Dhirubhai Ambani University

Requirement Analysis

FRs and NFRs

Functional requirements

FR-01 - User registration

Identified by: User surveys and user stories.

FR-02 - User login & session management

Identified by: User Stories

FR-03 - All-in-one tracking

Identified by: Analysis of existing system and user interviews.

FR-04 - Data analysis & visualization

Identified by: User surveys, interviews, and prototyping.

FR-05 - Group expense splitting

Identified by: Analysis of existing systems, user interviews, and user surveys

---

FR-06 - AI advisor

Identified by: Brainstorming sessions, user interviews, and user surveys.

FR-07 - Customer support

Identified by: Stakeholder analysis

FR-08 - Reminders and notifications

Identified by: User surveys and user interviews

FR-10 - Budgeting and alerts

Identified by: User surveys and interviews

FR-11 - Recurring expense automation

Identified by: User interviews

Non-functional requirements

NFR-01 - Usability(user-friendly)

Identified by: User surveys

NFR-02 - Performance (fast and smooth)

Identified by: User experience best practices

NFR-03 - Security

Identified by: Industry standards for data privacy.

NFR-04 - Reliability

Identified by: Stakeholder expectations

NFR-05 - Maintainability(clean and organized code)

Identified by: Development team best practices.

NFR-06 - Scalability

Identified by: Brainstorming

User stories

User authentication and registration

Front of card:

As a new user, I want to register using my email and password, but if I already have an account, I want to be easily guided to the login page.

Back of card:

- The signup form must include fields for email, password, and confirm password.
- The system checks if the email already exists and, if so, displays an error: "Account already exists. Please log in," with a direct link to the login page.
- Successful signup creates an account and redirects the user to their dashboard.
- The form must include a clear link that says, "Already have an account? Login."

Front of card: As a registered user, I want to log in with my email and password so I can securely access my expense dashboard.

Back of card:

- The login form includes fields for email and password.
- Credentials are validated against the database.
- If correct, the user is redirected to the dashboard; otherwise, an error message is shown.

Front of card: As any user, I want to sign up or log in instantly using my Google account to avoid filling out forms.

Back of card:

- The screen must show distinct "Continue with Google" buttons.
- Clicking triggers the standard OAuth process with the chosen provider.
- On success, the system either logs in the existing user or creates a new account using their social profile information, then redirects them to the dashboard.

Front of card: As a user filling out any form with a password, I want to toggle the password's visibility so I can confirm I entered it correctly.

Back of card:

- All password and "confirm password" fields must have an "eye" icon.
- By default, the password text is masked (dots or asterisks).
- Clicking the icon reveals the password in plain text; clicking again re-masks it.

Front of card:

As a user who has forgotten my password, I want a simple and secure way to reset it so that I can regain access to my account.

Back of card:

- The login form displays a "Forgot password?" link.
- Clicking the link takes the user to a page where they can enter their registered email.
- The system sends a time-sensitive password reset link to that email.
- Following the link allows the user to set a new password and log in.

Front of card:

As a user filling out a form, I want instant and clear feedback on my inputs and helpful error messages if something goes wrong.

Back of card:

- The system provides real-time validation for email formatting and password strength requirements as the user types.
- A clear "Passwords do not match" error is shown if the password fields are different.
- Specific, helpful error messages are shown for incorrect login credentials, server issues, or other problems.

Front of card:

As a user, I want the option to stay logged in and access my data in real-time from multiple devices.

Back of card:

- A "Remember me" checkbox on the login form will maintain the user's session after the app is closed.

- The system supports concurrent logins, and any data changes (like a new expense) will sync in real-time across all active devices.
- A user can view and manage their active sessions from their account settings.

Personal Expenses

Front of card:

As a user, I want to add an expense with details so I can maintain a complete record of my spending.

Back of card:

- Must allow entry of amount, category, date, payment mode, and optional notes.
- The expense should appear instantly in the transaction history.
- Prevents saving if the amount is invalid.
- Supports quick-add for frequent expenses.
- If offline, expenses must be stored locally and synced later.

Front of card:

As a user, I want to delete expenses so that I can correct mistakes or remove old entries.

Back of card:

- Must allow deletion from history.
- The system must confirm before deletion.
- Once deleted, the expense is removed from reports.
- Deleted entries can be undone briefly.

Front of card:

As a user, I want to edit the details of an expense I've already logged, so that I can correct any mistakes or update information without having to delete and re-enter the entire transaction.

Back of card:

- The user must be able to access an "Edit" option from the transaction history list or the detailed expense view.
- Clicking "Edit" must open a pre-filled form with all the existing expense details.
- The user can change any of the fields in the form.
- Upon saving, the changes must be instantly reflected in the transaction history and all related reports or charts.

- The system must re-validate the new information before saving.
- There must be a clear "Cancel" button to discard any changes and return to the previous screen.

Front of card:

As a user, I want to track money I owe or money owed to me so I can settle balances easily.

Back of card:

- Must allow tagging expenses as 'paid by me' or 'shared'.
- The system automatically calculates balances.
- Shows clear labels such as 'You owe X' or 'Y owes you'.
- Allows manual settlement once payment is made.
- Sends optional reminders for pending settlements.

Front of card:

As a user, I want to categorise expenses so that I can see where my money goes.

Back of card:

- Must provide predefined categories.
- Allows custom categories.
- Categories must be editable or deletable.
- Expenses must be filterable by category.
- Categories should be colour-coded for visualisation.

Front of card:

As a user, I want to see summaries and graphs so that I can track patterns in my spending.

Back of card:

- Must provide bar, pie, or line charts.
- Allows filtering by category or date.
- Graphs update dynamically.
- Exports insights into reports.
- Highlights unusual spikes in spending.

Front of card:

As a user, I want to see how much I saved each month so I can track progress toward my goals.

Back of card:

- Must compare income vs expenses.
- If no income, allow setting a budget baseline.
- Shows savings as a number and a percentage.
- Highlights months with negative savings.
- Displays trend over past months.

Front of card:

As a user, I want to check my total spending over time so that I can reflect on my habits.

Back of card:

- Must display cumulative total.
- Allows filtering by time range.
- Updates dynamically as expenses change.
- Supports breakdown by category over history.

Front of card:

As a user, I want to view my most recent expenses so that I don’t lose track of new entries.

Back of card:

- Must display at least the last 5–10 entries.
- Each entry shows date, amount, category, and notes.
- Clicking a transaction shows full details.
- Must allow swipe actions like delete or edit.

Front of card:

As a user, I want budget alerts so I can be notified when I am nearing or exceeding my spending limit.

Back of card:

- Must allow setting monthly or category budgets.
- Alerts at 75, 90, and 100 per cent.
- Alerts via push notifications or in-app messages.

- Must allow adjusting thresholds.
- Visual warnings must be displayed.

Front of card:

As a user, I want recurring expenses to be managed automatically so that I don't need to add them manually each time.

