SEM II 2023/2024

CSE3023 - Web-Based Application and Development (K1)

GROUP PROJECT

Group 13: Cooking Delight (Knowledge Management System)

Prepared By:

Name	Matric No.
Ahmad Syaiful Hafizi Bin Azhar	S63606
Nur Hidayatul Ain Bt Haluwi	S62367
NurSyaza Amira Binti Selamat	S62318

Prepared For:

Dr. Mohamad Nor Hassan

Faculty of Computer Science and Mathematics (FCSM)

UNIVERSITI MALAYSIA TERENGGANU

Table of Contents

Executive Summary

Introduction

Cooking Delight is an innovative web-based Knowledge Management System designed to revolutionize how cooking enthusiasts store, share, and discover recipes. Tailored for home cooks, professional chefs, and food lovers, Cooking Delight aims to centralize all aspects of recipe management and community interaction into one seamless platform.

Problem Statement

In today's digital age, there is an overwhelming amount of culinary information scattered across various platforms, making it difficult for users to manage their recipes, and find reliable cooking advice. Existing solutions often need more comprehensive features to integrate these functionalities effectively.

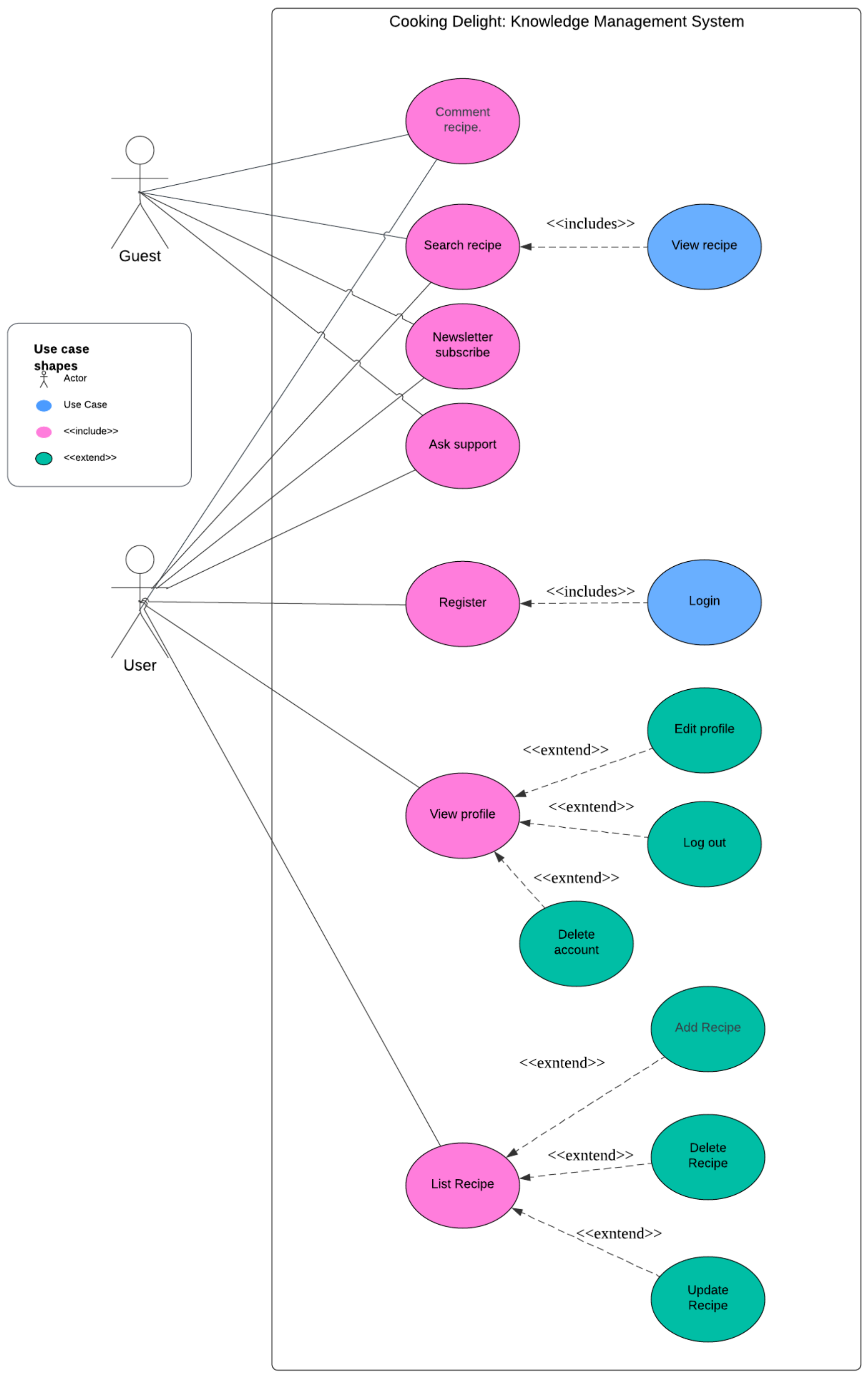
Objectives

- To provide a user-friendly, centralized repository for storing and managing recipes.
- To offer a search option to easily discover new recipes based on input.
- To foster a community of cooking enthusiasts through interactive features like comments.

Key Features

- User Management: Secure user authentication with a secure process.
- Recipe Repository**:** Basic recipe management with barebones feature.
- Comment Section: Allow users to comment on recipes, share tips, and provide feedback.
- Newsletter: Users can subscribe to newsletters to receive regular updates.

Use Case Diagram



Class Diagram

The class diagram for the Recipe Archive Management System outlines the structure and relationships between different classes in the system. This diagram includes classes for User, Guest, Recipe, Comment, and Feedback, showing their attributes and methods as well as the associations between them. Figure 2.0 shows a class diagram for the Recipe Archive System.

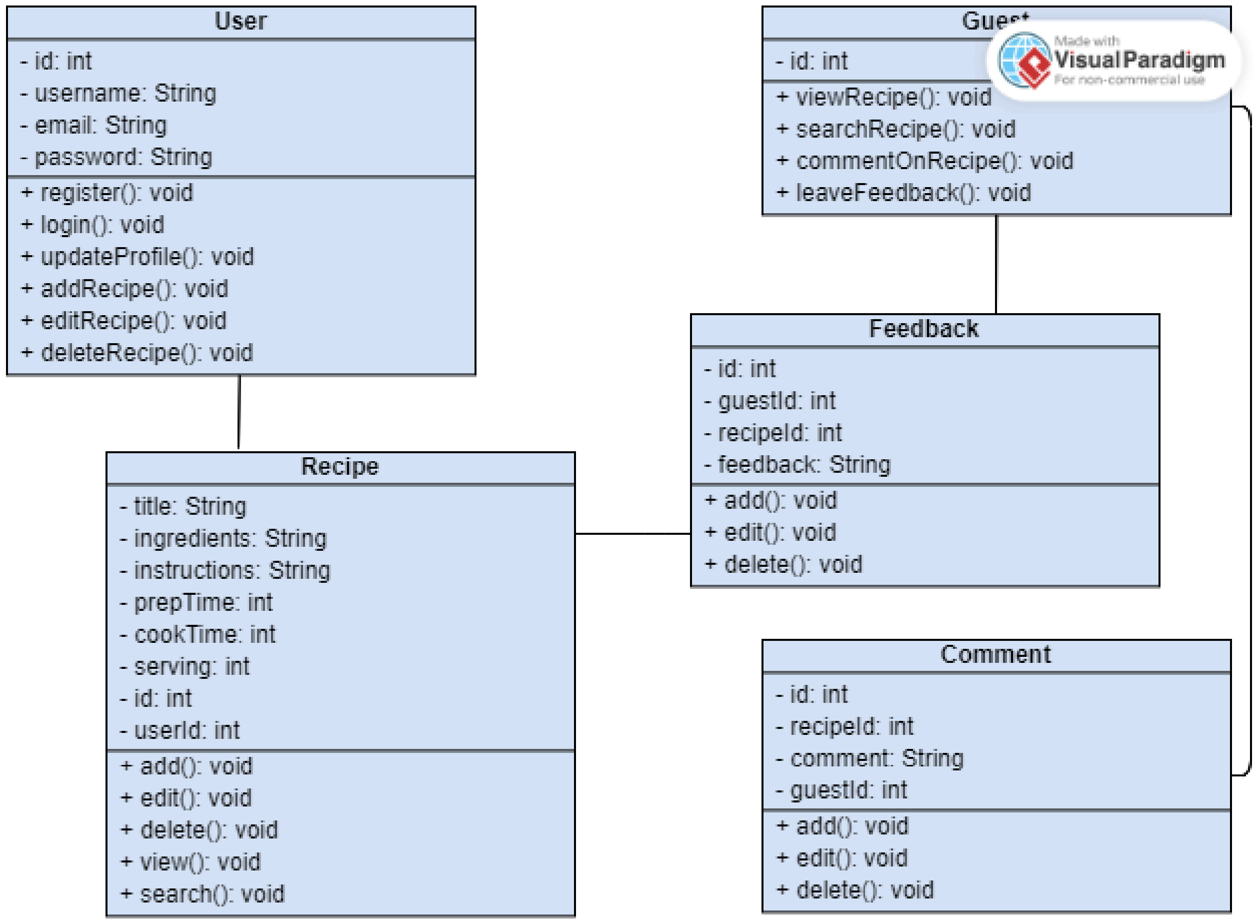


Figure 2.0

Classes and Attributes

User Class:

1. Attributes:

- id: Unique identifier for the user.
- username: The username chosen by the user.
- email: The user's email address.
- password: The user's password.

