Abstract

The proposed system for the coursework is “**expensoft**” an expense tracking system implemented using java web-based technology Java EE server-side, where the user can add their expense and income view summary statistics regarding their expenditure. The role base authentication for user and admin is developed along with other functionality likes user profile, user and admin dashboard, homepage, reset password field etc.

The overall design of the system is show through class diagram, Different test cases for evidence of the features, tools and libraries used to develop the project, project system development phase, method description all are included in this report. The wireframe with the actual design implementation is also included in the report.

Table of Contents

Table of Figures

Table of tables

1. Introduction

‘Expensoft’ is an expense tracking system that tracks the user expense and income, display their total expense and income summary. The roles-based login authentication is made with the help of session and cookies management where the access to pages for the user and admin are separated. The admin can view the user details, update the user details, delete user, add new user, can see overall system statistics with the help of admin dashboard.

For the dynamic web content JSP and JSTL of Java EE server-side technology are used, for Database MYSQL is used and for server development java servlet is used, for frontend HTML, CSS and basic JavaScript is being used in this system. It has pages for CRUD database operation like user profile page where user can update their data, delete account, change password, and their detail is displayed.

1.1 Aim

"To develop java web based application using Java EE server side technology with CRUD database operation ".

1.2 Objective

- Implement of CRUD database operation
- Develop a project using Java EE server-side technology.
- Implement Role base login auth entication for user and admin.
- Display the user information, with the functionality to add update and delete the user from the admin page.
- Display the user expense and income information with the functionality of CRUD database operation in user page.
- Create a Profile page for user where the user detail is displayed, and CRUD operation is included.
- Implementation of change password field in user profile
- Design the database with meaningful data and appropriate data type.
- Implement Session tracking and cookies in the system.

2. UI/UX Design

2.1 Wireframe

2.1.1 Login page

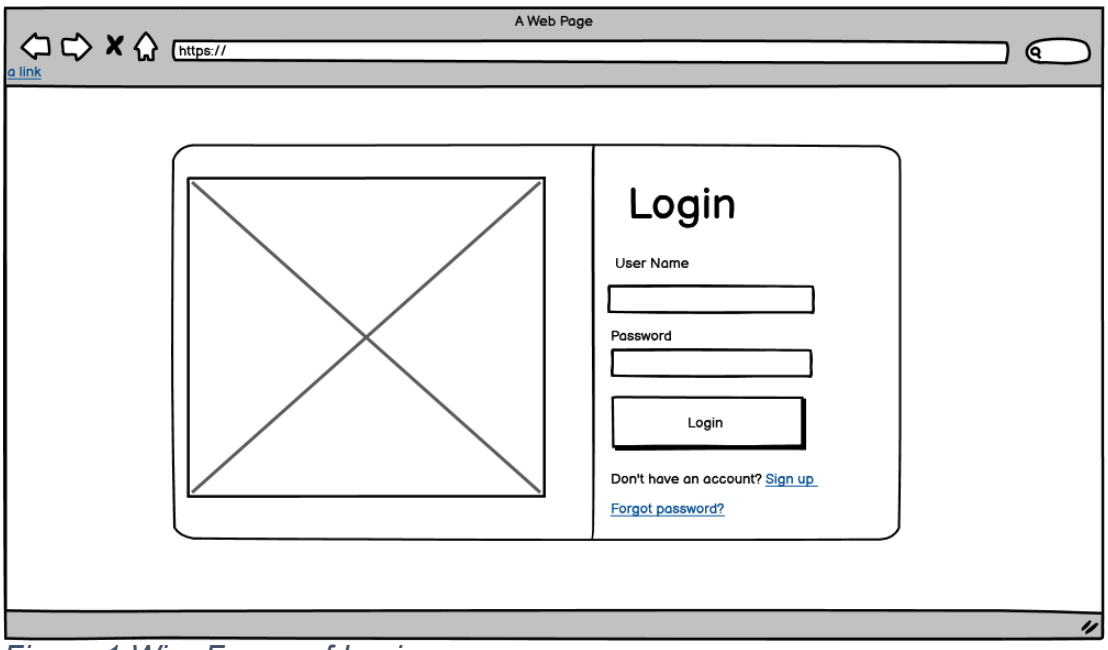


Figure 1 Wire Frame of Login page

2.1.2 Sign Up page

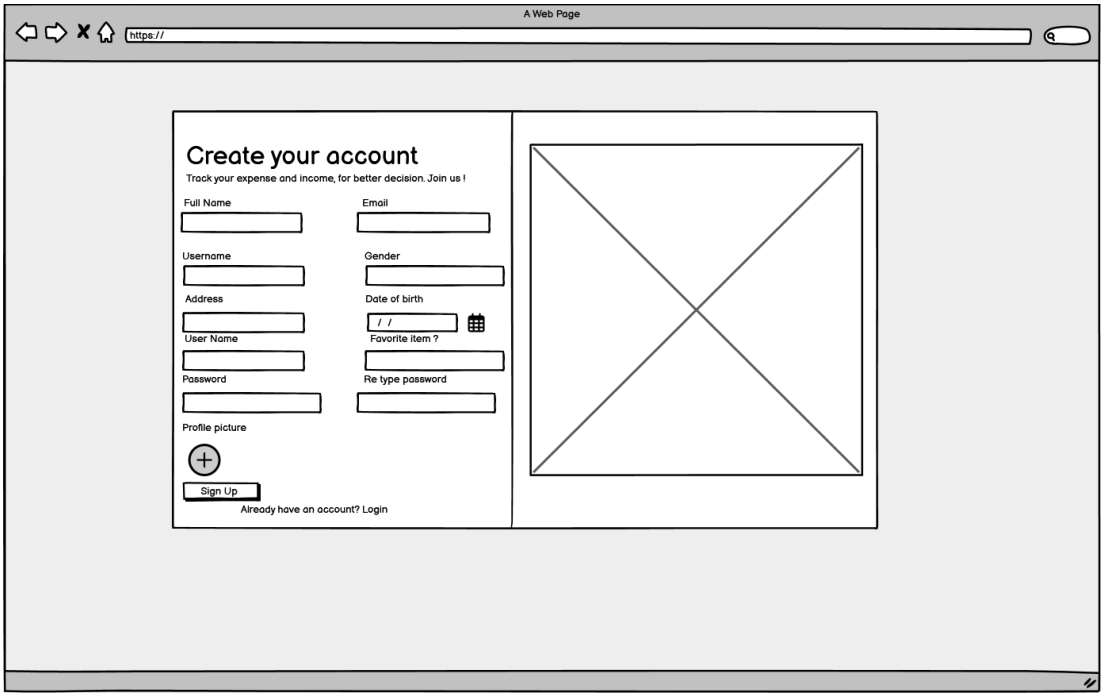


Figure 2 Wire frame of Sign Up page

2.1.3 User home page

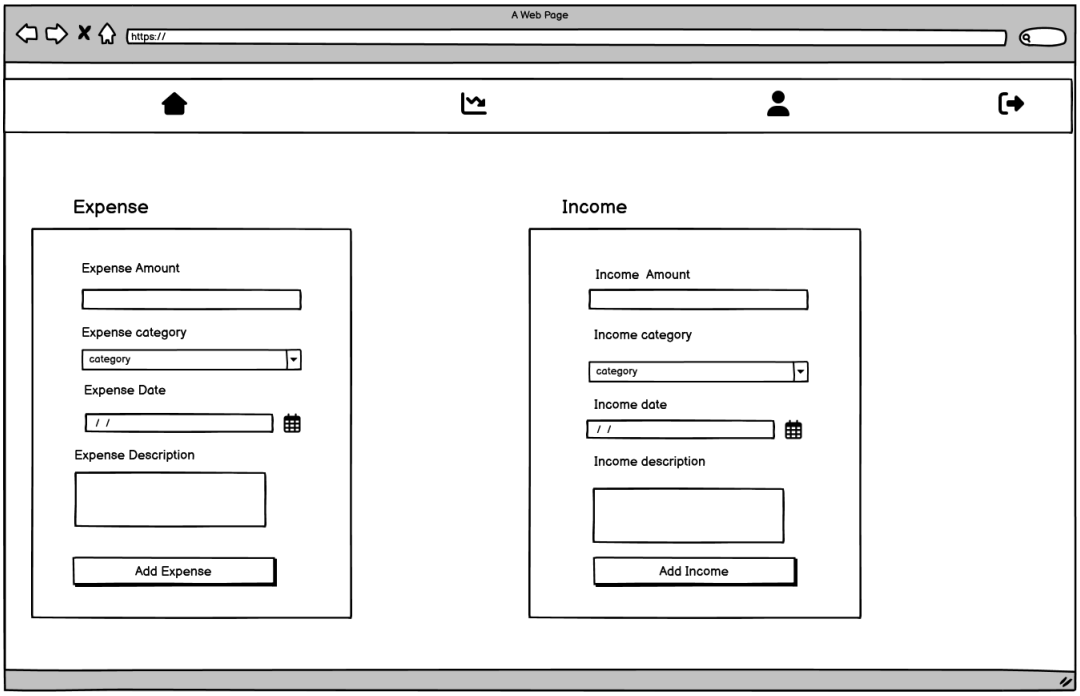


Figure 3 Wire frame of user home page.

2.1.4 Admin home page

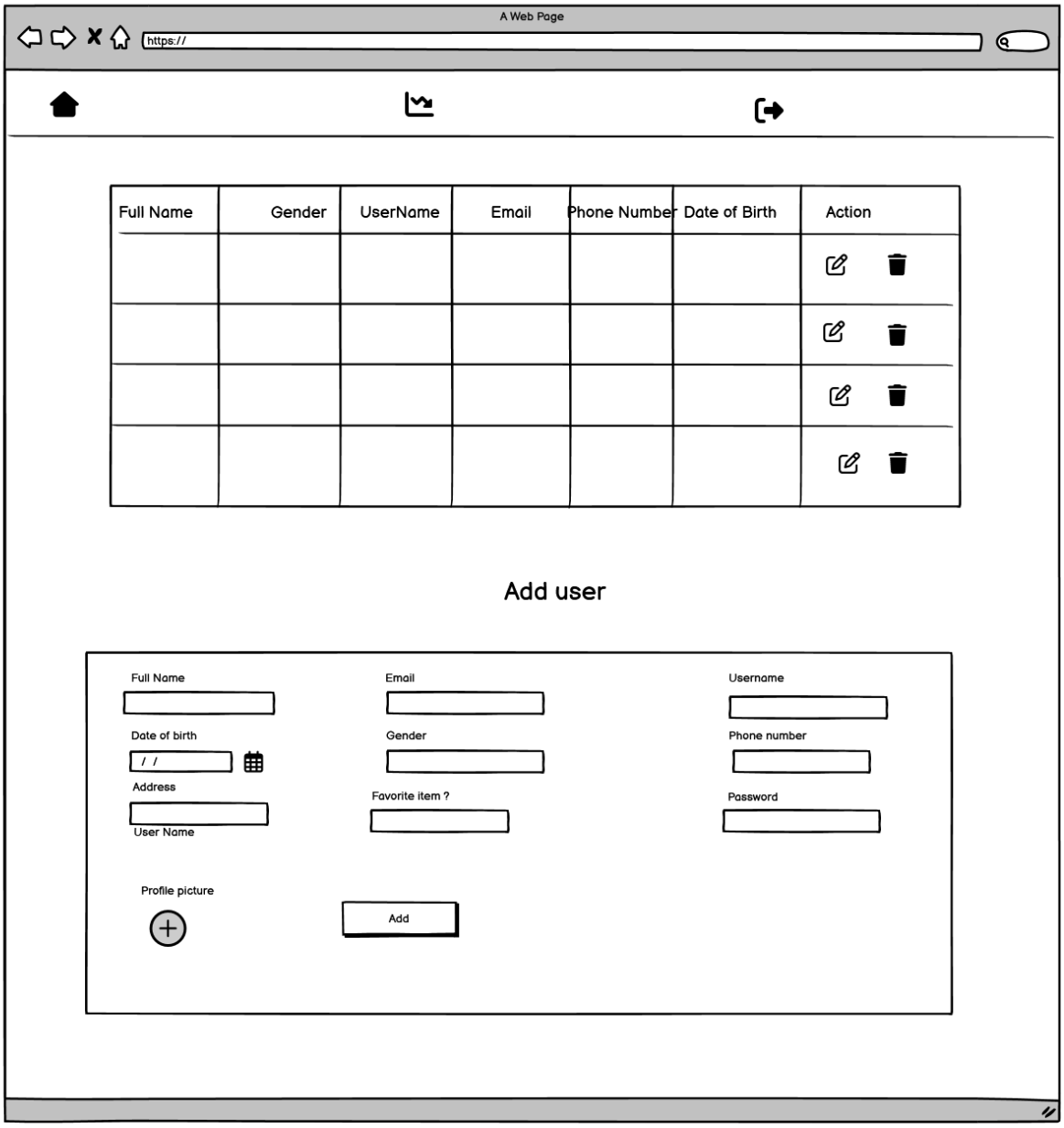


Figure 4 Wire frame of Admin home page.

2.1.5 User Profile page

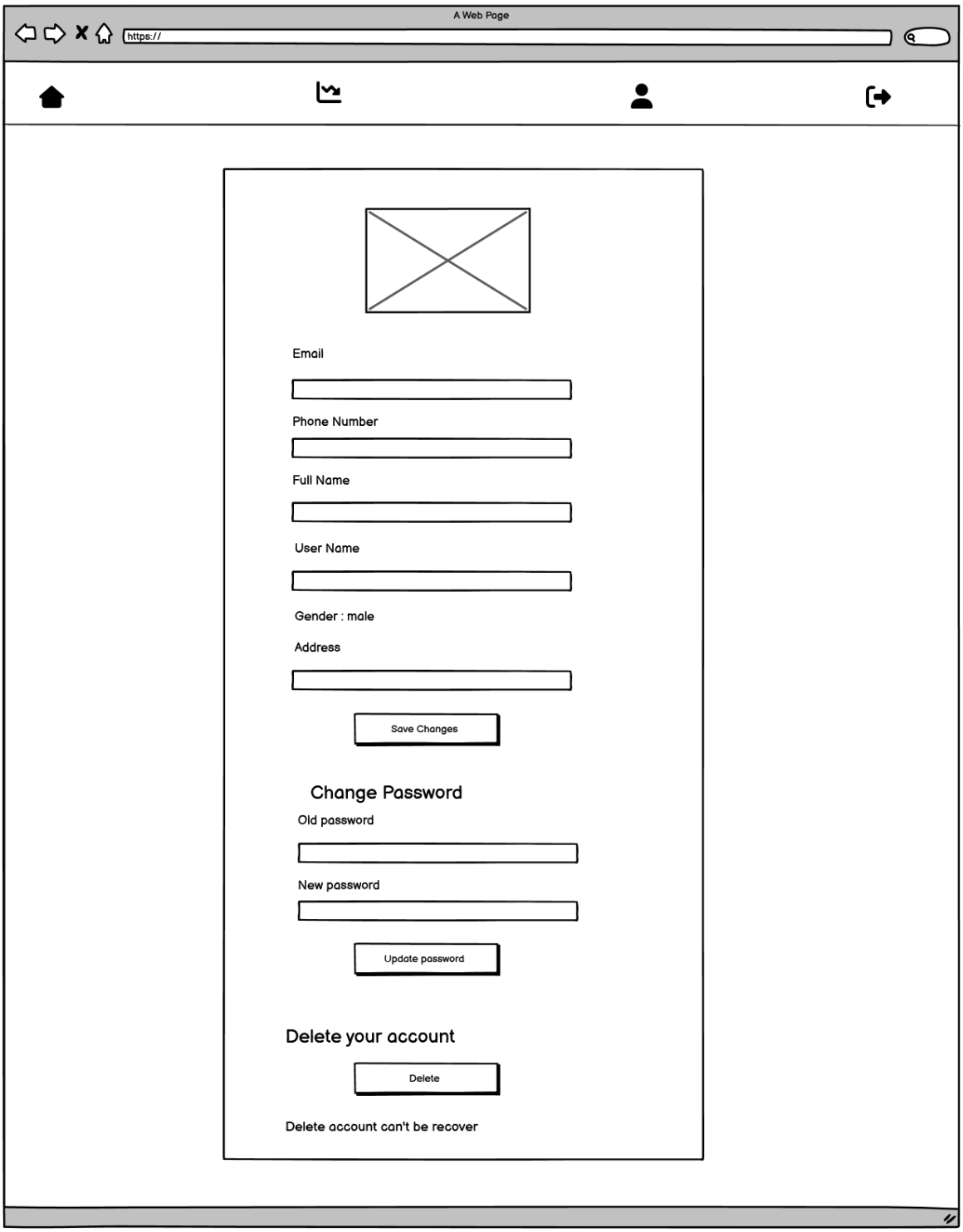


Figure 5 Wire Frame of User profile page.

2.1.6 Admin Dashboard

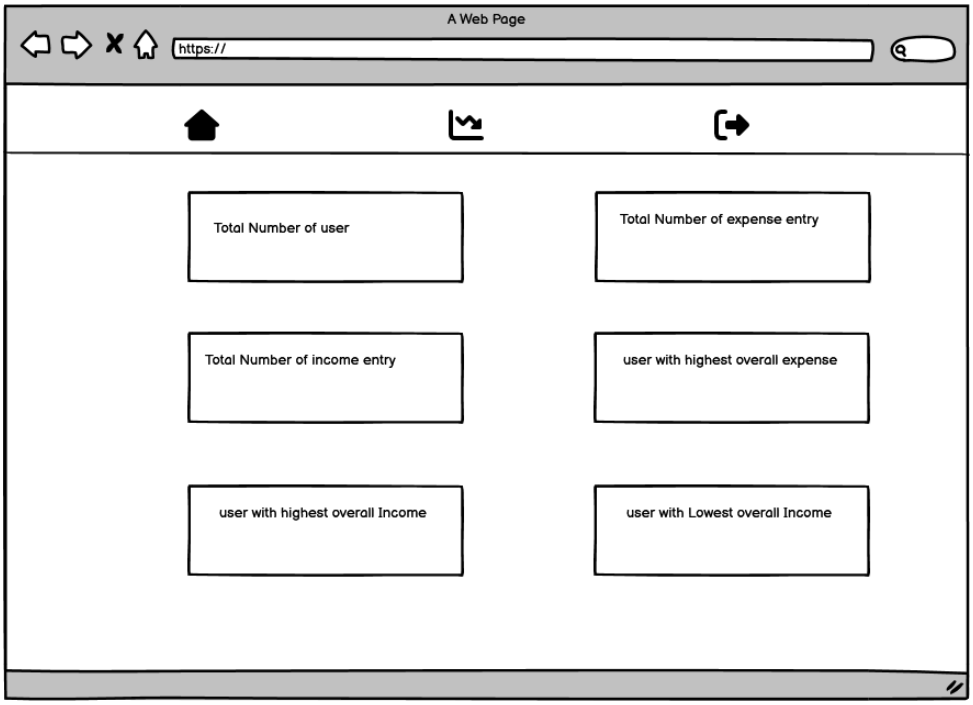


Figure 6 Wire Frame of Admin Dashboard

2.1.7 User dashboard

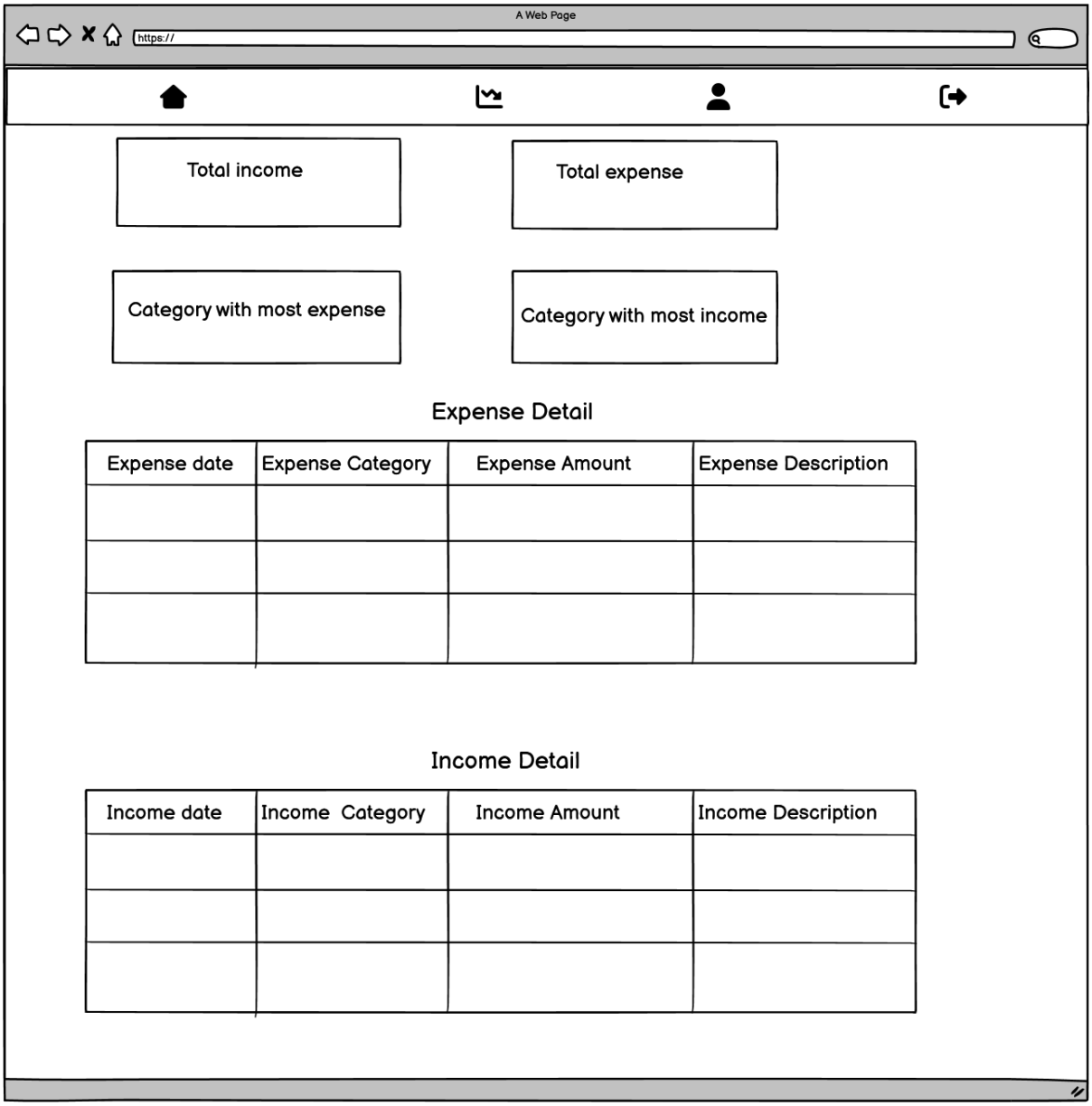


Figure 7 Wire frame for user dashboard.

2.1.8 Welcome Page

Figure 8 Wireframe of Welcome page.