Back of card:

- Must allow marking an expense as recurring (daily, weekly, monthly).
- Recurring expenses are added automatically to the history.
- Notification must be sent before adding a recurring expense.
- Must allow pausing or cancelling recurring expenses.

Front of card:

As a user, I want to export my expenses for a specific time period to a CSV file, so that I can share and view the expenses in one place together

Back of card:

- The user must be able to select a date range for the export
- The generated file must be in a valid .csv format that opens correctly in standard spreadsheet software.
- The CSV file must contain a header row with clear column titles
- Only the expenses that fall within the selected date range are included in the export.
- If there are no expenses in the selected period, the system displays a friendly message instead of generating an empty file.

Group Expenses

Front of card:

As a user, I want to create groups and invite members so that we can manage shared expenses.

Back of card:

- Must allow group creation with name and description.
- Members can be invited via email, phone, or link.
- Users can accept or decline invitations.
- Must support joining via link or code.
- Group admin can remove members.

Front of card:

As a group member, I want to add shared expenses so that everyone's contributions are visible.

Back of card:

- Must allow entry of amount, payer, participants, and category.
- By default, it is split equally among all members.
- Appears instantly in the group transaction list.
- Invalid entries must be blocked.
- Must allow attaching notes or receipts.

Front of card:

As a group member, I want to split costs unequally so that amounts reflect real spending.

Back of card:

- Must allow manual split by percentages or fixed amounts or even by parts.
- The system must validate totals.
- Supports mixed contributions.
- Displays how much each member owes.

Front of card:

As a group member, I want to view balances so that we can settle easily.

Back of card:

- The system must auto-calculate balances.
- Balances are shown in simple terms.
- Allows marking balances as settled.
- Supports partial settlements.
- Updates balance dynamically.

Front of card:

As a group member, I want to receive notifications when expenses are updated so that I always stay informed.

Back of card:

- Must notify when a new expense is added.
- Notify when expense is edited or deleted.
- Send reminders for settlements.
- Allow toggling notifications per group.

Parental Mode

Front of card:

As a parent, I want to monitor my child's account so that I can guide their spending.

Back of card:

- Must allow linking child accounts with email or ID.
- Parents can view all child transactions.
- The system generates alerts for limits set by parents.
- Parents can see categorised reports of the child's expenses.
- Parents are notified of unusual transactions.
- Linking requires consent from both parent and child.

AI Features

Front of card:

As a user, I want to chat with an AI assistant so that I can quickly get answers about my expenses.

Back of card:

- An AI chatbot is integrated into the app for expense-related queries
- Respond to questions like "How much did I spend on food last week?"
- Displays text answers and shows fallback suggestions if it cannot answer.

Front of card:

As a user, I want AI-based insights so that I can understand and improve my spending habits.

Back of card:

- AI generates personalised suggestions based on expense history
- Provides graphical insights alongside text explanations
- Highlights overspending trends in specific categories

Front of card:

As a user, I want to set goals with AI assistance so that I can stick to my budget.

Back of card:

- AI recommends realistic budget goals based on spending patterns
- Tracks progress and sends timely reminders
- Notifies me if I exceed the budget and congratulates me on achieving goals

---

Front of card:

As a user adding an expense, I want the AI to intelligently suggest a category based on my description so that I can save time and keep my records consistent.

Back of card:

- The AI must analyse the transaction description in real-time to suggest the most likely expense category.
- The user must always have the final choice to accept the AI's suggestion or manually select a different category.

Front of card:

As a user, I want the AI to handle my queries in natural language so that I don’t need to click through menus.

Back of card:

- Supports NLP for free-form expense questions
- Provides context-aware responses for accurate results
- Handles queries like expense breakdowns or category comparisons

---

Non-Functional Stories

Front of card:

As a user, I want the web-app to open quickly and feel responsive so that I don’t get frustrated waiting.

Back of card:

- The system must launch and display the main dashboard within a few seconds.
- Navigating between main screens (e.g., Dashboard, Transactions) must be smooth.
- Scrolling through long lists, such as transaction history, must remain fluid and responsive.

Front of card:

As a user, I want theweb-app to always work when I open it so that I can rely on it anytime.

Back of card:

- The system must handle network or server issues by displaying a friendly error message with a retry option, rather than crashing.
- Unsaved progress in forms, such as a partially entered expense, must be stored temporarily to prevent data loss from an interruption.

USE CASE

USER AUTHENTICATION

UC-01: User Registration

Goal: Allow a new user to create an account.

Actor: New User

Preconditions:

- User is not logged in
- User does not already have an account

Postconditions:

- Account is created
- User is redirected to dashboard

Main Flow

1. User opens the signup page.
2. Enters email, password, confirm password.
3. System validates format & matching passwords.
4. System checks if email already exists.
5. System creates the account.
6. User is redirected to the dashboard.

Alternative Flow

- A1: Email already exists → “Account exists, login instead”
- A2: Passwords do not match → show error
- A3: Weak password → show strength warning

UC-02: User Login

Goal: Allow a registered user to securely log in.

Actor: Registered User

Preconditions:

- User must have an existing account

Postconditions:

- User logs in successfully
- User is redirected to dashboard

Main Flow

1. User opens login page.
2. Enters email & password.
3. System validates credentials.
4. User is redirected to dashboard.

Alternative Flow

- A1: Incorrect credentials → show error
- A2: Server offline → show retry message

UC-03: Social Login (Google OAuth)

Goal: Enable quick login without entering credentials manually.

Actor: User

Preconditions:

- Google OAuth enabled

Postconditions:

- User account created or logged in
- Redirect to dashboard

Main Flow

1. User clicks “Continue with Google”.
2. OAuth popup opens.
3. User selects Google account.
4. System verifies token & logs in/creates account.

Alternative Flow

- A1: OAuth permission denied
- A2: Google authentication fails

UC-04: Forgot Password

Goal: Allow a user to securely reset their password.

Actor: User

Preconditions:

- User must be registered

Postconditions:

- Password updated successfully

Main Flow

1. User clicks “Forgot Password?”.
2. Enters email.
3. System sends reset link.
4. User opens link.
5. Enters new password.
6. System updates password.

Alternative Flow

- A1: Unregistered email → show error
- A2: Reset link expired

UC-05: Remember Me

Goal: Keep the user logged in across sessions.

Actor: User

Preconditions:

- User logged in manually

Postconditions:

- User remains logged in on future visits

Main Flow

1. User checks “Remember me”.
2. System generates long-session token.
3. Next time user visits → auto-login.

Alternative Flow

- A1: Token expired → require login
- A2: User revoked session manually

PERSONAL EXPENSE MANAGEMENT

UC-06: Add Expense

Goal: Allow the user to record a new personal expense.