2. Methods:
    - register(): Allows a new user to register.
    - login(): Allows a user to log in.
    - updateProfile(): Allows a user to update their profile information.
    - addRecipe(): Allows a user to add a new recipe.
    - editRecipe(): Allows a user to edit their own recipe.
    - deleteRecipe(): Allows a user to delete their own recipe.

Guest Class:

1. Attributes:
    - id: Unique identifier for the user.
2. Methods:
    - viewRecipe(): Allows a guest to view recipes.
    - searchRecipes(): Allows a guest to search for recipes based on various criteria.
    - leaveFeedback(): Allows a guest to leave feedback on a recipe.
    - commentOnRecipe(): Allows a guest to comment on a recipe.

Recipe class:

1. Attributes:
    - id: Unique identifier for the recipe.
    - userId: Identifier of the user who created the recipe.
    - title: Title of the recipe.
    - ingredients: Ingredients required for the recipe.
    - instructions: Step-by-step instructions for preparing the recipe.
    - prepTime: Preparation time in minutes.
    - cookTime: Cooking time in minutes.
    - servings: Number of servings the recipe yields.
2. Methods:

- add(): Allows a user to add a new recipe.
- edit(): Allows a user to edit their own recipe.
- delete(): Allows a user to delete their own recipe.
- view(): Allows viewing a recipe's details.
- search(): Allows searching for recipes.

Comment Class:

1. Attributes:
    - id: Unique identifier for the comment.
    - guestId: Identifier of the guest who made the comment.
    - recipeId: Identifier of the recipe the comment is associated with.
    - comment: The text of the comment.
2. Methods:
    - add(): Allows a guest to add a new comment.
    - edit(): Allows a guest to edit their own comment.
    - delete(): Allows a guest to delete their own comment.

Feedback Class:

1. Attributes:
    - id: Unique identifier for the feedback.
    - guestId: Identifier of the guest who left the feedback.
    - recipeId: Identifier of the recipe the feedback is related to.
    - feedback: The text of the feedback.
2. Methods:
    - add(): Allows a guest to add new feedback.
    - edit(): Allows a guest to edit their own feedback.
    - delete(): Allows a guest to delete their own feedback.

User Traceability Matrix

User Requirement	**Use Case**	**Web Page/Features**	**Controller/Servlet**	**Model/DAO**
View User Profile	View Profile	userProfile.jsp	UserController.java	UserDao.java
Update User Profile	Update Profile	updateUserProfile.jsp	UserController.java	userDao.java
Create Comment	Create Comment	createComment.jsp	RecipeController.java	CommentDao.java
View Recipe	View Recipe	viewRecipe.jsp	RecipeController.java	RecipeDao.java
Login	Login	login.jsp	LoginServlet	UserDao.java
Register	Register	register.jsp	RegisterServlet.java	UserDao.java
Manage Recipes	Manage Recipes	manageRecipes.jsp	RecipeController	Recipe.Dao.java

ER Diagram

The Entity-Relationship Diagram (ERD) for the Recipe Archive Management System provides a detailed visual representation of the data and relationships within the system. It highlights the key entities involved and their interconnections, ensuring a clear understanding of the data structure and interactions. Figure 3.0 shows a class diagram for the Recipe Archive System.

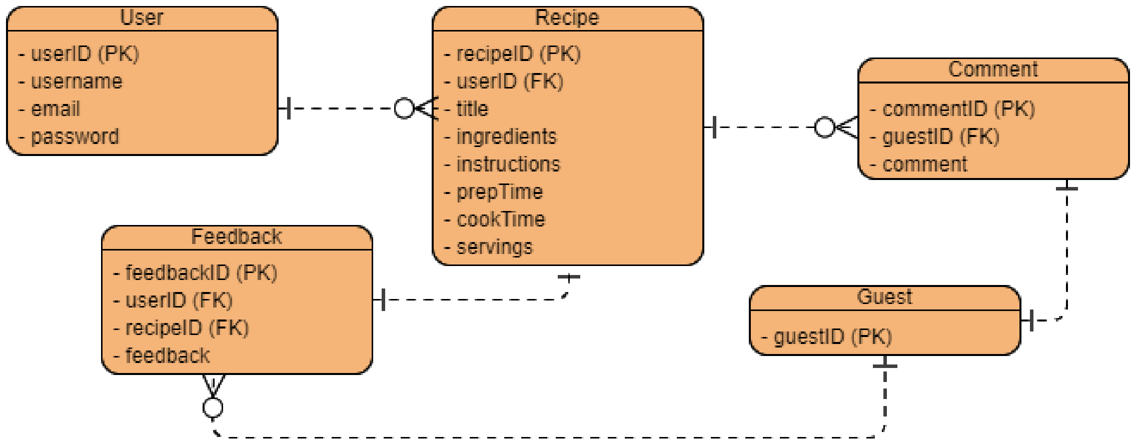


Figure 3.0

The User entity represents registered users and contains attributes such as userID, username, email, and password, which store user information and facilitate user management activities like registration and login. The Guest entity contains attributes guestID and capable viewing and searching recipes and leaving feedback.

The Recipe entity captures the details of each recipe, including attributes like recipeID, userID, title, ingredients, instructions, prepTime, cookTime, and servings. This entity is linked to the User entity through a one-to-many relationship, as each user can create multiple recipes.

Comments on recipes are managed through the Comment entity, which includes attributes such as commentID, guestID, recipeID, and comment.The Comment entity establishes relationships with both the Guest and Recipe entities, indicating that guest can leave comments on recipes, and each recipe can have multiple comments.

Additionally, the Feedback entity is designed to allow guests to leave feedback on recipes, containing attributes like feedbackID, guestID, recipeID, and feedback. This entity is associated with both the Guest and Recipe entities, highlighting that guests can provide feedback on multiple recipes, and each recipe can receive feedback from various guests.

The ERD effectively organizes these entities and their relationships, using primary and foreign keys to link them and ensuring a structured and coherent database design. This clear framework facilitates efficient data handling and interaction within the Recipe Archive Management System, providing a solid foundation for developing and managing the system's database.

MVC Model

The Model-View-Controller (MVC) design pattern is a widely used architectural pattern in software development, especially for web applications. It separates an application into three interconnected components: the Model, the View, and the Controller. This separation helps in organizing code and improving scalability and maintainability.

User

The *user* MVC model consists of User.java as JavaBean, UserDao.java as model/Data Access Object, UserController.java as controller, and all JSP pages under the /profileView directory as view.

Recipe

The *recipe* MVC model is comprised of Recipe.java as JavaBean, RecipeDao.java as model/Data Access Object, RecipeController.java as controller, and all JSP pages under the /recipeView directory as view.

