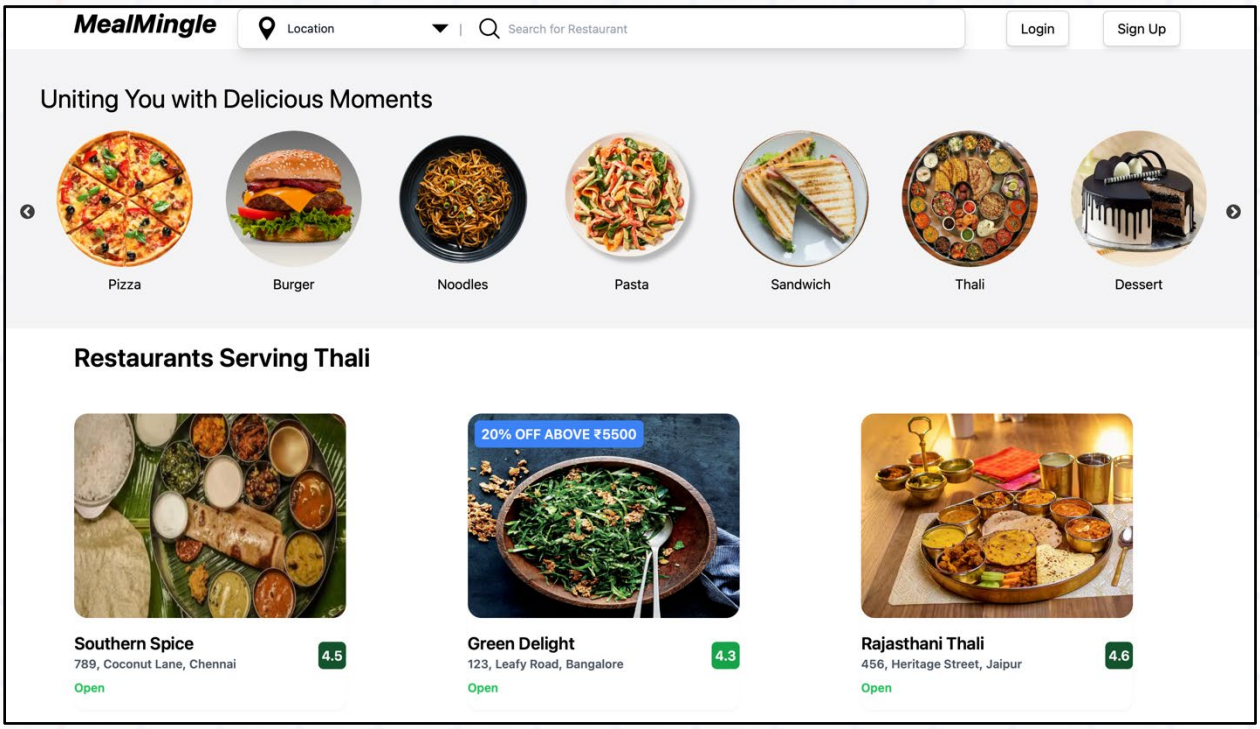
Build an E-marketplace web application



Contents

Prerequisites

Setting Up the Development Environment

Setting Up the Project Structure

Running your Project

Clean your Project

Edit your Project

Firebase Usage Guide

Creating Components

Prerequisites

Installation of VS Code

https://code.visualstudio.com/download

What is HTML?

- HTML stands for Hyper Text Markup Language.
- It is the standard markup language for creating Web pages.
- It describes the structure of a Web page.
- It consists of a series of elements.
- HTML elements tell the browser how to display the content.

HTML Page Structure

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Page Title</title>
</head>

<body>
    <!-- Content goes here -->
</body>

</html>
```

HTML Boilerplate

https://www.freecodecamp.org/news/basic-html5-template-boilerplate-code-example/

First HTML Program

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
</head>
<body>
<h1>Meal Mingle!</h1>
</body>
</html>
```

Tags and Elements

- Tags are keywords enclosed in angle brackets (< >) that specify how web browsers should format and display content on the web page.
  Examples: <h1>, , <img>, <a>.
- An Element consists of an opening tag, content, and a closing tag, forming a complete unit that defines a specific piece of content or functionality on a web page.
  Examples:
- Heading: <h1>My Heading</h1>
- Paragraph: This is a Paragraph.
- Image: <img src="MealMingle-logo.jpg" alt="MealMingle Logo">

Reference Links

- https://www.geeksforgeeks.org/html-tags-a-to-z-list/
- https://www.w3schools.com/html/html_elements.asp

Comments in HTML

- Comments are used to add notes or annotations within the code that are not displayed in the web browser.

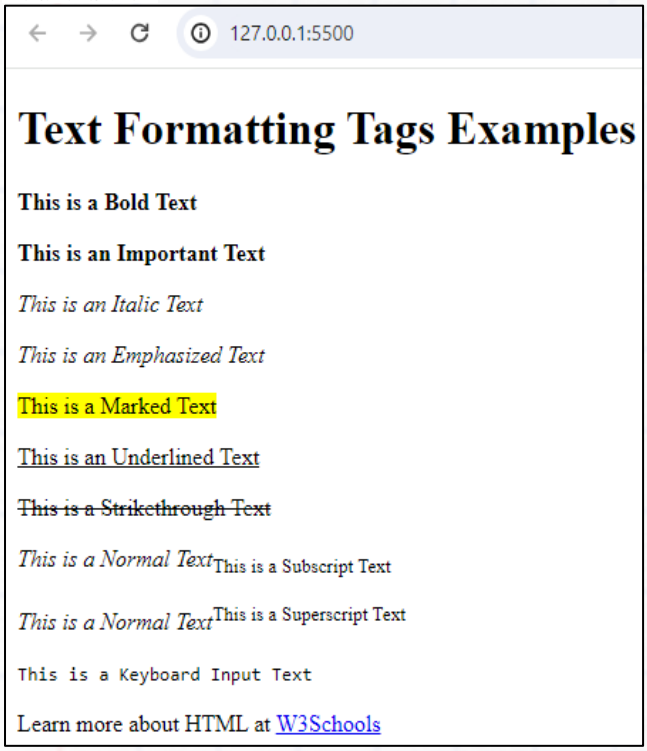
- They are useful for documenting your code, explaining complex sections, or temporarily disabling code without deleting it.

Reference Links

- https://www.w3schools.com/html/html_comments.asp

Text Formatting Tags

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>Text Formatting Tags Examples</title>
</head>

<body>
    <!-- Header Section -->
    <header>
        <h1>Text Formatting Tags Examples</h1>
    </header>

    <!-- Main Content Section -->
    <section>
        <p><b>This is a Bold Text</b></p>
        <p><strong>This is an Important Text</strong></p>
        <p><i>This is an Italic Text</i></p>
        <p><em>This is an Emphasized Text</em></p>
        <p><mark>This is a Marked Text</mark></p>
        <p><u>This is an Underlined Text</u></p>
        <p><s>This is a Strikethrough Text</s></p>
        <p><i>This is a Normal Text</i><sub>This is a Subscript Text</sub></p>
        <p><i>This is a Normal Text</i><sup>This is a Superscript Text</sup></p>
        <p><kbd>This is a Keyboard Input Text</kbd></p>
    </section>

    <!-- Footer Section -->
    <footer>
        <p>Learn more about HTML at <a href="https://www.w3schools.com/html/">W3Schools</a></p>
    </footer>

</body>

</html>
```



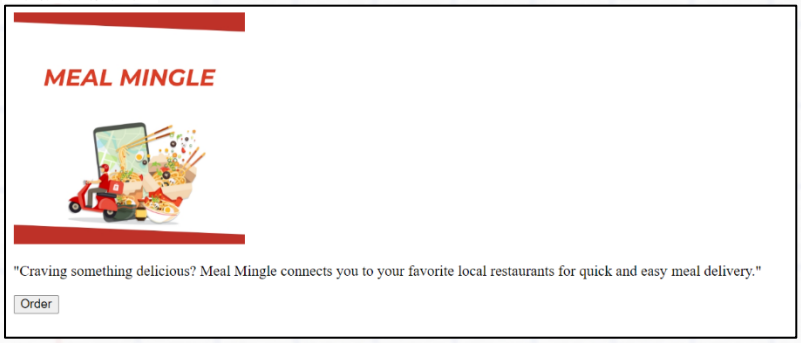
Reference Links

- https://www.w3schools.com/html/html_formatting.asp
- https://www.geeksforgeeks.org/html-text-formatting/

Attributes

- Attributes in HTML provide additional information about an element and are always specified in the opening tag.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" >
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>HTML Attributes Examples</title>
</head>
<body>
<!-- Image Tag -->
<img src="MealMingle-logo.jpg" alt="MealMingle-logo" width="250" height="250">
</a>
<!-- Paragraph Tag -->
<p align="left">"Craving something delicious? Meal Mingle connects you to your favorite local restaurants for quick and easy meal delivery.". </p>
<!-- Button Tag -->
<button type="button">Apply</button>
</body>
</html>

```



Reference Links

- https://www.w3schools.com/html/html_attributes.asp
- https://www.geeksforgeeks.org/html-attributes/

Semantic & Non-Semantic Elements

- Semantic Elements describe their meaning to both the browser and the developer, making the HTML code more readable and understandable.
  Examples of Semantic Elements include:

1. <header>: Represents a container for introductory content or navigation links.
2. <nav>: Defines a section with navigation links.
3. <main>: Contains the primary content of the document.
4. <article>: Represents a self-contained composition in a document.
5. <section>: Defines a section in a document, such as chapters, headers, footers, or any other content.
6. <aside>: Represents content that is tangentially related to the main content.
7. <footer>: Contains the footer information like author details, copyright, etc.

- Non-Semantic Elements are generic containers used to style content but do not provide any contextual meaning.
  Examples of Non-Semantic Elements include:

8. : A generic container for flow content.
9. : An inline container used to mark up a part of a text.
10.
: Line break.
11. <hr>: Horizontal rule (divider line).

Reference Links

- https://www.w3schools.com/html/html5_semantic_elements.asp
- https://www.geeksforgeeks.org/html5-semantics/

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Semantic and Non-Semantic Elements Examples</title>
</head>
<body>
<!-- Semantic Elements -->
<!-- Header -->
<header>
<h1>Welcome to Meal Mingle!</h1>
</header>
<!--Navigation -->
<nav>
<ul>
<li><a href="#home">Home</a></li>
<li><a href="#Restaurants">Restaurants</a></li>
<li><a href="#about">About Us</a></li>
<li><a href="#Meal">Meal</a></li>
</ul>
</nav>
<!-- Main Content -->
<main>
<!-- Article -->
<article>
<h2>About Meal Mingle</h2>
<p>"Craving something delicious? Meal Mingle connects you to your favorite local restaurants for quick and easy meal delivery."</p>
</article>
<!--Section -->
<section>
<h2>Our Services</h2>
<p>Satisfy your hunger with Meal Mingle, your ultimate destination for fast, reliable food delivery from the best local restaurants.
    Whether you're in the mood for a quick bite or a full-course meal, we've got you covered.
    Browse through a variety of cuisines, place your order, and let us handle the rest.
    Your favorite dishes are just a few taps away, delivered hot and fresh right to your door. Experience the convenience of dining with Meal Mingle today!.</p>
</section>
</main>
<!-- Footer -->
<footer>
<p>&copy; 2024 Meal Mingle</p>
</footer>
<!-- Non-Semantic Elements -->
<!-- Generic Container -->
<div class="container">
<!-- Inline Container -->
<span>Connect with us
</span>
</div>
<!-- Line Break -->
<br>
<!-- Horizontal Rule -->
<hr>
</body>
</html>

```

Welcome to Meal Mingle!

- Restaurants
- Meal

About Meal Mingle

"Craving something delicious? Meal Mingle connects you to your favorite local restaurants for quick and easy meal delivery."

Our Services

Satisfy your hunger with Meal Mingle, your ultimate destination for fast, reliable food delivery from the best local restaurants. Whether you're in the mood for a quick bite or a full-course meal, we've got you covered. Browse through a variety of cuisines, place your order, and let us handle the rest. Your favorite dishes are just a few taps away, delivered hot and fresh right to your door. Experience the convenience of dining with Meal Mingle today!.

© 2024 Meal Mingle

Connect with us

Block & Inline Elements

- Block-level Elements typically start on a new line and take up the full width available, pushing subsequent content to a new line.

Examples of Block-Level Elements are , <h1> to <h6>, <p>, <table>, <form>, <header>, <footer>, <section>, <article>, <aside>, <nav>, etc.

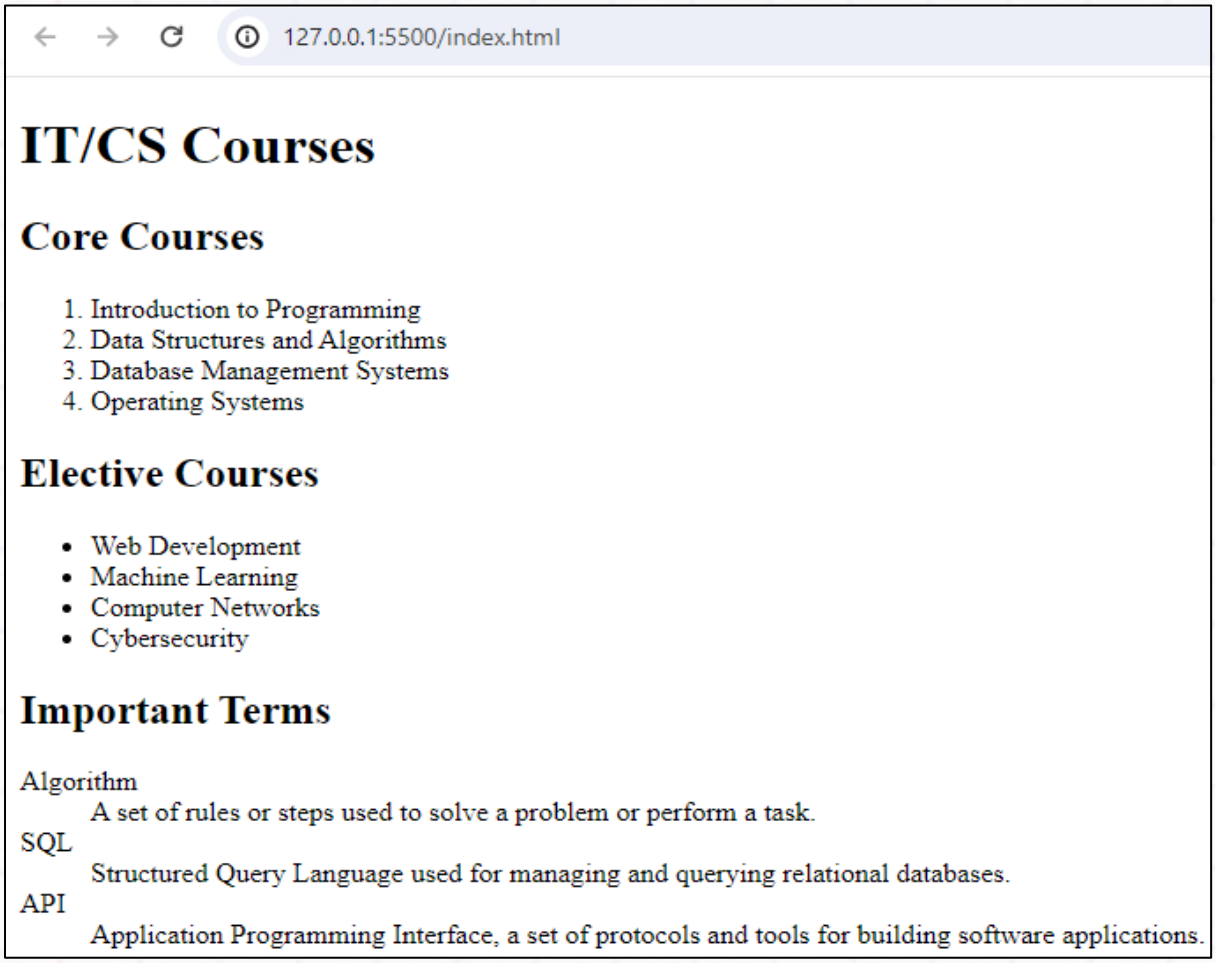
- Inline Elements, on the other hand, do not start on a new line and only take up as much width as necessary, allowing other elements to sit beside them.
  Examples of Inline Elements are , <a>, , , <img>, <input>, <code>, <label>, <select>, <textarea>, etc.

Reference Links

- https://www.w3schools.com/html/html_blocks.asp
- https://www.geeksforgeeks.org/html-block-and-inline-elements/

Lists

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>HTML Lists Examples</title>
</head>

<body>
    <h1>IT/CS Courses</h1>

    <!-- Ordered List: Core Courses -->
    <h2>Core Courses</h2>
    <ol>
        <li>Introduction to Programming</li>
        <li>Data Structures and Algorithms</li>
        <li>Database Management Systems</li>
        <li>Operating Systems</li>
    </ol>

    <!-- Unordered List: Elective Courses -->
    <h2>Elective Courses</h2>
    <ul>
        <li>Web Development</li>
        <li>Machine Learning</li>
        <li>Computer Networks</li>
        <li>Cybersecurity</li>
    </ul>

    <!-- Definition List: Important Terms -->
    <h2>Important Terms</h2>
    <dl>
        <dt>Algorithm</dt>
        <dd>A set of rules or steps used to solve a problem or perform a task.</dd>

        <dt>SQL</dt>
        <dd>Structured Query Language used for managing and querying relational databases.</dd>

        <dt>API</dt>
        <dd>Application Programming Interface, a set of protocols and tools for building software applications.</dd>
    </dl>
</body>

</html>
```



Reference Links

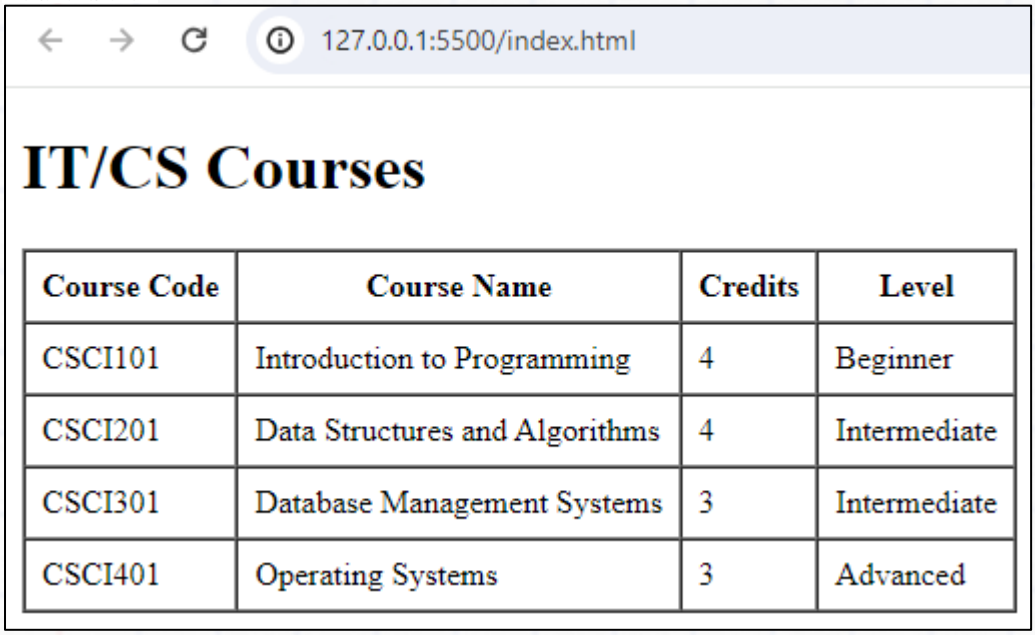
- https://www.w3schools.com/html/html_lists.asp
- https://www.geeksforgeeks.org/html-lists/

Tables

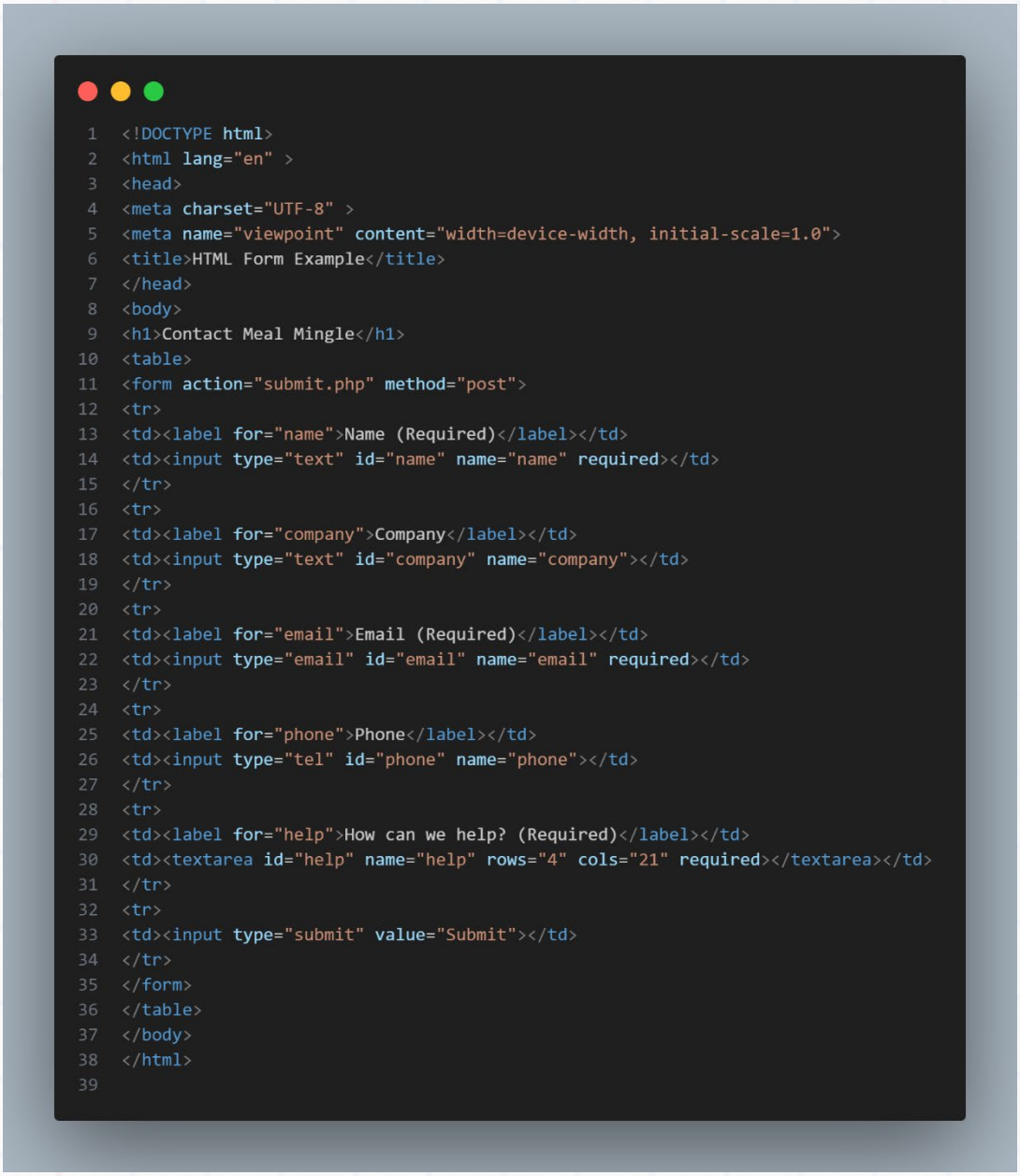
Reference Links

- https://www.w3schools.com/html/html_tables.asp
- https://www.w3schools.com/html/html_table_borders.asp
- https://www.w3schools.com/html/html_table_padding_spacing.asp
- https://www.w3schools.com/html/html_table_colspan_rowspan.asp
- https://www.geeksforgeeks.org/html-tables/

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>HTML Table Example</title>
</head>

<body>
    <h1>IT/CS Courses</h1>

    <!-- Table: Course Details -->
    <table border="1" cellspacing="0" cellpadding="8">
        <thead>
            <tr>
                <th>Course Code</th>
                <th>Course Name</th>
                <th>Credits</th>
                <th>Level</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>CSCI101</td>
                <td>Introduction to Programming</td>
                <td>4</td>
                <td>Beginner</td>
            </tr>
            <tr>
                <td>CSCI201</td>
                <td>Data Structures and Algorithms</td>
                <td>4</td>
                <td>Intermediate</td>
            </tr>
            <tr>
                <td>CSCI301</td>
                <td>Database Management Systems</td>
                <td>3</td>
                <td>Intermediate</td>
            </tr>
            <tr>
                <td>CSCI401</td>
                <td>Operating Systems</td>
                <td>3</td>
                <td>Advanced</td>
            </tr>
        </tbody>
    </table>
</body>

</html>
```


127.0.0.1:5500/index.html
IT/CS Courses
Course Code
Course Name
Credits
Level
CSCI101
Introduction to Programming
4
Beginner
CSCI201
Data Structures and Algorithms
4
Intermediate
CSCI301
Database Management Systems
3
Intermediate
CSCI401
Operating Systems
3
Advanced

Forms



```html
<!DOCTYPE html>
<html lang="en" >
<head>
<meta charset="UTF-8" >
<meta name="viewpoint" content="width=device-width, initial-scale=1.0">
<title>HTML Form Example</title>
</head>
<body>
<h1>Contact Meal Mingle</h1>
<table>
<form action="submit.php" method="post">
<tr>
<td><label for="name">Name (Required)</label></td>
<td><input type="text" id="name" name="name" required></td>
</tr>
<tr>
<td><label for="company">Company</label></td>
<td><input type="text" id="company" name="company"></td>
</tr>
<tr>
<td><label for="email">Email (Required)</label></td>
<td><input type="email" id="email" name="email" required></td>
</tr>
<tr>
<td><label for="phone">Phone</label></td>
<td><input type="tel" id="phone" name="phone"></td>
</tr>
<tr>
<td><label for="help">How can we help? (Required)</label></td>
<td><textarea id="help" name="help" rows="4" cols="21" required></textarea></td>
</tr>
<tr>
<td><input type="submit" value="Submit"></td>
</tr>
</form>
</table>
</body>
</html>

```

Reference Links

- https://www.w3schools.com/html/html_forms.asp
- https://www.geeksforgeeks.org/html-forms/

What is CSS?

- CSS stands for Cascading Style Sheets, a stylesheet language used to style and format HTML documents, determining their appearance on the web.
- It enables developers to control the visual aspects of web pages, such as colors, fonts, spacing, and layout, to create attractive and engaging designs.

First CSS Program

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
<link rel="stylesheet" href="Style.css">
</head>
<body>
<h1>Hello Meal Mingle!</h1>
</body>
</html>
```

```css
h1 {
    color: #F1A01F;
    }
```

Types of CSS

Inline CSS

```
<!DOCTYPE html>
<html lang="en" >
<head>
<meta charset="UTF-8" >
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
</head>
<body>
<h1 style="color: #F1A01F;">Hello Meal Mingle!</h1>
</body >
</html >
```

Internal CSS

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" >
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
<style>
h1 {
color: #F1A01F;
}
</style>
</head>
<body>
<h1>Hello Meal Mingle!</h1>
</body>
</html>
```

External CSS

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
<link rel="stylesheet" href="Style.css">
</head>
<body>
<h1>Hello GenSpark!</h1>
</body>
</html>
```

```css
h1 {
    color: #F1A01F;
    }
```

Reference Links

- https://www.geeksforgeeks.org/types-of-css-cascading-style-sheet/

CSS Selectors

Reference Links

- https://www.w3schools.com/css/css_selectors.asp
- https://www.geeksforgeeks.org/css-selectors/

CSS Comments

Reference Links

https://www.w3schools.com/css/css_comments.asp

Specificity

- Specificity in CSS determines which style rule is applied to an element when there are conflicting styles.
- It's essential to understand specificity to ensure that the intended styles are applied correctly.
- Specificity is calculated based on the combination of selectors used to target an element.

- It follows a hierarchy, where selectors with higher specificity override those with lower specificity.
  The specificity hierarchy is as follows:
  1. Inline Styles: Styles applied directly to an element using the style attribute. These have the highest specificity.
  2. ID Selectors: Selectors targeting elements by their ID attribute (#id). They have higher specificity than class selectors and element selectors.
  3. Class Selectors, Attribute Selectors, and Pseudo-Classes: Selectors targeting elements by class (.class), attribute ([attribute]), or pseudo-classes (:hover, :active, etc.).
  4. Type Selectors and Pseudo-Elements: Selectors targeting elements by their type (e.g., div, p, a) or pseudo-elements (::before, ::after).

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Specificity Example</title>
<link rel="stylesheet" href="styles.css">
</head>
<body>
<div id="content" class="container">
<!-- Inline CSS -->
<p style="color: #F1A01F;">Hello Meal Mingle!</p>
</div>
</body>
</html>
```

```css
/* ID Selector */
#content {
color: blue;
}
/* Class Selector */
.container {
 color: red;
}
/* Element Selector */
p {
color: green;
}
```

Hello Meal Mingle!

Colors

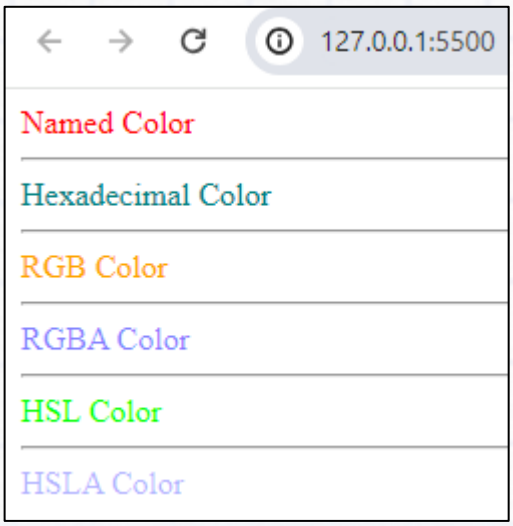
Reference Links

https://www.w3schools.com/css/css_colors.asp
https://www.w3schools.com/css/css_colors_rgb.asp
https://www.w3schools.com/css/css_colors_hex.asp

https://www.w3schools.com/css/css_colors_hsl.asp
https://www.w3schools.com/cssref/css_colors_legal.php

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>CSS Color Examples</title>
    <link rel="stylesheet" href="styles.css">
</head>

<body>
    <div id="named-color">Named Color</div>
    <hr>
    <div id="hex-color">Hexadecimal Color</div>
    <hr>
    <div id="rgb-color">RGB Color</div>
    <hr>
    <div id="rgba-color">RGBA Color</div>
    <hr>
    <div id="hsl-color">HSL Color</div>
    <hr>
    <div id="hsla-color">HSLA Color</div>
</body>

</html>
```

```css
/* Named Color */
#named-color {
    color: red;
}

/* Hexadecimal Color */
#hex-color {
    color: #008080;
}

/* RGB Color */
#rgb-color {
    color: rgb(255, 165, 0);
}

/* RGBA Color */
#rgba-color {
    color: rgba(0, 0, 255, 0.5);
}

/* HSL Color */
#hsl-color {
    color: hsl(120, 100%, 50%);
}

/* HSLA Color */
#hsla-color {
    color: hsla(240, 100%, 50%, 0.3);
}
```



Backgrounds

Reference Links

https://www.w3schools.com/cssref/pr_background-color.php
https://www.w3schools.com/cssref/pr_background-image.php
https://www.w3schools.com/cssref/css3_pr_background-size.php
https://www.w3schools.com/cssref/pr_background-attachment.php
https://www.w3schools.com/cssref/pr_background-position.php
https://www.w3schools.com/css/css_background_shorthand.asp

```html
<!DOCTYPE html>
<html lang="en" >
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>CSS Background Example</title>
<link rel="stylesheet" href="styles1.css">
</head>
<body>
<div class="background-example">
<h2>About Meal Mingle</h2>
<br>
<ul>
<li>Welcome to Meal Mingle! Satisfy your hunger with Meal Mingle, your ultimate destination for fast, reliable food delivery
<br>from the best local restaurants. </li>
<br>
<li>Your favorite dishes are just a few taps away, delivered hot and fresh right to your door.</li>
<br>
</ul>
</div>
</body>
</html >

```

```css
* {
    margin: 0;
    padding: 0;
    }
    .background-example {
    width: 100vw;
    height: 100vh;
    padding: 20px;
    font-size: 24px;
    color: black;
    background-color: white;
    background-image: url('MealMingle-Banner.jpg');
    background-repeat: no-repeat;
    background-size: contain;
    background-attachment: fixed;
    background-position: bottom;
    }
    h2 {
    text-align: center;
    }
```

Box Model

Reference Links

https://www.w3schools.com/css/css_boxmodel.asp
https://www.w3schools.com/css/tryit.asp?filename=trycss_boxmodel

Margin Collapse

Reference Links

https://www.w3schools.com/css/css_margin_collapse.asp
https://www.joshwcomeau.com/css/rules-of-margin-collapse/

Box Sizing

- The box-sizing property in CSS controls how the width and height of an element are calculated, including padding and border, or excluding them from the calculation.
- This property helps maintain consistency in layout and simplifies sizing calculations.

Reference Links

https://www.w3schools.com/css/css3_box-sizing.asp
https://www.w3schools.com/cssref/playdemo.php?filename=playcss_box-sizing

By default, the box-sizing property is set to content-box, where the width and height of an element only include the content area, excluding padding and border.

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Box Sizing Example</title>
<link rel="stylesheet" href="Styles2.css">
</head>
<body>
<div class="box">
<img src="MealMingle-logo.jpg" alt="MealMingle-Logo">
</div>
</body>
</html>
```

```css
.box {
    width: 500px;
    height: 500px;
    padding: 20px;
    border: 2px solid #f11f1f;
    margin: 20px;
    }
```

In this example, the Total Width of the .box element is calculated as follows:

- Content Width: 410px (specified)
- Padding: 20px (on both sides, so 40px in total)
- Border: 2px (on both sides, so 4px in total)

Total Width = Content Width + Padding + Border = 410px + 40px + 4px = 454px. Similarly, Total Height = Content Height + Padding + Border = 150px + 40px + 4px = 194px.

When box-sizing is set to border-box, the width and height of an element include padding and border, and only the content area is resized.

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Box Sizing Example</title>
<link rel="stylesheet" href="Styles3.css">
</head>
<body>
<div class="box">
<img src="MealMingle-logo.jpg" alt="MealMingle-Logo">
</div>
</body>
</html>

```

```css
.box {
    width: 480px;
    height: 550px;
    padding: 20px;
    border: 2px solid #f11f1f;
    margin: 20px;
    box-sizing: border-box;
    }
```

In this example, the Total Width of the .box element is calculated as follows:

- Content Width: 410px (specified)
- Padding: 20px (on both sides, included in the width calculation)
- Border: 2px (on both sides, included in the width calculation)

Total Width = Content Width = 410px.
Similarly, Total Height = Content Height = 150px.

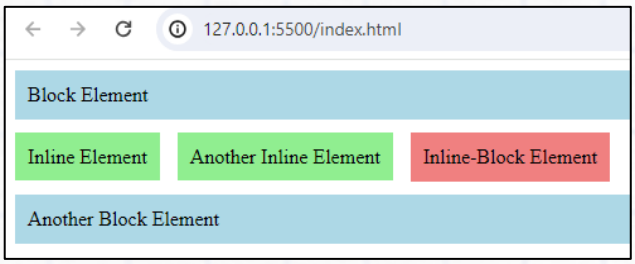
Display Property

Reference Links

https://www.w3schools.com/cssref/pr_class_display.php

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Display Property Example</title>
    <link rel="stylesheet" href="styles.css">
</head>

<body>
    <div class="block">Block Element</div>
    <span class="inline">Inline Element</span>
    <span class="inline">Another Inline Element</span>
    <div class="inline-block">Inline-Block Element</div>
    <div class="block">Another Block Element</div>
    <div class="none">Hidden Element</div>
</body>

</html>
```

```css
.block {
    display: block;
    background-color: lightblue;
    padding: 10px;
    margin-bottom: 10px;
}

.inline {
    display: inline;
    background-color: lightgreen;
    padding: 10px;
    margin-right: 10px;
}

.inline-block {
    display: inline-block;
    background-color: lightcoral;
    padding: 10px;
    margin-bottom: 10px;
}

.none {
    display: none;
}
```



Text Properties

Reference Links

https://www.geeksforgeeks.org/css-text-formatting/

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Text Properties Example</title>
<link rel="stylesheet" href="Styles4.css">
</head>
<body>
<h1>About Meal Mingle</h1>
<p>Welcome to Meal Mingle! Satisfy your hunger with Meal Mingle, your ultimate destination for fast, reliable food delivery from the best local restaurants.</p>
<p>Whether you're in the mood for a quick bite or a full-course meal, we've got you covered.</p>
<p>Browse through a variety of cuisines, place your order, and let us handle the rest.Your favorite dishes are just a few taps away, delivered hot and fresh right to your door.</p>
<p class="highlight">Experience the convenience of dining with Meal Mingle today!</p>
<p class="center">Connect with us: Meal Mingle</a></p>
</body>
</html>
```

```css
body {
    font-family: Arial, sans-serif;
    line-height: 1.6;
}
h1 {
    font-size: 24px;
    color: #F1A01F;
    text-align: center;
    text-decoration: underline;
}
p {
    font-size: 16px;
}
.highlight {
    color: white;
    background-color: #18243F;
    font-style: italic;
    font-weight: bold;
}
.center {
    text-align: center;
}
```

About Meal Mingle

Welcome to Meal Mingle! Satisfy your hunger with Meal Mingle, your ultimate destination for fast, reliable food delivery from the best local restaurants.

Whether you're in the mood for a quick bite or a full-course meal, we've got you covered.

Browse through a variety of cuisines, place your order, and let us handle the rest.Your favorite dishes are just a few taps away, delivered hot and fresh right to your door.

Experience the convenience of dining with Meal Mingle today!

Connect with us: Meal Mingle

Meal Mingle Frontend Guide

1. Setting Up the Development Environment

Prerequisites:

- Node.js and npm installed on your machine.
- A code editor like VS Code.

Steps:

1. **Install Node.js and npm:**
   - Download and install Node.js from the official website.
   - Verify the installation by running node -v and npm -v in your terminal.

```
node -v
v20.13.1
```

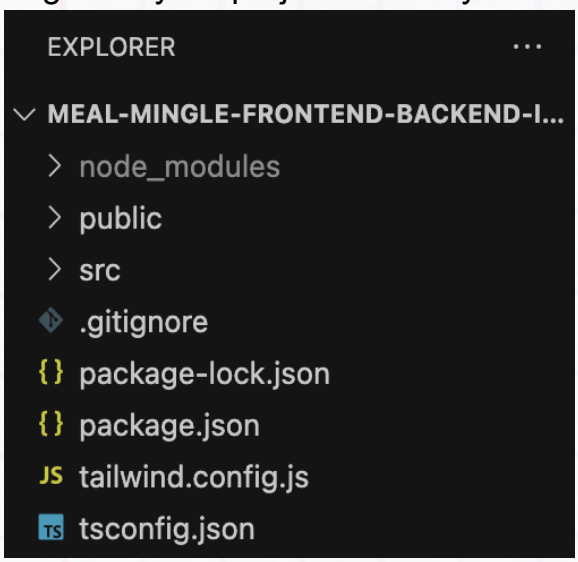
```
npm -v
10.5.2
```

2. **Create a React App with TypeScript:**

```
npx create-react-app mealmingle --template typescript
cd mealmingle
```

2. Setting Up the Project Structure

Organize your project directory as follows:

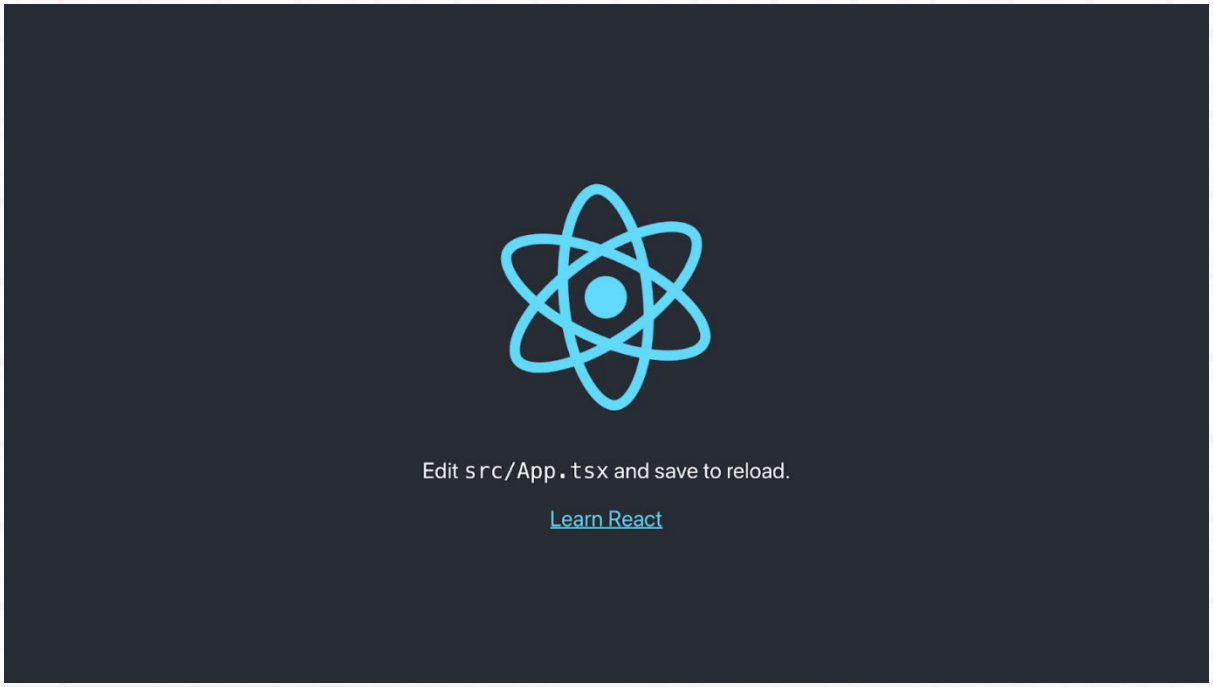


3. Running your Project

Start the Development Server:

- To run your React application in development mode, use:

```
npm start
```



4. Cleanup your Project

1. Delete the following files from the public directory:

- public/favicon.ico
- public/logo192.png
- public/logo512.png
- public/manifest.json
- public/robots.txt

2. Delete the following files from the src directory:

- src/App.test.tsx
- src/logo.svg
- src/reportWebVitals.ts
- src/setupTests.ts

3. Delete README.md.

5. Edit your Project

1. Remove unnecessary code from public/index.html.

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>
```

2. Remove all the code from src/App.css.

3. Install Tailwind CSS.

Reference Link: https://tailwindcss.com/docs/guides/create-react-app

Run the following commands in your project directory:

```
npm install -D tailwindcss
npx tailwindcss init
```

4. Edit tailwind.config.js file.

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: ["./src/**/*.{js,jsx,ts,tsx}",],
  theme: {
    extend: {
      colors: {
        customOrange: '#F1A01F',
      },
    },
  },
  plugins: [],
}
```

5. Edit src/App.tsx file.

```
import React from 'react'
const App = () => {
return (
<h1 className="text-3xl font-bold underline text-customOrange"> Hello Meal Mingle!
</h1>
)
}
export default App
```

6. Edit src/index.css file.

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

7. Edit src/index.tsx file.

```tsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';

const root = ReactDOM.createRoot(
  document.getElementById('root') as HTMLElement
);
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

8. Run your project.

Firebase Usage Guide

To add a project in Firebase, follow these steps:

1. Create a Firebase Project

    1. **Sign in to Firebase**:
        - Go to the Firebase Console.
        - Sign in with your Google account.
    2. **Add a New Project**:
        - Click on the **"Add project"** button.
        - Enter a name for your project.
        - (Optional) You can also enable Google Analytics for your project. If you choose to do so, you'll need to select an existing Google Analytics account or create a new one.
        - Click on **"Continue"** and follow the prompts to finish setting up your project.

2. Register Your App with Firebase

    1. **Choose Your Platform**:
        - In the Firebase Console, go to your project's overview page.
        - Click on the icon for the platform you want to add (Web, iOS, Android).

    2. **Register Your App**:

        - Enter the app's nickname.
        - Click on **"Register app"**.

    3. **Add Firebase SDK**:

For a web project using npm, follow these steps:

Install Firebase:

```
npm install firebase
```

Initialize Firebase in Your Project:

- Firebase will provide you with a code snippet that you need to add to your app.
- Create a new folder, firebase, inside the src directory and add setup.tsx file.

- Write the following code in setup.tsx file.

```
import { initializeApp } from "firebase/app";
import { getAuth, GoogleAuthProvider } from "firebase/auth";

const firebaseConfig = {
    apiKey: "YOUR_API_KEY",
    authDomain: "YOUR_PROJECT_ID.firebaseapp.com",
    projectId: "YOUR_PROJECT_ID",
    storageBucket: "YOUR_PROJECT_ID.appspot.com",
    messagingSenderId: "YOUR_MESSAGING_SENDER_ID",
    appId: "YOUR_APP_ID"
};

const app = initializeApp(firebaseConfig);
export const auth = getAuth(app);
export const googleAuthProvider = new GoogleAuthProvider();
```

3. Add Firebase Authentication

1. Enable Authentication Method:
    - In the Firebase Console, go to Authentication -> Sign-in method.
    - Enable the Authentication Methods you plan to use (e.g., Email/Password, Google, Phone).

Note: Before creating each component, make sure to refer to the `App.tsx` file at the end of this documentation to understand how the routes are set up and where each component is used.

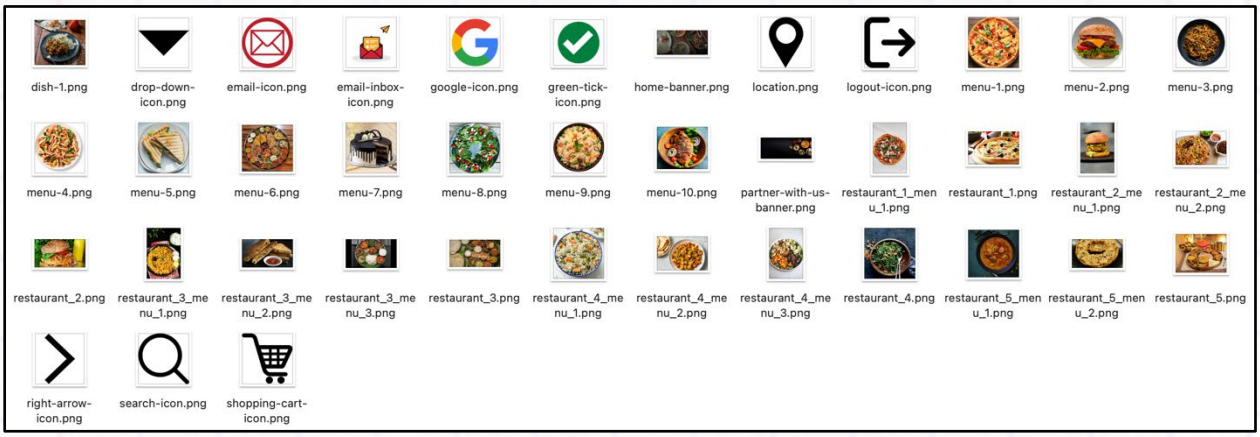
6. Creating Components

Create a components folder and an images folder inside the src directory.

Components folder would contain all the components (for User as well as Admin dashboards) of the Meal Mingle application.

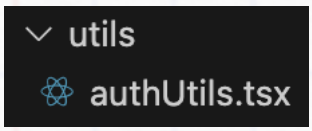
The images folder would contain all the images required for developing the application.

Restaurant and its Menu images were first used locally which was then replaced by the backend code.



Before diving deep into the Components creation, let us create a new folder, utils, inside the src directory.

- The isAuthenticated function checks if a user is logged in by looking for a token in localStorage.
- If the token exists, it returns true, indicating the user is authenticated; otherwise, it returns false.




```
export const isAuthenticated = (): boolean => {

    const token = localStorage.getItem('token');



    return !!(token);

};
```

6.1. Signup Component

- The Signup Component is a React component designed for User Registration in a web application.
- It provides a primary method for user registration: Email and Password.
- The purpose of this component is to handle User Sign-Up processes and ensure that new users are properly registered within the application.

Create a new file inside the components folder and name it Signup.tsx.

Now, install the following dependencies:

1. **React Router DOM**: For routing in your React application.

```
npm install react-router-dom
```

2. **React Toastify**: To display toast notifications.

```
npm install react-toastify
```

3. **libphonenumber-js**: For phone number validation.

```
npm install libphonenumber-js
```

1. Imports

- **React**: The library used to build the component.
- **useState**: A React hook for managing state (data that changes over time) in the component.
- **Firebase Functions**: Functions from Firebase used for authentication (e.g., createUserWithEmailAndPassword).
- **auth**: This is your Firebase authentication setup imported from a separate file.
- **useNavigate**: A hook from React Router to navigate between different pages.
- **ToastContainer and toast**: Used for showing notifications (toasts) on the screen.
- **parsePhoneNumberFromString**: A utility to validate phone numbers.

```
import React, { useState } from 'react';

import { createUserWithEmailAndPassword, sendEmailVerification,

signInWithPopup, GoogleAuthProvider, onAuthStateChanged } from

'firebase/auth';
```

```
import { auth } from '../firebase/setup';

import { Link, useNavigate } from 'react-router-dom';

import { ToastContainer, toast } from 'react-toastify';

import 'react-toastify/dist/ReactToastify.css';

import { parsePhoneNumberFromString } from 'libphonenumber-js';
```

2. Component Definition

- **Signup**: The functional component definition.
- **useNavigate**: Initializes the navigation function to move between pages.
- **useState**: Initializes state variables (name, email, password, phone, error) with default values. These hold the data input by the user.

```
const Signup = () => {

const navigate = useNavigate();

const [name, setName] = useState("");

const [email, setEmail] = useState("");

const [password, setPassword] = useState("");

const [phone, setPhone] = useState("");

const [error, setError] = useState<string | null>(null);
```

3. User Sign Up Function

- **userSignup**: Sends user data to a backend API to register the user.
- **fetch**: Makes an HTTP request to the server.
- **response.json()**: Converts the response to a JSON object.

```
const userSignup = async () => {

const response = await fetch('http://localhost:8090/api/users/register/user',

{
```

```
method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ userName: name, userEmail: email, userPassword:

password, userPhone: phone })

});

const data = await response.json();

console.log(data);

}
```

4. Validation Functions
   - **validateEmail**: Checks if the email format is correct.
   - **validatePassword**: Checks if the password is at least 6 characters long.
   - **validatePhoneNumber**: Validates if the phone number is valid using the libphonenumber-js library.
   - **checkPhoneNumberExists**: Checks if the phone number already exists in the database.

```
const validateEmail = (email: string) => {

const re = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

return re.test(String(email).toLowerCase());

};


const validatePassword = (password: string) => {

return password.length >= 6;

};


const validatePhoneNumber = (phone: string) => {
```

```
const phoneNumber = parsePhoneNumberFromString(phone, 'IN');

return phoneNumber && phoneNumber.isValid();

};



const checkPhoneNumberExists = async (phone: string) => {

try {

const response = await

fetch(`http://localhost:8090/api/users/phone/verify?phone=${phone}`, {

method: 'GET',

headers: {

'Content-Type': 'application/json'

}

});

const data = await response.json();

return data.exists;

} catch (error) {

console.error(error);

return false;

}

};
```

5. Email Sign-Up Function

- **emailSignUp**: Handles the entire sign-up process:
  1. Validates input fields.
  2. Checks if the phone number is already used.
  3. Calls userSignup to register the user on the backend.
  4. Uses Firebase to create the user account.
  5. Shows appropriate success or error messages using toast.
  6. Redirects the user to the login page after a short delay.

```
const emailSignUp = async () => {

if (!name.trim()) {

toast.error('Name is Required');

return;

}

if (!validateEmail(email)) {

toast.error('Invalid Email Format');

return;

}


if (!validatePassword(password)) {

toast.error('Password must be at least 6 characters long!');

return;

}


if (!validatePhoneNumber(phone)) {

toast.error('Invalid Phone Number');

return;

}


try {

const phoneExists = await checkPhoneNumberExists(phone);

if (phoneExists) {

toast.error('Phone Number is Already in Use!');

return;
```

```
}

userSignup();
const userCredential = await createUserWithEmailAndPassword(auth, email,
password);
const user = userCredential.user;

toast.success('Signed Up Successfully! Redirecting to Login page.');

onAuthStateChanged(auth, async (user) => {
if (user) {
console.log('User Signed Up Successfully:', user.uid);
}
});

setTimeout(() => {
navigate('/login');
}, 3000);
}
catch (err: any) {
console.error(err);
if (err.code === 'auth/email-already-in-use') {
toast.error('Email Address is Already in Use!');
} else if (err.code === 'auth/invalid-email') {
toast.error('Invalid Email Format');
```

```
} else {

toast.error('Failed to Sign Up. Please try again later.');

}

}

};
```

6. handleClose Function

- **handleClose**: Closes the sign-up modal and navigates back to the home page.

