Up In the Air

Finance Flow

Software Engineering Project

Name: Brian Larry, Dagm Kebede, Chrishelle Culmer, Vinh Le, Nkosi Boyce

Semester: Fall 2024

Group number: 3

Coordinator: Chrishelle Culmer

Name of the Guide: Dr. Tushara Sadasivuni

Date: 11/22/24

Table of Contents

Table of Contents 2
Section 1: Introductions 4
1.1 Software Engineers' information 4
1.2 Planning and Scheduling 5
1.3 Teamwork Basics 5
Section 2: Problem Statement 6
High-Level Overview of the Product 6
Target Audience 7
Problem Solved 7
Alternatives 7
Why It's Worth Developing 7
Top-Level Objectives and Differentiators 7
Competitors and Novel Approach 8
Feasibility and Technology 8
Technical Interest 8
User and Admin Access 8
Section 3: Context Diagram 9
Section 4: Activity Diagram 11
Section 5: System Requirements 12
5.1 Use Cases 12
5.2 Requirements 28
5.3 Use Case Diagrams 42
Section 6: Database Management 45
Section 7: Class Diagram 50
Section 8: Behavioral Modeling 51
Section 9: Implementation 52
Section 10: Testing 53
10.1 Test cases 53

10.2 Unit Testing 68
Section 11: Architectural Modeling 81
References 84

Section 1: Introductions

1.1 Software Engineers' information

Brief resume, skills, and CS experience:

Brian Larry; Senior for Computer Science.

Experience:

Productive Software; Microsoft office, Google Drive, Communication software.

Programming languages; Python, HTML, JavaScript.

Database management; SQL, Amazon RDS,

Operating Systems; Windows, Ubuntu Linux.

Hardware; Software installation, Tech support, Computer building and parts.

Other skills; Critical thinking, adaptive, problem solving.

Nkosi Boyce; Senior for Computer Science

Experience:

Product Software: Microsoft Office, Google Drive, Github, Comunication Software

Programming Languages: Python, C, HTML

Operating Systems: Windows, Linux

Hardware: Computer building, Software Installation

Other skills: Troubleshooting, Problem solving

Vinh Le; Junior for Computer Science

Experience:

Product Software: Visual Studio Code, Google Drive, GitHub, Communication Software

Programming Languages: Python, HTML, Java, CSS

Database management; Firestore

Operating Systems: Windows, Linux

Hardware: Computer building, Software Installation

Other skills: Problem solving

Chrishelle Culmer; Senior for Computer Science

Experience:

Product Software: Microsoft Office, Communication Software

Programming Languages: Java, HTML

Operating Systems: Windows, IOS

Hardware: Software Installation, Security Systems

Other Skills: Organization, Critical thinking

Dagm Kebede; Senior for Computer Science

Experience:

Product Software: Microsoft Office, Google Drive, GitHub, Communication Software, MySQL

Product Languages: Java, HTML, CSS,

Operating Systems: Windows, Linux, MacOS

Hardware: Computer building, Software Installation, Technical Support, Troubleshooting

1.2 Planning and Scheduling

Assignee Name	Email address	Task	Duration (hours)	Dependency	Due Date	Evaluation
Brian Larry	Blarry2@student.gsu.edu	Test cases	2 hours	None	11/22/2024	100%
Vinh Le	vle34@student.gsu.edu	Test cases	2 hours	None	11/22/2024	100%
Nkosi Boyce	nboyce1@student.gsu.edu	Test cases	2 hours	None	11/22/2024	100%
Chrishelle Culmer (coordinator)	cculmer1@student.gsu.edu	Test cases	2 hours	None	11/22/2024	100%
Dagm Kebede	dkebede1@student.gsu.edu	Test cases and testing	4 hours	None	11/22/2024	100%

1.3 Teamwork Basics

1. Work Norms: Work on this project should be distributed firstly based on each member's strengths and weaknesses. For example, if one member is better at artwork than the others, they might get the diagrams or other artistic assignments more often than the other group members. Afterward, the work would be split evenly for the remainder of the workload. The work would then be reviewed as a group, and a discussion

would be had if any member has an opinion on how the work could be changed. Split decisions will be voted on.

2. Facilitator Norms: The role of the facilitator will rotate each sprint so it will be fair to the group members. The facilitator is responsible for splitting the work between themselves and the other members. The facilitator also makes sure that the project is progressing at an acceptable rate.

3. Communication Norms: Communication within the group will mainly be held through the Discord application.

4. Meeting Norms: As most of the group members' schedules are too varied to meet in person, the meeting will also be held on Discord when necessary. If meetings are held, they will happen in the evening so it won't conflict with any classes that a group member or members will be in.

5. Consideration Norms: As meetings will be held online, eating/drinking during meetings is allowed as long as it doesn't distract the group member too much. If someone is dominating the conversation, we'll pause and have a section for if anyone wants to ask questions or give their opinion on a given topic.

Difficult behavior can be challenging when trying to get work done as a group. If a group member is displaying a form of difficult behavior, it's best to resolve it as soon as possible. If a member is talking too much, it would be best to talk to that group member in private or find a way to open the floor to the other members such as asking, "Any questions?" or "Would anyone like to add to what this person said?". The same can be used for people who don't speak as much, giving them a chance to speak if they felt too shy to interject in the conversation. Furthermore, if a group member is being difficult to work with, such as arguing or constantly complaining, The facilitator or other group member can talk to that member in private and explain to them that their behavior is a hinderance to the group's progress.

Additionally, to combat the possibility that group doesn't know what to do, we will have a discussion of what we had accomplished already a create a plan to improve what we have or begin a new section of the project. Taking breaks can also help the group refresh themselves and avoid burnout. Group members who don't seem to be completing their portion of the project will be politely reminded that their contributions to the project are vital and need to be finished before the due date. To keep the team on track, we'll have a recap at the end of each meeting that describes what was discussed and what needs to be done.

Section 2: Problem Statement

High-Level Overview of the Product

This financial management web application is designed as a comprehensive tool that goes beyond mere expense tracking. It empowers users to cultivate better spending habits and achieve overall financial wellness. The app actively monitors all linked accounts and transactions, providing real-time alerts when users approach their spending limits. Utilizing artificial intelligence, it tracks recurring payments like subscriptions and offers personalized insights to help users manage their finances effectively. The overarching goal is to promote healthy financial habits through regular engagement, smart budgeting, and proactive notifications.

Target Audience

The app is targeted at individuals seeking greater control over their finances, from casual savers to those aiming to build robust financial habits. It's ideal for anyone needing an intuitive way to monitor multiple accounts and receive timely updates without constant manual checks. Additionally, users who struggle with debt management or tracking subscriptions will find significant value in this application.

Problem Solved

While similar tools like Mint, Credit Karma, and bank applications from Chase offer basic financial tracking, they often fall short in providing personalized interactions or real-time notifications that encourage better financial management. This app addresses these gaps by actively engaging users and helping them develop better financial practices.

Alternatives

Existing solutions like Mint and Credit Karma deliver some level of financial overview, but this app differentiates itself with a high level of interactivity and a proactive approach to user engagement. Real-time notifications, visual alerts for budget status, and AI-driven insights set this application apart from competitors.

Why It's Worth Developing

In today's fast-paced digital landscape, users require a financial management tool that simplifies complexity and enhances their understanding of their financial status. This app is compelling not just as a tracking tool but as a financial coach that actively aids users in managing their money more effectively. It provides actionable insights and guidance, making financial literacy accessible and engaging.

Top-Level Objectives and Differentiators

- **Objective**: The primary goal is to enhance users' financial habits through continuous interaction and accessibility. The color-coded interface offers immediate visual feedback on spending patterns, while real-time alerts keep users informed and engaged. This high level of interactivity fosters mindfulness in financial management compared to traditional apps.
- **Differentiators**: Key features include:
    - Real-time spending notifications.
    - Color-coded UI reflecting budget status (green for low spending, red for nearing/exceeding limits).
    - Proactive subscription management with alerts for any changes in charges.

- AI insights for budget setting and spending prioritization.

Competitors and Novel Approach

While Mint, Credit Karma, and traditional banking apps provide some level of financial oversight, this application's real-time alerts and visually engaging cues make it unique. The interactive design, alongside regular notifications and AI-driven insights, ensures users maintain awareness of their financial behaviors and are motivated to improve their habits over time.

Feasibility and Technology

The application can be built using modern, scalable technologies. The front end will utilize React to create a dynamic and responsive user experience. Cloud Firestore will be employed as the database for secure user data storage. The integration of Plaid will facilitate secure bank account connections, enabling real-time transaction tracking and updates. Furthermore, AI will be leveraged to analyze spending patterns, set budgets, and notify users of changes in recurring payments, such as increased subscription charges.

Technical Interest

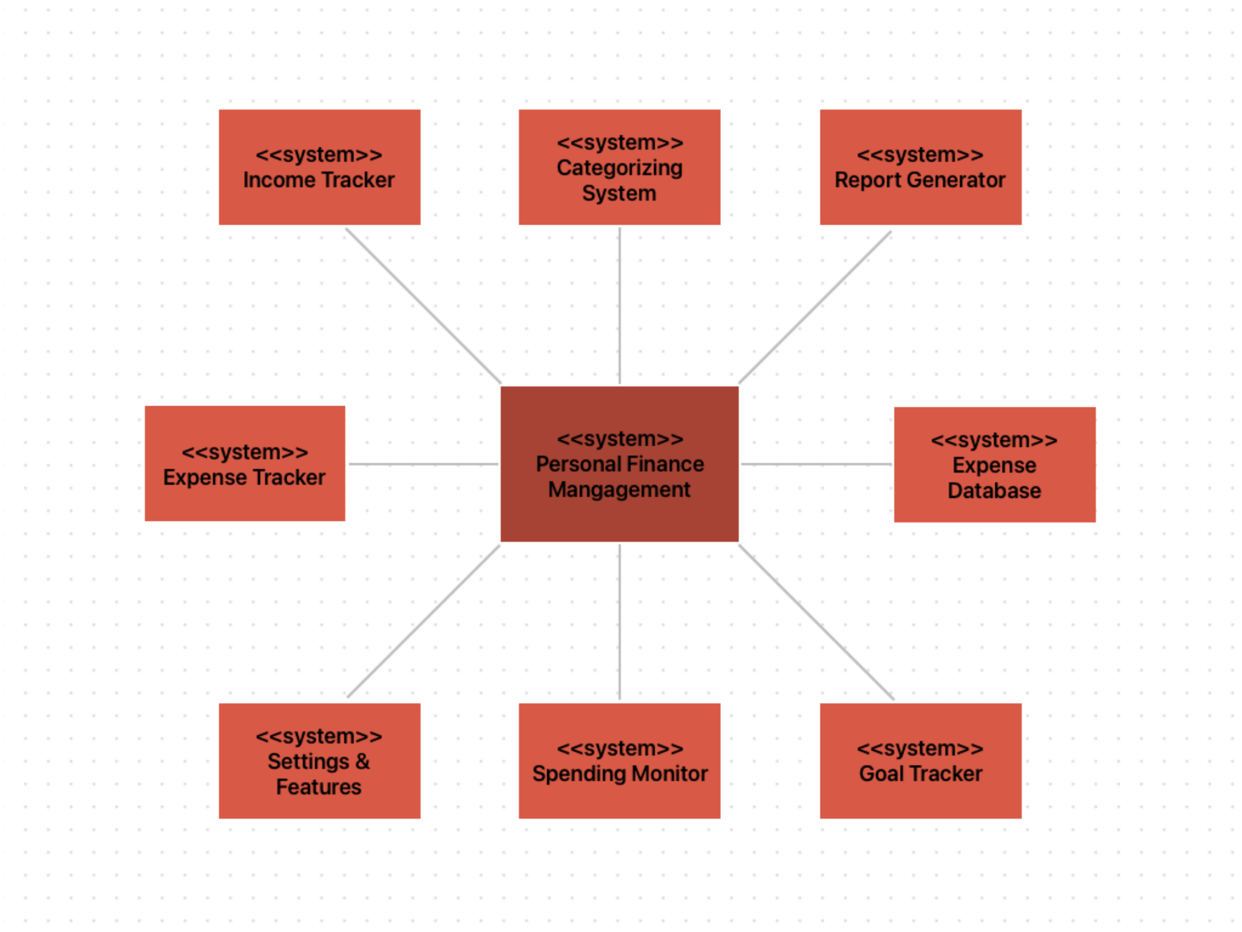
From a technical perspective, the application transforms a straightforward concept—financial management—into a dynamic, user-friendly solution that alleviates the mental burden associated with budgeting. It goes beyond tracking by providing features like spending prioritization, cost-cutting suggestions, and automatic flagging of low-priority transactions. The AI component ensures accuracy in spending oversight and minimizes the risk of overspending.

User and Admin Access

The system will include secure client logins to ensure that user data remains personalized and protected. Admin access will be established to manage user accounts and oversee data management, allowing for improved system oversight and maintenance.

This refined overview encapsulates the essential features and goals of your financial management application, highlighting its unique value proposition in the market.

Section 3: Context Diagram


<<system>> Income Tracker
<<system>> Categorizing System
<<system>> Report Generator
<<system>> Expense Tracker
<<system>> Personal Finance Mangagement
<<system>> Expense Database
<<system>> Settings & Features
<<system>> Spending Monitor
<<system>> Goal Tracker

Income Tracker:

- tracks income from your job and other secondary source of income
- provides the ability to add income manually

Expense Tracker:

- tracks expense made
- able to search for expense made by amount date, and the type of expense
- provides the option to add expense manually

Spending Monitor

- tracks mostly spending and lets user know increase or decrease in percentage from previous month
- notifies user every morning (time can be adjusted when to receive notification) about their available spending per day/month
- checks overall limit that is set
- checks spending limit set per category (ex: entertainment, bills, etc…)
- color of app changing the closer they get to their monthly spending

Expense Database

- database that contains user spending and other tracked data

- updates and removes data

Categorizing System

- used for reading the charges from user’s bank account and placing them in a category of charges

Goal tracker

- calculates your yearly income and current spending to make suggestions on where to cut spending plus what the daily and monthly spending should be

