Technical University of Cluj-Napoca

Meal Planner

Data Transmission Project
Sem. II, Year 2022-2023

Team Members

Apetrei Andrei-Octavian
Bokotey Monika-Imola
Dafinoiu Andreea-Raluca

Group: 30333

Coordinating Teacher Dr. Ing. Avram Camelia Claudia

Contents

1 Introduction

Welcome to our Meal Planner! This application is designed to assist users in planning their meals. Whether you're a seasoned cook or just starting out in the kitchen, this app aims to simplify the process of meal planning by providing access to a comprehensive database of recipes. With features like user registration, recipe browsing, search functionality, and the ability to add new recipes, Meal Planner is your go-to solution for organizing your culinary adventures.

2 Objectives

The Meal Planner App has been developed with the following objectives in mind:

1. **Simplify Meal Planning:** The primary objective of the app is to simplify the process of meal planning for users. By providing access to a diverse range of recipes, users can easily explore and discover new meal ideas that suit their preferences and dietary requirements.
2. **Enhance User Experince:** The app aims to deliver an intuitive user experience. It strives to provide a user-friendly interface that enables users to navigate through the app effortlessly, access recipes quickly, and perform actions with ease.
3. **Offer Recipe Variety:** The app's recipe database is designed to offer a wide variety of recipes, ensuring that users have multiple options to choose from, enabling them to create diverse and exciting meals.
4. **Enable Recipe Search:** The app provides a robust search functionality, allowing users to find specific recipes based on keywords, ingredients, or recipe names. This objective ensures that users can easily locate recipes that meet their specific needs or preferences.

3 Application Design

3.1 Database Design

In order to implement the database, we have used SQL Server. The database design is simple, containing two tables: Users and Recipes.

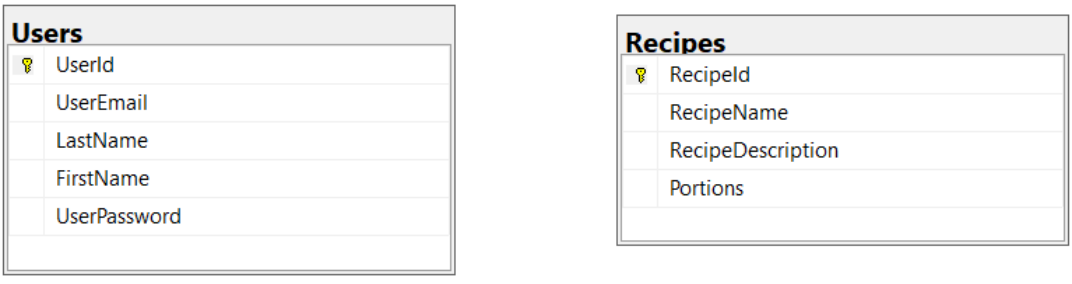

Figure 1: Database Design

1. **Users table:** The Users table stores information about registered users of the app. It contains columns such as UserId, UserEmail, LastName, FirstName, UserPassword.

The UserId is auto-incremented, while the other fields are requested during sign up. For login, only UserEmail and Password are required.

2. **Recipes:** The Recipes table serves as a repository for all the available recipes in the app. Each recipe is represented by a record in this table, which includes columns such as RecipeId, RecipeName, RecipeDescription and Portions.

