KOLEJ PROFESIONAL MARA BERANANG

DIPLOMA IN COMPUTER SCIENCE

COURSE NAME	:	**OBJECT ORIENTED PROGRAMMING**
COURSE CODE	:	**CSC2744**
ACADEMIC SESSION	:	**SESSION 1 2023/2024**
TYPE OF ASSESSMENT	:	**FINAL ASSIGNMENT**
DURATION	:	**20/6/2023-10/07/2023**

CLO 3: Employ third party data in object oriented application development using graphical user interface (GUI) application framework

INSTRUCTION TO CANDIDATES:

1. Late submissions after given due date will not be accepted.
2. Report should be written using:
   Font type: Arial
   Size: 12 pts
   Line Spacing: 1.5
3. Coding format:
   Font type: Consolas
   Size: 10 pts
   Line Spacing: Single

Personal Details	
Name	Dyana binti Azizol
I/D Number	BCS2207-071
Class	DCS 4A
Lecturer	Puan Nini Aniza

Section / Question No.	**Marks**
Total	**/ 50**

Question

Electron is a powerful framework that enables developers to create cross-platform applications using web technologies such as HTML, CSS, and JavaScript. You need to choose one of the applications below to develop a desktop application using Electron that integrates the given API. The application needs to be developed with specific requirements or functionalities.

Name of Application	Description	Requirements
Dictionary and Thesaurus	An app that lists words in groups of synonyms and related concept.	• Word search. • Information Output: give the meaning of the searched word for different part of speech (noun/adjective), antonyms and the example of word usage, sounds and related URL for the searched word. • CRUD words of the day https://api.dictionaryapi.dev/api/v2/entries/en/digital
Meal planner	An app that displays suggestion of recipe based on food item entered by user	• Suggest recipe based on food item. • Information Output: One recipe suggestion that comprises of ingredients, instruction on how to cook and URL of the related site for the food and the link on how to prepare the food. • CRUD meal planner https://www.themealdb.com/api/json/v1/1/search.php?s=shawarma
Makeup Box	An app that finds makeup products based on brand and category entered.	• Display the product info according to search criteria. • Information Output: product description based on brand and name, product image, product website and related link for the searched product. • CRUD makeup top 5 list http://makeup-api.herokuapp.com/api/v1/products.json?brand=maybelline

Tasks:

1. Create desktop app using electron and apply third party data fetched from API and the requirements given. Your application should have at least 2 pages and you may add extra functionality or features of your choice to the application.
2. Implement CRUD (create, read, update, delete) process to the application as given in the requirements.
3. Apply HTML and CSS for user interface and provide evidence for application.
4. GUI Elements:
    i. Apply GUI elements that assist users in using application.
    ii. The application's 'look and feel' is attractive and informative.
5. Produce a report on your application functionalities and features. Include the following:
    i. Overview of your application with a brief description.
    ii. Screenshots of the application with explanations on how to use it.
    iii. Program codes of your system
6. Submit files in GitHub.

Assessment Rubric

ATTRIBUTES	CRITERIA	POOR (1 mark)	FAIR (2 marks)	GOOD (3 marks)	VERY GOOD (4 marks)	EXCELLENT (5 marks)	Mark Obtained
Reproduce and Process Information	1. Create desktop app using electron and apply third party data fetched from API and the requirements given. You may add extra functionality or features of your choice to the application.	The application is an extensive collection and rehash of other people's ideas, products, and images. There is no evidence of new thought.	The application is somewhat a collection and rehash of other people's ideas, products, and images. There is little evidence of new thought or inventiveness.	The application is a minimal collection or rehash of other people's ideas, products, and images. There is a few evidence of new thought or inventiveness.	The application shows a lot of evidence of originality and inventiveness.	The application shows significant evidence of originality and inventiveness. Most of the content and many of the ideas are fresh,original, and inventive.	
		Able to display part of the data from the API and does not fulfill the requirements.	Able to display sufficient data from the API that meet with the requirements together with the description.	Able to display sufficient data from the API that meet with the requirements together with the description.	Able to display extra data from the API beyond the application requirements together with no description.	Able to display extra data from the API beyond the application requirements together with the description.	
		The data from the API does not reflect the whole purpose of the application developed.	The data from the API is sufficient but does not reflect the whole purpose of the application developed.	The data from the API is meaningful but does not reflect the purpose of the application developed.	The data from the API is meaningful and reflect the purpose of the application developed.	The data from the API is meaningful to come up with extra idea for the application developed.	
		The developed application does not fulfill the requirements stated for the chosen	The developed application fulfills all the requirements stated for the chosen application with	The developed application fulfills all the requirements stated for the chosen application.	The developed application fulfills all the requirements stated for the chosen application.	The developed application fulfills all the requirements stated for the chosen application that	

		application.	no extra functionalities.	Add extra functionality or features to the application.	Add extra functionality or features to the application that enhances the user experience or adds value to the application.	utilizes the data from the API Add extra functionality or features to the application that enhances the user experience or adds value to the application.	
	2. Implement CRUD (create, read, update, delete) process to the application as given in the requirements.	• Able to create only 2 of the CRUD processes according to the requirements. • No feedback for the CRUD process. • The design for data input is poor	• Able to create only 3 of the CRUD processes according to the requirements. • No feedback for the CRUD process. • The design for data input is good with some room for improvement	• Able to perform all the CRUD processes according to the requirements. • No feedback for the CRUD process. • Well-designed data input for CRUD process.	• Able to perform all the CRUD processes according to the requirements. • No feedback for the CRUD process. • Well-designed data input for CRUD process.	• Able to perform all the CRUD processes according to the requirements. • Appropriate feedback for the CRUD process. • Well-designed and user-friendly data input for CRUD process.	
	3. Apply HTML and CSS for user interface and provide evidence for application.	• **Text -** All text used is too small to view or the font type is wrongly chosen. • **Graphics** - Graphics seem randomly chosen, are of low quality, OR distract the reader.	• **Text** – Some of the text used is too small to view or the font type is wrongly chosen. • **Graphics -** Graphics seem randomly chosen, are	• **Text** - Most text used is clear but does not describe the content well. • **Graphics** - Graphics are related to the theme/purpose of the application	• **Text** - All text used is clear but does not describe the content well. • **Graphics -** Graphics are related to the theme/purpose of the	• **Text** - All text used is clear and able to describe the content well. • **Graphics** - Graphics are related to the theme/purpose of the application, are thoughtfully	

			of low quality, OR distract the reader.	and are of excellent quality.	application, are of excellent quality and enhance reader interest or understanding	cropped, are of high quality and enhance reader interest or understanding.	
Curate	4.GUI Elements: i. Apply GUI elements that assist users in using application. i.	Not able to curate for required content. • **Layout** - The HTML elements in the application are cluttered looking or confusing. • **Navigation** Links do not take the reader to the sites/ pages described. User typically feels lost.	Limited curation for required content. • **Layout** - The HTML elements in the application is messy, may appear busy or boring. • **Navigation** Links seem to be missing and don't allow the user to easily navigate.	Satisfactory curation for required content. • **Layout** - The HTML elements are suitable. • **Navigation** Links allow the reader to move from page to page, but some links seem to be missing.	Good curation for required content. • **Layout** - The HTML elements are suitable and usable. • **Navigation** Links are labelled and allow the user to easily move from page to page.	Excellent curation for required content. • **Layout** - The HTML elements are well structured, attractive, and usable layout. • **Navigation** Links are clearly labelled, consistently placed, and allow the user to easily move from page to page.	
	ii. The application's 'look and feel' is attractive and informative.	• The application is in need of polish in its visual design and is not appropriate for the target audience. • **Color**	• The application is in need of polish in its visual design, but it is still appropriate for the target audience. • **Color**	• The application mostly follows good visual design principles (e.g.: alignment, contrast,	• The application demonstrates good visual design principles (e.g.: alignment, contrast,	• The application clearly demonstrates good visual design principles (e.g.: alignment,	