2.2 Actual Design

2.2.1 Signup Page

Figure 9 Actual Design of Signup page

2.2.2 Login page

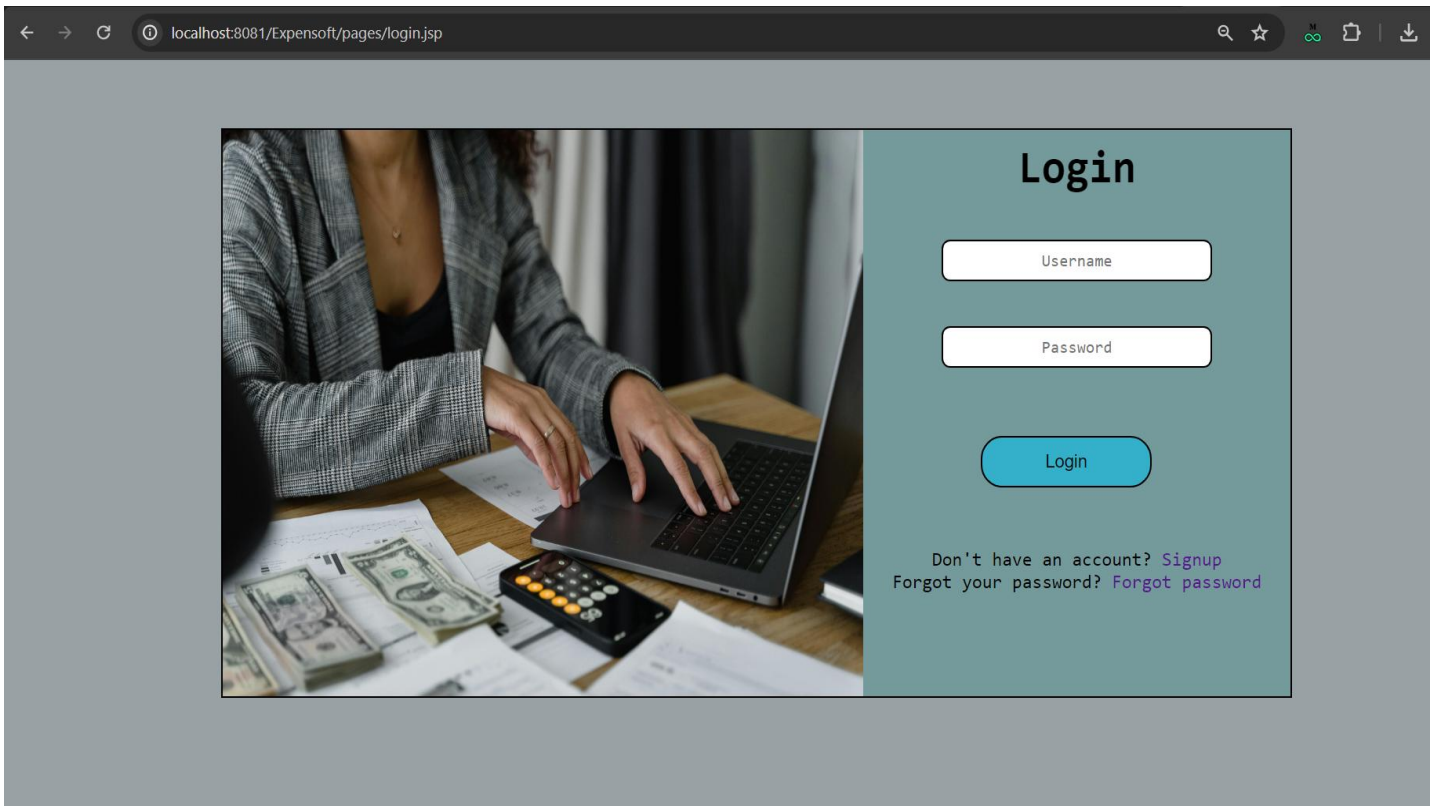


Figure 10 Actual Design of Login Page.

2.2.3 User home page

Figure 11 Actual Desing of User home page.

2.2.4 Actual Design of Admin home page

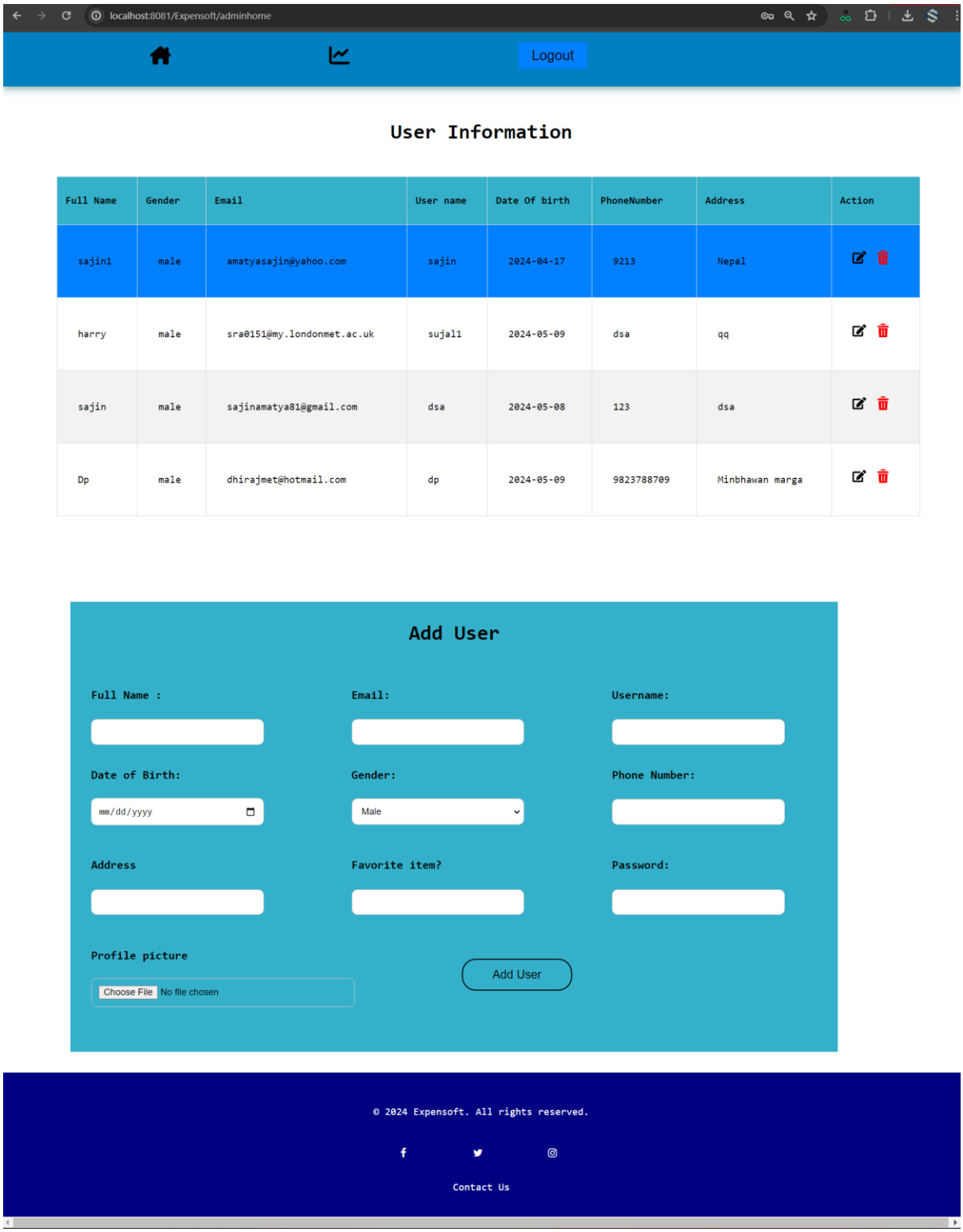


Figure 12 Actual Design of Admin home page.

2.2.5 Actual Design of user profile

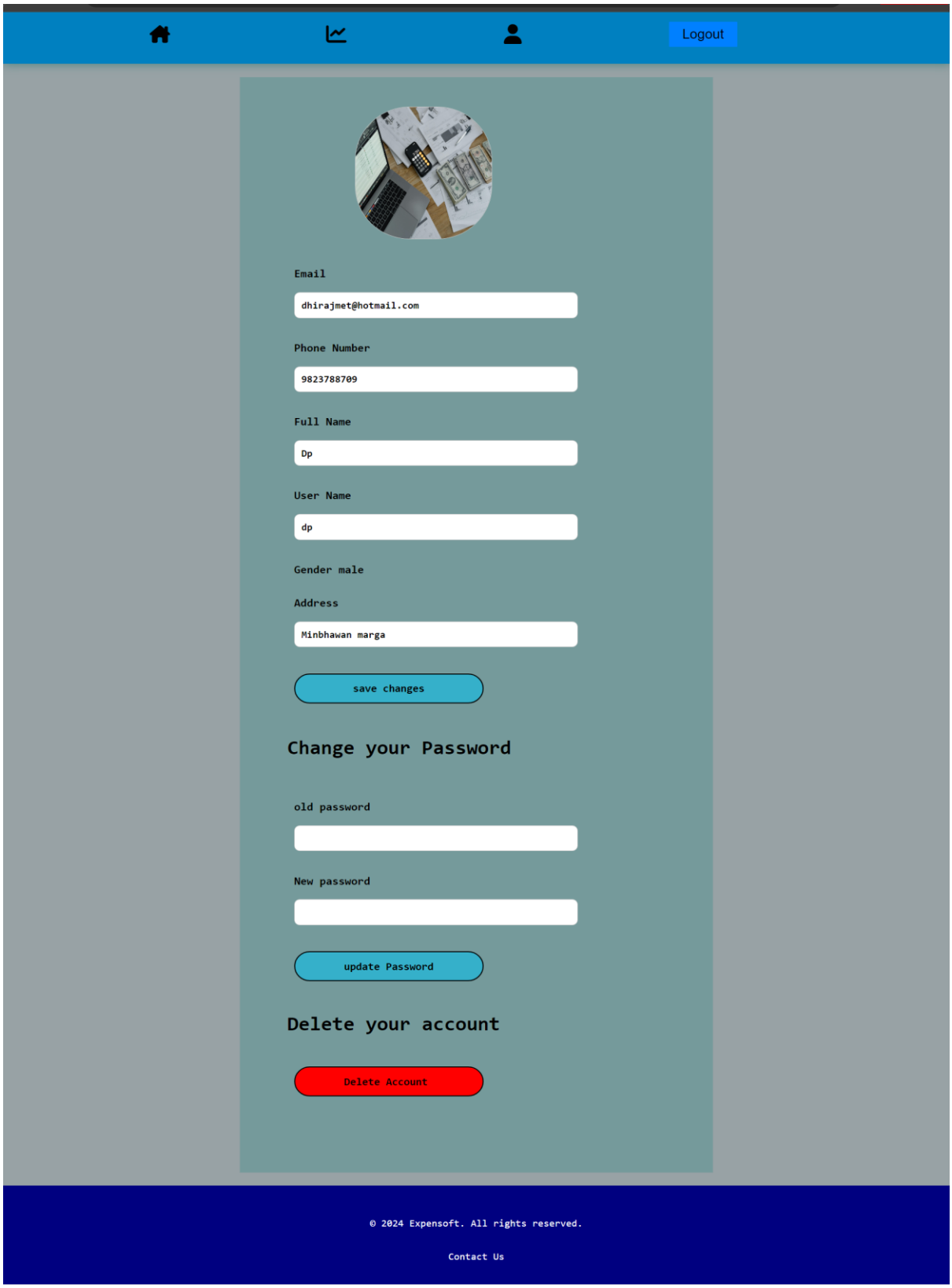


Figure 13 Actual Design of user profile.

2.2.6 Actual Design of User dashboard

Figure 14 Actual design of user dashboard.

2.2.7 Actual image of the Admin Dashboard

Figure 15 Actual design of the Admin Dashboard

2.2.8 Actual Design of the Home page

Figure 16 Actual design of the home page

3. Class Diagram

3.1 Combined class diagram

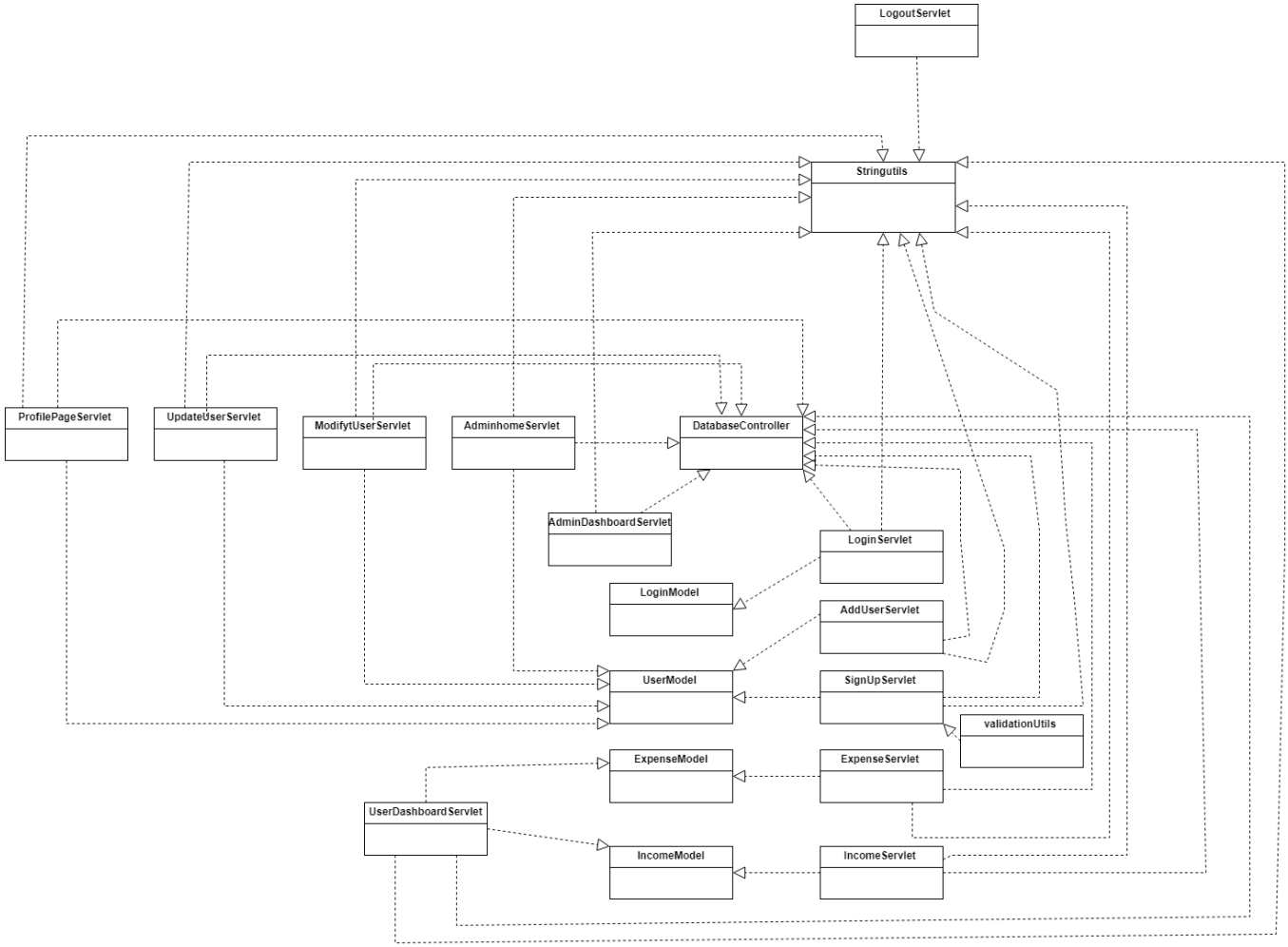


Figure 17 Overall Class Diagram

3.2 Individual class diagram

3.2.1 Individual UserModel class diagram

UserModel
- fullName : String - email : String - userName :String - gender : String - phoneNumber : String - address : String - password : String - securityQn : String - dateOfBirth : LocalDate - imageUrlFromPart : String
<<constructor>> + UserModel() + UserModel(String fullName, String email, String userName, String gender, String phoneNumber, String address, String password, String securityQn, LocalDate dateOfBirth, Part imagePart) + UserModel(String fullName, String email, String userName, String phoneNumber, String address) - getImageUrl(Part part) : String + setDateOfBirth(LocalDate dateOfBirth) : void + getDateOfBirth : LocalDate + LocalDate getDateOfBirth() : void + setSecurityQn(String securityQn) : void + getSecurityQn() : String + setPassword(String password) : void + getPassword() : String + setAddress(String address) : void + getAddress() : String + setPhoneNumber(String phoneNumber) : void + getPhoneNumber() : String + setGender(String gender) : void + getGender() : String + setUserName(String userName) : void + getUserName() : String

+ setEmail(String email) : void
+ getEmail(): String
+ setFullName(String fullName) : String
+ getFullName() : String
+ setImageUrlFromPart(String imageUrlFromPart) : void
+ getImageUrlFromPart() : String

Table 1 Individual Class Diagram of UserModel

3.2.2 Individual Class Diagram of loginModel

loginModel
- Username: String - password : String
<<constructor>> + LoginModel(String userName, String password) + getUsername() : String + setUsername(String userName) : String + setPassword(String password) : String + getPassword() : String

Table 2 individual Class Diagram of Login Model

3.2.3 Individual class Diagram of ExpenseModel

ExpenseModel
- user_id : int - expense_amount : Float - expense_date : LocalDate - expense_category : String - expense_description : String
<<constructor>> + ExpenseModel() + ExpenseModel(int user_id, float expense_amount, LocalDate expense_date, String expense_category, String expense_description) + getExpense_amount() : Float + getExpense_date() : LocalDate + setExpense_date(LocalDate expense_date) : void + setExpense_amount(float expense_amount) : void + getExpense_category() : String + setExpense_category(String expense_category) : void + getExpense_description() : String + setExpense_description(String expense_description) : void + getUser_id() : Int + setUser_id(int user_id) : void

Table 3 Individual Class Diagram of ExpenseModel

3.2.4 Individual class diagram of IncomeModel

IncomeModel
- user_id : int - income_amount : Float - income_date : LocalDate - income_category : String - income_description : String
<<constructor>> + IncomeModel() + IncomeModel(int user_id, float expense_amount, LocalDate expense_date, String expense_category, String expense_description) + getIncome_amount() : Float + getIncome_date() : LocalDate + setIncome_date(LocalDate expense_date) : void + setIncome_amount(float expense_amount) : void + getIncome_category() : String + setIncome_category(String expense_category) : void + getIncome_description() : String + setIncome_description(String expense_description) : void + getUser_id() : Int + setUser_id(int user_id) : void

Table 4 Individual Class Diagram of IncomeModel

3.2.5 Individual class diagram of DatabaseController

DatabaseController
+ getConnection() : Connection + signupUser(UserModel user) ; int + checkEmailIfExists(String email) : Boolean + checkNumberIfExists(String number) : Boolean + checkUsernameIfExists(String username) : Boolean + getUserLoginInfo(LoginModel loginModel) : int + extractUser_Id(String username) : int + userExpense(ExpenseModel expense) : int + getCategoryExpenseOfUser(int user_id) : Map<String, Double> + getCategoryIncomeOfUser(int user_id) : Map<String, Double> + getTotalExpenseOfUser(int user_id) : double + getTotalIncomeOfUser(int user_id) : double + getTotalNoOfUser() : int + userIncome(IncomeModel income) : int + updateUserProfile(UserModel user, int user_id) : int + changePassword(String newpass, String oldpass, String username) : int + getAllIncomeUserInfo(int user_id) : ArrayList<IncomeModel> + getAllExpenseUserInfo(int user_id) : ArrayList<ExpenseModel> + getAllUserInfo() : ArrayList<UserModel> + setUserdetailToSession(String username,HttpServletRequest request, HttpServletResponse response) : void + getTotalNoExpenseOfALLUser() : int + getTotalNoIncomeOfALLUser() : int + getHighestExpenseOfALLUser() : Map<String, Float> + getHighestIncomeOfALLUser() : Map<String, Float> + deleteUserInfo(String username) : int

Table 5 Individual Class diagram of Database Controller

3.2.6 Individual class diagram of Stringutils

Stringutils
+ DRIVER_NAME: String + LOCALHOST_URL: String + LOCALHOST_USERNAME: String + LOCALHOST_PASSWORD: String + QUERY_SIGNUP_USER: String + QUERY_USER_EXPENSE: String + QUERY_USER_INCOME: String + QUERY_USER_CHECK: String + QUERY_USER_CHECKPASSWORD: String + QUERY_GET_ALL_USER: String + QUERY_GET_ALL_INCOME: String + QUERY_GET_ALL_EXPENSE: String + QUERY_CHECK_EMAIL: String + QUERY_CHECK_PHONENUMBER: String + QUERY_CHECK_USERNAME: String + QUERY_GET_USER_ID: String + QUERY_DELETE_USER: String + QUERY_DELETE_USER_EXPENSES: String + QUERY_DELETE_USER_INCOMES: String + QUERY_UPDATE_USER_PROFILE: String + QUERY_USER_PASSWORD_CHECK: String +QUERY_USER_PASSWORD_UPDATE: String + QUERY_TOTAL_INCOME: String + QUERY_TOTAL_EXPENSE: String + QUERY_EXPENSE_CATEGORY: String + QUERY__INCOME_CATEGORY: String + QUERY_TOTAL_USER: String + QUERY_TOTAL_EXPENSE_ENTRY: String + QUERY_TOTAL_INCOME_ENTRY: String + QUERY_USER_MOST_EXPENSE: String + QUERY_USER_MOST_INCOME: String + USERNAME: String + FULL_NAME: String + ADDRESS: String + SECURITY_QUESTION: String + DATE_OF_BIRTH: String + GENDER: String + EMAIL: String + PHONE_NUMBER: String + PASSWORD: String + NEW_PASSWORD: String + RETYPE_PASSWORD: String

+ USER_NAME: String
+ USER_ID: String

+ EXPENSE_AMOUNT: String
+ EXPENSE_CATEGORY: String
+ EXPENSE_DATE: String
+ INCOME_DESCRIPTION: String

+ MESSAGE_SUCCESS_REGISTER: String

+ MESSAGE_ERROR_SIGNUP : 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_ERROR_PASSWORD: String

+ MESSAGE_SUCCESS_LOGIN

+ MESSAGE_SUCCESS_PASSWORD: String

+ MESSAGE_ERROR_LOGIN: String

+ MESSAGE_ERROR_CREATE_ACCOUNT: String

+ MESSAGE_SUCCESS_PROFILE: String

+ MESSAGE_ERROR_SERVER: String

+ MESSAGE_SUCCESS_DELETE: String

+ MESSAGE_ERROR_DELETE: String

+ MESSAGE_ERROR_PROFILE: String

+ MESSAGE_SUCCESS: String

+ MESSAGE_ERROR: String

+ PAGE_URL_LOGIN: String

+ PAGE_URL_SIGNUP: String

+ PAGE_URL_USER_HOME: String

+ PAGE_URL_HEADER: String

+ PAGE_URL_ADMIN: String

+ PAGE_URL_ADMIN_HOME: String

+ PAGE_URL_PROFILE: String

+ PAGE_URL_INDEX: String

+ PAGE_URL_USERDASHBOARD: String
+ SERVLET_URL_ADMIN_HOME: String
+ SERVLET_URL_ADMIN_ADD: String
+ SERVLET_URL_USER_DASHBOARD: String
+ SERVLET_URL_ADMIN_DASHBOARD: String
+ SERVLET_URL_LOGIN: String
+ SERVLET_URL_SIGNUP: String
+ SERVLET_URL_LOGOUT: String
+ SERVLET_URL_EXPENSE: String
+ SERVLET_URL_INCOME: String
+ SERVLET_URL_MODIFY_USER: String
+ SERVLET_URL_PROFILE: String
+ USER: String
+ SUCCESS: String
+ TRUE: String
+ JSESSIONID: String
+ LOGIN: String
+ LOGOUT: String
+ USER_LIST: String
+ DELETE_ID: String
+ UPDATE_ID: String
+ IMAGE_DIR: String
+ IMAGE_DIR_SAVE_PATH: String
+ IMAGE_ROOT_PATH: String
+ IMAGE_DIR_USER: String

Table 6 Individual Class diagram of Stringutils

3.2.7 Individual class diagram of LoginServlet

LoginServlet
- serialVersionUID : long - DatabaseController : DatabaseController
<<constructor>. + LoginServlet() # doPost(HttpServletRequest request, HttpServletResponse response) : void

Table 7 Individual Class diagram of Login Servlet

3.2.8 Individual class diagram of LogoutServlet

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

Table 8 Individual class diagram of Logout Servlet

3.2.9 Individual class diagram of SignupServlet

SignupServlet
- serialVersionUID : long - databaseController : DatabaseController
doPost(HttpServletRequest request, HttpServletResponse response) : void

Table 9 individual class diagram of Signup Servlet

3.2.10 Individual class diagram of ExpenseServlet

ExpenseServlet
- serialVersionUID : long - DatabaseController :DatabaseController
<<constructor>> + ExpenseServlet() # doPost(HttpServletRequest request, HttpServletResponse response) : void

Table 10 individual Class diagram for Expense Servlet

3.2.11 Individual class diagram of IncomeServlet

IncomeServlet
- serialVersionUID : long - DatabaseController :DatabaseController
<<constructor>> + IncomeServlet() # doPost(HttpServletRequest request, HttpServletResponse response) : void

Table 11 Individual Class diagram of Income Servlet

3.2.12 Individual Class diagram of AdminHomeServlet

AdminHomeServlet
- serialVersionUID : long - DatabaseController :DatabaseController
<<constructor>> + AdminHomeServlet() # doPost(HttpServletRequest request, HttpServletResponse response) : void # doGet(HttpServletRequest request, HttpServletResponse response)

Table 12 Individual Class Diagram of AdminHome Servlet

3.2.13 Individual Class diagram of ProfilePageServlet

ProfilePageServlet
- serialVersionUID : long - DatabaseController :DatabaseController
<<constructor>> + ProfilePageServlet() # doPost(HttpServletRequest request, HttpServletResponse response) : void # doGet(HttpServletRequest request, HttpServletResponse response) : void # doDelete(HttpServletRequest req, HttpServletResponse resp) : void

Table 13 Individual Class Diagram of ProfilePageServlet

3.2.14 Individual Class diagram of UserDashboardServlet

UserDashboardServlet
- serialVersionUID : long - controller : DatabaseController
doGet(HttpServletRequest request, HttpServletResponse response) : void

Table 14 Individual Class diagram of UserDashboardServlet

3.2.14 Individual Class diagram of AdminDashboardServlet

AdminDashboardServlet
- serialVersionUID : long - controller : DatabaseController
doGet(HttpServletRequest request, HttpServletResponse response) : void

Table 15 individual Class diagram of AdminDashboardServlet

3.2.15 Individual class diagram of UpdateUserServlet

UpdateUserServlet
- serialVersionUID : long - controller : DatabaseController
<<constructo>> + UpdateUserServlet() : # doPost(HttpServletRequest request, HttpServletResponse response) : void

Table 16 Individual class diagram of UpdateUserServlet

3.2.16 Individual class diagram of ModifyUserServlet

ModifytUserServlet
- serialVersionUID : long - dbController: DatabaseController
<<constructor>> + ModifyUserServlet () : # doPost(HttpServletRequest request, HttpServletResponse response) : void # doPut(HttpServletRequest req, HttpServletResponse resp) : void # doDelete(HttpServletRequest req, HttpServletResponse resp) : void

Table 17 individual Class diagram of ModifyUserServlet

3.2.17 Individual class diagram of AddUserServlet

AddUserServlet
- serialVersionUID : long - controller: DatabaseController
<<constructor>> + AdduserServlet() : # doPost(HttpServletRequest request, HttpServletResponse response) : void

Table 18 Individual class diagram of AddUserServlet

3.2.18 Individual Class diagram of ValidationUtils

ValidationUtils
+ isTextOnly(String text) : Boolean + isNumbersOnly(String text) : Boolean + isAlphanumeric(String text) : Boolean + isEmail(String email) : Boolean + hasNoSpecialCharacters(String text): Boolean + isValidPassword(String password) : Boolean

Table 19 Individual Class diagram of ValidationUtils

4. Database details

4.1 Normalization

4.1.1 UNF
All the attributes are listed down

User(**user_id**, user_name, password, security key, date_of_birth, email, gender, full_name , phone_number, address, user_image { **expense_id**, expense_amount

expense_date, expense_category, expense_description ,

income_id, income_amount, income_date, income_category,

income_description})

4.1.2 1NF
The repeating group are separated into new tables with primary and foreign key identification.

