SW Engineering CSC648-848-01 Summer 2025

Limóney (Financial Budgeting Webapp)

Team 02
https://limoney.org
8/5/2025 at 6:00 AM

Name	**Role**
Emily Perez (eperez@sfsu.edu)	Team Lead, Database Administrator, Github Master
Ishaank Zalpuri	Frontend Lead, UI Designer, Frontend Developer
Andrew Brockenborough	Technical Writer
Dani Luna	Frontend Lead, UI Designer, Frontend Developer
Jonathan Gonzalez	Software Architect, Frontend Lead, UX Specialist,
Gene Orias	Backend Lead

Milestone	**Version**	**Date**
Milestone 5	No Version	8/5/25
Milestone 4	Version 2	8/5/25
Milestone 4	Version 1	7/31/25
Milestone 3	Version 2	7/31/25
Milestone 3	Version 1	7/22/25
Milestone 2	Version 2	7/10/25
Milestone 2	Version 1	7/3/25
Milestone 1	Version 2	7/2/25

Milestone 1	Version 1	6/17/25

Table of Contents

Table of Contents **3**
Product Summary **4**
Product Name: 4
Final Functional Requirements (P1 and P2) 4
Major Functions and Services Provided by the Software Product 7
Unique Features 8
Deployment URL: https://limoney.org 8
Milestone Documents **9**
Milestone 1 9
Milestone 2 44
Milestone 3 83
Milestone 4 121
Team Member Contributions **159**
Post-Analysis – Lessons Learned **166**

Product Summary

Product Name:

Limoney

Final Functional Requirements (P1 and P2)

Priority 1

1. User

1.1. A user shall be able to create an account.

1.2. A user shall be able to securely log in to their account.

1.3. A user shall be able to recover or reset their password.

1.7. A user shall be associated with one or more accounts.

1.10. A user shall be able to view and edit their financial data history

2. Account

2.1. An account shall store income entries for a user.

2.2. An account shall store expense entries for a user.

2.4. An account shall track a balance value.

2.5. An account may be connected to one or more banks.

2.7. An account shall have a transaction history log.

3. Bank

3.1. A bank shall be linked to one or more user accounts.

3.2. A bank shall provide transaction data to the system.

4. Transaction

4.1. A transaction shall be classified as income or expense.

4.3. A transaction may be manually entered or synced from a bank.

5. Subscription

5.1. A subscription shall be tied to a recurring transaction.

6. Budget

6.1. A budget shall be created for one or more spending categories.

6.2. A budget shall belong to one and only one account.

6.3. A budget shall allow setting and tracking of goals.

13. ReimbursementRequest

13.1. A reimbursement request may be a request for money to a valid user.

13.2. A reimbursement request may be sending money to a valid user.

13.2. A reimbursement request shall have a status: pending, approved, or rejected.

14. SupportTicket

14.1. A support ticket shall be created by a user.

14.2. A support ticket shall include a subject, message, and status.

15. AccountBankLink

15.1. An account may be linked to one or more banks.

15.2. A bank may be linked to one or more accounts via AccountBankLink.

16. Bank

16.1. A bank shall store balance and credit data for users.

16.2. A bank shall provide transaction data to the system.

16.3. A bank shall be viewable in the user's linked accounts page.

17. Reviews

17.1. A review shall be optionally submitted by a registered user.

17.2. A review shall include a rating value (1 to 5) and an optional message.

17.3. A review shall be timestamped at the time of creation.

17.5. A review may exist independently as general user feedback not tied to any LemonAid.

17.7. A review shall be categorized by type (chatbot or user_experiance).

Priority 2

1. User

1.4. A user shall be able to select between manual or bank-linked financial tracking.

1.15. A user that is an administrator shall be able to respond to support tickets.

2. Account

2.6. An account shall be able to categorize transactions.

2.9. An account shall track subscriptions.

3. Bank

3.3. A bank shall store balance and credit data for users.

4. Transaction

4.7. A transaction may be associated with a reimbursement request.

5. Subscription

5.2. A subscription shall be viewable in a centralized dashboard.

13. ReimbursementRequest

13.3. A reimbursement request shall be visible to administrators for review.

14. SupportTicket

14.3. A support ticket shall be responded to by an administrator.

14.4. A support ticket shall track the admin response and resolution state.

Major Functions and Services Provided by the Software Product

The software offers a wide selection of tools designed to help users take control of their personal finances through intuitive, secure, and intelligent features. Core functions and services include:

1. Account Aggregation

- Connect and manage multiple bank accounts in one place.
- Support for both manual and third-party-linked account integration.

2. Transaction Tracking

- Automatically imports and categorizes transactions by type, date, and account.
- Supports search, filter, and manual entry/editing of financial activity.

3. Budgeting Tools

- Create and manage personalized budgets across various categories (e.g., groceries, rent, entertainment).

- Real-time budget tracking and spending alerts.

4. Subscription Monitoring

- Detects recurring payments and subscriptions.

5. Reimbursement Requests

- Allows users to submit, track, and manage reimbursement requests tied to specific transactions.
- Includes approval statuses and ranking tools for prioritization.

6. AI Financial Assistant: LemonAid

- Provides personalized insights and recommendations using AI.

- Helps users optimize budgets, identify saving opportunities, and understand spending patterns.

6. Security and Privacy

- Session-based authentication with optional multi-factor support.
- Users can control data sharing, linked accounts, and notification preferences.

7. Currency Converter

- Built-in tool for converting between multiple global currencies in real time.

Unique Features

- **LemonAid AI**
  Our product uses AI to analyze spending behavior and create budgets for users.

- **Reimbursement Request System**
  This feature helps track reimbursements. These transfers are designed for Limóney users to pay each other back. Users can send each other reimbursement requests on Limóney. Users can also receive these requests and pay back vice versa.

Overall, what makes them superior to competitors is that it is an all in one app used to make dealing with money easier for the common folk. These personalized insights from the AI will make users learn at a faster rate and save money along the way.

Deployment URL:
https://limoney.org

Milestone Documents

Milestone 1

SW Engineering CSC648-848-05 Summer 2025

Limóney (Financial Budgeting WebApp)

Team 02

Milestone 1

7/3/25

Name	**Role**
Emily Perez (eperez@sfsu)	Team Lead
Ishaank Zalpuri	Database Administrator
Andrew Brockenborough	Technical Writer
Dani Luna	Github Master, Database Administrator, Frontend Lead
Jonathan Gonzalez	Software Architect
Gene Orias	Backend Lead

Milestone	**Version**	**Date**
Milestone 1	Version 2	7/3/25
Milestone 1	Version 1	6/17/25

Table of Contents

1. Title Page

2. Table of Contents

3. Executive Summary

4. Main Use Cases

5. List of Main Data Items and Entities

6. Initial List of Functional Requirements

7. List of Non-Functional Requirements

8. Competitive Analysis

9. Checklist

10. High-Level System Architecture and Technologies Used

11. List of Team Contributions

Executive Summary

Limóney is an AI-enhanced budgeting web application designed to help users manage their finances all in one platform. We live in a time where many people struggle with managing recurring expenses, forgotten subscriptions, and inconsistent saving habits. Because of this, there's a growing need for financial tools that are not just useful, but also simple and easy to use. Our motivation is to keep the application both functional and accessible so everyday users can stay on top of their finances without feeling overwhelmed by data or complex interfaces.

Inspired by *EveryDollar* and *Rocket Money*, what sets *Limóney* apart is that it combines the best features and usability of both apps into one platform, while also integrating AI to provide personalized, intelligent budgeting insights, which is something neither of those apps currently offer. The built-in assistant provides tailored spending suggestions, identifies unnecessary expenses, and recommends ways to save based on the user's financial habits. The result is a smarter, more intuitive budgeting experience that helps users improve their financial well-being over time.

Main Use Cases

Main Use Case 1

Create Budget

Actors:

- Erik
- *Limóney*

Assumptions:

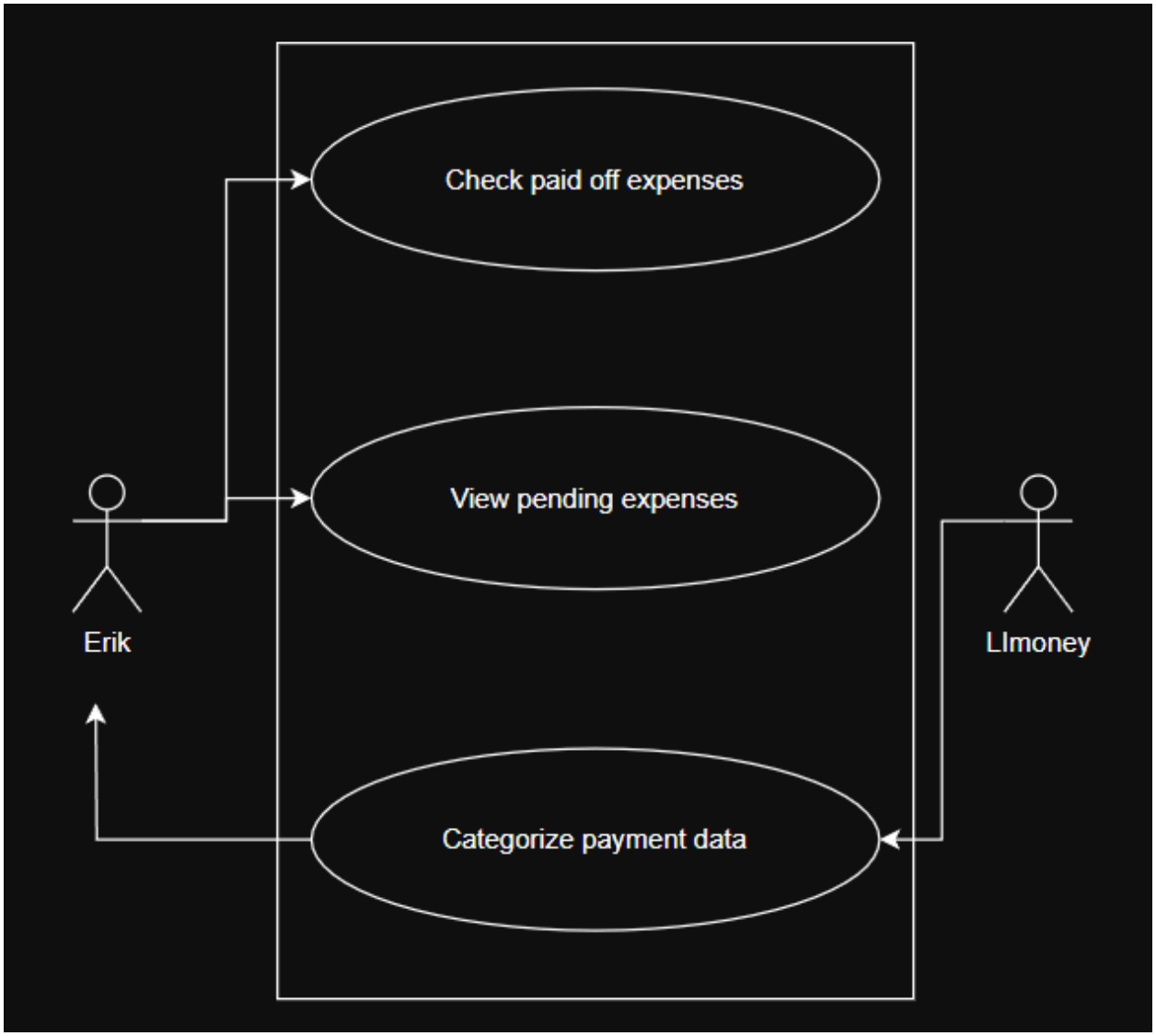
- Erik has multiple credit cards.
- He has made recent payments, but is unsure of which expenses got paid.
- Erik has connected his bank info to *Limóney* to keep track of his expenses.

Use Case:

- Erik made a credit card payment, but is unsure of what was actually paid. He opens up *Limóney*, and the app provides a clear list of which charges were paid off. For ex: his textbooks, school supplies, and what is still needed. Being able to see which expenses have been paid has given Erik peace of mind and helps him plan out his next payments with confidence.

Benefits for ...:

- Users:
    - Users gets to see which expenses have been paid.
    - Users gets help to decide which expenses to pay off next.
    - Users feel more in control of their credit and their spending



Main Use Case 2

Tracking multiple reimbursements and predicting balances

Actors:

- Student A
- Student A's group of friends

Assumptions:

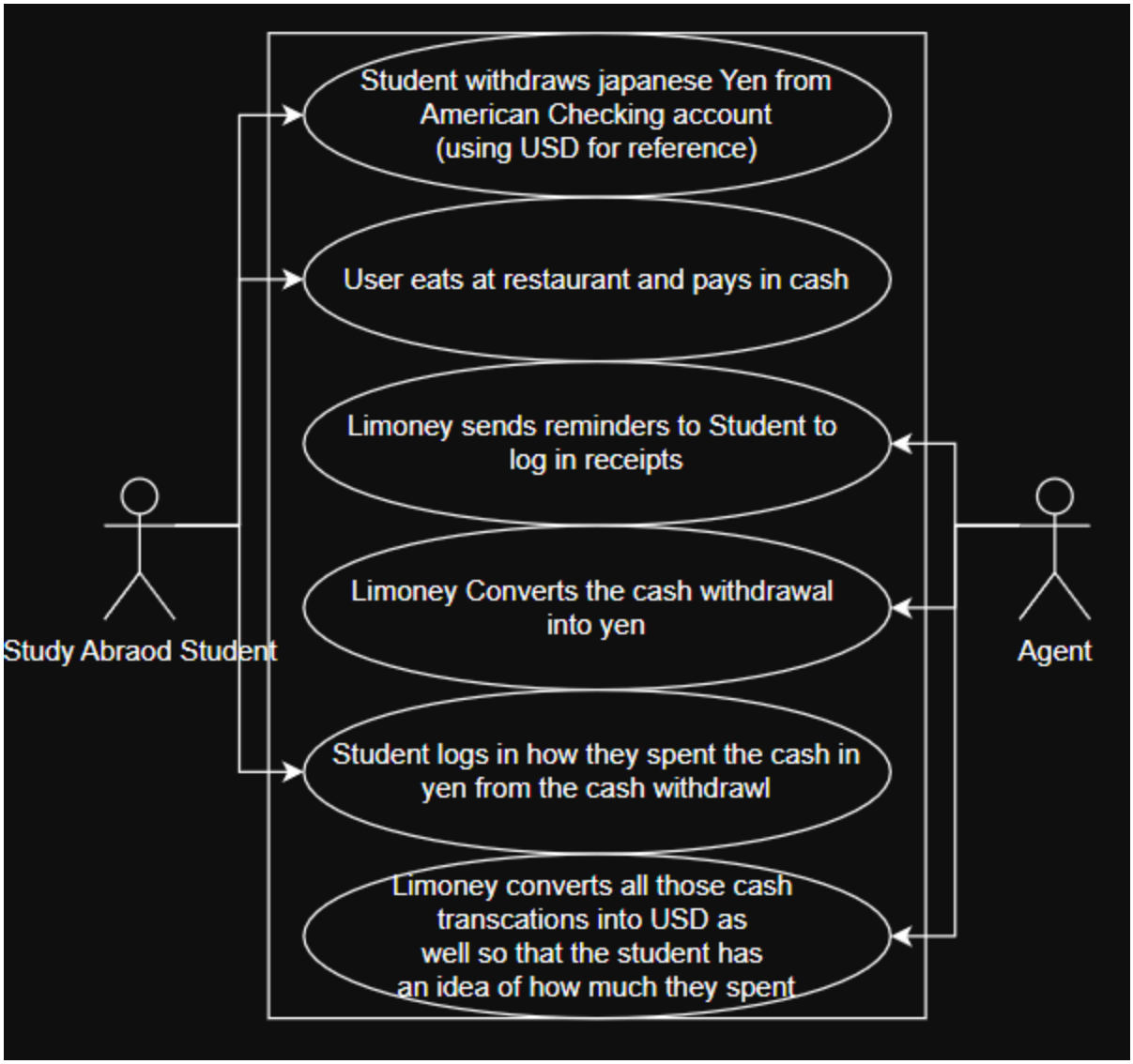
- Student A is somewhat financially stable

Use Case:

- Student A goes eating out with a group of friends once a month. Since they are a large group, it is usually difficult to split the bill among all of them. He pays the bill and expects his friends to reimburse him later, but everyone gets busy and can't keep track of how much needs to be paid back. The next time they go eating out, he puts the meal on his card. The student goes to the app right away and logs in the transaction. He also has the option of splitting the bill by a number of friends or splitting it by what they order and adding their names. These splits are called reimbursement requests, and the student can even send them to his friends as reminders. The request will be active until it is complete, and the money paid back will reflect on his account balances.

Benefits for...:

- Users
    - Users can keep track of who owes money back to him
    - Users are held accountable and become more financially responsible



Main Use Case 3

Log Transactions

Actors:

- Erik (Busy College Student)
- *Limóney* (Company)

Assumptions:

- Erik juggles school and work on a packed schedule.
- He stores physical receipts but forgets to log them at the end of the day.
- Erik prefers to manually enter his expenses for privacy reasons.

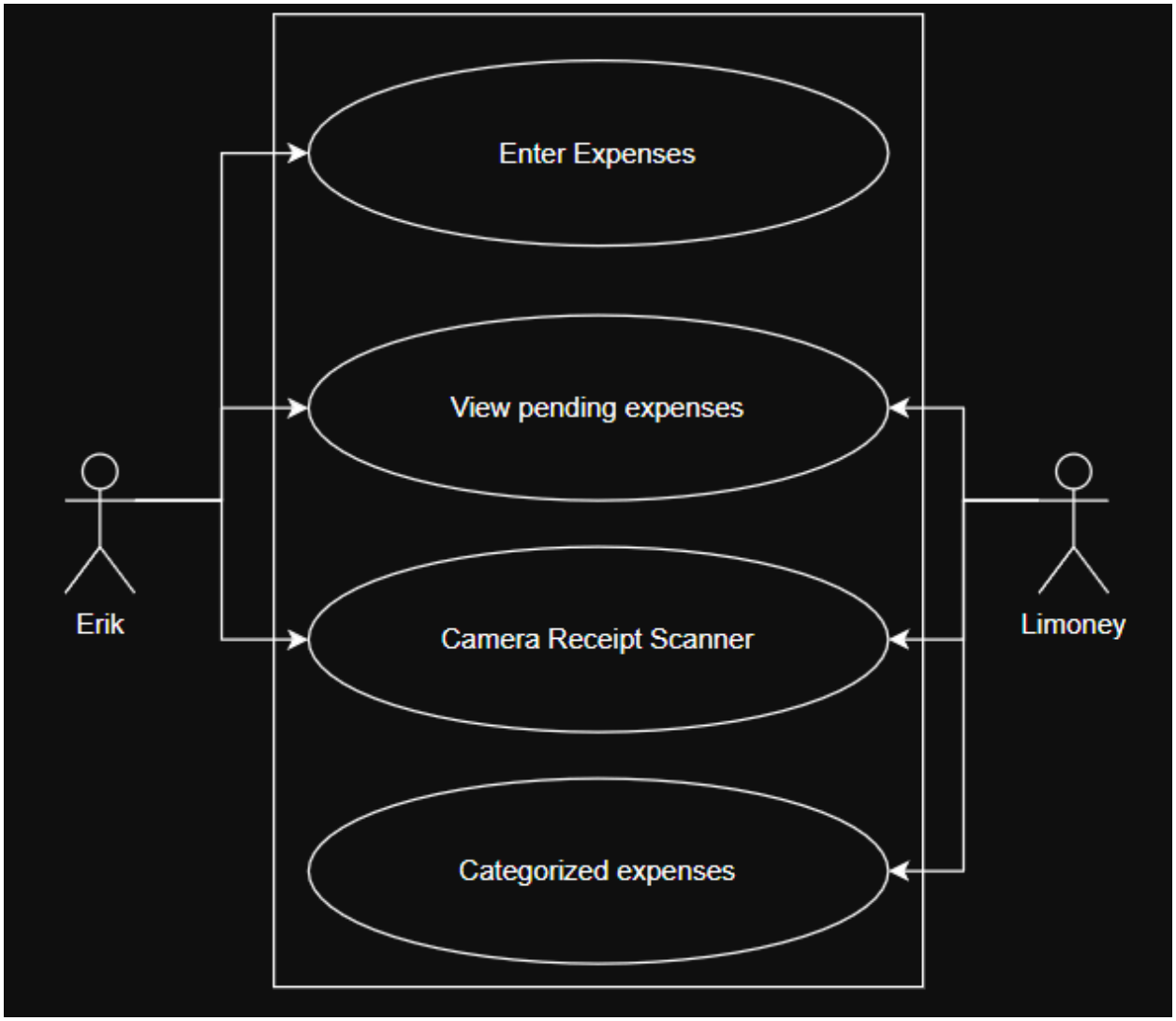
- He does not use updated bank tools.

Use Case:

- Erik tries to keep track of his spending, but with work and school, he tends to forget about his receipts, which pile up. Logging the receipts by hand feels like too much. One day, *Limóney* sends out a reminder and suggests that he can scan the receipts via his camera instead. Erik tries it and is surprised by how quickly and efficiently it works. Erik is now logging expenses through *Limóney* because it provides a simple way to keep track of his receipts and has become a nightly routine.

Benefits for ...:

- Users:
    - Users have an easier time logging receipts with the camera scanner.
    - Users get helpful reminders help Erik stay on track.
    - Users with no technical skill have an easier time managing their finances.



Main Use Case 4

Security Concern

Actors:

- Andrew
- *Limóney*

Assumptions:

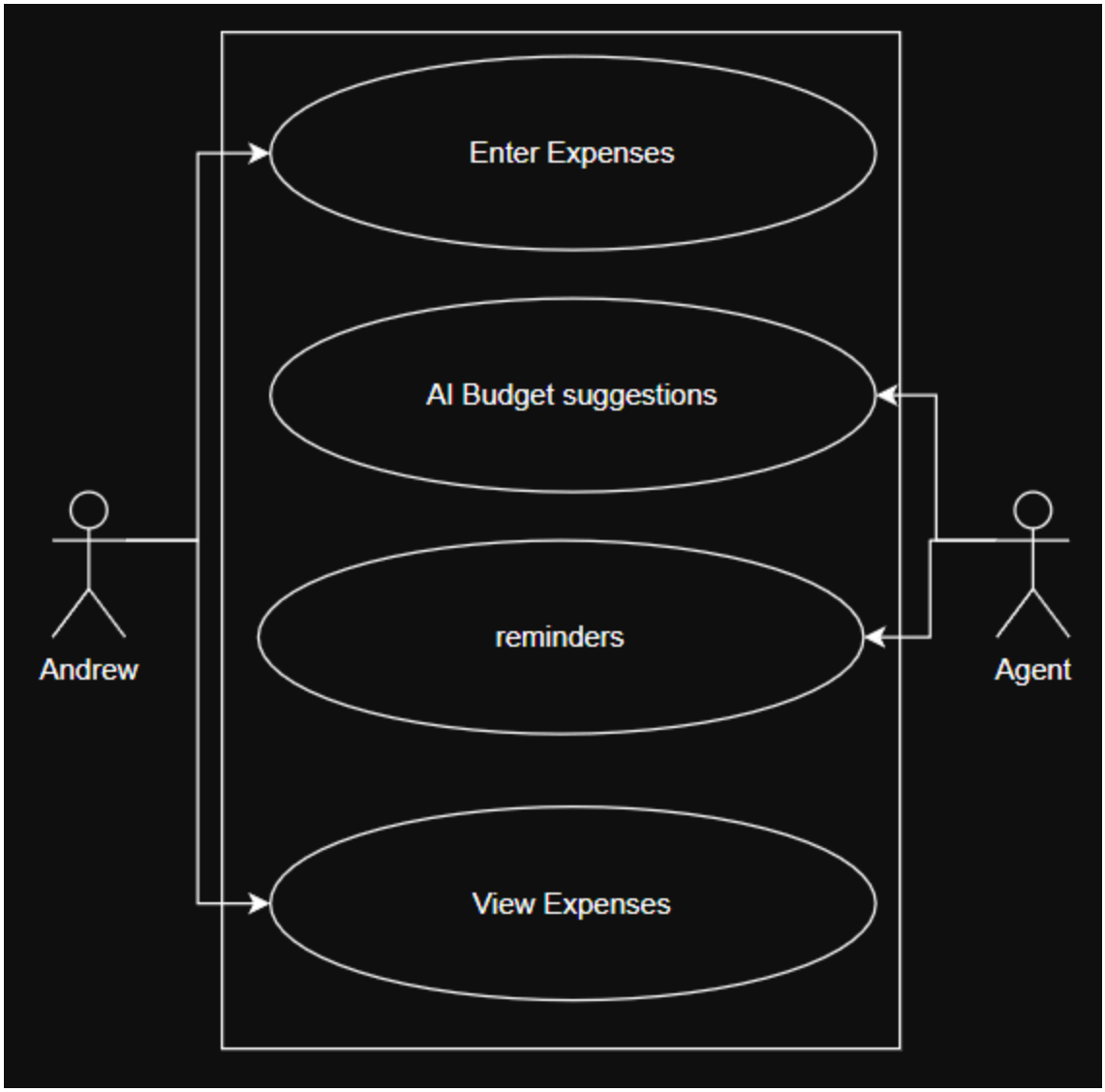
- The user manually enters receipts
- They don’t have the latest technologies
- They don’t want to use apps that access their financial information

Use Case:

- Andrew is serious about his privacy and refuses to use financial apps that require access to his bank account or store data externally. For years, he has been using spreadsheets to track his spending manually. The process has become overwhelming to do. He discovers *Limóney* , a secure, AI-enhanced budgeting app designed for users like him. *Limóney* allows users to enter transactions manually and get personalized budgeting insights without ever syncing financial accounts. Andrew starts using *Limóney* interface to log purchases. The built-in AI analyzes his patterns and suggests simple ways to save like avoiding frequent late-night food orders. Andrew gets skeptical on AI and has the choice of turning off the AI suggestions. What surprises Andrew most is how nice it feels: he stays in control, but still benefits from smart financial planning tools

Benefits for ...:

- Users:
    - Users can manage finances without linking sensitive financial accounts.
    - Users get smart recommendations based on spending habits.
    - Users know their data stays local and private.



Main Use Case 5

Receive AI Suggestions

Actors:

- Aaron
- Derrick (Aaron's Roommate)
- *Limóney*

Assumptions:

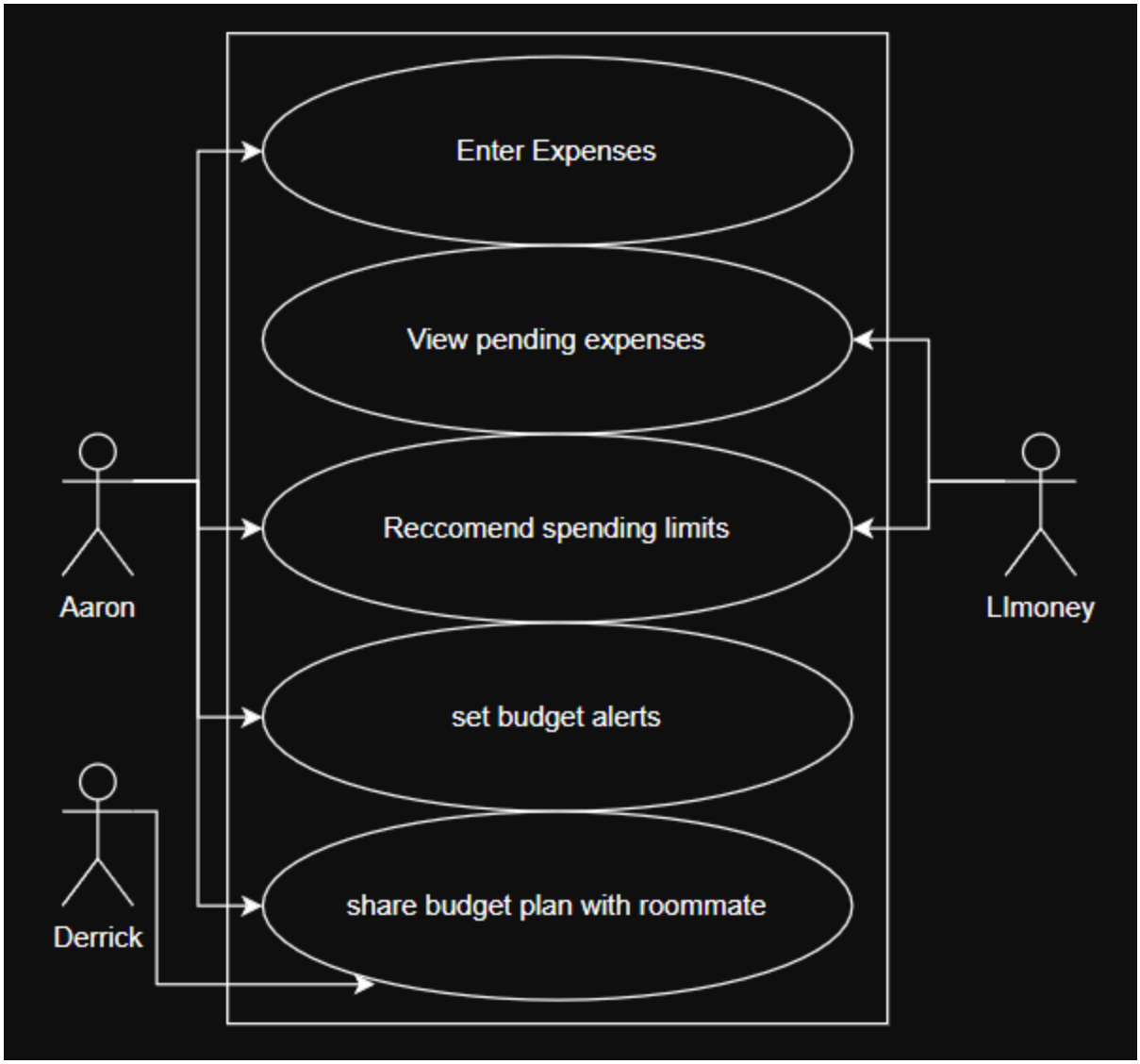
- Aaron and Derrick have separate bank accounts and budgets in *Limóney*
- Aaron has connected his banking info to the *Limóney* App

Use Case:

- On a Friday evening, Aaron opens the *Limóney* App to figure out why he keeps running low on money every month. The *Limóney* app, quickly points out that his dining expenses are higher than usual and suggests a closer look. Aaron notices that he has several delivery charges that had been adding on. The *Limóney* App recommends that Aaron sets up a weekly limit which will allows him to see how much he can potentially be saving. Relieved by this, Aaron sets a spending alert and feels in control of his budget and shares it with Derrick. Aaron and Derrick decide that cutting back on dining expenses can save them a lot so they begin cooking meals at home together and decrease their spending.

Benefits for ...:

- Users:
    - Users can understand exactly where their money is going.
    - Users can set spending alerts and adjust their budget in real time.
    - Users feel confident and are more proactive with their savings.



Main Use Case 6

Currency Exchange

Actors:

- Study Abroad Students

Assumptions:

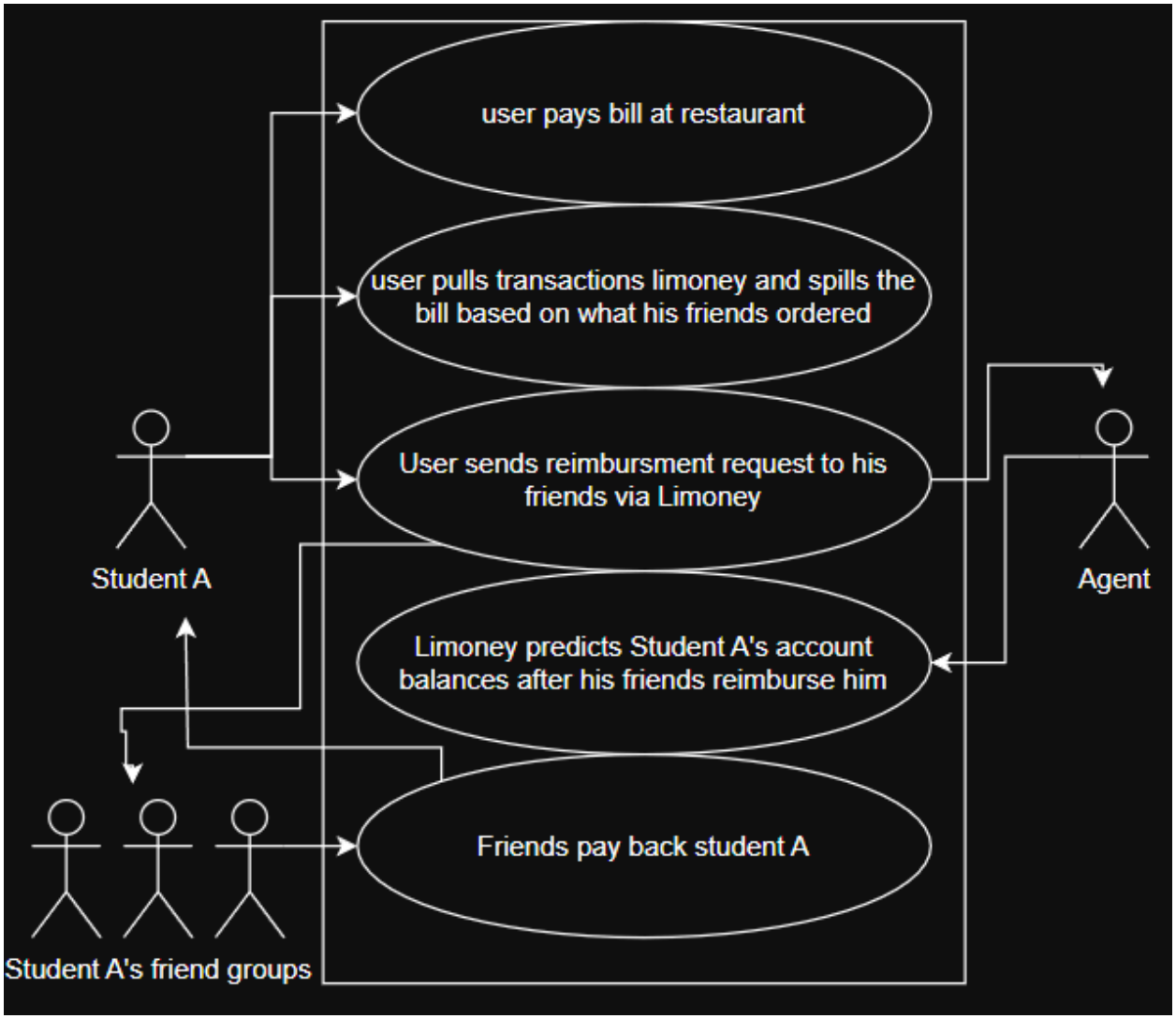
- Some Students struggle to adjust to foreign currency and want to get used to the system

Use Case:

- Study abroad student goes to Japan. The country paper based, most restaurants only take cash. Whether the student opts to manually track expenses or link his bank account, he needs to withdraw cash and keep a track of how he spends it. It gets confusing however as he collects receipts and the currency rate changes every day. By the time he sits down to enter his cash transactions, he has to spend extra time converting currency. Instead of stressing out, he chooses to switch the currency on the app to Japanese Yen so he can get used to the currency abroad and accurately log how he used his cash withdrawal. At the end, he also converts all those transactions back into USD to get an idea of how much he is spending and what the Japanese economy is like

Benefits for...:

- Users:
    - Users can keep a track of their expenses abroad and get used to foreign currency.



