Software Requirements Specification

for

CampusPay

Version 1.2

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	ravjiac21@iitk.ac.in
Shantanu Kolte	210958	shantanu21@iitk.ac.in

Course: CS253

Mentor TA: Sumit Lahiri

Date: Jan 27, 2023

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.0	Aditya Bangar Akshat Rajani Harsh Bihany Pulkit Gopalani	Initial Version of the document. A brief outline of Section 1.1 was written to get an overview of the scope.	Jan 22, 2023
1.1	Aditya Bangar Akshat Rajani Harsh Bihany Kalika Monil Lodha Pratham Sahu Pulkit Gopalani Ravija Chandel Shantanu Kolte Siddhant Jakhotiya	A change was proposed regarding the mechanism to facilitate transactions- instead of linking directly with the bank, an in-app wallet was suggested. All sections were filled in based on functionality discussed so far.	Jan 27, 2023
1.2	Aditya Bangar Akshat Rajani Monil Lodha Pratham Sahu Ravija Chandel Siddhant Jakhotiya	Based on the discussion with the TA mentor, a few edits were made before the release of the final document.	Jan 27, 2023

1 Introduction

1.1 Product Scope

The project aims to solve a common problem faced by the campus community, especially students, and those who make payments at the end of the month to different vendors. Almost all students residing on campus avail various facilities like Hall canteen, Hall mess, laundry etc. However, maintaining a track of payments and paying for these facilities becomes cumbersome due to "accounts" maintained everywhere, and clearing dues on the last day of semester is often a task in itself. Also, we are left handicapped on various occasions when bank servers are down or when we hit the transaction limit.

To solve this, we propose a unified portal for the campus where students (customers) and vendors can register, add dues, clear existing dues and maintain an account of transactions. For example, instead of maintaining paper registers, the canteen owner can set up his account and add any customers to his "organization". Any transaction recorded on paper could be recorded digitally on the portal, and the balance would add up (until the customer clears the dues). The customer can either use the wallet (maintain balance upfront, similar to PayTM wallet), or pay directly from a registered UPI account to clear dues. The portal will also facilitate peer to peer transactions (student to student). The utility of this feature is for situations wherein multiple friends purchase certain items together and only one student pays directly to the vendor. This saves the hassle of maintaining a record of the payments for the students. The software will also enable easier tracking of monthly expenses, dividing them up into categories like Food, Utilities etc.

1.2 Intended Audience and Document Overview

The intended users are the whole campus community, in particular for hostellers and vendors having a business on campus. The document can also be referred by the developer team to review features and functionality during the entire development process. It will also be useful to any team that may want to work upon any sort of changes, up-gradation of the software.

The remaining subheadings of Section-1 provide relevant terms used throughout the document. Familiarization with these terms would be essential to understand the document.

Section-2 provides an overview of the software, its functionality, constraints and assumptions. This would provide an idea regarding the actors and major components involved in the usage of the software, the technologies used in the development.

Section-3 explores the detailed requirements of the software. There are a few pictorial representations of the interface of the software (tentative) and different interactions and planned features.

Section-4 highlights the non-functional requirements of this software. Since this software deals with finances, it must have robust security. This, and other performance requirements are detailed in this section.

Users can understand the features of the software by going through Sections 2.1 and 2.2.
For the team developing/improving the software, it is imperative to go through all the sections in the order that they have been written in this document. Testers may read Section-2 (2.2 and 2.3) and all sections from Sections 3.2 to 4.1 to understand the functionalities that they need to test.

1.3 Definitions, Acronyms and Abbreviations

For the purpose of this document, these terms mean the following:

- Super account: An account that can view and edit the details of all accounts.
- User: Any student or faculty or administration who wishes to buy these commodities or services from the vendors and who has access to an IITK Email-ID.
- Vendor: Any person/organization registered with the institute and authorized to sell commodities or provide services (for eg. Canteen owners, washerman/washerwoman, shops providing stationery etc.).

Standard web-app based terms have been used like frontend, backend, framework, WebScripting, database.

Abbreviations used are:

- API : Application Programming Interface
- CC : Computer Centre
- CSS : Cascading Style Sheet
- HTML : Hyper Text Markup Language
- IITK : Indian Institute of Technology, Kanpur
- IMAP : Internet Message Access Protocol
- OTP : One Time Password
- SMTP : Simple Mail Transfer Protocol
- SQL : Structured Query Language
- SRS : Software Requirement Specification
- UPI : Unified Payments Interface

1.4 Document Conventions

1.4.1 Formatting Conventions

The font used throughout this document is Arial.

- Titles have font size 18 and are in bold.
- Section headings have font size 14 and are in bold.
- Subsection headings have font size 12 and are in bold.
- Body text has font size 11.

1.4.2 Naming Conventions

All headings have been named as per the standard template of SRS of IEEE. Certain sections have been divided into subsections/bulleted lists for easier readability.

