Module Code & Module Title
CS5003NI Data Structure and Specialist Programming

Assessment Type

30% Individual Coursework 2

Semester

2023-24 Autumn

Student Name: Aakarshan Khadka

Project Title: DaalBhaatGym

London Met ID: 22085855

College ID: np01ai4s230002

Assignment Due Date: Friday, May 10, 2024

Assignment Submission Date: Friday, May 10, 2024

Submitted to: Mr. Prithivi Maharjan

Word Count: 3283

Contents

1. Introduction

1.1. Introduction to DaalBhaatGym

DaalBhaatGym is a web-based dynamic system that is designed specifically tailored for the Nepalese Gym Enthusiasts. It will be designed to enhance the experience of the fellow gym rats and streamline the process by integrating technology making the process quicker, more reliable, and efficient. It aims to provide a smooth user-friendly platform for managing and tracking the members and membership plans of the gym. It offers features ensuring ease of use and quality of life for both the Gym owner and its users.

1.2. Aim

The main aim of this coursework is to design and develop a fully functional and efficient dynamic webpage tailored to DaalBhaatGym and its gym rats. We aim to provide features that make a positive change in the experience for both the admin and the users.

1.3. Objectives

- To create a login logout system, with personal sessions and cookies to greatly customize the user experience.
- Create a robust database system that stores data's safely to maintain user privacy and data integrity.
- Provide user features to customize and update their user profile with profile pictures.
- Develop multiple pages with CRUD database operations.
- Enable scalability and flexibility to accommodate for future growth of DaalBhaatGym.
- Design a straightforward and user-friendly system, that's easy to learn and use.
- Give the admin privilege to view the gym's data and manage membership.

1.4. List Of Features

- User Authentication: Secure login system for both admins and users.
- Password Encryption: Passwords will be encrypted to main data security.
- Personalized Sessions: User sessions will be kept making for a personalized experience.
- Membership Management: CRUD operations for managing membership details.
- User Profiles: User can view their personal profiles.

2. Database Design

2.1. Database Structure:

Admin(adminID(PK),username,password,firstName,lastName,email,phoneNumber)

Plans(planID(PK),planDurationDays,planPrice,planDsecription,adminID(FK))

User(userID(PK),firstName,lastName,username,email,phoneNumber,image)

2.2. Entity Relationship Diagram (ERD)

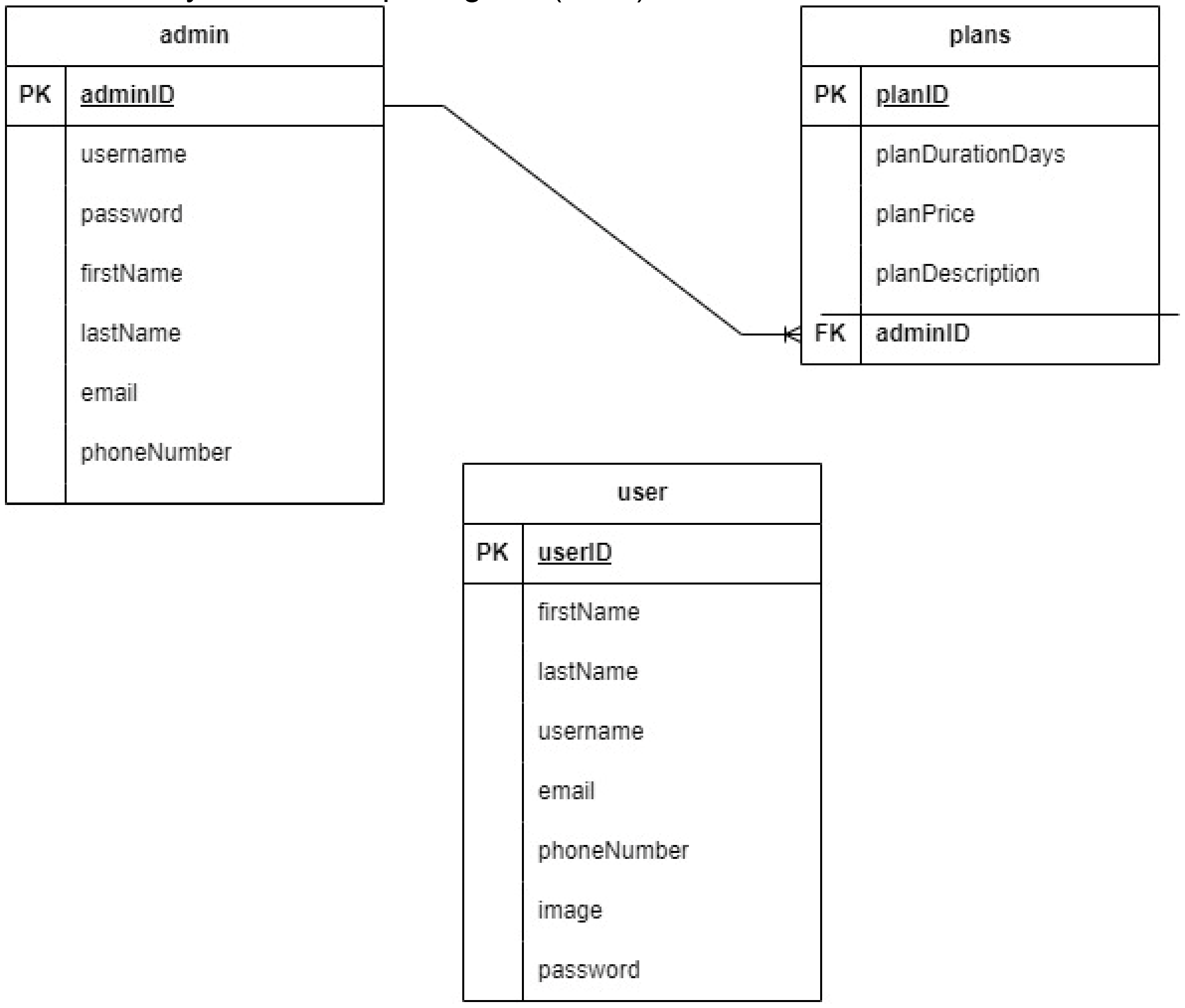


Figure 1: ER Diagram

2.3. Database:

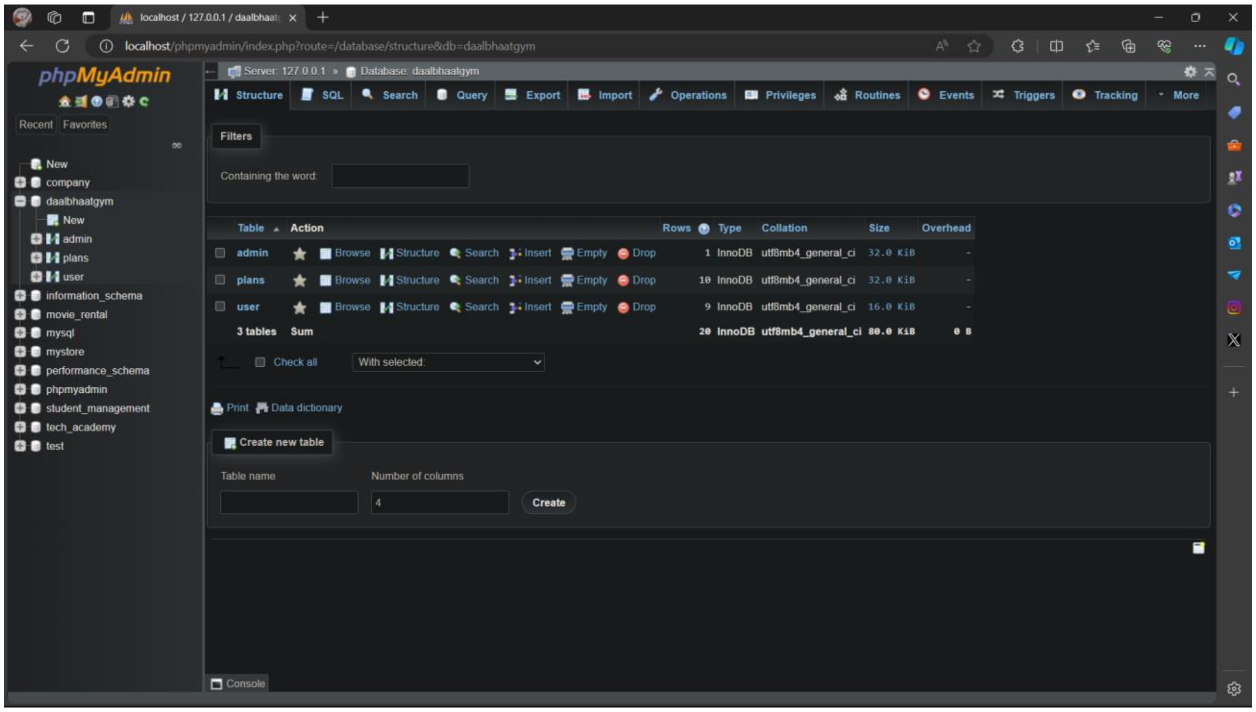
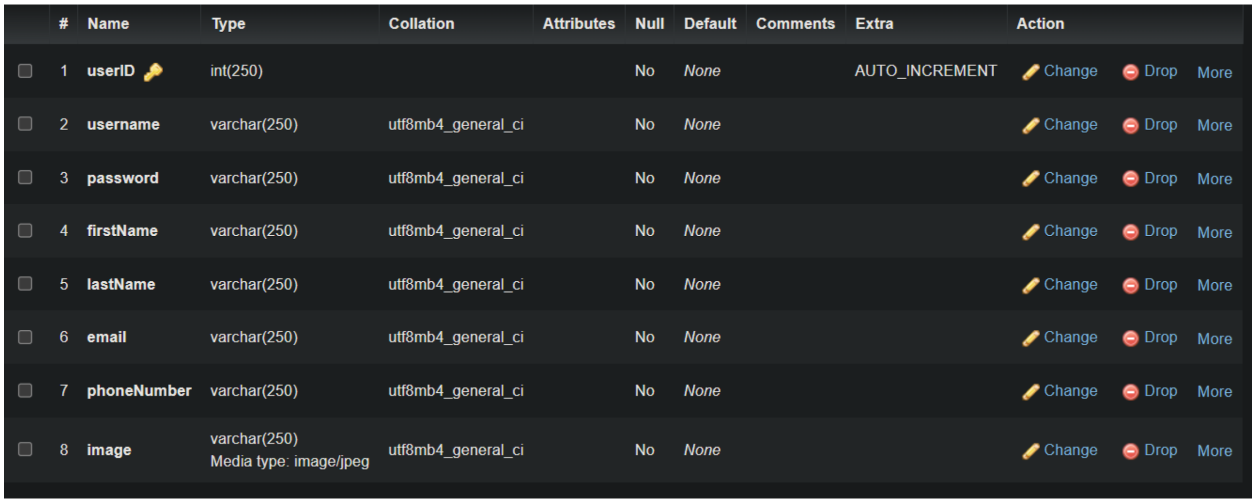


	Table	Action	Rows	Type	Collation	Size	Overhead
☐	admin	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	32.0 KiB	-
☐	plans	Browse Structure Search Insert Empty Drop	10	InnoDB	utf8mb4_general_ci	32.0 KiB	-
☐	user	Browse Structure Search Insert Empty Drop	9	InnoDB	utf8mb4_general_ci	16.0 KiB	-
	3 tables	Sum	20	InnoDB	utf8mb4_general_ci	80.0 KiB	0 B

Figure 2: Overall Database

2.4 Table Design

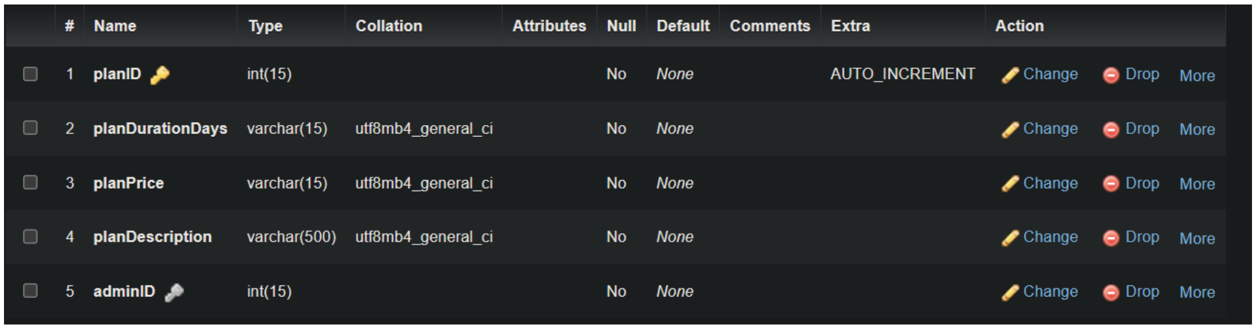
2.3.1. User



	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
☐	1	userID	int(250)			No	*None*		AUTO_INCREMENT	Change Drop More
☐	2	username	varchar(250)	utf8mb4_general_ci		No	*None*			Change Drop More
☐	3	password	varchar(250)	utf8mb4_general_ci		No	*None*			Change Drop More
☐	4	firstName	varchar(250)	utf8mb4_general_ci		No	*None*			Change Drop More
☐	5	lastName	varchar(250)	utf8mb4_general_ci		No	*None*			Change Drop More
☐	6	email	varchar(250)	utf8mb4_general_ci		No	*None*			Change Drop More
☐	7	phoneNumber	varchar(250)	utf8mb4_general_ci		No	*None*			Change Drop More
☐	8	image	varchar(250) Media type: image/jpeg	utf8mb4_general_ci		No	*None*			Change Drop More

Figure 3: User Table

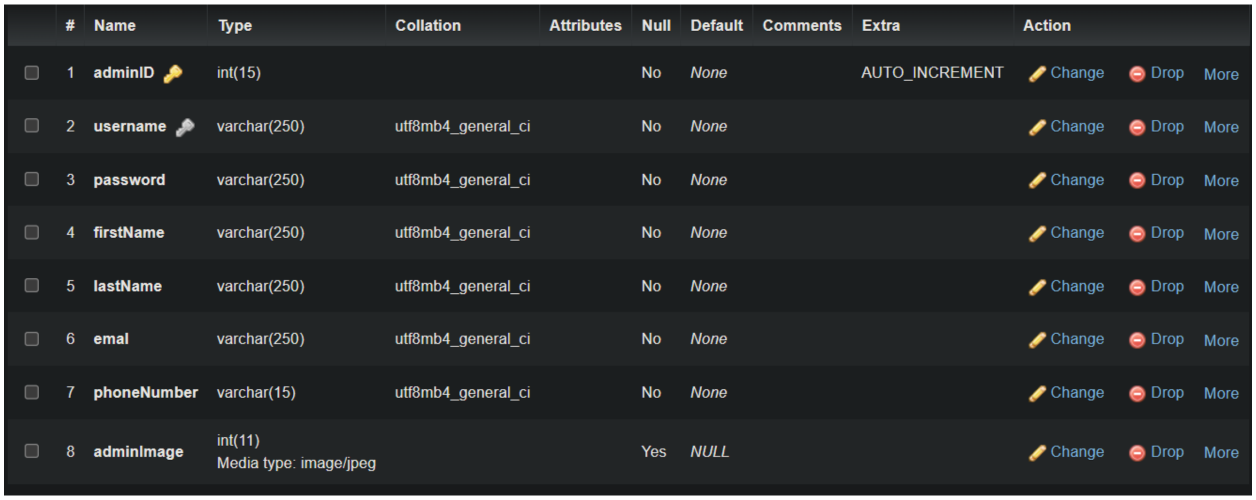
2.3.2. Plans



	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action		
	1	planID	int(15)			No	*None*		AUTO_INCREMENT	Change	Drop	More
	2	planDurationDays	varchar(15)	utf8mb4_general_ci		No	*None*			Change	Drop	More
	3	planPrice	varchar(15)	utf8mb4_general_ci		No	*None*			Change	Drop	More
	4	planDescription	varchar(500)	utf8mb4_general_ci		No	*None*			Change	Drop	More
	5	adminID	int(15)			No	*None*			Change	Drop	More

Figure 4: Plans Table

2.3.3. Admin



	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action		
	1	adminID	int(15)			No	*None*		AUTO_INCREMENT	Change	Drop	More
	2	username	varchar(250)	utf8mb4_general_ci		No	*None*			Change	Drop	More
	3	password	varchar(250)	utf8mb4_general_ci		No	*None*			Change	Drop	More
	4	firstName	varchar(250)	utf8mb4_general_ci		No	*None*			Change	Drop	More
	5	lastName	varchar(250)	utf8mb4_general_ci		No	*None*			Change	Drop	More
	6	emal	varchar(250)	utf8mb4_general_ci		No	*None*			Change	Drop	More
	7	phoneNumber	varchar(15)	utf8mb4_general_ci		No	*None*			Change	Drop	More
	8	adminImage	int(11) Media type: image/jpeg			Yes	*NULL*			Change	Drop	More