3.2 GUI Design

1. **Home**

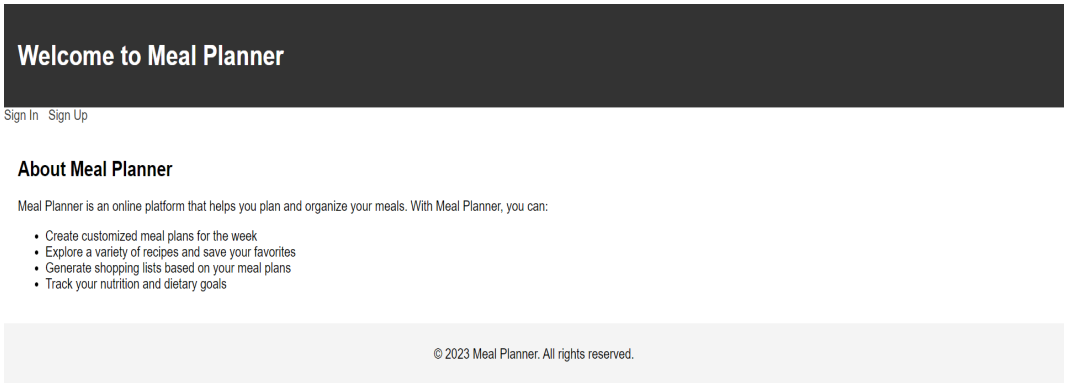

Figure 2: Home Page

2. **Sign Up**

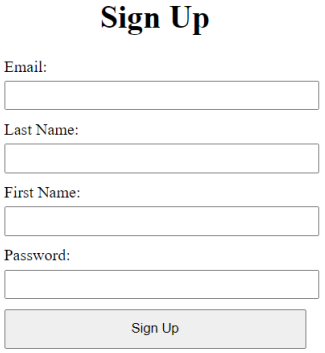

Figure 3: Sign Up Page

3. **Sign In**

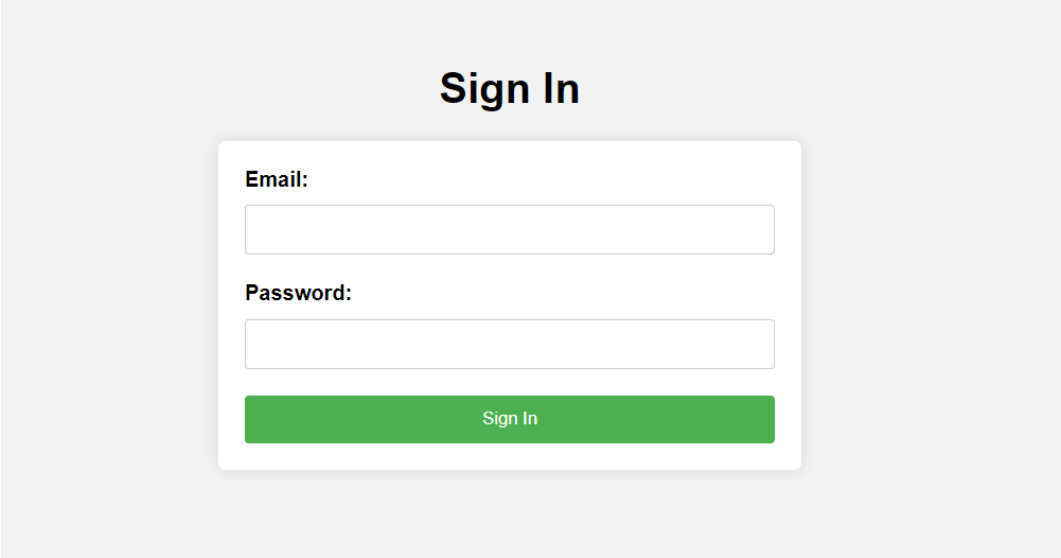

Figure 4: Sign In Page

4. Recipes

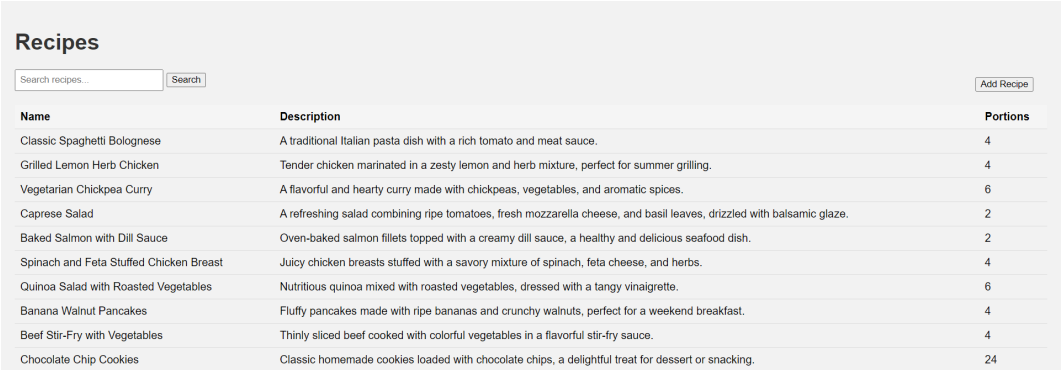

Figure 5: Recipes Page

5. New Recipe

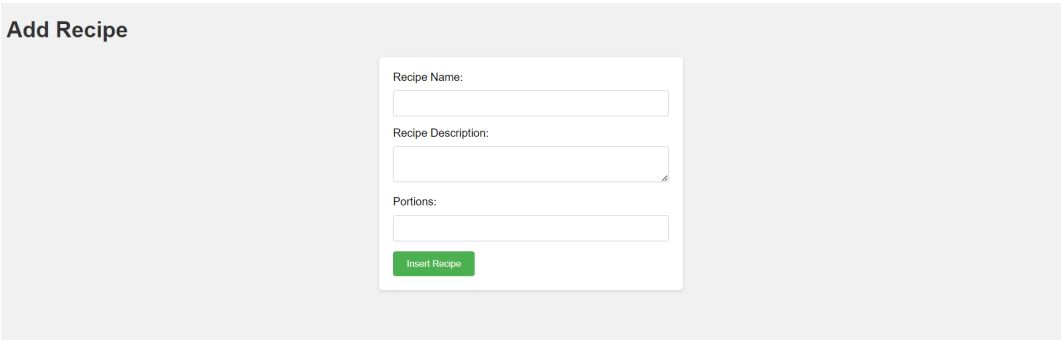

Figure 6: New Recipe Page

4 Implementation

Each page of the Meal Planner App was implemented using the server-side and client-side approach. The server-side was implemented using JavaScript, while the client-side was implemented using HTML. Additionally, CSS was used to add styling and enhance the visual appeal of the pages.

4.1 Sign Up

This subchapter provides an overview of the Sign Up method's implementation, both on the server and client side. The server side provides a connection to the database through a connection String, then it receives a post request containing the data about the user: email, last name, first name and password. The data is then added to an SQL query to be inserted into the table.

```
const express = require('express');
const path = require('path');
const sql = require('msnodesqlv8');
const bodyParser = require('body-parser');

const app = express();
var connectionString =
↪  "server=DESKTOP-LT7G6FF\\SQLEXPRESS;Database=MealPlanner;Trusted_Connection=Yes;Driver={SQL
↪  Server}";

app.use(express.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());

const signUpPath = path.join(__dirname, '..','SignUp');
app.use(express.static(path.join(signUpPath)));

app.post('/signup', (req, res) => {
  console.log(req.body);
  const { email, lastName, firstName, password } = req.body;