```
const handleClose = () => {

navigate('/');

};
```

7. Return JSX

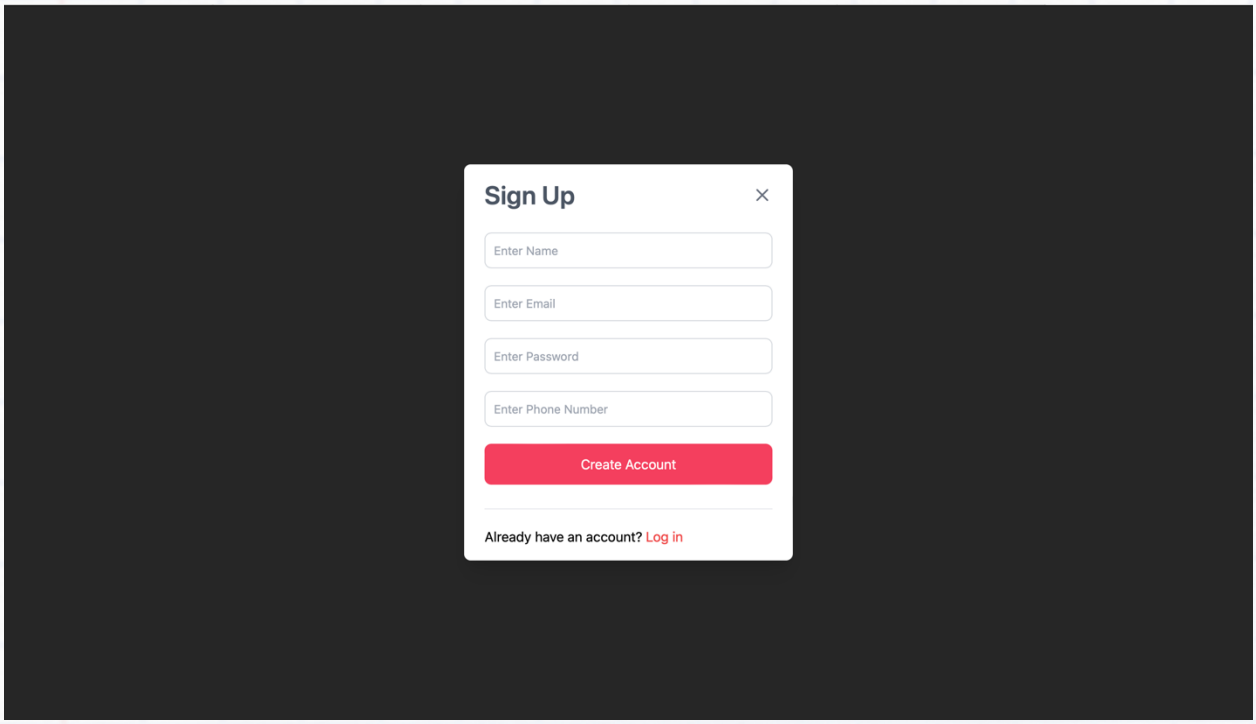
- **ToastContainer**: Displays toast notifications on the screen.
- **Modal Structure**: Uses div elements to create a modal dialog for the sign-up form.
- **Input Fields**: Fields for name, email, password, and phone number. They update the component's state when changed.
- **Create Account Button**: Calls emailSignUp when clicked.
- **Login Link**: Navigates to the login page if the user already has an account.

```
return (

<>

<ToastContainer />

<div className="relative z-10" aria-labelledby="modal-title" role="dialog"

aria-modal="true">

<div className="fixed inset-0 bg-black bg-opacity-85 transition-

opacity"></div>
```

```
<div className="fixed inset-0 z-10 w-screen overflow-y-auto">
<div className="flex min-h-full items-end justify-center p-4 text-center
sm:items-center sm:p-0">
<div className="relative transform overflow-hidden rounded-lg bg-white text-
left shadow-xl transition-all sm:my-8 sm:w-97 sm:max-w-lg">
<div className="bg-white px-4 pb-4 pt-5 sm:p-6 sm:pb-4">
<div className='flex'>
<h3 className="text-3xl font-semibold leading-6 text-gray-600" id="modal-
title">Sign Up</h3>
<button
onClick={handleClose}
className="text-gray-500"
>
<svg
xmlns="http://www.w3.org/2000/svg"
className="h-6 w-6 ml-52"
fill="none"
viewBox="0 0 24 24"
stroke="currentColor"
>
<path
strokeLinecap="round"
strokeLinejoin="round"
strokeWidth="2"
d="M6 18L18 6M6 6l12 12"
```

```
/>
</svg>
</button>
</div>
{error && <div className="text-red-500 text-sm">{error}</div>}
<input onChange={(e) => setName(e.target.value)} className="mt-8 outline-none
border border-gray-300 text-gray-900 text-sm rounded-lg block w-full p-2.5"
placeholder="Enter Name" required />
<input onChange={(e) => setEmail(e.target.value)} className="mt-5 outline-
none border border-gray-300 text-gray-900 text-sm rounded-lg block w-full p-
2.5" placeholder="Enter Email" required />
<input type='password' onChange={(e) => setPassword(e.target.value)}
className="mt-5 outline-none border border-gray-300 text-gray-900 text-sm
rounded-lg block w-full p-2.5" placeholder="Enter Password" required />
<input onChange={(e) => setPhone(e.target.value)} className="mt-5 outline-
none border border-gray-300 text-gray-900 text-sm rounded-lg block w-full p-
2.5" placeholder="Enter Phone Number" required />
<button onClick={emailSignUp} className="mt-5 mb-3 bg-rose-500 w-full h-12
text-white py-2 px-4 rounded-lg">
Create Account
</button>
<hr className='mt-4' />
<div className='text-base mt-5'>Already have an account? <Link
to='/login'><span className='text-red-500'>Log in</span></Link></div>
</div>
```

```
</div>

</div>

</div>

</div>

</>

);

}


export default Signup;
```



6.2. Login Component

- The Login Component is a React component designed for User Authentication in a web application.
- It provides three Authentication Methods: Phone Number OTP (One Time Password), Google Sign-In, and Email Login.

- The primary purpose of this component is to handle User Login processes.

1. Install react-phone-input-2 package which provides a ready-to-use phone input component that includes country codes and flags, making it more intuitive and user-friendly.
   Reference Link: https://www.npmjs.com/package/react-phone-input-2

   Open your terminal and run the following command in your project directory:

```
npm install react-phone-input-2
```

2. Create a new file inside the components folder and name it Login.tsx.

1. Imports

- **React**: Library used for building the component.
- **useState**: React hook for managing state within the component.
- **Firebase Functions**: Used for authentication operations (RecaptchaVerifier, signInWithPhoneNumber, signInWithPopup).
- **auth**: Firebase authentication instance.
- **googleAuthProvider**: Firebase provider for Google sign-in.
- **useNavigate**: Hook for navigation within the app.
- **PhoneInput**: Component for entering phone numbers.
- **GoogleIcon & EmailIcon**: Images used for Google and email sign-in options.
- **ToastContainer & toast**: Libraries for showing notifications on the screen.

```
import React, { useState } from 'react';

import { RecaptchaVerifier, signInWithPhoneNumber, signInWithPopup } from

'firebase/auth';

import { auth, googleAuthProvider } from '../firebase/setup';

import { Link, useNavigate } from 'react-router-dom';

import PhoneInput from 'react-phone-input-2';

import 'react-phone-input-2/lib/style.css';

import GoogleIcon from '../images/google-icon.png';

import EmailIcon from '../images/email-icon.png';
```

```
import { ToastContainer, toast } from 'react-toastify';

import 'react-toastify/dist/ReactToastify.css';
```

2. Component Definition

- **Login**: Functional component for the login screen.
- **useNavigate**: Initializes navigation for redirecting users.
- **useState**: Initializes state variables for phone number, OTP, and confirmation result.

```
const Login = () => {

const navigate = useNavigate();

const [phone, setPhone] = useState("");

const [otp, setOtp] = useState("");

const [confirmationResult, setConfirmationResult] = useState<any>(null);
```

3. verifyPhoneNumber Function

- **verifyPhoneNumber**: Checks if the phone number exists in the backend database by making a GET request.

```
const verifyPhoneNumber = async (phone: string) => {

try {

const response = await

fetch(`http://localhost:8090/api/users/phone/verify?phone=${phone}`, {

method: 'GET',

headers: {

'Content-Type': 'application/json'

}

});

const data = await response.json();
```

```
return data.exists;

} catch (error) {

console.error('Failed to verify phone number:', error);

return false;

}

};
```

4. sendOtp Function

- **sendOtp**: Sends an OTP to the user's phone number:
  1. Verifies if the phone number exists.
  2. Uses Firebase's RecaptchaVerifier to handle reCAPTCHA.
  3. Sends OTP via signInWithPhoneNumber.
  4. Saves confirmationResult to be used for OTP verification.

```
const sendOtp = async () => {

try {

const phoneExists = await verifyPhoneNumber(phone);



if (!phoneExists) {

toast.error('Failed to Verify Phone Number!');

return;

}



const recaptcha = new RecaptchaVerifier(auth, "recaptcha", {});

const confirmationResult = await signInWithPhoneNumber(auth, phone,

recaptcha);

setConfirmationResult(confirmationResult);

toast.success('OTP Sent Successfully!');
```

```
} catch (err: any) {

console.error(err);

toast.error('Failed to Send OTP!');

}

};
```

5. verifyOtp Function

- **verifyOtp**: Verifies the OTP entered by the user:
  1. Confirms the OTP using confirmationResult.
  2. Retrieves the current user and token from Firebase.
  3. Stores the token in localStorage and redirects the user to the home page.

```
const verifyOtp = async () => {

try {

if (!confirmationResult) throw new Error("Confirmation Result Not Found");

await confirmationResult.confirm(otp);

const user = auth.currentUser;

if (user) {

const token = await user.getIdToken();

localStorage.setItem("token", token);

toast.success('Logged In with OTP Successfully!');

setTimeout(() => {

navigate("/");

}, 2000);

} else {

toast.error('User Authentication Failed!');

}

} catch (err: any) {
```

```
console.error(err);

toast.error('Failed to Verify OTP!');

}

};
```

6. googleSignIn Function

- **googleSignIn**: Handles Google sign-in:
  1. Uses signInWithPopup to authenticate the user via Google.
  2. Saves the user's token in localStorage and redirects to the home page.

```
const googleSignIn = async () => {

try {

const data = await signInWithPopup(auth, googleAuthProvider);

const user = auth.currentUser;

if (user) {

const token = await user.getIdToken();

localStorage.setItem("token", token);

toast.success('Logged In with Google Successfully!');

setTimeout(() => {

navigate("/");

}, 2000);

} else {

toast.error('User Authentication Failed!');

}

} catch (err: any) {

console.error(err);

toast.error('Failed to Sign In with Google!');

}
```

```
};
```

7. handleClose Function

- **handleClose**: Closes the login modal and navigates back to the home page.

```
const handleClose = () => {

navigate('/');

};
```

8. Return JSX

- **ToastContainer**: Displays notifications for success or error messages.
- **Modal Structure**: Provides a modal dialog layout with a dark overlay.
- **Phone Input**: Allows users to input their phone number with country code.
- **Send OTP Button**: Triggers the sendOtp function to send an OTP.
- **OTP Input Field**: Allows users to enter the received OTP.
- **Verify OTP Button**: Calls verifyOtp to authenticate the user with the entered OTP.
- **Alternative Sign-in Options**: Provides links for email login and Google sign-in.
- **Create Account Link**: Navigates to the sign-up page for new users.

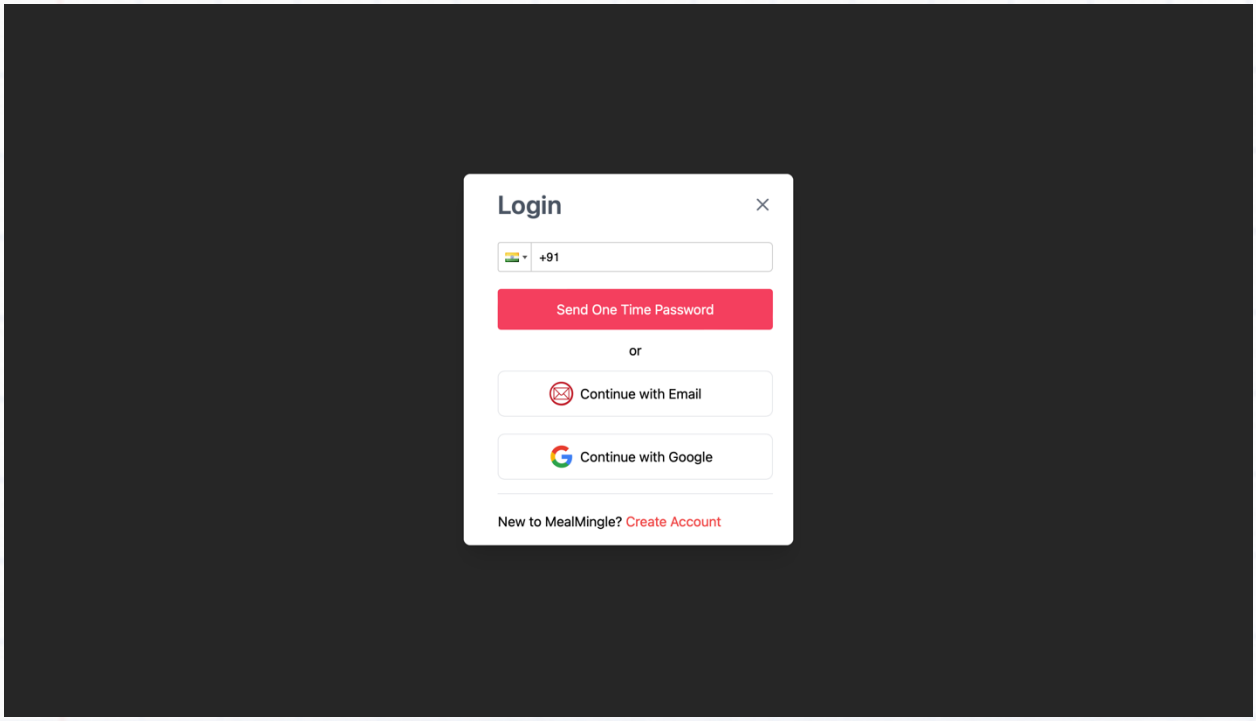
```
return (

<div className="relative z-10" aria-labelledby="modal-title" role="dialog"

aria-modal="true">

<ToastContainer />

<div className="fixed inset-0 bg-black bg-opacity-85 transition-

opacity"></div>

<div className="fixed inset-0 z-10 w-screen overflow-y-auto">

<div className="flex min-h-full items-end justify-center p-4 text-center

sm:items-center sm:p-0">
```

```
<div className="relative transform overflow-hidden rounded-lg bg-white text-
left shadow-xl transition-all sm:my-8 sm:w-97 sm:max-w-lg">
<div className="bg-white px-4 pb-4 pt-5 sm:p-6 sm:pb-4">
<div className="sm:flex sm:items-start">
<div className="mt-3 text-center sm:ml-4 sm:mt-0 sm:text-left">
<div className='flex '>
<h3 className="text-3xl font-semibold leading-6 text-gray-600" id="modal-
title">Login</h3>
<button
onClick={handleClose}
className="text-gray-500"
>
<svg
xmlns="http://www.w3.org/2000/svg"
className="h-6 w-6 ml-56"
fill="none"
viewBox="0 0 24 24"
stroke="currentColor"
>
<path
strokeLinecap="round"
strokeLinejoin="round"
strokeWidth="2"
d="M6 18L18 6M6 6l12 12"
/>
```

```
</svg>
</button>
</div>
<div className='mt-8'>
<PhoneInput
country={'in'}
value={phone}
onChange={(phone) => setPhone("+" + phone)}
buttonStyle={{ backgroundColor: "white" }}
inputStyle={{ width: "100%" }} />
</div>
<button onClick={sendOtp} className="mt-5 mb-3 bg-rose-500 w-full h-12 text-
white py-2 px-4 rounded">
Send One Time Password
</button>
<div id="recaptcha"></div>
{phone && <input onChange={(e) => setOtp(e.target.value)} className="mb-3 mt-
3 outline-none border border-gray-300 text-gray-900 text-sm rounded-sm block
w-full p-2.5" placeholder="Enter OTP" required />}
{otp && <button onClick={verifyOtp} className="mt-5 mb-3 bg-rose-500 w-80 h-
12 text-white py-2 px-4 rounded">
Verify One Time Password
</button>}
{!phone && <div>
<div className='text-center mb-3'>or</div>
```

```
<Link to='/emailLogin'><div className='flex items-center text-center border
border-spacing-1 rounded-lg p-3'>
<img src={EmailIcon} alt='Email Icon' className='w-7 h-7 ml-12' />
<button className='ml-2'>Continue with Email</button>
</div>
</Link>
<div onClick={googleSignIn} className='mt-5 flex items-center text-center
border border-spacing-1 rounded-lg p-3'>
<img src={GoogleIcon} alt='Google Icon' className='w-7 h-7 ml-12' />
<button className='ml-2'>Continue with Google</button>
</div>
</div>}
<hr className='mt-4' />
<div className='text-base mt-5'>New to MealMingle? <Link to='/signup'><span
className='text-red-500'>Create Account</span></Link></div>
</div>
</div>
</div>
</div>
</div>
</div>
);
};
```

```
export default Login;
```



6.3. Email Login Component

- The EmailLogin Component is a React component designed for User Authentication in a web application.
- It provides an Email and Password login method using Firebase Authentication.
- The purpose of this component is to handle User Login processes via Email and Password and to facilitate user access to the application.

Create a new file inside the components folder and name it EmailLogin.tsx.

1. Imports

- **React**: Library for building the component.
- **useState**: React hook for managing state within the component.
- **useNavigate**: Hook for navigation within the app.
- **EmailInboxIcon**: Image used for the email login icon.
- **signInWithEmailAndPassword**: Firebase function for email/password authentication.
- **auth**: Firebase authentication instance.
- **ToastContainer & toast**: Libraries for showing notifications on the screen.

```
import React, { useState } from 'react';

import { Link, useNavigate } from 'react-router-dom';

import EmailInboxIcon from '../images/email-inbox-icon.png';

import { signInWithEmailAndPassword } from 'firebase/auth';

import { auth } from '../firebase/setup';

import { ToastContainer, toast } from 'react-toastify';

import 'react-toastify/dist/ReactToastify.css';
```

2. Component Definition

- **EmailLogin**: Functional component for the email login screen.
- **useNavigate**: Initializes navigation for redirecting users.
- **useState**: Initializes state variables for email, password, and error handling.

```
const EmailLogin = () => {

const navigate = useNavigate();

const [email, setEmail] = useState("");

const [password, setPassword] = useState("");

const [error, setError] = useState<string | null>(null);
```

3. User Email Login Function

- **userEmailLogin**: Handles email login through the backend API:
  1. Sends a POST request with email and password.
  2. Displays error or success messages based on the response.
  3. Stores the token in localStorage and redirects to the home page after 2 seconds.

```
const userEmailLogin = async () => {

const response = await fetch('http://localhost:8090/api/users/login/user', {

method: 'POST',
```

```
headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ userEmail: email, userPassword: password })

})

const data = await response.json();

console.log(data)

if (data.error != "") {

toast.error(data.message)

return Error('Some Error Occurred!')

}

else {

toast.success(data.message);

localStorage.setItem('token', data.data.token)

setTimeout(() => {

navigate("/");

}, 2000);

}

}
```

4. emailLogin Function

emailLogin: Manages user login with Firebase authentication:

- Calls the userEmailLogin function to handle backend authentication.
- Uses Firebase's signInWithEmailAndPassword to authenticate the user.
- Catches and displays any errors that occur during the process.

```
const emailLogin = async () => {

try {
```

```
userEmailLogin();

const userCredential = await signInWithEmailAndPassword(auth, email,

password);

const user = userCredential.user;

} catch (err: any) {

console.error(err);

toast.error(err.message || "An Unknown Error Occurred during Login");

}

}
```

5. handleClose Function

- **handleClose**: Closes the login modal and navigates back to the home page.

```
const handleClose = () => {

navigate('/');

};
```

6. Return JSX

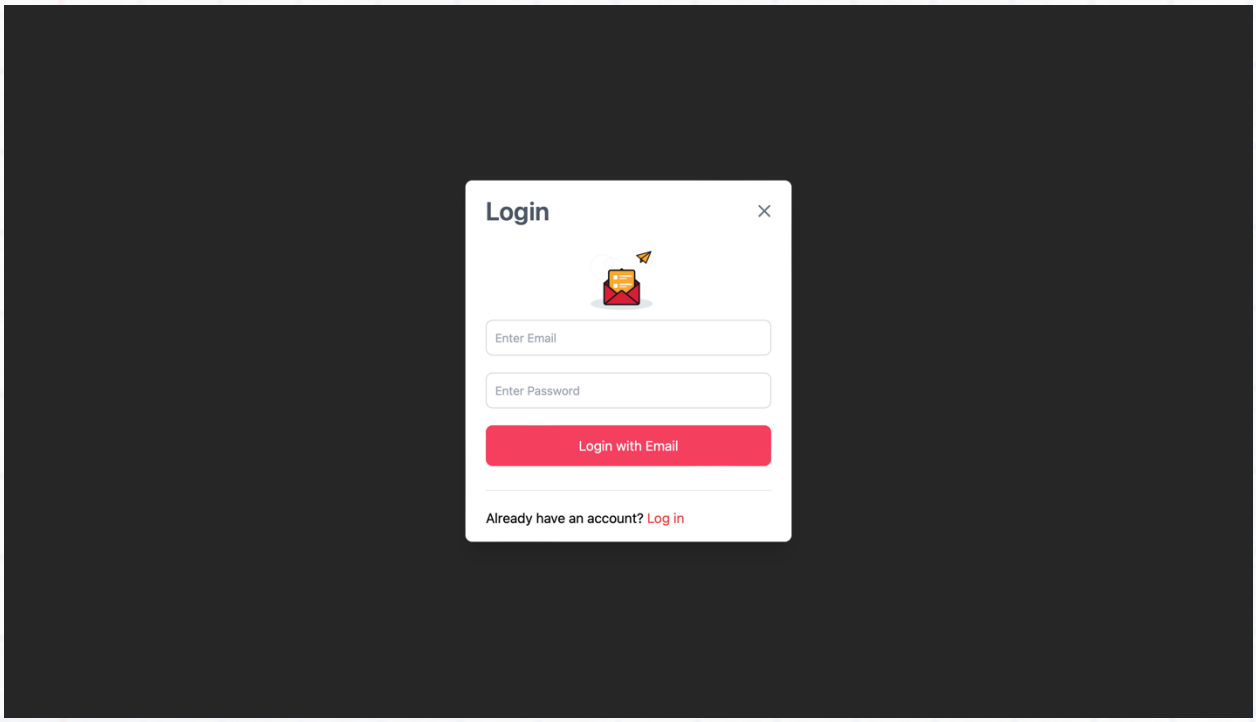
- **ToastContainer**: Displays notifications for success or error messages.
- **Modal Structure**: Provides a modal dialog layout with a dark overlay.
- **Email Input**: Allows users to enter their email address.
- **Password Input**: Allows users to enter their password.
- **Login Button**: Triggers the emailLogin function to handle the login process.
- **Error Display**: Shows an error message if there is a login error.
- **Alternative Navigation**: Provides a link for users to log in if they already have an account.

```
return (

<div className="relative z-10" aria-labelledby="modal-title" role="dialog"

aria-modal="true">
```

```
<ToastContainer />

<div className="fixed inset-0 bg-black bg-opacity-85 transition-

opacity"></div>

<div className="fixed inset-0 z-10 w-screen overflow-y-auto">

<div className="flex min-h-full items-end justify-center p-4 text-center

sm:items-center sm:p-0">

<div className="relative transform overflow-hidden rounded-lg bg-white text-

left shadow-xl transition-all sm:my-8 sm:w-96 sm:max-w-lg">

<div className="bg-white px-4 pb-4 pt-5 sm:p-6 sm:pb-4">

<div className='flex'>

<h3 className="text-3xl font-semibold leading-6 text-gray-600" id="modal-

title">Login</h3>

<button

onClick={handleClose}

className="text-gray-500"

>

<svg

xmlns="http://www.w3.org/2000/svg"

className="h-6 w-6 ml-60"

fill="none"

viewBox="0 0 24 24"

stroke="currentColor"

>

<path

strokeLinecap="round"
```

```
strokeLinejoin="round"
strokeWidth="2"
d="M6 18L18 6M6 6l12 12"
/>
</svg>
</button>
</div>
<img src={EmailInboxIcon} alt='Email Inbox Icon' className='w-24 h-24 mt-5
ml-28' />
<input onChange={(e) => setEmail(e.target.value)} className="outline-none
border border-gray-300 text-gray-900 text-sm rounded-lg block w-full p-2.5"
placeholder="Enter Email" required />
<input type='password' onChange={(e) => setPassword(e.target.value)}
className="mt-5 outline-none border border-gray-300 text-gray-900 text-sm
rounded-lg block w-full p-2.5" placeholder="Enter Password" required />
<button onClick={emailLogin} className="mt-5 mb-3 bg-rose-500 w-full h-12
text-white py-2 px-4 rounded-lg">
Login with Email
</button>
{error && <div className="text-red-500 mt-2">{error}</div>}
<hr className='mt-4' />
<div className='text-base mt-5'>Already have an account? <Link
to='/login'><span className='text-red-500'>Log in</span></Link></div>
</div>
</div>
```

```
</div>

</div>

</div>

);

}


export default EmailLogin;
```



6.4. Home Component

- The Home Component is a React component that serves as the landing page for a web application called "Meal Mingle".
- It provides an overview of the application, which focuses on discovering and exploring restaurants.
- The component displays a banner, login and signup buttons, and a list of restaurants.
- Users can navigate to specific restaurant details by clicking on the restaurant links.

Create a new file inside the components folder and name it Home.tsx.

1. Imports

- **React**: Library used for building the component.
- **HomeBanner**: Background image for the home page banner.
- **Link**: Component from react-router-dom for navigation.
- **RightArrowIcon**: Icon used for navigation arrows in the location list.
- **RestaurantData**: JSON file containing restaurant information.
- **isAuthenticated**: Utility function to check if a user is authenticated.

```
import React from 'react';

import HomeBanner from '../images/home-banner.png';

import { Link } from 'react-router-dom';

import RightArrowIcon from '../images/right-arrow-icon.png';

import RestaurantData from '../restaurants.json';

import { isAuthenticated } from '../utils/authUtils';
```

2. Component Definition

- **Home**: Functional component representing the home page.
- **restaurants**: Variable holding data.

```
const Home = () => {

const restaurants = RestaurantData;
```

3. Function

- **handleCitySelection**: Saves the selected city to localStorage for future use.

```
const handleCitySelection = (city: string) => {

localStorage.setItem('location', city);

};
```

4. Return JSX

1. Banner Section:

- **Style**: Uses inline styles to set a gradient overlay and background image for the home page.
- **Conditional Rendering**: If the user is not authenticated, shows buttons for adding a restaurant, logging in, and signing up.

2. Main Heading and Subheading:

- **Title**: Displays the main title "Meal Mingle" and a subheading describing the service.

3. Description:

- **Text**: Provides a description of the service, highlighting various restaurant options and features.

4. Restaurant Locations:

- **Grid Layout**: Uses a grid to display restaurant locations. Each location is a clickable link that leads to the main page (/main).
- **Location Handling**: When a location is clicked, it saves the selected city to localStorage and passes the city as state to the next page.
- **Location Card**: Each card displays the city name and an arrow icon for navigation.

```
return (

<>

<div

style={{

backgroundImage: `linear-gradient(rgba(0, 0, 0, 0.3), rgba(0, 0, 0, 0.7)),

url(${HomeBanner})`,

backgroundRepeat: 'no-repeat',

backgroundSize: 'cover',

minHeight: '100vh',

display: 'flex',

flexDirection: 'column',
```

```
justifyContent: 'center',
alignItems: 'center',
position: 'relative'
}}
className='text-white text-center'
>
{!isAuthenticated() && (
<div style={{ position: 'absolute', top: '40px', right: '20px' }}>
<Link to='/partner-with-us'>
<button className='text-base px-4 py-2 text-white rounded-md border border-
gray-300 mr-10'>
Add Restaurant
</button>
</Link>
<Link to='/login'>
<button className='text-base px-4 py-2 text-white rounded-md border border-
gray-300 mr-10'>
Log In
</button>
</Link>
<Link to='/signup'>
<button className='text-base px-4 py-2 text-white rounded-md border border-
gray-300 mr-40'>
Sign Up
</button>
```

```
</Link>

</div>

)}

<h1 className='text-6xl font-extrabold italic'>Meal Mingle</h1>

<h2 className='text-4xl mt-5'>

Mingle with Deliciousness <br />

from India

</h2>

</div>

<div className='text-4xl text-center mt-8'>Explore All the Locations</div>

<div className='text-xl text-gray-600 text-center mt-4'>

From swanky upscale restaurants to the cosiest hidden gems serving the most

incredible food, <br />

MealMingle covers it all. Explore menus, and millions of restaurant photos

and reviews from users <br />

just like you, to find your next great meal.

</div>

<div className='grid grid-cols-3 gap-y-6 mt-6'>

{restaurants.map((restaurant) => (

<Link key={restaurant.restaurantId} to='/main' state={{ city:

restaurant.restaurantAddress?.city }} onClick={() =>

handleCitySelection(restaurant.restaurantAddress?.city)}>

<div className='flex justify-between items-center border border-spacing-1

shadow-lg w-80 p-5 rounded-lg mx-auto mt-6'>

<div className='text-xl'>{restaurant.restaurantAddress?.city}</div>
```

```
<img src={RightArrowIcon} alt='Right Arrow Icon' className='w-4 h-4' />

</div>

</Link>

))}

</div>

</>

);

};


export default Home;
```

JSON Data Structure

The restaurants.json file contains an array of restaurant objects with detailed information about each restaurant. Here's a sample structure:

Note: This JSON file was only used before project integration.

```
[

{

"restaurantId": "1",

"restaurantName": "The Pizza Kings",

"restaurantAddress": {

"streetNumber": "12/92",

"streetName": "Babar Pur, Geeta Colony",

"city": "Delhi",

"country": "India"

},

"restaurantRating": 3.6,
```

```
"restaurantMinimumOrderAmount": 3500,
"restaurantDiscountPercentage": 5,
"restaurantAvailability": true,
"restaurantImageUrl": "restaurant_1.png",
"restaurantOperationDays": "Mon-Sun",
"restaurantOperationHours": "10:00AM-11:00PM",
"restaurantPhoneNumber": 7978654356,
"restaurantItems": [
{
"restaurantItemId": "1_1",
"restaurantItemName": "Margherita Pizza",
"restaurantItemPrice": 390,
"restaurantItemCategory": "Pizza",
"restaurantItemImageUrl": "restaurant_1_menu_1.png",
"restaurantItemCuisineType": "North Indian",
"restaurantItemVeg": true
}
]
},
{
"restaurantId": "2",
"restaurantName": "Punjab Depot",
"restaurantAddress": {
"streetNumber": "456",
"streetName": "Rose Street",
```

```
"city": "Mumbai",
"country": "India"
},
"restaurantRating": 4.7,
"restaurantMinimumOrderAmount": 6500,
"restaurantDiscountPercentage": 15,
"restaurantAvailability": true,
"restaurantImageUrl": "restaurant_2.png",
"restaurantOperationDays": "Mon-Wed",
"restaurantOperationHours": "09:00AM-06:00PM",
"restaurantPhoneNumber": 8978654367,
"restaurantItems": [
{
"restaurantItemId": "2_1",
"restaurantItemName": "Paneer Tikka Burger",
"restaurantItemPrice": 400,
"restaurantItemCategory": "Burger",
"restaurantItemImageUrl": "restaurant_2_menu_1.png",
"restaurantItemCuisineType": "North Indian",
"restaurantItemVeg": true
},
{
"restaurantItemId": "2_2",
"restaurantItemName": "Punjabi Chilli Garlic Noodles",
"restaurantItemPrice": 350,
```

```
"restaurantItemCategory": "Noodles",
"restaurantItemImageUrl": "restaurant_2_menu_2.png",
"restaurantItemCuisineType": "North Indian",
"restaurantItemVeg": true
}
]
},
{
"restaurantId": "3",
"restaurantName": "Southern Spice",
"restaurantAddress": {
"streetNumber": "789",
"streetName": "Coconut Lane",
"city": "Chennai",
"country": "India"
},
"restaurantRating": 4.5,
"restaurantMinimumOrderAmount": 0,
"restaurantDiscountPercentage": 0,
"restaurantAvailability": true,
"restaurantImageUrl": "restaurant_3.png",
"restaurantOperationDays": "Mon-Thu",
"restaurantOperationHours": "11:00AM-11:00PM",
"restaurantPhoneNumber": 6956654340,
"restaurantItems": [
```

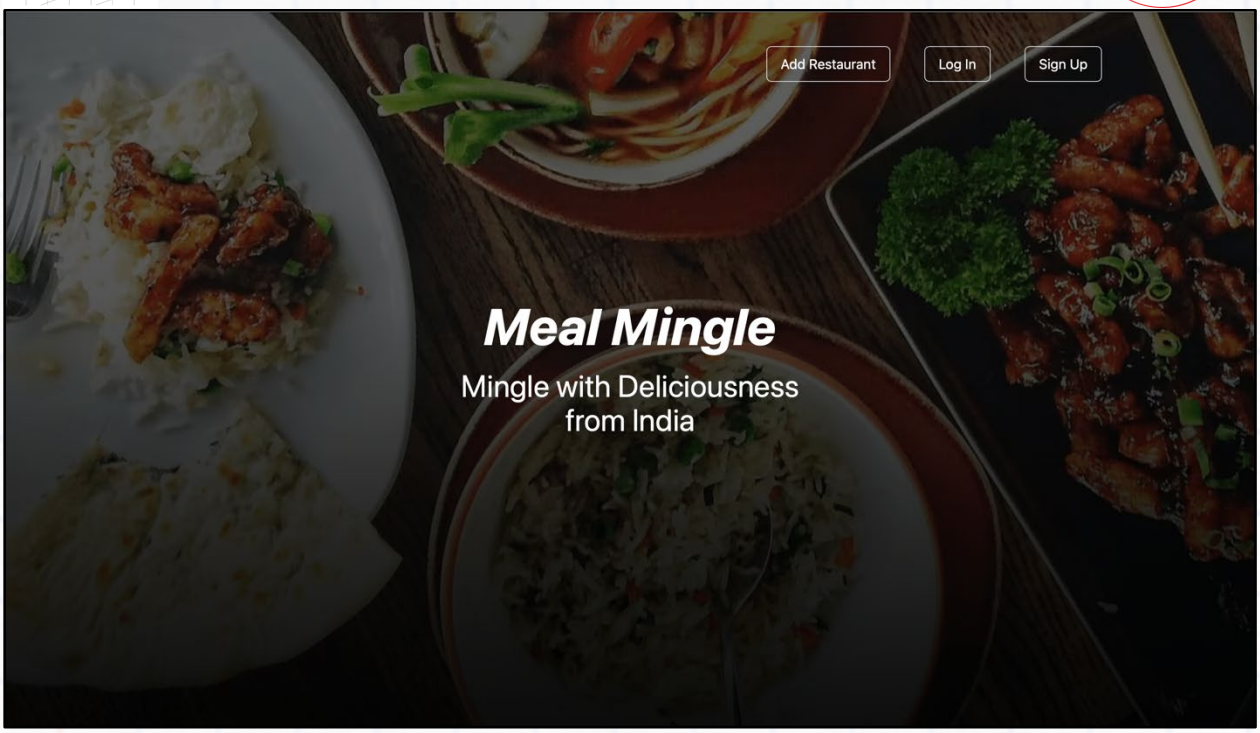
```
{
"restaurantItemId": "3_1",
"restaurantItemName": "Veg Biryani",
"restaurantItemPrice": 200,
"restaurantItemCategory": "Biryani",
"restaurantItemImageUrl": "restaurant_3_menu_1.png",
"restaurantItemCuisineType": "South Indian",
"restaurantItemVeg": true
},
{
"restaurantItemId": "3_2",
"restaurantItemName": "Veg Sandwich",
"restaurantItemPrice": 150,
"restaurantItemCategory": "Sandwich",
"restaurantItemImageUrl": "restaurant_3_menu_2.png",
"restaurantItemCuisineType": "South Indian",
"restaurantItemVeg": true
},
{
"restaurantItemId": "3_3",
"restaurantItemName": "Southern Spice Thali",
"restaurantItemPrice": 100,
"restaurantItemCategory": "Thali",
"restaurantItemImageUrl": "restaurant_3_menu_3.png",
"restaurantItemCuisineType": "South Indian",
```

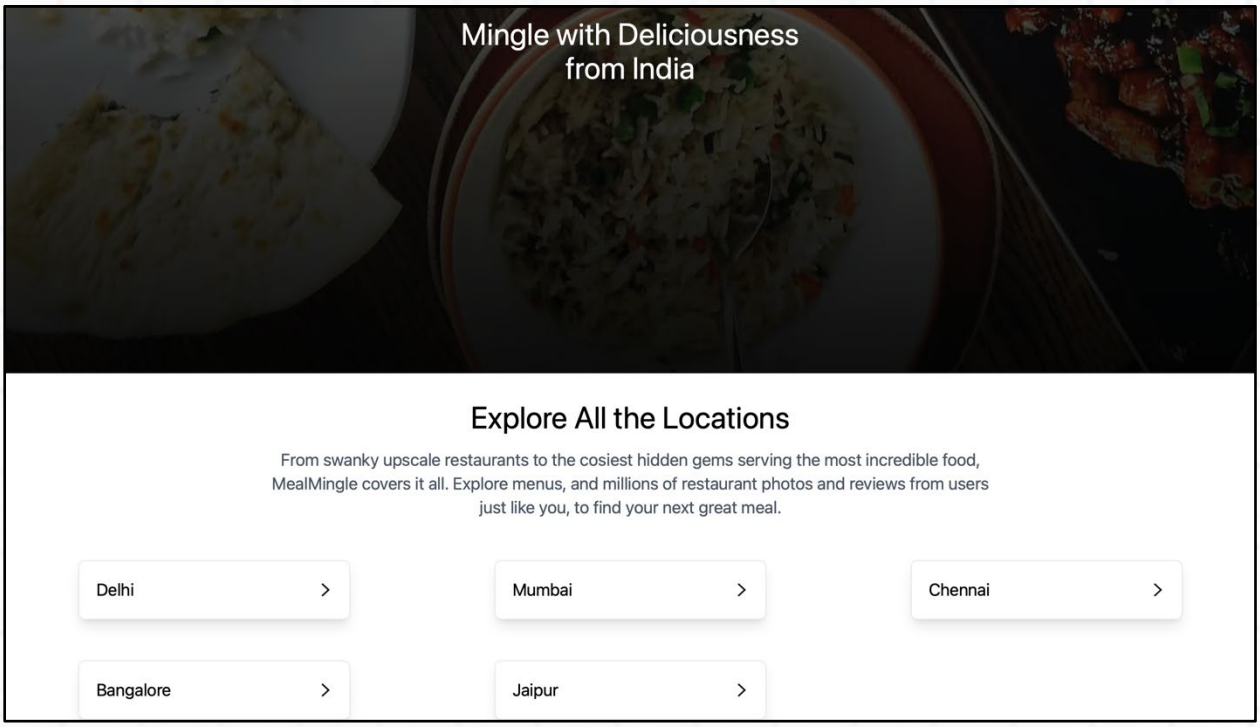
```
"restaurantItemVeg": true
}
]
},
{
"restaurantId": "4",
"restaurantName": "Green Delight",
"restaurantAddress": {
"streetNumber": "123",
"streetName": "Leafy Road",
"city": "Bangalore",
"country": "India"
},
"restaurantRating": 4.3,
"restaurantMinimumOrderAmount": 5500,
"restaurantDiscountPercentage": 20,
"restaurantAvailability": true,
"restaurantImageUrl": "restaurant_4.png",
"restaurantOperationDays": "Mon-Sat",
"restaurantOperationHours": "01:00AM-04:00PM",
"restaurantPhoneNumber": 7956678345,
"restaurantItems": [
{
"restaurantItemId": "4_1",
"restaurantItemName": "Veg Biryani",
```

```
"restaurantItemPrice": 300,
"restaurantItemCategory": "Biryani",
"restaurantItemImageUrl": "restaurant_4_menu_1.png",
"restaurantItemCuisineType": "South Indian",
"restaurantItemVeg": true
},
{
"restaurantItemId": "4_2",
"restaurantItemName": "Aloo Gobi",
"restaurantItemPrice": 250,
"restaurantItemCategory": "Thali",
"restaurantItemImageUrl": "restaurant_4_menu_2.png",
"restaurantItemCuisineType": "North Indian",
"restaurantItemVeg": true
},
{
"restaurantItemId": "4_3",
"restaurantItemName": "Harvest Bowl Salad",
"restaurantItemPrice": 200,
"restaurantItemCategory": "Salad",
"restaurantItemImageUrl": "restaurant_4_menu_3.png",
"restaurantItemCuisineType": "South Indian",
"restaurantItemVeg": true
}
]
```

```
},
{
"restaurantId": "5",
"restaurantName": "Rajasthani Thali",
"restaurantAddress": {
"streetNumber": "456",
"streetName": "Heritage Street",
"city": "Jaipur",
"country": "India"
},
"restaurantRating": 4.6,
"restaurantMinimumOrderAmount": 0,
"restaurantDiscountPercentage": 0,
"restaurantAvailability": true,
"restaurantImageUrl": "restaurant_5.png",
"restaurantOperationDays": "Mon-Sat",
"restaurantOperationHours": "10:00AM-06:00PM",
"restaurantPhoneNumber": 8706721345,
"restaurantItems": [
{
"restaurantItemId": "5_1",
"restaurantItemName": "Gatte ki Sabzi",
"restaurantItemPrice": 280,
"restaurantItemCategory": "Thali",
"restaurantItemImageUrl": "restaurant_5_menu_1.png",
```

```
"restaurantItemCuisineType": "North Indian",
"restaurantItemVeg": true
},
{
"restaurantItemId": "5_2",
"restaurantItemName": "Ghevar",
"restaurantItemPrice": 220,
"restaurantItemCategory": "Dessert",
"restaurantItemImageUrl": "restaurant_5_menu_2.png",
"restaurantItemCuisineType": "North Indian",
"restaurantItemVeg": true
}
]
}
]
```





Clicking on any restaurant location (say Delhi) redirects to the main page of the application.

6.5. Main Component

- The Main Component is a React component that serves as the primary interface for viewing and interacting with the restaurant listings in the "Meal Mingle" web application.
- It provides functionality for filtering, searching, and displaying restaurant information.

Create a new file inside the components folder and name it Main.tsx.

1. Imports

- **React**: Library used for building the component.
- **useLocation**: Hook from react-router-dom to access the current location object.
- **RestaurantData**: JSON file containing restaurant information.
- **Menubar, Navbar, RestaurantFilters, Restaurant**: Custom components used in the Main component.
- **toast**: Function from react-toastify to display notifications.

```
import React, { useState, useEffect } from 'react';

import { useLocation } from 'react-router-dom';

import RestaurantData from '../restaurants.json';

import Menubar from './Menubar';

import Navbar from './Navbar';

import RestaurantFilters from './RestaurantFilters';

import Restaurant from './Restaurant';

import { toast } from 'react-toastify';
```

2. Component Definition

- **Main**: Functional component representing the main page.
- **location**: Holds the current location object.
- **restaurants**: State to store the fetched restaurants data.
- **filteredRestaurants**: State to store the filtered restaurants data based on applied filters.
- **restaurantFilters**: State to store the status of applied filters (rating and offers).

```
const Main = () => {

const location = useLocation();

const [restaurants, setRestaurants] = useState([]);

const [filteredRestaurants, setFilteredRestaurants] =

useState(RestaurantData);

const [restaurantFilters, setRestaurantFilters] = useState({

rating: false,

offers: false,

});
```

3. Functions

- **fetchRestaurants**: Fetches restaurant data from the API based on the selected city. Displays an error message if there's an error and sets the fetched data to the restaurants state.

```
const fetchRestaurants = async () => {

const response = await

fetch(`http://localhost:8091/api/restaurants/city/${location.state.city}`)

const data = await response.json();

if (data.error != "") {

toast.error(data.message);

}

setRestaurants(data.data.restaurants);

}
```

useEffect for fetching restaurants

- **useEffect**: Calls fetchRestaurants when the component mounts.

```
useEffect(() => {
```

```
fetchRestaurants();

}, [])
```

useEffect for applying filters

useEffect: Calls applyRestaurantFilters whenever the search parameters in the URL change.

```
useEffect(() => {

applyRestaurantFilters(new URLSearchParams(location.search));

}, [location.search]);
```

applyRestaurantFilters

It filters the restaurant data based on the city, rating, and offers. Updates the restaurantFilters state to reflect the applied filters.

```
const applyRestaurantFilters = (params: URLSearchParams) => {

let filtered = RestaurantData;

const city = params.get('city') || location.state?.city;


if (city) {

filtered = filtered.filter(restaurant =>

restaurant.restaurantAddress.city.toLowerCase() === city.toLowerCase());

}


if (params.get('rating')) {

filtered = filtered.filter(restaurant => restaurant.restaurantRating >= 4.0);

setRestaurantFilters(prev => ({ ...prev, rating: true }));

} else {
```

```
setRestaurantFilters(prev => ({ ...prev, rating: false }));
}

if (params.get('offers')) {
filtered = filtered.filter(restaurant =>
restaurant.restaurantDiscountPercentage > 0);
setRestaurantFilters(prev => ({ ...prev, offers: true }));
} else {
setRestaurantFilters(prev => ({ ...prev, offers: false }));
}

setFilteredRestaurants(filtered);
};
```

handleSearch

It filters the restaurants based on the search query and selected city. Updates the filteredRestaurants state with the filtered data.

```
const handleSearch = (query: string) => {
const city = location.state?.city || localStorage.getItem('location');
const filtered = RestaurantData.filter(restaurant =>
restaurant.restaurantName.toLowerCase().startsWith(query.toLowerCase()) &&
(!city || restaurant.restaurantAddress.city.toLowerCase() ===
city.toLowerCase())
);
setFilteredRestaurants(filtered);
};
```

4. Return JSX

1. Navbar:

- **Props**: Passes the selected city and handleSearch function to the Navbar component.
- **Functionality**: Allows users to search for restaurants and displays the selected city.

2. RestaurantFilters:

- **Props**: Passes applyRestaurantFilters and restaurantFilters to the RestaurantFilters component.
- **Functionality**: Allows users to apply filters (rating and offers) to the restaurant list.

3. Menubar:

- **Component**: Displays a menu bar for navigation.

4. Restaurant:

- **Props**: Passes the fetched restaurants and the selected city to the Restaurant component.
- **Functionality**: Displays the list of restaurants based on the selected city and applied filters.

```
return (

<div>

<Navbar city={location.state?.city} onSearch={handleSearch} />

<RestaurantFilters applyRestaurantFilters={applyRestaurantFilters}

restaurantFilters={restaurantFilters} />

<Menubar />

<Restaurant restaurant={restaurants} city={location.state?.city} />

</div>

);

};
```

```
export default Main;
```

6.6. Navbar Component

- The Navbar Component is a versatile navigation bar for the MealMingle application.
- It manages User Authentication, Location Input, Search Functionality, and links to different parts of the app, such as the Cart and Order History.

Create a new file inside the components folder and name it Navbar.tsx.

Now, install the following dependencies:

1. **React Avatar**: For displaying user avatars.

```
npm install react-avatar
```

1. Imports
The component begins with importing necessary modules and assets, including images, Firebase authentication methods, React Router components, CartContext, and Toast notifications.

```
import React, { useEffect, useState } from 'react';

import SearchIcon from '../images/search-icon.png';

import Avatar from 'react-avatar';

import { auth } from '../firebase/setup';

import { onAuthStateChanged, signOut } from 'firebase/auth';

import { Link } from 'react-router-dom';

import Location from '../images/location.png';

import DropDownIcon from '../images/drop-down-icon.png';

import LogoutIcon from '../images/logout-icon.png';

import ShoppingCartIcon from '../images/shopping-cart-icon.png';

import { useCart } from '../context/CartContext';
```

```
import { toast, ToastContainer } from 'react-toastify';

import 'react-toastify/dist/ReactToastify.css';

import { useNavigate } from 'react-router-dom';
```

2. Props Interface

The cityProp interface defines the props that can be passed to the Navbar component, including city (optional) and onSearch (optional function).

```
interface cityProp {

city?: string;

onSearch?: (query: string) => void;

}
```

3. Component Definition

The Navbar Component function takes city and onSearch as props.

```
const Navbar = ({ city, onSearch }: cityProp) => {
```

4. State Variables

- authStore: Holds the user authentication data.
- searchQuery: Stores the search input value.
- location: Stores the location input value, initialized from local storage.

```
const [authStore, setAuthStore] = useState<any>({});

const { cart, getTotalQuantity, clearCart } = useCart();

const navigate = useNavigate();

const [searchQuery, setSearchQuery] = useState('');

const [location, setLocation] = useState(() => {

return localStorage.getItem('location') || 'Location';

});
```

5. Effect Hooks

- useEffect to listen to authentication state changes and update authStore accordingly.
- Another useEffect to update location state based on the city prop.

```
useEffect(() => {

const unsubscribe = onAuthStateChanged(auth, (user) => {

setAuthStore(user || {});

});


return () => unsubscribe();

}, []);


useEffect(() => {

if (city) {

setLocation(city);

localStorage.setItem('location', city);

}

else {

setLocation(localStorage.getItem('location') || 'Location');

}

}, [city]);
```

6. Logout Function

Defines an async function to handle user logout. It signs out the user, clears the cart, shows a toast notification, and navigates to the home page.

```
const logout = async () => {
```

```
try {
await signOut(auth);
toast.success('Logged Out Successfully!');
clearCart();
setTimeout(() => {
navigate('/');
}, 2000);
} catch (err) {
console.error(err);
}
localStorage.clear();
};
```

7. Event Handlers

- **handleSearchChange**: Updates the search query state and calls the onSearch function if provided.
- **handleLocationChange**: Updates the location state and local storage with the new location value.

```
const handleSearchChange = (e: React.ChangeEvent<HTMLInputElement>) => {
setSearchQuery(e.target.value);
if (onSearch) {
onSearch(e.target.value);
}
};

const handleLocationChange = (e: React.ChangeEvent<HTMLInputElement>) => {
const newLocation = e.target.value;
```

```
setLocation(newLocation);

localStorage.setItem('location', newLocation);

};
```

8. Return JSX

- The main JSX structure of the component, containing various elements such as the app logo, location input, search input, user profile, cart, order history, login/signup buttons, and logout button.
- Conditional rendering is used to show different elements based on whether the user is authenticated (auth.currentUser).

