Implementation and Testing

Group: Shahd Mustafa, Mike Nasser, Nathaniel Leonardo, Hassan Radwan, Hussen Al-Jubury

Table of contents:

User login ____ 3
Group management ____ 4
Expense creation ____ 4
Notification system ____ 5
Balance tracking ____ 7
Payment tracking ____ 7
Customer report ____ 10
Class Diagrams ____ 11
Database schema ____ 12
Test cases ____ 13

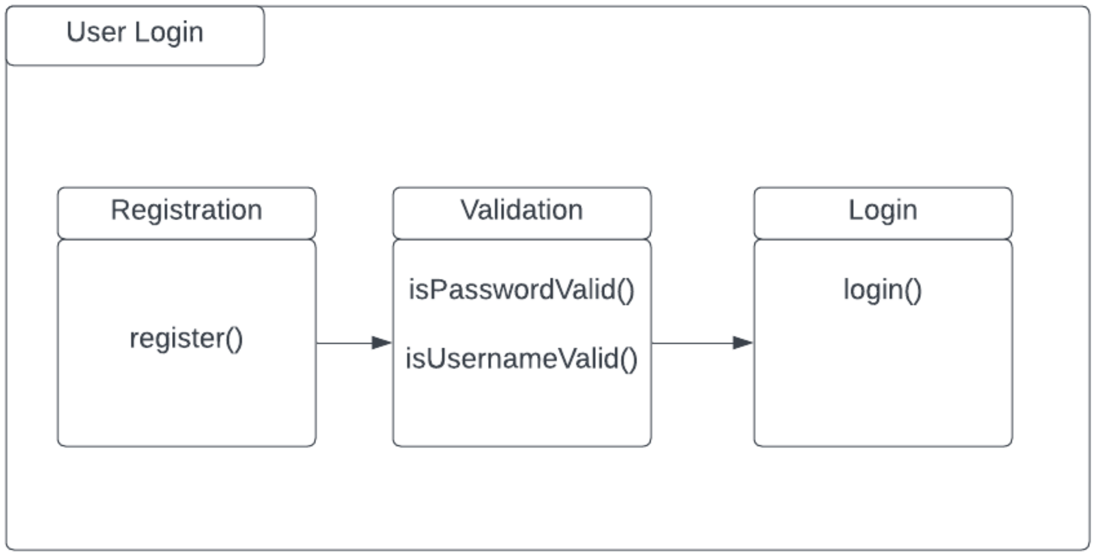
User login:

The user login handles account registration and user login. The user registration allows for users to enter in a valid user name and password to register into the system. A valid user name consists of 5 characters including 1 letter and 1 number, valid password consists of 8 characters including 1 uppercase letter, 1 lowercase, 1 number and 1 special character. If the user enters an invalid username or password to register a message will be displayed and the user will need to re-enter the input. When a valid username and password is registered the information is stored, the user can now login to the system. Using the valid username and password the user is directed to the Split Smart homepage where they can access all the site's features and enter in their account details. However if the user enters an invalid username or password, a message will be displayed and they will need to re-enter valid information to be directed to the homepage.

After entering a valid user login information the user is able to enter their account details that consists of creating their profile. Users are to provide their first and last name, email address, phone number, language, default currency and their timezone. The profile information is then stored and displayed in a new window for the user to confirm.

In regards to the design pattern for the user login, an interceptor pattern matches most closely with the ideal setup for this feature, as a user would input either their registration or login information which will then be taken and verified by the database, thus the need for an interceptor to handle that verification, as our current iteration remains local to the users machine this is not the ideal way it would be represented.

Component Diagram:

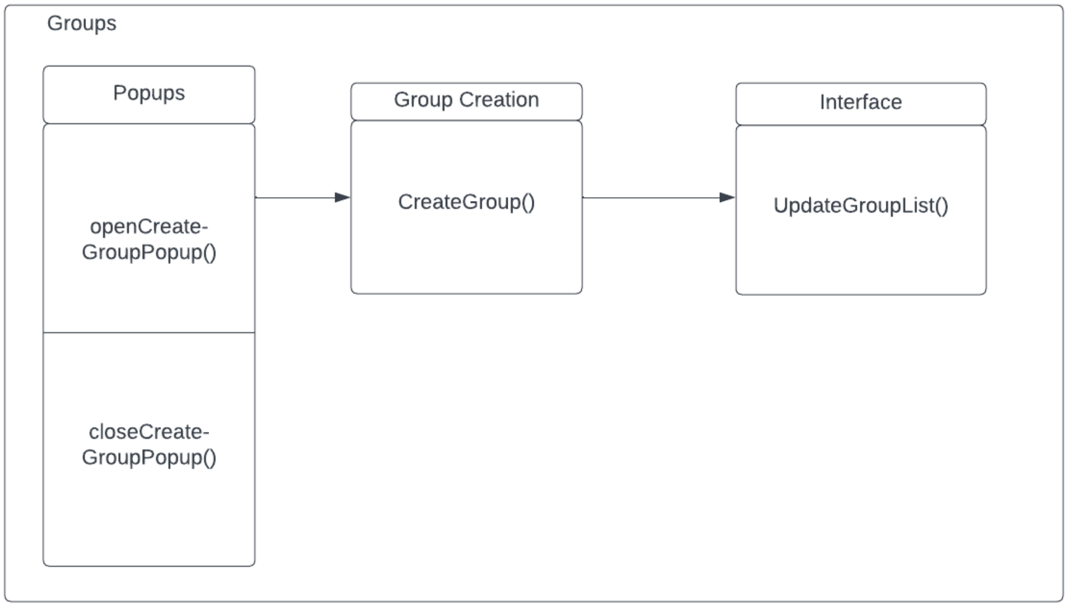


Group Management:

Users are able to create groups and invite friends to join the split smart application. When creating a group the user needs to enter a name and the members they wish to add. The system then checks if the members they wish to add have a valid account, if not they cannot be added. Upon creating a new group, the group name will be displayed on the users dashboard, of which they can select again to access the information on said group, and on creating multiple groups the list of Groups is then displayed on a list on the dashboard.

The Factory Method Design pattern would best apply to this section of the program, the groups as objects are created by the Class, they can then be adjusted later on the dashboard. The interface the user has is designed to allow the user to create multiple instances of groups so the Factory Method of design is appropriate.

Component Diagram:

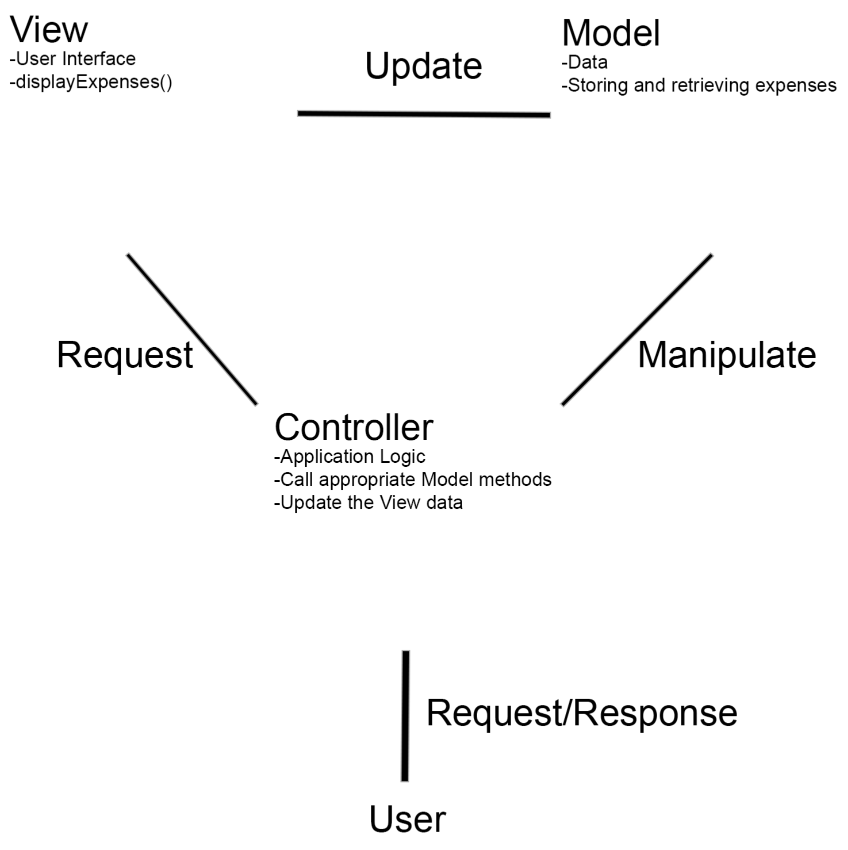


Expense Creation:

The user can create new expenses by inputting the following information: amount, sent to, date, description and split %. The expense is sent to the members entered and stored within the system.

The design pattern applied to create this expense management system is the Model-View-Controller (MVC) pattern, which helps to separate and organize code in a way that promotes maintainability and scalability. In the case of the expense creation feature, the model represents the data and business logic of the application and handles defining the expense data structure (amount, date, description, person, etc). The view, which is responsible for presenting the user interface to the user and is represented by the HTML template with the form and UI elements to input and display information. The controller whose job is to receive input from the view and interface it with the model can be seen in the manner in which the program handles the submissions of expense forms and extracting expense details from user input.