Figure 5: Admin Table

3. UI/UX Design

3.1. Wireframe:

3.1.1. Register Page:

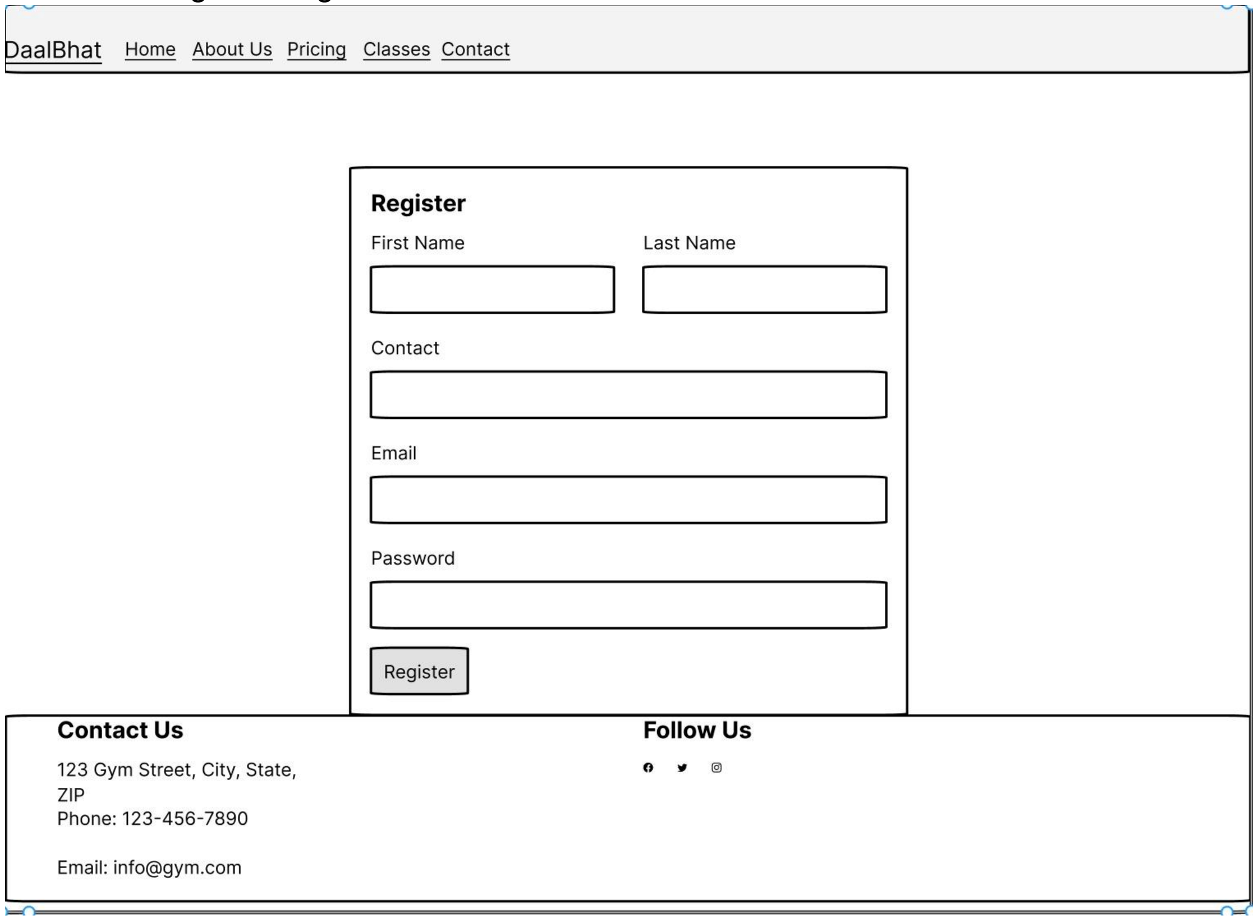


Figure 6:Register Wireframe

3.1.2. Login Page:

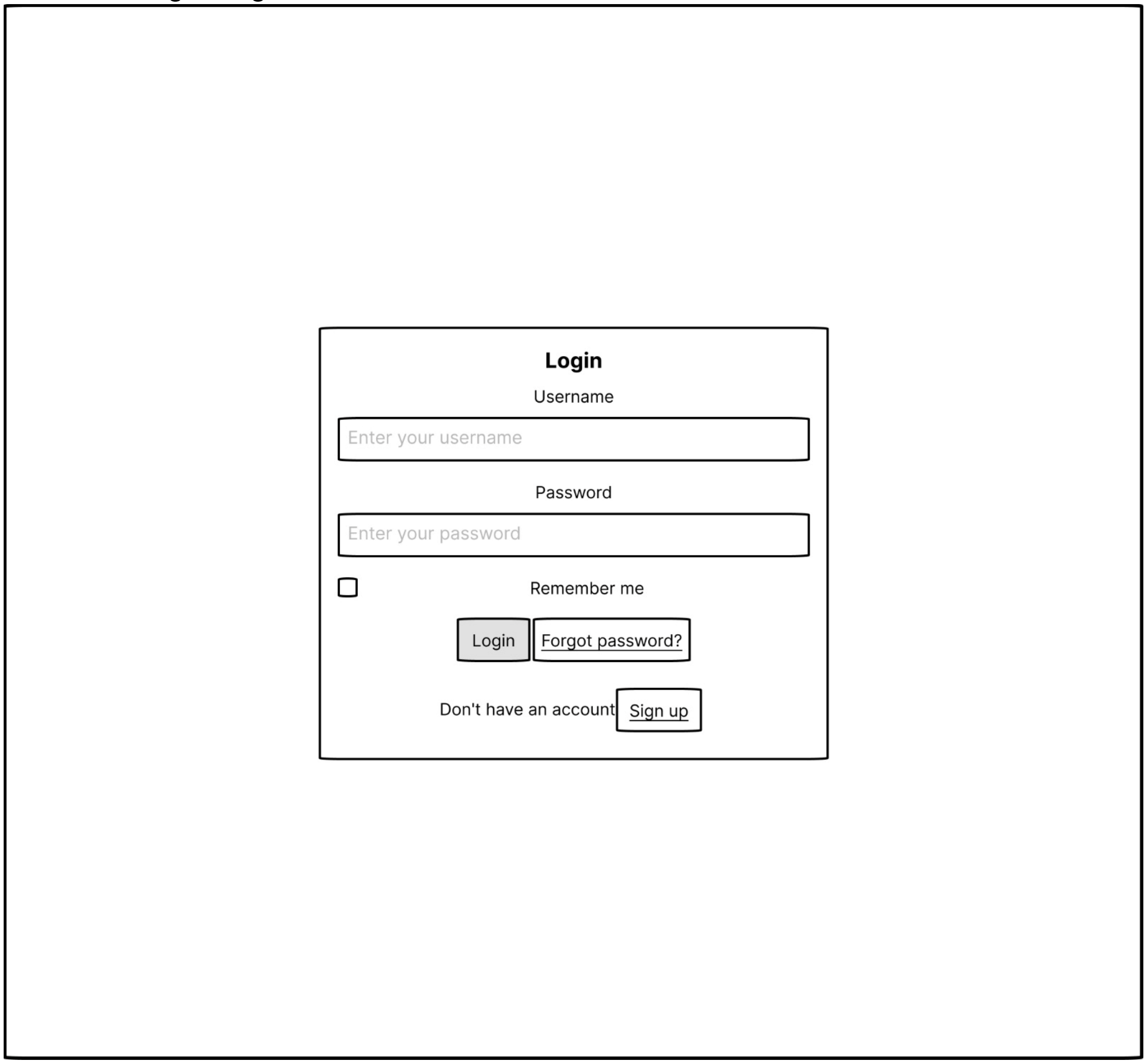


Figure 7:Wireframe Login

3.1.3. Home Page:

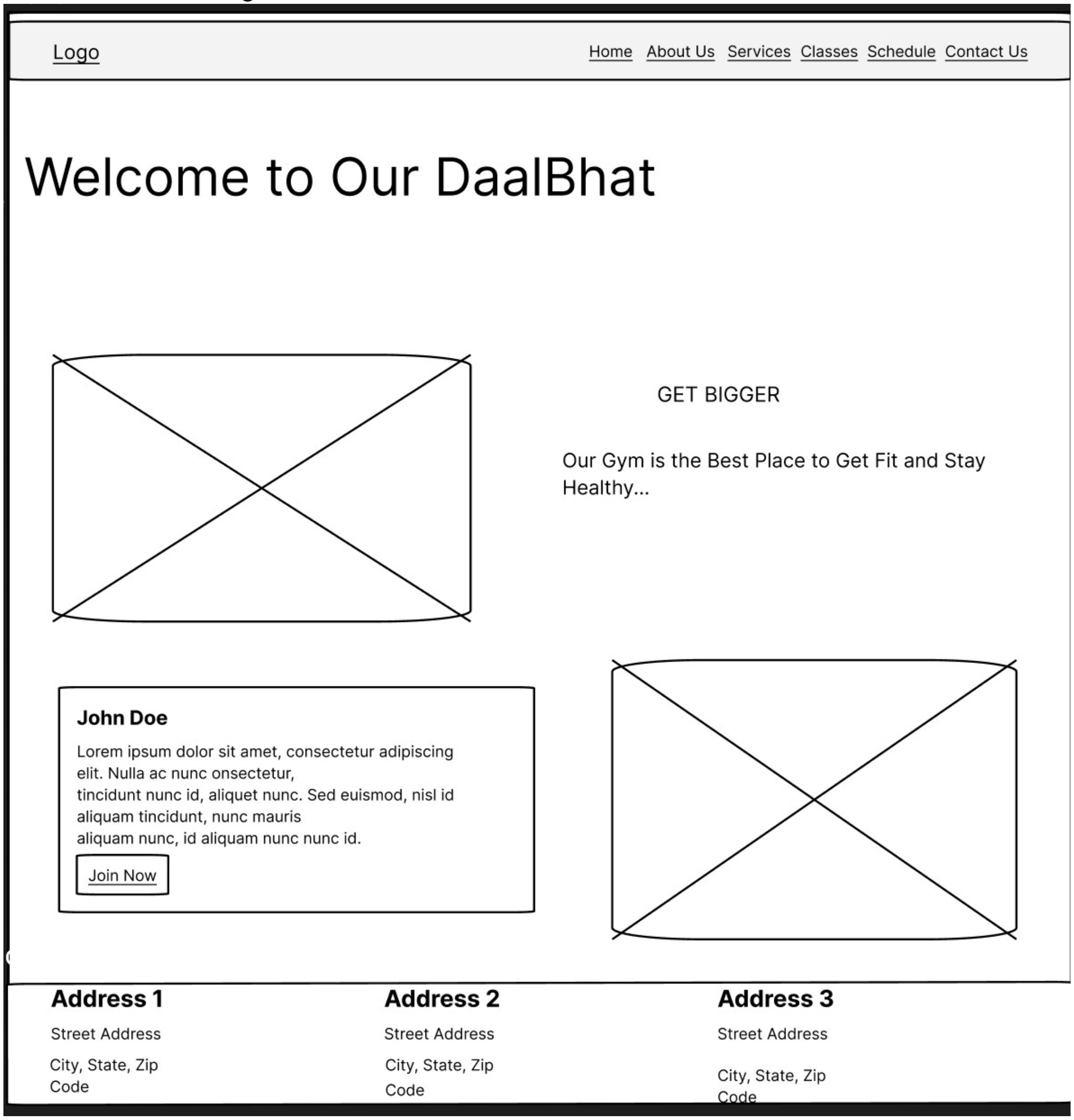

Logo
Home
About Us
Services
Classes
Schedule
Contact Us
Welcome to Our DaalBhat
GET BIGGER
Our Gym is the Best Place to Get Fit and Stay Healthy...
John Doe
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla ac nunc onsectetur, tincidunt nunc id, aliquet nunc. Sed euismod, nisl id aliquam tincidunt, nunc mauris aliquam nunc, id aliquam nunc nunc id.
Join Now
Address 1
Street Address
City, State, Zip Code
Address 2
Street Address
City, State, Zip Code
Address 3
Street Address
City, State, Zip Code

Figure 8:Home Wireframe

3.1.4. Plan Page:

3.1.5. Admin Page:

3.1.6. User Profile:

3.2. Actual Design

3.2.1. Register Page:

3.2.2. Login Page:

3.2.3. Home Page:

3.2.4. Plan Page:

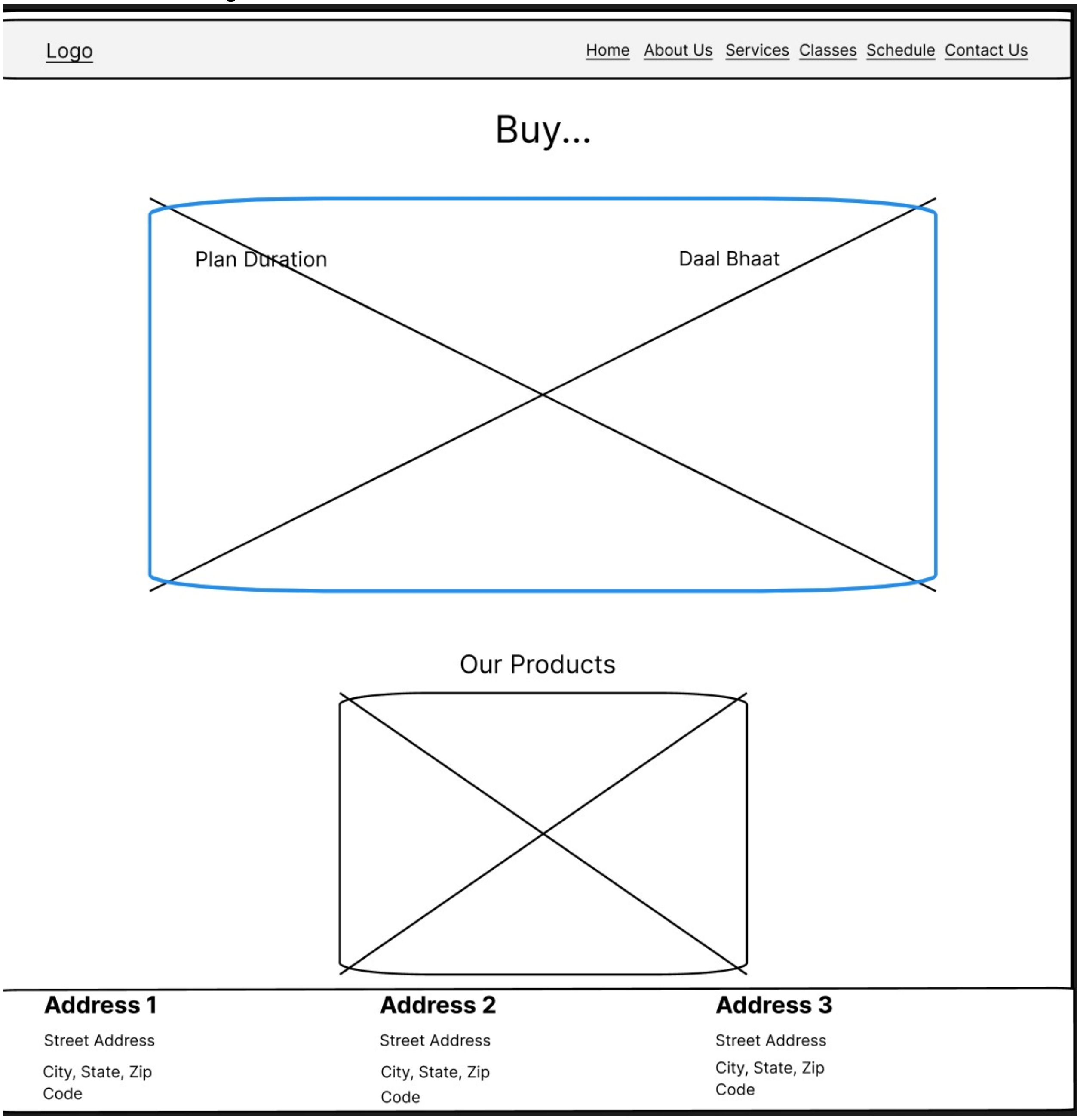


Figure 9:Wireframe plan page

3.2.5. Admin Page:

3.2.6. User Profile:

4. Class Diagram

4.1. Individual Class Diagram

4.1.1. Login Model

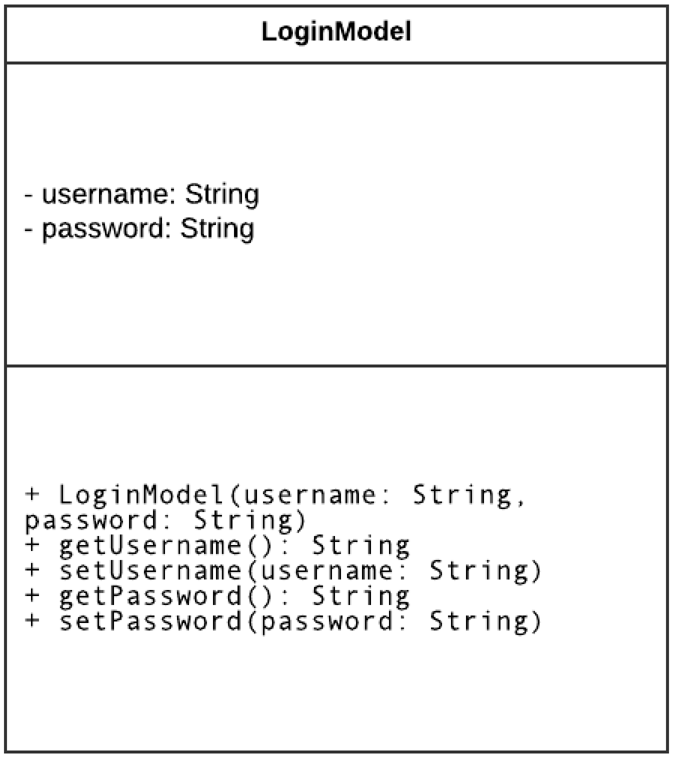
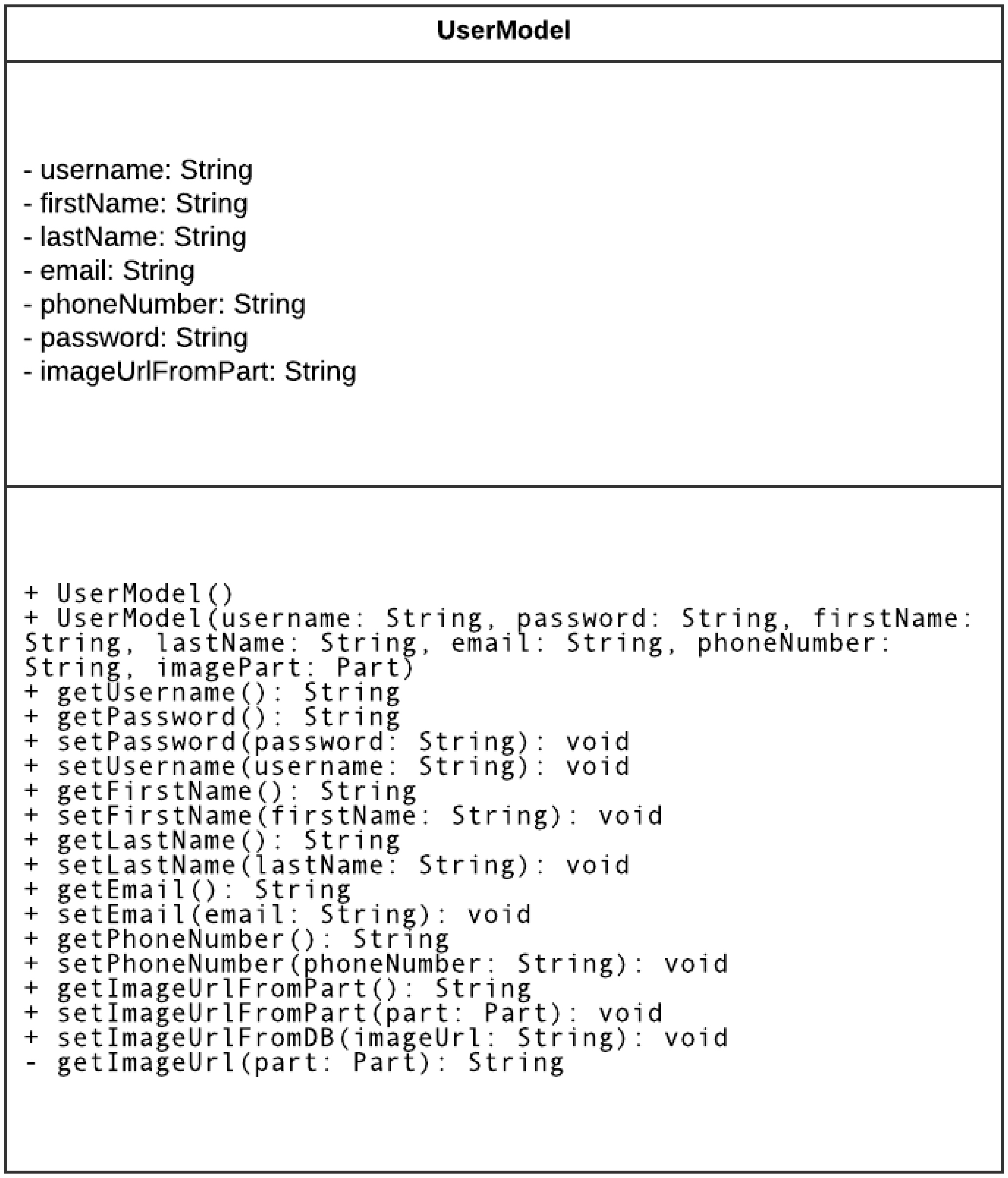

LoginModel
- username: String
- password: String
+ LoginModel(username: String, password: String)
+ getUsername(): String
+ setUsername(username: String)
+ getPassword(): String
+ setPassword(password: String)

Figure 10: Class Diagram: Login Model

4.1.2. User Model



```
- username: String
- firstName: String
- lastName: String
- email: String
- phoneNumber: String
- password: String
- imageUrlFromPart: String
```

```
+ UserModel()
+ UserModel(username: String, password: String, firstName:
String, lastName: String, email: String, phoneNumber:
String, imagePart: Part)
+ getUsername(): String
+ getPassword(): String
+ setPassword(password: String): void
+ setUsername(username: String): void
+ getFirstName(): String
+ setFirstName(firstName: String): void
+ getLastName(): String
+ setLastName(lastName: String): void
+ getEmail(): String
+ setEmail(email: String): void
+ getPhoneNumber(): String
+ setPhoneNumber(phoneNumber: String): void
+ getImageUrlFromPart(): String
+ setImageUrlFromPart(part: Part): void
+ setImageUrlFromDB(imageUrl: String): void
- getImageUrl(part: Part): String
```