User-1 (**user_id,** user_name, password, security key, date_of_birth, email, gender, full_name , phone_number, address,user_image)

Expense-income-1(**user_id**, **expense_id**,expense_amount,expense_date,

expense_category, expense_description, **income_id**, income_amount, income_date , income_category, income_description)

4.1.3 2NF
The partial functional dependency is removed into new table in second normal form.

For user table

user_id -> user_name, password, security key, date_of_birth, email, gender, full_name , phone_number, address, user_image

user_id,expense_id,income_id -> ×

expense_id,income_id -> ×

user_id, expense_id -> expense_amount,expense_date, expense_category, expense_description

user_id, income_id -> income_amount, income_date , income_category, income_description

User-2 (user_id, user_name, password, security key, date_of_birth, email, gender, full_name , phone_number, address)

Expense-2(user_id, expense_id,expense_amount,expense_date,

expense_category, expense_description)

Income – 2 (user_id, income_id, income_amount, income_date ,

income_category, income_description)

4 .1.3 3NF

The transitive dependencies is separated into new table

There aren't any transitive dependencies between non-prime attributes so the final table are

User-3 (user_id, user_name, password, security key, date_of_birth, email, gender, full_name , phone_number, address)

Expense-3 (user_id, expense_id,expense_amount,expense_date,

expense_category, expense_description)

Income – 3 (user_id, income_id, income_amount, income_date ,

income_category, income_description)

3.2 Final ERD

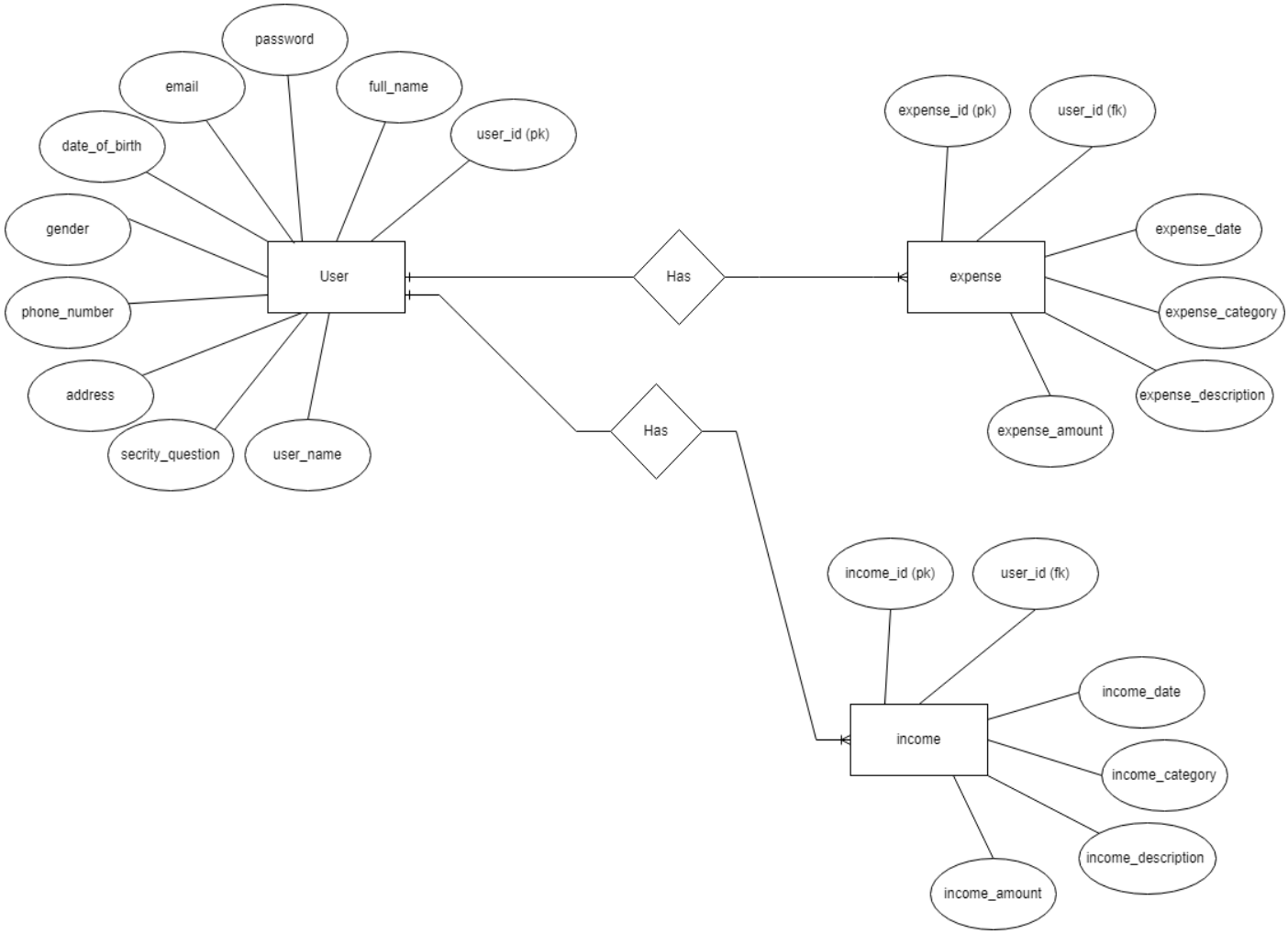


Figure 18 Final ERD of the database

4.2 Database Design

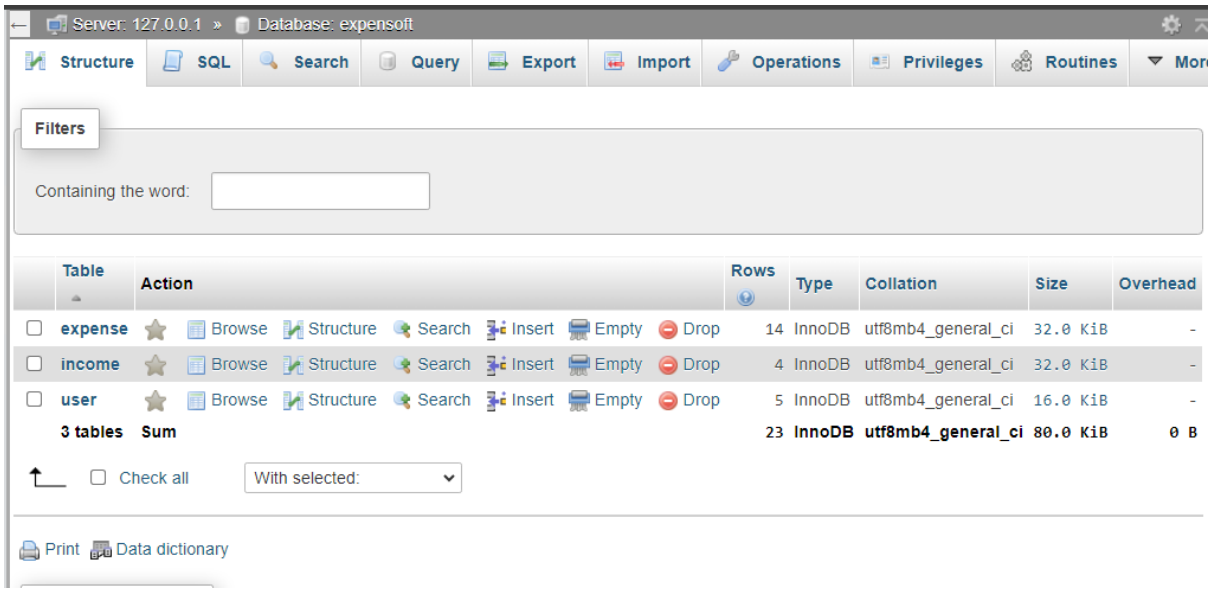


Figure 19 Database and Overall table

4.3 Table Design

4.3.1 expense table

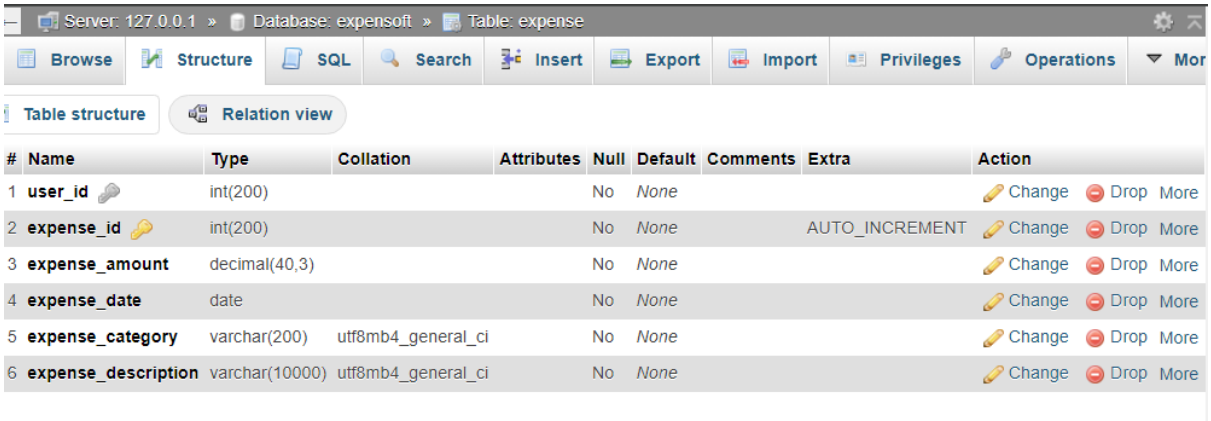


Figure 20 Structure of expense table

4.3.2 Income Table

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	user_id	int(200)			No	None			Change Drop More
2	income_id	int(200)			No	None		AUTO_INCREMENT	Change Drop More
3	income_amount	decimal(40,3)			No	None			Change Drop More
4	income_date	date			No	None			Change Drop More
5	income_category	varchar(200)	utf8mb4_general_ci		No	None			Change Drop More
6	income_description	varchar(10000)	utf8mb4_general_ci		No	None			Change Drop More

Check all With selected: Browse Change Drop Primary Unique Index Spatial

Figure 21 Structure for income table

4.3.2 User table

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	user_id	int(200)			No	None		AUTO_INCREMENT	Change Drop More
2	full_name	varchar(200)	utf8mb4_general_ci		No	None			Change Drop More
3	email	varchar(300)	utf8mb4_general_ci		No	None			Change Drop More
4	date_of_birth	date			No	None			Change Drop More
5	gender	varchar(50)	utf8mb4_general_ci		No	None			Change Drop More
6	phone_number	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
7	address	varchar(200)	utf8mb4_general_ci		No	None			Change Drop More
8	user_name	varchar(200)	utf8mb4_general_ci		No	None			Change Drop More
9	password	varchar(2000)	utf8mb4_general_ci		No	None			Change Drop More
10	security_question	varchar(3200)	utf8mb4_general_ci		No	None			Change Drop More
11	image	varchar(300)	utf8mb4_general_ci		No	None			Change Drop More

Figure 22 Structure for user table.

5. Method description

SignupUser(UserModel user)

This method takes the UserModel model as parameter with integer return type. It inserts the data entered by the user from Signup page to the database for registration purposes by extracting the data stored in the model class.

```
/**
 * This method register the user and save their details into the database
 * @param user object of SignupModel class
 * @return
 */
public int signupUser(UserModel user) {

    try {
        // Prepare a statement using the predefined sql query for user sign up
        PreparedStatement st = getConnection()
                .prepareStatement(Stringutils.QUERY_SIGNUP_USER);

        // Set the user detail in the prepared statement
        st.setString(1, user.getFullName());
        st.setString(2, user.getEmail());
        st.setDate(3, Date.valueOf(user.getDateOfBirth()));
        st.setString(4, user.getGender());
        st.setString(5, user.getPhoneNumber());
        st.setString(6, user.getAddress());
        st.setString(7, user.getUserName());
        st.setString(8, PasswordEncryptionWithAes.encrypt(
                user.getUserName(), user.getPassword()));
        st.setString(9,user.getSecurityQn());
        st.setString(10, user.getImageUrlFromPart());

        // Execute the update statement and store the number of affected rows
        int result = st.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 23 SignupUser method

CheckNumberIfExists(String number)

This method takes String as a parameter with return type Boolean. It checks if the user entered number from signup page already exists in the database or not with the help of SQL query defined for checking number existence.

```
/**
 * This method check if the Phone number exist in the database or not
 * @param number
 * @return
 */
public Boolean checkNumberIfExists(String number) {

    try {
        PreparedStatement stm = getConnection()
                .prepareStatement(Stringutils.QUERY_CHECK_PHONENUMBER);
        stm.setString(1, number);
        ResultSet result = stm.executeQuery();
        if(result.next()) {
            return true;
        }
     }
    catch(ClassNotFoundException | SQLException ex)
    {
          ex.printStackTrace();

    }
    return false;
}
```

Figure 24 CheckNumberIfExists method

CheckEmailIfExists(String email)

This method takes String as a parameter with return type Boolean. It checks if the user entered email address from signup page already exists in the database or not with the help of SQL query defined for checking email existence.

```
/**
 * This method check if the email is already exist or not
 * @param email
 * @return
 */
public Boolean checkEmailIfExists(String email) {

    try {
        PreparedStatement stm = getConnection()
                .prepareStatement(Stringutils.QUERY_CHECK_EMAIL);
        stm.setString(1, email);
        ResultSet result = stm.executeQuery();
        if(result.next()) {
            return true;
        }
     }
    catch(ClassNotFoundException | SQLException ex)
    {
          ex.printStackTrace();

    }
    return false;

}
```

Figure 25 CheckEmailIfExists method

CheckUsernameIfExists(String username)

This method takes String as a parameter with return type Boolean. It checks if the user entered username from signup page already exists in the database or not with the help of SQL query defined for checking username existence.

```
/**
 * This method check if the username exist in the database or not
 * @param username
 * @return
 */
public Boolean checkUsernameIfExists(String username) {
    try {
        PreparedStatement stm = getConnection()
                .prepareStatement(Stringutils.QUERY_CHECK_USERNAME);
        stm.setString(1, username);
        ResultSet result = stm.executeQuery();
        if(result.next()) {
            return true;
        }
     }
    catch(ClassNotFoundException | SQLException ex)
    {
          ex.printStackTrace();

    }
    return false;

}
```

Figure 26 CheckUsernameIFExists method

getUserLoginInfo(LoginModel loginModel)

This method takes LoginModel Model as parameter with a return type of integer. It validates if the entered user name and password by the user from login page matches the data of system database then, it authenticate the use to the user home page after login process succeed.

```
/**
 * This method perform login operation by checking if the user entered username and password matches the database data and redirect them to the system page
 * @param loginModel
 * @return
 */
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_USER_CHECK);

        // Set the user name 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.USER_NAME);

            // 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("admin") && decryptedPwd.equals("admin")) {
                return 3;
            }
            else 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 27 getUserLoginInfo Method

extractUser_Id(String username)

This method takes username as a parameter with return type integer. It retrieves the the user Id of the user from the database by performing SQL query define for it and return the user Id when the method is called.

```
/**
 * This method extract user_id of the user from the database
 * @param username
 * @return
 */
public int extractUser_Id(String username) {
    try{
        PreparedStatement st = getConnection()
                .prepareStatement(Stringutils.QUERY_GET_USER_ID);
        st.setString(1, username);
        ResultSet result = st.executeQuery();
        if(result.next()) {
            int user_id = result.getInt(Stringutils.USER_ID);
            return user_id;
        }
        else { return -1; }


    }
    catch(ClassNotFoundException | SQLException ex) {
        // Print the stack trace for debugging purposes
        ex.printStackTrace();
        return -2; // Internal error
    }

}
```

Figure 28 extractUser_id method

userExpense(ExpenseModel expense)

This method takes ExpenseModel model object as a parameter with return type integer. It adds the user enter expenses with its detail to the expense database where the user entered data is extracted from the model object. The SQL query that is already defined to perform expense add is executed.

```
/**
 *  This method add the expense of the user to the system
 * @param expense  entered by the user
 * @return
 */
public int userExpense(ExpenseModel expense) {
    try {

        PreparedStatement st = getConnection()
                .prepareStatement(Stringutils.QUERY_USER_EXPENSE);
         // Set the user detail in the prepared statement
        st.setInt(1,expense.getUser_id());
        st.setFloat(2,expense.getExpense_amount() );
        st.setDate(3, Date.valueOf(expense.getExpense_date()));
        st.setString(4, expense.getExpense_category());
        st.setString(5, expense.getExpense_description());


        // Execute the update statement and store the number of affected rows
        int result = st.executeUpdate();
      // Check if the update was successful (i.e., at least one row affected)
        if (result > 0) {
            return 1; // Expense added successful
        }
        else {
            return 0; // Expense adding process  failed (no rows affected)
        }



    }

    // Check if the update was successful (i.e., at least one row affected)

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

}
```

Figure 29 userExpense method

getCategoryExpenseOfUser(int user_id)

This method takes user id as a parameter with the return type Map which contains string and double value. It is used to get the Highest expense by expense categories of the user calculated by using SQL queries define for it.

```
/**
 * This method perform sql queries for extracting the highest total expense by category.
 * @param user_id
 * @return categoryexpense
 */
public Map<String, Double> getCategoryExpenseOfUser(int user_id) {
    Map<String, Double> ExpenseMap = new HashMap<>();

    try {
        PreparedStatement st1 = getConnection().prepareStatement(Stringutils.QUERY_EXPENSE_CATEGORY);
        st1.setInt(1, user_id);
        ResultSet result = st1.executeQuery();

        while(result.next())
        {
            String expensecategory = result.getString("expense_category");
            double categoryexpense = result.getDouble("total_expense");
            ExpenseMap.put(expensecategory, categoryexpense);

        }

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

    }
    return ExpenseMap;
}
```

Figure 30 getCategoryExpenseOfUser method

getCategoryIncomeOfUser(int user_id)

This method takes user id as a parameter with the return type Map which contains string and double value. It is used to get the Highest income by income categories of the user calculated by using SQL queries define for it.

```
/**
 * This method perform sql queries for extracting the highest total income by category.
 * @param user_id
 * @return
 */
public Map<String, Double> getCategoryIncomeOfUser(int user_id) {
    Map<String, Double> IncomeMap = new HashMap<>();

    try {
        PreparedStatement st1 = getConnection().prepareStatement(Stringutils.QUERY__INCOME_CATEGORY);
        st1.setInt(1, user_id);
        ResultSet result = st1.executeQuery();

        while(result.next())
        {
            String incomecategory = result.getString("income_category");
            double incomeam = result.getDouble("total_income");
            IncomeMap.put(incomecategory,incomeam);

        }

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

    }
    return IncomeMap;
```

Figure 31 getCategoryIncomeOfUser method

getTotalExpenseOfUser(int user_id)

This method takes user_id as a parameter with the return type Double. It is used to calculate the sum of all the expense of the user from the database with the help of SQL Query defined for it.

```
/**
 * This method return the sum of total expense of the user
 * @param user_id
 * @return totalexpense
 */
public double getTotalExpenseOfUser(int user_id) {

    try {
        PreparedStatement st1 = getConnection().prepareStatement(Stringutils.QUERY_TOTAL_EXPENSE);
        st1.setInt(1, user_id);
        ResultSet result = st1.executeQuery();

        if(result.next())
        {

            double totalexpense = result.getDouble("total_expense");

            return totalexpense;
        }
        else {
            return 1;
        }
    }
    catch (ClassNotFoundException | SQLException ex) {
        // Print the stack trace for debugging purposes
        ex.printStackTrace();
        // Internal error
        return 0;
    }
}
```

Figure 32 getTotalExpenseOfUser method

getTotalIncomeOfUser(int user_id)

This method takes user_id as a parameter with the return type Double. It is used to calculate the sum of all the income of the user from the database with the help of SQL Query defined for it

```
/**
 * This method return the sum of total expense of the user
 * @param user_id
 * @return totalexpense
 */
public double getTotalExpenseOfUser(int user_id) {

    try {
        PreparedStatement st1 = getConnection().prepareStatement(Stringutils.QUERY_TOTAL_EXPENSE);
        st1.setInt(1, user_id);
        ResultSet result = st1.executeQuery();

        if(result.next())
        {

            double totalexpense = result.getDouble("total_expense");

            return totalexpense;
        }
        else {
            return 1;
        }
    }
    catch (ClassNotFoundException | SQLException ex) {
        // Print the stack trace for debugging purposes
        ex.printStackTrace();
        // Internal error
        return 0;
    }
}
```

Figure 33 getTotalExpenseOfUser() method

getTotalNoOfUser()

This method takes no parameter with return type void. It retrieves the total number of of the user from the database with the help of defined SQL query.

```
/**
 * This method Reform the sql query to get total number of user from the database
 * @return total number of user
 */
public int getTotalNoOfUser() {

    try {
        PreparedStatement st2 = getConnection().prepareStatement(Stringutils.QUERY_TOTAL_USER);

        ResultSet result = st2.executeQuery();

        if(result.next())
        {

            int totaluser = result.getInt("totaluser");

            return totaluser;
        }
        else {
            return 1;
        }
    }
    catch (ClassNotFoundException | SQLException ex) {
        // Print the stack trace for debugging purposes
        ex.printStackTrace();
        // Internal error
        return 0;
    }
}
```

Figure 34 getTotalNoOfUser method

userIncome(IncomeModel income)

This method takes IncomeModel model as a parameter with return type integer. It inserts the user entered data related to their income in the database by executing SQL query.

```
/**
 * This method adds the income of the user to the system
 * @param income  given by the user
 * @return
 */
public int userIncome(IncomeModel income) {
    try {

        PreparedStatement st = getConnection()
                .prepareStatement(Stringutils.QUERY_USER_INCOME);
         // Set the user detail in the prepared statement
        st.setInt(1,income.getUser_id());
        st.setFloat(2,income.getIncome_amount() );
        st.setDate(3, Date.valueOf(income.getIncome_date()));
        st.setString(4, income.getIncome_category());
        st.setString(5, income.getIncome_description());


        // Execute the update statement and store the number of affected rows
        int result = st.executeUpdate();
     // Check if the update was successful (i.e., at least one row affected)
        if (result > 0) {
            return 1; // Expense added successful
        }
        else {
            return 0; // Expense adding process  failed (no rows affected)
        }



    }


    // Check if the update was successful (i.e., at least one row affected)


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

Figure 35 userIncome() method

updateUserProfile(UserModel user, int user_id)

This method takes UserModel model and user_id as a parameter with the return type of integer. This method updates the user data to the database using update SQL query where the required data is extracted from the model.