Component Diagram:

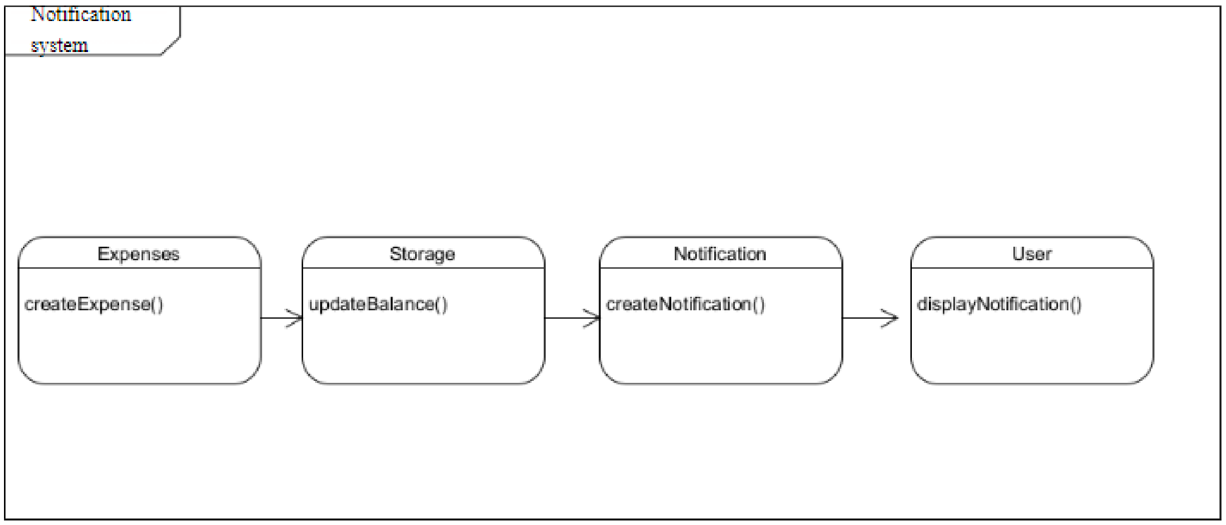


Notification System:

Notifications are sent out to users emails when a new expense has been created and when an invite has been sent. The notification of the new expense includes the name of the user who created the expense, the amount, date, description and the link to the website. Through this users are notified and are able to pay off the expense. On the other hand when an invite is sent, the notification consists of the group name, user who sent it, and the website link.

Due to limitation of resources and time of the project this feature was not implemented fully in the project site.

Component Diagram:



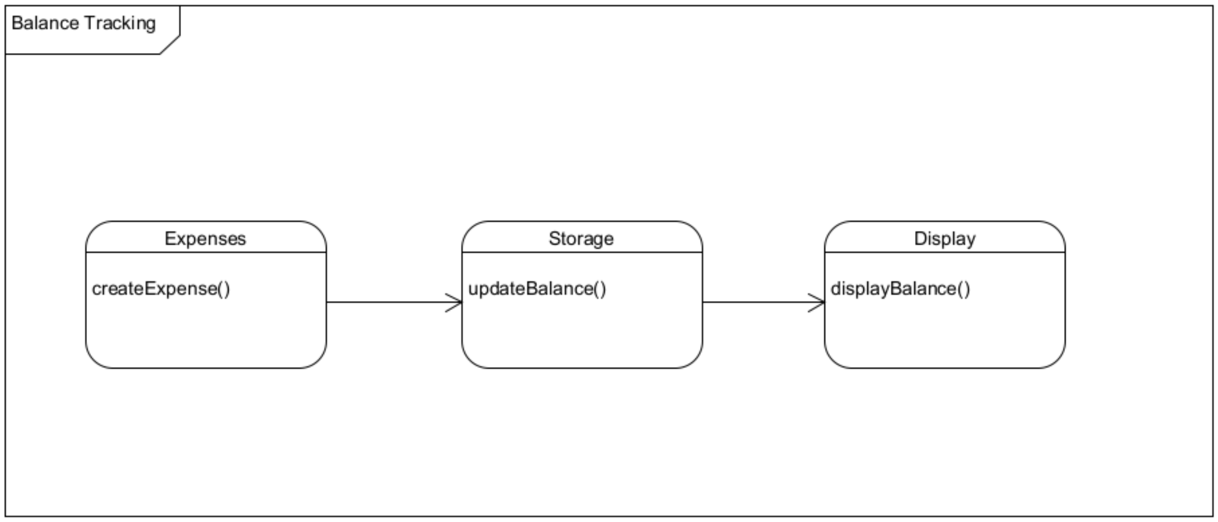
Code:
```
class NotificationSystem {
  emailStruct() {
    super();
    this.email = '';
  }

  setEmail(email) {
    this.email = email;
  }

  sendInviteNotification(inviteData) {
    // Logic to send invite notification
    const notification = `You have received an invite: ${inviteData}`;
    this.notify(notification);
  }

  sendExpenseNotification(expenseData) {
    // Logic to send expense notification
    const notification = `A new expense has been made: ${expenseData}`;
    this.notify(notification);
  }
}
```