Main Use Case 7

Tracking ongoing expense

Actors:

- Max who has a lot of active subscriptions
- Max doesn't really know much about the digital landscape

Assumptions:

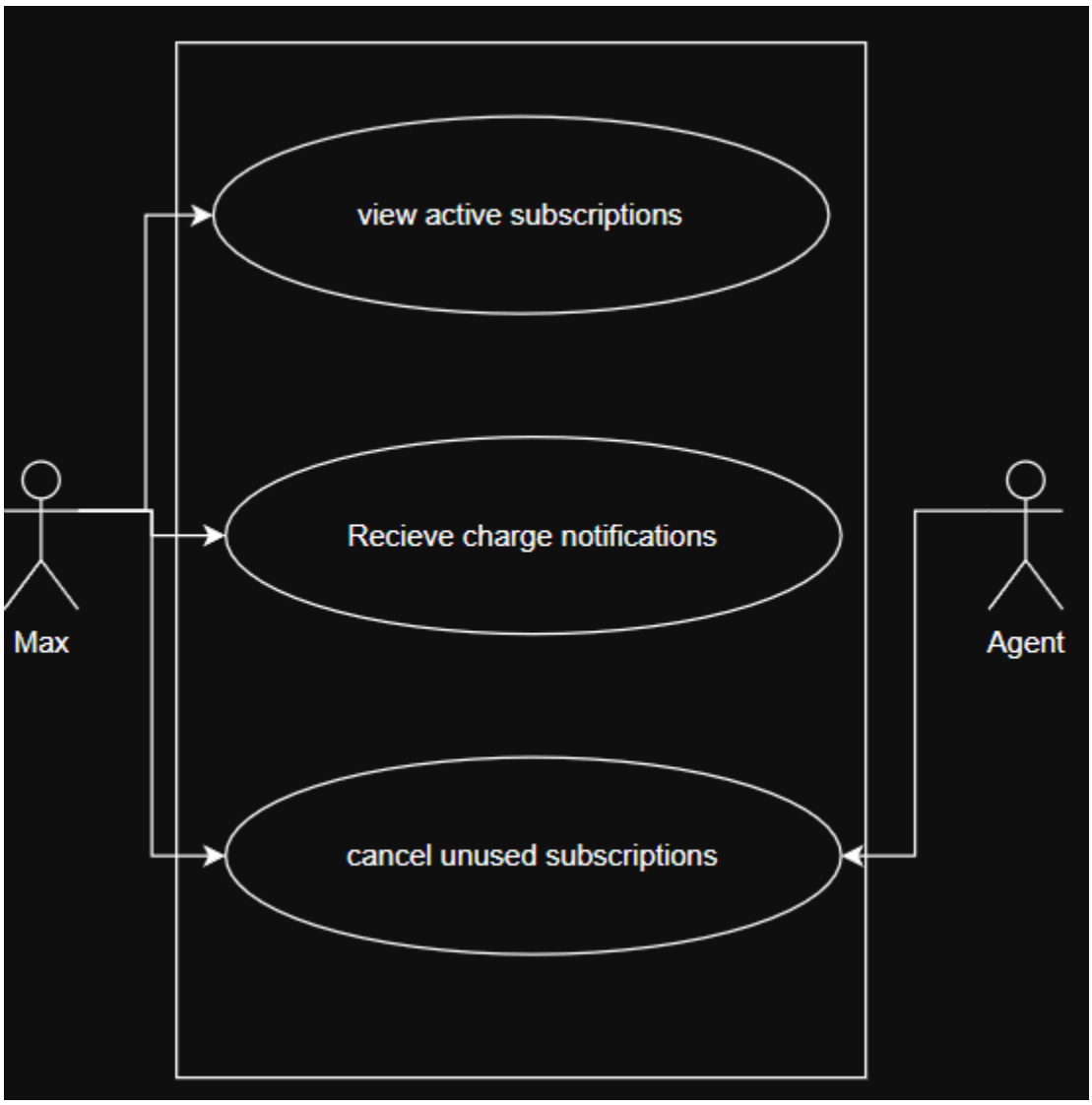
- Max has a lot of active subscriptions

Use Case:

- Max is a busy college student juggling classes, a part time job, and taking care of his siblings. Lately he has been feeling very overwhelmed and doesn't have time to check his bank account. Max suddenly got a notification on his phone that he has been charged 20$ for a subscription he didn't even know he had. Max then downloaded the *Limóney* app which can help him manage his subscriptions. Instantly he gets greeted by the active subscriptions he has on right now! With a few taps those useless subscriptions have been cancelled!

Benefits for ...:

- Users:
    - Users can prevent ongoing subscriptions as to easily see everything.
    - Users have the ability to see unwanted or unused subscriptions.



Main Use Case 8

Predicting balances end of month

Actors:

- Alex who loves planning ahead

Assumptions:

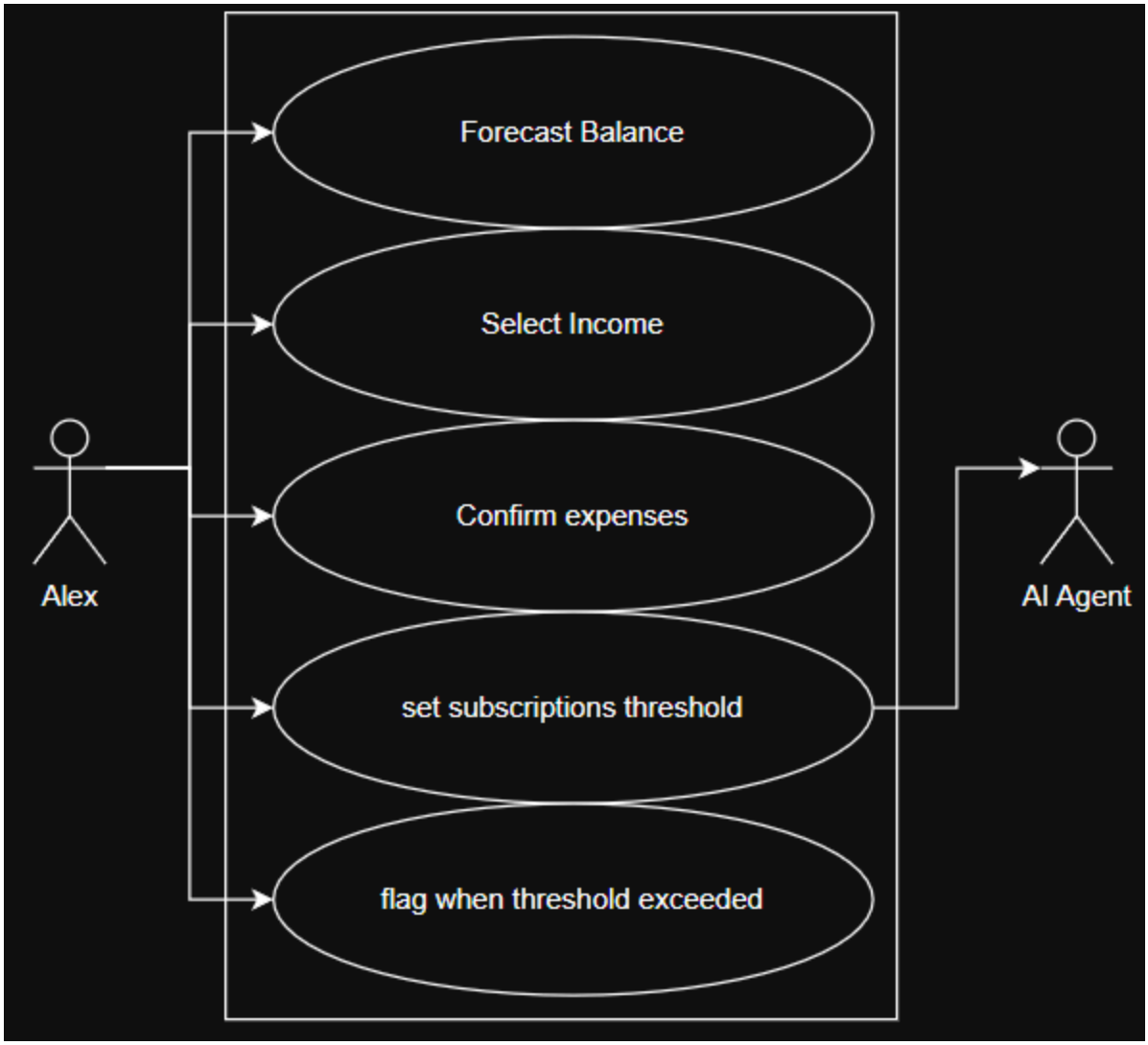
- Alex has stability within expense and income
- Alex updates their transactions regularly

Use Case:

- Alex, who is a planner by nature, wants to know if he will be financially comfortable by the end of the month. He opens the *Limóney* app and selects “Forecast Balance”. The app prompts him to select his income and confirms his current expenses like subscriptions. Alex then sets a threshold of 100$ within subscriptions so the app can flag Alex whenever he goes above that mark.

Benefits for ...:

- Users:
    - Users can completely avoid disturbance within the stability of expenses
    - User can be aware of any balances that could surprise then and can see their subscription bills easily and adjust accordingly.



Main Use Case 9

Signing up/Intro UI

Actors:

- John
- *Limóney*

Assumptions:

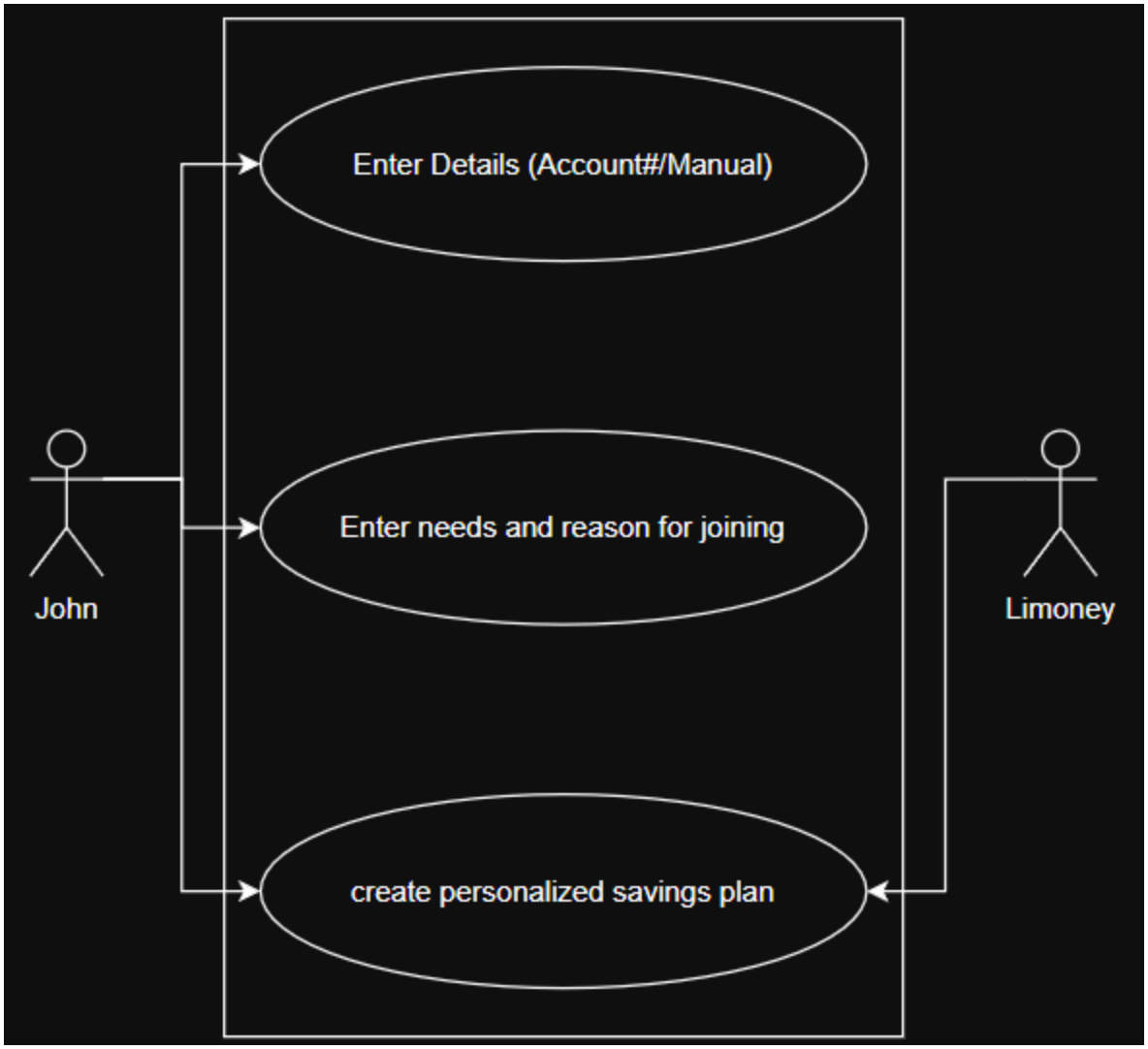
- John wants help with budgeting his expenses
- John finds *Limóney* ad interesting and clicks

Use Case:

- John is worried about his finances. He is concerned about going broke. Enter *Limóney*, a budgeting app he found in an ad. He is now signed up, and it is now asking him questions. It begins with why John is on the app. What does he need help with? (budgeting, managing expenses, investment, tracking subscriptions) Afterwards, it asks if John wants to share his data using his SS and banking info, or if he prefers to manually enter his info. If he chooses the latter, it takes him to fill out the information he needs help with (income, average expenses, current bills, etc.). He then receives a message welcoming him to *Limóney* and a well-wishing of “Happy Savings!”, with an immediate option to work with *Limóney* to start saving money.

Benefits for...:

- Users
    - Users can feel welcomed and impressed by the application while giving them confidence that they will be taken care of.
- Company
    - The company will benefit from the increase of the application' traffic through word of mouth.



Main Use Case 10

Chatbot/AI assistance

Actors:

- John
- LemonAid

Assumptions:

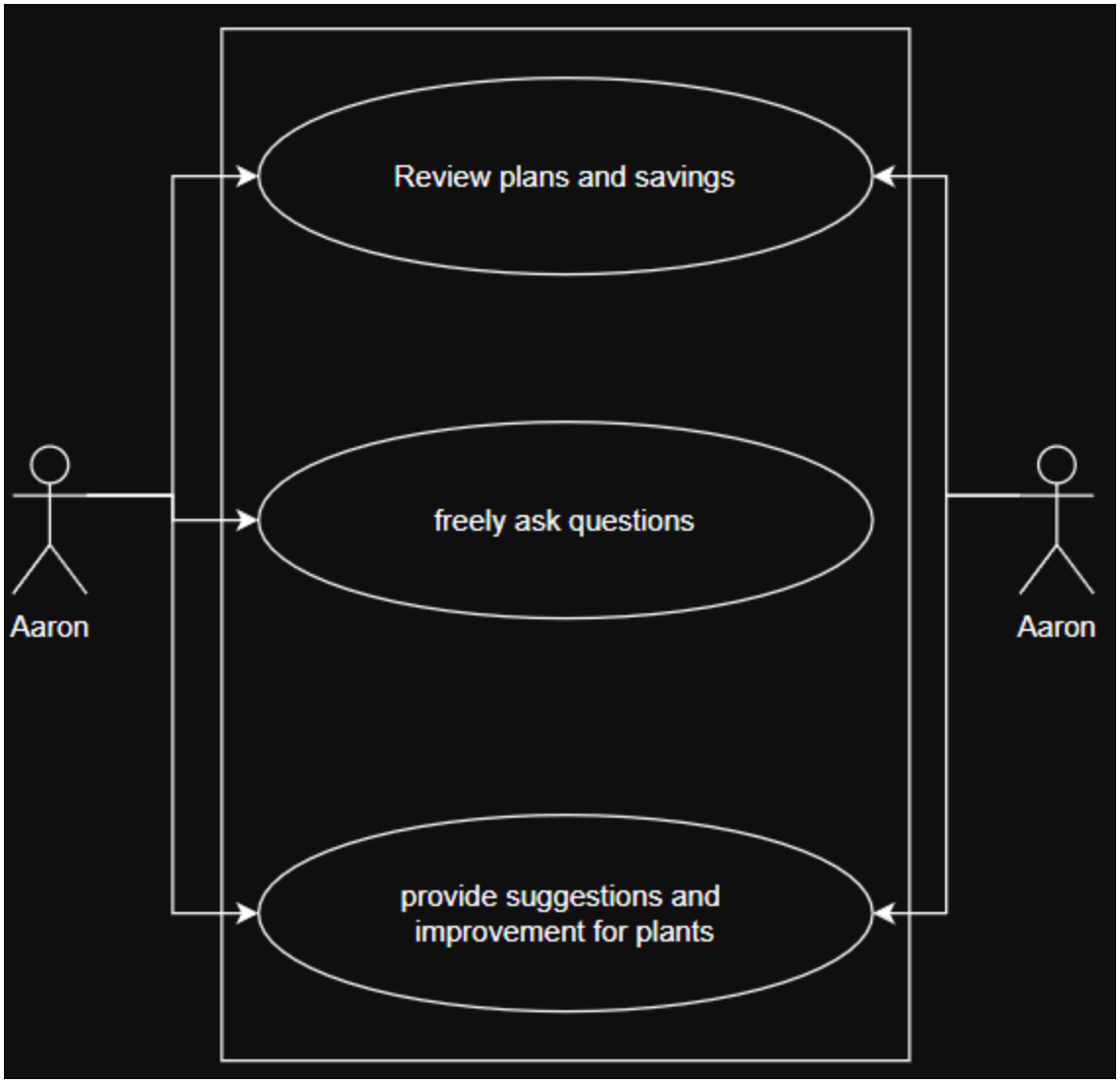
- John is familiar with the concept of a Chatbot and feels comfortable using it, understanding it is only for reference and will not be 100% accurate
- John likes the idea of having his entries analyzed and would like to have an outside perspective on his savings without sacrificing his privacy

Use Case:

- John has successfully signed up and is enjoying using *Limóney*. However, all the technical jargon and the routine of inputting and reading numbers gets tiring. He wants to take a bit of a break and wants to be able to discuss his finances with others. He opens LemonAid, the company's financial aid assistance, and asks it for help. To streamline the process, LemonAid asks John how exactly it can help him today, while providing several recommendations that he can click on or directing him to enter his own question in the text field.

Benefits for …:

- Users:
    - Users shall be able to discuss their finances with an AI assistant, allowing them not only to receive feedback from it, but also allowing them to think more deeply on the subject matter they are asking about. All while maintaining their privacy and not being required to share their finances with others.
- Company:
    - The company shall be able to streamline the feedback process and make communication much easier. If users choose to share their chat information, the company can use it to further develop the AI and train it to be prepared for more situations.



List of Main Data Items and Entities

1. Main Entity: Users

Defines all types of users within the application.

1.1 Role Type: Customer

- **Definition:** Basic user type who utilizes the app's core features.
- **Behavior:** Can view finances, complete tasks, interact with AI, and earn rewards.

1.2 Role Type: Premium

- **Definition:** Enhanced user role with access to exclusive financial features.
- **Behavior:** Includes all customer features, plus extras.

1.3 Role Type: Tester

- **Definition:** Temporary or experimental users who test new features or versions.
- **Behavior:** Provides feedback based on testing sessions.

1.4 Role Type: Administrator

- **Definition:** Pinnacle of users with control over platform settings and user management.
- **Behavior:** Can manage users, resolve tickets, and oversee operations.

---

2. Main Entity: Tasks

- **Definition:** Actions users can perform to manage or improve finances.
- **Behavior:** Users can complete, receive, and view tasks, may be may be recommended by LemonAid.

---

3. Main Entity: Account

Central system for tracking money flow, balances, and budgeting

3.1 Role Type: Income

- **Definition:** Tracks the sources and amounts of funds received.
- **Behavior:** Adds to account balance and informs budgeting and progress metrics.

3.2 Role Type: Expense

- **Definition:** Records money spent by the user.
- **Behavior:** Subtracts from account balance, contributes to budgeting, and is analyzed for trends.

3.3 Role Type: Saving

- **Definition:** Entity related to saving money.
- **Behavior:** Tracks actions taken, or that can be taken, by user in order to improve this in a stat based fashion.

---

4. Main Entity: Bank

Banks connected to user accounts

4.1 Role Type: Credit

- **Definition:** Amount of money that the User is in debt.
- **Behavior:** Tracks user debt from using cards, taking loans, etc.

4.2 Role Type: Total

- **Definition:** Total of user bank assets.
- **Behavior:** Can determine User net-worth, and be used with other entities to determine savings and other things.

---

5. Main Entity: Transaction

- **Definition:** Represents the movement of money within an account.
- **Behavior:** Provides data to other entities to help improve user savings.

---

6. Main Entity: Budget

- **Definition:** Helps users control and plan spending.
- **Behavior:** Tracks and provides limitations on user spending.

---

7. Main Entity: Subscription

- **Definition:** Payments for installments towards different services and plans.
- **Behavior:** Used to calculate user spending and improve spending by eliminating subscription over-population.

---

8. Main Entity: Reimbursement Request

- **Definition:** Request for financial compensation.

- **Behavior:** Is created by user and reviewed by Admin that will take action to aid user.

---

9. Main Entity: Notifications

- **Definition:** In-app messages that alert users to important events or tasks.
- **Behavior:** Informs users about tasks, budget limits, etc.

---

10. Main Entity: Support Ticket

- **Definition:** A message submitted to administrators for technical or financial assistance.
- **Behavior:** Sent to Admins for resolutions.

---

11. Main Entity: LemonAid (AI Assistant)

- **Definition:** AI Assistant integrated into the app.
- **Behavior:** Interacts with users, logs actions, and improves user guidance using AI.

Initial List of Functional Requirements

1. User

1.1. A user shall be able to create an account.
1.2. A user shall be able to securely log in to their account.
1.3. A user shall be able to recover or reset their password.
1.4. A user shall be able to select between manual or bank-linked financial tracking.
1.5. A user shall be able to customize their interface (e.g., dark mode).
1.6. A user shall be able to receive AI-generated financial guidance.
1.7. A user shall be associated with one or more accounts.
1.8. A user shall be classified as a customer, premium, tester, or administrator.
1.9. A user shall be able to interact with the AI chatbot for financial assistance.
1.10. A user shall be able to view and edit their financial data history.

---

2. Account

2.1. An account shall store income entries for a user.
2.2. An account shall store expense entries for a user.
2.3. An account shall belong to one and only one user.
2.4. An account shall track a balance value.
2.5. An account may be connected to one or more banks.
2.6. An account shall be able to categorize transactions.
2.7. An account shall have a transaction history log.
2.8. An account shall support spending limits and alerts.
2.9. An account shall track subscriptions and bill payments.
2.10. An account shall be able to forecast end-of-month balances.

---

3. Bank

3.1. A bank shall be linked to one or more user accounts.
3.2. A bank shall provide transaction data to the system.
3.3. A bank shall store balance and credit data for users.

---

4. Transaction

4.1. A transaction shall be classified as income or expense.
4.2. A transaction shall belong to exactly one account.
4.3. A transaction may be manually entered or synced from a bank.
4.4. A transaction shall be categorizable by the user or AI.
4.5. A transaction shall be editable and deletable by the user.
4.6. A transaction shall be linked to one or more receipts.
4.7. A transaction may be associated with a reimbursement request.

---

5. Subscription

5.1. A subscription shall be tied to a recurring transaction.
5.2. A subscription shall be viewable in a centralized dashboard.
5.3. A subscription shall trigger periodic spending alerts.

---

6. Budget

6.1. A budget shall be created for one or more spending categories.
6.2. A budget shall belong to one and only one account.
6.3. A budget shall allow setting and tracking of goals.
6.4. A budget shall support notifications based on spending limits.
6.5. A budget shall be able to generate monthly recaps.

---

7. Task

7.1. A task shall be assigned to a user account.
7.2. A task shall be of one type: savings, investing, setup, admin, testing.
7.3. A task shall be marked completed upon user action.
7.4. A recommended task shall be generated based on user activity or AI analysis.

---

8. AI Assistant (LemonAid)

8.1. The AI assistant shall analyze user transactions and spending habits.
8.2. The AI assistant shall recommend saving opportunities and budget changes.
8.3. The AI assistant shall forecast recurring charges and balances.
8.4. The AI assistant shall provide suggestions for uncategorized transactions.
8.5. The AI assistant shall simulate different financial scenarios for planning.

---

9. Notification System

9.1. The system shall send reminders for logging receipts.
9.2. The system shall notify users of overspending events.
9.3. The system shall notify users of upcoming bills or subscriptions.
9.4. The system shall deliver daily, weekly, or monthly financial summaries.
9.5. The system shall alert users when savings goals are off track.

---

10. Administrator

10.1. An administrator shall be able to access user account data.
10.2. An administrator shall be able to view flagged transactions.
10.3. An administrator shall be able to respond to support tickets.
10.4. An administrator shall be able to manage and moderate platform use.
10.5. An administrator shall be able to suspend or maintain site operations.

List of Non-Functional Requirements

1. Performance
1.1. The system shall respond to user actions within 2 seconds under normal load.
1.2. Backend APIs shall respond to simple requests in under 200 milliseconds.
1.3. The dashboard shall load within 3 seconds over a stable 10 Mbps connection.
1.4. AI recommendations shall be delivered within 5 seconds of user input.
1.5. The system shall support real-time syncing of transactions without freezing the UI.

2. Security
2.1. All communications shall be encrypted using HTTPS and TLS protocols.
2.2. All sensitive data, including user credentials, shall be encrypted.
2.3. Passwords shall be hashed using bcrypt or equivalent secure hashing algorithms.
2.4. Role-based access control shall be implemented to separate admin and user privileges.
2.5. The system shall prevent common web vulnerabilities such as SQL injection, XSS, and CSRF.
2.6. Only authenticated and authorized users shall be able to change, delete, or insert sensitive data.

3. Scalability
3.1. The system shall support up to 500 concurrent users without performance degradation.
3.2. The database shall support horizontal scaling across distributed servers via Docker and AWS.
3.3. The system shall efficiently scale with a 10x increase in user data and transaction volume.
3.4. Database schema shall support sharding for large financial datasets.

4. Usability
4.1. The UI shall be intuitive and require no documentation for key tasks such as setting up budgets or tracking expenses.
4.2. The system shall support both dark mode and light mode themes.
4.3. The AI assistant shall use natural, conversational language to support non-technical users.
4.4. Navigation, task completion, and interaction shall follow established UX best practices.

5. Reliability
5.1. The system shall maintain at least 99.5% uptime over a 30-day period.
5.2. Transaction data shall be backed up every 12 hours automatically.
5.3. Background processes such as syncing and reminders shall automatically retry on failure.
5.4. Error handling shall provide feedback to users without crashing the system.

6. Maintainability
6.1. The codebase shall follow JavaScript linting rules using ESLint and Prettier.
6.2. Docker containers shall be used to ensure parity between local and production environments.
6.3. The codebase shall follow a modular architecture to isolate and decouple components.
6.4. Critical modules shall maintain a minimum of 70% unit test coverage.
6.5. Version control shall be managed through Git and GitHub with frequent commits and reviews.

7. Portability
7.1. Docker containers shall allow the app to run across local development, staging, and AWS cloud

environments.
7.2. The app shall run identically across Windows, macOS, and Linux-based systems.

8. Compatibility
8.1. The system shall integrate with major U.S. banks using secure open banking APIs.
8.2. The AI engine (for ex: ChatGPT API or Gemini API) shall be modular and replaceable without impacting core functionality.
8.3. The system shall support the following browsers:
• Google Chrome v80+ • Mozilla Firefox v75+ • Microsoft Edge v80+ • Safari v11.1.1+

9. Privacy
9.1. The app shall allow users to manually enter financial data without linking any bank accounts.
9.2. User data shall not be sold to third-party services.
9.3. All stored data shall remain local unless explicit user consent is given.
9.4. Privacy mode shall restrict AI access to only local anonymized data.

10. Compliance
10.1. The system shall comply with GDPR regulations for data privacy and user consent.
10.2. Users shall be able to permanently delete their data on request.
10.3. Consent prompts shall be clearly visible during signup.

11. Supportability
11.1. System logs shall track user behavior, feature usage, and AI errors for debugging.
11.2. An admin dashboard shall allow staff to review flagged transactions and feedback.
11.3. The system shall support browser versions listed under Compatibility.
11.4. Support documentation shall be maintained for installation, setup, and deployment.

12. Efficiency
12.1. The system shall operate at no more than 60% CPU usage under normal load conditions.
12.2. Memory usage shall not exceed 75% of allocated memory on the EC2 instance.
12.3. Efficient logging shall be implemented to prevent log flooding and performance drag.

13. Look and Feel
13.1. The system shall maintain a clean, minimal aesthetic with responsive design.
13.2. UI elements shall follow a cohesive design language across all pages and devices.
13.3. Accessibility standards (WCAG 2.1 AA) shall be considered for inclusive design.

14. Coding Standards
14.1. All backend code shall conform to Node.js and Express.js best practices.
14.2. Frontend code shall follow React component structure and state management conventions.
14.3. Continuous Integration tools shall run linting and testing on every merge request.

15. Environmental Sustainability
15.1. The system shall optimize backend server usage to reduce idle time and energy waste.
15.2. Docker containers shall be configured to auto-scale down during periods of low usage.
15.3. The app shall minimize data transfer size (e.g., compress JSON responses, lazy-load images) to

reduce bandwidth and energy consumption.
15.4. The frontend shall use efficient rendering practices in React (e.g., avoiding unnecessary re-renders, using memoization).
15.5. System architecture shall consider the use of renewable-energy-based data centers (e.g., AWS sustainability zones).
15.6. Logs and backups shall be archived using energy-efficient cold storage where applicable.
15.7. Documentation shall be provided digitally only (no paper printing by default).
15.8. The system shall avoid unnecessary background polling or constant sync unless essential for real-time use.

Competitive Analysis

High-Level Competitor Overview

Competitor	URL	Primary Audience	Core Offering	Mobile Support
RocketMoney	https://www.rocketmoney.com	Budget-conscious individuals	Manages subscriptions, tracks spending, automates savings, negotiates bills.	Excellent
YNAB	https://www.youneedabudget.com	Proactive planners	Zero-based budgeting, emphasizes forward-looking control.	Strong
Mint (Intuit)	https://mint.intuit.com	Broad consumer financial users	Tracks spending, credit scores, budgeting, account syncing in one free app.	Good
Copilot	https://www.copilot.money	iOS users seeking modern UX	AI-powered categorization, real-time insights, sleek design	iOS-only

Emma	https://emma-app.com	Younger, tech-savvy users	Subscription tracking, budgeting, crypto support, social sharing.	Cross-platform
EveryDollar	https://www.everydollar.com	Envelope-budgeting enthusiasts	Zero-based budgeting with envelope-style allocation, expense tracking, debt payoff focus.	Excellent

Research Table (Qualitative Finds)

Competitor	**Key Differentiators**	**Weaknesses**	**User Target Fit**
RocketMoney	Subscription canceling; bill negotiation	Limited deeper planning; lacks personalized AI guidance	Passive optimizers
YNAB	Goal-based, zero-sum budgeting	Steep learning curve; higher subscription cost	Active planners
Mint (Intuit)	Free, long-established platform	Being phased out; inconsistent updates	Casual overview seekers
Copilot	Premium UX; AI-based category suggestions	iOS-only; premium price	Modern insight seekers
Emma	Crypto integration; social features	Feature overload; can overwhelm	Gen Z/Millennials
EveryDollar	Envelope-style zero-based budgeting; debt focus	No free bank sync for envelopes; advanced features behind paywall	Users who follow Ramsey's methodology

Feature Comparison Table

Feature	Rocket Money	YNAB	Mint	Copilot	Emma	EveryDollar	Our Product
Budgeting Tools	+	+	+	+	+	+	++
Bill Negotiation	+	-	-	-	-	-	++
Subscription Cancellation	+	-	+	-	+	-	++
Real-Time Spend Categorization	+	+	+	+	+	-	++
Credit Score Monitoring	-	-	+	-	+	-	+
Smart Saving Suggestions (AI)	+	-	-	+	+	-	++
Crypto Asset Integration	-	-	-	-	+	-	+
Bill Reminders / Notifications	+	+	+	+	+	+	++
Financial Goals Tracking	+	+	+	+	+	+	++
Envelope-Style Allocation	-	-	-	-	-	+	+
Cross-Platform Support	+	+	+	-	+	+	++

Legend:

- - = Not implemented
- + = Implemented
- ++ = Planned enhancement by our team

Summary of Competitive Insights

- **Automation & AI-driven guidance are underutilized.** RocketMoney and Copilot do well with smart suggestions, but none combine seamless AI advice with active bill negotiation and cancellation in one place.
- **Zero-based vs. passive approaches.** YNAB and EveryDollar champion envelope and zero-based budgeting, which appeals to planners, but casual users find those models too rigid.
- **Platform reach gaps.** Copilot's iOS exclusivity and Emma's feature overload highlight a need for a balanced, cross-platform experience that's powerful yet simple.
- **Legacy vs. modern churn.** With Mint's phase-out, there's an opportunity to capture lapsed users looking for a familiar but improved solution.

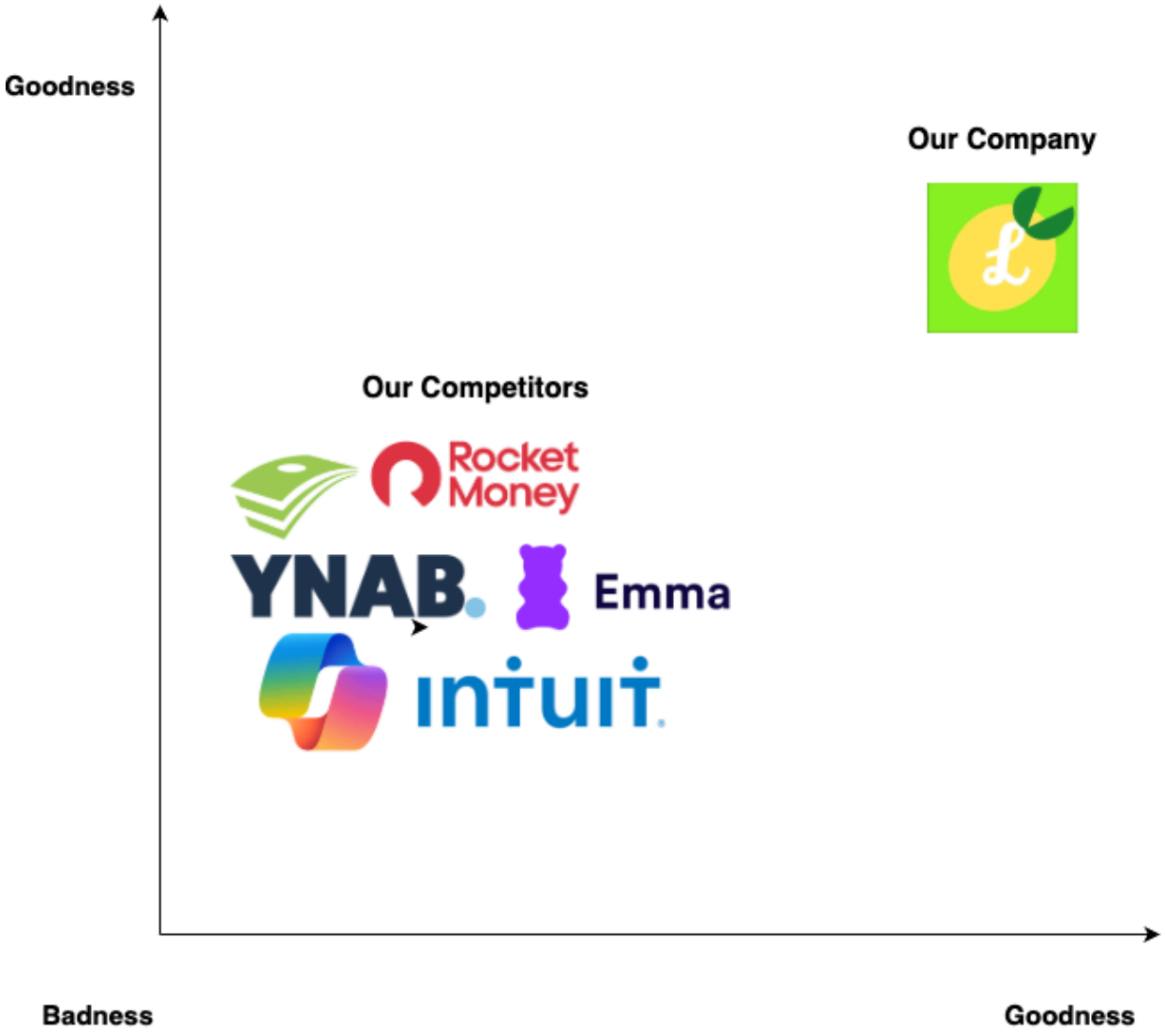
Summary of Competitive Insights

- **Unified AI & automation.** We blend proactive AI-driven savings tips with negotiated bill reductions and one-click subscription cancellations.
- **Flexible budgeting.** Support both envelope-style allocation for planners (à la EveryDollar) and passive tracking for casual users, all in one app.
- **Cross-platform clarity.** Available across web, iOS, and Android with an intuitive interface that scales from quick overviews to deep dives.
- **Stress-free control.** Built for clarity: users get actionable recommendations without the noise of endless toggles or paywalls.