```
/**
 * This method updates the user detail of the user by performing update query
 * @param user
 * @param user_id
 * @return integer
 */
public int updateUserProfile(UserModel user, int user_id) {

    try {
        PreparedStatement stmt = getConnection().prepareStatement(Stringutils.QUERY_UPDATE_USER_PROFILE);

         stmt.setString(1,user.getFullName());
         stmt.setString(2,user.getEmail());
         stmt.setString(3,user.getPhoneNumber());
         stmt.setString(4,user.getAddress());
         stmt.setString(5,user.getUserName());
         stmt.setInt(6,user_id);
         int result = stmt.executeUpdate();

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



    }
    catch (SQLException | ClassNotFoundException ex) {
        ex.printStackTrace();
        return -1;
    }

}
```

Figure 36 updateUserProfile() method

getAllIncomeUserInfo(int user_id)

This method takes user_id as the parameter with the return type Array List of IncomeModel model. It extracts all the detail regarding the user income from the database, stores it in the array list and return it when this method is called.

```
/**
 * This method gets all the user income details and stores it to the arraylist
 * @param user_id
 * @return ArrayList<IncomeModel>
 */
public ArrayList<IncomeModel> getAllIncomeUserInfo(int user_id) {
    try {
        PreparedStatement stmt = getConnection().prepareStatement(Stringutils.QUERY_GET_ALL_INCOME);
        stmt.setInt(1, user_id);
        ResultSet result = stmt.executeQuery();

        ArrayList<IncomeModel> IncomeArraylist = new ArrayList<IncomeModel>();

        while (result.next()) {
            IncomeModel Income = new IncomeModel();

            Income.setIncome_amount(result.getFloat("Income_amount"));
            Income.setIncome_category(result.getString("Income_category"));
            Income.setIncome_description(result.getString("Income_description"));
            Income.setIncome_date(result.getDate("Income_date").toLocalDate());
            IncomeArraylist.add(Income);
        }

        return IncomeArraylist;
    } catch (SQLException | ClassNotFoundException ex) {
        ex.printStackTrace();
        return null;
    }

}
```

Figure 37 getAllIncomeUserInfo() method

changePassword(String newpass, String oldpass, String username)

This method takes newpass, oldpass and username as a parameter with return type integer. It changes the password of the user if the old password and new password matches and updates the data to the database with the help of defined SQL Query.

```
/**
 * This method update the password of the user
 * @param newpass   new password enter by user
 * @param oldpass    old password enter by user
 * @param username   username of user
 * @return
 */
public int changePassword( String newpass, String oldpass, String username ) {

    try{
        PreparedStatement st = getConnection()
            .prepareStatement(Stringutils.QUERY_USER_CHECK);
        st.setString(1, username);

        ResultSet result = st.executeQuery();

        if(result.next()) {

            String encryptedPwd = result.getString(Stringutils.PASSWORD);
            String decryptedPwd = PasswordEncryptionWithAes.decrypt(encryptedPwd, username);
            if(decryptedPwd.equals(oldpass)) {
                PreparedStatement st1 = getConnection()
                        .prepareStatement(Stringutils.QUERY_USER_PASSWORD_UPDATE);
                st1.setString(1, PasswordEncryptionWithAes.encrypt(
                        username, newpass));
                st1.setString(2, username);
                int result1 = st1.executeUpdate();

             // Check if the update was successful (i.e., at least one row affected)
                    if (result1 > 0) {
                    return 1; // user password  updated   successful
                    }
                    else {
                        return 0; // user updating password    process  failed (no rows affected)
                    }

            }
            else
            {

                return -1; // password didn't match
                }

        }
        else
        {return 2;}
    }
    catch(ClassNotFoundException | SQLException ex) {
        // Print the stack trace for debugging purposes
            ex.printStackTrace();
            return -2; // Internal error
    }
}
```

Figure 38 changePassword() method

getAllExpenseUserInfo(int user_id)

This method takes user_id as the parameter with the return type Array List of ExpenseModel model. It extracts all the detail regarding the user Expense from the database, stores it in the array list and return it when this method is called.

```
/** This method extract all the user expense details and stories it in the arraylist
 * @param user_id
 * @return
 */
public ArrayList<ExpenseModel> getAllExpenseUserInfo(int user_id) {
    try {
        PreparedStatement stmt = getConnection().prepareStatement(Stringutils.QUERY_GET_ALL_EXPENSE);
        stmt.setInt(1, user_id);
        ResultSet result = stmt.executeQuery();

        ArrayList<ExpenseModel> ExpenseArraylist = new ArrayList<ExpenseModel>();

        while (result.next()) {
            ExpenseModel expense = new ExpenseModel();
            expense.setExpense_amount(result.getFloat("expense_amount"));
            expense.setExpense_category(result.getString("expense_category"));
            expense.setExpense_description(result.getString("expense_description"));
            expense.setExpense_date(result.getDate("expense_date").toLocalDate());
            ExpenseArraylist.add(expense);
        }

        return ExpenseArraylist;
    } catch (SQLException | ClassNotFoundException ex) {
        ex.printStackTrace();
        return null;
    }

}
```

Figure 39 getAllExpenseUserInfo() method

getAllExpenseUserInfo()

This method takes no parameter with the return type Array List of UserModel model. It extracts all the detail of every user from the database, stores it in the array list and return it when this method is called.

```
/**
 * This method retreives details of all the user from the database  of the system and stores into ArrayList
 * @return ArrayList which contain detail of every user in the system
 */
public ArrayList<UserModel> getAllUserInfo() {
    try {
        PreparedStatement stmt = getConnection().prepareStatement(Stringutils.QUERY_GET_ALL_USER);
        ResultSet result = stmt.executeQuery();

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

        while (result.next()) {
            UserModel users = new UserModel();
            users.setFullName(result.getString("full_name"));
            users.setDateOfBirth(result.getDate("date_of_birth").toLocalDate());
            users.setEmail(result.getString("email"));
            users.setGender(result.getString("gender"));
            users.setPhoneNumber(result.getString("phone_number"));
            users.setUserName(result.getString("user_name"));
            users.setAddress(result.getString("address"));
            userArraylist.add(users);
        }

        return userArraylist;
    } catch (SQLException | ClassNotFoundException ex) {
        ex.printStackTrace();
        return null;
    }

}
```

Figure 40 getAllUserInfo() method

setUserdetailToSession ()

This method takes username, request, response as a parameter with the return type of void. It is called when the user login into the system and set the user detail to the session and cookies.

```
/**
 * This method retrieves the user detail from the database and create a session and update the details to it if the login is sucessful
 * @param username
 * @param request
 * @param response
 * @throws SQLException  ClassNotFoundException
 */
public void setUserdetailToSession(String username,HttpServletRequest request, HttpServletResponse response) {
    try {
        PreparedStatement stmt = getConnection().prepareStatement(Stringutils.QUERY_USER_CHECK);
        stmt.setString(1,username);
        ResultSet result = stmt.executeQuery();

        while (result.next()) {
            String user_id = String.valueOf(result.getInt("user_id"));
            String fullname = result.getString("full_name");
            String email = result.getString("email");
            String gender = result.getString("gender");
            String address = result.getString("address");
            String user_name = result.getString("user_name");
            String phone = result.getString("phone_number");
            Date date = result.getDate("date_Of_birth");
            String imageurl = result.getString("image");

            HttpSession session = request.getSession(true);
            session.setAttribute(Stringutils.USERNAME, user_name);
            session.setAttribute(Stringutils.USER_ID, user_id);
            session.setAttribute("email", email);
            session.setAttribute("gender", gender);
            session.setAttribute("address", address);
            session.setAttribute("phone", phone);
            session.setAttribute("fullname", fullname);
            session.setAttribute("image", imageurl);
            session.setAttribute("date",date);
            session.setMaxInactiveInterval(30*60);

            Cookie userCookie= new Cookie(Stringutils.USER, user_name);
            Cookie userCookieId = new Cookie("user_id",user_id);
            Cookie userCookieEmail = new Cookie("email", email);
            Cookie userCookiePhone = new Cookie("phonenumber",phone);
            Cookie userCookieGender = new Cookie("gender",gender);
            Cookie userCookieDate = new Cookie("date",date.toString());
            userCookie.setMaxAge(30*60);
            response.addCookie(userCookie);
            response.addCookie(userCookieId);
            response.addCookie(userCookieEmail);
            response.addCookie(userCookiePhone);
            response.addCookie(userCookieGender);
            response.addCookie(userCookieDate);
        }

    } catch (SQLException | ClassNotFoundException ex) {
        ex.printStackTrace();
```

Figure 41 setUserDetailToSession

deleteUserInfo(String username)

This method is takes username as the parameter with the return type of integer. It deletes the user detail with called by performing DELETE SQL query.

```
/**
 * This method perfrom deletion of the user account when called
 * @param username of the user
 * @return integer
 */
public int deleteUserInfo(String username) {
    try (Connection con = getConnection()) {

        try (PreparedStatement st1 = con.prepareStatement(Stringutils.QUERY_DELETE_USER_EXPENSES)) {
            st1.setString(1, username);
            st1.executeUpdate();
        }


        try (PreparedStatement st2 = con.prepareStatement(Stringutils.QUERY_DELETE_USER_INCOMES)) {
            st2.setString(1, username);
            st2.executeUpdate();
        }


        try (PreparedStatement st3 = con.prepareStatement(Stringutils.QUERY_DELETE_USER)) {
            st3.setString(1, username);
            return st3.executeUpdate();
        }
    }catch (SQLException | ClassNotFoundException ex) {
        ex.printStackTrace();
        return -1;
    }


}
```

Figure 42 deleteUsersInfo() method

isAlphanumberic()

This method check whether the user entered detail contains letter and digit only.

```
public static boolean isAlphanumeric(String text) {
    return text.matches("[a-zA-Z0-9]+"); // Match let
}
```

Figure 43 isAlphanumeric() method

getConnection()

This method is used to create the connection to the database which returns the database connection with given username, password and URL of the database we are trying to connect.

```
/**
 * Create a connection to the database using pre-defined credentials and driver information.
 *
 * @return A `Connection` object representing the established connection to the database.
 * @throws SQLException if a database access error occurs.
 * @throws ClassNotFoundException if the JDBC driver class is not found.
 */
public Connection getConnection() throws SQLException, ClassNotFoundException {

    // Load the JDBC driver class specified by the StringUtils.DRIVER_NAME constant
    Class.forName(Stringutils.DRIVER_NAME);

    // Create a connection to the database using the provided credentials from Stringutils
    return DriverManager.getConnection(Stringutils.LOCALHOST_URL, Stringutils.LOCALHOST_USERNAME,
                                       Stringutils.LOCALHOST_PASSWORD);
}
```

Figure 44 getConnection() method

doPost()

This method handles the HTTP post request sent by the client from the user interface to the server when the data is submitted by client.

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

Figure 45 doPost Method

doGet()

This method is used to handles the HTTP Get request sent by the client to the server which retrieves the data from the server and display it to the user interface for dynamic web content

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

Figure 46 doGet() method

doDelete()

This method is overridden by the HTTPServlet that is used to invoke the servlet when the delete request is send by the client.

```
@Override
protected void doDelete(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    System.out.println("delete triggered");
```

Figure 47 doDelete()

isTextOnly()

This method checks whether the entered text by the user in the text field contains only letters and white space only.

```
public static boolean isTextOnly(String text) {
    return text.matches("[a-zA-Z\\s]+"); // Mat
}
```

Figure 48 isTextOnly() method

isNumberOnly()

This method check whether the user enter detail in the text if contains only number only.