```
return (

<>

<ToastContainer />

<div className='flex'>

<Link to='/'><h1 className='text-3xl font-extrabold italic ml-

20'>MealMingle</h1></Link>

{auth.currentUser ? (<div className='ml-6 shadow-lg flex items-center border

border-gray-300 w-6/12 rounded-lg p-3 h-12'>

<img src={Location} alt='Location Icon' className='w-7 h-7 ml-2' />

<input className="outline-none text-gray-900 text-sm block w-40 p-2.5"

placeholder='Location' value={location} onChange={handleLocationChange}

required />

<img src={DropDownIcon} alt='Drop Down Icon' className='w-5 h-5 ml-5' />

<div className='ml-3 text-gray-400'>|</div>

<img src={SearchIcon} alt='Search Icon' className='w-6 h-6 ml-5' />

<input className="outline-none text-gray-900 text-sm block w-96 p-2.5"

placeholder="Search for Restaurant" value={searchQuery}
```

```
onChange={handleSearchChange} required />
</div>) : (<div className='ml-6 shadow-lg flex items-center border border-
gray-300 w-7/12 rounded-lg p-3 h-12'>
<img src={Location} alt='Location Icon' className='w-7 h-7 ml-2' />
<input className="outline-none text-gray-900 text-sm block w-40 p-2.5"
placeholder='Location' value={location} onChange={handleLocationChange}
required />
<img src={DropDownIcon} alt='Drop Down Icon' className='w-5 h-5 ml-5' />
<div className='ml-3 text-gray-400'>|</div>
<img src={SearchIcon} alt='Search Icon' className='w-6 h-6 ml-5' />
<input className="outline-none text-gray-900 text-sm block w-96 p-2.5"
placeholder="Search for Restaurant" value={searchQuery}
onChange={handleSearchChange} required />
</div>)}
<div className='flex items-center'>
{authStore?.photoURL ? (
<img src={authStore?.photoURL} alt='User Pic' className='w-12 h-12 ml-5
rounded-full' />
) : authStore?.displayName ? (
<Avatar name={authStore?.displayName} round={true} size='40' className='ml-
72' />
) : null}
<div className='ml-2'>
{authStore?.displayName ? authStore.displayName : authStore?.email ?
authStore.email : ''}
```

```
</div>
{authStore?.phoneNumber && <div className="text-gray-600 text-
lg">{authStore.phoneNumber}</div>}
{!auth.currentUser?.email && !auth.currentUser?.phoneNumber && (
<Link to='/login'>
<button
className={`px-4 py-2 rounded-md border bg-white text-black border-gray-300
shadow-md ml-10 cursor-pointer`}>
Login
</button>
</Link>
)}
{!auth.currentUser?.email && !auth.currentUser?.phoneNumber && (
<Link to='/signup'>
<button
className={`px-4 py-2 rounded-md border bg-white text-black border-gray-300
shadow-md ml-10 cursor-pointer`}>
Sign Up
</button>
</Link>
)}
{auth.currentUser && (
<Link to='/cart' className='relative'>
<img src={ShoppingCartIcon} alt='Shopping Cart Icon' className='ml-4 shadow-
lg p-2 rounded-xl text-gray-600 cursor-pointer w-10 h-10' />
```

```
{getTotalQuantity() > 0 && (
<span className='absolute top-0 right-0 inline-flex items-center justify-
center px-2 py-1 text-xs font-bold leading-none text-red-100 bg-red-600
rounded-full'>
{getTotalQuantity()}
</span>
)}
</Link>
)}
{auth.currentUser && (
<Link to='/order-history' className='relative'>
<div className='ml-6 shadow-lg p-2 rounded-xl text-black cursor-pointer w-15
h-15'>My Orders</div>
</Link>
)}
{auth.currentUser && (
<img onClick={logout} src={LogoutIcon} alt='Logout Icon' className='ml-6
shadow-lg p-2 rounded-xl text-gray-600 cursor-pointer w-10 h-10' />
)}
</div>
</div>
</>
);
};
```

```
export default Navbar;
```

For Unauthenticated Users:

6.7. RestaurantFilters Component

Note: This component is currently not integrated.

- It allows users to filter restaurants based on certain criteria like rating and offers.
- It uses React's state and effect hooks along with React Router's hooks to manage and apply these filters.

Create a new file inside the components folder and name it RestaurantFilters.tsx.

1. Imports
   - useState, useEffect from React.
   - useLocation, useNavigate from React Router for handling navigation and location.

```
import React, { useState, useEffect } from 'react';

import { useLocation, useNavigate } from 'react-router-dom';
```

2. Defining Props Interface

- **applyRestaurantFilters**: A function passed as a prop to apply filters.
- **restaurantFilters**: An object containing the initial state of filters.

```
interface RestaurantFiltersProps {

applyRestaurantFilters: (params: URLSearchParams) => void;

restaurantFilters: {

rating: boolean;

offers: boolean;

};
```

```
}
```

3. Component Definition

RestaurantFilters is defined as a functional component that takes applyRestaurantFilters and restaurantFiltersas props.

```
const RestaurantFilters: React.FC<RestaurantFiltersProps> = ({

applyRestaurantFilters, restaurantFilters }) => {

const navigate = useNavigate();

const location = useLocation();
```

4. State Variables

rating4Plus and offers are state variables initialized to false.

```
const [rating4Plus, setRating4Plus] = useState(false);

const [offers, setOffers] = useState(false);
```

5. Using useEffect

This hook is used to set the state based on the current URL query parameters whenever the URL changes.

```
useEffect(() => {

const params = new URLSearchParams(location.search);

setRating4Plus(params.get('rating') === 'true');

setOffers(params.get('offers') === 'true');

}, [location.search]);
```

6. Function to Apply Filters and Navigate

applyRestaurantFiltersAndNavigate constructs a URL with the selected filters and navigates to the new URL.

```
const applyRestaurantFiltersAndNavigate = () => {

const params = new URLSearchParams(location.search);


const city = location.state?.city || localStorage.getItem('location');

if (city) {

params.set('city', city);

}


if (rating4Plus) {

params.set('rating', 'true');

} else {

params.delete('rating');

}

if (offers) {

params.set('offers', 'true');

} else {

params.delete('offers');

}


navigate(`/restaurants/filter?${params.toString()}`, { state: { city } });

applyRestaurantFilters(params);

};
```

7. Handle Button Clicks

handleRatingClick and handleOffersClick toggle the filter states.

```
const handleRatingClick = () => {
```

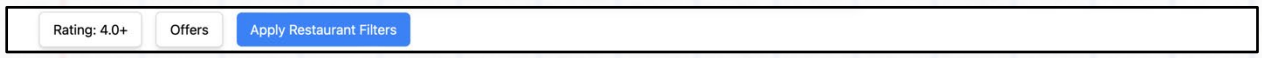
```
const newValue = !rating4Plus;

setRating4Plus(newValue);

};


const handleOffersClick = () => {

const newValue = !offers;

setOffers(newValue);

};
```

8. Return JSX

The component returns a div with buttons to toggle filters and apply them. The buttons' styles change based on whether the filter is active.

```
return (

<div className="bg-white p-4 text-black ml-5">

<div className="flex items-center space-x-4">

<button

className={`px-4 py-2 rounded-md border ${rating4Plus ? 'bg-gray-300 text-

black' : 'bg-white text-black'} border-gray-300 shadow-md`}

onClick={handleRatingClick}

>

Rating: 4.0+

</button>

<button

className={`px-4 py-2 rounded-md border ${offers ? 'bg-gray-300 text-black' :

'bg-white text-black'} border-gray-300 shadow-md`}

onClick={handleOffersClick}
```

```
>

Offers

</button>

<button

className="px-4 py-2 rounded-md bg-blue-500 text-white"

onClick={applyRestaurantFiltersAndNavigate}

>

Apply Restaurant Filters

</button>

</div>

</div>

);

};


export default RestaurantFilters;
```



6.8. Menubar Component

- The Menubar Component displays a carousel of dish categories that users can click on to navigate to specific category pages.
- It uses the react-slick library for the carousel functionality and React Router for navigation.

Create a new file inside the components folder and name it Menubar.tsx.

Now, install the following dependencies:

1. **@types/react-slick**: This package provides TypeScript type definitions for react-slick, which helps with type checking and IntelliSense in your TypeScript project. It's essential for TypeScript projects using react-slick.

```
npm install --save-dev @types/react-slick
```

2. **slick-carousel**: This is the main package for Slick Carousel, which you already need for the carousel functionality.

```
npm install slick-carousel
```

1. Imports

- React and useNavigate from React Router.
- Slider from react-slick for the carousel.
- CSS files for styling the carousel.
- Custom arrow components for the carousel.
- Images for the dish categories.

```
import React from 'react';

import { useNavigate } from 'react-router-dom';

import Slider from 'react-slick';

import 'slick-carousel/slick/slick.css';

import 'slick-carousel/slick/slick-theme.css';

import PreviousArrow from '../components/PreviousArrow';

import NextArrow from '../components/NextArrow';

import '../components/Slick-Arrows.css';


import Pizza from '../images/menu-1.png';

import Burger from '../images/menu-2.png';

import Noodles from '../images/menu-3.png';

import Pasta from '../images/menu-4.png';

import Sandwich from '../images/menu-5.png';

import Thali from '../images/menu-6.png';

import Dessert from '../images/menu-7.png';

import Salad from '../images/menu-8.png';
```

```
import Biryani from '../images/menu-9.png';

import Fish from '../images/menu-10.png';
```

2. Defining Dish Categories

An array dishes that contains objects representing each dish category with its name and image.

```
const dishes = [

{ name: 'Pizza', image: Pizza },

{ name: 'Burger', image: Burger },

{ name: 'Noodles', image: Noodles },

{ name: 'Pasta', image: Pasta },

{ name: 'Sandwich', image: Sandwich },

{ name: 'Thali', image: Thali },

{ name: 'Dessert', image: Dessert },

{ name: 'Salad', image: Salad },

{ name: 'Biryani', image: Biryani },

{ name: 'Fish', image: Fish },

];
```

3. Component Definition

Menubar is defined as a functional component.

```
const Menubar: React.FC = () => {

const navigate = useNavigate();
```

4. Handling Category Clicks

handleCategoryClick function sets a location in local storage and navigates to the category page.

```
const handleCategoryClick = (categoryName: string) => {

localStorage.setItem('location', 'Location');

navigate(`/category/${categoryName}`);

};
```

5. Carousel Settings

The settings for the react-slick carousel, including speed, number of slides to show, autoplay, and custom arrows.

```
const settings = {

dots: false,

infinite: true,

speed: 500,

slidesToShow: 7,

slidesToScroll: 1,

autoplay: false,

arrows: true,

prevArrow: <PreviousArrow />,

nextArrow: <NextArrow />,

};
```

6. Return JSX

The component returns a div containing the title and the Slider component. Each dish is displayed as an image with a label, and clicking on it navigates to the corresponding category page.

```
return (

<div className='bg-zinc-100 p-10'>

<div className='text-3xl'>Uniting You with Delicious Moments</div>
```

```
<Slider {...settings} className='mt-5'>

{dishes.map((dish, index) => (

<div key={index} className='text-center px-2 cursor-pointer' onClick={() =>

handleCategoryClick(dish.name)}>

<img src={dish.image} alt={dish.name} className='rounded-full w-40 h-40 mx-

auto' />

<p className='mt-2'>{dish.name}</p>

</div>

))}

</Slider>

</div>

);

};


export default Menubar;
```

Create three separate files inside components folder and name them: PreviousArrow.tsx, NextArrow.tsx and Slick-Arrows.css.

PreviousArrow Component

- **Purpose**: This component renders a custom button for the "previous" arrow in the carousel.
- **Props**: onClick is a function provided by react-slick that moves the carousel to the previous slide when the button is clicked.
- **Button Styling**: It uses slick-arrow and slick-prev CSS classes for styling, and text-black to set the text color to black. The button displays a less-than sign (&lt) as the arrow indicator.

```
import React from 'react';
```

```
const PreviousArrow = (props: any) => {

const { onClick } = props;

return (

<button onClick={onClick} className="slick-arrow slick-prev text-black" >

&lt

</button>

);

};


export default PreviousArrow;
```

NextArrow Component

- **Purpose**: This component renders a custom button for the "next" arrow in the carousel.
- **Props**: onClick is a function provided by react-slick that moves the carousel to the next slide when the button is clicked.
- **Button Styling**: It uses slick-arrow and slick-next CSS classes for styling. The button displays a greater-than sign (&gt) as the arrow indicator.

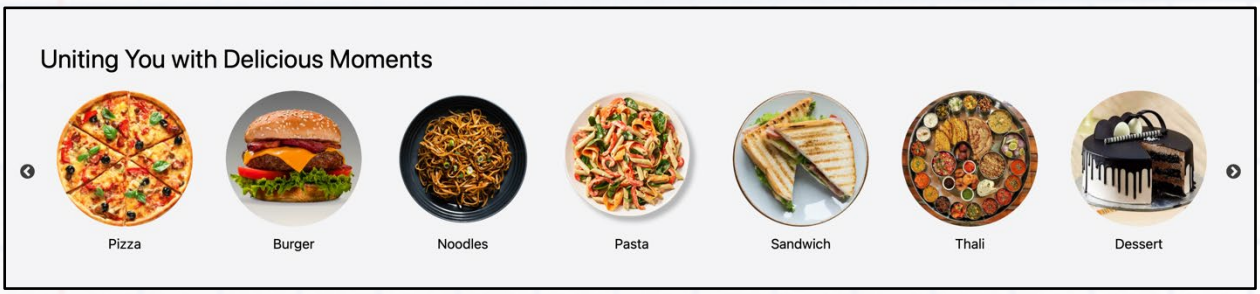
```
import React from 'react';


const NextArrow = (props: any) => {

const { onClick } = props;

return (

<button onClick={onClick} className="slick-arrow slick-next">

&gt

</button>

);
```

```
};


export default NextArrow;
```

Slick-Arrows.css (CSS for Arrows)

- **Purpose**: This CSS sets the color of the arrow icons in the carousel to black.
- **Classes Used**:
- .slick-prev::before: Targets the "previous" arrow.
- .slick-next::before: Targets the "next" arrow.
- **Color**: The color: black; rule ensures that the arrows are displayed in black.

```
.slick-prev::before,

.slick-next::before {

color: black;

}
```



6.9. Restaurant Component

- This component is part of a React application that displays a list of restaurants, shows their operational status, and allows users to navigate to a menu page or login page based on their authentication status.

Create a new file inside the components folder and name it Restaurant.tsx.

Now, install the following dependencies:

1. **React Icons**: For using the FaBan icon.

```
npm install react-icons
```

1. Imports

- **React**: Used to create components.
- **Link**: For navigation between pages.
- **isAuthenticated**: A utility function to check if the user is logged in.
- **FaBan**: An icon to show when a restaurant is closed.

```
import React from 'react';

import { Link } from 'react-router-dom';

import { isAuthenticated } from '../utils/authUtils';

import { FaBan } from 'react-icons/fa';
```

2. Interface Definition

RestaurantProp: Defines the types of props this component expects. It expects restaurant and city, but for simplicity, they are typed as any.

```
interface RestaurantProp {

restaurant: any,

city: any

}
```

3. Utility Functions

isRestaurantOpen: Checks if the restaurant is open. It converts days and hours into a format that can be compared with the current date and time.

```
const isRestaurantOpen = (operationDays: string, operationHours: string) => {

const daysMap: { [key: string]: number } = {

'Sun': 0,

'Mon': 1,
```

```
'Tue': 2,
'Wed': 3,
'Thu': 4,
'Fri': 5,
'Sat': 6
};

const currentDate = new Date();
const currentDay = currentDate.getDay();
const currentTime = currentDate.getHours() * 60 + currentDate.getMinutes();

const [startDay, endDay] = operationDays.split('-');
const startDayIndex = daysMap[startDay];
const endDayIndex = daysMap[endDay];

const withinOperatingDays = startDayIndex <= endDayIndex
? currentDay >= startDayIndex && currentDay <= endDayIndex
: currentDay >= startDayIndex || currentDay <= endDayIndex;

if (!withinOperatingDays) {
return false;
}

const [startTime, endTime] = operationHours.split('-').map(time => {
```

```
const [hours, minutes] = time.replace('AM', '').replace('PM',

'').split(':').map(Number);

const isPM = time.includes('PM');

const totalMinutes = (isPM ? hours + 12 : hours) * 60 + minutes;

return totalMinutes;

});


return currentTime >= startTime && currentTime <= endTime;

}
```

getOfferPhrase: Creates a string to show the discount offer of the restaurant.

```
const getOfferPhrase = (minOrderAmt: number, discountPercentage: number) => {

return `${discountPercentage}% OFF ABOVE ₹${minOrderAmt}`;

};
```

4. Component Definition

This is the main component function. It receives `restaurant` data and an optional `city` as props.

```
const Restaurant: React.FC<RestaurantProp> = ({ restaurant, city }) => {
```

5. Return JSX

- **Display Restaurant List**:
- The component shows a list of restaurants filtered by city.
- Each restaurant is displayed with an image, name, address, and rating.
- If the restaurant is open, its image is shown normally. If not open, its image is grayed out.
- If there is a discount, it displays the offer phrase on top of the restaurant image.
- If the restaurant is closed, it shows an overlay with a "ban" icon.

- **No Restaurants Message**: If there are no restaurants, it displays "No Restaurants Found!".

```
return (
        <div className='p-4 pl-20'>
            <div className='font-semibold text-3xl'>
                {city ? `Best Food in ${city}` : 'Best Food in Location'}
            </div>
            <div className='grid grid-cols-3 gap-4'>
                {restaurant
                    .filter((data: any) => !city ||
data.restaurantAddress.city.includes(city))
                    .map((data: any) => {
                        const isOpen =
isRestaurantOpen(data.restaurantOperationDays,
data.restaurantOperationHours);
                        const offerPhrase =
getOfferPhrase(data.restaurantMinimumOrderAmount,
data.restaurantDiscountPercentage);
                        return (
                            <div key={data.restaurantId}
className="relative max-w-xs rounded-xl overflow-hidden shadow-sm mt-12">
                                {isOpen ? (
                                    <Link to={isAuthenticated() ? '/menu'
: '/login'} state={{ data: data }}>
                                        <img className={`w-full rounded-
2xl h-60`} src={data.restaurantImageUrl} alt="Restaurant Image" />
```

```
                                    </Link>
                                ) : (
                                    <img className={`w-full rounded-2xl h-
60 filter grayscale`} src={`${data.restaurantImageUrl}`} alt="Restaurant
Image" />
                                )}
                                {data.restaurantDiscountPercentage > 0 &&
(
                                    <div className="absolute top-2 left-2
bg-blue-500 text-white font-semibold py-1 px-2 rounded-md">
                                        {offerPhrase}
                                    </div>
                                )}
                                <div className="py-4">
                                    <div className='flex justify-between
items-center'>
                                        <div className="font-semibold
text-xl mb-2">
                                            {data.restaurantName}
                                            <div className="text-sm text-
gray-600">{data.restaurantAddress.pincode},
{data.restaurantAddress.streetName}, {data.restaurantAddress.city}</div>
                                        </div>
```

```
                                    <div className={`text-white font-
semibold text-base rounded-md p-1 ${data.restaurantRating < 4.5 ? `bg-
green-600` : `bg-green-900`}`}>
                                        {data.restaurantRating}
                                    </div>
                                </div>
                                <div className={`font-semibold text-sm
${isOpen ? 'text-green-500' : 'text-red-500'}`}>
                                    {isOpen ? 'Open' : 'Closed'}
                                </div>
                            </div>
                            {!isOpen && (
                                <div className="absolute inset-0 bg-
black bg-opacity-50 flex items-center justify-center">
                                    <div className="w-16 h-16 rounded-
full flex items-center justify-center">
                                        <FaBan className="text-white
text-3xl" />
                                    </div>
                                </div>
                            )}
                        </div>
                    );
                })}
            {restaurant.length === 0 && (
```

```
                    <div className="text-center col-span-3 text-xl mt-24">
                        No Restaurants Found!
                    </div>
                )}
            </div>
        </div>
    );
}

export default Restaurant;
```

6.10. RestaurantsByFilter Component

Note: This component is currently not integrated.

- The RestaurantsByFilter Component is designed to display a list of restaurants that meet user-specified criteria.
- It allows users to filter restaurants by location, rating, and special offers, and provides a search functionality to find restaurants based on their names.

Create a new file inside the components folder and name it RestaurantsByFilter.tsx.

1. Imports
   - **React**: The core library for building the component.
   - **useState, useEffect**: Hooks from React to manage state and side effects.
   - **useLocation, useNavigate**: Hooks from react-router-dom to work with routing.
   - **RestaurantData**: Imported JSON data containing restaurant information.
   - **Navbar, RestaurantFilters, Menubar, Restaurant**: Components used in this component for different functionalities.

```
import React, { useState, useEffect } from 'react';

import { useLocation, useNavigate } from 'react-router-dom';
```

```
import RestaurantData from '../restaurants.json';

import Navbar from './Navbar';

import RestaurantFilters from './RestaurantFilters';

import Menubar from './Menubar';

import Restaurant from './Restaurant';
```

2. Component Definition

- **RestaurantsByFilter Component**: A functional component that manages restaurant filtering and display.
- **State Variables**:
- **filteredRestaurants**: Stores the list of restaurants after applying filters.
- **restaurantFilters**: Stores the status of filters (rating and offers).

```
const RestaurantsByFilter = () => {

const location = useLocation();

const navigate = useNavigate();

const [filteredRestaurants, setFilteredRestaurants] =

useState(RestaurantData);

const [restaurantFilters, setRestaurantFilters] = useState({

rating: false,

offers: false,

});
```

3. Effect Hook

useEffect: Runs applyRestaurantFilters whenever the query parameters in the URL change. This is triggered when the component mounts or when the URL query parameters update.

```
useEffect(() => {

applyRestaurantFilters(new URLSearchParams(location.search));

}, [location.search]);
```

4. Filtering Function

applyRestaurantFilters: Function to apply filters based on URL parameters:

- **City Filter**: Filters restaurants based on the city parameter.
- **Rating Filter**: Filters restaurants with a rating of 4.0 or higher.
- **Offers Filter**: Filters restaurants with available discounts.
- Updates the filteredRestaurants state with the filtered list.
- Updates restaurantFilters state to reflect the current filters applied.

```
const applyRestaurantFilters = (params: URLSearchParams) => {

let filtered = RestaurantData;

const city = params.get('city') || location.state?.city;


if (city) {

filtered = filtered.filter(restaurant =>

restaurant.restaurantAddress.city.toLowerCase() === city.toLowerCase());

}


if (params.get('rating')) {

filtered = filtered.filter(restaurant => restaurant.restaurantRating >= 4.0);

setRestaurantFilters(prev => ({ ...prev, rating: true }));

} else {

setRestaurantFilters(prev => ({ ...prev, rating: false }));

}


if (params.get('offers')) {

filtered = filtered.filter(restaurant =>

restaurant.restaurantDiscountPercentage > 0);

setRestaurantFilters(prev => ({ ...prev, offers: true }));
```

```
} else {

setRestaurantFilters(prev => ({ ...prev, offers: false }));

}



setFilteredRestaurants(filtered);

};
```

5. Search Handling

- **handleSearch**: Function to filter restaurants based on the search query:
  - Filters restaurants where the name starts with the search query.
  - Also considers the city filter if specified.

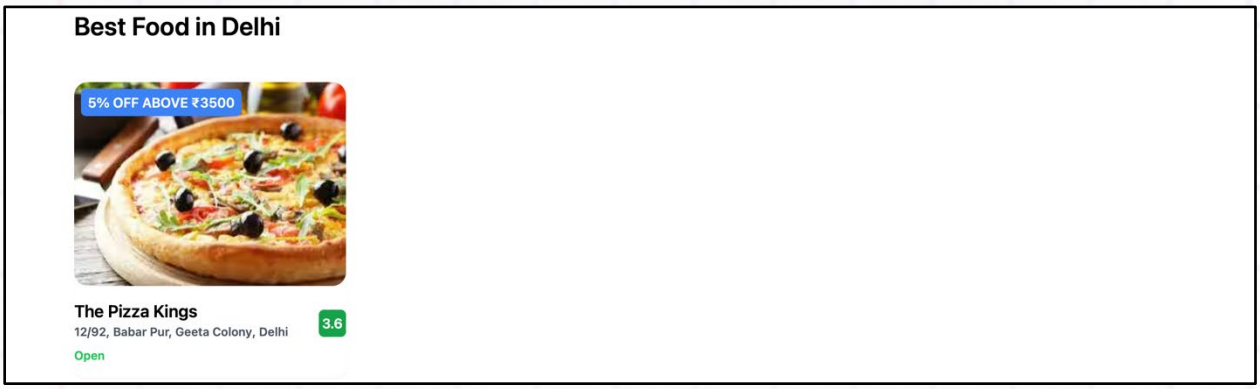
```
const handleSearch = (query: string) => {

const city = location.state?.city || localStorage.getItem('location');

const filtered = RestaurantData.filter(restaurant =>

restaurant.restaurantName.toLowerCase().startsWith(query.toLowerCase()) &&

(!city || restaurant.restaurantAddress.city.toLowerCase() ===

city.toLowerCase())

);

setFilteredRestaurants(filtered);

};
```

6. Return JSX

- **Navbar**: Component for the navigation bar, with a search functionality.
- **RestaurantFilters**: Component for applying and displaying filter options.
- **Menubar**: Component for displaying a menu bar, possibly with categories or other options.
- **Restaurant**: Component for displaying the list of filtered restaurants.

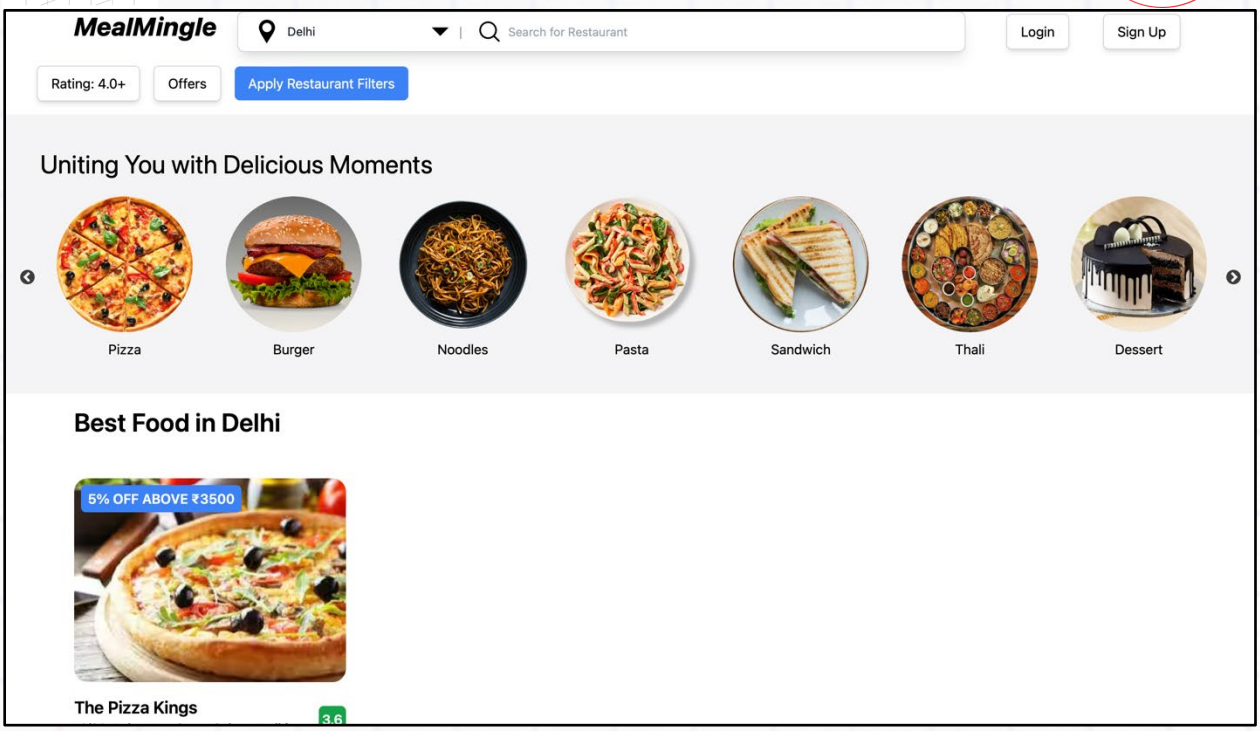
```
return (
```

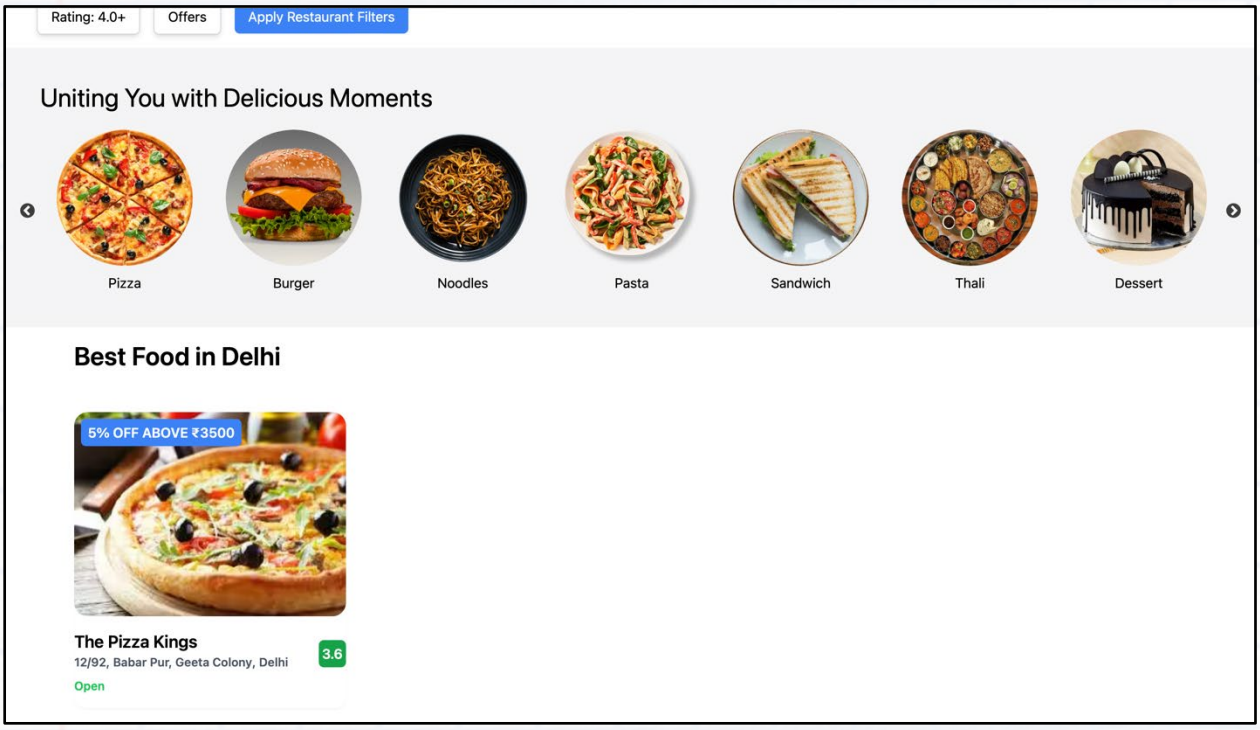
```
<div>
<Navbar city={location.state?.city} onSearch={handleSearch} />
<RestaurantFilters applyRestaurantFilters={applyRestaurantFilters}
restaurantFilters={restaurantFilters} />
<Menubar />
<Restaurant restaurant={filteredRestaurants} city={location.state?.city} />
</div>
);
};

export default RestaurantsByFilter;
```



Combining all the subcomponents of the Main Component, the UI will be as follows:

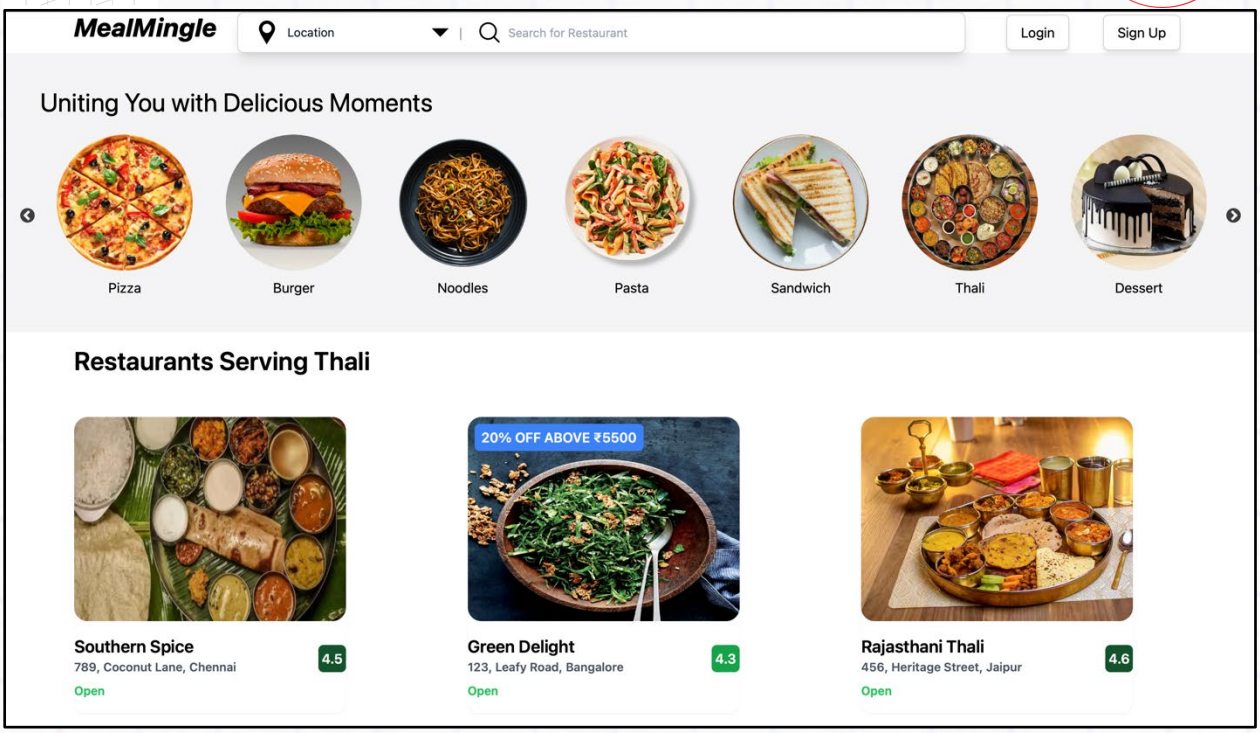
Note: The data is being fetched from the database here.



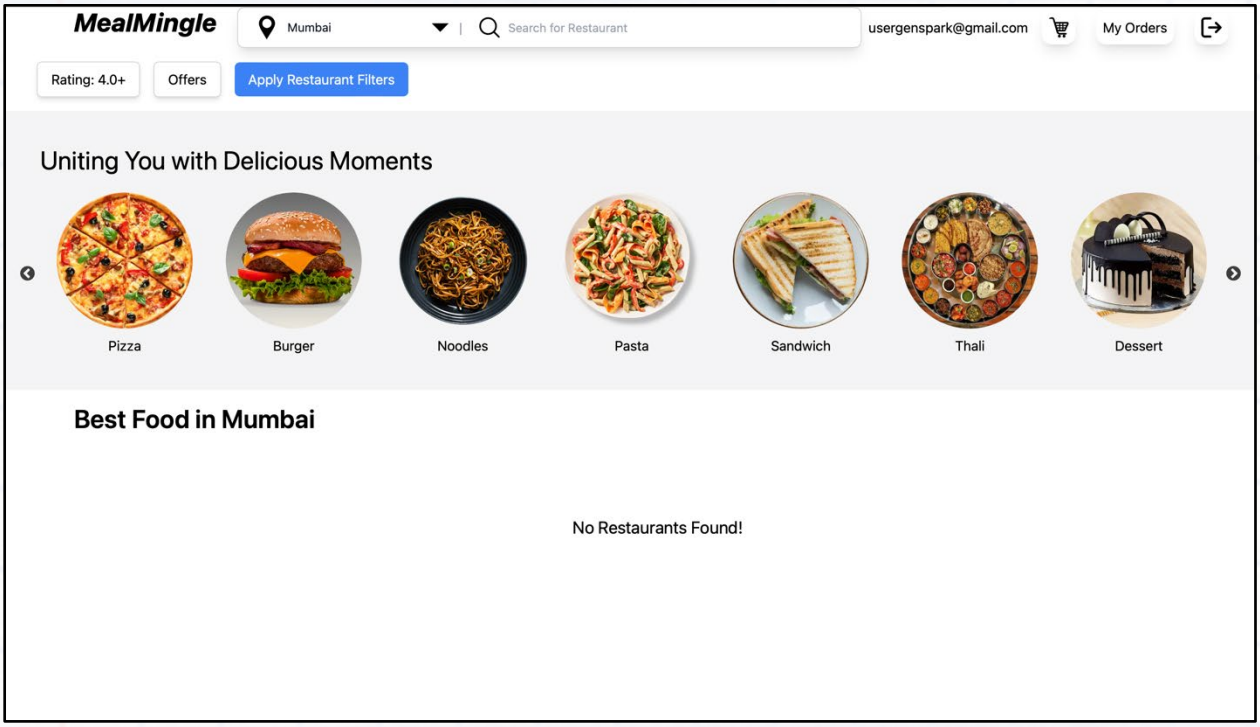


Clicking on the restaurant image of The Pizza Kings will take us to the Login page (for unauthenticated users).

Let's select Thali from the Menubar. The UI will be as shown below:



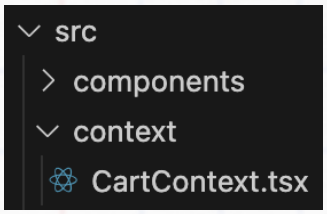
After login, the complete UI of the Main Component will be as shown below:



Cart Context

- The CartContext Component is essential for managing the shopping cart functionality within a React application.
- It allows different parts of the application to interact with the cart without needing to pass down props through multiple components.
- By creating a context, the cart state and related functions (like adding, removing items, checking the total quantity, etc) can be easily accessed anywhere within the application.
- This makes it easier to manage the cart and keep the application code clean and organized.

Create a new folder, context, inside the src directory and create a new file and name it CartContext.tsx.



1. Imports

- **createContext**: Creates a context object that allows you to pass data through the component tree without having to pass props manually.
- **useState**: Manages the state of the cart.
- **useContext**: Provides a way to consume the context within other components.
- **ReactNode**: Represents the type of children props in React components.

```
import React, { createContext, useState, useContext, ReactNode } from

'react';
```

2. Cart Item Interface

- **CartItem Interface**: Defines the structure of a cart item. Each item has properties like id, restaurantItemId, restaurantItemName, quantity, restaurantId, etc. This helps TypeScript ensure that the items in the cart have a consistent structure.

```
export interface CartItem {

id: string;

restaurantItemId: string;

restaurantItemName: string;

restaurantItemPrice: number;

restaurantItemImageUrl: string;

quantity: number;

restaurantItemCategory: string;

restaurantId: string,

restaurantMinimumOrderAmount: number;

restaurantDiscountPercentage: number;

restaurantName: string;

}
```

3. Cart Context Interface

- **CartContextType Interface**: Defines the functions and variables that will be provided by the CartContext. This includes functions like addToCart, removeFromCart, clearCart, and others. This interface helps ensure that any component consuming the context knows what to expect.

```
interface CartContextType {

cart: CartItem[];

addToCart: (item: CartItem, quantity: number) => boolean;

removeFromCart: (itemId: string) => void;

clearCart: () => void;

updateQuantity: (itemId: string, newQuantity: number) => void;

getTotalQuantity: () => number;

checkIfSameRestaurant: (newRestaurantId: string) => boolean;
```

```
placeOrder: () => void;

}
```

4. Creating the Context

- **CartContext**: This line creates the context object with an initial value of undefined. It will later be provided with the actual cart state and functions.

```
const CartContext = createContext<CartContextType | undefined>(undefined);
```

5. Cart Provider Component

- **CartProvider Component**: This is the component that will wrap around other components to provide access to the cart context.

```
export const CartProvider: React.FC<{ children: ReactNode }> = ({ children })

=> {

const [cart, setCart] = useState<CartItem[]>([]);
```

6. Cart Management Functions

- **addToCart**: Adds an item to the cart, checking if it's from the same restaurant as the current cart items. If the item is already in the cart, it updates the quantity.
- **removeFromCart**: Removes an item from the cart based on its id.
- **clearCart**: Empties the entire cart.
- **updateQuantity**: Changes the quantity of a specific item in the cart, ensuring it doesn't exceed 10.
- **getTotalQuantity**: Calculates the total number of items in the cart.
- **checkIfSameRestaurant**: Ensures that all items in the cart are from the same restaurant.
- **placeOrder**: Clears the cart when an order is placed.

```
const addToCart = (item: CartItem, quantity: number): boolean => {

if (!checkIfSameRestaurant(item.restaurantId)) {

const confirmReset = window.confirm(`Your Cart contains items from another

restaurant. Would you like to reset your cart for adding items from this

restaurant?`);
```

```
if (confirmReset) {
clearCart();
setCart([{ ...item, id: `${item.restaurantItemId}-${Date.now()}`, quantity
}]);
return true;
}
return false;
}

const existingItemIndex = cart.findIndex((cartItem) =>
cartItem.restaurantItemId === item.restaurantItemId);

if (existingItemIndex !== -1) {
const updatedCart = [...cart];
const newQuantity = updatedCart[existingItemIndex].quantity + quantity;
if (newQuantity > 10) {
alert('You cannot Add more than 10 Qty of the Same Item.');
return false;
}
updatedCart[existingItemIndex].quantity = newQuantity;
setCart(updatedCart);
} else {
setCart([...cart, { ...item, id: `${item.restaurantItemId}-${Date.now()}`,
quantity }]);
}
```

```
return true;
};

const removeFromCart = (itemId: string) => {
setCart(cart.filter(item => item.id !== itemId));
};

const clearCart = () => {
setCart([]);
};

const updateQuantity = (itemId: string, newQuantity: number) => {
if (newQuantity > 10) {
alert('You cannot Add more than 10 Qty of the Same Item.');
return;
}
const updatedCart = cart.map(item => {
if (item.id === itemId) {
return { ...item, quantity: newQuantity };
}
return item;
});
setCart(updatedCart);
};
```

```
const getTotalQuantity = () => {

return cart.reduce((total, item) => total + item.quantity, 0);

};


const checkIfSameRestaurant = (newRestaurantId: string) => {

if (cart.length === 0) {

return true;

}


return cart[0].restaurantId === newRestaurantId;

};


const placeOrder = () => {

clearCart();

};
```

8. Providing Context to Children

- **Context Provider**: This wraps children (the components that need access to the cart) with CartContext.Provider, passing down the cart state and functions.

```
return (

<CartContext.Provider value={{ cart, addToCart, removeFromCart, clearCart,

updateQuantity, getTotalQuantity, checkIfSameRestaurant, placeOrder }}>

{children}

</CartContext.Provider>

);
```

```
};
```

9. Custom Hook to Use the Cart Context

- **useCart Hook**: This custom hook makes it easy to use the cart context in any component. It ensures that the CartContextis not used outside of its provider.

```
export const useCart = () => {

const context = useContext(CartContext);

if (!context) {

throw new Error('useCart must be used within a CartProvider');

}

return context;

};
```

6.11. Menu Component

- The Menu Component is a part of a food ordering React application.
- It displays the menu items of a specific restaurant, allows users to select quantities, and add items to their cart.
- The component uses React hooks for managing state and side effects, and it interacts with other components and services, like fetching restaurant items from an API and using the cart context to manage the cart.

Create a new file inside the components folder and name it Menu.tsx.

1. Imports

- **React Imports:** We import useEffect and useState for handling side effects and state management.
- **useLocation:** This hook from react-router-dom is used to get the current route location, which includes the state passed to the component.
- **Navbar:** A component that displays the navigation bar.
- **useCart:** Custom hook to access cart functions and data.
- **toast:** A function from react-toastify to show notifications to the user.

```
import React, { useEffect, useState } from 'react';

import { useLocation } from 'react-router-dom';

import Navbar from './Navbar';

import { useCart, CartItem } from '../context/CartContext';

import { toast } from 'react-toastify';

import 'react-toastify/dist/ReactToastify.css';
```

2. Component Definition

- **location:** Retrieves the current route location which includes data passed from the previous component.
- **addToCart and checkIfSameRestaurant:** Functions from useCart to manage cart operations.
- **quantities:** State to track selected quantities for each menu item.
- **restaurantItems:** State to hold the list of menu items for the restaurant.

```
const Menu = () => {

    const location = useLocation();

    const { data } = location.state;

    const { addToCart, checkIfSameRestaurant } = useCart();

    const [quantities, setQuantities] = useState<{ [key: string]: number

}>({});

    const [restaurantItems, setRestaurantItems] = useState<CartItem[]>([]);
```

3. fetchRestaurantItems Function

- This asynchronous function fetches the restaurant's menu items from an API using the restaurant's ID.
- If there's an error, it shows an error message using toast.error.
- If the data is successfully fetched, it updates the restaurantItems state with the menu items.
- **useEffect**: This hook triggers the fetchRestaurantItems function whenever the data.restaurantId changes (which happens when the restaurant data is updated).

```
const fetchRestaurantItems = async (restaurantId: string) => {

const response = await

fetch(`http://localhost:8091/api/restaurant/${restaurantId}/items`)

const data = await response.json();

if (data.error != "") {

toast.error(data.message);

}

else {

console.log(data)

setRestaurantItems(data.data.restaurantItems);

}

}

useEffect(() => {

fetchRestaurantItems(data.restaurantId);

}, [data.restaurantId])
```

4. handleAddToCart Function

- When the user clicks "Add to Cart", this function is called.
- It gets the selected quantity for the item or defaults to 1.
- It adds extra restaurant details (like minimum order amount, discount percentage, and name) to the item.
- Then it tries to add the item to the cart using the addToCart function from the cart context.
- If the item is successfully added, it shows a success message.

```
const handleAddToCart = (item: CartItem) => {

const quantity = quantities[item.restaurantItemId] || 1;

item.restaurantMinimumOrderAmount = data.restaurantMinimumOrderAmount;

item.restaurantDiscountPercentage = data.restaurantDiscountPercentage;
```

```
item.restaurantName = data.restaurantName;



const success = addToCart(item, quantity);

if (success) {

toast.success(`${item.restaurantItemName} added to Cart!`);

}

};
```

5. handleQuantityChange Function

- This function updates the quantities state when the user selects a different quantity for a menu item.
- It takes the itemId and the new quantity and updates the state accordingly.

```
const handleQuantityChange = (itemId: string, quantity: number) => {

setQuantities(prevState => ({

...prevState,

[itemId]: quantity,

}));

};
```

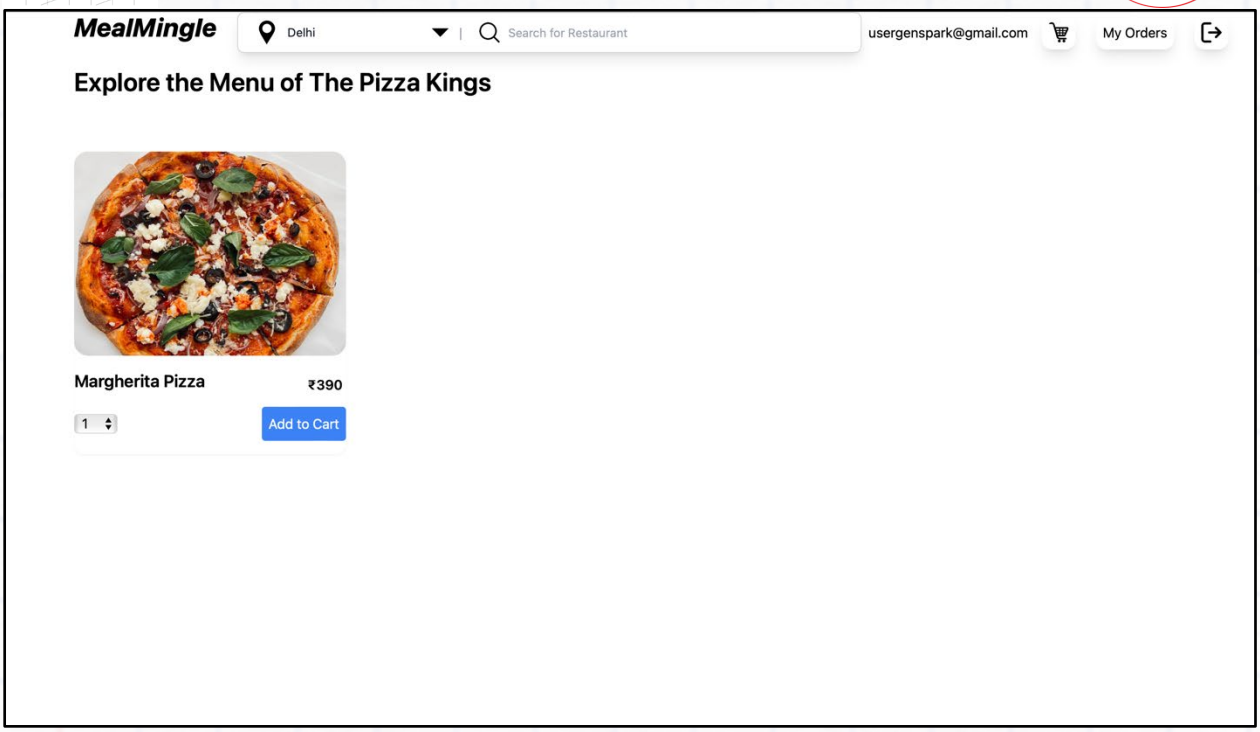
6. Return JSX

- The component renders a Navbar with the city's name from the restaurant data.
- It then displays a grid of the restaurant's menu items.
- For each item, it shows the item image, name, price, and options to select the quantity and add the item to the cart.

```
    return (

        <>

            <Navbar city={data.restaurantAddress.city} />

            <div className='p-4 pl-20'>
```

```
                <h1 className='font-semibold text-3xl mb-4'>Explore the Menu
of {data.restaurantName}</h1>
                <div className='grid grid-cols-3'>
                    {restaurantItems.map((item: CartItem) => (
                        <div key={item.restaurantItemId} className='max-w-xs
rounded-xl overflow-hidden shadow-sm mt-12'>
                            <img className='w-full rounded-2xl h-60 object-
cover' src={`${item.restaurantItemImageUrl}`} alt={item.restaurantItemName}
/>
                            <div className='py-4'>
                                <div className='flex justify-between items-
center'>
                                    <div className='font-semibold text-xl mb-
2'>{item.restaurantItemName}</div>
                                    <p className='font-semibold text-base p-
1'>₹{item.restaurantItemPrice}</p>
                                </div>
                                <div className='flex justify-between items-
center mt-2'>
                                    <select

value={quantities[item.restaurantItemId] || 1}
                                        onChange={(e) =>
handleQuantityChange(item.restaurantItemId, parseInt(e.target.value))}
```

```
                                        className='mr-2 px-4 py-2 border
border-gray-300 rounded'
                                    >
                                        {[...Array(10)].map((_, index) => (
                                            <option key={index + 1}
value={index + 1}>{index + 1}</option>
                                        ))}
                                    </select>
                                    <button onClick={() =>
handleAddToCart(item)} className='inline-block px-2 py-2 bg-blue-500 text-
white rounded hover:bg-blue-600 transition duration-300'>Add to Cart</button>
                                </div>
                            </div>
                        </div>
                    ))}
                </div>
            </div>
        </>
    );
};

export default Menu;
```



6.12. RestaurantsByCategory Component

- It fetches and displays a list of restaurants that serve a specific category of food (like "Pizza" or "Burger").

Create a new file inside the components folder and name it RestaurantsByCategory.tsx.

1. Imports

- **React**: Used to create the component.
- **useState** and **useEffect**: Hooks from React for managing state and side effects.
- **useParams**: Gets the category name from the URL (like /category/Pizza).
- **useNavigate**: Lets you navigate to different pages.
- **Navbar** and **Menubar**: Other components that are part of the page layout.
- **isAuthenticated**: A utility function to check if the user is logged in.
- **FaBan**: An icon used to indicate a restaurant is closed.

```
import React, { useState, useEffect } from 'react';

import { useParams, useNavigate } from 'react-router-dom';

import Navbar from './Navbar';
```

```
import RestaurantData from '../restaurants.json';

import Menubar from './Menubar';

import { isAuthenticated } from '../utils/authUtils';

import { FaBan } from 'react-icons/fa';
```

2. Helper Functions

isRestaurantOpen: This function checks if a restaurant is open based on its operating days and hours.

```
const isRestaurantOpen = (operationDays: string, operationHours: string) => {

const daysMap: { [key: string]: number } = {

'Sun': 0,

'Mon': 1,

'Tue': 2,

'Wed': 3,

'Thu': 4,

'Fri': 5,

'Sat': 6

};


const currentDate = new Date();

const currentDay = currentDate.getDay();

const currentTime = currentDate.getHours() * 60 + currentDate.getMinutes();


const [startDay, endDay] = operationDays.split('-');

const startDayIndex = daysMap[startDay];

const endDayIndex = daysMap[endDay];
```

```
const withinOperatingDays = startDayIndex <= endDayIndex
? currentDay >= startDayIndex && currentDay <= endDayIndex
: currentDay >= startDayIndex || currentDay <= endDayIndex;

if (!withinOperatingDays) {
return false;
}

const [startTime, endTime] = operationHours.split('-').map(time => {
const [hours, minutes] = time.replace('AM', '').replace('PM',
'').split(':').map(Number);
const isPM = time.includes('PM');
const totalMinutes = (isPM ? hours + 12 : hours) * 60 + minutes;
return totalMinutes;
});

return currentTime >= startTime && currentTime <= endTime;
}
```

getOfferPhrase: This function generates a discount phrase like "20% OFF ABOVE ₹500".

```
const getOfferPhrase = (minOrderAmt: number, discountPercentage: number) => {
    return `${discountPercentage}% OFF ABOVE ₹${minOrderAmt}`;
};
```

3. **Component Definition**

- **categoryName**: Retrieved from the URL, specifies the category of food.
- **navigate**: Allows navigation to other routes.
- **restaurants**: State variable to hold the list of restaurants serving the specified category.

```
const RestaurantsByCategory = () => {

    const { categoryName } = useParams();

    const navigate = useNavigate();

    const [restaurants, setRestaurants] = useState<any[]>([]);
```

4. Fetching Category Restaurants

Fetches the restaurants that serve the specified category from the API and updates the `restaurants` state.

```
const fetchCategoryRestaurants = async () => {

        const response = await

fetch(`http://localhost:8091/api/restaurants/${categoryName}`);

        const data = await response.json();

        setRestaurants(data.data.restaurants);

        console.log(data)

    }
```

useEffect Hook: Calls `fetchCategoryRestaurants` when the component mounts or when `categoryName` changes.