Figure 11 Class Diagram: User Model

4.1.3. Plan Model

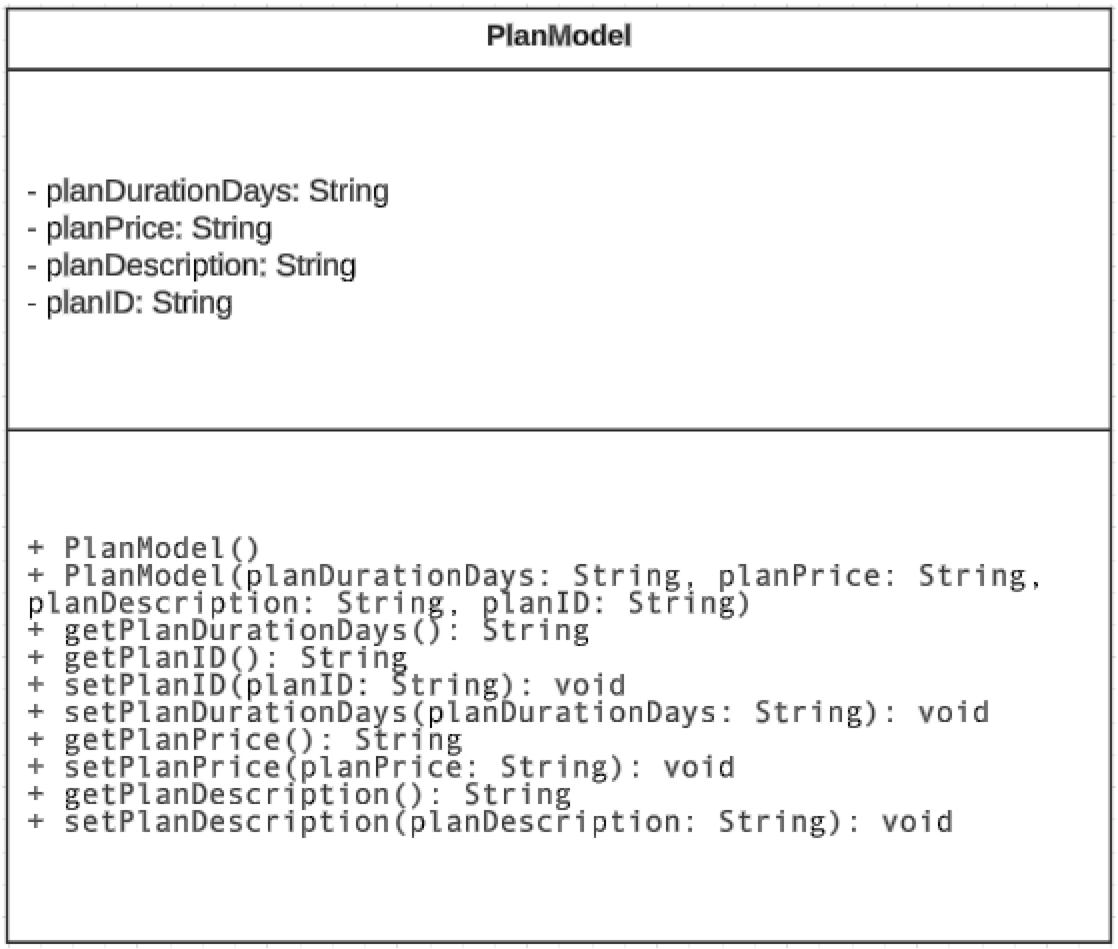

PlanModel
- planDurationDays: String
- planPrice: String
- planDescription: String
- planID: String
+ PlanModel()
+ PlanModel(planDurationDays: String, planPrice: String, planDescription: String, planID: String)
+ getPlanDurationDays(): String
+ getPlanID(): String
+ setPlanID(planID: String): void
+ setPlanDurationDays(planDurationDays: String): void
+ getPlanPrice(): String
+ setPlanPrice(planPrice: String): void
+ getPlanDescription(): String
+ setPlanDescription(planDescription: String): void

Figure 12: Class Diagram: Plan Model

4.1.4. PasswordEncryptionWithAes

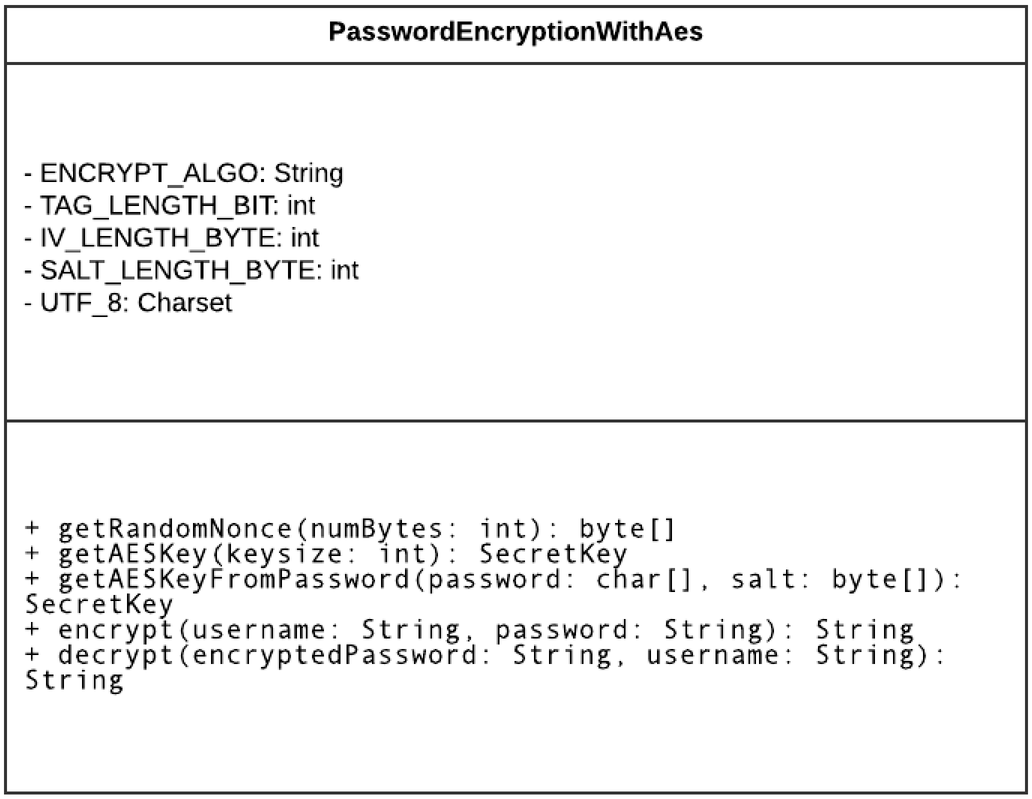
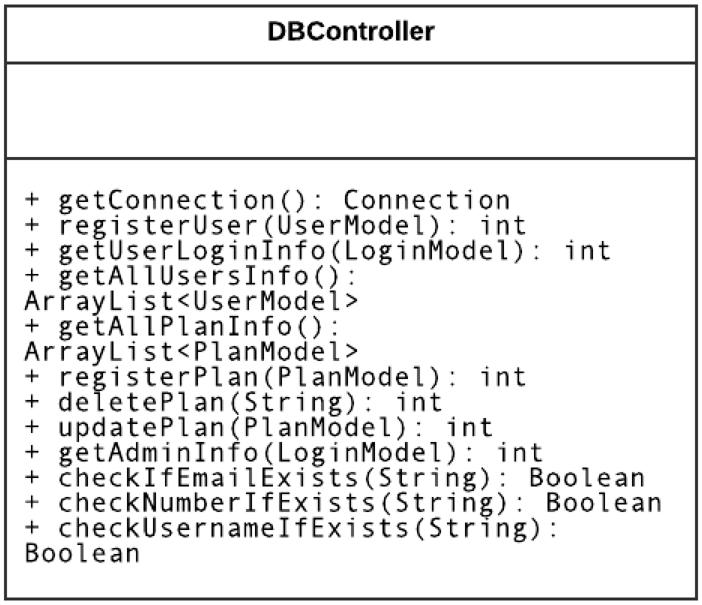

PasswordEncryptionWithAes
- ENCRYPT_ALGO: String
- TAG_LENGTH_BIT: int
- IV_LENGTH_BYTE: int
- SALT_LENGTH_BYTE: int
- UTF_8: Charset
+ getRandomNonce(numBytes: int): byte[]
+ getAESKey(keysize: int): SecretKey
+ getAESKeyFromPassword(password: char[], salt: byte[]): SecretKey
+ encrypt(username: String, password: String): String
+ decrypt(encryptedPassword: String, username: String): String

Figure 13: Class Diagram: Password Encryption with Aes

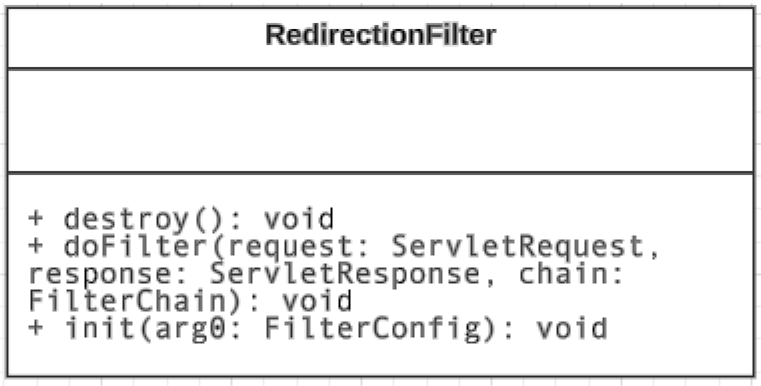
4.1.5. DB Controller



```
+ getConnection(): Connection
+ registerUser(UserModel): int
+ getUserLoginInfo(LoginModel): int
+ getAllUsersInfo():
ArrayList<UserModel>
+ getAllPlanInfo():
ArrayList<PlanModel>
+ registerPlan(PlanModel): int
+ deletePlan(String): int
+ updatePlan(PlanModel): int
+ getAdminInfo(LoginModel): int
+ checkIfEmailExists(String): Boolean
+ checkNumberIfExists(String): Boolean
+ checkUsernameIfExists(String):
Boolean
```

Figure 14: Class Diagram: Controller

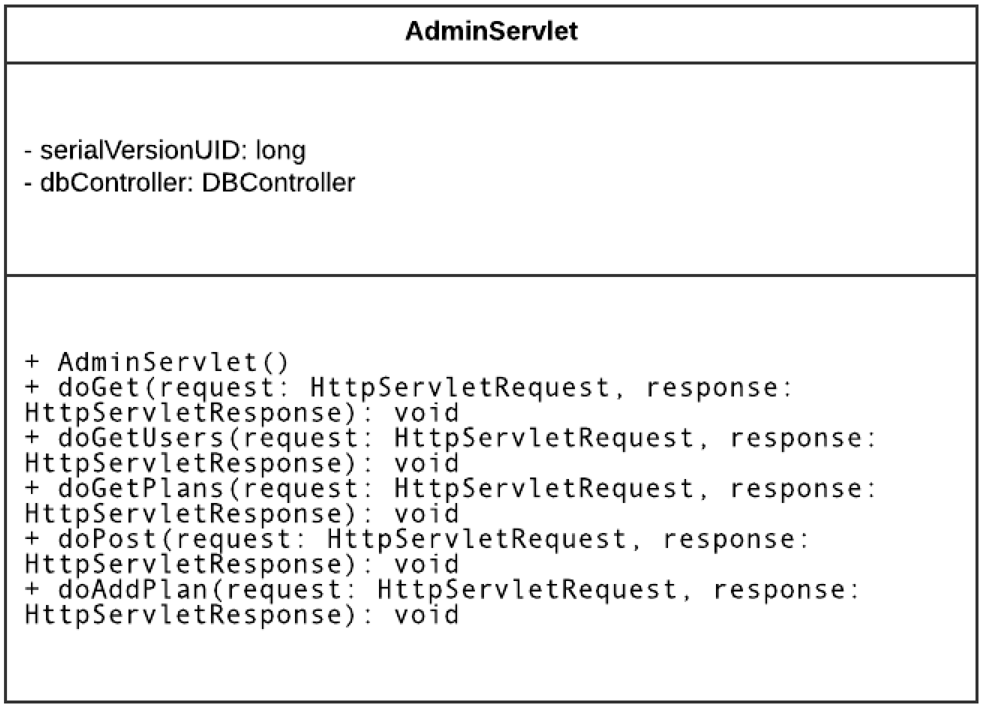
4.1.6. Redirection Filter



```
+ destroy(): void
+ doFilter(request: ServletRequest,
response: ServletResponse, chain:
FilterChain): void
+ init(arg0: FilterConfig): void
```

Figure 15: Class Diagram: Redirection Filter

4.1.7. Admin Servlet



```
- serialVersionUID: long
- dbController: DBController

+ AdminServlet()
+ doGet(request: HttpServletRequest, response:
HttpServletResponse): void
+ doGetUsers(request: HttpServletRequest, response:
HttpServletResponse): void
+ doGetPlans(request: HttpServletRequest, response:
HttpServletResponse): void
+ doPost(request: HttpServletRequest, response:
HttpServletResponse): void
+ doAddPlan(request: HttpServletRequest, response:
HttpServletResponse): void
```

Figure 16:Class Diagram: Admin Servlet

4.1.8. Home Servlet

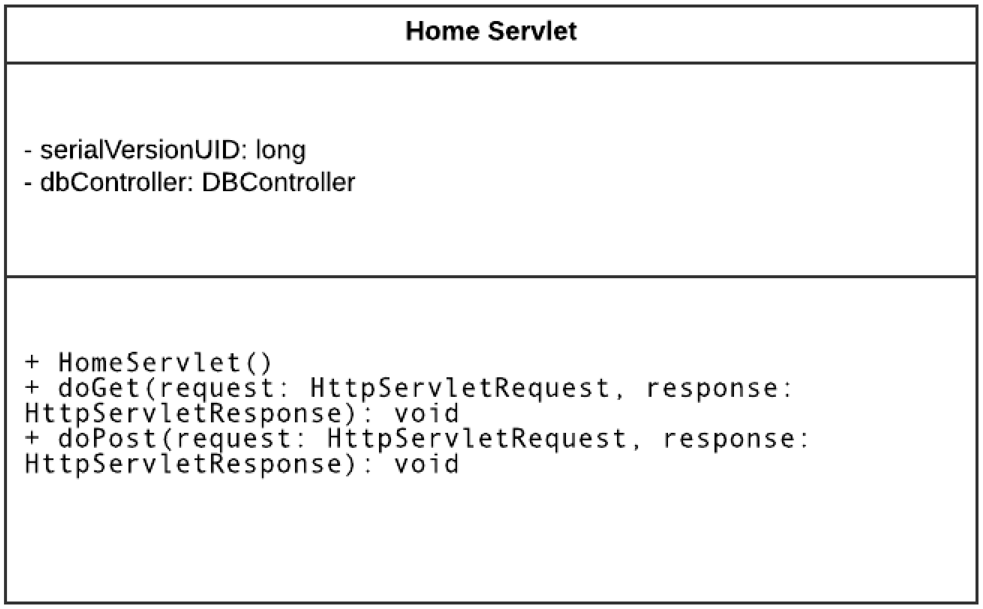

Home Servlet
- serialVersionUID: long
- dbController: DBController
+ HomeServlet()
+ doGet(request: HttpServletRequest, response: HttpServletResponse): void
+ doPost(request: HttpServletRequest, response: HttpServletResponse): void

Figure 17:Class Diagram: Home Servlet

4.1.9. Login Servlet

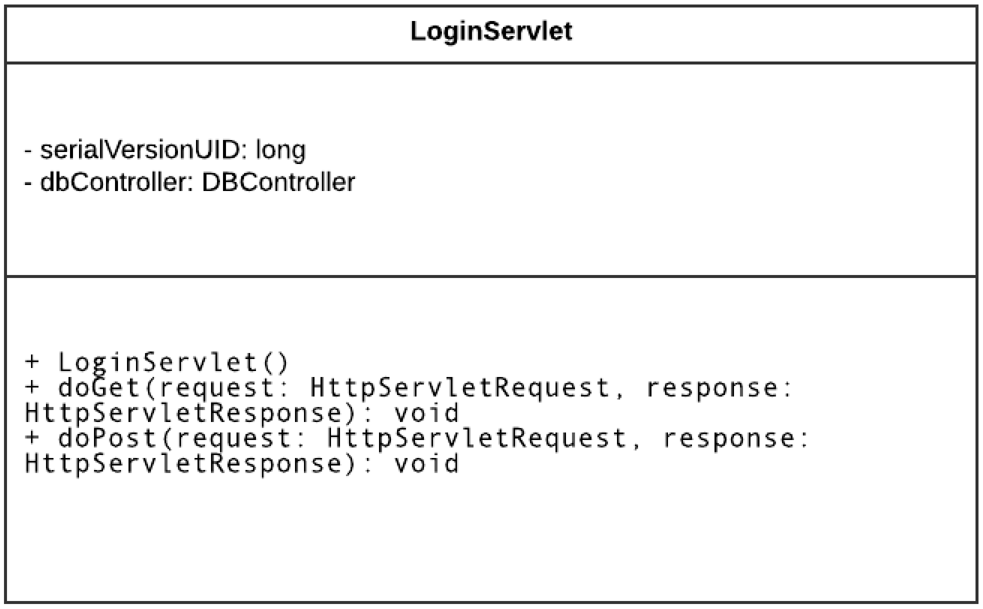

LoginServlet
- serialVersionUID: long
- dbController: DBController
+ LoginServlet()
+ doGet(request: HttpServletRequest, response: HttpServletResponse): void
+ doPost(request: HttpServletRequest, response: HttpServletResponse): void

Figure 18:Class Diagram: Login Servlet

4.1.10. Logout Servlet

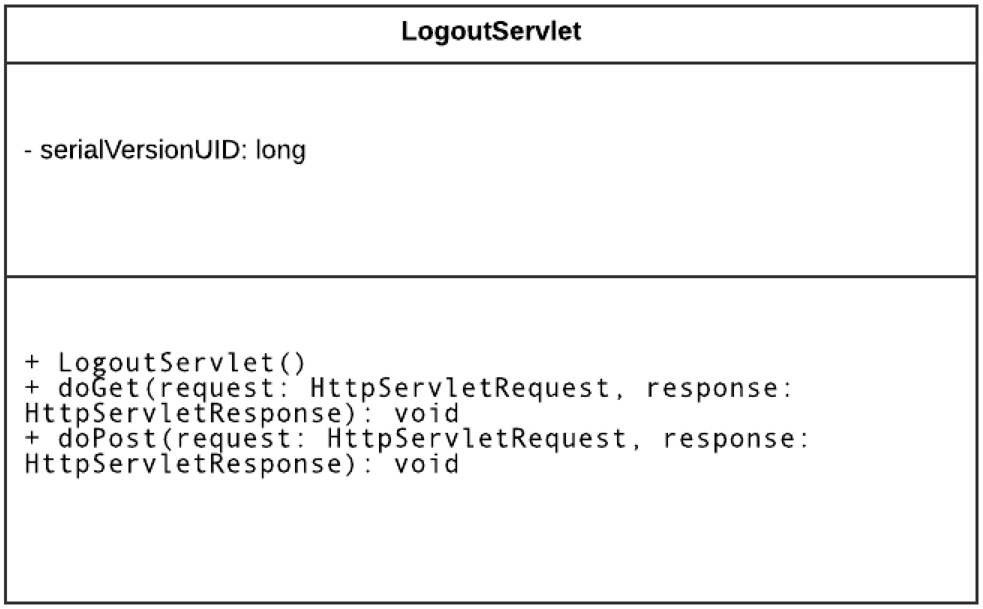

LogoutServlet
- serialVersionUID: long
+ LogoutServlet()
+ doGet(request: HttpServletRequest, response: HttpServletResponse): void
+ doPost(request: HttpServletRequest, response: HttpServletResponse): void

Figure 19:Class Diagram:Logout Servlet

4.1.11. Manage Plan Servlet

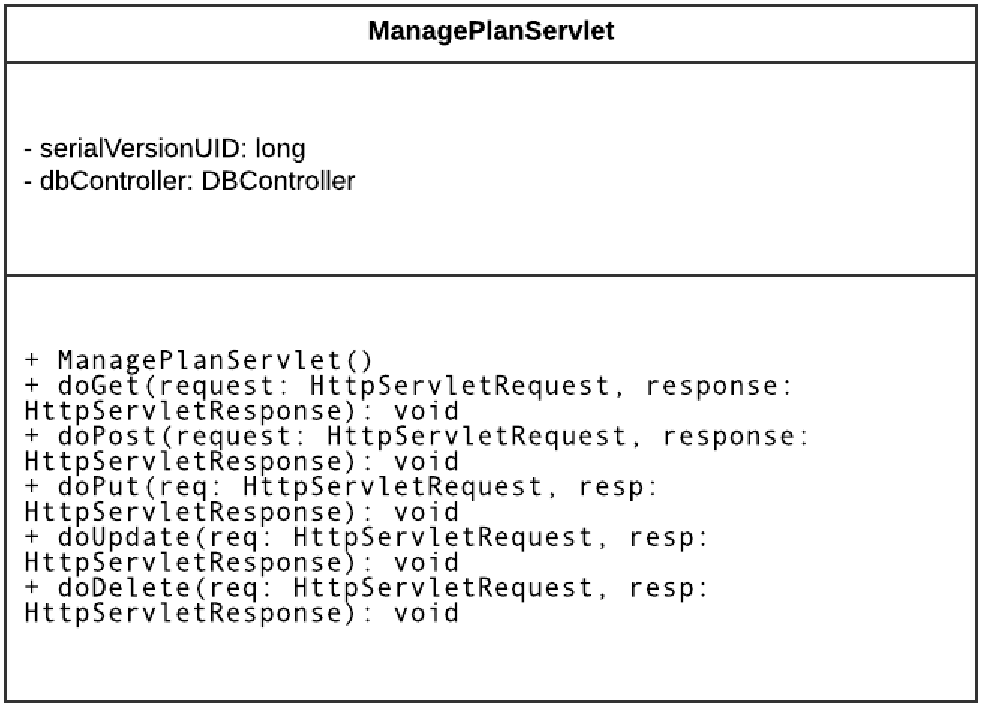

ManagePlanServlet
- serialVersionUID: long
- dbController: DBController
+ ManagePlanServlet()
+ doGet(request: HttpServletRequest, response: HttpServletResponse): void
+ doPost(request: HttpServletRequest, response: HttpServletResponse): void
+ doPut(req: HttpServletRequest, resp: HttpServletResponse): void
+ doUpdate(req: HttpServletRequest, resp: HttpServletResponse): void
+ doDelete(req: HttpServletRequest, resp: HttpServletResponse): void

