
Design Document

for

CampusPay

Version <1.0>

Prepared by

Group 2:

Aditya Bangar
Akshat Rajani
Harsh Bihany
Kalika
Monil Lodha
Pratham Sahu
Pulkit Gopalani
Ravija Chandel
Shantanu Kolte
Siddhant Jakhotiya

Group Name: TheMissingSemiColon

210069
210812
210406
210482
210630
210755
180564
210835
210958
211030

adityavb21@iitk.ac.in
rajanias21@iitk.ac.in
harshb21@iitk.ac.in
kalika21@iitk.ac.in
monill21@iitk.ac.in
spratham21@iitk.ac.in
gpulkit@iitk.ac.in
ravijac21@iitk.ac.in
shantanu21@iitk.ac.in
siddhantj21@iitk.ac.in

Course: CS253

Mentor TA: Sumit Lahiri

Date: Feb 10, 2023

Contents

CONTENTS	I
REVISIONS	II
1 CONTEXT DESIGN	1
1.1 CONTEXT MODEL	1
1.2 HUMAN INTERFACE DESIGN	1
2 ARCHITECTURE DESIGN	2
3 OBJECT-ORIENTED DESIGN	3
3.1 USE CASE DIAGRAM	3
3.2 CLASS DIAGRAM	3
3.3 SEQUENCE DIAGRAM	3
3.4 STATE DIAGRAM	3
4 PROJECT PLAN	4
5 OTHER DETAILS	5
APPENDIX A - GROUP LOG	6

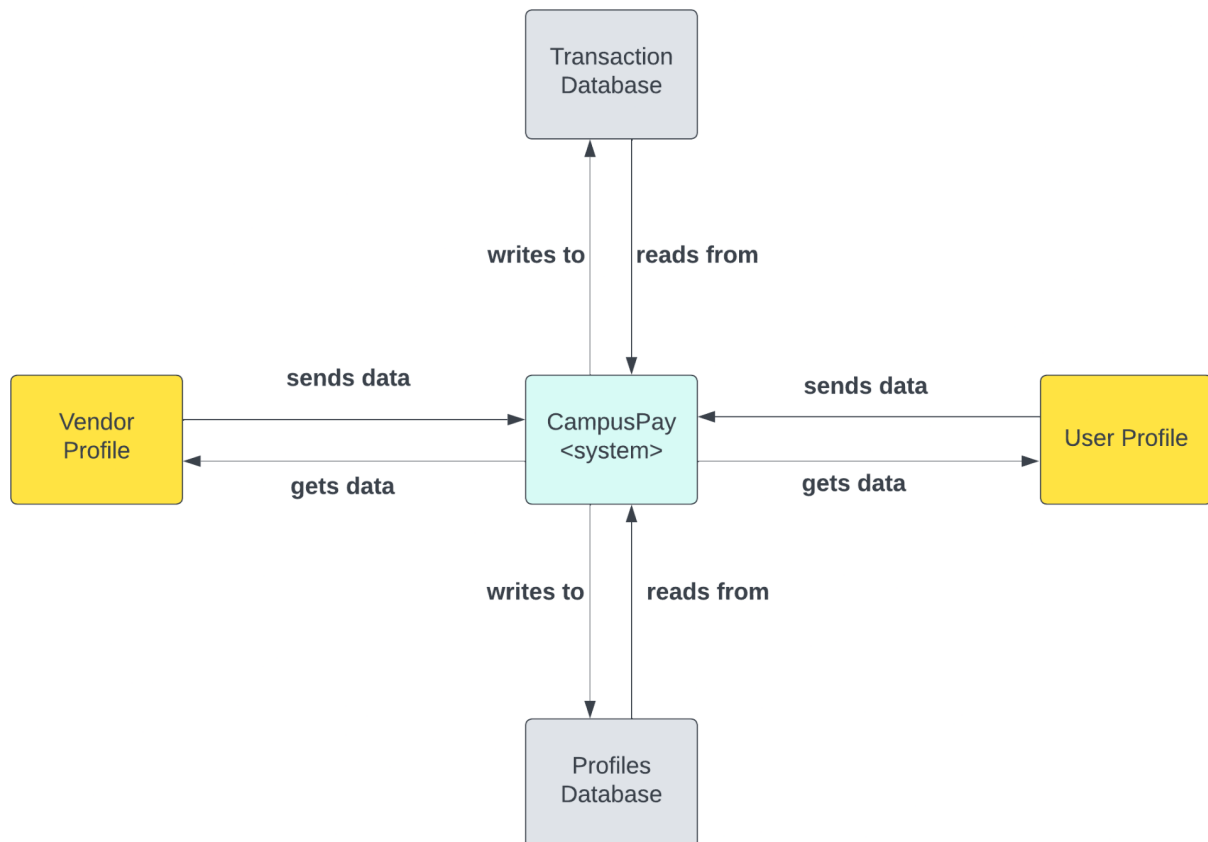
Revisions

Version	Primary Author(s)	Description of Version	Date Completed
Draft Type and Number	Full Name	Information about the revision. This table does not need to be filled in whenever a document is touched, only when the version is being upgraded.	00/00/00

1 Context Design

1.1

1.2 Context Model



1.3 Human Interface Design

1. Homepage

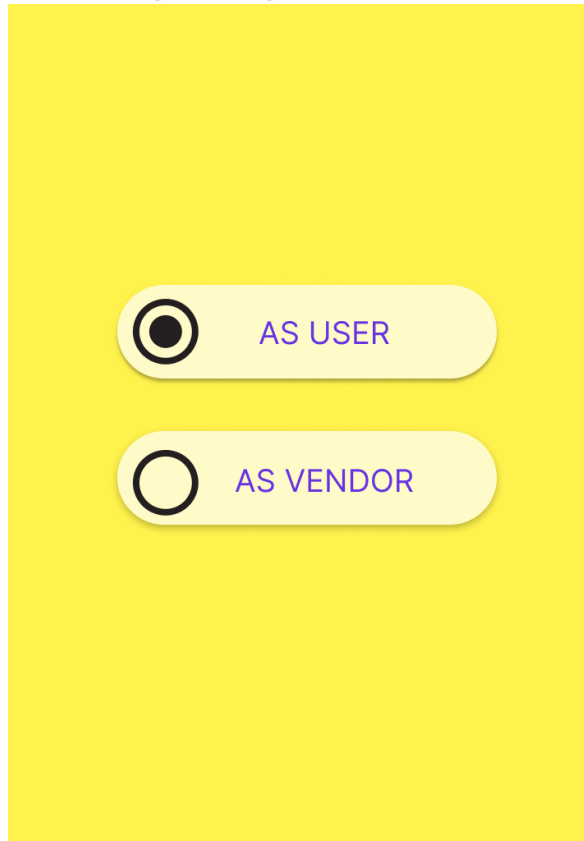
CAMPUS PAY

LOGIN

REGISTER



2. Login Page



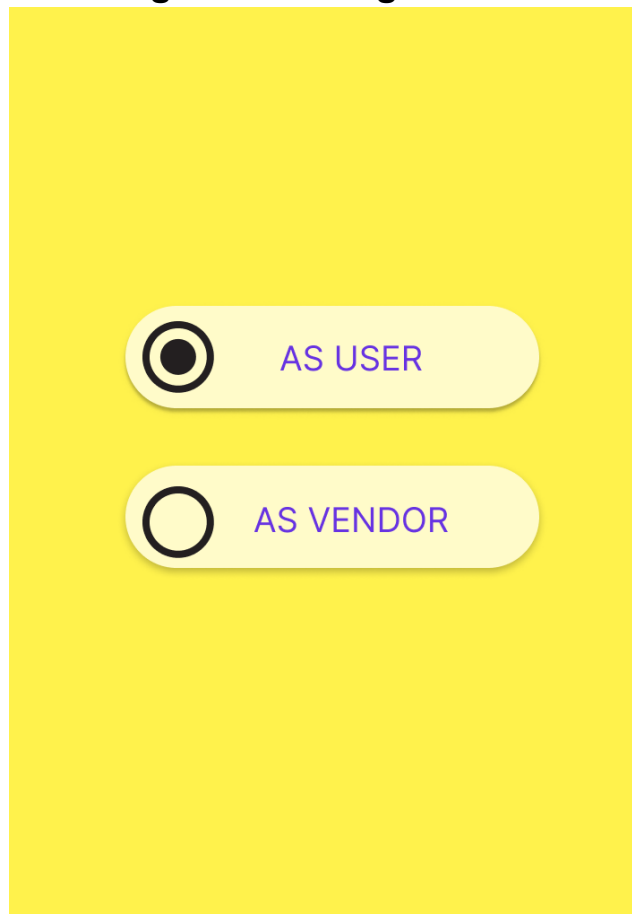
The image shows a yellow rectangular area representing a login page. Inside this area, there are two rounded rectangular buttons. The top button is light yellow and contains a black radio button icon followed by the text 'AS USER' in purple. The bottom button is also light yellow and contains a black radio button icon followed by the text 'AS VENDOR' in purple.