```
useEffect(() => {

        fetchCategoryRestaurants();

    }, [categoryName])
```

5. Handling Restaurant Click

- Checks if the user is authenticated.
- If authenticated, navigates to the restaurant's menu.
- If not authenticated, navigates to the login page.

```
const handleRestaurantClick = (restaurant: any) => {

        if (isAuthenticated()) {

            navigate(`/category/${categoryName}/menu`, { state: { data:

restaurant, category: categoryName } });

        } else {

            navigate('/login');

        }

    };
```

6. Return JSX

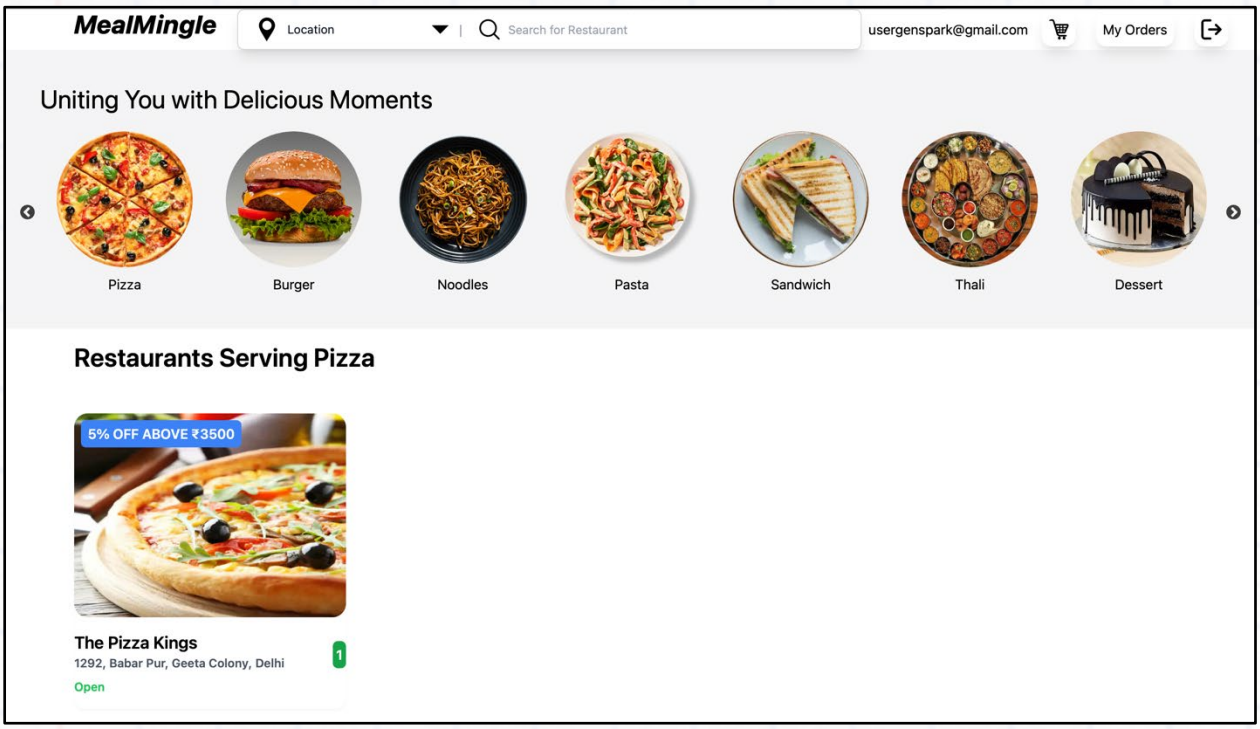
- Renders a list of restaurants in a grid format.
- Shows if the restaurant is open or closed, and displays an offer if applicable.
- If no restaurants are found, displays a message stating that no restaurants serve the selected category.

```
    return (

        <>

            <Navbar />

            <Menubar />

            <div className='p-4 pl-20'>

                <h1 className='font-semibold text-3xl'>Restaurants Serving

{categoryName}</h1>

                {restaurants.length > 0 ? (

                    <div className='grid grid-cols-3 gap-4'>
```

```
                        {restaurants.map((restaurant) => {
                            const isOpen =
isRestaurantOpen(restaurant.restaurantOperationDays,
restaurant.restaurantOperationHours);
                            const offerPhrase =
getOfferPhrase(restaurant.restaurantMinimumOrderAmount,
restaurant.restaurantDiscountPercentage);
                            return (
                                <div key={restaurant.restaurantId}
className="relative max-w-xs rounded-xl overflow-hidden shadow-sm mt-12
cursor-pointer" onClick={() => handleRestaurantClick(restaurant)}>
                                    {isOpen ? (
                                        <img className={`w-full rounded-2xl
h-60`} src={`${restaurant.restaurantImageUrl}`} alt="Restaurant Image" />
                                    ) : (
                                        <img className={`w-full rounded-2xl
h-60 filter grayscale`} src={`${restaurant.restaurantImageUrl}`}
alt="Restaurant Image" />
                                    )}
                                    {restaurant.restaurantDiscountPercentage
> 0 && (
                                        <div className="absolute top-2 left-2
bg-blue-500 text-white font-semibold py-1 px-2 rounded-md">
                                            {offerPhrase}
                                        </div>
```

```
                                    )}
                                    <div className="py-4">
                                        <div className='flex justify-between
items-center'>
                                            <div className="font-semibold
text-xl mb-2">
                                                {restaurant.restaurantName}
                                                <div className="text-sm text-
gray-600">{restaurant.restaurantAddress.pincode},
{restaurant.restaurantAddress.streetName},
{restaurant.restaurantAddress.city}</div>
                                            </div>
                                            <div className={`text-white font-
semibold text-base rounded-md p-1 ${restaurant.restaurantRating < 4.5 ? `bg-
green-600` : `bg-green-900`}`}>
                                                {restaurant.restaurantRating}
                                            </div>
                                        </div>
                                        <div className={`font-semibold text-
sm ${isOpen ? 'text-green-500' : 'text-red-500'}`}>
                                            {isOpen ? 'Open' : 'Closed'}
                                        </div>
                                    </div>
                                    {!isOpen && (
```

```
                                        <div className="absolute inset-0 bg-
black bg-opacity-50 flex items-center justify-center">
                                            <div className="w-16 h-16
rounded-full flex items-center justify-center">
                                                <FaBan className="text-white
text-3xl" />
                                            </div>
                                        </div>
                                    )}
                                </div>
                            );
                        })}
                    </div>
                ) : (
                    <div className="text-center text-xl mt-24">
                        No Restaurants are serving {categoryName} at the
moment!
                    </div>
                )}
            </div>
        </>
    );
};

export default RestaurantsByCategory;
```



6.13. RestaurantsByCategoryMenu Component

- This component displays a restaurant's menu based on a selected category.

Create a new file inside the components folder and name it RestaurantsByCategoryMenu.tsx.

1. Imports
    - **React Imports:** useState and useEffect are used for managing state and side effects.
    - **useLocation:** A hook from react-router-dom to access the current route's location, which includes the state (data) passed to this component.
    - **Navbar:** A component that displays the navigation bar.
    - **useCart:** A custom hook to access cart-related functions and data.
    - **toast:** A function from react-toastify to display notifications to the user.

```
import React, { useState, useEffect } from 'react';

import { useLocation } from 'react-router-dom';

import Navbar from './Navbar';
```

```
import { useCart, CartItem } from '../context/CartContext';

import { toast } from 'react-toastify';

import 'react-toastify/dist/ReactToastify.css';
```

2. Component Definition

- **location:** Gets the current route location, which includes data and category passed from the previous component.
- **data:** Contains information about the restaurant.
- **category:** The specific menu category to display.
- **addToCart:** A function from useCart to add items to the cart.
- **quantities:** State to keep track of the quantity selected for each menu item.
- **restaurantItems:** State to store the filtered list of menu items for the selected category.

```
const RestaurantsByCategoryMenu = () => {

    const location = useLocation();

    const { data, category } = location.state;

    const { addToCart } = useCart();

    const [quantities, setQuantities] = useState<{ [key: string]: number

}>({});

    const [restaurantItems, setRestaurantItems] = useState<CartItem[]>([]);
```

3. Fetching Restaurant Items

- **fetchRestaurantItems:** This function fetches the restaurant's menu items from a server using the restaurant ID.
- **Filtering Items:** Once the items are fetched, they are filtered to include only those that match the specified category.
- **Error Handling:** If there is an error, a notification is shown using toast.error.
- **useEffect:** Ensures that fetchRestaurantItems is called when the component is mounted. The empty dependency array [] means it runs only once.

```
const fetchRestaurantItems = async (restaurantId: string) => {
```

```
        const response = await
fetch(`http://localhost:8091/api/restaurant/${restaurantId}/items`);
        const data = await response.json();
        if (data.error == "") {
            const filteredRestaurantItems =
data.data.restaurantItems.filter((restaurantItem: CartItem) =>
restaurantItem.restaurantItemCategory === category)
            setRestaurantItems(filteredRestaurantItems);
        }
        else {
            toast.error(data.message);
        }
    }
    useEffect(() => {
        fetchRestaurantItems(data.restaurantId);
    }, []);
```

4. Adding Items to the Cart

handleAddToCart: This function is triggered when the user clicks the "Add to Cart" button.

- It sets the item quantity (using the state value or defaulting to 1).
- It updates the item's data with the restaurant's details (like minimum order amount and discount).
- It then calls the addToCart function from the CartContext to add the item to the cart.
- If the item is successfully added, it shows a success notification.

```
const handleAddToCart = (item: CartItem) => {
       const quantity = quantities[item.restaurantItemId] || 1;
```

```
        item.restaurantMinimumOrderAmount =
data.restaurantMinimumOrderAmount;
        item.restaurantDiscountPercentage =
data.restaurantDiscountPercentage;
        item.restaurantName = data.restaurantName;


        const success = addToCart(item, quantity);
        if (success) {
            toast.success(`${item.restaurantItemName} added to Cart!`);
        }
    };
```

5. Handling Quantity Changes

handleQuantityChange: This function updates the quantity for a specific item when the user selects a different quantity from the dropdown.

```
const handleQuantityChange = (itemId: string, quantity: number) => {
        setQuantities(prevState => ({
            ...prevState,
            [itemId]: quantity,
        }));
    };
```

6. Return JSX

It renders:

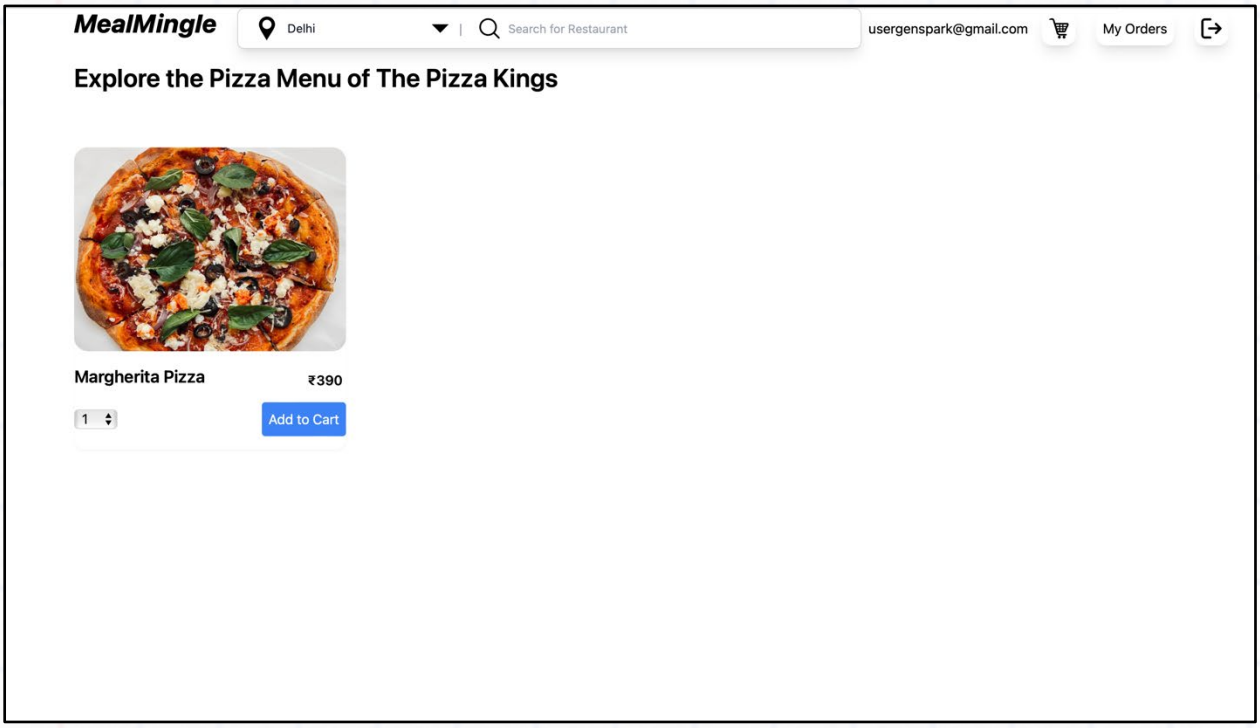
- **Navbar**: A navigation bar at the top of the page.
- **Restaurant Menu**: The menu items from the restaurant, each with a dropdown to select quantity and a button to add to the cart.

- Each item is displayed in a grid format, with an image, name, price, and an "Add to Cart" button.

```
    return (
        <>
            <Navbar city={data.restaurantAddress.city} />
            <div className='p-4 pl-20'>
                <h1 className='font-semibold text-3xl mb-4'>Explore the
{category} Menu of {data.restaurantName}</h1>
                <div className='grid grid-cols-3'>
                    {restaurantItems.map((item: CartItem) => (
                        <div key={item.restaurantItemId} className='max-w-
xs rounded-xl overflow-hidden shadow-sm mt-12'>
                            <img className='w-full rounded-2xl h-60
object-cover' src={`${item.restaurantItemImageUrl}`}
alt={item.restaurantItemName} />
                            <div className='py-4'>
                                <div className='flex justify-between
items-center'>
                                    <div className='font-semibold text-xl
mb-2'>{item.restaurantItemName}</div>
                                    <p className='font-semibold text-base
p-1'>₹{item.restaurantItemPrice}</p>
                                </div>
                                <div className='flex justify-between
items-center mt-2'>
                                    <select
```

```
value={quantities[item.restaurantItemId] || 1}
                                        onChange={(e) =>
handleQuantityChange(item.restaurantItemId, parseInt(e.target.value))}
                                        className='mr-2 px-4 py-2 border
border-gray-300 rounded'
                                    >
                                        {[...Array(10)].map((_, index) =>
(
                                            <option key={index + 1}
value={index + 1}>{index + 1}</option>
                                        ))}
                                    </select>
                                    <button onClick={() =>
handleAddToCart(item)} className='inline-block px-2 py-2 bg-blue-500 text-
white rounded hover:bg-blue-600 transition duration-300'>Add to
Cart</button>
                                </div>
                            </div>
                        </div>
                    ))}
                </div>
            </div>
        </>
    );
```

```
};


export default RestaurantsByCategoryMenu;
```



6.14. Cart Component

- It manages and displays the items that a user has added to their shopping cart.

Create a new file inside the components folder and name it Cart.tsx.

1. Imports

These imports bring in the necessary libraries, components, and data required by the Cart component.

- **React**: The core library for building the component.
- **Link, useNavigate**: For handling navigation within the app.
- **Navbar**: A custom navigation bar component used at the top of the page.

- **useCart, CartItem**: Custom hooks and types from the context that manage the cart's state.

```
import React from 'react';

import { Link, useNavigate } from 'react-router-dom';

import Navbar from './Navbar';

import { useCart, CartItem } from '../context/CartContext';
```

2. Cart Component and Hook Initialization

- The Cart component itself is a functional component.
- It uses the useCart hook to access the cart's state and the functions to manipulate it, such as removing items, clearing the cart, and updating item quantities.

```
const Cart = () => {

    const { cart, removeFromCart, clearCart, updateQuantity } = useCart();

    const navigate = useNavigate();
```

3. Calculating Total Price

This function calculates the total price of all items in the cart by iterating over each item and summing up the price multiplied by the quantity.

```
    const calculateTotalPrice = () => {

        return cart.reduce((total, item) => total + (item.restaurantItemPrice
* item.quantity), 0);

    };
```

4. Calculating Discounted Price

This function calculates the discounted price if the total price exceeds a certain minimum order amount. It also calculates the discount amount. If the total price is below the minimum order amount, the discount is not applied.

```
const calculateDiscountedPrice = (totalPrice: number, minOrderAmount: number,
discountPercentage: number) => {
        if (totalPrice > minOrderAmount) {
            const discountAmount = (totalPrice * discountPercentage) / 100;
            const discountedPrice = totalPrice - discountAmount;
            return { discountedPrice, discountAmount };
        } else {
            return { discountedPrice: totalPrice, discountAmount: 0 };
        }
    };
```

5. Setting Up Discount Variables

These variables store the discount percentage, the total amount, and the calculated discounted price and discount amount. This part of the code ensures that the calculations are only done if the cart has items.

```
let discountPercentage = cart.length > 0 ?
Number(cart[0].restaurantDiscountPercentage) : 0;
    let totalAmount = calculateTotalPrice();
    let discounts = cart.length > 0 ? calculateDiscountedPrice(totalAmount,
        Number(cart[0].restaurantMinimumOrderAmount), discountPercentage) : {
discountedPrice: 0, discountAmount: 0 };
    let discountedPrice = discounts.discountedPrice;
    let discountedAmount = discounts.discountAmount
```

6. Proceed to Payment Function

This function navigates the user to the payment page if there are items in the cart. It passes the cart items, restaurant details, and the total price (after discounts) as state to the payment page.

```
const proceedToPayment = () => {

      if (cart.length > 0) {



          navigate('/payment', {

              state: {

                  items: cart,

                  restaurantId: cart[0].restaurantId,

                  restaurantName: cart[0].restaurantName,

                  totalPrice: discountedPrice,

              }

          });

      }

  };
```

7. Return JSX

This section returns the JSX that defines what the user sees on the screen:

- **Navbar**: A navigation bar at the top.
- **Cart Heading**: A heading displaying "Your Cart".
- **Empty Cart Message**: If the cart is empty, a message saying "Your Cart is Empty!" is displayed.
- **Cart Items**: If there are items in the cart, they are displayed in a grid. Each item shows its name, price, quantity (which can be adjusted), and a remove button.
- **Total Price and Discount**: The total price, discount amount, and discounted price are displayed.

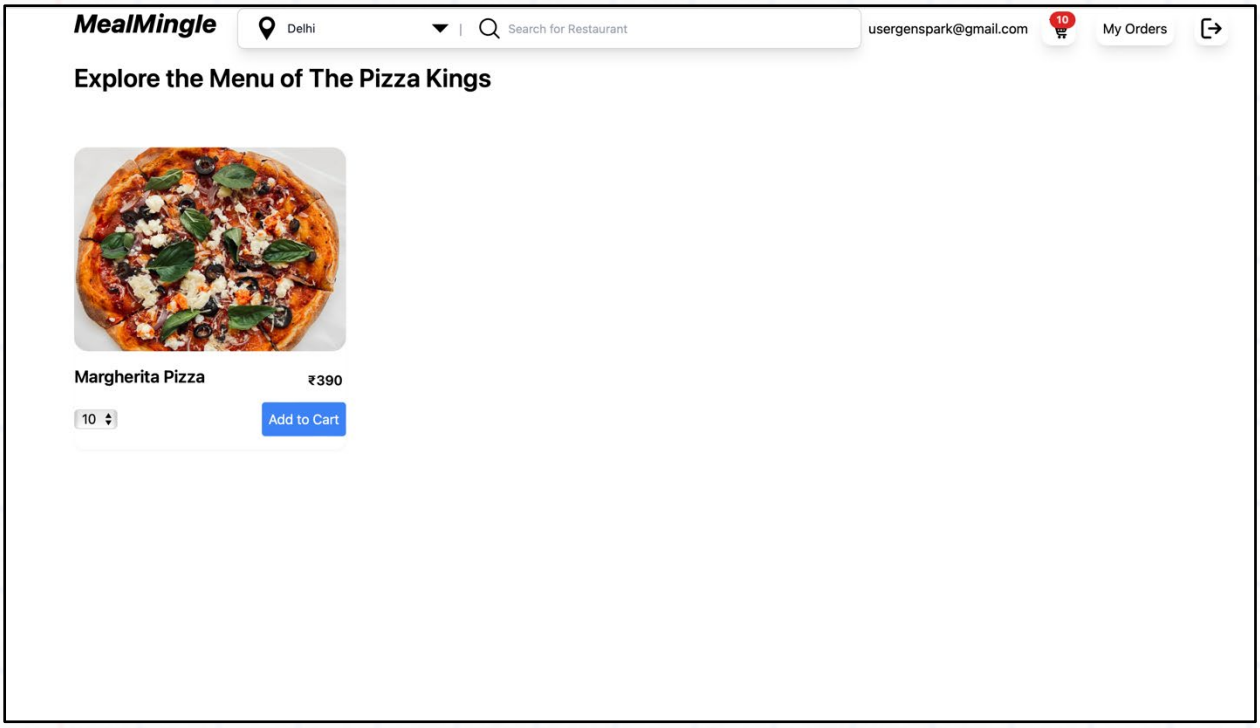
- **Clear Cart and Proceed to Payment Buttons**: Buttons that allow the user to clear the cart or proceed to payment.

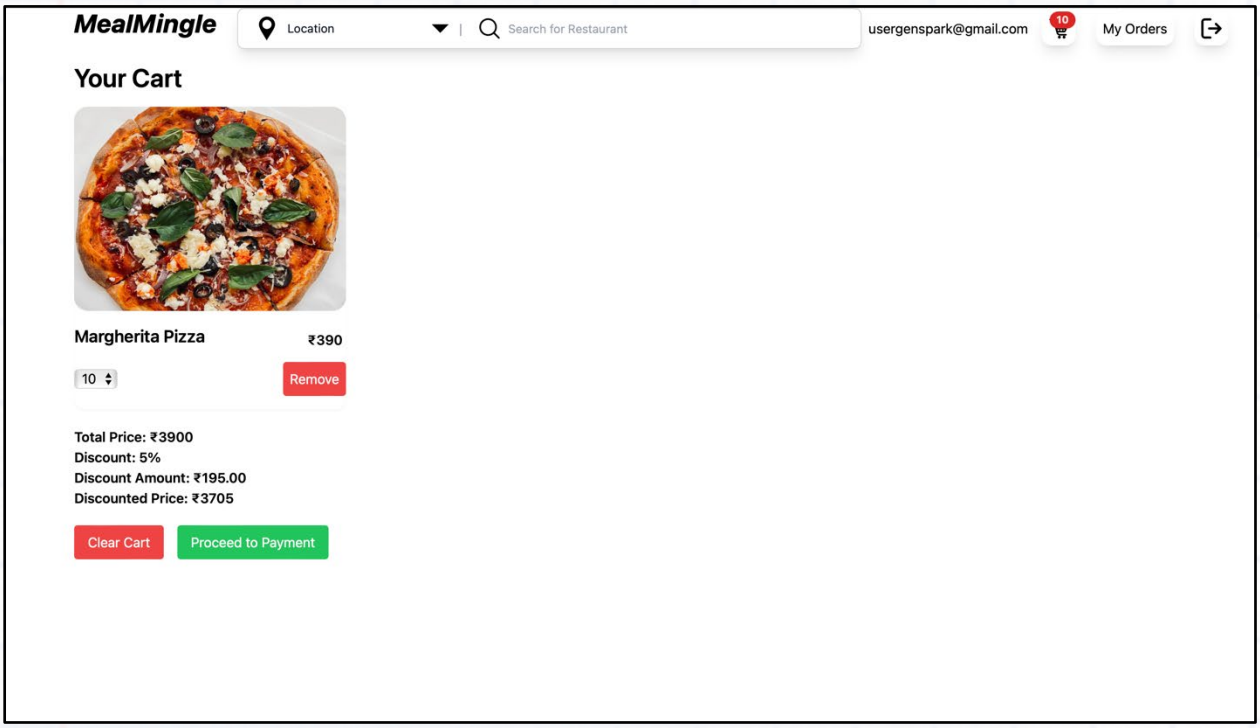
```
    return (
        <>
            <Navbar city="Location" />
            <div className='p-4 pl-20'>
                <h1 className='font-semibold text-3xl mb-4'>Your Cart</h1>
                {cart.length === 0 ? (
                    <p className="text-center text-xl mt-72">Your Cart is
Empty!</p>
                ) : (
                    <div>
                        <div className='grid grid-cols-3 gap-4'>
                            {cart.map((item: CartItem) => (
                                <div key={item.id} className='max-w-xs
rounded-xl overflow-hidden shadow-sm'>
                                    <img className='w-full rounded-2xl h-
60 object-cover' src={`${item.restaurantItemImageUrl}`}
alt={item.restaurantItemName} />
                                    <div className='py-4'>
                                        <div className='flex justify-
between items-center'>
                                            <div className='font-semibold
text-xl mb-2'>{item.restaurantItemName}</div>
                                            <p className='font-semibold
text-base p-1'>₹{item.restaurantItemPrice}</p>
```

```
                                        </div>
                                        <div className='flex justify-
between items-center mt-2'>
                                            <select
                                                value={item.quantity}
                                                onChange={(e) =>
updateQuantity(item.id, parseInt(e.target.value))}
                                                className='mr-2 px-4 py-2
border border-gray-300 rounded'
                                            >
                                                {[...Array(10)].map((_,
index) => (
                                                    <option key={index +
1} value={index + 1}>{index + 1}</option>
                                                ))}
                                            </select>
                                            <button onClick={() =>
removeFromCart(item.id)} className='inline-block px-2 py-2 bg-red-500
text-white rounded'>Remove</button>
                                        </div>
                                    </div>
                                </div>
                            ))}
                        </div>
                        <div className='mt-5'>
```

```
                            <p className='font-semibold'>Total Price:
₹{totalAmount}</p>
                            {cart &&
Number(cart[0].restaurantDiscountPercentage) > 0 && totalAmount >
cart[0].restaurantMinimumOrderAmount && (
                                <>
                                    <p className='font-semibold'>Discount:
{cart[0].restaurantDiscountPercentage}%</p>
                                    <p className='font-semibold'>Discount
Amount: ₹{Number(discountedAmount).toFixed(2)}</p>
                                    <p className='font-
semibold'>Discounted Price: ₹{Number(discountedPrice)}</p>
                                </>
                            )}
                            <button onClick={clearCart} className='mr-4
mt-5 px-4 py-2 bg-red-500 text-white rounded'>Clear Cart</button>
                            <button onClick={proceedToPayment}
className='px-4 py-2 bg-green-500 text-white rounded'>Proceed to
Payment</button>
                        </div>
                    </div>
                )}
            </div>
        </>
    );
```

```
};


export default Cart;
```





6.15. Payment Component

Create a new file inside the components folder and name it Payment.tsx.

1. Imports

These imports bring in necessary libraries, components, and types needed for the Payment component:

- **React, useState, ChangeEvent, FormEvent, useEffect**: Core React functionality, hooks for state management and handling form events.
- **redirect, useLocation, useNavigate**: For handling navigation and routing within the app.
- **Navbar**: A custom navigation bar component.
- **CartItem, useCart**: Custom hooks and types from the context that manage the cart's state.

```
import React, { useState, ChangeEvent, FormEvent, useEffect } from

'react';

import { redirect, useLocation, useNavigate } from 'react-router-dom';

import Navbar from './Navbar';

import { CartItem, useCart } from '../context/CartContext';
```

2. Interfaces for Restaurant and PaymentState

These interfaces define the structure of the data used in the Payment component:

- **Restaurant**: Describes the structure of the restaurant details, including nested properties like address and items.
- **PaymentState**: Represents the state passed via navigation, including items in the cart, the associated restaurant, the total price, and the restaurant name.

```
interface Restaurant {

    restaurantId: string;

    restaurantName: string;

    restaurantAddress: {
```

```
        streetNumber: string;
        streetName: string;
        city: string;
        country: string;
    };
    restaurantRating: number;
    restaurantMinimumOrderAmount: number;
    restaurantDiscountPercentage: number;
    restaurantAvailability: boolean;
    restaurantImageUrl: string;
    restaurantOperationDays: string;
    restaurantOperationHours: string;
    restaurantPhoneNumber: number;
    restaurantItems: {
        restaurantItemId: string;
        restaurantItemName: string;
        restaurantItemPrice: number;
        restaurantItemCategory: string;
        restaurantItemImageUrl: string;
        restaurantItemCuisineType: string;
        restaurantItemVeg: boolean;
    }[];
}


interface PaymentState {
```

```
    items: CartItem[];

    restaurantId: Restaurant;

    totalPrice: number;

    restaurantName: string

}
```

3. OrderItem Type

Defines the structure of an order item, including the name, quantity, and price.

```
type OrderItem = {

    orderItemName: string,

    orderItemQuantity: number,

    orderItemPrice: number,

}
```

4. Payment Component Initialization

The Payment component uses various hooks and initializes state:

- **useLocation, useNavigate**: These hooks are used to access the state passed from the previous component (like the cart) and to navigate to other routes.
- **useCart**: Custom hook to access cart-related functionality, like placing an order.
- **useState**: Used for managing local state within the component, like storing orders and payment details.

```
const Payment = () => {

    const location = useLocation();

    const navigate = useNavigate();

    const { state } = location;
```

```
    const { items, restaurantId, totalPrice, restaurantName } = state as
PaymentState;

    const { cart, placeOrder } = useCart();

    const [orders, setOrders] = useState<any[]>([]);
```

5. payNow Function

This function simulates the payment process:

1. **Timeout**: Redirects the user to a mock payment page after 5 seconds.
2. **Token Retrieval**: Gets the authentication token from localStorage.
3. **Order Data Preparation**: Prepares the order data, including the items, total price, shipping address, restaurant name, and order date.
4. **API Call**: Sends a POST request to the server with the order data. The response is used to update the state with the order details.

```
const payNow = async () => {

        setTimeout(() => {

            window.location.href = 'http://localhost:8089';

        }, 5000);



        const token = localStorage.getItem('token')



        const orderItems: OrderItem[] = [];



        items.forEach((orderItem, i) => {

            orderItems.push({

                orderItemName: orderItem.restaurantItemName,

                orderItemPrice: orderItem.restaurantItemPrice,

                orderItemQuantity: orderItem.quantity

            })
```

```
    });


    const orderDate = new Date().toLocaleString();


    const orderData = {
        orderItems: orderItems,
        orderTotalPrice: totalPrice,
        shippingAddress: "Manchester House, Near Nagwara 564500",
        restaurantName: restaurantName,
        orderDate: orderDate
    }
    console.log(orderData)
    const response = await fetch('http://localhost:8093/api/orders', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'Authorization': `Bearer ${token}`
        },
        body: JSON.stringify(orderData)
    })
    const data = await response.json();
    setOrders(data);
}
```

6. Payment Details State and Input Handling

- **paymentDetails**: Manages the state of the payment details entered by the user, such as card number, expiry date, and CVV.
- **handleInputChange**: Updates the `paymentDetails` state whenever the user changes an input field.

```
const [paymentDetails, setPaymentDetails] = useState({
    cardNumber: '',
    expiryDate: '',
    cvv: ''
});

const handleInputChange = (event: ChangeEvent<HTMLInputElement>) => {
    const { name, value } = event.target;
    setPaymentDetails({
        ...paymentDetails,
        [name]: value
    });
};
console.log(orders);
```

7. Return JSX

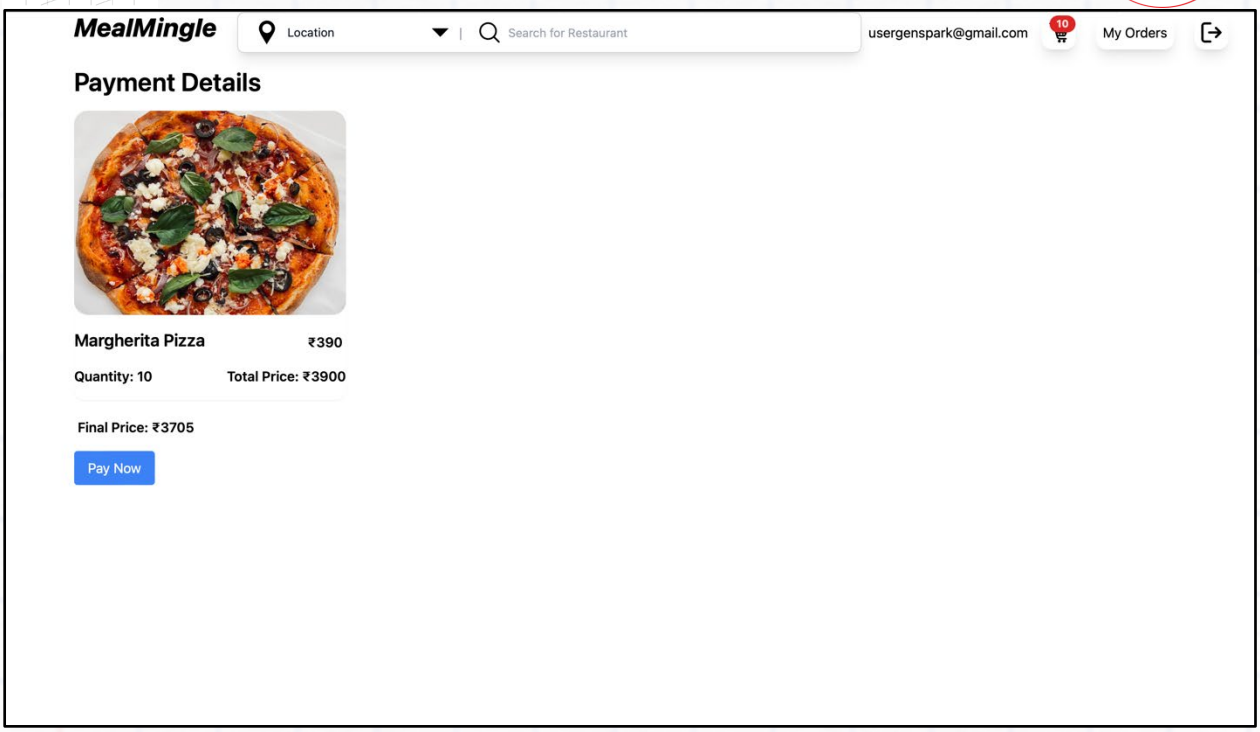
This section returns the JSX defining what the user sees on the screen:

- **Navbar**: Displays the navigation bar at the top.
- **Payment Details Heading**: A heading that indicates the section for entering payment details.
- **Cart Items**: Displays each item in the cart with its image, name, price, and quantity.
- **Final Price**: Shows the total price to be paid.

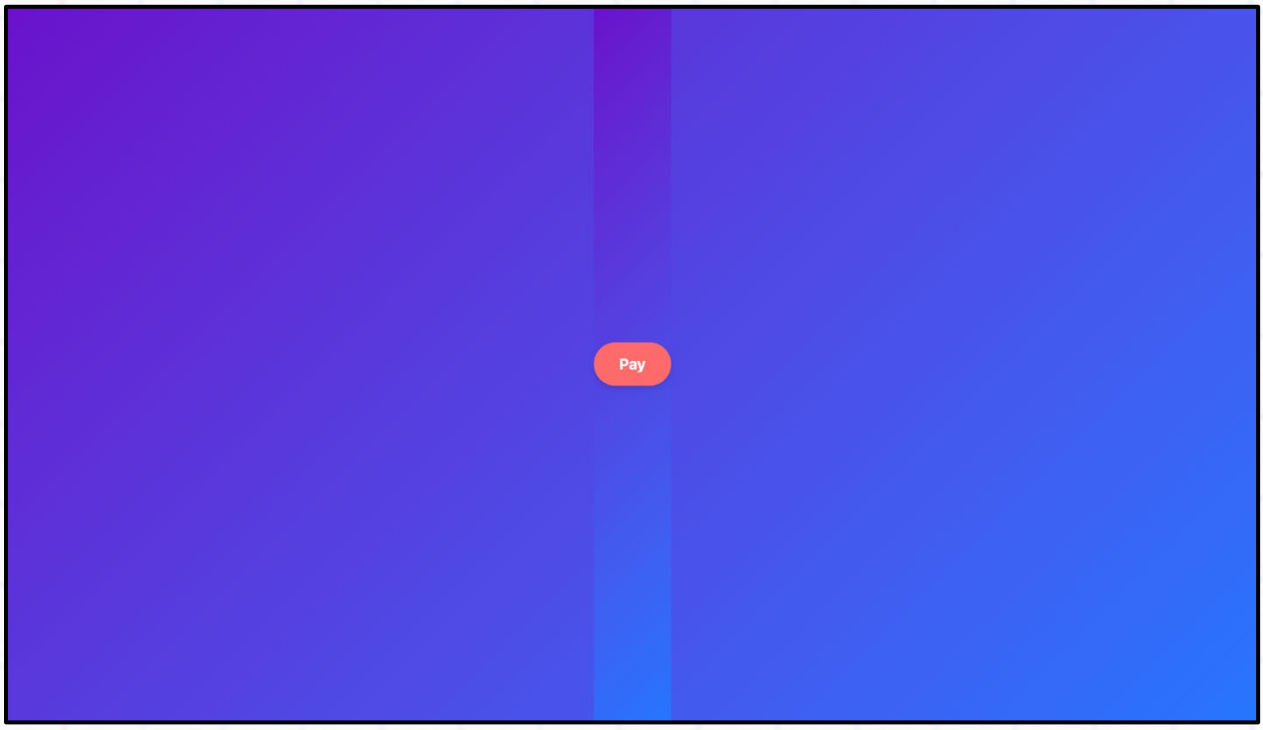
- **Pay Now Button**: Triggers the `payNow` function to simulate the payment process.

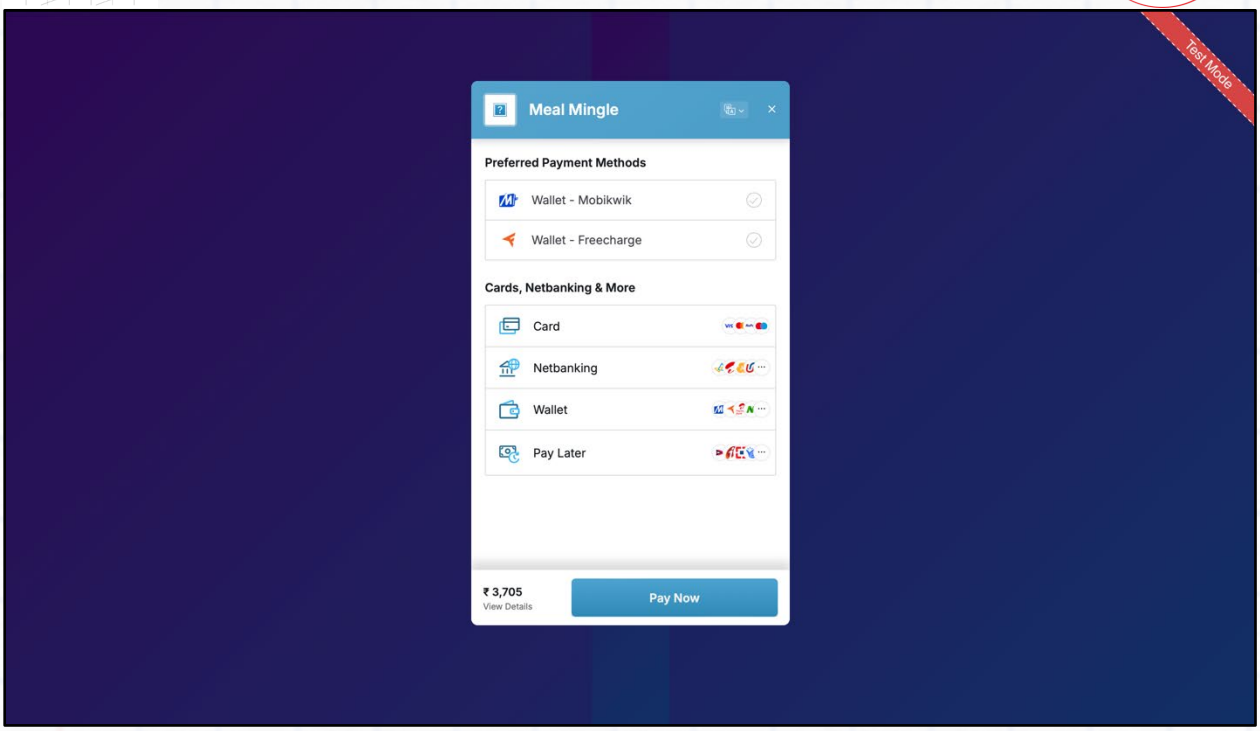
```
return (
        <>
            <Navbar />
            <div className='p-4 pl-20'>
                <h1 className='font-semibold text-3xl mb-4'>Payment
Details</h1>
                <div className='grid grid-cols-3 gap-4'>
                    {items.map((item, index) => (
                        <div key={index} className='max-w-xs rounded-xl
overflow-hidden shadow-sm'>
                            <img className='w-full rounded-2xl h-60
object-cover' src={`${item.restaurantItemImageUrl}`}
alt={item.restaurantItemName} />
                            <div className='py-4'>
                                <div className='flex justify-between'>
                                    <div className='font-semibold text-xl
mb-2'>{item.restaurantItemName}</div>
                                    <p className='font-semibold text-base
p-1'>₹{item.restaurantItemPrice}</p>
                                </div>
                                <div className='mt-2 flex justify-
between'>
                                    <p className='font-semibold text-
base'>Quantity: {item.quantity}</p>
```

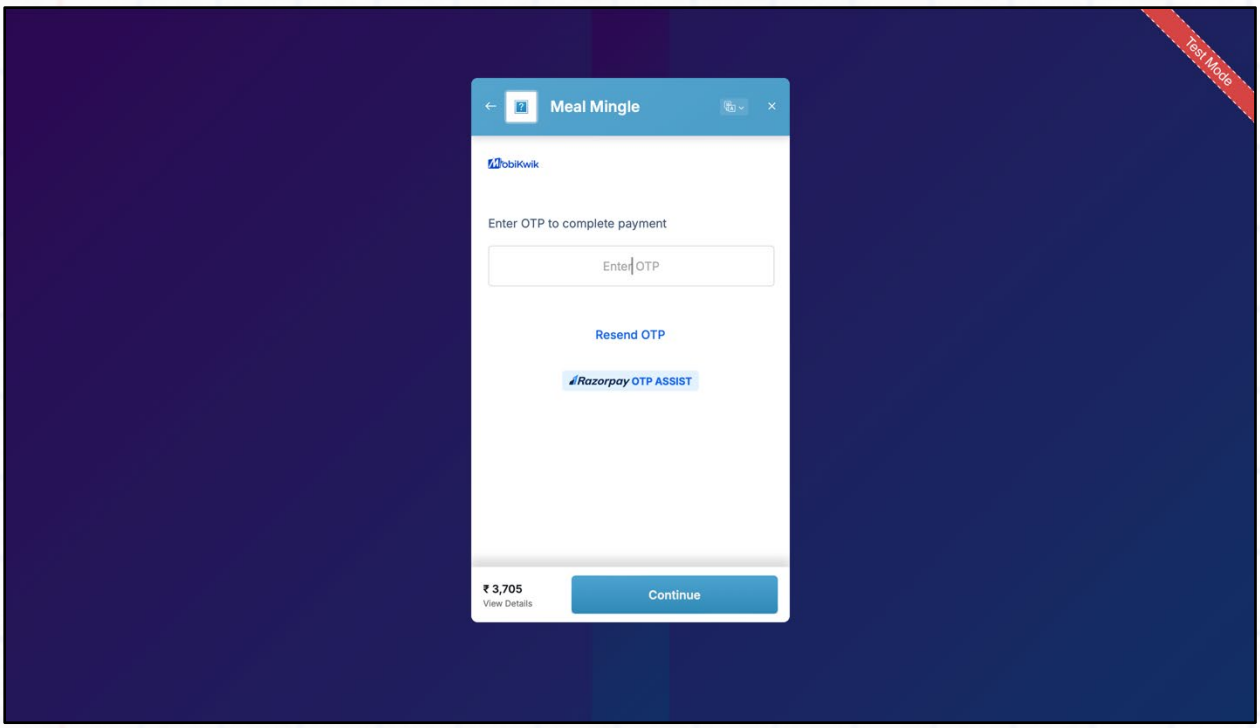
```
                                    <p className='font-semibold text-
base'>Total Price: ₹{item.restaurantItemPrice * item.quantity}</p>
                                </div>
                            </div>
                        </div>
                    ))}
                </div>
                <div className='font-semibold text-base p-1 mt-4'>
                    Final Price: ₹{totalPrice}
                </div>
                <button onClick={payNow} type='submit' className='px-4 py-
2 bg-blue-500 text-white rounded mt-3'>Pay Now</button>
            </div>
        </>
    );
};

export default Payment;
```

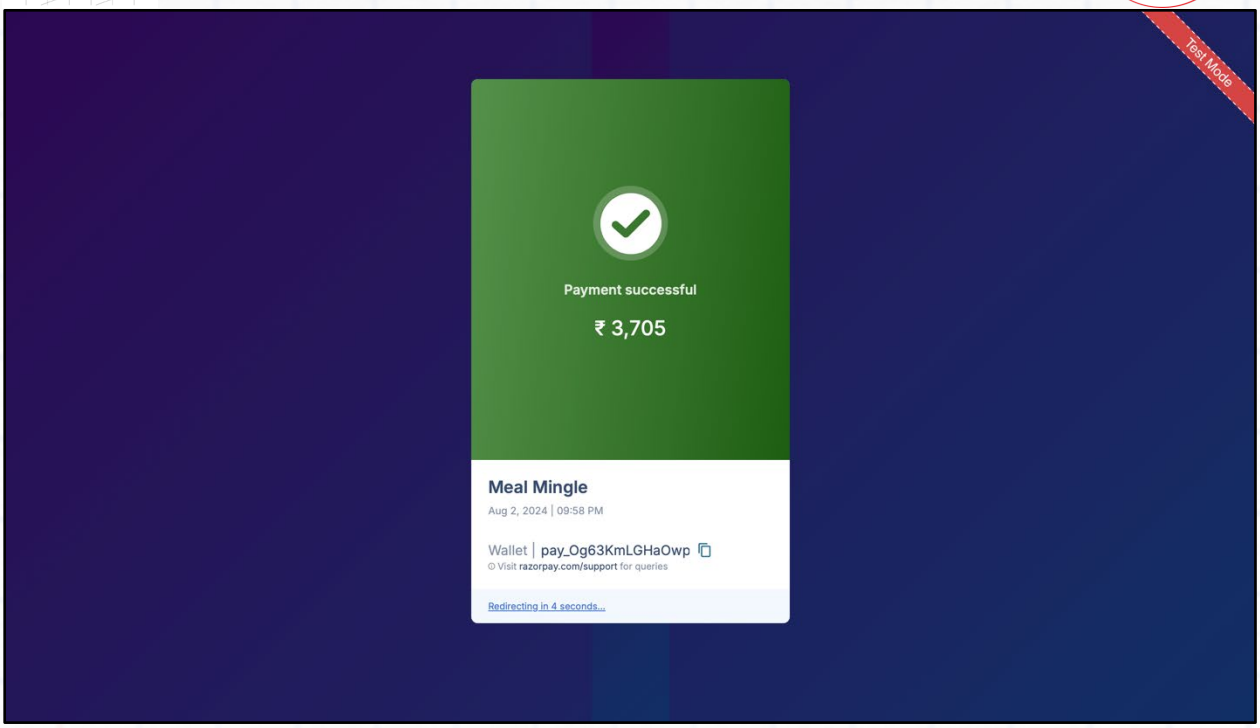


On clicking on Pay Now button,




Test Mode
Meal Mingle
Preferred Payment Methods
Wallet - Mobikwik
Wallet - Freecharge
Cards, Netbanking & More
Card
Netbanking
Wallet
Pay Later
₹ 3,705
View Details
Pay Now


Test Mode
Meal Mingle
MobiKwik
Enter OTP to complete payment
Enter OTP
Resend OTP
Razorpay OTP ASSIST
₹ 3,705
View Details
Continue



After this page, we will get to see the Payment Success page followed by the Order History page.

6.16. Payment Success Component

Create a new file inside the components folder and name it PaymentSuccess.tsx.

1. Imports

These imports bring in the necessary libraries, components, and assets for the PaymentSuccess component:

- **React, useEffect**: Core React library and a hook that allows performing side effects in function components.
- **useNavigate**: A hook from `react-router-dom` used for navigating programmatically between routes.
- **Navbar**: A custom navigation bar component, likely used across different pages for consistent navigation.
- **GreenTickIcon**: An image asset representing a green tick, used to visually indicate successful payment.

```
import React, { useEffect } from 'react';

import { useNavigate } from 'react-router-dom';
```

```
import Navbar from './Navbar';

import GreenTickIcon from '../images/green-tick-icon.png';
```

2. PaymentSuccess Component Initialization

This component displays a success message when a payment is completed and redirects the user to the order history page after a short delay:

- **useNavigate**: The `useNavigate` hook is used to navigate to the order history page (`/order-history`) after the payment success message is displayed for a few seconds.
- **useEffect**: The `useEffect` hook handles the side effect of setting up a timer to navigate to the order history page after 3 seconds. It also cleans up the timer when the component unmounts to prevent memory leaks.

```
const PaymentSuccess = () => {

    const navigate = useNavigate();



    useEffect(() => {

        const timer = setTimeout(() => {

            navigate('/order-history');

        }, 3000);



        return () => clearTimeout(timer);

    }, [navigate]);
```

3. Return JSX

This section defines what the user sees on the screen when they visit the payment success page:

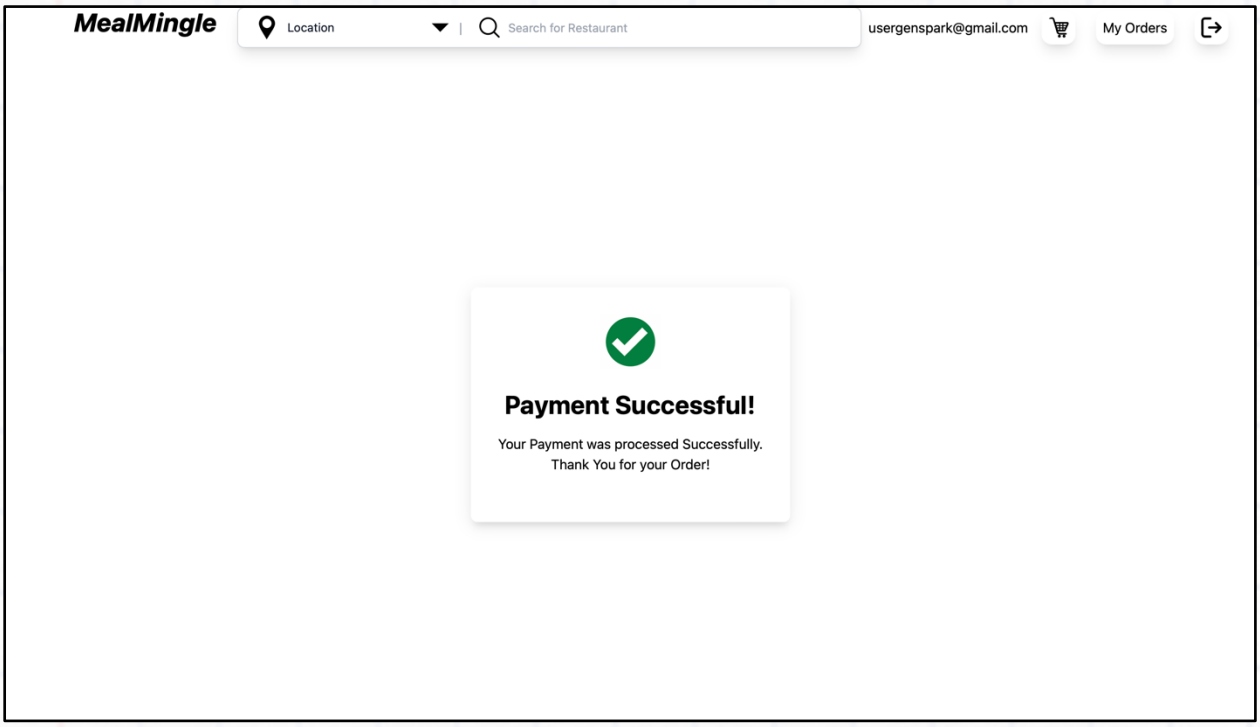
- **Navbar**: Displays the navigation bar at the top of the page, ensuring consistent navigation.
- **Main Container**: A `div` element that centers the success message both vertically and horizontally using Flexbox. The entire container takes up the

minimum height of the screen (`min-h-screen`), ensuring it fills the viewport.

- **Success Message Box**: A `div` element styled to look like a card, containing the success message. It has a white background, rounded corners, and shadow effects for a clean, modern look.
- **Green Tick Icon**: An image that visually represents the success of the payment. It is displayed at the top of the message box.
- **Success Message**: A heading (`h1`) and a paragraph (`p`) that inform the user that their payment was successful and thank them for their order.