Figure 20:Class Diagram: Manage Plan Servlet

4.1.12. Register Servlet

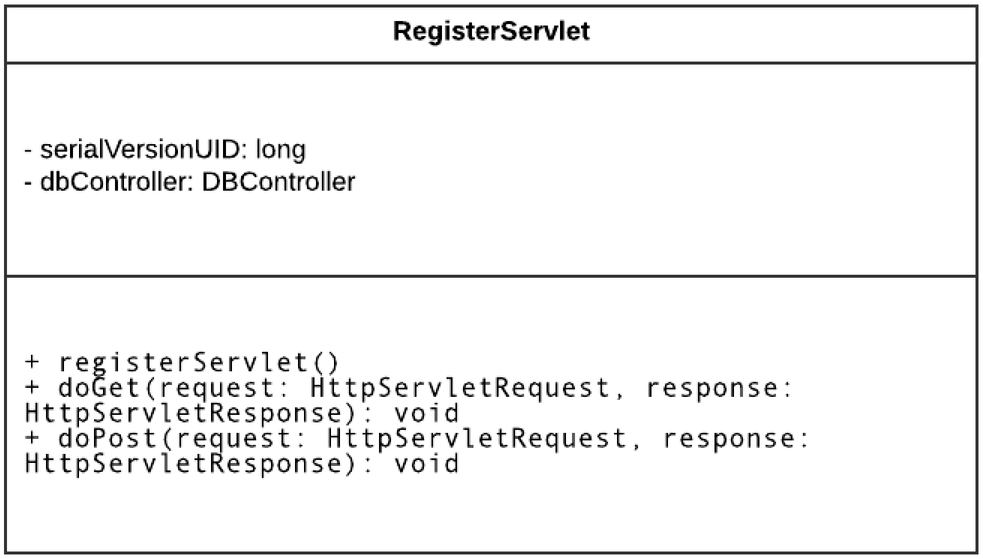

RegisterServlet
- serialVersionUID: long
- dbController: DBController
+ registerServlet()
+ doGet(request: HttpServletRequest, response: HttpServletResponse): void
+ doPost(request: HttpServletRequest, response: HttpServletResponse): void

Figure 21:Class Diagram: Register Servlet

4.1.13. Plan Servlet

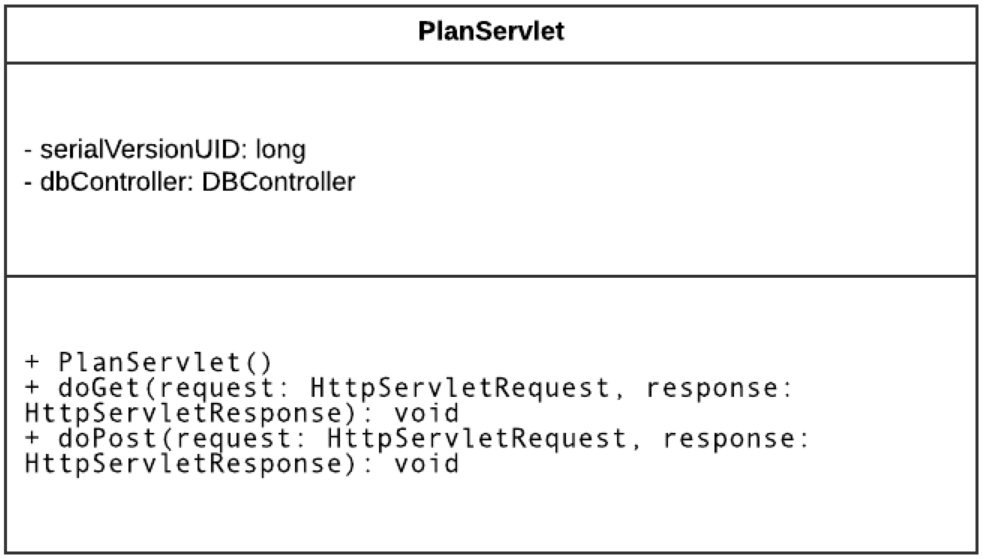

PlanServlet
- serialVersionUID: long
- dbController: DBController
+ PlanServlet()
+ doGet(request: HttpServletRequest, response: HttpServletResponse): void
+ doPost(request: HttpServletRequest, response: HttpServletResponse): void

Figure 22:Class Diagram: Plan Servlet

4.1.14. String Utils

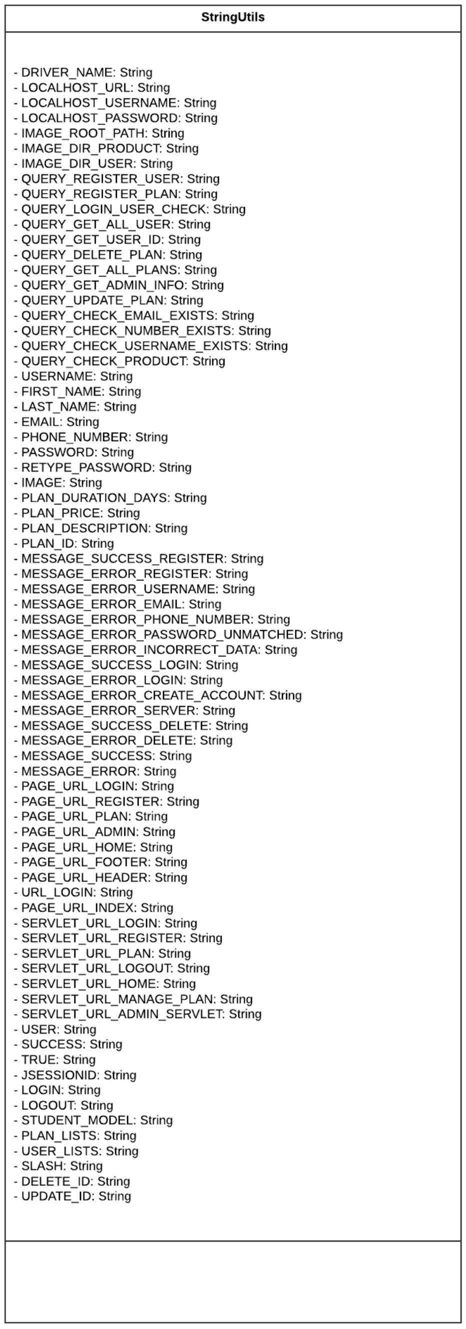

StringUtils
- DRIVER_NAME: String
- LOCALHOST_URL: String
- LOCALHOST_USERNAME: String
- LOCALHOST_PASSWORD: String
- IMAGE_ROOT_PATH: String
- IMAGE_DIR_PRODUCT: String
- IMAGE_DIR_USER: String
- QUERY_REGISTER_USER: String
- QUERY_REGISTER_PLAN: String
- QUERY_LOGIN_USER_CHECK: String
- QUERY_GET_ALL_USER: String
- QUERY_GET_USER_ID: String
- QUERY_DELETE_PLAN: String
- QUERY_GET_ALL_PLANS: String
- QUERY_GET_ADMIN_INFO: String
- QUERY_UPDATE_PLAN: String
- QUERY_CHECK_EMAIL_EXISTS: String
- QUERY_CHECK_NUMBER_EXISTS: String
- QUERY_CHECK_USERNAME_EXISTS: String
- QUERY_CHECK_PRODUCT: String
- USERNAME: String
- FIRST_NAME: String
- LAST_NAME: String
- EMAIL: String
- PHONE_NUMBER: String
- PASSWORD: String
- RETYPE_PASSWORD: String
- IMAGE: String
- PLAN_DURATION_DAYS: String
- PLAN_PRICE: String
- PLAN_DESCRIPTION: String
- PLAN_ID: String
- MESSAGE_SUCCESS_REGISTER: String
- MESSAGE_ERROR_REGISTER: String
- MESSAGE_ERROR_USERNAME: String
- MESSAGE_ERROR_EMAIL: String
- MESSAGE_ERROR_PHONE_NUMBER: String
- MESSAGE_ERROR_PASSWORD_UNMATCHED: String
- MESSAGE_ERROR_INCORRECT_DATA: String
- MESSAGE_SUCCESS_LOGIN: String
- MESSAGE_ERROR_LOGIN: String
- MESSAGE_ERROR_CREATE_ACCOUNT: String
- MESSAGE_ERROR_SERVER: String
- MESSAGE_SUCCESS_DELETE: String
- MESSAGE_ERROR_DELETE: String
- MESSAGE_SUCCESS: String
- MESSAGE_ERROR: String
- PAGE_URL_LOGIN: String
- PAGE_URL_REGISTER: String
- PAGE_URL_PLAN: String
- PAGE_URL_ADMIN: String
- PAGE_URL_HOME: String
- PAGE_URL_FOOTER: String
- PAGE_URL_HEADER: String
- URL_LOGIN: String
- PAGE_URL_INDEX: String
- SERVLET_URL_LOGIN: String
- SERVLET_URL_REGISTER: String
- SERVLET_URL_PLAN: String
- SERVLET_URL_LOGOUT: String
- SERVLET_URL_HOME: String
- SERVLET_URL_MANAGE_PLAN: String
- SERVLET_URL_ADMIN_SERVLET: String
- USER: String
- SUCCESS: String
- TRUE: String
- JSESSIONID: String
- LOGIN: String
- LOGOUT: String
- STUDENT_MODEL: String
- PLAN_LISTS: String
- USER_LISTS: String
- SLASH: String
- DELETE_ID: String
- UPDATE_ID: String

Figure 23:Class Diagram: String Utils

4.1.15. Validation Utils

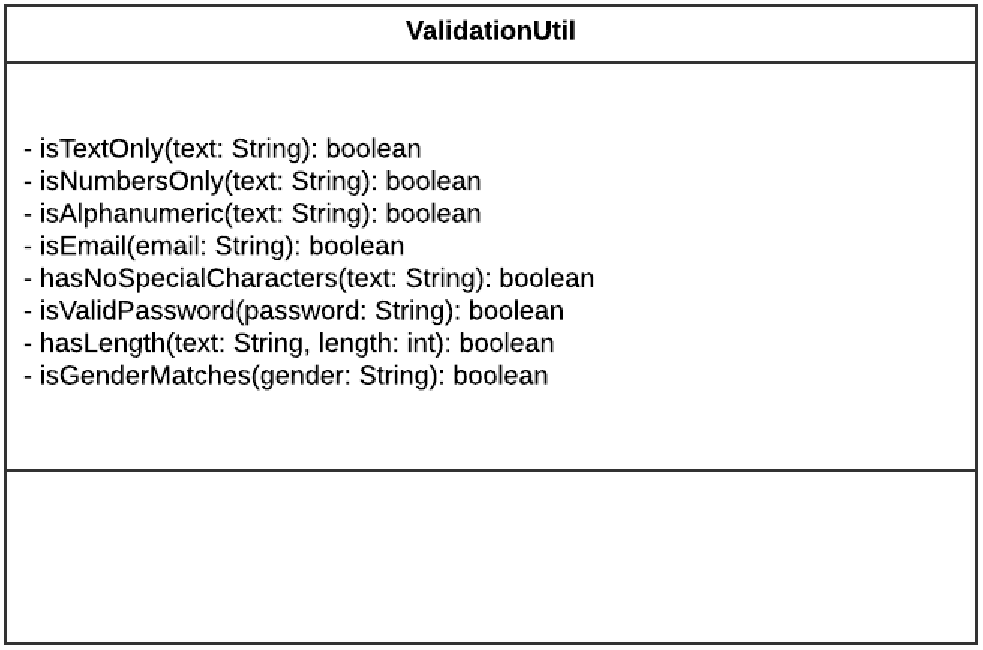

ValidationUtil
- isTextOnly(text: String): boolean
- isNumbersOnly(text: String): boolean
- isAlphanumeric(text: String): boolean
- isEmail(email: String): boolean
- hasNoSpecialCharacters(text: String): boolean
- isValidPassword(password: String): boolean
- hasLength(text: String, length: int): boolean
- isGenderMatches(gender: String): boolean

Figure 24:Class Diagram: Validation Util

4.2. Final Class Diagram

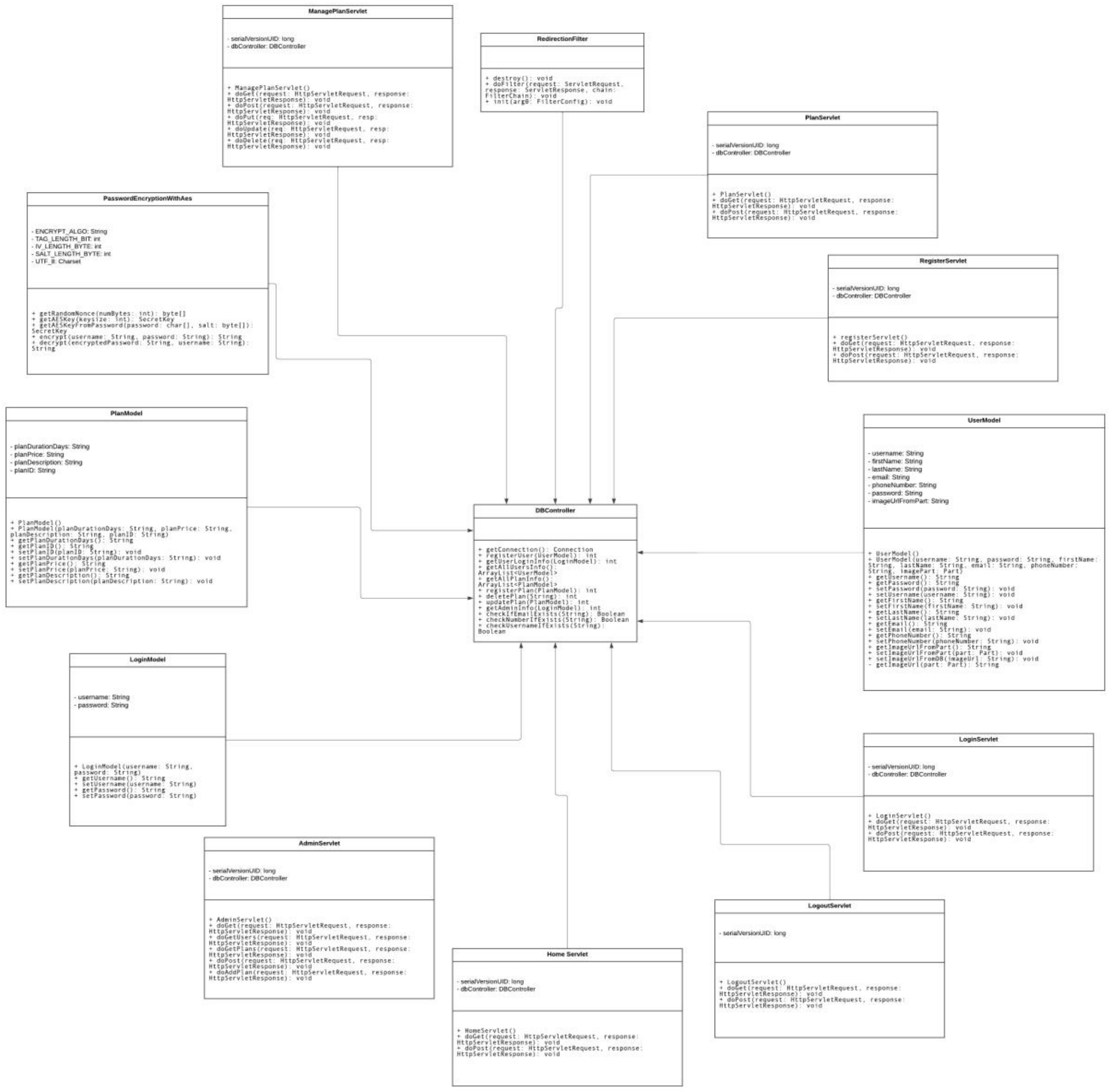

ManagePlanServlet
- serialVersionUID: long
- dbController: DBController
+ ManagePlanServlet()
+ doGet(request: HttpServletRequest, response: HttpServletResponse): void
+ doPost(request: HttpServletRequest, response: HttpServletResponse): void
+ doPut(req: HttpServletRequest, resp: HttpServletResponse): void
+ doUpdate(req: HttpServletRequest, resp: HttpServletResponse): void
+ doDelete(req: HttpServletRequest, resp: HttpServletResponse): void
RedirectionFilter
+ destroy(): void
+ doFilter(request: ServletRequest, response: ServletResponse, chain: FilterChain): void
+ init(arg0: FilterConfig): void
PlanServlet
- serialVersionUID: long
- dbController: DBController
+ PlanServlet()
+ doGet(request: HttpServletRequest, response: HttpServletResponse): void
+ doPost(request: HttpServletRequest, response: HttpServletResponse): void
PasswordEncryptionWithAes
- ENCRYPT_ALGO: String
- TAG_LENGTH_BIT: int
- IV_LENGTH_BYTE: int
- SALT_LENGTH_BYTE: int
- UTF_8: Charset
+ getRandomNonce(numBytes: int): byte[]
+ getAESKey(keysize: int): SecretKey
+ getAESKeyFromPassword(password: char[], salt: byte[]): SecretKey
+ encrypt(username: String, password: String): String
+ decrypt(encryptedPassword: String, username: String): String
RegisterServlet
- serialVersionUID: long
- dbController: DBController
+ registerServlet()
+ doGet(request: HttpServletRequest, response: HttpServletResponse): void
+ doPost(request: HttpServletRequest, response: HttpServletResponse): void
PlanModel
- planDurationDays: String
- planPrice: String
- planDescription: String
- planID: String
+ PlanModel()
+ PlanModel(planDurationDays: String, planPrice: String, planDescription: String, planID: String)
+ getPlanDurationDays(): String
+ getPlanID(): String
+ setPlanID(planID: String): void
+ setPlanDurationDays(planDurationDays: String): void
+ getPlanPrice(): String
+ setPlanPrice(planPrice: String): void
+ getPlanDescription(): String
+ setPlanDescription(planDescription: String): void
DBController
+ getConnection(): Connection
+ registerUser(UserModel): int
+ getUserLoginInfo(LoginModel): int
+ getAllUsersInfo(): ArrayList<UserModel>
+ getAllPlanInfo(): ArrayList<PlanModel>
+ registerPlan(PlanModel): int
+ deletePlan(String): int
+ updatePlan(PlanModel): int
+ getAdminInfo(LoginModel): int
+ checkIfEmailExists(String): Boolean
+ checkNumberIfExists(String): Boolean
+ checkUsernameIfExists(String): Boolean
UserModel
- username: String
- firstName: String
- lastName: String
- email: String
- phoneNumber: String
- password: String
- imageUrlFromPart: String
+ UserModel()
+ UserModel(username: String, password: String, firstName: String, lastName: String, email: String, phoneNumber: String, imagePart: Part)
+ getUsername(): String
+ getPassword(): String
+ setPassword(password: String): void
+ setUsername(username: String): void
+ getFirstName(): String
+ setFirstName(firstName: String): void
+ getLastName(): String
+ setLastName(lastName: String): void
+ getEmail(): String
+ setEmail(email: String): void
+ getPhoneNumber(): String
+ setPhoneNumber(phoneNumber: String): void
+ getImageUrlFromPart(): String
+ setImageUrlFromPart(part: Part): void
+ setImageUrlFromDB(imageUrl: String): void
- getImageUrl(part: Part): String
LoginModel
- username: String
- password: String
+ LoginModel(username: String, password: String)
+ getUsername(): String
+ setUsername(username: String)
+ getPassword(): String
+ setPassword(password: String)
LoginServlet
- serialVersionUID: long
- dbController: DBController
+ LoginServlet()
+ doGet(request: HttpServletRequest, response: HttpServletResponse): void
+ doPost(request: HttpServletRequest, response: HttpServletResponse): void
AdminServlet
- serialVersionUID: long
- dbController: DBController
+ AdminServlet()
+ doGet(request: HttpServletRequest, response: HttpServletResponse): void
+ doGetUsers(request: HttpServletRequest, response: HttpServletResponse): void
+ doGetPlans(request: HttpServletRequest, response: HttpServletResponse): void
+ doPost(request: HttpServletRequest, response: HttpServletResponse): void
+ doAddPlan(request: HttpServletRequest, response: HttpServletResponse): void
LogoutServlet
- serialVersionUID: long
+ LogoutServlet()
+ doGet(request: HttpServletRequest, response: HttpServletResponse): void
+ doPost(request: HttpServletRequest, response: HttpServletResponse): void
Home Servlet
- serialVersionUID: long
- dbController: DBController
+ HomeServlet()
+ doGet(request: HttpServletRequest, response: HttpServletResponse): void
+ doPost(request: HttpServletRequest, response: HttpServletResponse): void