```
public static boolean isNumbersOnly(String text) {
    return text.matches("\\d+"); // Match digits only
}
```

Figure 49 isNumberOnly() method

6 Test Case

For user

Test 1: Testing the user Sign up and data insertion functionality in database.

Test No. 1	1
Objective	To Sign up the user and insert the user entered data in the database.
Action	➢ The following data was entered in the text field of the sign-up page. Full Name: Sajin Raj Amatya Email : sajinamatya81@gmail.com Username : sajinamatya Date of Birth : 05/14/2004 Gender : Male Phone number : 9823788709 Address : sankhamul Favorite item ? : Car Password : sajin Image ➢ Sign Up button was clicked to submit it
Expected Result	The account will be created with data insertion in database
Actual Result	The account was created with the data insertion in database
Conclusion	The Test is successful

Table 20 Test 1 Testing user signup and data insertion functionality in database

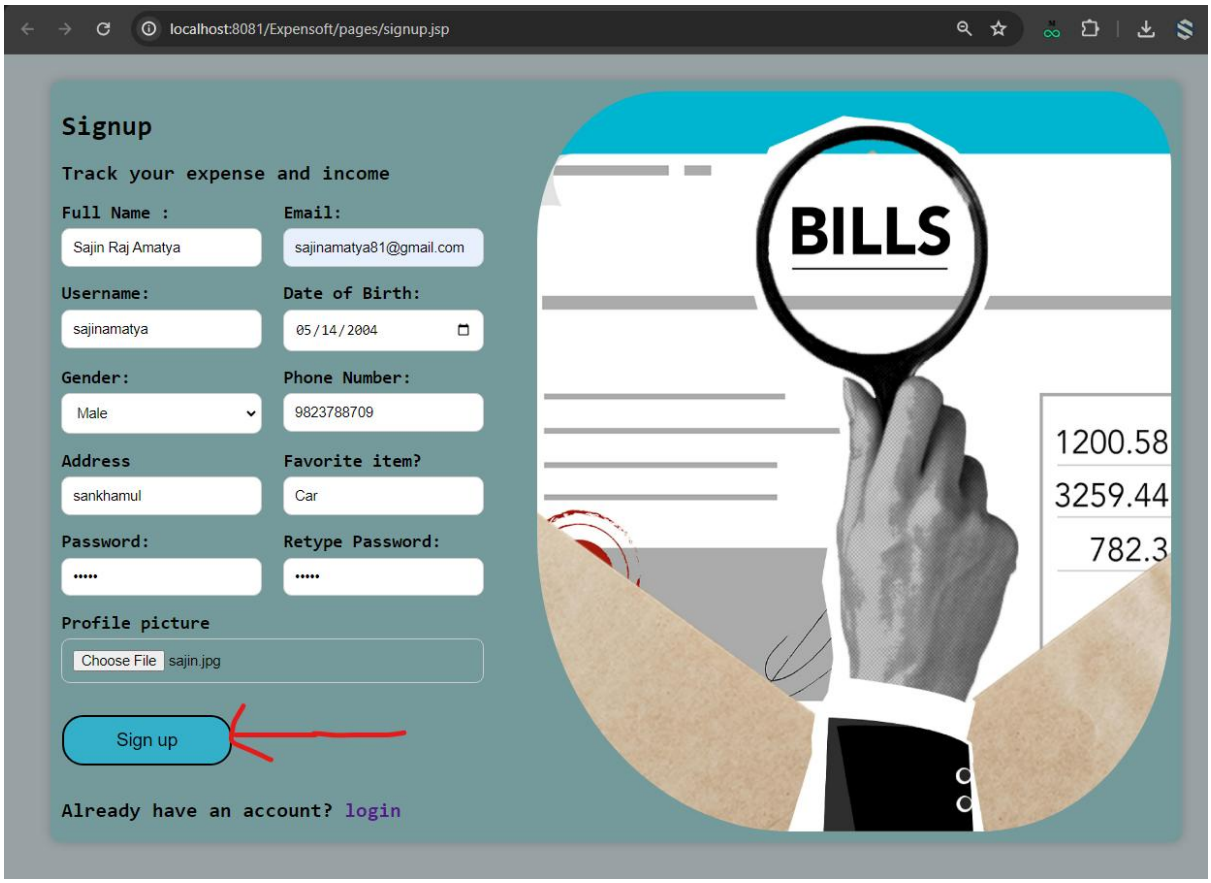


Figure 50 Test 1 Sign Up page

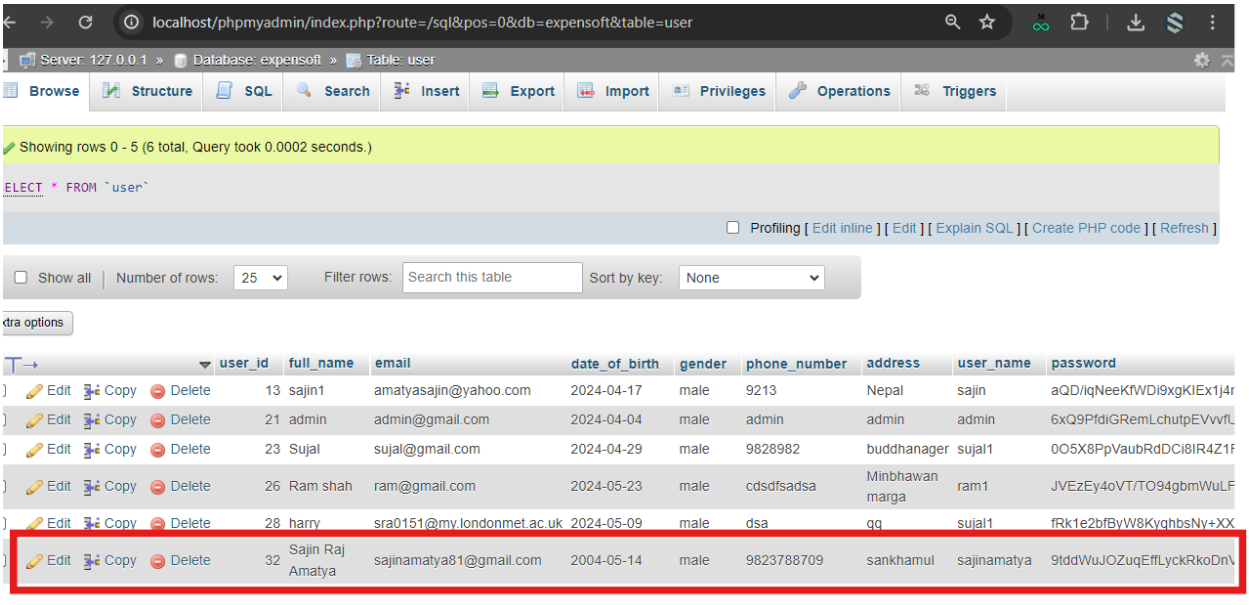


Figure 51 Test 1 insertion in database

Test 2: Testing login Authentication of user

Test No. 2	2
Objective	To check whether the login Authentication works or not
Action	➢ The following data was entered in the text field of the login page. UserName : sajinamatya Password : sajin ➢ Login button was clicked to submit it
Expected Result	The user will be authenticated to the system and redirect to the user home page
Actual Result	The user was authenticated to the system and redirect to the user home page
Conclusion	The Test is successful

Table 21 Test 2 Testing login Authentication of user

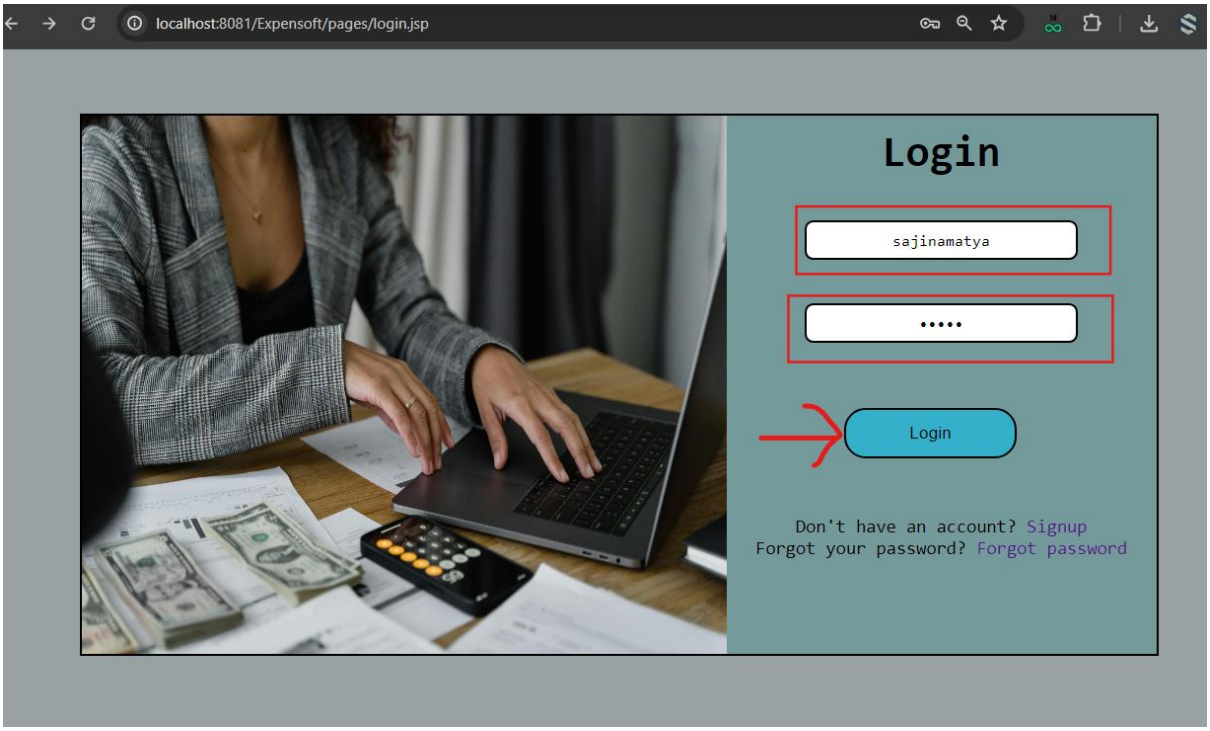


Figure 52 Test 2 login page

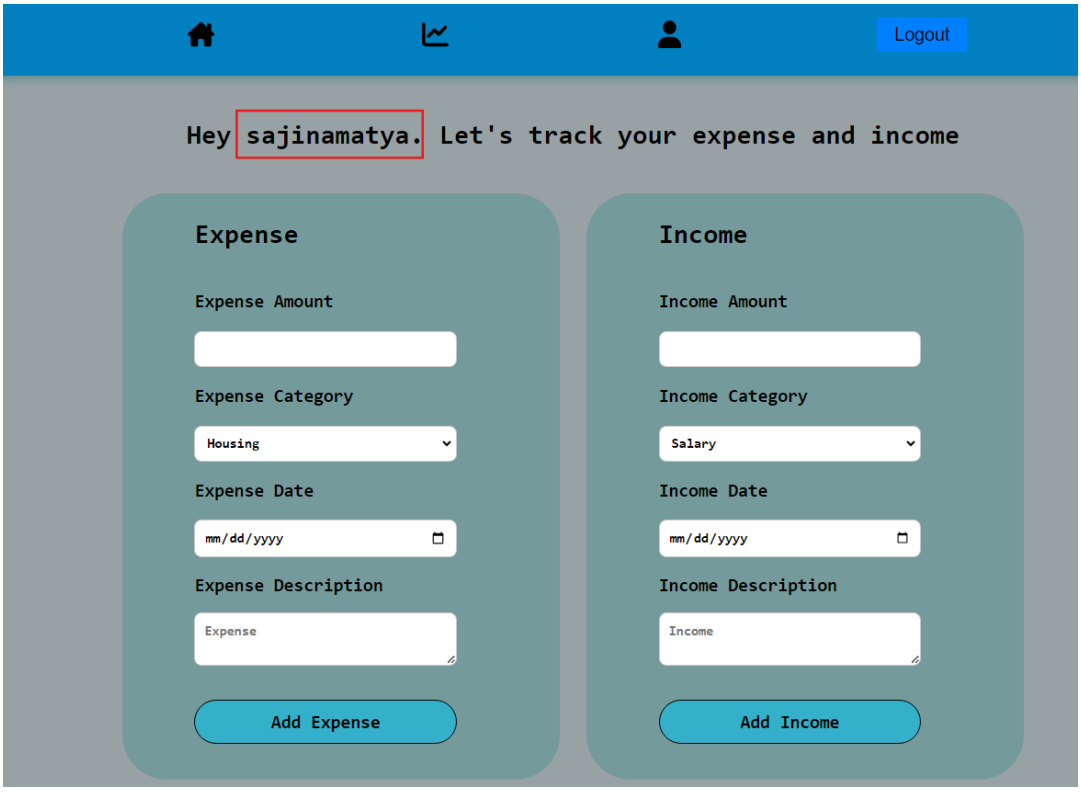


Figure 53 Test 2 redirection to home page after success login.

Test 3 : Testing the implementation of cookies and Session tracking

Test No. 3	3
Objective	To check whether session and cookies are created or not when the user login is logged in
Action	➢ The following data was entered in the text field of the login page. UserName : sajinamatya Password : sajin ➢ Login button was clicked to submit it ➢ Inspection of the web page was opened, and network section was clicked to view session and cookie
Expected Result	The user session and cookies will be created
Actual Result	The user session and cookies were created
Conclusion	The Test is successful

Table 22 Test 3 : Testing the implementation of cookies and Session tracking

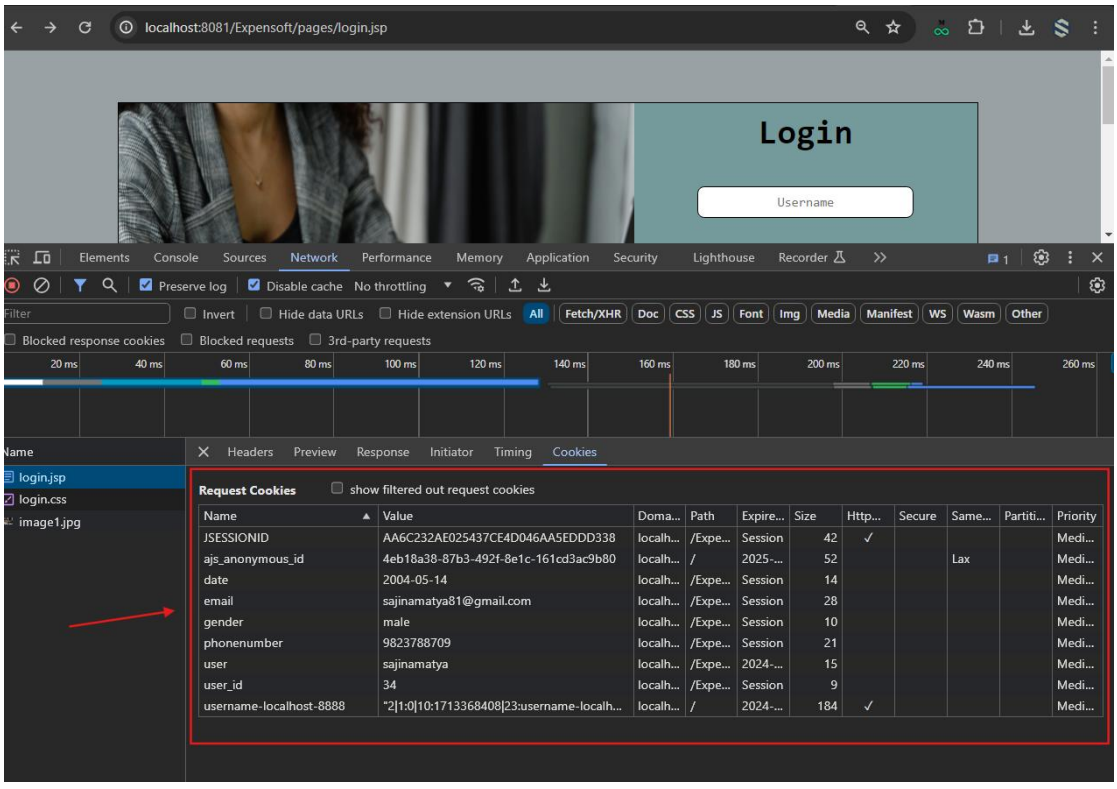


Figure 54 Test 3 User session and cookie.

Test 4: Testing the insertion of user expense and income detail in database.

Test No. 4	4
Objective	To check whether the user expense and income details are added to the database or not
Action	➢ The following data was entered in the text field of the user home page. Expense Amount : 2000 ExpenseCategory : Transportation Expense Date : 05/15/2024 Expense Description : Taxi ➢ Add expense button was clicked Income Amount : 30000 IncomeCategory : salary Income Date : 05/02/2024 Income Description : Taxi ➢ Add Income button was clicked
Expected Result	The expense and income detail will be added to the database
Actual Result	The expense and income detail were added to the database
Conclusion	The Test is successful

Table 23 Test 4: Testing the insertion of user expense and income detail in database.

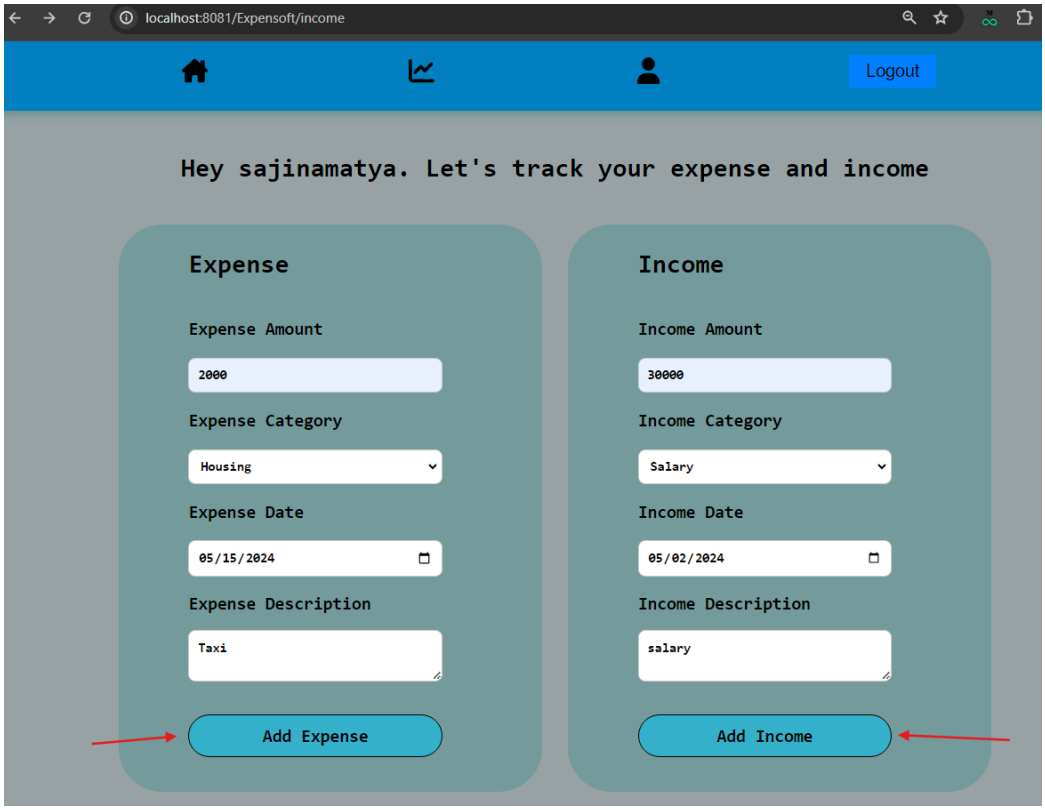


Figure 55 Test 4 user expense and income page.

Extra options

	user_id	expense_id	expense_amount	expense_date	expense_category	expense_description
Edit Copy Delete	13	5	30000.000	2024-02-13	Entertainment	movies
Edit Copy Delete	13	6	30000.000	2024-04-17	Food	kfc
Edit Copy Delete	13	7	2000.000	2024-04-25	Entertainment	Movie with sister
Edit Copy Delete	13	9	34000.000	2024-04-26	Healthcare	MRI
Edit Copy Delete	13	10	321.000	2024-03-20	Transportation	taxi
Edit Copy Delete	13	11	123.000	2024-02-14	Food	juice
Edit Copy Delete	13	12	123.000	2024-02-14	Food	juice
Edit Copy Delete	13	13	123.000	2024-02-14	Food	juice
Edit Copy Delete	13	14	123.000	2024-02-14	Food	juice
Edit Copy Delete	13	19	30000.000	2024-05-16	Housing	mom gave me money
Edit Copy Delete	13	20	34000.000	2024-05-22	Entertainment	clothuing
Edit Copy Delete	34	21	2000.000	2024-05-15	Transportation	Taxi

Figure 56 Test 4 Insertion of expense detail in database.

	user_id	income_id	income_amount	income_date	income_category	income_description
Edit Copy Delete	13	1	123.000	2024-05-07	Transportation	dsa
Edit Copy Delete	13	2	3000.000	2024-05-14	Food	Data analysis
Edit Copy Delete	13	3	1200.000	2024-05-16	Transportation	share bonus
Edit Copy Delete	34	4	30000.000	2024-05-02	Salary	Office allowance

Figure 57 Test 4 Insertion of income detail in database.

Test 5: Testing if the user expense and income detail are displayed in dashboard

Test No. 5	5
Objective	To check whether the user expense and income detail are displayed in the dashboard or not
Action	➢ User dashboard page was opened
Expected Result	The expense and income detail will be displayed .
Actual Result	The expense and income detail were displayed.
Conclusion	The Test is successful

Table 24 Test 5: Testing if the user expense and income detail are displayed in dashboard

Figure 58 Test 5 User dashboard

Test 6: Testing Update operation of user profile

Test No. 6	6
Objective	To check whether the user update operation works or not
Action	➢ User Profile page was opened. In the text field was entered Full Name: Sajin Amatya Address: Buddhanagar ➢ Save change button was clicked.
Expected Result	The expense and income detail will be added to the database
Actual Result	The expense and income detail were added to the database
Conclusion	The Test is successful

Table 25 Test 6: Testing Update operation of user profile

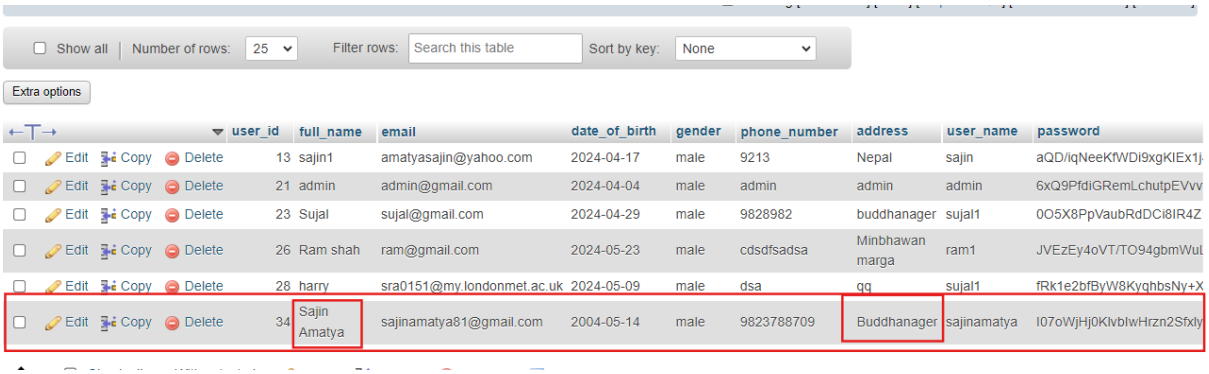


Figure 59 Test 6 UPDATE Operation in profile page.

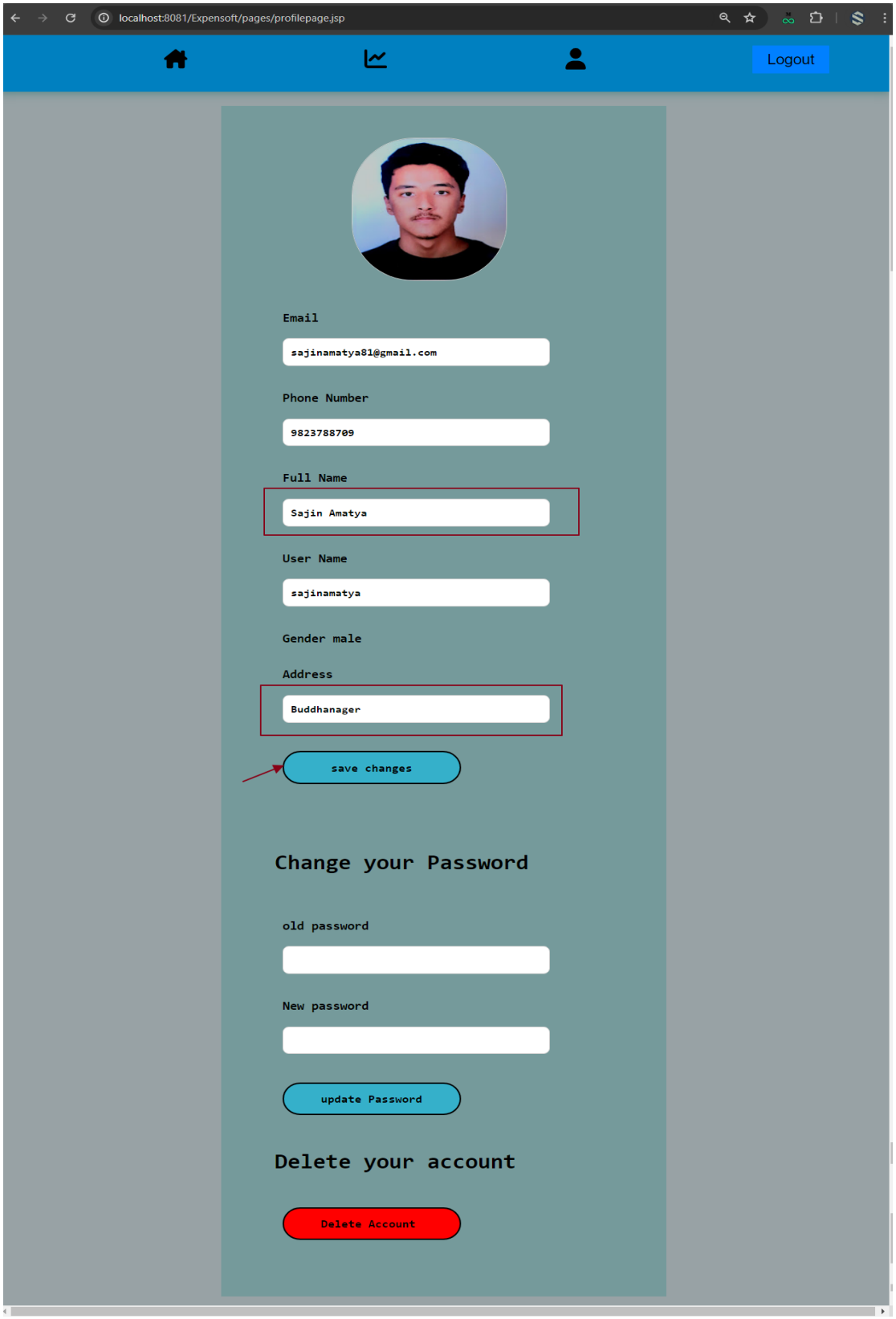


Figure 60 Test 6 User profile page

Test 7: Testing the deleting account functionality.

Test No. 7	7
Objective	To check whether the user delete operation works or not
