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

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.

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.

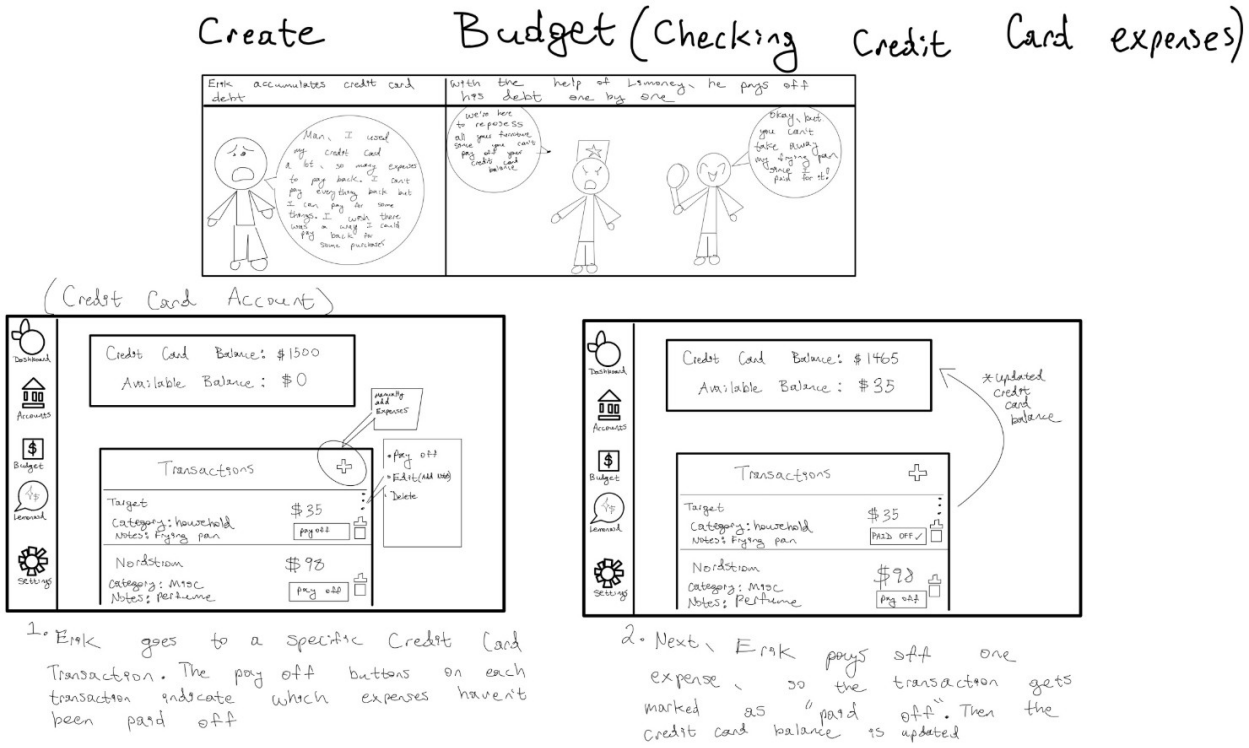
17. Reviews

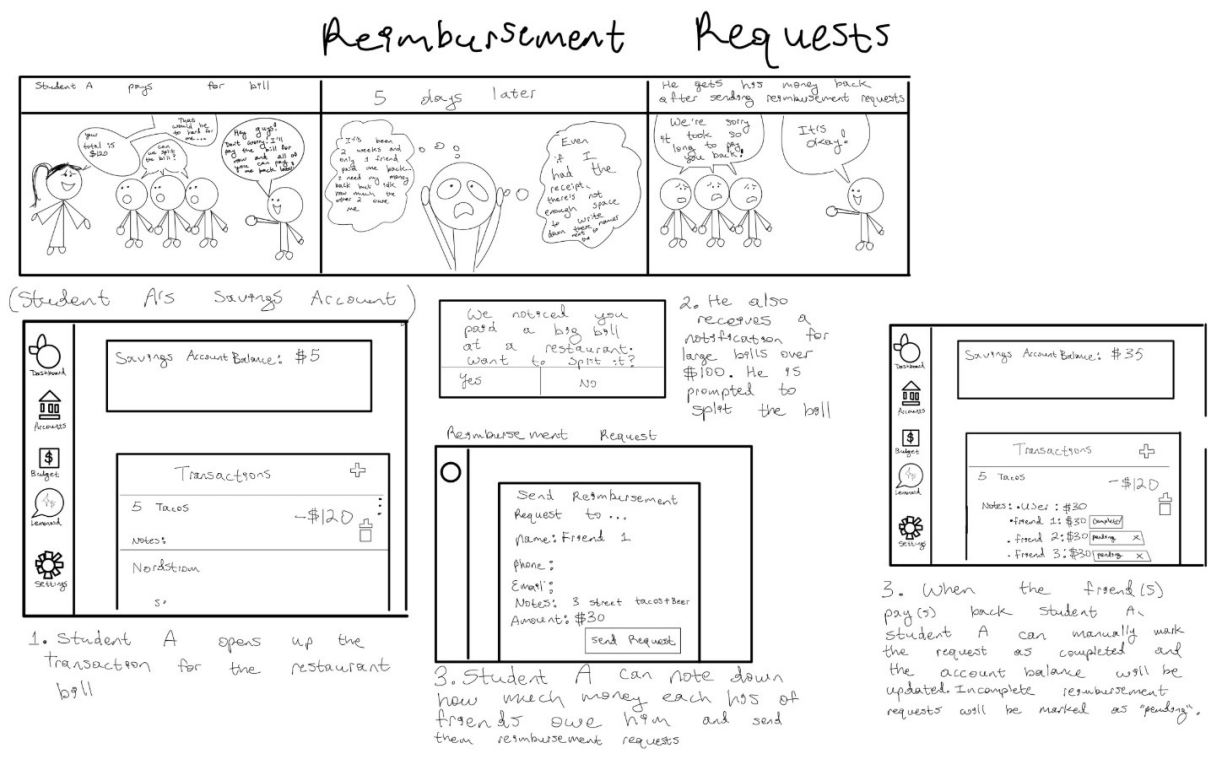
17.8. A review shall be viewable by system administrators for quality assurance and analysis.