Actor: User

Preconditions:

- User must be logged in

Postconditions:

- Expense saved to database

Main Flow

1. User opens **Add Expense** screen.
2. Enters amount, date, category, mode, notes.
3. User can optionally click **AI Categorize** (if no manual category selected).
4. System validates inputs.
5. System saves the new expense.
6. System updates dashboard, calendar, and graphs.

Alternative Flow

- A1: Invalid amount → show error
- A2: Network loss → “Connection lost. Retry?”

UC-07: Edit Expense

Goal: Allow a user to modify an existing expense.

Actor: User

Preconditions:

- Expense exists
- User logged in

Postconditions:

- Updated expense saved

Main Flow

1. User opens the Expense Calendar.
2. User clicks a date.
3. System shows expenses for that date.
4. User selects an expense.
5. User clicks **Edit**.
6. System shows pre-filled form.
7. User updates values.
8. System validates input.
9. System saves updated expense.
10. Dashboard recalculates values.
11. System shows “Updated successfully”.

Alternative Flow

- A1: Invalid data → error
- A2: Network issue → Retry option

UC-08: Delete Expense

Goal: Allow the user to remove an unwanted expense.

Actor: User

Preconditions:

- Expense exists
- User logged in

Postconditions:

- Expense deleted

Main Flow

1. User opens the Expense Calendar.
2. User selects a date.

3. System shows expenses for that date.
4. User selects one expense.
5. Clicks **Delete**.
6. Confirmation popup appears.
7. User confirms.
8. System deletes the expense.
9. System updates graphs and totals.
10. System shows success message.

Alternative Flow

- A1: User cancels delete
- A2: Network error → “Retry deletion?”

UC-09: View Expense

Goal: Allow the user to view full details of an expense.

Actor: User

Preconditions:

- At least one expense should exist for that date

Postconditions:

- Expense details shown

Main Flow

1. User opens the Expense Calendar.
2. Clicks on a date containing expenses.
3. System shows expense list.
4. User selects an expense.
5. System displays:
    - Title
    - Amount
    - Category
    - Date & Time
    - Tags
6. User closes detail view.

Alternative Flow

- A1: Date has no expenses → show “No expenses”
- A2: Expense fails to load → show error
- A3: User closes popup

GROUP EXPENSE MANAGEMENT

UC-10: Create Group

Goal: Allow users to create groups for shared expenses.

Actor: User

Preconditions:

- User logged in

Postconditions:

- New group created

Main Flow

1. User opens Group tab.
2. Enters group name & description.
3. Invites members.
4. Group is created.

Alternative Flow

- A1: Group name duplicate → show error

UC-11: Add Group Expense

Goal: Allow users to add a shared expense in a group.

Actor: Group Member

Preconditions:

- Group exists
- User is member

Postconditions:

- Expense added

Main Flow

1. User selects group.
2. Enters amount, payer, participants.
3. System applies **default equal split**.
4. Saves expense.

Alternative Flow

- A1: Missing participant → show error

UC-12: Split Expense Unequally

Goal: Allow user to split a group expense manually.

Actor: Group Member

Preconditions:

- Group exists
- User is member

Postconditions:

- Custom split saved

Main Flow

1. User opens Groups.
2. Selects group.
3. Clicks **Add Expense**.
4. Enters expense details.
5. Selects **Unequal Split**.
6. Enters custom amounts/percentages.
7. System checks if total matches expense amount.
8. System saves expense.
9. Group balances update.
10. Success message shown.

Alternative Flow

- A1: Total not matching → show error
- A2: Blank participant amount → show error

UC-13: View Group Balances

Goal: Allow members to see total group spending and balances.

Actor: Group Member

Preconditions:

- Group has expenses

Postconditions:

- Balance sheet displayed

Main Flow

1. User opens Group tab.
2. Selects group.
3. System loads all group expenses.
4. System calculates:
    - Total group expense
    - Who owes whom
5. Displays balance sheet.

Alternative Flow

- A1: No expenses → show ₹0
- A2: Database fetch error

AI FEATURES

UC-14: AI Chat Assistant

Goal: Allow users to get insights by asking natural language questions.

Actor: User

Preconditions:

- AI service live

Postconditions:

- AI response shown

Main Flow

1. User types a question.
2. AI processes query.
3. AI returns answer + suggestions.

Alternative Flow

- A1: AI can't understand → fallback suggestion

UC-15: AI Budget Goal Recommendations

Goal: Help user set realistic budget goals.

Actor: User

Preconditions:

- User has spending history

Postconditions:

- New goal recommended or created

Main Flow

1. User opens Goal screen.
2. AI analyzes data.
3. Suggests budget goal.
4. User accepts.

Alternative Flow

- A1: AI cannot generate → manual goal

UC-16: AI Auto-Categorization

Goal: Let AI automatically suggest expense category.

Actor: User

Preconditions:

- Adding expense
- Description entered

- No manual category selected

Postconditions:

- Category suggested

Main Flow

1. User enters description.
2. User clicks **AI Categorize**.
3. System sends text to AI.
4. AI predicts category.
5. System shows the suggestion.
6. User applies or overrides category.

Alternative Flow

- A1: AI low confidence → “Select manually”
- A2: User edits description → AI re-enabled
- A3: AI service error → “Try again later”