1.5 References and Acknowledgments

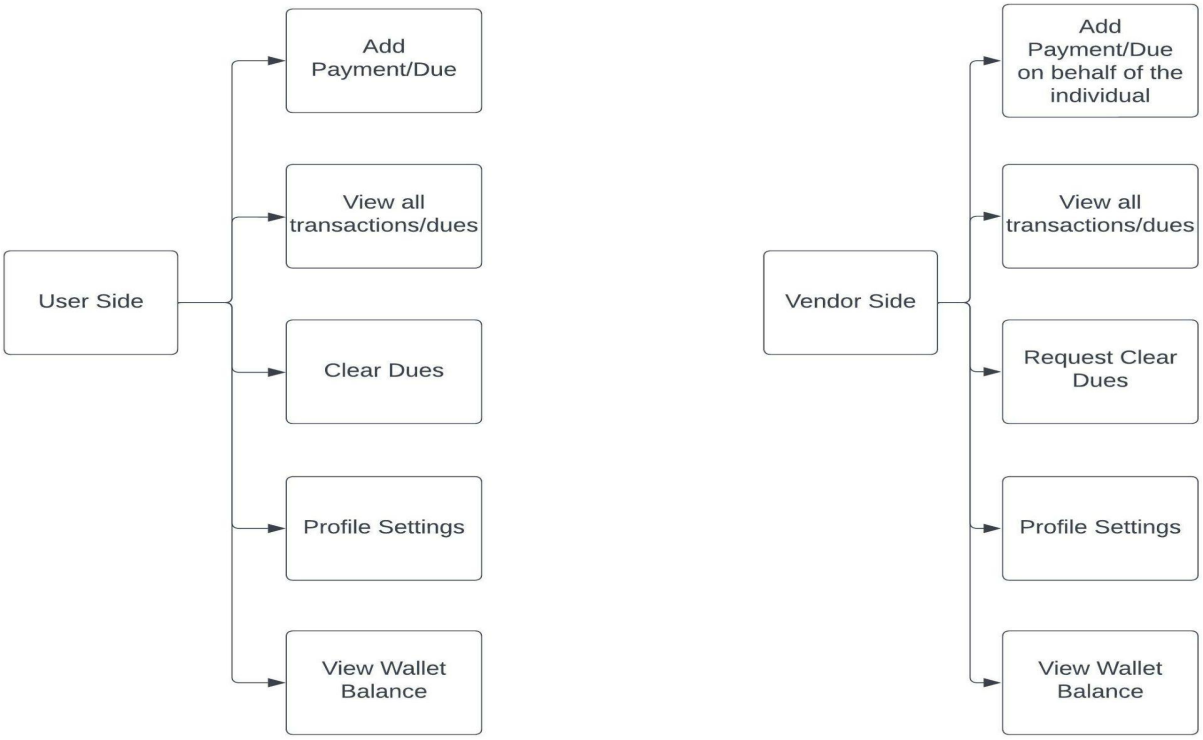
https://igotanoffer.com/blogs/tech/latency-throughput-availability-system-design-interview -
Referred for latency and availability inputs of similar systems

https://lucid.app/documents#/dashboard -
To create use case UML diagram
To create wireframes

2 Overall Description

2.1 Product Overview

The product was conceptualized for making expense tracking and payments on campus simpler and convenient. This is a self-contained product, to address issues faced by the campus community regarding payments, hall dues etc. The idea is to eliminate the need for multiple payment platforms and provide a single platform for both the vendors and customers. Our platform keeps track of dues pending at various vendors (e.g. hall canteen, mess, hall general store) and allows the user to clear these dues at any time, hassle free. In addition, the platform can be used to make instant payments to any registered user. All of this is achieved through wallet functionality, deployed by other apps (e.g. PayTM), but with campus specific functions as above.



2.2 Product Functionality

[Please note: The following functions have been discussed in more detail in Section 3.2 later.]

- User and vendor registration (profile management)
- Wallet (deposit / withdraw money), pay immediately to anyone on the portal through wallet through a unique Wallet ID
- Maintain accounts with registered vendors, clear dues anytime
- Transaction Verification – report suspicious transactions
- Default prevention - impose limit on dues going above wallet balance
- Hall specific No-dues clubbed together for convenience (incl Mess, Electricity etc.)

- Alert user about pending dues, mandate payment at end of semester; auto-pay facility to avoid delays in payment
- Vendor can manage registered users, accept / decline user registration request and impose limit on amount of dues pending
- Vendor can add dues for a user based on their unique ID (subject to transaction verification as above)
- In case the vendor is unable to, users can add dues (which the vendor can verify)
- User friendly interface which displays wallet balance, pending dues, recent transactions, accounts with vendors

2.3 Design and Implementation Constraints

Specific technologies used: Django as a backend framework, ReactJS as a frontend framework. Language requirements: Python, JavaScript and SQL. WebScripting languages like HTML, CSS. Security considerations: in-built form security provided by the django framework. Databases to be used: SQL database (like PostgreSQL)