Action	➢ User Profile page was opened. ➢ Delete account button was clicked. ➢ Confirmation message was shown and clicked ok
Expected Result	The user account will be deleted from the database
Actual Result	The user account was deleted from the database
Conclusion	The Test is successful

Table 26 Test 7: Testing the deleting account functionality.

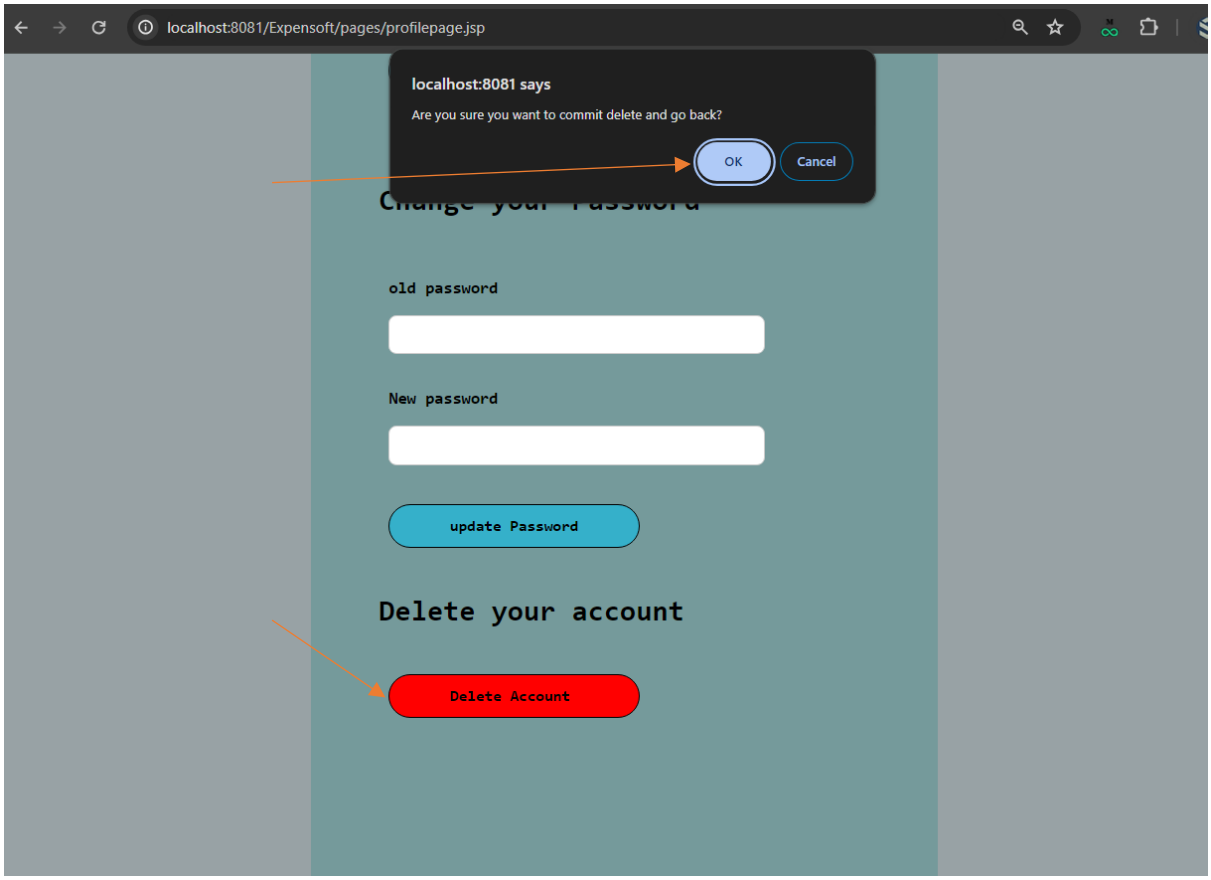


Figure 61 Test 7 deleting user account.

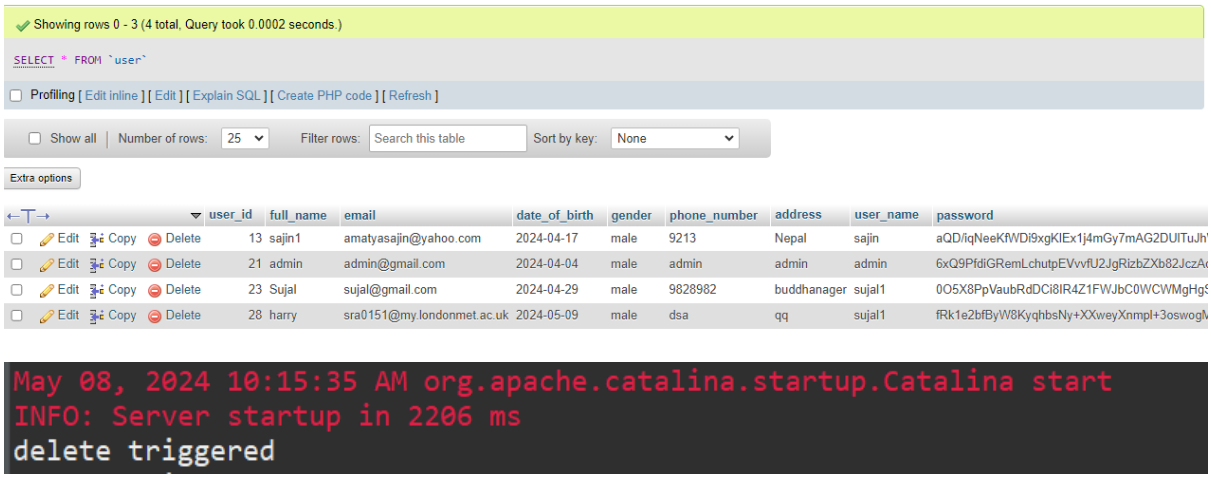


Figure 62 Test 7 details of the user deleted from the database.

For Admin

Test 8: Testing the Admin login Authentication.

Test No. 8	8
Objective	To check whether the admin authentication works or not
Action	➢ Login page was opened. The following data was entered in textfield Username = “ admin” Password = “ admin” ➢ Login button was clicked
Expected Result	The admin will be authenticated to admin page.
Actual Result	The admin was authenticated to admin page.
Conclusion	The Test is successful

Table 27 Test 8: Testing the Admin login Authentication.

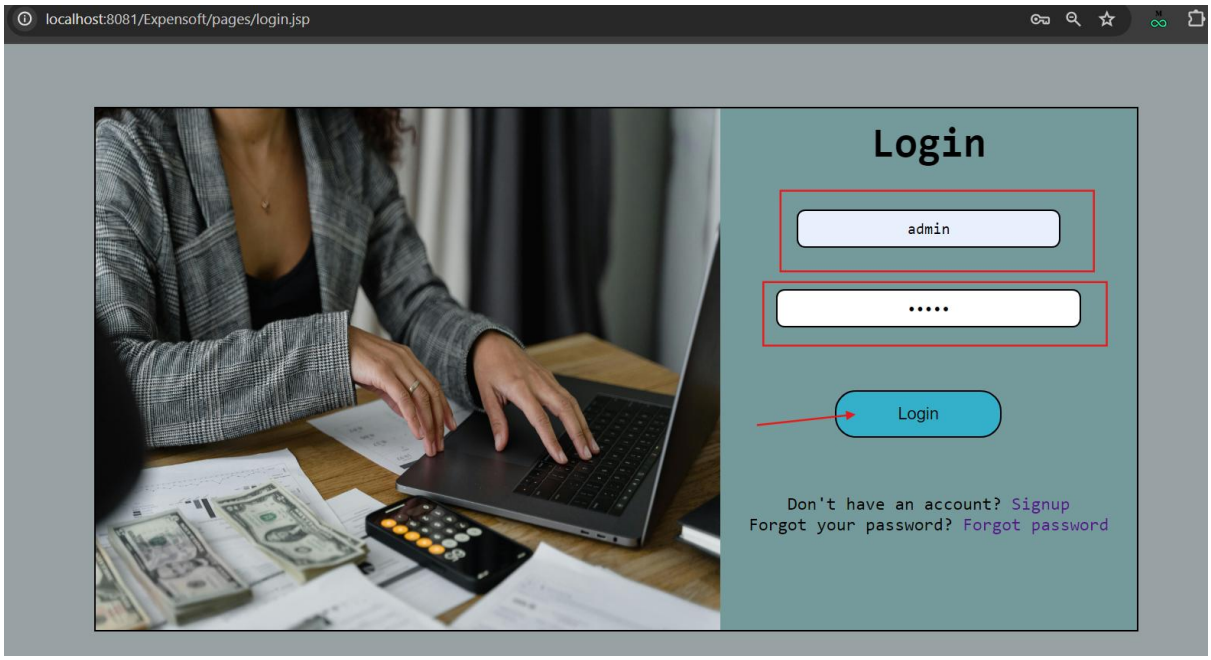


Figure 63 Test 8 Admin login

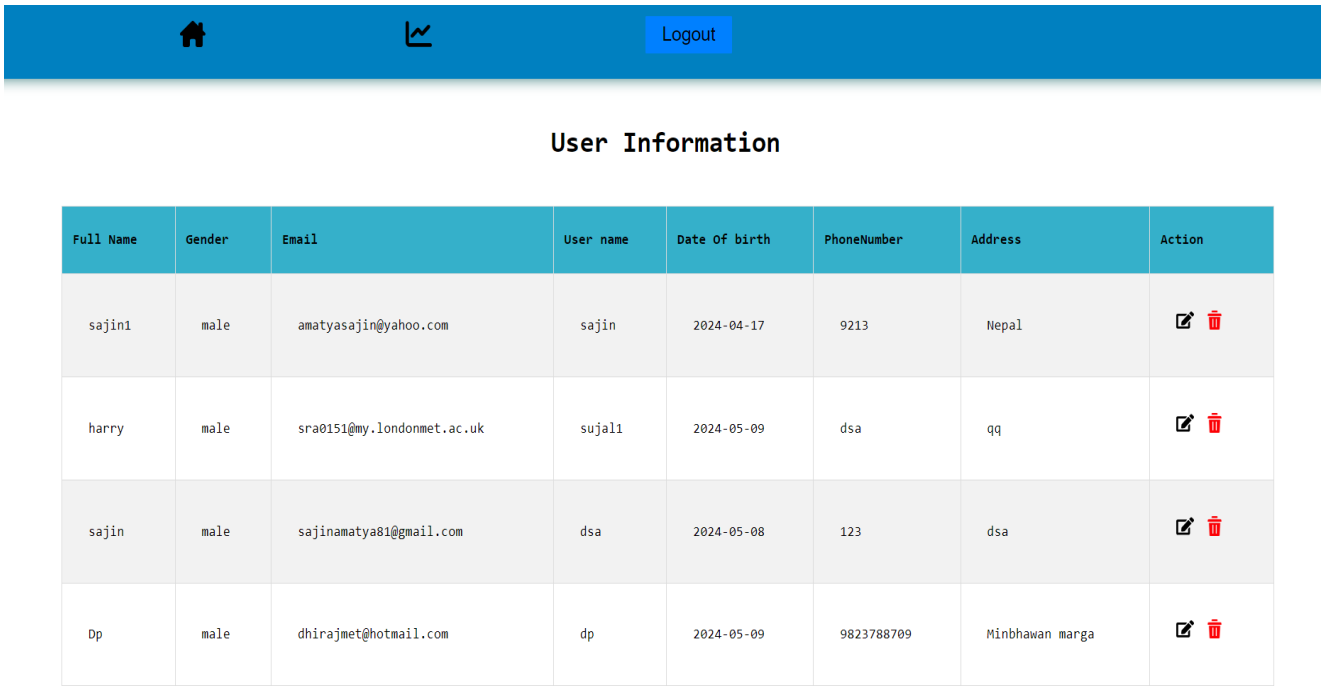


Figure 64 Test Redirection to admin page.

Test 9: Testing the Admin user account deletion functionality.

Test No. 9	9
Objective	To check whether the admin user deletion functionality works or not
Action	➢ Login page was opened. The following data was entered in textfield Username = " admin" Password = " admin" ➢ Login button was clicked ➢ Admin home page was redirected ➢ Delete button was clicked to remove account of username "sanjay"
Expected Result	The user account will be deleted by admin.
Actual Result	The user account was deleted by admin.
Conclusion	The Test is successful

Table 28 Test 9: Testing the Admin user account deletion functionality.

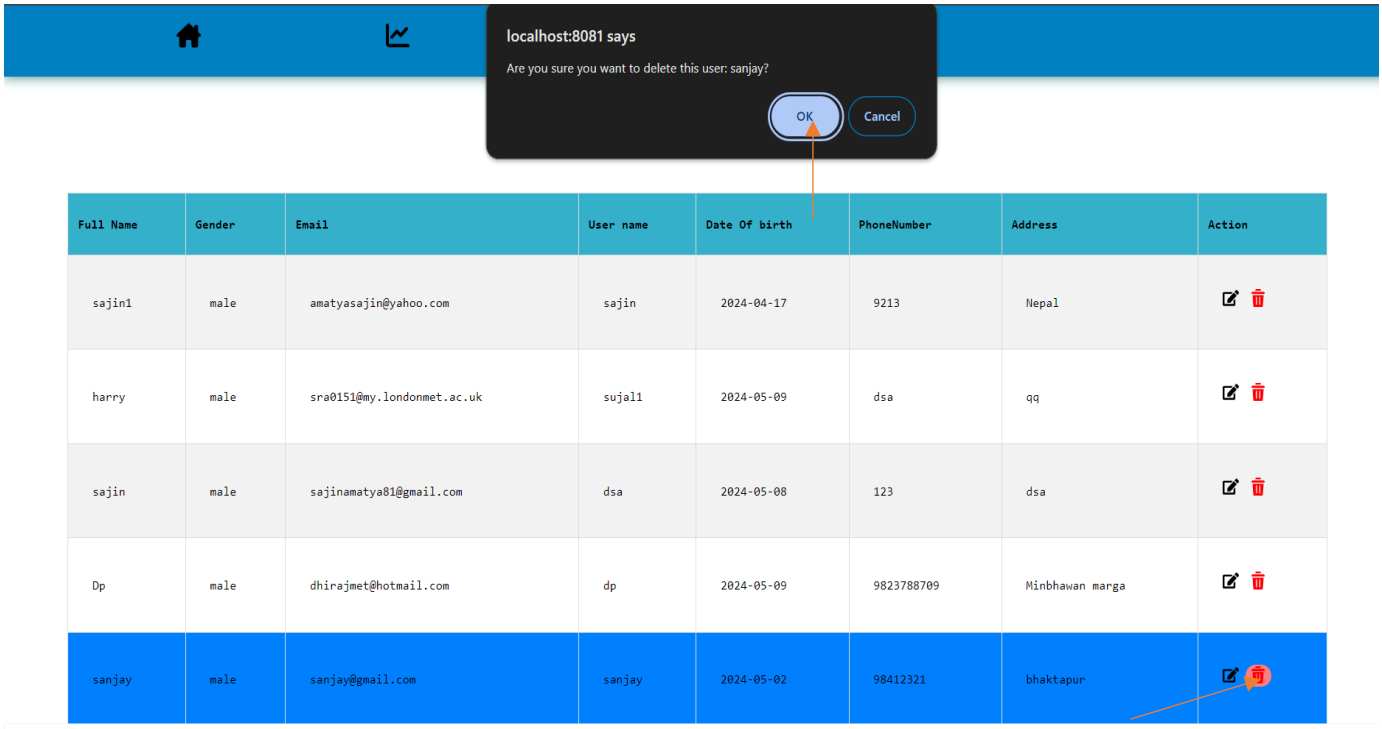


Figure 65 Test 9 Deleting user account by admin.

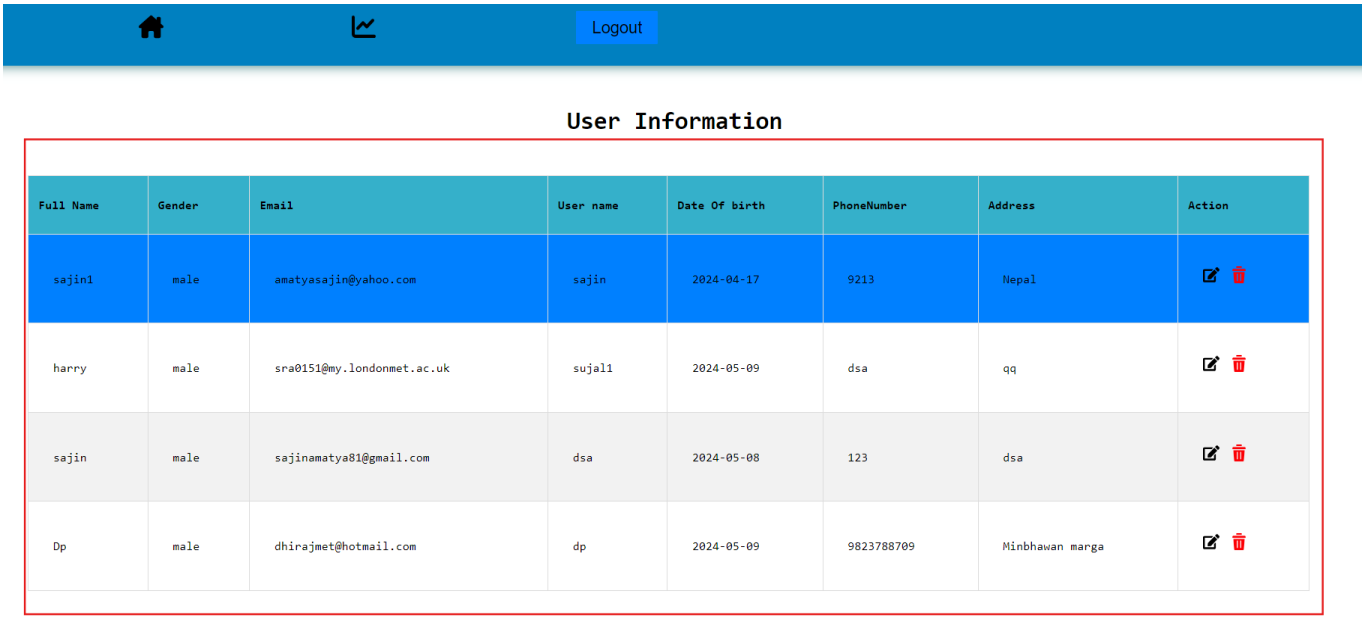


Figure 66 Test 9 Table after deletion operation.

Test 10: Testing admin dashboard summary statistic functionality

Test No. 10	10
Objective	To check whether the admin user deletion functionality works or not
Action	➢ Login page was opened. The following data was entered in textfield Username = “ admin” Password = “ admin” ➢ Login button was clicked ➢ Admin home page was redirected ➢ Admin dashboard was page was opened
Expected Result	The user account will be deleted by admin.
Actual Result	The user account was deleted by admin.
Conclusion	The Test is successful

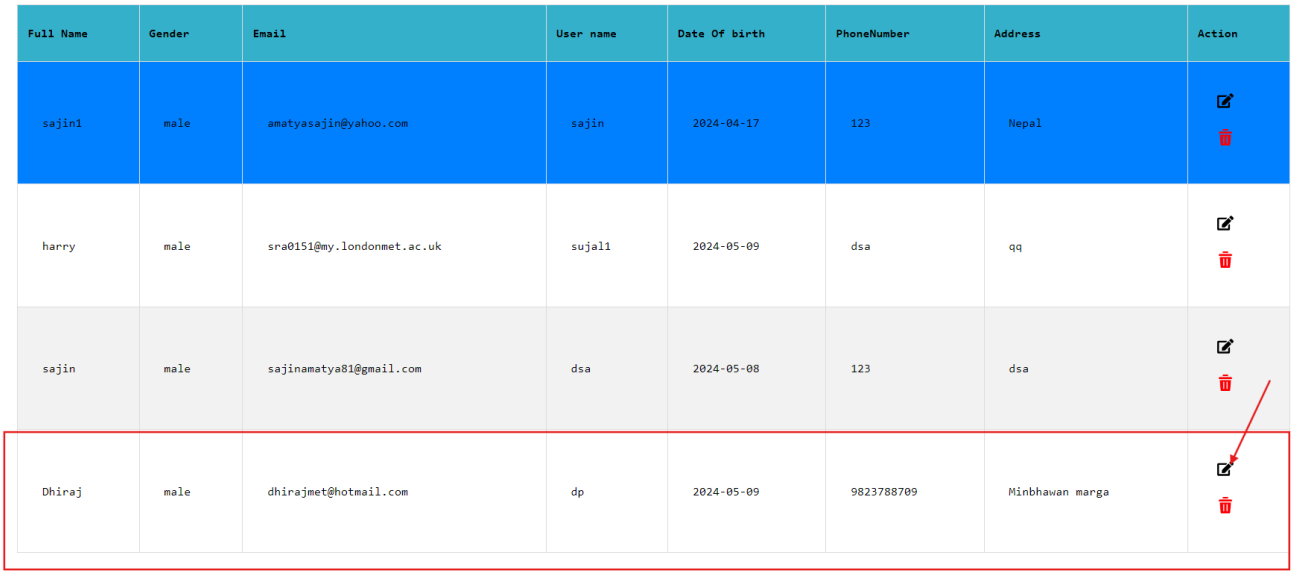
Table 29 Test 10: Testing admin dashboard summary statistic functionality

Figure 67 Test 10 Admin dashboard.

Test 11: Testing the update user functionality of admin.

Test No. 11	11
Objective	To check whether the admin update user functionality works or not
Action	➢ Login page was opened. The following data was entered in textfield Username = “ admin” Password = “ admin” ➢ Login button was clicked ➢ Admin home page was redirected ➢ From user details table modify button was clicked for
Expected Result	The user account will be deleted by admin.
Actual Result	The user account was deleted by admin.
Conclusion	The Test is successful

Table 30 Test 11: Testing the update user functionality of admin.



Full Name	Gender	Email	User name	Date Of birth	PhoneNumber	Address	Action
sajin1	male	amatyasajin@yahoo.com	sajin	2024-04-17	123	Nepal	
harry	male	sra0151@my.londonmet.ac.uk	sujal1	2024-05-09	dsa	qq	
sajin	male	sajinamatya81@gmail.com	dsa	2024-05-08	123	dsa	
Dhiraj	male	dhirajmet@hotmail.com	dp	2024-05-09	9823788709	Minbhawan marga	

Figure 68 Test 11 Modifying user data

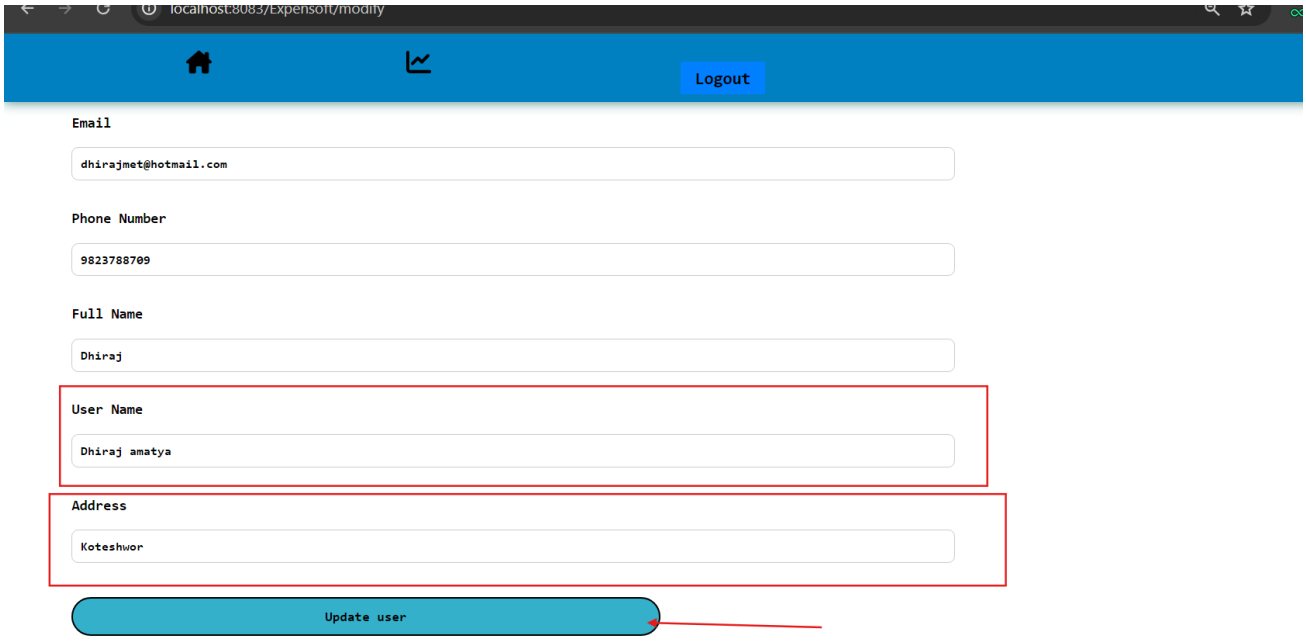


Figure 69 Test 11 Updating user details.

Full Name	Gender	Email	User name	Date Of birth	PhoneNumber	Address	Action
sajin1	male	amatyasajin@yahoo.com	sajin	2024-04-17	123	Nepal	
harry	male	sra0151@my.londonmet.ac.uk	sujal1	2024-05-09	dsa	qq	
sajin	male	sajinamatya81@gmail.com	dsa	2024-05-08	123	dsa	
Dhiraj	male	dhirajmet@hotmail.com	Dhiraj amatya	2024-05-09	9823788709	Koteshwor	

Figure 70 Test 11 user detail after updating by admin

7. Tools and Libraries Used

7.1 Software

7.1.1 Eclipse IDE

Eclipse IDE is an integrated Development environment which is used to develop java application with the help of installed java module, plugins, and other tools which can be integrated as per our needs. (tutorialspoint, 2024)

Using Eclipse IDE our java-based web project is developed and managed. Development process both frontend which consist of html, CSS, JSP and JavaScript and backend which consist of servlet, filter, database connection, is carried out in this IDE.

Figure 71 Eclipse IDE (Foundation, n.d.)

- Easy to integrate the Apache server to run, test and deploy java-based web application.
- Support for JSP, Java Servlet and JSTL for creating java-based web app.
- Addition of JAR file as per out project requirements.
- Option to install different packages, server, plugins within its marketplace.

7.1.2 XAMPP

Xampp is the open-source cross platform web server package used for the website development where we can set up a web server and database with the help of different scripting languages. It consists of Apache web server, MYSQL database, PERL, Tomcat server etc. (Gyaan, 2023).

Figure 72 XAMPP (seeklogo, 2024)

- Easy to install and configure it into our system.
- It can be operated in different operating systems (Cross platfrom).
- Easy to setup and configure Apache server, Tomcat server and MYSQL database from phpMyAdmin which is need for our project to function properly.

7.1.3 balsamiq Wireframe

Balsamiq is the desktop application design tools which is used for creating wireframes for the user interface design of the application (balsamiq, 2024). It is available on both mac and windows to download.

Figure 73 Balsamiq (Balsamiq, 2024)

- It is simple to use in compared to other tools like Figma and adobe XD
- It consists of variety of elements prototypes like button, text field , etc. for web design

7.2 Languages

7.2.1 Java