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

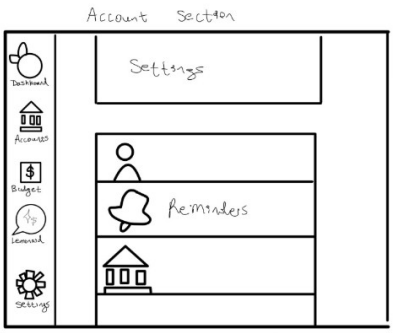
17.10. A review shall support analytics use cases for measuring system effectiveness and user satisfaction.

Mockups/Storyboards

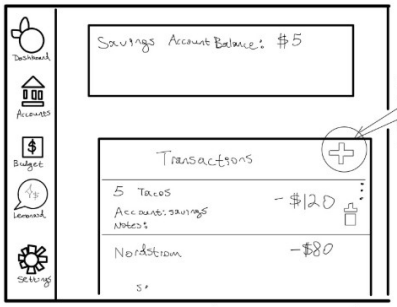


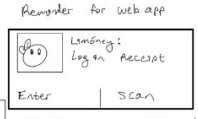


Log Transactions



or





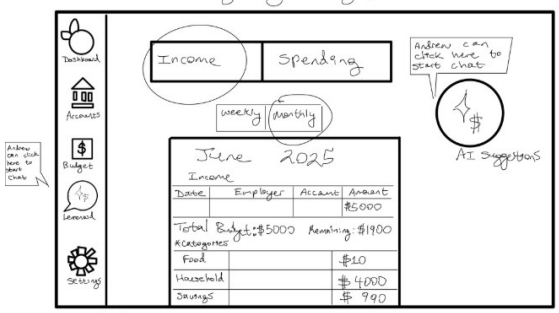
1. Erik can set reminders via settings or the account transaction dashboard

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



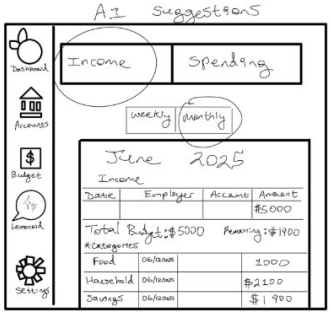
1. Andrew opens budget page where he can manually enter his monthly or weekly income in the income section

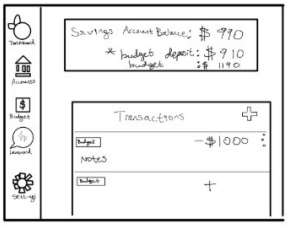
2. Andrew, being a premium user, can access the Lemonaid chatbot through the budgeting page or nav bar

3.

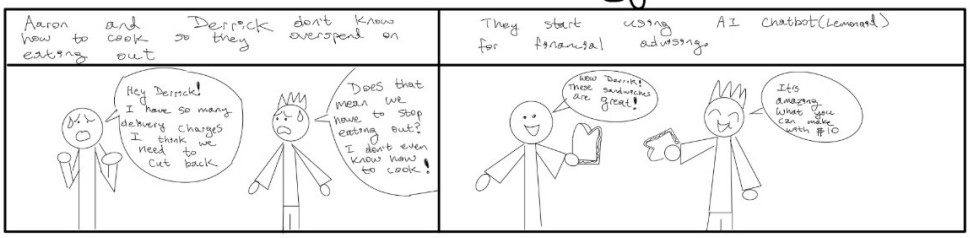
Modified Budgeting page

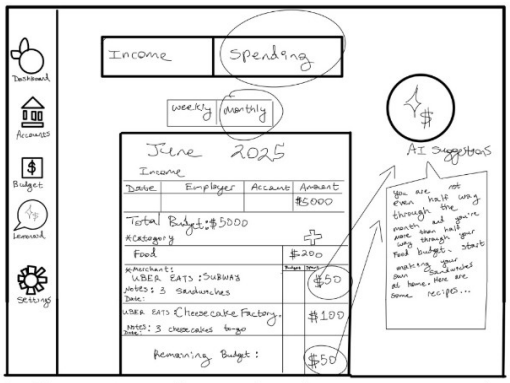
AI Suggestions



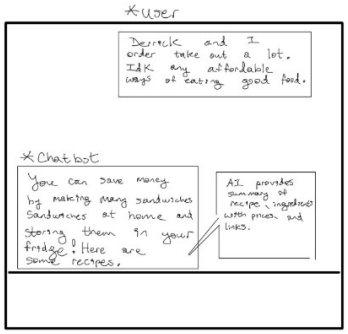


Receive AI suggestions

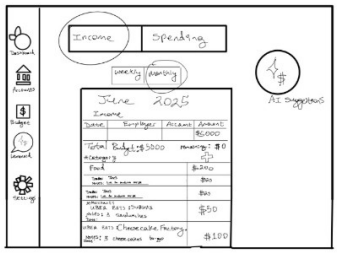




1. This is the "Spending" section of the budget, subtracting the budget total from the income section with transactions from account. The AI warns him he's overspending and the visual page helps him realize