Comment

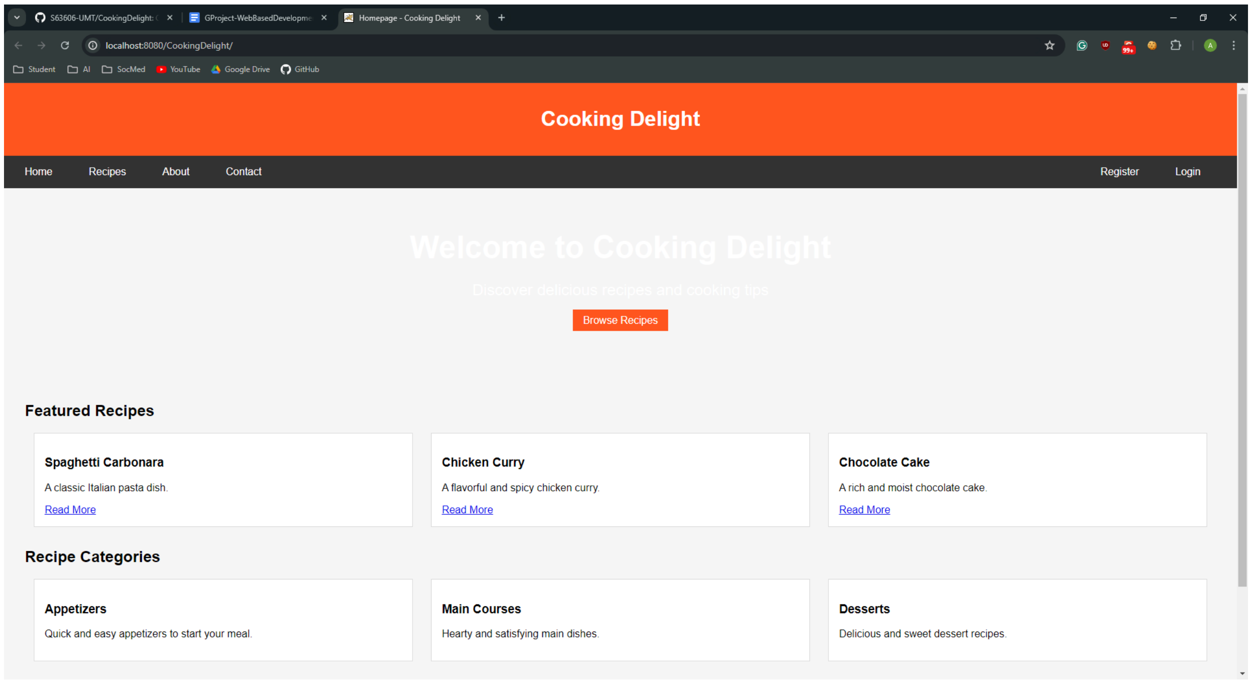
The *comment* MVC model is composed of Comment.java as JavaBean, and CommentDao.java as model/Data Access Object, but has none of its own Controller or View since it's using Recipe's Controller and View instead.

Newsletter

The *newsletter* MVC model is comprised of Newsletter.java as JavaBean, NewsletterDao.java as model/Data Access Object, and NewsletterController.java as controller. It only uses *about.jsp* as the View.

Sample of Interface

Index.jsp:


S63606-UMT/CookingDelight
GProject-WebBasedDevelopme
Homepage - Cooking Delight
localhost:8080/CookingDelight/
Student
AI
SocMed
YouTube
Google Drive
GitHub

Cooking Delight

Home Recipes About Contact Register Login

Welcome to Cooking Delight

Discover delicious recipes and cooking tips

Browse Recipes

Featured Recipes

Spaghetti Carbonara
A classic Italian pasta dish.
Read More

Chicken Curry
A flavorful and spicy chicken curry.
Read More

Chocolate Cake
A rich and moist chocolate cake.
Read More

Recipe Categories

Appetizers
Quick and easy appetizers to start your meal.

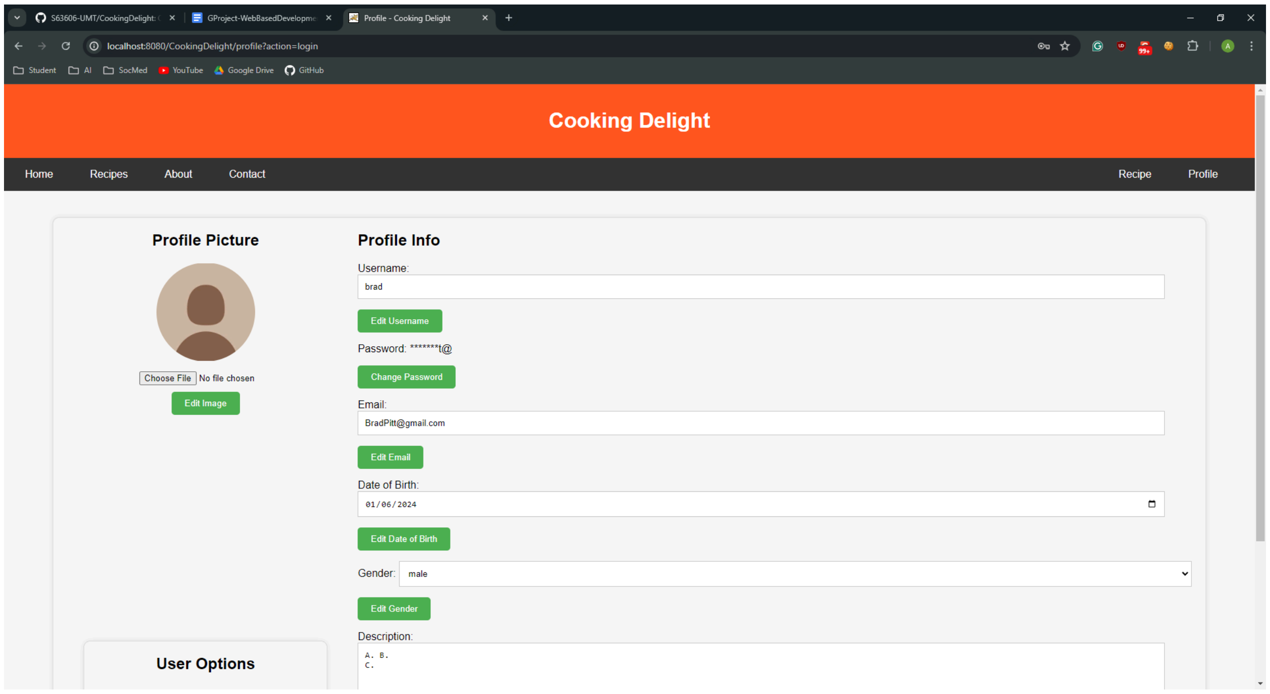
Main Courses
Hearty and satisfying main dishes.

Desserts
Delicious and sweet dessert recipes.

Figure 4.0

Figure is a homepage of this web application. It serves as the entry point for users, providing them with an overview of what this application offers. This page includes a welcoming message, links to other pages, and a navigation bar to facilitate easy navigation.

ProfileView/profile.jsp:



Username: brad

Password: *******t@

Email: BradPitt@gmail.com

Date of Birth: 01/06/2024

Gender: male

Description:
```
A. B.
C.
```

Figure 5.0

Figure 5.0 shows a ProfileView/profile.jsp page. Located in the profileView directory, this page's purpose is to provide a display of the user's data and the capability to edit any info, log out, and delete the user's account. Only accessible when logged in.

register.jsp:

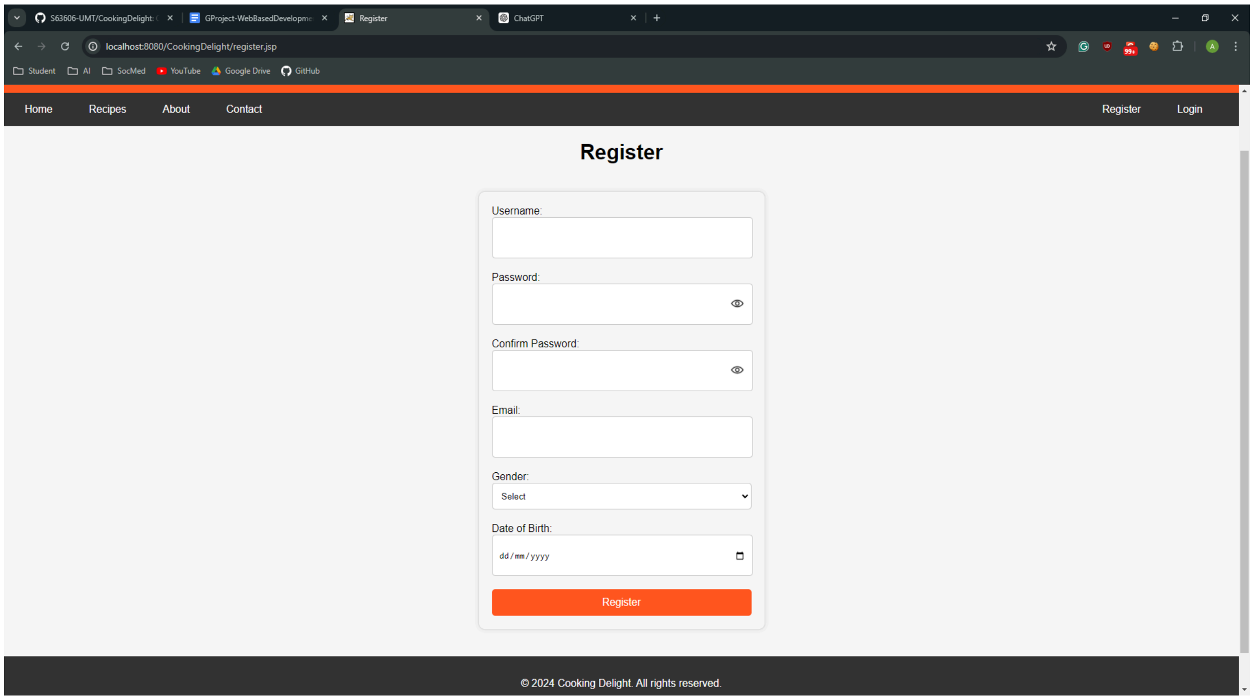
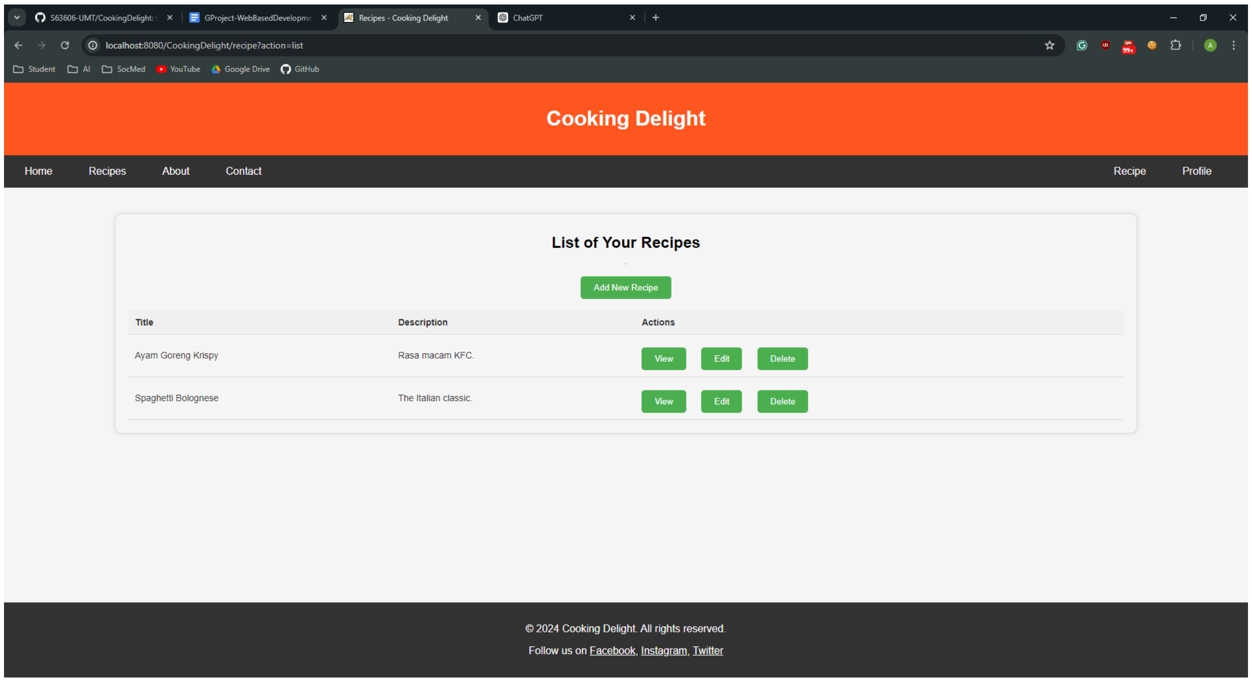


Figure 6.0

Figure 6.0 shows register.jsp page where this page shows a form for registration for a new account. Users can fill out this form to create a new profile by providing essential information such as their username, password, and other relevant details. The form ensures that all necessary fields are completed and validates the input to maintain data integrity and security. After submitting the form, users will receive a confirmation email to verify their account. Once verified, they can log in and access the full features of the application, such as personalizing their profile, interacting with other users, and utilizing the services offered.

recipeView/RecipeList.jsp:


Cooking Delight

Home Recipes About Contact Recipe Profile

List of Your Recipes

Add New Recipe

Title	Description	Actions
Ayam Goreng Krispy	Rasa macam KFC.	View Edit Delete
Spaghetti Bolognese	The Italian classic.	View Edit Delete

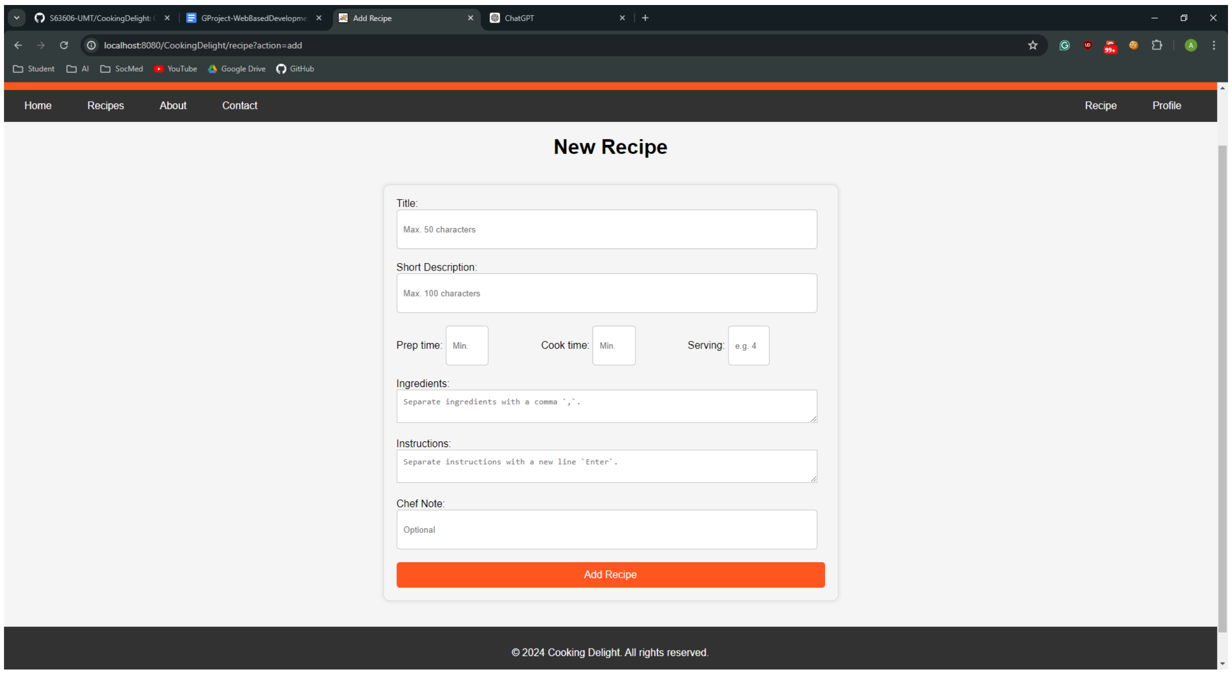
© 2024 Cooking Delight. All rights reserved.

Follow us on Facebook, Instagram, Twitter

Figure 7.0

Figure 7.0 shows recipeView/Recipe.jsp where a list of recipes made and added by the user is displayed. This page is only accessible when the user is logged in. Users can view, edit, and delete their own recipes, as well as browse through the details of each recipe, including ingredients, instructions, and other relevant details.

recipeView/Add.jsp:



New Recipe

Title: Max. 50 characters

Short Description: Max. 100 characters

Prep time: Min. Cook time: Min. Serving: e.g. 4

Ingredients: Separate ingredients with a comma `,`.

Instructions: Separate instructions with a new line `Enter`.