Username

Password

LOGIN

3. Registration Page



The registration page features a yellow background. On the left, there are two radio button options: 'AS USER' (selected) and 'AS VENDOR'. The 'AS USER' option is represented by a black circle with a white dot in the center, and the 'AS VENDOR' option is represented by a white circle with a black outline.

First Name

Last Name

Username

Phone

Password

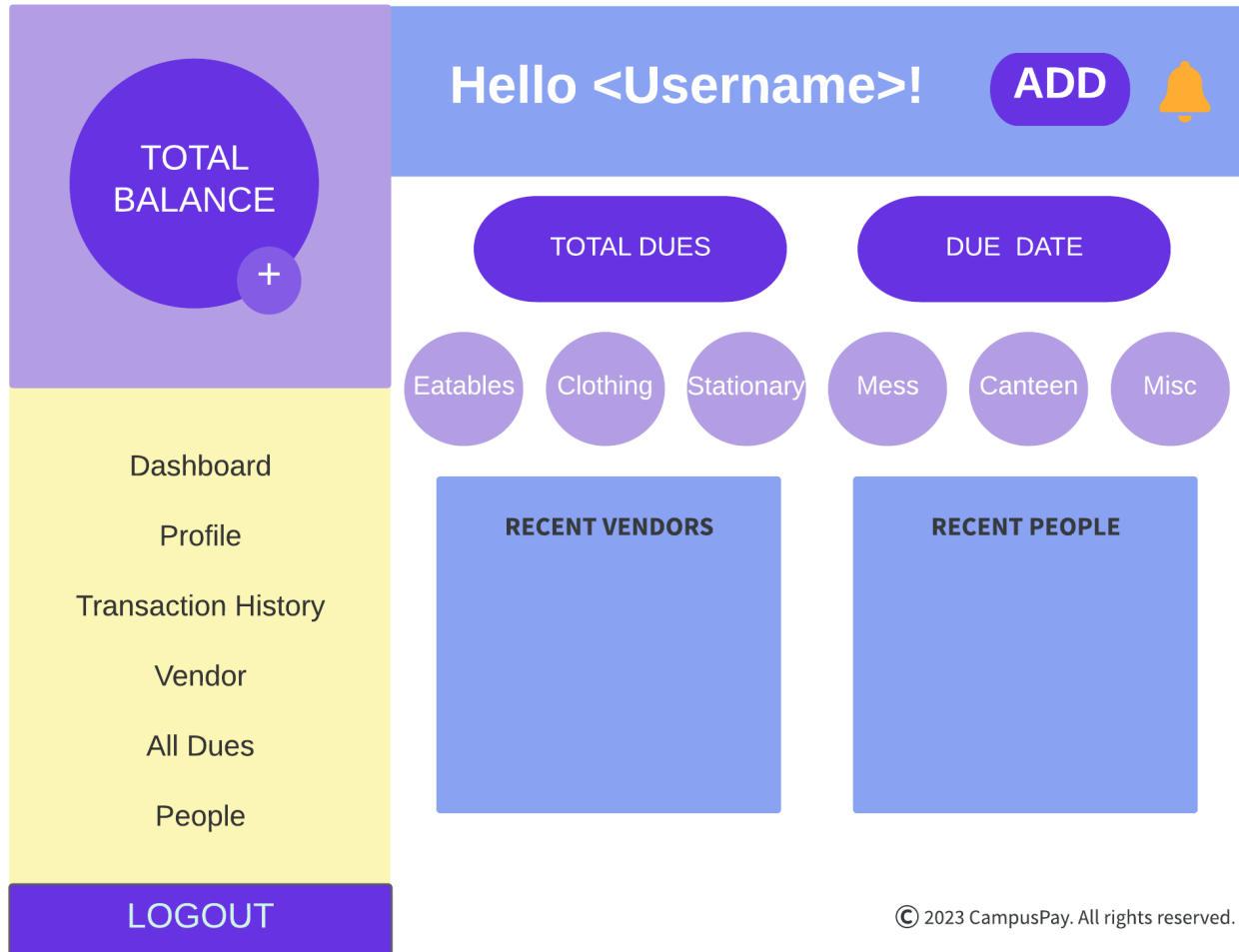
Confirm Password

Roll No./Alternate ID

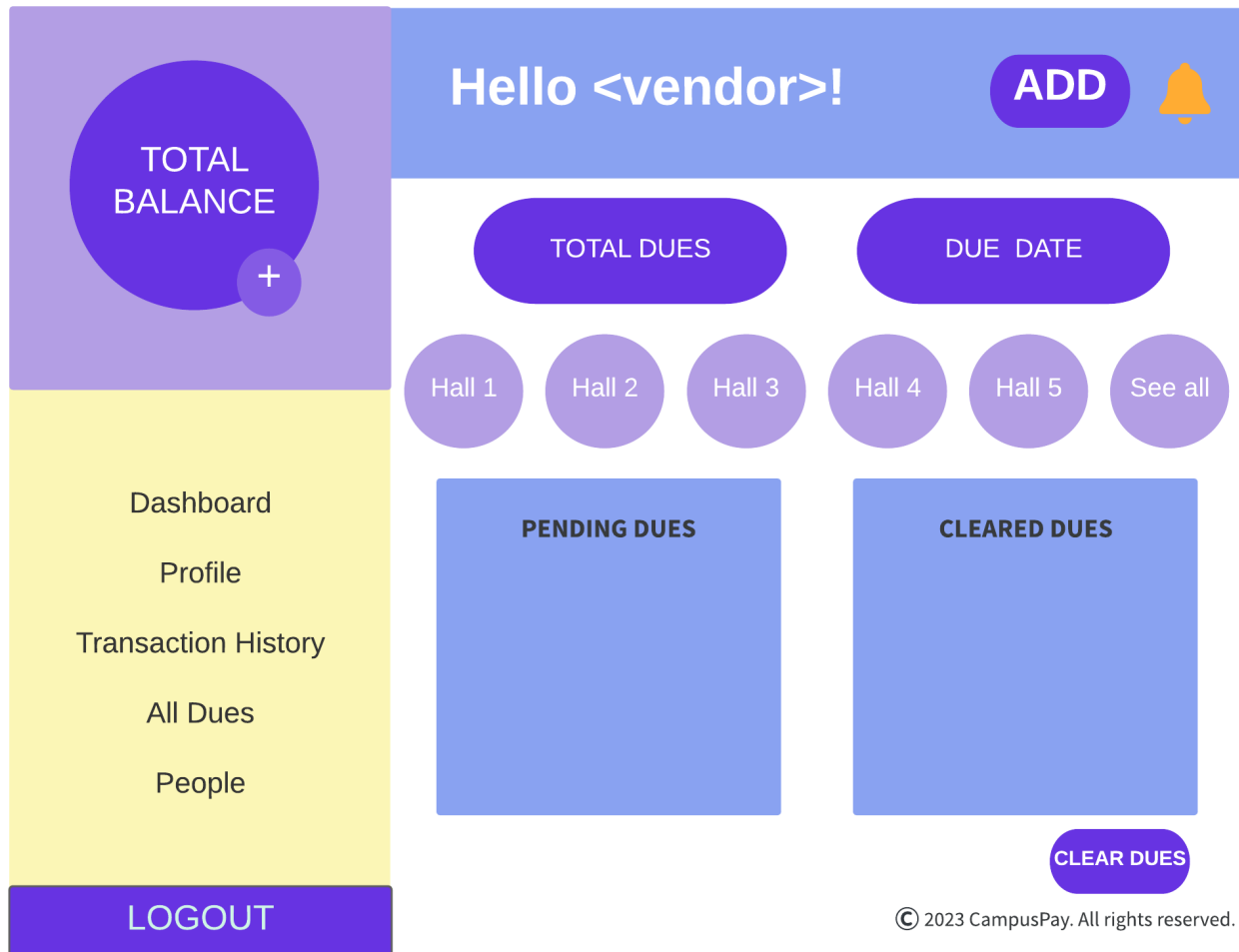