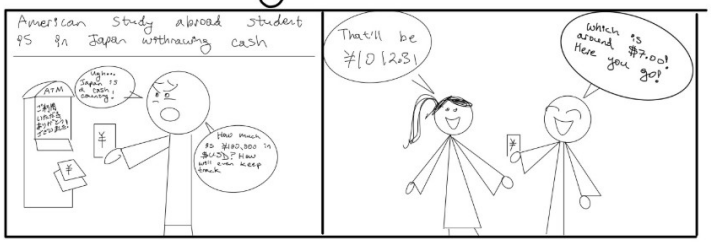


2. Aaron consults with the Lemonaid chatbot, asking how he and Derrick can stick to their budget.

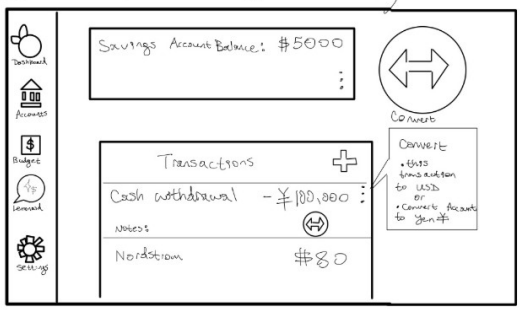


3. The AI chatbot updates the income section of the budget to help Aaron plan his food expenses for the rest of the month.

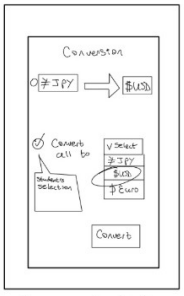
Currency Exchange



(Student A's Savings Account)

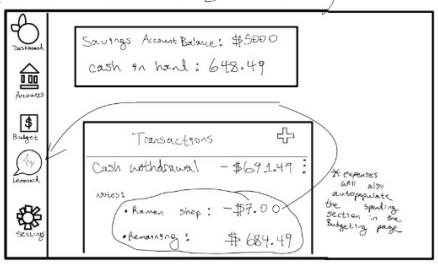


1. American Study Abroad student manually logs in the ATM cash withdrawal in ¥yen. He has three buttons to help him make currency conversions: the arrow button on the transaction itself (usually triggered only if transaction is in foreign currency), the big "Convert" button on the right, and the drop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



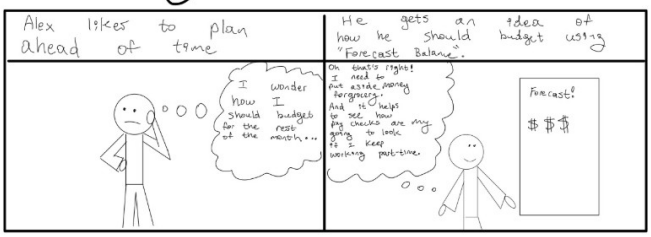
2. The student has the option of converting all transactions/expenses into $USD or converting just that one expense into $USD.

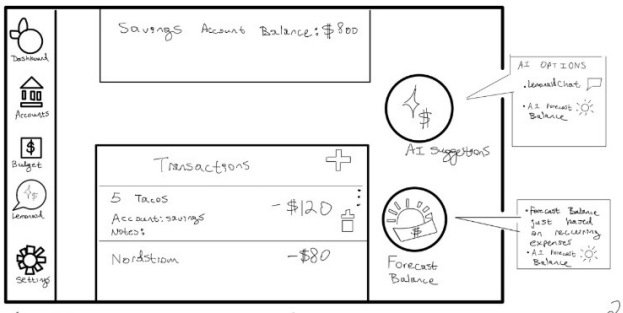
(Student A's Savings Account)



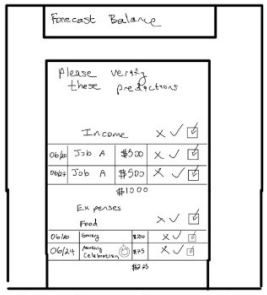
3. The student converts that cash withdrawal into $USD and can log in how he spent that cash which will also autopopulate his budgeting page.

Predicting Balances End of Month



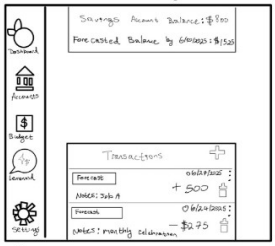


1. Alex goes to his Savings account. His balance sits at $800 at the beginning of the month but he wants to know what his balance will look like at the end of the month given his upcoming expenses and paychecks (income).



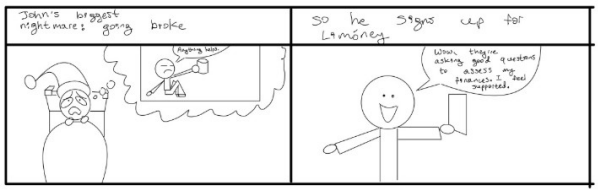
2. The Forecast presents Alex options to accept, reject, or edit for his bank balances and budget.

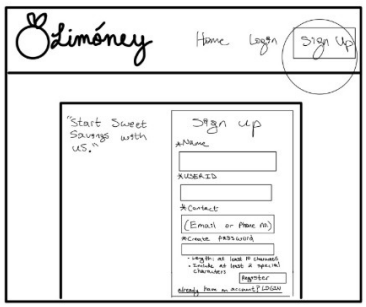
Modified Budgeting page and Accounts pages



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.