Chef Note: Optional

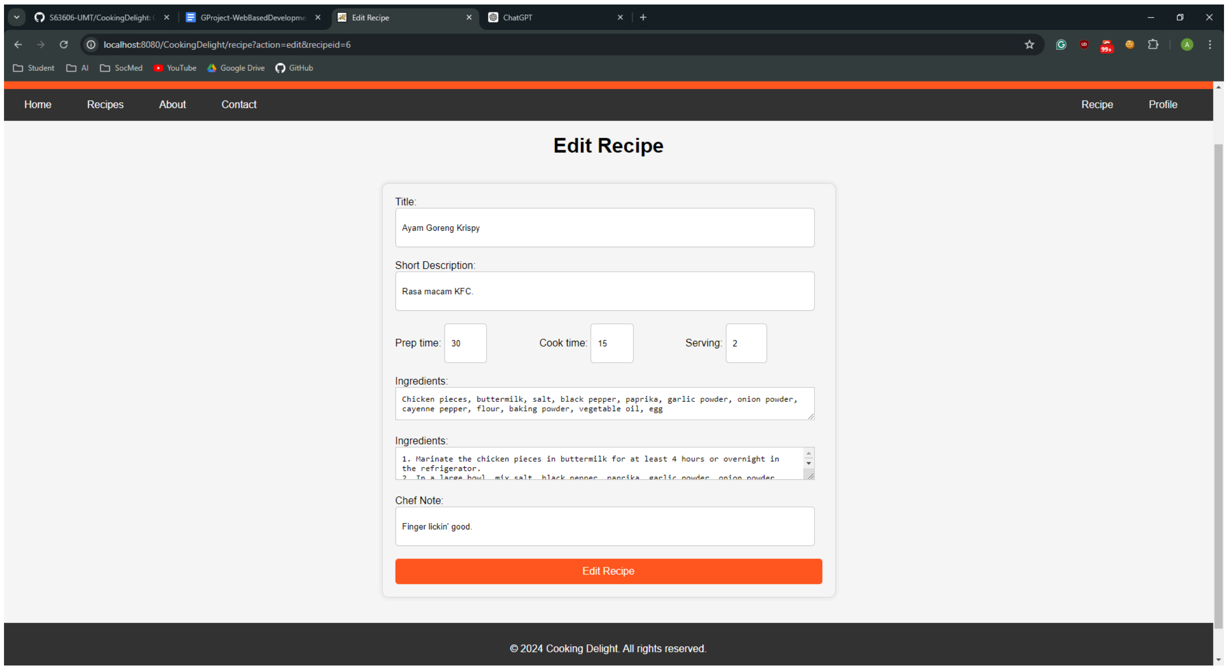
Add Recipe

© 2024 Cooking Delight. All rights reserved.

Figure 8.0

Figure 8.0 shows recipeView/Add.jsp where this page shows a form for adding a new recipe. Users can enter details such as the recipe name, ingredients, preparation steps, cooking time, and any additional notes. The form includes input fields for each of these details, ensuring that users provide all necessary information for a complete recipe. Once the form is filled out, users can submit it by clicking the 'Add Recipe' button, which will save the new recipe to the database and make it available for viewing by other users.

recipeView/edit.jsp:



Edit Recipe

Title:
Ayam Goreng Krispy

Short Description:
Rasa macam KFC.

Prep time: 30 Cook time: 15 Serving: 2

Ingredients:
Chicken pieces, buttermilk, salt, black pepper, paprika, garlic powder, onion powder, cayenne pepper, flour, baking powder, vegetable oil, egg

Ingredients:
1. Marinate the chicken pieces in buttermilk for at least 4 hours or overnight in the refrigerator.
2. In a large bowl, mix salt, black pepper, paprika, garlic powder, onion powder

Chef Note:
Finger lickin' good.

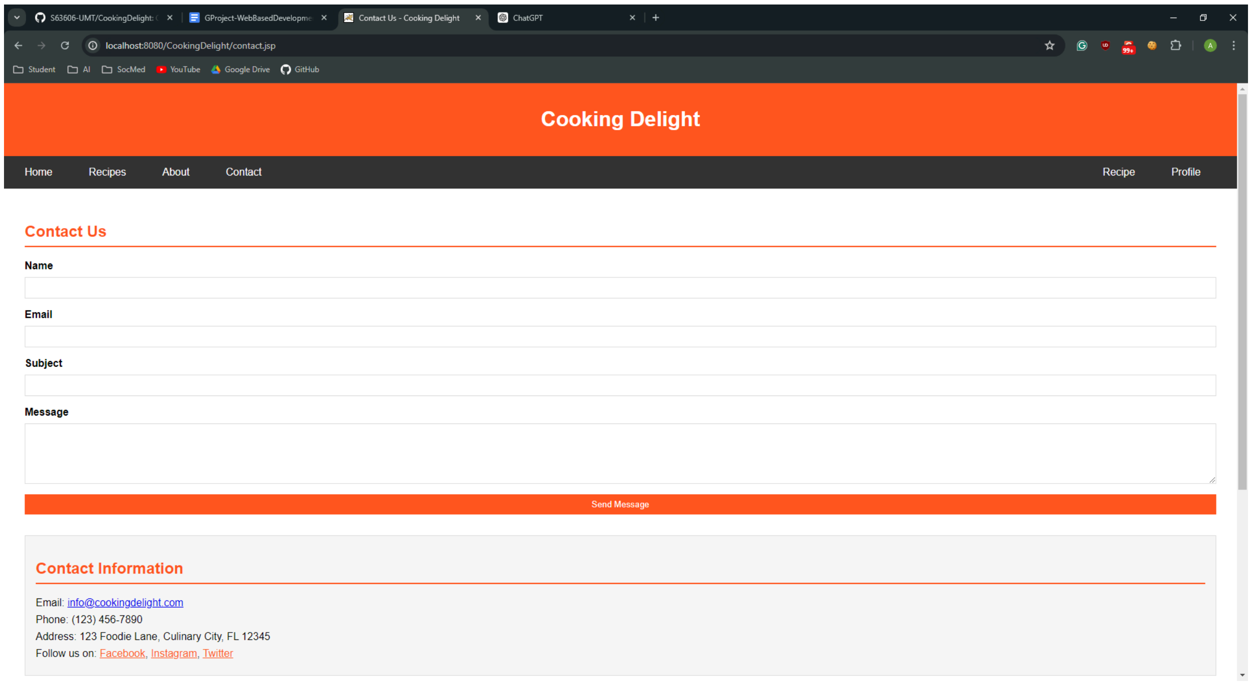
Edit Recipe

© 2024 Cooking Delight. All rights reserved.

Figure 9.0

Figure 9.0 shows recipeView/edit.jsp where the same as previous, except inputs are filled with existing values from the selected recipe. This page allows users to view and edit the details of a specific recipe. The form fields are pre-populated with the current information for the recipe, making it easy for users to update any part of the recipe, such as the title, ingredients, and instructions. Once the desired changes are made, the user can submit the form to save the updates, ensuring that the recipe information remains accurate and up-to-date.

Contact.jsp:


localhost:8080/CookingDelight/contact.jsp

Cooking Delight

Contact Us

Name

Email

Subject

Message

Contact Information

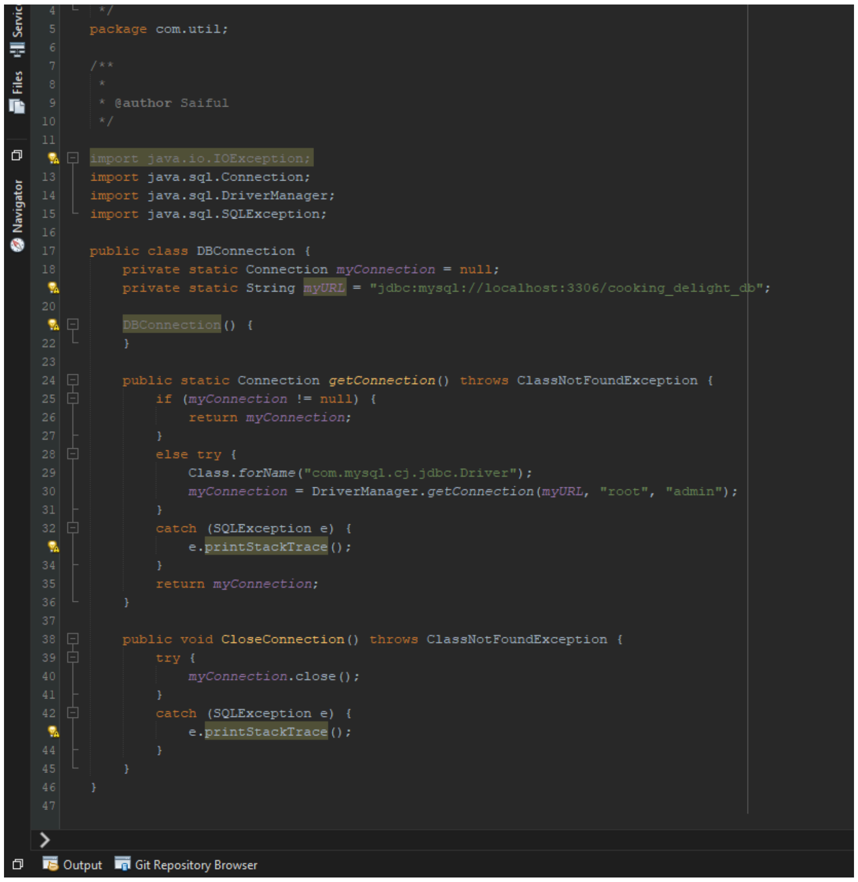
Email: info@cookingdelight.com
Phone: (123) 456-7890
Address: 123 Foodie Lane, Culinary City, FL 12345
Follow us on: Facebook, Instagram, Twitter