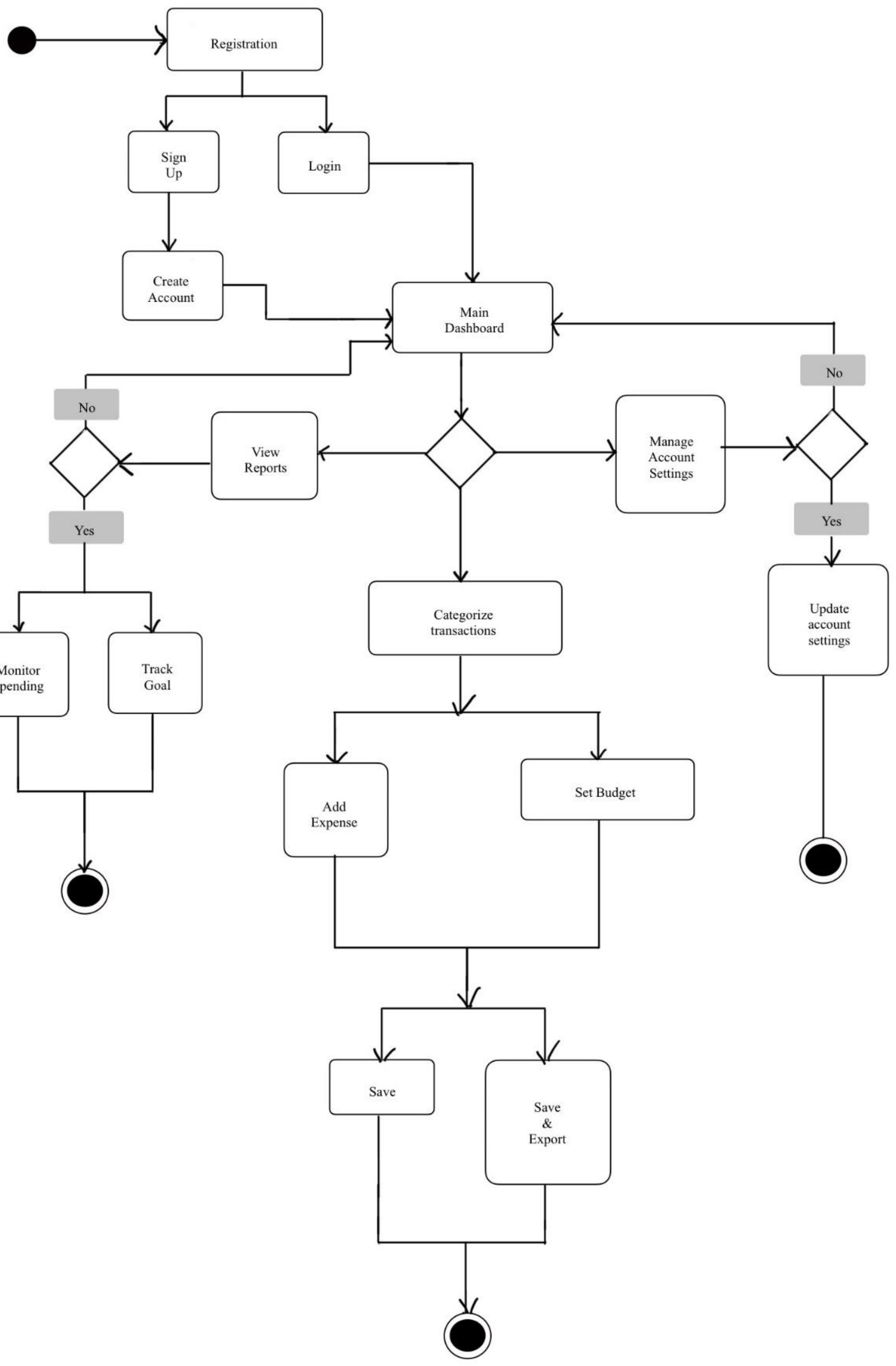
Settings and Features

- allows user to adjust the settings such as account information or changing features (dark mode, notification, etc…)

Report Generator

- creates yearly report of income, expense, and goals made, goals achieved, and so forth throughout the year
- creates report for budget limit created in categories of expenses

Section 4: Activity Diagram



Section 5: System Requirements

5.1 Use Cases

Use Case no.: 1

Use Case Name: Adding an Expense

Actors: User, Finance App

Description:

- The user opens the app.
- The user chooses the "Add Expense" option.
- The user adds a name and the amount of the expense.
- The user saves these changes.
- The expense is added to their "Expense" list.

Alternate Path: The user can back out of the tab to cancel their expense addition.

Exception Path: If the user tries to save an added expense without a name or a price value, the program will ask the user to complete the addition properly

Pre-condition: The user is required to be logged into the application

Post-condition: The list of expenses is updated along with the predetermined budget.

Use Case no.: 2

Use Case Name: Deleting an Expense

Actors: User, Finance App

Description:

1. The user selects the expense they want to delete.
2. The user selects the "delete" option.
3. The user confirms their decision.
4. The expense is deleted from the expenses list.

Alternate Path: The user can cancel their deletion and return to the expense list.

Exception Path: The app returns an error if the user tries to delete an expense that doesn't exist.

Pre-condition: There must be an existing expense to delete.

Post-condition: The expense will be removed from the user's expense list.

Use Case no.: 3

Use Case Name: Creating Monthly Reminders

Actors: User, Finance App

Description:

- User selects an expense to set a reminder for
- User selects the reminder frequency (weekly, monthly, etc.)
- User selects the day to send reminder
- User saves their options

Alternate Path: The user can cancel this reminder and select another reminder or use the application's other functions.

Exception Path: If there is no expense to set a reminder for, the app will notify the user of this and suggest that they create an expense first.

Pre-condition: There must be an existing expense to set a reminder for

Post-condition: The reminder will be placed and notify the user at the specified time.

Use case #: 4
Name of the Use case: Login/Sign up
Actors: User
Description:

- User can create a new account by providing their email and password.
- Once it's been made they would be able to log in.
- If they already have an account they can log in using existing credentials.
- Once they have logged in they will be directed to the main dashboard

[Exception path]:

- If the user enters incorrect credentials, they are prompted with an error message stating that what they inputted was wrong and that they must change.
- Creating an account will fail if they're already a registered email.

[Alternate path]:

- There will be a forgot password link to reset passwords if ever forgotten.
- (Possible sign-in through google?)

[Pre-condition]:

- Must have an email address.

[Post-condition]:

- Once logged in they'll be put into the app's dashboard

- Signing up creates a new account.

Use case #: 5
Name of the Use case: Income Tracking
Actors: User
Description:

- User are able to check their income from their jobs, salary, and/or any other way of getting income.
- Can also input income manually with date, amount, and source for information
- All income sources are stored.

[Exception path]:

- If given invalid data(example: negative numbers) will be prompted with an error message
- If not saved, will prompt the user of unsaved data.

[Alternate path]:

- Can put what kind of income they have (example: investing, salary, etc.)
- Can edit or delete income entries.

[Pre-condition]:

- Must be logged into the app.

[Post-condition]:

- Income is stored and recorded.

Use case #: 6
Name of the Use case: Track expenses
Actors: User
Description:

- Can view all expenses made by entering the amount, date, and type of expense.
- Would have all of the expenses tracked and stored.

[Exception path]:

- If given invalid data (example: negative numbers) will be prompted with an error message.
- If close or over income would give an error warning to the user.
- If not saved, will prompt the user of unsaved data.

[Alternate path]:

- Can set up recurring and subscriptions for tracking.
- Can put expenses into categories

[Pre-condition]:

- Must be logged in

- Must have income stored.
- Must have added an expense or deleted expense from other cases

[Post-condition]:

- Expenses are put into stored and recorded.

Use case #: 7
Name of the Use case: Expense Categories
Actors: User
Description:

- User can create expense categories examples being Groceries, Rent, Utilities, etc.
- Categories can be put into a priority setting where Rent could be 10 for priority which is the highest and entertainment can be 0 for its not needed but wanted.

[Exception path]:

- Will have errors if creating a category with the same name.
- If not saved, will prompt the user of unsaved data.

[Alternate path]:

-

[Pre-condition]:

- Must be logged in.

[Post-condition]:

- A category will be made when created.

Use case #: 8
Name of the Use case: Set Financial Goals
Actors: User
Description:

- User can set financial goals for either emergency funds, a major purchase, etc.
- App will track progress by monitoring the input income, expenses, etc and putting it towards a goal.
- User will be able to set a target amount and/or timeframe
- App will update on progress if wanted.

[Exception path]:

- Inputting invalid data example (negative numbers) prompts and error
- Inputting Goals of the same name prompts an error
- If not saved will be prompted about unsaved data

[Alternate path]:

- User can adjust the goal or timeline

- Can have multiple goals at once

[Pre-condition]:

- Must be logged in
- Must have at least Income and expenses inputted.

[Post-condition]:

- Financial goal is saved and tracked.

Use case #: 9
Name of the Use case: Budget in Categories
Actors: User
Description:

1. User can set specific budget limits for various categories (example: food, rent, entertainment, etc.)
2. Will be prompted when approaching or exceeding their budget based on how much is spent
3. Will show how much has been spent overall in said budget category.

[Exception path]:

1. Inputting invalid data (example: negative numbers) prompts an error message
2. Exceeding budget will prompt a notification
3. If a budget hasn't been saved will be prompted with an unsaved message

[Alternate path]:

1. Can adjust budget at any time

[Pre-condition]:

1. Must be logged in
2. Must have income imputed

[Post-condition]:

1. Budget limits are set for categories once made.

Use case #: 10
Name of the Use case: Notification for bills/payments
Actors: User
Description:

1. The app can be set up to have reminders for upcoming payments to avoid late fees or missing it completely.
2. Notifications can be customized for different types of payments.
3. Can also have recurring reminders for bills or payments that happen on a set schedule.

[Exception path]:

1. Notifications might fail due to connection issues.

[Alternate path]:

1. User can disable notifications
2. User can snooze notifications for later reminders
3. Can add or remove notifications at will
4. Can add them to the calendar case.

[Pre-condition]:

1. Must be logged in
2. Must have expenses that would want notifications for

[Post-condition]:

1. Notifications are made and sent whenever any upcoming payments or bills appear.

Use case #: 11
Name of the Use case: View Financial Reports
Actors: User
Description:

1. User can view their reports and charts based on their income and expenses/spending.
2. The app will generate reports based on all the financial data input and can be filtered by date, categories, goals, and amount.
3. Will be updated in real time to reflect the latest financial activity.

[Exception path]:
Report generation might fail if any data is corrupted or incompleted.
[Alternate path]:
User can filter reports by category, time, goal, etc.
[Pre-condition]:
User must be logged in
User must have financial data inputted
[Post-condition]:
A report is displayed.

Use case #: 12

Name of the Use case: Export Financial Data
Actors: User
Description:

- User can export their financial data, income, expenses, budgets, etc.
- The export can either be structured in the format of CSV or PDF.

[Exception path]:

- Fails if there are any connection errors.

[Alternate path]:

- User can select different file formats
- Can have the system export data on a schedule.

[Pre-condition]:

- Must be logged in
- Must have financial data inputted

[Post-condition]:

- A downloadable file containing the financial data is generated.

Use case #: 13
Name of the Use case: Backup and Restore Data
Actors: User
Description:

App gives the user an option to backup their financial data to either cloud or local storage device
If the device is changed or data is lost, the user can restore their data.

[Exception path]:

Backup fails if there are any connection errors
Also fails if not enough storage
Data could become corrupt

[Alternate path]:

Can manually initiate a backup manually or automatically
Data can be restored on a new device

[Pre-condition]:

Must be logged in.
Must be connected to the internet for cloud backup

[Post-condition]:

Data is safely backed up and able to be restored later.

Use case #: 14
Name of the Use case: AI spending Assistant
Actors: User, AI assistant.
Description:

The AI will analyze the user’s income and spending pattern and provide insight or recommendations.
User will be given advice that can be personalized for either saving money, reducing expenses, reaching financial goals or even spending more efficiently.
Ai will guide the user to better financial habits if allowed.

[Exception path]:
AI could give inaccurate data
[Alternate path]:
User can customize suggestions based on what suits their need the most
[Pre-condition]:
Must be logged in
Must have financial data
[Post-condition]:
AI will provide insight and suggestions.

Use case #: 15
Name of the Use case: Log out
Actors: User
Description:

The user logs out of their account so no one else can view their info
This will clear the data and disconnect the user from the app.

[Exception path]:
App may fail to log out if there are connection issues with the network
[Alternate path]:
User can clear session cookies
Can also have force logout in the settings
[Pre-condition]:
User must be logged in
[Post-condition]:

User is logged out without any data remaining.

Use Case #16: Password Reset

Actor: User

Description: A user resets their forgotten password.

Preconditions: User has an account and access to their registered email.

Postconditions: User resets their password successfully.

Steps:

1. User clicks "Forgot Password" on the login page.
2. User enters their email and submits the request.
3. System sends a password reset link to the user’s email.
4. User clicks the link and is redirected to a reset password page.
5. User enters a new password and confirms it.
6. System updates the password and confirms the change.
7. User logs in with the new password.

Alternative Path:

User can choose to cancel the password reset process at any time.

Exception Path:

If the user enters an unregistered email, the system prompts an error message indicating that the email does not exist.

Use Case #17: Edit Transaction

Actor: User

Description: User edits details of an existing transaction.

Preconditions: User is logged in and transaction exists.

Postconditions: Transaction details are updated.

Steps:

1. User navigates to the "Transactions" page.

2. User selects a transaction to edit.
3. User modifies the details in the transaction form.
4. User submits the changes.
5. System updates the transaction details.

Alternative Path:

User can choose to cancel the edit and return to the transactions page without saving changes.

Exception Path:

If the user tries to submit without entering required fields, an error message prompts them to complete the form properly.

Use Case #18: View Budget Overview

Actor: User

Description: User views an overview of all set budgets and their current status.

Preconditions: User is logged in.

Postconditions: User sees budget statuses, including any overages.

Steps:

1. User navigates to the "Budget Overview" page.
2. System displays all budgets along with spent amounts and remaining balances.
3. User reviews the budget performance.

Alternative Path:

User can filter the view by specific categories or timeframes.

Exception Path:

If there are no budgets set, the system prompts the user to create a budget first.

Use Case #19: Edit Financial Goals

Actor: User

Description: User modifies details of an existing financial goal.

Preconditions: User is logged in and goals exist.

Postconditions: Goal details are updated.

Steps:

1. User navigates to the "Financial Goals" section.
2. User selects a goal to edit.

3. User changes the goal details (amount, deadline).
4. User submits the changes.
5. System updates the goal details.

Alternative Path:

User can cancel the edit process and return to the financial goals section without saving changes.

Exception Path:

If the user tries to submit an invalid goal, the system prompts an error message indicating the issue.

Use Case #20: Delete Financial Goals

Actor: User

Description: User deletes a financial goal that is no longer relevant or desired.

Preconditions: User is logged in and goals exist.

Postconditions: Goal is removed from the system.

Steps:

1. User navigates to the "Financial Goals" section.
2. User selects a goal and chooses to delete it.
3. System confirms the action with the user.
4. User confirms deletion.
5. System removes the goal from the records.

Alternative Path:

User can cancel the deletion process and return to the financial goals section without deleting anything.

Exception Path:

If the user tries to delete a goal that does not exist, the system returns an error message.

Use Case #21: Customize Dashboard

Actor: User

Description: User customizes the layout and information displayed on their dashboard.

Preconditions: User is logged in.

Postconditions: Dashboard displays information according to user preferences.

Steps:

1. User navigates to the "Customize Dashboard" section.
2. User selects widgets and information to display (transactions, budgets, goals, etc.).

3. User arranges the layout of the dashboard.
4. User saves their settings.
5. System updates the dashboard to reflect user preferences.

Alternative Path:

User can revert to default settings if they wish to reset their dashboard.

Exception Path:

If the user tries to save an invalid layout, the system prompts an error message indicating the issue.

Use Case #22: Sync Transactions Automatically

Actor: User

Description: Transactions are automatically fetched and synced from user's bank accounts via Plaid API.

Preconditions: User has linked their bank accounts through Plaid.

Postconditions: New transactions are updated in the user’s account regularly.

Steps:

1. System periodically triggers a sync operation using Plaid API.
2. Plaid API returns recent transaction data.
3. System processes and categorizes transactions based on predefined rules or user settings.
4. Transactions are stored in the database and displayed on the user’s transactions page.

Alternative Path:

User can manually trigger a sync operation from their account settings.

Exception Path:

If the sync fails due to connectivity issues, the system alerts the user and suggests retrying.

Use Case #23: Link/Unlink Bank Accounts

Actor: User

Description: User links or unlinks their bank accounts using Plaid.

Preconditions: User is logged in and has access to account settings.

Postconditions: User’s bank accounts are linked to or unlinked from their application profile.

Steps:

1. User navigates to "Bank Accounts" settings.
2. User chooses to link a new account and follows prompts to authenticate and link their bank via Plaid.

3. For unlinking, user selects an existing linked account and confirms they want to unlink.
4. System updates the account link status in the database.

Alternative Path:

User can review linked accounts before making changes.

Exception Path:

If the linking fails due to invalid credentials, the system prompts an error message indicating the problem.

Use Case #24: Visualize Spending Trends Over Time

Actor: User

Description: User accesses visualizations showing spending trends over time.

Preconditions: User has transaction history stored.

Postconditions: Visual data representation is available for user analysis.