Java is a cross platform, object oriented, fast, secure, and reliable programming, which is used for developing the web applications, games, mobile applications, big data application etc. (Amazon, 2024).

Figure 74 Java (logos-world, n.d.)

- Java is fast and efficient compared to other programing languages like python, Golang etc.
- It is platform independence as its program in converted into bytecode that can be complied on any platform with the help of Java virtual machine.
- Support of object-oriented analysis for code reusability and memory optimization.

7.2.2 SQL (Structured Query Language)

SQL is a query language which is used to retrieve, manage, manipulate, aggregate data from the relational databases like Postgres SQL, MySQL etc. (Kanade, 2022) Using SQL the data can be extracted from the database for further analysis , implementation of dynamic webapp and full stack application.

Figure 75 SQL (alphaservesp, 2023)

- It is the industry standard, official query language for interaction with the different database.
- Its syntax is easier to understand and implement.
- It is used to define the query for extracting summarized information.
- It is used to insert the new data to the database, updated and delete by writing simple queries.

7.2.3 Java Server programming (JSP)

Java server page is the java web technology for creating dynamic and engaging web pages where the java code can be embedded in the html page using JSP tag (Tomar, 2021). The JSP code written in HTML page is converted into the server when complied.

Figure 76 JSP (logo, 2024)

- With the help of JSP we can directly embed the java code into the HTML page to create a dynamic web page.
- It is efficient as it is compiled into servlet when the page is runed.
- Easy to integrate with servlet for handling dynamic content.

7.2.4 JSTL (JSP Standard Tag Library)

JSP Standard Tag Library controls the JSP page behavior with collection of custom tag for iteration, control statements, database connection, etc operation in the web pages (Pankaj, 2022). JSTL major Tags category are Core tag, formatting tag, SQL tag, XML tag, and function tags.

Figure 77 JSTL (Nakoma, 2024)

- Support for the iteration and loop to reduce the code size by Core tag.
- Support for the condition statement.
- Database connection can be done easily with the help of SQL tag of JSTL.
- It can Display the request value from the server to the html page

8. Development process

8.1 Creation of Dynamic Web Project

A dynamic web project is required to operate our development process. Creating the dynamic web project process is show below.

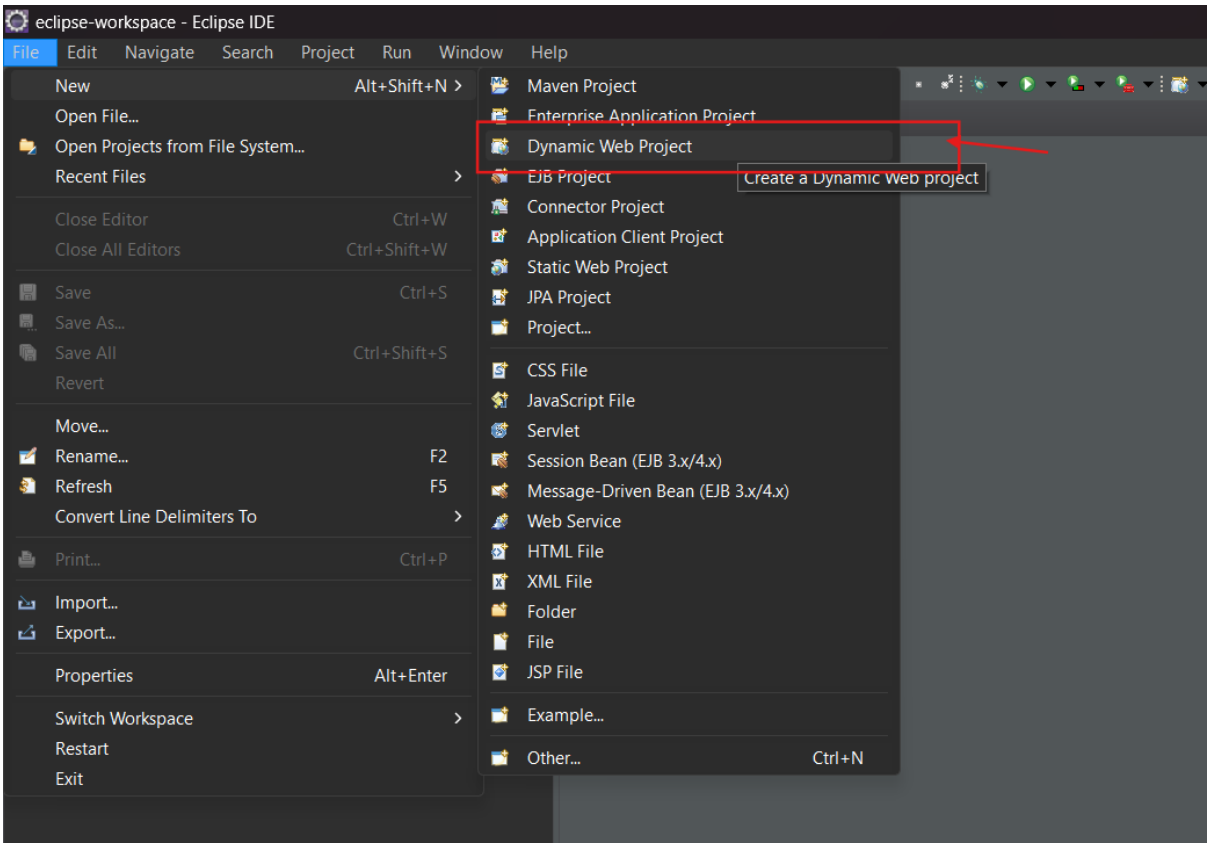


Figure 78 Creating dynamic web project.

Specify the name of our project for development, then selecting the target runtime and configuration to server Apache tomcat v8.5. Then click on ‘next’.

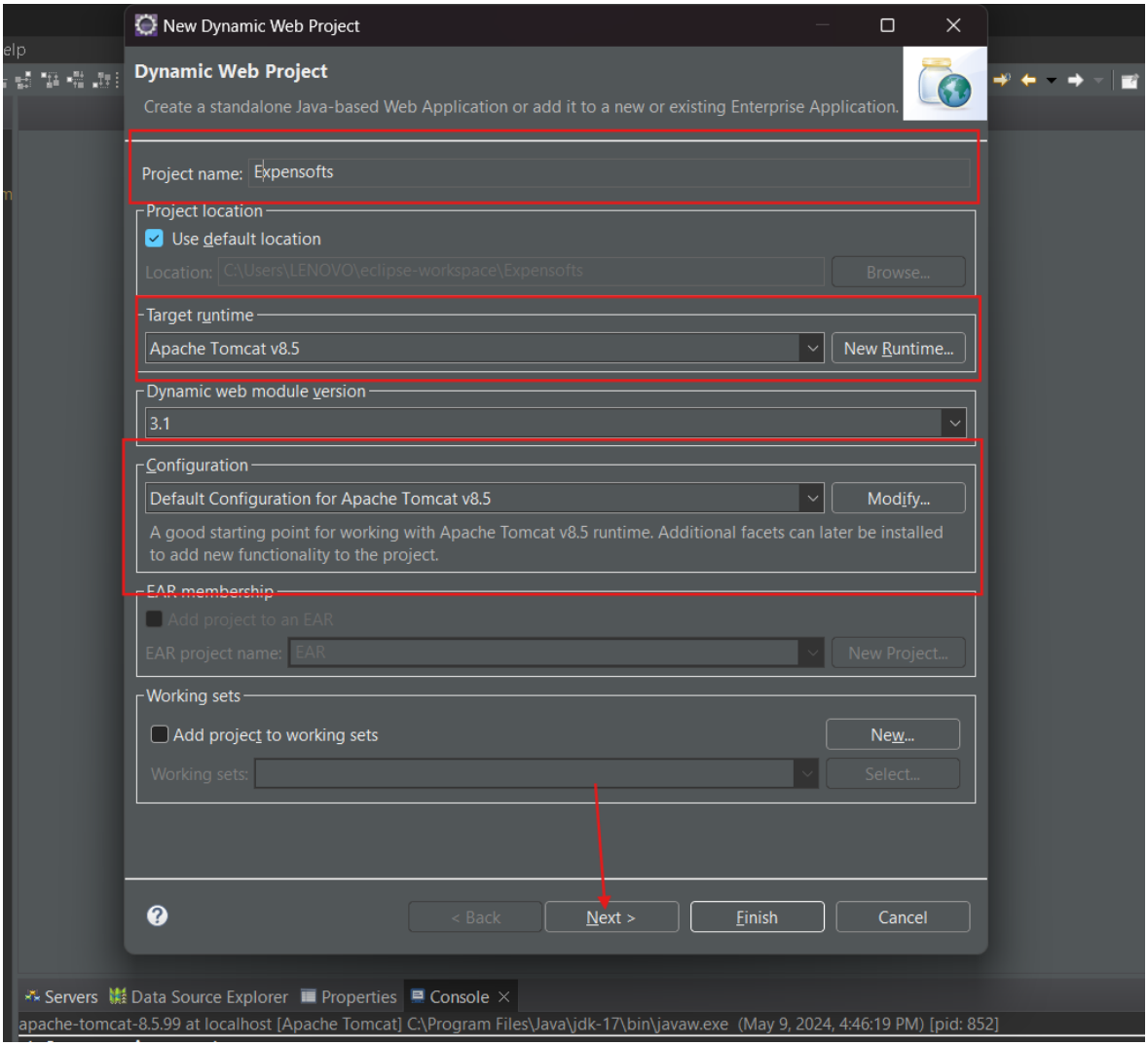


Figure 79 Setup for dynamic web project.

Select on generate web.xml deployment descriptor which would allow us to configure the map routing of URL to specific servlets.

Then click on finish, your project is created.

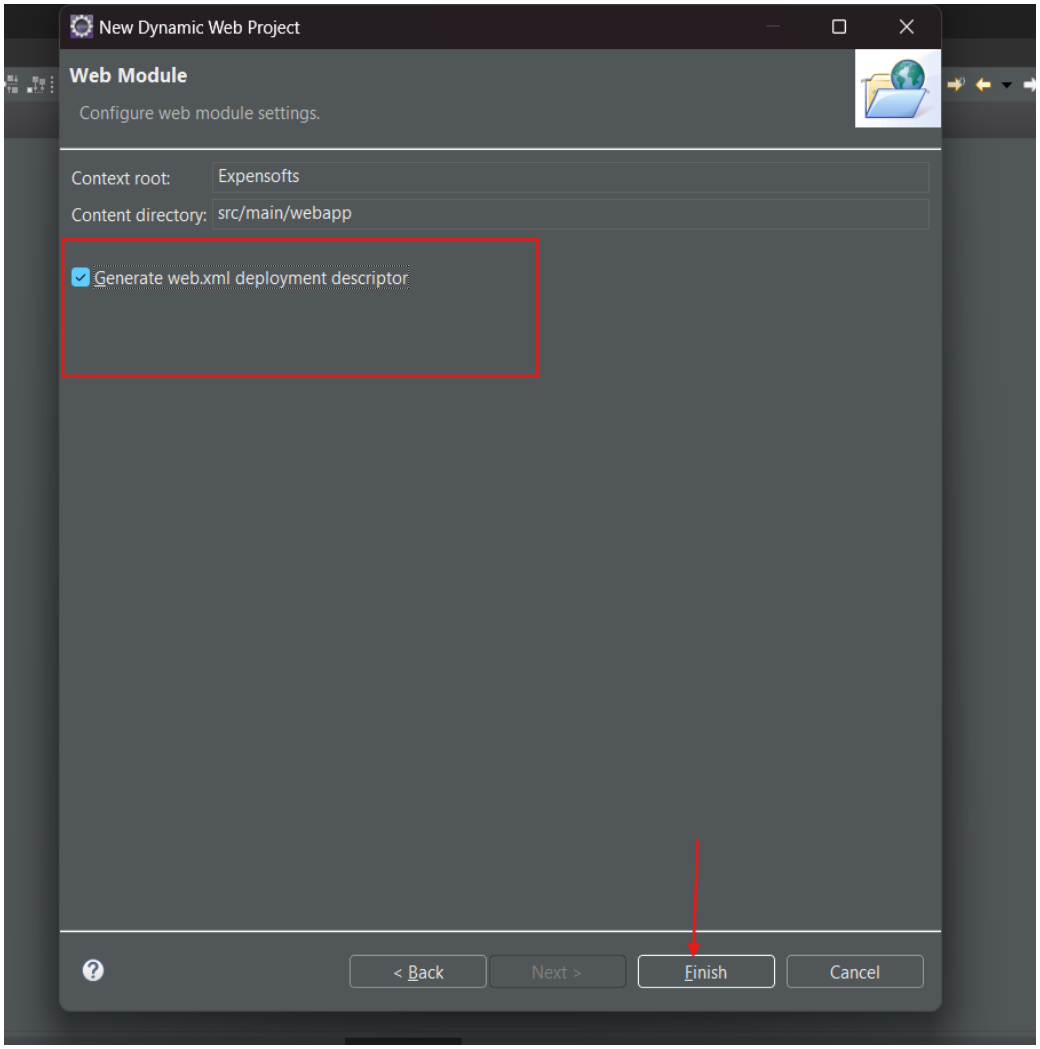


Figure 80 Confirming the addition of web.xml file in the project.

8.2 Creation of required packages or folder for the project

Here our project is created but we need a software development design to follow for clean and proper software development.

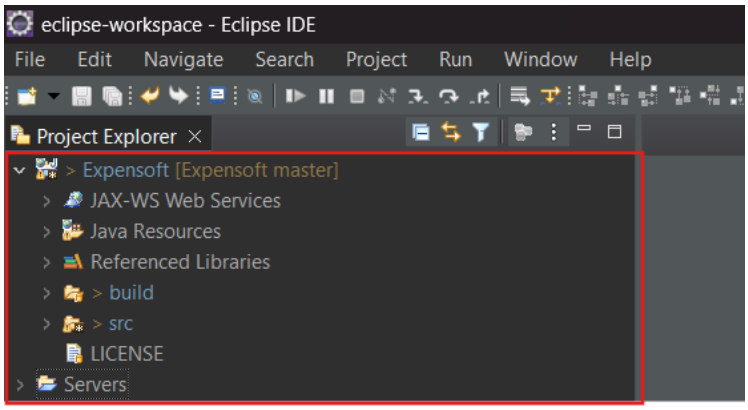


Figure 81 Project creation.

The MVC (Model view controller) design pattern is chosen for our project, where we have create a separated folder for View (webapp folder) , Model (model folder) and controller(controller folder).Also for other functionality like filter, validation and database folder is created inside the controller folder.

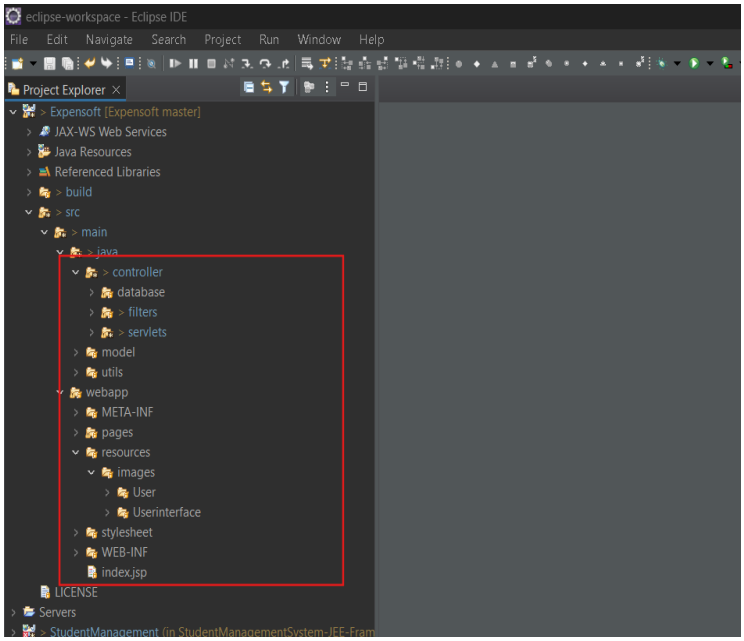


Figure 82 Overall project folder structure.

8.3 Adding Necessary libraries to the project.

Our project requires MySQL connector libraries for database connection and taglibs library for implementation of JSTL in frontend. To add those libraries firstly, install the MySQL connector and Taglibs jar file.

Right click on project folder > select build path > click on configure build path

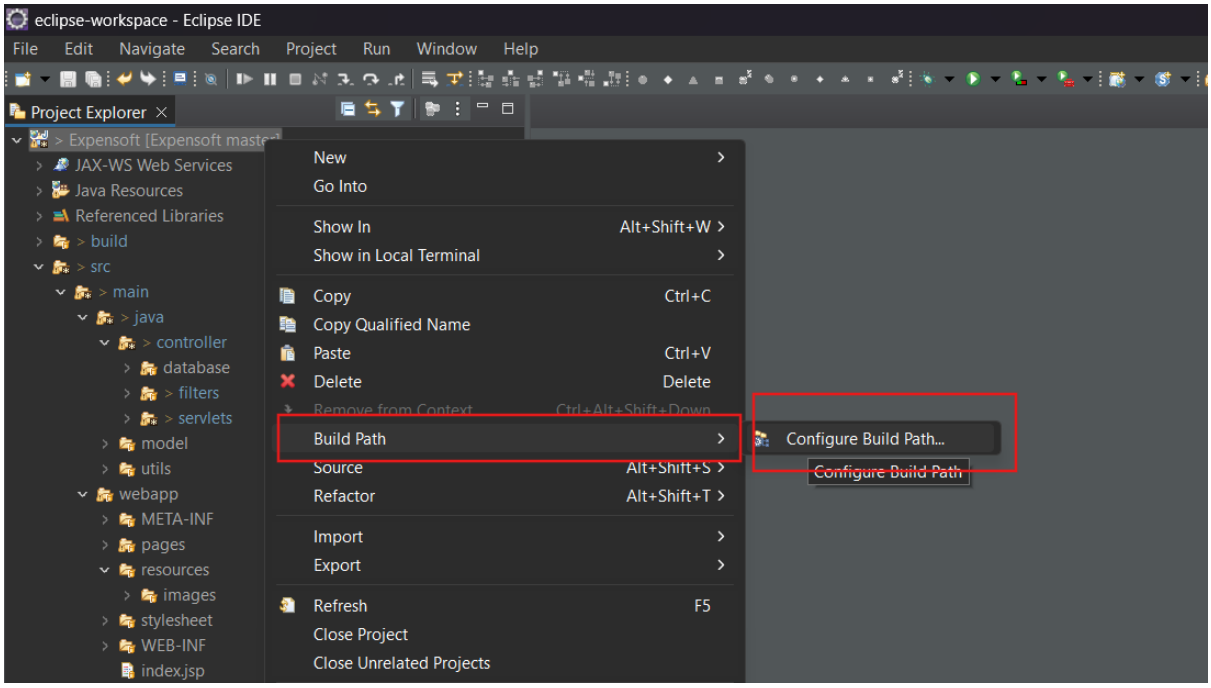


Figure 83 Process of adding libraries in the project

Click on add External Jars option > Select the required jar file for the project > then click on apply.

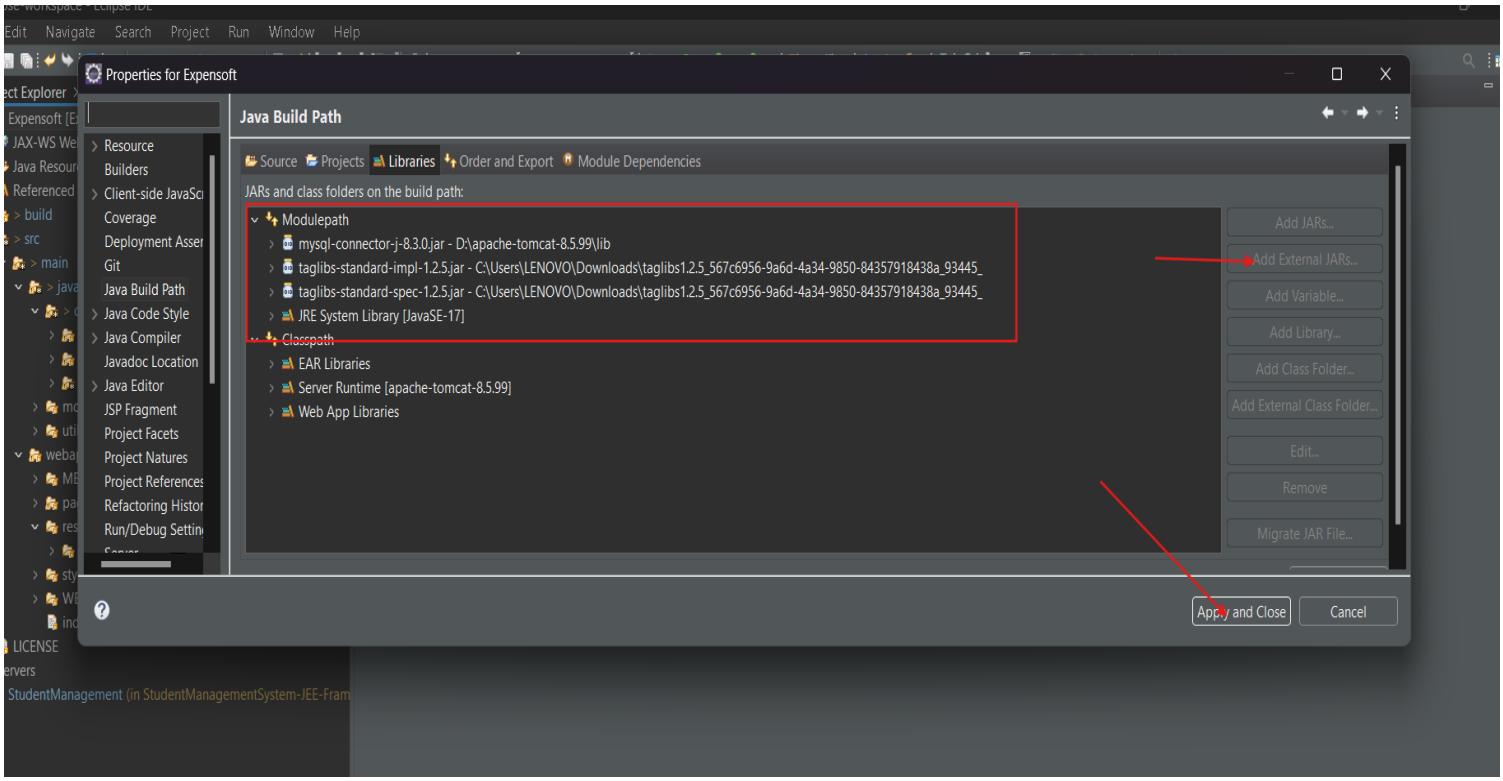


Figure 84 Adding required JAR file for the project.

8.4 Configuration of server and database

For creation of the database connection, server connection both tomcat and Apache XAMPP software is required.

Open Xampp > Start Apache, MySQl and Tomcat

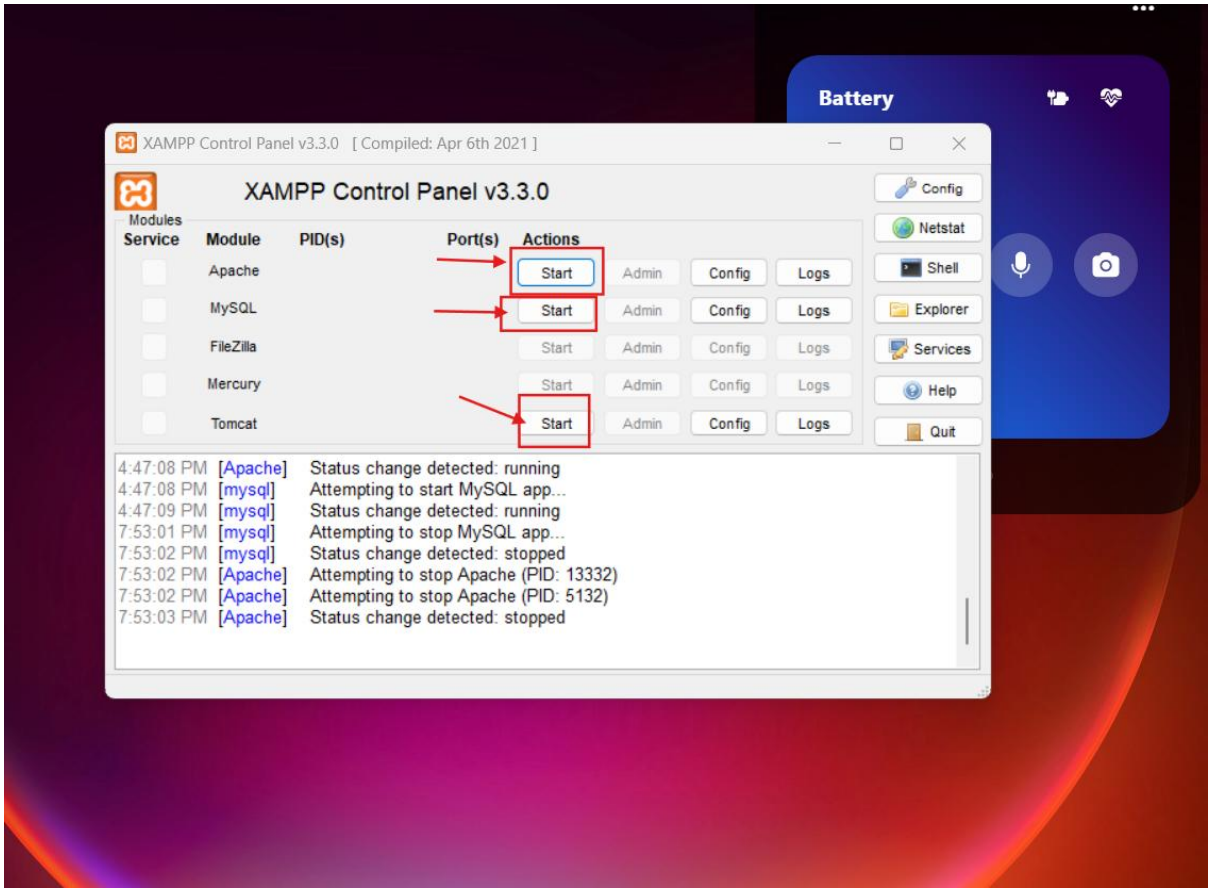


Figure 85 Starting XAMPP

When starting Apache server and Tomcat we have to change the port number of both of these servers.

Navigate to server > click on the port number > change tomcat to 8006 and HTTP/1.1 to 8081

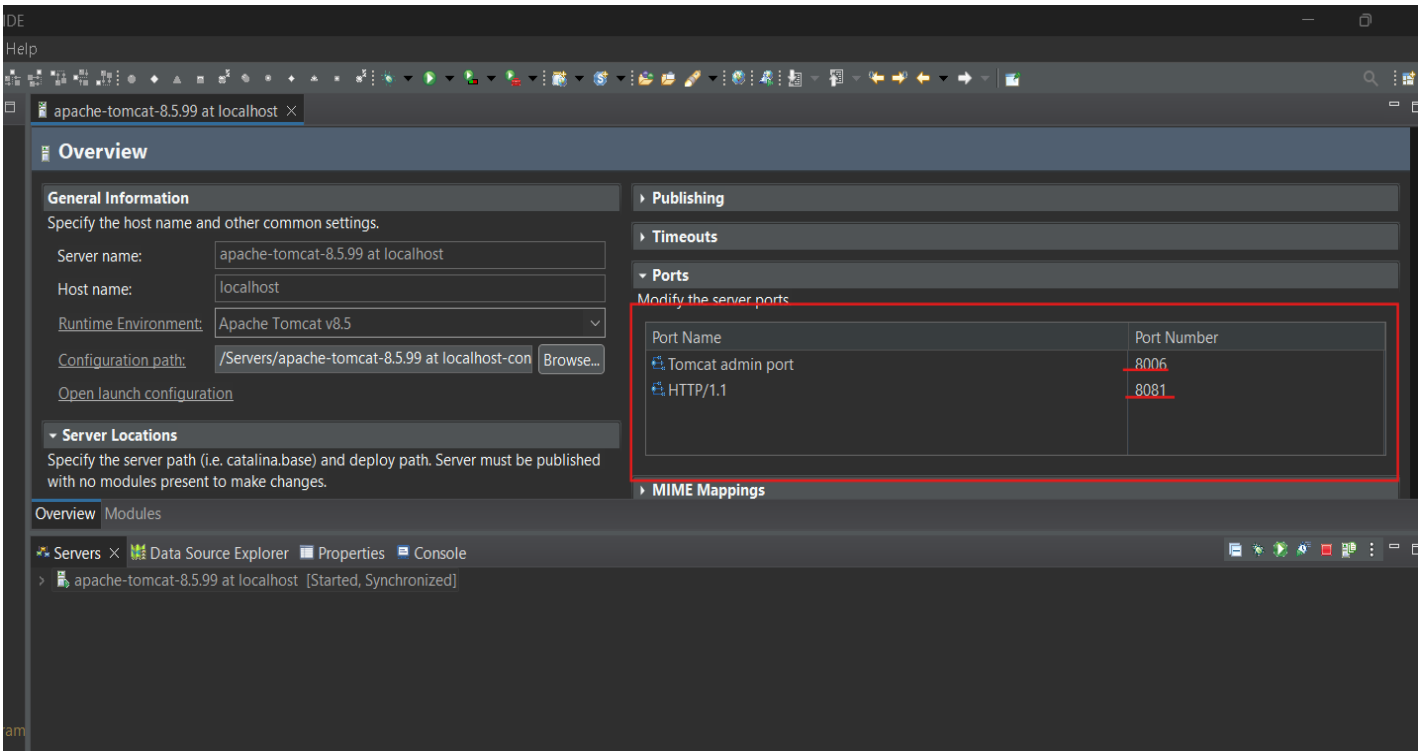


Figure 86 Changing the port number of the server.

8.5 Setting up web.xml file for authentication Filter and URL mapping.

When running the server, we want the server to hit certain servlet URL or jsp file. For authentication filter setup to implement authentication for accessing the certain resources we would add filter servlet name to the web.xml

To set up this, open web.xml > In welcome-file tag enter your Jsp file name. > for filter write the name of the servlet with authentication filter in filter-name tag

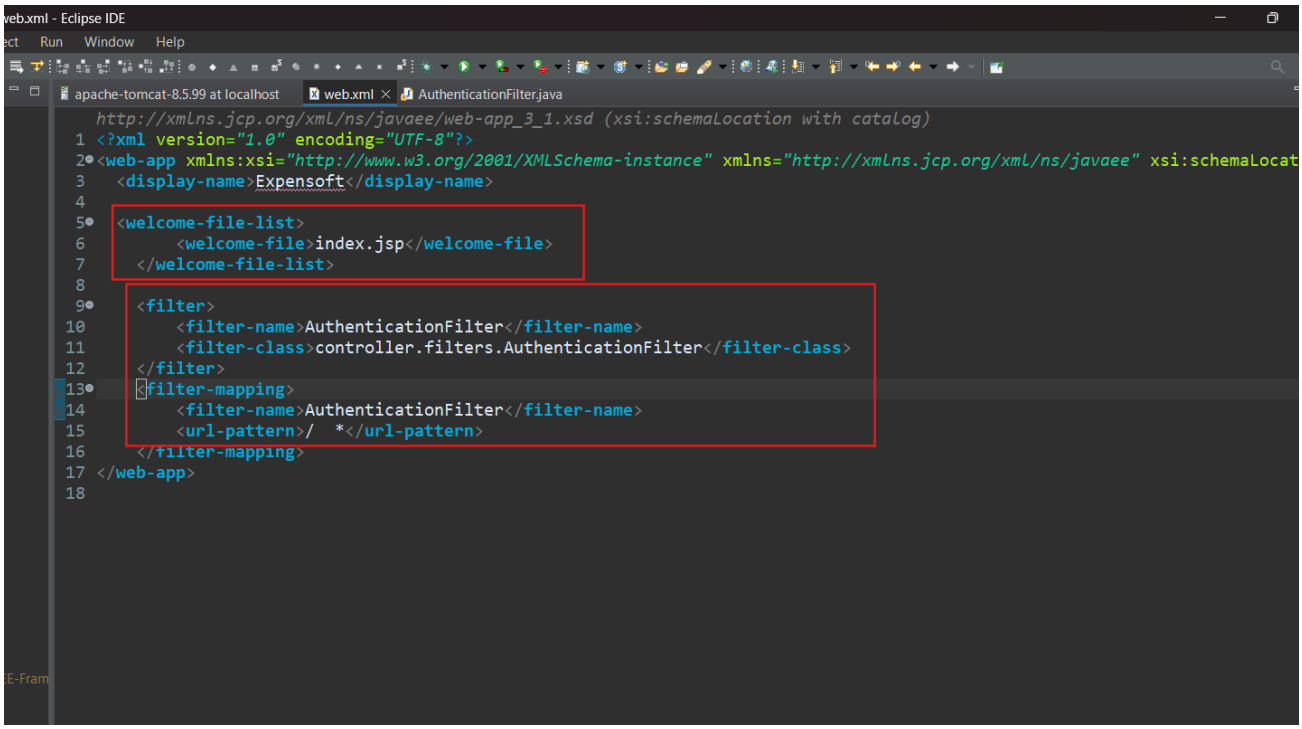


Figure 87 Configuration of web.xml file

9. Critical Analysis

9. 1 Development

9.1.1 Unable to perform user delete operation.

While performing delete operation from the admin user management table, error was occurred because parent table primary key attribute user_id was linked with foreign key of expense and income child table.

Figure 88 Development Error 1

To solve this problem, I have created two SQL query and executed them along with user delete query in database controller for deleting the data of the user from both expense and income table as those are linked with parent table ‘user’.