```
    return (
        <>
            <Navbar />
            <div className='flex flex-col items-center justify-center min-h-screen'>
                <div className='flex flex-col items-center justify-center bg-white rounded-lg shadow-md p-8' style={{ boxShadow: '0px 8px 8px -4px rgba(0, 0, 0, 0.1), 0px 4px 20px 0px rgba(0, 0, 0, 0.08)' }}>
                    <img src={GreenTickIcon} alt='Green Tick Icon' className='w-auto h-16 mb-6' />
                    <h1 className='font-bold text-3xl text-black mb-4'>Payment Successful!</h1>
                    <p className='text-md text-black mb-6 text-center'>
                        Your Payment was processed Successfully.
                        <br />
                        Thank You for your Order!
                    </p>
                </div>
            </div>
        </>
```

```
    );
};


export default PaymentSuccess;
```



6.17. MyOrders Component

Create a new file inside the components folder and name it MyOrders.tsx.

1. Imports

These imports bring in the necessary libraries and components for the MyOrders component:

- **React, useEffect, useState**: Core React library and hooks. `useState` manages the component's state, and `useEffect` handles side effects such as fetching data.
- **Navbar**: A custom navigation bar component used to provide consistent navigation across the application.
- **toast**: A function from `react-toastify` used to show toast notifications for success or error messages.

```
import React, { useEffect, useState } from 'react';

import Navbar from './Navbar';

import { toast } from 'react-toastify';
```

2. Interfaces

These interfaces define the structure of the data used in the component:

- **OrderItem**: Represents an item in an order, including its name, price, and quantity.
- **Order**: Represents an entire order, including its ID, items, total price, date, and status.

```
interface OrderItem {

    orderItemName: string;

    orderItemPrice: number;

    orderItemQuantity: number;

}



interface Order {

    orderId: string;

    orderItems: OrderItem[];

    orderTotalPrice: number;

    orderDate: string;

    orderStatus: 'Pending' | 'Confirmed' | 'Preparing' | 'Ready' |

'Delivered';

}
```

3. Component Initialization

The MyOrders component fetches and displays the user's orders and allows them to cancel pending orders:

- **State Management**:
  - `orders`: Manages the list of orders fetched from the server.
  - `token`: Retrieves the authentication token from `localStorage` for API requests.

```
const MyOrders = () => {

    const [orders, setOrders] = useState<Order[]>([]);

    const token = localStorage.getItem('token');
```

Fetching Orders:

- `fetchOrders`: An asynchronous function that retrieves orders from the API. It adds the current date if the order does not have one and updates the component state with the fetched orders.

```
    const fetchOrders = async () => {

        const response = await fetch('http://localhost:8093/api/orders', {

            method: 'GET',

            headers: {

                'Content-Type': 'application/json',

                'Authorization': `Bearer ${token}`

            },

        });

        const data = await response.json();

        const ordersWithDate = data.data.order.map((order: Order) => {

            if (!order.orderDate) {

                order.orderDate = new Date().toLocaleString();

            }
```

```
            return order;
        });
        console.log(ordersWithDate)
        setOrders(ordersWithDate)
    }
```

useEffect Hook: Calls `fetchOrders` when the component mounts to load the user's orders.

```
useEffect(() => {
        fetchOrders();
    }, []);
```

Cancel Order:

- `cancelOrder`: An asynchronous function that sends a delete request to the API to cancel an order. It shows a confirmation prompt and handles success or error responses using `toast` notifications.

```
const cancelOrder = async (order: any) => {
        const isConfirmed = window.confirm('Are you sure you want to
Cancel this Order?');
        if (isConfirmed) {
            const token = localStorage.getItem('token');
            const response = await
fetch(`http://localhost:8093/api/orders/${order.orderId}`, {
                method: 'DELETE',
                headers: {
                    'Authorization': `Bearer ${token}`
                }
```

```
            })

            const data = await response.json();


            if (data.error != "") {

                toast.error(data.message);

            }

            else {

                toast.success(data.message);

                fetchOrders();

            }

        }

    };
```

4. Return JSX

This section defines the user interface of the MyOrders component:

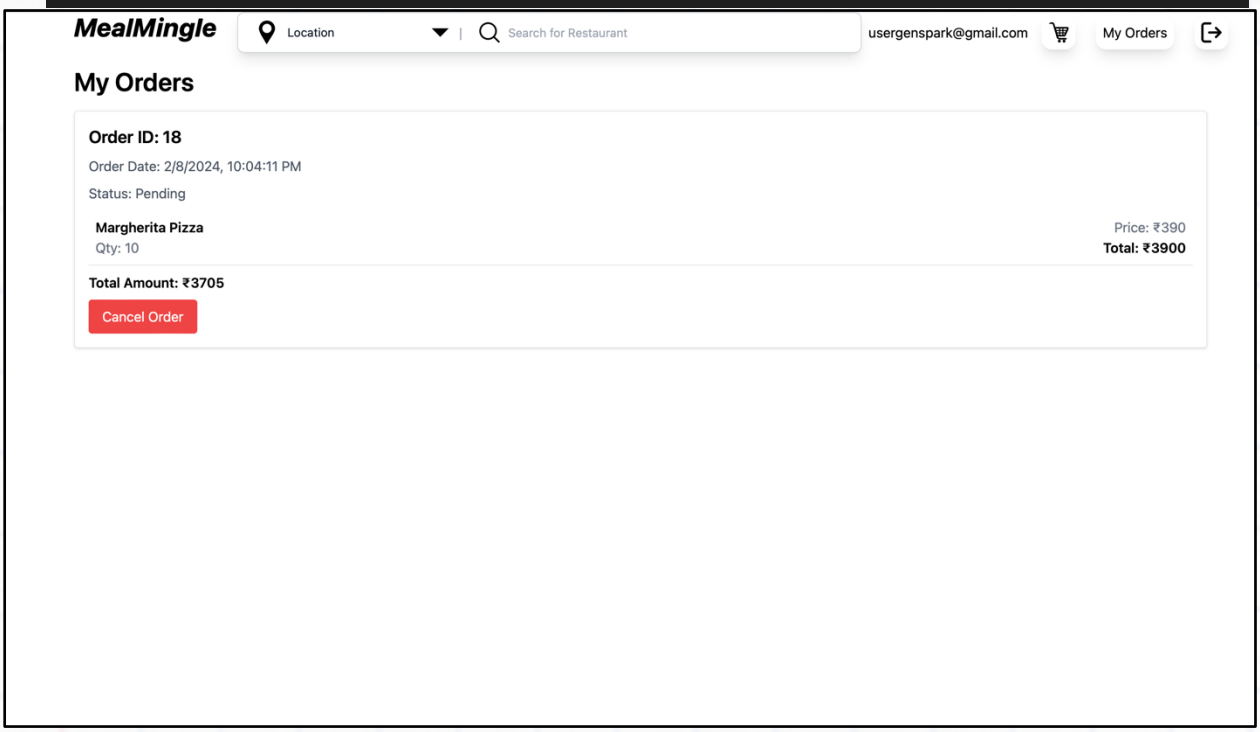
- **Navbar**: Displays the navigation bar at the top of the page.
- **Order Display**:
    - Displays a message if no orders are found.
    - If there are orders, it maps over them to display each order's details, including:
        - Order ID
        - Order Date
        - Order Status
        - Items in the order (name, quantity, price)
        - Total amount of the order
    - A "Cancel Order" button is displayed if the order is still pending.

```
return (

        <>

            <Navbar />

            <div className='p-4 pl-20'>
```

```
                <h1 className='font-semibold text-3xl mb-4'>My Orders</h1>
                {orders.length === 0 ? (
                    <p className="text-center text-xl mt-72">You have No
Orders yet!</p>
                ) : (
                    <div className='space-y-4'>
                        {orders.map((order) => (
                            <div key={order.orderId} className='border p-4
rounded shadow mr-10'>
                                <h2 className='text-xl font-semibold mb-
2'>Order ID: {order.orderId}</h2>
                                <p className='text-gray-600 mb-2'>Order
Date: {order.orderDate}</p>
                                <p className='text-gray-600 mb-2'>Status:
{order.orderStatus}</p>
                                <div className='space-y-2'>
                                    {order.orderItems.map((item, index) =>
(
                                        <div key={index} className='flex
justify-between items-center p-2 border-b'>
                                            <div className='flex flex-
col'>
                                                <span className='font-
semibold'>{item.orderItemName}</span>
```

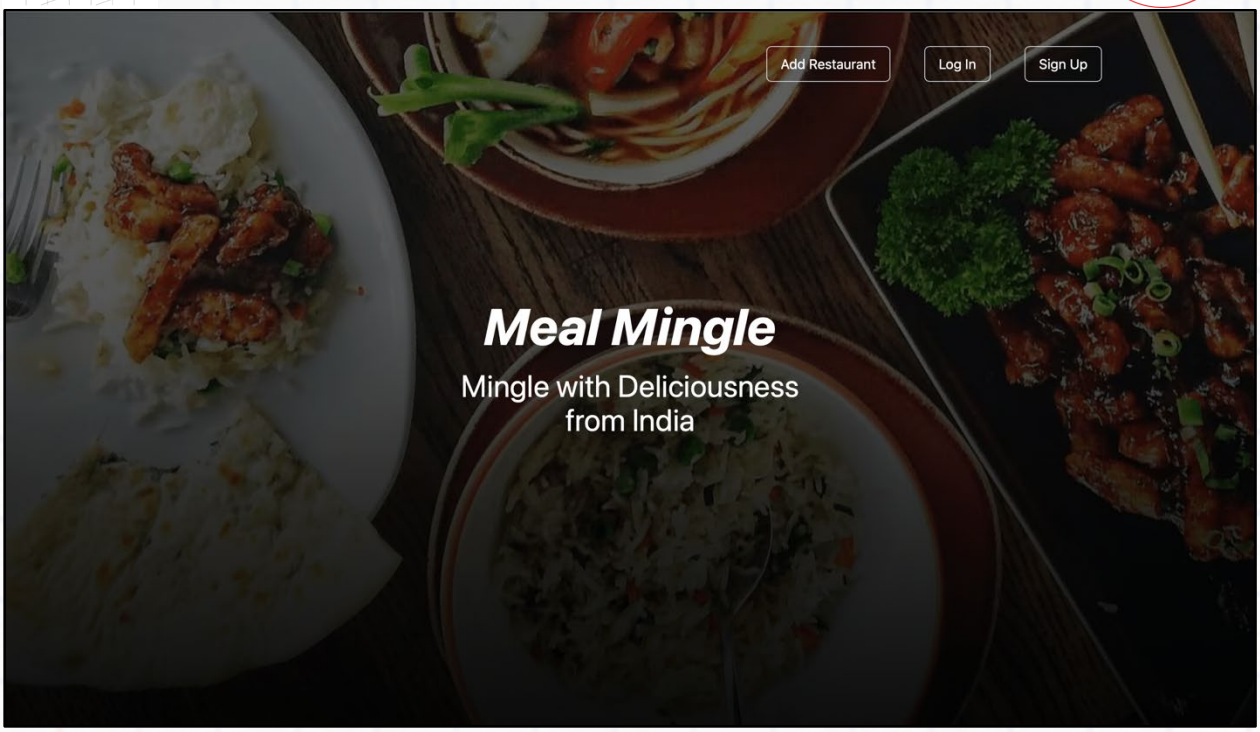
```
                                        <span className='text-
gray-500'>Qty: {item.orderItemQuantity}</span>
                                    </div>
                                    <div className='flex flex-col
items-end'>
                                        <span className='text-
gray-500'>Price: ₹{item.orderItemPrice}</span>
                                        <span className='font-
semibold'>Total: ₹{item.orderItemPrice * item.orderItemQuantity}</span>
                                    </div>
                                </div>
                            ))}
                        </div>
                        <div className='font-semibold mt-2'>Total
Amount: ₹{order.orderTotalPrice}</div>
                        {order.orderStatus === 'Pending' && (
                            <button
                                onClick={() => cancelOrder(order)}
                                className='mt-2 px-4 py-2 bg-red-
500 text-white rounded'
                            >
                                Cancel Order
                            </button>
                        )}
                    </div>
```

```
                        ))}
                    </div>
                ))}
            </div>
        </>
    );
};

export default MyOrders;
```



This marks the end of the User Dashboard.

Now, let us move to the Admin Dashboard.



On clicking on Add Restaurant button, it will navigate to the Partner With Us page.

6.18. PartnerWithUs Component

Create a new file inside the components folder and name it PartnerWithUs.tsx.

1. Imports

The component imports various elements needed for functionality and styling:

- **PartnerWithUsBanner**: An image used as the background banner for the component.
- **isAuthenticated**: A utility function from `authUtils` to check if the user is authenticated.
- **Link, useNavigate**: Hooks and components from `react-router-dom` for navigation and linking.

```
import PartnerWithUsBanner from '../images/partner-with-us-banner.png';

import { isAuthenticated } from '../utils/authUtils';

import { Link, useNavigate } from 'react-router-dom';
```

2. Component Function

The PartnerWithUs component provides a page where users can either register a restaurant or view existing restaurants based on their authentication status.

```
const PartnerWithUs = () => {

    const navigate = useNavigate();
```

3. Event Handlers

These functions handle navigation based on the user's authentication status:

- **`handleRegisterRestaurantClick`**: Navigates to the `/register-restaurant` page if the user is authenticated; otherwise, it navigates to the `/admin/login` page.

```
const handleRegisterRestaurantClick = () => {

        if (isAuthenticated()) {

            navigate('/register-restaurant');

        } else {

            navigate('/admin/login');

        }

    };
```

`handleViewRestaurantsClick`: Navigates to the `/view-admin-restaurants` page if the user is authenticated; otherwise, it navigates to the `/admin/login` page.

```
const handleViewRestaurantsClick = () => {

        if (isAuthenticated()) {

            navigate('/view-admin-restaurants');

        } else {

            navigate('/admin/login');
```

```
        }

    };
```

4. Return JSX

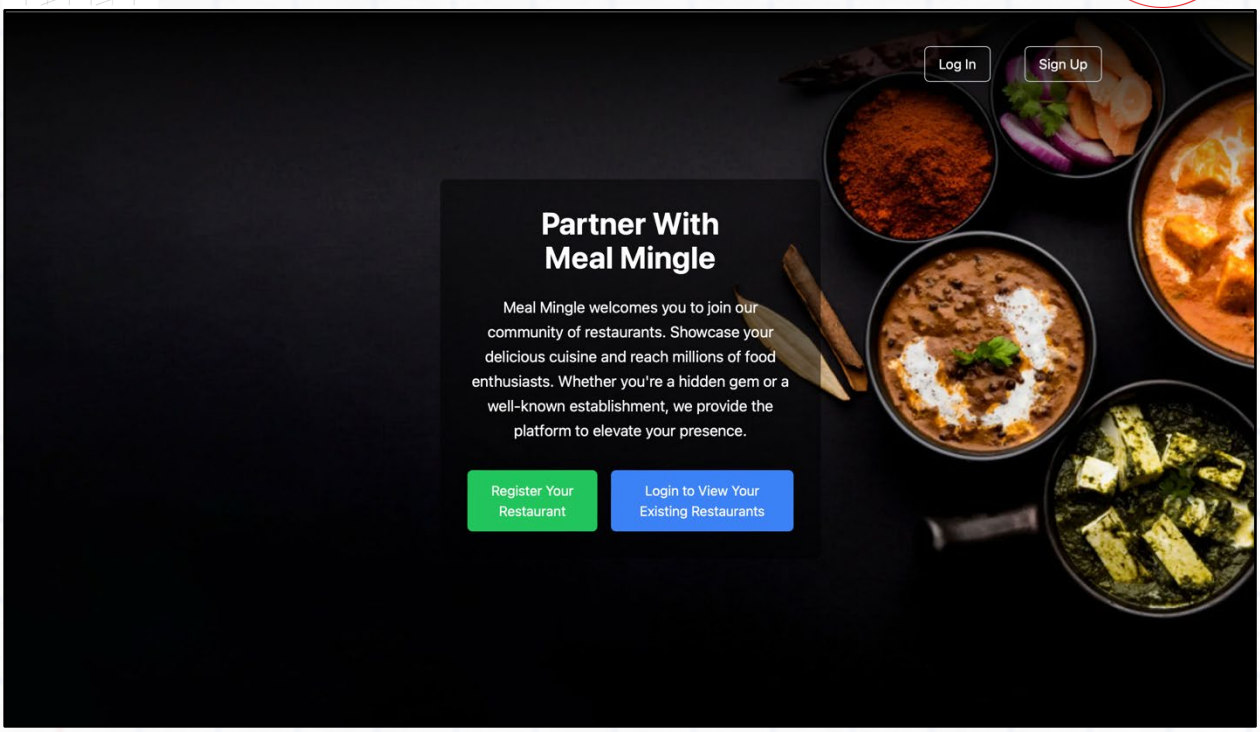
1. **Background Container**:
    - Uses a full-screen background image (`PartnerWithUsBanner`).
2. **Conditional Login/Signup Buttons**:
    - If not authenticated, displays login and signup buttons in the top-right corner.
3. **Main Content**:
    - **Overlay Container**: A semi-transparent black background with centered text.
    - **Heading and Description**: Promotes joining Meal Mingle with a title and description.
    - **Action Buttons**:
        - **Register Your Restaurant**: Navigates to restaurant registration if authenticated; otherwise, to login.
        - **Login to View Your Existing Restaurants**: Navigates to view existing restaurants if authenticated; otherwise, to login.

The component provides clear navigation options and information based on user authentication status.

```
    return (
        <div
            className="flex items-center justify-center min-h-screen bg-cover
bg-center"
            style={{
                backgroundImage: `url(${PartnerWithUsBanner})`,
            }}
        >
            {!isAuthenticated() && (
                <div style={{ position: 'absolute', top: '40px', right:
'20px' }}>
```

```
                    <Link to='/admin/login'>
                        <button className='text-base px-4 py-2 text-white
rounded-md border border-gray-300 mr-10'>
                            Log In
                        </button>
                    </Link>
                    <Link to='/admin/signup'>
                        <button className='text-base px-4 py-2 text-white
rounded-md border border-gray-300 mr-40'>
                            Sign Up
                        </button>
                    </Link>
                </div>
            )}
            <div className="bg-black bg-opacity-50 p-8 rounded-lg text-center
text-white max-w-md">
                <h1 className="text-4xl font-bold mb-6">Partner With <br
/>Meal Mingle</h1>
                <p className="text-lg mb-8 leading-relaxed">
                    Meal Mingle welcomes you to join our community of
restaurants. Showcase your delicious cuisine and reach millions of food
enthusiasts. Whether you're a hidden gem or a well-known establishment, we
provide the platform to elevate your presence.
                </p>
                <div className="flex justify-center space-x-4">
```

```
                    <button
                        className="bg-green-500 hover:bg-green-600 text-white
px-6 py-3 rounded-md transition duration-300"
                        onClick={handleRegisterRestaurantClick}
                    >
                        Register Your Restaurant
                    </button>
                    <button
                        className="bg-blue-500 hover:bg-blue-600 text-white
px-6 py-3 rounded-md transition duration-300"
                        onClick={handleViewRestaurantsClick}
                    >
                        Login to View Your Existing Restaurants
                    </button>
                </div>
            </div>
        </div>
    );
};

export default PartnerWithUs;
```



6.19. Admin Signup Component

- The AdminSignup component is a React functional component that allows new administrators to sign up.
- It provides a form where administrators can enter their name, email, password, and phone number.

Create a new file inside the components folder and name it AdminSignup.tsx.

1. Imports
   - useState is used to manage the component's state.
   - createUserWithEmailAndPassword is a Firebase function to create a new user.
   - auth is the Firebase authentication object.
   - useNavigate is used for navigating to different pages.
   - ToastContainer and toast from react-toastify are used for displaying notifications.
   - parsePhoneNumberFromString is used to validate phone numbers.

```
import React, { useState } from 'react';

import { createUserWithEmailAndPassword, sendEmailVerification,

signInWithPopup, GoogleAuthProvider, onAuthStateChanged } from

'firebase/auth';
```

```
import { auth } from '../firebase/setup';

import { Link, useNavigate } from 'react-router-dom';

import GoogleIcon from '../images/google-icon.png';

import { ToastContainer, toast } from 'react-toastify';

import 'react-toastify/dist/ReactToastify.css';

import { parsePhoneNumberFromString } from 'libphonenumber-js';
```

2. State Variables

- State variables store the values entered by the user and any error messages.
- name, email, password, and phone store the user's input.
- error stores any error messages.

```
const AdminSignup = () => {

    const navigate = useNavigate();

    const [name, setName] = useState("");

    const [email, setEmail] = useState("");

    const [password, setPassword] = useState("");

    const [phone, setPhone] = useState("");

    const [error, setError] = useState<string | null>(null);
```

3. `adminData` Object

This object is created to hold the admin's registration data.

```
    const adminData = { userEmail: email, userName: name, userPassword:

password, userPhone: phone };
```

4. `addAdminToThePortal` **Function**

This function sends a POST request to a backend server to register the admin.

- Sends registration data to the server.
- Displays success or error messages based on the server's response.

```
const addAdminToThePortal = async () => {
        const response = await
fetch('http://localhost:8090/api/users/register/admin', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify(adminData)
        })
        const data = await response.json();
        if (data.error != "") {
            toast.error(data.message);
        }
        else {
            toast.success(data.message);
        }
        console.log(data);
    }
```

5. Validation Functions

These functions check if the user input is valid.

- validateEmail checks if the email format is correct.
- validatePassword ensures the password is at least 6 characters long.
- validatePhoneNumber validates the phone number using the libphonenumber-js library.

```
  const validateEmail = (email: string) => {
        const re = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
```

```
        return re.test(String(email).toLowerCase());
    };


    const validatePassword = (password: string) => {
        return password.length >= 6;
    };


    const validatePhoneNumber = (phone: string) => {
        const phoneNumber = parsePhoneNumberFromString(phone, 'IN');
        return phoneNumber && phoneNumber.isValid();
    };
```

6. `checkPhoneNumberExists` Function

- This function checks if the phone number is already in use.
- Sends a GET request to check if the phone number already exists.

```
    const checkPhoneNumberExists = async (phone: string) => {
        try {
            const response = await
fetch(`http://localhost:8090/api/users/phone/verify?phone=${phone}`, {
                method: 'GET',
                headers: {
                    'Content-Type': 'application/json'
                }
            });
            const data = await response.json();
            return data.exists;
```

```
        } catch (error) {
            console.error(error);
            return false;
        }
    };
```

7. `emailSignUp` Function

- This function handles the sign-up process, including validation and user creation.
- Validates user input.
- Checks if the phone number is already in use.
- Registers the admin and handles Firebase authentication.
- Displays success or error messages and redirects to the login page.

```
const emailSignUp = async () => {
        if (!name.trim()) {
            toast.error('Name is Required');
            return;
        }

        if (!validateEmail(email)) {
            toast.error('Invalid Email Format');
            return;
        }

        if (!validatePassword(password)) {
            toast.error('Password must be at least 6 characters long!');
            return;
        }
```

```
        if (!validatePhoneNumber(phone)) {
            toast.error('Invalid Phone Number');
            return;
        }

        try {
            const phoneExists = await checkPhoneNumberExists(phone);
            if (phoneExists) {
                toast.error('Phone Number is Already in Use!');
                return;
            }

            addAdminToThePortal();
            const userCredential = await createUserWithEmailAndPassword(auth,
email, password);
            const user = userCredential.user;

            toast.success('Signed Up Successfully! Redirecting to Admin Login
page.');

            onAuthStateChanged(auth, async (user) => {
                if (user) {
                    console.log('Admin Signed Up Successfully:', user.uid);
                }
```

```
            });


            setTimeout(() => {
                navigate('/admin/login');
            }, 3000);
        }
        catch (err: any) {
            console.error(err);
            if (err.code === 'auth/email-already-in-use') {
                toast.error('Email Address is Already in Use!');
            } else if (err.code === 'auth/invalid-email') {
                toast.error('Invalid Email Format');
            } else {
                toast.error('Failed to Sign Up. Please try again later.');
            }
        }
    };
```

8. `handleClose` Function

This function navigates the user back to the "Partner With Us" page.

```
const handleClose = () => {
        navigate('/partner-with-us');
    };
```

9. Return JSX

- Renders a modal dialog with a form for admin sign-up.
- Includes form fields for name, email, password, and phone number.

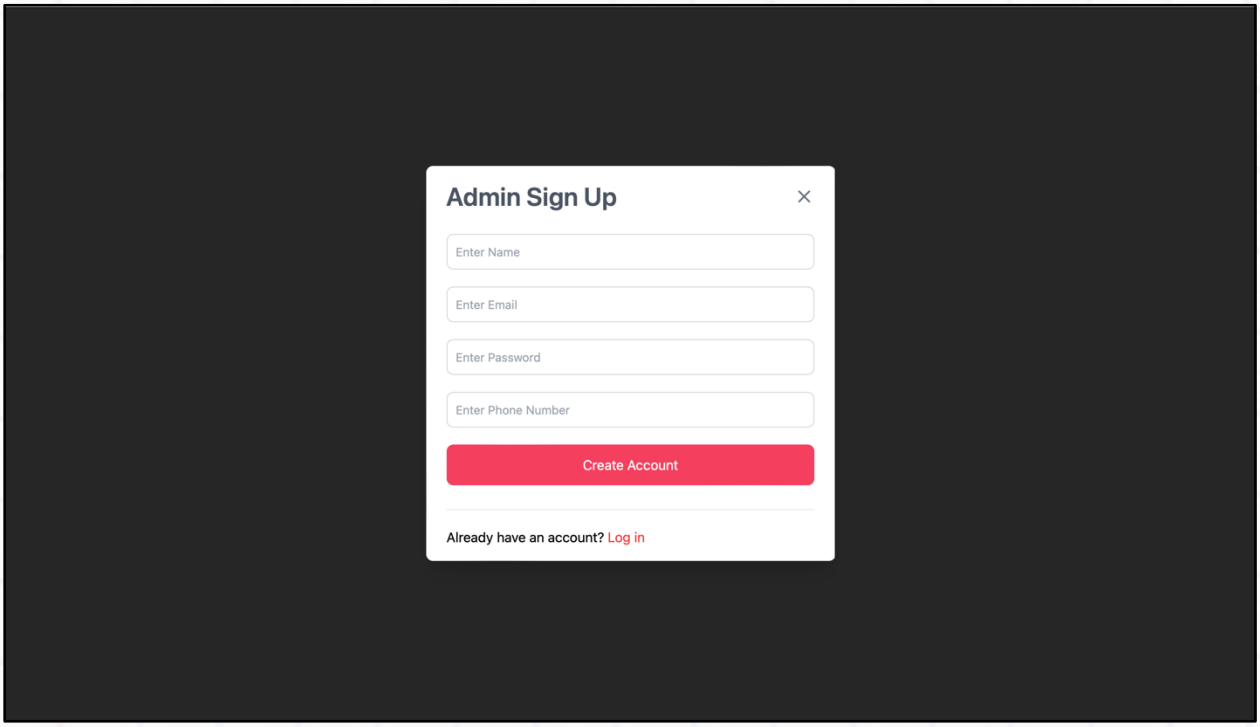
- Displays error messages and success notifications.
- Provides a button to close the modal and navigate to the "Partner With Us" page.
- Includes a link to navigate to the admin login page if the user already has an account.

```
return (

        <>

            <ToastContainer />

            <div className="relative z-10" aria-labelledby="modal-title"
role="dialog" aria-modal="true">

                <div className="fixed inset-0 bg-black bg-opacity-85
transition-opacity"></div>

                <div className="fixed inset-0 z-10 w-screen overflow-y-auto">

                    <div className="flex min-h-full items-end justify-center
p-4 text-center sm:items-center sm:p-0">

                        <div className="relative transform overflow-hidden
rounded-lg bg-white text-left shadow-xl transition-all sm:my-8 sm:w-97
sm:max-w-lg">

                            <div className="bg-white px-4 pb-4 pt-5 sm:p-6
sm:pb-4">

                                <div className='flex'>

                                    <h3 className="text-3xl font-semibold
leading-6 text-gray-600" id="modal-title">Admin Sign Up</h3>

                                    <button

                                        onClick={handleClose}

                                        className="text-gray-500"

                                    >
```

```
                                    <svg
xmlns="http://www.w3.org/2000/svg"
                                        className="h-6 w-6 ml-52"
                                        fill="none"
                                        viewBox="0 0 24 24"
                                        stroke="currentColor"
                                    >
                                        <path
                                            strokeLinecap="round"
                                            strokeLinejoin="round"
                                            strokeWidth="2"
                                            d="M6 18L18 6M6 6l12 12"
                                        />
                                    </svg>
                                </button>
                            </div>
                            {error && <div className="text-red-500 text-
sm">{error}</div>}
                            <input onChange={(e) =>
setName(e.target.value)} className="mt-8 outline-none border border-gray-300
text-gray-900 text-sm rounded-lg block w-full p-2.5" placeholder="Enter Name"
required />
                            <input onChange={(e) =>
setEmail(e.target.value)} className="mt-5 outline-none border border-gray-300
```

```
text-gray-900 text-sm rounded-lg block w-full p-2.5" placeholder="Enter
Email" required />
                                <input type='password' onChange={(e) =>
setPassword(e.target.value)} className="mt-5 outline-none border border-gray-
300 text-gray-900 text-sm rounded-lg block w-full p-2.5" placeholder="Enter
Password" required />
                                <input onChange={(e) =>
setPhone(e.target.value)} className="mt-5 outline-none border border-gray-300
text-gray-900 text-sm rounded-lg block w-full p-2.5" placeholder="Enter Phone
Number" required />
                                <button onClick={emailSignUp} className="mt-5
mb-3 bg-rose-500 w-full h-12 text-white py-2 px-4 rounded-lg">
                                    Create Account
                                </button>
                                <hr className='mt-4' />
                                <div className='text-base mt-5'>Already have
an account? <Link to='/admin/login'><span className='text-red-500'>Log
in</span></Link></div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </>
    );
```

```
}


export default AdminSignup;
```



6.20. Admin Login Component

Create a new file inside the components folder and name it AdminLogin.tsx.

1. Imports

- `useState` is used to manage the component's state.
- `RecaptchaVerifier`, `signInWithPhoneNumber`, and `signInWithPopup` are Firebase functions for authentication.
- `auth` and `googleAuthProvider` are Firebase configuration objects.
- `PhoneInput` is a component for phone number input.
- `ToastContainer` and `toast` from `react-toastify` are used for notifications.
- `GoogleIcon` and `EmailIcon` are icons used in the UI.

```
import React, { useState } from 'react';
import { RecaptchaVerifier, signInWithPhoneNumber, signInWithPopup } from
'firebase/auth';
import { auth, googleAuthProvider } from '../firebase/setup';
import { Link, useNavigate } from 'react-router-dom';
import PhoneInput from 'react-phone-input-2';
import 'react-phone-input-2/lib/style.css';
import GoogleIcon from '../images/google-icon.png';
import EmailIcon from '../images/email-icon.png';
import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
```

2. State Variables

- State variables store user input and Firebase confirmation results.
- phone stores the phone number entered by the user.
- otp stores the one-time password sent to the user.
- confirmationResult stores the result of the OTP confirmation.

```
const AdminLogin = () => {

    const navigate = useNavigate();

    const [phone, setPhone] = useState("");

    const [otp, setOtp] = useState("");

    const [confirmationResult, setConfirmationResult] = useState<any>(null);
```

3. `verifyPhoneNumber` Function

- This function checks if the phone number exists in the backend.
- Sends a GET request to check if the phone number is registered.
- Returns true if the phone number exists, false otherwise.

```
const verifyPhoneNumber = async (phone: string) => {
        try {
            const response = await
fetch(`http://localhost:8090/api/users/phone/verify?phone=${phone}`, {
                method: 'GET',
                headers: {
                    'Content-Type': 'application/json'
                }
            });
            const data = await response.json();
            return data.exists;
        } catch (error) {
            console.error('Failed to verify phone number:', error);
            return false;
        }
    };
```

4. `sendOtp` Function

- This function sends an OTP to the user's phone number.
- Verifies if the phone number exists.
- If valid, sends an OTP and sets confirmationResult.
- Displays appropriate success or error messages.

```
const sendOtp = async () => {
        try {
            const phoneExists = await verifyPhoneNumber(phone);

            if (!phoneExists) {
```

```
                toast.error('Failed to Verify Phone Number!');
                return;
            }

            const recaptcha = new RecaptchaVerifier(auth, "recaptcha", {});
            const confirmationResult = await signInWithPhoneNumber(auth,
phone, recaptcha);
            setConfirmationResult(confirmationResult);
            toast.success('OTP Sent Successfully!');
        } catch (err: any) {
            console.error(err);
            toast.error('Failed to Send OTP!');
        }
    };
```

5. `verifyOtp` Function

This function verifies the OTP entered by the user.

- Confirms the OTP using confirmationResult.
- If successful, stores the token and redirects to the partner page.
- Displays appropriate success or error messages.

```
const verifyOtp = async () => {
        try {
            if (!confirmationResult) throw new Error("Confirmation Result Not
Found");
            await confirmationResult.confirm(otp);
            const user = auth.currentUser;
```

```
            if (user) {
                const token = await user.getIdToken();
                localStorage.setItem("token", token);
                toast.success('Logged In with OTP Successfully!');
                setTimeout(() => {
                    navigate("/partner-with-us");
                }, 2000);
            } else {
                toast.error('User Authentication Failed!');
            }
        } catch (err: any) {
            console.error(err);
            toast.error('Failed to Verify OTP!');
        }
    };
```

6. `googleSignIn` Function

This function handles Google sign-in.

- Uses Google sign-in via a popup.
- If successful, stores the token and redirects to the partner page.
- Displays appropriate success or error messages.

```
const googleSignIn = async () => {
        try {
            const data = await signInWithPopup(auth, googleAuthProvider);
            const user = auth.currentUser;
            if (user) {
```

```
                const token = await user.getIdToken();
                localStorage.setItem("token", token);
                toast.success('Logged In with Google Successfully!');
                setTimeout(() => {
                    navigate("/partner-with-us");
                }, 2000);
            } else {
                toast.error('User Authentication Failed!');
            }
        } catch (err: any) {
            console.error(err);
            toast.error('Failed to Sign In with Google!');
        }
    };
```

7. `handleClose` Function

This function handles closing the modal and navigating back.

```
const handleClose = () => {
        navigate('/partner-with-us');
    };
```

8. Return JSX

- **Modal Dialog**: Renders the modal dialog with a background overlay.
- **Phone Number Input**: Displays PhoneInput for entering a phone number.
- **OTP Verification**: Shows OTP input and verification button if phone number is provided.
- **Alternative Login Options**: Provides options to log in with Google or email if no phone number is entered.

- **Navigation**: Includes links for creating an account and redirects to the partner page on successful login.

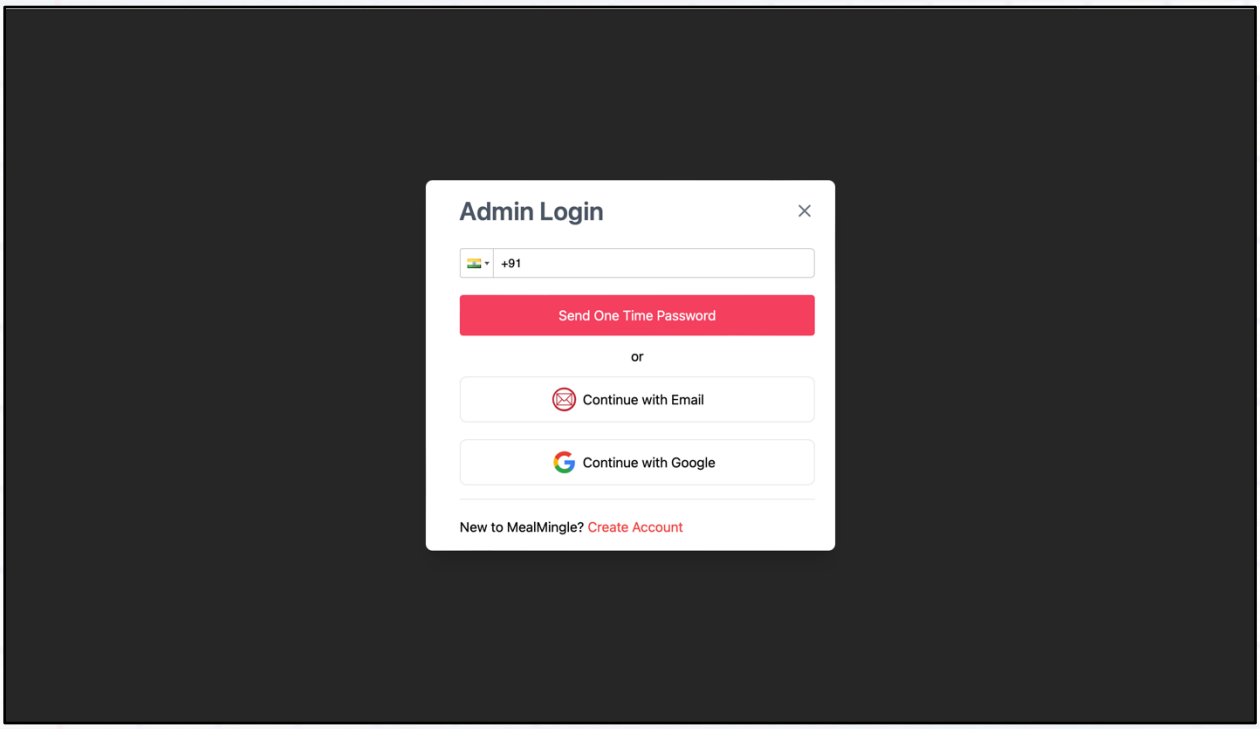
```
return (
        <div className="relative z-10" aria-labelledby="modal-title"
role="dialog" aria-modal="true">
            <ToastContainer />
            <div className="fixed inset-0 bg-black bg-opacity-85 transition-
opacity"></div>
            <div className="fixed inset-0 z-10 w-screen overflow-y-auto">
                <div className="flex min-h-full items-end justify-center p-4
text-center sm:items-center sm:p-0">
                    <div className="relative transform overflow-hidden
rounded-lg bg-white text-left shadow-xl transition-all sm:my-8 sm:w-97
sm:max-w-lg">
                        <div className="bg-white px-4 pb-4 pt-5 sm:p-6 sm:pb-
4">
                            <div className="sm:flex sm:items-start">
                                <div className="mt-3 text-center sm:ml-4
sm:mt-0 sm:text-left">
                                    <div className='flex '>
                                        <h3 className="text-3xl font-semibold
leading-6 text-gray-600" id="modal-title">Admin Login</h3>
                                        <button
                                            onClick={handleClose}
                                            className="text-gray-500"
                                        >
```

```
                                    <svg


xmlns="http://www.w3.org/2000/svg"

                                        className="h-6 w-6 ml-56"

                                        fill="none"

                                        viewBox="0 0 24 24"

                                        stroke="currentColor"

                                    >

                                        <path

                                            strokeLinecap="round"

                                            strokeLinejoin="round"

                                            strokeWidth="2"

                                            d="M6 18L18 6M6 6l12 12"

                                        />

                                    </svg>

                                </button>

                            </div>

                            <div className='mt-8'>

                                <PhoneInput

                                    country={'in'}

                                    value={phone}

                                    onChange={(phone) => setPhone("+"

+ phone)}

                                    buttonStyle={{ backgroundColor:

"white" }}
```

```
                                        inputStyle={{ width: "100%" }} />
                                    </div>
                                    <button onClick={sendOtp} className="mt-5
mb-3 bg-rose-500 w-full h-12 text-white py-2 px-4 rounded">
                                        Send One Time Password
                                    </button>
                                    <div id="recaptcha"></div>
                                    {phone && <input onChange={(e) =>
setOtp(e.target.value)} className="mb-3 mt-3 outline-none border border-gray-
300 text-gray-900 text-sm rounded-sm block w-full p-2.5" placeholder="Enter
OTP" required />}
                                    {otp && <button onClick={verifyOtp}
className="mt-5 mb-3 bg-rose-500 w-full h-12 text-white py-2 px-4 rounded">
                                        Verify One Time Password
                                    </button>}
                                    {!phone && <div>
                                        <div className='text-center mb-
3'>or</div>
                                        <Link to='/admin/emailLogin'><div
className='flex items-center text-center border border-spacing-1 rounded-lg
p-3'>
                                            <img src={EmailIcon} alt='Email
Icon' className='w-7 h-7 ml-24' />
                                            <button className='ml-2'>Continue
with Email</button>
```

```
                                    </div>
                                    </Link>
                                    <div onClick={googleSignIn}
classNameName='mt-5 flex items-center text-center border border-spacing-1
rounded-lg p-3'>
                                        <img src={GoogleIcon} alt='Google
Icon' classNameName='w-7 h-7 ml-24' />
                                        <button classNameName='ml-2'>Continue
with Google</button>
                                    </div>
                                </div>}
                                <hr classNameName='mt-4' />
                                <div classNameName='text-base mt-5'>New to
MealMingle? <Link to='/admin/signup'><span classNameName='text-red-500'>Create
Account</span></Link></div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};
```

```
export default AdminLogin;
```



6.21. Admin Email Login Component

- The AdminEmailLogin component is a React functional component that provides a login form for admin users to sign in using their email and password.
- It uses Firebase for authentication and react-toastify for displaying notifications.

Create a new file inside the components folder and name it AdminEmailLogin.tsx.

1. Imports
   - useState is used for managing component state.
   - Link and useNavigate are from react-router-dom for navigation.
   - signInWithEmailAndPassword and auth are from Firebase for email authentication.
   - ToastContainer and toast from react-toastify are used for displaying notifications.
   - EmailInboxIcon is an icon used in the UI.

```
import React, { useState } from 'react';
```

```
import { Link, useNavigate } from 'react-router-dom';

import EmailInboxIcon from '../images/email-inbox-icon.png';

import { signInWithEmailAndPassword } from 'firebase/auth';

import { auth } from '../firebase/setup';

import { ToastContainer, toast } from 'react-toastify';

import 'react-toastify/dist/ReactToastify.css';
```

2. State Variables

State variables manage the login form's data and error messages.

- email stores the email entered by the user.
- password stores the password entered by the user.
- error stores any error message to display.

```
const AdminEmailLogin = () => {

    localStorage.removeItem('token');

    const navigate = useNavigate();

    const [email, setEmail] = useState("");

    const [password, setPassword] = useState("");

    const [error, setError] = useState<string | null>(null);
```

3. `loginAdminToPortal` Function

This function sends the login request to the backend and handles responses.

- Sends a POST request to the backend with email and password.
- If the response contains an error, it shows an error message using toast.error.
- If successful, it stores the token in localStorage, shows a success message, and navigates to the /partner-with-uspage after a short delay.

```
  const loginAdminToPortal = async () => {
```

```
        const adminData = { userEmail: email, userPassword: password };
        const response = await
fetch('http://localhost:8090/api/users/login/admin', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify(adminData)
        })
        const data = await response.json();
        console.log(data);
        if (data.error != "") {
            toast.error(data.message);
        }
        else {
            toast.success(data.message);
            localStorage.setItem('token', data.data.token);
            setTimeout(() => {
                navigate("/partner-with-us");
            }, 2000);
        }
    }
```

4. `emailLogin` Function

This function handles the login process.

- Calls loginAdminToPortal to perform the login.
- Catches and handles any errors, displaying an error message using toast.error.

```
const emailLogin = async () => {

        try {

            loginAdminToPortal();

        } catch (err: any) {

            console.error(err);

            toast.error(err.message || "An Unknown Error Occurred during

Login");

        }

    }
```

5. `handleClose` **Function**

This function handles closing the modal and navigating back.

```
const handleClose = () => {

        navigate('/partner-with-us');

    };
```

6. Return JSX

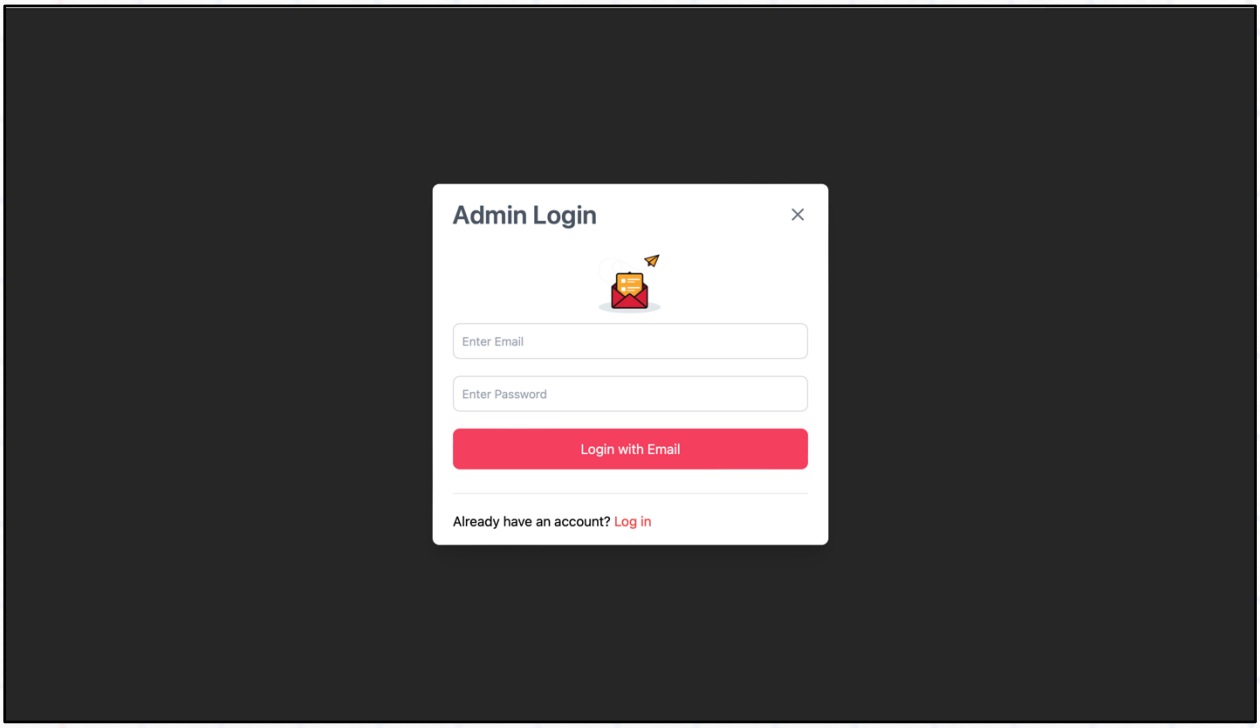
- **Modal Dialog**: Renders the modal dialog with a background overlay.
- **Login Form**: Includes email and password input fields and a login button.
- **Notifications**: Displays toast notifications for success or error messages.
- **Navigation**: Includes a link to navigate to the admin login page if the user already has an account.

```
return (

        <div className="relative z-10" aria-labelledby="modal-title"

role="dialog" aria-modal="true">
```

```
            <ToastContainer />
            <div className="fixed inset-0 bg-black bg-opacity-85 transition-
opacity"></div>
            <div className="fixed inset-0 z-10 w-screen overflow-y-auto">
                <div className="flex min-h-full items-end justify-center p-4
text-center sm:items-center sm:p-0">
                    <div className="relative transform overflow-hidden
rounded-lg bg-white text-left shadow-xl transition-all sm:my-8 sm:w-97
sm:max-w-lg">
                        <div className="bg-white px-4 pb-4 pt-5 sm:p-6 sm:pb-
4">
                            <div className='flex'>
                                <h3 className="text-3xl font-semibold
leading-6 text-gray-600" id="modal-title">Admin Login</h3>
                                <button
                                    onClick={handleClose}
                                    className="text-gray-500"
                                >
                                    <svg
                                        xmlns="http://www.w3.org/2000/svg"
                                        className="h-6 w-6 ml-56"
                                        fill="none"
                                        viewBox="0 0 24 24"
                                        stroke="currentColor"
                                    >
```

```
                                    <path
                                        strokeLinecap="round"
                                        strokeLinejoin="round"
                                        strokeWidth="2"
                                        d="M6 18L18 6M6 6l12 12"
                                    />
                                </svg>
                            </button>
                        </div>
                        <img src={EmailInboxIcon} alt='Email Inbox Icon'
classNameName='w-24 h-24 mt-5 ml-40' />
                        <input onChange={(e) => setEmail(e.target.value)}
classNameName="outline-none border border-gray-300 text-gray-900 text-sm rounded-
lg block w-full p-2.5" placeholder="Enter Email" required />
                        <input type='password' onChange={(e) =>
setPassword(e.target.value)} classNameName="mt-5 outline-none border border-gray-
300 text-gray-900 text-sm rounded-lg block w-full p-2.5" placeholder="Enter
Password" required />
                        <button onClick={emailLogin} classNameName="mt-5 mb-3
bg-rose-500 w-full h-12 text-white py-2 px-4 rounded-lg">
                            Login with Email
                        </button>
                        {error && <div classNameName="text-red-500 mt-
2">{error}</div>}
                        <hr classNameName='mt-4' />
```

```
                        <div className='text-base mt-5'>Already have an
account? <Link to='/admin/login'><span className='text-red-500'>Log
in</span></Link></div>
                    </div>
                </div>
            </div>
        </div>
    );
}

export default AdminEmailLogin;
```



6.22. Admin Navbar Component

- The AdminNavbar component is a React functional component that serves as a navigation bar for the admin panel.
- It includes user information, navigation links, and a logout button. It also handles fetching and displaying bank details.

Create a new file inside the components folder and name it AdminNavbar.tsx.

1. Imports
   - React, useEffect, and useState are from React.
   - Avatar is a component for displaying user avatars.
   - auth, onAuthStateChanged, and signOut are from Firebase for authentication.
   - Link is from react-router-dom for navigation.
   - LogoutIcon is an image used for the logout button.
   - ToastContainer and toast from react-toastify are used for notifications.
   - useNavigate is from react-router-dom for programmatic navigation.

```
import React, { useEffect, useState } from 'react';

import Avatar from 'react-avatar';

import { auth } from '../firebase/setup';

import { onAuthStateChanged, signOut } from 'firebase/auth';

import { Link } from 'react-router-dom';

import LogoutIcon from '../images/logout-icon.png';

import { toast, ToastContainer } from 'react-toastify';

import 'react-toastify/dist/ReactToastify.css';

import { useNavigate } from 'react-router-dom';
```

2. Interfaces

Define TypeScript interfaces for the data structures used.

- Admin represents the admin's information.
- BankDetails represents the bank details associated with the admin.

```
interface Admin {
    adminId: string;
    adminName: string;
    adminEmail: string;
    adminPhone: string;
}


interface BankDetails {
    user: Admin;
    accountNumber: string;
    bankName: string;
    branchName: string;
    ifscCode: string;
    panNumber: string;
    aadhaarNumber: string;
    gstNumber: string;
}
```

3. State Variables

Manage component state using hooks.

- authStore stores the current user's authentication state.
- bankDetails stores the fetched bank details.
- navigate is used for navigation.

```
const AdminNavbar = () => {
    const initialBankDetails: BankDetails = {
        user: {
```

```
            adminId: '',
            adminName: '',
            adminEmail: '',
            adminPhone: ''
        },
        accountNumber: '',
        bankName: '',
        branchName: '',
        ifscCode: '',
        panNumber: '',
        aadhaarNumber: '',
        gstNumber: '',
    };

    const [authStore, setAuthStore] = useState<any>({});
    const [bankDetails, setBankDetails] =
useState<BankDetails>(initialBankDetails);
    const navigate = useNavigate();
```

4. `useEffect` Hooks

Handle side effects like authentication state changes and fetching bank details.

- Listens for authentication state changes and updates authStore accordingly.
- Cleans up the subscription when the component unmounts.

```
useEffect(() => {
        const unsubscribe = onAuthStateChanged(auth, (user) => {
```

```
            setAuthStore(user || {});
        });


        return () => unsubscribe();
    }, []);
```

- Fetches bank details from the server and updates bankDetails state if the response is successful.

```
useEffect(() => {
        async function fetchBankDetails() {
            const response = await
fetch('http://localhost:8090/api/users/details', {
                method: 'GET',
                headers: {
                    'Content-Type': 'application/json',
                    'Authorization': `Bearer
${localStorage.getItem('token')}`
                }
            });
            const data = await response.json();
            if (data.error == "" && data.data != null) {
                console.log(data.data);
                setBankDetails(data.data);
            }
        }
        fetchBankDetails();
```

```
    }, [])
```

5. `logout` Function

Handles user logout and navigates to another page.

- Signs out the user using Firebase.
- Shows a success message using toast.success.
- Navigates to /partner-with-us after a short delay.
- Clears localStorage to remove stored tokens.