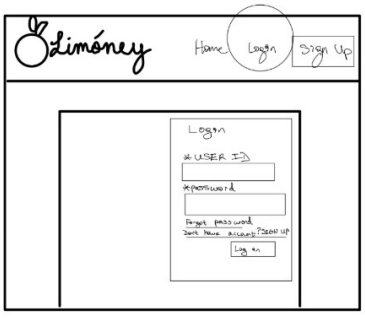
Signing up / Intro UI





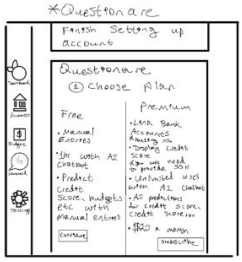
1. John goes to Sign up page.

or



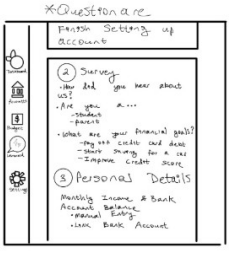
1. John went to sign up account and was already told he has an account so he logs in.

*Questionare



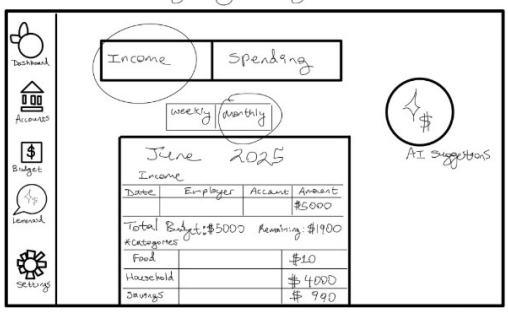
2. John chooses to be a premium user and answers the questionair so the Limóney webapp can give him a personalized experience based on his

*Questionare



Chatbot / AI assistance (Lemonaid)

Budgeting page

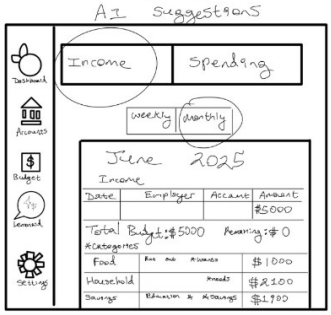


1. John opens his budget to start planning for June 2025. Overwhelmed, he clicks on "AI suggestions" which directs him to the Lemonaid Chatbot. This demonstrates accessibility: a reminder for John that he seek support when feeling overwhelmed

2. He asks the Chatbot for advice and it gives a response based on John's Account Balances, recurring expenses, income, demographic, credit card debt, etc.

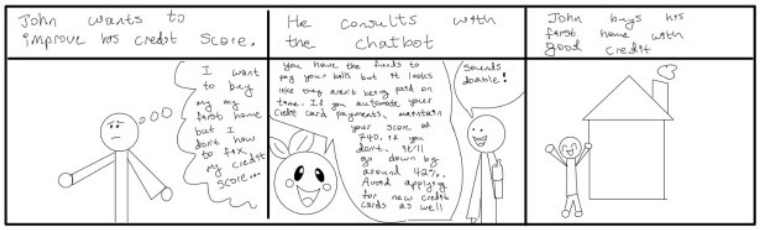
Modified Budgeting page

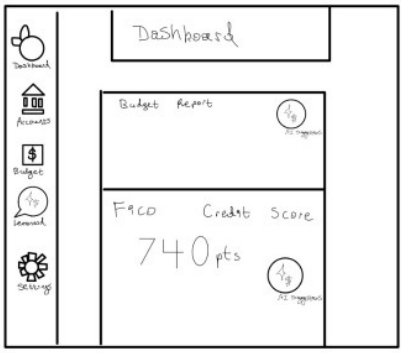
AI suggestions



3. The AI updates John's budget with the agreements based upon their conversation.

Credit Score Inquiry





1. John can check his Credit score on the dashboard using his

2. John consults with Lemonaid about improving his Credit score. Although Lemonaid provides suggestions, John will be responsible for automating his credit card

Cursor

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)

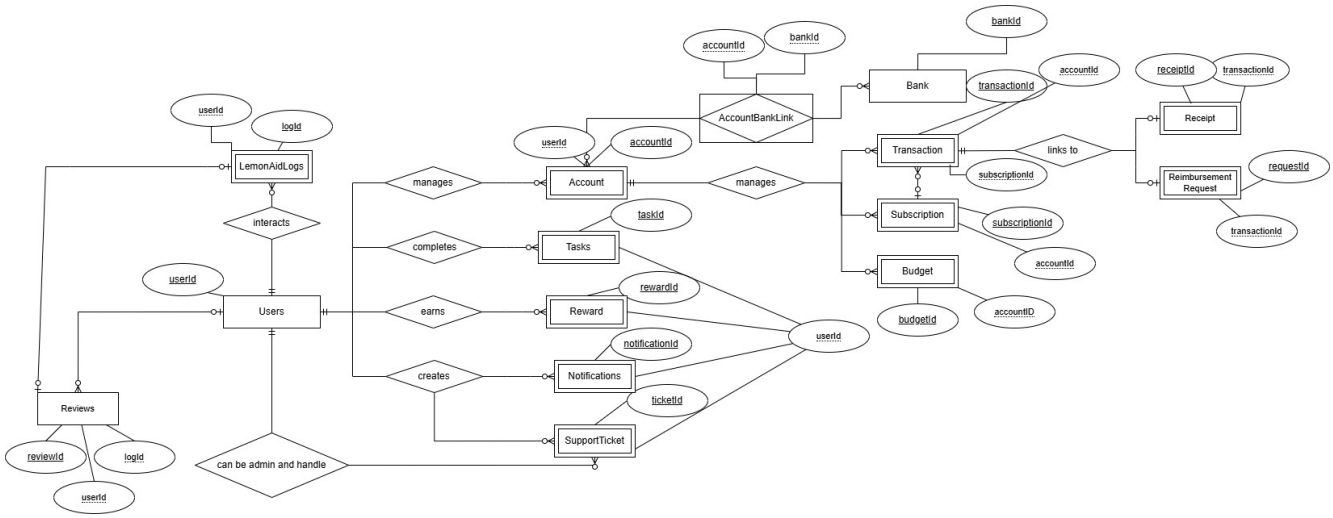
Attributes:
Connects user accounts to external banks for multi-bank linking.