This code would be implemented using the observer design pattern, the subject is the notification class and the observer is the email function. The code runs when a new invite is sent or a new expense has been created, the subject is notified and the observer (email function) sends out the notification. This is done by sending an email to the user based on the pre registered email address.

Balance Tracking:

The balance is updated when new expenses have been charged to a user and when payments are made. It is displayed at the top of the dashboard.

Component Diagram:



Payment Tracking:

The user is able to fulfill ongoing payments through the system, using their credit card information. Paying off an expense will update the ongoing balance of the user account and will be saved within the system. All previously made payments will be stored and could be viewed through the user report.

Due to the time limit and limitations of the project this feature could not be implemented in the split smart site. However below is an example of how it would be coded.

Code:

```
<!--Payment feature-->

  <div class="popup" id="paymentPopup">
    <div class="overlay"></div>
    <div class="content">
      <div class="close-btn" onclick="closePaymentPopup()">&times;</div>
      <h1>Make Payment</h1>
      <p>Select expense to pay:</p>
      <select id="expenseSelect">
```

```
          <option value="">Select an expense</option>
        </select>
        <button onclick="makePayment()">Payment completed</button>
      </div>
    </div>

    <script>
      function displayExpenses() {
  const expenseList = document.getElementById('expenseList');
  expenseList.innerHTML = '';

  expenses.forEach((expense) => {
    const expenseItem = document.createElement('div');
    expenseItem.innerHTML = `
      <p>Amount: $${expense.amount}</p>
      <p>Sent to: ${expense.person}</p>
      <p>Date: ${expense.date}</p>
      <p>Description: ${expense.description}</p>
      <p>Payment Status: ${expense.paymentStatus}</p>
      <button onclick="openPaymentPopup(${expense.id})">Make Payment</button>
      <hr>
    `;

    expenseList.appendChild(expenseItem);
  });
}

    // expense payment popup
    function paymentExpense() {
    const expenseSelect = document.getElementById('expenseSelect');
    expenseSelect.innerHTML = '<option value="">Select an expense</option>';

    expenses.forEach((expense) => {
        if (expense.paymentStatus === 'Pending') {
        const option = document.createElement('option');
        option.value = expense.id;
        option.textContent = `${expense.description} ($${expense.amount})`;
        expenseSelect.appendChild(option);
        }
    });
    }

    // payment popup - open
    function openPopup(expenseId) {
    populateExpenseSelect();
    const paymentPopup = document.getElementById('paymentPopup');
    paymentPopup.classList.add('active');
    }

    // payment popup - close
    function closePopup() {
    const paymentPopup = document.getElementById('paymentPopup');
    paymentPopup.classList.remove('active');
    }

    // Make a payment and update the payment status
    function makePayment() {
    const expenseSelect = document.getElementById('expenseSelect');
    const selectedExpenseId = parseInt(expenseSelect.value, 10);

    if (!selectedExpenseId || isNaN(selectedExpenseId)) {
      alert('Please select an expense to mark as paid.');
      return;
    }

    const selectedExpense = expenses.find(expense => expense.id == selectedExpenseId);
    if (selectedExpense) {
      selectedExpense.paymentStatus = 'Paid';
      closePaymentPopup();
      displayExpenses();
    }
}
     </script>
```