		Choice of colors and combinations are not suitable.	Choice of colors and combinations do not match the concept of the application.	easily read text) and is appropriate for the target audience. • **Color** Choice of colors and combinations match the concept of the application.	easily read text) and is appropriate for the target audience. • **Color** Appropriate colors used to produce an atmosphere that expresses the concept of the application.	contrast, easily read text) and is appropriate for the target audience. • **Color** Appropriate colors used to produce an atmosphere that expresses the concept of the application.	
Convey	5. Produce a report on your application functionalities and features that includes: i. Overview of the application. ii. Screenshots of the application with explanations on how to use it.	The overview of the application is vague. The user guide is incomplete and cannot be recognized as a user guide.	The overview of the application is very brief and does not describe the whole functionalities of the application. The user guide provides limited information with no screenshots of the application.	The overview of the application is clearly described the whole application and its functionalities. The user guide provides basic information with limited screenshots of the application.	The overview of the application is clearly described the whole application and its functionalities. The user guide provides adequate information with complete screenshots of the application.	The overview of the application is clearly described the whole application and its functionalities. The user guide provides extensive information with complete screenshots and labelling of the application.	
	iii. Program codes of the system	HTML, CSS and JavaScript codes attached are not complete. The codes are hardly read.	HTML, CSS and JavaScript codes attached are complete. The codes are hardly read.	HTML, CSS and JavaScript codes attached are complete. The codes are readable but not organized.	HTML, CSS and JavaScript codes attached are complete. The codes are readable and	HTML, CSS and JavaScript codes attached are complete and include comments for the important parts of the codes.	

		Does not submit complete electron files in GitHub	Completely submit all the electron file in GitHub.	Completely submit all the electron file in GitHub.	organized. Completely submit all the electron file in GitHub.	The codes are readable and organized. Completely submit all the electron file in GitHub.	
Total Marks Earned							**/50**
Total Percentage (40%)							**/40%**

Overview of the application

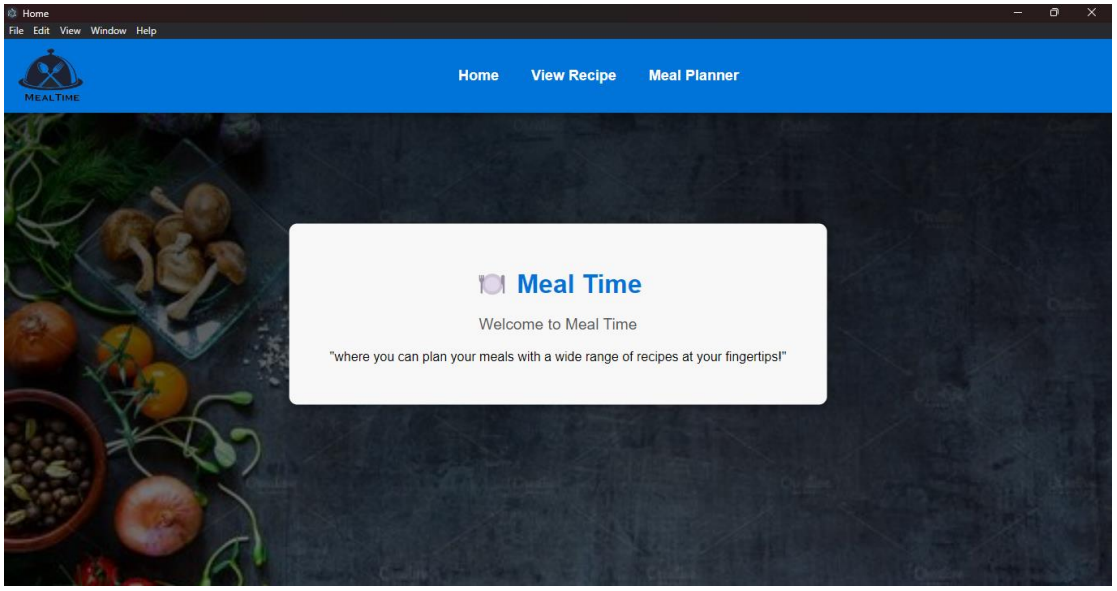
“Meal Time” is a user-friendly system that simplifies the process of meal planning and recipe discovery. This versatile platform offers two main features, first, it allows users to explore an extensive range of recipes categorized by their preferences, such as “dessert” or “Starter” Upon selecting a category, the system presents a list of suggested recipes in boxes, each accompanied by detailed information fetched from the API, including a list of ingredients, cooking instructions, a link to a helpful YouTube video, and the source of the recipe. Second, “Meal Time” empowers users to effortlessly plan their meals. With this system, users can create, read, and update their meal plans, providing an organized and flexible approach to managing their dietary choices. In essence, “Meal Time” serves as a user-friendly tool for both discovering new recipes and streamlining the meal planning process, catering to the diverse needs of its users.

Screenshots of the application with explanations on how to use it.

(User guide)