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

Domains:

- Purely foreign key based.

- No extra data beyond IDs referencing accounts and banks.



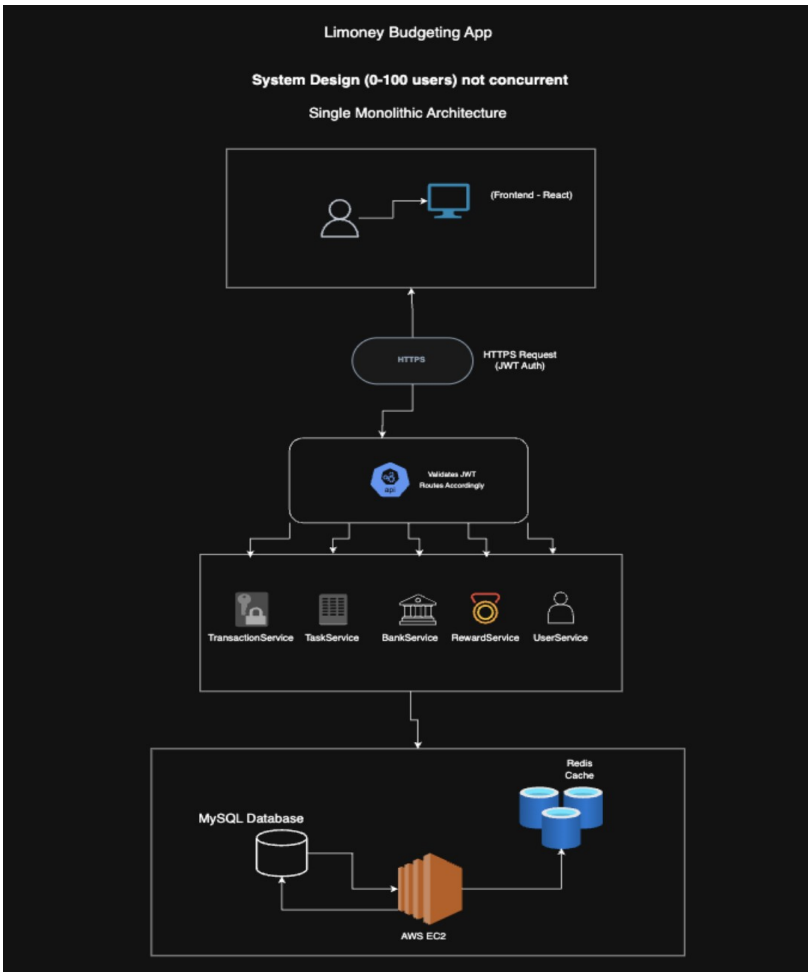
Media Storage

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

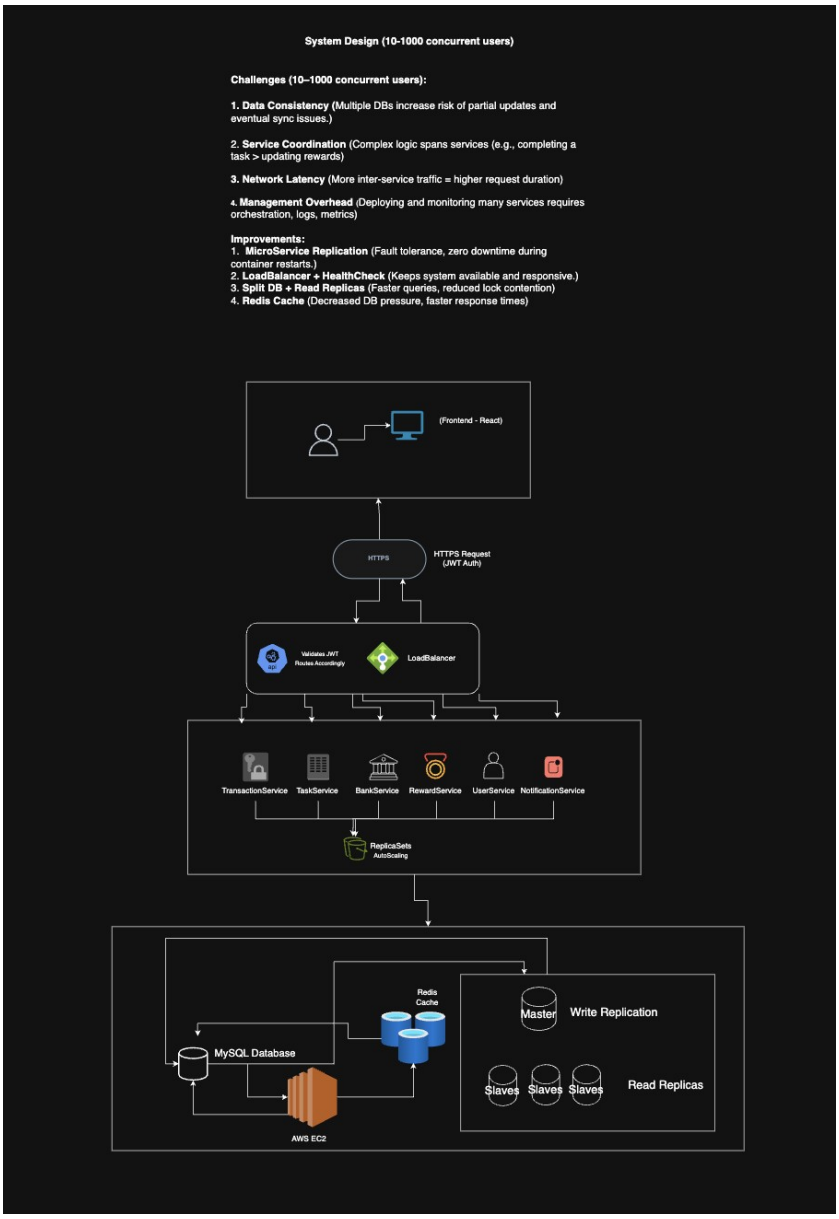
ALL SCALE DESIGN

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.