Figure 25: Final Class Diagram

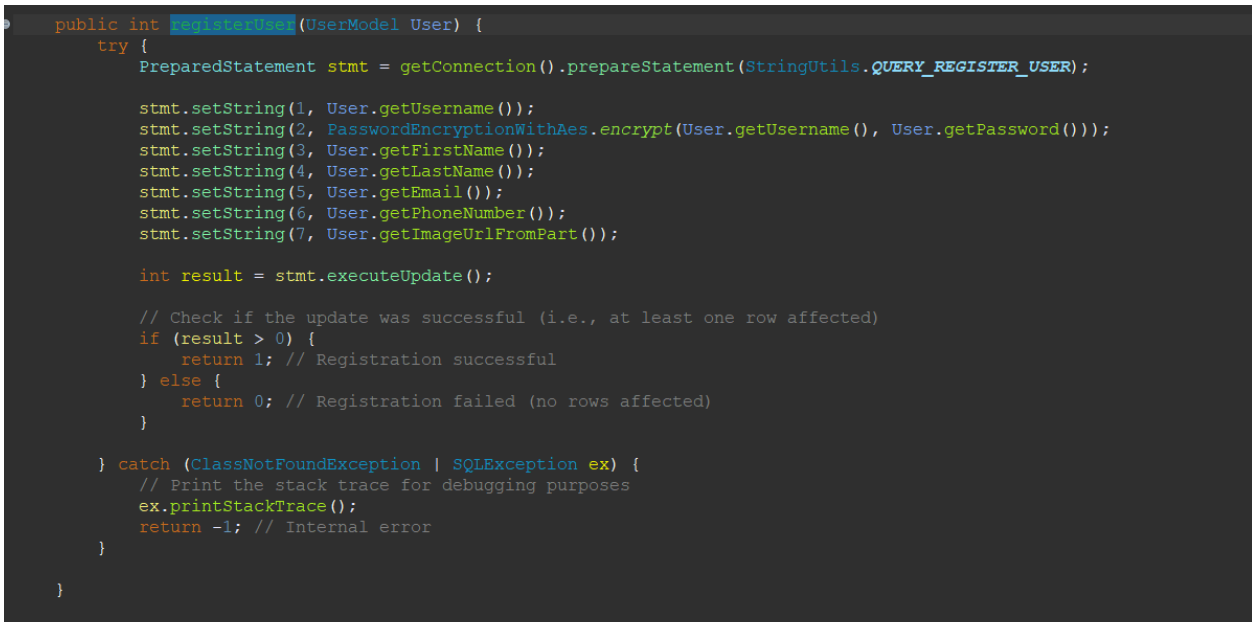
5. Method Description:

5.1. DB Controller

5.1.1. registerUser:

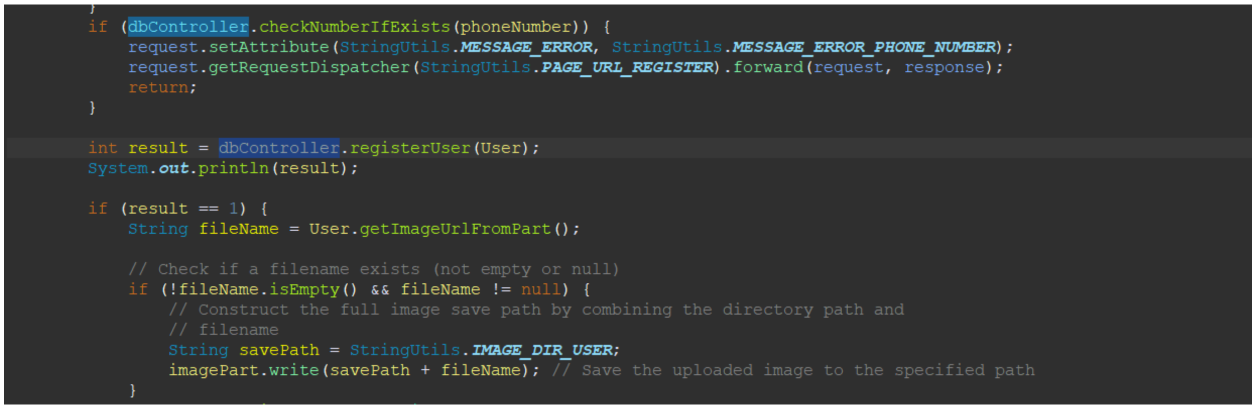
Method Name:	registerUser(UserModel User)
Method Description:	This method is used to register a new user in the database.
Triggered:	This method is invoked by the 'doPost' method in the registerServlet when a new user hits the register button 'register.jsp'.
How it works:	This method is called when a user hits the submit button on the registration form. It takes the model class object. containing user details as a parameter. It uses the INSERT query in sql.
Result:	It registers the user by inserting the user entered information into the database.

Table 1:registerUser method



```java
public int registerUser(UserModel User) {
    try {
        PreparedStatement stmt = getConnection().prepareStatement(StringUtils.QUERY_REGISTER_USER);

        stmt.setString(1, User.getUsername());
        stmt.setString(2, PasswordEncryptionWithAes.encrypt(User.getUsername(), User.getPassword()));
        stmt.setString(3, User.getFirstName());
        stmt.setString(4, User.getLastName());
        stmt.setString(5, User.getEmail());
        stmt.setString(6, User.getPhoneNumber());
        stmt.setString(7, User.getImageUrlFromPart());

        int result = stmt.executeUpdate();

        // Check if the update was successful (i.e., at least one row affected)
        if (result > 0) {
            return 1; // Registration successful
        } else {
            return 0; // Registration failed (no rows affected)
        }

    } catch (ClassNotFoundException | SQLException ex) {
        // Print the stack trace for debugging purposes
        ex.printStackTrace();
        return -1; // Internal error
    }

}
```

Figure 26: registerUser Method



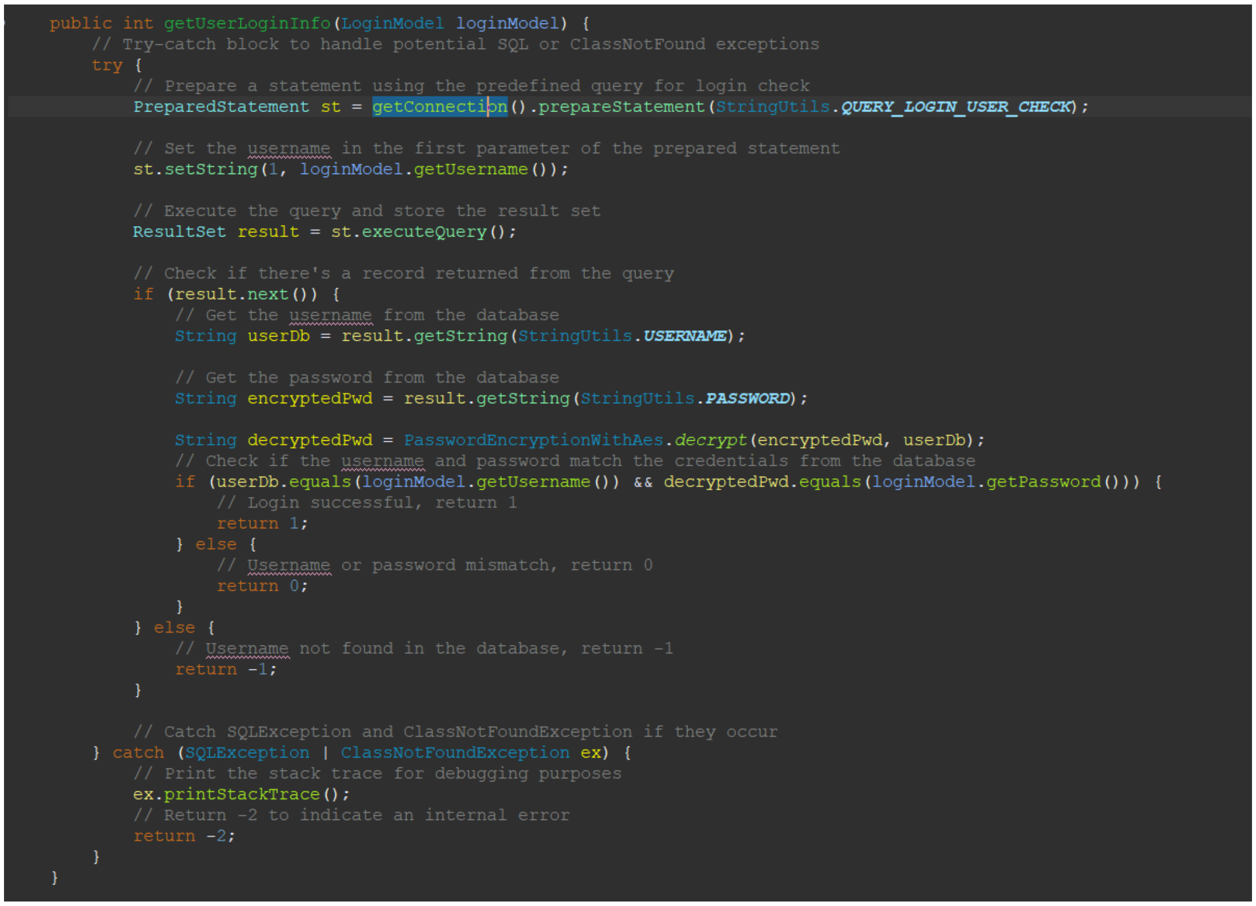
```
        if (dbController.checkNumberIfExists(phoneNumber)) {
            request.setAttribute(StringUtils.MESSAGE_ERROR, StringUtils.MESSAGE_ERROR_PHONE_NUMBER);
            request.getRequestDispatcher(StringUtils.PAGE_URL_REGISTER).forward(request, response);
            return;
        }

        int result = dbController.registerUser(User);
        System.out.println(result);

        if (result == 1) {
            String fileName = User.getImageUrlFromPart();

            // Check if a filename exists (not empty or null)
            if (!fileName.isEmpty() && fileName != null) {
                // Construct the full image save path by combining the directory path and
                // filename
                String savePath = StringUtils.IMAGE_DIR_USER;
                imagePart.write(savePath + fileName); // Save the uploaded image to the specified path
            }
```

Figure 27: registerUser method call

5.1.2. getAdminLoginInfo:

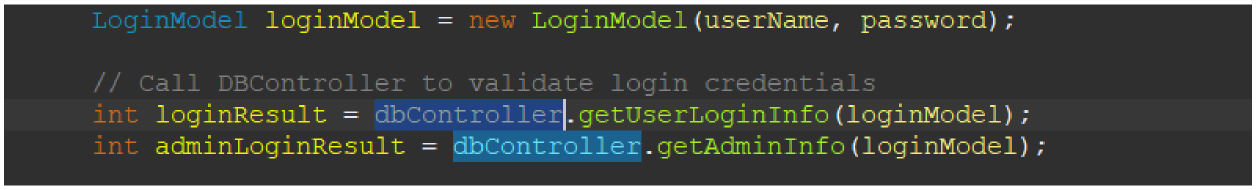
Method Name:	getAdminLoginInfo (LoginModel loginModel)
Method Description:	This method retrieves the information of the admin from the database.
Triggered:	This method is invoked by the 'doPost' method in 'LoginServlet'.
How it works:	This method is called when a admin logs in through 'login.jsp'. It takes a the 'LoginModel' object which contains the username and the password as its parameter. If the entered username or password is in the database, it logs the user in else throws appropriate message.
Result:	If it matches it returns 1 indicating successful login or else, it returns 0 showing error.

Table 2: getAdminLoginInfo method description



```java
public int getUserLoginInfo(LoginModel loginModel) {
    // Try-catch block to handle potential SQL or ClassNotFound exceptions
    try {
        // Prepare a statement using the predefined query for login check
        PreparedStatement st = getConnection().prepareStatement(StringUtils.QUERY_LOGIN_USER_CHECK);

        // Set the username in the first parameter of the prepared statement
        st.setString(1, loginModel.getUsername());

        // Execute the query and store the result set
        ResultSet result = st.executeQuery();

        // Check if there's a record returned from the query
        if (result.next()) {
            // Get the username from the database
            String userDb = result.getString(StringUtils.USERNAME);

            // Get the password from the database
            String encryptedPwd = result.getString(StringUtils.PASSWORD);

            String decryptedPwd = PasswordEncryptionWithAes.decrypt(encryptedPwd, userDb);
            // Check if the username and password match the credentials from the database
            if (userDb.equals(loginModel.getUsername()) && decryptedPwd.equals(loginModel.getPassword())) {
                // Login successful, return 1
                return 1;
            } else {
                // Username or password mismatch, return 0
                return 0;
            }
        } else {
            // Username not found in the database, return -1
            return -1;
        }

        // Catch SQLException and ClassNotFoundException if they occur
    } catch (SQLException | ClassNotFoundException ex) {
        // Print the stack trace for debugging purposes
        ex.printStackTrace();
        // Return -2 to indicate an internal error
        return -2;
    }
}
```

Figure 28: getAdminLoginInfo method



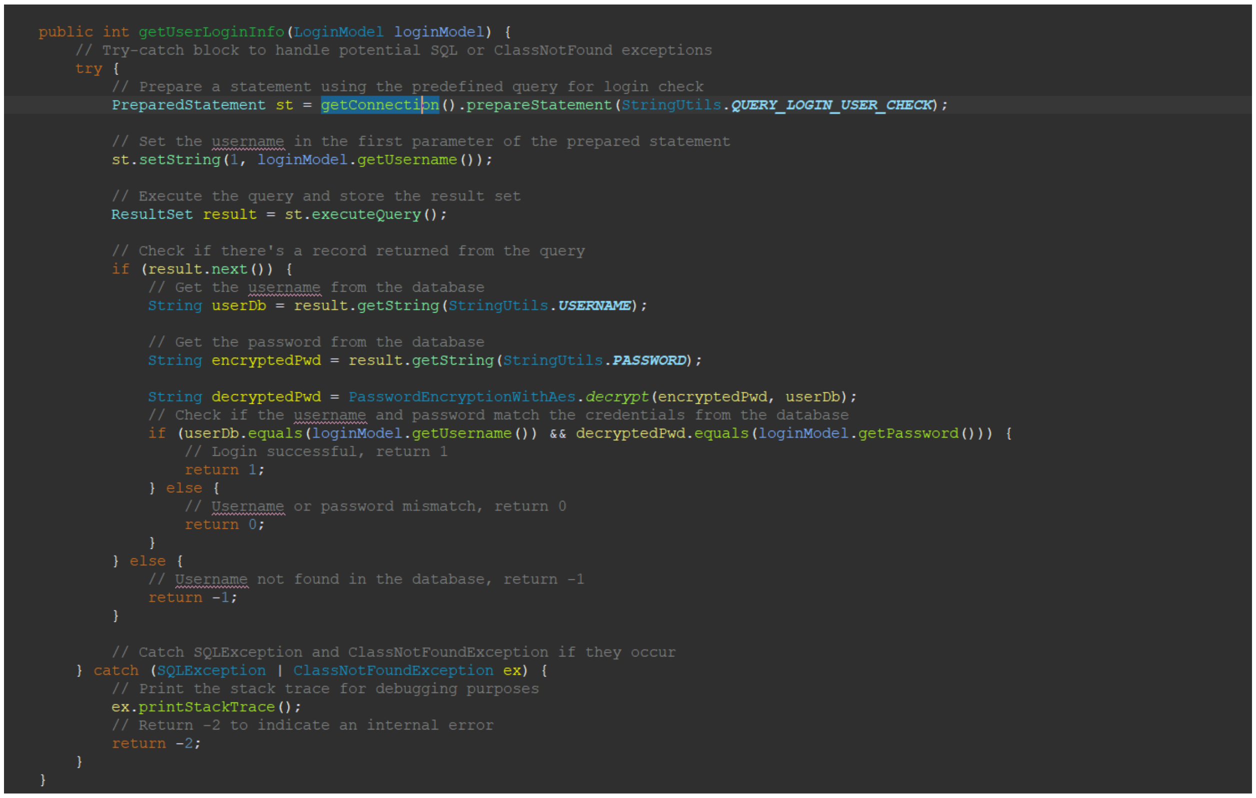
```java
LoginModel loginModel = new LoginModel(userName, password);

// Call DBController to validate login credentials
int loginResult = dbController.getUserLoginInfo(loginModel);
int adminLoginResult = dbController.getAdminInfo(loginModel);
```

Figure 29: getAdminLoginInfo method called

5.1.3. getUserLoginInfo:

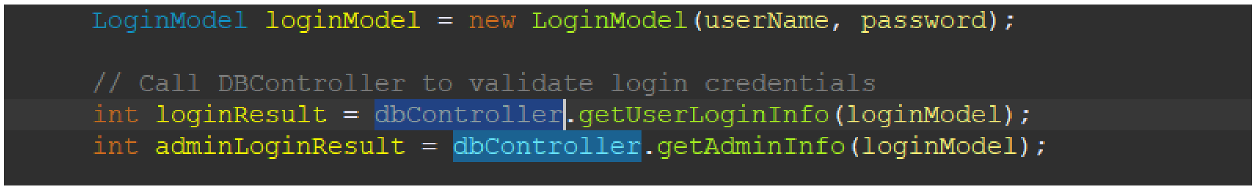
Method Name:	getUserLoginInfo(LoginModel loginModel)
Method Description:	This method is used to retrieve the login information of the user form the database.
Triggered:	This method is invoked by the 'doPost' method in 'LoginServlet' when user logs in through 'login.jsp'.
How it works:	This method is called when a user logs in through 'login.jsp'. It takes a the 'LoginModel' object which contains the username and the password as its parameter. If the entered username or password is in the database, it logs the user in else throws appropriate message.
Result:	If it matches it returns 1 indicating successful login or else it returns 0 or -1 showing error.