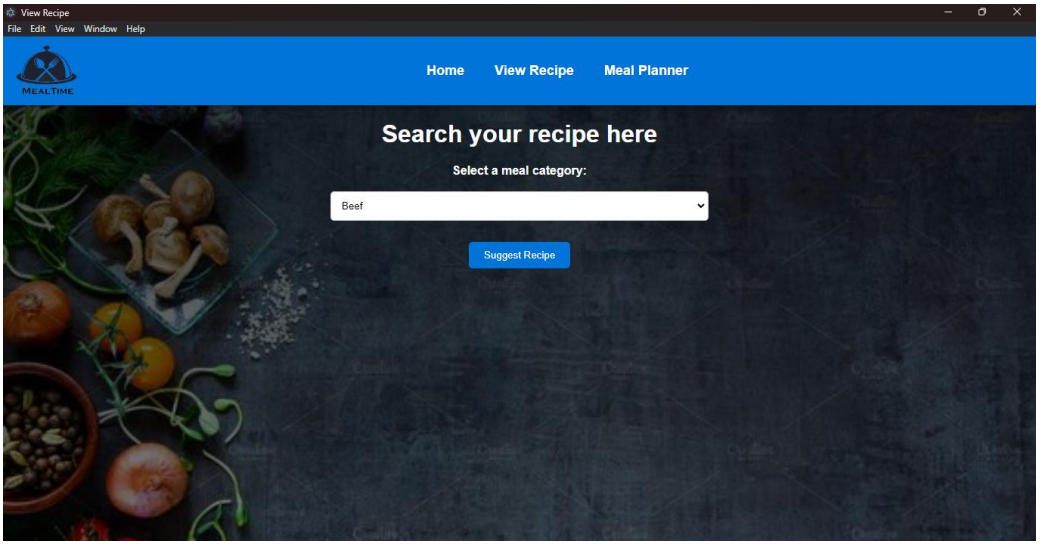
File name: index.html

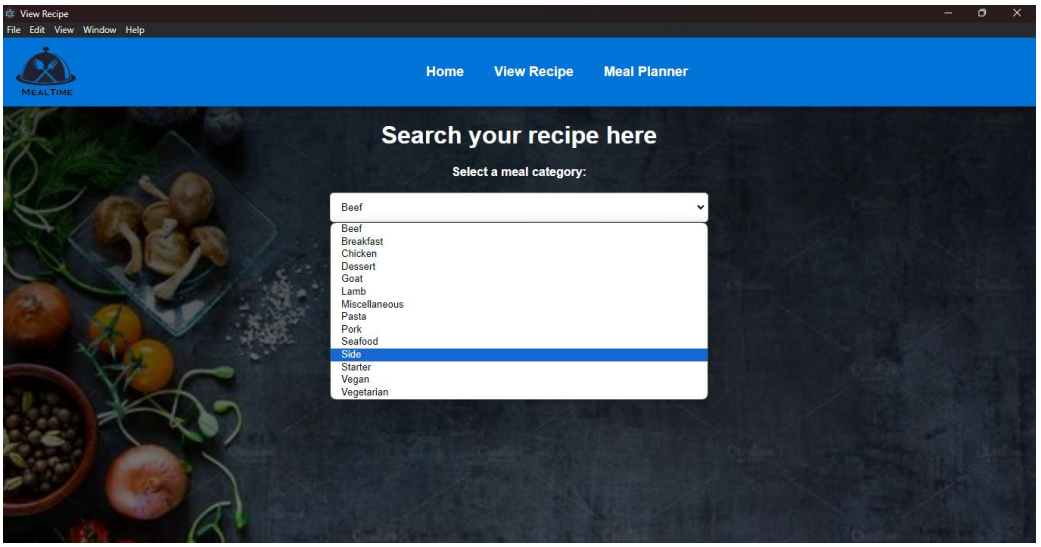
This is the ‘index.html’ file, known as the ‘Home’ page in my system. This page is the first thing a user will see when they view my application. On this page, there is a navigation menu with options such as ‘Home’, ‘View Recipe’, and ‘Meal Planner’. When a user clicks ‘Home’ in the navigation menu, it will display this page. Clicking ‘View Recipe’ will lead the user to the view recipe page, and clicking ‘Meal Planner’ will take them to the meal planner page.



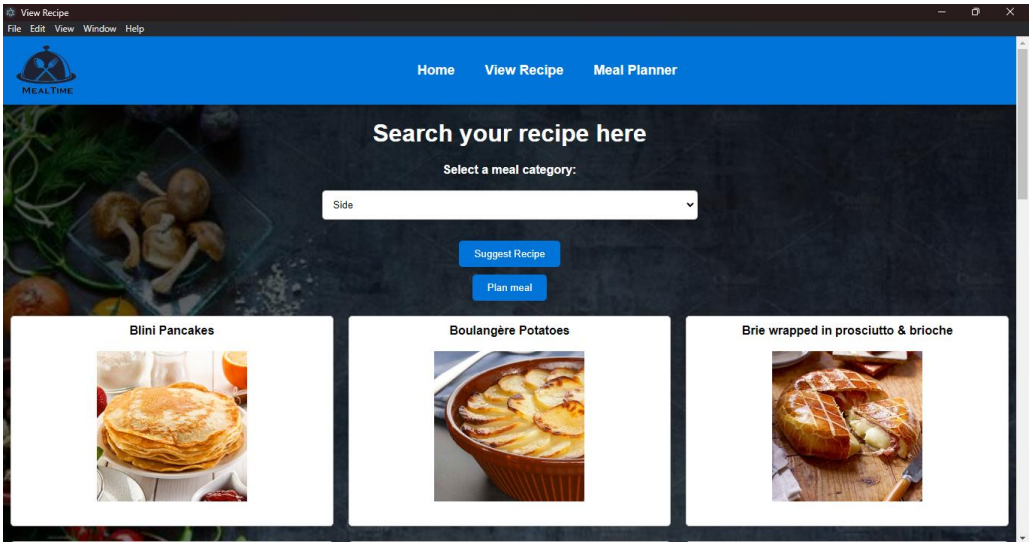
File name: viewrecipe.html

This is the ‘viewrecipe.html’ file, known as the ‘View Recipe’ page in my system. On this page, users are provided with search fields to allow them to search for any category item and view suggested recipes based on their search. When users click the search fields, a dropdown appears containing a list of category items. After selecting a category and clicking the ‘Suggest Recipe’ button, the system will display all the suggested recipes. Users can click on the image of any suggested recipe to access details about the recipe, including ingredients, instructions, the source, and a YouTube URL for the recipe. Additionally, after clicking the ‘Suggest Recipe’ button, a ‘Plan Meal’ button will appear, enabling users to navigate to the ‘Plan Meal’ page (‘plan.html’). The purpose of the ‘Plan Meal’ button is to facilitate meal planning for users after they have viewed the recipes.

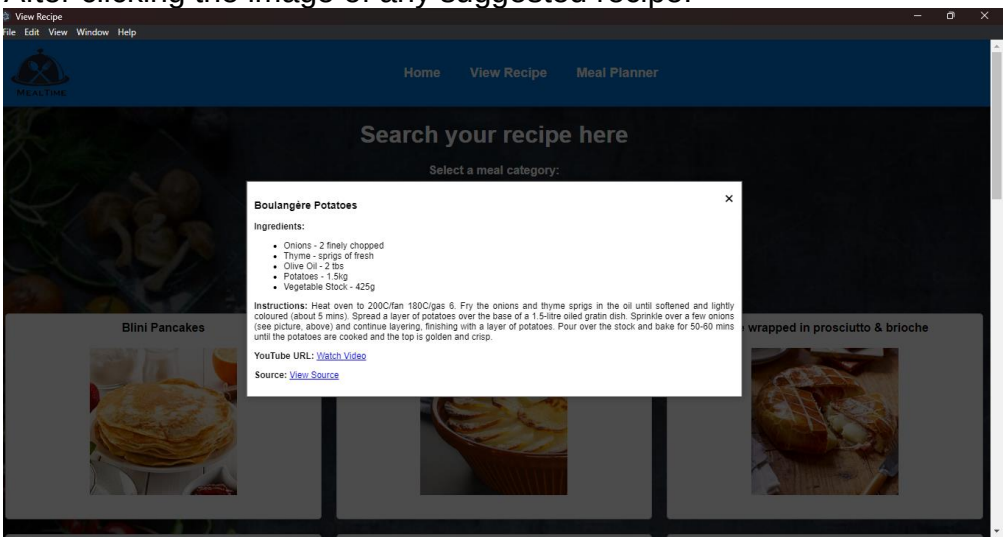




After clicking the ‘Suggest Recipe’ button:

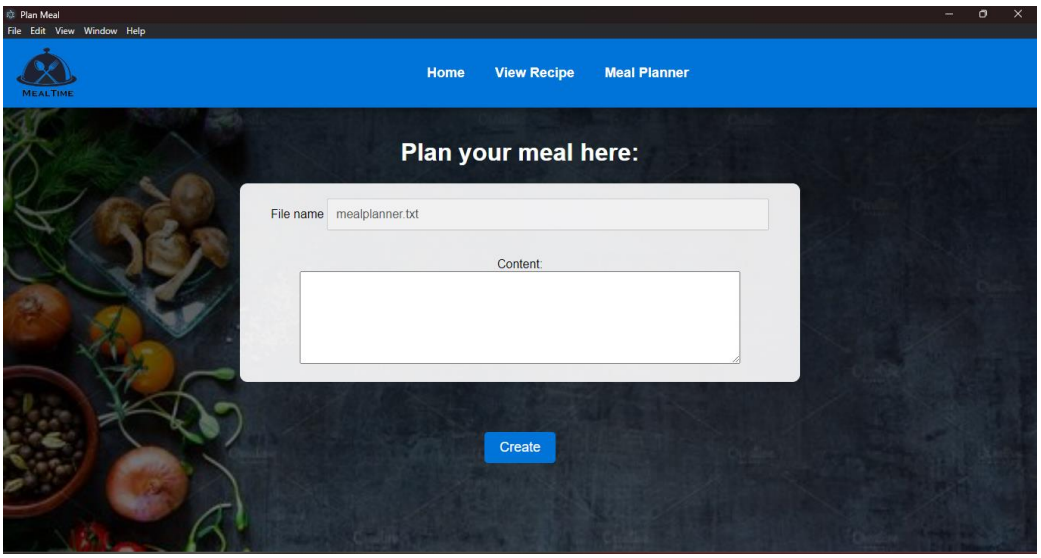


After clicking the image of any suggested recipe:

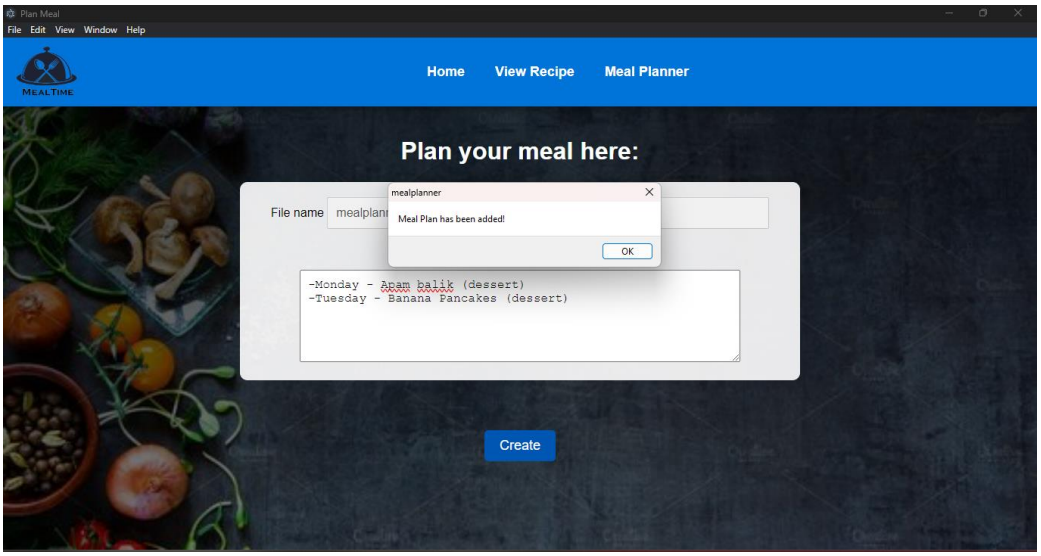


File name: plan.html

This is the ‘plan.html’ file, known as the ‘Plan Meal’ page in my system. On this page, users are provided with a form that allows them to enter their planned meals in the content fields. In the form, the file name (‘mealplanner.txt’) is already fixed, so all the content entered by the user will be stored in a single file. After the user fills out the form, if they click the ‘Create’ button, the system will display an alert, ‘Meal Plan has been added!’ to alert the user that the content has been successfully created and added to the file.

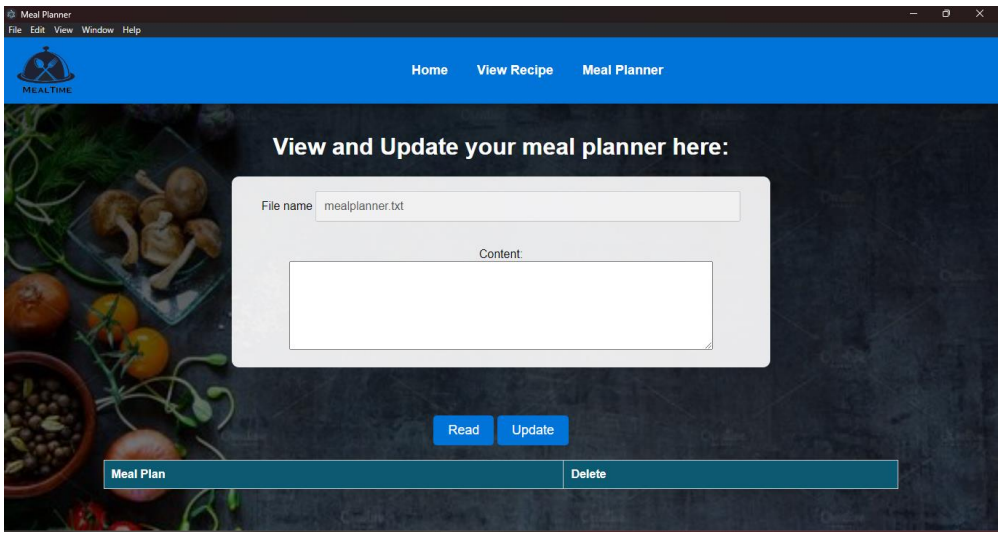


After user entered the content fields then click the “Create” button:

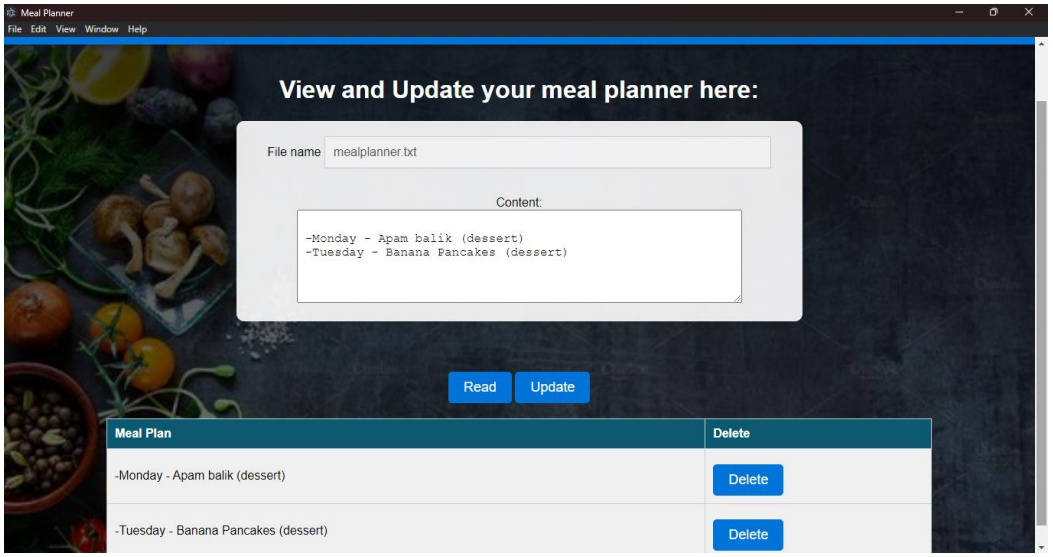


File name: CRUD.html

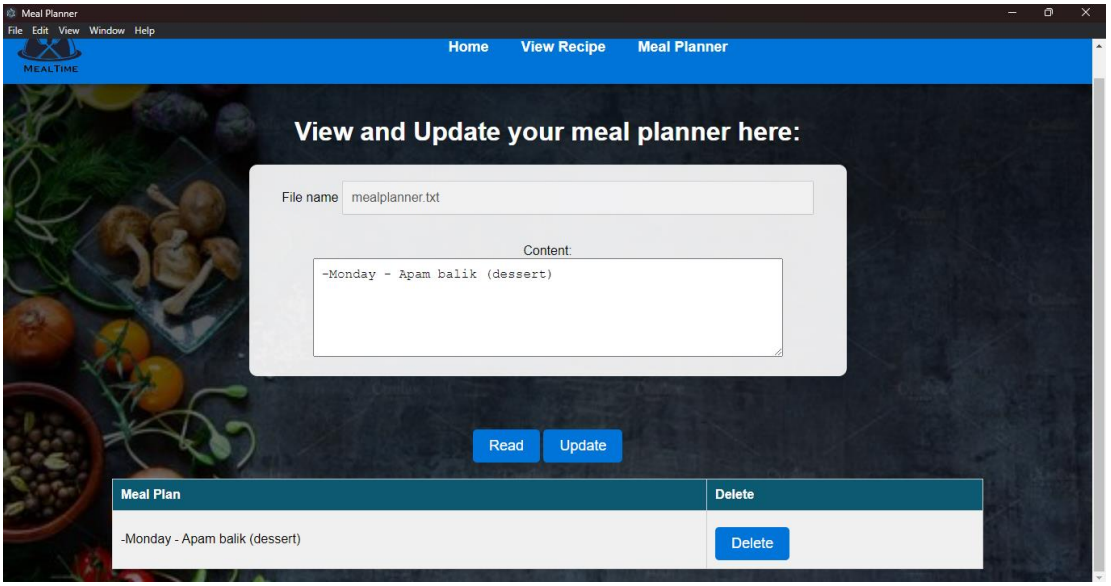
This is the ‘CRUD.html’ file, know as the ‘Meal Planner’ page in my system. On this page, users are provided with a form that allows them to read and update the ‘mealplanner.txt’ file that they have previously created and added on the ‘Plan Meal’ page. The picture below is displayed before the user clicks either the ‘Read’ or ‘Update’ button:



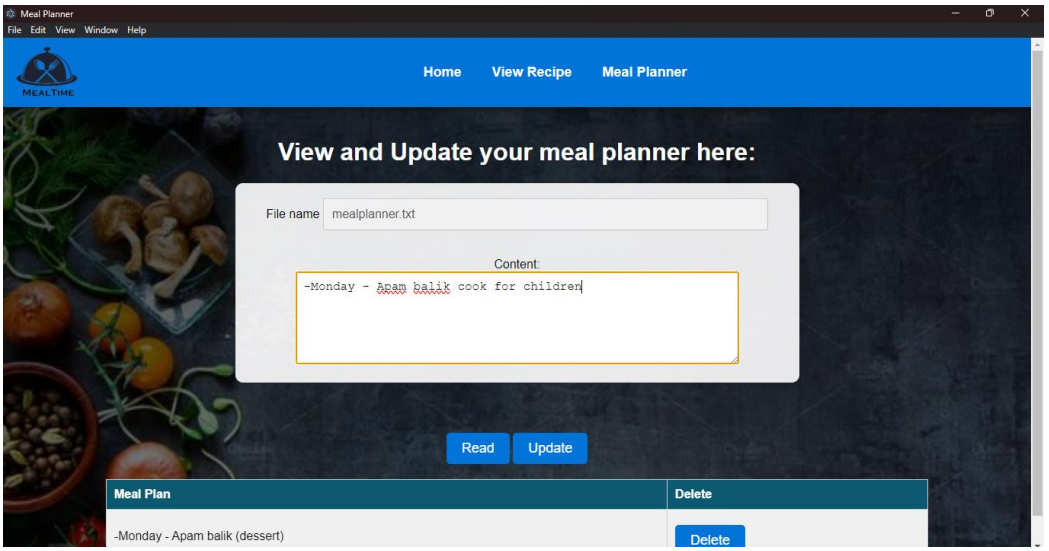
If the user clicks the ‘Read’ button, the content from ‘mealplanner.txt’ will be displayed in the form also in the table below the form, allowing the user to read the content they had previously created and added on the ‘Plan Meal’ page. In the table, user can click the ‘Delete’ button if they want to delete any of the content in the ‘mealplanner.txt’ like shown below:



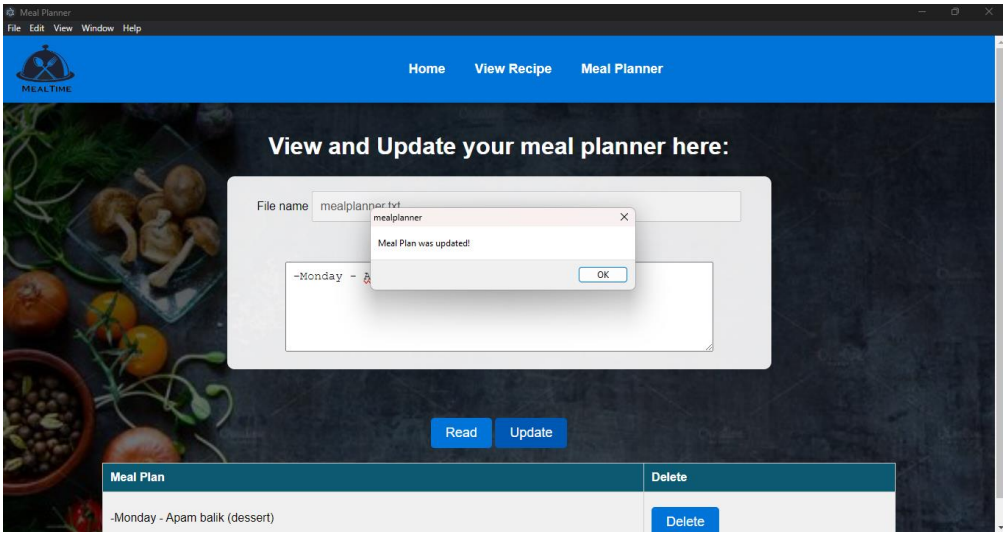
after click the “Delete” button:

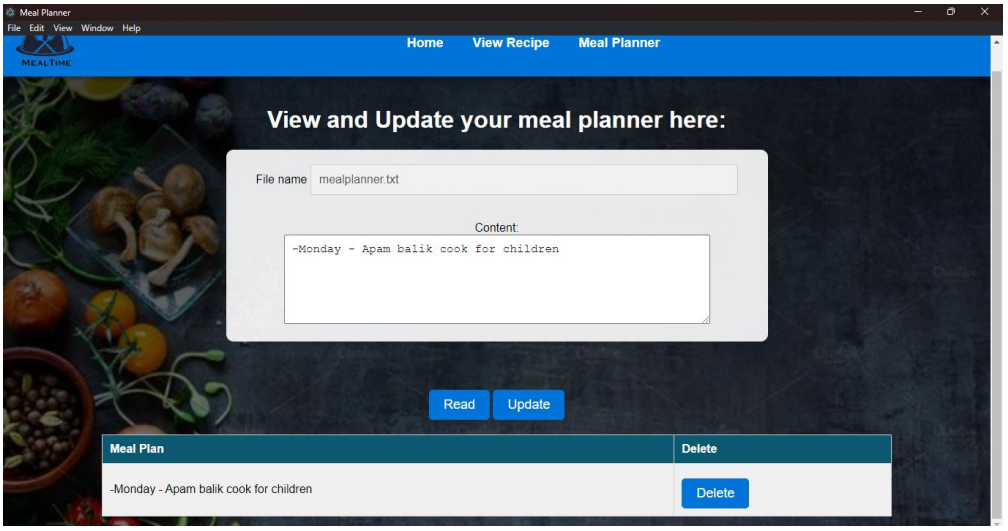


If a user wants to update the ‘mealplanner.txt’ file, they need to click the ‘Read’ button first. After clicking ‘Read’ the system will display the content in the form. Once the content is displayed, the user can make updates within the form and then click the ‘Update’ button. After clicking ‘Update’ the user will receive an alert that says ‘Meal Plan was updated!’ to inform the user that the file has been successfully updated. Here’s an example of updated content in the form:



After clicking the “Update” button:





Program codes of the system

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Home</title> <!-- Set the title of the web page -->
    <link rel="stylesheet" href="index.css" />
  </head>
  <body>
    <div class="header">
      <div class="left-content">
        <img src="logo1.png" width="90px" > <!-- Logo and header section -->
      </div>
      <div class="center-content">
        <nav>
          <ul class="navbar">
            <li><a href="index.html">Home</a></li> <!-- Navigation links to Home,
View Recipe, and Meal Planner -->
            <li><a href="viewrecipe.html">View Recipe</a></li>
            <li><a href="CRUD.html">Meal Planner</a></li>
          </ul>
        </nav>
      </div>
    </div>
    <div class="container">
      <h1>🍽 Meal Time</h1> <!-- Main heading -->
      <p class="p1">Welcome to Meal Time</p> <!-- Welcome message -->
      <p>"where you can plan your meals with a wide range of recipes at your
fingertips!"</p> <!-- Description -->
    </div>
  </body>
</html>
```

index.css

```
body, h1, p, a {
  margin: 0;
  padding: 0;
}

/* background color to the entire page with a semi-transparent overlay */
body {
  background-image: linear-gradient(rgba(0, 0, 0, 0.75), rgba(0, 0, 0, 0.5)),
url("meal.jpg");
  background-size: cover;
  background-repeat: no-repeat;
  background-attachment: fixed;
  font-family: Arial, sans-serif;
}