Competitive Visual

On both axes, we measure overall "goodness" (ease of use, helpful features, delight). Competitors cluster in the middle: RocketMoney, YNAB, Emma, Intuit (Mint), and EveryDollar each excel in pockets but have notable trade-offs. Limoney sits in the top right, reflecting its unique blend of AI automation, envelope flexibility, and seamless

cancellation tools, delivering the most user-centric experience.



Checklist

The team needs to find timeslots outside of class time to meet

- Done - Met on 06/07/25

GitHub Master has been chosen.

- Done - The GitHub master is Dani

The team has collectively decided on and agreed to use the listed software tools and deployment server.

- Done

The team is ready to use the chosen front-end and back-end frameworks, and those who need to learn are actively working on it.

- Done - Team is mostly familiar with things we are using, currently learning/reviewing React and Sequel

The Team Lead has ensured that all members have read and understand the final M1 before submission.

- On-track
- Will be done as tech doc is turned in and explained to everyone

GitHub is organized as discussed in class (e.g., master branch, development branch, folder for milestone documents, etc.).

- Issue - Need to speak with professor regarding how to submit work and how credit is looked at.
- On-track - Met with professor, have discussed how it works, currently working on cohesion
- Done - Have determined how to pull and create branches for the project, as well as settled on using Gitbook

Members have sent an email reviewing all teammates contributions

- On - track
- Done

High-Level System Architecture and Technologies Used

- Cloud Server Host: AWS EC2 – t2.micro (1 vCPU, 1 GB RAM)
- Operating System: Ubuntu 22.04 LTS
- Database: MySQL 8.0.42
- Web Server Framework: Express.js 4.19.2
- Backend Language: JavaScript
    - Backend Framework: Node.js
- Frontend Language: HTML, CSS, JavaScript

    - Frontend Framework: React
- Containerization: Docker
- SSL Certificate: Let's Encrypt with Certbot
- Version Control & Collaboration: Git, GitHub
- Optional Tools:
    - Firebase
    - AI-Based Budgeting Assistant: Powered by ChatGPT API or Gemini API for personalized financial suggestions

List of Team Contributions

Name	Contributions (C2&C3)	Score (1-10)
Emily Perez	Wrote executive summary, wrote high level system..., set up cloud server, helped with credentials md, helped with main use cases	10
Ishaank Zalpuri	Wrote list of main data...,wrote main use cases, wrote checklist, helped set up permissions with cloud server, did what was assigned	10
Andrew Brockenborough	wrote competitive analysis, helped w main use cases, did what was assigned	10
Dani Luna	made the title page, wrote main use cases, set up aboutus page, did what was assigned	10
Jonathan Gonzalez	wrote main use cases, wrote functional req, wrote non functional req, did what was assigned	10
Gene Orias	wrote table of contents, wrote main use cases, helped with credentials md, did what was assigned	10

Milestone 2

SW Engineering CSC648-848-01 Summer 2025

Limóney (Financial Budgeting Webapp)

Team 02

Milestone 2

7/10/25

Name	**Role**
Emily Perez (eperez@sfsu.edu)	Team Lead
Ishaank Zalpuri	Database Administrator
Andrew Brockenborough	Technical Writer
Dani Luna	Frontend Lead, Backend Lead
Jonathan Gonzalez	Software Architect
Gene Orias	Backend Lead,

Milestone	**Version**	**Date**
Milestone 2	Version 2	7/10/25
Milestone 2	Version 1	7/3/25
Milestone 1	Version 2	7/2/25
Milestone 1	Version 1	6/17/25

Table of Contents

Table of Contents **1**
Data Definitions **2**
Core User & Identity Entities 2
Financial Management Entities 2
Task & Reward System 2
Communication & Logging Entities 2
Support & Admin Interaction 3
Integration & Association 3
User Privileges and Registration 3
Prioritized High-Level Functional Requirements **4**
Mockups/Storyboards **10**
High-Level System Design **15**
High Level Database Architecture 15
Initial Database Requirements 15
DBMS Selection 17
Database Organization 17
Backend Architecture 25
ALL SCALE DESIGN 25
MEDIUM SCALE DESIGN 25
LARGE SCALE DESIGN 26
ARCHITECTURE SUMMARY 27
UML Design 29
High Level Application Network Protocols and Deployment Design 30
Network & Development Diagram 30
Application Networks Diagram 31
Deployment Diagram 32
Integration with External Components 32
High Level APIs and Main Algorithms 33
High-Level APIs 33
Main Algorithms and Processes 33
Software Tools and Frameworks 34
Key Project Risks **35**
Project Management **37**
List of Team Contributions **38**

Data Definitions

Core User & Identity Entities

1. **User**
   Represents any person registered in the system. Each user has an account profile and can be either a standard user or an administrator. Used for login, personalization, and access control. Email is the primary identifier, and all actions in the system are associated with a user.

Financial Management Entities

1. **Account**
   Represents a user's financial container (checking, savings, or credit). All financial activities, such as transactions, subscriptions, and budgets, are linked to specific accounts. Accounts maintain running balances and support multi-bank integration.
2. **Transaction**
   Represents a monetary event tied to an account (like income, expense, or fund transfer). Transactions are categorized for budgeting and reporting purposes and are timestamped to track financial behavior over time.
3. **Receipt**
   Stores digital copies of receipts (image files) linked to individual transactions. Used for personal finance tracking, tax documentation, and reimbursement validation.
4. **ReimbursementRequest**
   Represents a formal request to be reimbursed for a transaction, often reviewed by an administrator. Tracks status and supports workflows such as pending, approved, or rejected.
5. **Budget**
   Defines spending limits for an account over a specified period (weekly, monthly, etc.). Used to monitor and enforce financial discipline and to trigger alerts when limits are approached or exceeded.
6. **Subscription**
   Represents recurring payments (like Netflix, utilities) tied to an account. Helps users track recurring expenses, manage due dates, and receive reminders before payments occur.

Task & Reward System

1. **Task**
   Represents actionable to-dos or habits assigned to users (like weekly budget check-ins). Tasks can be recurring or one-time, and their completion status contributes to user engagement and reward incentives.
2. **Reward**
   Represents points or incentives earned by users for completing tasks, staying within budget, or interacting with system features. Can be redeemed or used to motivate positive financial behavior.

Communication & Logging Entities

1. **Notification**
   Delivers system-generated messages to users, including reminders (like budget deadline), alerts (like low balance), or informational updates (like task completion). Time-stamped and categorized for clarity.
2. **LemonAidLogs**
   Logs each interaction between a user and the LemonAid AI assistant. Each log includes the AI's response text and a timestamp. User ratings and comments are now handled in the separate Reviews entity to support better modularity and review flexibility.

Support & Admin Interaction

1. **SupportTicket**
   Represents a help or issue report submitted by a user. Includes the subject, detailed message, ticket status, and any admin responses. Used to facilitate two-way support communication.

Integration & Association

1. **AccountBankLink**
   Represents the relationship between internal accounts and external banks. Enables multi-bank integration and prepares the system for open banking API compatibility. Supports syncing and aggregation of transaction data.
2. **Reviews**
   Captures user feedback in the form of ratings and messages. Reviews are linked either to LemonAid logs (chatbot type) or general user experience (user_experiance type). Supports analytics and system improvement initiatives.

User Privileges and Registration

- **Standard users can:**
    - Register, log in, and manage their accounts, tasks, budgets, and AI interactions.
    - View only their own data.
- **Admins can:**
    - View user accounts, support tickets, and analytics logs.
    - Respond to support tickets and manage platform-level settings.
- **Registration includes:** name, email, and password, with email verification planned in a future milestone.

Prioritized High-Level Functional Requirements

Priority 1

1. User

1.1. A user shall be able to create an account.

1.2. A user shall be able to securely log in to their account.

1.3. A user shall be able to recover or reset their password.

1.7. A user shall be associated with one or more accounts.

2. Account

2.1. An account shall store income entries for a user.

2.2. An account shall store expense entries for a user.

2.3. An account shall belong to one and only one user.

2.4. An account shall track a balance value.

2.5. An account may be connected to one or more banks.

2.7. An account shall have a transaction history log.

3. Bank

3.1. A bank shall be linked to one or more user accounts.

3.2. A bank shall provide transaction data to the system.

4. Transaction

4.1. A transaction shall be classified as income or expense.

4.2. A transaction shall belong to exactly one account.

4.3. A transaction may be manually entered or synced from a bank.

4.4. A transaction shall be categorizable by the user or AI.

5. Subscription

5.1. A subscription shall be tied to a recurring transaction.

6. Budget

6.1. A budget shall be created for one or more spending categories.

6.3. A budget shall allow setting and tracking of goals.

9. AI Assistant (LemonAid)

9.6. The system shall store AI responses and user feedback in LemonAidLogs.

10. Notification System

10.2. The system shall deliver daily, weekly, or monthly financial summaries.

13. ReimbursementRequest

13.1. A reimbursement request may be associated with one transaction.

13.2. A reimbursement request shall have a status: pending, approved, or rejected.

15. AccountBankLink

15.1. An account may be linked to one or more banks.

15.2. A bank may be linked to one or more accounts via AccountBankLink.

16. Bank

16.1. A bank shall store balance and credit data for users.

16.2. A bank shall provide transaction data to the system.

16.3. A bank shall be viewable in the user's linked accounts dashboard**.**

17. Reviews

17.1. A review shall be optionally submitted by a registered user.

17.2. A review shall include a rating value (1 to 5) and an optional message.

17.3. A review shall be timestamped at the time of creation.

17.4. A review may be linked to one and only one LemonAidLog.

17.5. A review may exist independently as general user feedback not tied to any LemonAidLog.

17.6. A LemonAidLog shall be associated with at most one review.

17.7. A review shall be categorized by type (chatbot or user_experiance).

Priority 2

1. User

1.4. A user shall be able to select between manual or bank-linked financial tracking.

1.15. A user that is an administrator shall be able to respond to support tickets.

1.16. A user that is administrator shall be able to access user account data.

2. Account

2.6. An account shall be able to categorize transactions.

2.8. An account shall support spending limits and alerts.

2.9. An account shall track subscriptions and bill payments.

2.10. An account shall be able to forecast end-of-month balances.

3. Bank

3.3. A bank shall store balance and credit data for users.

4. Transaction

4.7. A transaction may be associated with a reimbursement request.

5. Subscription

5.2. A subscription shall be viewable in a centralized dashboard.

5.3. A subscription shall trigger periodic spending alerts.

6. Budget

6.2. A budget shall belong to one and only one account.

6.4. A budget shall support notifications based on spending limits.

6.5. A budget shall be able to generate monthly recaps.

10. Notification System

10.3. The system shall notify users of upcoming bills or subscriptions.

10.4. The system shall notify users of overspending events.

10.5. The system shall alert users when savings goals are off track.

Priority 3

1. User

1.5. A user shall be able to customize their interface (like dark mode).

1.6. A user shall be able to receive AI-generated financial guidance.

1.8. A user shall be classified as a customer, premium, tester, or administrator.

1.9. A user shall be able to interact with the AI chatbot for financial assistance.

1.10. A user shall be able to view and edit their financial data history.

1.11. A user that is an administrator shall be able to suspend or maintain site operations.

11.12. A user that is an administrator shall be able to view flagged transactions.

1.13. A user that is an administrator shall be able to suspend or maintain site operations.

1.14. A user that is an administrator shall be able to view flagged transactions.

4. Transaction

4.5. A transaction shall be editable and deletable by the user.

4.6. A transaction shall be linked to one or more receipts.

7. Task

7.1. A task shall be assigned to a user account.

7.2. A task shall be marked completed upon user action.

7.3. A task shall be of one type: savings, investing, setup, admin, testing.

7.4. A recommended task shall be generated based on user activity or AI analysis.

8. Reward System

8.1. A user shall be able to earn rewards for completing tasks.

8.2. A user shall be able to redeem rewards through the system.

8.3. A user shall be assigned a level based on completed actions.

8.4. A user shall be able to view rankings if competitive mode is enabled.

9. AI Assistant (LemonAid)

9.1. The AI assistant shall analyze user transactions and spending habits.

9.2. The AI assistant shall simulate different financial scenarios for planning.

9.3. The AI assistant shall forecast recurring charges and balances.

9.4. The AI assistant shall recommend saving opportunities and budget changes.

9.5. The AI assistant shall provide suggestions for uncategorized transactions.

10. Notification System

10.1. The system shall send reminders for logging receipts.

11. Administrator

11.5. An administrator shall be able to manage and moderate platform use.

12. Receipt

12.1. A receipt may be linked to a transaction.

12.2. A receipt may be stored as a binary image (JPG/PNG, max 5MB).P2

12.3. A receipt shall include the upload date.

12.4. The system shall send reminders for users to upload receipts.

12.5. An administrator shall be able to access user account data.

13. ReimbursementRequest

13.3. A reimbursement request shall be visible to administrators for review.

14. SupportTicket

14.1. A support ticket shall be created by a user.

14.2. A support ticket shall include a subject, message, and status.

14.3. A support ticket shall be responded to by an administrator.

14.4. A support ticket shall track the admin response and resolution state.

15. AccountBankLink

15.3. The system shall sync data from bank APIs using these links.

17. Reviews

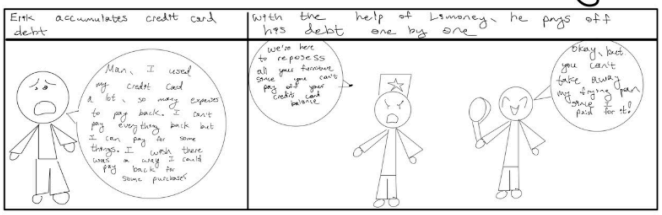
17.8. A review shall be viewable by system administrators for quality assurance and analysis.

17.9. A review shall be stored securely and associated with the submitting user, if applicable.

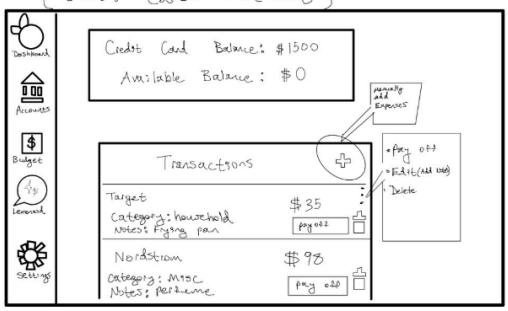
17.10. A review shall support analytics use cases for measuring system effectiveness and user satisfaction.

Mockups/Storyboards

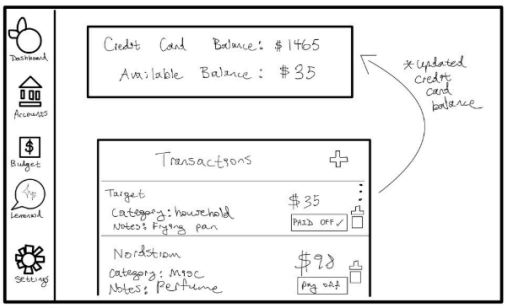
Create Budget (Checking Credit Card expenses)



(Credit Card Account)



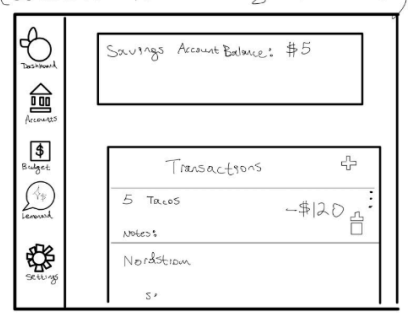
1. Erik goes to a specific Credit Card Transaction. The pay off buttons on each transaction indicate which expenses haven't been paid off



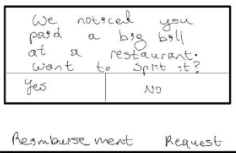
2. Next, Erik pays off one expense, so the transaction gets marked as "paid off". Then the credit card balance is updated

Reimbursement Requests

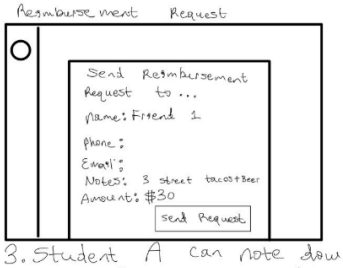
(Student A's Savings Account)



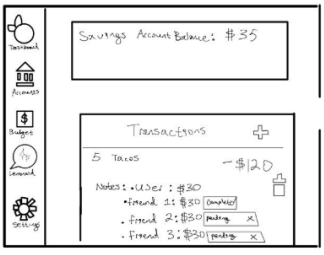
1. Student A opens up the transaction for the restaurant bill



2. He also receives a notification for large bills over $100. He is prompted to split the bill



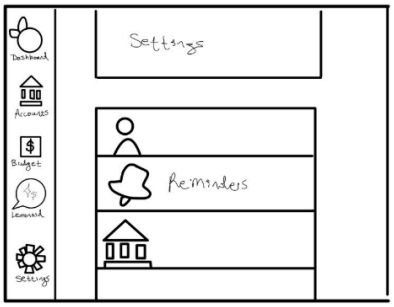
3. Student A can note down how much money each of friends owe him and send them reimbursement requests



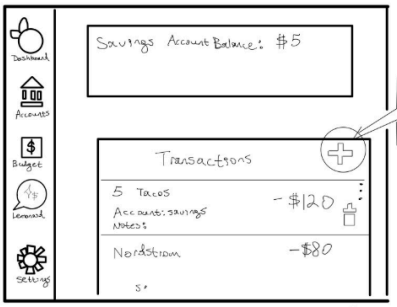
3. When the friend(s) pay(s) back Student A, Student A can manually mark the request as completed and the account balance will be updated. Incomplete reimbursement requests will be marked as "pending".

Log Transactions

Account Section



Or



Reminder for web app

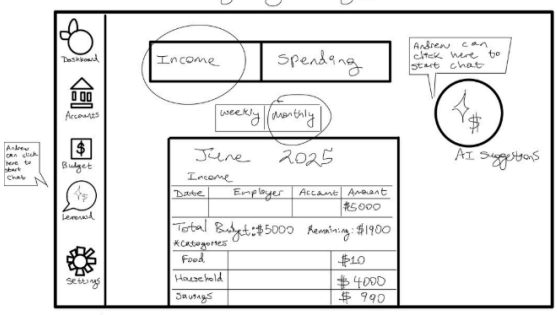
1. Erik can set reminders via settings or the account transaction dashboard

2. These are the reminders to log in receipts sent to Erik exclusively from Lemoney. The scan feature will use a camera, scan photo of Erik's receipt, and auto fill the data fields such as

---

Security Concern

Budgeting page



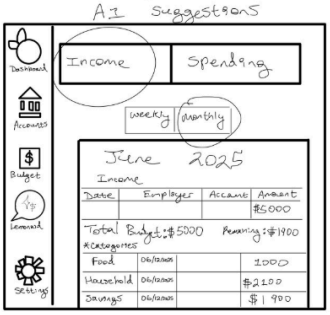
1. Andrew opens budget page where he can manually enter his monthly or weekly income in the income section

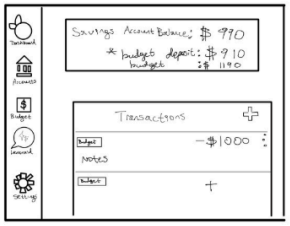
2. Andrew, being a premium user, can access the Lemonaid chatbot through the budgeting page or nav bar

3.

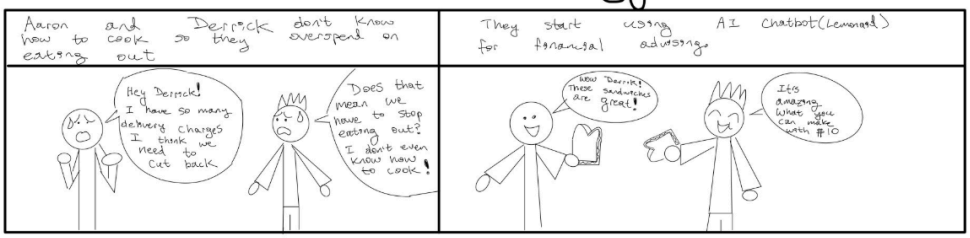
Modified Budgeting page

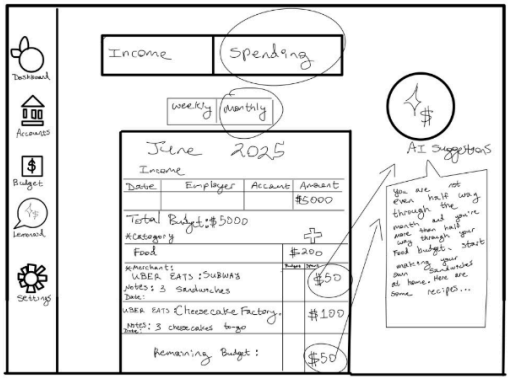
AI Suggestions





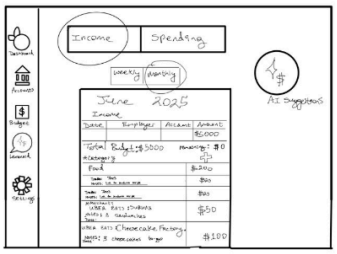
Receive AI suggestions





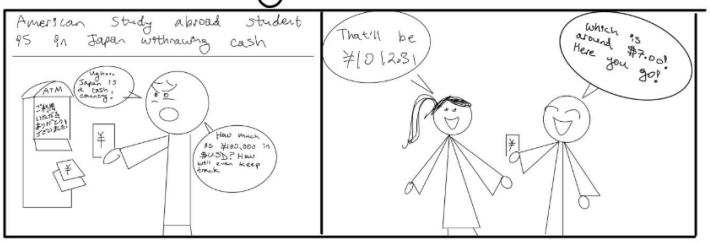
1. This is the "Spending" section of the budget, subtracting the budget total from the income section with transactions from account. The AI warns him he's overspending and the visual page helps him realize

2. Aaron consults with the Lemonaid chatbot, asking how he and Derrick can stick to their budget.

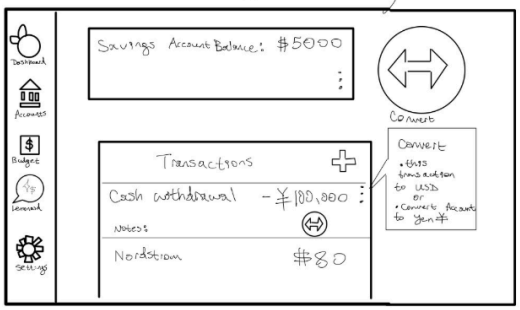


3. The AI chatbot updates the income section of the budget to help Aaron plan his food expenses for the rest of the month.

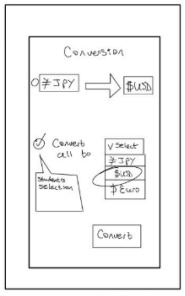
Currency Exchange



(Student A's Savings Account)

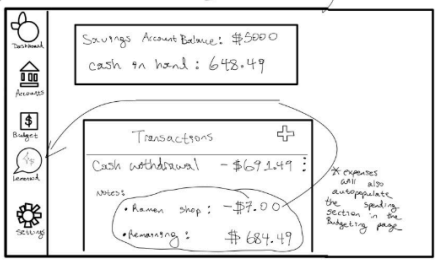


1. American Study Abroad student manually logs in the ATM cash withdrawal in Yen. He has three buttons to help him make currency conversions: the arrow button on the transaction itself (usually triggered only if transaction is in foreign currency), the big "Convert" button on the right, and the drop down menu.



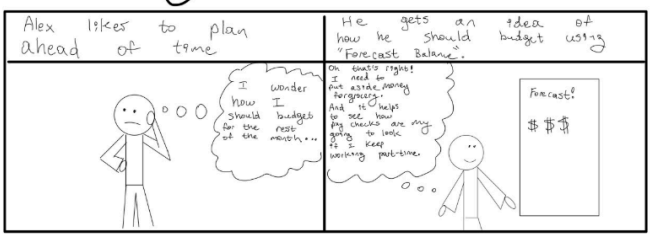
2. The student has the option of converting all transactions/expenses into $USD or converting just that one expense into $USD.

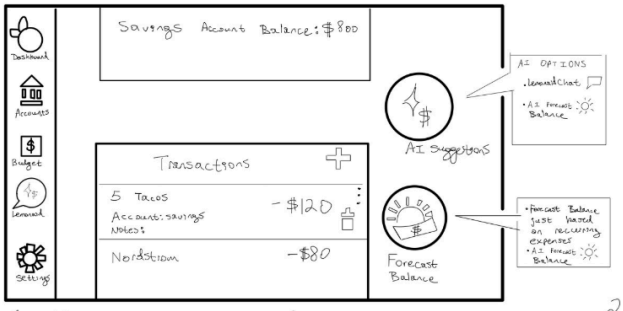
(Student A's Savings Account)



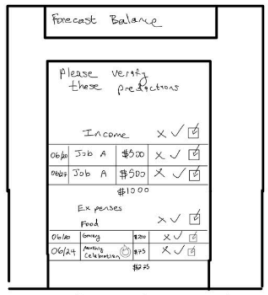
3. The student converts that cash withdrawal into $USD and can log in how he spent that cash which will also auto-populate his budgeting page.

Predicting Balances End of Month



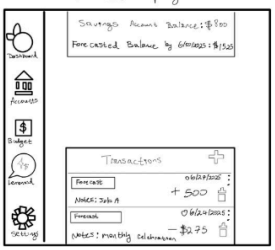


1. Alex goes to his Savings account. His balance sits at $800 at the beginning of the month but he wants to know what his balance will look like at the end of the month given his upcoming expenses and paychecks (income)



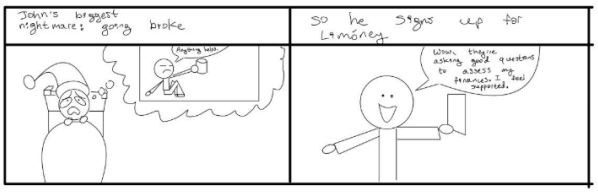
2. The Forecast presents Alex options to accept, reject, or edit for his bank balances and budget.

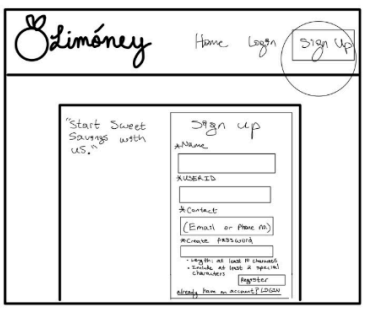
Modified Budgeting page and Accounts pages



3. Alex's bank balance is not updated, rather the upcoming expenses are displayed as "forecast" and there's a calculated balance for the end of the month seperate from the current balance.

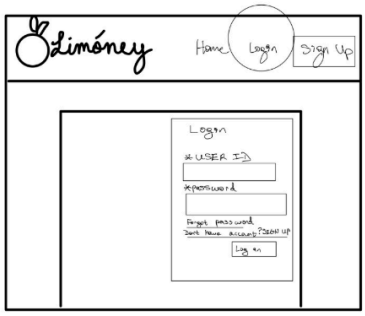
Signing up / Intro UI



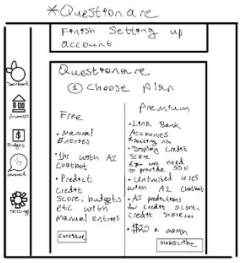


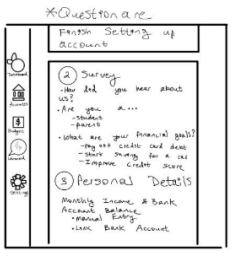
1. John goes to Sign up page.

or

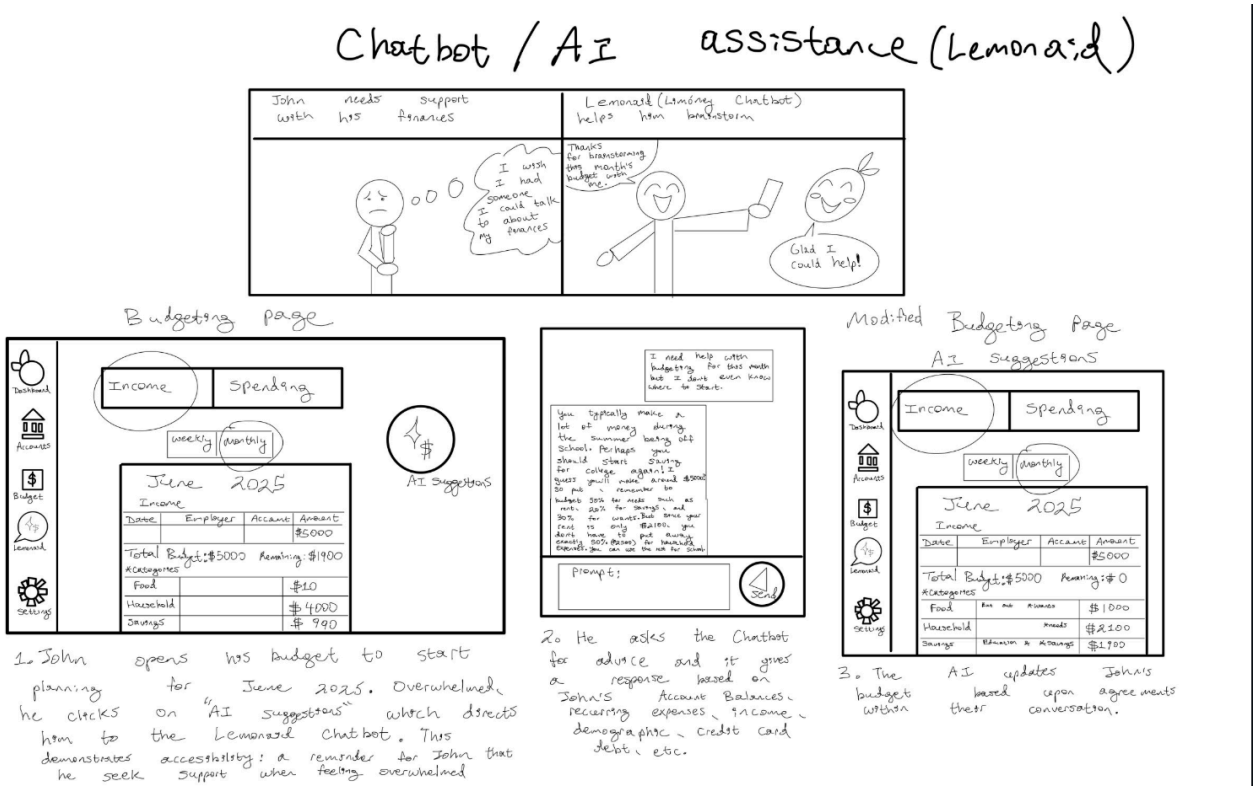


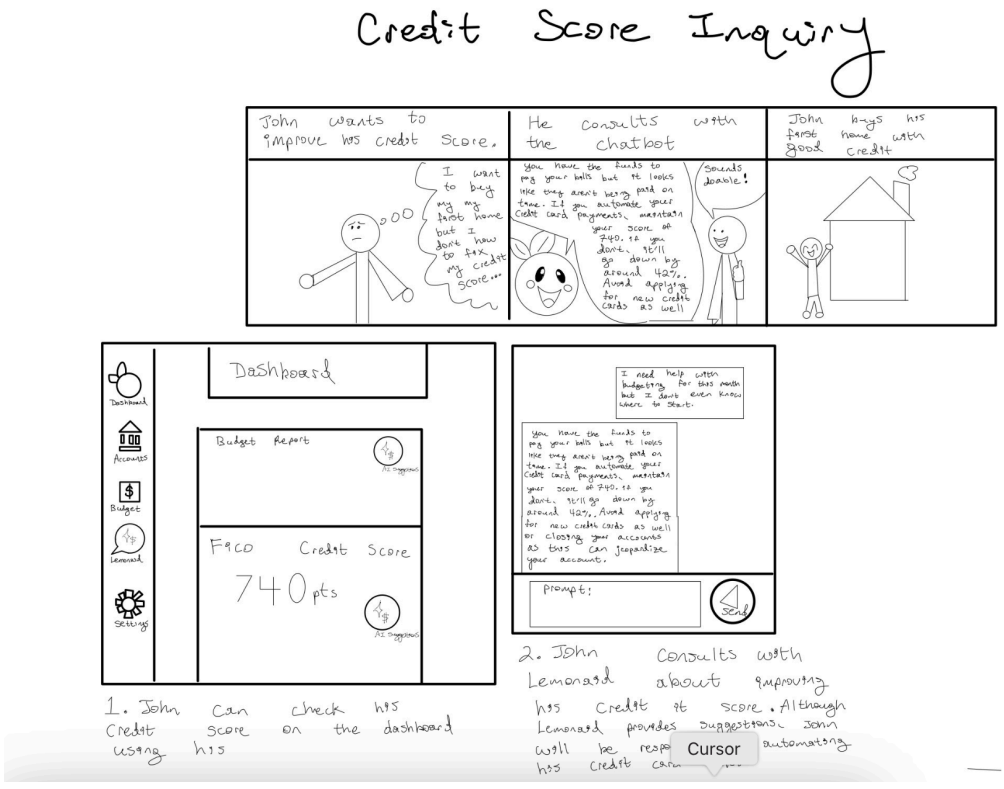
1. John went to sign up account and was already told he has an account so he logs in.





2. John chooses to be a premium user and answers the questionair so the Limoney webapp can give him a personalized experience based on his





High-Level System Design

High Level Database Architecture

Initial Database Requirements

- **User**
    - A User shall be associated with 0 to many Accounts.
    - A User shall be able to create and manage 0 to many Tasks.
    - A User shall be rewarded with 0 to many Rewards.
    - A User shall receive 0 to many Notifications.
    - A User shall be able to submit 0 to many SupportTickets.
    - A User shall have 0 to many LemonAidLogs.
    - A User can be a subtype Administrator.
    - A User may receive responses from users that are admin via associated SupportTickets.
    - A User shall be able to submit 0 to many Reviews.