  console.log('Received data:', email, lastName, firstName, password);

  const query = `INSERT INTO Users (UserEmail, LastName, FirstName,
  ↪  UserPassword) VALUES ('${email}', '${lastName}', '${firstName}',
  ↪  '${password}')`;

  sql.query(connectionString, query, (err, rows) => {
    if (err) {
      console.log('Error:', err);
      res.status(500).json({ error: 'An error occurred while signing up.'
      ↪  });
    } else {
      console.log('User signed up successfully');
      res.redirect('http://localhost:3001');
    }
  });
});
```

```
app.get('/', (req, res) => {
    res.redirect('/signup');
  });

  app.get('/signup', (req, res) => {
    res.sendFile(path.join(signUpPath, 'signup.html'))
  });


app.listen(3002, () => {
  console.log('Server is running on http://localhost:3002');
});
```

The client side consists of an HTML file that sets up the sign up form with input fields for email, last name, first name and password, allowing users to submit their information.

```
<!DOCTYPE html>
<html>
<head>
    <title>Sign Up</title>
    <link rel="stylesheet" href="signup.css">
</head>
<body>
    <h1>Sign Up</h1>
    <form id="signup-form" action=/signup method="POST">
        <label for="email">Email:</label>
        <input type="email" id="email" name="email" autocomplete = "off"
        ↪  required>

        <label for="lastName">Last Name:</label>
        <input type="text" id="lastName" name="lastName" required>

        <label for="firstName">First Name:</label>
        <input type="text" id="firstName" name="firstName" required>

        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required>

        <button type="submit">Sign Up</button>
    </form>

</body>
</html>
```