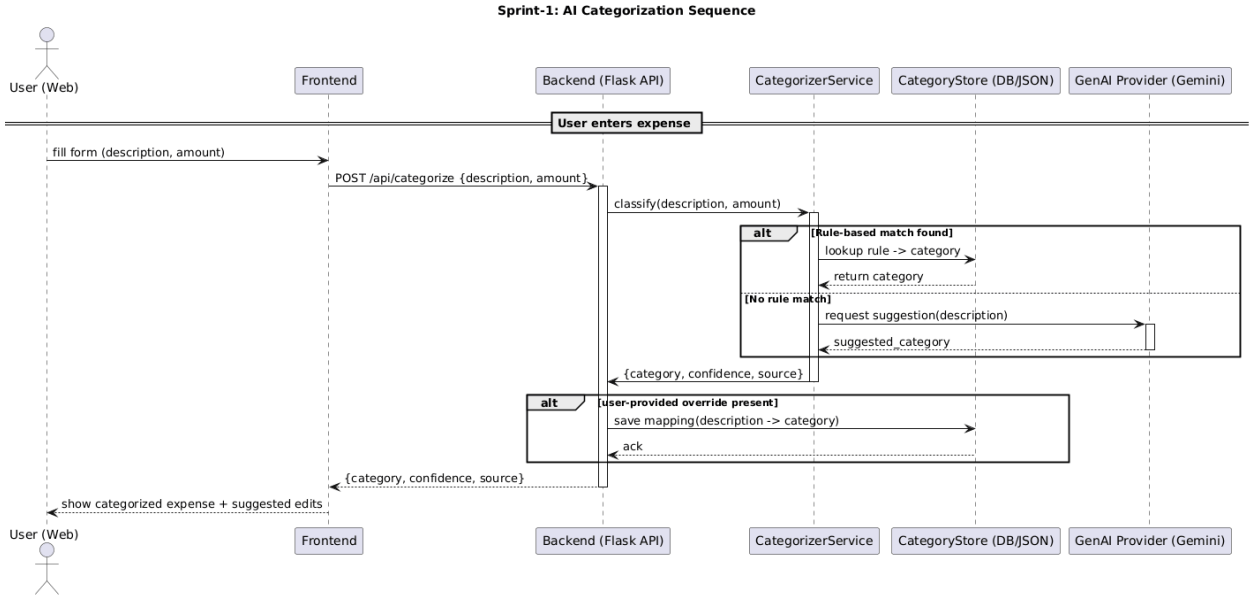
Sprints

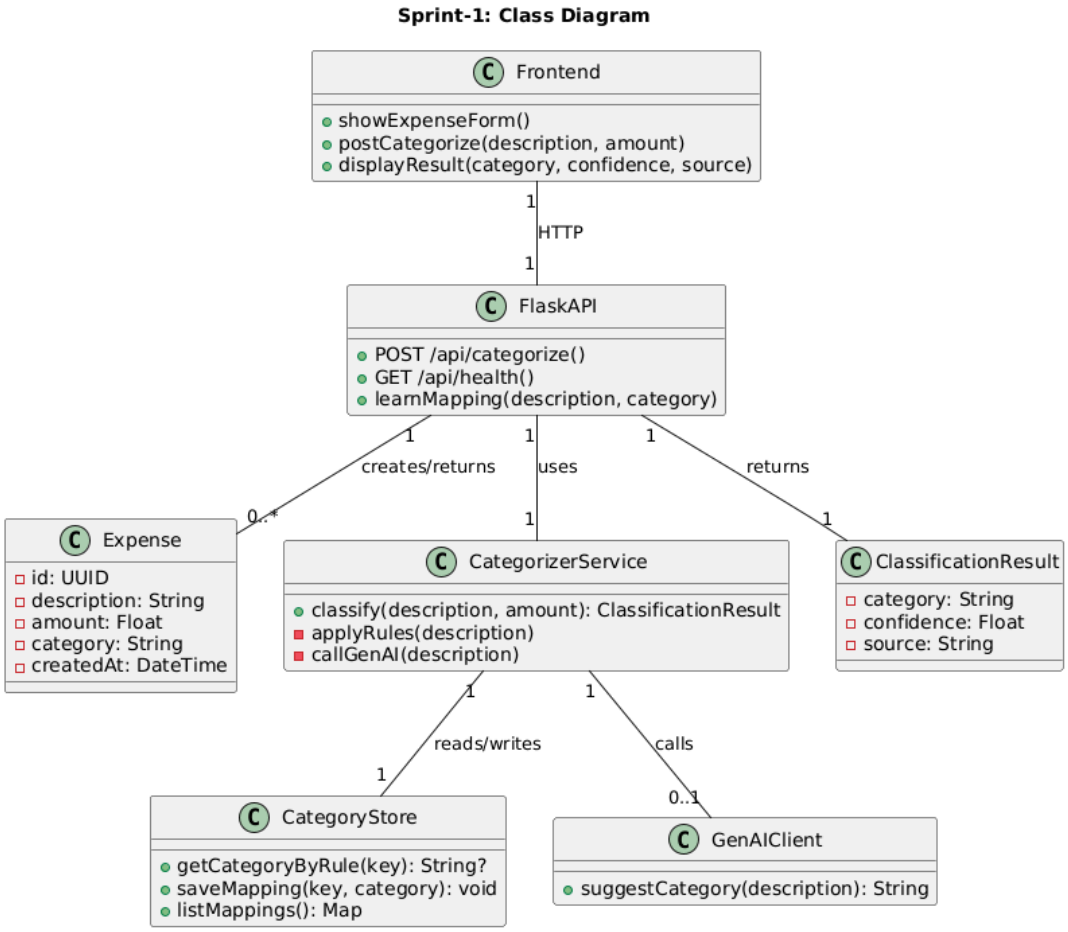
Sprint 1: The plan, the design, and our first AI test

Goal: Our primary objective is to determine precisely what we're building, design the app's visual appearance, and test our core AI concept to ensure its feasibility.

Key Activities:

- **Figuring out what to build:**
  - We'll decide on the "must-have" features for our first version.
  - We'll gather feedback from real people by sending out our survey and discussing their money habits with a few friends.
  - Based on that feedback, we'll write down our final list of project requirements.
- **What the frontend team will do:**
  - Create the tentative design and a prototype of the web app in Figma.
- **What the backend & AI team will do:**
  - They will build a simple server with api endpoints for basic AI categorisation that proves our AI categorisation idea works. This won't be a full working app, just a small terminal-side test program.
  - They will share this script with the whole team. Everyone can then run it, test it under different situations, and see how accurate our AI is from day one.





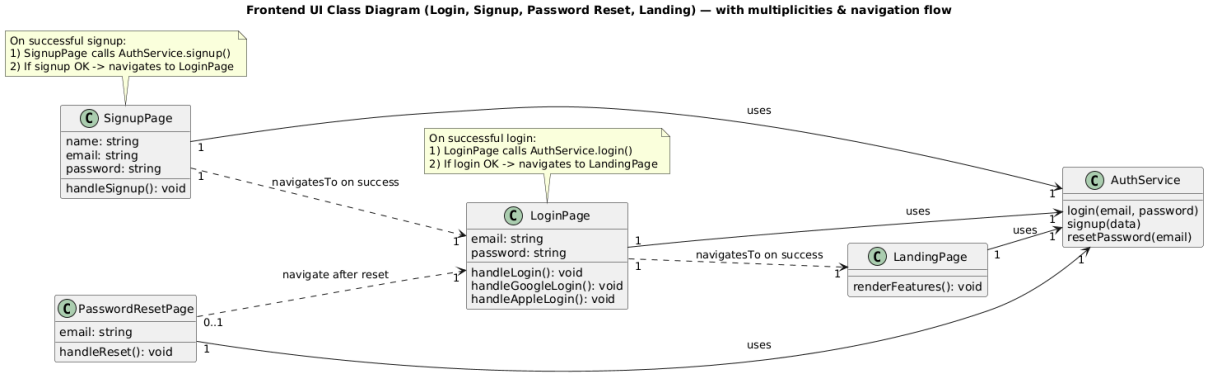
Sprint 2: The Core Foundation - Database & Authentication