Email

REGISTER

4. User Dashboard



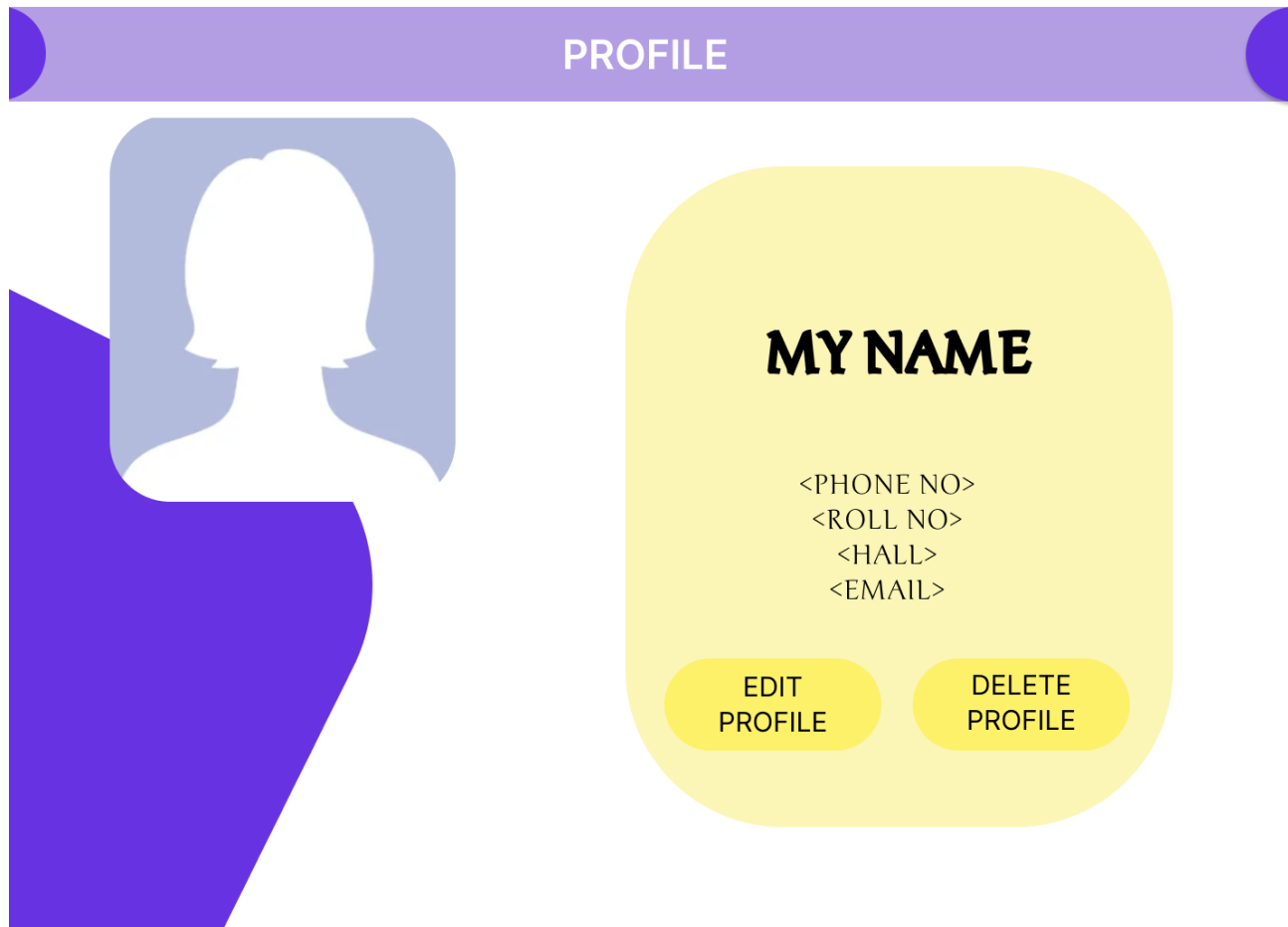
5. Vendor Dashboard



6. All Dues

ALL DUES	
VENDOR	DUE AMOUNT
Vendor A	₹ 1000
Vendor B	₹ 1500
Vendor C	₹ 2000

7. Profile

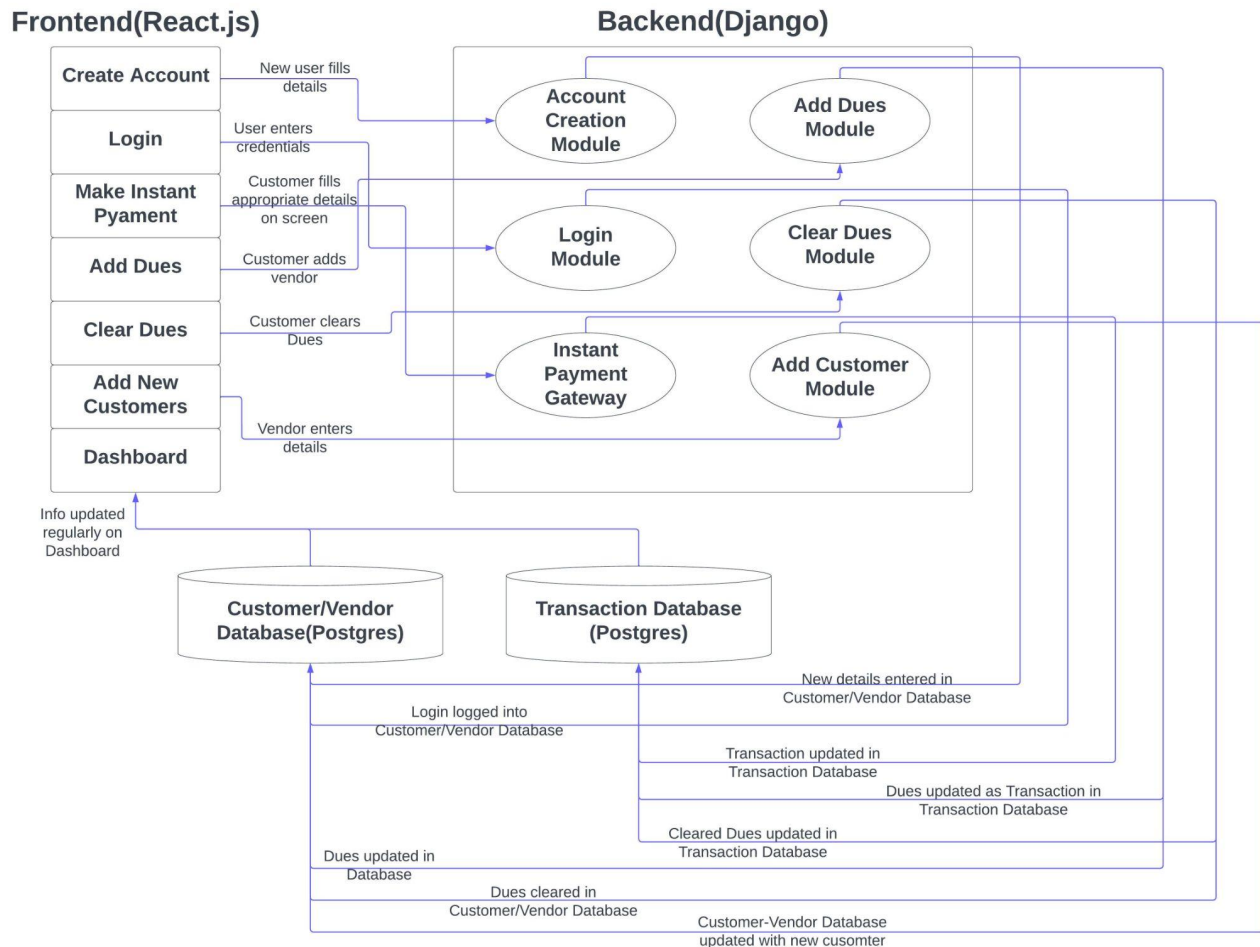


8. Transaction History

TRANSACTION HISTORY				
DATE	TIME	PAYMENT TO	AMOUNT	EMAIL
07/09/22	9:00 AM	xyz	₹ 50	xyz@iitk.ac.in
07/09/22	3:00 PM	abc	₹ 50	abc@iitk.ac.in
07/09/22	8:00 PM	mnp	₹ 50	mnp@iitk.ac.in

2 Architecture Design

We have the following tentative architecture diagram:

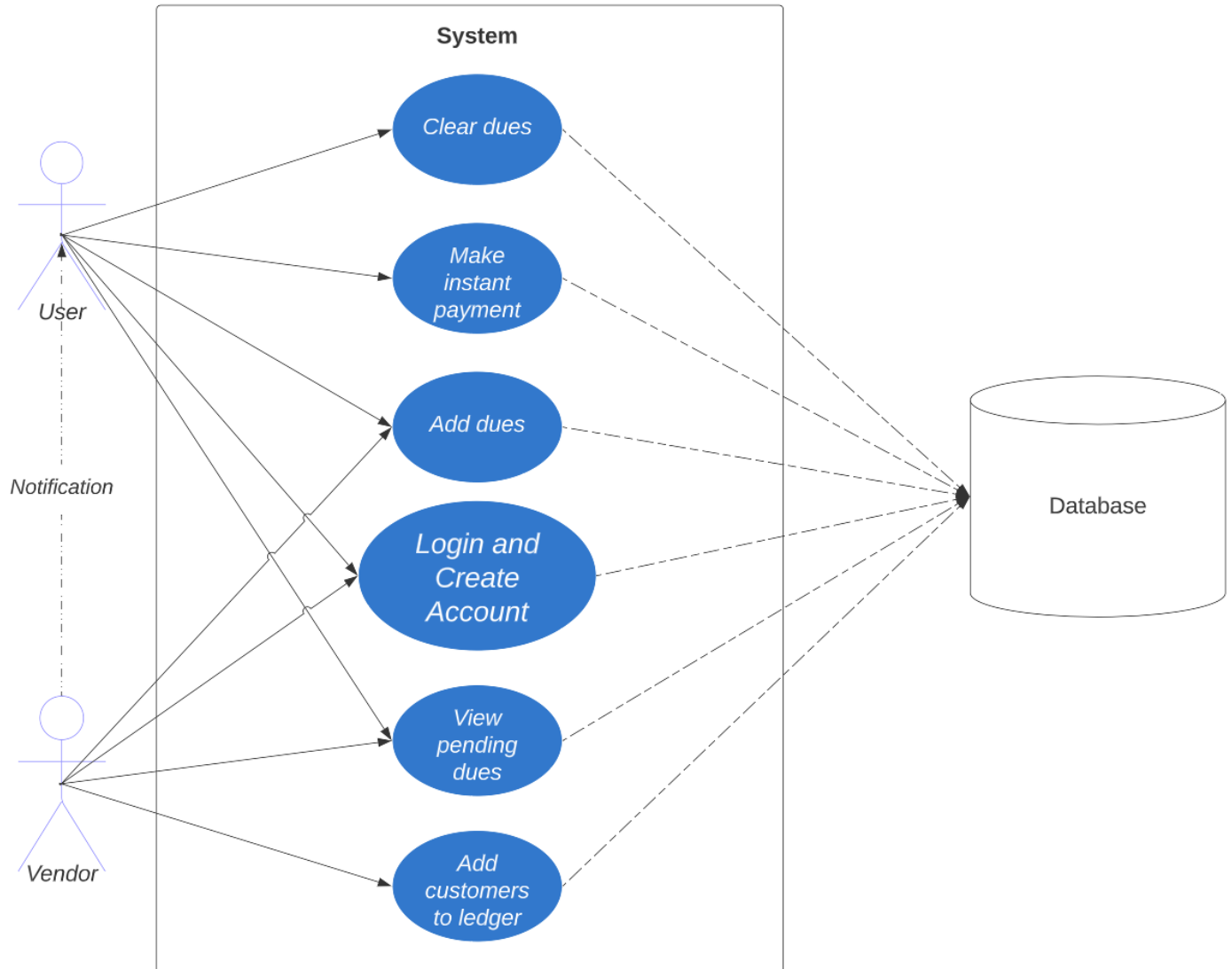


The Frontend section consists of the various actions which a user could take (in a broad sense). Corresponding to each such action, we show how the program logic works through the backend and the databases.

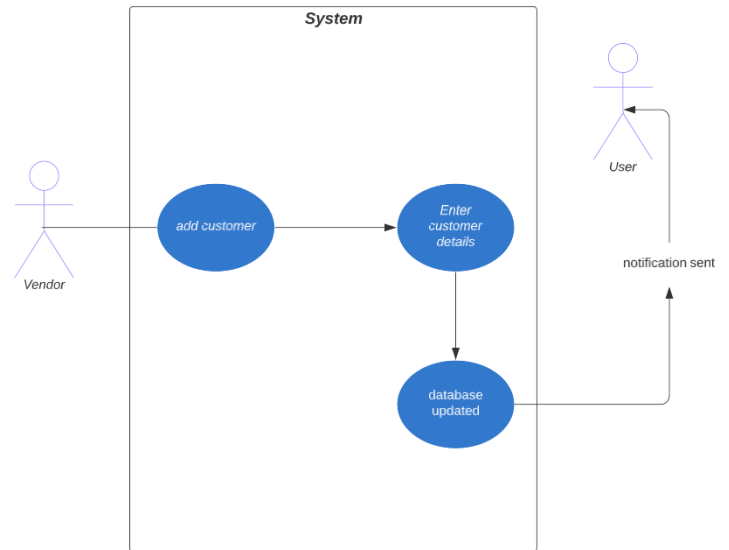
With the frameworks used herein, the non-functional requirements related to latency etc. are satisfied (please refer to our requirement document where we discuss these requirements in greater detail). Specifically, as indicated in the document, we choose Django as our choice of backend framework, React for Frontend and Postgres for database management.

3 Object Oriented Design

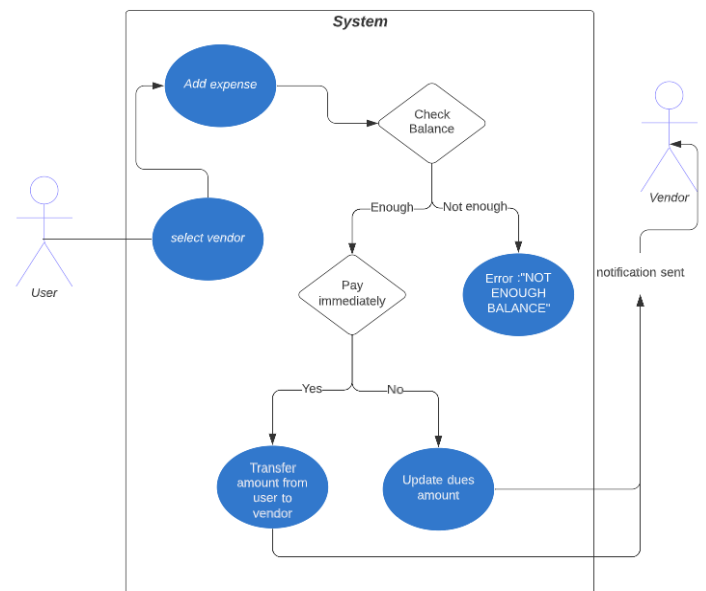
3.1 Use Case Diagrams



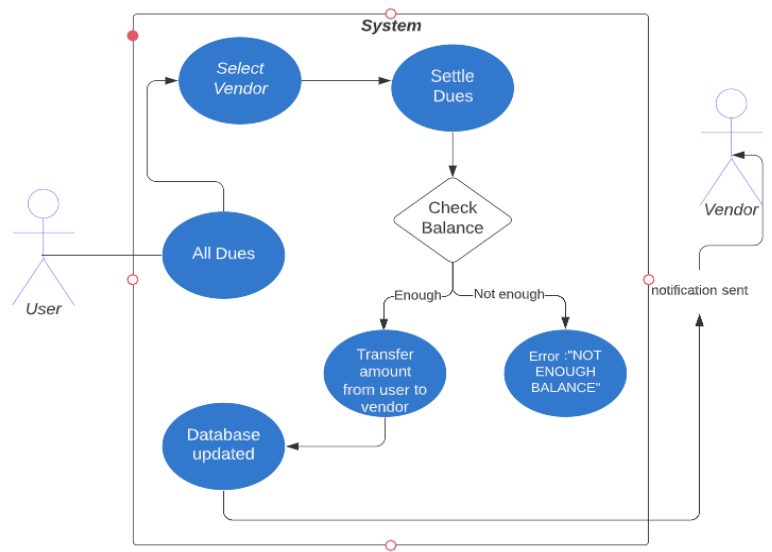
Vendor adding the user to his ledger



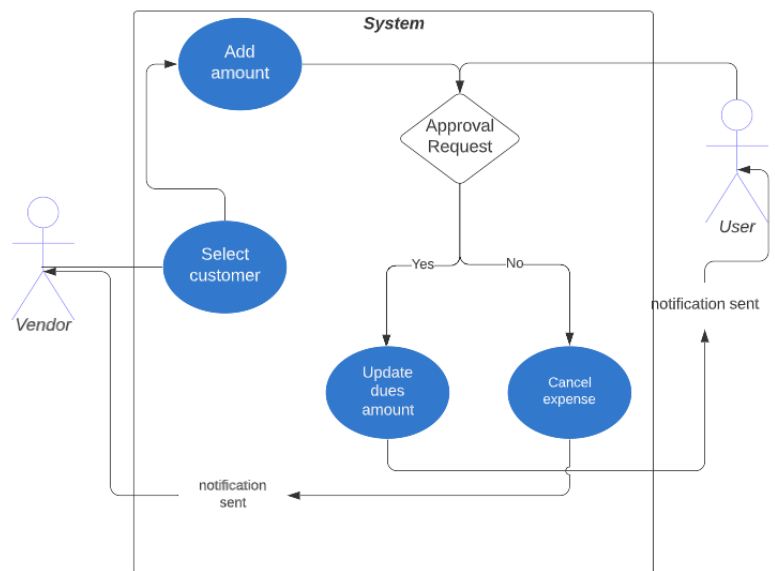
User adding an expense as dues
User making instant payment



