SW Engineering CSC648-848-01 Summer 2025

Limóney (Financial Budgeting Webapp)

Team 02

Milestone 4

7/31/25

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 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

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

Effectiveness Table

Function	Success Rate	Comments

Reimbursement	100%	Why didn't you guys use Zelle? (No public api access, however, Venmo may be feasible)
Currency Converter	90%	You should use real-time rates
LemonAid AI	80%	Needs page-specific interaction or should just be on the side
Budget	75%	Is confusing to use, may need a tutorial, why am I setting a budget for last month
Account Tracking	60%	Isn't up to date

Efficiency Table (Avg Time to Complete Tasks)

Function	Avg Time (s)	Optimal (s)
Reimbursement	12	10
LemonAid AI	8	7
Currency Converter	5	5
Budget Visualization	15	10

Likert User Satisfaction

User rated on a scale 1-5 (Strongly Disagree to Strongly Agree)

Reimbursement Request

1. I would frequently use this feature (4)
2. This feature was easy to understand (3)
3. I will share this feature with people I know (5)

Currency Converter

4. I would frequently use this feature (3)
5. This feature was easy to understand (5)
6. I will share this feature with people I know (2)

LemonAid AI

7. I would frequently use this feature (3)
8. This feature was easy to understand (5)
9. I will share this feature with people I know (5)

Budget Visualization

10. I would frequently use this feature (3)
11. This feature was easy to understand (2)
12. I will share this feature with people I know (2)

QA Test Plan

Selected Non-Functional Requirements

1. Performance
2. Usability
3. Load Tolerance
4. Reliability
5. Compatibility

QA Test Table

Performance: The system shall respond to user actions within 2 seconds under normal load..

Test No.	Function	Input	Expected Output	Result
1	Load Dashboard	User logs in and is redirected to dashboard	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	about 15 seconds	Took a little while for the accounts to load about 20 seconds FAIL
	Navigate to Accounts Page	User clicks on accounts page from dashboard	Accounts page takes about 3 seconds to load all the accounts	about 1 second PASS

Usability: The UI shall be intuitive and require no documentation for key tasks such as setting up budgets or tracking expenses.

Test No.	Function	Input	Expected Output	Result
1	Usage of Accounts page	User clicks on Accounts page in Navbar	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	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	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 reimbursement request PASS

Load Tolerance: AI recommendations shall be delivered within 5 seconds of user input.

Test No.	Function	Input	Expected Output	Result
1	Budget AI generates budget for user according to 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 chat provides financial advice to user when prompted	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 sets budget for user	User accepts generated budget	It updates the db and ui for user	Very buggy FAIL

.Reliability:The system shall maintain at least 99.5% uptime over a 30-day period. **DONE**

Test No.	Function	Input	Expected Output	Result
1	Up for 5 days?	Up 5 days	Up 5 days	PASS
2	Up for 15 days?	Up 15 days	Up 15 days	PASS
3	Up for 30 days?	Up 30 days	Up 30 days	PASS

Compatibility: The system shall support the following browsers:

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

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

Localization Testing

Plan

- Default Language: English
- Alternative: Spanish
- Attempted at using i18 library but there were so many version conflicts that it had to be dropped + was not part of nonfunctional requirements

Sample Test Case

Page	Test	Translation OK?
Home.js	N/A	No
Budget.js	N/A	No
About.js	N/A	No

Results

All UI text components have not passed the initial Spanish translation test.