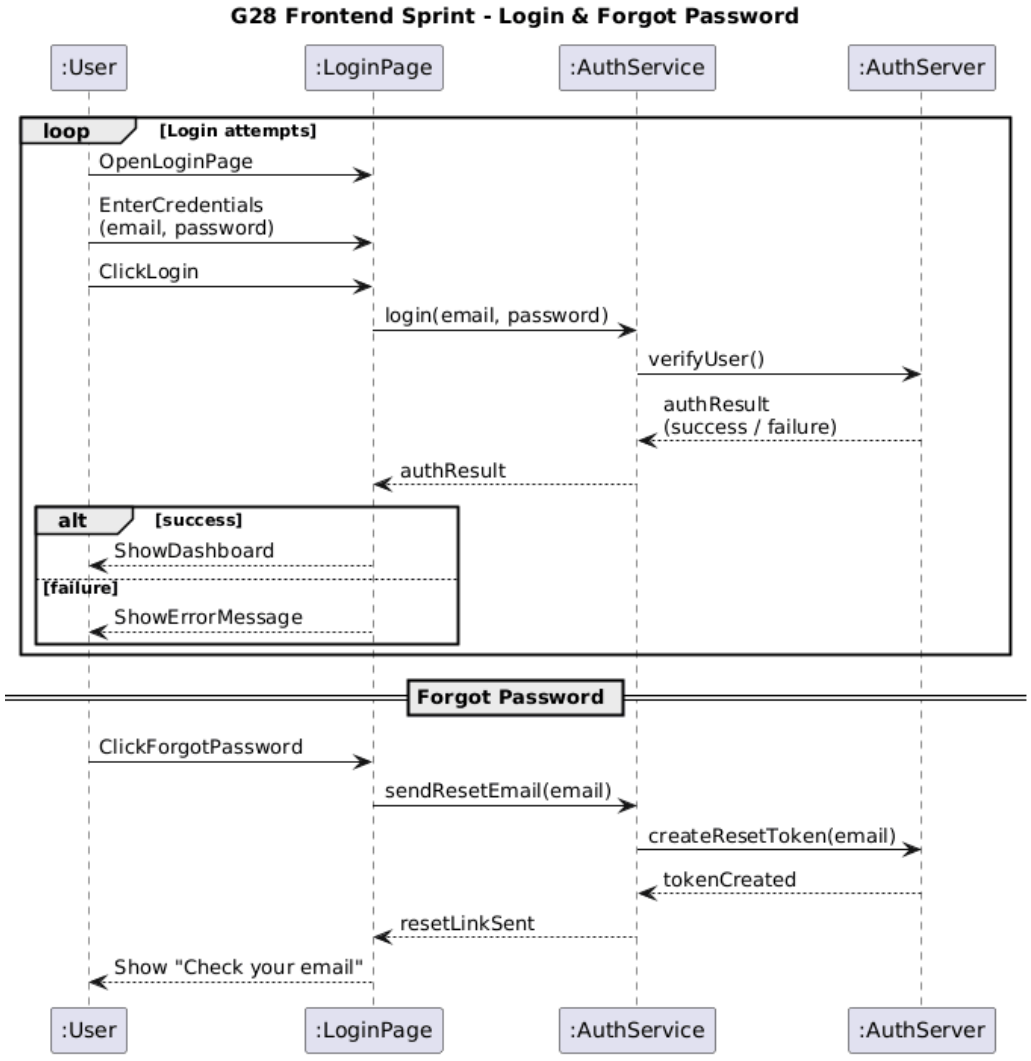
Goal: To build the complete database blueprint and a secure, working user authentication system.

Key Activities:

- **Frontend Team:**
  - Build the final React components for the **Login**, **Sign Up**, **Password Reset**, and **Landing** pages based on the Figma designs.
  - Implement the UI for the "Login with Google" and "Login with Apple" buttons.
- **Backend Team:**
  - Set up our Supabase project and enable **Authentication** (Email/Password, Google).

- Define and create the **entire database schema** in SQL (all the tables we planned).
- Write and test all the critical **SQL Triggers**, **Functions**, and **Row Level Security (RLS) policies** to make the database secure from day one.





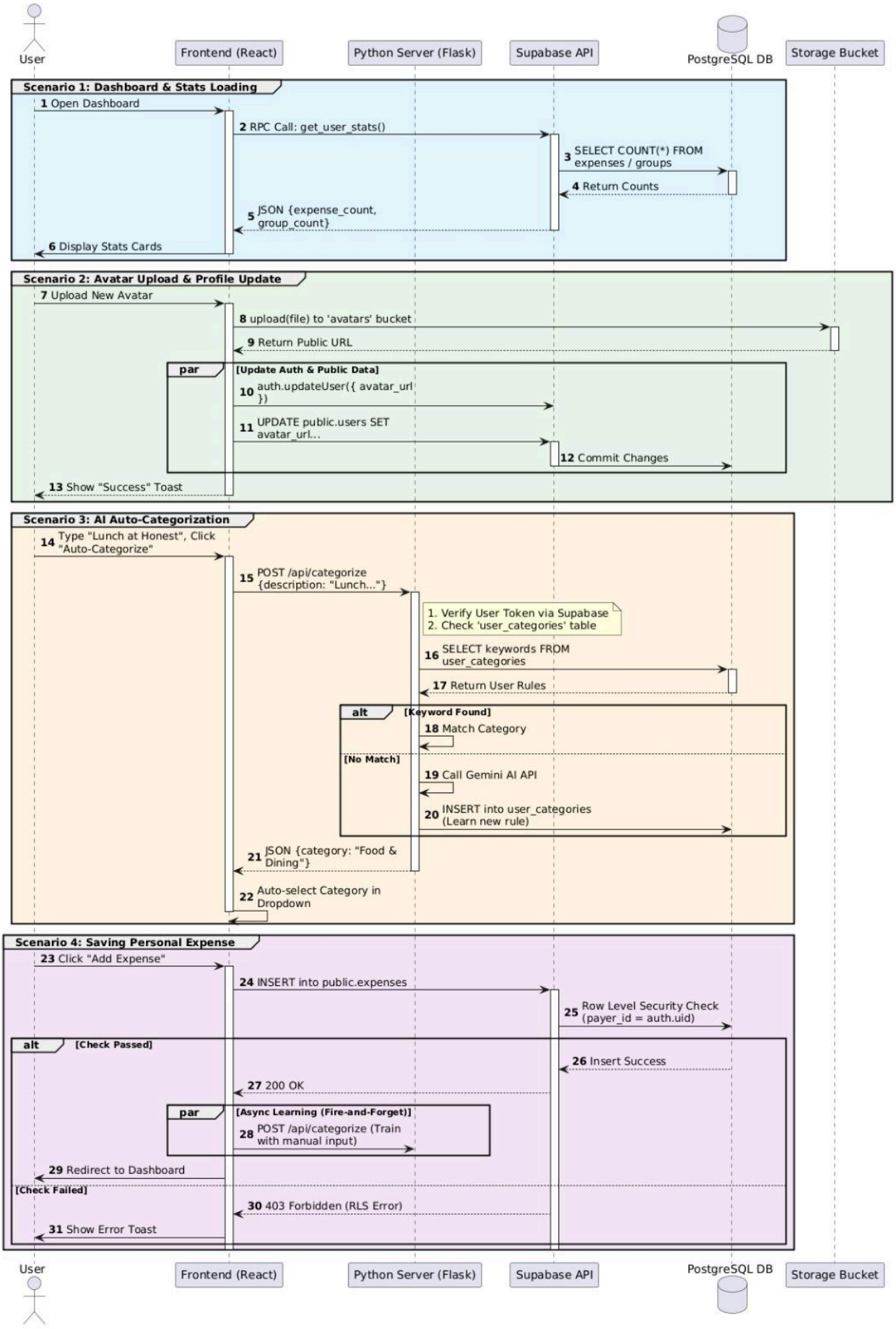
Sprint 3: Core Features - Profile, Dashboard & Personal Expenses

Goal: To build the core "logged-in" experience. Users will be able to manage their profile, see their dashboard, and track personal (non-group) expenses.