Table 3: getUserLoginInfo method description



```java
public int getUserLoginInfo(LoginModel loginModel) {
    // Try-catch block to handle potential SQL or ClassNotFound exceptions
    try {
        // Prepare a statement using the predefined query for login check
        PreparedStatement st = getConnection().prepareStatement(StringUtils.QUERY_LOGIN_USER_CHECK);

        // Set the username in the first parameter of the prepared statement
        st.setString(1, loginModel.getUsername());

        // Execute the query and store the result set
        ResultSet result = st.executeQuery();

        // Check if there's a record returned from the query
        if (result.next()) {
            // Get the username from the database
            String userDb = result.getString(StringUtils.USERNAME);

            // Get the password from the database
            String encryptedPwd = result.getString(StringUtils.PASSWORD);

            String decryptedPwd = PasswordEncryptionWithAes.decrypt(encryptedPwd, userDb);
            // Check if the username and password match the credentials from the database
            if (userDb.equals(loginModel.getUsername()) && decryptedPwd.equals(loginModel.getPassword())) {
                // Login successful, return 1
                return 1;
            } else {
                // Username or password mismatch, return 0
                return 0;
            }
        } else {
            // Username not found in the database, return -1
            return -1;
        }

        // Catch SQLException and ClassNotFoundException if they occur
    } catch (SQLException | ClassNotFoundException ex) {
        // Print the stack trace for debugging purposes
        ex.printStackTrace();
        // Return -2 to indicate an internal error
        return -2;
    }
}
```

Figure 30: getUserLoginInfo method



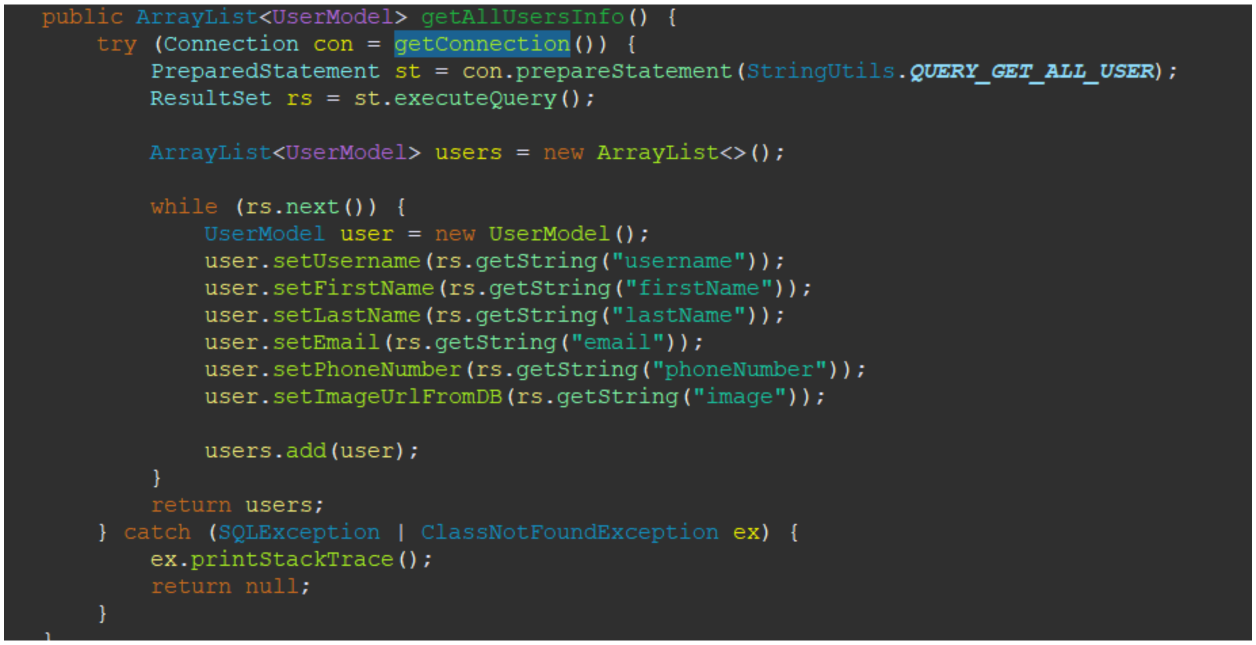
```
LoginModel loginModel = new LoginModel(userName, password);

// Call DBController to validate login credentials
int loginResult = dbController.getUserLoginInfo(loginModel);
int adminLoginResult = dbController.getAdminInfo(loginModel);
```

Figure 31: getUserLoginInfo method called

5.1.4. getAllUsersInfo:

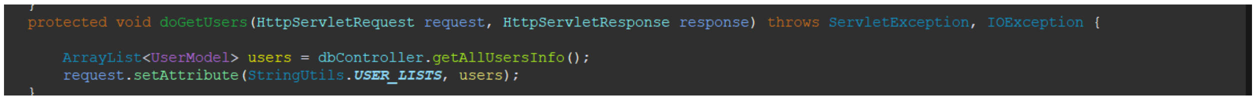
Method Name:	getAllUsersInfo( )
Method Description:	This method uses the SELECT * query to gell all the user information from the database.
Triggered :	This method is triggered when the 'AdminServlet' sends a request to fetch the user details to then display on the Admin Dashbord
How it works:	This method retrieves all user details in the 'UserModel' object and stores it in a arraylist and when the method is called by 'UserProfileServlet' it returns the arraylist for it to be displayed in 'UserProfile.jsp'
Result:	It returns the details to the servlet for it to be displayed.

Table 4: getAllUsersInfo method description



```
public ArrayList<UserModel> getAllUsersInfo() {
    try (Connection con = getConnection()) {
        PreparedStatement st = con.prepareStatement(StringUtils.QUERY_GET_ALL_USER);
        ResultSet rs = st.executeQuery();

        ArrayList<UserModel> users = new ArrayList<>();

        while (rs.next()) {
            UserModel user = new UserModel();
            user.setUsername(rs.getString("username"));
            user.setFirstName(rs.getString("firstName"));
            user.setLastName(rs.getString("lastName"));
            user.setEmail(rs.getString("email"));
            user.setPhoneNumber(rs.getString("phoneNumber"));
            user.setImageUrlFromDB(rs.getString("image"));

            users.add(user);
        }
        return users;
    } catch (SQLException | ClassNotFoundException ex) {
        ex.printStackTrace();
        return null;
    }
```

Figure 32: getAllUsersInfo method



```
protected void doGetUsers(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    ArrayList<UserModel> users = dbController.getAllUsersInfo();
    request.setAttribute(StringUtils.USER_LISTS, users);
}
```

Figure 33:getAllUsersInfo method called

Method Name	Method Description
getAllPlanInfo()	This method retrives all the information from the plans table to be displayed at plan.jsp and at the admin dashboard for modification. This
registerPlan(PlanModel)	This method is used to register a new plan in the database. This method is called when the admin hits the submit button on the register plan forum. This method uses the INSERT query.
deletePlan(String)	This method is used to delete a plan form the database. It takes planID as a parameter which is used in the DELETE query to remove a plan from the database.
updatePlan(PlanModel)	This method is used to update an existing plan in the database. This method is called when the admin submits the update forum in the admin dashboard. This method uses the UPDATE query to update a plan with planID that is set by the parameter.
checkIfEmailExists(String)	This method is used to validate if a specific email already exists in the database or not.
checkNumberIfExists(String)	This method is used to validate if a specific phone number already exists in the database or not.
checkUsernameIfExists(String)	This method is used to validate if a specific email already exists in the database or not.

Table 5: Method Description DBController

5.2. Register Servlet:

Method Name	Method Description
registerServlet()	This method is the constructor that initializes the object of DBController Class
doPost()	This method handles the HTTP post request sent to the servlet.This method is invoked when a user hits submit in the registration form.It extracts the user information from the form parameter. After validating the input it calls the method from DBController to register a new user.

Table 6: Register Servlet Methods

5.3. Login Servlet:

Table 7:Login Servlet Method Description

Method Name:	Method Description
doPost()	This method handles HTTP POST requests sent to the servlet. This method is invoked when the user hits the login button on the login form. It extracts the username and password from the parameter before validating the login credentials to see if they exist on the database. If they do exist, they log the user In and create cookies and sessions.

5.3. Admin Servlet:

Method Name:	Method Description
public AdminServlet()	This method is the constructor of the class. It initializes ar object for DBController.
doGet()	This method calls the doGetUsers() and doGetPlans(method.
doGetUsers()	This method is called to populate the array, to display the details of the user. This method uses the SELECT query to get information of the user from the database.
doGetPlans()	This method is called to populate the array, to display the details of the plans. This method uses the SELECT query to get information of the plans from the database.
doAddPlans()	This method is used to add a new plan to the database. I is invoked when the admin hits the submit button on the register plans form in from the admin dashboard. This method uses the INSERT query.

Table 8:Admin Servlet Method Description

5.5. Redirection Filter:

Method Name	Method Description
doFilter()	The doFilter function serves as the primary method of the filter responsible for authentication. It takes ServletRequest, ServletResponse, and FilterChain objects as arguments. It fetches the attribute of the session which is created at Login and uses it to filter. If no session is created, it assumes the user is not logged in and redirects them to the login page.

5.6. Manage Plan Servlet:

Method Name:	Method Description
doPost()	This method handles HTTP POST requests to update or delete product. This method is called when it receives the request from user. It determines whether to update the product details by calling the 'doUpdate' method or delete the product by calling 'doDelete' method.
doUpdate()	This method is invoked when a admin requests to update a plan. If the doPost() method receives a value for update id, this method is called. This method uses the UPDATE query to update existing plans in the database.
doDelete()	This method is invoked when a admin requests to delete a plan. If the doPost() method receives a value for delete id, this method is called. This method uses the DELETE query to update existing plans in the database.

5.7 Home Servlet:

Method Name:	Method Description
HomeServlet()	This method is the constructor of the class. This method initializes an object of DBController.
doGet()	This method gets the value of the user form the database. It uses the SELECT method. The value is then displayed on home.jsp. It only displays the value of the current user.

5.8. Logout Servlet:

Method Name:	Method Description
doPost()	This method deletes the user session and cookies. It is used to logout the user.

5.9. Plan Servlet:

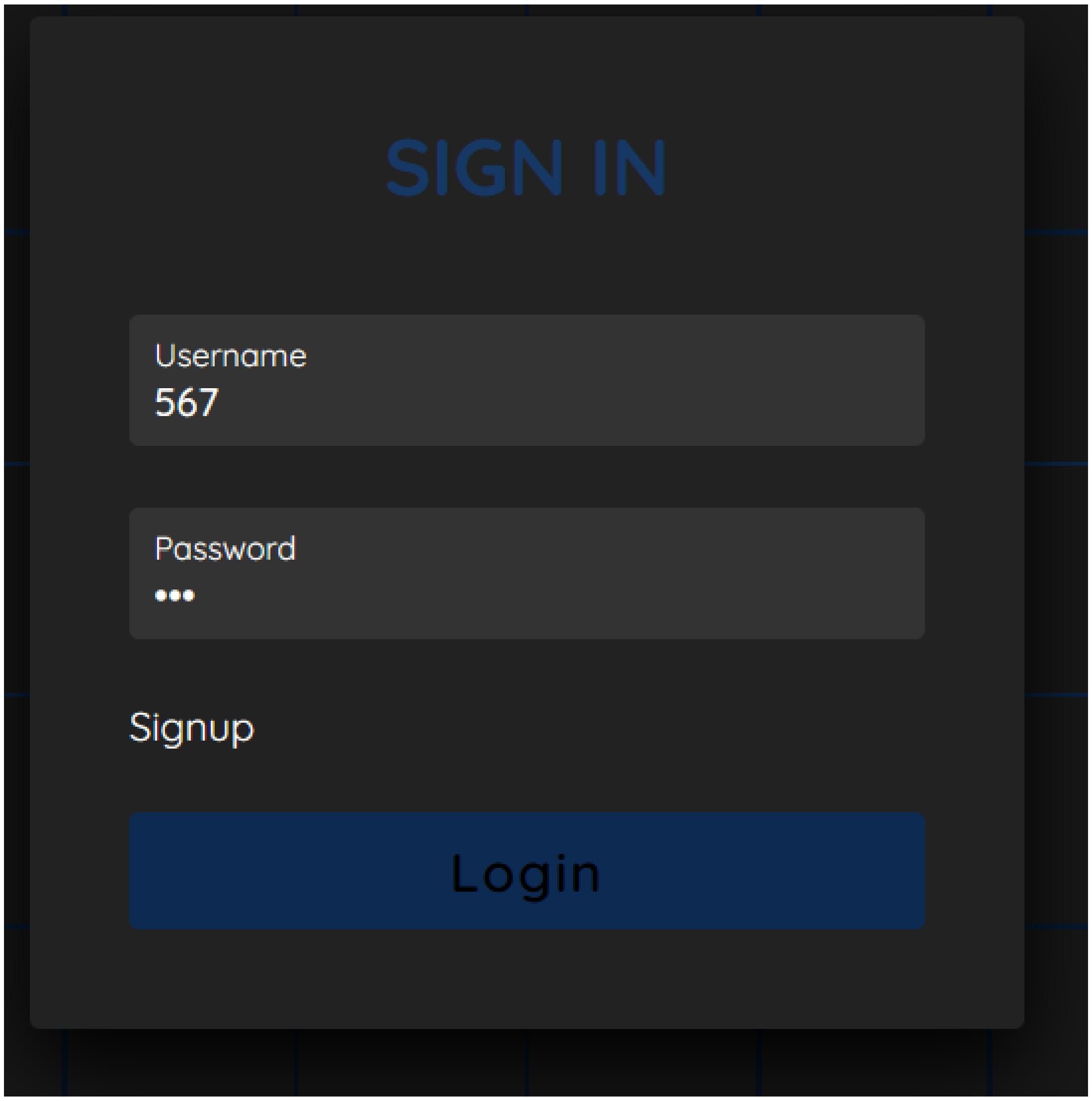
Method Name:	Method Description
Planservlet()	This method is the constructor of the class. This method initializes an object of DBController.
doGet()	This method gets the plans from the database to be display in the plans page. It uses the SELECT query.
doPost()	Ths method is used to add a new plan to the database. It uses the INSERT query.

6.Test Cases:

6.1.Validation:

6.1.1. Validation for login:

Objective	To log in to the website based on stored user
Action	The credential of the user is input and the login button is pressed.
Expected Result	The user is then identified and sent to the home page.
Actual Result	The user was identified and sent to the home page.
Conclusion	The test was successful.



567

●●●

Figure 34 : Entering Credentials

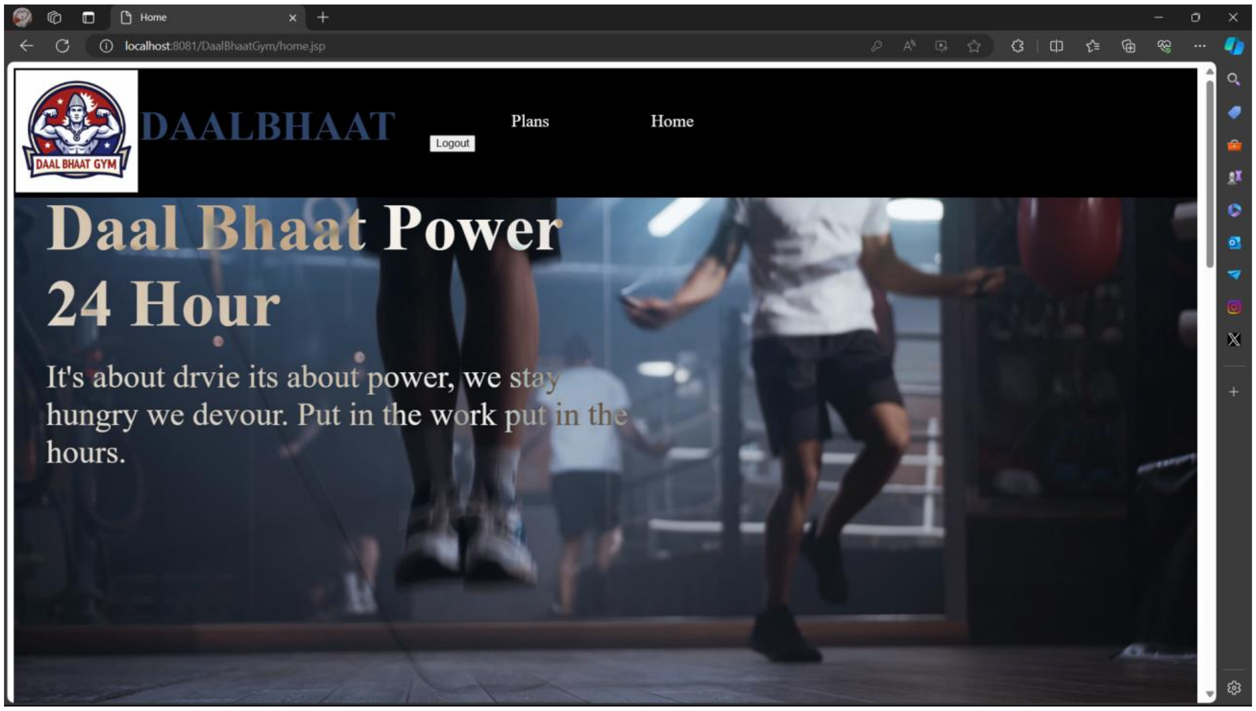
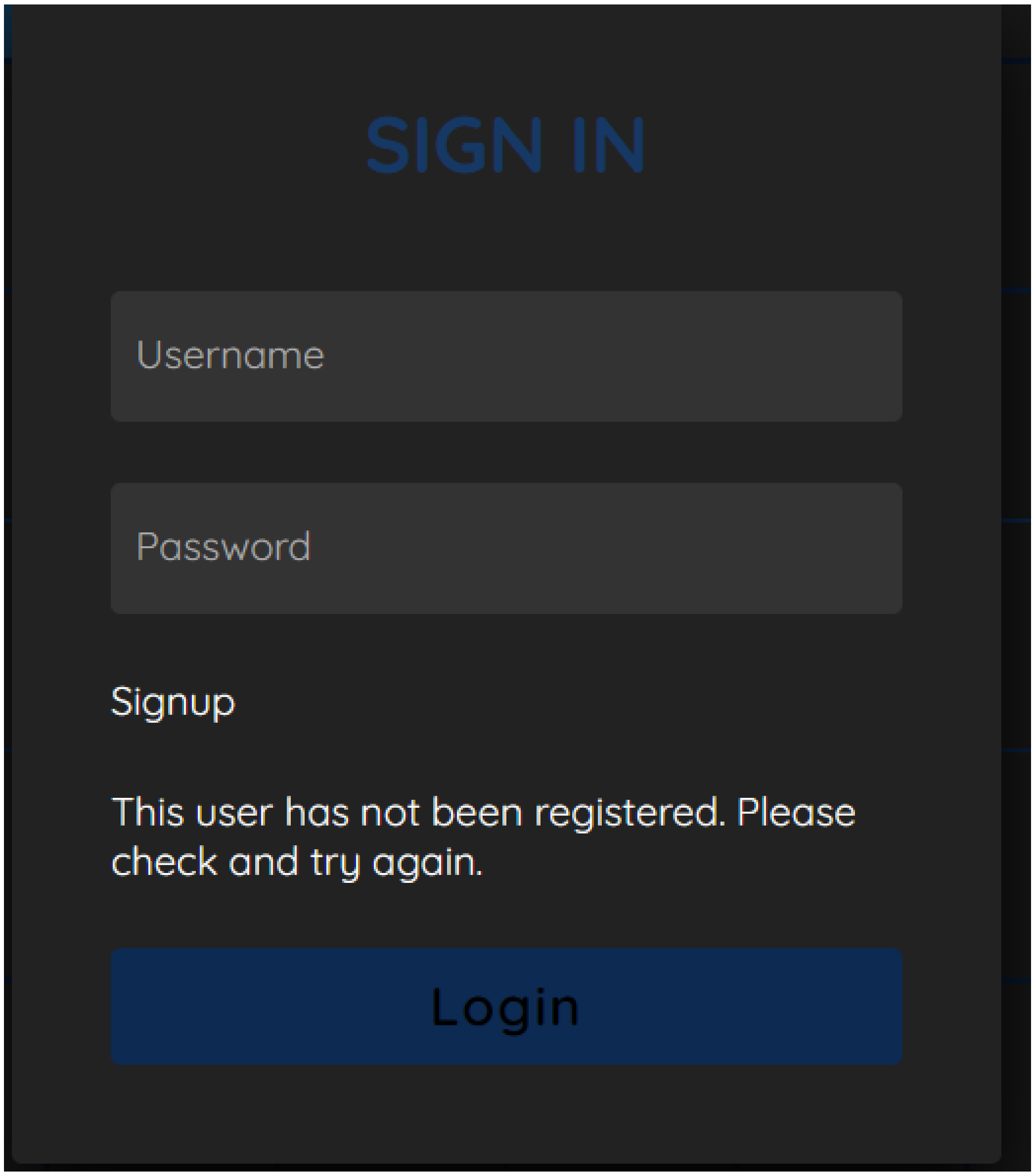

Home
localhost:8081/DaalBhaatGym/home.jsp
DAAL BHAAT GYM
DAALBHAAT
Logout
Plans
Home
Daal Bhaat Power
24 Hour
It's about drvie its about power, we stay hungry we devour. Put in the work put in the hours.

Figure 35:Login successful.

6.1.2. Login Value Validation:

Objective	To validate the login credentials.
Action	The user enters the log in credentials with proper values
Expected Result	The system checks if the value is valid and performs validation accordingly.
Actual Result	The user input was validated properly.
Conclusion	The test was successful.