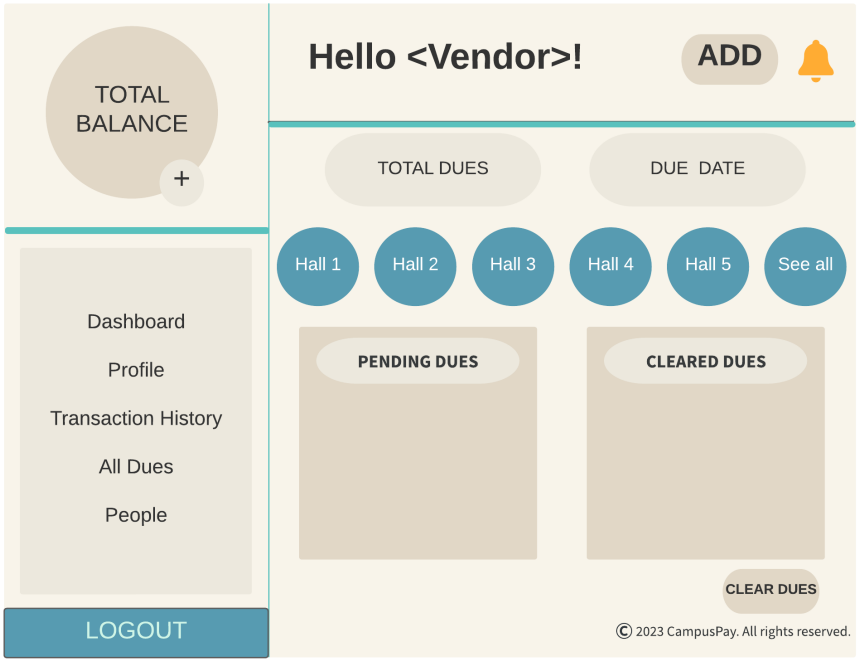
2.4 Assumptions and Dependencies

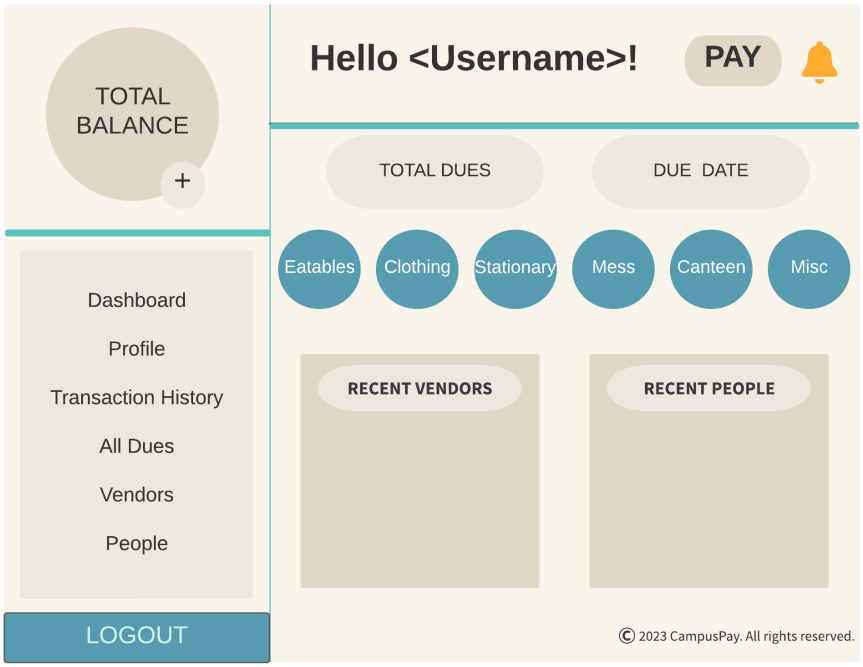
- We assume that the database for student and authorised vendors will be provided by the authorities (eg. CC) to prevent malicious use of the softwares by external vendors and non-students.
- We assume that a third-party access will be given in the process of clearing dues to make it a hassle-free process which the web-app aims to achieve.
- We assume that the vendor and the students (clients) will have access to smartphones and an internet connection to access the web-application.
- We assume that the vendors and students will be provided a one time training to operate the web-application. This can be made available in the form of a tutorial.
- The web browser being used must be compatible with the latest versions of ReactJS, Django and Postgres.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces





The user interface of the software will be web-based and user-friendly. Allowing users to access the software via any modern web browser like Google Chrome, Mozilla Firefox, Microsoft Edge, etc. Users have to log in/create an account via their registered mobile number and a password. They will be provided a unique 4-digit alpha numeric ID just after login .After that they can see the homepage which will be different for vendors and users.

Vendors-

- The homepage will mainly display the total amount of dues of the customers, the due date to pay the dues.
- It will display top 5 halls with maximum combined dues of its residents and other halls can be viewed by clicking see all options.
- In the left panel there will be options like Dashboard, Profile, Transaction history, All dues and people on the app.
- There will also be an indicator to display the total balance available in the bank.
- A list of pending and cleared dues will be visible.

Users-

- The homepage will mainly display the total amount of dues of the customers, the due date to pay the dues.
- It will display the total amount spent by the customer in all categories.
- In the left panel there will be options like Dashboard, Profile, Transaction history, All dues , vendors and people on the app.
- There will also be an indicator to display the total balance available in the bank.
- A list of money transaction history will be visible to both people and the vendors.

3.1.2 Hardware Interfaces

Since the application must run over the internet, all the hardware shall be required to connect to the internet, which will be the hardware interface for the system. As for e.g. Wi-Fi, Ethernet Cross-Cable, etc. Any smartphone or PC with proper internet connection will serve the purpose.

3.1.3 Software Interfaces

Any modern updated browser. In our web application the various buttons at the frontend connect with the backend via api calls, for example, the registration page does a POST request to the backend which updates entries in the database. The Dashboard and other pages on loading do a GET request to the server which serves the pages to be shown to the user. The pay button does an UPDATE request on the backend which changes the amount of money in the wallets of the users participating in the transaction.