/* Header styles */
.header {
  display: flex;
  justify-content: space between;
  align-items: center;
  background-color: #0074d9; /* Light blue background color for the header */
  text-align: center;
  padding: 10px 0;
  box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.24);
}

/* Styles for the left content within the header */
.left-content {
  margin-left: 10px;
}

/* Styles for the center content within the header */
.center-content {
  flex-grow: 1;
}
```

```
/* Navigation bar styles */
.navbar {
  list-style: none;
  padding: 0;
  display: flex;
  justify-content: center;
  background-color: transparent;
}

/* Style list items within the navigation bar */
.navbar li {
  margin: 0 20px;
}

/* Style hyperlinks within the navigation bar */
.navbar a {
  text-decoration: none;
  color: white;
  font-weight: bold;
  font-size: 18px;
  transition: color 0.3s;
  position: relative;
}

/* sliding underline effect for hyperlinks */
.navbar a:before {
  content: '';
  position: absolute;
  width: 100%;
  height: 2px;
  bottom: 0;
  left: 0;
  background-color: white;
  visibility: hidden;
  transform: scaleX(0);
  transition: all 0.3s ease-in-out;
}

/* sliding underline effect on hyperlink hover */
.navbar a:hover:before {
```

```
  visibility: visible;
  transform: scaleX(1);
}

/* container in the middle of the page */
.container {
  text-align: center;
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  background-color: #f8f8f8;
  padding: 50px;
  border-radius: 10px;
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.349);
}

/* Style the main heading */
h1 {
  font-size: 2em;
  margin-bottom: 20px;
  color: #0074d9;
}

/* Style the welcome message */
.p1 {
  font-size: 1.2em;
  color: #555;
  margin-bottom: 20px;
}
```

index.js

```
const { app, BrowserWindow } = require('electron');
const fs = require('fs')
const path = require('path')

// Handle creating/removing shortcuts on Windows when installing/uninstalling.
if (require('electron-squirrel-startup')) {
  // eslint-disable-line global-require
  app.quit();
```

```
}

const createWindow = () => {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      nodeIntegration: true,
      contextIsolation: false,
    }

  });

  // and load the index.html of the app.
  mainWindow.loadFile(path.join(__dirname, 'index.html'));

  // Open the DevTools.
  //mainWindow.webContents.openDevTools();
};

// This method will be called when Electron has finished
// initialization and is ready to create browser windows.
// Some APIs can only be used after this event occurs.
app.on('ready', createWindow);

// Quit when all windows are closed, except on macOS. There, it's common
// for applications and their menu bar to stay active until the user quits
// explicitly with Cmd + Q.
app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') {
    app.quit();
  }
});

app.on('activate', () => {
  // On OS X it's common to re-create a window in the app when the
  // dock icon is clicked and there are no other windows open.
  if (BrowserWindow.getAllWindows().length === 0) {
    createWindow();
```

```
  }
});
```

viewrecipe.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>View Recipe</title>
    <link rel="stylesheet" href="viewrecipe.css" />
  </head>
  <body>
    <div class="header">
      <div class="left-content">
        <img src="logo1.png" width="90px" > <!-- Logo image in the header -->
      </div>
      <div class="center-content">
        <nav>
          <ul class="navbar"> <!-- Navigation bar links -->
            <li><a href="index.html">Home</a></li>
            <li><a href="viewrecipe.html">View Recipe</a></li>
            <li><a href="CRUD.html">Meal Planner</a></li>
          </ul>
        </nav>
      </div>
    </div>
    <h1> Search your recipe here</h1> <!-- Page title -->
    <center>
      <label for="searchData">Select a meal category:</label> <!-- Dropdown label --
>
      <br>
      <select id="searchData"> <!-- Dropdown select for meal category -->
      </select>
    </center>
    <br>
    <button onclick="buttonClicked()">Suggest Recipe</button> <!-- Button to suggest
a recipe -->
    <button id="planMealButton" onclick="navigateToCRUDPage()" style="display:
none;">Plan meal</button> <!-- Button to plan a meal-->
    <div id="suggestRecipe"></div> <!-- Display suggested recipes-->
    <div id="planMealContainer"></div> <!-- Container for plan meals -->
```

```
    <!-- the modal HTML structure for displaying meal details -->
    <div id="mealModal" class="modal">
      <div class="modal-content">
        <span id="mealModalClose" class="close">&times;</span>
        <div id="mealModalContent"></div>
      </div>
    </div>
  </body>
  <script src="fetchapi.js"></script>
</html>
```

viewrecipe.css

```
body {
  margin: 0;
  padding: 0;
  background-image: linear-gradient(rgba(0, 0, 0, 0.75), rgba(0, 0, 0, 0.5)),
url("meal.jpg"); /* Background image with overlay */
  background-size: cover;
  background-repeat: no-repeat;
  background-attachment: fixed;
  font-family: Arial, sans-serif;
}

/* Header styles for the page title */
h1 {
  text-align: center;
  margin-top: 20px;
  color: white;
}

/* Paragraph styles */
p {
  text-align: justify;
  color: black;
}

.p1 {
  text-align: center;
```

```
  color: black;
}

/* Dropdown label styles */
label {
  display: block;
  margin-top: 10px;
  font-weight: bold;
  color: white;
}

/* Dropdown select styles */
select {
  width: 500px;
  padding: 10px;
  margin: 0 auto;
  border: 1px solid #ccc;
  border-radius: 5px;
}

/* Button styles */
button {
  display: block;
  margin: 10px auto;
  padding: 10px 20px;
  background-color: #0074d9;
  color: #fff;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  transition: background-color 0.3s;
}

button:hover {
  background-color: #0056b3;
}

/* Header styles for planned meals */
h2 {
  text-align: center;
```

```
  margin-top: 20px;
  color: white;
}

/* Styles for suggested recipe display */
#suggestRecipe div {
  text-align: center;
  background-color: #fff;
  border: 1px solid #ccc;
  border-radius: 5px;
  padding: 10px;
  margin: 10px;
  width: calc(33.33% - 20px);
  display: inline-block;
  box-sizing: border-box;
}

#suggestRecipe {
  text-align: center;
}

/* Header styles */
.header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  background-color: #0074d9;
  text-align: center;
  padding: 10px 0;
  box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.24);
}

.left-content {
  margin-left: 10px;
}

.center-content {
  flex-grow: 1;
}
```

```
/* Navigation bar styles */
.navbar {
  list-style: none;
  padding: 0;
  display: flex;
  justify-content: center;
  background-color: transparent;
}

.navbar li {
  margin: 0 20px;
}

.navbar a {
  text-decoration: none;
  color: white;
  font-weight: bold;
  font-size: 18px;
  transition: color 0.3s;
  position: relative;
}

.navbar a:before {
  content: '';
  position: absolute;
  width: 100%;
  height: 2px;
  bottom: 0;
  left: 0;
  background-color: white;
  visibility: hidden;
  transform: scaleX(0);
  transition: all 0.3s ease-in-out;
}

.navbar a:hover:before {
  visibility: visible;
  transform: scaleX(1);
}
```

```
/* Modal styles */
.modal {
  display: none;
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: rgba(0, 0, 0, 0.7);
}

.modal-content {
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  background-color: white;
  padding: 10px;
  border: 1px solid #333;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.3);
  max-width: 100%;
  font-size: 12px;
  font-family: Arial;
}

.close {
  position: absolute;
  top: 10px;
  right: 10px;
  font-size: 24px;
  cursor: pointer;
}

/* container settings*/

#planMealContainer {
  margin-top: 20px;
  text-align: left;
}
```