Figure 10.0

Figure 10.0 shows Contact.jsp where this page displays a form for sending an email to the supposed manager/moderator/admin. For simulation, the form used Web3Forms, a Simple Mail Transfer Protocol (SMTP) service, to send an email to one of the members, S63606. This form includes fields for the user's name, email address, subject, and message. Upon submission, the form captures the user input and utilizes the Web3Forms API to securely transmit the message. This setup allows for efficient communication within the application, ensuring that any inquiries or issues raised by users are promptly addressed by the appropriate personnel. Additionally, the page includes validation to ensure all required fields are filled out correctly before the form can be submitted.

Sample of Coding

DBConnection.java:

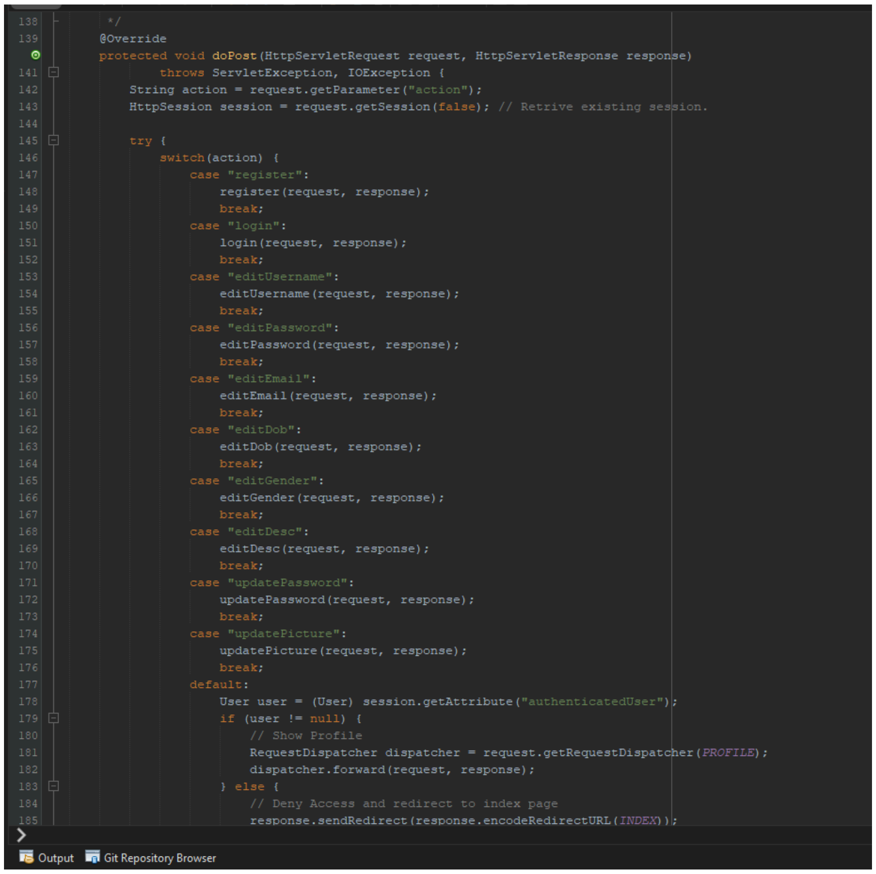


```
 */
package com.util;

/**
 *
 * @author Saiful
 */

import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnection {
    private static Connection myConnection = null;
    private static String myURL = "jdbc:mysql://localhost:3306/cooking_delight_db";

    DBConnection() {
    }

    public static Connection getConnection() throws ClassNotFoundException {
        if (myConnection != null) {
            return myConnection;
        }
        else try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            myConnection = DriverManager.getConnection(myURL, "root", "admin");
        }
        catch (SQLException e) {
            e.printStackTrace();
        }
        return myConnection;
    }

    public void CloseConnection() throws ClassNotFoundException {
        try {
            myConnection.close();
        }
        catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Figure 11.0

Figure 11.0 shows DBConnection.java, which is responsible for making connections to the database. This class handles the creation, management, and closing of database connections, ensuring that the application can securely and efficiently interact with the database. By centralizing the database connection logic, DBConnection.java helps maintain a clean and organized codebase, promoting reusability and reducing redundancy.

UserController.java:

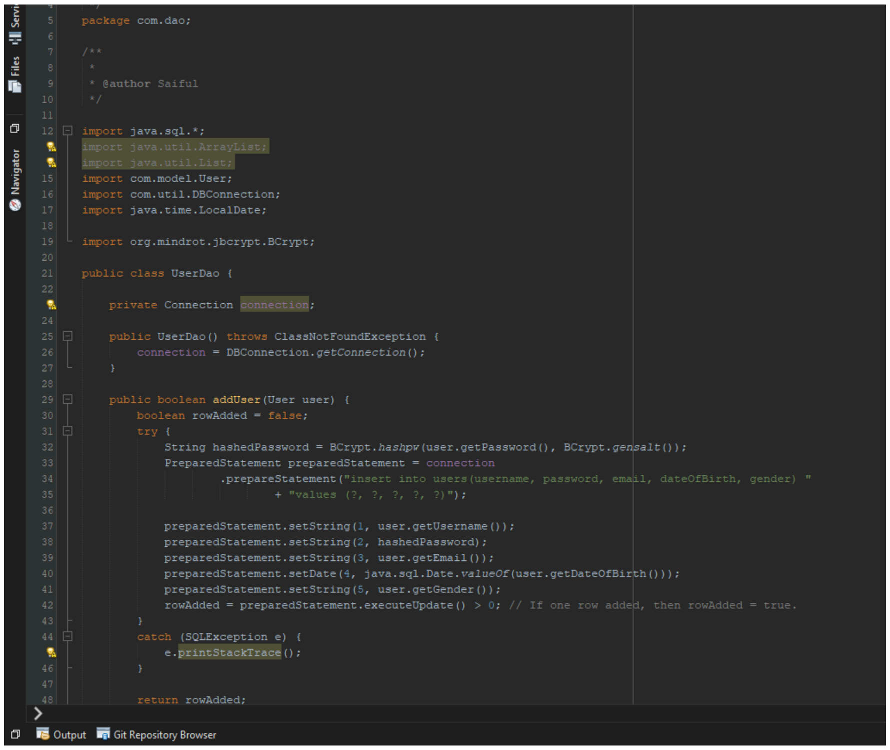


```java
     */
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String action = request.getParameter("action");
        HttpSession session = request.getSession(false); // Retrive existing session.