Steps:

1. User selects the "Trends" option from the dashboard or menu.
2. User chooses parameters for the data visualization (e.g., time period, categories).
3. System fetches relevant data and generates visualizations like line charts or heat maps.
4. User reviews the visual data to understand spending habits and trends.

Alternative Path:

User can select predefined reports or create custom visualizations.

Exception Path:

If there is insufficient data to generate visualizations, the system alerts the user.

Use Case #25: Manage Savings Goals with Visual Trackers

Actor: User

Description: User sets and manages savings goals with the help of visual trackers that show progress.

Preconditions: User is logged in and interested in setting savings goals.

Postconditions: Savings goals are set and tracked visually.

Steps:

1. User navigates to the "Savings Goals" section.
2. User sets a new goal, specifying the target amount and timeline.
3. System creates a visual tracker for the goal.

4. User can view and update their progress on the goal, with the system adjusting the visual tracker accordingly.

Alternative Path:

User can receive alerts when nearing their goal amount.

Exception Path:

If the user tries to set an unrealistic goal (e.g., negative amount), the system prompts an error message.

Use Case #26: Financial Challenges and Goals

Actor: User

Description: User participates in financial challenges and sets personal financial goals.

Preconditions: User is logged in and interested in improving financial habits.

Postconditions: Challenges are active and goals are tracked.

Steps:

1. User navigates to the "Challenges" section and views available challenges (e.g., "No Spend November", "Save $500 in 3 months").
2. User joins a challenge and sets up personal financial goals.
3. System tracks user’s progress against these challenges and goals.
4. User receives updates and notifications on milestones or completions.

Alternative Path:

User can opt-out of challenges at any time.

Exception Path:

If the user fails to meet a milestone, the system sends a reminder with motivational tips.

Use Case #27: Bill Splitting and Reminders (continued)

Actor: User

Description: User splits bills with friends and sets reminders for shared expenses.

Preconditions: User has shared expenses and linked contacts or other users.

Postconditions: Expenses are split and reminders are set.

Steps:

1. User enters a transaction that includes a shared expense.

2. User selects the option to split the bill and enters the contacts' details or selects from saved contacts.
3. System calculates the split amount and sends notifications to involved parties if they are app users, or creates a reminder for the user to inform others.
4. System sets up reminders for the user to collect or send payments based on the split.

Alternative Path:

User can edit the split details after saving.

Exception Path:

If the user tries to split an amount that exceeds the total transaction, the system prompts an error message indicating the issue.

Use Case #28: Enhanced Subscription Management

Actor: User

Description: User manages and tracks subscriptions, including automated detection of duplicate charges or unused subscriptions.

Preconditions: User has active subscriptions stored in the app.

Postconditions: Subscriptions are managed efficiently with optimization suggestions.

Steps:

1. User navigates to the "Subscriptions" section.
2. System displays all active subscriptions with details such as cost, renewal date, and usage activity.
3. System alerts the user to potential duplicate subscriptions or suggests cancelling rarely used services.
4. User can update subscription details or cancel directly through the app interface.

Alternative Path:

User can set reminders for upcoming subscription renewals.

Exception Path:

If the user tries to modify a subscription that does not exist, the system prompts an error message.

Use Case #29: Predictive Budgeting with AI

Actor: User

Description: AI models predict future spending and budget requirements based on historical data.

Preconditions: Sufficient historical spending data is available.

Postconditions: Predictive budgets are created.

Steps:

1. User accesses the "Predictive Budgeting" feature.
2. System analyzes past spending habits, recurring expenses, and financial events.
3. AI predicts future spending for upcoming periods and suggests budget allocations.
4. User reviews predictions, adjusts budgets as necessary, and sets them into action.

Alternative Path:

User can compare predictive budgets against their current spending.

Exception Path:

If the historical data is insufficient for predictions, the system prompts the user to input more data.

Use Case #30: Currency Exchange Feature

Actor: User

Description: User converts transactions between different currencies using real-time exchange rates.

Preconditions: User is logged in and views transaction details.

Postconditions: Transactions are displayed in the user's preferred currency.

Steps:

1. User navigates to the "Transactions" page.
2. User selects a transaction listed in a foreign currency.
3. User clicks on "Convert Currency".
4. System fetches real-time exchange rates.
5. User selects the target currency.
6. System displays the transaction amount in the selected currency.

Alternative Path:

User can set their preferred currency for future transactions.

Exception Path:

If the currency conversion fails due to API issues, the system alerts the user with an error message.

Use Case #31: Real-Time Fraud Detection Alerts

Actor: System, User

Description: The system monitors for and alerts users to potential fraudulent activities.

Preconditions: User’s account is active with transaction monitoring enabled.

Postconditions: User is alerted to potential fraud immediately.

Steps:

1. System continuously analyzes transaction patterns using AI algorithms.
2. System identifies a transaction that deviates significantly from established patterns.
3. System flags the transaction as potentially fraudulent.
4. User receives an immediate alert via app notification and email.
5. User reviews the transaction and confirms whether it was legitimate or fraudulent.

Alternative Path:

User can dispute a flagged transaction through the app.

Exception Path:

If the user does not respond to the alert within a specified time, the system locks the account for security.

5.2 Requirements

Use Case number: 1

Introduction: Simple list and UI for the user to add their expenses to and manage

Inputs: Expense name and associated value along with the category it associates with

Requirements Description: The user should be able to add expenses to the list. That can only happen once these conditions are met.

1. The system offers the user an interface for the user to put the expense information.
2. The input data must be valid.

Outputs: The expense should be logged in its respective category and should update the budget.

Use Case number: 2

Introduction: An option that allows users to delete an existing expense and will update the current budget

Inputs: The expense that the user wants to delete

Requirements Description: When the user chooses the “Delete Expense” button all available expense should have a selection box that the user can select if they want to delete them. The user should be able to select multiple expenses to delete at once. To acheive this, there must be:

1. A delete button on the Expense tab
2. A selection confirmation box for each expense
3. The ability to select more than one expense.

Outputs: The expense(s) will be removed from the expense list and the budget should be updated.

Use Case number: 3

Introduction: The system sets a recurring notification of subscription payments

Inputs: The due date of the expense (Subscriptions, rent, etc.)

Requirements Description:

1. The system needs a calender function
2. The system needs a notification system

Outputs: The expense’s due date will be saved so a notification can be sent at a later date.

Requirement #: 4
Use Case #: 4

Name: Login/Sign up
Introduction:
The user securely logs in or signs up to access the app.
Rationale:
The access will be secured to make sure personal information is protected.
Requirement Description:
The app will support new user sign-ups and existing user logins.

1. The system must allow new users to create an account using their email and a password.
2. Existing users should be able to log in using their existing credentials
3. Supports log-in with Google authentication.

Output:

Requirement #: 5
Use Case #: 5

Name: Track Income
Introduction:
Users are allowed to pull and categorize their income in a structured manner.
Rationale:
Tracking income will help the user manage their funds for any future endeavors.
Requirement Description:
The user should be able to add income entries, specifying the source.

4. Allows the user to enter the amount, source and date for each income entry
5. Income entries entries must be categorized(example: salary, investment)
6. Must validate the amounts to make sure there are no negatives.

Output:
Income data is stored for future use of the user.

Requirement #: 6
Use Case #: 6

Name: Track Expenses
Introduction:
Users are allowed to log and categorize their expenses to track their habits.
Rationale:
Tracking expenses allows the user to better manage their finances and understand where their money is going.
Requirement Description:
Users should be able to log their expenses, categorize them, and track their spending history

4. Users are allowed to input their amount, category, and date.
5. Categories can be preferenced by the users.
6. The user will be notified if any expenses exceed their funds.

Output:
Expenses are logged and deducted from the user's funds.

Requirement #: 7
Use Case #: 7

Name: Create expense categories
Introduction:
User is allowed to create, edit, and delete custom categories to organize their expenses.
Rationale:
Categories provide flexibility allowing the user to organize their spending.
Requirement Description:
App enables users to create custom categories for better organization/

2. Users must be able to create new custom categories.
3. Does not allow duplicate names for categories.
4. Allows the user to edit or delete categories at any time.

Output:
Categories are created and ready to be used.

Requirement #: 8
Use Case #: 8

Name: Set Financial Goals
Introduction:
Users are allowed to set financial goals and track their progress based on their income and expenses.
Rationale:
Setting financial goals helps users stay focused on a certain task or objective.
Requirement Description:
Users should be able to specify financial targets and the system should track the progress.

3. Must allow users to set target amounts and deadlines for their financial goals.
4. Should be able to track user progress toward each goal based on their income and expenses.
5. Multiple financial goals should be allowed.

Output:
Financial goals are saved and tracked.

Requirement #: 9
Use Case #: 9

Name: Budget in Categories
Introduction:
Users can set budget limits for specific categories and track them based on them
Rationale:
Budgeting helps the user control and maintain discipline.
Requirement Description:
Users should be able to create and customize different categories and receive notifications when approaching or exceeding the set limit.

2. Users are allowed to set budget limits in specific categories.
3. User will be able to view, edit or remove budgets.
4. User will be notified when approaching or exceeding said budget.

Output:
Budgets are made and able to be monitored.

Requirement #: 10
Use Case #: 10

Name: Notifications for Bills/Payments
Introduction:
Users are notified of upcoming bills and payments to ensure they are paid on time.
Rationale:
Reminders for bills help the user stay on top of payments so there are no missed ones.

Requirement Description:
Users should be able to set up recurring bill notifications and customize how frequent they appear.

4. Users are allowed to create reminders.
5. They’re able to receive recurring notifications.
6. Users can customize notification settings or disable them

Output:
Users receive notifications based on their preferences.

Requirement #: 11
Use Case #: 11

Name: View Financial Reports
Introduction:
Users is provided with financial reports that summarize their income, expenses, budgets, and progress toward financial goals
Rationale:
Financial reports allow users to analyze their financial health and make decisions based on what they view.
Requirement Description:
The app should generate a visual report based on the user's selected filters

2. User is allowed to view reports filtered by date, category, goals, etc.
3. Reports should visually be represented in either graphs, charts, tables or a combination of the three
4. Reports must be updated in real-time

Output:
A real-time financial report is given to the user.

Requirement #: 12
Use Case #: 12

Name: Export Financial Data
Introduction:
Users are allowed to export their financial data in various formats for offline use.
Rationale:
Exporting this data enables the user to have offline records for personal or business use.
Requirement Description:
Users should be able to export their financial data within given filters and file formats.

5. App must allow the user to export their financial data in CSV and PDF formats.
6. Users are able to filter which data is included in the export.
7. Exported data must be accurate and able to be downloaded.

Output:
A downloadable file containing the user's data is exported.

Requirement #: 13
Use Case #: 13

Name: Backup and restore data.
Introduction:
The app allows the user to back up their data to the cloud or a local system so it can be restored.
Rationale:
Backing up data prevents data loss and lets the user swap to devices.
Requirement Description:
Users should be able to either have automatic backups or manual ones.

3. Users are allowed to have manual and scheduled backups
4. Backups are stored securely in the cloud or local device
5. Users can restore their data on any device that is supported.

Output:
The user's data is securely backed up and able to be restored.

Requirement #: 14
Use Case #: 14

Name: AI spending assistant
Introduction:
AI feature analyzes the user's spending patterns and offers financial advice and suggestions.
Rationale:
AI assistant helps the user to make smarter financial decisions by suggesting how to optimize their finances
Requirement Description:
The AI should analyze their income, expenses, goals, and budget to provide useful information.

2. The AI must analyze the user's given financial data.
3. The AI should constantly learn from the data given and give insight
4. Users must be able to request specific financial insights from the AI.

Output:
The AI provides the user with any advice or suggestions that would help them.

Requirement #: 15
Use Case #: 15

Name: Log out
Introduction:
The app allows the user to securely log out of their account and clears any session data for their financial information to be protected.
Rationale:
Logging out is needed to maintain the security and privacy of the user's data.
Requirement Description:
Users should have the ability to log out from the app at any time and any sensitive data must no longer be available.

4. App must have an easily accessible log-out option in the interface
5. The log-out process must securely remove the user's session and clear sensitive data.
6. Provide an option to log out of all devices

Output:
The user is securely logged out of the app and the data is cleared.

Use Case number: 16

Introduction: A user resets their forgotten password through an email verification process.

Inputs: User's registered email address and new password.

Requirements Description:

- The user should be able to request a password reset via the "Forgot Password" link on the login page.
- The system must validate that the entered email is associated with an existing account before sending a reset link.
- The password reset link must expire after a predetermined time for security reasons.
- The user must be able to enter and confirm a new password, ensuring it meets defined security standards (e.g., minimum length, complexity).
- The system should update the password in the database once confirmed.

Outputs: The user successfully resets their password and can log in with the new credentials.

Use Case number: 17

Introduction: User edits details of an existing transaction.

Inputs: Transaction details to be modified, such as amount, date, category, and notes.

Requirements Description:

- The user must navigate to the "Transactions" page and select a transaction to edit.
- The system must ensure that the selected transaction exists before allowing edits.
- The input data for the transaction must be validated (e.g., required fields must not be empty, amounts must be positive).
- The system should update the transaction details in the database upon successful modification.

Outputs: The updated transaction details are saved and reflected in the user's transaction history.

Use Case number: 18

Introduction: User views an overview of all set budgets and their current status.

Inputs: N/A (viewing data).

Requirements Description:

- The user must navigate to the "Budget Overview" page to view their budget statuses.
- The system must display all budgets, including spent amounts, remaining balances, and any overages.
- The user should have the option to filter the view by specific categories or timeframes.

Outputs: The user sees a comprehensive overview of their budget statuses, enabling better financial management.

Use Case number: 19

Introduction: User modifies details of an existing financial goal.

Inputs: Financial goal details, including target amount and deadline.

Requirements Description:

- The user must navigate to the "Financial Goals" section and select a goal to edit.
- The system should ensure that the selected goal exists before allowing edits.
- The input data for the goal must be validated (e.g., no negative amounts, deadlines must be future dates).
- The system should save the updated goal details in the database.

Outputs: The modified financial goal details are saved and reflected in the user's profile.

Use Case number: 20

Introduction: User deletes a financial goal that is no longer relevant or desired.

Inputs: Selected financial goal to be deleted.

Requirements Description:

- The user must navigate to the "Financial Goals" section and select a goal to delete.
- The system must confirm the deletion action with the user to prevent accidental deletions.
- If the user attempts to delete a non-existing goal, the system should provide an error message.

Outputs: The selected financial goal is removed from the system.

Use Case number: 21

Introduction: User customizes the layout and information displayed on their dashboard.

Inputs: Preferences for dashboard widgets and layout arrangement.

Requirements Description:

- The user must navigate to the "Customize Dashboard" section to modify their layout preferences.
- The system should allow users to select and arrange widgets (transactions, budgets, goals, etc.) according to their preference.
- The user preferences must be saved and applied upon the next login.

Outputs: The dashboard displays the user-selected layout and information as per their preferences.

Use Case number: 22

Introduction: Transactions are automatically fetched and synced from the user's bank accounts via the Plaid API.

Inputs: Linked bank account information.

Requirements Description:

- The system must periodically trigger a sync operation using the Plaid API to fetch recent transactions.

- The fetched transaction data must be processed and categorized based on predefined rules or user settings.
- Users should have the option to manually trigger a sync operation if desired.

Outputs: New transactions are regularly updated in the user's account and reflected in their transaction history.

Use Case number: 23

Introduction: User links or unlinks their bank accounts using Plaid.

Inputs: User's bank account credentials for linking or unlinking.

Requirements Description:

- The user must navigate to the "Bank Accounts" settings to link or unlink an account.
- The system should validate user credentials during the linking process to ensure secure access.
- Users must be able to view all linked accounts and their statuses.

Outputs: The user's bank accounts are linked or unlinked successfully, reflecting the current status in their profile.

Use Case number: 24

Introduction: User accesses visualizations showing spending trends over time.

Inputs: Parameters for data visualization (e.g., time period, categories).