```
const logout = async () => {

        try {

            await signOut(auth);

            toast.success('Logged Out Successfully!');

            setTimeout(() => {

                navigate('/partner-with-us');

            }, 2000);

        } catch (err) {

            console.error(err);

        }

        localStorage.clear();

    };
```

6. Return JSX

- **Header**: Displays a heading with a link to the /partner-with-us page.
- **User Info**: Shows user photo or avatar, name, and phone number if available.
- **Navigation Buttons**: Includes links to login and sign-up pages if the user is not authenticated.
- **Bank Details**: Shows a button to view bank account summary if aadhaarNumber is not empty.
- **Logout**: Displays a logout button if the user is authenticated.

```
return (
        <>
            <ToastContainer />
            <div className='flex items-center justify-between p-4'>
                <Link to='/partner-with-us'>
                    <h1 className='text-3xl font-extrabold italic ml-
20'>MealMingle for Business</h1>
                </Link>
                <div className='flex items-center ml-auto'>
                    {authStore?.photoURL ? (
                        <img src={authStore.photoURL} alt='User Pic'
className='w-12 h-12 rounded-full' />
                    ) : authStore?.displayName ? (
                        <Avatar name={authStore.displayName} round={true}
size='40' className='' />
                    ) : null}
                    <div className='ml-2 mr-10'>
                        {authStore?.displayName ? authStore.displayName :
authStore?.email ? authStore.email : ''}
                    </div>
                    {authStore?.phoneNumber && <div className="text-gray-600
text-lg ml-2">{authStore.phoneNumber}</div>}
                    {!auth.currentUser?.email &&
!auth.currentUser?.phoneNumber && (
                        <Link to='/admin/login'>
```

```
                            <button
                                className={`px-4 py-2 rounded-md border bg-
white text-black border-gray-300 shadow-md mr-10 cursor-pointer`}>
                                Login
                            </button>
                        </Link>
                    )}
                    {!auth.currentUser?.email &&
!auth.currentUser?.phoneNumber && (
                        <Link to='/admin/signup'>
                            <button
                                className={`px-4 py-2 rounded-md border bg-
white text-black border-gray-300 shadow-md mr-32 cursor-pointer`}>
                                Sign Up
                            </button>
                        </Link>
                    )}
                    {bankDetails.aadhaarNumber !== '' && (
                        <button
                            onClick={() => navigate('/view-admin-bank-
details', { state: { bankDetails } })}
                            className='relative mr-12 shadow-lg p-2 rounded-
xl text-black cursor-pointer w-15 h-15'
                        >
                            Bank Account Summary
```

```
                        </button>
                    )}
                    {auth.currentUser && (
                        <div>
                            <img onClick={logout} src={LogoutIcon}
alt='Logout Icon' className='shadow-lg p-2 rounded-xl text-gray-600 cursor-
pointer w-10 h-10 mr-20' />
                        </div>
                    )}
                </div>
            </div>
        </>
    );
};

export default AdminNavbar;
```

In the absence of the Bank Details, the Admin Navbar would look like:

6.23. Register Restaurant Component

- It's a form used for registering a new restaurant, where you can fill in details like the restaurant's name, address, phone number, and other relevant information.
- The component also handles form submission, validation, and navigation.

Create a new file inside the components folder and name it `RegisterRestaurant`.tsx.

1. Imports

- **React**: A library for building user interfaces.
- **useState**: A React hook to manage state (data that changes).
- **ChangeEvent** and **FormEvent**: TypeScript types for event handling.
- **useNavigate**: A React Router hook to navigate to different pages.
- **AdminNavbar**: A component to display the navigation bar.
- **ToastContainer** and **toast**: For showing notifications to the user.

```
import React, { useState, ChangeEvent, FormEvent } from 'react';

import { useNavigate } from 'react-router-dom';

import AdminNavbar from './AdminNavbar';

import { ToastContainer, toast } from 'react-toastify';

import 'react-toastify/dist/ReactToastify.css';
```

2. **TypeScript Interfaces**

These define the structure of the data used in the component.

- **Address**: Defines the structure of the restaurant's address.
- **RestaurantData**: Defines the complete data structure for a restaurant.
- **RegisterRestaurantProps**: Props (properties) the component can accept.

```
interface Address {

    streetNumber: string;

    streetName: string;

    city: string;

    country: string;

}



interface RestaurantData {

    restaurantId: string;

    restaurantName: string;
```

```
    restaurantAddress: Address;
    restaurantRating: number;
    restaurantMinimumOrderAmount: number;
    restaurantDiscountPercentage: number;
    restaurantAvailability: boolean;
    restaurantImageUrl: string;
    restaurantOperationDays: string;
    restaurantOperationHours: string;
    restaurantPhoneNumber: string;
    restaurantOwnerMail: string;
}


interface RegisterRestaurantProps {
    onSubmit?: () => void;
}
```

3. Component Definition and Initial State

- **initialRestaurantData**: Sets the initial state of the restaurant data with empty or default values.
- **restaurantData**: State to store and manage the restaurant data.
- **errors**: State to store validation errors.
- **navigate**: Used for programmatically navigating to different routes.

```
const RegisterRestaurant: React.FC<RegisterRestaurantProps> = ({ onSubmit })
=> {
    const initialRestaurantData: RestaurantData = {
        restaurantId: '',
        restaurantName: '',
        restaurantAddress: {
```

```
            streetNumber: '',
            streetName: '',
            city: '',
            country: 'India',
        },
        restaurantRating: 1,
        restaurantMinimumOrderAmount: 0,
        restaurantDiscountPercentage: 0,
        restaurantAvailability: true,
        restaurantImageUrl: '',
        restaurantOperationDays: '',
        restaurantOperationHours: '',
        restaurantPhoneNumber: '',
        restaurantOwnerMail: '',
    };

    const [restaurantData, setRestaurantData] =
useState<RestaurantData>(initialRestaurantData);
    const [errors, setErrors] = useState<string[]>([]);
    const navigate = useNavigate();
```

4. **Save Restaurant Data**

saveRestaurantData: An asynchronous function that sends the restaurant data to a server using `fetch` and displays success or error messages.

```
async function saveRestaurantData() {
        const postRestaurantData = {
            restaurant: {
```

```
                restaurantName: restaurantData.restaurantName,
                restaurantAddress: {
                    pincode: restaurantData.restaurantAddress.streetNumber,
                    streetName: restaurantData.restaurantAddress.streetName,
                    city: restaurantData.restaurantAddress.city,
                    country: restaurantData.restaurantAddress.country
                },
                restaurantRating: 1,
                restaurantMinimumOrderAmount:
restaurantData.restaurantMinimumOrderAmount,
                restaurantDiscountPercentage: 10,
                restaurantAvailability: restaurantData.restaurantAvailability
? "Open" : "Closed",
                restaurantImageUrl: restaurantData.restaurantImageUrl,
                restaurantOperationDays:
restaurantData.restaurantOperationDays,
                restaurantOperationHours:
restaurantData.restaurantOperationHours,
                restaurantPhoneNumber: restaurantData.restaurantPhoneNumber,
            }
        }
        const response = await fetch('http://localhost:8091/api/restaurants',
{
            method: 'POST',
            headers: {
```

```
                'Content-Type': 'application/json',
                'Authorization': `Bearer ${localStorage.getItem('token')}`
            },
            body: JSON.stringify(postRestaurantData)
        })
        const data = await response.json();
        if (data.error === "") {
            toast.success(data.message);
            setRestaurantData(data.data.restaurant);
        }
        else {
            toast.error(data.message);
        }
    }
```

5. **Handle Input Changes**

handleChange: Updates the `restaurantData` state when the user types in the form fields.

```
    const handleChange = (e: ChangeEvent<HTMLInputElement |
HTMLTextAreaElement | HTMLSelectElement>) => {
        const { name, value } = e.target;
        if (name.includes('.')) {
            const [parent, child] = name.split('.');
            setRestaurantData(prevData => ({
                ...prevData,
                [parent as keyof RestaurantData]: {
                    ...(prevData[parent as keyof RestaurantData] as Address),
```

```
                    [child]: value,
                },
            }));
        }
        else {
            if (name === 'restaurantMinimumOrderAmount' || name ===
'restaurantDiscountPercentage') {
                const numericValue = parseFloat(value);
                if (numericValue >= 0) {
                    setRestaurantData({ ...restaurantData, [name]:
numericValue });
                }
            }
            else {
                setRestaurantData({ ...restaurantData, [name]: value });
            }
        }
    };
```

6. **Handle Form Submission**

handleSubmit: Prevents the default form submission, validates the form, and if valid, calls `saveRestaurantData`. Shows a success message and redirects to the admin restaurants view after 2 seconds.

```
const handleSubmit = (e: FormEvent<HTMLFormElement>) => {
        e.preventDefault();

        const isValid = validateForm();
```

```
        if (!isValid) {
            return;
        }

        if (onSubmit) onSubmit();
        toast.success('Restaurant Added Successfully!');
        setTimeout(() => {
            navigate('/view-admin-restaurants');
        }, 2000);
    };
```

7. **Handle Close**

handleClose: Redirects to the "Partner With Us" page when the close button is clicked.

```
const handleClose = () => {
        navigate('/partner-with-us');
    };
```

8. **Validate Form**

validateForm: Checks if all form fields have valid values. If there are errors, they are displayed using `toast`.

```
const validateForm = () => {
        let isValid = true;
        const errors: string[] = [];

        // Restaurant Name
```

```
        if (!restaurantData.restaurantName.trim()) {
            errors.push('Restaurant Name is Required.');
            isValid = false;
        }
        else if (/^\d+$/.test(restaurantData.restaurantName.trim())) {
            errors.push('Restaurant Name must be a String, not a Number.');
            isValid = false;
        }
        else if (restaurantData.restaurantName.trim().length < 5 ||
restaurantData.restaurantName.trim().length > 20) {
            errors.push('Restaurant Name must be between 5 to 20 Characters
long.');
            isValid = false;
        }

        // Street Number
        if (!restaurantData.restaurantAddress.streetNumber.trim()) {
            errors.push('Street Number is Required.');
            isValid = false;
        }
        else if
(isNaN(Number(restaurantData.restaurantAddress.streetNumber))) {
            errors.push('Street Number must be a Number, not a String.');
            isValid = false;
        }
```

```
        // Street Name
        if (!restaurantData.restaurantAddress.streetName.trim()) {
            errors.push('Street Name is Required.');
            isValid = false;
        }
        else if
(/^\d+$/.test(restaurantData.restaurantAddress.streetName.trim())) {
            errors.push('Street Name must be a String, not a Number.');
            isValid = false;
        }
        else if (restaurantData.restaurantAddress.streetName.trim().length <
5 || restaurantData.restaurantAddress.streetName.trim().length > 30) {
            errors.push('Street Name must be between 5 to 30 Characters
long.');
            isValid = false;
        }

        // City
        if (!restaurantData.restaurantAddress.city.trim()) {
            errors.push('City is Required.');
            isValid = false;
        }

        // Operation Days
```

```
        const operationDays = restaurantData.restaurantOperationDays.trim();


        if (!operationDays) {
            errors.push('Operation Days are Required.');
            isValid = false;
        }
        else {
            const operationDaysRegex = /^[a-zA-Z]{3}-[a-zA-Z]{3}$/;
            if (!operationDaysRegex.test(operationDays)) {
                errors.push('Operation Days must be of the format: Mon-
Fri.');
                isValid = false;
            }
            else {
                const [startDay, endDay] = operationDays.split('-');
                const daysOfWeek = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat",
"Sun"];


                if (!daysOfWeek.includes(startDay) ||
!daysOfWeek.includes(endDay)) {
                    errors.push('Invalid Days Provided. Days must be one of
Mon, Tue, Wed, Thu, Fri, Sat, Sun.');
                    isValid = false;
                }
                else if (startDay === endDay) {
```

```
                    errors.push('Start Day and End Day must not be the
Same.');
                    isValid = false;
                }
            }
        }

        // Operation Hours
        const operationHours =
restaurantData.restaurantOperationHours.trim();

        if (!operationHours) {
            errors.push('Operation Hours are Required.');
            isValid = false;
        }
        else {
            const operationHoursRegex = /^(0?[1-9]|1[0-2]):[0-5][0-9]([AP]M)-
(0?[1-9]|1[0-2]):[0-5][0-9]([AP]M)$/;

            if (!operationHoursRegex.test(operationHours)) {
                errors.push('Operation Hours must be in the format: 10:00AM-
06:00PM.');
                isValid = false;
            }
            else {
```

```
                const [start, end] = operationHours.split('-');

                const convertTo24Hour = (time: any) => {
                    let [hours, minutes] = time.slice(0, -2).split(':');
                    const period = time.slice(-2);
                    hours = parseInt(hours, 10);
                    minutes = parseInt(minutes, 10);
                    if (period === 'PM' && hours !== 12) {
                        hours += 12;
                    }
                    else if (period === 'AM' && hours === 12) {
                        hours = 0;
                    }
                    return hours * 60 + minutes;
                };

                const startTime = convertTo24Hour(start);
                const endTime = convertTo24Hour(end);

                if (startTime === endTime) {
                    errors.push('Start Time and End Time must not be the
Same.');
                    isValid = false;
                }
                else if (startTime >= endTime) {
```

```
                    errors.push('End Time must be after Start Time.');
                    isValid = false;
                }
            }
        }

        // Image URL
        if (!restaurantData.restaurantImageUrl.trim()) {
            errors.push('Image URL is Required.');
            isValid = false;
        }

        // Phone Number
        if (!restaurantData.restaurantPhoneNumber.trim()) {
            errors.push('Phone Number is Required.');
            isValid = false;
        }
        else if (!/^\d+$/.test(restaurantData.restaurantPhoneNumber.trim()))
{
            errors.push('Phone Number must contain only Numeric
Characters.');
            isValid = false;
        }
        else if (restaurantData.restaurantPhoneNumber.trim().length !== 10) {
            errors.push('Phone Number must be exactly 10 Digits.');
```

```
            isValid = false;
        }
        else if (!/^[6-
9]\d{9}$/.test(restaurantData.restaurantPhoneNumber.trim())) {
            errors.push('Phone Number must start with 6, 7, 8, or 9');
            isValid = false;
        }

        setErrors(errors);
        if (errors.length > 0) {
            toast.error(errors[0]);
        }

        if (isValid) {
            saveRestaurantData();
        }

        return isValid;
    };
```

9. `cityOptions` Array

This array defines a list of cities that will be available for selection in the dropdown menu for the city field in the form.

```
const cityOptions = [
        'Delhi',
```

```
        'Mumbai',
        'Chennai',
        'Bangalore',
        'Jaipur'
    ];
```

10. Return JSX

1. **<AdminNavbar />**:

- This component is rendered at the top of the form, likely providing navigation options specific to admin users.

2. **Form Structure**:

- The form includes various input fields for registering a new restaurant, such as:
  - **Restaurant Name**
  - **Street Number and Street Name**
  - **City**: Populated using the `cityOptions` array, allowing the user to select from predefined cities.
  - **Country**: Displayed as read-only.
  - **Operation Days and Hours**
  - **Image URL**
  - **Phone Number**
  - **Minimum Order Amount and Discount Percentage**

3. **Handling Form Submission**:

- The `onSubmit` event of the form triggers the `handleSubmit` function, which is expected to handle the form submission logic.

4. **Form Inputs**:

- Each input field is controlled by React state, managed through the `restaurantData` object.
- The `onChange` event handler (`handleChange`) updates the state as the user types in the form fields.

5. **City Dropdown**:

- The `<select>` element uses the `cityOptions` array to populate the list of available cities.
- It maps over `cityOptions` to create `<option>` elements for each city.

6. **Styling**:

- The form is styled using Tailwind CSS classes to ensure a consistent look and feel.

7. **Button**:

- A submit button with a class for styling and handling form submission.

```
return (
        <>
            <AdminNavbar />
            <div className="flex justify-center items-center min-h-
screen">
                <div className="w-full max-w-2xl p-6 bg-white rounded-lg
shadow-md relative">
                    <div className="absolute top-0 right-0 m-4">
                        <button
                            onClick={handleClose}
                            className="text-gray-500"
                        >
                            <svg
                                xmlns="http://www.w3.org/2000/svg"
                                className="h-6 w-6"
                                fill="none"
                                viewBox="0 0 24 24"
```

```
                            stroke="currentColor"
                        >
                            <path
                                strokeLinecap="round"
                                strokeLinejoin="round"
                                strokeWidth="2"
                                d="M6 18L18 6M6 6l12 12"
                            />
                        </svg>
                    </button>
                </div>
                <h3 className="text-3xl font-semibold text-gray-800
mb-4">
                    Register New Restaurant
                </h3>
                <form onSubmit={handleSubmit} className="space-y-4">
                    <div className="grid grid-cols-2 gap-4">
                        <div className="col-span-2">
                            <label className="block text-base font-
medium text-gray-700">
                                Restaurant Name
                            </label>
                            <input
                                type="text"
                                name="restaurantName"
```

```
                                    value={restaurantData.restaurantName}
                                    onChange={handleChange}
                                    className="form-input mt-1 block w-
full rounded-md shadow-sm border h-10"
                                />
                            </div>
                        </div>
                        <div className="grid grid-cols-2 gap-4">
                            <div className="col-span-1">
                                <label className="block text-base font-
medium text-gray-700">
                                    Street Number
                                </label>
                                <input
                                    type="text"
                                    name="restaurantAddress.streetNumber"

value={restaurantData.restaurantAddress.streetNumber}
                                    onChange={handleChange}
                                    className="form-input mt-1 block w-
full rounded-md shadow-sm border h-10"
                                />
                            </div>
                            <div className="col-span-1">
```

```
                                <label className="block text-base font-
medium text-gray-700">
                                    Street Name
                                </label>
                                <input
                                    type="text"
                                    name="restaurantAddress.streetName"

value={restaurantData.restaurantAddress.streetName}
                                    onChange={handleChange}
                                    className="form-input mt-1 block w-
full rounded-md shadow-sm border h-10"
                                />
                            </div>
                        </div>
                        <div className="grid grid-cols-2 gap-4">
                            <div className="col-span-1">
                                <label className="block text-base font-
medium text-gray-700">
                                    City
                                </label>
                                <select
                                    id="city"
                                    name="restaurantAddress.city"
```

```
value={restaurantData.restaurantAddress.city}
                                    onChange={handleChange}
                                    className="form-select mt-1 block w-
full rounded-md shadow-sm border h-10"
                                >
                                    <option value="">Select a
City</option>
                                    {cityOptions.map(city => (
                                        <option key={city} value={city}>
                                            {city}
                                        </option>
                                    ))}
                                </select>
                            </div>
                            <div className="col-span-1">
                                <label className="block text-base font-
medium text-gray-700">
                                    Country
                                </label>
                                <input
                                    type="text"
                                    name="restaurantAddress.country"

value={restaurantData.restaurantAddress.country}
```

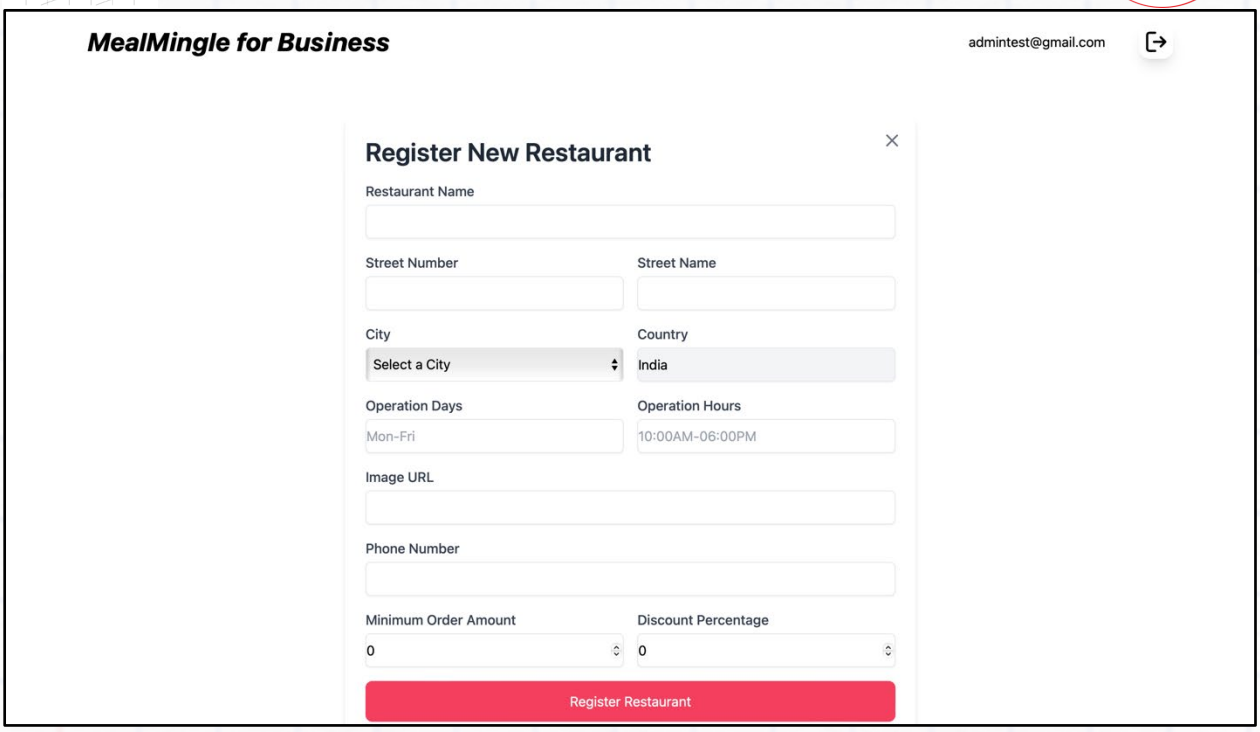
```
                                readOnly
                                className="form-input mt-1 block w-
full rounded-md shadow-sm bg-gray-100 border h-10"
                            />
                        </div>
                    </div>
                    <div className="grid grid-cols-2 gap-4">
                        <div className="col-span-1">
                            <label className="block text-base font-
medium text-gray-700">
                                Operation Days
                            </label>
                            <input
                                type="text"
                                name="restaurantOperationDays"

value={restaurantData.restaurantOperationDays}
                                onChange={handleChange}
                                placeholder="Mon-Fri"
                                className="form-input mt-1 block w-
full rounded-md shadow-sm border h-10"
                            />
                        </div>
                        <div className="col-span-1">
```

```
                                <label className="block text-base font-
medium text-gray-700">
                                    Operation Hours
                                </label>
                                <input
                                    type="text"
                                    name="restaurantOperationHours"

value={restaurantData.restaurantOperationHours}
                                    onChange={handleChange}
                                    placeholder="10:00AM-06:00PM"
                                    className="form-input mt-1 block w-
full rounded-md shadow-sm border h-10"
                                />
                            </div>
                        </div>
                        <div className="border-gray-300">
                            <label className="block text-base font-medium
text-gray-700">
                                Image URL
                            </label>
                            <input
                                type="text"
                                name="restaurantImageUrl"
                                value={restaurantData.restaurantImageUrl}
```

```
                                onChange={handleChange}
                                className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                            />
                        </div>
                        <div className="border-gray-300">
                            <label className="block text-base font-medium
text-gray-700">
                                Phone Number
                            </label>
                            <input
                                type="text"
                                name="restaurantPhoneNumber"

value={restaurantData.restaurantPhoneNumber}
                                onChange={handleChange}
                                className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                            />
                        </div>
                        <div className="grid grid-cols-2 gap-4">
                            <div className="col-span-1">
                                <label className="block text-base font-
medium text-gray-700">
                                    Minimum Order Amount
```

```
                                </label>
                                <input
                                    type="number"
                                    name="restaurantMinimumOrderAmount"

value={restaurantData.restaurantMinimumOrderAmount}
                                    onChange={handleChange}
                                    className="form-input mt-1 block w-
full rounded-md shadow-sm border h-10"
                                />
                            </div>
                            <div className="col-span-1">
                                <label className="block text-base font-
medium text-gray-700">
                                    Discount Percentage
                                </label>
                                <input
                                    type="number"
                                    name="restaurantDiscountPercentage"

value={restaurantData.restaurantDiscountPercentage}
                                    onChange={handleChange}
                                    className="form-input mt-1 block w-
full rounded-md shadow-sm border h-10"
                                />
```

```
                            </div>
                        </div>
                        <button
                            type="submit"
                            className="bg-rose-500 w-full h-12 text-white
py-2 px-4 rounded-lg"
                        >
                            Register Restaurant
                        </button>
                    </form>
                </div>
            </div>
        </>
    );
};

export default RegisterRestaurant;
```



6.24. View Admin Restaurants Component

- This code is a React component called ViewAdminRestaurants that allows an admin user to view, update, delete, and manage restaurants they have registered.
- It also integrates bank details for further operations like adding restaurant menus.

Create a new file inside the components folder and name it `ViewAdminRestaurants`.tsx.

1. Imports

- **React** is imported to use its features like useState and useEffect.
- **AdminNavbar** is a component likely for navigation within the admin panel.
- **useNavigate** from react-router-dom helps in navigation within the app.
- **toast** is used for showing pop-up notifications (like success or error messages).
- **'react-toastify/dist/ReactToastify.css'** is the CSS file for styling the toast notifications.

```
import React , { useEffect, useState }from 'react';

import AdminNavbar from './AdminNavbar';

import { useNavigate } from 'react-router-dom';
```

```
import { toast } from 'react-toastify';

import 'react-toastify/dist/ReactToastify.css';
```

2. **Defining Data Structures**

These interfaces define the structure of different objects used in this component, like `Address`, `RestaurantData`, `RestaurantItem`, `User`, and `BankDetails`. These are TypeScript interfaces that help to ensure type safety.

```
interface Address {

    pincode: string;

    streetName: string;

    city: string;

}


interface RestaurantItem {

    restaurantItemId: string;

    restaurantItemName: string;

    restaurantItemPrice: number;

    restaurantItemCategory: string;

    restaurantItemImageUrl: string;

    restaurantItemCuisineType: string;

    restaurantItemVeg: boolean;

}


interface RestaurantData {

    restaurantId: string;

    restaurantName: string;
```

```
    restaurantAddress: Address;
    restaurantRating: number;
    restaurantMinimumOrderAmount: number;
    restaurantDiscountPercentage: number;
    restaurantImageUrl: string;
    restaurantOperationDays: string;
    restaurantOperationHours: string;
    restaurantPhoneNumber: string;
}

interface RestaurantProp {
    restaurants: RestaurantData[];
    onDelete: (id: string) => void;
}

interface User {
    userName: string;
    userId: string;
    userEmail: string;
    userPhone: string;
}
interface BankDetails {
    user:User
    accountNumber: string;
    bankName: string;
```

```
    branchName: string;
    ifscCode: string;
    panNumber: string;
    aadhaarNumber: string;
    gstNumber: string;
}
```

3. **Component Setup and State Management**
   - ViewAdminRestaurants is a functional component that receives onDelete as a prop.
   - initialBankDetails sets up a default structure for bank details.
   - useState hooks are used to manage the state of bankDetails and restaurants.

```
const ViewAdminRestaurants: React.FC<RestaurantProp> = ({ onDelete }) => {
    const initialBankDetails: BankDetails = {
        user: {
            userEmail:'',
            userId:'',
            userName:'',
            userPhone:''
        },
        accountNumber: '',
        bankName: '',
        branchName: '',
        ifscCode: '',
        panNumber: '',
        aadhaarNumber: '',
        gstNumber: '',
```

```
    };


const [bankDetails,setBankDetails] =

useState<BankDetails>(initialBankDetails);

const [restaurants, setRestaurants] = useState<RestaurantData[]>([]);
```

4. **Fetching Bank Details**
   - This useEffect runs once when the component mounts.
   - It fetches bank details from a backend API and updates the bankDetails state if successful.

```
useEffect(() => {

        async function fetchBankDetails() {



            const response = await

fetch('http://localhost:8090/api/users/details',{

                method: 'GET',

                headers: {

                    'Content-Type': 'application/json',

                    'Authorization': `Bearer

${localStorage.getItem('token')}`

                }

            });

            const data = await response.json();


            if (data.error == "" && data.data != null) {

                console.log(data.data);

                setBankDetails(data.data);
```

```
            }


        }

        fetchBankDetails();

    },[])
```

5. **Fetching Restaurant Data**
   - This function fetches the list of restaurants from another backend API.
   - If successful, it updates the restaurants state with the data received.
   - Toast notifications inform the user whether the fetch was successful or if there was an error.

```
const fetchRestaurants = async () => {

        try {

            const response = await

fetch('http://localhost:8091/api/restaurants', {

                method: 'GET',

                headers: {

                    'Content-Type': 'application/json',

                    'Authorization': `Bearer

${localStorage.getItem('token')}`

                }

            });

            const data = await response.json();

            if (data.error == "" && data.data != null) {

                toast.success('Restaurants Fetched Successfully!');

                setRestaurants(data.data.restaurants);

            }
```

```
            else {
                toast.error(data.error);
            }
        } catch (error) {
            console.error('Error:', error);
        }
    }

    useEffect(() => {
        fetchRestaurants();
    },[])
```

6. **Handling Different User Actions**

- **Updating a Restaurant**

This function navigates the user to a page where they can update the details of a restaurant by passing the restaurant's ID.

```
const navigate = useNavigate();


    const handleUpdate = (id: string) => {
        navigate(`/update-restaurant/${id}`);
    };
```

- **Deleting a Restaurant**

  This function triggers the `onDelete` function passed as a prop and shows a success message using a toast notification.

```
const handleDelete = (id: string) => {
        onDelete(id);
        toast.success('Restaurant Deleted Successfully!');
    };
```

- **Viewing Restaurant Details**

  This function either navigates the user to enter their bank details (if not already entered) or directly to the restaurant's details page.

```
const handleClick = (restaurantId: string) => {


        if (bankDetails.aadhaarNumber === '') {
            navigate(`/admin/enter-bank-details/${restaurantId}`, {
                state: { nextPage: `/view-admin-restaurant-
items/${restaurantId}` }
            });
        } else {
            navigate(`/view-admin-restaurant-items/${restaurantId}`);
        }
    };
```

- **Adding a Menu to a Restaurant**

This function allows the admin to add menu items to a restaurant, ensuring bank details are entered first if not already.

```
const handleAddMenu = (restaurantId: string) => {
        if (bankDetails.aadhaarNumber === '') {
            navigate(`/admin/enter-bank-details/${restaurantId}`, {
```

```
                state: { nextPage: `/register-restaurant-
item/${restaurantId}`}
            });
        } else {
            navigate(`/register-restaurant-item/${restaurantId}`, {
                state: {
                    restaurantId: restaurantId
                }
            });
        }
    }
```

7. Return JSX

- The component first renders the AdminNavbar for navigation.
- It displays either a message if no restaurants are registered or a grid of restaurant cards, each showing details like image, name, rating, and discount.
- Each card has buttons for actions like "Add Menu," "Update," and "Delete," and clicking the image leads to more details.

```
return (
        <>
            <AdminNavbar />
            <div className='p-4 pl-20'>
                <h1 className='font-semibold text-3xl mb-4'>Explore All Your
Registered Restaurants</h1>
                {restaurants.length === 0 ? (
                    <div className="text-center text-xl">
```

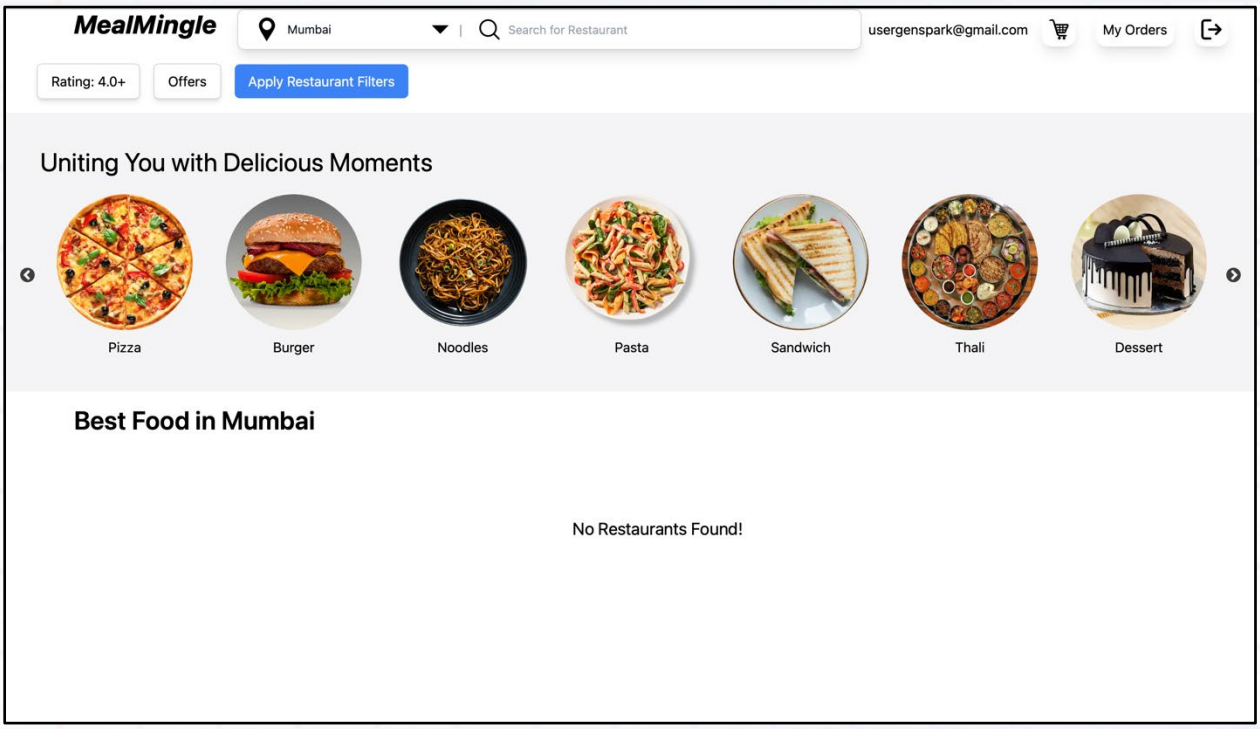
```
                        <p className="mt-60 mb-4">No Restaurants Registered
Yet!</p>
                        <p>Please <a href="/register-restaurant"
className="text-blue-500 underline">Register a Restaurant</a> to View.</p>
                    </div>
                ) : (
                    <div className='grid grid-cols-3 gap-4'>
                        {restaurants.map((data) => (
                            <div className="relative max-w-xs rounded-xl
overflow-hidden shadow-sm mt-12 cursor-pointer">
                                <img className={`w-full rounded-2xl h-60`}
src={data.restaurantImageUrl} alt="Restaurant Image" onClick={() =>
handleClick(data.restaurantId)} />
                                {data.restaurantDiscountPercentage > 0 && (
                                    <div className="absolute top-2 left-2 bg-
blue-500 text-white font-semibold py-1 px-2 rounded-md">

{`${data.restaurantDiscountPercentage}% OFF ABOVE
${data.restaurantMinimumOrderAmount}`}
                                    </div>
                                )}
                                <div className="py-4">
                                    <div className='flex justify-between
items-center'>
```

```
                                    <div className="font-semibold text-xl
mb-2">
                                        {data.restaurantName}
                                        <div className="text-sm text-
gray-600">{`${data.restaurantAddress.pincode },
${data.restaurantAddress.streetName}, ${data.restaurantAddress.city}`}</div>
                                    </div>
                                    <div className={`text-white font-
semibold text-base rounded-md p-1 ${data.restaurantRating < 4.5 ? `bg-green-
600` : `bg-green-900`}`}>
                                        {data.restaurantRating}
                                    </div>
                                </div>
                                <div className="flex justify-between mt-
4">
                                    <button
                                        onClick={() =>
handleAddMenu(data.restaurantId)}
                                        className="inline-block px-2 py-2
bg-green-500 text-white rounded hover:bg-green-600 transition duration-300"
                                    >
                                        Add Menu
                                    </button>
                                    <button
```

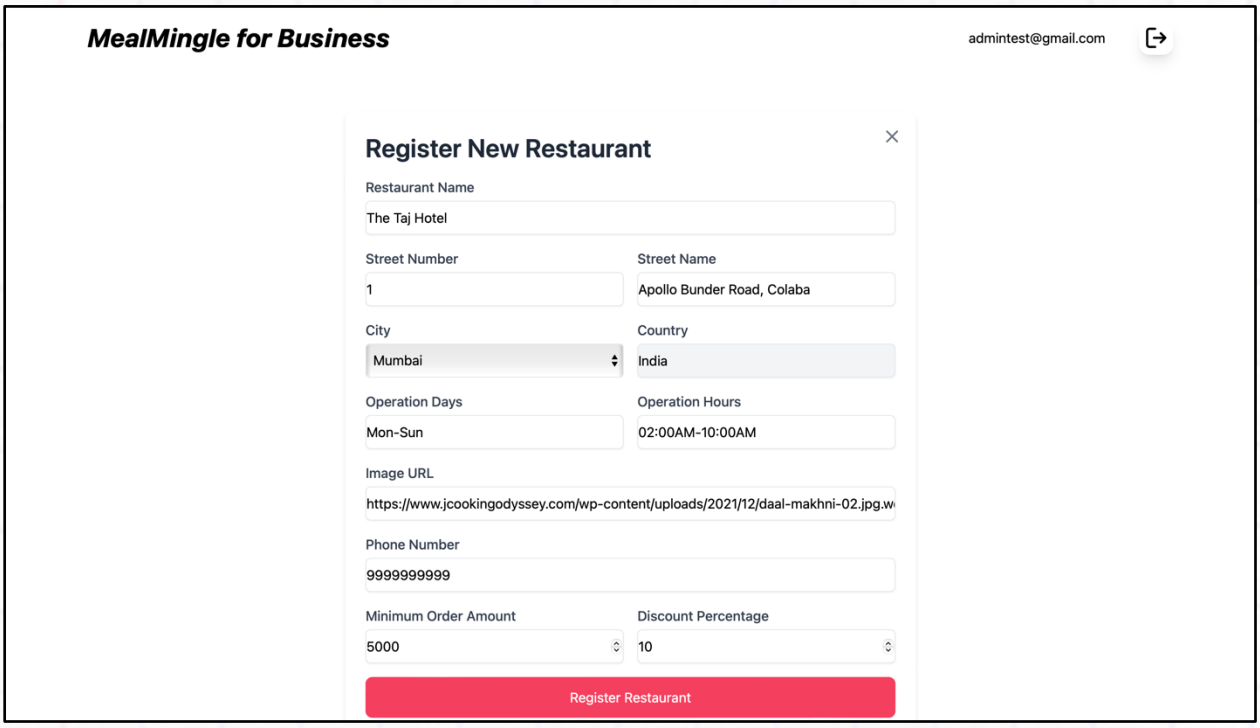
```
                                        onClick={() =>
handleUpdate(data.restaurantId)}
                                        className="inline-block px-2 py-2
bg-blue-500 text-white rounded hover:bg-blue-600 transition duration-300"
                                    >
                                        Update
                                    </button>
                                    <button
                                        onClick={() =>
handleDelete(data.restaurantId)}
                                        className="inline-block px-2 py-2
bg-red-500 text-white rounded hover:bg-red-600 transition duration-300"
                                    >
                                        Delete
                                    </button>
                                </div>
                            </div>
                        </div>
                    ))}
                </div>)}
            </div>
        </>
    );
};
```

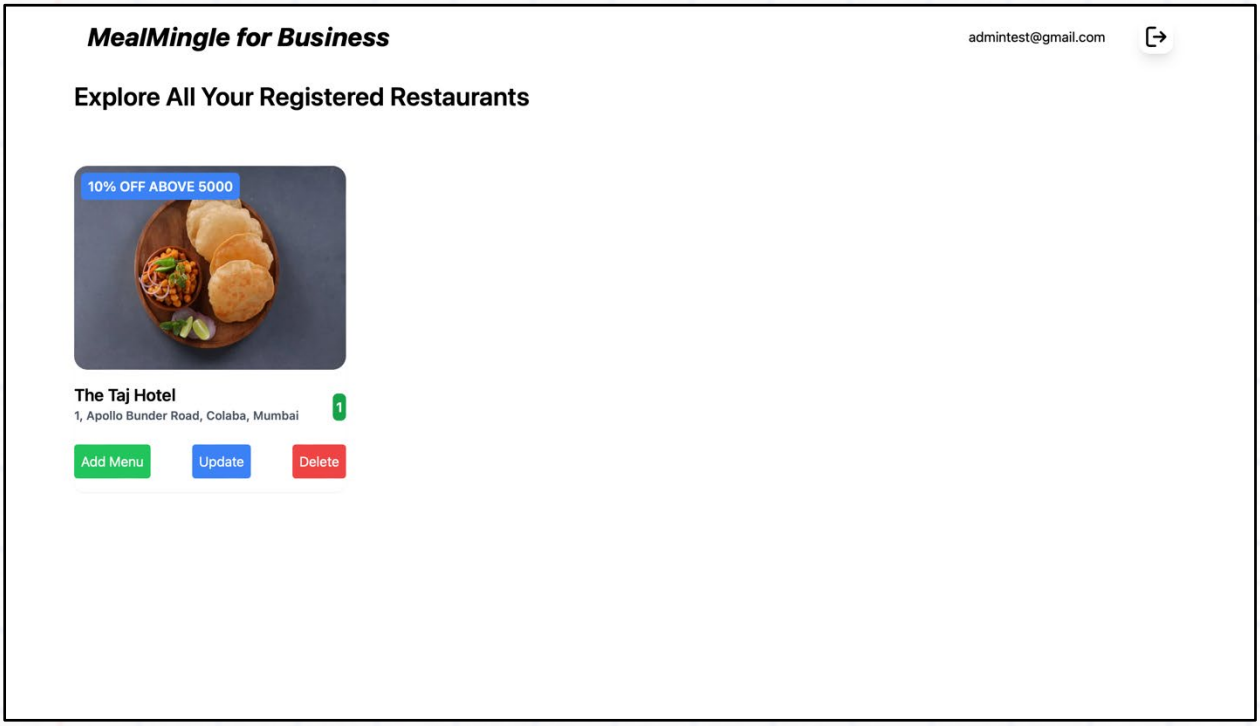
```
export default ViewAdminRestaurants;
```

In the User Dashboard, we had seen that there were no restaurants in Mumbai.



Now, let's add a restaurant in Mumbai.





6.25. Update Restaurant Component

- The UpdateRestaurant component is a React functional component used to handle updating the details of a restaurant.
- It utilizes React hooks and interacts with a REST API to perform update operations.

Create a new file inside the components folder and name it UpdateRestaurant.tsx.

1. **Imports and Interfaces**
   - **Imports**: useState, useEffect, ChangeEvent, and FormEvent are React hooks and types used for state management and handling events. useParams and useNavigate are from react-router-dom for routing. ToastContainer and toastare used for showing notifications.
   - **Interfaces**: Define the shape of data for address, restaurant items, and restaurant information.

```
import React, { useState, useEffect, ChangeEvent, FormEvent } from 'react';

import { useParams, useNavigate } from 'react-router-dom';

import AdminNavbar from './AdminNavbar';

import { ToastContainer, toast } from 'react-toastify';
```

```
import 'react-toastify/dist/ReactToastify.css';


interface Address {
    streetNumber: string;
    streetName: string;
    city: string;
    country: string;
}


interface RestaurantItem {
    restaurantItemId: string;
    restaurantItemName: string;
    restaurantItemPrice: number;
    restaurantItemCategory: string;
    restaurantItemImageUrl: string;
    restaurantItemCuisineType: string;
    restaurantItemVeg: boolean;
}


interface RestaurantData {
    restaurantId: string | undefined;
    restaurantName: string;
    restaurantAddress: Address;
    restaurantRating: number;
    restaurantMinimumOrderAmount: number;
```

```
    restaurantDiscountPercentage: number;

    restaurantImageUrl: string;

    restaurantOperationDays: string;

    restaurantOperationHours: string;

    restaurantPhoneNumber: string;

    restaurantItems: RestaurantItem[];

}
```

2. **Component Definition**
   - **Initialization**: useParams extracts the restaurant ID from the URL. useNavigate is used for programmatic navigation.
   - **State**: restaurantData holds the form data, initialized with default values. errors keeps track of validation errors.

```
const UpdateRestaurant: React.FC = () => {

    const { id } = useParams();

    const navigate = useNavigate();



    const initialRestaurantData: RestaurantData = {

        restaurantId: id,

        restaurantName: '',

        restaurantAddress: {

            streetNumber: '',

            streetName: '',

            city: '',

            country: 'India',

        },

        restaurantRating: 1,
```

```
        restaurantMinimumOrderAmount: 0,
        restaurantDiscountPercentage: 0,
        restaurantImageUrl: '',
        restaurantOperationDays: '',
        restaurantOperationHours: '',
        restaurantPhoneNumber: '',
        restaurantItems: [],
    };

    const [restaurantData, setRestaurantData] =
useState<RestaurantData>(initialRestaurantData);
    const [errors, setErrors] = useState<string[]>([]);
```

3. **Handle Change**

- **Handling Input Changes**: Updates `restaurantData` state based on input field changes. Handles nested fields (e.g., `restaurantAddress.streetNumber`) and numeric fields with specific validation.

```
const handleChange = (e: ChangeEvent<HTMLInputElement |
HTMLTextAreaElement | HTMLSelectElement>) => {
        const { name, value } = e.target;
        if (name.includes('.')) {
            const [parent, child] = name.split('.');
            setRestaurantData(prevData => ({
                ...prevData,
                [parent as keyof RestaurantData]: {
```

```
                    ...(prevData[parent as keyof RestaurantData] as
Address),
                    [child]: value,
                },
            }));
        } else {
            if (name === 'restaurantMinimumOrderAmount' || name ===
'restaurantDiscountPercentage') {
                const numericValue = parseFloat(value);
                if (numericValue >= 0) {
                    setRestaurantData({ ...restaurantData, [name]:
numericValue });
                }
            } else {
                setRestaurantData({ ...restaurantData, [name]: value });
            }
        }
    };
```

4. **Update Restaurant**

API Request: Sends a PUT request to update the restaurant data. If successful, shows a success message and navigates to the view restaurants page. If there's an error, it displays an error message.

```
const updateRestaurant = async () => {
        const modifiedRestaurantData = {
            restaurant: {
```

```
                restaurantId: id,
                restaurantName: restaurantData.restaurantName,
                restaurantAddress: {
                    pincode: restaurantData.restaurantAddress.streetNumber,
                    streetName: restaurantData.restaurantAddress.streetName,
                    city: restaurantData.restaurantAddress.city,
                    country: restaurantData.restaurantAddress.country,
                },
                restaurantRating: restaurantData.restaurantRating,
                restaurantMinimumOrderAmount:
restaurantData.restaurantMinimumOrderAmount,
                restaurantDiscountPercentage:
restaurantData.restaurantDiscountPercentage,
                restaurantImageUrl: restaurantData.restaurantImageUrl,
                restaurantOperationDays:
restaurantData.restaurantOperationDays,
                restaurantOperationHours:
restaurantData.restaurantOperationHours,
                restaurantPhoneNumber: restaurantData.restaurantPhoneNumber,
                restaurantItems: restaurantData.restaurantItems,
                restaurantAvailability: "Open",
            }
        }
```

```
        const response = await
fetch('http://localhost:8091/api/restaurants/update', {
            method: 'PUT',
            headers: {
                'Content-Type': 'application/json',
                'Authorization': `Bearer ${localStorage.getItem('token')}`,
            },
            body: JSON.stringify(modifiedRestaurantData),
        });

        const data = await response.json();
        if (data.error === "") {
            toast.success('Restaurant Updated Successfully!');
            console.log(data);
            setTimeout(() => {
                navigate('/view-admin-restaurants');
            }, 2000);
        }
        else {
            toast.error(data.error);
        }
    }
```

5. **Handle Submit**

Form Submission: Prevents default form submission, validates the form, and calls `updateRestaurant` if the form is valid.

```
const handleSubmit = (e: FormEvent<HTMLFormElement>) => {

        e.preventDefault();



        const isValid = validateForm();

        if (!isValid) {

            return;

        }



        updateRestaurant();

    };
```

6. **Handle Close**

Close Button: Navigates to the view restaurants page when the close button is clicked.

```
const handleClose = () => {

        navigate('/view-admin-restaurants');

    };
```

7. **Form Validation**

Validation: Checks if the form data meets the required criteria. Shows error messages if validation fails.

```
const validateForm = () => {

        let isValid = true;

        const errors: string[] = [];



        // Restaurant Name
```

```
        if (!restaurantData.restaurantName.trim()) {
            errors.push('Restaurant Name is Required.');
            isValid = false;
        } else if (/^\d+$/.test(restaurantData.restaurantName.trim())) {
            errors.push('Restaurant Name must be a String, not a Number.');
            isValid = false;
        } else if (restaurantData.restaurantName.trim().length < 5 ||
restaurantData.restaurantName.trim().length > 20) {
            errors.push('Restaurant Name must be between 5 to 20 Characters
long.');
            isValid = false;
        }

        // Street Number
        if (!restaurantData.restaurantAddress.streetNumber.trim()) {
            errors.push('Street Number is Required.');
            isValid = false;
        } else if
(isNaN(Number(restaurantData.restaurantAddress.streetNumber))) {
            errors.push('Street Number must be a Number, not a String.');
            isValid = false;
        }

        // Street Name
        if (!restaurantData.restaurantAddress.streetName.trim()) {
```

```
            errors.push('Street Name is Required.');
            isValid = false;
        } else if
(/^\d+$/.test(restaurantData.restaurantAddress.streetName.trim())) {
            errors.push('Street Name must be a String, not a Number.');
            isValid = false;
        } else if (restaurantData.restaurantAddress.streetName.trim().length
< 5 || restaurantData.restaurantAddress.streetName.trim().length > 30) {
            errors.push('Street Name must be between 5 to 30 Characters
long.');
            isValid = false;
        }

        // City
        if (!restaurantData.restaurantAddress.city.trim()) {
            errors.push('City is Required.');
            isValid = false;
        } else if
(/^\d+$/.test(restaurantData.restaurantAddress.city.trim())) {
            errors.push('City must be a String, not a Number.');
            isValid = false;
        } else if (restaurantData.restaurantAddress.city.trim().length < 3 ||
restaurantData.restaurantAddress.city.trim().length > 30) {
            errors.push('City must be between 3 to 30 Characters long.');
            isValid = false;
```

```
        }

        // Operation Days
        const operationDays = restaurantData.restaurantOperationDays.trim();

        if (!operationDays) {
            errors.push('Operation Days are Required.');
            isValid = false;
        } else {
            const operationDaysRegex = /^[a-zA-Z]{3}-[a-zA-Z]{3}$/;
            if (!operationDaysRegex.test(operationDays)) {
                errors.push('Operation Days must be of the format: Mon-
Fri.');
                isValid = false;
            } else {
                const [startDay, endDay] = operationDays.split('-');
                const daysOfWeek = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat",
"Sun"];

                if (!daysOfWeek.includes(startDay) ||
!daysOfWeek.includes(endDay)) {
                    errors.push('Invalid Days Provided. Days must be one of
Mon, Tue, Wed, Thu, Fri, Sat, Sun.');
                    isValid = false;
                } else if (startDay === endDay) {
```

```
                    errors.push('Start Day and End Day must not be the
Same.');
                    isValid = false;
                }
            }
        }

        // Operation Hours
        const operationHours =
restaurantData.restaurantOperationHours.trim();

        if (!operationHours) {
            errors.push('Operation Hours are Required.');
            isValid = false;
        } else {
            const operationHoursRegex = /^(0?[1-9]|1[0-2]):[0-5][0-9]([AP]M)-
(0?[1-9]|1[0-2]):[0-5][0-9]([AP]M)$/;

            if (!operationHoursRegex.test(operationHours)) {
                errors.push('Operation Hours must be in the format: 10:00AM-
06:00PM.');
                isValid = false;
            } else {
                const [start, end] = operationHours.split('-');
```

```
            const convertTo24Hour = (time: any) => {
                let [hours, minutes] = time.slice(0, -2).split(':');
                const period = time.slice(-2);
                hours = parseInt(hours, 10);
                minutes = parseInt(minutes, 10);
                if (period === 'PM' && hours !== 12) {
                    hours += 12;
                } else if (period === 'AM' && hours === 12) {
                    hours = 0;
                }
                return hours * 60 + minutes;
            };

            const startTime = convertTo24Hour(start);
            const endTime = convertTo24Hour(end);

            if (startTime === endTime) {
                errors.push('Start Time and End Time must not be the
Same.');
                isValid = false;
            } else if (startTime >= endTime) {
                errors.push('End Time must be after Start Time.');
                isValid = false;
            }
        }
```

```
        }

        // Image URL
        if (!restaurantData.restaurantImageUrl.trim()) {
            errors.push('Image URL is Required.');
            isValid = false;
        }

        // Phone Number
        if (!restaurantData.restaurantPhoneNumber.trim()) {
            errors.push('Phone Number is Required.');
            isValid = false;
        } else if
(!/^\d+$/.test(restaurantData.restaurantPhoneNumber.trim())) {
            errors.push('Phone Number must contain only Numeric
Characters.');
            isValid = false;
        } else if (restaurantData.restaurantPhoneNumber.trim().length !== 10)
{
            errors.push('Phone Number must be exactly 10 Digits.');
            isValid = false;
        } else if (!/^[6-
9]\d{9}$/.test(restaurantData.restaurantPhoneNumber.trim())) {
            errors.push('Phone Number must start with 6, 7, 8, or 9');
            isValid = false;
```

```
        }


        setErrors(errors);
        if (errors.length > 0) {
            toast.error(errors[0]);
        }


        return isValid;
    };
```

8. `cityOptions` Array

This array defines a list of cities that will be available for selection in the dropdown menu for the city field in the form.