- **Account**
    - An Account shall belong to exactly 1 User.
    - An Account shall be associated with 0 to many Transactions.
    - An Account shall be linked to 0 to many Budgets.
    - An Account shall be linked to 0 to many Subscriptions.
    - An Account shall be associated with 0 to many Banks through AccountBankLink.
- **Bank**
    - A Bank shall be linked to 0 to many Accounts via AccountBankLink.
    - A Bank may serve multiple Users through its linked Accounts.

- **Transaction**
    - A Transaction shall belong to exactly 1 Account.
    - A Transaction shall have 0 or 1 associated Receipt.
    - A Transaction shall be associated with 0 or 1 ReimbursementRequest.
- **Receipt**
    - A Receipt shall be associated with exactly 1 Transaction.
- **ReimbursementRequest**
    - A ReimbursementRequest shall be linked to exactly 1 Transaction.
- **Subscription**
    - A Subscription shall belong to exactly 1 Account.
- **Budget**
    - A Budget shall belong to exactly 1 Account.
- **Task**
    - A Task shall be assigned to exactly 1 User.
- **Reward**
    - A Reward shall be associated with exactly 1 User.
- **Notification**
    - A Notification shall be sent to exactly 1 User.
- **LemonAidLogs**
    - A LemonAidLog shall be created by exactly 1 User.
- **SupportTicket**
    - A SupportTicket shall be created by exactly 1 User.

    - A SupportTicket may be responded to by 0 or 1 Users that are admin.

- **Reviews**
    - A Review shall be submitted by 0 or 1 Users.
    - A Review may be associated with 0 or 1 LemonAidLogs.
    - A Review shall include a required rating and message.
    - A Review shall be classified by type (e.g., chatbot or user experience).
    - A Review shall automatically store the timestamp of submission.

DBMS Selection

We will use **MySQL** because it is a popular, reliable relational database that integrates seamlessly with Node.js/Express, handles financial and user data easily, and is approved by our CTO.

Database Organization

User (strong)

Attributes:
Represents individual system participants. Stores identity info, authentication credentials, and role classification.

Relationships:

- 1 to 0…* with **Account**
- 1 to 0…* with **Tasks**
- 1 to 0…* with **Reward**
- 1 to 0…* with **Notifications**
- 1 to 0…* with **SupportTicket**
- 1 to 0…* with **LemonAidLogs**

Domains:

- Text (names, emails) with format validation.
- Encrypted strings for passwords.
- Predefined roles (like standard, admin).
- IDs as integers, unique per user.

Account (weak)

Attributes:
Stores a user's financial container, such as checking, savings, or credit. Tracks balance and account type.

Relationships:

- 0…* to 1 with **User**
- 1 to 0…* with **Transaction**
- 1 to 0…* with **Budget**
- 1 to 0…* with **Subscription**
- 0…* to 0…* with **Bank** (via AccountBankLink)

Domains:

- Account types (like checking, savings, credit) via ENUM.
- Balances in integer cents to avoid float errors.
- IDs as integers.

Bank (strong)

Attributes:
Represents financial institutions users can link to their accounts.

Relationships:

- 0…* to 0…* with **Account** (via AccountBankLink)

Domains:

- Short text for bank names.
- Unique integer IDs.

Transaction (weak)

Attributes:
Tracks money movements on accounts, including income, expenses, and transfers. Also includes category, date, and amount.

Relationships:

- 0…* to 1 with **Account**
- 1 to 0…1 with **Receipt**
- 1 to 0…1 with **ReimbursementRequest**

Domains:

- Transaction types (income, expense, transfer) via ENUM.
- Categories as short text (user/system defined).
- Amounts in integer cents.
- Dates in standard DATE format.
- IDs as integers.

Receipt (weak)

Attributes:
Stores uploaded receipt images tied to transactions.

Relationships:

- 0…1 to 1 with **Transaction**

Domains:

- Binary files (JPG/PNG only, max ~5MB).
- Upload date as DATE.
- Foreign key reference to transaction.

ReimbursementRequest (weak)

Attributes:
Represents refund requests linked to transactions, including workflow state.

Relationships:

- 0…1 to 1 with **Transaction**

Domains:

- ENUM for status (pending, approved, rejected).
- Reference to associated transaction.

Subscription (weak)

Attributes:
Recurring charges linked to accounts. Includes name, interval, and next billing date.

Relationships:

- 0…* to 1 with **Account**

Domains:

- Service names as short text (e.g., "Netflix").
- Amounts in cents.
- ENUM interval (daily, weekly, monthly, yearly).
- Dates as DATE.

Budget (weak)

Attributes:
Defines spending limits over a given time window for an account.

Relationships:

- 0…* to 1 with **Account**

Domains:

- Limit amounts in cents.
- Start/end dates as DATE.
- Integer IDs.

Task (weak)

Attributes:
User-assigned or system-generated to-dos, with status and type (e.g., recurring, custom).

Relationships:

- 0…* to 1 with **User**

Domains:

- ENUM types (daily, weekly, one-timer, custom).
- ENUM status (pending, completed, overdue).
- Integer IDs.

Reward (weak)

Attributes:
Points awarded to users for engagement or successful financial behavior.

Relationships:

- 0…* to 1 with **User**

Domains:

- Point totals as integers.

- IDs as integers.

Notification (weak)

Attributes:
System messages for users, such as alerts, updates, and reminders.

Relationships:

- 0…* to 1 with **User**

Domains:

- ENUM types (info, alert, reminder).
- Message content as text.
- Timestamp as DATETIME.

LemonAidLogs (weak)

Attributes:
Interaction logs between the user and AI assistant.

Relationships:

- 0…* to 1 with **User**

Domains:

- AI output as text.
- Timestamp as TIMESTAMP.

SupportTicket (weak)

Attributes:
User-submitted help requests. Includes issue description, status, and optional admin reply.

Relationships:

- 0…* to 1 with **User**
- 0…* to 1 with Administrator **(User)**

Domains:

- Subject/message as text.
- Status ENUM (open, in progress, closed).
- Optional response as text.
- Integer IDs.

Reviews (strong)

Attributes:
User or non user submitted feedback on either the chatbot ai or the user experience.

Relationships:

- 0…* to 0…1 with **User**
- 0…1 to 0…1 with **LemonAidLogs**

Domains:

- Integer IDs
- Feedback as text.
- Possible foreign keys from User and LemonAidLogs.
- Rating (1-5).

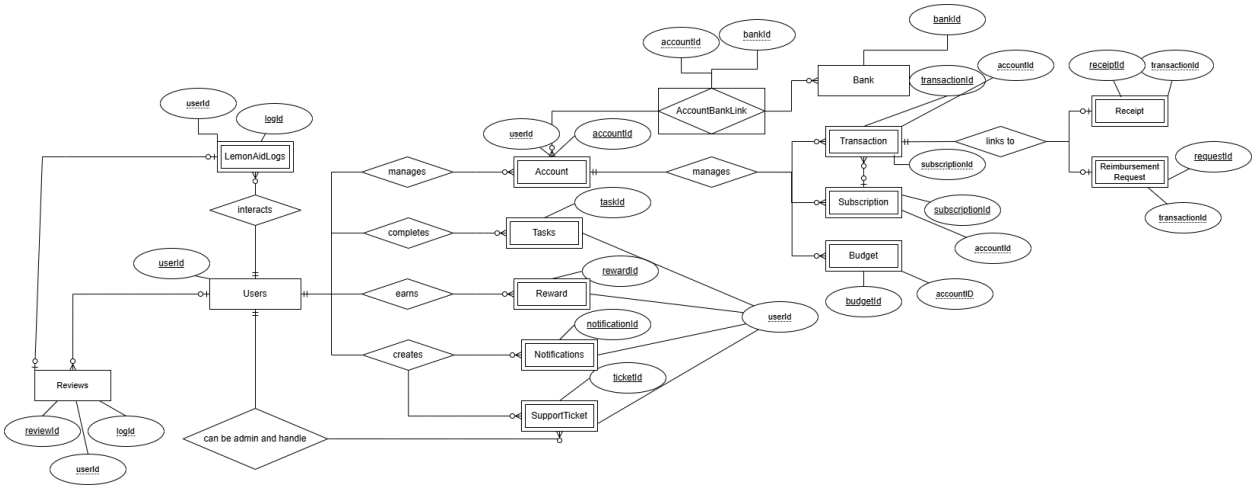
AccountBankLink (associative)

Attributes:
Connects user accounts to external banks for multi-bank linking.

Purpose:
Resolves many-to-many between **Account** and **Bank**

Domains:

- Purely foreign key based.
- No extra data beyond IDs referencing accounts and banks.



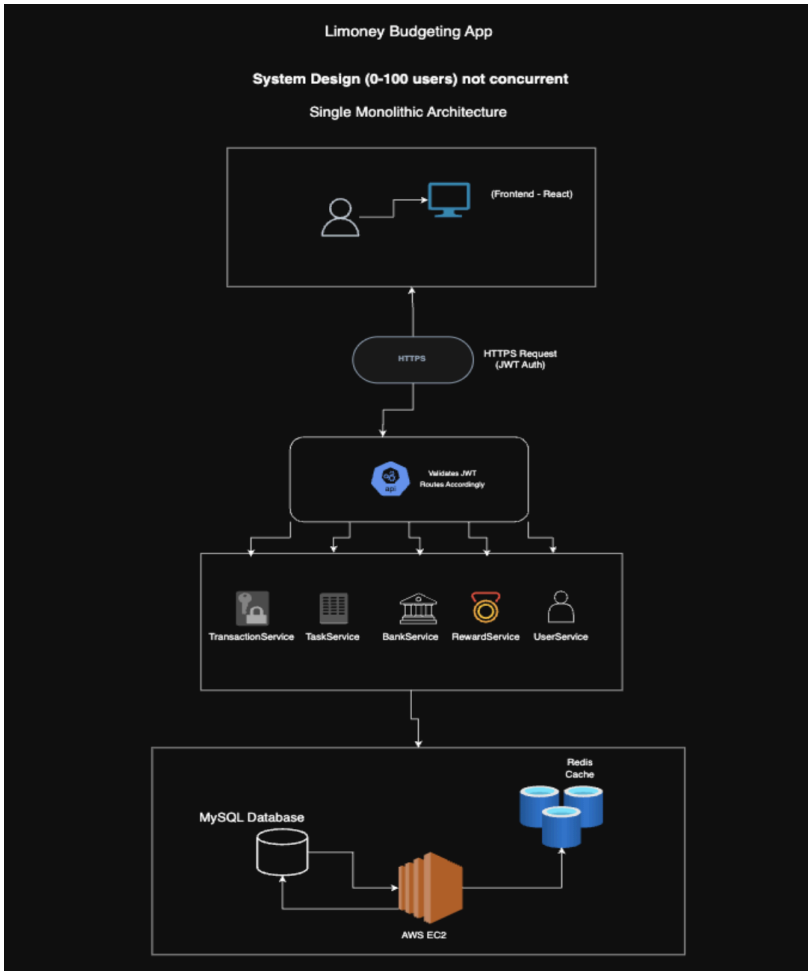
Media Storage

For now, we plan to store images as BLOBs for things like receipts, since they are small and directly tied to transactions. We do not currently need to store video, audio, or GPS data, but if needed in the future, we may switch to a file system to handle larger files more efficiently.

Backend Architecture

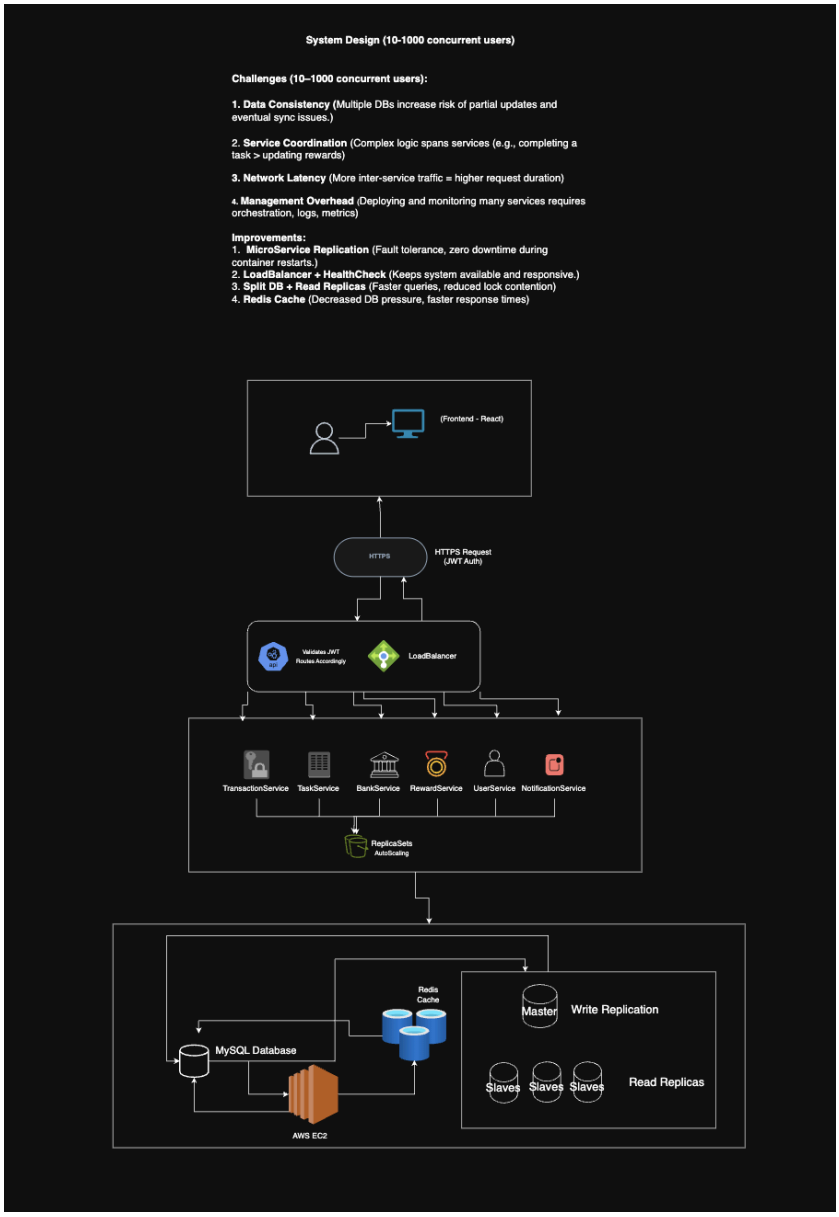
ALL SCALE DESIGN

The Limoney Budgeting App for 0-100 (not concurrent) users uses a single monolithic architecture. The frontend (React) communicates via HTTPS (JWT Auth) to a backend API, which validates JWT and routes requests to services such as TransactionService, TaskService, BankService, RewardService, and UserService. All services interact with a single MySQL database (hosted on AWS EC2) and an optional Redis cache for session/AI caching.



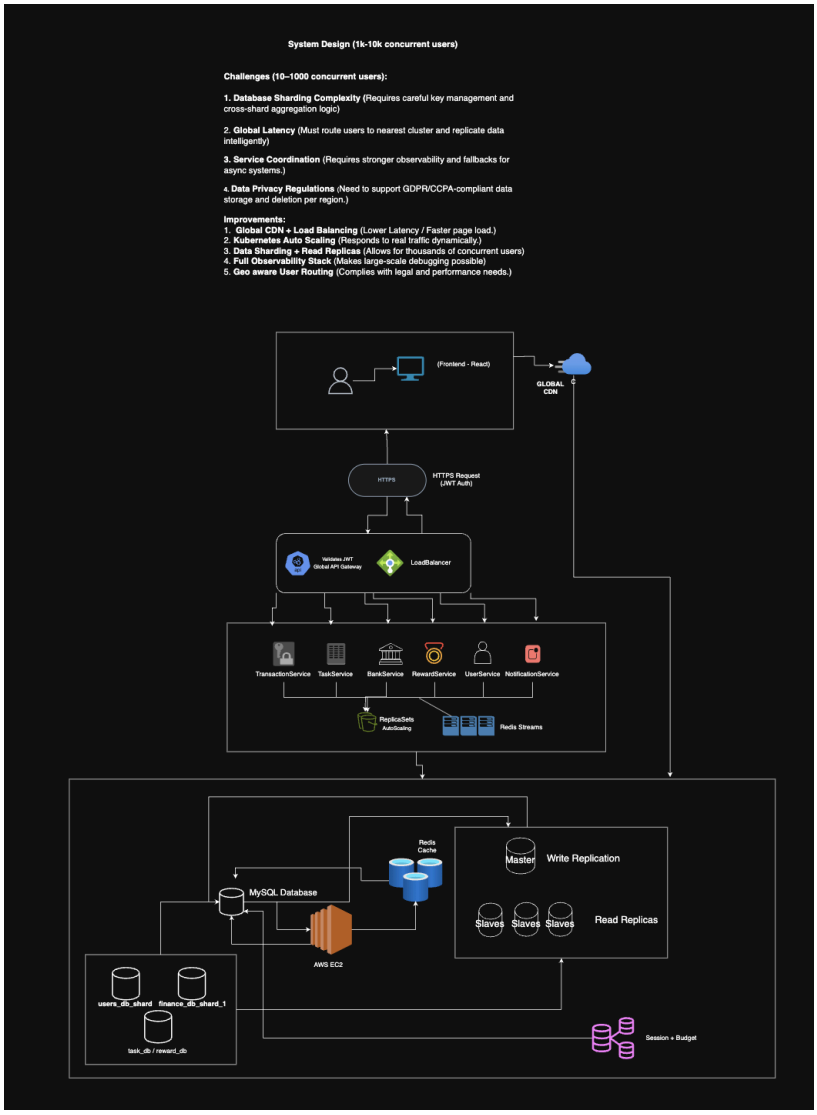
MEDIUM SCALE DESIGN

For 10-1000 concurrent users, the system introduces microservice replication, load balancing, and read replicas for the database. The backend is horizontally scalable with Dockerized microservices, an auto-scaling NGINX load balancer, and health checks. The MySQL database uses master-read replica architecture for scalability, and Redis is used for caching AI and frequent queries. Improvements include fault tolerance, health checks, and reduced DB pressure.



LARGE SCALE DESIGN

For 1k-10k concurrent users, the system leverages Kubernetes-managed microservices, global CDN, and database sharding by user region. The backend is orchestrated with Kubernetes for auto-scaling and CI/CD. The database layer uses sharding and a mix of read/write replicas, with multi-layer Redis caching. Security is enhanced with rate-limiting, API key enforcement, and mutual TLS between services. The architecture supports global latency reduction and compliance with data privacy regulations.



ARCHITECTURE SUMMARY

Microservices Architecture

- Domain-driven microservices: each major business capability is its own independently deployable service
    - Services: UserService, BankService, BudgetService, SubscriptionService, TaskService, AIRecommendationService, RewardService
- Service communication:
    - Synchronous calls via REST APIs

- Asynchronous messaging for AI triggers and reward events

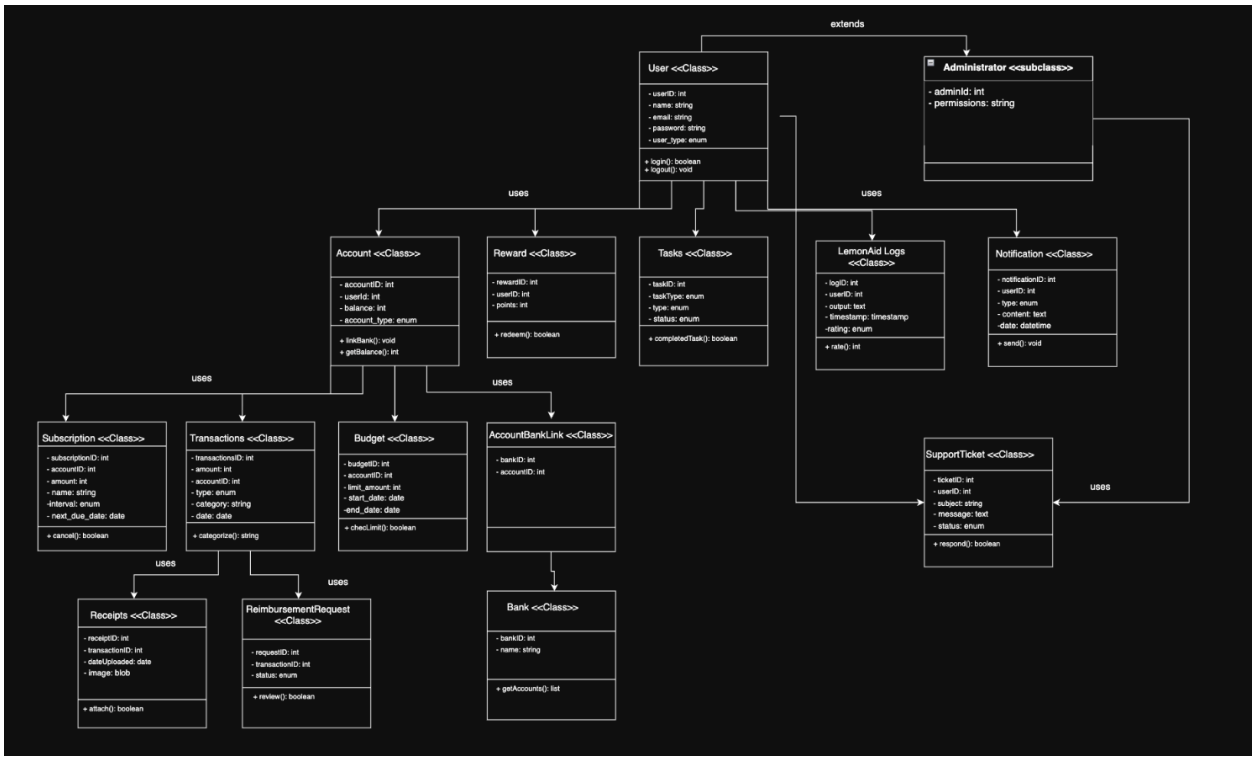
Backend System Design

- **Load Balancing**
  - NGINX reverse proxy or AWS ALB to evenly route traffic
  - Health checks to verify service availability
  - Supports horizontal scaling for better availability and responsiveness
- **Caching Strategies**
  - Cache-aside pattern for database query results (e.g., user budget history)
  - LRU eviction policy to optimize memory usage
- **Reliability & Fault Tolerance**
  - Each service runs in its own Docker container managed by Kubernetes
  - Circuit breakers (Resilience4j) to prevent cascading failures
  - Retry and timeout policies to stabilize inter-service connections
- **Containers & Orchestration**
  - Docker containers ensure portable environments (local → staging → production)
  - Kubernetes handles deployment, health checks, auto-scaling, and rolling updates
- **Data Replication & Consistency**
  - Master-Slave replication for MySQL to boost read throughput
  - Synchronous writes for critical services (e.g., TransactionService, SavingsService)
  - Eventual consistency acceptable for non-critical operations to improve performance
- **Security Considerations**

- JWT authentication for all incoming requests
- Role-Based Access Control (RBAC) distinguishing Admins vs. Customers
- Mutual TLS for service-to-service communication
- Rate limiting and request logging at the API gateway
- Data encryption: AES-256 at rest and TLS in transit

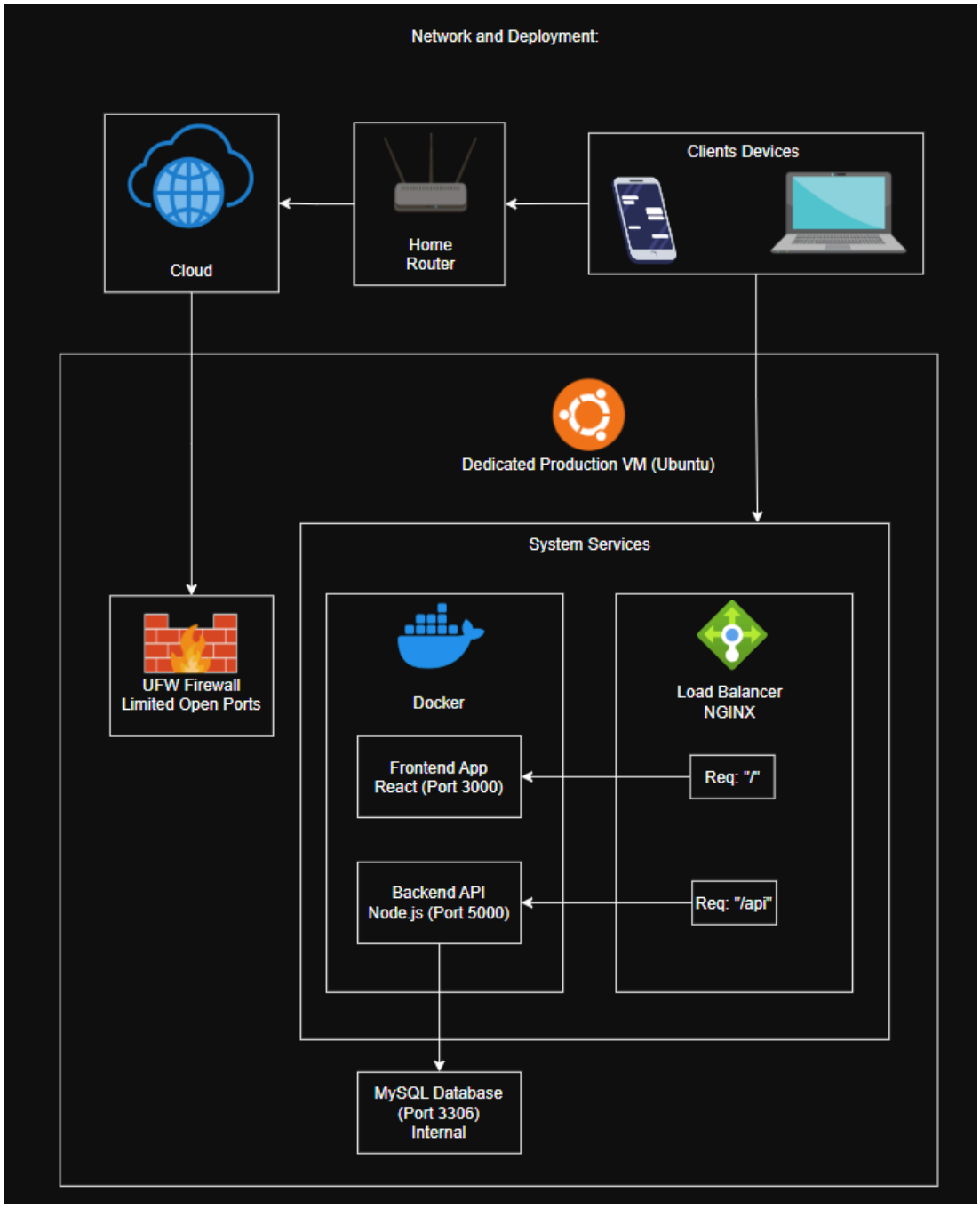
UML Design

- The following UML diagram illustrates the key domain classes and service interfaces for the Limóney backend system.



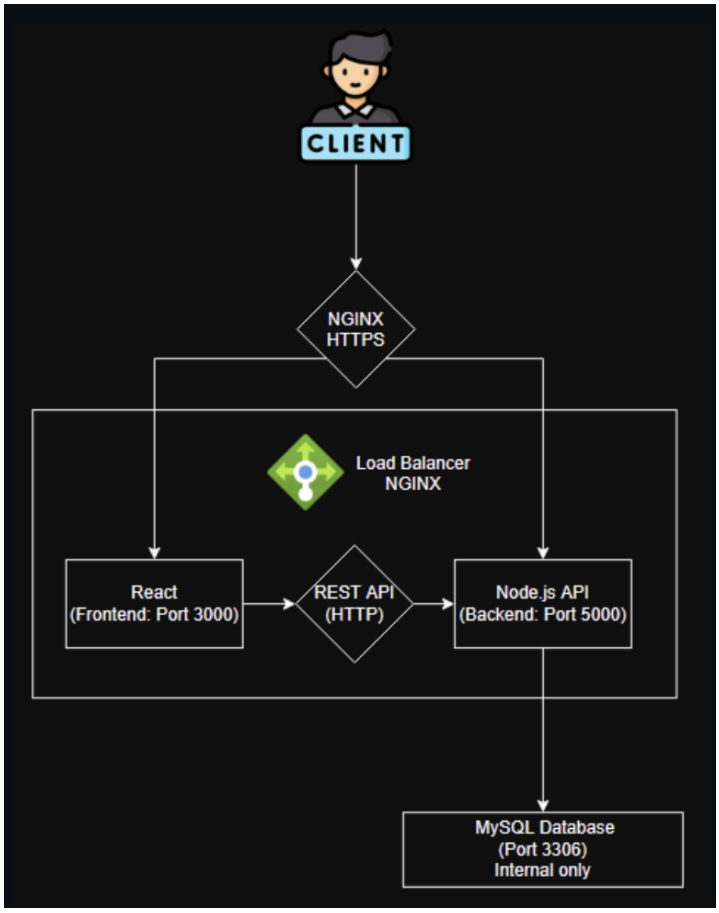
High Level Application Network Protocols and Deployment Design

Network & Development Diagram



This diagram shows a secure full-stack web app deployment. Users connect with HTTPS to a cloud Virtual Machine protected by a UFW firewall. NGINX routes traffic to Dockerized React frontend and Node.js backend services, which access a MySQL database internally.

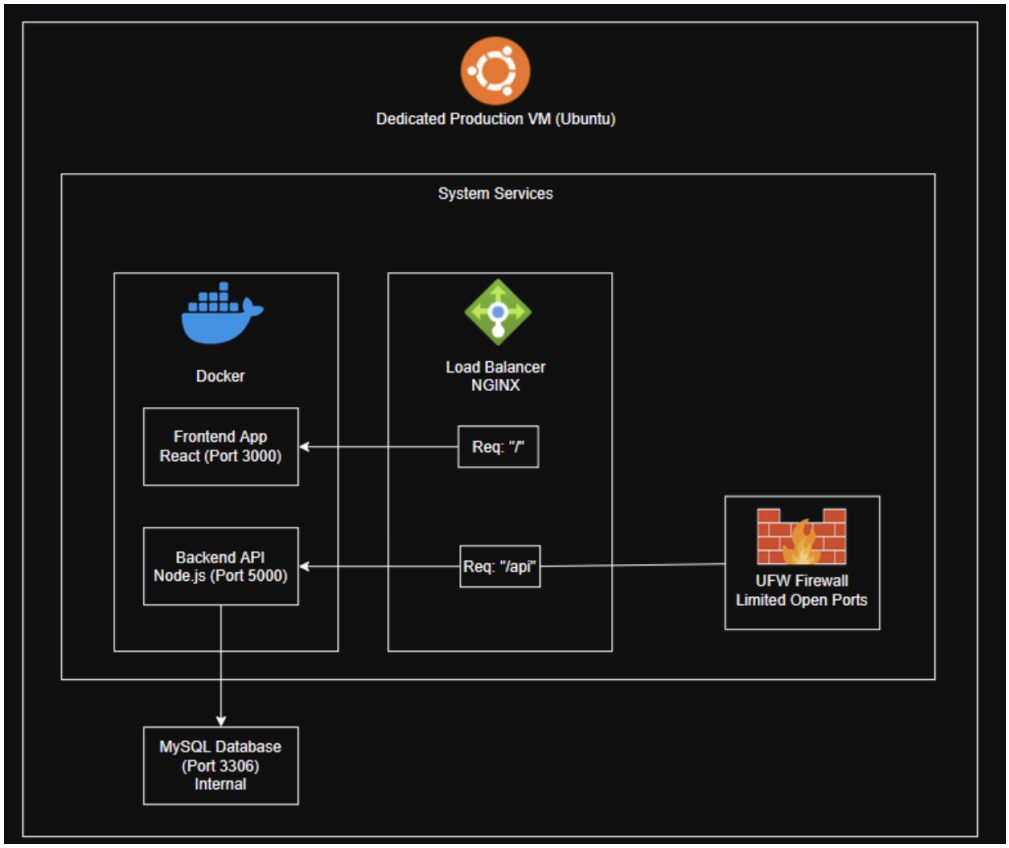
Application Networks Diagram



Protocols

Network uses HTTPS for securing client access with NGINX, routing to React frontend and Node.Js backed. The backend accesses MYSQL database internally over TCP/IP. Services run on Docker bridge network, with security enforced through SSL/TLS, CORS. and a UFW firewall that restricts access only to essential ports to the application to work. NGINX is the main gateway and proxy which isolates the internal services from the public.

Deployment Diagram



This diagram illustrates the system's structure on a dedicated Ubuntu server. Docker manages the React frontend and Node.js backend processes. NGINX handles incoming traffic and routes it to the appropriate service. A UFW firewall restricts network access, allowing only essential ports. The backend securely connects to an internal or cloud-hosted MySQL database.

Integration with External Components

External Components:

- No third party API's so far

Internal Libraries:

- Bycrypt: for hashing passwords
- Dotenv: for ".env" variable handling
- Cors: for securing cross-origin request
- Mysql12: for database connection

High Level APIs and Main Algorithms

High-Level APIs

- **Account & Transaction Management API**
    - Allows users to securely manage accounts, log income/expenses, categorize transactions, and attach receipts.
- **Budget & Subscription API**
    - Helps users create, update, and track budgets and recurring payments with due-date reminders.
- **Task & Reward System API**
    - Supports gamification by letting users complete tasks and earn reward points based on activity.
- **Notification API**
    - Delivers alerts, reminders, and system messages tied to user activity and financial goals.
- **Support & Admin API**
    - Enables users to submit support tickets and allows admins to manage and respond to them.
- **LemonAid AI Interaction API**
    - Logs interactions between users and the LemonAid assistant and allows users to rate each AI response.

Main Algorithms and Processes

AI Rating System: After each LemonAid interaction, users rate the response on a scale of 1–5. These ratings will track AI quality over time and may later feed into an admin review process or an AI fine-tuning feedback loop.

Spending Categorization: Transactions are initially auto-labeled via simple keyword matching, with a roadmap to evolve into a smarter, pattern-based classification engine.

Budget Alerts: The system runs periodic checks of current spending against set budget limits and sends notifications when users approach or exceed those thresholds.

Reward Calculation: Points are awarded for completing financial tasks or goals; the calculation factors in task type, frequency, and any completion streaks to drive user engagement.

Software Tools and Frameworks

Current Technology Stack

- Node.js + Express (backend API framework)
- MySQL (relational database)
- React (or plain HTML/CSS/JavaScript) for the frontend
- Docker (for containerized deployment)
- AWS EC2 running Ubuntu 22.04 (cloud hosting)
- Let’s Encrypt + Certbot (SSL certificates)

Planned Integrations

- Firebase Authentication (optional, for user auth & password resets)
- OpenAI API (for the LemonAid assistant)
- Tesseract OCR or Google Vision API (automatic receipt scanning)
- *Any new tools will be approved by the instructor before implementation.*

Key Project Risks

Skills Risks

- Team members each have strengths in different areas (documentation, prototyping, etc.) but share a strong desire to learn.
- If someone encounters an unfamiliar task, they'll proactively consult multiple sources (peers, tutorials, official docs).

Schedule Risks

- All members balance outside responsibilities, limiting available hours.
- Mitigation: detailed milestone planning, shared calendars, and regular check-ins to surface conflicts early.