Requirements Description:

- The user must select the "Trends" option from the dashboard or menu.
- The system should fetch relevant data based on the selected parameters and generate visualizations (e.g., line charts, heat maps).
- If there is insufficient data to generate visualizations, the system should alert the user.

Outputs: Visual data representation is available for user analysis, enabling better understanding of spending habits and trends.

Use Case number: 25

Introduction: User sets and manages savings goals with the help of visual trackers.

Inputs: Target amount and timeline for the savings goal.

Requirements Description:

- The user must navigate to the "Savings Goals" section to set a new goal.
- The system should create a visual tracker for each savings goal set by the user.
- Users should receive alerts when nearing their goal amount.

Outputs: The savings goals are set and tracked visually, providing motivation and progress updates.

Use Case number: 26

Introduction: User participates in financial challenges and sets personal financial goals.

Inputs: Details of the challenge and personal financial goals.

Requirements Description:

- The user must navigate to the "Challenges" section and view available challenges.
- The system should track the user’s progress against these challenges and goals.
- Users should receive updates and notifications on their progress.

Outputs: The challenges are active, and the user can track their financial goals effectively.

Use Case number: 27

Introduction: User splits bills with friends and sets reminders for shared expenses.

Inputs: Transaction details, contacts for bill splitting, and reminder settings.

Requirements Description:

- The user must enter a transaction that includes a shared expense.
- The system should allow the user to select contacts to split the bill with and calculate the split amount.
- Users must be able to set reminders for payment collection based on the split.
- If the user tries to split an amount that exceeds the total transaction, the system should provide an error message.

Outputs: The expenses are split among the selected contacts, and reminders are set for payment collection.

Use Case number: 28

Introduction: User manages and tracks subscriptions, including detecting duplicate charges or unused subscriptions.

Inputs: Subscription details including cost, renewal date, and usage activity.

Requirements Description:

- The user must navigate to the "Subscriptions" section to view their active subscriptions.
- The system should display all subscription details and alert the user to any duplicate subscriptions or rarely used services.
- Users should have the option to update subscription details or cancel subscriptions directly through the app interface.

Outputs: Subscriptions are managed efficiently, providing the user with optimization suggestions and alerts for action.

Use Case number: 29

Introduction: AI models predict future spending and budget requirements based on historical data.

Inputs: Historical spending data, recurring expenses, and financial events.

Requirements Description:

- The user must access the "Predictive Budgeting" feature to initiate predictions.
- The system should analyze the user’s past spending habits and provide budget allocation suggestions based on predictions.
- Users must be able to adjust their budgets based on these predictions and set them into action.

Outputs: Predictive budgets are created, allowing users to plan future spending effectively.

Use Case number: 30

Introduction: User converts transactions between different currencies using real-time exchange rates.

Inputs: Foreign currency transaction details and the target currency for conversion.

Requirements Description:

- The user must navigate to the "Transactions" page and select a foreign currency transaction to convert.
- The system should fetch real-time exchange rates from a reliable source and display the converted amount in the selected target currency.

- If the currency conversion fails due to API issues, the system should alert the user with an error message.

Outputs: Transactions are displayed in the user's preferred currency, allowing for accurate financial tracking.

Use Case number: 31

Introduction: The system monitors for and alerts users to potential fraudulent activities.

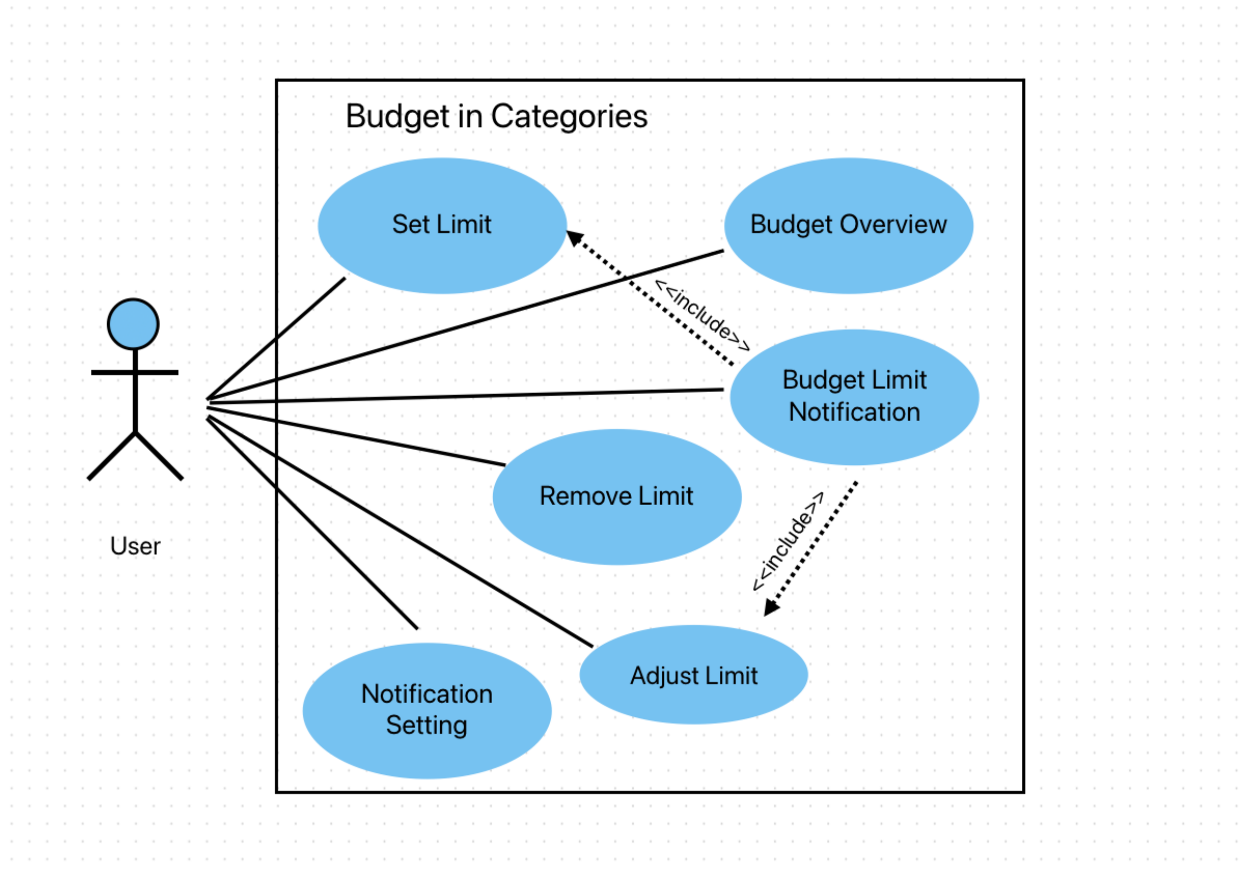
Inputs: User's transaction data and real-time monitoring parameters.

Requirements Description:

- The system must continuously analyze transaction patterns using AI algorithms to detect anomalies.
- If a transaction deviates significantly from established patterns, the system should flag it as potentially fraudulent.
- Users should receive immediate alerts via app notifications and email for any flagged transactions.
- If the user does not respond to the alert within a specified time, the system should lock the account for security.

Outputs: Users are alerted to potential fraud immediately, allowing for quick action to secure their accounts.

5.3 Use Case Diagrams

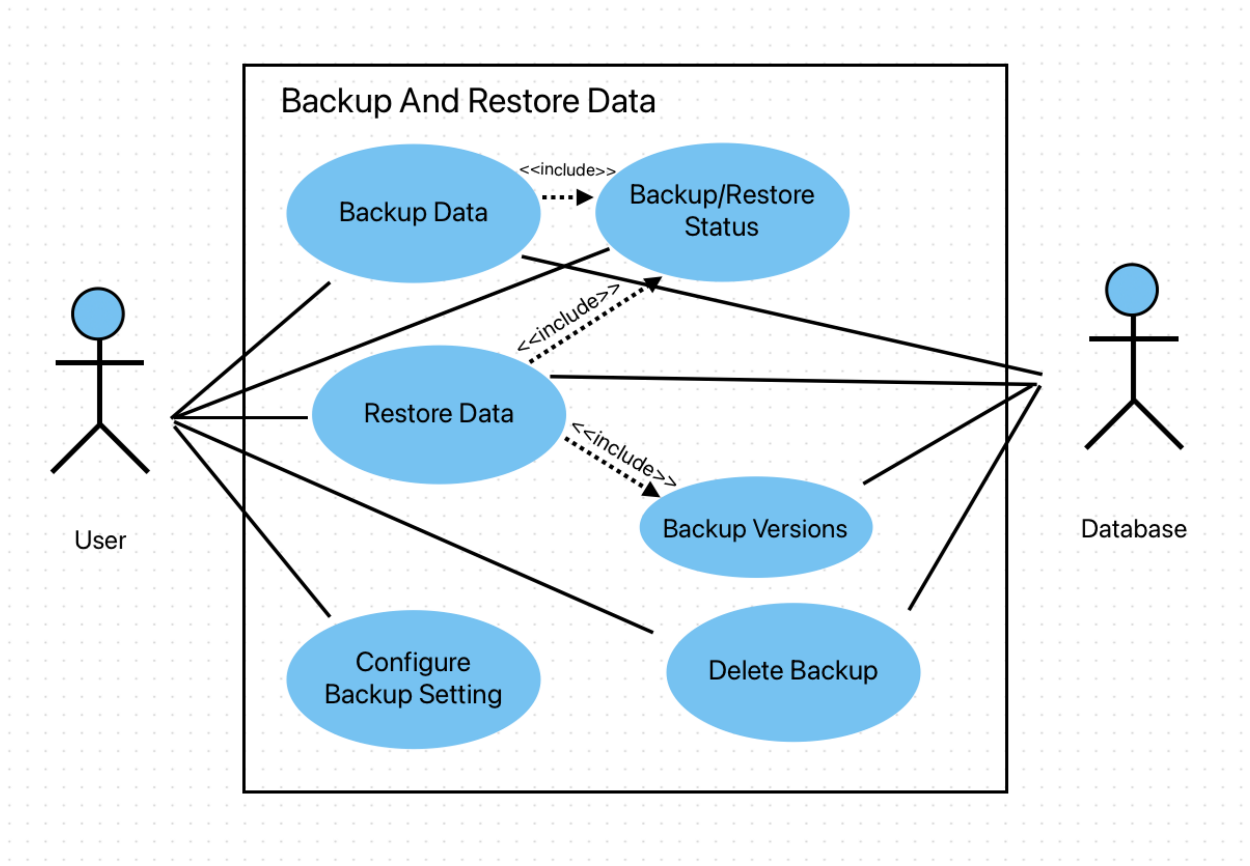


The diagram shows the following:
User can:

- set budget limit per category
- adjust budget limit per category
- remove budget limit
- receive notification when approaching budget limit or has already reached limit
- adjust notification settings (e.g., frequency of notification or turning off feature)
- provides report on spending within category
- view backup/restore status

Interactions:

- setting limit provides information to the budget notification class as it will be used if user decides to receive notification when they get past a certain threshold in their limit
- adjusting limit provides information to the budget notification class as it will be used if user decides to receive notification when they get past a certain threshold in their limit



The diagram shows the following:

User can:

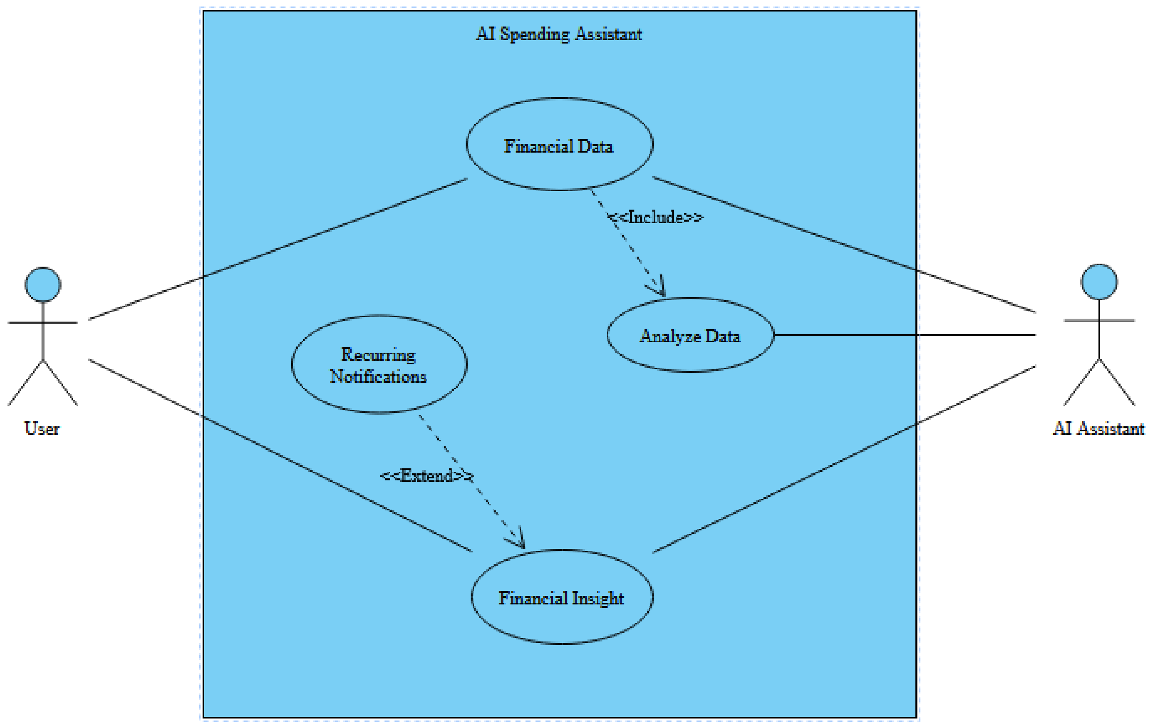
- backup data
- restore data
- configure backup settings (e.g., how often it should backup automatically, choose local or cloud storage)
- delete backup

Database can:

- store backup data
- allow user to restore data
- contain backup version to allow user to restore data from it
- delete backup

Interactions:

- once backup starts, the user would be able to see the backup status
- once restore starts, the user would be able to see the restore status
- when user initiates restore process, they will have the option to choose between backup versions if more than one exists



The diagram shows the following:

User can:

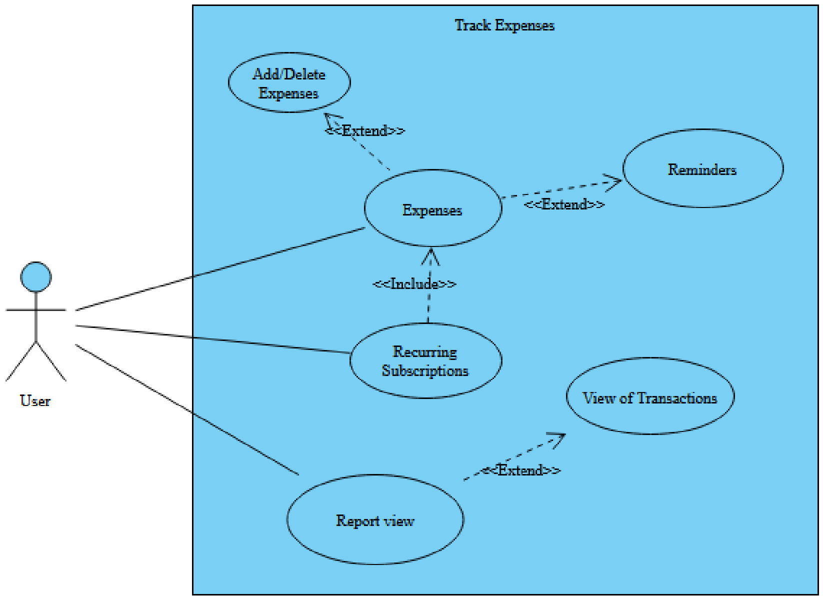
- View their own financial data
- Set recurring notifications for spending
- View financial insight from the AI assistant

AI Assistant can:

- View financial data and analyze the information
- Provides financial insight dependent on the user’s data

Interaction:

- User requests insight from the AI assistant based on their spending data.