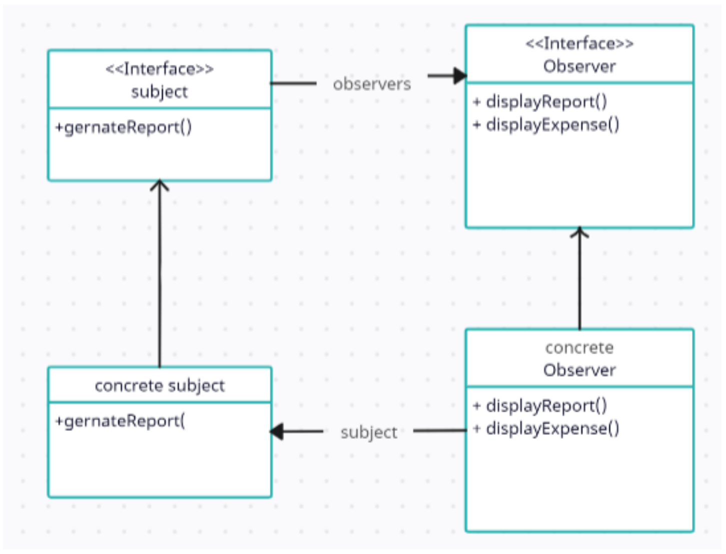
The payment code could be coded with the observer design pattern or the singleton design pattern. If the observer pattern was used to change the payment status when new expenses are paid off, the observer system waits for changes to be made to the payment status and it will automatically update. When the “makePayment()” function is updated it notifies the display function. On the other hand, if the singleton pattern is used, it would allow for one payment popup per expense. However, this may not be ideal if the user wants to pay off the expense in parts. Or one payment pop up could be created to be used through the entire site.

Customer Report:

When the user requests for a report to be created a popup window appears that asks the user for a general report or a custom report. A general report displays all payments that the user has made through the system. Meanwhile, a custom report will request for a date range, if the system finds any payments made within the time given a report will be displayed, if the system doesn't find any payments the system will display a message to the user.

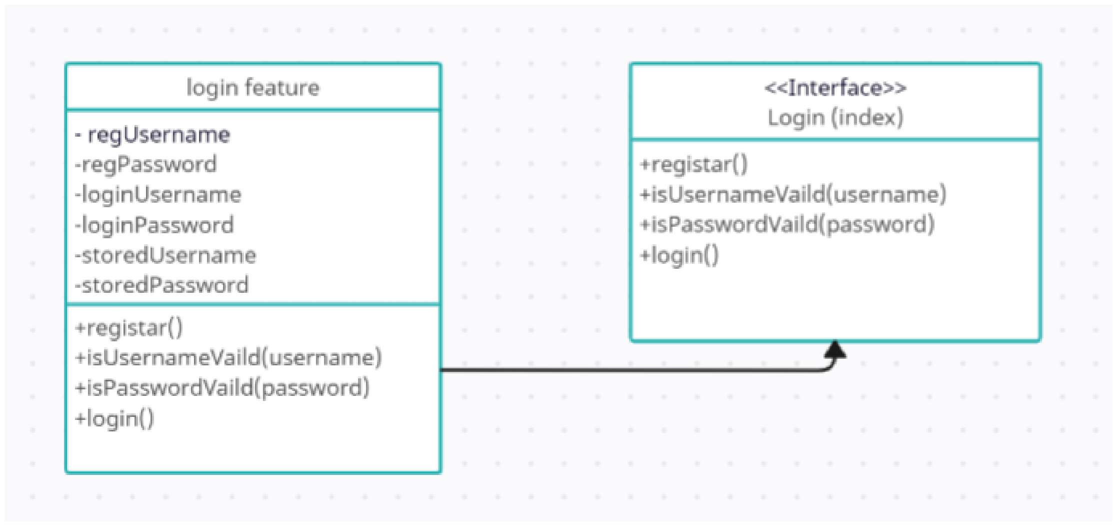
The design pattern implemented within the customer report code is the observer pattern, it notifies other functions when changes occur. Within the code the functions “displayExpense and “displayReport()” are the observers and the function “generateReport() is the subject. When the user requests a custom report the “generateRport()” function looks for any expenses with a matching date then notifies the “displayReport()” function to update the interface and display the data.

Component Diagram:

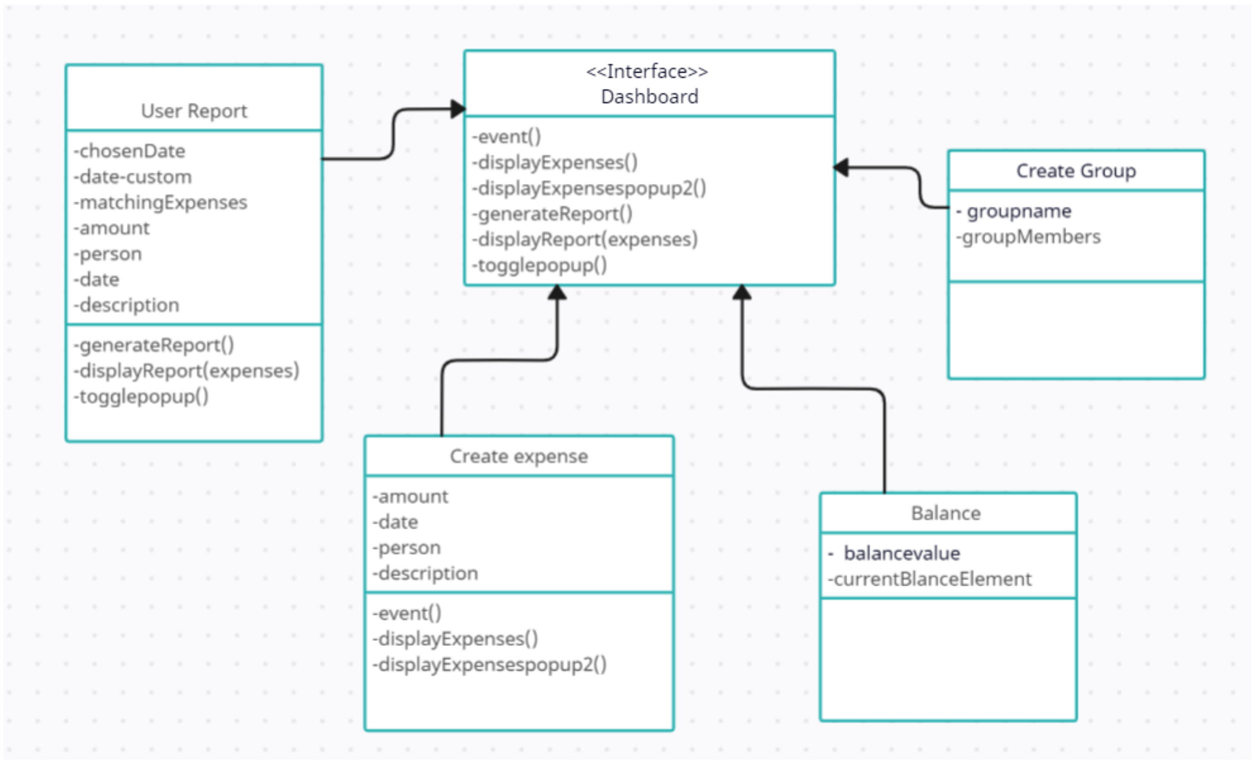


Class diagrams:

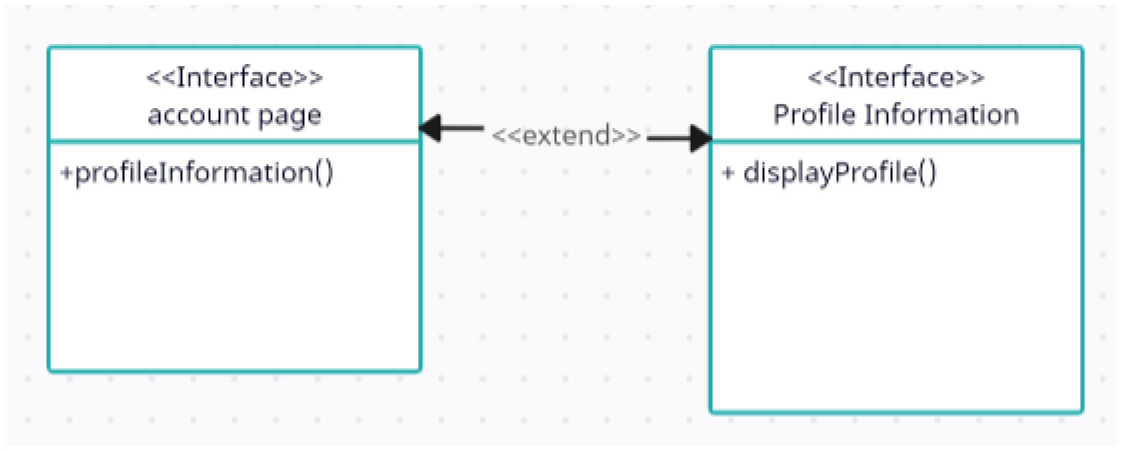
Login page:



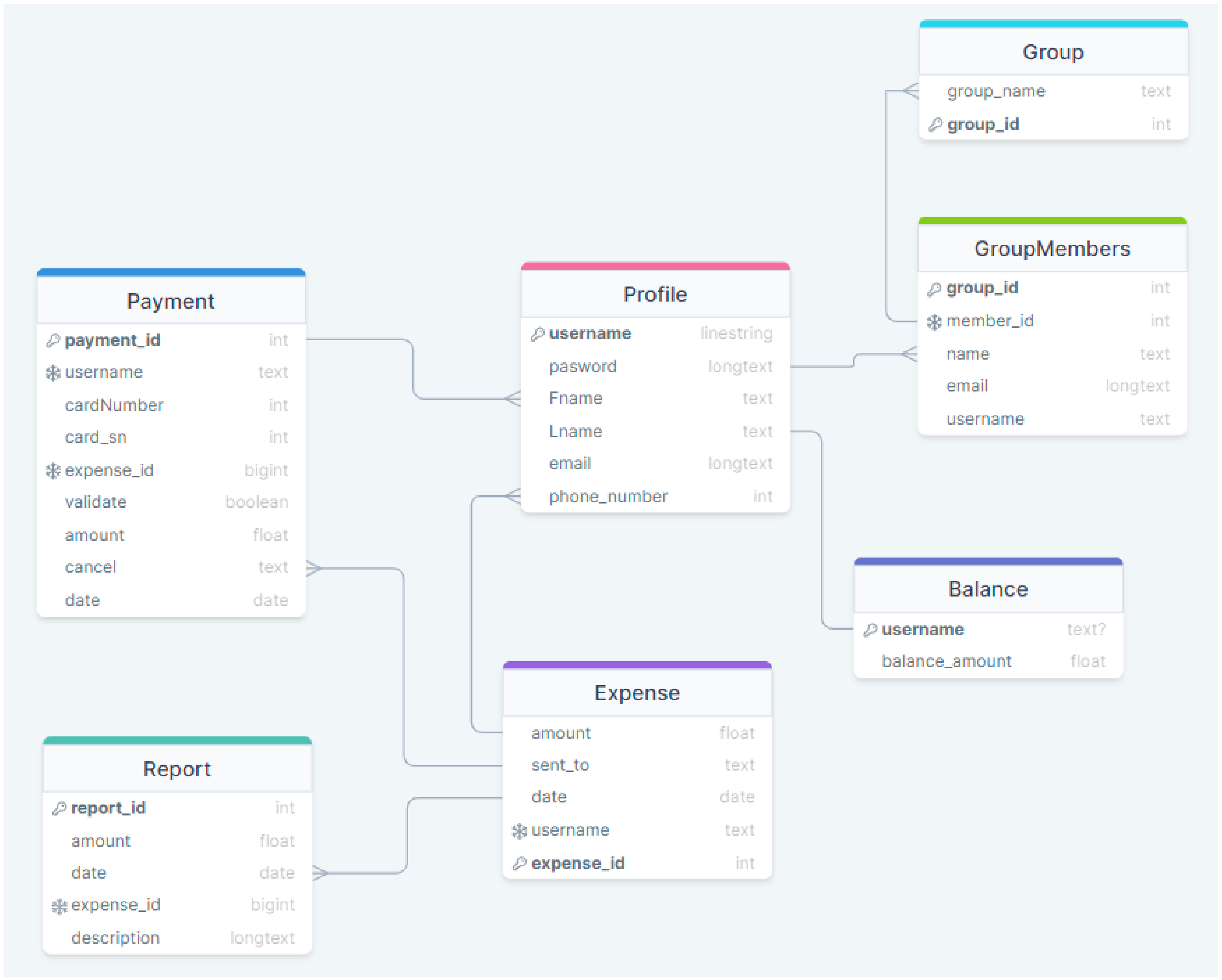
Dashboard:



Account:



Database Schema:



Schema overview:

- Profile (**username(primary key)**, password, name, first name , last name, email, phone number)
- Expense(amount, sent to , date, description, username(foreign key from profile)**, expense_id(primary key)**)
- Balance(**username(foreign key to profile)**, balance)
- Payment(**payment id(primary key),** username(foreign key from profile), card number, card security number, expense id(foreign key from expense), cancel**,** validate, amount, date)
- Report(**report id,** amount, date, description,expense id(foreign key from expense))
- Group(group_name,**group_id(primary key)**)
- GroupMembers(group_id(foreign key from group),**member_id**(primary key), name, email, username(foreign key from profile))

Relationships:

- Profile and expense (one to many): one user profile can have multiple expenses created. Using the foreign key "username to access the expense table).
- Profile and balance (one to one): one user profile can only have one balance. Using the "username" key to access the balance table. The balance is updated for each account.
- Profile and payment (one to many): one user profile can make multiple payments.
- Profile and group members (one to many): one profile can be a member to multiple groups. A member can join multiple groups but members in the group can not repeat.
- Group and group members (one to many): one group can have multiple members.
- Expense and payment (one to many): one expense can have multiple payments. They share the foreign key "expense_id" to access the table.
- Expense and report (one to one): one expense can only be reported once.