4.2 Sign In

The Sign In method also receives a post request containing the user email and password required. The destructuring assignment email, password = req.body extracts the email and password properties from the request body. The query then selects all columns from Users table where the UserEmail and UserPassword match the ones provided. If there is such a row, a success message is logged to the console and redirects the user to http://localhost:3001. In the case that no match is found, the user receives an error message "Invalid email or password".

```
const express = require('express');
const path = require('path');
const sql = require('msnodesqlv8');
const bodyParser = require('body-parser');

const app = express();
var connectionString =
↪  "server=DESKTOP-LT7G6FF\\SQLEXPRESS;Database=MealPlanner;Trusted_Connection=Yes;Driver={SQL
↪  Server}";

app.use(express.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());

const signInPath = path.join(__dirname, '..','SignIn');
app.use(express.static(path.join(signInPath)));

app.post('/signin', (req, res) => {
  console.log(req.body);
  const { email, password } = req.body;
  console.log('Received data:', email, password);

  const query = `SELECT * FROM Users WHERE UserEmail = '${email}' AND
  ↪  UserPassword = '${password}'`;

  sql.query(connectionString, query, (err, rows) => {
    if (err) {
      console.log('Error:', err);
      res.status(500).json({ error: 'An error occurred while signing in.'
      ↪  });
    } else {
      if (rows.length > 0) {
        console.log('User signed in successfully');
        res.redirect('http://localhost:3001');
      } else {
        console.log('Invalid email or password');
        res.status(401).json({ error: 'Invalid email or password.' });
      }
    }
  });
});


app.get('/', (req, res) => {
  res.redirect('/signin');
});

app.get('/signin', (req, res) => {
  res.sendFile(path.join(signInPath, 'signin.html'));
});

app.listen(3000, () => {
  console.log('Server is running on http://localhost:3000');
```

```
});
```

For the client side, the HTML file sets up a form with fields for email and password, that allow the user to authenticate.

```
<!DOCTYPE html>
<html>
<head>
    <title>Sign In</title>
    <link rel="stylesheet" href="signin.css">
</head>
<body>
    <h1>Sign In</h1>
    <form id="signin-form" action="/signin" method="POST">
        <label for="email">Email:</label>
        <input type="email" id="email" name="email" autocomplete="off"
        ↪  required>

        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required>

        <button type="submit">Sign In</button>
    </form>
</body>
</html>
```