The diagram shows the following:

User can:

- Add/Delete Expenses
- View their expenses
- View transactions in an easy to view format
- Set reminders about expenses

Interactions:

- Viewing expenses and transactions
- Adding and deleting transactions

Section 6: Database Management

FireStore Schema

1. **Users Collection**
   Stores user profile information, including authentication details and links to financial data.

Field	Data Type	Description

user_id	String (UID)	Firebase Authentication User ID
email	String	User's email
name	String	User's name
plaid_access_token	String	Plaid API access token
notifications	Boolean	Default for receiving notifications
dashboard_layout	String	User's preferred dashboard layout
theme_preference	String	User's preferred theme (e.g., light/dark)

2. **Transactions Collection** (Subcollection under Users)
This collection stores user-specific financial transactions.

Field	**Data Type**	**Description**
transaction_id	String	Unique transaction ID (from Plaid)
description	String	Description of the transaction
amount	Number	Transaction amount
category	String	Automatically categorized (e.g., groceries)
custom_category	String	User-defined category (if applicable)

date	Timestamp	Date of the transaction
merchant_name	String	Merchant details (if available from Plaid)
transaction_type	String	Transaction type (credit, debit)

3. **Subscriptions Collection** (Subcollection under Users)
   This collection tracks subscription details for each user.

Field	**Data Type**	**Description**
subscription_id	String	Unique subscription ID
service_name	String	Name of the subscription service
amount	Number	Recurring payment amount
renewal_date	Timestamp	Date of next renewal
frequency	String	Renewal frequency (monthly, yearly)
status	String	Subscription status (active, canceled)

4. **Spending Limits Collection (Subcollection under Users)**
   This collection tracks the user's spending limits and custom categories.

Field	**Data Type**	Description
limit_id	String	Unique limit ID

monthly_limit	Number	Monthly spending limit
notifications	Boolean	Whether notifications are enabled
custom_categories	Array	Array of user-defined categories with limits

5. **Notifications Collection (Subcollection under Users)**
   This collection manages alerts related to overspending, bill reminders, and subscriptions.

Field	Data Type	Description
notification_id	String	Unique notification ID
message	String	Notification content (e.g., bill reminder)
notification_type	String	Type (overspending, subscription renewal, bill reminder)
created_at	Timestamp	Timestamp when the notification was created
read_status	Boolean	Whether the notification has been read

6. **Insights Collection (Subcollection under Users)**
   This collection stores AI-generated recommendations for users based on their financial behavior.

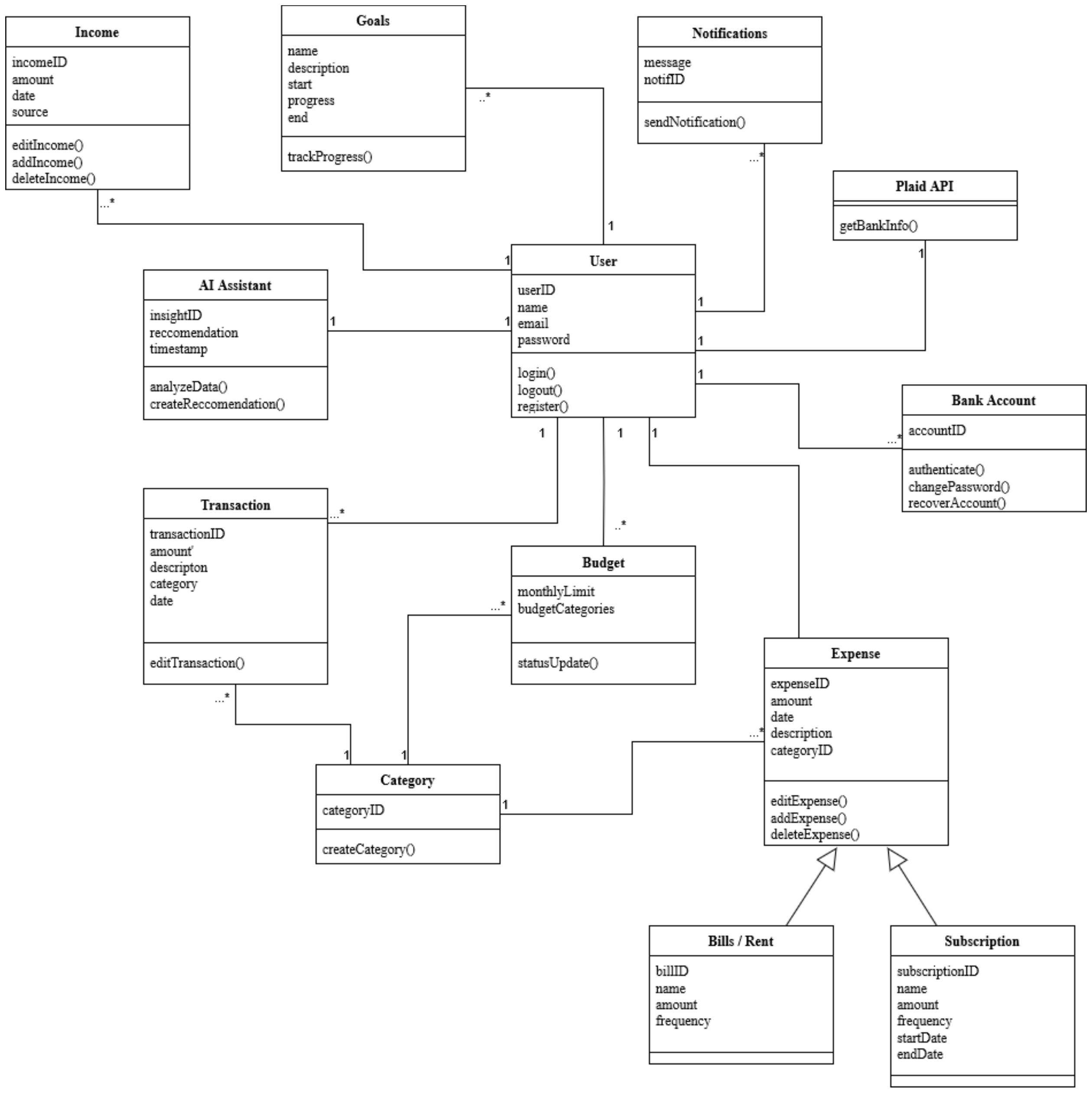
Field	Data Type	Description
insight_id	String	Unique insight ID

recommendation	String	AI-generated recommendation
created_at	Timestamp	Timestamp when the recommendation was created

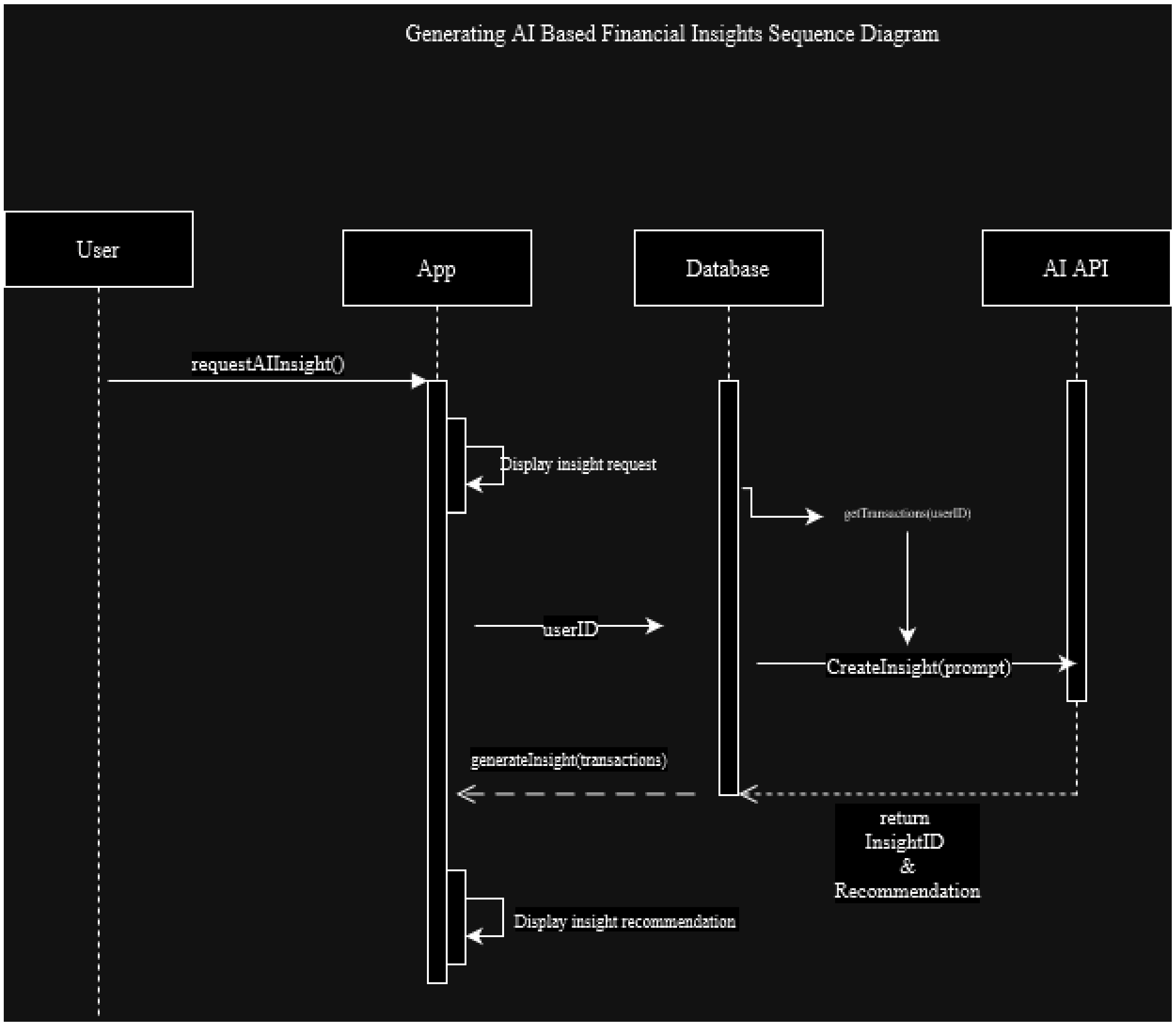
7. **Challenges Collection (Subcollection under Users)**
This collection tracks gamified financial challenges users can participate in.

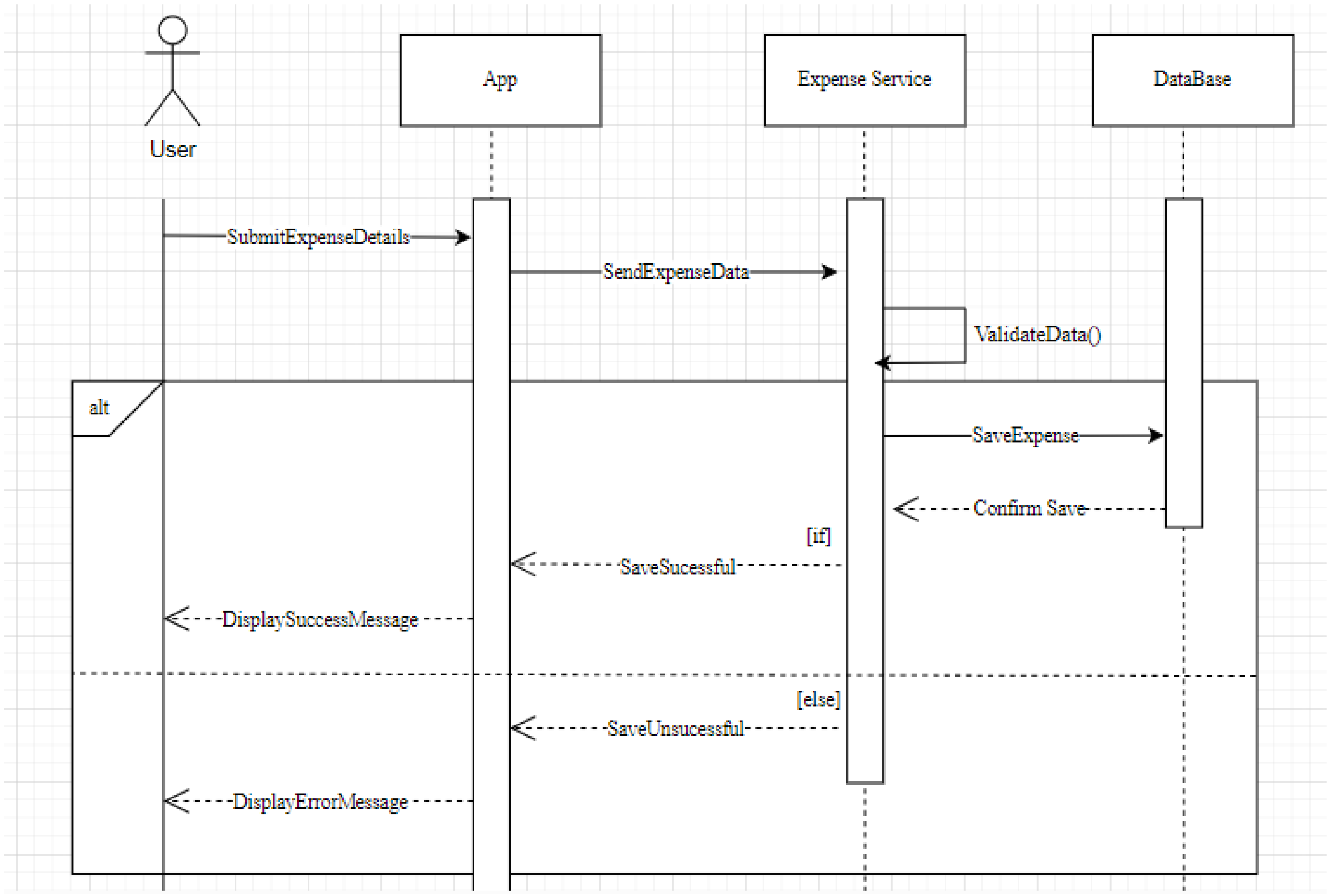
Field	**Data Type**	**Description**
challenge_id	String	Unique challenge ID
name	String	Name of the challenge
description	String	Challenge details
start_date	Timestamp	Start date of the challenge
end_date	Timestamp	End date of the challenge
progress	Number	User's progress toward completing the challenge

Section 7: Class Diagram



Section 8: Behavioral Modeling


Generating AI Based Financial Insights Sequence Diagram
User
App
Database
AI API
requestAIInsight()
Display insight request
getTransactions(userID)
userID
CreateInsight(prompt)
generateInsight(transactions)
return InsightID & Recommendation
Display insight recommendation

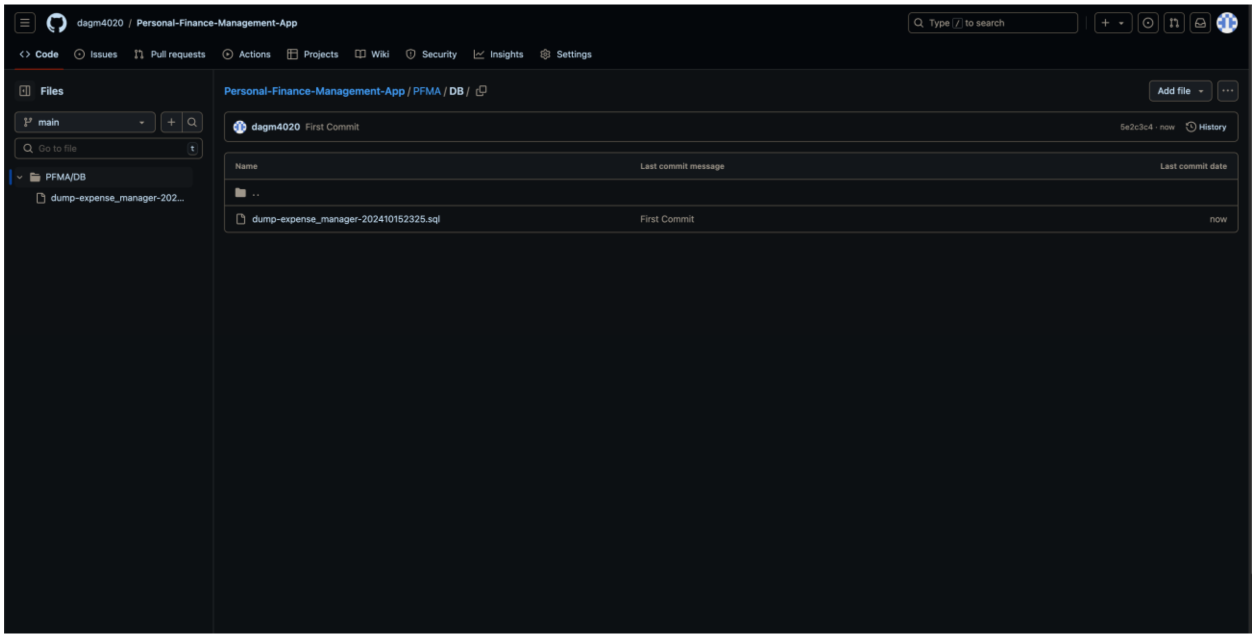


Section 9: Implementation

GitHub link:

https://github.com/dagm4020/Personal-Finance-Management-App/tree/main/PFMA/DB

Screenshot:



Personal-Finance-Management-App / PFMA / DB /

dagm4020 First Commit 5e2c3c4 · now

Name	Last commit message	Last commit date
..		
dump-expense_manager-202410152325.sql	First Commit	now

Section 10: Testing

10.1 Test cases

Test Case 1

Test Case ID: TC1.1

Description: Adding an expense with a valid name and description.