```
const cityOptions = [
        'Delhi',
        'Mumbai',
        'Chennai',
        'Bangalore',
        'Jaipur'
    ];
```

9. Return JSX

- Renders the form with fields for updating restaurant details.
- Includes navigation buttons and validation messages.
- Utilizes Tailwind CSS for styling.

```
return (
        <>
            <AdminNavbar />
            <div className="flex justify-center items-center min-h-screen">
                <div className="w-full max-w-2xl p-6 bg-white rounded-lg
shadow-md relative">
                    <div className="absolute top-0 right-0 m-4">
                        <button
                            onClick={handleClose}
                            className="text-gray-500"
                        >
                            <svg
                                xmlns="http://www.w3.org/2000/svg"
                                className="h-6 w-6"
                                fill="none"
                                viewBox="0 0 24 24"
                                stroke="currentColor"
                            >
                                <path
                                    strokeLinecap="round"
                                    strokeLinejoin="round"
                                    strokeWidth="2"
                                    d="M6 18L18 6M6 6l12 12"
                                />
                            </svg>
```

```
                        </button>
                    </div>
                    <h3 className="text-3xl font-semibold text-gray-800 mb-
4">
                        Update Restaurant
                    </h3>
                    <form onSubmit={handleSubmit} className="space-y-4">
                        <div className="grid grid-cols-2 gap-4">
                            <div className="col-span-2">
                                <label className="block text-base font-medium
text-gray-700">
                                    Restaurant Name
                                </label>
                                <input
                                    type="text"
                                    id="restaurantName"
                                    name="restaurantName"
                                    value={restaurantData.restaurantName}
                                    onChange={handleChange}
                                    className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                                />
                            </div>
                        </div>
                        <div className="grid grid-cols-2 gap-4">
```

```
                        <div className="col-span-1">
                            <label className="block text-base font-medium
text-gray-700">
                                Street Number
                            </label>
                            <input
                                type="text"
                                id="streetNumber"
                                name="restaurantAddress.streetNumber"

value={restaurantData.restaurantAddress.streetNumber}
                                onChange={handleChange}
                                className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                            />
                        </div>
                        <div className="col-span-1">
                            <label className="block text-base font-medium
text-gray-700">
                                Street Name
                            </label>
                            <input
                                type="text"
                                id="streetName"
                                name="restaurantAddress.streetName"
```

```
value={restaurantData.restaurantAddress.streetName}
                                    onChange={handleChange}
                                    className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                                />
                            </div>
                        </div>
                        <div className="grid grid-cols-2 gap-4">
                            <div className="col-span-1">
                                <label className="block text-base font-medium
text-gray-700">
                                    City
                                </label>
                                <select
                                    id="city"
                                    name="restaurantAddress.city"

value={restaurantData.restaurantAddress.city}
                                    onChange={handleChange}
                                    className="form-select mt-1 block w-full
rounded-md shadow-sm border h-10"
                                >
                                    <option value="">Select a City</option>
                                    {cityOptions.map(city => (
```

```
                                    <option key={city} value={city}>
                                        {city}
                                    </option>
                                ))}
                            </select>
                        </div>
                        <div className="col-span-1">
                            <label className="block text-base font-medium
text-gray-700">
                                Country
                            </label>
                            <input
                                type="text"
                                id="country"
                                name="restaurantAddress.country"

value={restaurantData.restaurantAddress.country}
                                readOnly
                                className="form-input mt-1 block w-full
rounded-md shadow-sm bg-gray-100 border h-10"
                            />
                        </div>
                    </div>
                    <div className="grid grid-cols-2 gap-4">
                        <div className="col-span-1">
```

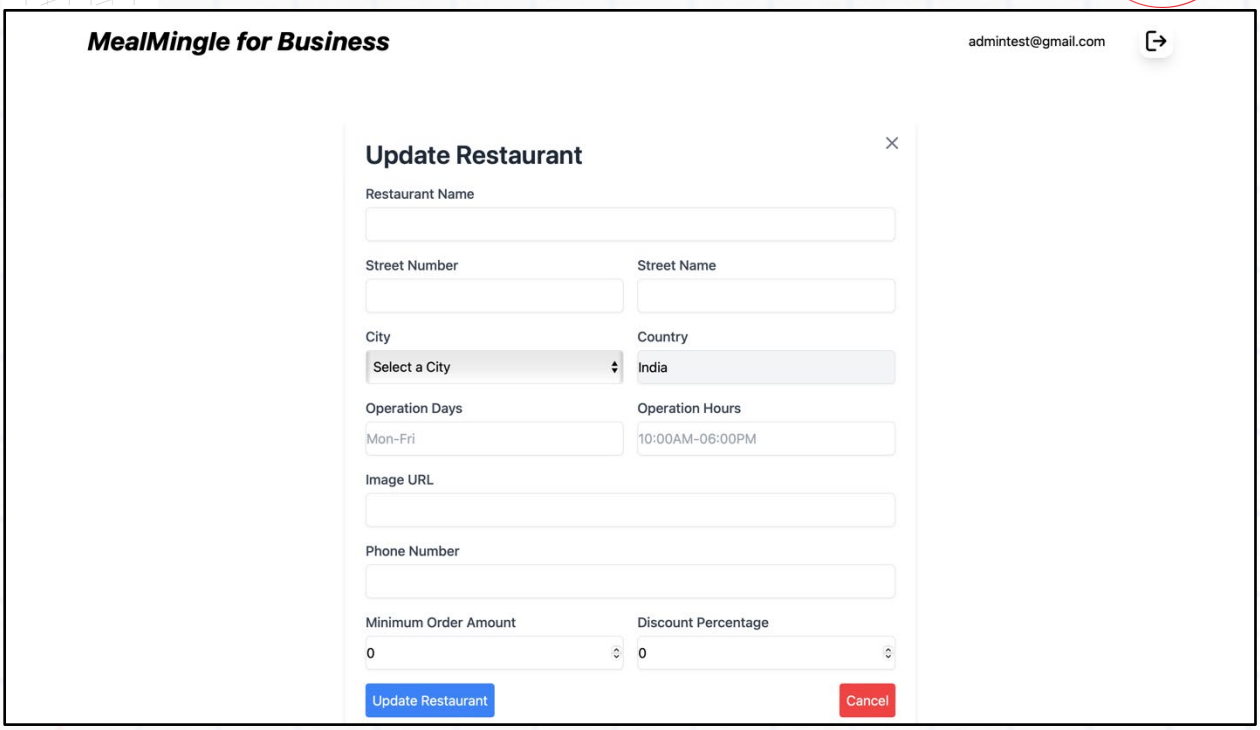
```
                                <label className="block text-base font-medium
text-gray-700">
                                    Operation Days
                                </label>
                                <input
                                    type="text"
                                    id="restaurantOperationDays"
                                    name="restaurantOperationDays"

value={restaurantData.restaurantOperationDays}
                                    onChange={handleChange}
                                    placeholder="Mon-Fri"
                                    className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                                />
                            </div>
                            <div className="col-span-1">
                                <label className="block text-base font-medium
text-gray-700">
                                    Operation Hours
                                </label>
                                <input
                                    type="text"
                                    id="restaurantOperationHours"
                                    name="restaurantOperationHours"
```

```
value={restaurantData.restaurantOperationHours}
                                    onChange={handleChange}
                                    placeholder="10:00AM-06:00PM"
                                    className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                                />
                            </div>
                        </div>
                        <div className="border-gray-300">
                            <label className="block text-base font-medium
text-gray-700">
                                Image URL
                            </label>
                            <input
                                type="text"
                                id="restaurantImageUrl"
                                name="restaurantImageUrl"
                                value={restaurantData.restaurantImageUrl}
                                onChange={handleChange}
                                className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                            />
                        </div>
                        <div className="border-gray-300">
```

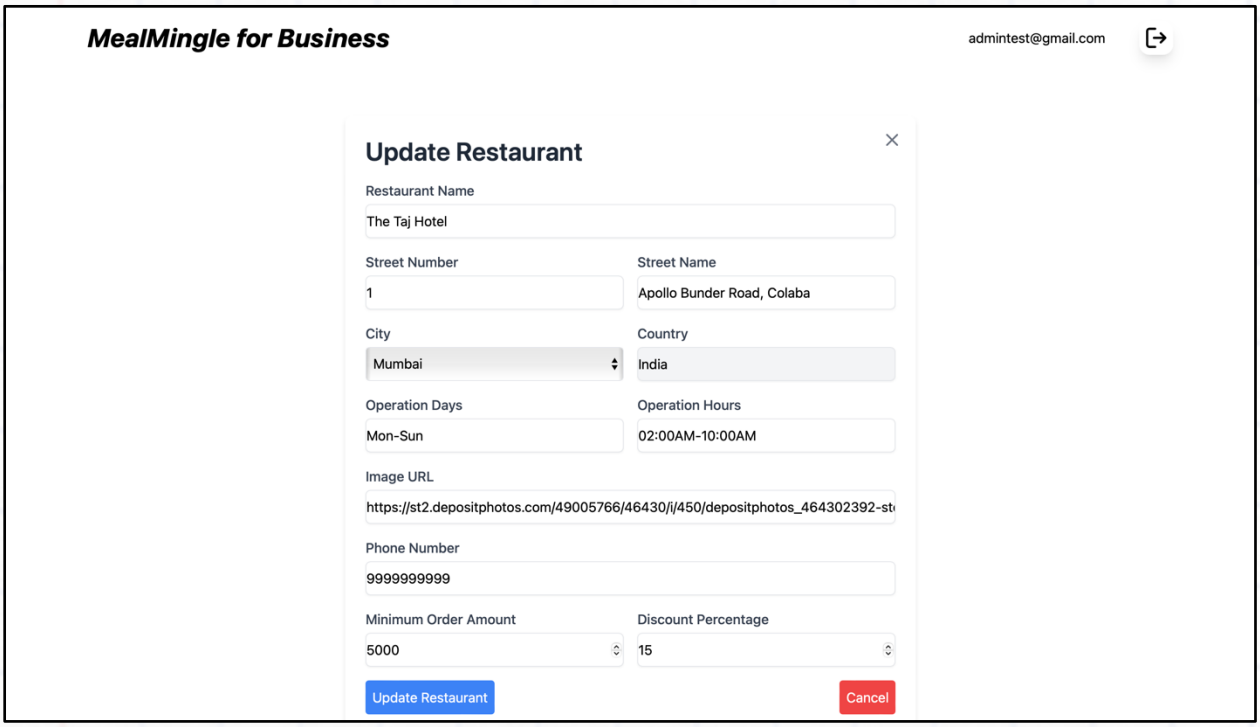
```
                        <label className="block text-base font-medium
text-gray-700">
                            Phone Number
                        </label>
                        <input
                            type="text"
                            id="restaurantPhoneNumber"
                            name="restaurantPhoneNumber"
                            value={restaurantData.restaurantPhoneNumber}
                            onChange={handleChange}
                            className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                        />
                    </div>
                    <div className="grid grid-cols-2 gap-4">
                        <div className="col-span-1">
                            <label className="block text-base font-medium
text-gray-700">
                                Minimum Order Amount
                            </label>
                            <input
                                type="number"
                                id="restaurantMinimumOrderAmount"
                                name="restaurantMinimumOrderAmount"
```

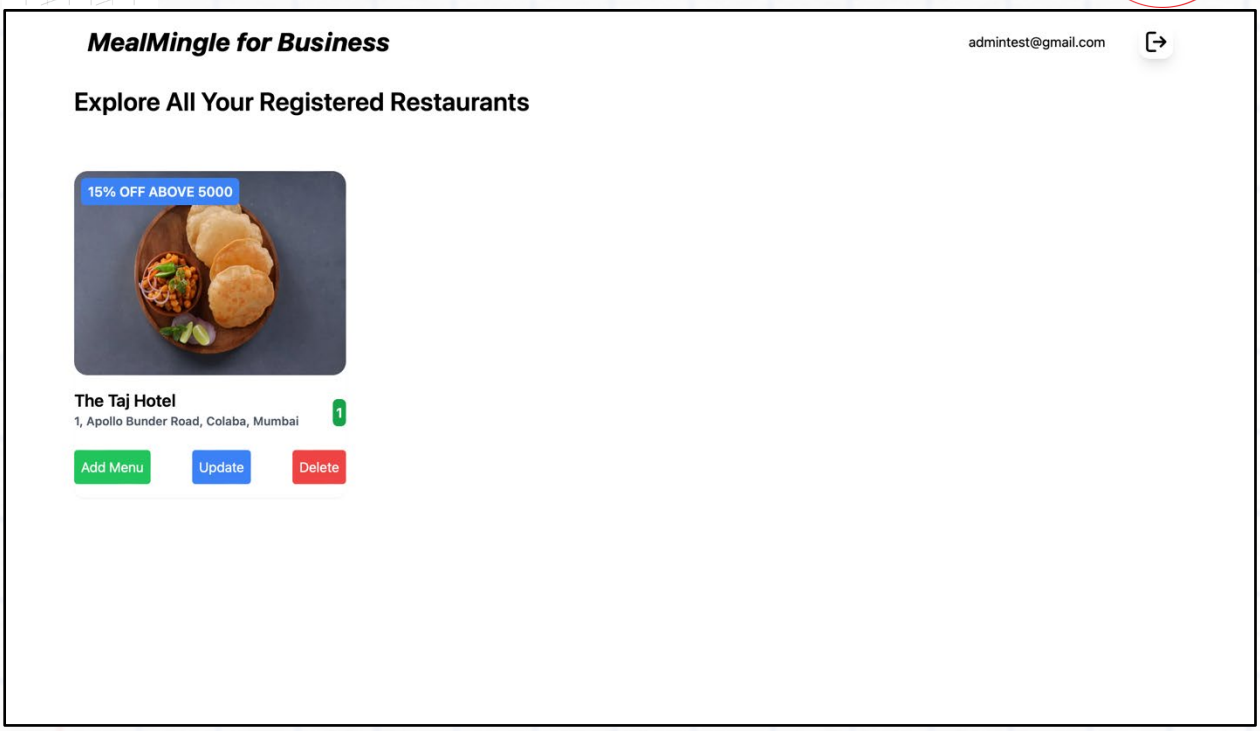
```
value={restaurantData.restaurantMinimumOrderAmount}
                                    onChange={handleChange}
                                    className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                                />
                            </div>
                            <div className="col-span-1">
                                <label className="block text-base font-medium
text-gray-700">
                                    Discount Percentage
                                </label>
                                <input
                                    type="number"
                                    id="restaurantDiscountPercentage"
                                    name="restaurantDiscountPercentage"

value={restaurantData.restaurantDiscountPercentage}
                                    onChange={handleChange}
                                    className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                                />
                            </div>
                        </div>
                        <div className="flex justify-between">
```

```
                            <button
                                type="submit"
                                className="inline-block px-2 py-2 bg-blue-500
text-white rounded hover:bg-blue-600 transition duration-300"
                            >
                                Update Restaurant
                            </button>
                            <button
                                type="button"
                                onClick={handleClose}
                                className="inline-block px-2 py-2 bg-red-500
text-white rounded hover:bg-red-600 transition duration-300"
                            >
                                Cancel
                            </button>
                        </div>
                    </form>
                </div>
            </div >
        </>
    );
};

export default UpdateRestaurant;
```



For The Taj Hotel, let's update the restaurantDiscountPercentage and make it to 15%.





6.26. Restaurant Bank Details Component

- The RestaurantBankDetails component is a React functional component used to handle and display a form for entering bank details for a restaurant.
- It includes validation, form handling, and error messaging.

Create a new file inside the components folder and name it RestaurantBankDetails.tsx.

1. **Imports**
   - **React and Hooks**: Importing React along with useState, FormEvent, and ChangeEvent from React for managing state and handling form events.
   - **Routing**: useLocation, useNavigate, and useParams are used for navigation and retrieving route parameters.
   - **Toast Notifications**: react-toastify is used for showing notifications to users.

```
import React, { useState, FormEvent, ChangeEvent } from 'react';

import { useLocation, useNavigate, useParams } from 'react-router-dom';

import AdminNavbar from './AdminNavbar';
```

```
import { ToastContainer, toast } from 'react-toastify';

import 'react-toastify/dist/ReactToastify.css';
```

2. Interface Definition

BankDetails Interface: Defines the structure of the bank details data, which includes fields like account number, bank name, etc.

```
interface BankDetails {

    accountNumber: string;

    bankName: string;

    branchName: string;

    ifscCode: string;

    panNumber: string;

    aadhaarNumber: string;

    gstNumber: string;

}
```

3. Component Initialization

- **Navigation and Params**: useNavigate is used to programmatically navigate, and useParams retrieves the restaurantId from the route.
- **Initial State**: initialBankDetails sets up the initial empty state for the form.
- **State Variables**: bankDetails holds the current form data, and errors stores validation errors.

```
const RestaurantBankDetails: React.FC = () => {

    const navigate = useNavigate();

    const { restaurantId } = useParams<{ restaurantId: string }>();

    console.log(restaurantId);

    const location = useLocation();

    const nextPage = location.state?.nextPage || '/view-admin-restaurants';
```

```
    const initialBankDetails: BankDetails = {
        accountNumber: '',
        bankName: '',
        branchName: '',
        ifscCode: '',
        panNumber: '',
        aadhaarNumber: '',
        gstNumber: '',
    };

    const [bankDetails, setBankDetails] =
useState<BankDetails>(initialBankDetails);
    const [errors, setErrors] = useState<string[]>([]);
```

4. Handle Form Changes

handleChange: Updates the `bankDetails` state whenever an input field changes.

```
const handleChange = (e: ChangeEvent<HTMLInputElement>) => {
        const { name, value } = e.target;
        setBankDetails({ ...bankDetails, [name]: value });
    };
```

5. Add Owner Details Function

addOwnerDetails: Sends the form data to the backend server using a POST request. If successful, it shows a success message and navigates to the next page; otherwise, it displays an error message.

```
async function addOwnerDetails() {
        const userData = {
```

```
            accountNumber: bankDetails.accountNumber,
            bankName: bankDetails.bankName,
            branchName: bankDetails.branchName,
            ifscCode: bankDetails.ifscCode,
            panNumber: bankDetails.panNumber,
            adharNumber: bankDetails.aadhaarNumber,
            gstNumber: bankDetails.gstNumber
        }
        const response = await
fetch('http://localhost:8090/api/users/details', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'Authorization': `Bearer ${localStorage.getItem('token')}`
            },
            body: JSON.stringify(userData)
        });
        console.log(userData)
        const data = await response.json();
        if (data.error == "") {
            toast.success('Bank Details Added Successfully!');
            setTimeout(() => {
                navigate(nextPage);
            }, 2000);
        }
```

```
        else {
            toast.error(data.error);
        }
    }
```

6. Handle Form Submission

handleSubmit: Prevents the default form submission behavior, validates the form, and if valid, calls `addOwnerDetails` to submit the data.

```
const handleSubmit = (e: FormEvent<HTMLFormElement>) => {
        e.preventDefault();

        const isValid = validateForm();
        if (!isValid) {
            return;
        }
        addOwnerDetails();
    };
```

7. Close Button Handler

handleClose: Navigates to the `/view-admin-restaurants` page when the close button is clicked.

```
const handleClose = () => {
        navigate('/view-admin-restaurants');
    };
```

8. Validate Form

validateForm: Checks if the form fields are filled correctly and sets appropriate error messages. Shows the first error if there are any.

```
const validateForm = () => {

        let isValid = true;

        const errors: string[] = [];


        // Account Number

        if (!bankDetails.accountNumber.trim()) {

            errors.push('Account Number is Required.');

            isValid = false;

        } else if (isNaN(Number(bankDetails.accountNumber))) {

            errors.push('Account Number must be a Number, not a String.');

            isValid = false;

        } else if (bankDetails.accountNumber.trim().length !== 12) {

            errors.push('Account Number must be exactly 12 Digits.');

            isValid = false;

        }


        // Bank Name

        if (!bankDetails.bankName.trim()) {

            errors.push('Bank Name is Required.');

            isValid = false;

        }


        // Branch Name

        if (!bankDetails.branchName.trim()) {

            errors.push('Branch Name is Required.');
```

```
            isValid = false;
        }

        // IFSC Code
        if (!bankDetails.ifscCode.trim()) {
            errors.push('IFSC Code is Required.');
            isValid = false;
        } else if (bankDetails.ifscCode.trim().length !== 11) {
            errors.push('IFSC Code must be exactly 11 Characters.');
            isValid = false;
        }

        // PAN Number
        if (!bankDetails.panNumber.trim()) {
            errors.push('PAN Number is Required.');
            isValid = false;
        } else if (!/^([A-Z]){5}([0-9]){4}([A-
Z]){1}$/.test(bankDetails.panNumber.trim())) {
            errors.push('Invalid PAN Number Format. Example: ABCDE1234F');
            isValid = false;
        }

        // Aadhar Number
        if (!bankDetails.aadhaarNumber.trim()) {
            errors.push('Aadhar Number is Required.');
```

```
            isValid = false;
        } else if (isNaN(Number(bankDetails.aadhaarNumber))) {
            errors.push('Aadhar Number must contain only Numeric
Characters.');
            isValid = false;
        } else if (bankDetails.aadhaarNumber.trim().length !== 12) {
            errors.push('Aadhar Number must be exactly 12 Digits.');
            isValid = false;
        }

        setErrors(errors);
        if (errors.length > 0) {
            toast.error(errors[0]);
        }

        return true;
    };
```

9. Return JSX

- Renders the component with a navigation bar, form fields, and buttons.
- The form collects bank details, and errors or success messages are shown using toast notifications.

```
return (
        <>
            <AdminNavbar />
            <div className="flex justify-center items-center min-h-screen">
```

```
                <div className="w-full max-w-2xl p-6 bg-white rounded-lg
shadow-md relative">
                    <div className="absolute top-0 right-0 m-4">
                        <button
                            onClick={handleClose}
                            className="text-gray-500"
                        >
                            <svg
                                xmlns="http://www.w3.org/2000/svg"
                                className="h-6 w-6"
                                fill="none"
                                viewBox="0 0 24 24"
                                stroke="currentColor"
                            >
                                <path
                                    strokeLinecap="round"
                                    strokeLinejoin="round"
                                    strokeWidth="2"
                                    d="M6 18L18 6M6 6l12 12"
                                />
                            </svg>
                        </button>
                    </div>
                    <h3 className="text-3xl font-semibold text-gray-800 mb-
4">
```

```
                        Enter Bank Details
                    </h3>
                    <form onSubmit={handleSubmit} className="space-y-4">
                        <div className="grid grid-cols-2 gap-4">
                            <div className="col-span-1">
                                <label className="block text-base font-medium
text-gray-700">
                                    Account Number
                                </label>
                                <input
                                    type="text"
                                    id="accountNumber"
                                    name="accountNumber"
                                    value={bankDetails.accountNumber}
                                    onChange={handleChange}
                                    className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                                />
                            </div>
                            <div className="col-span-1">
                                <label className="block text-base font-medium
text-gray-700">
                                    Bank Name
                                </label>
                                <input
```

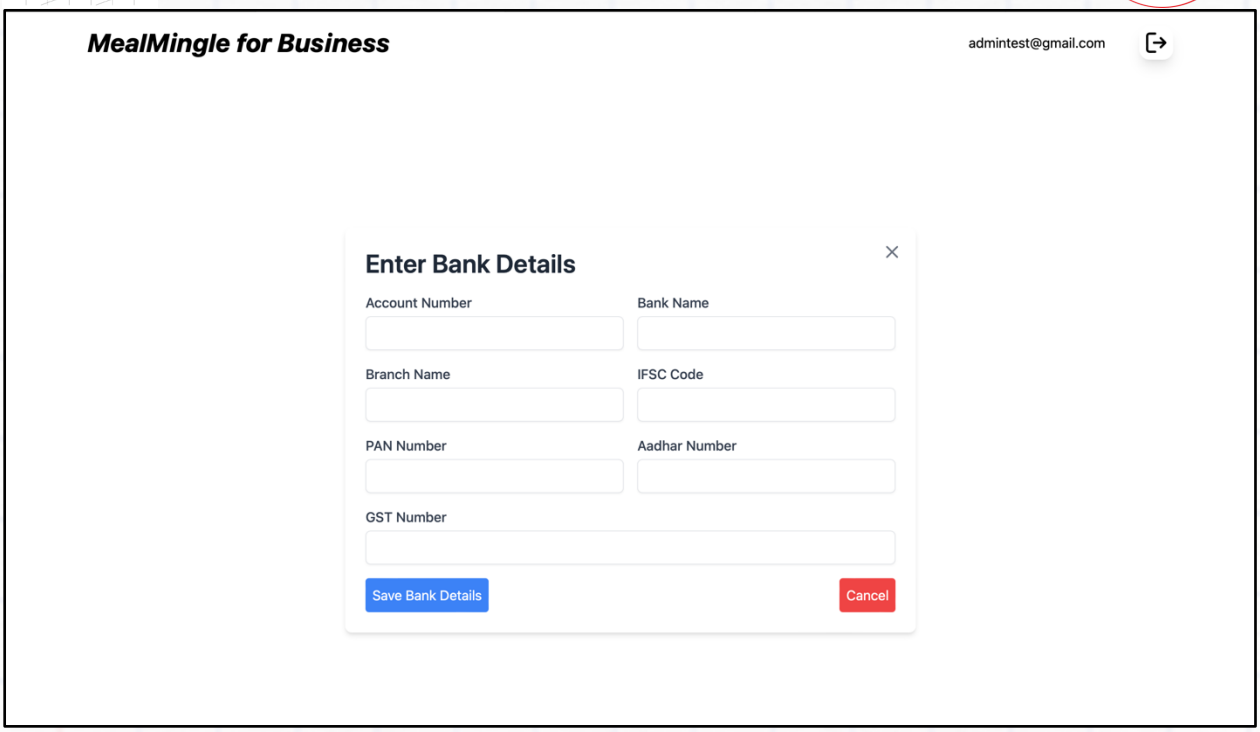
```
                                type="text"
                                id="bankName"
                                name="bankName"
                                value={bankDetails.bankName}
                                onChange={handleChange}
                                className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                            />
                        </div>
                    </div>
                    <div className="grid grid-cols-2 gap-4">
                        <div className="col-span-1">
                            <label className="block text-base font-medium
text-gray-700">
                                Branch Name
                            </label>
                            <input
                                type="text"
                                id="branchName"
                                name="branchName"
                                value={bankDetails.branchName}
                                onChange={handleChange}
                                className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                            />
```

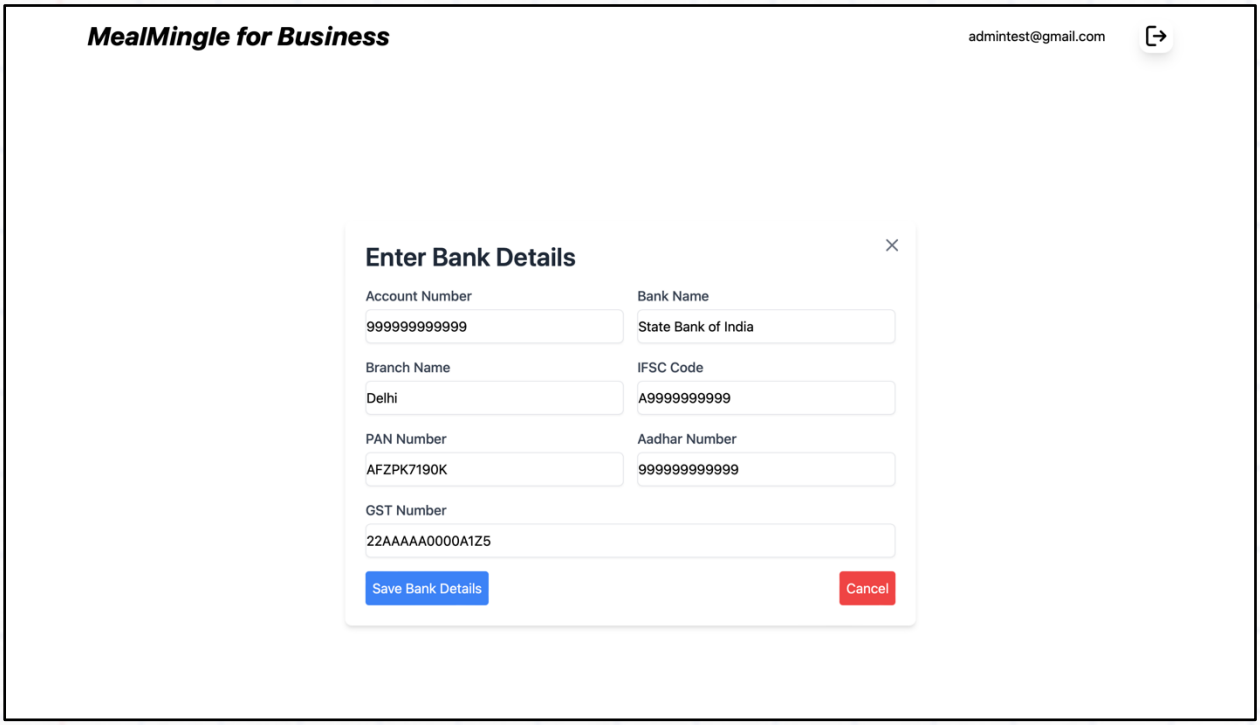
```
                        </div>
                        <div className="col-span-1">
                            <label className="block text-base font-medium
text-gray-700">
                                IFSC Code
                            </label>
                            <input
                                type="text"
                                id="ifscCode"
                                name="ifscCode"
                                value={bankDetails.ifscCode}
                                onChange={handleChange}
                                className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                            />
                        </div>
                    </div>
                    <div className="grid grid-cols-2 gap-4">
                        <div className="col-span-1">
                            <label className="block text-base font-medium
text-gray-700">
                                PAN Number
                            </label>
                            <input
                                type="text"
```

```
                                    id="panNumber"
                                    name="panNumber"
                                    value={bankDetails.panNumber}
                                    onChange={handleChange}
                                    className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                                />
                            </div>
                            <div className="col-span-1">
                                <label className="block text-base font-medium
text-gray-700">
                                    Aadhar Number
                                </label>
                                <input
                                    type="text"
                                    id="aadhaarNumber"
                                    name="aadhaarNumber"
                                    value={bankDetails.aadhaarNumber}
                                    onChange={handleChange}
                                    className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                                />
                            </div>
                        </div>
                        <div className="border-gray-300">
```

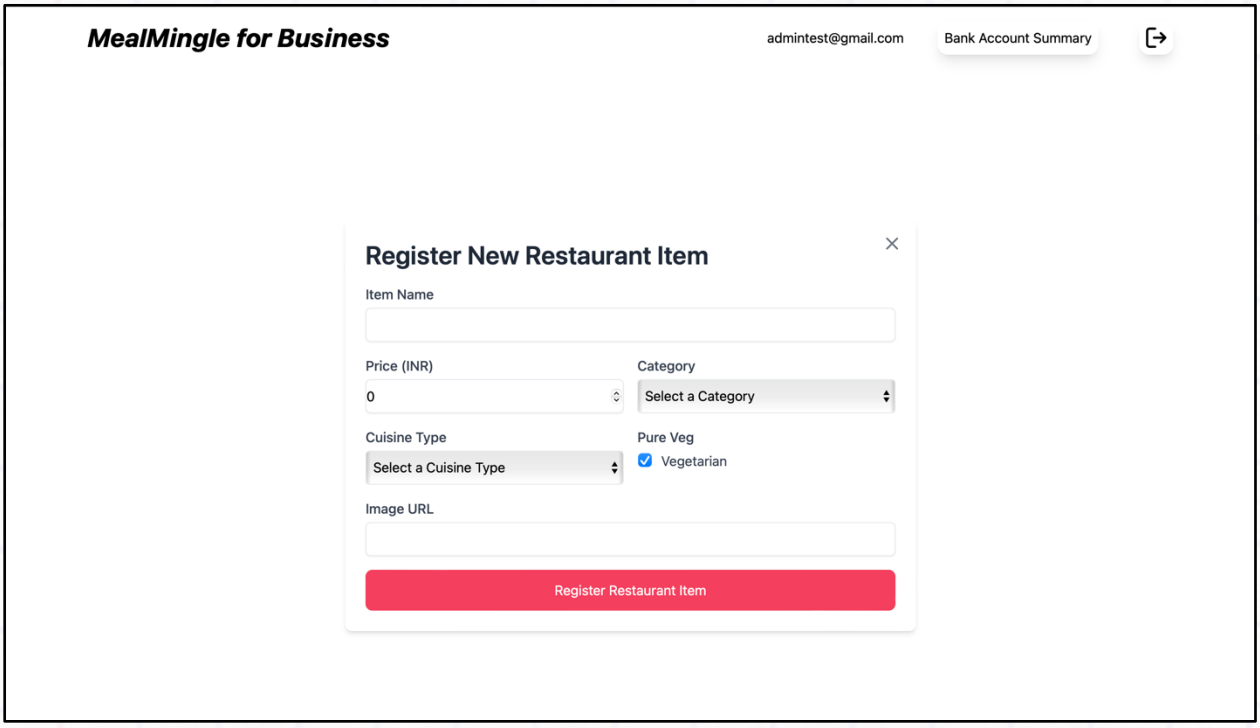
```
                            <label className="block text-base font-medium
text-gray-700">
                                GST Number
                            </label>
                            <input
                                type="text"
                                id="gstNumber"
                                name="gstNumber"
                                value={bankDetails.gstNumber}
                                onChange={handleChange}
                                className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                            />
                        </div>
                        <div className="flex justify-between">
                            <button
                                type="submit"
                                className="inline-block px-2 py-2 bg-blue-500
text-white rounded hover:bg-blue-600 transition duration-300"
                            >
                                Save Bank Details
                            </button>
                            <button
                                type="button"
```

```
                                onClick={() => navigate('/view-admin-
restaurants')}
                                className="inline-block px-2 py-2 bg-red-500
text-white rounded hover:bg-red-600 transition duration-300"
                            >
                                Cancel
                            </button>
                        </div>
                    </form>
                </div>
            </div>
        </>
    );
};

export default RestaurantBankDetails;
```


MealMingle for Business
admintest@gmail.com
Enter Bank Details
Account Number
Bank Name
Branch Name
IFSC Code
PAN Number
Aadhar Number
GST Number
Save Bank Details
Cancel


MealMingle for Business
admintest@gmail.com
Enter Bank Details
Account Number
999999999999
Bank Name
State Bank of India
Branch Name
Delhi
IFSC Code
A9999999999
PAN Number
AFZPK7190K
Aadhar Number
999999999999
GST Number
22AAAAA0000A1Z5
Save Bank Details
Cancel

On clicking on Save Bank Details button, it navigates us to the Register Restaurant Item page and the Admin Navbar is modified accordingly.



6.27. View Admin Bank Details Component

- The ViewAdminBankDetails component is used to display bank details for an admin.
- It fetches the bank details from a server and shows them on the page. The admin can also update or cancel the view.

Create a new file inside the components folder and name it ViewAdminBankDetails.tsx.

1. **Imports**
   - React and hooks like useEffect and useState are imported from the react library. They help manage component behavior and state.
   - useLocation, useNavigate, and useParams are from react-router-dom and help with navigation and route parameters.
   - AdminNavbar is a component that displays the navigation bar for admins.

```
import React, { useEffect, useState } from 'react';

import { useLocation, useNavigate, useParams } from 'react-router-dom';

import AdminNavbar from './AdminNavbar';
```

2. Creating the Component

- ViewAdminBankDetails is a functional component.
- useNavigate is used to navigate between different pages.
- useState is used to manage the state of bankDetails. Initially, it's set to null.

```
const ViewAdminBankDetails = () => {

    const navigate = useNavigate();

    const [bankDetails, setBankDetails] = useState<any>(null);
```

3. Fetching Data with `useEffect`

- useEffect is used to perform side effects, such as fetching data from a server.
- fetchBankDetails is an asynchronous function that makes a GET request to the server to get bank details.
- If the server response is successful and contains data, it updates bankDetails with the fetched data.

```
useEffect(() => {

        async function fetchBankDetails() {

            const response = await

fetch('http://localhost:8090/api/users/details', {

                method: 'GET',

                headers: {

                    'Content-Type': 'application/json',

                    'Authorization': `Bearer

${localStorage.getItem('token')}`

                }

            });

            const data = await response.json();

            if (data.error === "" && data.data !== null) {

                setBankDetails(data.data);
```

```
            }

        }

        fetchBankDetails();

    }, []);
```

4. Handling Button Clicks

- handleUpdate navigates to the page where you can update bank details.
- handleCancel navigates back to the page showing all restaurants.

```
const handleUpdate = () => {

        navigate('/update-admin-bank-details');

    };

    const handleCancel = () => {

        navigate('/view-admin-restaurants');

    };
```

5. Return JSX

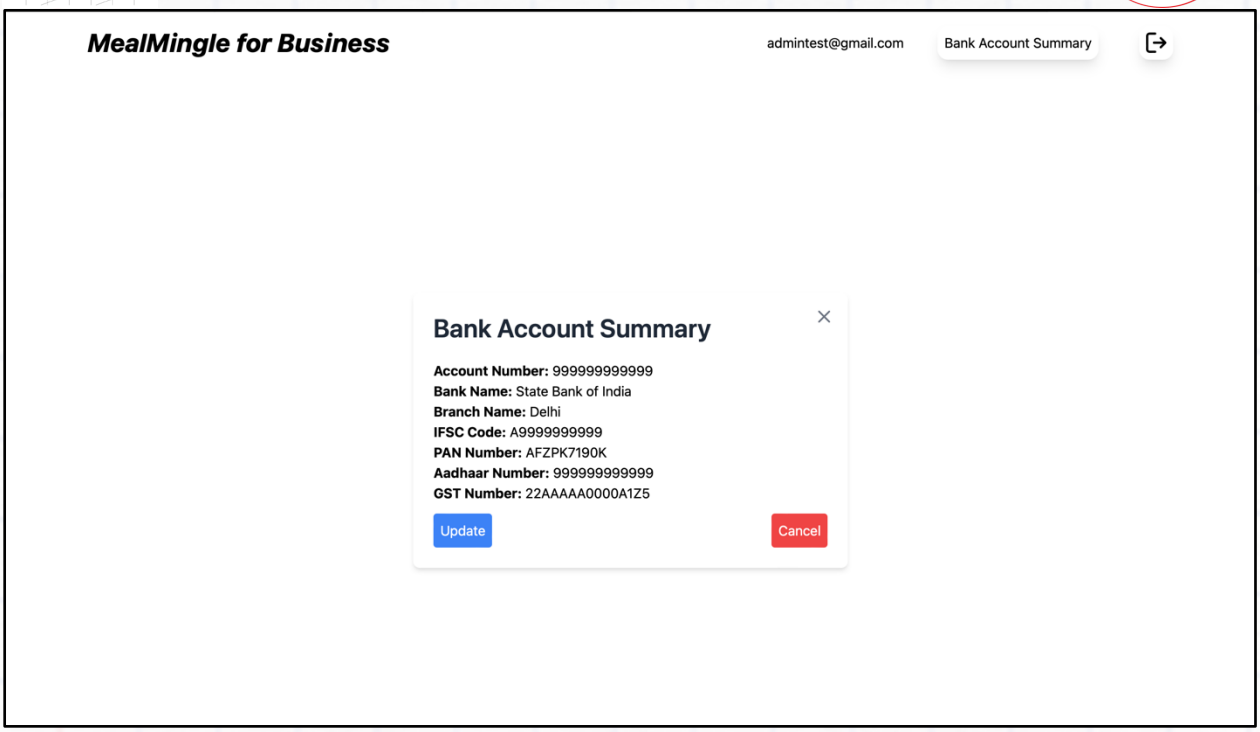
- `AdminNavbar` is rendered at the top.
- The main content is centered on the page with a white background and shadow.
- If `bankDetails` is not null, it displays the bank details in a formatted way.
- If `bankDetails` is null, it shows "No Bank Details Found!".
- Two buttons are provided:
  - **Update**: Navigates to the update page.
  - **Cancel**: Navigates back to the restaurant view page.

```
return (

        <>

            <AdminNavbar />

            <div className="flex justify-center items-center min-h-screen">
```

```
                <div className="w-full max-w-lg p-6 bg-white rounded-lg
shadow-md relative">
                    <div className="absolute top-0 right-0 m-4">
                        <button
                            onClick={handleCancel}
                            className="text-gray-500"
                        >
                            <svg
                                xmlns="http://www.w3.org/2000/svg"
                                className="h-6 w-6"
                                fill="none"
                                viewBox="0 0 24 24"
                                stroke="currentColor"
                            >
                                <path
                                    strokeLinecap="round"
                                    strokeLinejoin="round"
                                    strokeWidth="2"
                                    d="M6 18L18 6M6 6l12 12"
                                />
                            </svg>
                        </button>
                    </div>
                    <h3 className="text-3xl font-semibold text-gray-800 mb-
4">
```

```
                        Bank Account Summary
                    </h3>
                    {bankDetails ? (
                        <div className="pt-1">
                            <p><strong>Account Number:</strong>
{bankDetails.accountNumber}</p>
                            <p><strong>Bank Name:</strong>
{bankDetails.bankName}</p>
                            <p><strong>Branch Name:</strong>
{bankDetails.branchName}</p>
                            <p><strong>IFSC Code:</strong>
{bankDetails.ifscCode}</p>
                            <p><strong>PAN Number:</strong>
{bankDetails.panNumber}</p>
                            <p><strong>Aadhaar Number:</strong>
{bankDetails. adharNumber}</p>
                            <p><strong>GST Number:</strong>
{bankDetails.gstNumber}</p>
                            <div className="flex justify-between mt-3">
                                <button
                                    type="submit"
                                    onClick={handleUpdate}
                                    className="inline-block px-2 py-2 bg-
blue-500 text-white rounded hover:bg-blue-600 transition duration-300"
                                >
                                    Update
```

```
                                </button>
                                <button
                                    type="button"
                                    onClick={handleCancel}
                                    className="inline-block px-2 py-2 bg-red-
500 text-white rounded hover:bg-red-600 transition duration-300"
                                >
                                    Cancel
                                </button>
                            </div>
                        </div>
                    ) : (
                        <p>No Bank Details Found!</p>
                    )}
                </div>
            </div>
        </>
    );
};

export default ViewAdminBankDetails;
```



6.28. Update Admin Bank Details Component

- The UpdateAdminBankDetails component is part of a React application where an admin can update their bank details.
- It includes a form for entering and updating bank details, form validation, and navigation to different pages.

Create a new file inside the components folder and name it UpdateAdminBankDetails.tsx.

1. **Imports**
   - **React** and **useState, useEffect**: Import React and some hooks. useState manages the state of the component, and useEffect handles side effects.
   - **useNavigate, useLocation**: Hooks from react-router-dom to handle navigation and access route location.
   - **AdminNavbar**: A custom component to show the navigation bar.
   - **react-toastify**: A library for displaying toast notifications (pop-up messages).

```
import React, { useState, useEffect, FormEvent, ChangeEvent } from 'react';

import { useNavigate, useLocation } from 'react-router-dom';
```

```
import AdminNavbar from './AdminNavbar';

import { ToastContainer, toast } from 'react-toastify';

import 'react-toastify/dist/ReactToastify.css';
```

2. **BankDetails Interface**

BankDetails: Defines the structure of bank details data.

```
interface BankDetails {

    accountNumber: string;

    bankName: string;

    branchName: string;

    ifscCode: string;

    panNumber: string;

    adharNumber: string;

    gstNumber: string;

}
```

3. **Component Definition**

UpdateAdminBankDetails: Defines the functional component. `React.FC` is a type for functional components in TypeScript.

```
const UpdateAdminBankDetails: React.FC = () => {
```

4. **State and Effects**

- **navigate**: A function to programmatically navigate to different routes.
- **location**: Accesses the current route's location and state.
- **nextPage**: Determines where to navigate after updating details.
- **initialBankDetails**: Default values for the bank details form.
- **bankDetails**: State to hold current bank details.
- **errors**: State to store validation errors.

```
const navigate = useNavigate();
```

```
    const location = useLocation();
    const nextPage = location.state?.nextPage || '/view-admin-restaurants';


    const initialBankDetails: BankDetails = {
        accountNumber: '',
        bankName: '',
        branchName: '',
        ifscCode: '',
        panNumber: '',
        adharNumber: '',
        gstNumber: '',
    };


    const [bankDetails, setBankDetails] =
useState<BankDetails>(initialBankDetails);
    const [errors, setErrors] = useState<string[]>([]);
```

5. **Functions**

Save Bank Details

- **saveBankDetails**: Sends a request to update bank details and handles the response. Shows success or error messages using `toast`.

```
const saveBankDetails = async () => {
        const response = await
fetch('http://localhost:8090/api/users/details', {
            method: 'PUT',
            headers: {
```

```
                'Content-Type': 'application/json',
                'Authorization': `Bearer ${localStorage.getItem('token')}`
            },
            body: JSON.stringify(bankDetails)
        });
        const data = await response.json();
        console.log(data)
        if (data.error === '') {
            toast.success('Bank Details Updated Successfully!');
            setTimeout(() => {
                navigate(nextPage);
            }, 2000);
        } else {
            toast.error(data.error);
        }
    }
```

Handle Input Change

handleChange: Updates `bankDetails` state when user types in the form fields.

```
const handleChange = (e: ChangeEvent<HTMLInputElement>) => {
        const { name, value } = e.target;
        setBankDetails({ ...bankDetails, [name]: value });
    };
```

Handle Form Submission

- **handleSubmit**: Prevents the default form submission and validates the form. If valid, it calls `saveBankDetails`.

```
const handleSubmit = (e: FormEvent<HTMLFormElement>) => {

        e.preventDefault();



        const isValid = validateForm();

        if (!isValid) {

            return;

        }

        saveBankDetails();

    };
```

Handle Close

handleClose: Navigates to the `/view-admin-restaurants` page.

```
const handleClose = () => {

        navigate('/view-admin-restaurants');

    };
```

Form Validation

validateForm: Checks each form field for validity and adds errors to the `errors` array if needed.

```
const validateForm = () => {

        let isValid = true;

        const errors: string[] = [];



        // Account Number

        if (!bankDetails.accountNumber.trim()) {
```

```
        errors.push('Account Number is Required.');
        isValid = false;
    } else if (isNaN(Number(bankDetails.accountNumber))) {
        errors.push('Account Number must be a Number, not a String.');
        isValid = false;
    } else if (bankDetails.accountNumber.trim().length !== 12) {
        errors.push('Account Number must be exactly 12 Digits.');
        isValid = false;
    }

    // Bank Name
    if (!bankDetails.bankName.trim()) {
        errors.push('Bank Name is Required.');
        isValid = false;
    }

    // Branch Name
    if (!bankDetails.branchName.trim()) {
        errors.push('Branch Name is Required.');
        isValid = false;
    }

    // IFSC Code
    if (!bankDetails.ifscCode.trim()) {
        errors.push('IFSC Code is Required.');
```

```
            isValid = false;
        } else if (bankDetails.ifscCode.trim().length !== 11) {
            errors.push('IFSC Code must be exactly 11 Characters.');
            isValid = false;
        }

        // PAN Number
        if (!bankDetails.panNumber.trim()) {
            errors.push('PAN Number is Required.');
            isValid = false;
        } else if (!/^([A-Z]){5}([0-9]){4}([A-
Z]){1}$/.test(bankDetails.panNumber.trim())) {
            errors.push('Invalid PAN Number Format. Example: ABCDE1234F');
            isValid = false;
        }

        // Aadhar Number
        if (!bankDetails.adharNumber.trim()) {
            errors.push('Aadhar Number is Required.');
            isValid = false;
        } else if (isNaN(Number(bankDetails.adharNumber))) {
            errors.push('Aadhar Number must contain only Numeric
Characters.');
            isValid = false;
        } else if (bankDetails.adharNumber.trim().length !== 12) {
```

```
            errors.push('Aadhar Number must be exactly 12 Digits.');
            isValid = false;
        }

        // GST Number
        if (!bankDetails.gstNumber.trim()) {
            errors.push('GST Number is Required.');
            isValid = false;
        } else if (!/^\d{2}[A-Z]{5}\d{4}[A-Z]{1}\d[Z]{1}[A-Z\d]{1}$/.test(bankDetails.gstNumber.trim())) {
            errors.push('Invalid GST Number Format.');
            isValid = false;
        }

        setErrors(errors);
        if (errors.length > 0) {
            toast.error(errors[0]);
        }

        return isValid;
    };
```

6. Return JSX

- **AdminNavbar**: Displays the navigation bar.
- **Form**: Contains input fields for bank details. Submits data to update it.
- **Buttons**: "Update" to submit the form and "Cancel" to go back to the previous page.

