Design Document

for

CampusPay

Version 1.1

Prepared by

Group 2: **Group Name:** TheMissingSemiColon

Aditya Bangar	210069	adityavb21@iitk.ac.in
Akshat Rajani	210812	rajanias21@iitk.ac.in
Harsh Bihany	210406	harshb21@iitk.ac.in
Kalika	210482	kalika21@iitk.ac.in
Monil Lodha	210630	monill21@iitk.ac.in
Pratham Sahu	210755	spratham21@iitk.ac.in
Pulkit Gopalani	180564	gpulkit@iitk.ac.in
Ravija Chandel	210835	ravijac21@iitk.ac.in
Shantanu Kolte	210958	shantanu21@iitk.ac.in
Siddhant Jakhotiya	211030	siddhantj21@iitk.ac.in

Course: CS253

Mentor TA: Sumit Lahiri

Date: Apr 23, 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
1.0	Aditya Bangar Akshat Rajani Harsh Bihany Kalika Monil Lodha Pulkit Gopalani Ravija Chandel Shantanu Kolte Siddhant Jakhotiya	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.	10/02/2023
1.1	Shantanu Kolte Siddhant Jakhotiya	Edits based upon feedback from the TA	31/03/2023

1 Context Design

1.1 Context Model

1.2 Human Interface Design

1. Login Page

2. Registration Page

3. Customer Overview

4. Vendor Dashboard

5. All Dues

6. Profile

7. Transaction History

ReceiverID	Date	Status	TransactionID	Amount
9ZEVSBLD	23/04/2023	Pending	WFW02KW32L	50
YH01E55P	23/04/2023	Pending	9WESGVO4UG	35
YH01E55P	17/04/2023	Failed	O8QVCF9M0U	4000
YH01E55P	17/04/2023	Paid	L6PJP35VHU	123
9ZEVSBLD	17/04/2023	Paid	O3BKCAYZBS	12
YH01E55P	17/04/2023	Paid	UFHDWTFD94	123
9ZEVSBLD	17/04/2023	Cleared	PZ81MOOL6M	12
YH01E55P	17/04/2023	Cleared	6F6G0P2VEL	123
YH01E55P	17/04/2023	Paid	U5NEB4UXET	23
YH01E55P	17/04/2023	Paid	9A7YQRMN12	123
YH01E55P	17/04/2023	Cleared	MYLJLE8COU	23
ZCT51SPG	17/04/2023	Paid	DYSEMYTLA0	45
YH01E55P	17/04/2023	Paid	6F9LQFCYFO	123
58HRZQZ2	17/04/2023	Paid	R11E7WVT7R	76
ZCT51SPG	17/04/2023	Cleared	0YVLXGYU26	45
58HRZQZ2	17/04/2023	Cleared	19V88U36TH	76
YH01E55P	17/04/2023	Cleared	S79ZNM19W0	123
YH01E55P	17/04/2023	Paid	8ED4JF6OOD	123
YH01E55P	17/04/2023	Paid	VN3709ZQV0	1000

8. Initiating Transaction

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 SQLfor database management.

3 Object Oriented Design

3.1 Use Case Diagrams

Vendor adding the customer to his ledger (Implemented differently)	

Customer adding an expense as dues Customer making instant payment	
Customer settling his/her dues	

Vendor adding an expense as dues/ reminding for a pending due	
Customer adding money to wallet	

User deleting his account	

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 the 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 a 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 systems.

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 notifications at the time of addition of client to the server.

i) Data Members

(1) clientID: It stores 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 the content variable of base class.

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

i) Data Members

(1) userID: It stores 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 clearance and stores it in the content variable of base class.

d) Transaction_notification: Notification : This class is derived from the Notification class and it handles the notifications when a user or vendor adds a due.

i) Data Members

(1) userID: It stores the ID of the user associated with the 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

The state diagram for the login process. It facilitates the features to create a new account and to login (if the account already exists) to access the app.

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

<u>An example of a journey for a user can be:</u>

Logged in as a <u>customer</u> and currently viewing the ‘Dashboard’. To add dues with a vendor, the customer has two options: they can either click on the 'Make Payment' button and then select 'Add Dues', or they can navigate directly to the 'Add Dues' tab. Once there, the customer can enter the Receiver ID and the desired amount to complete the transaction.Transaction status will be updated accordingly in the ‘Transaction History’ tab.

<u>An example of a journey for a user can be:</u>

Logged in as a <u>customer</u> and currently viewing the ‘Dashboard’. To add money to their wallet, the customer can click on the 'Add Money to Wallet' button and enter the desired amount. After clicking the 'Submit' button, the amount will be added to the customer's wallet and will be reflected immediately reflected in their wallet balance

<u>An example of a journey for a user can be:</u>

Logged in as a <u>vendor</u> and currently viewing the ‘Dashboard’. If the vendor needs to verify whether a customer has submitted a pending or paid transaction after a failed transaction, they can navigate to the 'Transaction History' tab. Failed transactions are clearly marked in red and are listed alongside the customer involved. From there, the vendor can check whether the customer has added another transaction with a 'paid' or 'pending' status. This allows the vendor to easily keep track of transaction status and follow up with customers if necessary.

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 vendor can view all the customers associated through the application (users that have had transactions with the vendor) and if a new user wishes to register, they can be added simply by initiating their first transaction with the vendor.

<u>An example of a journey for a user can be:</u>
Logged in as a customer and currently viewing the ‘Dashboard’. The customer may choose to view all associated vendors and clear dues with a particular vendor. To accomplish this, the customer would navigate to the 'Vendors' tab, which displays a card-based layout of all currently associated vendors. To clear dues with a specific vendor, say 'V', the customer can click on the 'clear dues' button located on V's card. If the customer has sufficient balance in their wallet, this action will settle the dues with that vendor.

In the system, a user can access their profile which includes their username, userID, mobile number, email ID, and userType. The user has the ability to modify all the fields except for their userID and userType.

<u>An example of a journey for a user can be:</u>
Logged in as a vendor and currently viewing Dashboard. The vendor might want to update their mobile number and Email ID. The vendor could navigate to the 'Profile' tab and select the 'Edit Profile' option to modify their profile details. They could then change their mobile number and Email ID, and save the changes by clicking on the 'Save' button. After successfully updating their profile, the vendor may choose to log out by clicking on the 'Logout' button located in the bottom left corner of the page. This would end their user session and log them out of their account.

With just a click of the mouse, the user has the ability to seamlessly switch between light and dark themes. Additionally, all notifications can be accessed in one convenient location via the ‘notifications’ tab. This allows the user to stay up-to-date with any important notifications.

<u>An example of a journey for a user can be:</u>
Logged in as Customer and currently viewing ‘Dashboard’. The customer may need to prioritize which vendor to pay first due to insufficient wallet balance. The user can check which vendors have requested to clear dues recently by navigating to the 'Notifications' tab. Once there, the user can see all notifications, including requests to clear dues. After identifying the specific vendor, the user can navigate to the 'Vendors' tab and clear dues with that vendor by clicking on ‘clear dues’ button. This allows the user to efficiently manage their payments.

4 Project Plan

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

- **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).
- **22nd April**
  - Akshat added state diagrams highlighting the features of the app.
  - Pratham updated the class diagram.
  - Monil updated the use-case models.