3.2 Functional Requirements

3.2.1 Feature #1 [User and vendor registration]

The first registration for anyone would be as a user of the platform. We plan to collect basic information in this registration, for example, Name, Mobile Number, Email ID, Relevant Campus Information (Hall of Residence associated with, etc). In case the user wishes to register as a vendor, there would be an option to do so, subject to subsequent physical verification.

3.2.2 Feature #2 [Wallet facility]

Every user is provided a Wallet (similar to PayTM), which can be used for transactions on the platform. In the current project, we have abstracted out the wallet recharge / withdrawal process since that would require actual banking transactions. Every user will be assigned a unique 4-character alphanumeric code upon registration, which can be used for instant payments (eg. payments can be made directly by entering this 4 digit code and payable amount).

3.2.3 Feature #3 [Dues management]

Each user can register with a vendor on the platform, to start an account at their establishment. Once the vendor approves this registration, pending dues will be displayed on both ends. A user can add / remove vendors through the platform (removal allowed only if there are no pending dues). Dues can be updated both by the vendor as well as the user themselves.

3.2.4 Feature #4 [Instant Payment]

In addition to accounts in Feature #3, users can also pay other users (including vendors) instantly using their wallet. This is similar to the facility provided in wallets like PayTM.

3.2.5 Feature #5 [Transaction verification and reporting]

In case of a suspicious dues update by a vendor, users have the option to report the transaction for further review. It would also be possible to enable a real-time approval facility, in which such a transaction will be completed only when approved by the user.

3.2.6 Feature #6 [Hall specific dues management]

Various dues corresponding to the hall of residence of a user, including Mess, Electricity, Canteen, General store will be clubbed together for convenience. The No-Dues certificate required at the end of each semester will be generated automatically and sent to the Hall Office once these dues are cleared by the user.

3.2.7 Feature #7 [Pending Dues Alert, Autopay]

Users will be alerted about their pending dues on the platform, at time intervals specified by the user themselves. In addition, admin like the Hall office can mandate payments at times like semester-end, or before mid-semester recess for canteens. Users can also enable an autopay feature through which dues will be cleared automatically at given time intervals, provided the wallet has sufficient balance.

3.2.8 Feature #8 [Default prevention]

To prevent cases of default when the user does not pay their dues by the stipulated deadline, we plan to impose a restriction on the dues possible to be accumulated by a user. There will be a minimum balance required in every user's wallet in case they wish to add a due for some vendor. This minimum balance will be relaxable till a fixed grace amount (e.g. Rs. 100). This means that, for a current wallet balance of Rs. X, one can add dues upto Rs. X+200, after which the platform will not allow any more dues to be added. W.r.t. feature #7, if the user does not pay their dues by the admin specified deadline, the pending amount will be deducted automatically from their wallet. Remaining dues (< Rs. 100) will be added to the institute pending dues for the user.

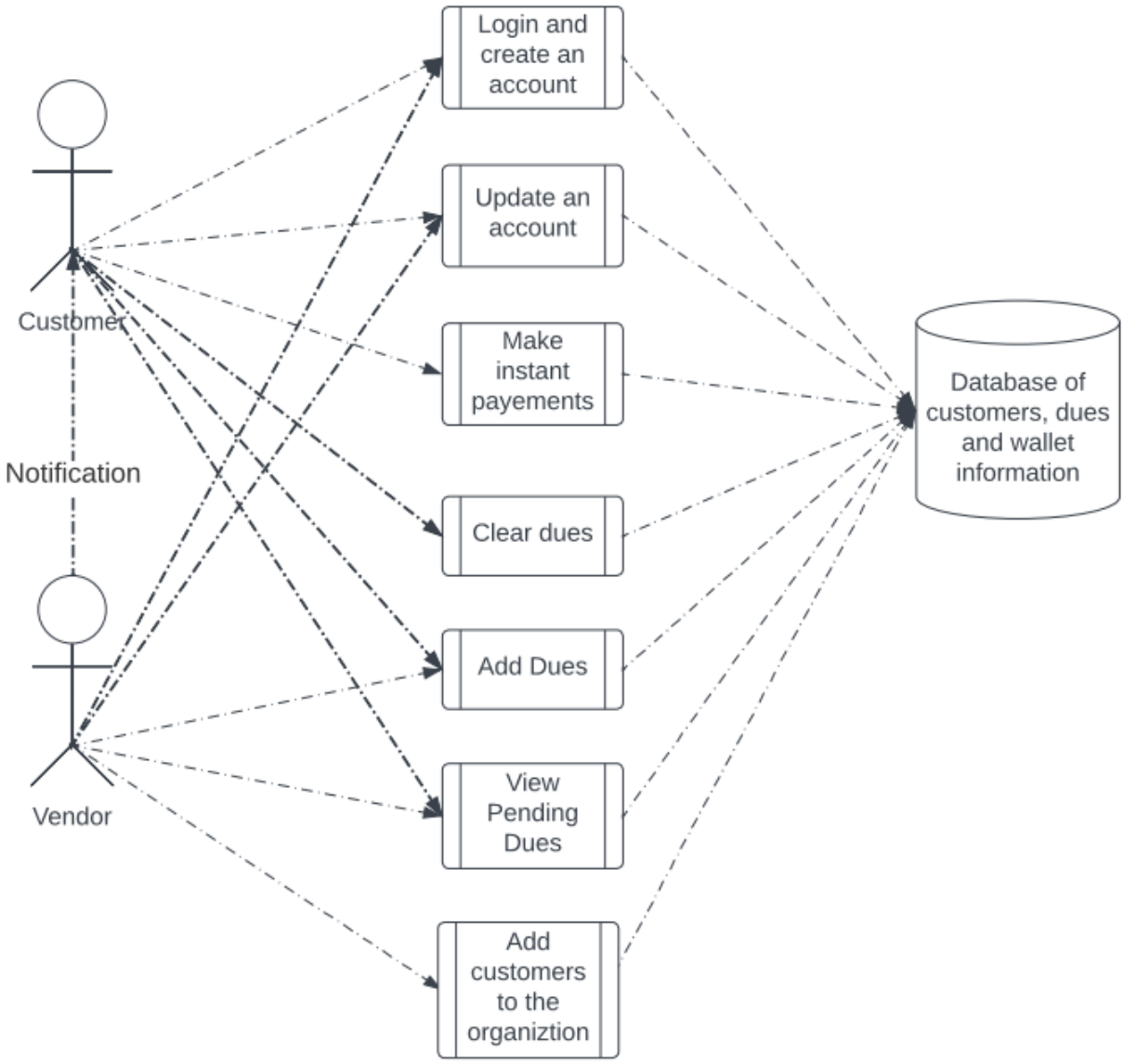
3.2.9 Feature #9 [Vendor privileges]