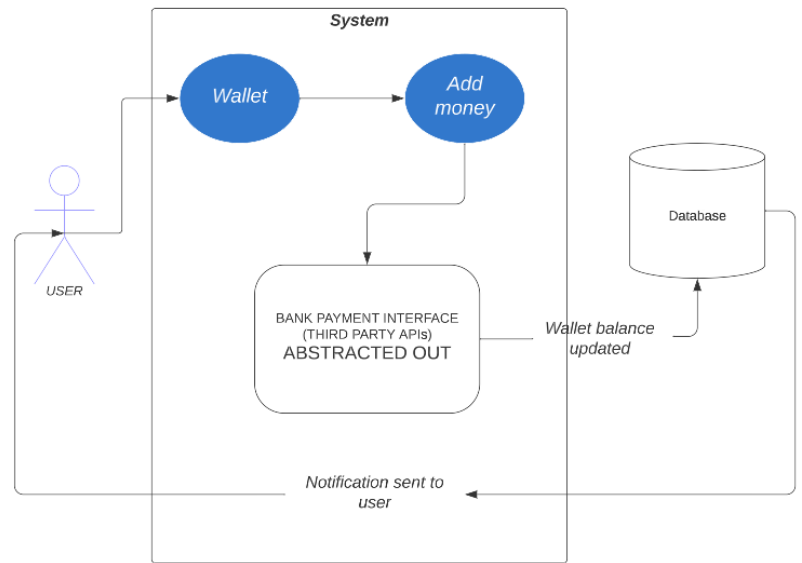
User settling his/her dues



Vendor adding an expense as dues/ reminding for a pending due

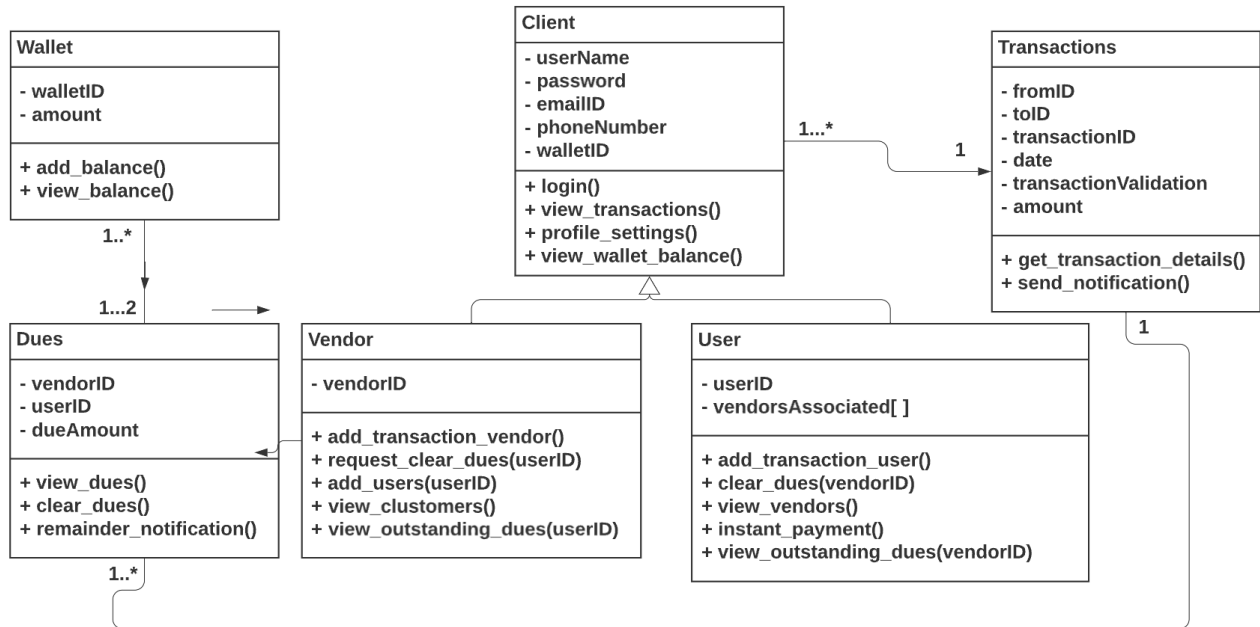


User adding money to wallet

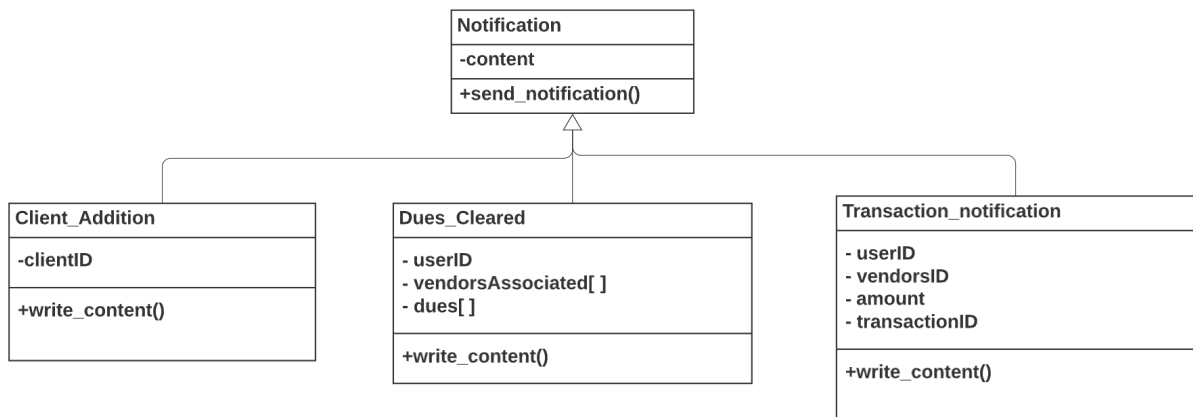


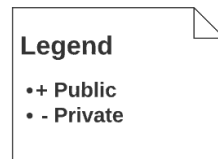
3.2 Class Diagrams

Class Diagram 1



Class Diagram 2





1) Class Diagram 1

a) **Client:** This class is the base class for all the end-users of the application.

i) **Data Members**

- (1) userName: It stores the login username of the client
- (2) password: It stores the login password of the client
- (3) emailID: It stores the email-id of the client.
- (4) phoneNumber: It stores the phone number of the client.
- (5) walletID: It stores the walletID of the wallet associated with the client.

ii) **Member Functions**

- (1) login(): This logs in the client to the application
- (2) view_transactions(): This allows the client to view all the transactions associated with his/her account.
- (3) profile_settings(): This allows a client to view his/her profile settings.
- (4) view_wallet_balance(): This allows the client to view his/her wallet balance.

b) **User: Client :** This class is the derived class indicating the users(Students and Professors) who use the application.

i) **Data Members**

- (1) userID: Unique alphanumeric code for each user
- (2) vendorsAssociated[]: This stores the associated vendors of the given users

ii) **Member Functions**

- (1) add_transaction_user() : This function is used to add transactions from the user-end
- (2) clear_dues(vendorID) : This function is used to clear dues with a given Vendor
- (3) view_vendors(): This is used to display vendors associated with a given ID
- (4) instant_payment(): This is used to make an instant payment to a given vendor.
- (5) view_outstanding_dues(): This is used to display outstanding dues of a given vendor.

- c) **Vendor: Client:** This class is the derived class indicating the vendors who use the application.
- i) **Data Members:**
 - (1) vendorID: Unique alphanumeric code for each vendor.
 - ii) **Member Functions:**
 - (1) add_transaction_vendor() : This function is used to add transactions from the vendor-end
 - (2) request_clear_dues(userID) : This is used to request clearance of dues
 - (3) add_users(userID): This is used to add a user associated with the vendor
 - (4) view_customers(): This displays the customers associated with the given vendor
 - (5) view_outstanding_dues(): This displays the list of outstanding dues of each user associated with the vendor.
- d) **Wallet:** This class templates an object that represents a wallet and has the following data members and member functions
- i) **Data Members:**
 - (1) walletID: This stores the unique alphanumeric ID of each wallet
 - (2) amount: This stores the balance amount in the wallet
 - ii) **Data functions:**
 - (1) add_balance(): This is used to add money to the wallet.
 - (2) view_balance(): This is used to view balance amount in the wallet.
- e) **Dues:** This class represents a due amount. It is composed of the following data members and member functions.
- i) **Data Members:**
 - (1) vendorID: This denotes the vendorID associated with the given due.
 - (2) userID: This denotes the userID associated with the given due.
 - (3) dueAmount: This denotes the amount of the due.
 - ii) **Member functions:**
 - (1) view_dues(): This is used to view the details of the views.
 - (2) clear_dues(): This is used to clear the dues.
 - (3) reminder_notification(): This is used to send a reminder notification for due clearance.
- f) **Transactions:** This class represents a transaction. It is composed of the following data members and member functions.
- i) **Data Members:**
 - (1) fromID: This denotes the ID of the sender
 - (2) userID: This denotes the ID of the receiver

- (3) transactionID: This denotes unique alphanumeric transaction Id of the transaction.
- (4) date: This stores the date of the transaction
- (5) transactionValidation: This stores the validation status of the transaction.
- (6) amount: This stores the amount associated with the transaction.

ii) **Member functions:**

- (1) get_transaction_details(): It gets the transaction details of the given object
- (2) send_notification(): It sends a notification to notify a transaction.

2) Class Diagram 2

- a) **Notification:** This class is the base class controlling all the notifications among users, vendors and system.

i) **Data Members**

- (1) content: It store the message to be delivered while sending the notification

ii) **Member Functions**

- (1) send_notification(): It sends the notification to the target client

- b) **Client_Addition: Notification** : This class is derived from the Notification class and it handles the notificaitons at the time of addition of client to the server.

i) **Data Members**

- (1) clientID: It stroes the ID of the client to be added.

ii) **Member Functions**

- (1) write_content(): This function generates the message to be delivered at the time of client addition and stores it in content variable of base class.