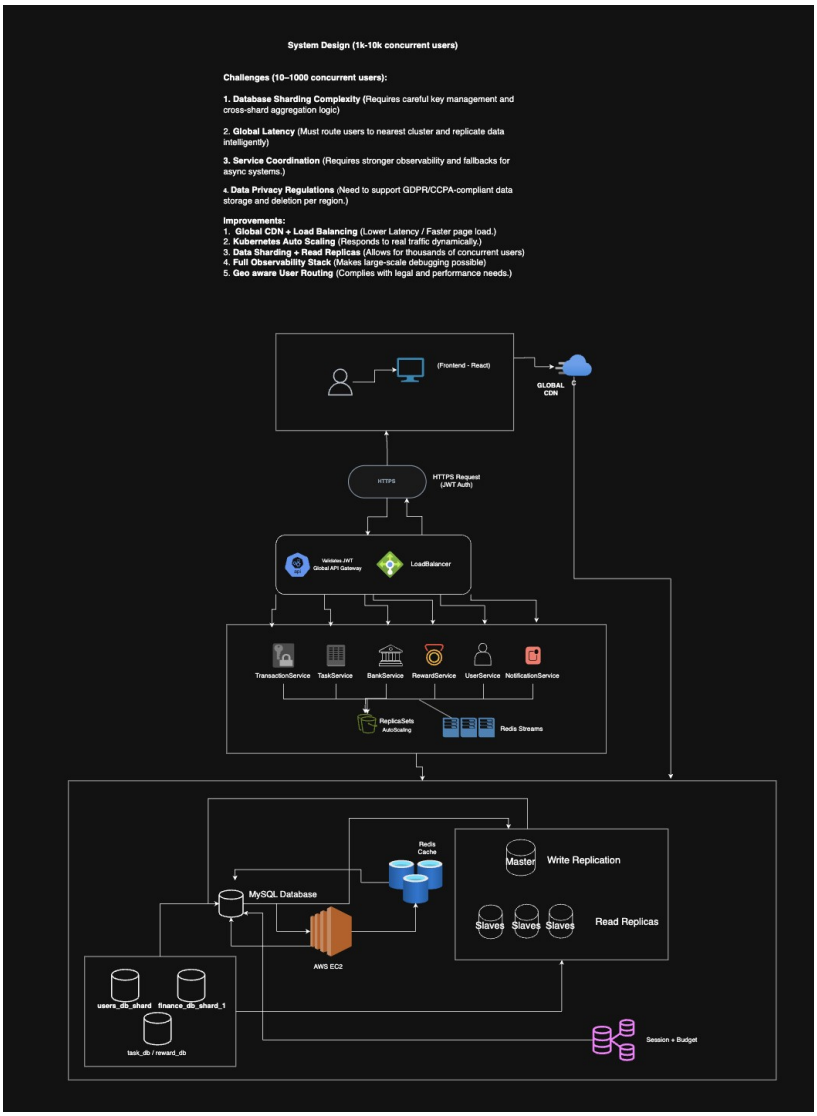
MEDIUM SCALE DESIGN

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

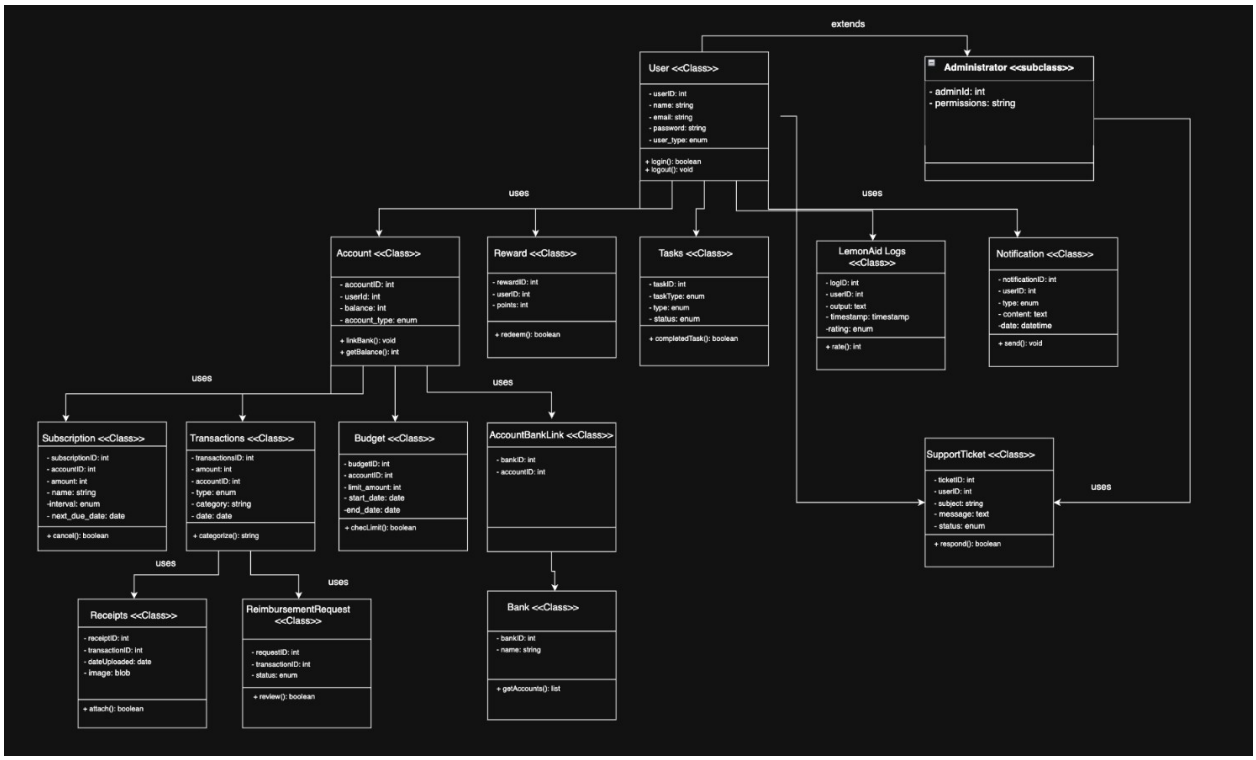
Backend System Design

- **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