Test Inputs: Expense Name: “Groceries,” Amount: 50, Category: “Food”

Expected Results: Expense entry is saved and displayed in expense history.

Dependencies: None.

Initialization: User logged in.

Test Steps:

Go to “Add Expense”.

Enter the expense name “Groceries.”

Enter amount 50.

Select the category “Food.”

Save expense.

Post-conditions: Expenses are visible in history.

Test Case ID: TC1.2

Description: Attempt to add expense with an invalid amount.

Test Inputs: Expense Name: “Groceries,” Amount: -50, Category: “Food”

Expected Results: Error message: “Invalid amount.”

Dependencies: None.

Initialization: User logged in.

Test Steps:

Go to “Add Expense”.

Enter the expense name “Groceries.”

Enter amount -50.

Select the category “Food.”

Save expense.

Post-conditions: An error is shown, and the expense is not saved.

Test Case 2

Test Case ID: TC2.1

Description: Delete an expense entry.

Test Inputs: Select “Groceries” from the expense list.

Expected Results: The expense “Groceries” is removed from the list

Initialization: Expense “Groceries” exists in the list.

Test Steps:

Go to the “Expenses” section.

Select “Delete Expense”.

Choose “Groceries”.

Confirm deletion.

Post-conditions: The expense list no longer displays “Groceries”.

Test Case 3

Test Case ID: TC3.1

Description: Set a recurring notification for a subscription payment.

Test Inputs: Expense type: “Subscription”, Due Date: “MM/DD/YYYY”.

Expected Results: Notification is saved and set to recur on the specified date.

Dependencies: Valid date input.

Initialization: The user is logged in

Test Steps:

Go to the “Add Subscription Notification” section.

Select “Subscription” type.

Enter “MM/DD/YYYY” as Due Date.

Set recurrence as monthly.

Save.

Post-conditions: Notification is set to recur monthly on a specified date.

Test Case 4

Test Case ID: TC4.1

Description: Successful login with valid credentials.

Test Inputs: Email: “testuser@example.com”, Password: “ValidPass123”

Expected Results: The user logs in successfully, and the app displays the user’s dashboard.

Dependencies: Valid user credentials in the system.

Initialization: The app is on the login screen.

Test Steps:

Enter “testuser@example.com” as Email.

Enter “ValidPass123” as a Password.

Click “Login”.

Post-conditions: The user is directed to the dashboard.

Test Case ID: TC4.2

Description: Attempt login with incorrect password.

Test Inputs: Email: “testuser@example.com”, Password: “WrongPass”

Expected Results: The system denies login with an error message for incorrect credentials.

Dependencies: Valid email in the system.

Initialization: App at the login screen.

Test Steps:

Enter “testuser@example.com” as Email.

Enter “WrongPass” as Password.

Click “Login”.

Post-conditions: Login fails with an error message.

Test case 5

Test Case ID: TC5.1

Description: Add a valid income entry.

Test Inputs: Income source: “Salary”, Amount: 2000, Date: “MM/DD/YYYY”, Category: “Work”

Expected Results: Income entry is added to the “Work” category and is visible in income records.

Dependencies: Valid date and amount inputs.

Initialization: The user is logged in, with a set of categories defined.

Test Steps:

Go to the “Add Income” section.

Enter “Salary” as Income Source.

Enter “2000” as Amount.

Select “MM/DD/YYYY” as Date.

Choose “Work” as a Category.

Submit.

Post-conditions: Income is displayed in the “Work” category of income records.

Test Case ID: TC5.2

Description: Track an income entry with an invalid amount (zero).

Test Inputs: Income source: “Freelance Work”, Amount: 0, Date: “MM/DD/YYYY”, Category: “Side Job”

Expected Results: Error message for zero value.

Dependencies: None.

Initialization: User logged in.

Test Steps:

Go to “Track Income”.

Enter “Freelance Work” as the Source.

Enter “0” for Amount.

Select “Side Job” as a Category.

Submit.

Post-conditions: Entry is not saved; error is shown.

Test Case 6

Test Case ID: TC6.1

Description: Verify that the user can log and categorize their expenses to track spending habits and view expense history.

Test Inputs: Amount:$50 Category: Food Date: 10/30/2024

Expected Results: Shows the expense matching the descriptions

Dependencies: The user has sufficient funds in their account. The "Groceries" category is pre-existing or can be selected as a category for expenses.

Initialization: The user is logged in and has access to the "Add Expense" feature. User has a balance in their account that exceeds the expense amount.

Test Steps:

Navigate to the "Track Expenses" page.

Search based on descriptions

Submit.

Post-conditions: The user’s history of their expenses will be revealed.

Test Case 7

Test Case ID: TC7.1

Description: Verify that the user can create, edit, and delete custom categories for organizing expenses.

Test Inputs: Category Name: "Entertainment", Edit Name: Change "Entertainment" to "Leisure", Duplicate Name Attempt: "Entertainment"

Expected Results: The "Entertainment" category is successfully created. The user receives an error message if trying to create a duplicate category named "Entertainment."

Dependencies: The user is logged in and has access to category management features.

Initialization: The user is logged in with access to the "Manage Categories" feature.

Test Steps:

Navigate to the "Manage Categories" page.

Create a new category named "Entertainment."

Attempt to create a category with the name "Entertainment" again.

Either cancel or rename

Post-conditions: The category "Entertainment" is created.

Test Case 8

Test Case ID: TC8.1

Description: Verify that the user can set financial goals with target amounts and deadlines and track progress toward these goals.

Test Inputs: Goal Name: "Vacation Fund", Target Amount: $2000, Deadline: "6/14/2025", Progress Update: Adding income and expenses that affect goal-tracking

Expected Results: The "Vacation Fund" goal is created with a $2000 target and a deadline of "2025-06-01." The User can view progress toward the goal based on income and expenses. Multiple goals can be tracked simultaneously.

Dependencies: The user has income and expense tracking enabled.

Initialization: The user is logged in and has access to the "Financial Goals" feature.

Test Steps:

Navigate to the "Financial Goals" page.

Create a new goal named "Vacation Fund" with a $2000 target and a "6/14/2025" deadline.

Track the progress by adding income or expenses to see updates reflected in goal progress.

Post-conditions: Financial goals, including progress, are tracked and updated based on income and expense changes.

Test Case 9

Test Case ID: TC9.1

Description: Verify that the user can set budget limits for specific categories and receive notifications when nearing or exceeding the budget.

Test Inputs: Category: "Groceries", Budget Limit: $500, Expenses: $450 in "Groceries" category

Expected Results: The user receives a notification when $450 of the $500 limit is reached. The user can view, edit, or delete the budget.

Dependencies: Notifications enabled for budget alerts.

Initialization: The user is logged in and has set up budgets in categories.

Test Steps:

Navigate to the "Budget" section.

Set a budget limit of $500 for "Groceries."

Add expenses totaling $450 in the "Groceries" category.

Edit or delete the budget if necessary.

Post-conditions: Budget notifications are sent, and the budget can be viewed, edited, or removed.

Test Case 10

Test Case ID: TC10.1

Description: Verify that the user can set recurring notifications for upcoming bills and payments.

Test Inputs: Bill Name: "Electricity Bill", Due Date: "11/12/2024", Notification Frequency: Monthly

Expected Results: The user receives recurring notifications for "Electricity Bill" on the specified schedule. Notification settings can be customized or disabled.

Dependencies: Notifications are enabled in user settings.

Initialization: The user is logged in and has access to "Notifications" for bills.

Test Steps:

Navigate to the "Bills" or "Payments" section.

Add a new bill named "Electricity Bill" with a due date of "11/12/2024" and set notification frequency to "Monthly."

Customize or disable the notification if needed.

Post-conditions: The user receives notifications according to the specified frequency.

Test Case 11

Test Case ID: TC11.1

Description: Verify that the user can view financial reports summarizing income, expenses, budgets, and goals, with filter options for categories and dates.

Test Inputs: Filter: Date range, Display Options: Bar chart, Pie chart

Expected Results: The financial report displays income, expenses, budgets, and goal progress. The report can be filtered by date and category and displayed in selected chart formats.

Dependencies: Historical data of income and expenses.

Initialization: The user is logged in and has access to the "Reports" feature.

Test Steps:

Navigate to the "Financial Reports" section.

Set the filter to a date range

Select bar chart and pie chart display options.

View the report for accuracy and relevance.

Post-conditions: The user sees an updated, real-time financial report according to selected filters and display options.

Test Case 12

Test Case ID: TC12.1

Description: Verify that the user can export their financial data in CSV and PDF formats with selected filters.

Test Inputs: File Format: CSV/PDF, Date Filters, Category Filters

Expected Results: The user successfully downloads a CSV file with filtered data. The data in the downloaded file is accurate.

Dependencies: The user has historical data available for export.

Initialization: The user is logged in and has access to the "Export Data" feature.

Test Steps:

Navigate to the "Export Data" section.

Select CSV as the file format.

Apply the filters

Download and verify the data in the exported file.

Post-conditions: Data is successfully exported in the chosen format with applied filters.

Test Case 13

Test Case ID: TC13.1

Description: Verify that users can back up their data to the cloud and restore it.

Test Inputs: Backup and restore

Expected Results: Data is successfully backed up and can be restored on another device.

Dependencies: Internet connection for cloud backup

Initialization: The user is logged into the app and has data to back up.

Test Steps:

Navigate to the "Backup" section.

Select the manual backup option and confirm.

Log in on a different device.

Navigate to the "Restore" section.

Select the most recent backup and confirm.

Post-conditions: The user's data is restored accurately on the new device.

Test Case 14

Test Case ID: TC14.1

Description: Verify that the AI spending assistant provides financial insights based on user data.

Test Inputs: User's Income, Monthly Expenses, Financial Goals

Expected Results: AI suggests ways to optimize savings based on user data.

Dependencies: The user's data must be available in the system.

Initialization: The user has entered their income, expenses, and financial goals.

Test Steps:

Navigate to the "AI Spending Assistant" section.

Request financial insights.

Post-conditions: The user receives actionable financial advice from the AI assistant.

Test Case 15

Test Case ID: TC15.1

Description: Verify that users can securely log out of the application.

Test Inputs: The user clicks the "Logout" button.

Expected Results: The user is logged out, and session data is cleared.

Dependencies: The user must be logged in.

Initialization: The user is on any app page.

Test Steps:

Click on the "Logout" button.

Confirm logout action.

Post-conditions: The user is redirected to the login page, and session data is not accessible.

Test Case 16

Test Case ID: TC16.1

Description: Verify that users can reset their password via email verification

Test Inputs: Registered Email, New password.

Expected Results: The user successfully resets the password and can log in with new credentials.

Dependencies: Email service must be operational.

Initialization: User has forgotten their password and requests a reset.

Test Steps:

Click on the "Forgot Password" link.

Enter your registered email and submit.

Check your email for the reset link.

Click the link and enter a new password.

Confirm new password.

Post-conditions: The user can log in using the new password.

Test Case 17

Test Case ID: TC17.1

Description: Verify that users can edit details of an existing transaction.

Test Inputs: Transaction ID, New Amount, New date

Expected Results: Transaction details are updated successfully.

Dependencies: Transaction must exist in the database.

Initialization: The user is on the "Transactions" page.

Test Steps:

Select the transaction to edit.

Enter new details (amount, date, etc.).

Save changes.

Post-conditions: Updated transaction details are reflected in the user's transaction history.

Test Case 18

Test Case ID: TC18.1

Description: Verify that users can view an overview of all budgets and their current status.

Test Inputs: Click to view the page

Expected Results: All budgets, including spent amounts and remaining balances, are displayed.

Dependencies: The user must have set budgets in the system.

Initialization: The user is logged in.

Test Steps:

Navigate to the "Budget Overview" page.

Post-conditions: The user sees a comprehensive overview of budget statuses.

Test Case 19

Test Case ID: TC19.1

Description: Verify that users can modify the details of an existing financial goal.

Test Inputs: Goal ID, New target amount, New deadline

Expected Results: Financial goal details are updated successfully.

Dependencies: The goal must exist in the database.

Initialization: The user is on the "Financial Goals" page.

Test Steps:

Select the financial goal to modify.

Enter new details

Save changes.

Post-conditions: Modified goal details are reflected in the user's profile.

Test Case 20

Test Case ID: TC20.1

Description: Verify that users can delete a financial goal that is no longer relevant.

Test Inputs: Goal ID

Expected Results: The selected financial goal is removed from the system.

Dependencies: The goal must exist in the database.

Initialization: The user is on the "Financial Goals" page.

Test Steps:

Select the goal to delete.

Confirm deletion action.

Post-conditions: The goal is no longer listed in the user's financial goals.

Test Case 21

Test Case ID: TC21.1

Description: Verify that users can customize the layout and information displayed on their dashboard.

Test Inputs: Widget Preferences: Transactions, Budgets

Expected Results: The dashboard reflects user-selected layout and information.

Dependencies: The user must be logged in.

Initialization: The user is on the "Customize Dashboard" page.

Test Steps:

Select desired widgets and layout preferences.

Save changes.

Post-conditions: The dashboard is updated with the new layout upon the next login.

Test Case 22

Test Case ID: TC22.1

Description: Verify that transactions are automatically fetched and synced from linked bank accounts.

Test Inputs: Linked Bank Account Credentials

Expected Results: New transactions are updated in the user's account.

Dependencies: The user must have linked bank accounts.

Initialization: The user is logged in and has access to the bank account.

Test Steps:

Trigger sync operation manually or wait for automatic sync.

Post-conditions: New transactions appear in the user's transaction history.

Test Case 23

Test Case ID: TC23.1

Description: Verify that users can link or unlink their bank accounts

Test Inputs: Bank Account Credentials

Expected Results: The user's bank account is successfully linked or unlinked.

Dependencies: The user must have valid credentials.