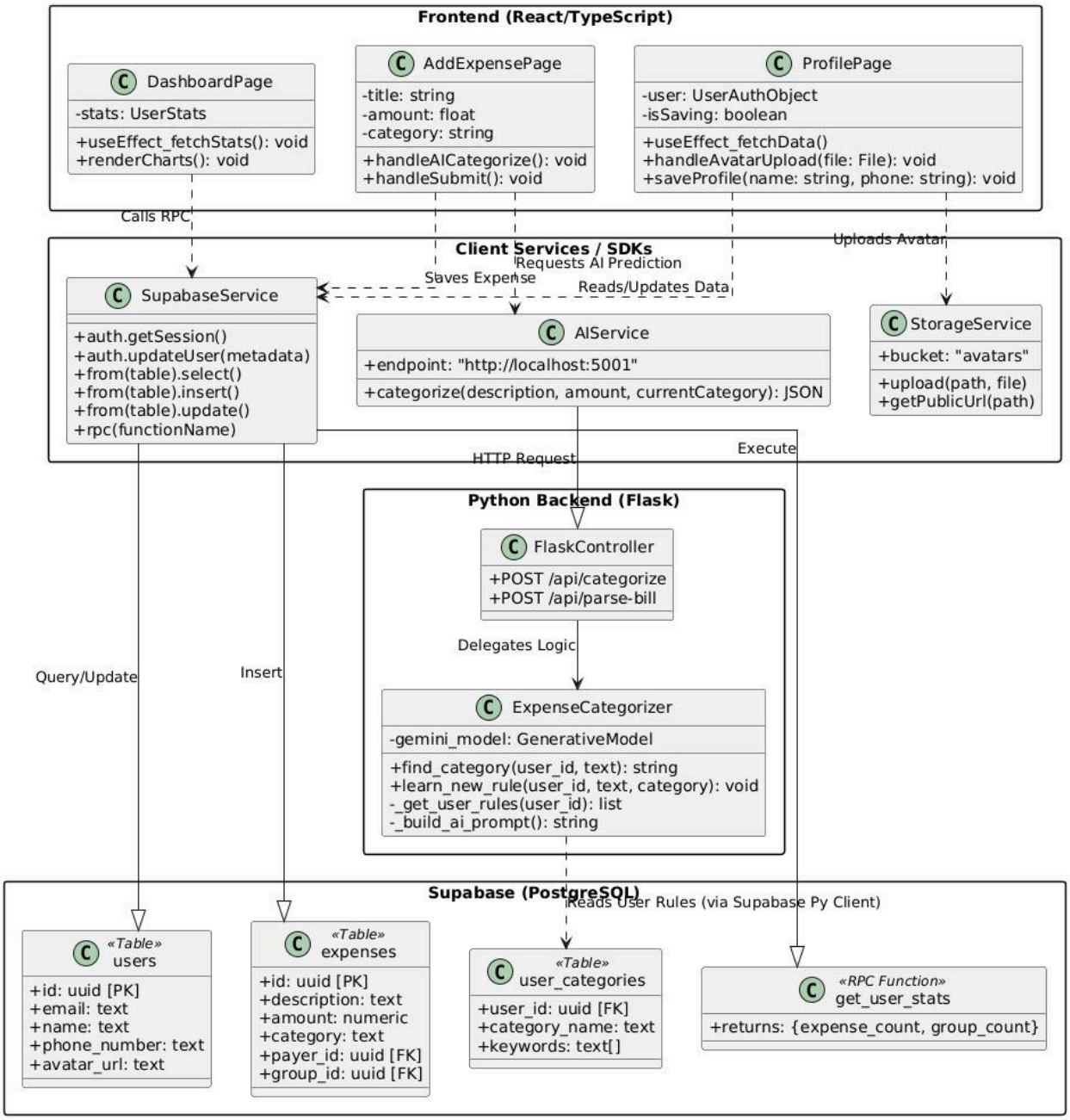
Key Activities:

- **Frontend Team:**
  - Build the `Profile.tsx` page, including the "edit/save" logic and the avatar upload UI.
  - Build the `Dashboard.tsx` component, including the layout for charts and analytics.
  - Build the `AddExpense.tsx` component, focusing on the "Personal" tab.
- **Backend & Integration Team:**
  - Connect the Profile page to the `users` table and **Supabase Storage** (for avatars).
  - Implement the `get_user_stats` function and connect it to the Dashboard.
  - Connect the `AddExpense.tsx` form's "Personal" tab to the `expenses` table in Supabase.
  - **Integrate the AI:** Connect the "Auto-Categorize" button to the Python AI endpoint. Update the `handleSubmit` function to also send manual entries to the AI for learning.