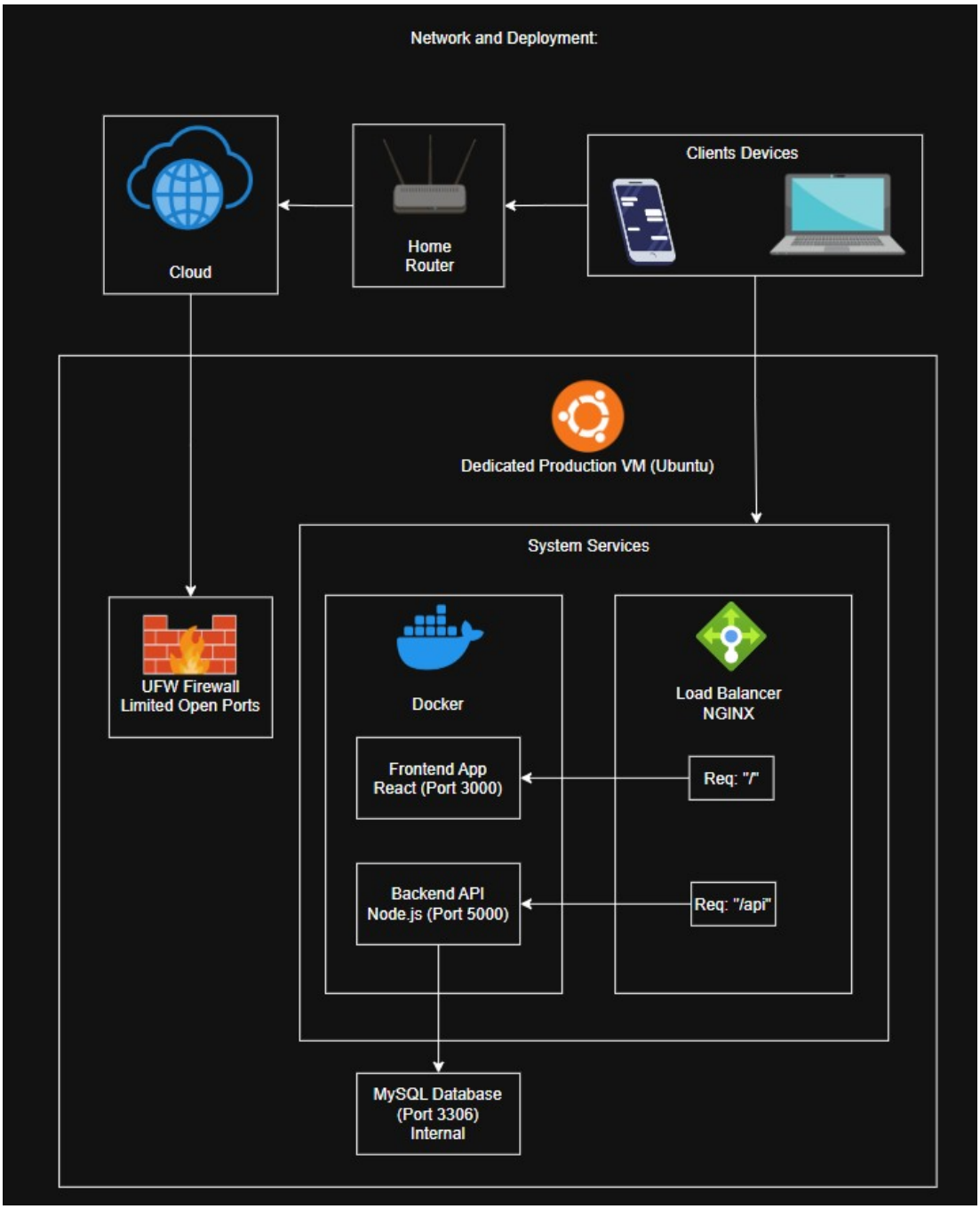
UML Design

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



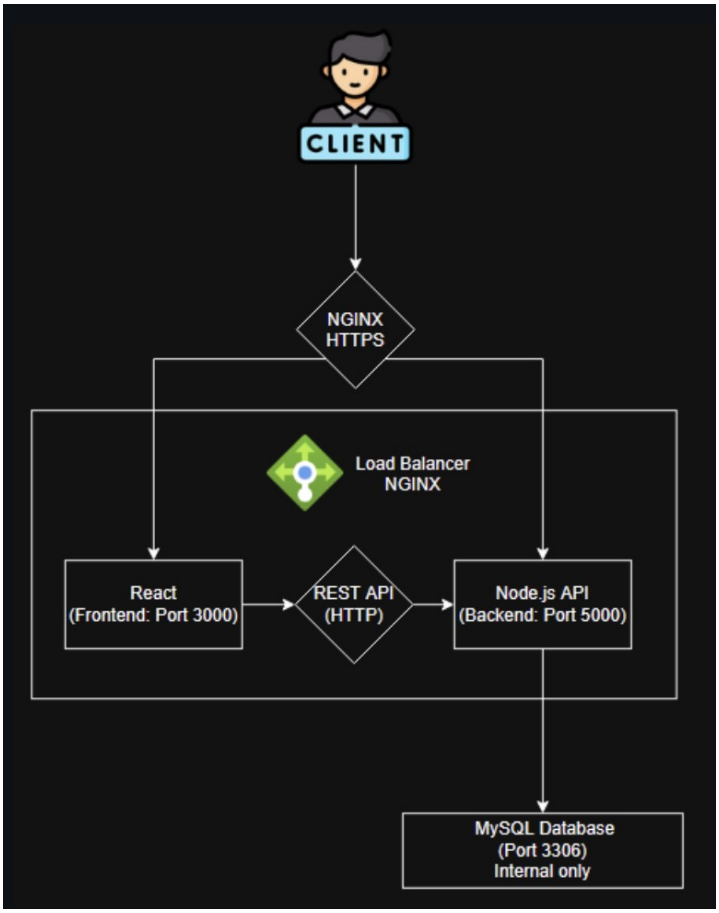
High Level Application Network Protocols and Deployment Design