- c) **Dues_Cleared: Notification** : This class is derived from the Notification class and it handles the notificaitons at the time of dues clearance.

i) **Data Members**

- (1) userID: It stroes the ID of the user that has cleared the dues.
- (2) vendorsAssociated[]: It stores the ID of the vendors associated with the dues.
- (3) dues[]: It stores the amount of dues associated with each vendor.

ii) **Member Functions**

- (1) write_content(): This function generates the message to be delivered at the time of dues claerance and stores it in content variable of base class.

- d) **Transaction_notification: Notification** : This class is derived from the Notification class and it handles the notificaitons when a user of vendor adds a due.

i) **Data Members**

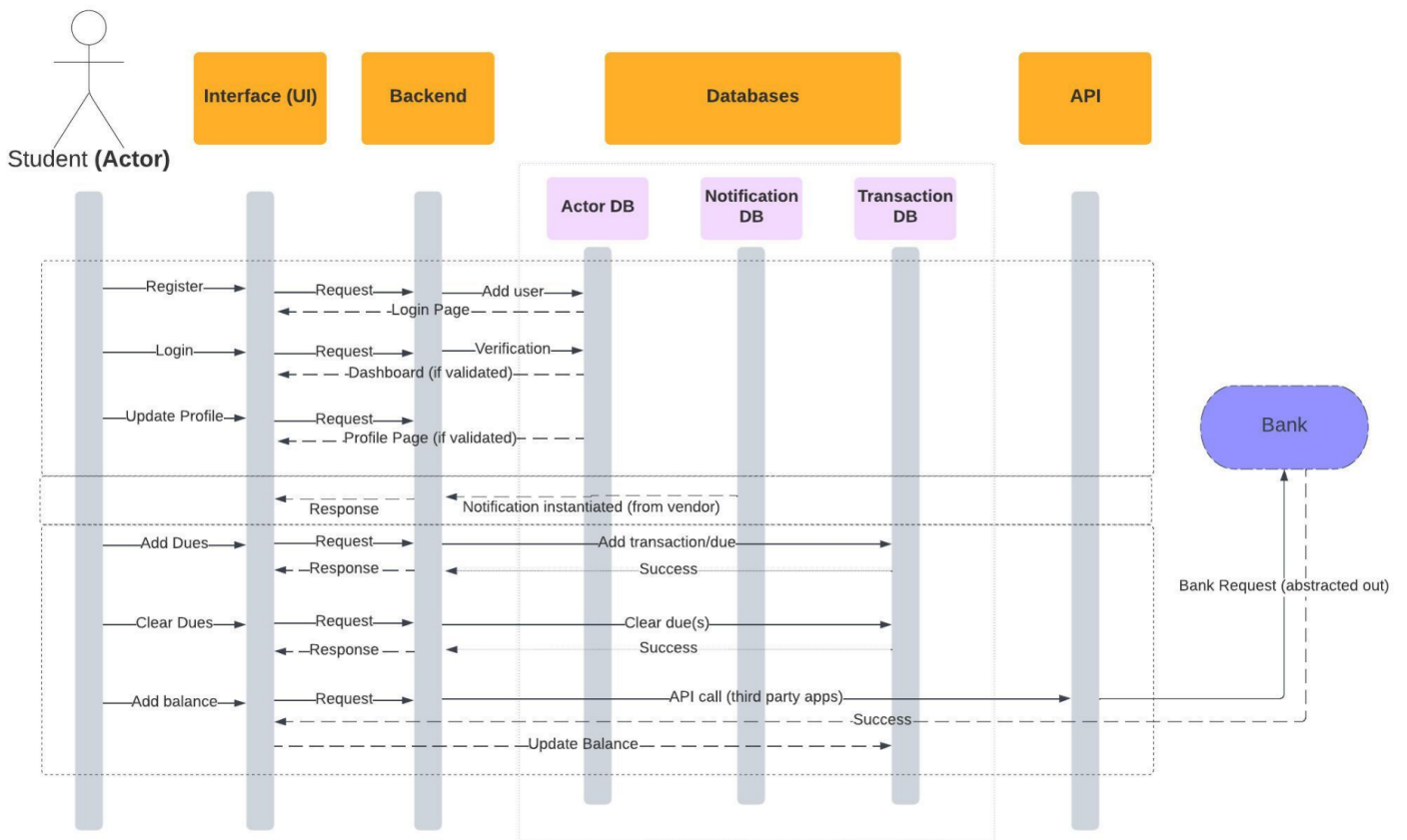
- (1) userID: It stroes the ID of the user associated with transaction.

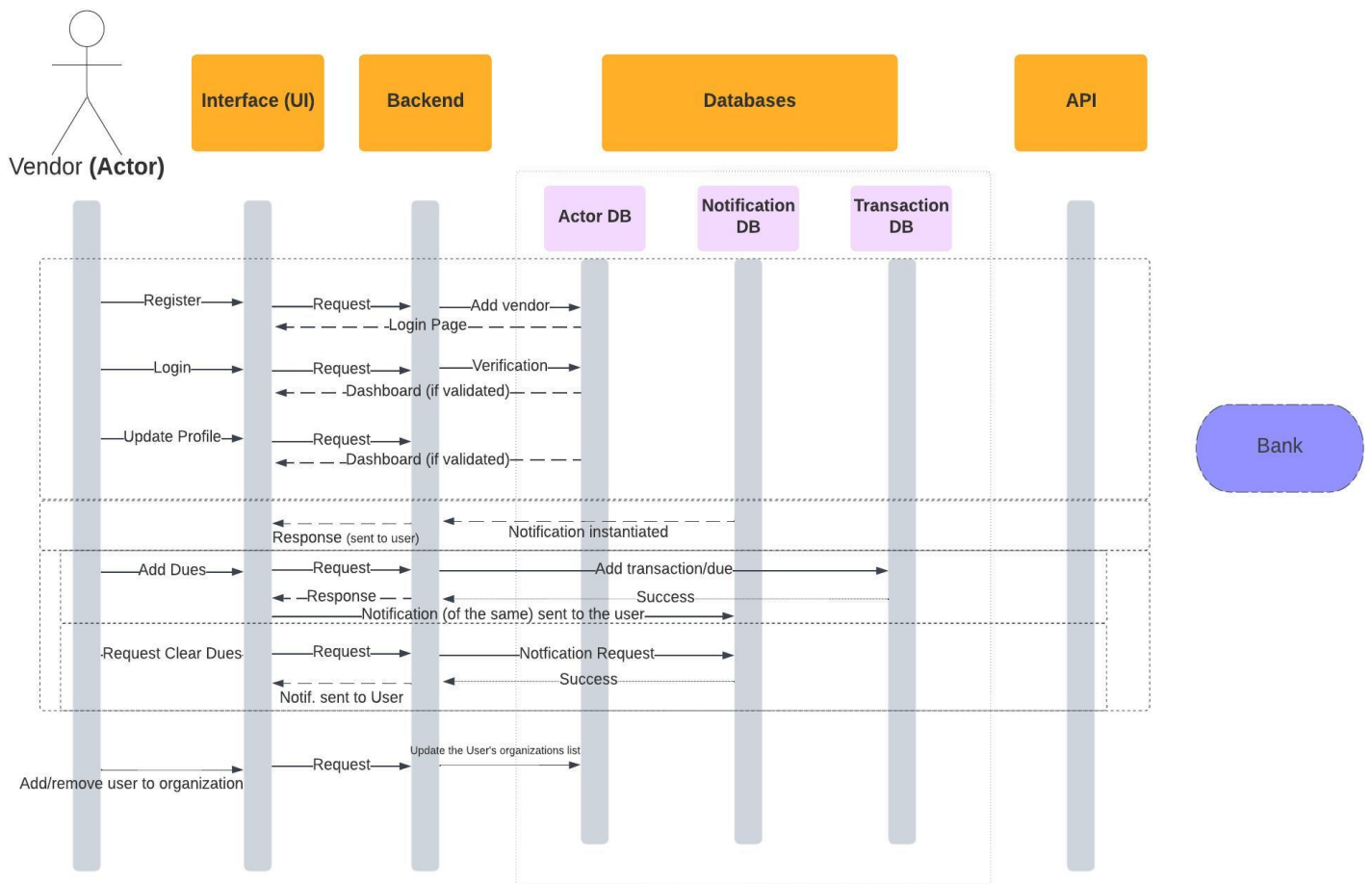
- (2) vendorID: It stores the ID of the vendor associated with the transaction
- (3) amount: It stores the amount of dues added during transaction
- (4) transactionID: It stores the same value as the “transactionID” variable of “Transaction” Class

ii) Member Functions

- (1) write_content(): This function generates the message to be delivered when the transaction is added.