The vendor can control whom to add as an account user for their establishment. The vendor can decline user registration requests, and set an upper limit for pending dues, after which no transactions would be allowed. That is the user would have to clear their existing dues to resume using the account facility at that establishment.

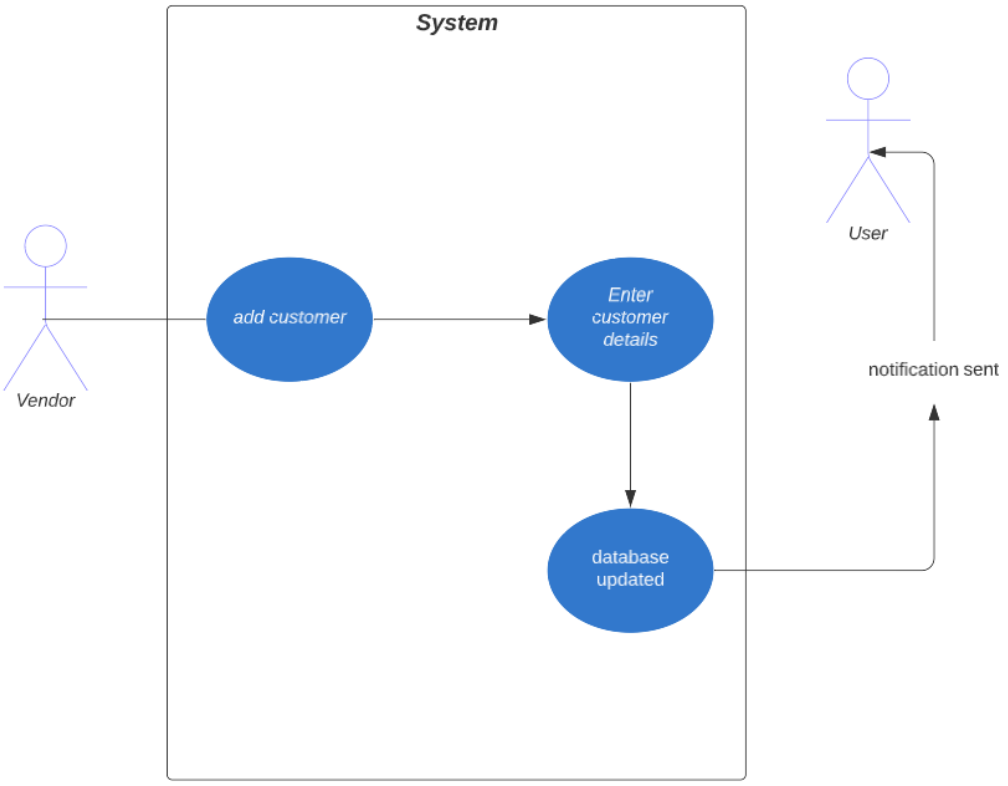
3.2.10 Feature #10 [User Interface]

The User Interface (UI) will display Wallet balance and pending dues on all pages the user visits, tentatively on the left hand side. On the landing page, the most recent transactions will be displayed to the user. Further, on clicking one of the transactions (which includes vendor details), the user will be shown the most recent transactions corresponding to the vendor they clicked on. There will also be options to add or remove vendors, deposit or withdraw money from the wallet, and instant payments.

3.3 Use Case Model



3.3.1 U1 : Vendor adding the user to his ledger



Purpose - Vendor has to add the user(one time operation) to his list with whom he wishes to maintain an account.