Initialization: The user is on the "Bank Accounts" settings page.

Test Steps:

Enter bank account credentials and submit to the link.

Select an account to unlink and confirm.

Post-conditions: Linked accounts are displayed or removed from the user's profile as appropriate.

Test Case 24

Test Case ID: TC24.1

Description: Verify that users can access visualizations showing spending trends over time.

Test Inputs: Time Period, Categories

Expected Results: Visual representations of spending trends are displayed accurately.

Dependencies: Historical transaction data must be available.

Initialization: The user is on the "Spending Trends" page.

Test Steps:

Select the desired period and categories.

Post-conditions: The user views the correct trend visualizations.

Test Case 25

Test Case ID: TC25.1

Description: Verify that users can set and visually track savings goals with progress alerts.

Test Inputs: Target Amount, Timeline

Expected Results: A savings goal is created with a visual progress tracker, and alerts are set when nearing the goal amount.

Dependencies: The user must have financial data entered in the app.

Initialization: The user is logged in and has navigated to the "Savings Goals" section.

Test Steps:

Go to the "Savings Goals" section.

Enter the target amount and timeline for the goal.

Confirm goal creation,

Post-conditions: Savings goal is displayed with a visual tracker, and progress alerts are set.

Test Case 26

Test Case ID: TC26.1

Description: Verify that users can participate in financial challenges, set personal goals, and receive updates.

Test Inputs: Challenge, Personal Goal

Expected Results: The user can view challenge progress, set personal goals, and receive updates on progress.

Dependencies: Challenges must be available in the system.

Initialization: The user is logged in and on the "Challenges" section.

Test Steps:

Navigate to the "Challenges" section.

Select a challenge to participate in and set a personal goal.

Track progress over time and confirm notifications for updates.

Post-conditions: The user receives progress updates and challenge results in the app.

Test Case 27

Test Case ID: TC27.1

Description: Verify that users can split bills with contacts and set reminders for shared expenses.

Test Inputs: Total Transaction Amount, Contacts, Reminders

Expected Results: Expense is split among selected contacts, and reminders are set for payment collection.

Dependencies: Contacts must be added to the user's profile.

Initialization: The user is logged in and has selected a transaction to split.

Test Steps:

Enter transaction details and select contacts to split the bill with.

Set reminders for each contact.

Confirm split and reminders.

Post-conditions: A split amount is displayed for each contact, and reminders are scheduled.

Test Case 28

Test Case ID: TC28.1

Description: Verify that users can view, update, and manage subscriptions, including duplicate and low-usage alerts.

Test Inputs: Subscription, Renewal date

Expected Results: Users can view and manage subscriptions, and receive duplicate and low-usage alerts.

Dependencies: Subscription data must be entered in the app.

Initialization: The user is logged in and on the "Subscriptions" page.

Test Steps:

View list of subscriptions.

Update details for a subscription.

Confirm updates and check for alerts on duplicate or rarely used subscriptions.

Post-conditions: Updated subscription details and relevant alerts are displayed.

Test Case 29

Test Case ID: TC29.1

Description: Verify that users can access predictive budgeting based on historical data.

Test Inputs: Historical spending data, Prediction Period

Expected Results: AI generates predictive budgets with suggested allocations for the chosen period.

Dependencies: Sufficient historical data must be present for analysis.

Initialization: The user is logged in and navigates to the "Predictive Budgeting" feature.

Test Steps:

Access "Predictive Budgeting" and select a prediction period.

Confirm prediction generation.

Post-conditions: Predictive budget suggestions are displayed, with options to set them.

Test Case 30

Test Case ID: TC30.1

Description: Verify that users can convert foreign currency transactions using real-time exchange rates.

Test Inputs: Transaction Amount, Current Currency, Target Currency

Expected Results: Transaction is displayed in the selected target currency using the real-time exchange rate

Dependencies: Internet connection for real-time exchange rates.

Initialization: The user is logged in and has a foreign currency transaction recorded.

Test Steps:

Navigate to the "Transactions" page and select the foreign currency transaction.

Choose a target currency and initiate the conversion.

Post-conditions: The converted amount is displayed, and the transaction record is updated.

Test Case 31

Test Case ID: TC31.1

Description: Verify that the system detects and alerts users about potential fraudulent activities.

Test Inputs: Unusual Transaction

Expected Results: The system flags the transaction and sends an alert to the user.

Dependencies: AI monitoring algorithm and notification service must be active.

Initialization: The user has transaction data that AI can monitor.

Test Steps:

Simulate a transaction outside normal spending patterns.

Check for alert notification and response options.

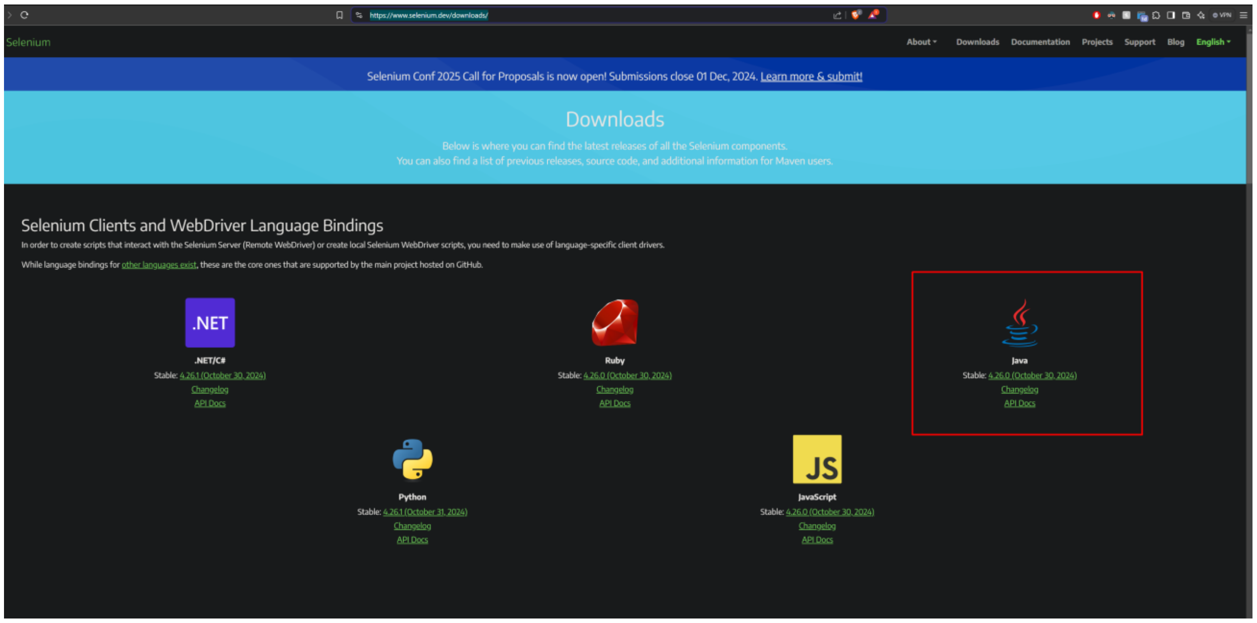
Post-conditions: The user receives an alert, and action options are available (e.g., confirm, dispute, or lock account).

10.2 Unit Testing

Framework setup and instructions on running test case:

We will be utilizing Selenium for this project since it has a GUI feature unlike HTMLUnit. In addition, it has many other benefits, but it is an ideal choice as in general as it is a more comprehensive and realistic testing environment.

- Go to Selenium and download the stable Java version


https://www.selenium.dev/downloads/

Selenium

About Downloads Documentation Projects Support Blog English

Selenium Conf 2025 Call for Proposals is now open! Submissions close 01 Dec, 2024. Learn more & submit!

Downloads

Below is where you can find the latest releases of all the Selenium components.
You can also find a list of previous releases, source code, and additional information for Maven users.

Selenium Clients and WebDriver Language Bindings

In order to create scripts that interact with the Selenium Server (Remote WebDriver) or create local Selenium WebDriver scripts, you need to make use of language-specific client drivers.

While language bindings for other languages exist, these are the core ones that are supported by the main project hosted on GitHub.

.NET/C#
Stable: 4.26.1 (October 30, 2024)
Changelog
API Docs

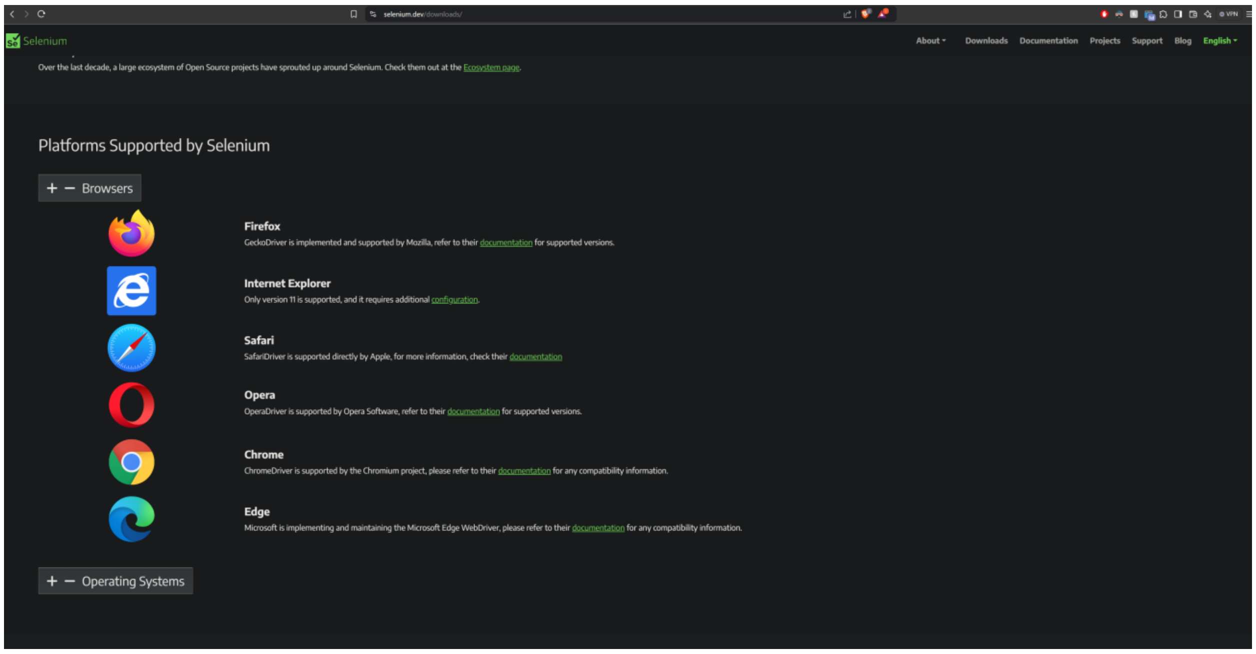
Ruby
Stable: 4.26.0 (October 30, 2024)
Changelog
API Docs

Java
Stable: 4.26.0 (October 30, 2024)
Changelog
API Docs

Python
Stable: 4.26.1 (October 31, 2024)
Changelog
API Docs

JavaScript
Stable: 4.26.0 (October 30, 2024)
Changelog
API Docs

- On the same page, scroll down to download browser driver
    - Note: you need to know what version your browser is running on to download the correct browser driver


selenium.dev/downloads/

Selenium

About Downloads Documentation Projects Support Blog English

Over the last decade, a large ecosystem of Open Source projects have sprouted up around Selenium. Check them out at the Ecosystem page.

Platforms Supported by Selenium

+ − Browsers

Firefox
GeckoDriver is implemented and supported by Mozilla, refer to their documentation for supported versions.

Internet Explorer
Only version 11 is supported, and it requires additional configuration.

Safari
SafariDriver is supported directly by Apple, for more information, check their documentation

Opera
OperaDriver is supported by Opera Software, refer to their documentation for supported versions.

Chrome
ChromeDriver is supported by the Chromium project, please refer to their documentation for any compatibility information.

Edge
Microsoft is implementing and maintaining the Microsoft Edge WebDriver, please refer to their documentation for any compatibility information.

+ − Operating Systems

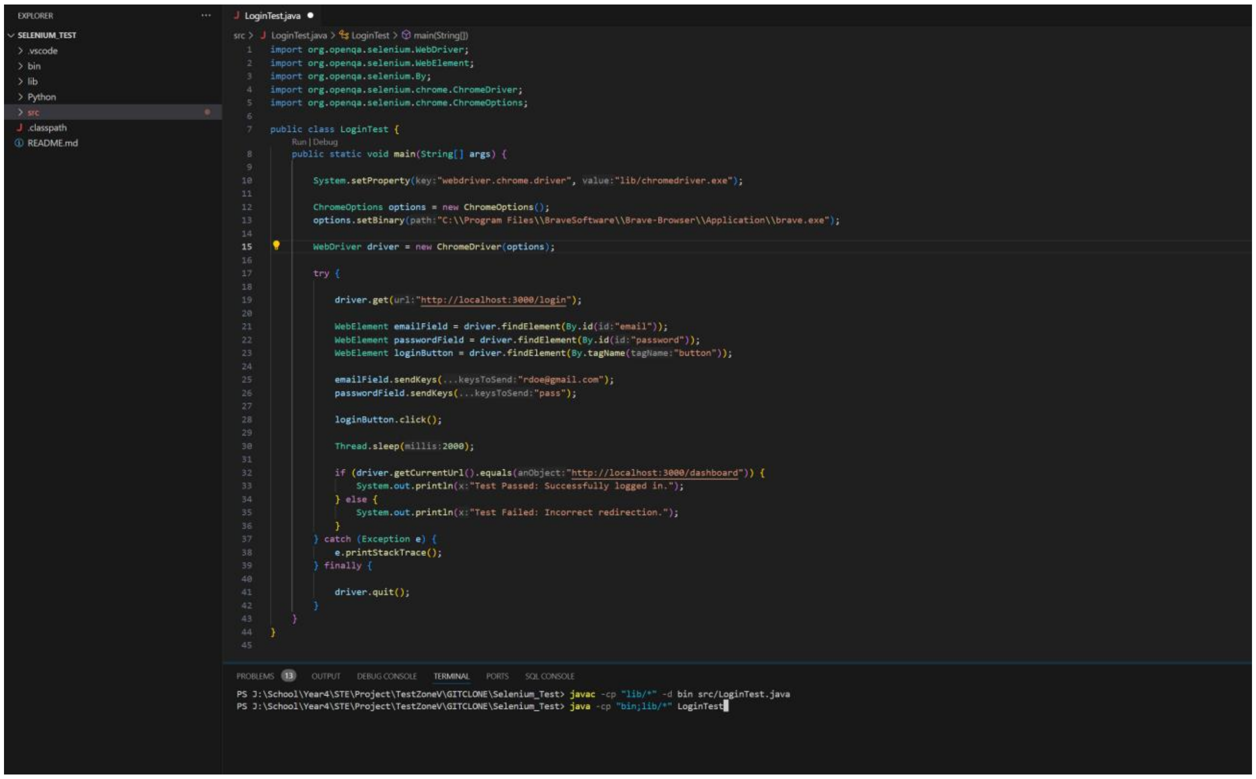
    - If the browser isn't listed, you might be using a browser that's based off of Chromium. The browser that is in use currently is Brave which uses Chromium as the engine, therefore, we can use the Chrome driver

- If you haven't already, make sure to have Java SDK downloaded and set it up
- This instruction will use VS Code for the IDE, download the extension pack for Java