4.3 Recipes

The Recipes file contains both the client and server side. In this method a query is used to display all the recipe names, descriptions and portions. A get request is used for fetching the recipes. Also, the html variable is initialized with the HTML code for the page, including the DOCTYPE declaration, HTML tags, and the basic structure. The code also includes a search bar that uses the function "Search recipes". This function is triggered when the user clicks the "Search" button on the web page. This function performs a search on a table of recipes by comparing the search input with the recipe names and descriptions. It dynamically shows or hides table rows based on whether the search input matches the recipe information, allowing the user to filter and display specific recipes.

```
const express = require('express');
const path = require('path');
const sql = require('msnodesqlv8');

const app = express();
var connectionString =
↪  "server=DESKTOP-LT7G6FF\\SQLEXPRESS;Database=MealPlanner;Trusted_Connection=Yes;Driver={SQL
↪  Server}";

const recipesPath = path.join(__dirname, '..', 'Recipes');
app.use(express.static(path.join(recipesPath)));

app.get('/recipes', (req, res) => {
  const query = "SELECT RecipeName, RecipeDescription, Portions FROM
  ↪  Recipes";
```

```
sql.query(connectionString, query, (err, rows) => {
  if (err) {
    console.log('Error:', err);
    res.status(500).json({ error: 'An error occurred while fetching
    ↪  recipes.' });
  } else {
    console.log('Recipes fetched successfully');

    // Generate the HTML code for the table
    let html = `
      <!DOCTYPE html>
      <html>
      <head>
        <title>Recipes</title>
        <style>
          body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 20px;
            background-color: #f2f2f2;
          }

          h1 {
            color: #333;
          }

          #recipes-table {
            border-collapse: collapse;
            width: 100%;
            margin-top: 20px;
          }

          #recipes-table th,
          #recipes-table td {
            padding: 8px;
            text-align: left;
            border-bottom: 1px solid #ddd;
          }

          #recipes-table th {
            background-color: #f5f5f5;
          }

          #recipes-table tr:hover {
            background-color: #f9f9f9;
          }

          .search-bar {
            margin-bottom: 20px;
          }

          .search-bar input[type="text"] {
            padding: 6px;
```

```
            width: 200px;
          }

          .add-recipe-btn {
            float: right;
            margin-top: -40px;
            margin-right: 20px;
          }
        </style>
      </head>
      <body>
        <h1>Recipes</h1>
        <div class="search-bar">
          <input type="text" id="search-input" placeholder="Search
↪ recipes..." />
          <button onclick="searchRecipes()">Search</button>
        </div>
        <button class="add-recipe-btn" onclick="window.location.href =
↪ 'http://localhost:3003'">Add Recipe</button>
        <table id="recipes-table">
          <thead>
            <tr>
              <th>Name</th>
              <th>Description</th>
              <th>Portions</th>
            </tr>
          </thead>
          <tbody>`;

    rows.forEach((recipe) => {
      html += `
        <tr>
          <td>${recipe.RecipeName}</td>
          <td>${recipe.RecipeDescription}</td>
          <td>${recipe.Portions}</td>
        </tr>`;
    });

    html += `
        </tbody>
      </table>

      <script>
        function searchRecipes() {
          var input = document.getElementById('search-input');
          var filter = input.value.toLowerCase();
          var table = document.getElementById('recipes-table');
          var rows = table.getElementsByTagName('tr');

          for (var i = 0; i < rows.length; i++) {
            var recipeName = rows[i].getElementsByTagName('td')[0];
            var recipeDescription =
↪ rows[i].getElementsByTagName('td')[1];
```

```
            if (recipeName && recipeDescription) {
              var recipeNameText = recipeName.textContent ||
↪  recipeName.innerText;
              var recipeDescriptionText = recipeDescription.textContent
↪  || recipeDescription.innerText;
              var recipeText = recipeNameText.toLowerCase() +
↪  recipeDescriptionText.toLowerCase();

              if (recipeText.indexOf(filter) > -1) {
                rows[i].style.display = '';
              } else {
                rows[i].style.display = 'none';
              }
            }
          }
        }
      </script>

      </body>
      </html>`;

    res.send(html);
  }
 });
});


app.get('/', (req, res) => {
  res.redirect('/recipes');
});

app.get('/recipes', (req, res) => {
  res.sendFile(path.join(recipesPath, 'recipes.html'));
});


app.listen(3001, () => {
  console.log('Server is running on http://localhost:3001');
});
```

4.4 New Recipe

Like the Sign Up file, the New Recipe file contains a server and client part. The server side receives the recipe name, description and portions through a post request and inserts the provided values into the database through the SQL query.

```
const express = require('express');
const path = require('path');
const sql = require('msnodesqlv8');
const bodyParser = require('body-parser');

const app = express();
```

```
var connectionString =
↪  "server=DESKTOP-LT7G6FF\\SQLEXPRESS;Database=MealPlanner;Trusted_Connection=Yes;Driver={SQL
↪  Server}";

app.use(express.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());

const recipePath = path.join(__dirname, '..','Newrecipe');
app.use(express.static(path.join(recipePath)));

app.post('/newrecipe', (req, res) => {
  console.log(req.body);
  const { recipeName, recipeDescription, portions } = req.body;
  console.log('Received data:', recipeName, recipeDescription, portions);

  const query = `INSERT INTO Recipes (RecipeName, RecipeDescription,
  ↪  Portions) VALUES ('${recipeName}', '${recipeDescription}',
  ↪  '${portions}')`;

  sql.query(connectionString, query, (err, rows) => {
    if (err) {
      console.log('Error:', err);
      res.status(500).json({ error: 'An error occurred while adding.' });
      return;
    }

    console.log('Added successfully');
    res.status(200).json({ message: 'Added successfully.' });
  });
});



app.get('/', (req, res) => {
  res.redirect('/newrecipe');
});

app.get('/newrecipe', (req, res) => {
  res.sendFile(path.join(recipePath, 'newrecipe.html'));
});

app.listen(3003, () => {
  console.log('Server is running on http://localhost:3003');
});
```