Requirements Traceability – Feature #9 (Vendor Privileges)

Priority - High

Preconditions - NA

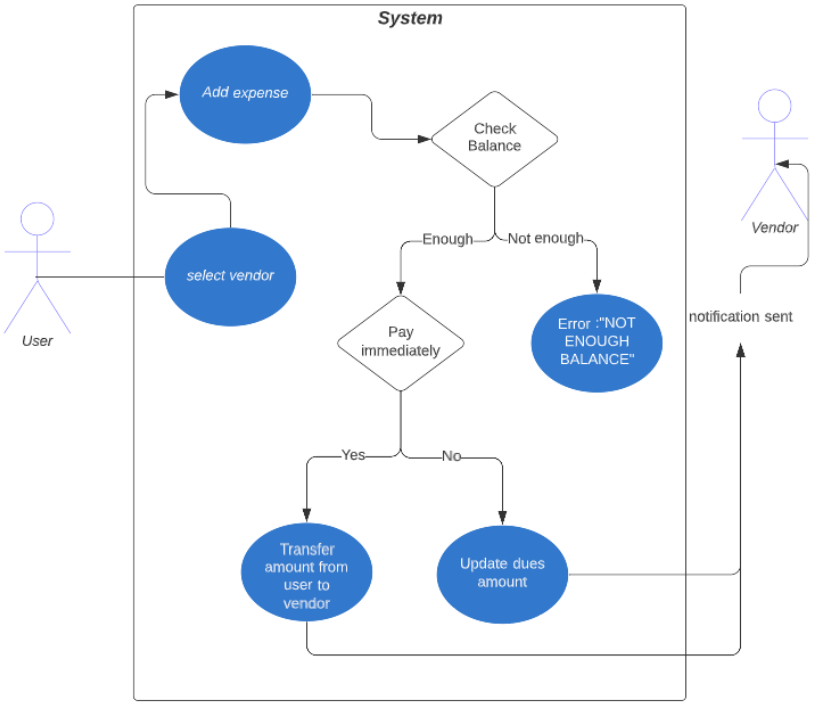
Post conditions - The user will be able to maintain dues with the vendor after successful registration.

Actors – Vendor, User

Exceptions - NA

Includes (other use case IDs)
Notes/Issues -

3.3.2 U2: User adding an expense as dues



Purpose - Users can add an expense anytime he/she buys commodities from a vendor, without paying them instantaneously. Its main objective is to decrease redundancy of multiple payments to the same vendor within a specified period.

Requirements Traceability – Feature #3 (Dues Management)

Priority - High

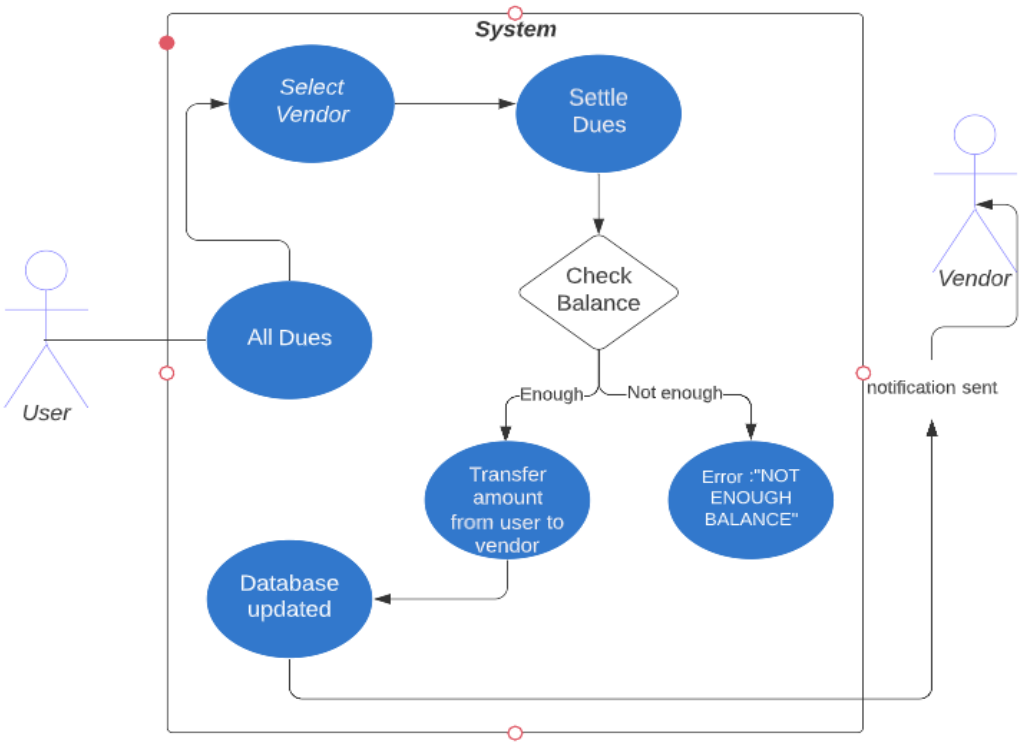
Preconditions - 1. The user must be registered with the vendor beforehand.
2. User must have sufficient wallet balance

Post conditions - The total dues will be updated in the databases and will be displayed to both user and vendor.

Actors – User

Exceptions -NA
Includes- NA
Notes/Issues - NA

3.3.3 U3: User settling his/her dues



Purpose - A user can settle his existing dues with a vendor as per his own convenience.