fetchapi.js

```
// Function to fetch meal categories and populate the dropdown
function fetchMealCategories() {
  fetch("https://www.themealdb.com/api/json/v1/1/list.php?c=list")
    .then((response) => response.json())
    .then((data) => {
      const categoryDropdown = document.getElementById("searchData");

      data.meals.forEach((category) => {
        const option = document.createElement("option");
        option.value = category.strCategory;
        option.textContent = category.strCategory;
        categoryDropdown.appendChild(option);
      });
    })
    .catch((error) => {
      console.error("Error fetching meal categories: " + error);
    });
}

// Call the fetchMealCategories function to load meal categories
fetchMealCategories();

// Get references to the modal and its content
const modal = document.getElementById("mealModal");
const modalContent = document.getElementById("mealModalContent");

function openModal(mealName) {
  modalContent.innerHTML = "Loading...";

  // Fetch meal details based on the provided meal name
  fetch(`https://www.themealdb.com/api/json/v1/1/search.php?s=${mealName}`)
    .then((response) => response.json())
    .then((data) => {
      const meal = data.meals[0];
      if (meal) {
        // Generate HTML for displaying meal details
        const html = `
```

```
          <h3>${meal.strMeal}</h3>
          <p><strong>Ingredients:</strong></p>
          <ul>
            ${Object.keys(meal)
              .filter((key) => key.startsWith("strIngredient") && meal[key])
              .map((key) => `<li>${meal[key]} - ${meal[`strMeasure${key.slice(-
1)}`]}</li>`)
              .join("")}
          </ul>
          <p><strong>Instructions:</strong> ${meal.strInstructions}</p>
          <p><strong>YouTube URL:</strong> <a href="${meal.strYoutube}"
target="_blank">Watch Video</a></p>
          <p><strong>Source:</strong> <a href="${meal.strSource}"
target="_blank">View Source</a></p> <!-- Add this line -->
        `;
        modalContent.innerHTML = html;
      } else {
        modalContent.innerHTML = "Meal details not found.";
      }
    })
    .catch((error) => {
      console.error("Error fetching meal details: " + error);
      modalContent.innerHTML = "Error fetching meal details.";
    });

  modal.style.display = "block";
}


// Function to close the modal
function closeModal() {
  modal.style.display = "none";
}

// Add event listener to close the modal when the close button is clicked
document.getElementById("mealModalClose").addEventListener("click", closeModal);

// Add event listener to open the modal when a meal image is clicked
document.body.addEventListener("click", (event) => {
  if (event.target.matches(".meal-image")) {
```

```
    openModal(event.target.getAttribute("data-meal-name"));
  }
});

// Function to handle button click for fetching and displaying recipes
function buttonClicked() {
  var mealCategory = document.getElementById("searchData").value;

  if (mealCategory) {
    // Fetch and display recipes based on the selected meal category
    fetch(`https://www.themealdb.com/api/json/v1/1/filter.php?c=${mealCategory}`)
      .then((response) => response.json())
      .then((data) => {
        if (data.meals && data.meals.length > 0) {
          displaysuggestRecipe(data.meals);
          displayPlanMealButton(); // Display the "Plan meal" button
        } else {
          document.getElementById("suggestRecipe").innerHTML = "No meals found for
the selected category.";
        }
      })
      .catch((error) => {
        console.error("Error fetching data:", error);
        document.getElementById("suggestRecipe").innerHTML = "Error fetching data.";
      });
  }
}

// Function to display suggested recipes
function displaysuggestRecipe(meals) {
  const suggestRecipe = document.getElementById("suggestRecipe");
  suggestRecipe.innerHTML = '';

  meals.forEach((meal) => {
    const div = document.createElement('div');
    div.innerHTML = `<b>${meal.strMeal}</b><br><br>`;

    if (meal.strMealThumb) {
      const img = document.createElement('img');
      img.src = meal.strMealThumb;
```

```
      img.style.maxWidth = '200px';
      img.classList.add('meal-image');
      img.setAttribute('data-meal-name', meal.strMeal); // Add meal name as a data
attribute
      div.appendChild(img);
    }

    div.innerHTML += '<br><br>';
    suggestRecipe.appendChild(div); // Append to the 'suggestRecipe' div
  });
}

// Function to navigate to the meal planning page
function navigateToCRUDPage() {
  window.location.href = 'plan.html';
}

// Function to display the "Plan meal" button
function displayPlanMealButton() {
  const planMealButton = document.getElementById("planMealButton");
  if (planMealButton) {
    planMealButton.style.display = "block";
  }
}
```

plan.html

```
<!DOCTYPE html>
<html>

<head>
    <title>Plan Meal</title> <!-- Set the title of the web page -->
    <link rel="stylesheet" type="text/css" href="CRUD.css">
</head>

<body>
    <div class="header">
        <div class="left-content">
            <img src="logo1.png" width="90px"> <!-- Logo and header section -->
        </div>
        <div class="center-content">
```

```
            <nav>
                <ul class="navbar">
                    <li><a href="index.html">Home</a></li> <!-- Navigation links to
Home, View Recipe, and Meal Planner -->
                    <li><a href="viewrecipe.html">View Recipe</a></li>
                    <li><a href="CRUD.html">Meal Planner</a></li>
                </ul>
            </nav>
        </div>
    </div>

    <main>
        <h1> Plan your meal here:</h1>

        <form>
            <div class="container">
                <div class="form-group">
                    <label>File name</label> <!-- Label for the file name input -->
                    <input id="fileName" type="text" class="form-control"
value="mealplanner.txt" disabled> <!-- Input field for the file name (disabled) -->
                </div>

                <br><br>

                <div class="form-group">
                    <label for="fileContents">Content:</label> <!-- Label for the
content input area -->
                    <br>
                    <textarea id="fileContents" class="form-control"
rows="5"></textarea> <!-- Textarea for entering file content -->
                </div>
            </div>
        </form>

        <br><br>

        <button id="btnCreate" class="btn btn-default">Create</button> <!-- Button
for creating a new text file -->

    </main>
```

```
    <script src="plan.js"></script>
</body>

</html>
```

plan.js

```
const { app, BrowserWindow } = require('electron');
const fs = require('fs');
const path = require('path');

// Get references to HTML elements
var btnAppend = document.getElementById('btnCreate');
var fileName = document.getElementById('fileName');
var fileContents = document.getElementById('fileContents');

// Define the path for file operations
let pathName = path.join(__dirname, 'Files');

// Event listener for the "CREATE" button
btnCreate.addEventListener('click', function () {
  // Get the file path and contents
  let file = path.join(pathName, fileName.value);
  let contents = fileContents.value;

  // Check if the file already exists
  if (fs.existsSync(file)) {
    // If the file exists, append the contents with a newline character
    contents = '\n' + contents;
    fs.appendFile(file, contents, function (err) {
      if (err) {
        return console.log(err);
      }
      var txtfile = document.getElementById('fileName').value;
      alert(' Meal Plan has been added! '); // Display success message
      console.log('Meal Plan has been added!');
    });
  } else {
    alert('File does not exist.'); // Display an error message if the file does not
exist
```

```
  }
});
```

CRUD.html

```
<!DOCTYPE html>
<html>

<head>
    <title>Meal Planner</title> <!-- Set the title of the web page -->
    <link rel="stylesheet" type="text/css" href="CRUD.css">
</head>

<body>
    <div class="header">
        <div class="left-content">
            <img src="logo1.png" width="90px"> <!-- Logo and header section -->
        </div>
        <div class="center-content">
            <nav>
                <ul class="navbar">
                    <li><a href="index.html">Home</a></li> <!-- Navigation links to
Home, View Recipe, and Meal Planner -->
                    <li><a href="viewrecipe.html">View Recipe</a></li>
                    <li><a href="CRUD.html">Meal Planner</a></li>
                </ul>
            </nav>
        </div>
    </div>

    <main>
        <h1> View and Update your meal planner here:</h1>

        <form>
            <div class="container"> <!-- Form section for file management -->
                <div class="form-group">
                    <label>File name</label>
                    <input id="fileName" type="text" class="form-control"
value="mealplanner.txt" disabled> <!-- File name input field -->
                </div>

                <br><br>
```

```
                <div class="form-group">
                    <label for="fileContents">Content:</label>
                    <br>
                    <textarea id="fileContents" class="form-control"
rows="5"></textarea> <!-- Text area for file content -->
                </div>
            </div>
        </form>

        <br><br>

        <button id="btnRead" class="btn btn-default">Read</button> <!-- Read button
-->
        <button id="btnUpdate" class="btn btn-default">Update</button> <!-- Update
button -->

        <table id="fileTable">
            <thead>
                <tr>
                    <th>Meal Plan</th>
                    <th>Delete</th>
                </tr>
            </thead>
            <tbody id="fileTableBody"></tbody> <!-- Table for displaying meal plans
-->
        </table>
    </main>

    <script src="CRUD.js"></script>
</body>

</html>
```