Test cases:

Test number	Feature testing	Description	Input data	valid/ invalid data	Expected output	Output data	Pass / fail
1	User login - registration	Valid username and password inputted	Username: Shahdmustafa1 Password: Shahdmustafa12!	valid	"Registration successful! You can now login."	"Registration successful! You can now login."	PASS
2	User login - registration	Invalid username and	Username: Shahdmustafa	invalid	"Invalid username or	"Invalid username or	PASS

		password inputted	Password: Shahdmustafa12		password. Please try again.”	password. Please try again.”	
3	User login	Valid username and password inputted	Username: Shahdmustafa1 Password: Shahdmustafa12!	valid	"Login successful! Redirecting to the dashboard..."	"Login successful! Redirecting to the dashboard..."	PASS
4	User login	Invalid username and password inputted	Username: Shahdmustafa Password: Shahdmustafa12	invalid	“Invalid username or password. Please try again.”	“Invalid username or password. Please try again.”	PASS
5	Group management	Upon clicking “create group” button on dashboard, a popup will display prompting users for numerous inputs	Text, drop window selection	valid	Popup is displayed to user with input prompts	Popup is displayed to user with input prompts	PASS
6	Group management	Upon creating multiple groups, the different groups are displayed on the dashboard on a list	Multiple group creations	valid	List of groups generated	List of groups properly generated	PASS
7	Group management	Attempting to leave a group name unentered would stop the user and prompt to input name	Group creation, exception of inputs	Invalid	Popup notification prompting user to input field	Popup notification prompting user to input field	PASS

8	Expense creation	Upon clicking the submit “Submit” button, an expense will be added to the list and be displayed below the prompts.	100, Man, 7/23/2023,Money money	valid	Amount: $100 Sent to: Man Date:7/23/2023 Description: Money money Approval Status: Pending	Amount: $100 Sent to: Man Date: Description: Money money Approval Status: Pending	FAIL-Date missing
9	Expense creation	Upon clicking submit “Submit” button, an expense will be added to the list and be displayed below the prompts.	100, Man, 7/23/2023,Money money	valid	Amount: $100 Sent to: Man Date:7/23/2023 Description: Money money Approval Status: Pending	Amount: $100 Sent to: Man Date:7/23/2023 Description: Money money Approval Status: Pending	PASS
10	Expense creation	Upon clicking submit “Submit” button, an expense will be added to the list and be displayed below the prompts.	abc, dfe, 7/23/2023,123	invalid	Unable to input incorrect data	Unable to input incorrect data	PASS
11	Report - general	Past expenses are to be displayed in the popup	General button selection	Valid	General button selected (previously created an	Amount: $90 Sent to: shahd	FAIL (date is missi

		window. All components should appear.			expense)	Date: Description: lunch. Approval Status: Pending	ng)
12	Report - general	All expenses made to the system should appear when requested.	General button selection	Valid	General button selected (previously created an expense)	Amount: $90 Sent to: shahd Date: 2023 - 09 - 09 Description: lunch. Approval Status: Pending	PASS
13	Report - custom	All expenses done for the date chosen are displayed in the popup window	Date: 2023-07 - 20	Valid	Amount: $90 Sent to: shahd Date: 2023-07-20 Description: lunch. Approval Status: Pending	Amount: $90 Sent to: shahd Date: 2023-07-20 Description: lunch. Approval Status: Pending	PASS
14	Balance Tracking	It should perform the proper calculations and show the balance either owed or is owed to the user.	Create expense button	Valid	50	Amount: $100 Sent to: Hussen Date: 2023 - 07 - 31 Description:	PASS

						lunch. Split %: 50	
15	Balance Tracking	It should perform the proper calculations and show the balance either owed or is owed to the user.	Create expense button	Valid	130 - >180	Amount: $100 Sent to: Hussen Date: 2023 - 07 - 31 Description: lunch. Split %: 50	PASS
16	Profile information	All sections of the account information are inputted to the system. A new window should open with a confirmation of the information.	First name: shahd Last name: Mustafa Email: shahdmu@umich.edu Phone number: 313-333-3333	Valid	First name: shahd Last name: mustafa Email: shahdmu@umich.edu Number: 3133333333	First name: shahd Last name: mustafa Email: shahdmu@umich.edu Number: 3133333333	PASS
17	Profile information	Sections of the account information will be left blank to test that they are required in order to proceed.	first name: shahd Last name: Mustafa Email: Phone number: 313-333-3333	Invalid	“Please fill out this field” -Warning message should appear below the email field.	“Please fill out this field”	PASS