Sequence Diagram - Sprint 3: Core Features & AI Integration



Class Diagram - Sprint 3: Core Features & AI Integration

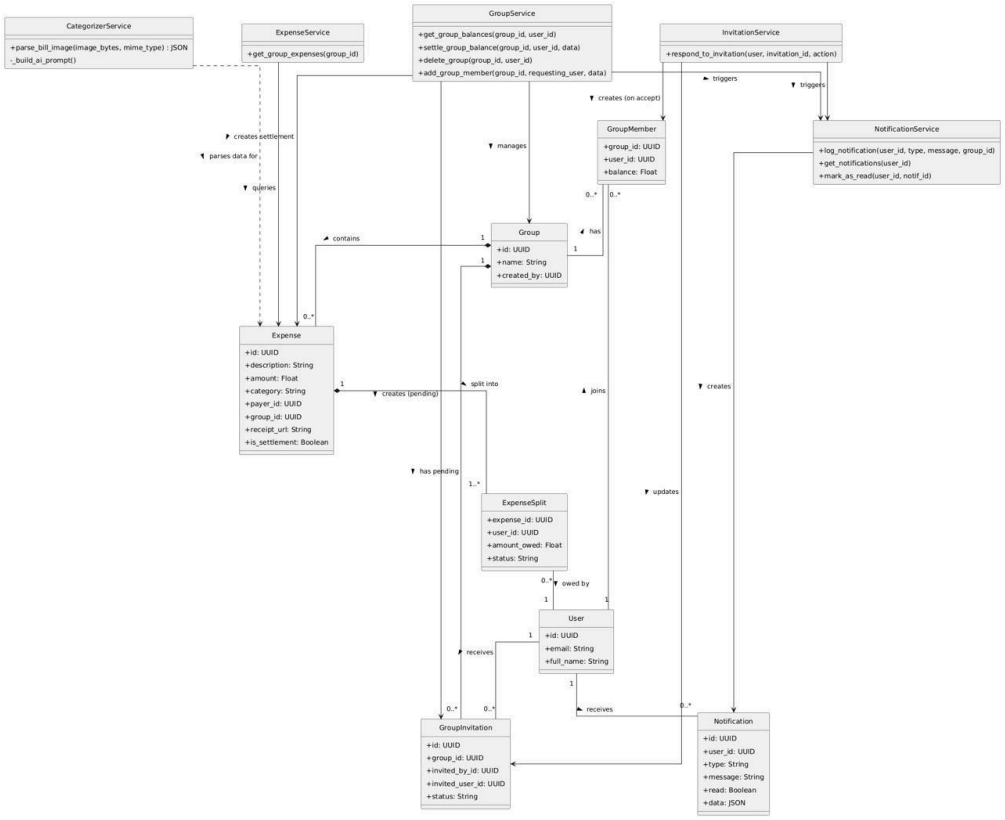


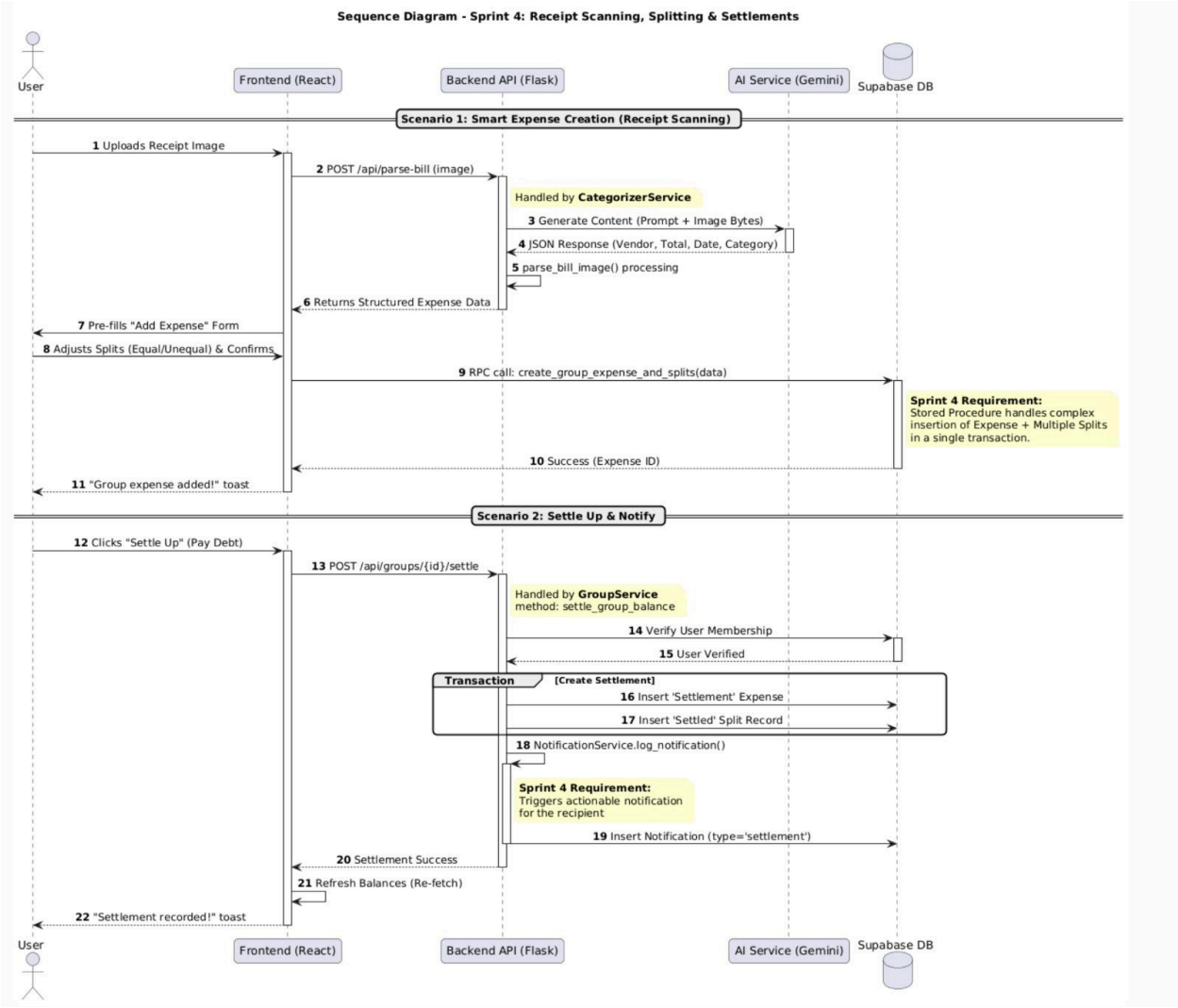
Sprint 4: Group Expenses, Receipt Scanning & Notifications

Goal: To build the app's social features: group expenses and bill splitting. We'll also add receipt scanning and a notification system.

Key Activities:

- **Frontend Team:**
  - Build the "Group" tab in the `AddExpense.tsx` component, including the UI for "Equal" vs. "Unequal" splitting.
  - Build the pages for creating and managing groups.
  - Implement the UI for **receipt scanning** (the upload button).
  - Build the UI for the new **notification** feature.
- **Backend & Integration Team:**
  - Implement the `create_group_expense_and_splits` SQL function to handle complex group expenses.
  - Build the "Settle Up" logic.
  - Implement the **Python AI endpoint for receipt scanning** (`/api/parse-bill`).
  - Modify the database and create backend logic for the new **notification system**.
  - Connect all new UI elements (group splits, receipt scanning) to their backend endpoints.