        try {
            switch(action) {
                case "register":
                    register(request, response);
                    break;
                case "login":
                    login(request, response);
                    break;
                case "editUsername":
                    editUsername(request, response);
                    break;
                case "editPassword":
                    editPassword(request, response);
                    break;
                case "editEmail":
                    editEmail(request, response);
                    break;
                case "editDob":
                    editDob(request, response);
                    break;
                case "editGender":
                    editGender(request, response);
                    break;
                case "editDesc":
                    editDesc(request, response);
                    break;
                case "updatePassword":
                    updatePassword(request, response);
                    break;
                case "updatePicture":
                    updatePicture(request, response);
                    break;
                default:
                    User user = (User) session.getAttribute("authenticatedUser");
                    if (user != null) {
                        // Show Profile
                        RequestDispatcher dispatcher = request.getRequestDispatcher(PROFILE);
                        dispatcher.forward(request, response);
                    } else {
                        // Deny Access and redirect to index page
                        response.sendRedirect(response.encodeRedirectURL(INDEX));
```

Figure 12.0

Figure 12.0 shows UserController.java where the doPost method is used to receive POST requests from the client. This method is responsible for handling client requests, specifically for the User View, featuring user profile management functionalities. It processes data sent by the user, such as profile updates, password changes, and other personal information modifications. Upon receiving a request, the doPost method validates the input data, interacts with the model layer to update the database.

UserDao.java:

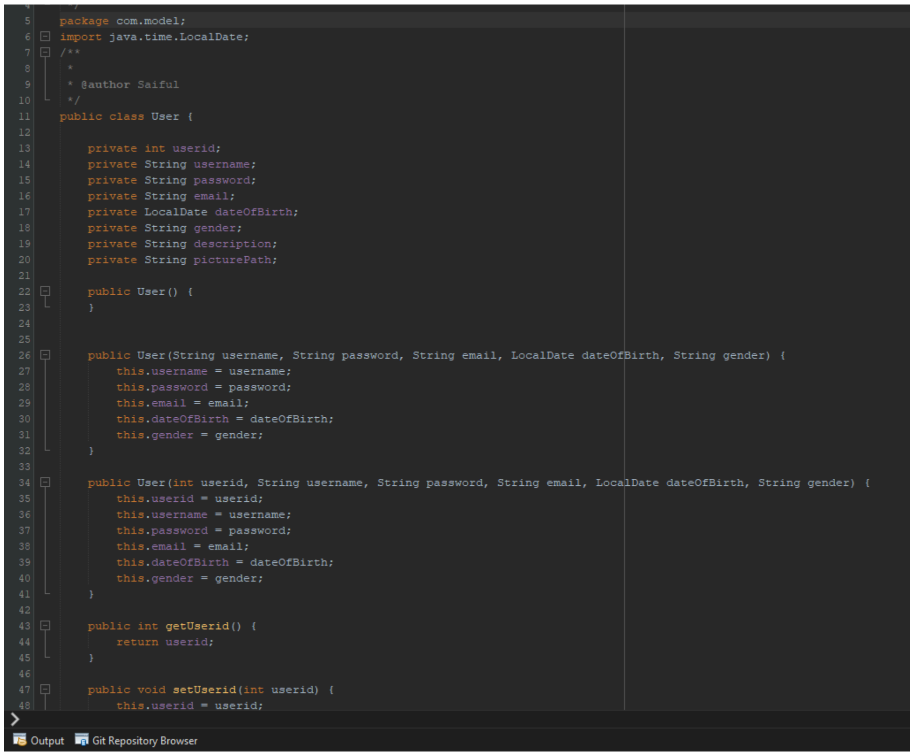


```java
package com.dao;

/**
 *
 * @author Saiful
 */

import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import com.model.User;
import com.util.DBConnection;
import java.time.LocalDate;

import org.mindrot.jbcrypt.BCrypt;

public class UserDao {

    private Connection connection;

    public UserDao() throws ClassNotFoundException {
        connection = DBConnection.getConnection();
    }

    public boolean addUser(User user) {
        boolean rowAdded = false;
        try {
            String hashedPassword = BCrypt.hashpw(user.getPassword(), BCrypt.gensalt());
            PreparedStatement preparedStatement = connection
                    .prepareStatement("insert into users(username, password, email, dateOfBirth, gender) "
                            + "values (?, ?, ?, ?, ?)");

            preparedStatement.setString(1, user.getUsername());
            preparedStatement.setString(2, hashedPassword);
            preparedStatement.setString(3, user.getEmail());
            preparedStatement.setDate(4, java.sql.Date.valueOf(user.getDateOfBirth()));
            preparedStatement.setString(5, user.getGender());
            rowAdded = preparedStatement.executeUpdate() > 0; // If one row added, then rowAdded = true.
        }
        catch (SQLException e) {
            e.printStackTrace();
        }

        return rowAdded;
```

Figure 13.0

Figure 13.0 shows UserDao.java, an early portion of the Java class responsible for the CRUD process specifically for the user process. This class handles the Create, Read, Update, and Delete operations for user data in the database. It includes methods for adding new users, retrieving user information, updating existing user details, and deleting users from the system. The UserDao.java class interacts with the database through SQL queries and ensures that all user-related data management tasks are performed efficiently and securely.

User.java:

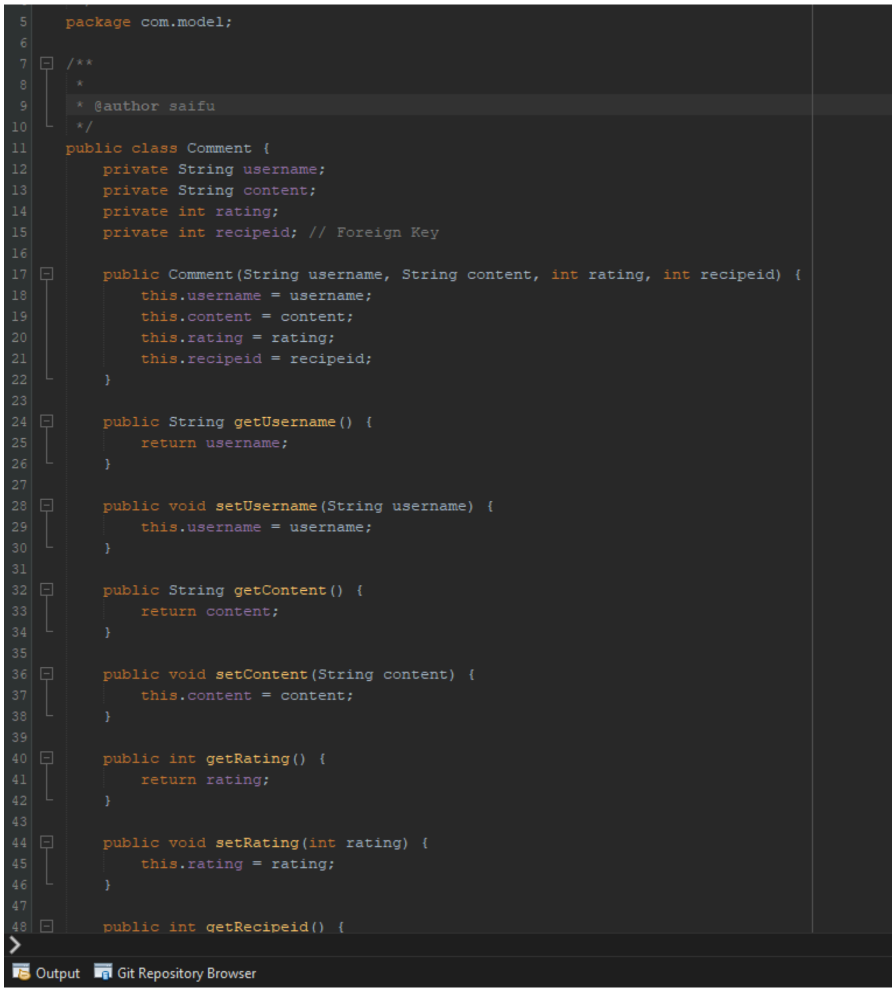


```java
package com.model;
import java.time.LocalDate;
/**
 *
 * @author Saiful
 */
public class User {

    private int userid;
    private String username;
    private String password;
    private String email;
    private LocalDate dateOfBirth;
    private String gender;
    private String description;
    private String picturePath;

    public User() {
    }


    public User(String username, String password, String email, LocalDate dateOfBirth, String gender) {
        this.username = username;
        this.password = password;
        this.email = email;
        this.dateOfBirth = dateOfBirth;
        this.gender = gender;
    }

    public User(int userid, String username, String password, String email, LocalDate dateOfBirth, String gender) {
        this.userid = userid;
        this.username = username;
        this.password = password;
        this.email = email;
        this.dateOfBirth = dateOfBirth;
        this.gender = gender;
    }