```
QUERY_DELETE_USER = " DELETE FROM user where user_name = ?";
QUERY_DELETE_USER_EXPENSES = "DELETE FROM expense WHERE user_id = (SELECT user_id FROM user WHERE user_name = ?)";
QUERY_DELETE_USER_INCOMES = "DELETE FROM income WHERE user_id = (SELECT user_id FROM user WHERE user_name = ?)";
```

```
public int deleteUserInfo(String username) {
    try (Connection con = getConnection()) {

        try (PreparedStatement st1 = con.prepareStatement(Stringutils.QUERY_DELETE_USER_EXPENSES)) {
            st1.setString(1, username);
            st1.executeUpdate();
        }


        try (PreparedStatement st2 = con.prepareStatement(Stringutils.QUERY_DELETE_USER_INCOMES)) {
            st2.setString(1, username);
            st2.executeUpdate();
        }


        try (PreparedStatement st3 = con.prepareStatement(Stringutils.QUERY_DELETE_USER)) {
            st3.setString(1, username);
            return st3.executeUpdate();
        }
```

Figure 89 Error 1 Solution code.

Reference : https://www.geeksforgeeks.org/mysql-deleting-rows-when-there-is-a-foreign-key/

https://www.codejava.net/coding/jsp-servlet-jdbc-mysql-create-read-update-delete-crudexample

9.1.2 Unable to start tomcat server from xampp

Tomcat error occurred while working with the user profile image storage solution.

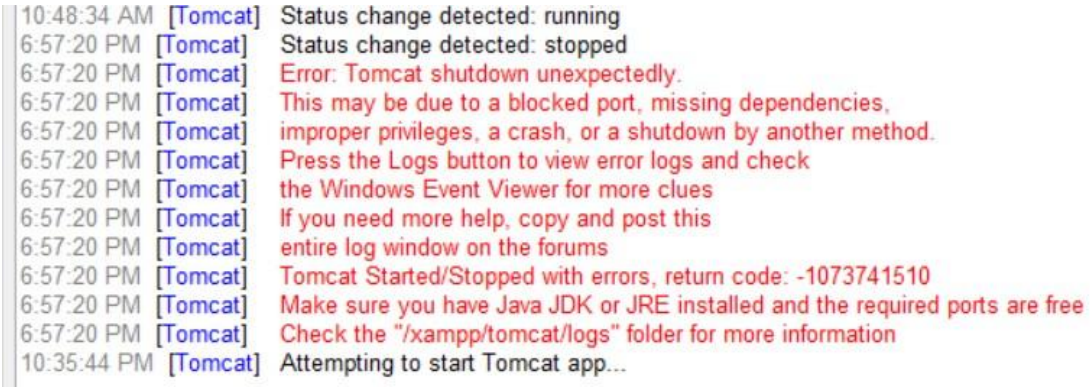


Figure 90 Tomcat error (Error 2)

From researching different forum the site like reddit, article sites I understood the issues which was creating this problem. I have changed the port of the tome cat server to 8806 with some changes made in server.xml file.

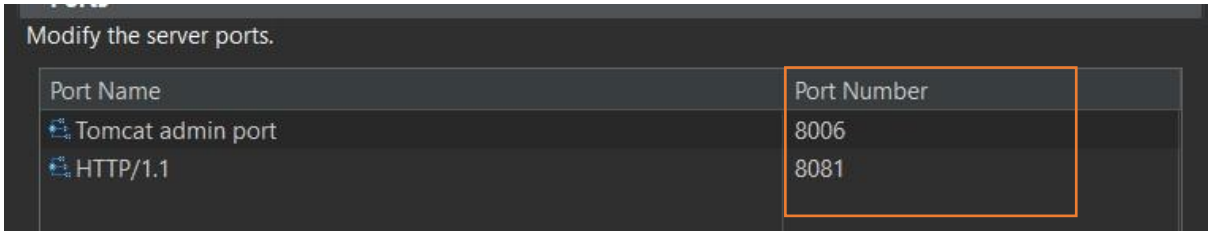


Figure 91 Solution for Error 3

Reference

https://muhammadtriwibowo.medium.com/xampp-tomcat-cant-start-with-exceptionaddress-already-in-use-net-bind-windows-397ccc748aef

https://www.codejava.net/servers/tomcat/how-to-change-port-numbers-for-tomcat-ineclipse

9.1.3 Unable to register the user.

All the fields with correct data was enter in the registration form but while try to submit it error occurred.

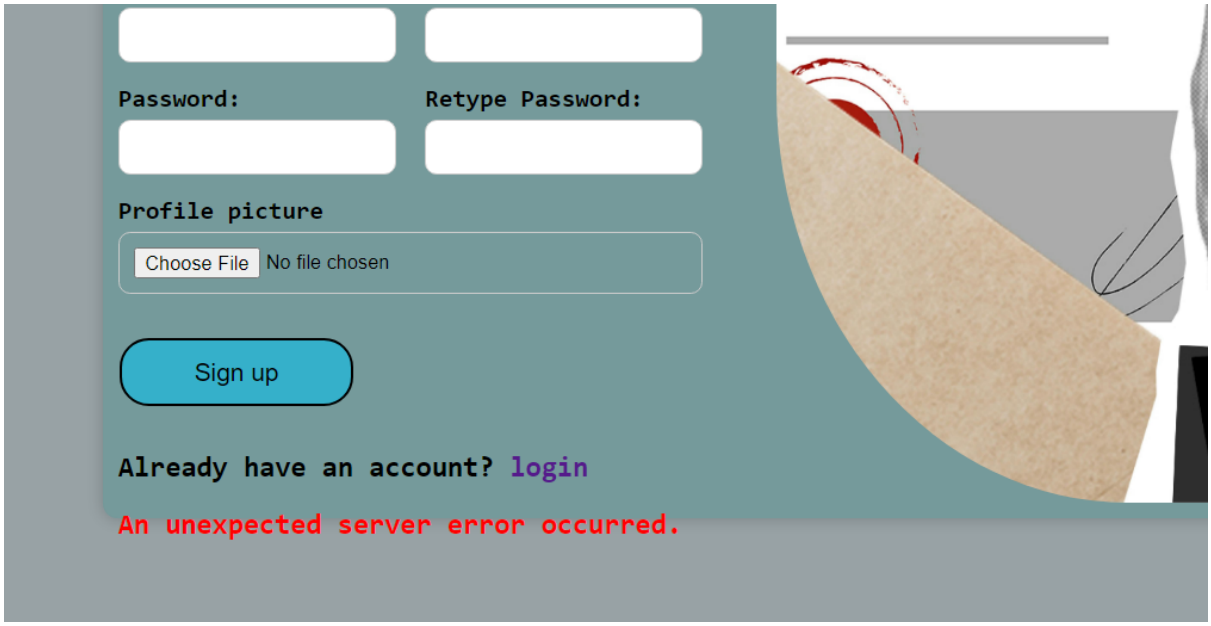


Figure 92 Development error 3

After checking overall working flow of the sign-up servlet, I found that there was error in SQL Query, the table of the database name was misspelled to ‘users’ but the actual table name was ‘user’.

```
public static final String QUERY_SIGNUP_USER = "INSERT INTO user ("
        + "full_name, email, date_of_birth, gender,phone_number, address, user_name, password,security_question,
        + "VALUES (?, ?, ?, ?, ?, ?, ?, ?,?,?)";
```

Figure 93 Development error 3 solution.

9.1.4 Unable to start MySQL server.

While running the server the MySQL was facing error shown below and was unable to start.

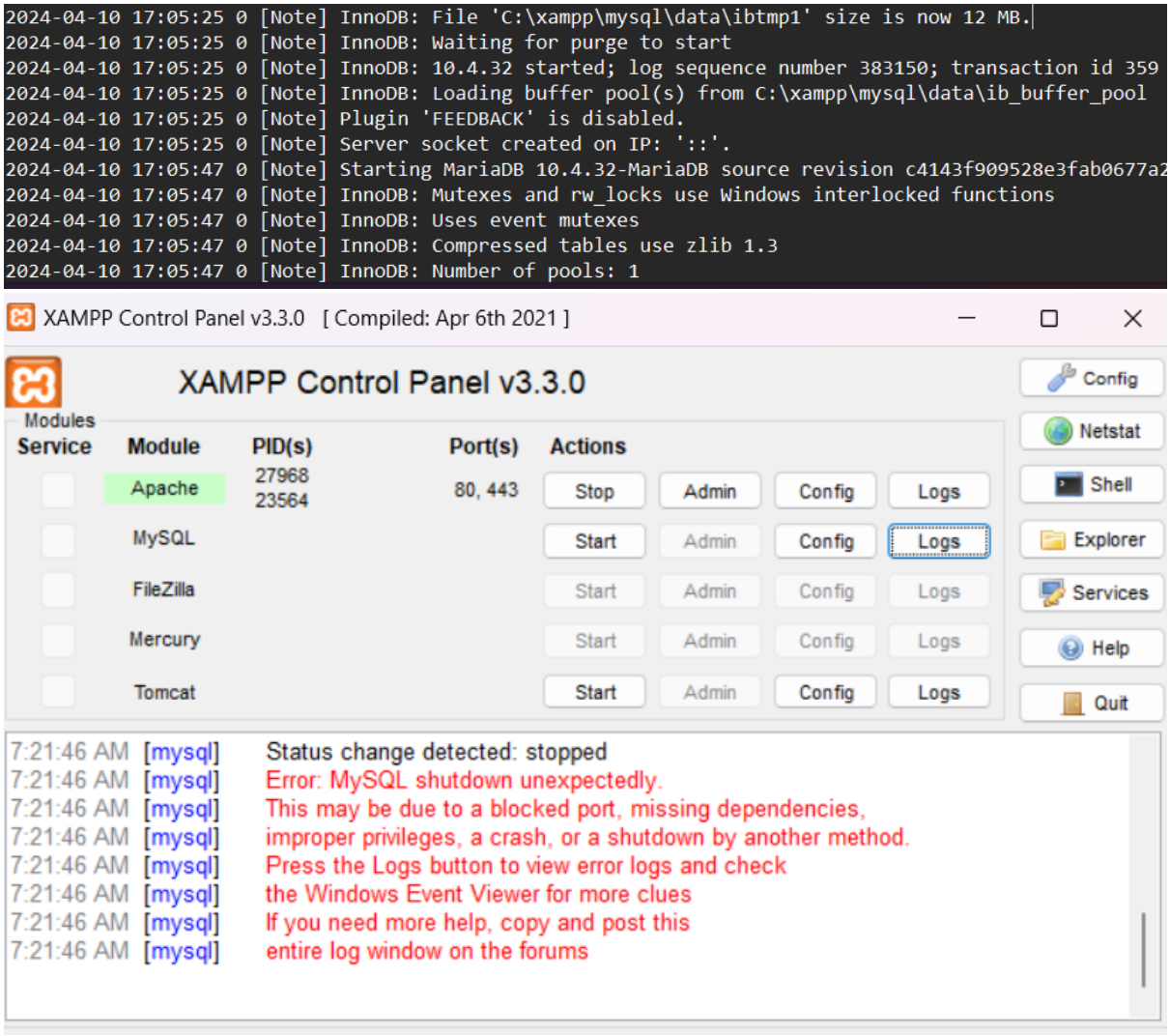


Figure 94 Development error 4

To solve this problem navigate to xampp folder > mysql folder > create a another copy of data folder > remove MySQL, performance schema and phpMyAdmin > and paste it again after copying the same file from the back up

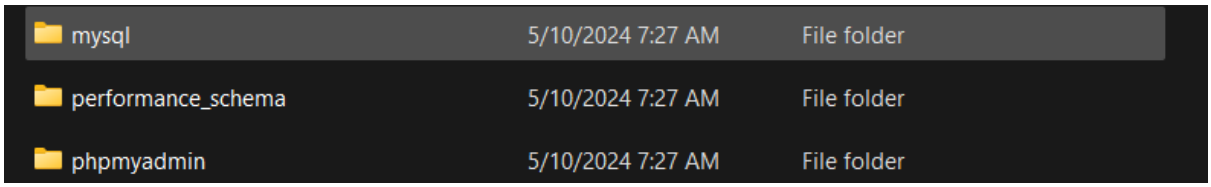


Figure 95 Development Error 4 solution

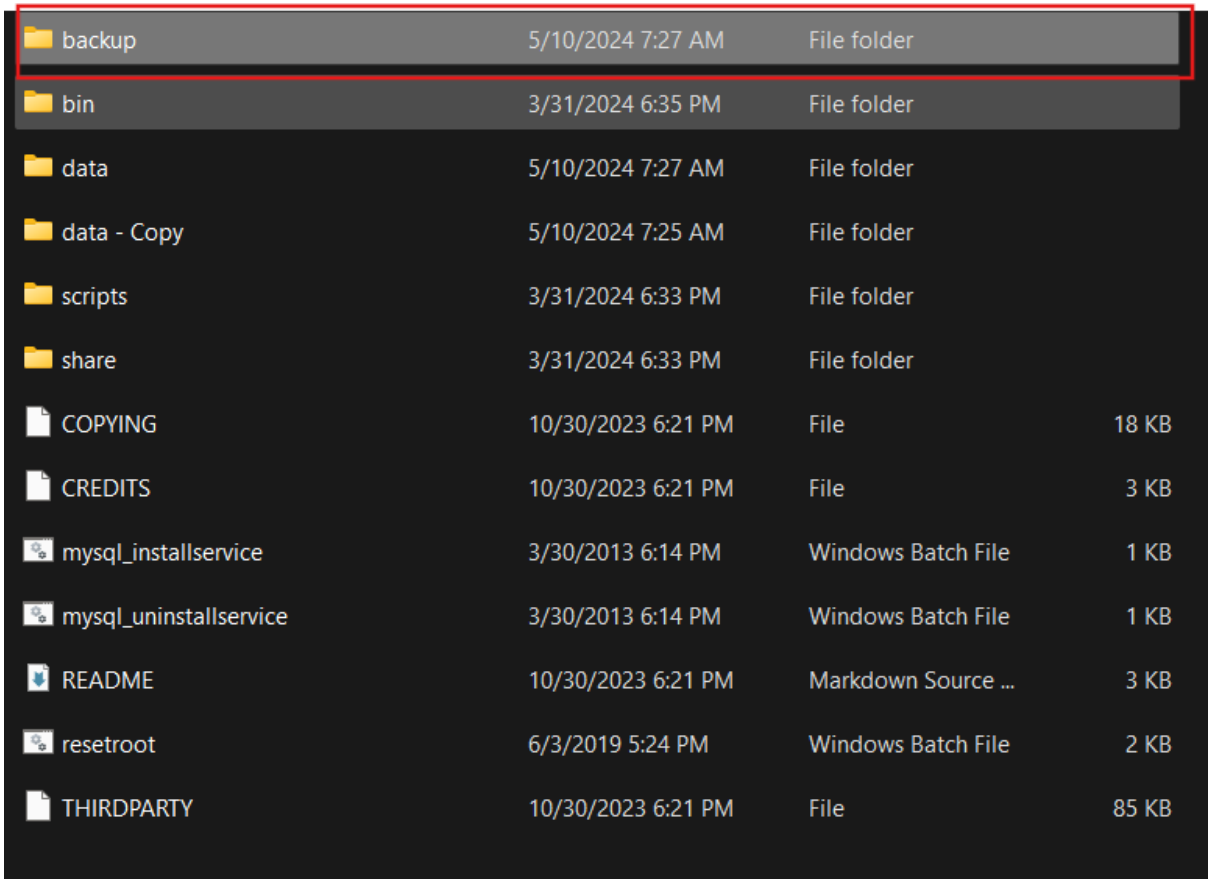


Figure 96 Development Error 4 solution 2.

Reference:
https://www.youtube.com/watch?v=84IOtc05TuA&ab_channel=BoostMyTool

9.2 Design

9.2.1 Unable to implement the image with different border radius on all corners.
Here, all four sides border is with same border radius but, different border radius size is required for all four corners in sign up page.

Figure 97 Design Error 1

Solution

To achieve the desire output in border-radius property 4 different values in percent is given where first value indicates top-left, second value indicates top-right, third value indicates bottom-right, third value indicates bottom-left.

```
signup-right{
   background-image:url(../resources/44.jpg);
   background-size:cover;
   width:600px;
   margin-left:5px;
   border-radius: 10% 30% 50% 70% ;
```

Figure 98 After solution implemented for Error 1 of design

Reference Source : https://developer.mozilla.org/en-US/docs/Web/CSS/border-radius

9.2.2 Unable to create space between two flex items of the containers.

Unable Implementing the space or gap between two items of the container of a flexbox while developing the fronted part of user home page.

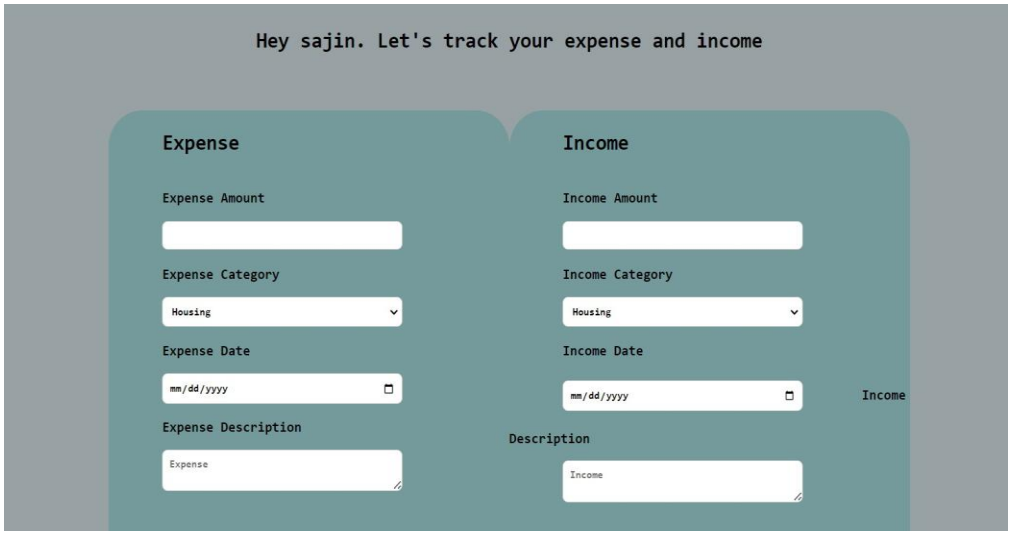


Figure 99 Design Error 2

Solution

To solve this problem, I have used justify-content properties of flex box and margin left properties in the left side flex item.

```
main{
    display:flex;
    width:1200px;
    height : 700px;
    margin: 50px auto;
    justify-content:space-between;
}
expense, .income {
    border-radius: 50px;
    width: 600px;
    height : 700px;
    background-color: #759A9B;
}
.income{
    margin-left:30px;
}
```

Figure 100 Design Error 2 solution code.

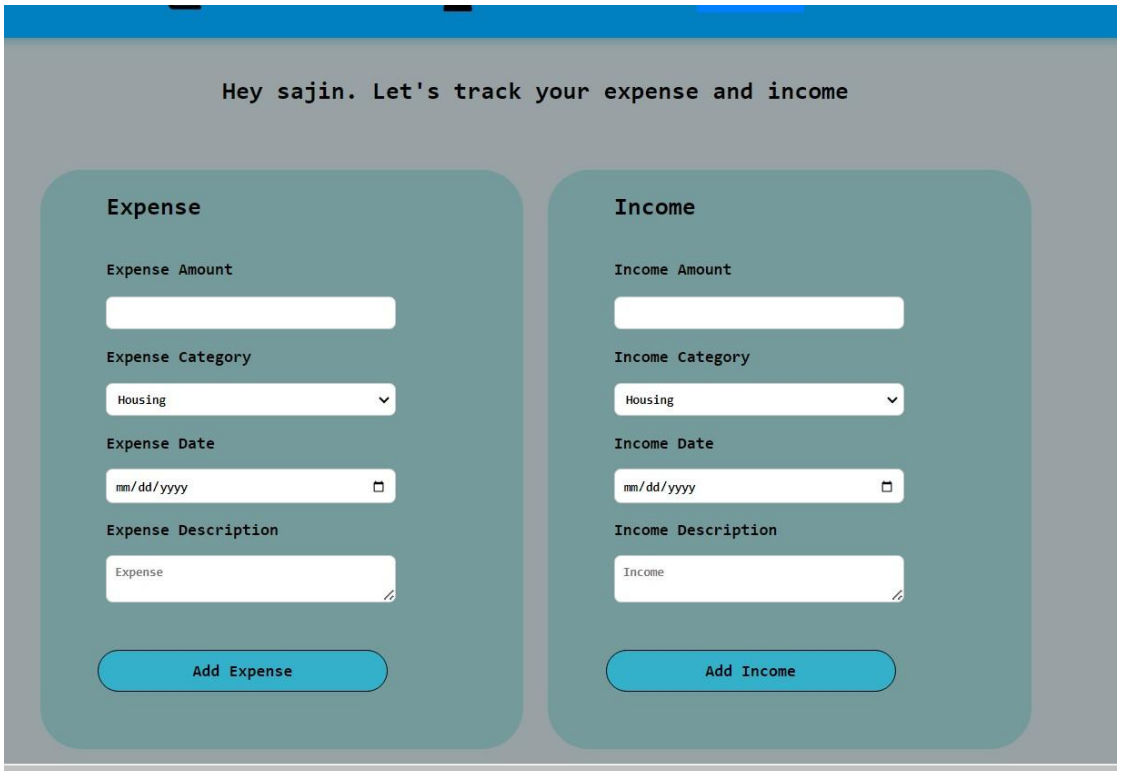


Figure 101 Design Error 2 after implementation of solution.

Reference source : https://www.geeksforgeeks.org/how-to-set-space-between-the-flexbox/

10. Conclusion

Hence, after the completion of the project “expensoft”, expense tracking system develop by using java web based technology, I got a chance to gain an experience on the field of full stack development which includes designing the user interface protype and developing it using html, CSS, and JavaScript, designing the database structure and performing CRUD operation with the help of SQL and servlet, learned the concept of JSP and JSTL for implementation of dynamic content in webpage that is being fetch by the server from the database, working mechanism of Request and response cycle for web server.

Developing this project was difficulty for me as I must focus on frontend, backend, and database development. By doing research, design analysis of database, help from platform like stack flow, medium and YouTube, I was able to clear my doubt regarding the technology that I found difficult. Continuous improvement and effort to the project led me to complete the project on time. Sometimes I was frustrated by the error generated during the development that wasn’t solving. From the guidance of teacher and fellow friends I was able to overcome the error.

11 References

alphaservesp. (2023, Februrary 8). *what-is-sql-database-structure-types-examples*. Retrieved from alphaservesp.com: https://www.alphaservesp.com/blog/what-is-sql-database-structure-types-examples

Amazon. (2024). *What is Java?* Retrieved from aws amazon: https://aws.amazon.com/what-is/java/

Balsamiq. (2024). *Balsamiq Brand Assets*. Retrieved from Balsamiq Assets: https://balsamiq.com/company/brandassets/

balsamiq. (2024). *balsamiq wireframe*. Retrieved from balsamiq.com: https://balsamiq.com/wireframes/desktop/docs/intro/#:~:text=Balsamiq%20Wireframes%20is%20a%20user,before%20any%20code%20is%20written.

Foundation, E. (n.d.). *download.eclipse*. Retrieved from Eclipse Foundation : https://download.eclipse.org/justj/?file=eclipse.org-common/themes/solstice/public/images/logo

Gyaan, S. (2023, september 27). *what-xampp-how-does-work-everything-you-need-know-server-site-gyaan*. Retrieved from linkedin: https://www.linkedin.com/pulse/what-xampp-how-does-work-everything-you-need-know-server-site-gyaan/

Kanade, V. (2022, July 11). *what-is-sql*. Retrieved from spiceworks: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-sql/

logo, F. (2024). *Flaticon logo*. Retrieved from Flaticon logo: https://www.flaticon.com/free-icon/jsp-open-file-format-with-java-logo_28968

logos-world. (n.d.). *logos-world.net* . Retrieved from logos-world: https://logos-world.net/allpng/

Nakoma. (2024). Retrieved from zenn.dev: https://zenn.dev/nakohama/articles/22bbb20398a5ad

Pankaj. (2022 , August 4). *JSTL Tutorial, JSTL Tags Example*. Retrieved from DigitalOcean : 2022

seeklogo. (2024). *xampp*. Retrieved from seeklogo: https://seeklogo.com/vector-logo/274098/xampp

Tomar, A. (2021, may 31). *JSP and Servlet in Java*. Retrieved from Medium : https://medium.com/nerd-for-tech/jsp-and-servlet-in-java-9981759972f0

tutorialspoint. (2024). *tutorialspoint/eclipse/*. Retrieved from tutorialspoint: https://www.tutorialspoint.com/eclipse/eclipse_overview.htm