Requirements Traceability - Feature #2 (Wallet Facility), #3 (Due Management), #7 (Pending Dues Alert, Autopay)

Priority - High

Preconditions - The dues category must not be empty and the user should have enough wallet balance

Post conditions - The dues of that particular vendor will be settled and money would be transferred from the user's account to the vendor's wallet and the database will be updated.

Actors – User

Exceptions - NA
Includes-NA
Notes/Issues - NA

3.3.4 U4: User making instant payment

Refer diagram of U2

Purpose - Apart from adding dues for an expense (as in U1), a user can also pay instantly using his/her wallet.

Requirements Traceability – Feature #2, #4

Priority - High

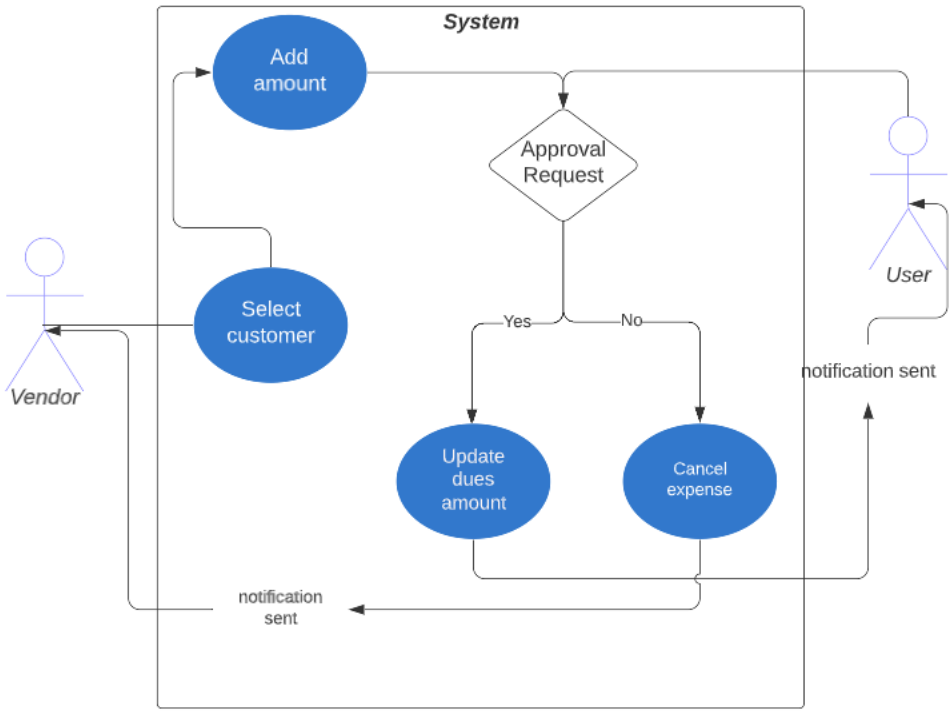
Preconditions - Sufficient wallet balance is required as per Feature #8

Post conditions - The wallet amount of both user and vendor will be updated in the databases.

Actors – User

Exceptions - NA
Includes- NA
Notes/Issues -NA

3.3.5 U5: Vendor adding an expense as dues/ reminding for a pending due



Purpose : Vendor can add an expense anytime a user buys an item. He can remind the user if he hasn't cleared the dues for a long time or if he is in need of urgent money

Requirements Traceability – Feature #3, #7

Priority - High

Scenario - The vendor adds the expense and immediately an approve notification goes to the user where he verifies the amount and approves.

Post conditions - The expense is updated in the database with the corresponding user and database and the corresponding total dues are also updated.
The user will receive a notification if the vendor has sent a reminder.

Actors – User,Vendor

Exceptions - User may disapprove of the expense in which he also has an option to report the vendor.

Includes -NA
Notes/Issues - The remainder notification -NA

3.3.6 U6: User adding money to wallet (abstracted out)

Purpose - User can recharge the wallet by any amount using bank account

Requirements Traceability- Feature #2
Priority - Low

Preconditions - User must be signed in to the system, have sufficient balance in bank account.

Post conditions - The updated amount will be stored in the database and displayed to the user under the wallet section.

Actors – User, UPI, bank account.

Exceptions - Payment may be declined due to server issues from the bank side or if the transaction limit has been exceeded.

Includes NA
Notes/Issues - Due to resource constraints and security concerns, we'll not link bank accounts to the wallet, for the scope of the project.

4 Other Non-functional Requirements

4.1 Performance Requirements

- The system should be able to respond to user requests in a timely fashion within 5s to ensure consistent operations.
- The system should be able to display error messages and save session settings within 5s.
- The system should update the user logs and interface after interaction within 5s to ensure no lag in the operations.
- The system should be able to retrieve information from the databases within 5s.
- The system should update the used databases within 5s to ensure smooth operations.
- In case of crashing, the system should have a recovery time within 10s.
- The system should have high availability and minimal downtime.
- The system should be able to cater to a huge number of concurrent users and transactions without crashing.
- The system should have proper storage to handle the data of all the students and vendors of IIT Kanpur.
- The system should have high availability and minimal downtime.
- The system should comply with relevant financial regulations and standards.
- The technicalities of implementation of the above stated requirements will be mentioned in the design document.