Technical Risks

- Our cloud host has very limited vCPU and RAM, leading to past crashes (e.g., "About Us" page overload).
- Mitigation:
    - Complete early prototypes to allow backend optimization
    - Continuously monitor CPU/memory metrics
    - Evaluate lightweight frameworks or alternative hosting if needed

Teamwork Risks

- Progress stalls if members struggle in silence.
- Mitigation:
    - Brief daily/bi-weekly stand-ups for status and blockers

    - “Struggling?” channel in team chat for immediate help requests
    - Pair programming on complex features

Legal/Content Risks

- Third-party UI kits, AI tools (e.g., OpenAI), and APIs require proper licensing for commercial use.
- Mitigation:
    - Verify and document licenses for all dependencies
    - Budget for any required paid service plans
    - Display clear disclaimers that the AI assistant offers suggestions only, not professional advice

---

Project Management

In Milestone 2, project management started off rough because we had a set up on Notion and then the free trial expired faster than expected. After that, we quickly moved over to Gitbook since we were familiar with it due to the technical documentation. However, the free trial expired and made it excessively inconvenient to use it for project management. From then, the team lead stayed active with making sure everyone was on top of things by messaging them for updates. For milestone 3, we will attempt at using Jira (very similar to Notion), which is another project management tool. If the team lead read the free trial rules correctly, it should be able to last until the summer semester is over.

List of Team Contributions

Name	Contributions	Score (1-10)
Emily Perez	wrote key project risks; wrote title page; finalized high-level functional requirements; organized and finalized technical documentation; assisted with setting up the directory; helped establish connection between frontend, backend, and database	6
Ishaank Zalpuri	wrote data definitions; helped with database architecture; helped with high-level functional requirements	4.5
Andrew Brockenborough	created table of contents; updated readme.md file in application folder; helped with high level apis and main algorithms; helped with high-level functional requirements	2
Dani Luna	created todo list; wrote data definitions; helped with database architecture; created mockups/storyboards; worked on the search bar algorithm for m2c2; helped with high-level functional requirements	5
Jonathan Gonzalez	helped with high-level functional requirements; worked on backend architecture	3
Gene Orias	establish connection between frontend, backend, and database; completed the signup aspect of m2c2; updated credentials files; encrypted server data into code (.env); worked on high-level application network protocols and deployment design; helped with high-level functional requirements	8

Milestone 3

SW Engineering CSC648-848-01 Summer 2025

Limóney (Financial Budgeting Webapp)

Team 02

Milestone 3

7/31/25

Name	**Role**
Emily Perez (eperez@sfsu.edu)	Team Lead, Database Administrator, Github Master, UX Specialist, Quality Assurance
Ishaank Zalpuri	Frontend Lead, UI Designer, Frontend Developer
Andrew Brockenborough	Technical Writer
Dani Luna	Frontend Lead, UI Designer, Frontend Developer
Jonathan Gonzalez	Software Architect, Frontend Lead, UX Specialist,
Gene Orias	Backend Lead, Frontend Developer

Milestone	**Version**	**Date**
Milestone 3	Version 2	7/31/25
Milestone 3	Version 1	7/22/25
Milestone 2	Version 2	7/10/25
Milestone 2	Version 1	7/3/25
Milestone 1	Version 2	7/2/25
Milestone 1	Version 1	6/17/25

Table of Contents

Table of Contents **2**
Data Definitions **3**
Core User & Identity Entities 3
Financial Management Entities 3
Task & Reward System 3
Communication & Logging Entities 4
Support & Admin Interaction 4
Integration & Association 4
Prioritized High-Level Functional Requirements **5**
UI/UX Wireframes **11**
High-Level System Design **17**
High Level Database Architecture 17
Initial Database Requirements 17
DBMS Selection 19
Database Organization 19
Entity Relationship Diagram: 27
Extended Entity Relationship Diagram: 27
Backend Architecture **27**
ALL SCALE DESIGN 28
MEDIUM SCALE DESIGN 28
LARGE SCALE DESIGN 29
ARCHITECTURE SUMMARY 30
Backend System Design 31
UML Design 32
High Level Application Network Protocols and Deployment Design **33**
Network & Development Diagram 33
Application Networks Diagram 34
Deployment Diagram 35
Network and Deployment Diagram 36
Integration with External Components 37
High Level APIs and Main Algorithms **37**
High-Level APIs 37
Main Algorithms and Processes 38
Software Tools and Frameworks 38
List of Team Contributions **39**

Data Definitions

Core User & Identity Entities

1. **User**
   Represents any person registered in the system. Each user has an account profile and can be either a standard user or an administrator. Used for login, personalization, and access control. Email is the primary identifier, and all actions in the system are associated with a user.

Financial Management Entities

1. **Account**
   Represents a user's financial container (checking, savings, or credit). All financial activities, such as transactions, subscriptions, and budgets, are linked to specific accounts. Accounts maintain running balances and support multi-bank integration.
2. **Transaction**
   Represents a monetary event tied to an account (like income, expense, or fund transfer). Transactions are categorized for budgeting and reporting purposes and are timestamped to track financial behavior over time.
3. **Receipt**
   Stores digital copies of receipts (image files) linked to individual transactions. Used for personal finance tracking, tax documentation, and reimbursement validation.
4. **ReimbursementRequest**
   Represents a formal request to be reimbursed for a transaction, often reviewed by an administrator. Tracks status and supports workflows such as pending, approved, or rejected.
5. **Budget**
   Defines spending limits for an account over a specified period (weekly, monthly, etc.). Used to monitor and enforce financial discipline and to trigger alerts when limits are approached or exceeded.
6. **Subscription**
   Represents recurring payments (like Netflix, utilities) tied to an account. Helps users track recurring expenses, manage due dates, and receive reminders before payments occur.

Task & Reward System

1. **Task**
   Represents actionable to-dos or habits assigned to users (like weekly budget check-ins). Tasks can be recurring or one-time, and their completion status contributes to user engagement and reward incentives.
2. **Reward**
   Represents points or incentives earned by users for completing tasks, staying within budget, or interacting with system features. Can be redeemed or used to motivate positive financial behavior.

Communication & Logging Entities

1. **Notification**
   Delivers system-generated messages to users, including reminders (like budget deadline), alerts

(like low balance), or informational updates (like task completion). Time-stamped and categorized for clarity.

2. **LemonAidLogs**
   Logs each interaction between a user and the LemonAid AI assistant. Each log includes the AI's response text and a timestamp. User ratings and comments are now handled in the separate Reviews entity to support better modularity and review flexibility.
3. **Reviews**
   Captures user feedback in the form of ratings and messages. Reviews are linked either to LemonAid logs (chatbot type) or general user experience (user_experiance type). Supports analytics and system improvement initiatives.

Support & Admin Interaction

1. **SupportTicket**
   Represents a help or issue report submitted by a user. Includes the subject, detailed message, ticket status, and any admin responses. Used to facilitate two-way support communication.

Integration & Association

1. **AccountBankLink**
   Represents the relationship between internal accounts and external banks. Enables multi-bank integration and prepares the system for open banking API compatibility. Supports syncing and aggregation of transaction data.

User Privileges and Registration

- **Standard users can:**
  - Register, log in, and manage their accounts, tasks, budgets, and AI interactions.
  - View only their own data.
- **Admins can:**
  - View user accounts, support tickets, and analytics logs.
  - Respond to support tickets and manage platform-level settings.
- **Registration includes:** name, email, and password, with email verification planned in a future milestone

---

Prioritized High-Level Functional Requirements

Priority 1

1. User

1.1. A user shall be able to create an account.

1.2. A user shall be able to securely log in to their account.

1.3. A user shall be able to recover or reset their password.

1.7. A user shall be associated with one or more accounts.

1.10. A user shall be able to view and edit their financial data history.

2. Account

2.1. An account shall store income entries for a user.

2.2. An account shall store expense entries for a user.

2.3. An account shall belong to one and only one user.

2.4. An account shall track a balance value.

2.5. An account may be connected to one or more banks.

2.7. An account shall have a transaction history log.

3. Bank

3.1. A bank shall be linked to one or more user accounts.

3.2. A bank shall provide transaction data to the system.

4. Transaction

4.1. A transaction shall be classified as income or expense.

4.2. A transaction shall belong to exactly one account.

4.3. A transaction may be manually entered or synced from a bank.

4.4. A transaction shall be categorizable by the user or AI.

5. Subscription

5.1. A subscription shall be tied to a recurring transaction.

6. Budget

6.1. A budget shall be created for one or more spending categories.

6.3. A budget shall allow setting and tracking of goals.

9. AI Assistant (LemonAid)

9.6. The system shall store AI responses and user feedback in LemonAidLogs.

10. Notification System

10.2. The system shall deliver daily, weekly, or monthly financial summaries.

13. ReimbursementRequest

13.1. A reimbursement request may be associated with one transaction.

13.2. A reimbursement request shall have a status: pending, approved, or rejected.

15. AccountBankLink

15.1. An account may be linked to one or more banks.

15.2. A bank may be linked to one or more accounts via AccountBankLink.

16. Bank

16.1. A bank shall store balance and credit data for users.

16.2. A bank shall provide transaction data to the system.

16.3. A bank shall be viewable in the user's linked accounts dashboard**.**

17. Reviews

17.1. A review shall be optionally submitted by a registered user.

17.2. A review shall include a rating value (1 to 5) and an optional message.

17.3. A review shall be timestamped at the time of creation.

17.4. A review may be linked to one and only one LemonAidLog.

17.5. A review may exist independently as general user feedback not tied to any LemonAidLog.

17.6. A LemonAidLog shall be associated with at most one review.

17.7. A review shall be categorized by type (chatbot or user_experiance).

Priority 2

1. User

1.4. A user shall be able to select between manual or bank-linked financial tracking.

1.8. A user shall be classified as a customer, premium, tester, or administrator.

1.15. A user that is an administrator shall be able to respond to support tickets.

1.16. A user that is administrator shall be able to access user account data.

2. Account

2.6. An account shall be able to categorize transactions.

2.8. An account shall support spending limits and alerts.

2.9. An account shall track subscriptions and bill payments.

2.10. An account shall be able to forecast end-of-month balances.

3. Bank

3.3. A bank shall store balance and credit data for users.

4. Transaction

4.7. A transaction may be associated with a reimbursement request.

5. Subscription

5.2. A subscription shall be viewable in a centralized dashboard.

5.3. A subscription shall trigger periodic spending alerts.

6. Budget

6.2. A budget shall belong to one and only one account.

6.4. A budget shall support notifications based on spending limits.

6.5. A budget shall be able to generate monthly recaps.

10. Notification System

10.3. The system shall notify users of upcoming bills or subscriptions.

10.4. The system shall notify users of overspending events.

10.5. The system shall alert users when savings goals are off track.

Priority 3

1. User

1.5. A user shall be able to customize their interface (like dark mode).

1.6. A user shall be able to receive AI-generated financial guidance.

1.9. A user shall be able to interact with the AI chatbot for financial assistance.

1.11. A user that is an administrator shall be able to suspend or maintain site operations.

11.12. A user that is an administrator shall be able to view flagged transactions.

1.13. A user that is an administrator shall be able to suspend or maintain site operations.

1.14. A user that is an administrator shall be able to view flagged transactions.

4. Transaction

4.5. A transaction shall be editable and deletable by the user.

4.6. A transaction shall be linked to one or more receipts.

7. Task

7.1. A task shall be assigned to a user account.

7.2. A task shall be marked completed upon user action.

7.3. A task shall be of one type: savings, investing, setup, admin, testing.

7.4. A recommended task shall be generated based on user activity or AI analysis.

8. Reward System

8.1. A user shall be able to earn rewards for completing tasks.

8.2. A user shall be able to redeem rewards through the system.

8.3. A user shall be assigned a level based on completed actions.

8.4. A user shall be able to view rankings if competitive mode is enabled.

9. AI Assistant (LemonAid)

9.1. The AI assistant shall analyze user transactions and spending habits.

9.2. The AI assistant shall simulate different financial scenarios for planning.

9.3. The AI assistant shall forecast recurring charges and balances.

9.4. The AI assistant shall recommend saving opportunities and budget changes.

9.5. The AI assistant shall provide suggestions for uncategorized transactions.

10. Notification System

10.1. The system shall send reminders for logging receipts.

11. Administrator

11.5. An administrator shall be able to manage and moderate platform use.

12. Receipt

12.1. A receipt may be linked to a transaction.

12.2. A receipt may be stored as a binary image (JPG/PNG, max 5MB).P2

12.3. A receipt shall include the upload date.

12.4. The system shall send reminders for users to upload receipts.

12.5. An administrator shall be able to access user account data.

13. ReimbursementRequest

13.3. A reimbursement request shall be visible to administrators for review.

14. SupportTicket

14.1. A support ticket shall be created by a user.

14.2. A support ticket shall include a subject, message, and status.

14.3. A support ticket shall be responded to by an administrator.

14.4. A support ticket shall track the admin response and resolution state.

15. AccountBankLink

15.3. The system shall sync data from bank APIs using these links.

17. Reviews

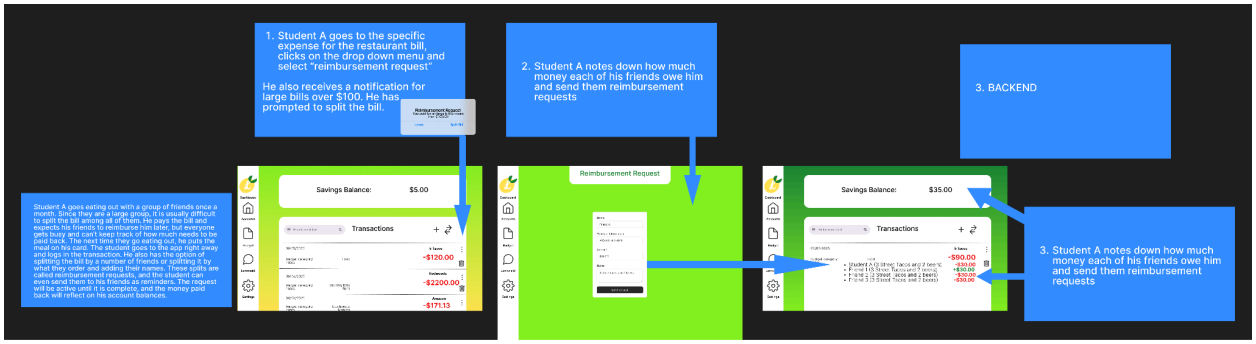
17.8. A review shall be viewable by system administrators for quality assurance and analysis.

17.9. A review shall be stored securely and associated with the submitting user, if applicable.

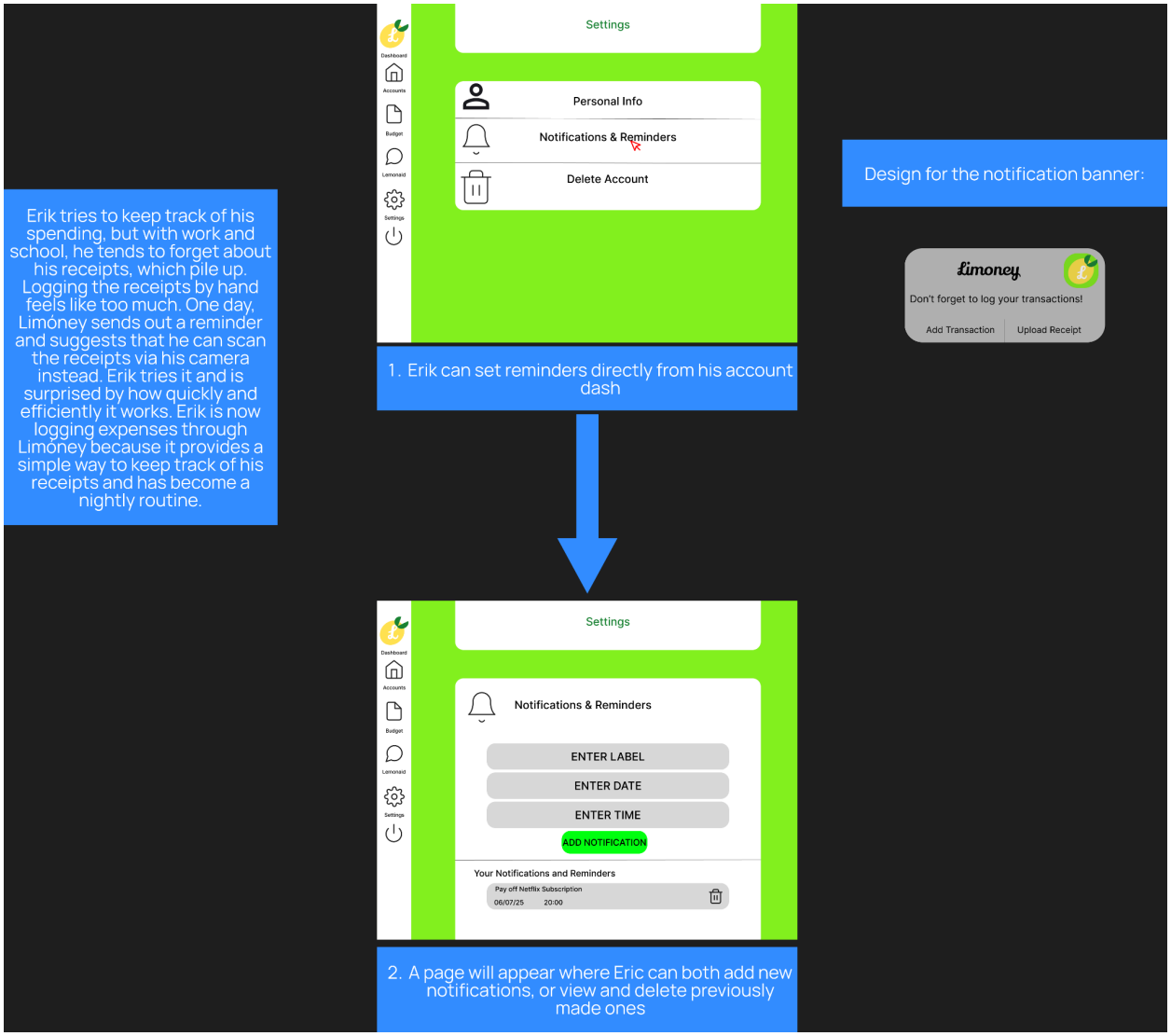
17.10. A review shall support analytics use cases for measuring system effectiveness and user satisfaction.

UI/UX Wireframes

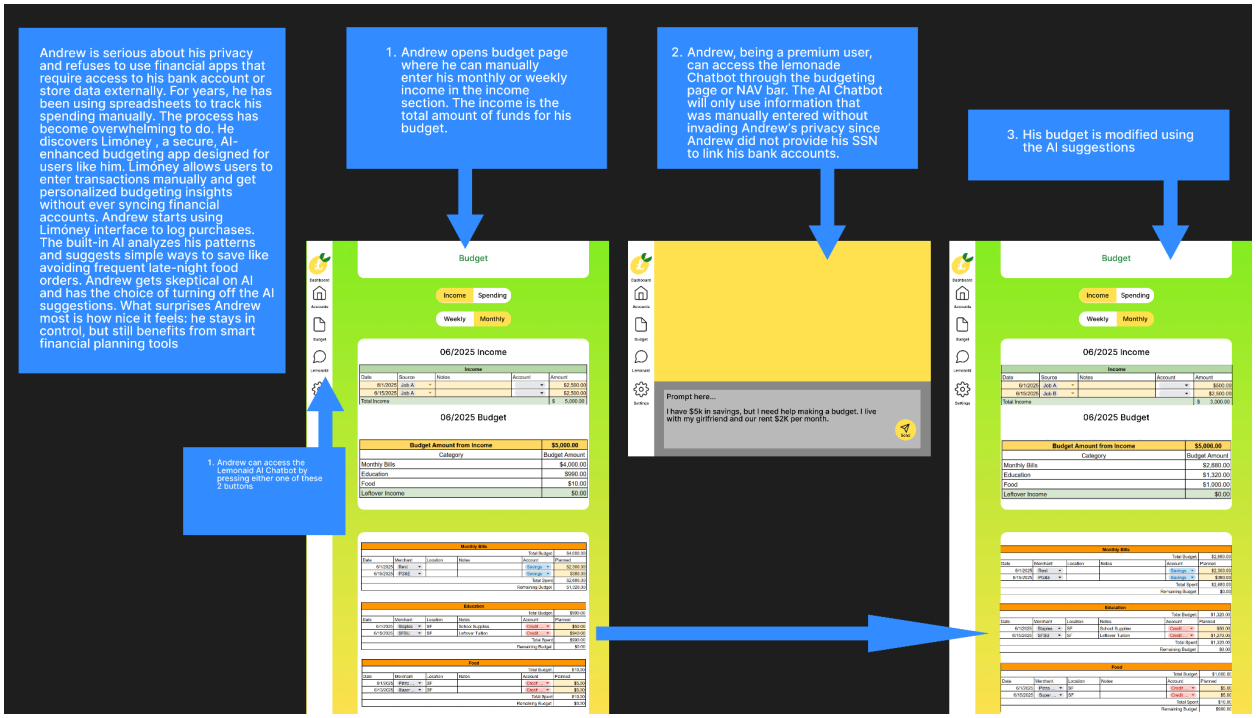
2.Tracking Multiple Reimbursements:



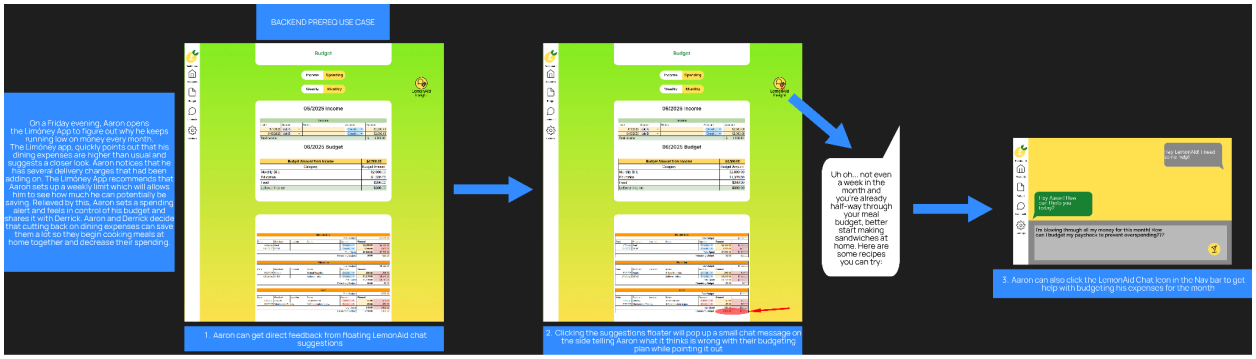
3. Log Transaction:



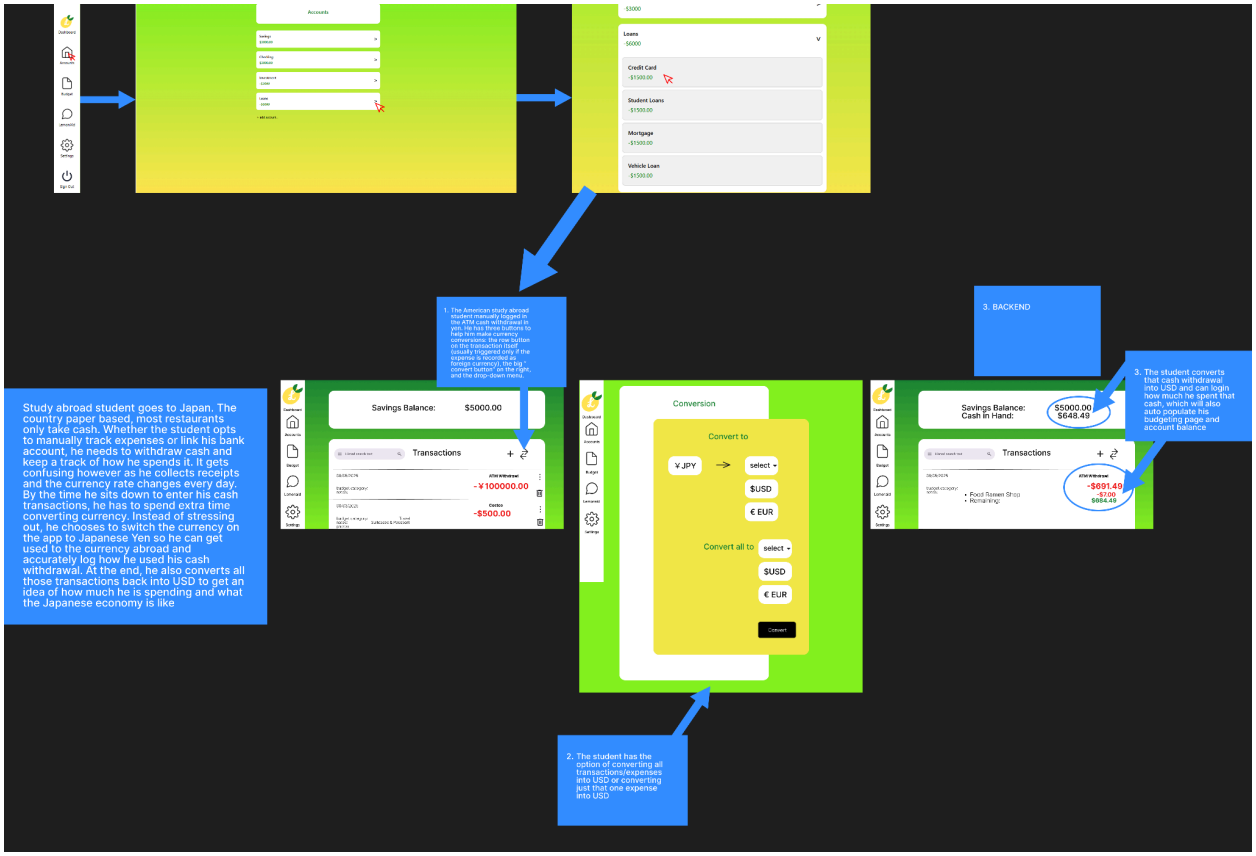
4. Security Concern:



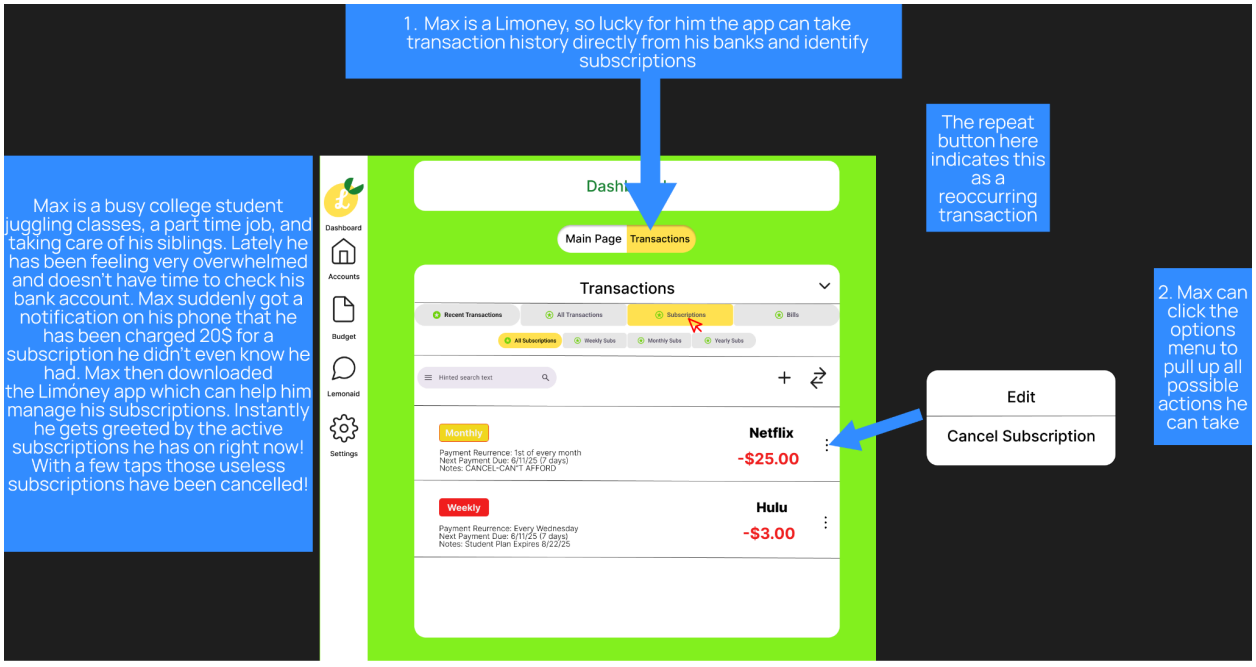
5. Receive AI Suggestions:



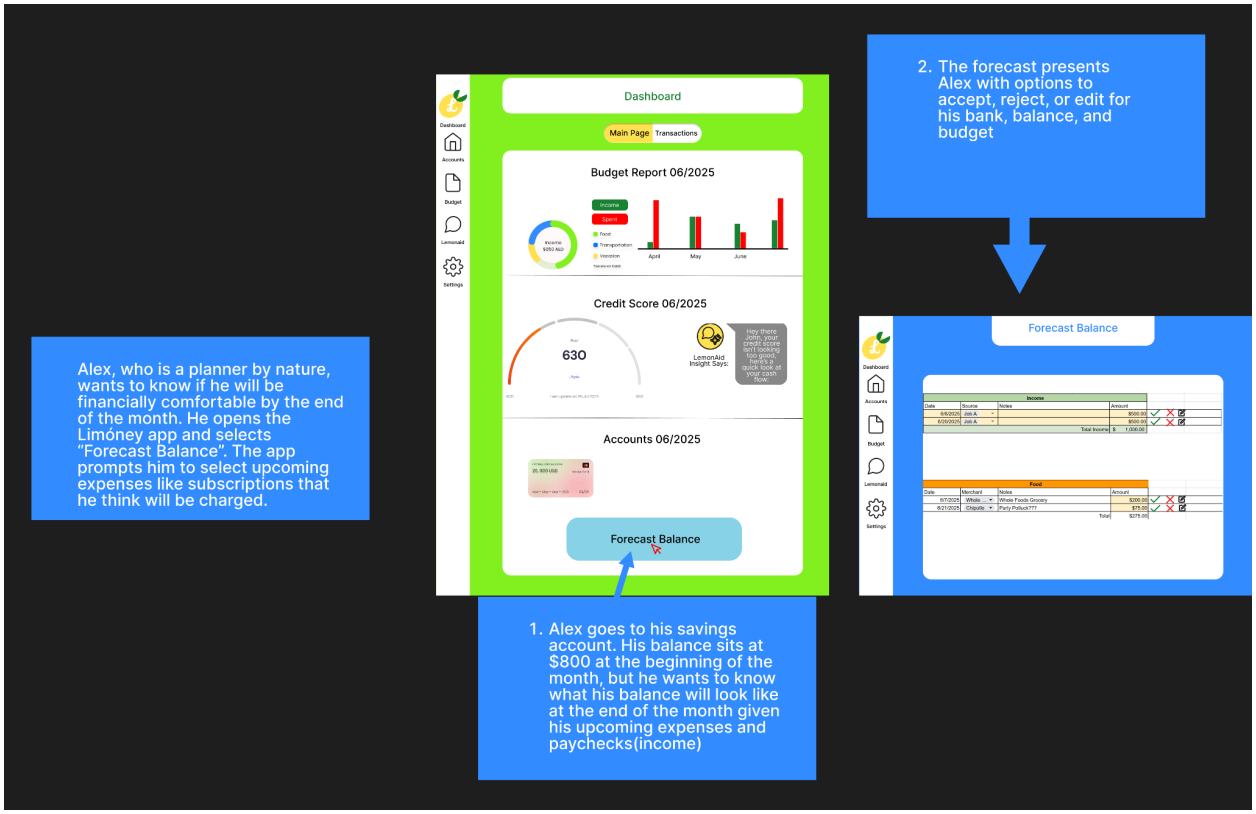
6. Currency Exchange:



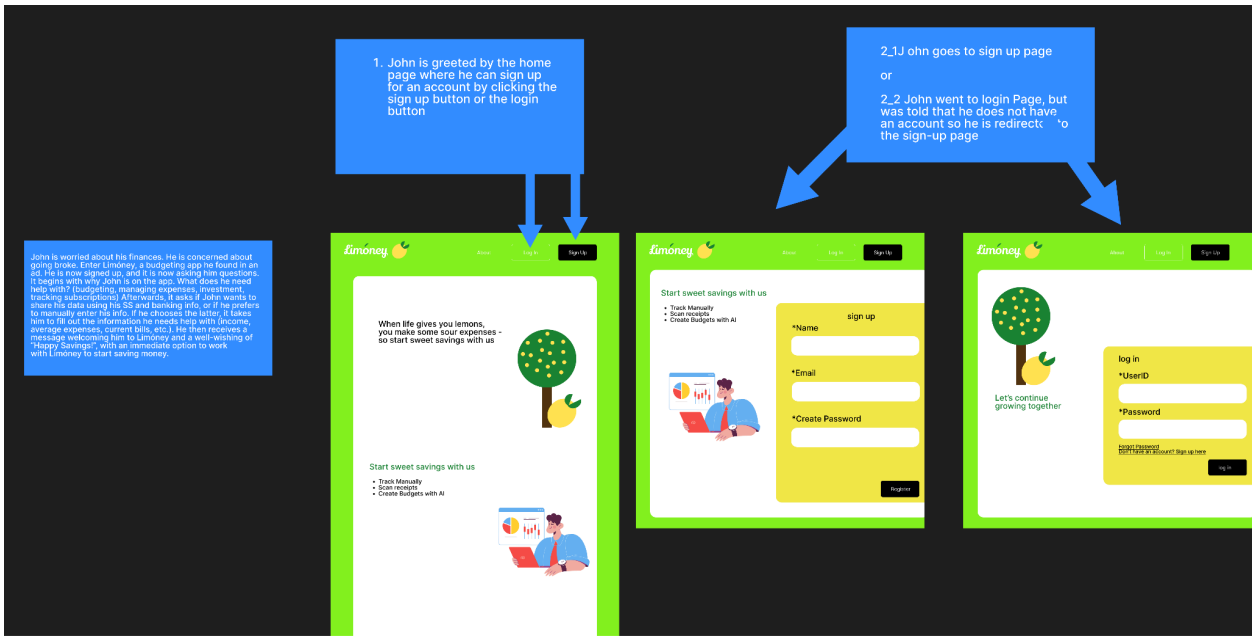
7. Tracking Ongoing Expenses:



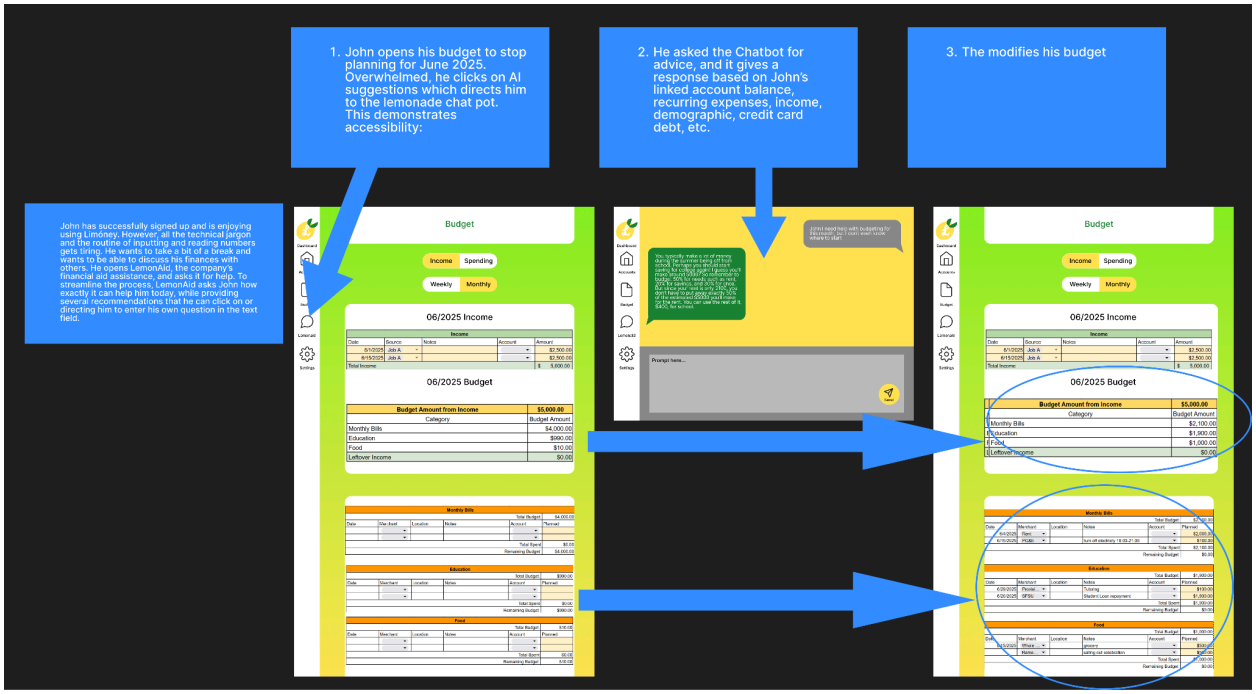
8. Predicting Balances at The End of The Month:



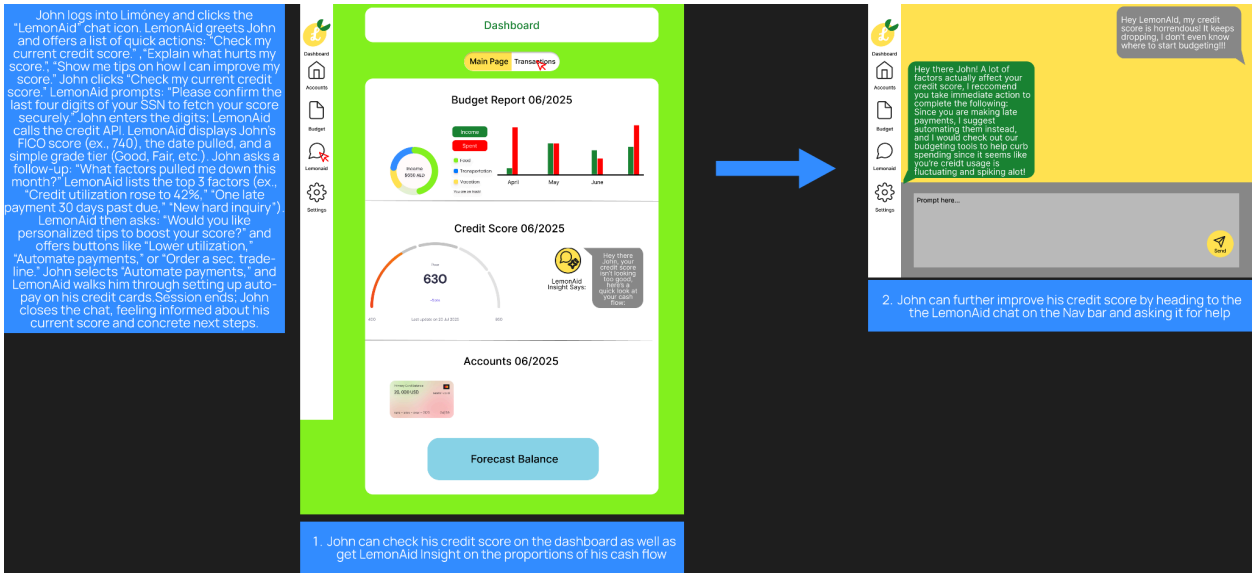
9. Signing Up/Intro UI:



10. Chatbot/AI Assistance:



11. Credit Score Inquiry:



FIGMA LINK:
https://www.figma.com/proto/VRQplRZM2PeWQ1DOF1hh7t/CSC-648-Team-02?node-id=0-1&t=XNTNx3me7X7fSEqa-1

High-Level System Design

High Level Database Architecture

Initial Database Requirements

User

Table: `User`

- 1 User to 0..* `UserAccount` (Accounts)
- 1 User to 0..* `ManualAccount` (Accounts)
- 1 User to 0..* `SupportTicket`
- 1 User to 0..* `Notification`
- 1 User to 0..* `LemonAidLogs`
- 1 User to 0..* `Reviews`
- ISA: `User` can be an **Administrator**
- 1 User (admin) ↔ 0..* `SupportTicket.responseByAdmin` (FK or link)
- 1 User ↔ 0..* `sessions`

Account (Manual and Plaid)

Tables:

- `UserAccount` (for Plaid-linked)
- `ManualAccount` (for manually created)

Relationships:

- 0 to many `UserAccount` / `ManualAccount` ↔ 1 User
- 1 Account to 0..* `UserTransaction` / `ManualTransaction`
- 1 Account to 0..* `Budget`
- 1 Account to 0..* `Subscription`

Transaction

Tables:

- `UserTransaction` (Plaid)
- `ManualTransaction`

Relationships:

- 0 to * Transaction ↔ 1 Account (`UserAccount` or `ManualAccount`)
- 1 Transaction ↔ 0 or 1 `ReimbursementRequest`

ReimbursementRequest

Table: `ReimbursementRequest`

- 0 to 1 ReimbursementRequest to 1 `Transaction` (Manual or User)

Budget

Table: `Budget`

- 0 to 1 Budget to 1 Account (`UserAccount` or `ManualAccount`)

Subscription

Table: `Subscription`

- 0 to * Subscription to 1 Account

SupportTicket

Table: `SupportTicket`

- 1 Ticket ↔ 1 User (creator)
- 0 or 1 admin (User) responds *(ISA relationship)*

Notification

Table: `Notification`

- 1 Notification ↔ 1 User

Reviews

Table: `Reviews`

- 0 or 1 User ↔ 1 Review
- 0 or 1 LemonAidLog ↔ 1 Review
- Fields: `rating` (required), `message` (required), `type`, `timestamp`

DBMS Selection

We will use **MySQL** because it is a popular, reliable relational database that integrates seamlessly with Node.js/Express, handles financial and user data easily, and is approved by our CTO.

User

Attributes:

- `userId`: unique identifier for each user `INT UNSIGNED`
- `name`: full name of the user `VARCHAR(45)`
- `email`: login credential and unique contact identifier `VARCHAR(45)`
- `password`: hashed user password `VARCHAR(255)`
- `userType`: determines user role (standard or admin) `ENUM('standard', 'admin')`

Relationships:

- 1 to 0…* with `ManualAccount` and `UserAccount`
- 1 to 0…* with `Task`
- 1 to 0…* with `Reward`
- 1 to 0…* with `Notification`
- 1 to 0…* with `SupportTicket` (creator)
- 1 to 0…* with `SupportTicket` (adminId responder — only if `userType = 'admin'`)
- 1 to 0…* with `LemonAidLogs`
- 1 to 0…* with `Reviews`

ManualAccount / UserAccount

Attributes:

- `accountId`: unique identifier for each account `INT UNSIGNED`
- `userId`: reference to owning user `INT UNSIGNED`
- `balance`: current account balance `INT UNSIGNED`
- `accountType`: financial category of the account `ENUM('checking', 'saving', 'credit')`

Relationships:

- 1 to 1 with `User`
- 1 to 0…* with `ManualTransaction` or `UserTransaction`
- 1 to 0…* with `Budget`
- 1 to 0…* with `Subscription`
- 1 to 0…* with `AccountBankLink`

Bank

Attributes:

- `bankId`: unique identifier for each bank `INT UNSIGNED`
- `name`: name of the financial institution `VARCHAR(45)`

Relationships:

- 1 to 0…* with `Account` via `AccountBankLink`

UserTransaction / ManualTransaction

Attributes:

- `transactionId`: unique identifier for each transaction `INT UNSIGNED`
- `accountId`: reference to the linked account `INT UNSIGNED`
- `amount`: transaction amount (positive or negative) `INT`
- `category`: user-defined or system-defined tag `VARCHAR(40)`
- `type`: defines purpose `ENUM('income', 'expense', 'transfer')`
- `date`: date of the transaction `DATE`
- `subscriptionId`: optional link to related subscription `INT UNSIGNED`

Relationships:

- 1 to 1 with `Account`
- 1 to 0…1 with `Receipt`
- 1 to 0…1 with `ReimbursementRequest`
- 1 to 0…1 with `Subscription`

ReimbursementRequest

Attributes:

- `requestId`: unique identifier for the request `INT UNSIGNED`
- `transactionId`: transaction that the reimbursement applies to `INT UNSIGNED`
- `status`: workflow status `ENUM('pending', 'approved', 'rejected')`

Relationships:

- 0..1 to 1 with `Transaction`

Subscription

Attributes:

- `subscriptionId`: unique identifier for each subscription `INT UNSIGNED`
- `accountId`: reference to the linked account `INT UNSIGNED`
- `name`: label for the subscription `VARCHAR(45)`
- `amount`: amount of recurring payment `INT UNSIGNED`
- `interval`: frequency `ENUM('daily', 'weekly', 'monthly', 'yearly')`
- `nextDueDate`: next billing date `DATE`

Relationships:

- 0…* to 1 with `Account`
- 0…1 to 0…* with `Transaction`

Budget

Attributes:

- `budgetId`: unique identifier for the budget `INT UNSIGNED`
- `accountId`: reference to the associated account `INT UNSIGNED`
- `limitAmount`: allowed budget limit `INT UNSIGNED`
- `startDate`: budget start date `DATE`
- `endDate`: budget end date `DATE`

Relationships:

- 0…* to 1 with `Account`

Notification

Attributes:

- `notificationId`: unique identifier `INT UNSIGNED`
- `userId`: recipient of the notification `INT UNSIGNED`
- `type`: nature of the message `ENUM('info', 'alert', 'reminder')`
- `content`: message content `TEXT`
- `date`: delivery timestamp `DATETIME`

Relationships:

- 0…* to 1 with `User`

Reviews

Attributes:

- `reviewId`: unique identifier for the review `INT UNSIGNED`
- `userId`: reviewer (nullable) `INT UNSIGNED`
- `logId`: linked LemonAidLog (nullable) `INT UNSIGNED`
- `message`: feedback left by the user `TEXT`
- `rating`: numeric level `ENUM('1', '2', '3', '4', '5')`
- `createdAt`: submission time `TIMESTAMP`
- `type`: targets chatbot or UX `ENUM('chatbot', 'user_experiance')`

Relationships:

- 0…1 to 1 with `User`
- 0…1 to 1 with `LemonAidLogs`

SupportTicket

Attributes:

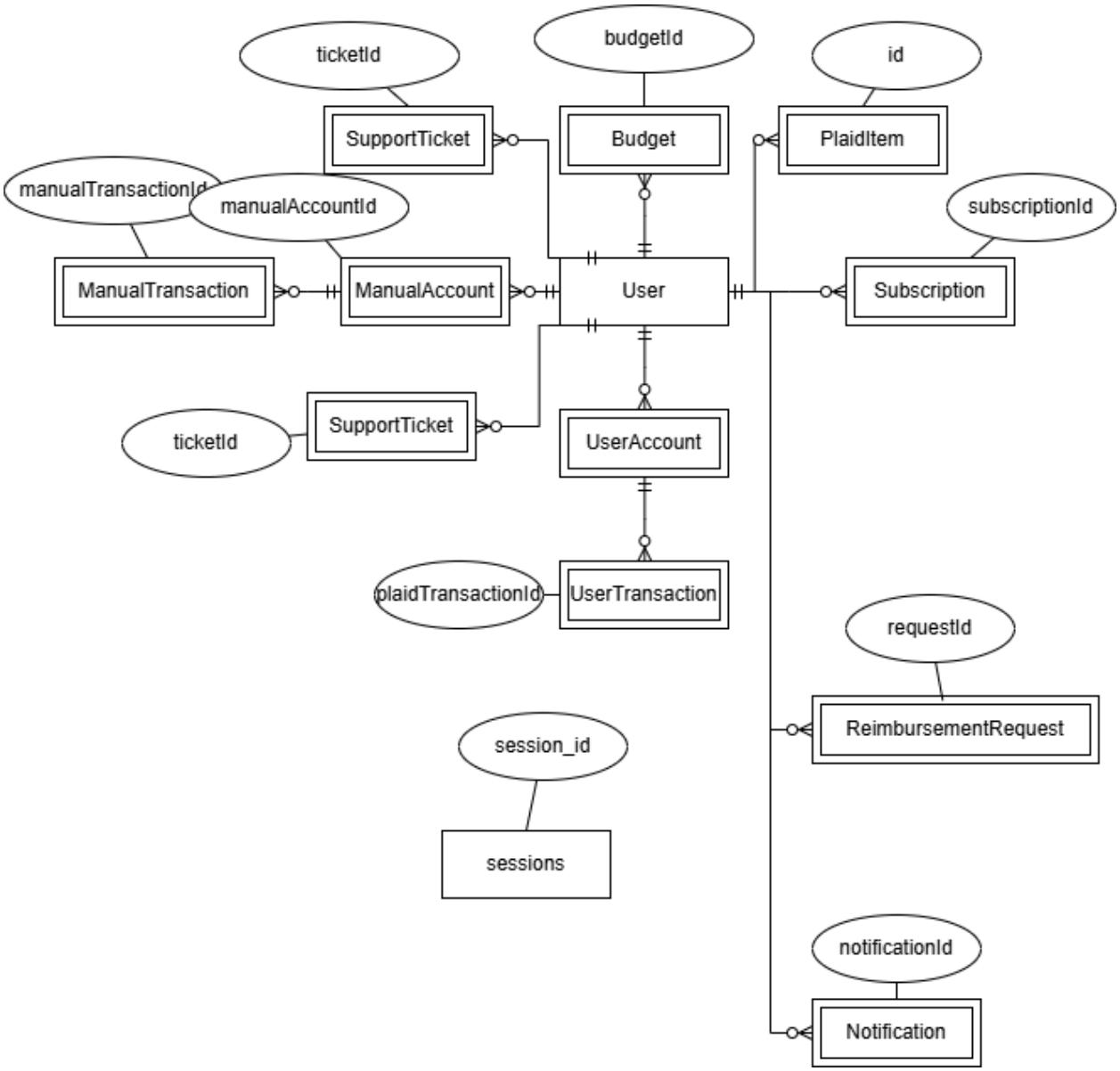
- `ticketId`: unique identifier `INT UNSIGNED`
- `userId`: creator of the ticket `INT UNSIGNED`
- `subject`: title of the issue `VARCHAR(45)`

- `message`: description of the issue `TEXT`
- `status`: progress status `ENUM('open', 'in progress', 'closed')`
- `adminResponse`: optional reply `TEXT`
- `adminId`: (optional) userId of admin responder `INT UNSIGNED`

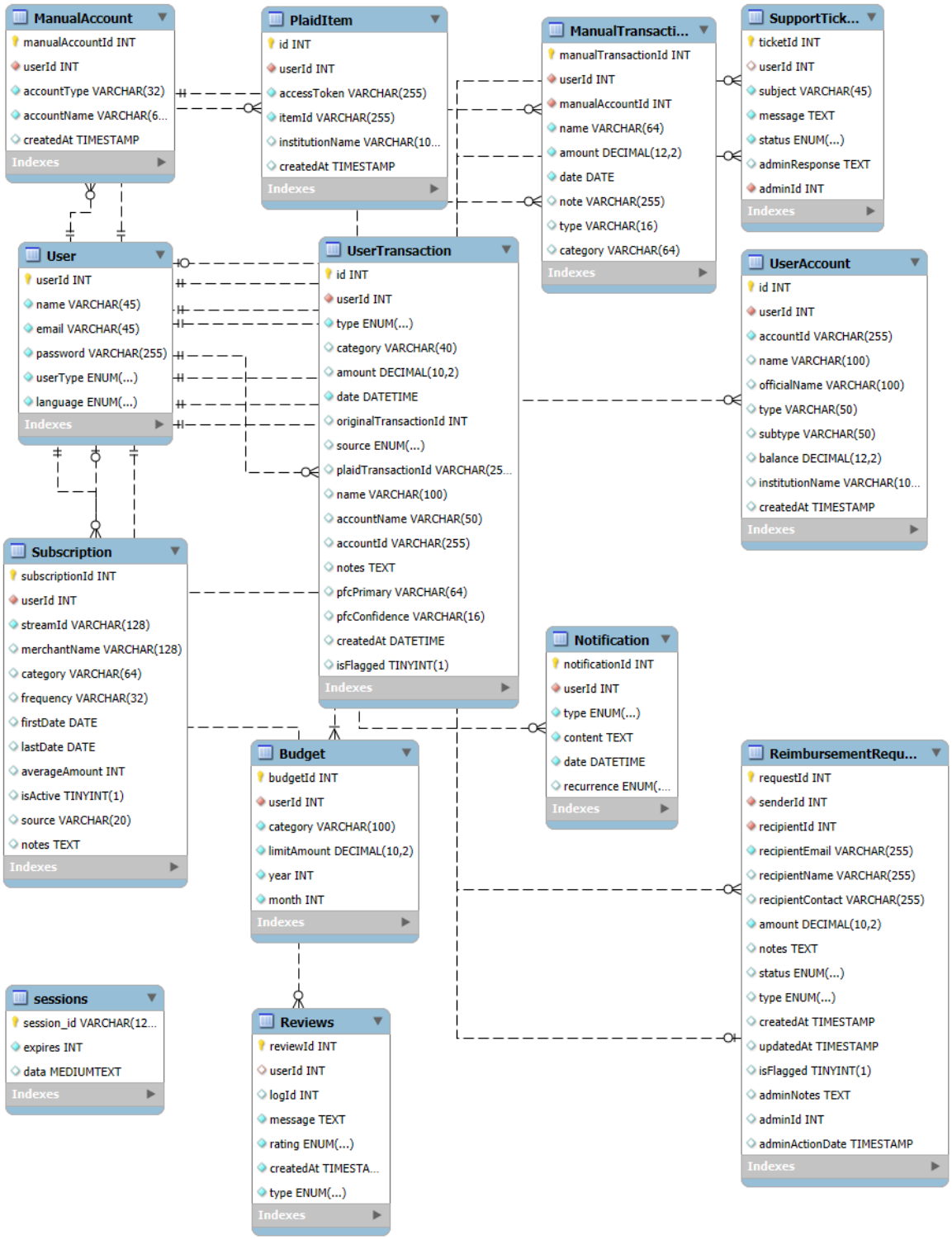
Relationships:

- 1 to 1 with `User` (creator)
- 0…1 to 1 with `User` (admin responder if `userType = 'admin'`)

Entity Relationship Diagram:



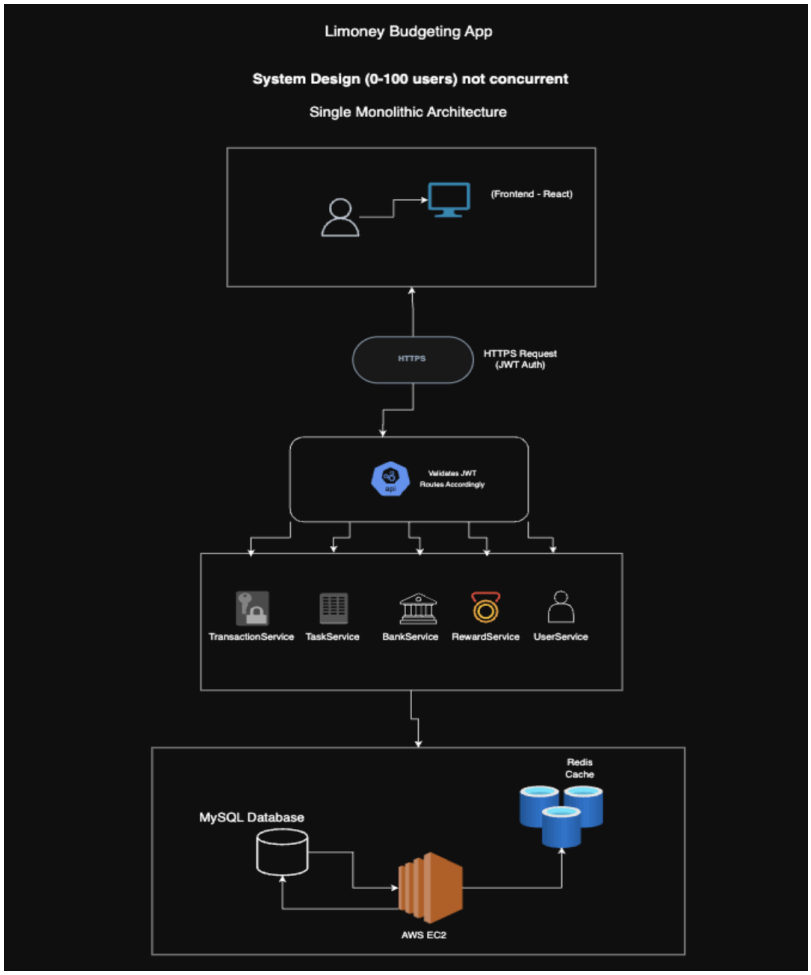
Extended Entity Relationship Diagram:



Backend Architecture

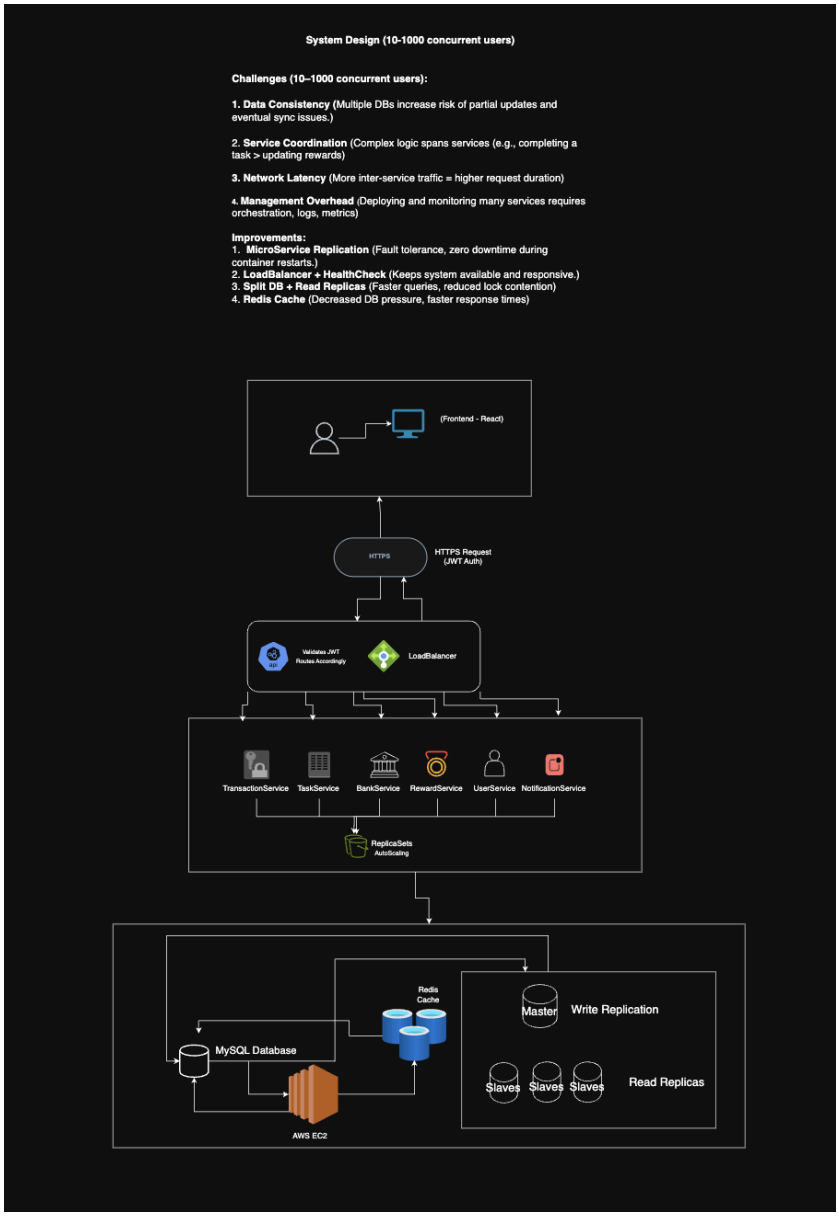
ALL SCALE DESIGN

The Limoney Budgeting App for 0-100 (not concurrent) users uses a single monolithic architecture. The frontend (React) communicates via HTTPS (JWT Auth) to a backend API, which validates JWT and routes requests to services such as TransactionService, TaskService, BankService, RewardService, and UserService. All services interact with a single MySQL database (hosted on AWS EC2) and an optional Redis cache for session/AI caching.



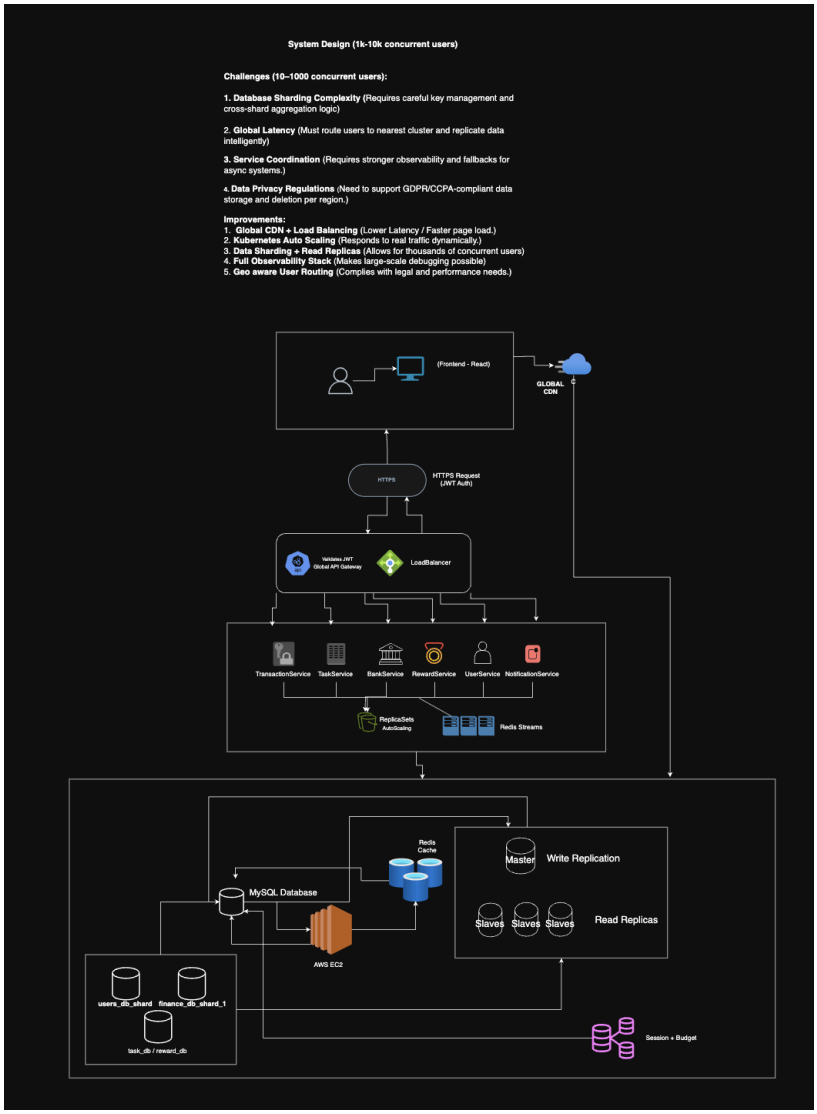
MEDIUM SCALE DESIGN

For 10-1000 concurrent users, the system introduces microservice replication, load balancing, and read replicas for the database. The backend is horizontally scalable with Dockerized microservices, an auto-scaling NGINX load balancer, and health checks. The MySQL database uses master-read replica architecture for scalability, and Redis is used for caching AI and frequent queries. Improvements include fault tolerance, health checks, and reduced DB pressure.



LARGE SCALE DESIGN

For 1k-10k concurrent users, the system leverages Kubernetes-managed microservices, global CDN, and database sharding by user region. The backend is orchestrated with Kubernetes for auto-scaling and CI/CD. The database layer uses sharding and a mix of read/write replicas, with multi-layer Redis caching. Security is enhanced with rate-limiting, API key enforcement, and mutual TLS between services. The architecture supports global latency reduction and compliance with data privacy regulations.



ARCHITECTURE SUMMARY

Microservices Architecture

- Domain-driven microservices: each major business capability is its own independently deployable service
    - Services: UserService, BankService, BudgetService, SubscriptionService, TaskService, AIRecommendationService, RewardService
- Service communication:
    - Synchronous calls via REST APIs

- Asynchronous messaging for AI triggers and reward events

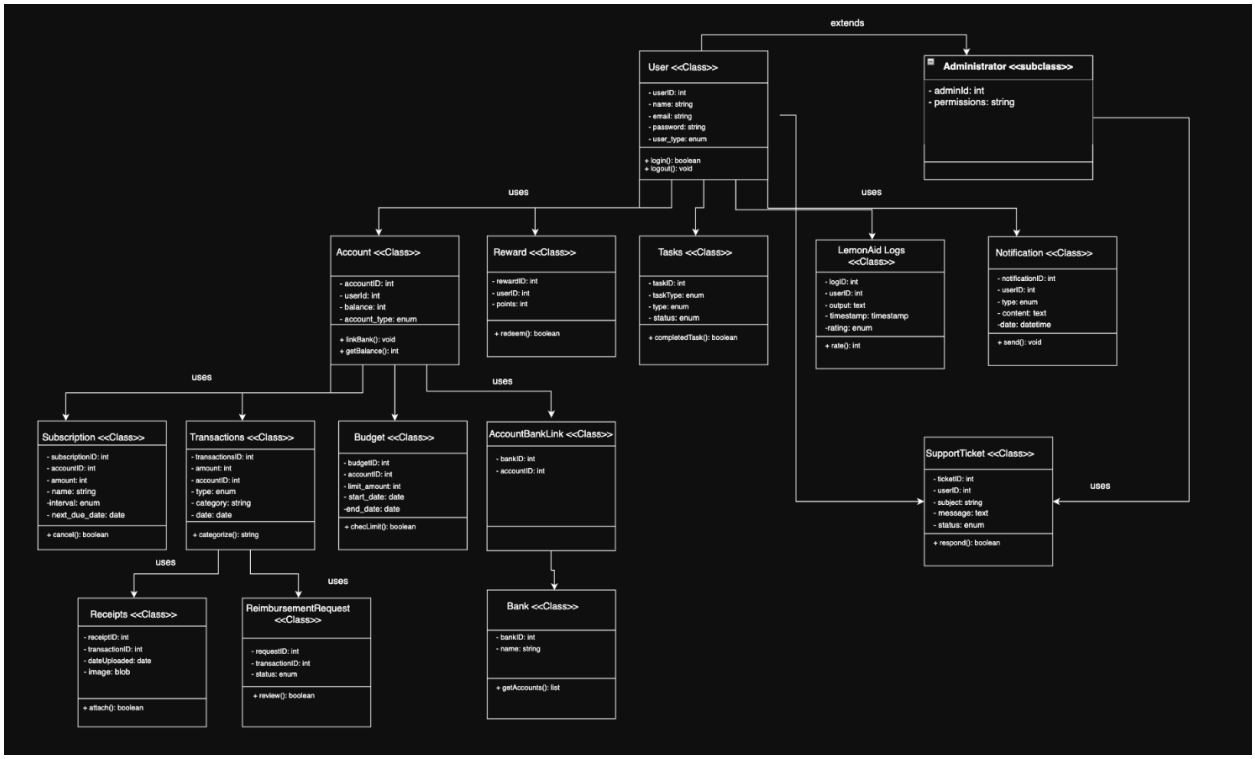
Backend System Design

- **Load Balancing**
  - NGINX reverse proxy or AWS ALB to evenly route traffic
  - Health checks to verify service availability
  - Supports horizontal scaling for better availability and responsiveness
- **Caching Strategies**
  - Cache-aside pattern for database query results (e.g., user budget history)
  - LRU eviction policy to optimize memory usage
- **Reliability & Fault Tolerance**
  - Each service runs in its own Docker container managed by Kubernetes
  - Circuit breakers (Resilience4j) to prevent cascading failures
  - Retry and timeout policies to stabilize inter-service connections
- **Containers & Orchestration**
  - Docker containers ensure portable environments (local → staging → production)
  - Kubernetes handles deployment, health checks, auto-scaling, and rolling updates
- **Data Replication & Consistency**
  - Master-Slave replication for MySQL to boost read throughput
  - Synchronous writes for critical services (e.g., TransactionService, SavingsService)
  - Eventual consistency acceptable for non-critical operations to improve performance
- **Security Considerations**

- JWT authentication for all incoming requests
- Role-Based Access Control (RBAC) distinguishing Admins vs. Customers
- Mutual TLS for service-to-service communication
- Rate limiting and request logging at the API gateway
- Data encryption: AES-256 at rest and TLS in transit

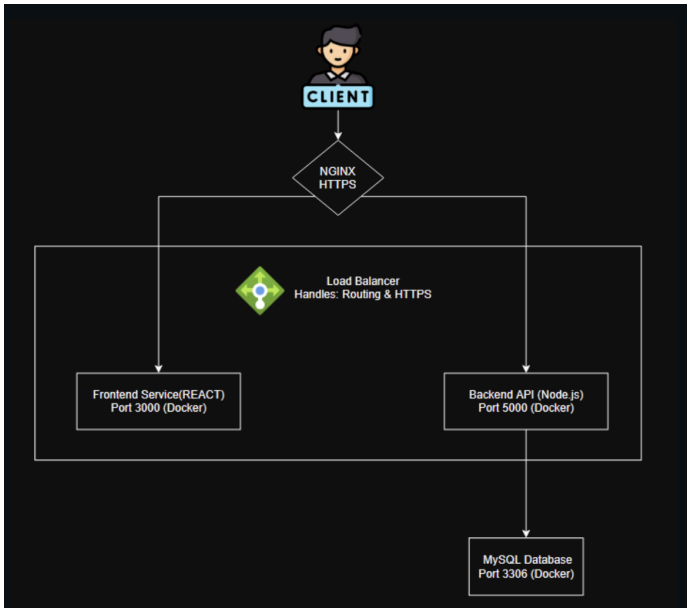
UML Design

- The following UML diagram illustrates the key domain classes and service interfaces for the Limóney backend system.