    public int getUserid() {
        return userid;
    }

    public void setUserid(int userid) {
        this.userid = userid;
```

Figure 14.0

Figure 14.0 shows User.java, which serves as the JavaBean or model for the User class. The image illustrates the class's attributes, constructors, and possibly methods that define its behavior and properties. This includes fields such as username, password, email, and possibly others like user ID or role. The constructors depicted in the figure initialize these attributes upon instantiation, ensuring that each User object is properly initialized with relevant data. Such a structure facilitates the encapsulation of user data, allowing for efficient management and manipulation within the application's logic.

Comment.java:



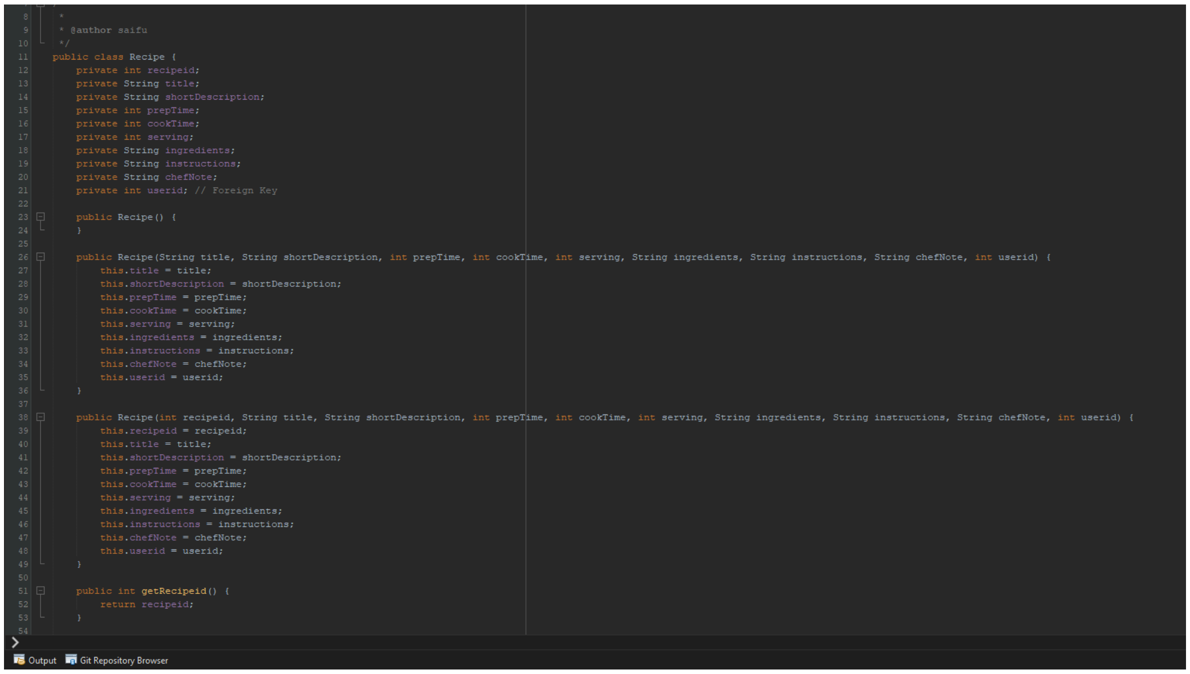
```java
package com.model;

/**
 *
 * @author saifu
 */
public class Comment {
    private String username;
    private String content;
    private int rating;
    private int recipeid; // Foreign Key

    public Comment(String username, String content, int rating, int recipeid) {
        this.username = username;
        this.content = content;
        this.rating = rating;
        this.recipeid = recipeid;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

    public int getRating() {
        return rating;
    }

    public void setRating(int rating) {
        this.rating = rating;
    }

    public int getRecipeid() {
```

Figure 15.0

The Comment.java class serves as the JavaBean or model for handling comments within the application. Despite having its own Data Access Object (DAO) for database interactions, there isn't a dedicated controller or servlet specifically for managing comments. Instead, the functionality related to comments is integrated into the RecipeController.java class.

The recipeid attribute within the Comment.java class acts as a foreign key linking comments to the corresponding recipe entries in the database. This relationship ensures that each comment is associated with the correct recipe, facilitating seamless navigation and data retrieval for users interacting with the application

Recipe.java:



```java
 *
 * @author saifu
 */
public class Recipe {
    private int recipeid;
    private String title;
    private String shortDescription;
    private int prepTime;
    private int cookTime;
    private int serving;
    private String ingredients;
    private String instructions;
    private String chefNote;
    private int userid; // Foreign Key

    public Recipe() {
    }

    public Recipe(String title, String shortDescription, int prepTime, int cookTime, int serving, String ingredients, String instructions, String chefNote, int userid) {
        this.title = title;
        this.shortDescription = shortDescription;
        this.prepTime = prepTime;
        this.cookTime = cookTime;
        this.serving = serving;
        this.ingredients = ingredients;
        this.instructions = instructions;
        this.chefNote = chefNote;
        this.userid = userid;
    }

    public Recipe(int recipeid, String title, String shortDescription, int prepTime, int cookTime, int serving, String ingredients, String instructions, String chefNote, int userid) {
        this.recipeid = recipeid;
        this.title = title;
        this.shortDescription = shortDescription;
        this.prepTime = prepTime;
        this.cookTime = cookTime;
        this.serving = serving;
        this.ingredients = ingredients;
        this.instructions = instructions;
        this.chefNote = chefNote;
        this.userid = userid;
    }

    public int getRecipeid() {
        return recipeid;
    }

```

Figure 16.0

Figure 16.0 shows Comment.java where the attribute 'userid' is used as a foreign key to the 'users' table, establishing a relationship between Comment.java and the users who have posted comments. This ensures that each comment in the application is associated with a specific user, linking their identity and comments seamlessly within the database. By referencing the 'userid' from the 'users' table, the Comment.java class can retrieve additional information about the user who made the comment, such as their username, profile picture, or other relevant details, enhancing the context and usability of the comment feature within the application.

Contribution

Name	Contribution
Ahmad Syaiful Hafizi	**Project Report:** i. Executive Summary ii. Use Case Diagram iii. Sample of Interface iv. Sample of Code **System Development:** i. Login process ii. Recipe process iii. Comment process iv. Newsletter process
Hidayatul Ain	**Project Report:** i. User Traceability Matrix ii. MVC Model iii. Conclusion iv. References **System Development:** i. ii.
NurSyaza Amira	**Project Report:** i. Class Diagram ii. ER Diagram iii. Sample of Interface (Explanation) iv. Sample of Code (Explanation) **System Development:** i. Interface

Conclusion

Cooking Delight represents a significant step forward in the realm of culinary knowledge management by providing a comprehensive, user-friendly platform tailored for cooking enthusiasts of all levels. The Recipe Archive Management System underpins this platform, employing advanced software design principles to deliver a seamless and efficient user experience.

Through utilization of the Model-View-Controller (MVC) architectural design, the system guarantees a distinct division of responsibilities, leading to improved code structure, expandability, and ease of maintenance. Each element - Model, View, and Controller - has a vital function in upholding the system's integrity and functionality. The Model controls data and business logic using clearly defined JavaBeans and Data Access Objects (DAOs). The View consists of JSP pages that offer a user-friendly interface, while the Controller handles user input to facilitate communication between the Model and View.

The MVC framework efficiently handles important aspects of Cooking Delight, including secure user management, a strong recipe repository, engaging comment sections, and a subscription-based newsletter. The User MVC model is responsible for managing user authentication and profile information to provide secure access to custom features. The Recipe MVC model offers an organized method for handling culinary content, allowing users to easily generate, find, and explore recipes. The Comment MVC model promotes community engagement through user sharing of advice and opinions, while the Newsletter MVC model ensures user retention through consistent updates on fresh content and features.

The Class Diagram and ERD provide a visual representation of the system's structure, emphasizing the connections and exchanges among different entities. These illustrations offer a graphical portrayal of the organization's framework, promoting a thorough comprehension of the flow of information and interconnections.

In general, Cooking Delight effectively solves the issue of scattered and untrustworthy culinary information by providing a centralized, feature-packed platform. It blends strong back-end data handling with an attractive user interface, supporting a lively group of cooking fans. The system's design addresses both current user needs and allows for future growth, ensuring Cooking Delight remains a valuable resource for home cooks, professional chefs, and food enthusiasts.

References

WhatisKnowledgeManagementSystem?-eGain.(2024,June24).eGain.https://www.egain.com/wha-is-knowledge-management-system

Fonseca, L. (2024, June 20). *Class Diagram: Examples, How to Make, Symbols, Benefits*. Venngage.https://venngage.com/blog/class-diagram/#:~:text=A%20class%20diagram%20is%20a,name%2C%20attributes%2C%20and%20operations.

Hanna, K. T., & Biscobing, J. (2024, March 26). *entity relationship diagram (ERD)*. Data Management.https://www.techtarget.com/searchdatamanagement/definition/entity-relationship-diagram-ERD

Sheldon, R. (2023, September 12). *model-view-controller (MVC)*. WhatIs. https://www.techtarget.com/whatis/definition/model-view-controller-MVC#:~:text=In%20programming%2C%20model%2Dview%2D,a%20specific%20set%20of%20tasks.