4.2 Safety and Security Requirements

- Registration for customers will take place using their institute email address or any other institute approved identification code.
- Registration for vendors can only occur after approval from the institute.
- Authentication will be required by any user to log in to their account after the initial registration.
- Administrators can only perform administrative tasks on pages they are privileged to access. Customers and vendors will not be allowed to access the administrative pages.
- The database should not be accessed by anyone except the superuser to allow addition of money to the customers wallet (also this functionality exists for demo purposes only, if the app goes into production no one can change money in anyone's wallet).

4.3 Software Quality Attributes

4.3.1 Availability and Scalability

This application will be made to clients with a IITK email ID which will be used for verification. The vendors will be verified by the institute before an account is made.
The app will be scalable to include all students, staff and vendors currently residing/operating on campus and the database will be updated each year to include new students and remove outgoing students.

4.3.2 Maintainability

The application should use continuous integration so that features and bug fixes can be deployed quickly without operational downtime. The website will be available for use 99.9% of the time and bug fixes will be deployed during times of least use.

4.3.3 Reliability

Based on the limited number of users (~15000 users) and database specifications, the web application will be able to perform with stability.

4.3.4 Latency

The latency of React and Django when there are ~3000 users will depend on several factors such as the complexity of the application, the resources available on the server, and the network infrastructure. It is not possible to provide a definitive answer without more information about the specific application and its deployment environment. However, in general, it is safe to say that the latency will increase as the number of users increases and the application becomes more heavily loaded. It is important to note that both React and Django are highly customizable, and the performance of an application built with them can be optimized through various techniques such as caching, load balancing, and database optimization.
Based on similar systems and load input, the web application should take less than 5000 ms to load under standard operating conditions and medium load (~3000 users).

5 Other Requirements

Some third party vendor requirements are as follows –

- Request the IITK CC for access to their SMTP and IMAP servers for Email-ID verification and OTP delivery.
- We do not work with real money and for simulation will add money to each user's wallet using a super account. However, if time permits, we will make use of a publicly available API (e.g. RazorPay) to facilitate transfer of money from a 'real' bank account to the wallets.
- We also assume that the linking of these dues to the Students' Pingala Account for their 'No Dues' certificate is abstracted out. However, if time permits, we will use a mock API of Pingala to outline the process of updating no-dues on Pingala.

Appendix A – Data Dictionary

Sr. No.	State Variable	Shown in Use Case#	Actor	Possible State#1	Possible State#2	Input	Output
1	Check Balance	3.3.2	User	Enough	Not enough	When user adds expense	Prompt to ask the user to pay immediately or an error message saying “not enough balance”.
2	Check Balance	3.3.3	User	Enough	Not enough	When user chooses to settle dues	Either transfer the due amount to the vendor or display an error message saying “not enough balance”.
3	Pay Immediately	3.3.2	User	Yes	No	When user adds expense	Either transfer amount from user to vendor or update the dues list.
4	Approve Request	3.3.5	Vendor, User	Yes	No	When vendor adds expense	If the user chooses yes then the pending dues are updated. If not, expense is cancelled.

Some other processes have been described without the use of state variables in this version of the document. The details for those will be included in the design document.

Appendix B - Group Log

- 7th January-
  - The whole team met for the first time and finalized the idea of CampusPay.
  - The team brainstormed on the idea and formulated the big picture for the application and its working.
- 13th January-
  - Pratham, Aditya and Harsh met to discuss the framework stack to be used in developing the web application. Django, ReactJS and Postgres were decided as the base stack being used.
- 18th January-
  - Entire group met with the TA for the first time to get started on the SRS documentation and get approval for the frameworks that were decided.
  - The initial idea that was finalized was to have an app that can maintain records and facilitate transactions *directly from the bank* on a month-to-month basis (or as and when required by the user). Some discussion was made about the security measures required for this implementation.
- 21st January-
  - Aditya, Kalika and Harsh discussed the overall user interface and schemas for the web application.
- 22nd January-
  - Monil, Harsh,Pulkit and Aditya met to discuss the functionality, features and a general idea of wireframes. The idea to facilitate transactions from the bank was discarded and an in-app wallet based approach was finalized.
  - The other group members were briefed about the same.
- 23rd January -
  - Akshat and Monil met to discuss developments about use case models.
  - Siddhant wrote 1.1 based on the latest discussion of the group.
  - Kalika made the design of the wireframes.
  - Aditya wrote 2.1 based on discussion of the group.
- 24th January -
  - The entire group met online with the TA to discuss the non-functional requirements.
  - Siddhant finished 1.2 while referring to other sections that had been completed by others.
  - Kalika completed the 3.3 section with the wireframes, its descriptions , software and hardware interfaces.
  - Pratham and Aditya discussed the assumptions and software third-party requirements required to be incorporated in the application
  - Pratham completed 2.4 and 4.3 based on the group meeting.
  - Shantanu completed section 4.2 based on the group meeting
  - Ravija completed section 4.1 based on the group meeting.

- 27th January
    - Harsh added the general use case diagram for section 3.3.
    - Akshat andMonil completed the use case description in section 3.3.1 to 3.3.6. Also added the UML diagrams for each use case in section 3.3.
    - Team meet with TA Mentor.
    - Aditya and Kalika discussed the changes in the software interface and updated it.
    - Pratham updated latency parameters of the application based on inputs from the team and TA.
    - Siddhant filled in the 'Revisions' table and completed Sections 1.3, 1.4, 5 and Appendix A. Also proofread the entire document and suggested minor corrections to the team for the same.