The HTML file provides a form with input fields which allow the user to submit the information required to add a new recipe.

```
<!DOCTYPE html>
<html>
<head>
  <title>Add Recipe</title>
  <style>
    body {
```

```
      font-family: Arial, sans-serif;
      background-color: #f1f1f1;
    }

    h1 {
      color: #333;
    }

    form {
      max-width: 400px;
      margin: 0 auto;
      background-color: #fff;
      padding: 20px;
      border-radius: 5px;
      box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
    }

    label {
      display: block;
      margin-bottom: 10px;
    }

    input[type="text"],
    input[type="number"],
    textarea {
      width: 100%;
      padding: 10px;
      border: 1px solid #ccc;
      border-radius: 4px;
      box-sizing: border-box;
      margin-bottom: 15px;
    }

    button[type="submit"] {
      background-color: #4caf50;
      color: #fff;
      padding: 10px 20px;
      border: none;
      border-radius: 4px;
      cursor: pointer;
    }

    button[type="submit"]:hover {
      background-color: #45a049;
    }
  </style>
</head>
<body>
  <h1>Add Recipe</h1>
  <form id="recipeForm">
    <label for="recipeName">Recipe Name:</label>
    <input type="text" id="recipeName" name="recipeName" required><br>
```

```
  <label for="recipeDescription">Recipe Description:</label>
  <textarea id="recipeDescription" name="recipeDescription"
  ↪  required></textarea><br>

  <label for="portions">Portions:</label>
  <input type="number" id="portions" name="portions" required><br>

  <button type="submit">Insert Recipe</button>
</form>

<script>
  document.getElementById('recipeForm').addEventListener('submit',
  ↪  function(event) {
    event.preventDefault();

    var recipeName = document.getElementById('recipeName').value;
    var recipeDescription =
    ↪  document.getElementById('recipeDescription').value;
    var portions = document.getElementById('portions').value;

    var data = {
      recipeName: recipeName,
      recipeDescription: recipeDescription,
      portions: portions
    };

    fetch('/newrecipe', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(data)
    })
    .then(function(response) {
      if (response.ok) {
        console.log('Recipe inserted successfully');
        alert('Recipe inserted successfully');

        fetch('/newrecipe')
          .then(function(response) {
            if (response.ok) {
              return response.json();
            } else {
              throw new Error('Error fetching ingredients');
            }
          })
          .then(function(data) {
            var ingredientList =
            ↪  document.getElementById('ingredient-list');
            ingredientList.innerHTML = '';

            data.ingredients.forEach(function(ingredient) {
              var li = document.createElement('li');
```

```
            li.textContent = ingredient.IngredientName;
            ingredientList.appendChild(li);
          });

          var allIngredientsList =
          ↪  document.getElementById('all-ingredients-list');
          allIngredientsList.innerHTML = '';

          data.ingredients.forEach(function(ingredient) {
            var li = document.createElement('li');
            li.textContent = ingredient.IngredientName;
            allIngredientsList.appendChild(li);
          });
        })
        .catch(function(error) {
          console.error('Error fetching ingredients:', error.message);
        });
      } else {
        console.error('Error inserting recipe');
        alert('Error inserting recipe');
      }
    })
    .catch(function(error) {
      console.error('Error inserting recipe:', error.message);
      alert('Error inserting recipe');
    });
  });
</script>

</body>
</html>
```