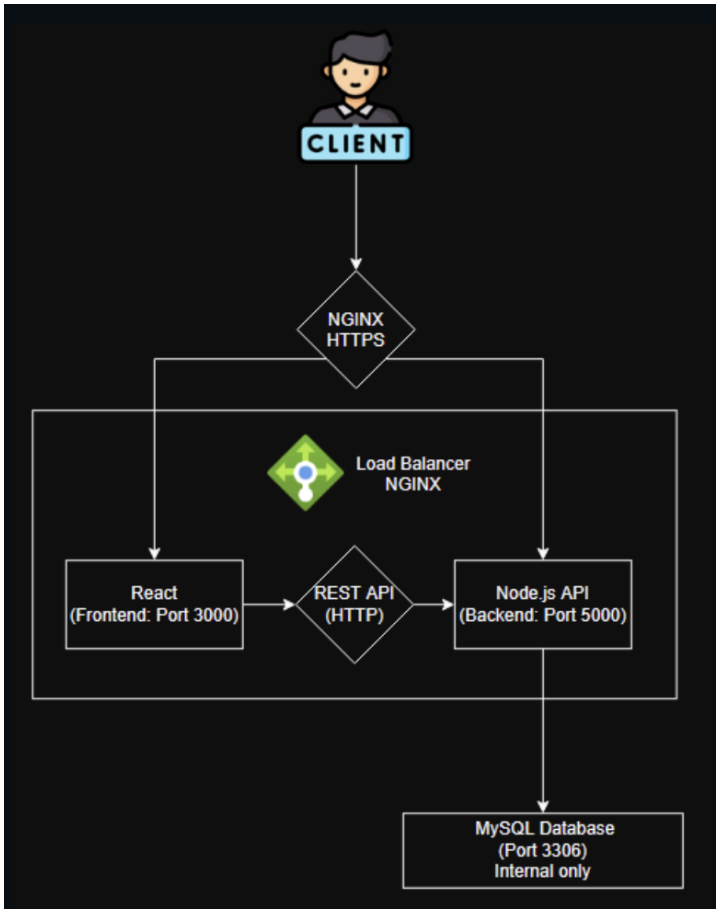


High Level Application Network Protocols and Deployment Design

Network & Development Diagram



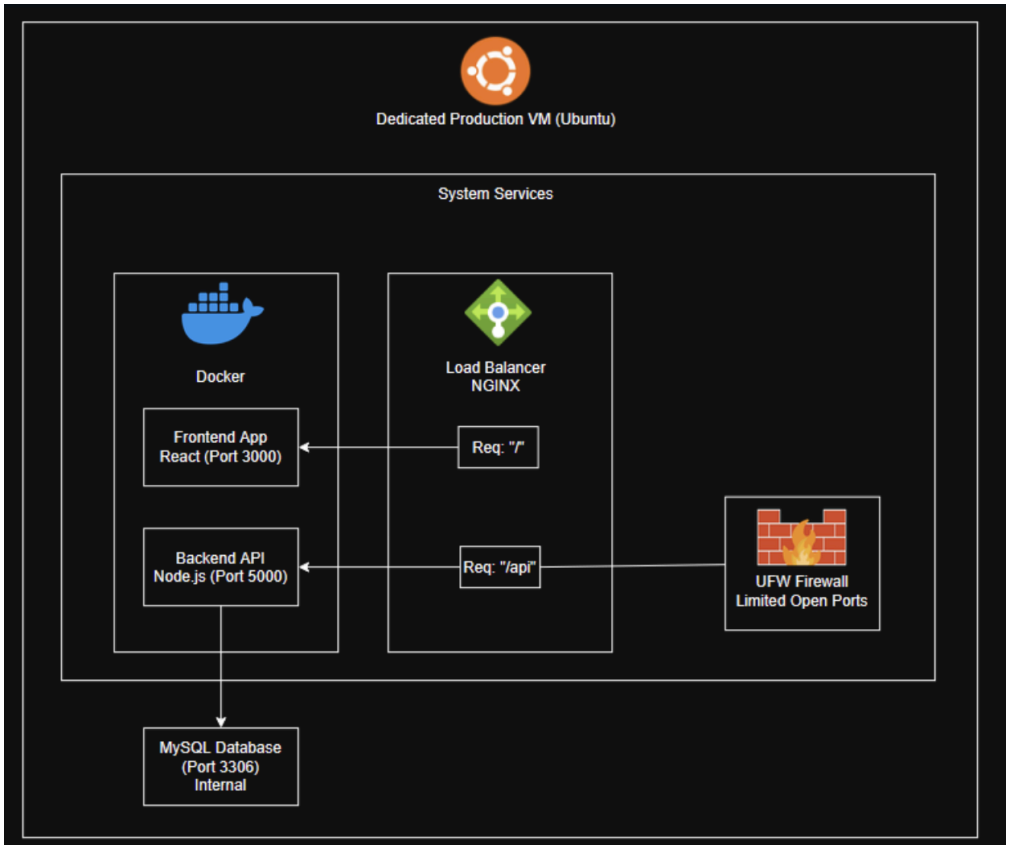
Application Networks Diagram



Protocols

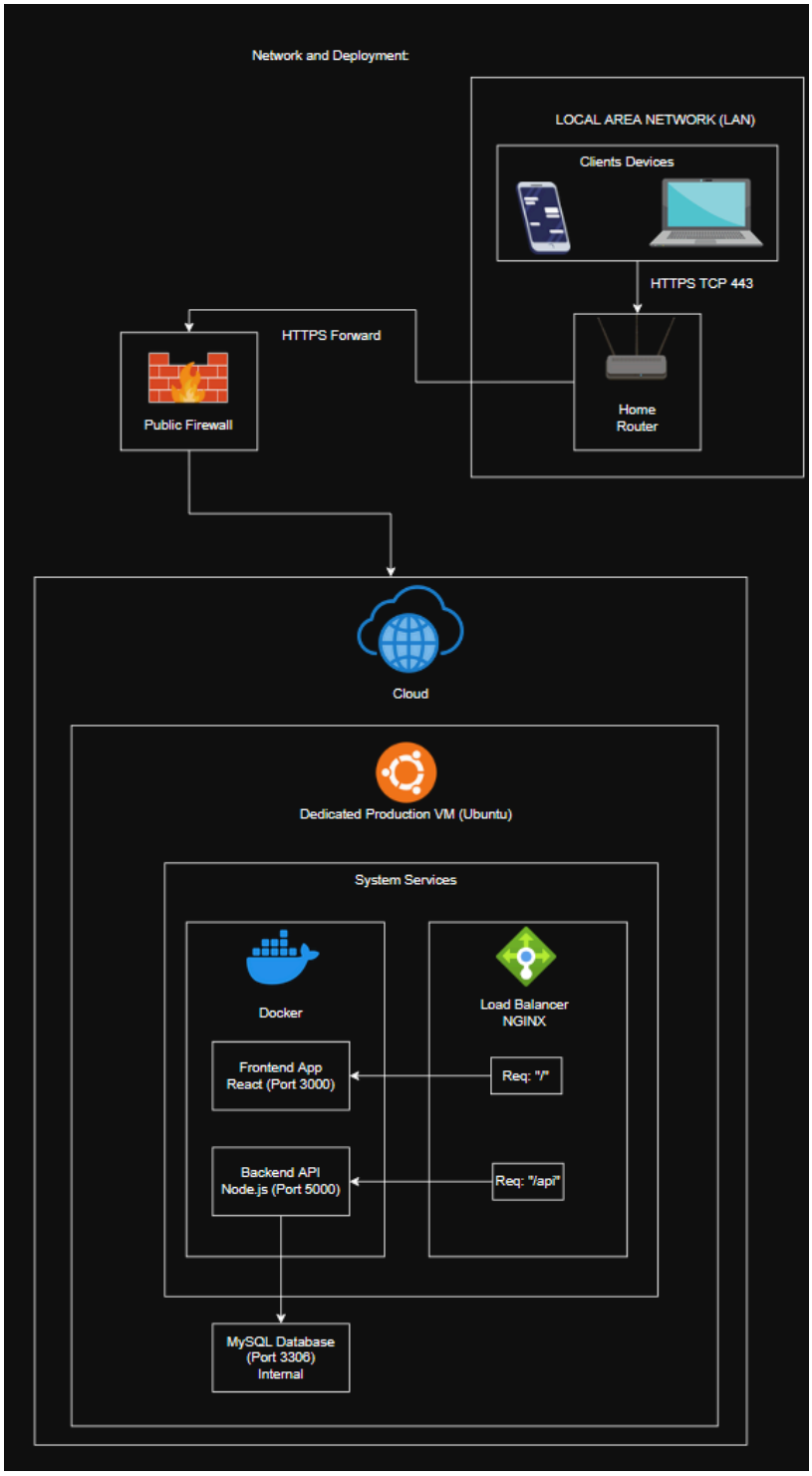
Network uses HTTPS for securing client access with NGINX, routing to React frontend and Node.Js backed. The backend accesses MYSQL database internally over TCP/IP. Services run on Docker bridge network, with security enforced through SSL/TLS, CORS. and a UFW firewall that restricts access only to essential ports to the application to work. NGINX is the main gateway and proxy which isolates the internal services from the public.

Deployment Diagram



This diagram illustrates the system's structure on a dedicated Ubuntu server. Docker manages the React frontend and Node.js backend processes. NGINX handles incoming traffic and routes it to the appropriate service. A UFW firewall restricts network access, allowing only essential ports. The backend securely connects to an internal or cloud-hosted MySQL database.

Network and Deployment Diagram



Integration with External Components

External Components:

- Gemini API for our LemonAid Chatbot component

Internal Libraries:

- Bycrypt: for hashing passwords
- Dotenv: for ".env" variable handling
- Cors: for securing cross-origin request
- Mysql12: for database connection

High Level APIs and Main Algorithms

High-Level APIs

- **Account & Transaction Management API**
    - Allows users to securely manage accounts, log income/expenses, categorize transactions, and attach receipts.
- **Budget & Subscription API**
    - Helps users create, update, and track budgets and recurring payments with due-date reminders.
- **Task & Reward System API**
    - Supports gamification by letting users complete tasks and earn reward points based on activity.
- **Notification API**
    - Delivers alerts, reminders, and system messages tied to user activity and financial goals.
- **Support & Admin API**
    - Enables users to submit support tickets and allows admins to manage and respond to them.
- **LemonAid AI Interaction API**
    - Logs interactions between users and the LemonAid assistant and allows users to rate each AI response.

Main Algorithms and Processes

AI Rating System: After each LemonAid interaction, users rate the response on a scale of 1–5. These ratings will track AI quality over time and may later feed into an admin review process or an AI fine-tuning feedback loop.

Spending Categorization: Transactions are initially auto-labeled via simple keyword matching, with a roadmap to evolve into a smarter, pattern-based classification engine.

Budget Alerts: The system runs periodic checks of current spending against set budget limits and sends notifications when users approach or exceed those thresholds.

Reward Calculation: Points are awarded for completing financial tasks or goals; the calculation factors in task type, frequency, and any completion streaks to drive user engagement.

Software Tools and Frameworks

Current Technology Stack

- Node.js + Express (backend API framework)
- MySQL (relational database)
- React (or plain HTML/CSS/JavaScript) for the frontend
- Docker (for containerized deployment)
- AWS EC2 running Ubuntu 22.04 (cloud hosting)
- Let’s Encrypt + Certbot (SSL certificates)

Planned Integrations

- Firebase Authentication (optional, for user auth & password resets)
- GenAI API (for the LemonAid assistant)
- Any new tools will be approved by the instructor before implementation.

List of Team Contributions

Name	Contributions	Score (1-10)
Emily Perez	started and finished tech doc for m3, fixed fetch errors, refined data definitions, refined erd, created eer, set up guide for setting up working environment, connected pages together, set up and enforced code standards, handled pull requests, assigned tasks to members, implemented final database(forward engineered eer), adjusted algorithms to new database, implemented all parts of the settings page, implemented public page ui, got domain for site, got ssl certificates, implemented ui for reimbursement requests ranking algo, deployed code onto server	8
Ishaank Zalpuri	helped with UI/UX Wireframes, created accounts page in the userlogin pages, organized file directory, implemented ui for delete accounts, implemented ui for transactions, fixed figma to new implemented design, implemented ui for adding accounts	7.5
Andrew Brockenborough	Added navbar for the userlogin pages, created parts of the settings page, helped with organizing the technical documentation, kept the readmemd up to date,	5
Dani Luna	led and designed the UI/UX Figma Wireframes, created TODO list, implemented ui for budget page, helped handle pull requests, implemented ui for	9

	balance forecas, implemented ui for dashboard	
Jonathan Gonzalez	fixed uml diagram for m3, helped with public page ui implementation, fixed ui for settings, implemented reimbursement request and its functionalities	7.5
Gene Orias	implemented search functionality logic, implemented rating functionality logic, set up all the jsx files (pages), set up gemini api for lemonaidchat, fixed ui for lemonaid chat, fixed ui for accounts page, set up ui for transactions of accounts, helped organize userloginpages for coordination,	9

Milestone 4

SW Engineering CSC648-848-01 Summer 2025

Limóney (Financial Budgeting Webapp)

Team 02

Milestone 4

8/5/25

Name	**Role**
Emily Perez (eperez@sfsu.edu)	Team Lead, Database Administrator, Github Master
Ishaank Zalpuri	Frontend Lead, UI Designer, Frontend Developer

Andrew Brockenborough	Technical Writer
Dani Luna	Frontend Lead, UI Designer, Frontend Developer
Jonathan Gonzalez	Software Architect, Frontend Lead, UX Specialist,
Gene Orias	Backend Lead

Milestone	**Version**	**Date**
Milestone 4	Version 2	8/5/25
Milestone 4	Version 1	7/31/25
Milestone 3	Version 2	7/31/25
Milestone 3	Version 1	7/22/25
Milestone 2	Version 2	7/10/25
Milestone 2	Version 1	7/3/25
Milestone 1	Version 2	7/2/25
Milestone 1	Version 1	6/17/25

Table of Contents

Table of Contents **2**
Product Summary **3**
Product Name: 3
Final Functional Requirements (P1 and P2) 3
Unique Features 6
Deployment URL: https://limoney.org 6
Usability Test Plan **7**
Test Objectives 7
Test Setup 7
Test Functions 7
1. Reimbursement Effectiveness Table 8

2. Currency Converter
Effectiveness Table....8
3. LemonAid AI
Effectiveness Table....9
4. Budget Visualization
Effectiveness Table....10
5. Account Tracking
Effectiveness Table....10
6. Transactions Search
Effectiveness Table....11
Likert User Satisfaction....11
QA Test Plan....13
Selected Non-Functional Requirements....13
QA Test Table
1. Performance....13
Code Review....19
Coding Standards....19
Github Code Review....19
External Review....27
Security Practices....28
Assets Protected....28
Password Encryption....28
Input Validation....28
Self-Check: Adherence to Non-Functional Specs....31
List of Team Contributions....34

Product Summary

Product Name:

Limoney

Final Functional Requirements (P1 and P2)

Priority 1

1. User

1.1. A user shall be able to create an account.

1.2. A user shall be able to securely log in to their account.

1.3. A user shall be able to recover or reset their password.

1.7. A user shall be associated with one or more accounts.

1.10. A user shall be able to view and edit their financial data history

2. Account

2.1. An account shall store income entries for a user.

2.2. An account shall store expense entries for a user.

2.4. An account shall track a balance value.

2.5. An account may be connected to one or more banks.

2.7. An account shall have a transaction history log.

3. Bank

3.1. A bank shall be linked to one or more user accounts.

3.2. A bank shall provide transaction data to the system.

4. Transaction

4.1. A transaction shall be classified as income or expense.

4.3. A transaction may be manually entered or synced from a bank.

5. Subscription

5.1. A subscription shall be tied to a recurring transaction.

6. Budget

6.1. A budget shall be created for one or more spending categories.

6.2. A budget shall belong to one and only one account.

6.3. A budget shall allow setting and tracking of goals.

13. ReimbursementRequest

13.1. A reimbursement request may be a request for money to a valid user.

13.2. A reimbursement request may be sending money to a valid user.

13.2. A reimbursement request shall have a status: pending, approved, or rejected.

14. SupportTicket

14.1. A support ticket shall be created by a user.

14.2. A support ticket shall include a subject, message, and status.

15. AccountBankLink

15.1. An account may be linked to one or more banks.

15.2. A bank may be linked to one or more accounts via AccountBankLink.

16. Bank

16.1. A bank shall store balance and credit data for users.

16.2. A bank shall provide transaction data to the system.

16.3. A bank shall be viewable in the user's linked accounts page.

17. Reviews

17.1. A review shall be optionally submitted by a registered user.

17.2. A review shall include a rating value (1 to 5) and an optional message.

17.3. A review shall be timestamped at the time of creation.

17.5. A review may exist independently as general user feedback not tied to any LemonAid.

17.7. A review shall be categorized by type (chatbot or user_experiance).

Priority 2

1. User

1.4. A user shall be able to select between manual or bank-linked financial tracking.

1.15. A user that is an administrator shall be able to respond to support tickets.

2. Account

2.6. An account shall be able to categorize transactions.

2.9. An account shall track subscriptions.

3. Bank

3.3. A bank shall store balance and credit data for users.

4. Transaction

4.7. A transaction may be associated with a reimbursement request.

5. Subscription

5.2. A subscription shall be viewable in a centralized dashboard.

13. ReimbursementRequest

13.3. A reimbursement request shall be visible to administrators for review.

14. SupportTicket

14.3. A support ticket shall be responded to by an administrator.

14.4. A support ticket shall track the admin response and resolution state.

Unique Features

- **LemonAid AI**
  Our product uses AI to analyze spending behavior and create budgets for users.

- **Reimbursement Request System**
  This feature helps track reimbursements. These transfers are designed for Limóney users to pay each other back. Users can send each other reimbursement requests on Limóney. Users can also receive these requests and pay back vice versa.

Overall, what makes them superior to competitors is that it is an all in one app used to make dealing with money easier for the common folk. These personalized insights from the AI will make users learn at a faster rate and save money along the way.

Deployment URL:

https://limoney.org

Usability Test Plan

Test Objectives

Test the user interface intuitiveness, response flow, and ease of use for key features that differentiate Limóney from competitors. Necessary to ensure a positive user interaction with the app, encouraging them to come back because the app is easy to use and understand. To successfully test this, we will be testing five major functions from our superior features. To ensure accurate results, we enlisted the help of a tester with no prior knowledge of the application. Most of the tests will be performed by this user with little to no guidance (in the form of questions). We shall ask the user to navigate to a specific location or feature and ask them to interact with it before inquiring about their thoughts on it. The user shall be asked to locate the features for the reimbursement request and the currency converter, and what they think of these features. The user shall be pointed to LemonAid's pop-up present on the Budget page and asked what they think of it. The user shall be asked to locate and use the main feature of the app before being asked what they think said feature is and what they think of it.

Test Setup

- **Users:** College students and young professionals aged 18-infinite
- **URL:** https://limoney.org

Test Functions

1. Reimbursement Request
2. Currency Converter
3. LemonAid AI Insights
4. Budget Visualization
5. Account Tracking
6. Transactions Search

1. Reimbursement
Effectiveness Table

Test/Use Case	% completed	errors	comments
Requested money from user	70%	no actual money involved reimbursement doesnt pop up intuitively	integrate api implement proper ranking of reimbusements
Sending money as new user(shouldnt allow because no accounts)	100%	N/A	
Sending money as used user (with accounts)	80%	No actual money involved Approving or denying does nothing right now	
Clicking the buttons approval/reject	50%	does nothing no real money involved	
Clicking flagged button	80%	shows up in admin page, but doesnt really allow for any further action	
Updating accounts	30%	doesnt updated accounts	

2. Currency Converter
Effectiveness Table

Test/Use Case	% completed	errors	comments
Convert 10USD to Euro	85%	Not up to date Result: 8.50	

		Answer: 8,65	
Convert 10USD to USD	100%	wow so accurate	
Convert 10 USD to JPY	75%	Result: 1093.70 Answer: 1,471.91	
Convert 10 JPY to USD	88%	Result: 0.09 Answer:0.068	
Convert 10 JPY to Euro	88%	Result: 0.08 Answer: 0.059	
Convert 10 Euro to Jpy	79%	Result: 1302.56 Answer: 1,702.16	

3. LemonAid AI

Effectiveness Table

Test/Use Case	% completed	errors	comments
How can I cut down on food this month?	50%	None	Does not reference users' finances even when prompted to. Gives general advice.
Which subscriptions should I let go of?	50%	None	Does not reference users' finances and expects user to provide expenses. Gives general advice.
I want to save for a house. What should I do about my finances?	50%	Exception error	Does not work
Which subscriptions should I let go of? (budget page)	50%	Exception error	Does not work

Help me set realistic budget limits? (budget page)	50%	Exception error	Does not work
Give financial advice personalized	75%	Not the best formatting of writing	

4. Budget Visualization

Effectiveness Table

Test/Use Case	% completed	errors	comments
Ask for budget for past month	100%	N/A	
Ask for budget for current month	100%	N/A	
Ask for budget for future months	85%	doesnt adjust to new plans	Just wish that they was more creative freedom to changing on aspect of the budget plan
Budget easy to read?	75%	confusing for new users	
Budget ai	65%	could be refined . the outputs formats sometimes arent good and they are too long to keep the attention of the common user	
usability	75%	typing on budget is kind of annoying because the initial number wont go away	

5. Account Tracking

Effectiveness Table

Test/Use Case	% completed	errors	comments
Connecting plaid account	100%	N/A	
Creating account in private mode	100%	N/A	
Delete transaction	80%	Can only delete manual transactions	
Add transaction	100%	N/A	
Account Balance Updating when adding	100%	N/A	
Account Balance Updating when subtracting	100%	N/A	

6. Transactions Search

Effectiveness Table

Test/Use Case	% completed	errors	comments
Search netflix	100%	N/A	
Search starbucks	100%	N/A	
Check list from high to low	100%	N/A	
Search ‘nothing’	100%	N/A	
Check list for	100%	N/A	

flagged			
Check list from oldest to newst	100%	N/A	

Likert User Satisfaction

Scale:
1 – Strongly Disagree
2 – Disagree
3 – Neutral
4 – Agree
5 – Strongly Agree

1. Reimbursement Request

1. I found it easy to submit a reimbursement request through the system.
2. The system clearly communicated the status of my reimbursement requests.
3. I felt confident that the reimbursement process worked as expected.

2. Currency Converter

4. I found the currency converter feature easy to use.
5. The exchange rates provided by the system were clear and understandable.
6. The currency conversion results felt accurate and trustworthy.

3. LemonAid AI Insights

7. The AI insights I received were relevant to my financial habits.
8. The assistant's suggestions helped me improve my spending decisions.

9. I felt that the LemonAid assistant added meaningful value to the app.

4. Budget Visualization

10. The budget charts and graphs made it easy to understand my finances.
11. The visual layout of my budgets was clear and well-organized.
12. Tracking spending visually helped me stay on top of my budget.

5. Account Tracking

13. I could easily view and manage my linked accounts in one place.
14. The system correctly grouped my accounts by type (e.g., savings, checking).
15. I felt confident that my account balances were accurately reflected.

6. Transactions Search

16. I was able to quickly find specific transactions using the search feature.
17. The filters available in transaction search made it easier to narrow results.
18. I found the transaction search feature intuitive and helpful.

QA Test Plan

Selected Non-Functional Requirements

1. Performance
2. Usability

3. Load Tolerance
4. Reliability
5. Compatibility

QA Test Table

1. Performance

Test Objectives: Ensure the system responds to user actions within 2 seconds under normal load.
HW and SW Setup:
-Hardware: AWS EC2 t2.micro, laptop
-Software: Ubuntu 22.04, Node.js backend, React frontend
-URL: https://limoney.org/signin
Feature to be Tested: Page and feature responsiveness (login, navigation, account linking)
QA Test Plan:

Test No.	Title	Task Description	Input	Expected Output	Result
1	Load Dashboard	User logs in and is redirected to dashboard	Login credentials	Dashboard UI loads and becomes interactive in ≤ 2 seconds	<1 second PASS
2	Add New Account	User clicks link bank account and enters their bank credentials	Bank login credentials	about 15 seconds	Took a little while for the accounts to load about 20 seconds FAIL
3	Navigate to Accounts Page	User clicks on accounts page from dashboard	Click on navigation tab	Accounts page takes about 3 seconds to load all the accounts	about 1 second PASS

2. Usability
Test Objectives: Ensure the UI is intuitive and users can perform key tasks without documentation.
HW and SW Setup:

-Hardware: Laptop or mobile
-Software: React frontend tested on Chrome, Firefox, Safari
-URL: https://limoney.org
Feature to be Tested: Ease of use and understanding of the navigation and main pages

Test No.	Title	Task Description	Input	Expected Output	Result
1	Usage of Accounts page	User clicks on Accounts page in Navbar	Click accounts tab	User understands the purpose of the Accounts page	User understood majority purpose, was not immediately sure of benefit PASS
2	Usage of Budgets page	User clicks on Budget page in Navbar	Click budgets tab	User understands purpose of the Budget page and how to create an budget	User struggled to understand the purpose of the page due to being able to change passed month's budgets FAIL
3	Usage of Dashboard	User clicks on Dashboard in Navbar	Click dashboard tab	User understands purpose of Dashboard page as a summary of their financial status	User understood what the dashboard was for, had momentary confusion regarding create a reimburseme nt request

					PASS

3. Scalability
Test Objectives: Ensure AI tools respond within 5 seconds even under moderate system load.
HW and SW Setup:
-Hardware: AWS EC2
-Software: Node.js, AI recommendation module
-URL: https://limoney.org
Feature to be Tested: Response time of Budget AI and Lemonaid chatbot

Test No.	Title	Task Description	Input	Expected Output	Result
1	Budget AI Suggestion	AI generates budget from user's transactions	User types prompt to budget ai asking for budget to be generated	Ai recommends in less than 5 seconds	Makes budget in about 2 seconds PASS
2	Lemonaid Advice	AI gives personal financial advice	User types prompt asking lemonaid chat for financial help	Ai gives through and personal advice	Makes response in about 3-5 seconds PASS
3	Budget AI Save	User accepts AI suggestion, DB/UI updates	User accepts generated budget	It updates the db and ui for user	PASS

4. .Reliability:The system shall maintain at least 99.5% uptime over a 30-day period. **DONE**
Test Objectives: Ensure the system maintains at least 99.5% uptime over a 30-day period.
HW and SW Setup:
-Hardware: AWS EC2 with uptime monitoring
-Software: PM2 (process manager), Nginx
-URL: https://limoney.org
Feature to be Tested: Service uptime consistency

Test No.	Title	Task	Input	Expected	Result

		Description		Output	
1	Up for 5 days	Monitor uptime for 5 days	Up 5 days	System stays up 5 days	PASS
2	Up for 15 days	Monitor uptime for 15 days	Up 15 days	System stays up 15 days	PASS
3	Up for 30 days	Monitor uptime for 30 days	Up 30 days	System stays up 30 days	PASS

5. Compatibility: The system shall support the following browsers:
Test Objectives: Ensure the system supports modern browsers (Chrome, Firefox, Edge, Safari).
HW and SW Setup:
-Hardware: Laptop
-Software: Browsers: Chrome v80+, Firefox v75+, Edge v80+, Safari v11.1.1+
-URL: https://limoney.org
Feature to be Tested: Site compatibility with various browsers:

• Google Chrome v80+ • Mozilla Firefox v75+ • Microsoft Edge v80+ • Safari v11.1.1+

Test No.	Title	Task Description	Input	Expected Output	Result
1	Google Chrome	Open app onGoogle Chrome	User opens on google chrome	It works with all its functions	PASS
2	Mozilla Firefox	Open app on Firefox	User opens on mozilla firefox	It works with all its functions	PASS
3	Microsoft Edge	Open app on Edge	User opens on microsoft Edge	It works with all its functions	PASS

Localization Testing

Tasks:

1. Home page
2. Budget page
3. About page

Translations: Spanish and Hindi

Cultural Nuances:

- Spanish dialects containing different vocabulary and phrases for Hispanic users of different backgrounds
- Hindi and Hinglish for North Indians who speak other languages than Hindi, use English as a lingua franca, and those who can't read Hindi, speak pure Hindi or English.

Regional Preferences:

Argentina, Mexico, Spain, and North India.

Sample Test Case:

Page	Test	Translation OK?
Home.js	Test es to es-MX to es-AR and hi to hi-Latin	Yes
Budget.js	Test es to es-MX to es-AR and hi to hi-Latin	Yes
About.js	Test es to es-MX to es-AR and hi to hi-Latin	Yes

Results

Most UI text components passed the translation tests. Due to time constraints, there are some pages in user layout that were not translated such as the subpages in settings. Hinglish could use more English loan words and the hooks could use a different idiom that sounds more native in different languages.

Code Review

Coding Standards

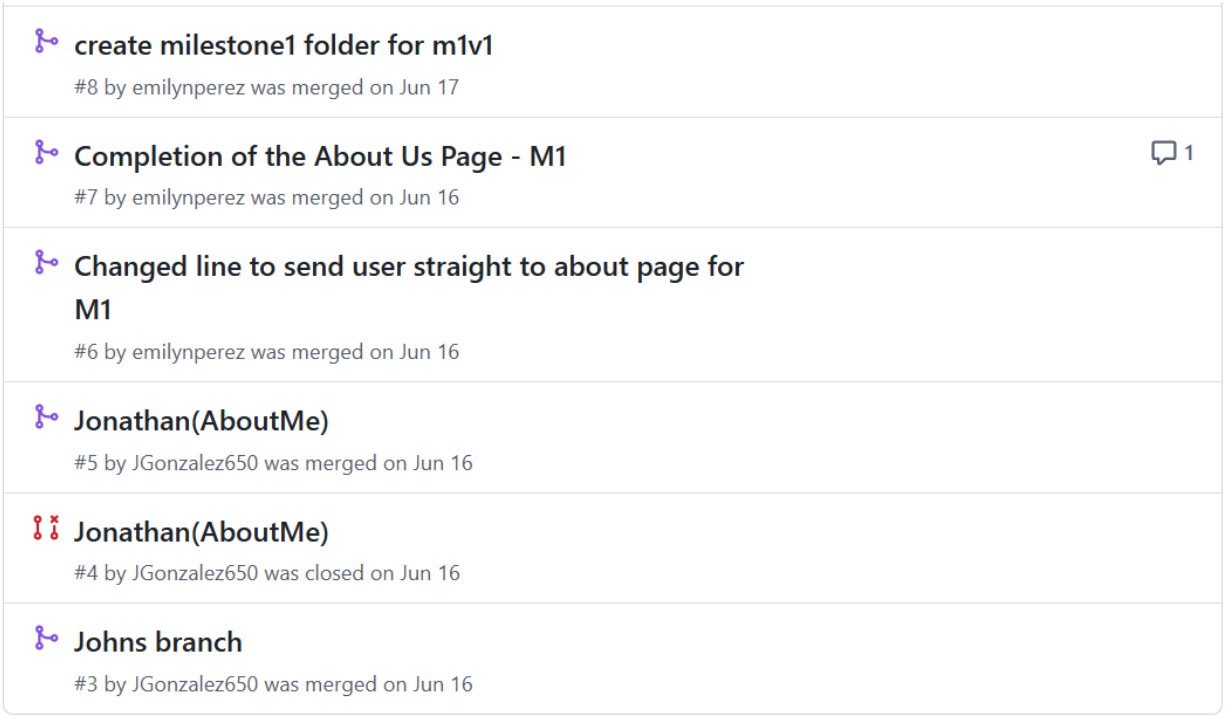
Here are the coding/working standards I have on a doc shared with all of my team members to ensure unity and reflecting what is asked from the nonfunctional requirements.

Coding standards:

- Name variables under camelCase convention.
    - Ex:
    - camelCase
    - somethingCool
    - iDontKnowMan
- Always work within your own branch and have it be named after the feature you are creating. Try not to create many features at once!
- Never push code onto development branch with code that is not functional or that you are still testing.
- Make sure the commits have proper names!
- Make comments throughout the code so anyone looking at it can understand what is going on.
- Always encrypt vulnerable information like database information or passwords of users
- Never hardcode ports or http://localhost. There are environment variables made for that.

Github Code Review

MILESTONE 1:


create milestone1 folder for m1v1
#8 by emilynperez was merged on Jun 17
Completion of the About Us Page - M1
1
#7 by emilynperez was merged on Jun 16
Changed line to send user straight to about page for M1
#6 by emilynperez was merged on Jun 16
Jonathan(AboutMe)
#5 by JGonzalez650 was merged on Jun 16
Jonathan(AboutMe)
#4 by JGonzalez650 was closed on Jun 16
Johns branch
#3 by JGonzalez650 was merged on Jun 16

Completion of the About Us Page - M1 #7

Merged **dluna2** merged 7 commits into `master` from `development` on Jun 16

Conversation 1 | Commits 7 | Checks 0 | Files changed 7

emilynperez commented on Jun 16

No description provided.