Code Review

Coding Standards

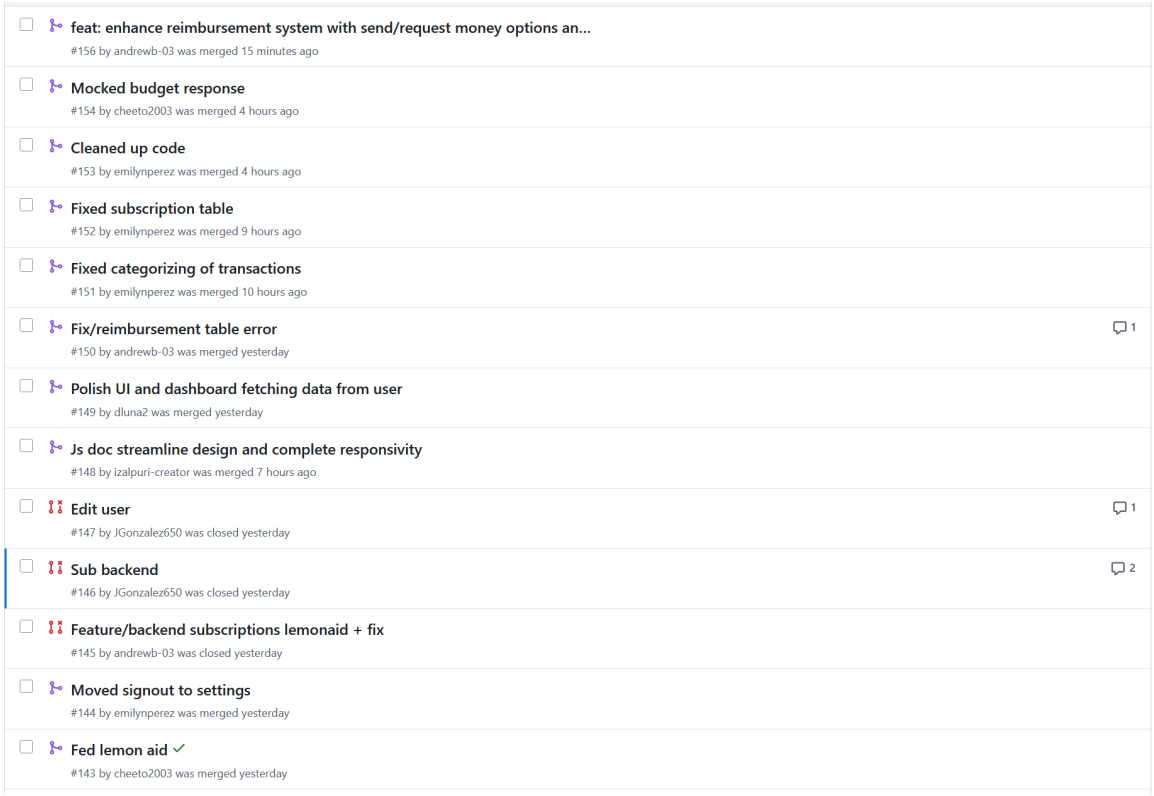
Here are the coding/working standards I have on a doc shared with all of my team members to ensure unity and reflecting what is asked from the nonfunctional requirements.

Coding standards:

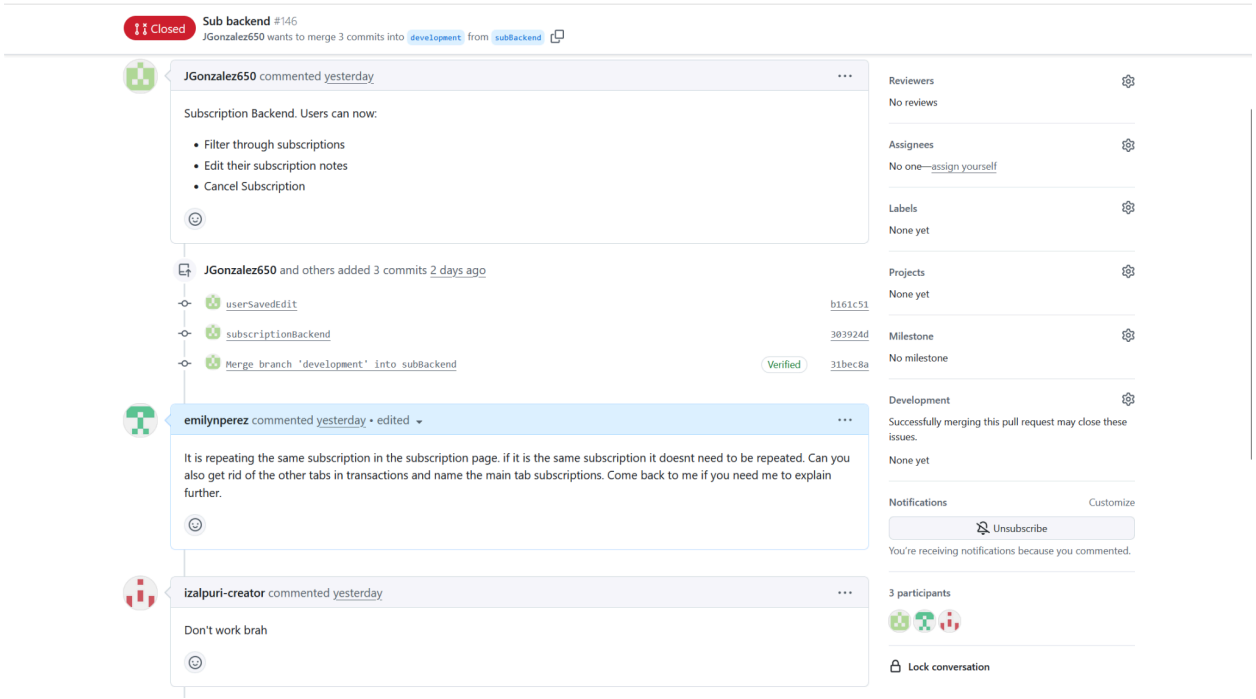
- Name variables under camelCase convention.
    - Ex:
    - camelCase
    - somethingCool
    - iDontKnowMan
- Always work within your own branch and have it be named after the feature you are creating. Try not to create many features at once!
- Never push code onto development branch with code that 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

Sample of some of our pull requests:



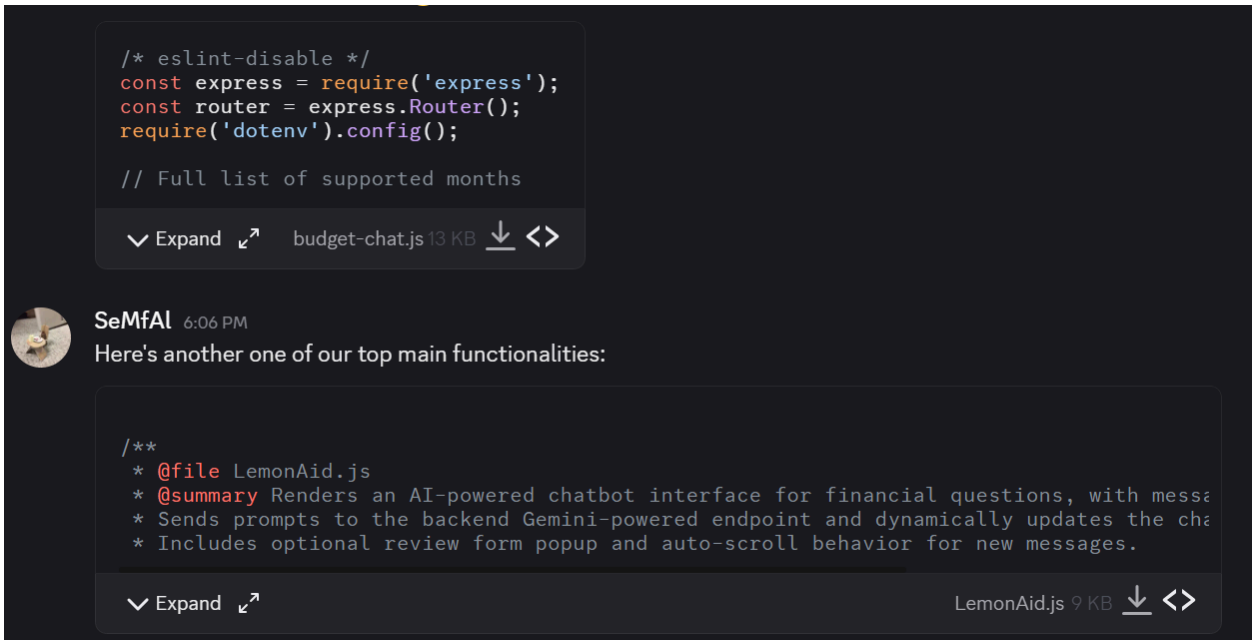
Sample of comments in a pull request:

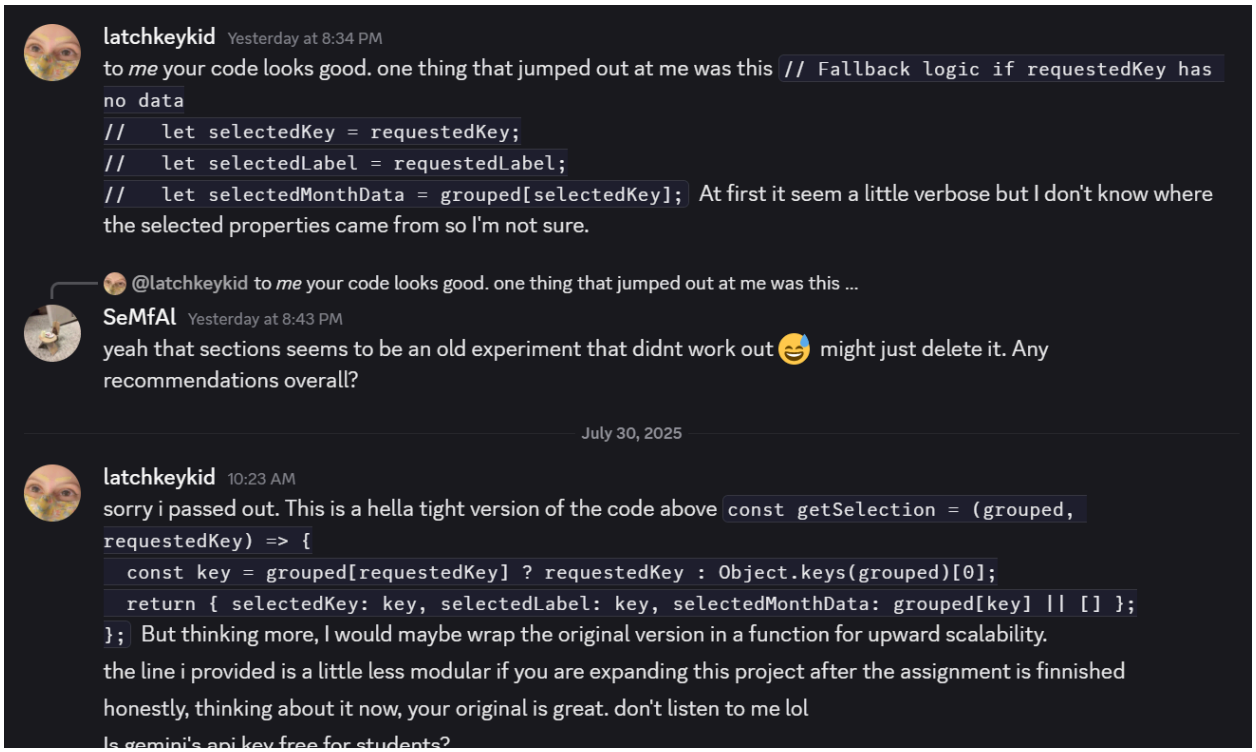


External Review

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. **ON TRACK**
1.2. Backend APIs shall respond to simple requests in under 200 milliseconds.**ISSUE - impossible**
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. **ON TRACK**
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. **ISSUE - not enough space in our database**
5.3. Background processes such as syncing and reminders shall automatically retry on failure.**ON TRACK**
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. **ON TRACK**
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. **ON TRACK**
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. **ON TRACK**

10. Compliance
10.1. The system shall comply with GDPR regulations for data privacy and user consent. **ON TRACK**
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. **ON TRACK**
11.2. An admin dashboard shall allow staff to review flagged transactions and feedback.**ON TRACK**
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