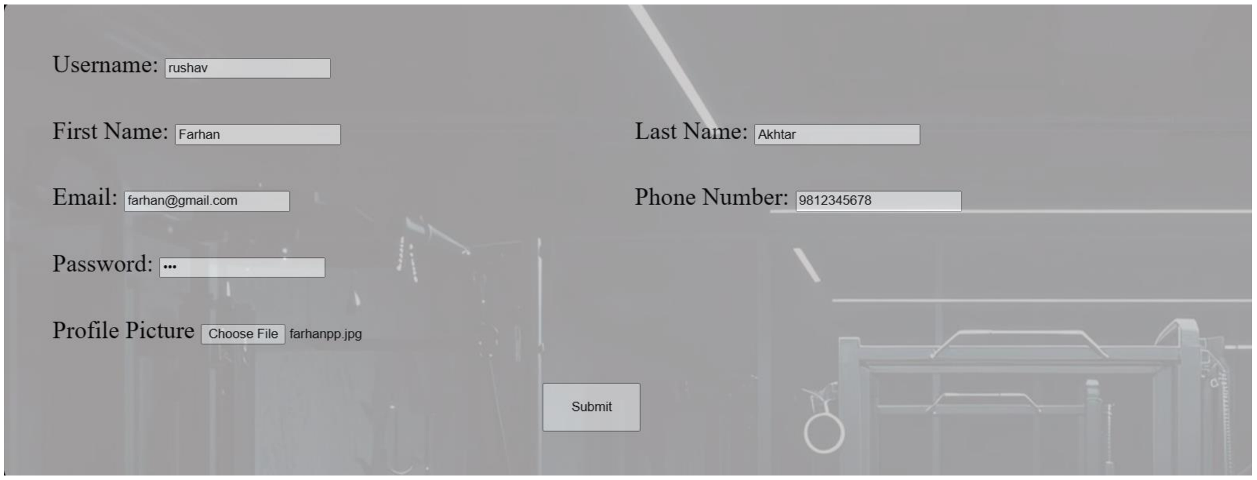


This user has not been registered. Please check and try again.

Figure 36: username checking validation

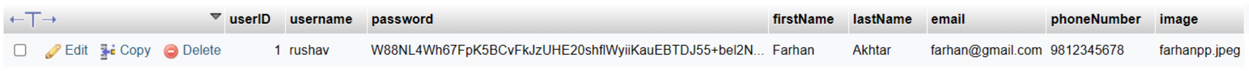
6.1.3. Register Validation:

Objective	To register new user
Action	The user submits all their details required for registration.
Expected Result	The user should be sent to the next page and the information should be stored in a database.
Actual Result	The user is forwarded to the next page and all of their data is stored in the database.
Conclusion	The test was successful.



Username:	rushav
First Name:	Farhan
Last Name:	Akhtar
Email:	farhan@gmail.com
Phone Number:	9812345678
Password:	•••
Profile Picture	farhanpp.jpg

Figure 37: Register Validation

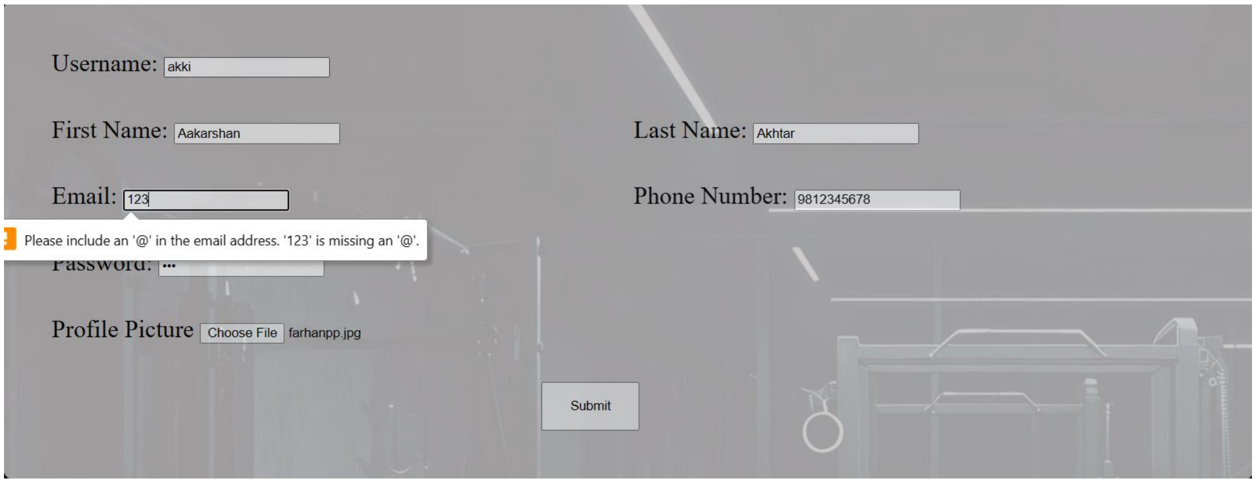


	userID	username	password	firstName	lastName	email	phoneNumber	image
Edit Copy Delete	1	rushav	W88NL4Wh67FpK5BCvFkJzUHE20shflWyiiKauEBTDJ55+bel2N...	Farhan	Akhtar	farhan@gmail.com	9812345678	farhanpp.jpeg

Figure 38: Registration Successful

6.1.4.Register Value Validation:

Objective	To validate the user details
Action	The user enters the credentials and submits the registration form
Expected Result	The system should validate the user input
Actual Result	The system validates the user input.
Conclusion	The test was successful.



Username: akki

First Name: Aakarshan

Last Name: Akhtar

Email: 123

Phone Number: 9812345678

Please include an '@' in the email address. '123' is missing an '@'.

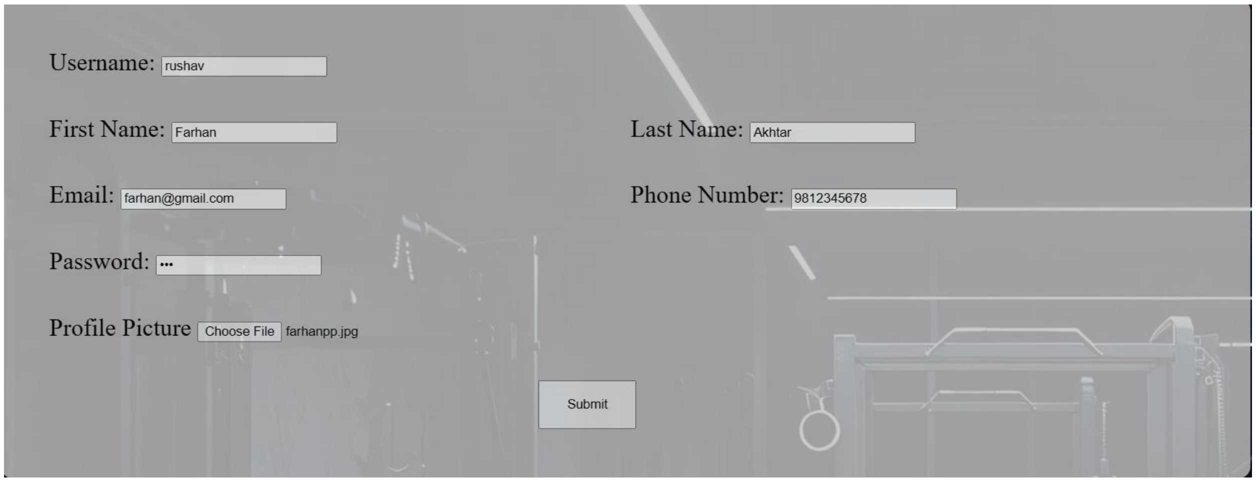
Password: ...

Profile Picture Choose File farhanpp.jpg

Figure 39: Validation successful Register

6.1.5. User Duplication Validation:

Objective	To check if the user already exist.
Action	The user enters their details
Expected Result	The system checks for any duplicate inputs.
Actual Result	The system successfully detects duplicate inputs.
Conclusion	The test was successful.



Username: rushav

First Name: Farhan

Last Name: Akhtar

Email: farhan@gmail.com

Phone Number: 9812345678

Password: ...

Profile Picture Choose File farhanpp.jpg

Figure 40:Duplicate Value Insertion

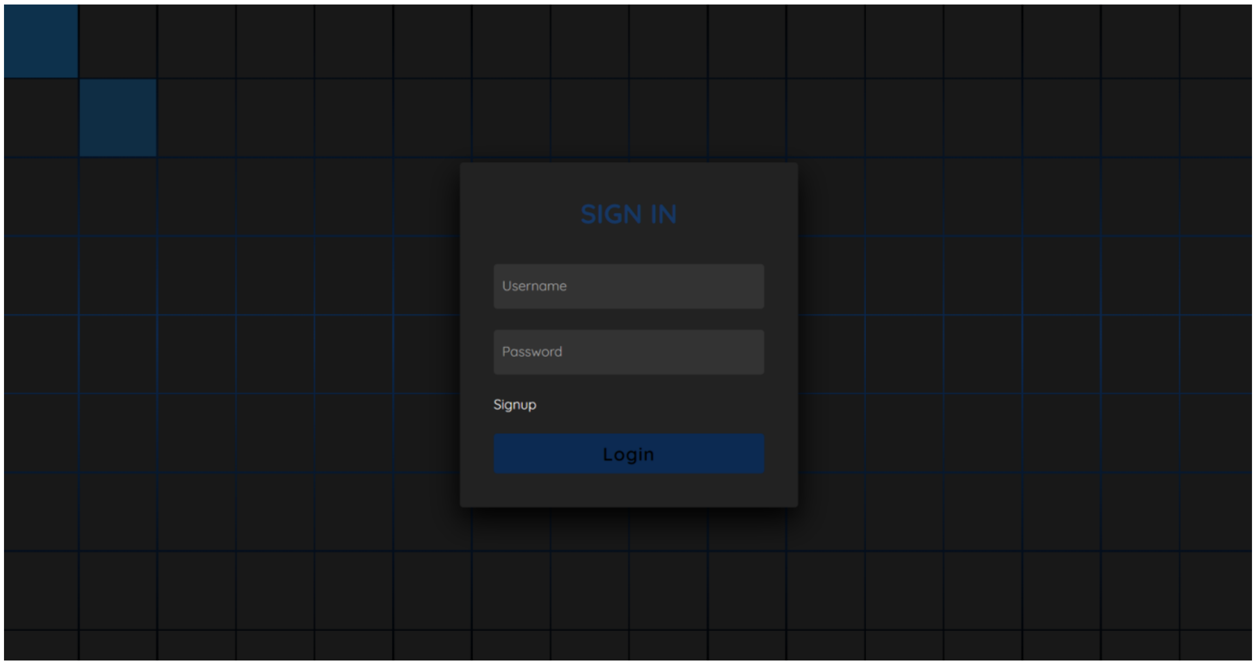
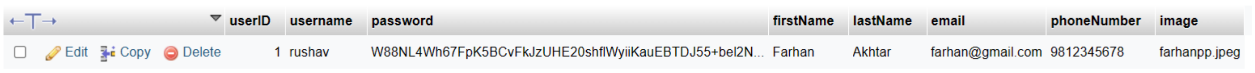


Figure 41: Login Page Redirection

6.1.6.Password Encryption:

Objective	To encrypt the password provided by the new user after registering.
Action	The user enters their password in the field.
Expected Result	When storing the password in the database it should be encrypted.
Actual Result	The passwords are encrypted and stored in the database.
Conclusion	The test was successful.

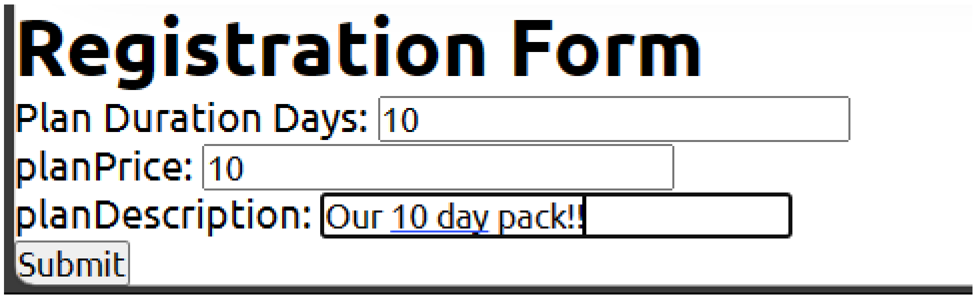


	userID	username	password	firstName	lastName	email	phoneNumber	image
Edit Copy Delete	1	rushav	W88NL4Wh67FpK5BCvFkJzUHE20shflWyiiKauEBTDJ55+bel2N...	Farhan	Akhtar	farhan@gmail.com	9812345678	farhanpp.jpeg

Figure 42:Password Encrypted in database

6.1.7. Product Addition Validation:

Objective	To validate the inputs while adding a product
Action	The user adds their preferred plan.
Expected Result	The system validates the plan details
Actual Result	The system properly validates all plan details.
Conclusion	The test was successful.



Registration Form

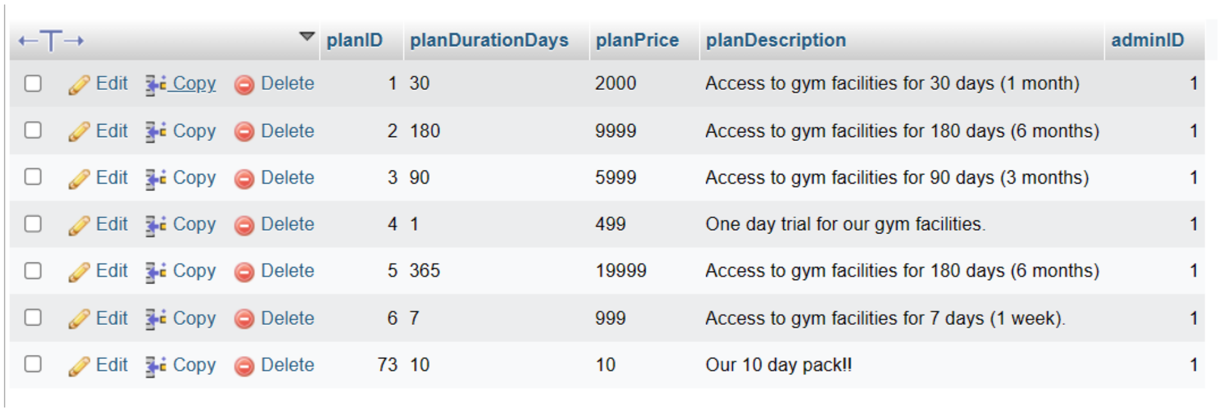
Plan Duration Days: 10

planPrice: 10

planDescription: Our 10 day pack!!

Submit

Figure 43: Adding Plan Details

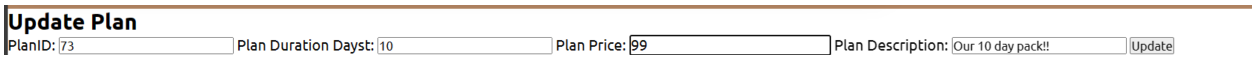


	planID	planDurationDays	planPrice	planDescription	adminID
Edit Copy Delete	1	30	2000	Access to gym facilities for 30 days (1 month)	1
Edit Copy Delete	2	180	9999	Access to gym facilities for 180 days (6 months)	1
Edit Copy Delete	3	90	5999	Access to gym facilities for 90 days (3 months)	1
Edit Copy Delete	4	1	499	One day trial for our gym facilities.	1
Edit Copy Delete	5	365	19999	Access to gym facilities for 180 days (6 months)	1
Edit Copy Delete	6	7	999	Access to gym facilities for 7 days (1 week).	1
Edit Copy Delete	73	10	10	Our 10 day pack!!	1

Figure 44:Addition Successful

6.1.9. Product Update Validation:

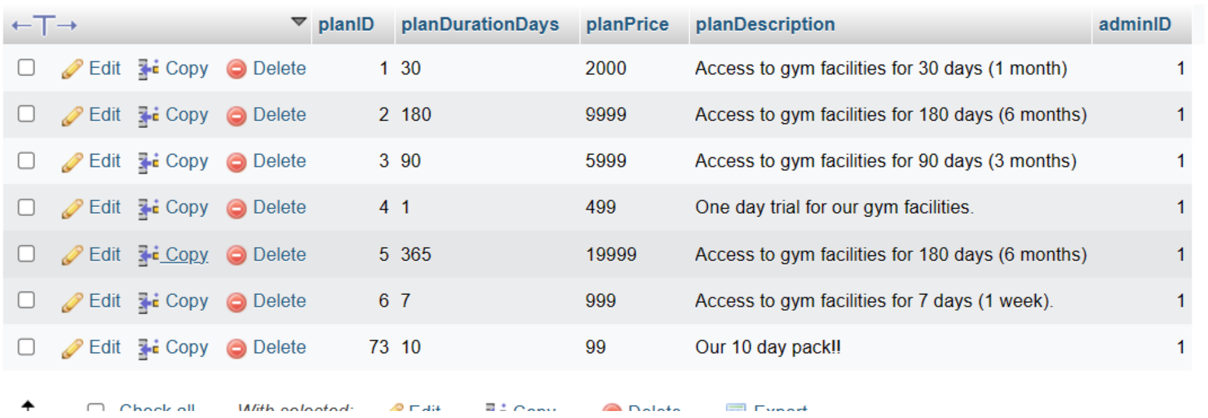
Objective	To check if the updated product details are valid
Action	The user updates for values of their plan
Expected Result	The system checks for the validity of the updated plan.
Actual Result	The system successfully checks for valid plans.
Conclusion	The test was successful.



Update Plan

PlanID: 73 Plan Duration Dayst: 10 Plan Price: 99 Plan Description: Our 10 day pack!! Update

Figure 45: Updating Value



	planID	planDurationDays	planPrice	planDescription	adminID
☐ Edit Copy Delete	1	30	2000	Access to gym facilities for 30 days (1 month)	1
☐ Edit Copy Delete	2	180	9999	Access to gym facilities for 180 days (6 months)	1
☐ Edit Copy Delete	3	90	5999	Access to gym facilities for 90 days (3 months)	1
☐ Edit Copy Delete	4	1	499	One day trial for our gym facilities.	1
☐ Edit Copy Delete	5	365	19999	Access to gym facilities for 180 days (6 months)	1
☐ Edit Copy Delete	6	7	999	Access to gym facilities for 7 days (1 week).	1
☐ Edit Copy Delete	73	10	99	Our 10 day pack!!	1

Figure 46: Updated Value in Database

6.1.10.User/Admin Login :

Objective	To log in to the website based on user/admin.
Action	The admin/user enters the credentials email and password specific to the admin and a user to log in
Expected Result	The system identifies if the user is an admin or not and proceeds accordingly
Actual Result	The type of user is successfully identified and redirected accordingly.
Conclusion	The test was successful.