JGonzalez650 and others added 7 commits 2 months ago

Jonathan(AboutMe)		4f038ff
Merge pull request #5 from sfsu-joseo/john-branch2	Verified	6d7bbdc
Updated About Us		ecf9994
added logo and description to my profile		6a68f14
added description about myself and put logo		e6f4353
fixed typos		1ab3b5e
accepted Ishaank's push		ea19457

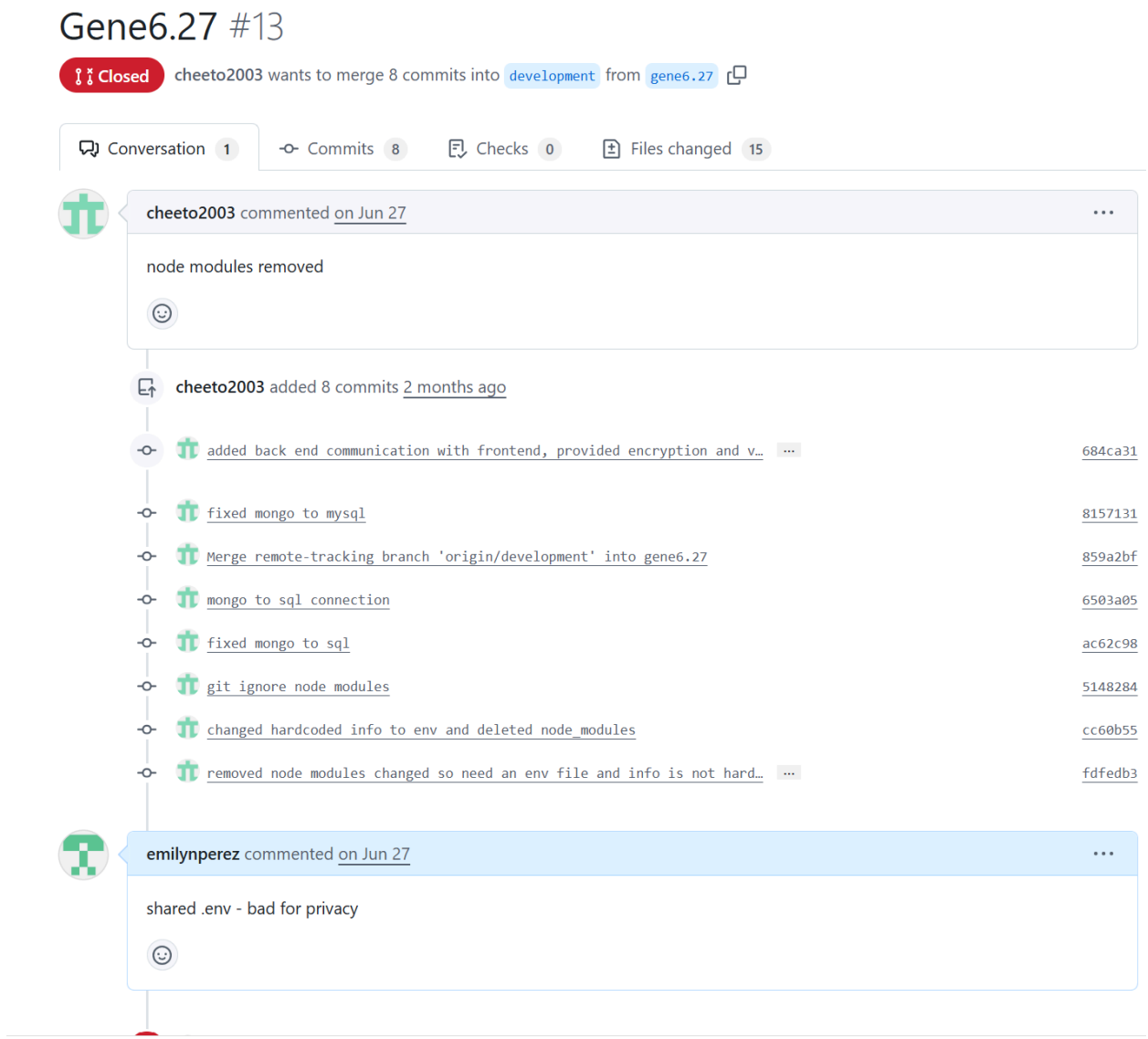
dluna2 commented on Jun 16

will merge

dluna2 merged commit `7ffb7e6` into `master` on Jun 16 — Revert

MILESTONE 2:

- Delete milestone2 directory
  #24 by emilynperez was merged on Jul 2
- Updated Database Organization ✓
  #23 by izalpuri-creator was merged on Jul 1
- modifed changes...deleted and kept
  #22 by cheeto2003 was merged on Jun 30
- Update README.md into accessing server 1
  #21 by cheeto2003 was closed on Jun 30
- Update README.md to accessing server ✓ 1
  #20 by cheeto2003 was closed on Jun 30
- Update README.md to accessing server ✓ 1
  #19 by cheeto2003 was closed on Jun 30 • Review required
- Created Dockerfile compose
  #18 by emilynperez was merged on Jun 30
- Updating .gitignore in root
  #17 by emilynperez was closed on Jun 27
- Updating global .gitignore
  #16 by emilynperez was merged on Jun 27
- removed git idea
  #15 by cheeto2003 was merged on Jun 27
- Gene6.27 1
  #14 by cheeto2003 was closed on Jun 27
- Gene6.27 1
  #13 by cheeto2003 was closed on Jun 27
- mongo -> mysql 1
  #12 by cheeto2003 was closed on Jun 27
- Clean up of code/directory + added signup page + search bar for milestone 2 (no functionalities)
  #11 by emilynperez was merged on Jun 26


Gene6.27 #13
Closed
cheeto2003 wants to merge 8 commits into development from gene6.27
Conversation 1
Commits 8
Checks 0
Files changed 15
cheeto2003 commented on Jun 27
node modules removed
cheeto2003 added 8 commits 2 months ago
added back end communication with frontend, provided encryption and v… 684ca31
fixed mongo to mysql 8157131
Merge remote-tracking branch 'origin/development' into gene6.27 859a2bf
mongo to sql connection 6503a05
fixed mongo to sql ac62c98
git ignore node modules 5148284
changed hardcoded info to env and deleted node_modules cc60b55
removed node modules changed so need an env file and info is not hard… fdfedb3
emilynperez commented on Jun 27
shared .env - bad for privacy

MILESTONE 3:

- Budget dashboard animation
  #92 by dluna2 was merged 2 weeks ago
- prettied up transaction page 1
  #91 by cheeto2003 was merged 2 weeks ago
- added kebab functionality
  #90 by cheeto2003 was closed 2 weeks ago
- lined up transaction page to look like figma
  #89 by cheeto2003 was closed 2 weeks ago
- modified search for searching and loading transactions
  #88 by cheeto2003 was merged 2 weeks ago
- Budget dashboard animation
  #87 by dluna2 was merged 2 weeks ago
- SettingsPageUI
  #86 by JGonzalez650 was merged 2 weeks ago
- transactoin and search is now live!
  #85 by cheeto2003 was merged 3 weeks ago
- logout fix
  #84 by andrewb-03 was closed 2 weeks ago
- Organized Directory
  #83 by izalpuri-creator was merged 3 weeks ago
- Fix public pages
  #82 by emilynperez was merged 3 weeks ago
- Modded accounts & lemon aid page to look pretty and match color scheme :)
  #81 by cheeto2003 was merged 3 weeks ago

prettied up transaction page #91

Merged **dluna2** merged 5 commits into `development` from `modded_search` 2 weeks ago

Conversation 1 | Commits 5 | Checks 0 | Files changed 3

cheeto2003 commented 2 weeks ago

prettied up transaction page

cheeto2003 and others added 5 commits 2 weeks ago

- `lined up transaction page to look like figma` `0dd4290`
- `added kebab functionality` `5a144a1`
- `added kebab functionality` `fef229c`
- `fixed search constantly searching for you` `802561b`
- `Merge branch 'development' of github.com:sfsu-joseo/csc648-848-01-su2…` `b4c3b74`

dluna2 commented 2 weeks ago

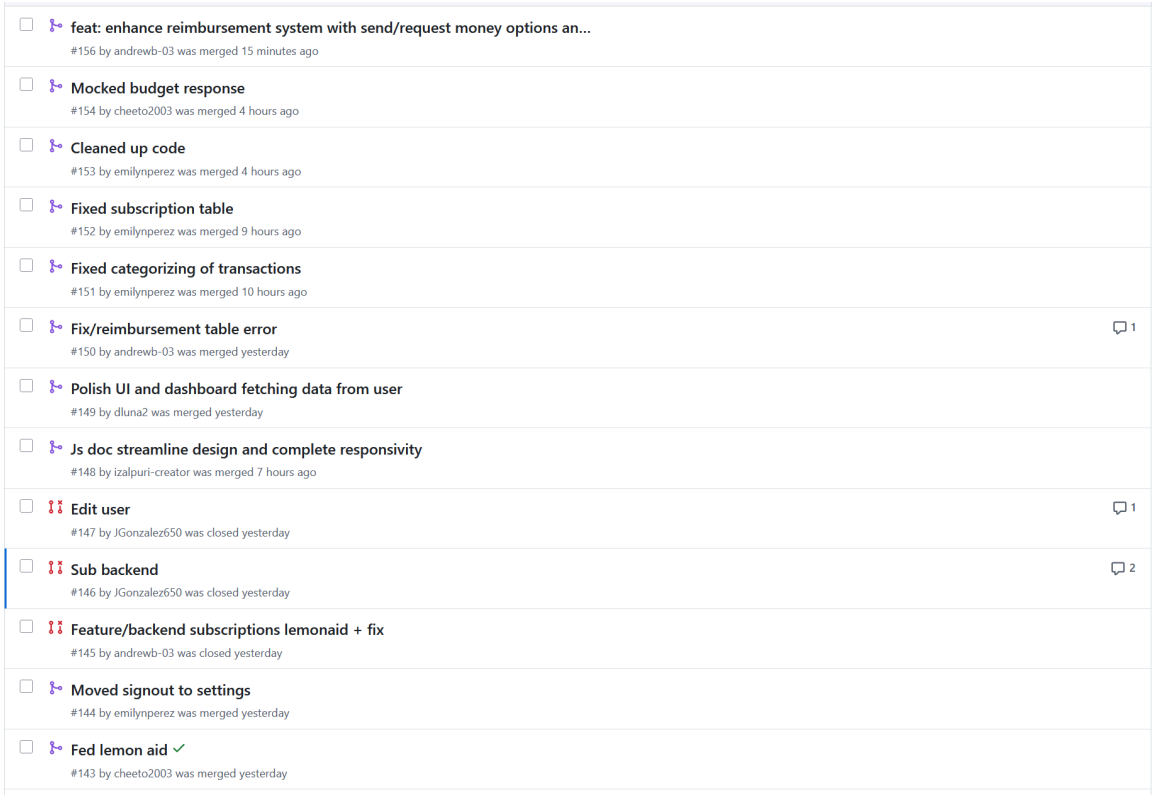
great work. we need to change the trash icon for deleting.

dluna2 merged commit `8e0d281` into `development` 2 weeks ago — Revert

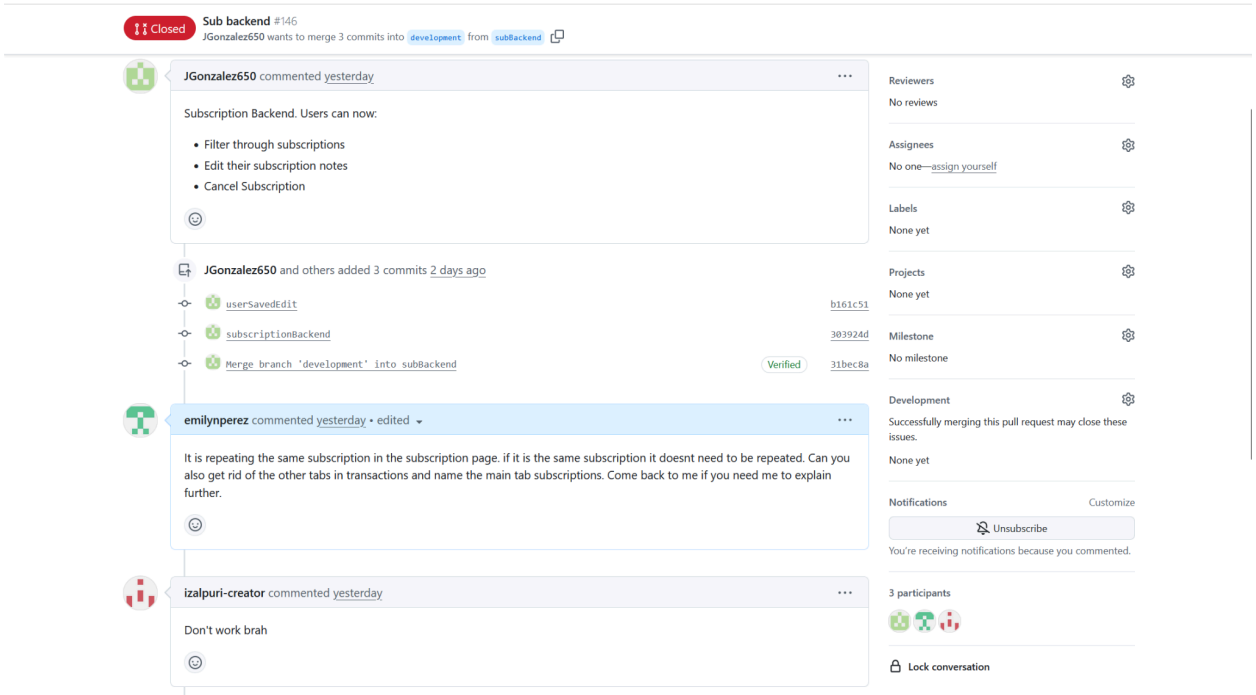
emilynperez deleted the `modded_search` branch 2 weeks ago — Restore branch

MILESTONE 4:

Sample of some of our pull requests:



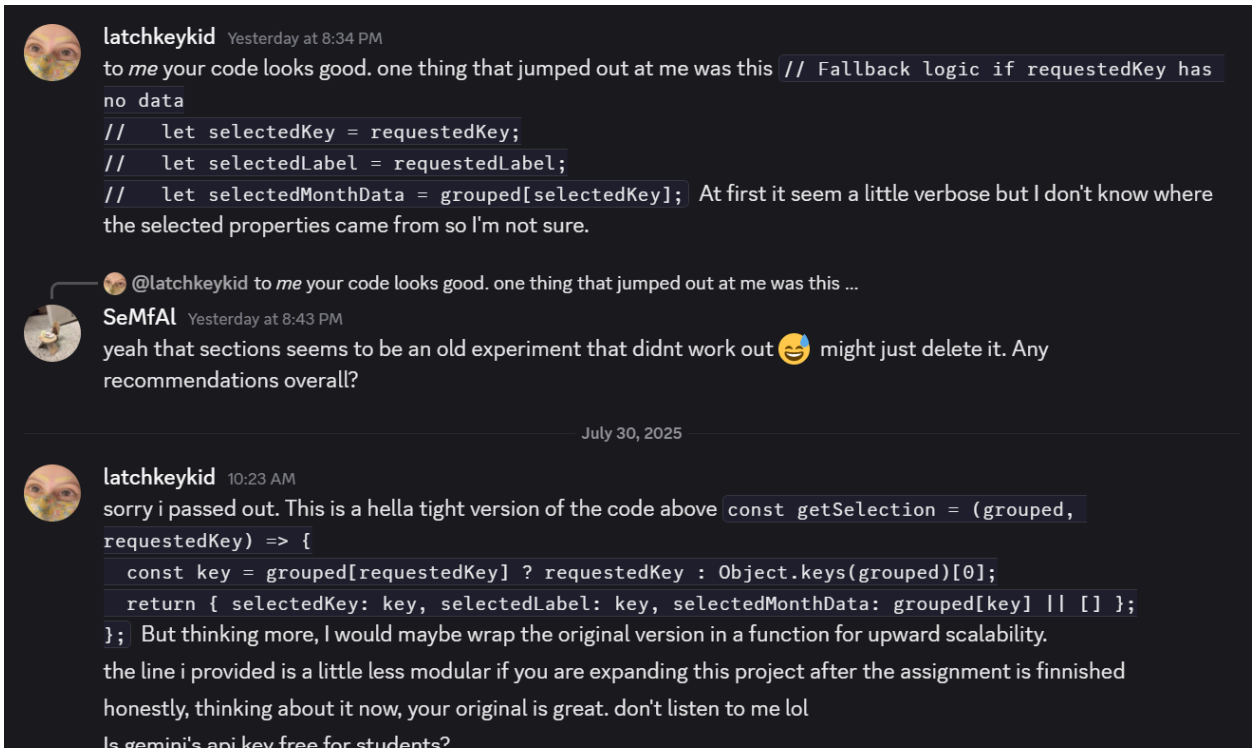
Sample of comments in a pull request:



External Review

Team 5 reviewed our budget and budget ai backend code. We decided to message through discord because we couldn't open eachother's files on our school emails.

- Names are too verbose. In milestone 5, we will clean and refine our code.
  Photo proof:



Security Practices

Assets Protected

- User personal info
- Banking credentials
- Transaction history
- AI recommendations

Password Encryption

- Bcrypt (12 salt rounds)
- Stored hashed in MySQL

```
mysql> SELECT * FROM User;
+--------+------------------+----------------------+--------------------------------------------------------------+----------+
| userId | name             | email                | password                                                     | userType |
+--------+------------------+----------------------+--------------------------------------------------------------+----------+
|      3 | test2            | test2@test.com       | $2b$10$Kj3QWcxjtucli.tM57TLWeVGYlQTd8D7M/R4tgOeo17jzZsZip4Ri | standard |
|      4 | test             | test@pleasework.com  | $2b$10$L060/fSX.DWsEiqM3c8Kqud1iIOVvoqIaG6Ht9iXzJk51KKvcHmrO | standard |
|      5 | 123              | 123@123.com          | $2b$10$vlj2lJXhQYCoUQ6g6HUWreT1HwxznsqEmTk6RZDsBH1cYuR/kCfQu | standard |
|      6 | rawr             | rawr@oowow.com       | $2b$10$H4KDEN/gnOXR7PFcM/HOKeH3g0trVhQox6q0MMQSpYMrWo1W6J6bq | standard |
|      8 | testing          | 123@gmail.com        | $2b$10$Ff7FziFTcMe9rh1EY8Pb8uxLm12PjsaqQ6k4jGO.EkduTXS7P5Hcm | standard |
|      9 | test2            | testing@gmail.com    | $2b$10$LrAm.bTu7OCYbnMSdE1XWeKcqby51UCh52pDURo1E7BnsfEM787cW | standard |
|     10 | random           | test@work.com        | $2b$10$blYOVzNfISPSEQqGsr/Ate.U0ggIgR/qIvBcrw1VNxjWq9Yb5ydC2 | standard |
|     11 | Ishaank          | izalpuri@gmail.com   | $2b$10$5zCnglj4cnXFIDto2cMiiOUNnWil3fQZszvCZQeM7vP9CVk1zlJC6 | standard |
|     12 | Alice Johnson    | alice@example.com    | hashedpass1                                                  | standard |
|     13 | Bob Lee          | bob@example.com      | hashedpass2                                                  | standard |
|     14 | Carol Kim        | carol@example.com    | hashedpass3                                                  | standard |
|     15 | fdafkldjasfdas   | sleep@sleep.com      | $2b$10$JgY9.ipudFRTn.2w/lo9wu9JMoK79sFEPv2FkqbBMKl8X9n079Cty | standard |
|     16 | Dechen           | dechen2020l@icloud.com | $2b$10$uQoi0Gi8eQCmw1zzYgMD4uJfa5eqtzEq6fn/exUeLOgZzqJ88PNsC | standard |
|     17 | Jose             | joseortizcosta@aol.com | $2b$10$l7gaI2owfsayQUAjuhs1feP9YJZG5hjdtwwemuoBZYSjoMQge6udS | standard |
|     18 | Hii              | 1@g.com              | $2b$10$HyQX7i9fSWYcX80Rf7eyVOPRJY6/MqWJMdCrrHITeTv6HjPPIWY1K | standard |
|     20 | bla              | bla@gmail.com        | $2b$10$c7cteCK0ZtgIFvIV08SX9O.tgKoJoT/NJV5rVaxJIKc2DqD/9iQA2 | standard |
|     24 | test2            | testing2@gmail.com   | $2b$10$07VH/eHSQginFyqnnvERmODMU06laYPYmi1DOG9zcVM0hyNjwhQHy | standard |
|     25 | test             | test@test.comte      | $2b$10$Zu9VIOb.XYSYoBnl/.Fmh.TenUV981d/E38WnqUCQWkDHQWTz/bQu | standard |
|     27 | usertestFP       | usertestfp@example.com | $2b$10$XQ8fdjFRDr4qyR40F97fAuCj2VT.rOzOkigwg/FWJy.raVNcIi5fy | standard |
|     28 | china            | china@china.com      | $2b$10$rHFSQBxDt1fHDqEoD9z85.OoR4l.nhQR/kgWcH5hf9RcU3/eBigDG | standard |
|     31 | test3            | test23@test.com      | $2b$10$zCs9WEs85zWvjh9XrCuYk.LgTBfG7vRzKNhqbuQJ0N/n0nbL4FiYa | standard |
|     42 | test             | test1@test.com       | $2b$10$ldxmZa6EIVou1KV66Q7/ieuKbrvb8av7AYIxp8unjnK50w8gQztce | standard |
|     55 | test2            | test@test.com        | $2b$10$H0EmdvTt9qVu9w3xhoLSkeIz65Z9Hr3.KXMzz85tgRDbEHE8TYQey | standard |
|     57 | 123              | 21@test.com          | $2b$10$LvdfonsSncnOjw.jOkG/h.D9mPaSsdHABep8forjT6/oEijbKAPPG | standard |
|     58 | testreimbursement | reimbursement@test.com | $2b$10$USk5TEGH/b9YlBvwNPUQ8u/ardJkhCLBpMCGzKp4Ycws2jt5FPNfK | standard |
|     59 | Recipient User   | recipient@test.com   | $2b$10$Me8gyTiMvlPgDEXY9aIvd.ae0oabH5Z.d7i6YCHSwgaNJSgXuHeii | standard |
|     60 | Sender User      | sender@test.com      | $2b$10$dV7QVme0WcjOei3bMBlB6.EaOE4RnO.7.1R6SQoEYW4MxL7GCp04. | standard |
+--------+------------------+----------------------+--------------------------------------------------------------+----------+
```

Input Validation

Validated inputs: email, password, search

- Signing up validation : email, password (at least 6 characters)

```
name = typeof name === 'string' ? name.trim() : '';
```

```
email = typeof email === 'string' ? email.trim().toLowerCase() : '';
```

```
    password = typeof password === 'string' ? password.trim() : '';

    if (!name || !email || !password) {
      return res.status(400).json({ message: 'All fields are
required' });
    }

    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailRegex.test(email)) {
      return res.status(400).json({ message: 'Invalid email format'
});
    }

    if (password.length < 6) {
      return res.status(400).json({ message: 'Password must be at
least 6 characters' });
    }

    try {
      const existing = await UserModel.findUserByEmail(db, email);
      if (existing) {
        return res.status(409).json({ message: 'Email already
registered' });
      }

      const hashedPassword = await bcrypt.hash(password, 10);
      await UserModel.createUser(db, name, email, hashedPassword);

      console.log(`New user created: ${email}`);
      res.json({ message: `Thanks for signing up, ${name}!` });
    } catch (err) {
      console.error('Error saving user:', err);
      res.status(500).json({ message: 'Server error', error:
err.message });
    }
  };
```

- Search:
  ensures an empty search isn’t sent

```
if (query.trim()) params.append('query', query);
```

Self-Check: Adherence to Non-Functional Specs

List of Non-Functional Requirements

1. Performance
1.1. The system shall respond to user actions within 2 seconds under normal load. **DONE**
1.2. Backend APIs shall respond to simple requests in under 200 milliseconds. **DONE**
1.3. The dashboard shall load within 3 seconds over a stable 10 Mbps connection. **DONE**
1.4. AI recommendations shall be delivered within 5 seconds of user input. **DONE**
1.5. The system shall support real-time syncing of transactions without freezing the UI. **DONE??**

2. Security
2.1. All communications shall be encrypted using HTTPS and TLS protocols. **DONE**
2.2. All sensitive data, including user credentials, shall be encrypted. **DONE**
2.3. Passwords shall be hashed using bcrypt or equivalent secure hashing algorithms. **DONE**
2.4. Role-based access control shall be implemented to separate admin and user privileges. 2.5. The system shall prevent common web vulnerabilities such as SQL injection, XSS, and CSRF. **DONE**
2.6. Only authenticated and authorized users shall be able to change, delete, or insert sensitive data.**DONE??**

3. Scalability
3.1. The system shall support up to 500 concurrent users without performance degradation. **DONE**
3.2. The database shall support horizontal scaling across distributed servers via Docker and AWS. **DONE**
3.3. The system shall efficiently scale with a 10x increase in user data and transaction volume. **DONE**
3.4. Database schema shall support sharding for large financial datasets. **DONE**

4. Usability
4.1. The UI shall be intuitive and require no documentation for key tasks such as setting up budgets or tracking expenses. **DONE**
4.2. The system shall support both dark mode and light mode themes. **DONE**
4.3. The AI assistant shall use natural, conversational language to support non-technical users. **DONE**
4.4. Navigation, task completion, and interaction shall follow established UX best practices. **DONE**

5. Reliability
5.1. The system shall maintain at least 99.5% uptime over a 30-day period. **DONE**
5.2. Transaction data shall be backed up every 12 hours automatically. **DONE**
5.3. Background processes such as syncing and reminders shall automatically retry on failure. **DONE**
5.4. Error handling shall provide feedback to users without crashing the system. **DONE**

6. Maintainability
6.1. The codebase shall follow JavaScript linting rules using ESLint and Prettier. **DONE**
6.2. Docker containers shall be used to ensure parity between local and production environments. **DONE**
6.3. The codebase shall follow a modular architecture to isolate and decouple components. **DONE**
6.4. Critical modules shall maintain a minimum of 70% unit test coverage. **DONE**

6.5. Version control shall be managed through Git and GitHub with frequent commits and reviews. **DONE**

7. Portability
7.1. Docker containers shall allow the app to run across local development, staging, and AWS cloud environments. **DONE**
7.2. The app shall run identically across Windows, macOS, and Linux-based systems. **DONE**

8. Compatibility
8.1. The system shall integrate with major U.S. banks using secure open banking APIs. **DONE**
8.2. The AI engine (for ex: ChatGPT API or Gemini API) shall be modular and replaceable without impacting core functionality. **DONE**
8.3. The system shall support the following browsers: **DONE**
• Google Chrome v80+ • Mozilla Firefox v75+ • Microsoft Edge v80+ • Safari v11.1.1+

9. Privacy
9.1. The app shall allow users to manually enter financial data without linking any bank accounts. **DONE**
9.2. User data shall not be sold to third-party services. **DONE**
9.3. All stored data shall remain local unless explicit user consent is given. **DONE**
9.4. Privacy mode shall restrict AI access to only local anonymized data. **DONE**

10. Compliance
10.1. The system shall comply with GDPR regulations for data privacy and user consent. **DONE**
10.2. Users shall be able to permanently delete their data on request. **DONE**
10.3. Consent prompts shall be clearly visible during signup.**DONE**

11. Supportability
11.1. System logs shall track user behavior, feature usage, and AI errors for debugging. **DONE**
11.2. An admin dashboard shall allow staff to review flagged transactions and feedback. **DONE**
11.3. The system shall support browser versions listed under Compatibility. **DONE**
11.4. Support documentation shall be maintained for installation, setup, and deployment. **DONE**

12. Efficiency
12.1. The system shall operate at no more than 60% CPU usage under normal load conditions. **DONE**
12.2. Memory usage shall not exceed 75% of allocated memory on the EC2 instance. **DONE**
12.3. Efficient logging shall be implemented to prevent log flooding and performance drag. **DONE**

13. Look and Feel
13.1. The system shall maintain a clean, minimal aesthetic with responsive design. **DONE**
13.2. UI elements shall follow a cohesive design language across all pages and devices. **DONE**
13.3. Accessibility standards (WCAG 2.1 AA) shall be considered for inclusive design. **DONE**

14. Coding Standards
14.1. All backend code shall conform to Node.js and Express.js best practices. **DONE**
14.2. Frontend code shall follow React component structure and state management conventions. **DONE**
14.3. Continuous Integration tools shall run linting and testing on every merge request.**DONE**

15. Environmental Sustainability

15.1. The system shall optimize backend server usage to reduce idle time and energy waste. **DONE**

15.2. Docker containers shall be configured to auto-scale down during periods of low usage. **DONE**

15.3. The app shall minimize data transfer size (e.g., compress JSON responses, lazy-load images) to reduce bandwidth and energy consumption. **DONE**
15.4. The frontend shall use efficient rendering practices in React (e.g., avoiding unnecessary re-renders, using memoization).**DONE**
15.5. System architecture shall consider the use of renewable-energy-based data centers (e.g., AWS sustainability zones).**DONE**
15.6. Logs and backups shall be archived using energy-efficient cold storage where applicable. **DONE**

15.7. Documentation shall be provided digitally only (no paper printing by default). **DONE**
15.8. The system shall avoid unnecessary background polling or constant sync unless essential for real-time use.**DONE**

List of Team Contributions

Name	Contributions	Score
Emily Perez	• set up live compiling for easier coding process • set up different environment for production vs development • implemented sessions • implemented plaid api • made accounts page dynamic to user • fixed transactions to where each transaction is put into the correct account • Implemented database for usertransactions, plaiditem, • deployed onto site • dealt with pull requests and merging • helped other debug their problems • implemented categorization using plaid api • implemented subscription into db using plaidapi	9
Ishaank Zalpuri	• made all public pages responsive to various device sizes • helped with m4v1 • attempted at localization	8
Andrew Brockenborough	• refactored server.js so it is cleaner, • helped with m4v1, • added tools icon on userlogin navbar and connected the pages • implemented backend for reimbursement request	6

	○ receive ○ send	
Dani Luna	● created todolist; ● inserted background colors for public layout, balance forecast, dashboard and lemonaid; ● turned off horizontal scrolling on budget page; ● helped with m4v1 ● attempted at localization ● drew background for lemonaidchat; ● fixed cut off for piechart	7
Jonathan Gonzalez	● implemented back end change email and name ● implement forgot password backend ● implemented forgot password frontend ● fixed ui for subsctiptions	7
Gene Orias	● added fake ssl certificates for coding environment ● implemented cookies, ● secured userloginpages for users only ● made userloginpages adjust to the user, ● implemented backend for settings: ○ editing name ○ notifications ○ delete account ● implemented backend for budget ● implemented backend for budget ai generator ● implemented backend for lemonaid general chat	10

Team Member Contributions

Name	Contributions	Score
Emily Perez	**MILESTONE 5** ● dealt with pull requests and merging errors ● helped debug ● assigned tasks ● fixed m4v1 ● finalzed m5 ● deploy site onto server **MILESTONE 4** ● set up live compiling for easier coding process ● set up different environment for production vs development ● implemented sessions ● implemented plaid api	10

	● made accounts page dynamic to user ● fixed transactions to where each transaction is put into the correct account ● Implemented database for usertransactions, plaiditem, ● deployed onto site ● dealt with pull requests and merging ● helped other debug their problems ● implemented categorization using plaid api ● implemented subscription into db using plaidapi **MILESTONE 3** ● started and finished tech doc for m3, ● fixed fetch errors, ● refined data definitions, ● refined erd, ● created eer, ● set up guide for setting up working environment, ● connected pages together, ● set up and enforced code standards, ● handled pull requests, ● assigned tasks to members, ● implemented final database(forward engineered eer), ● adjusted algorithms to new database, ● implemented all parts of the settings page, ● implemented public page ui, ● got domain for site, ● got ssl certificates, ● implemented ui for reimbursement requests	

	ranking algo, ● deployed code onto server **MILESTONE 2** ● wrote key project risks; ● wrote title page; ● finalized high-level functional requirements; organized and finalized technical documentation; ● assisted with setting up the directory; ● helped establish connection between frontend, backend, and database **MILESTONE 1** ● Wrote executive summary, ● wrote high level system..., ● set up cloud server, helped with credentials md, ● helped with main use cases	
Ishaank Zalpuri	**MILESTONE 5** ● integrated private mode ● fixed lemon ui ● cleaned up all eslint disable lines **MILESTONE 4** ● made all public pages responsive to various device sizes ● helped with m4v1 ● attempted at localization **MILESTONE 3** ● helped with UI/UX Wireframes, ● created accounts page in the userlogin pages, ● organized file directory, ● implemented ui for delete accounts, ● implemented ui for transactions,	8

	• fixed figma to new implemented design, • implemented ui for adding accounts **MILESTONE 2** • wrote data definitions; • helped with database architecture; • helped with high-level functional requirements **MILESTONE 1** • Wrote list of main data..., • wrote main use cases, wrote checklist, • helped set up permissions with cloud server	

Andrew Brockenborough	**MILESTONE 5**
	• set up general user feedback ui
	• set up general user feedback backend
	• set up webhooks for continuous syncing of transactions from plaid
	MILESTONE 4
	• refactored server.js so it is cleaner,
	• helped with m4v1,
	• added tools icon on userlogin navbar and connected the pages
	• implemented backend for reimbursement request
	  ○ receive
	  ○ send
	MILESTONE 3
	• Added navbar for the userlogin pages,
	• created parts of the settings page,
	• helped with organizing the technical documentation,
	• kept the readmemd up to date,
	MILESTONE 2
		6

	• created table of contents; updated readme.md file in application folder; helped with high level apis and main algorithms; helped with high-level functional requirements **MILESTONE 1** • wrote competitive analysis, • helped w main use cases,	
Dani Luna	**MILESTONE 5** • set up localization • tested for localization • set up dark mode for ui • helped with merge conflicts **MILESTONE 4** • created todolist; • inserted background colors for public layout, balance forecast, dashboard and lemonaid; • turned off horizontal scrolling on budget page; • helped with m4v1 • attempted at localization • drew background for lemonaidchat; • fixed cut off for piechart **MILESTONE 3** • led and designed the UI/UX Figma Wireframes, • created TODO list, • implemented ui for budget page, • helped handle pull requests, • implemented ui for balance forecas, • implemented ui for dashboard **MILESTONE 2**	8

	● created todo list; ● wrote data definitions; ● helped with database architecture; ● created mockups/storyboards; ● worked on the search bar algorithm for m2c2; ● helped with high-level functional requirements **MILESTONE 1** ● made the title page, ● wrote main use cases, ● set up aboutus page,	
Jonathan Gonzalez	**MILESTONE 5** ● set up admin front end ● set up admin backend **MILESTONE 4** ● implemented back end change email and name ● implement forgot password backend ● implemented forgot password frontend **MILESTONE 3** ● fixed ui for subsctiptions ● fixed uml diagram for m3, ● helped with public page ui implementation, ● fixed ui for settings, ● implemented reimbursement request and its functionalities **MILESTONE 2** ● helped with high-level functional requirements; ● worked on backend architecture **MILESTONE 1** ● wrote main use cases, ● wrote functional req, ● wrote non functional req,	7
Gene Orias	**MILESTONE 5** ● implemented manual	10

	transactions ● refined budget chatbot ai backend **MILESTONE 4** ● added fake ssl certificates for coding environment ● implemented cookies, ● secured userloginpages for users only ● made userloginpages adjust to the user, ● implemented backend for settings: ○ editing name ○ notifications ○ delete account ● implemented backend for budget ● implemented backend for budget ai generator ● implemented backend for lemonaid general chat **MILESTONE 3** ● implemented search functionality logic, ● implemented rating functionality logic, ● set up all the jsx files (pages), ● set up gemini api for lemonaidchat, ● fixed ui for lemonaid chat, ● fixed ui for accounts page, ● set up ui for transactions of accounts, ● helped organize userloginpages for coordination, **MILESTONE 2** ● establish connection between frontend, backend, and database; ● completed the signup aspect of m2c2; ● updated credentials files;	

	● encrypted server data into code (.env); ● worked on high-level application network protocols and deployment design; ● helped with high-level functional requirements **MILESTONE 1** ● wrote table of contents, ● wrote main use cases, ● helped with credentials md,	

Post-Analysis – Lessons Learned

This class is certainly an experience of all the emotions. It was both thrilling and draining but at the end, we all learned so much through this class. We encountered so many challenges together.

Challenges:

1. **Adapting Quickly**
   There have been many cases where we as a group had to adapt fast and quickly to sudden changes. For example, we relied on gitbook to accumulate our writing for the milestones but after the trial expired it became excruciatingly inconvenient to use because we couldn't export it as pdf. For easier editing for the team lead, we chose to move all of our writing into google docs because it was so much easier to work with than gitbook.
2. **Learning to Work with Others**
   Working in a fast-paced environment exposed the difficulties of team coordination. There were times when tasks were assigned to members who couldn't complete them on time, which led to others having to step in at the last minute. With better planning and more transparent communication, these situations could have been avoided.
   One of the most valuable lessons was learning how to navigate team dynamics. People had different levels of commitment, motivation, and capacity. Some wanted to contribute more but struggled to follow through, while others were more consistent but overloaded.
3. **Challenges as a Team Lead**
   As a team lead, one of the biggest lessons I learned was how important it is to understand each team member's strengths, limitations, and working style. It took time to

understand what people could realistically handle, and one of my early mistakes was assigning tasks based on what people volunteered for rather than what they could actually complete effectively.
Learning to say “no” was difficult but necessary. I had to develop the confidence to reassign work or step in when I knew someone wasn’t ready for a particular task, even if they were enthusiastic. Setting those boundaries helped the team function more smoothly and ensured that deadlines were met without compromising quality.

4. **Lack of Attention to Detail**
   Another recurring challenge was the general lack of attention to detail from members. Whether it was formatting issues, missing sections, or overlooked requirements, these small oversights often created larger problems down the line. As a result, additional time had to be spent on revisions and quality checks.

This class project was an emotional and educational journey.. As a team, we faced several challenges including adapting to sudden tool changes, coordinating effectively in a fast-paced environment, and managing uneven contributions. Transitioning from GitBook to Google Docs improved workflow efficiency. As a team lead, valuable lessons were learned about aligning task assignments with team members’ actual capabilities and enforcing boundaries to maintain quality and meet deadlines. A recurring obstacle was the lack of attention to detail, which led to avoidable rework. Despite the difficulties, the team is proud of what was accomplished.