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.

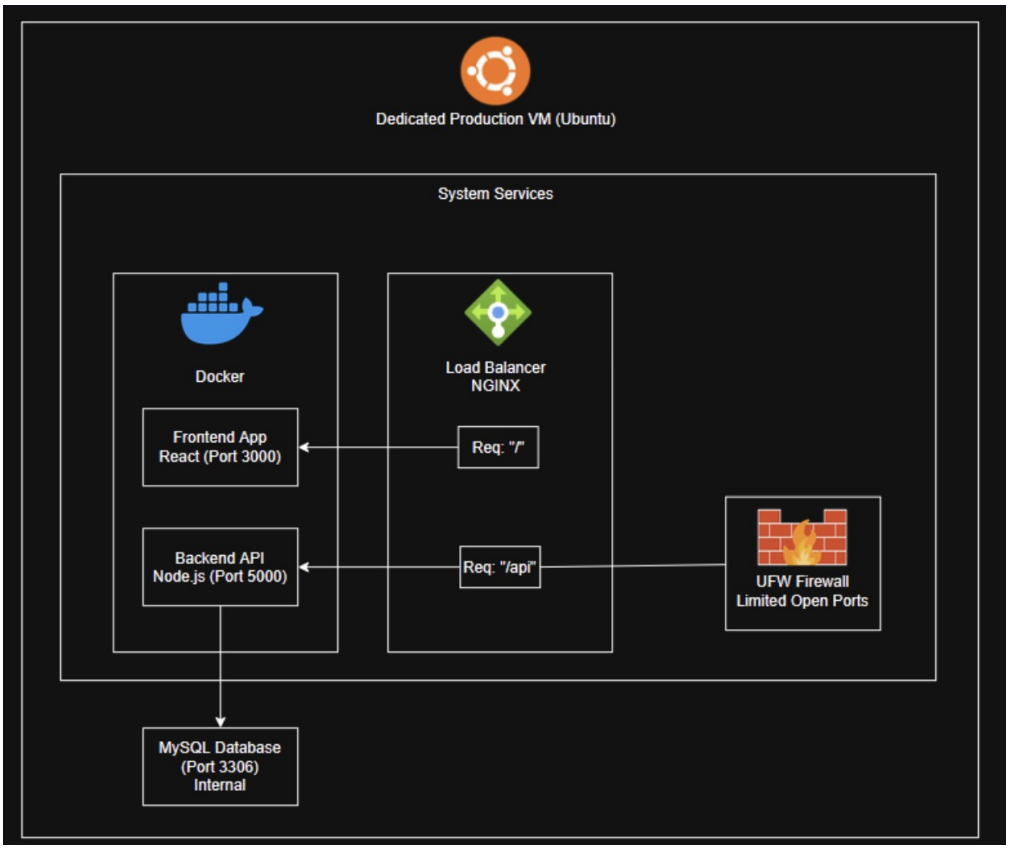
Application Networks Diagram



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