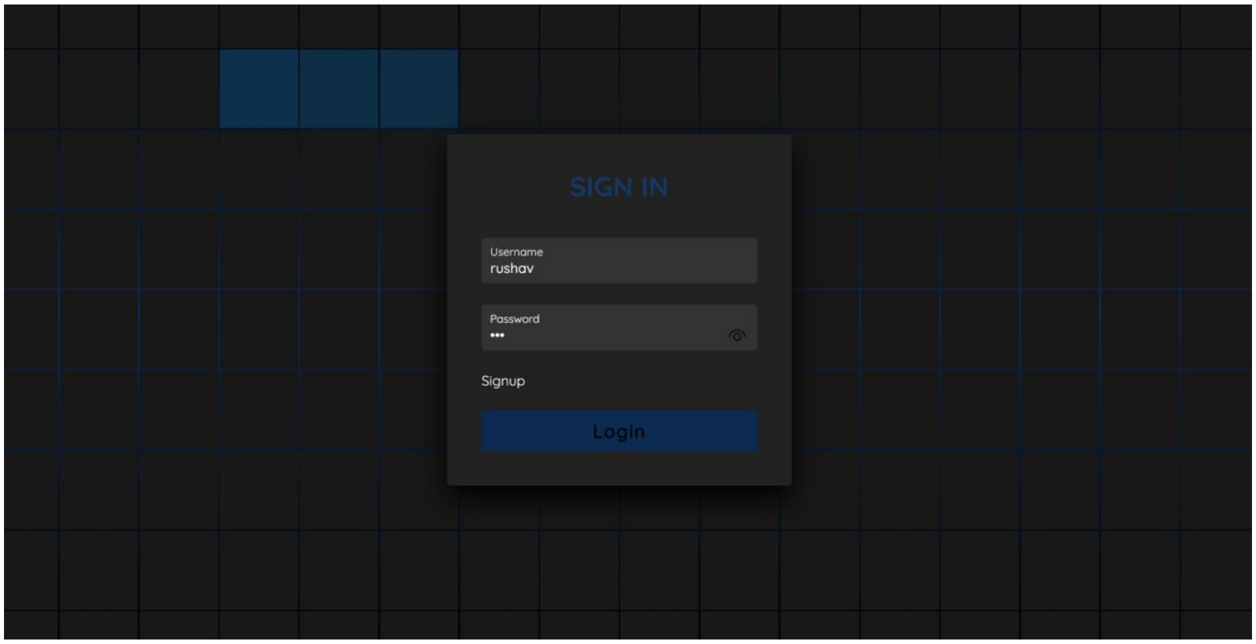
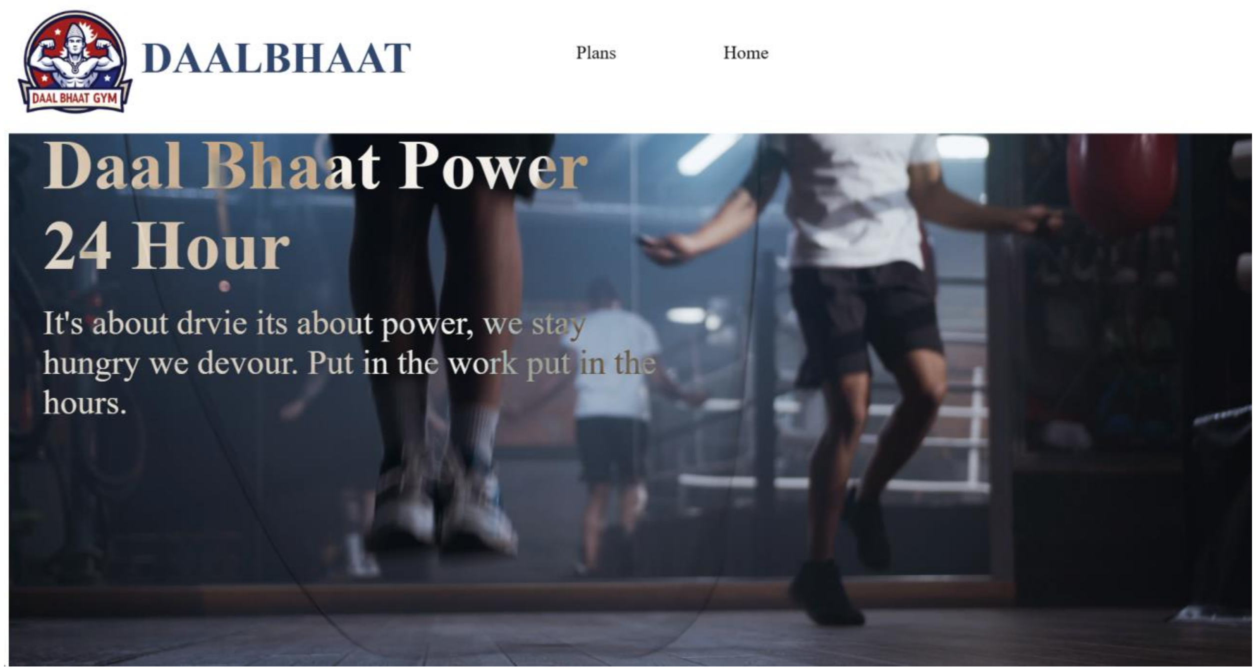
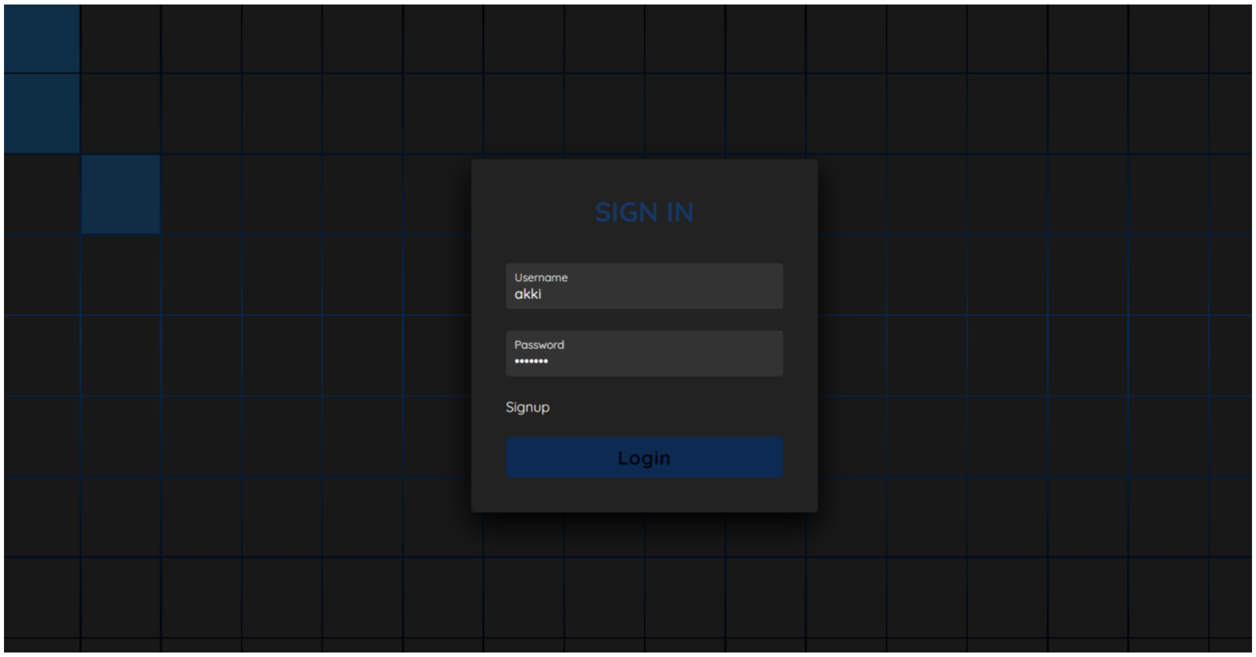


Figure 47:Log in as user



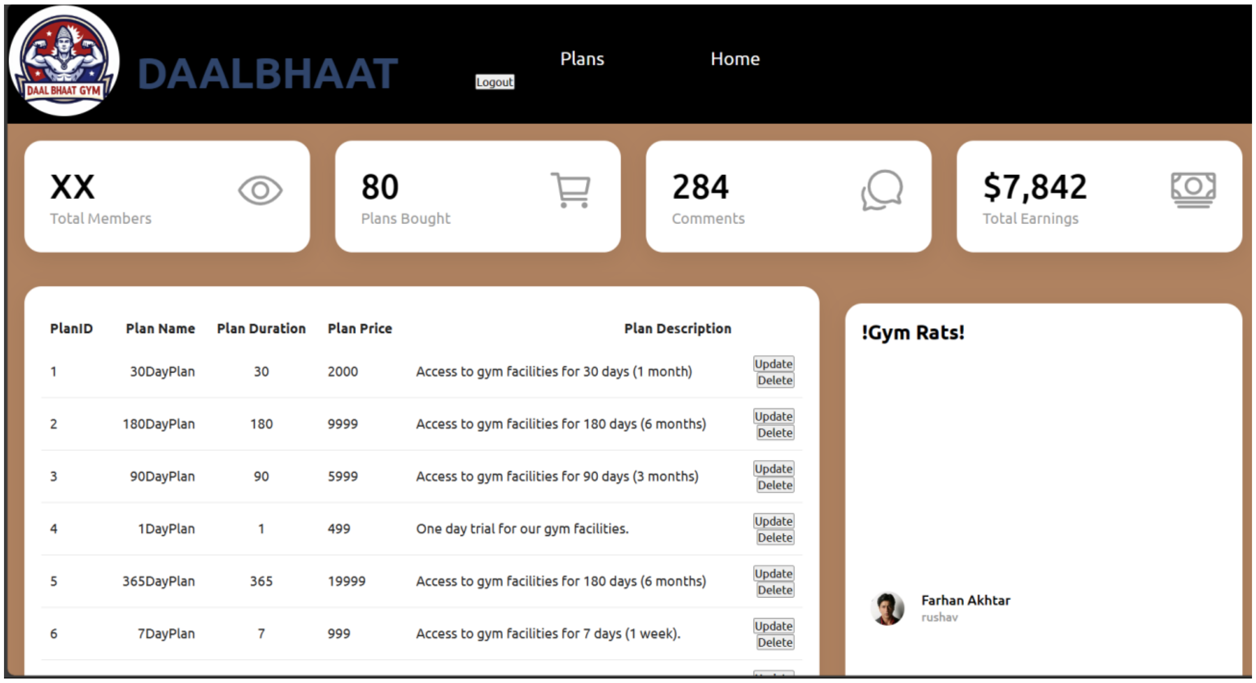
It's about drvie its about power, we stay hungry we devour. Put in the work put in the hours.

Figure 48: Directed to home page as user



Username
akki
Password
•••••••

Figure 49:Logging in as admin



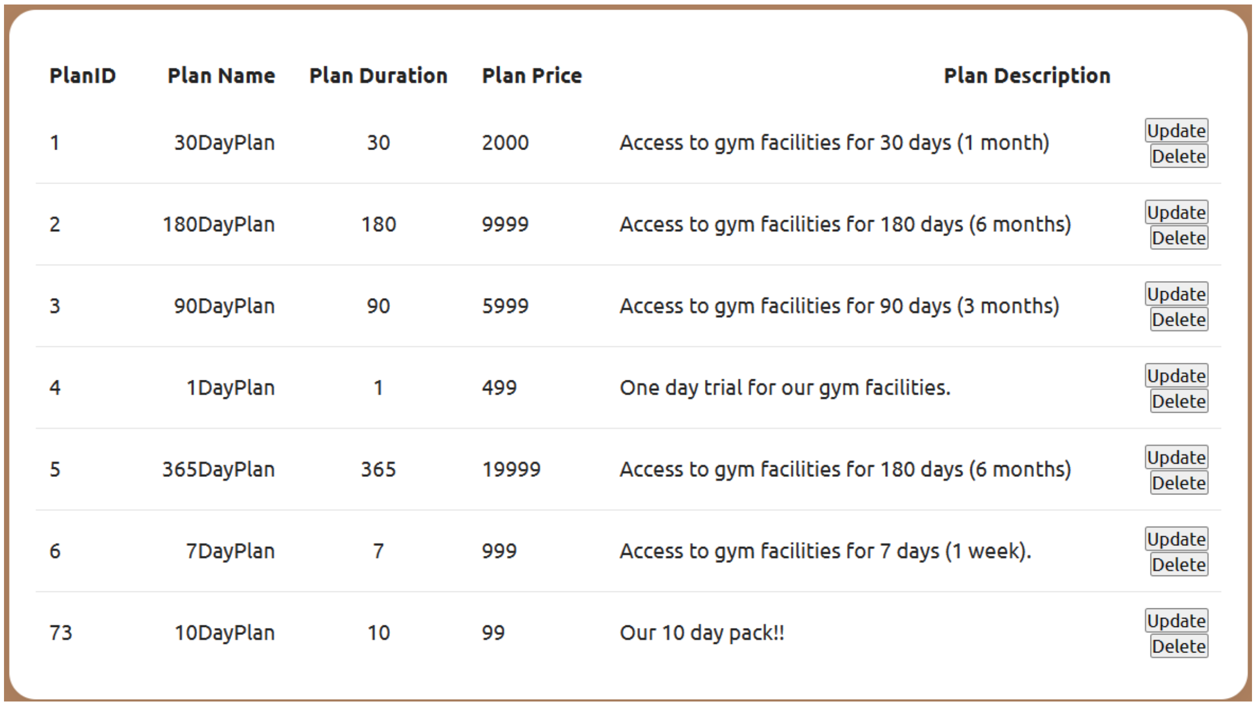
PlanID	Plan Name	Plan Duration	Plan Price	Plan Description	
1	30DayPlan	30	2000	Access to gym facilities for 30 days (1 month)	Update Delete
2	180DayPlan	180	9999	Access to gym facilities for 180 days (6 months)	Update Delete
3	90DayPlan	90	5999	Access to gym facilities for 90 days (3 months)	Update Delete
4	1DayPlan	1	499	One day trial for our gym facilities.	Update Delete
5	365DayPlan	365	19999	Access to gym facilities for 180 days (6 months)	Update Delete
6	7DayPlan	7	999	Access to gym facilities for 7 days (1 week).	Update Delete

Figure 50:Redirected to Admin

6.2. Table :

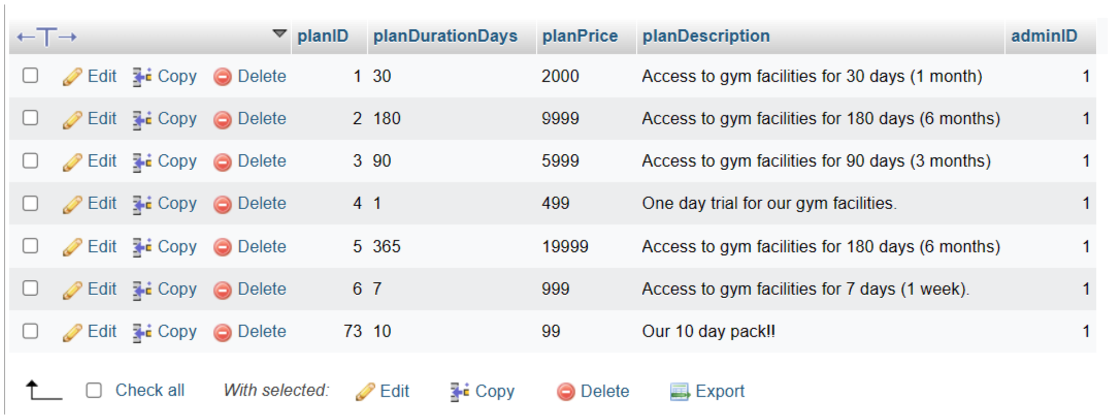
6.2.1. Table Display:

Objective	To display table values from database.
Action	The main page is opened to check for a table with all the values.
Expected Result	The information of the plans should be taken from the database and displayed in the screen.
Actual Result	The system properly displays all the necessary information in tabular form
Conclusion	The test was successful.



PlanID	Plan Name	Plan Duration	Plan Price	Plan Description	
1	30DayPlan	30	2000	Access to gym facilities for 30 days (1 month)	Update Delete
2	180DayPlan	180	9999	Access to gym facilities for 180 days (6 months)	Update Delete
3	90DayPlan	90	5999	Access to gym facilities for 90 days (3 months)	Update Delete
4	1DayPlan	1	499	One day trial for our gym facilities.	Update Delete
5	365DayPlan	365	19999	Access to gym facilities for 180 days (6 months)	Update Delete
6	7DayPlan	7	999	Access to gym facilities for 7 days (1 week).	Update Delete
73	10DayPlan	10	99	Our 10 day pack!!	Update Delete

Figure 51:Table Shown In Dashboard



←T→	planID	planDurationDays	planPrice	planDescription	adminID
Edit Copy Delete	1	30	2000	Access to gym facilities for 30 days (1 month)	1
Edit Copy Delete	2	180	9999	Access to gym facilities for 180 days (6 months)	1
Edit Copy Delete	3	90	5999	Access to gym facilities for 90 days (3 months)	1
Edit Copy Delete	4	1	499	One day trial for our gym facilities.	1
Edit Copy Delete	5	365	19999	Access to gym facilities for 180 days (6 months)	1
Edit Copy Delete	6	7	999	Access to gym facilities for 7 days (1 week).	1
Edit Copy Delete	73	10	99	Our 10 day pack!!	1

Check all *With selected:* Edit Copy Delete Export

Figure 52:Value in Database

6.2.2. Table Addition:

Objective	To add table values into the database.
Action	The user enters in the add plan form to add plans.
Expected Result	The system enters all the data given by the user into the database
Actual Result	The system properly adds all the data into the database.
Conclusion	The test was successful.

6.2.3. Table Deletion:

Objective	To delete table values from the database.
Action	The delete button in the table is pressed
Expected Result	All the values of the respective id should be deleted.
Actual Result	All the values that were to be deleted were removed from the database.
Conclusion	The test was successful.

6.2.4. Table Update:

Objective	To update table values into the database.
Action	The clicks on the update button.
Expected Result	The values entered should be added to the database,
Actual Result	The values are successfully added to the database.
Conclusion	The test was successful.

6.3. Session Creation:

6.3.1.Patient session:

Objective	To create user session when logged in.
Action	The user enters their username and password and presses the login button
Expected Result	The program should create a new ID for the user
Actual Result	A new Id was created.
Conclusion	The test was successful.

6.3.2.Pharmacist session:

Objective	To create admin session when logged in.
Action	The admin user inputs their credentials.
Expected Result	The system should create an admin ID.
Actual Result	A new admin ID was created
Conclusion	The test was successful.

7.Development Process :

7.1.Eclipse IDE:

Eclipse is an Integrated Development Environment (IDE) widely used for software development. It is mainly preferred over other IDEs because it has a vast number of tools and features at the disposal to make work faster and more efficient for the user. It is very widely used for many languages like Java, C++ and python.

7.1.1. Why Eclipse?

1. Java Support: Eclipse offers comprehensive support for Java development, including tools designed specifically for Java projects and features like code completion and syntax highlighting.

2. Server Integration: We can easily publish, execute, and debug our web applications straight from the Eclipse IDE thanks to its smooth integration with well-known application servers like Apache Tomcat.

3. Extensibility: Eclipse has a wide variety of plugins and extensions each catered to a different user needs. There are many plug in available, some of the most used are; database administration tools, Git version control systems, and JSP development.

7.2. Java:

Java is a widely used object-oriented programming language mainly used as It operates on billions of devices, including game consoles, mobile phones, laptop computers, medical equipment, and a host of other gadgets. The languages C and C++ served as the foundation for Java's conventions and grammar. It is also platform independent and supports multithreading.

7.2.1.Implementation of Java :

Java is implemented in this project in the servlets which are java classes used in web application.

7.3.Apache Tomcat Server:

Tomcat is an open-source web server and servlet.Its main purpose is to host java programs on the web. It is very compatible with java as it was build on Java and was specifically tailored to work with jsp. Tomcat helps join java programs to the internet, Reason for selectiom of Apachr Tomcat Server:

1. Java Servlet Support: It is specifically made to handle java servlets and jsp.
2. Servlet Container: Tomcat is used as a servlet container to provide runtime environments for java servlets.
3. Compatibility: Apache Tombat is compatible with many OS some common ones beingl windows and macOS.

7.4.Java Server Page(JSP):

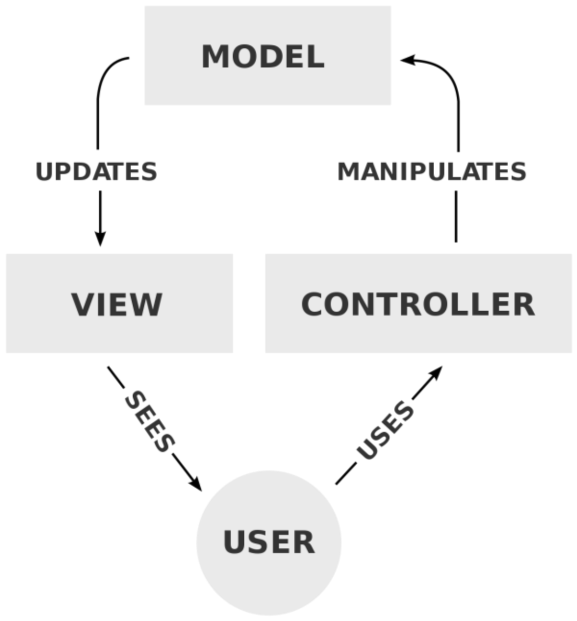
JSP is a server side technology which is used for creating dynamic web applications. JSP consists of both JSP and HTML files where the jsp files are used to implement java into html.

7.6. MVC Architecture:

MVC or Model-View-Controller is a pattern used in software design for the implementation of user interfaces, data, and controlling logic. It signifies the difference of user interface and code functionality.

The MVC has 3 parts;

1. Model: Manages data and business logic.
2. View: Handles layout and display.
3. Controller: Routes commands to the model and view parts.



7.7.Draw.io:

Draw.io is a web software used to create diagrams of software. It was used to create class diagrams for this project. Draw.io is a very widely used program that allows visual representation of the functionality and flow of programs.

7.8.Figma:

Figma is a very widely used software mainly used to create blueprints for websites. It has many tools like layering items and has flexibility with its components allowing for very accurate representation of websites on wireframes.

(Figma, 2022)