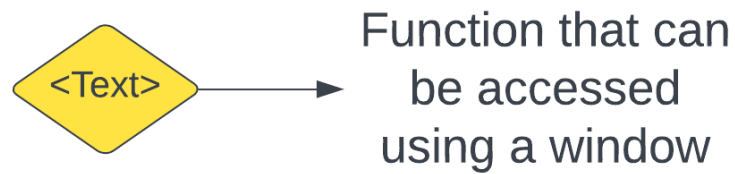
3.3 Sequence Diagrams



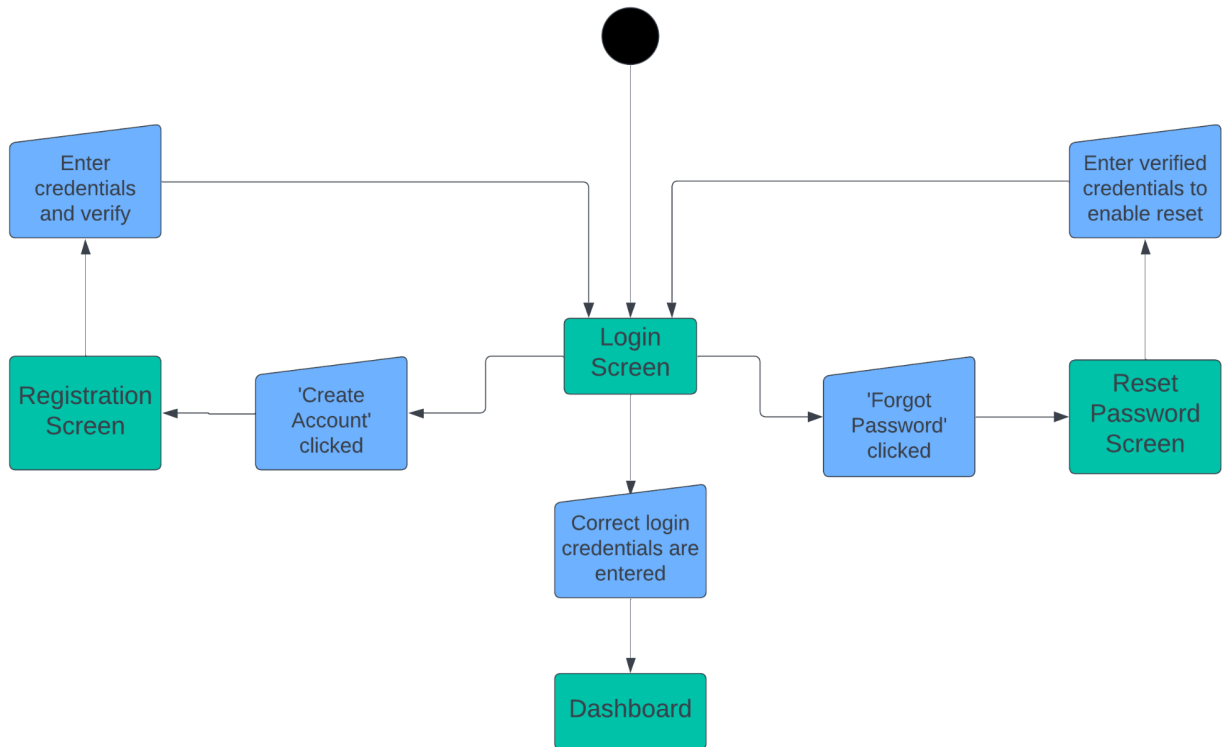


3.4 State Diagrams

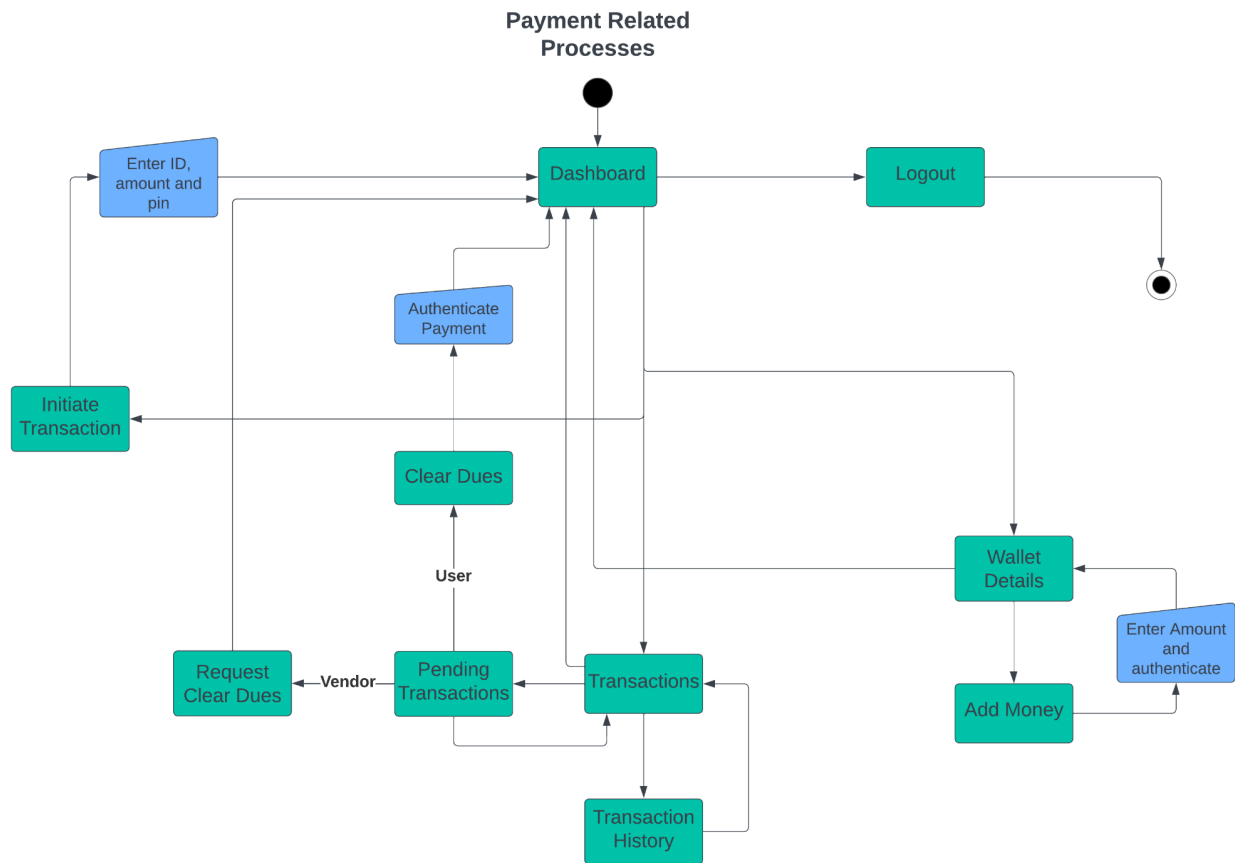
Key



Login Process



The state diagram for the login process. It facilitates the features to reset password, to create a new account and to login to access the app.



The state diagram for all payment related features on the app.

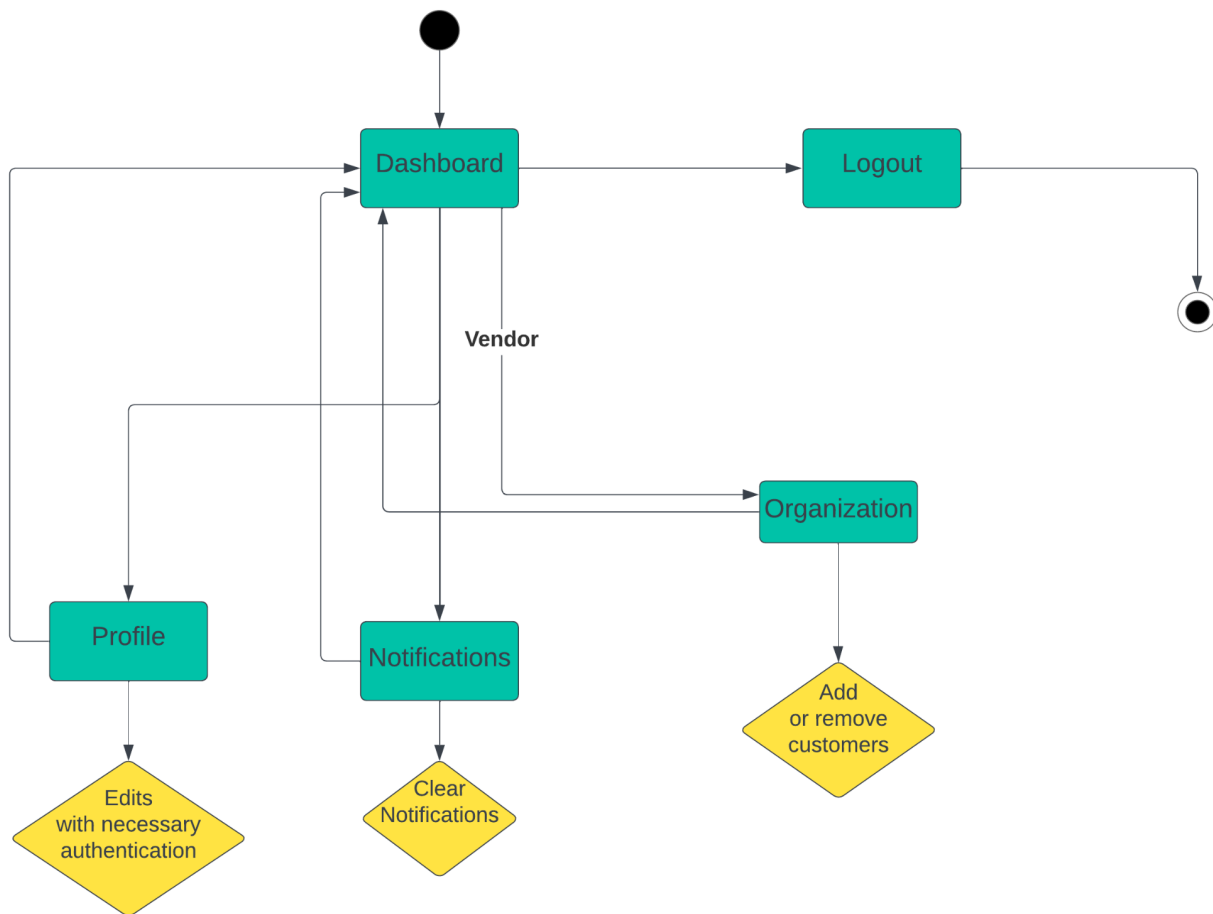
An example of a journey for a user can be:

Logged in and currently viewing the 'Dashboard'. Now the user wishes to clear dues. The user accesses the 'Transactions' tab and views the 'Pending Transactions'. They now tap 'Clear Dues', authenticate the payment and if successful, are redirected to the 'Dashboard'. Now, the user wishes to add money for the next time they clear dues. They tap on 'Wallet Details' and choose 'Add Money'. After the required input, if the process is successful, money is added to the wallet and the user is redirected to the 'Dashboard'. Now the user logs out.

An example of a journey for a vendor can be:

Logged in and currently viewing the 'Dashboard'. A particular user X has a huge amount of due. The vendor wishes to request that user to clear dues. They access the 'Transactions' tab and go to 'Pending Transactions' and then 'Request Clear Dues'. They can now select particular or all users and request to clear dues. The vendor selects the user X and requests clear dues. A notification is sent to user X and the vendor is redirected to the 'Dashboard'. Now they log out.

Other Features



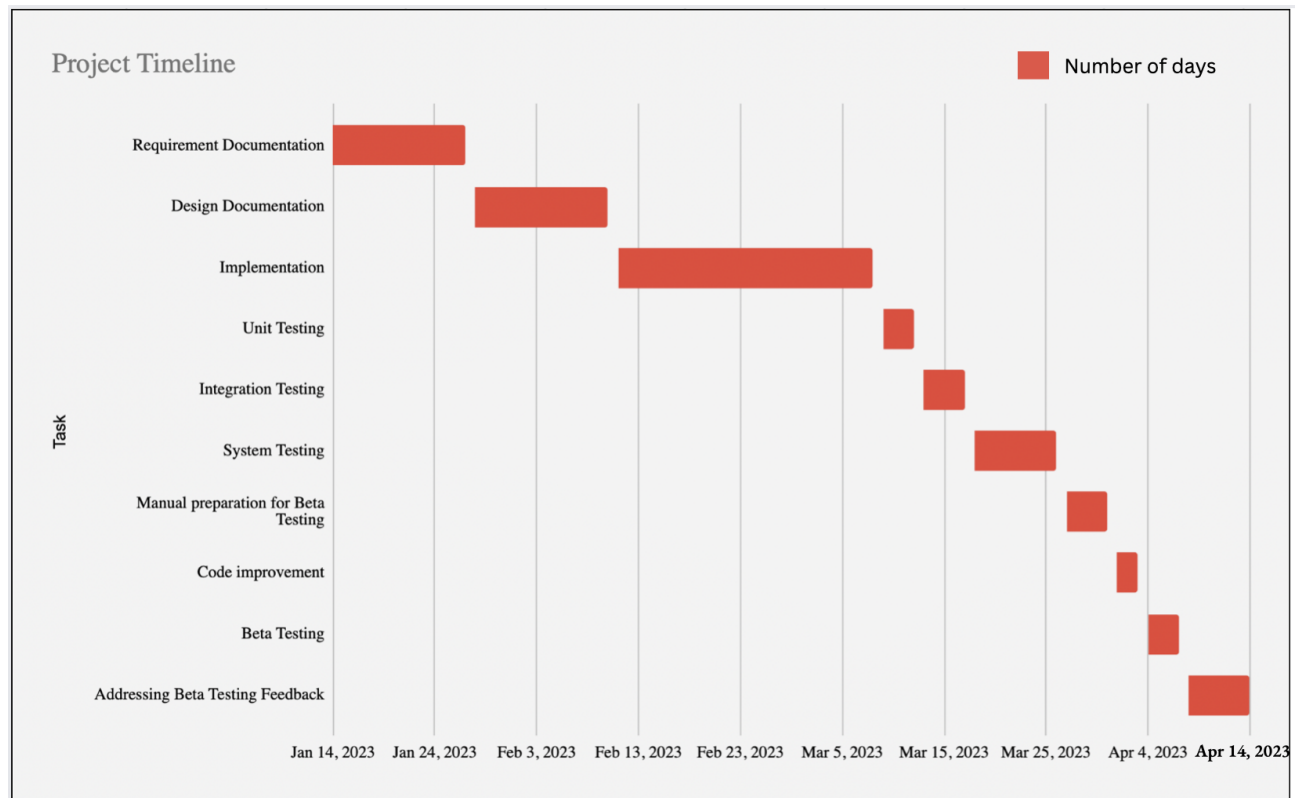
This includes other features that are accessible to users. Any user/vendor can access their profile and update selected fields (details to be decided during implementation).

A user/vendor can also view their notifications (sorted from most->least recent) and they can clear particular/all notifications.

A vendor can also view all the customers belonging to their organization (users that have had transactions with the vendor) and if a new user wishes to register, they can be added to the organization. If a vendor is unhappy with a particular customer, they can be removed from the organization.

4 Project Plan

<Provide a detailed plan for the implementation and testing of the software. Identify the major tasks and their dependencies. Capture the plan in a Gantt Chart. Also, for each task, identify the team member who will be responsible for completing the task.>



Task	Start Date	Number of Days To Complete
Requirement Documentation	14-Jan-2023	14
Design Documentation	28-Jan-2023	14
Implementation	11-Feb-2023	26
Unit Testing	09-Mar-2023	4
Integration Testing	13-Mar-2023	5
System Testing	18-Mar-2023	9
Manual preparation for Beta Testing	27-Mar-2023	5
Code improvement	01-Apr-2023	3
Beta Testing	04-Apr-2023	4
Addressing Beta Testing Feedback	08-Apr-2023	7

NAME	WORK
Aditya Bangar	Full-stack Implementation, Code Improvement, Beta Testing
Akshat Rajani	Frontend Implementation, Integration Testing, Beta Testing
Harsh Bihany	Full-stack Implementation, Integration Testing, Addressing Feedback
Kalika	Frontend Implementation, Manual for beta testing, Beta Testing
Monil Lodha	Backend Implementation, Integration Testing, Addressing Feedback
Pratham Sahu	Full-stack Implementation, System Testing, Code Improvement
Pulkit Gopalani	Backend Implementation, System Testing, Addressing Feedback
Ravija Chandel	Frontend Implementation , Unit Testing, Manual for beta testing, Beta Testing
Shantanu Kolte	Backend Implementation, Unit testing, Code Improvement
Siddhant Jakhotiya	Frontend-Implementation, Unit Testing, Code Improvement

5 Other Details

Some tools used:

1. <https://www.lucidchart.com/pages/>
2. <https://www.figma.com/>

Appendix A - Group Log

<Please include here all the minutes from your group meetings, your group activities, and any other relevant information that will assist in determining the effort put forth to produce this document>

- **4th February**
 - Harsh, Pratham and Siddhant met and discussed about class, state and sequence diagrams. Consequently, Siddhant made a state diagram for the entire app.
- **5th February**
 - Monil, Akshat and Ravija completed section 4 : project plan and 3.1 : use case diagrams.
 - Pratham completed 3.2: Class diagram 1.
 - Pulkit and Shantanu met to discuss architecture design
- **6th February**
 - The team had a meeting with the mentor.
 - As per the discussion of the meeting, certain changes were made in the state diagram by Siddhant.
- **7th February**
 - Pulkit and Shantanu met to discuss architecture design
- **8th February**
 - Harsh completed the sequence diagrams.
 - Pratham added notification classes in 3.2
 - Kalika discussed the basic designs of Human interface with Harsh and started working on it.
- **9th February**
 - Aditya completed the context model.
 - Team Meeting to discuss Work Distribution and Architecture Design.
 - Kalika completed the Human interface design(Section 1.2).
- **10th February**
 - Pratham and Akshat completed the description for the class diagrams.
 - Shantanu and Pulkit completed the architecture design (Section 2)