```
return (
```

```
        <>
            <AdminNavbar />
            <div className="flex justify-center items-center min-h-screen">
                <div className="w-full max-w-2xl p-6 bg-white rounded-lg
shadow-md relative">
                    <div className="absolute top-0 right-0 m-4">
                        <button
                            onClick={handleClose}
                            className="text-gray-500"
                        >
                            <svg
                                xmlns="http://www.w3.org/2000/svg"
                                className="h-6 w-6"
                                fill="none"
                                viewBox="0 0 24 24"
                                stroke="currentColor"
                            >
                                <path
                                    strokeLinecap="round"
                                    strokeLinejoin="round"
                                    strokeWidth="2"
                                    d="M6 18L18 6M6 6l12 12"
                                />
                            </svg>
                        </button>
```

```
                    </div>
                    <h3 className="text-3xl font-semibold text-gray-800 mb-
4">
                        Update Bank Details
                    </h3>
                    <form onSubmit={handleSubmit} className="space-y-4">
                        <div className="grid grid-cols-2 gap-4">
                            <div className="col-span-1">
                                <label className="block text-base font-medium
text-gray-700">
                                    Account Number
                                </label>
                                <input
                                    type="text"
                                    id="accountNumber"
                                    name="accountNumber"
                                    value={bankDetails.accountNumber}
                                    onChange={handleChange}
                                    className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                                />
                            </div>
                            <div className="col-span-1">
                                <label className="block text-base font-medium
text-gray-700">
```

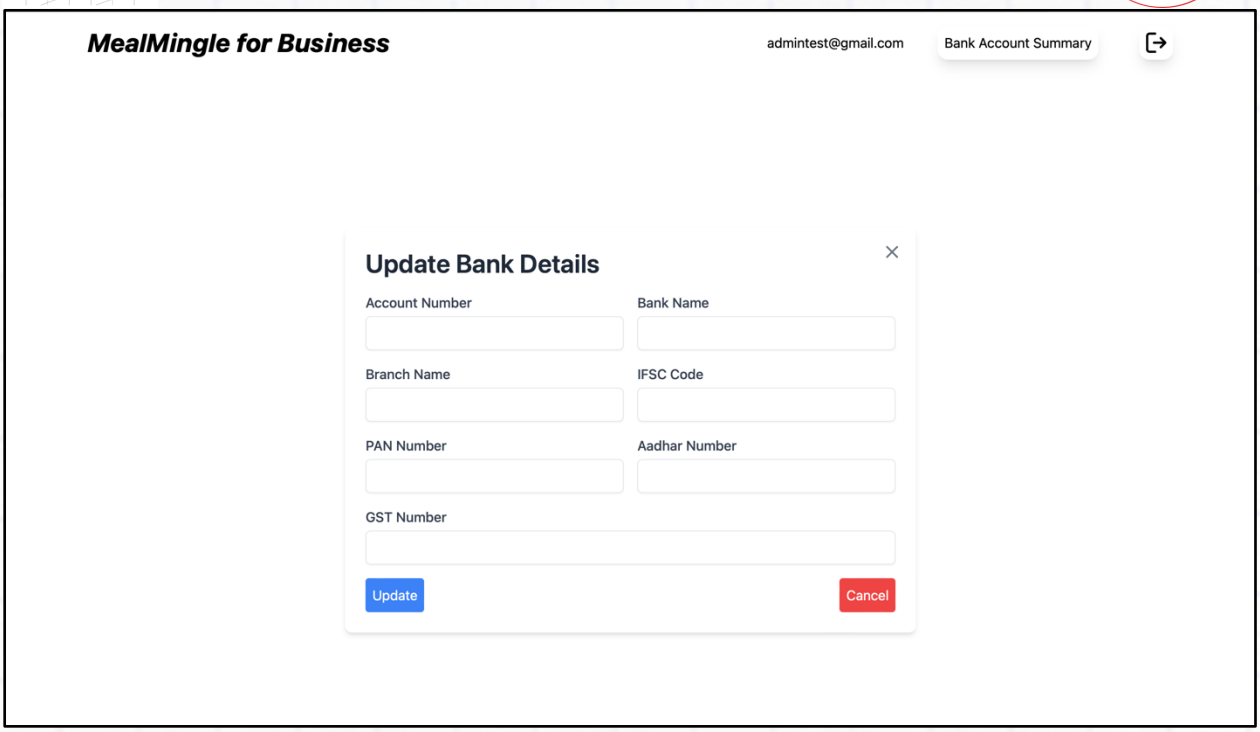
```
                                    Bank Name
                                </label>
                                <input
                                    type="text"
                                    id="bankName"
                                    name="bankName"
                                    value={bankDetails.bankName}
                                    onChange={handleChange}
                                    className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                                />
                            </div>
                        </div>
                        <div className="grid grid-cols-2 gap-4">
                            <div className="col-span-1">
                                <label className="block text-base font-medium
text-gray-700">
                                    Branch Name
                                </label>
                                <input
                                    type="text"
                                    id="branchName"
                                    name="branchName"
                                    value={bankDetails.branchName}
                                    onChange={handleChange}
```

```
                                    className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                                />
                            </div>
                            <div className="col-span-1">
                                <label className="block text-base font-medium
text-gray-700">
                                    IFSC Code
                                </label>
                                <input
                                    type="text"
                                    id="ifscCode"
                                    name="ifscCode"
                                    value={bankDetails.ifscCode}
                                    onChange={handleChange}
                                    className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                                />
                            </div>
                        </div>
                        <div className="grid grid-cols-2 gap-4">
                            <div className="col-span-1">
                                <label className="block text-base font-medium
text-gray-700">
                                    PAN Number
```

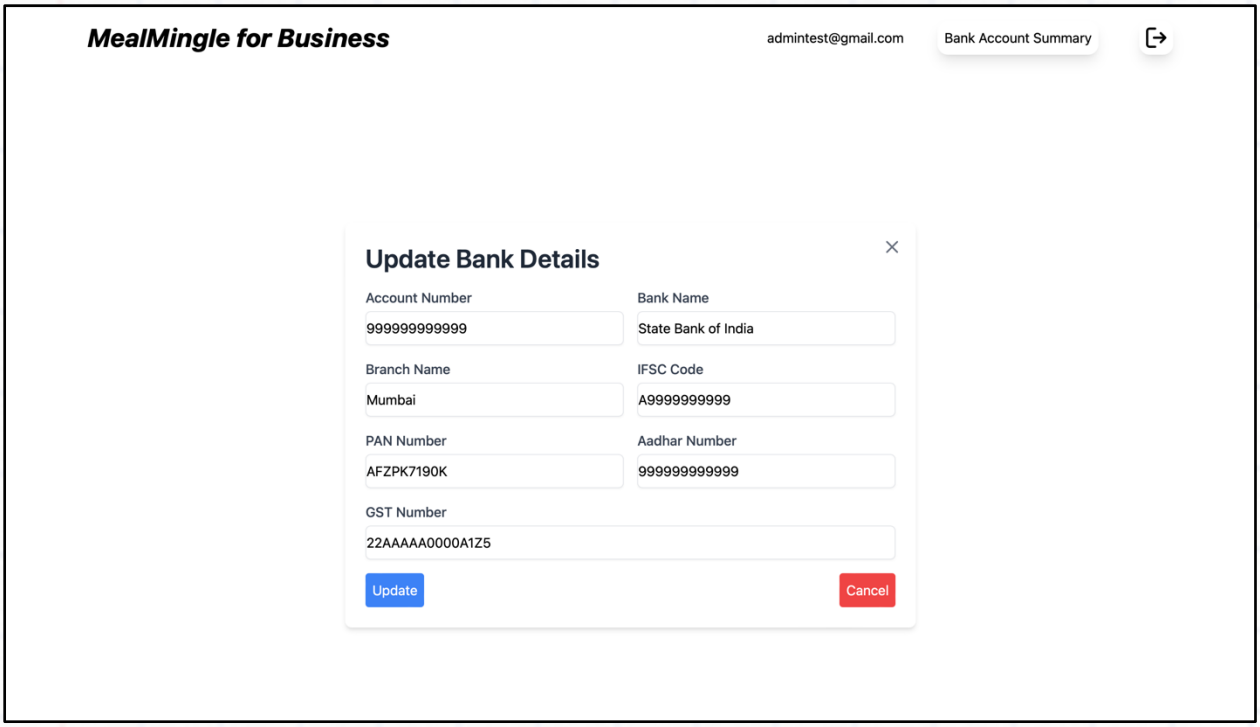
```
                                </label>
                                <input
                                    type="text"
                                    id="panNumber"
                                    name="panNumber"
                                    value={bankDetails.panNumber}
                                    onChange={handleChange}
                                    className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                                />
                            </div>
                            <div className="col-span-1">
                                <label className="block text-base font-medium
text-gray-700">
                                    Aadhar Number
                                </label>
                                <input
                                    type="text"
                                    id="adharNumber"
                                    name="adharNumber"
                                    value={bankDetails.adharNumber}
                                    onChange={handleChange}
                                    className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                                />
```

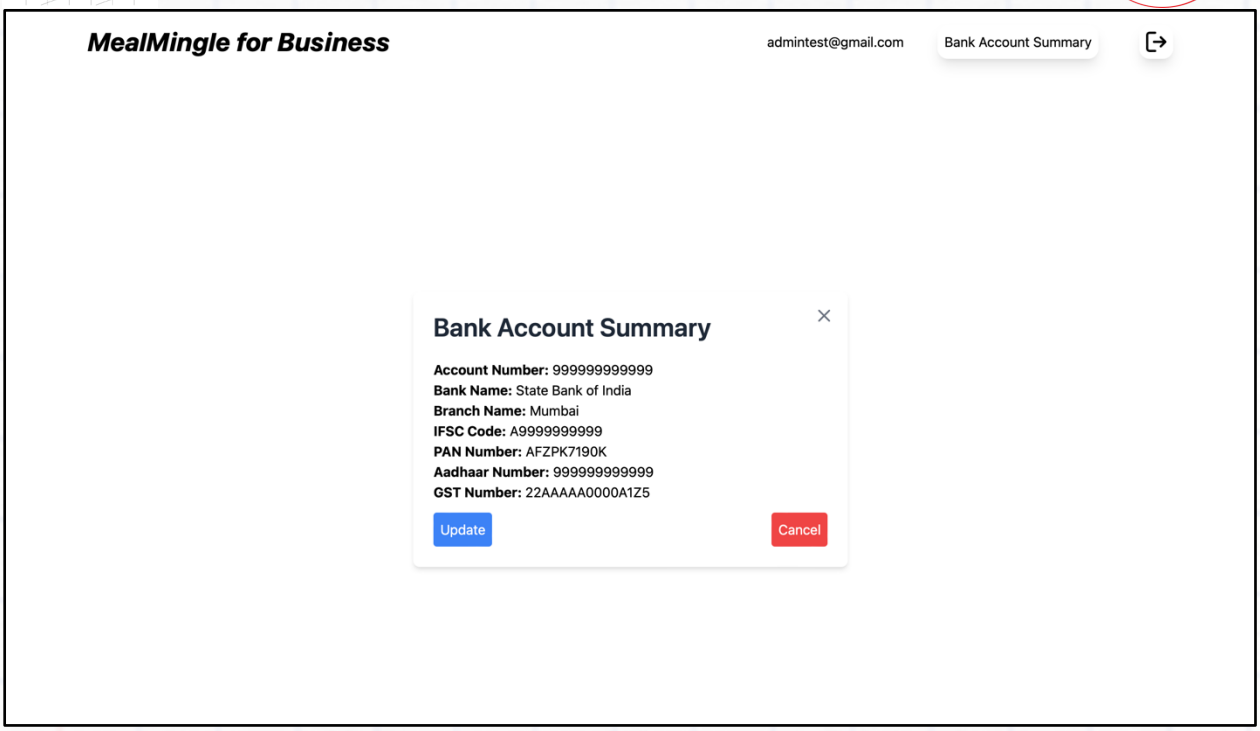
```
                        </div>
                    </div>
                    <div className="border-gray-300">
                        <label className="block text-base font-medium
text-gray-700">
                            GST Number
                        </label>
                        <input
                            type="text"
                            id="gstNumber"
                            name="gstNumber"
                            value={bankDetails.gstNumber}
                            onChange={handleChange}
                            className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                        />
                    </div>
                    <div className="flex justify-between">
                        <button
                            type="submit"
                            className="inline-block px-2 py-2 bg-blue-500
text-white rounded hover:bg-blue-600 transition duration-300"
                        >
                            Update
                        </button>
```

```
                            <button
                                type="button"
                                onClick={handleClose}
                                className="inline-block px-2 py-2 bg-red-500
text-white rounded hover:bg-red-600 transition duration-300"
                            >
                                Cancel
                            </button>
                        </div>
                    </form>
                </div>
            </div>
        </>
    );
};

export default UpdateAdminBankDetails;
```



Now, let’s change the Branch Name of The Taj Hotel from Delhi to Mumbai.





6.29. Register Restaurant Item Component

- The RegisterRestaurantItem component is a React functional component that provides a form for registering new items for a restaurant.
- This involves filling out various details about the item and submitting the form.
- It uses React hooks for managing state, handling form changes, and performing actions like submission.
- It also makes use of react-toastify for displaying success or error messages.

Create a new file inside the components folder and name it RegisterRestaurantItem.tsx.

1. **Imports**
   - useState is a React hook for managing state in functional components.
   - ChangeEvent and FormEvent are TypeScript types for event handling.
   - useNavigate is a hook from react-router-dom for navigation.
   - useParams is a hook to access URL parameters.
   - ToastContainer and toast from react-toastify are used for showing notifications.

```
import React, { useState, ChangeEvent, FormEvent } from 'react';

import { useNavigate, useParams } from 'react-router-dom';
```

```
import AdminNavbar from './AdminNavbar';

import { ToastContainer, toast } from 'react-toastify';

import 'react-toastify/dist/ReactToastify.css';
```

2. Interfaces

- **RestaurantItem**: Defines the structure of a restaurant item with properties such as restaurantItemName, restaurantItemPrice, etc.
- **Restaurant**: Defines the structure of a restaurant, including its items.
- **RegisterRestaurantItemProps**: Describes optional props for the component, specifically an onSubmit callback function.

```
interface RestaurantItem {

    restaurantItemId?: string;

    restaurantItemName: string;

    restaurantItemPrice: number;

    restaurantItemCategory: string;

    restaurantItemImageUrl: string;

    restaurantItemCuisineType: string;

    restaurantItemVeg: boolean;

    restaurantId: string | undefined

}


interface Restaurant {

    restaurantId: string;

    restaurantName: string;

    restaurantAddress: string;

    restaurantItems: RestaurantItem[];

}
```

```
interface RegisterRestaurantItemProps {

    onSubmit?: () => void;

}
```

3. Categories and Cuisine Types

- **Categories Array**: A list of predefined food categories used in the form dropdown for item registration.
- **Cuisine Types Array**: A list of predefined cuisine types available for selection during item registration.

```
onst categories = [

    'Pizza', 'Burger', 'Noodles', 'Pasta', 'Sandwich',

    'Thali', 'Dessert', 'Salad', 'Biryani', 'Fish'

];



const cuisineTypes = [

    'North Indian', 'South Indian'

];
```

4. Component Function

- **RegisterRestaurantItem**: The functional component.
- **useParams**: Gets route parameters (e.g., restaurantId) from the URL.

```
const RegisterRestaurantItem: React.FC<RegisterRestaurantItemProps> = ({

onSubmit }) => {

    const value = useParams();

    console.log(value);
```

5. Initial State and Handlers

- **Initial State**: `initialRestaurantItem` holds the initial values for the restaurant item form.

- **State Management**:
- `restaurantItem`: State for storing form values.
- `errors`: State for storing validation error messages.
- `useNavigate`: Hook for navigating between routes.

```
const initialRestaurantItem: RestaurantItem = {

        restaurantItemName: '',

        restaurantItemPrice: 0,

        restaurantItemCategory: '',

        restaurantItemImageUrl: '',

        restaurantItemCuisineType: '',

        restaurantItemVeg: true,

        restaurantId: value.restaurantId

    };



    const [restaurantItem, setRestaurantItem] =

useState<RestaurantItem>(initialRestaurantItem);

    const [errors, setErrors] = useState<string[]>([]);

    const navigate = useNavigate();
```

6. Event Handlers

- **handleChange**: Updates state when input values change.

- Specifically checks if the `restaurantItemPrice` is non-negative.
- **handleCheckboxChange**: Updates the restaurantItemVeg boolean when the checkbox is toggled.

```
const handleChange = (e: ChangeEvent<HTMLInputElement |

HTMLSelectElement>) => {

        const { name, value } = e.target;
```

```
        if (name === 'restaurantItemPrice' && +value < 0) {
            return;
        }
        setRestaurantItem(prevItem => ({
            ...prevItem,
            [name]: value,
        }));
    };

    const handleCheckboxChange = (e: ChangeEvent<HTMLInputElement>) => {
        const { name, checked } = e.target;
        setRestaurantItem(prevItem => ({
            ...prevItem,
            [name]: checked,
        }));
    };
```

7. Form Submission and Validation

- **addRestaurantItem**: Sends a POST request to the server to add the restaurant item.
    - If successful, it triggers a success toast; otherwise, it shows an error toast.
- **handleSubmit**: Validates the form and submits it.
    - If validation passes, the item is added, and the user is redirected to a list of items for that restaurant.

```
async function addResturantItem() {
        const restaurantItemData = {
            restaurantItem: {
```

```
                restaurantItemName: restaurantItem.restaurantItemName,
                restaurantItemPrice: restaurantItem.restaurantItemPrice,
                restaurantItemCategory:
restaurantItem.restaurantItemCategory,
                restaurantItemImageUrl:
restaurantItem.restaurantItemImageUrl,
                restaurantItemCuisineType:
restaurantItem.restaurantItemCuisineType,
                restaurantItemVeg: restaurantItem.restaurantItemVeg,
                restaurantId: restaurantItem.restaurantId
            }
        }
        const response = await
fetch(`http://localhost:8091/api/restaurant/items/add`, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'Authorization': `Bearer ${localStorage.getItem('token')}`
            },
            body: JSON.stringify(restaurantItemData),
        });
        const data = await response.json();
        console.log(data);
        if (data.error == "" && data.data != null) {
            toast.success('Restaurant Item Added Successfully!');
```

```
        } else {
            toast.error(data.error);
        }
    }
    const handleSubmit = (e: FormEvent<HTMLFormElement>) => {
        e.preventDefault();

        const isValid = validateForm();
        if (!isValid) {
            return;
        }
        addResturantItem();
        if (onSubmit) onSubmit();
        setTimeout(() => {
            navigate(`/view-admin-restaurant-
items/${value.restaurantId}`);
        }, 2000);
    };
```

8. Navigation and Validation

- **handleClose**: Navigates back to the list of restaurants if the user decides to cancel.
- **validateForm**: Validates the form input fields and updates the `errors` state with any validation messages.
    - Ensures that required fields are filled out, with additional checks on item name length and price.

```
const handleClose = () => {
        navigate('/view-admin-restaurants');
```

```
    };

    const validateForm = () => {
        let isValid = true;
        const errors: string[] = [];

        // Item Name
        if (!restaurantItem.restaurantItemName.trim()) {
            errors.push('Item Name is Required.');
            isValid = false;
        } else if (/^\d+$/.test(restaurantItem.restaurantItemName.trim()))
{
            errors.push('Item Name must be a String, not a Number.');
            isValid = false;
        } else if (restaurantItem.restaurantItemName.trim().length < 5 ||
restaurantItem.restaurantItemName.trim().length > 20) {
            errors.push('Item Name must be between 5 to 20 Characters
long.');
            isValid = false;
        }

        // Price (INR)
        if (restaurantItem.restaurantItemPrice <= 0) {
            errors.push('Price must be greater than 0.');
            isValid = false;
```

```
    }

    // Category
    if (!restaurantItem.restaurantItemCategory.trim()) {
        errors.push('Category is Required.');
        isValid = false;
    }

    // Cuisine Type
    if (!restaurantItem.restaurantItemCuisineType.trim()) {
        errors.push('Cuisine Type is Required.');
        isValid = false;
    }

    // Image URL
    if (!restaurantItem.restaurantItemImageUrl.trim()) {
        errors.push('Image URL is Required.');
        isValid = false;
    }

    setErrors(errors);
    if (errors.length > 0) {
        toast.error(errors[0]);
    }
```

```
        return isValid;

    }
```

9. Return JSX

- **Form Layout**: The form consists of multiple input fields for different properties of a restaurant item, such as name, price, category, etc.
- **Button to Close Form**: At the top-right, there's a button to close the form, which triggers handleClose.
- **Form Submission**: The form submission is handled by handleSubmit, which triggers the item registration process.
- **ToastContainer**: Renders any toasts triggered during form submission or validation.

```
return (

        <>

            <AdminNavbar />

            <div className="flex justify-center items-center min-h-
screen">

                <div className="w-full max-w-2xl p-6 bg-white rounded-lg
shadow-md relative">

                    <div className="absolute top-0 right-0 m-4">

                        <button

                            onClick={handleClose}

                            className="text-gray-500"

                        >

                            <svg

                                xmlns="http://www.w3.org/2000/svg"

                                className="h-6 w-6"

                                fill="none"

                                viewBox="0 0 24 24"
```

```
                            stroke="currentColor"
                        >
                            <path
                                strokeLinecap="round"
                                strokeLinejoin="round"
                                strokeWidth="2"
                                d="M6 18L18 6M6 6l12 12"
                            />
                        </svg>
                    </button>
                </div>
                <h3 className="text-3xl font-semibold text-gray-800
mb-4">
                    Register New Restaurant Item
                </h3>
                <form onSubmit={handleSubmit} className="space-y-4">
                    <div className="grid grid-cols-2 gap-4">
                        <div className="col-span-2">
                            <label className="block text-base font-
medium text-gray-700">
                                Item Name
                            </label>
                            <input
                                type="text"
                                name="restaurantItemName"
```

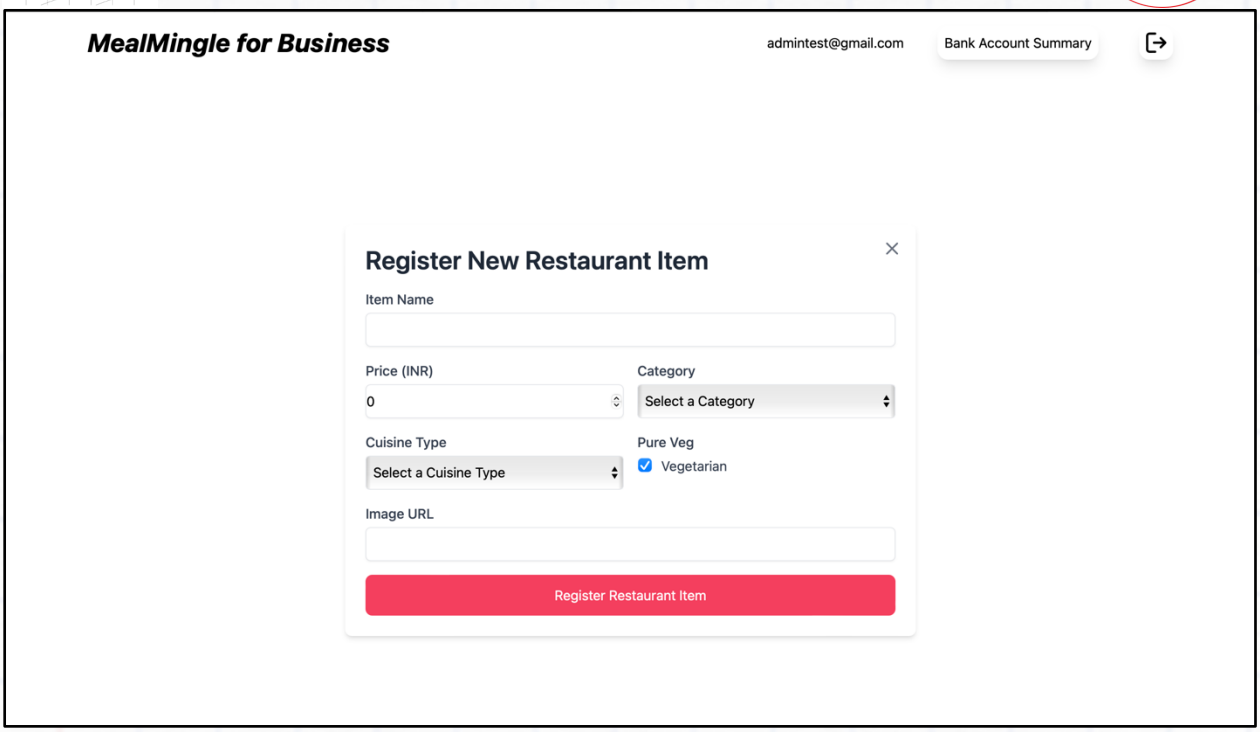
```
value={restaurantItem.restaurantItemName}
                                    onChange={handleChange}
                                    className="form-input mt-1 block w-
full rounded-md shadow-sm border h-10"
                                />
                            </div>
                        </div>
                        <div className="grid grid-cols-2 gap-4">
                            <div className="col-span-1">
                                <label className="block text-base font-
medium text-gray-700">
                                    Price (INR)
                                </label>
                                <input
                                    type="number"
                                    name="restaurantItemPrice"

value={restaurantItem.restaurantItemPrice}
                                    onChange={handleChange}
                                    className="form-input mt-1 block w-
full rounded-md shadow-sm border h-10"
                                />
                            </div>
                            <div className="col-span-1">
```

```
                                <label className="block text-base font-
medium text-gray-700">
                                    Category
                                </label>
                                <select
                                    name="restaurantItemCategory"

value={restaurantItem.restaurantItemCategory}
                                    onChange={handleChange}
                                    className="form-select mt-1 block w-
full rounded-md shadow-sm border h-10"
                                >
                                    <option value="">Select a
Category</option>
                                    {categories.map(category => (
                                        <option key={category}
value={category}>{category}</option>
                                    ))}
                                </select>
                            </div>
                        </div>
                        <div className="grid grid-cols-2 gap-4">
                            <div className="col-span-1">
                                <label className="block text-base font-
medium text-gray-700">
```

```
                                    Cuisine Type
                                </label>
                                <select
                                    name="restaurantItemCuisineType"

value={restaurantItem.restaurantItemCuisineType}
                                    onChange={handleChange}
                                    className="form-select mt-1 block w-
full rounded-md shadow-sm border h-10"
                                >
                                    <option value="">Select a Cuisine
Type</option>
                                    {cuisineTypes.map(cuisine => (
                                        <option key={cuisine}
value={cuisine}>{cuisine}</option>
                                    ))}
                                </select>
                            </div>
                            <div className="col-span-1">
                                <label className="block text-base font-
medium text-gray-700">

                                    Pure Veg
                                </label>
                                <div className="flex items-center mt-1">
                                    <input
```

```
                                    type="checkbox"
                                    name="restaurantItemVeg"

checked={restaurantItem.restaurantItemVeg}
                                    onChange={handleCheckboxChange}
                                    className="form-checkbox h-5 w-5
text-rose-600"
                                />
                                <span className="ml-2 text-base text-
gray-700">Vegetarian</span>
                            </div>
                        </div>
                    </div>
                    <div className="border-gray-300">
                        <label className="block text-base font-medium
text-gray-700">
                            Image URL
                        </label>
                        <input
                            type="text"
                            name="restaurantItemImageUrl"

value={restaurantItem.restaurantItemImageUrl}
                            onChange={handleChange}
```

```
                                    className="form-input mt-1 block w-full
rounded-md shadow-sm border h-10"
                                />
                            </div>
                            <button
                                type="submit"
                                className="bg-rose-500 w-full h-12 text-white
py-2 px-4 rounded-lg"
                            >
                                Register Restaurant Item
                            </button>
                        </form>
                    </div>
                </div>
            </>
        );
};

export default RegisterRestaurantItem;
```



6.30. View Admin Restaurant Items Component

- This React component, ViewAdminRestaurantItems, is designed to allow an admin user to view, update, and delete items from a specific restaurant's menu.

Create a new file inside the components folder and name it ViewAdminRestaurantItems.tsx.

1. **Imports**
   - **React, useEffect, useState**: These are imported from React to create components, manage side effects (like fetching data), and store and manage state (data that changes over time).
   - **AdminNavbar**: A custom component that likely contains navigation links for the admin.
   - **useNavigate, useParams**: Hooks from React Router for navigation and accessing URL parameters, respectively.
   - **toast**: Used for showing notifications to the user, such as success or error messages.
   - **'react-toastify/dist/ReactToastify.css'**: This imports the default styling for the toast notifications.

```
import React, { useEffect, useState } from 'react';

import AdminNavbar from './AdminNavbar';
```

```
import { Link, useNavigate, useParams } from 'react-router-dom';

import { toast } from 'react-toastify';

import 'react-toastify/dist/ReactToastify.css';
```

2. Interface Definition

This defines the structure of a `RestaurantItem` object. It's a TypeScript interface that describes the properties each restaurant item should have.

```
interface RestaurantItem {

    restaurantItemId: string;

    restaurantItemName: string;

    restaurantItemPrice: number;

    restaurantItemCategory: string;

    restaurantItemImageUrl: string;

    restaurantItemCuisineType: string;

    restaurantItemVeg: boolean;

    restaurantId: string;

}
```

3. Component Definition

- **useParams**: Extracts the `restaurantId` from the URL. This ID is used to fetch items for a specific restaurant.
- **useState**: Creates a state variable `restaurantItems` to store the list of items, and `setRestaurantItems` to update this state.
- **useNavigate**: A hook for programmatically navigating the user to different pages.

```
const ViewAdminRestaurantItems: React.FC = () => {

    const { restaurantId } = useParams<{ restaurantId: string }>();
```

```
    const [restaurantItems, setRestaurantItems] =
useState<RestaurantItem[]>([]);
    const navigate = useNavigate();
```

4. Fetching Restaurant Items

- **fetchRestaurantItems**: An asynchronous function to get the restaurant items from a backend API.
- It sends a GET request to the API endpoint, using the restaurantId from the URL.
- **Authorization**: Sends a token stored in localStorage to authorize the request.
- **toast.success** and **toast.error**: Displays success or error messages depending on the API response.
- **setRestaurantItems**: Updates the state with the fetched restaurant items.

```
const fetchRestaurantItems = async() => {
        const response = await
fetch(`http://localhost:8091/api/restaurant/${restaurantId}/items`, {
            method: 'GET',
            headers: {
                'Content-Type': 'application/json',
                'Authorization': `Bearer ${localStorage.getItem('token')}`
            }
        });
        const data = await response.json();
        if (data.error === "" && data.data !== null) {
            toast.success('Restaurant Items Fetched Successfully!');
            setRestaurantItems(data.data.restaurantItems);
        } else {
```

```
            toast.error(data.error);

        }

    }
```

useEffect Hook

useEffect: This hook runs `fetchRestaurantItems` when the component first renders. The empty array `[]` ensures it only runs once.

```
useEffect(() => {
        fetchRestaurantItems();
    },[])
```

5. Handling Update and Delete

- **handleUpdate**: Navigates to an update page for a specific restaurant item when the "Update" button is clicked.
- **handleDelete**: Sends a `DELETE` request to the API to remove a specific restaurant item. If successful, it refreshes the list of items by calling `fetchRestaurantItems` again.

```
const handleUpdate = (restaurantItemId: string) => {

        navigate(`/update-restaurant-

item/${restaurantId}/${restaurantItemId}`);

    };



    const handleDelete = async (restaurantItemId: string) => {

        if (restaurantId === undefined) {

            return;

        }

        const response = await

fetch(`http://localhost:8091/api/restaurants/${restaurantId}/items/${resta

urantItemId}`, {
```

```
            method: 'DELETE',
            headers: {
            'Content-Type': 'application/json',
            'Authorization': `Bearer ${localStorage.getItem('token')}`
            }
        });

        const data = await response.json();
        if (data.error === "") {
            toast.success('Restaurant Item Deleted Successfully!');
            fetchRestaurantItems();
        } else {
            toast.error(data.error);
        }
        return;
    };
```

6. Return JSX

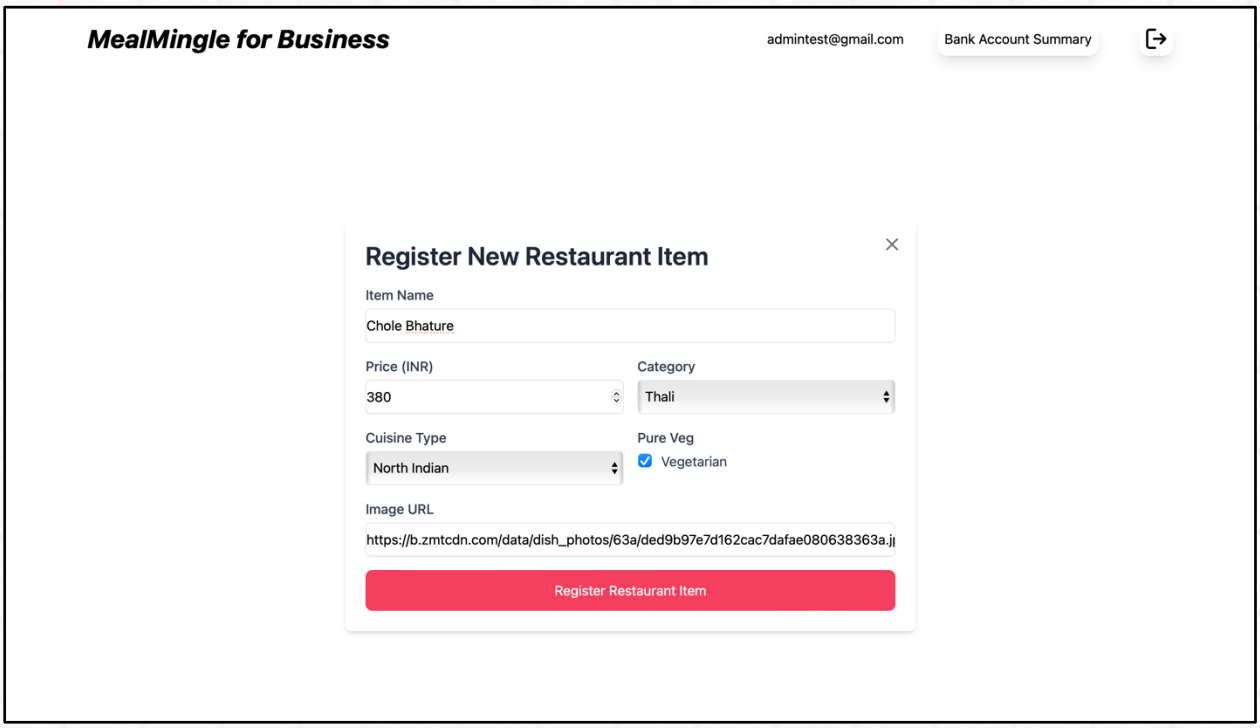
- **AdminNavbar**: Renders the admin navigation bar at the top.
- **Conditional Rendering**: If there are no restaurant items (`restaurantItems.length === 0`), it shows a message and a link to register a new item.
- **Grid Layout**: If there are items, they are displayed in a grid, each with options to update or delete the item.

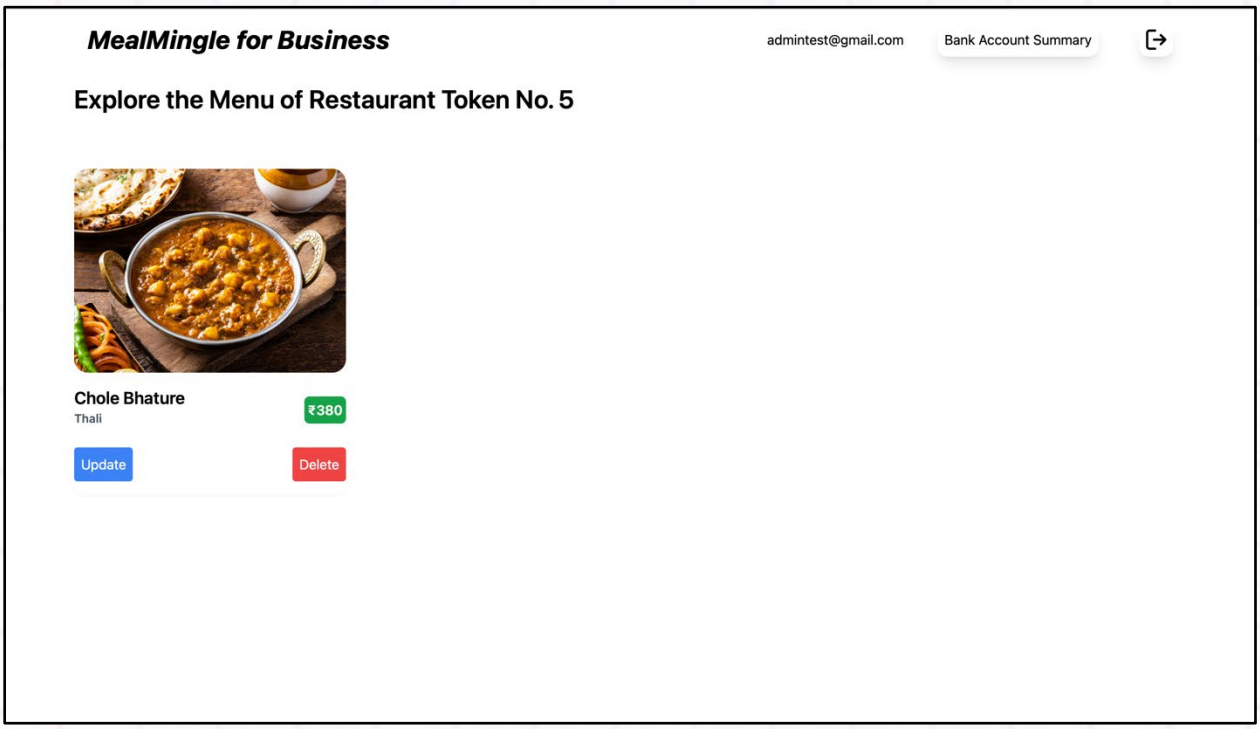
```
return (
        <>
            <AdminNavbar />
```

```
            <div className='p-4 pl-20'>
                <h1 className='font-semibold text-3xl mb-4'>Explore the
Menu of Restaurant Token No. {restaurantId}</h1>
                {restaurantItems.length === 0 ? (
                    <div className="text-center text-xl">
                        <p className="mt-60 mb-4">No Restaurant Items
Registered Yet!</p>
                        <p>Please <Link to={`/register-restaurant-
item/${restaurantId}`} className="text-blue-500 underline">Register a
Restaurant Item</Link> to View.</p>
                    </div>
                ) : (
                    <div className='grid grid-cols-3 gap-4'>
                        {restaurantItems.map((item: RestaurantItem) => (
                            <div key={item.restaurantItemId}
className="relative max-w-xs rounded-xl overflow-hidden shadow-sm mt-12
cursor-pointer">
                                <img className={`w-full rounded-2xl h-60`}
src={item.restaurantItemImageUrl} alt="Restaurant Item Image" />
                                <div className="py-4">
                                    <div className='flex justify-between
items-center'>
                                        <div className="font-semibold
text-xl mb-2">
                                            {item.restaurantItemName}
```

```
                                        <div className="text-sm text-
gray-600">{item.restaurantItemCategory}</div>
                                    </div>
                                    <div className="text-white font-
semibold text-base rounded-md p-1 bg-green-600">
                                        ₹{item.restaurantItemPrice}
                                    </div>
                                </div>
                                <div className="flex justify-between
mt-4">
                                    <button
                                        onClick={() =>
handleUpdate(item.restaurantItemId)}
                                        className="inline-block px-2
py-2 bg-blue-500 text-white rounded hover:bg-blue-600 transition duration-
300"
                                    >
                                        Update
                                    </button>
                                    <button
                                        onClick={() =>
handleDelete(item.restaurantItemId)}
                                        className="inline-block px-2
py-2 bg-red-500 text-white rounded hover:bg-red-600 transition duration-
300"
```

```
                                    >
                                        Delete
                                    </button>
                                </div>
                            </div>
                        </div>
                    ))}
                </div>)}
            </div>
        </>
    );
};

export default ViewAdminRestaurantItems;
```

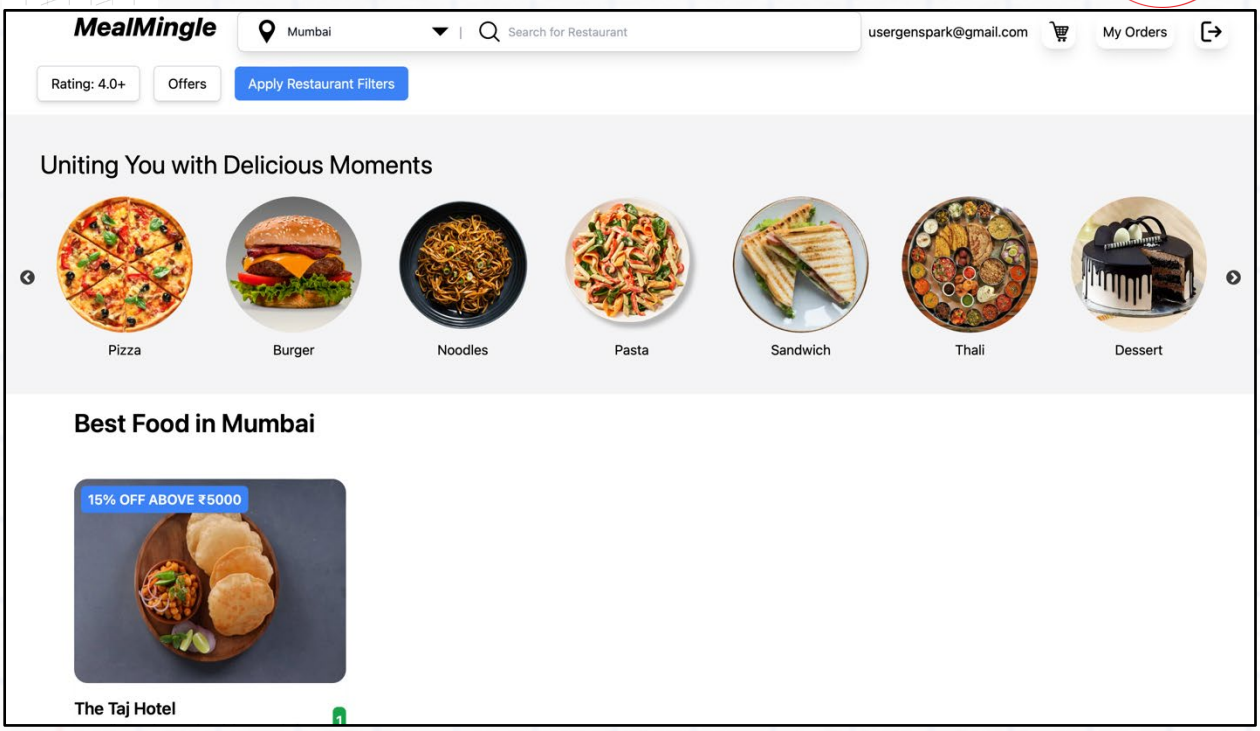


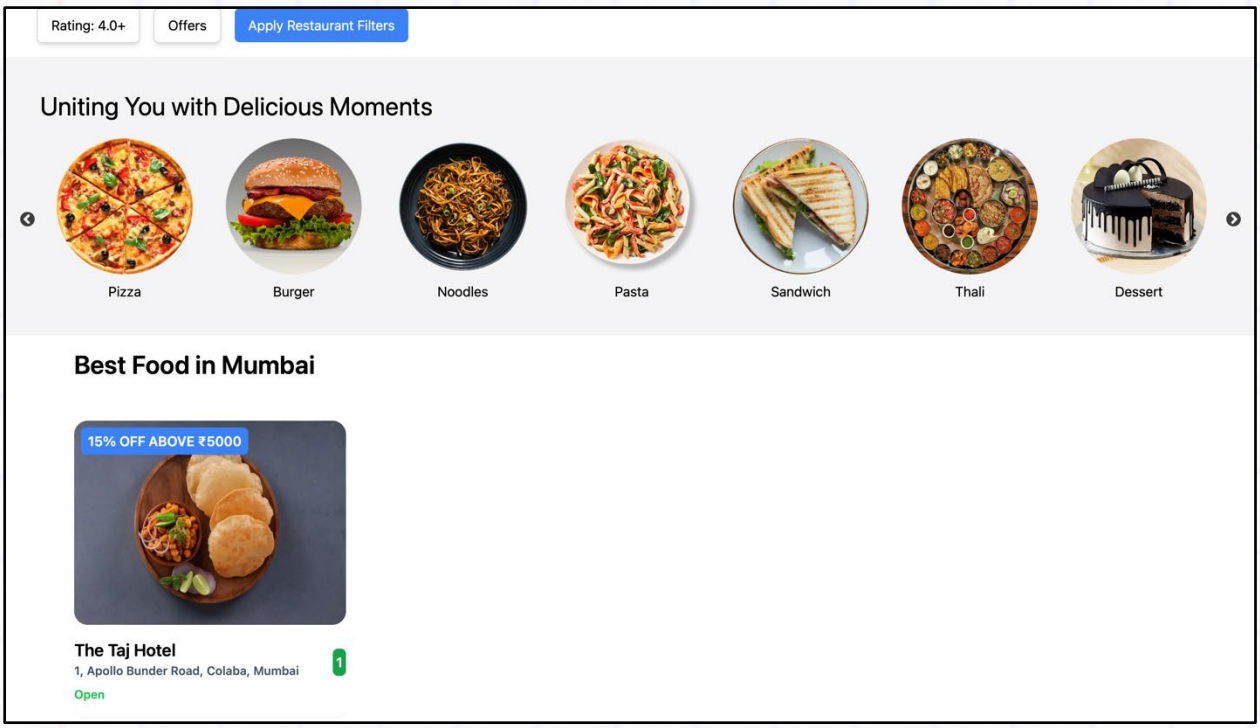


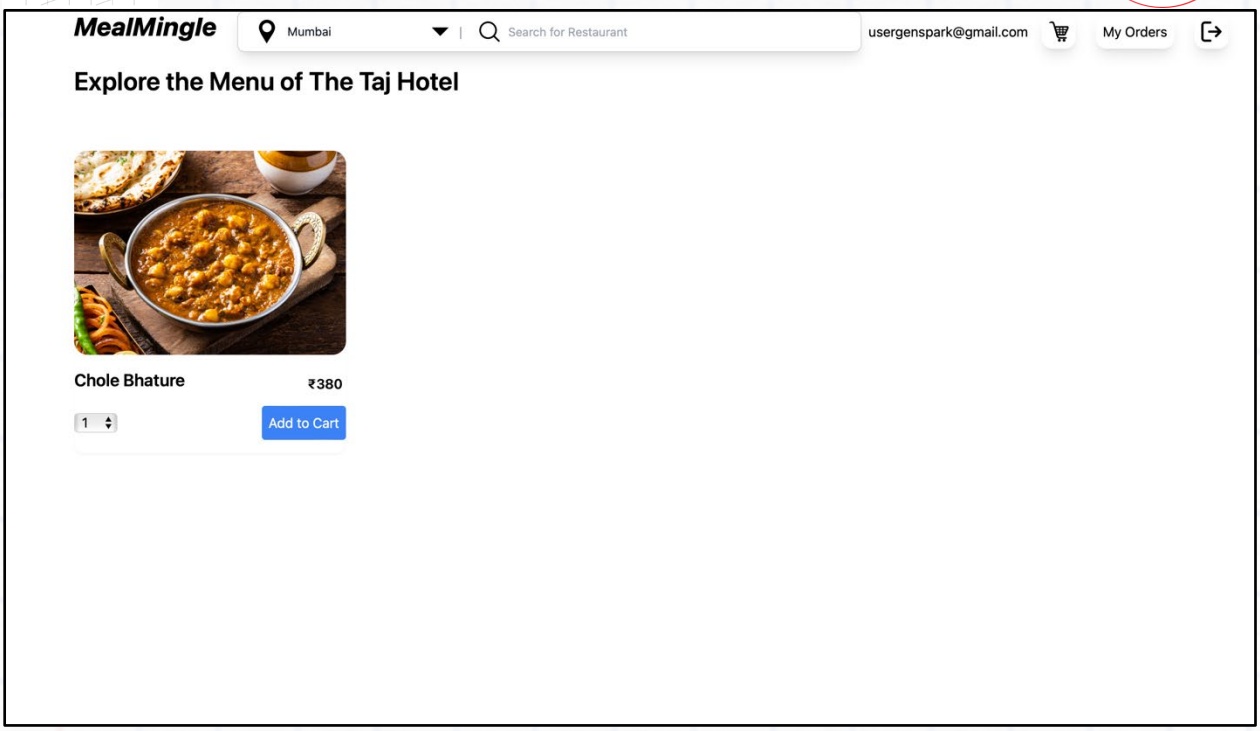
Task: Implement the functionality of Updating a Restaurant Item.

The file name is UpdateRestaurantItem.tsx and it is under components folder only.

Now, let's check if the newly registered restaurant, The Taj Hotel, is visible in the User Dashboard or not.


MealMingle
Mumbai
Search for Restaurant
usergenspark@gmail.com
My Orders
Rating: 4.0+
Offers
Apply Restaurant Filters
Uniting You with Delicious Moments
Pizza
Burger
Noodles
Pasta
Sandwich
Thali
Dessert
Best Food in Mumbai
15% OFF ABOVE ₹5000
The Taj Hotel


Rating: 4.0+
Offers
Apply Restaurant Filters
Uniting You with Delicious Moments
Pizza
Burger
Noodles
Pasta
Sandwich
Thali
Dessert
Best Food in Mumbai
15% OFF ABOVE ₹5000
The Taj Hotel
1, Apollo Bunder Road, Colaba, Mumbai
Open



App.tsx

This `App` component in React sets up the main structure of the Meal Mingle application, handling both user and admin functionalities.

1. Context & State Management:

- **`CartProvider`**: This wraps the entire application, providing cart-related state and functions across all components using React's Context API.
- **`useState` and `useEffect`**:
  - `useState` manages the state for the list of restaurants.
  - `useEffect` fetches restaurant data from `localStorage` when the app loads, and sets it in the state.

2. Routes:

The Routes component from react-router-dom is used to define the navigation structure of the app. It maps specific paths (URLs) to corresponding components:

User-Side Routes:

1. **`/`** (`Home`): The homepage of the application.
2. **`/signup`** (`Signup`): The user registration page.
3. **`/login`** (`Login`): The user login page.
4. **`/emailLogin`** (`EmailLogin`): Alternative login using email.
5. **`/main`** (`Main`): The main dashboard for users after logging in.
6. **`/menu`** (`Menu`): Displays the menu for a specific restaurant.
7. **`/cart`** (`Cart`): Shows the user's shopping cart.
8. **`/payment`** (`Payment`): Handles the payment process.
9. **`/payment-success`** (`PaymentSuccess`): Confirmation page after successful payment.
10. **`/category/:categoryName`** (`RestaurantsByCategory`): Shows restaurants filtered by a specific category.
11. **`/category/:categoryName/menu`** (`RestaurantsByCategory Menu`): Displays the menu for a restaurant within a specific category.
12. **`/restaurants/filter`** (`RestaurantsByFilter`): Shows restaurants based on user-applied filters.
13. **`/order-history`** (`MyOrders`): Displays the user's order history.

Admin-Side Routes:

1. **`/partner-with-us`** (`PartnerWithUs`): A page for restaurant owners to partner with the platform.
2. **`/admin/signup`** (`AdminSignup`): Admin registration page.
3. **`/admin/login`** (`AdminLogin`): Admin login page.
4. **`/admin/emailLogin`** (`AdminEmailLogin`): Admin login using email.
5. **`/register-restaurant`** (`RegisterRestaurant`): Allows admins to register a new restaurant.
6. **`/view-admin-restaurants`** (`ViewAdminRestaurants`): Admin view to see and manage all registered restaurants. It also handles deletion of restaurants via the `handleDelete` function.
7. **`/update-restaurant/:id`** (`UpdateRestaurant`): Admin page to update details of a specific restaurant.
8. **`/admin/enter-bank-details/:restaurantId`** (`RestaurantBankDetails`): Admin page to enter or update bank details for a specific restaurant.
9. **`/register-restaurant-item/:restaurantId`** (`RegisterRestaurantItem`): Admin page to register new items on a restaurant's menu.

10. **`/view-admin-restaurant-items/:restaurantId`** (`ViewAdminRestaurantItems`): Admin view to manage all items of a specific restaurant's menu.
11. **`/update-restaurant-item/:restaurantId/:restaurantItemId`** (`UpdateRestaurantItem`): Admin page to update details of a specific menu item.
12. **`/view-admin-bank-details`** (`ViewAdminBankDetails`): Admin view to see the bank details associated with restaurants.
13. **`/update-admin-bank-details`** (`UpdateAdminBankDetails`): Admin page to update bank details.

```
import React, { useEffect, useState } from 'react';

import { Routes, Route, Navigate } from 'react-router-dom';

import { CartProvider } from './context/CartContext';

import Login from './components/Login';

import Signup from './components/Signup';

import Home from './components/Home';

import EmailLogin from './components/EmailLogin';

import Main from './components/Main';

import Menu from './components/Menu';

import Payment from './components/Payment';

import PaymentSuccess from './components/PaymentSuccess';

import Cart from './components/Cart';

import RestaurantsByCategory from './components/RestaurantsByCategory';

import RestaurantsByCategoryMenu from

'./components/RestaurantsByCategoryMenu';

import RestaurantsByFilter from './components/RestaurantsByFilter';

import MyOrders from './components/MyOrders';

import PartnerWithUs from './components/PartnerWithUs';

import AdminSignup from './components/AdminSignup';
```

```
import AdminLogin from './components/AdminLogin';
import AdminEmailLogin from './components/AdminEmailLogin';
import RegisterRestaurant from './components/RegisterRestaurant';
import ViewAdminRestaurants from './components/ViewAdminRestaurants';
import UpdateRestaurant from './components/UpdateRestaurant';
import RestaurantBankDetails from './components/RestaurantBankDetails';
import RegisterRestaurantItem from './components/RegisterRestaurantItem';
import ViewAdminRestaurantItems from './components/ViewAdminRestaurantItems';
import UpdateRestaurantItem from './components/UpdateRestaurantItem';
import ViewAdminBankDetails from './components/ViewAdminBankDetails';
import UpdateAdminBankDetails from './components/UpdateAdminBankDetails';


const App = () => {
  const [restaurants, setRestaurants] = useState<any[]>([]);


  useEffect(() => {
    const storedRestaurants = JSON.parse(localStorage.getItem('restaurants')
|| '[]');
    setRestaurants(storedRestaurants);
  }, []);


  const handleDelete = (id: string) => {
    const updatedRestaurants = restaurants.filter(r => r.restaurantId !==
id);
    setRestaurants(updatedRestaurants);
```

```
    localStorage.setItem('restaurants', JSON.stringify(updatedRestaurants));
  };

  return (
    <CartProvider>
      <Routes>
        <Route path='/' element={<Home />} />
        <Route path='/signup' element={<Signup />} />
        <Route path='/login' element={<Login />} />
        <Route path='/emailLogin' element={<EmailLogin />} />
        <Route path='/main' element={<Main />} />
        <Route path='/menu' element={<Menu />} />
        <Route path='/cart' element={<Cart />} />
        <Route path='/payment' element={<Payment />} />
        <Route path="/payment-success" element={<PaymentSuccess />} />
        <Route path="/category/:categoryName" element={<RestaurantsByCategory
/>} />
        <Route path="/category/:categoryName/menu"
element={<RestaurantsByCategoryMenu />} />
        <Route path="/restaurants/filter" element={<RestaurantsByFilter />}
/>
        <Route path="/order-history" element={<MyOrders />} />
        <Route path="/partner-with-us" element={<PartnerWithUs />} />
        <Route path='/admin/signup' element={<AdminSignup />} />
        <Route path='/admin/login' element={<AdminLogin />} />
```

```
        <Route path='/admin/emailLogin' element={<AdminEmailLogin />} />
        <Route path="/register-restaurant" element={<RegisterRestaurant />}
/>
        <Route path="/view-admin-restaurants" element={<ViewAdminRestaurants
restaurants={restaurants} onDelete={handleDelete} />} />
        <Route path="/update-restaurant/:id" element={<UpdateRestaurant />}
/>
        <Route path="/admin/enter-bank-details/:restaurantId"
element={<RestaurantBankDetails />} />
        <Route path="/register-restaurant-item/:restaurantId"
element={<RegisterRestaurantItem />} />
        <Route path="/view-admin-restaurant-items/:restaurantId"
element={<ViewAdminRestaurantItems />} />
        <Route path="/update-restaurant-item/:restaurantId/:restaurantItemId"
element={<UpdateRestaurantItem />} />
        <Route path="/view-admin-bank-details" element={<ViewAdminBankDetails
/>} />
        <Route path="/update-admin-bank-details"
element={<UpdateAdminBankDetails />} />
      </Routes>
    </CartProvider>
  );
};

export default App;
```