CRUD.css

```
body {
    margin: 0;
    padding: 0;
    background-image: linear-gradient(rgba(0, 0, 0, 0.75), rgba(0, 0, 0, 0.5)),
url("meal.jpg");
    background-size: cover;
    background-repeat: no-repeat;
    background-attachment: fixed;
    font-family: Arial, sans-serif;
}

/* Header styles */
.header {
    display: flex;
    justify-content: space between;
    align-items: center;
    background-color: #0074d9;
    text-align: center;
    padding: 10px 0;
    box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.24);
}

.left-content {
    margin-left: 10px;
}

.center-content {
    flex-grow: 1;
}

/* Navigation bar styles */
.navbar {
    list-style: none;
    padding: 0;
    display: flex;
    justify-content: center;
    background-color: transparent;
```

```
}

.navbar li {
    margin: 0 20px;
}

.navbar a {
    text-decoration: none;
    color: white;
    font-weight: bold;
    font-size: 18px;
    transition: color 0.3s;
    position: relative;
}

/* Navigation bar link underline styles */
.navbar a:before {
    content: '';
    position: absolute;
    width: 100%;
    height: 2px;
    bottom: 0;
    left: 0;
    background-color: white;
    visibility: hidden;
    transform: scaleX(0);
    transition: all 0.3s ease-in-out;
}

.navbar a:hover:before {
    visibility: visible;
    transform: scaleX(1);
}

/* Main content area */
main {
    width: 80%;
    margin: 0 auto;
    padding: 20px;
    text-align: center;
```

```
}

h1 {
    color: rgb(255, 255, 255);
}

/* Form styles */
form {
    text-align: center;
    margin: 20px auto;
}

.container {
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0 0 20px rgba(0, 0, 0, 0.1);
    background-color: rgba(255, 255, 255, 0.9);
    max-width: 700px;
    margin: 0 auto;
}

buttons-container {
    margin-top: 20px;
}

/* Button styles */
button {
    background-color: #0074d9;
    color: #fff;
    border: none;
    border-radius: 5px;
    padding: 10px 20px;
    font-size: 18px;
    margin-top: 10px;
    cursor: pointer;
    transition: background-color 0.3s;
}

button:hover {
    background-color: #0056b3;
```

```
}

input[type="text"] {
    width: 80%;
    padding: 10px;
    font-size: 16px;
}

textarea {
    width: 80%;
    padding: 10px;
    font-size: 16px;
    resize: vertical;
}

/* Table styles */
table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 20px;
}

table th, table td {
    border: 1px solid #ccc;
    padding: 10px;
    text-align: left;
}

table tr {
    margin-bottom: 10px;
}

table th {
    background-color: rgb(11, 90, 114);
    color: #fff;
}

/* Specific styles for a table with id "fileTableBody" */
#fileTableBody {
    background-color: #f0f0f0;
```

```
}

/* Specific styles for an element with id "fileContents" */
#fileContents {
    min-height: 100px;
}
```

CRUD.js

```
// Import required modules
const { app, BrowserWindow } = require('electron');
const fs = require('fs');
const path = require('path');

// Get references to HTML elements
var btnRead = document.getElementById('btnRead');
var btnUpdate = document.getElementById('btnUpdate');
var fileName = document.getElementById('fileName');
var fileContents = document.getElementById('fileContents');

// Define the path for storing files
let pathName = path.join(__dirname, 'Files');

// Event listener for the "Read" button
btnRead.addEventListener('click', function() {
    let file = path.join(pathName, fileName.value);

    // Read the file asynchronously
    fs.readFile(file, 'utf8', function(err, data) {
        if (err) {
            console.error(err);
            return console.log(err);
        }
        fileContents.value = data;

        // Split the file content into lines
        const lines = data.split('\n');

        const tableBody = document.getElementById('fileTableBody');
        tableBody.innerHTML = '';
```

```
        // Iterate through the lines and add them to the table
        lines.forEach(function(line) {
            addContentToTable(line);
        });

        console.log("Meal Plan has been Read!");
    });
});

// Event listener for the "Update" button
btnUpdate.addEventListener('click', function () {
    let file = path.join(pathName, fileName.value);
    let contents = fileContents.value;

    // Write the updated content back to the file
    fs.writeFile(file, contents, function (err) {
        if (err) {
            return console.log(err);
        }
        console.log("Meal Plan was updated!");
        alert("Meal Plan was updated!");
    });
});

// Function to add content to the table
function addContentToTable(content) {
    const newRow = document.createElement('tr');

    const contentCell = document.createElement('td');
    contentCell.textContent = content;

    const deleteCell = document.createElement('td');
    const deleteButton = document.createElement('button');
    deleteButton.textContent = 'Delete';

    // Event listener for the "Delete" button
    deleteButton.addEventListener('click', function() {
        const row = this.parentNode.parentNode;
        const rowIndex = row.rowIndex;
```

```
            row.remove();

            // Read the file, remove the specified line, and write the updated content
            const file = path.join(pathName, fileName.value);
            fs.readFile(file, 'utf8', function(err, data) {
                if (err) {
                    console.error(err);
                    return console.log(err);
                }
                const lines = data.split('\n');
                lines.splice(rowIndex - 1, 1);
                const updatedData = lines.join('\n');
                fs.writeFile(file, updatedData, 'utf8', function(err) {
                    if (err) {
                        console.error(err);
                    }
                });
            });
        });

        deleteCell.appendChild(deleteButton);

        newRow.appendChild(contentCell);
        newRow.appendChild(deleteCell);

        const tableBody = document.getElementById('fileTableBody');
        tableBody.appendChild(newRow);
    }
```

main.js

```
const { app, BrowserWindow } = require('electron');
const fs = require('fs')
const path = require('path')

// Get references to HTML elements
var btnCreate = document.getElementById('btnCreate')
var btnRead = document.getElementById('btnRead')
var btnDelete = document.getElementById('btnDelete')
var fileName = document.getElementById('fileName')
var fileContents = document.getElementById('fileContents')

// Define the path for file operations
let pathName = path.join(__dirname, 'Files')

// Event listener for the "CREATE" button
btnCreate.addEventListener('click', function(){
  // Create a text file when the user clicks the CREATE button
  let file = path.join(pathName, fileName.value)
  let contents = fileContents.value
  fs.writeFile(file, contents, function(err){
    if(err){
      return console.log(err)
    }
    var txtfile = document.getElementById("fileName").value
    alert(txtfile + " text file was created")
    console.log("The file was created")
  })
})

// Event listener for the "READ" button
btnRead.addEventListener('click', function(){
  // Read contents of the created text file
  let file = path.join(pathName, fileName.value)

  fs.readFile(file, function(err, data){
    if(err){
      return console.log(err)
```

```
    }
    fileContents.value = data
    console.log("The file was read!")
  })
})

// Event listener for the "DELETE" button
btnDelete.addEventListener('click', function(){
  // Delete the specified text file
  let file = path.join(pathName, fileName.value)

  fs.unlink(file, function(err){
    if(err){
      return console.log(err)
    }
    fileName.value = ""
    fileContents.value = ""
    console.log("The file was deleted!")
  })
})
```

preload.js

```
// See the Electron documentation for details on how to use preload scripts:
// https://www.electronjs.org/docs/latest/tutorial/process-model#preload-scripts
```

Link : https://github.com/dyanaazizol/mealplanner.git