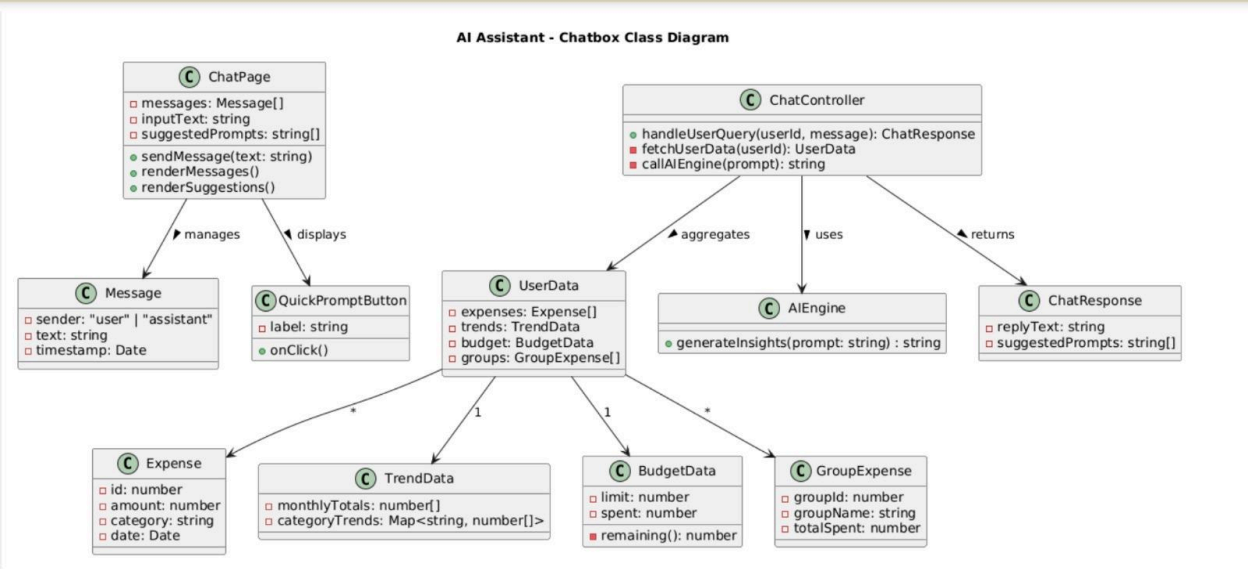
Sprint 5: AI Financial Advisor & Finishing Touches

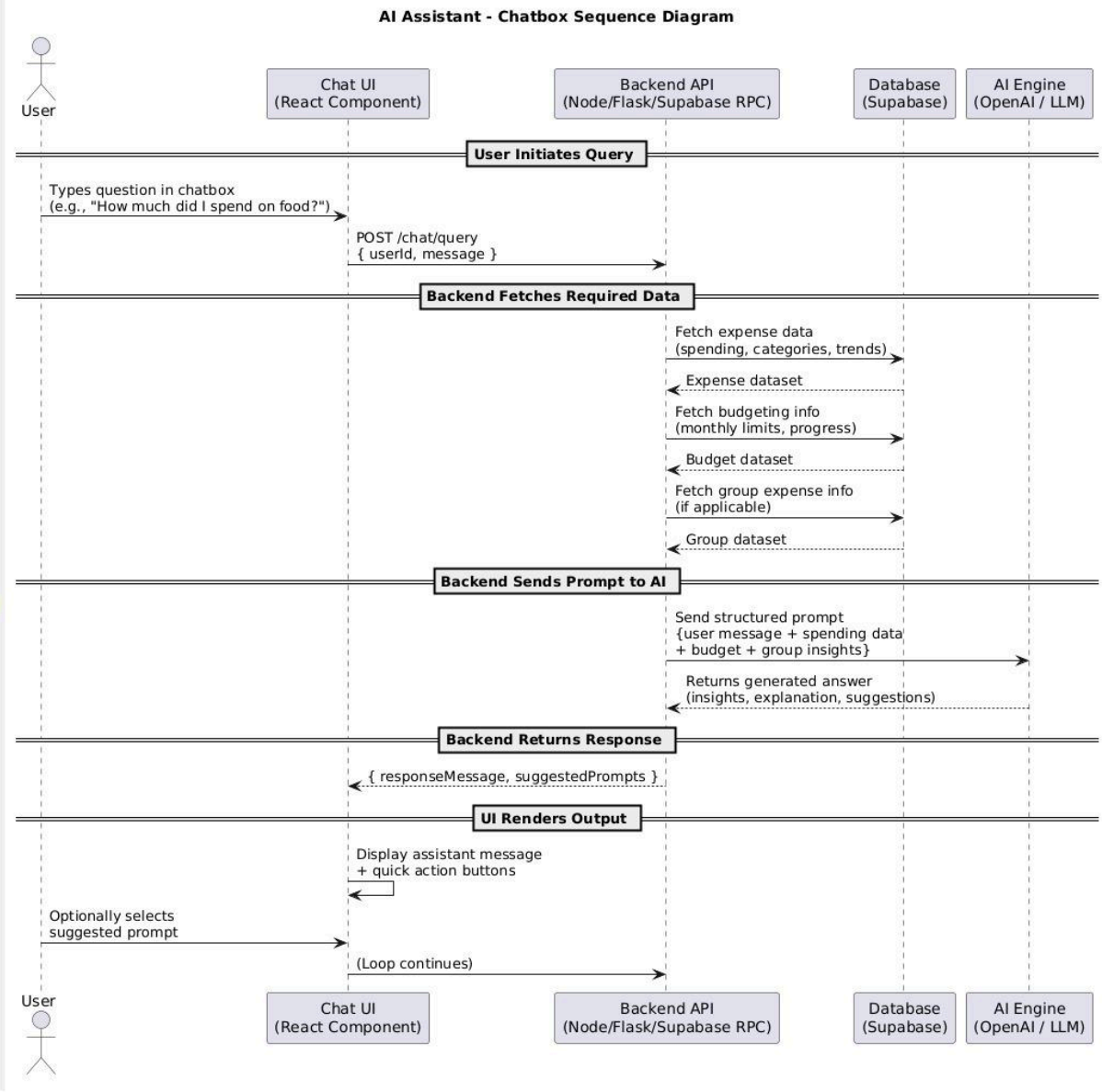
Goal: To implement the "smartest" feature of our app, the AI Financial Advisor, and complete all other planned features.

Key Activities:

- **Frontend Team:**
  - Build the final UI for the **AI Chatbot** (Financial Advisor).
  - Build all remaining pages, such as Settings, Support, and the UI for adding **Recurring Expenses**.
- **Backend & AI Team:**
  - Develop the complete backend logic for the **AI Financial Advisor**, including its prompts and API endpoint.

    - Implement the backend logic and database changes needed for the **Recurring Expenses** feature.
    - Review all existing endpoints, optimize for performance, and ensure all security is locked down.
- **Integration:**
    - Connect the new AI Chatbot UI to its backend endpoint.
    - Connect the Recurring Expenses UI to its new backend logic.





Sprint 6: The Final Push - Integration and Deployment

Goal: The goal for this final sprint is for all teams to come together, merge all the features, conduct final testing, and launch a stable version of the application.

Key Activities (All Teams):

- **Feature Integration:** We will connect all the independent parts of the app: personal tracking, group splitting, and the AI features, to ensure they work together seamlessly.
- **Final Testing:** The entire team will dedicate time to testing every part of the live application, trying to find and fix any remaining bugs or user experience issues.

- **Deployment:** We will deploy the final, polished version of our app to a live server, ready for our final demonstration.
- **Documentation:** We will complete the final project report and user guide to wrap up the project.