  - depending on what device you use, you might need to add the path to the environment variable (for Windows) / .zshrc file (for Mac)
  - Make sure that the java and java compiler versions are compatible (this might cause issues later on, check using java –version and javac –version)
- Unzip browser driver and Selenium
- For ease, next to project folder create a java project for testing purpose
  - In VS Code (in the explorer view) right click and click “New Java Project”
  - Go through the process of choosing location, providing the folder an appropriate name, and so forth until you have the project folder setup
- A “lib” folder should be present if previous instruction was done correctly. Transfer all of the JAR files from Selenium into this folder as well as the browser driver
- To avoid any issues, instead of using a development server, testing will be done in production build.
  - To do this we will generate the build folder that contains the optimized files using “npm run build” which should create a build folder in frontend project
-
  - We will then serve the production build locally using the “npx server –s build” command. This should open a browser with the frontend running. This project uses a different port for the backend so use a different terminal to get it running using “npm start”. Verify both front and backend are functioning correctly by attempting to login.
- For this example, we will demonstrate TC0.6_Login_Test_2. When developing the test code, you might need to specify the location of the browser. You will need to provide the URL for the site, location for Selenium JAR files and browser driver. Once you have the code setup you will need to compile the code in the terminal using (make sure you are in root directory of the Java project):
  - javac –cp “lib/*” -d bin src/LoginTest.java (replace LoginTest with the name of the java file you created)
- To run the compiled code, you need to use:
  - java –cp “bin;lib/*” LoginTest (replace LoginTest with the name of the java file you created, do not add .java after the file name to run code)

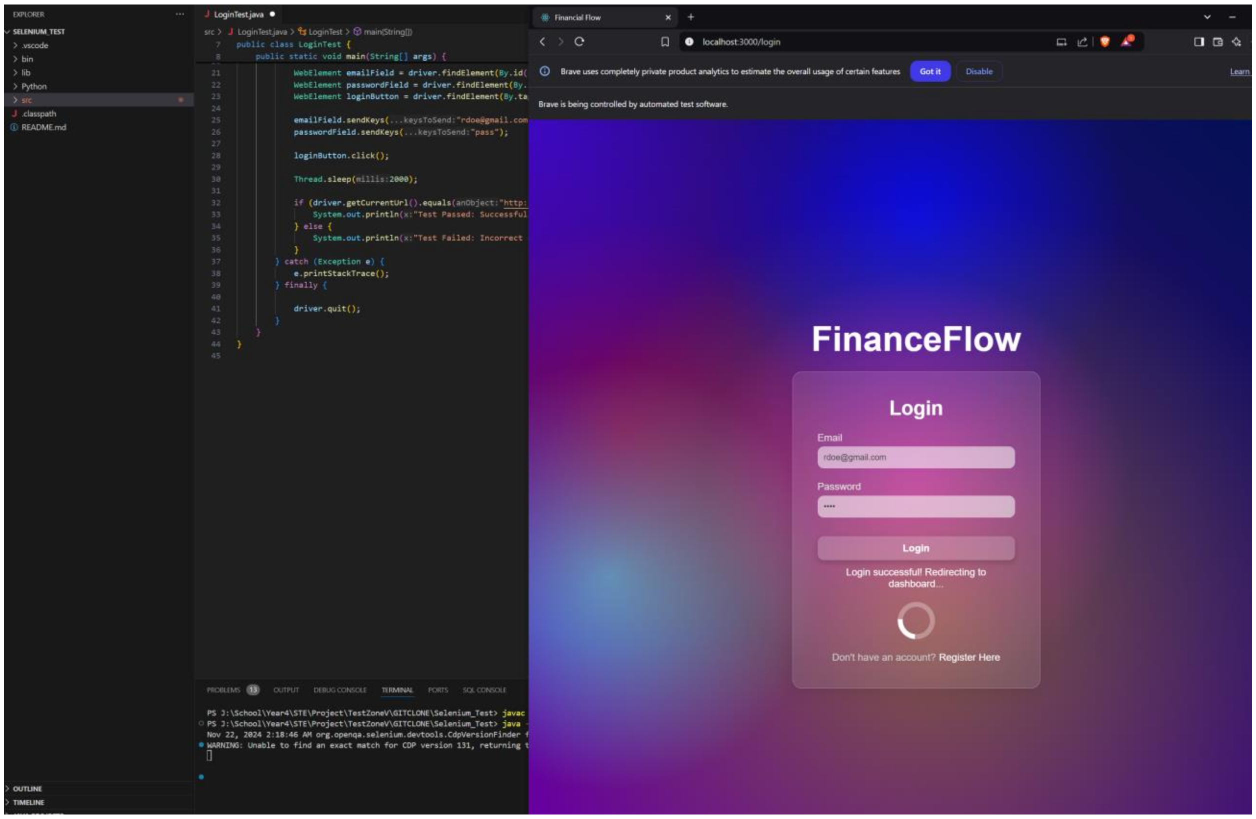
(Warning: do not copy and paste the code from above, Word might be using different characters.

Type it out manually)



```java
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.By;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;

public class LoginTest {
    public static void main(String[] args) {

        System.setProperty(key:"webdriver.chrome.driver", value:"lib/chromedriver.exe");

        ChromeOptions options = new ChromeOptions();
        options.setBinary(path:"C:\\Program Files\\BraveSoftware\\Brave-Browser\\Application\\brave.exe");

        WebDriver driver = new ChromeDriver(options);

        try {

            driver.get(url:"http://localhost:3000/login");

            WebElement emailField = driver.findElement(By.id(id:"email"));
            WebElement passwordField = driver.findElement(By.id(id:"password"));
            WebElement loginButton = driver.findElement(By.tagName(tagName:"button"));

            emailField.sendKeys(...keysToSend:"rdoe@gmail.com");
            passwordField.sendKeys(...keysToSend:"pass");

            loginButton.click();

            Thread.sleep(millis:2000);

            if (driver.getCurrentUrl().equals(anObject:"http://localhost:3000/dashboard")) {
                System.out.println(x:"Test Passed: Successfully logged in.");
            } else {
                System.out.println(x:"Test Failed: Incorrect redirection.");
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {

            driver.quit();
        }
    }
}
```

```
PS J:\School\Year4\STE\Project\TestZoneV\GITCLONE\Selenium_Test> javac -cp "lib/*" -d bin src/LoginTest.java
PS J:\School\Year4\STE\Project\TestZoneV\GITCLONE\Selenium_Test> java -cp "bin;lib/*" LoginTest
```



```
PS J:\School\Year4\STE\Project\TestZoneV\GITCLONE\Selenium_Test> javac
PS J:\School\Year4\STE\Project\TestZoneV\GITCLONE\Selenium_Test> java -
Nov 22, 2024 2:18:46 AM org.openqa.selenium.devtools.CdpVersionFinder f
WARNING: Unable to find an exact match for CDP version 131, returning t
```

Brave uses completely private product analytics to estimate the overall usage of certain features

Brave is being controlled by automated test software.

FinanceFlow

Login

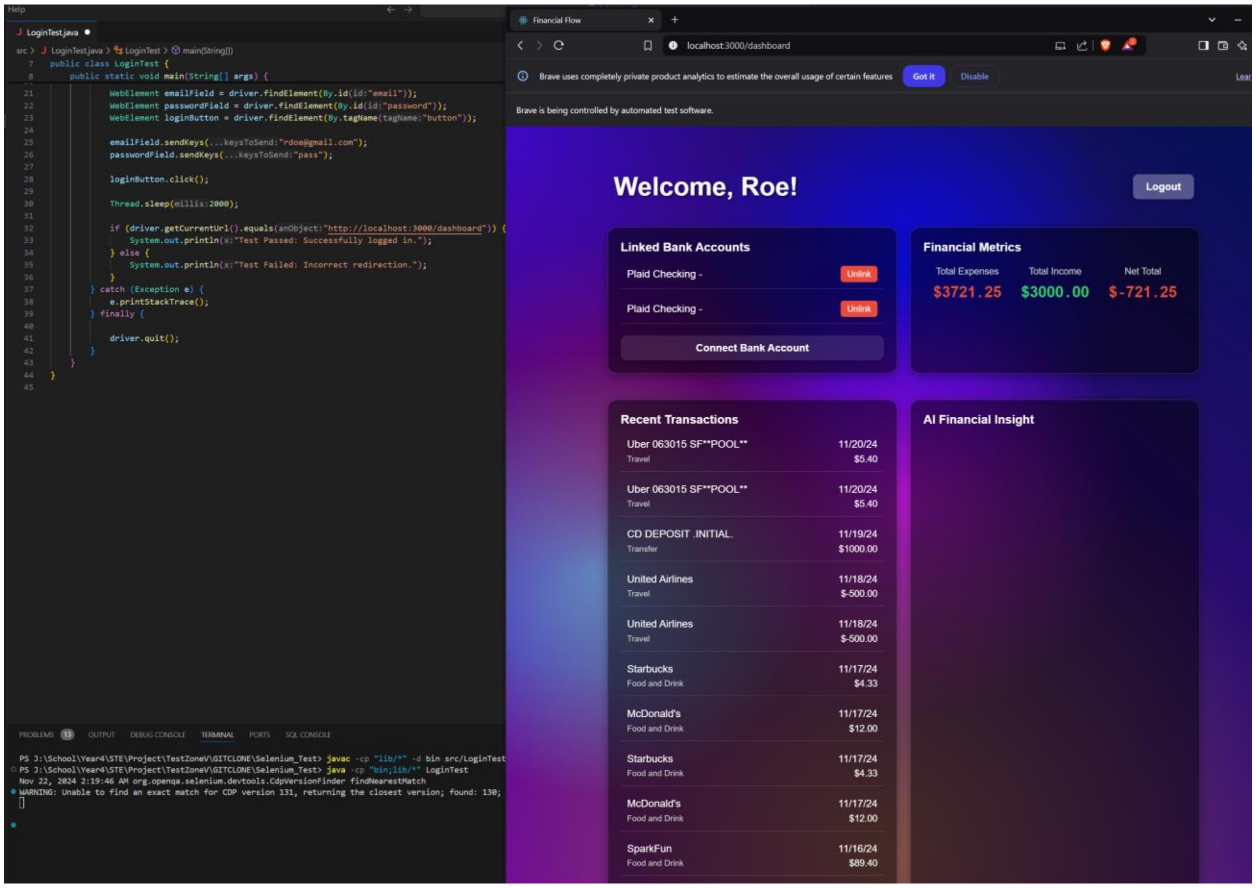
Email: rdoe@gmail.com

Password: ••••

Login

Login successful! Redirecting to dashboard...

Don't have an account? Register Here



LoginTest.java

src > LoginTest.java > LoginTest > main(String[])

```java
public class LoginTest {
    public static void main(String[] args) {
            WebElement emailField = driver.findElement(By.id(id:"email"));
            WebElement passwordField = driver.findElement(By.id(id:"password"));
            WebElement loginButton = driver.findElement(By.tagName(tagName:"button"));

            emailField.sendKeys(...keysToSend:"rdoe@gmail.com");
            passwordField.sendKeys(...keysToSend:"pass");

            loginButton.click();

            Thread.sleep(millis:2000);

            if (driver.getCurrentUrl().equals(anObject:"http://localhost:3000/dashboard")) {
                System.out.println(x:"Test Passed: Successfully logged in.");
            } else {
                System.out.println(x:"Test Failed: Incorrect redirection.");
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {

            driver.quit();
        }
    }
}
```

```
PS J:\School\Year4\STE\Project\TestZoneV\GITCLONE\Selenium_Test> javac -cp "lib/*" -d bin src/LoginTest
PS J:\School\Year4\STE\Project\TestZoneV\GITCLONE\Selenium_Test> java -cp "bin;lib/*" LoginTest
Nov 22, 2024 2:19:46 AM org.openqa.selenium.devtools.CdpVersionFinder findNearestMatch
WARNING: Unable to find an exact match for CDP version 131, returning the closest version; found: 130;
```

Financial Flow

localhost:3000/dashboard

Brave uses completely private product analytics to estimate the overall usage of certain features

Brave is being controlled by automated test software.

Welcome, Roe!

Logout

Linked Bank Accounts

Plaid Checking - Unlink

Plaid Checking - Unlink

Connect Bank Account

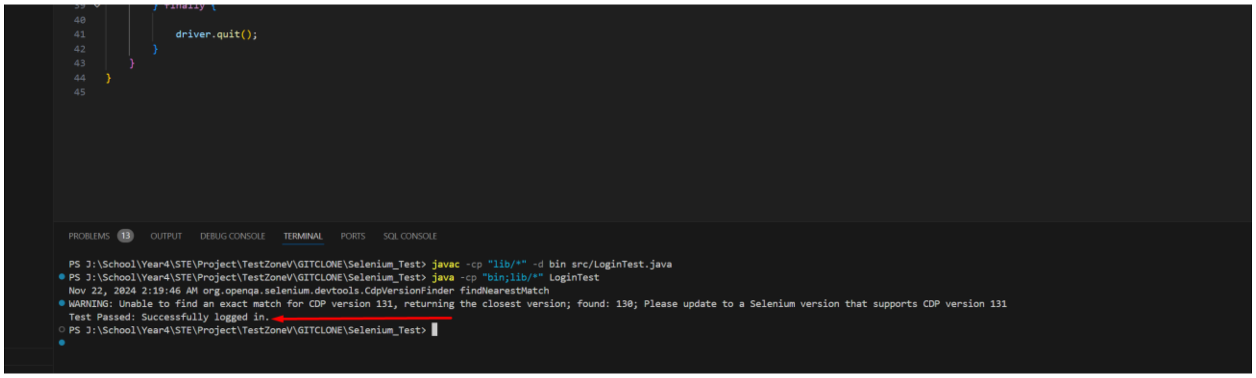
Financial Metrics

Total Expenses	Total Income	Net Total
$3721.25	$3000.00	$-721.25

Recent Transactions

Uber 063015 SF**POOL** Travel	11/20/24 $5.40
Uber 063015 SF**POOL** Travel	11/20/24 $5.40
CD DEPOSIT .INITIAL. Transfer	11/19/24 $1000.00
United Airlines Travel	11/18/24 $-500.00
United Airlines Travel	11/18/24 $-500.00
Starbucks Food and Drink	11/17/24 $4.33
McDonald's Food and Drink	11/17/24 $12.00
Starbucks Food and Drink	11/17/24 $4.33
McDonald's Food and Drink	11/17/24 $12.00
SparkFun Food and Drink	11/16/24 $89.40

AI Financial Insight



```java
            driver.quit();
        }
    }
}
```

```
PS J:\School\Year4\STE\Project\TestZoneV\GITCLONE\Selenium_Test> javac -cp "lib/*" -d bin src/LoginTest.java
PS J:\School\Year4\STE\Project\TestZoneV\GITCLONE\Selenium_Test> java -cp "bin;lib/*" LoginTest
Nov 22, 2024 2:19:46 AM org.openqa.selenium.devtools.CdpVersionFinder findNearestMatch
WARNING: Unable to find an exact match for CDP version 131, returning the closest version; found: 130; Please update to a Selenium version that supports CDP version 131
Test Passed: Successfully logged in.
PS J:\School\Year4\STE\Project\TestZoneV\GITCLONE\Selenium_Test>
```

- The tab automatically closes right after logging in and landing on the dashboard. Result will be printed in the terminal as well. Create new java files, compile, and run them for each test cases.

Login Screenshot

Test ID	TC0.3_Login_Test_1
Purpose of Test	To ensure the login module of the application works as intended to provide security for the user's sensitive information.
Test Environment	The test environment is the login screen of the FinanceFlow web application.
Test Steps	1. The unit test opens the web application to the login screen. 2. The program then inputs placeholder credentials (credentials for an account that doesn't exist) into the respective fields. 3. The program then tries to log in with these credentials.
Test Input	Username and password
Expected Result	The hypothetical user would be denied access and be told to try again.
Likely Problems/Bugs Revealed	N/A

Test ID	TC0.6_Login_Test_2
Purpose of Test	To ensure the login module of the application works as intended to provide security for the user's sensitive information.
Test Environment	The test environment is the login screen of the FinanceFlow web application.
Test Steps	1. The unit test opens the web application to the login screen. 2. The program then inputs the correct credentials 3. The program then tries to log in with these credentials.
Test Input	Username and Password

Expected Result	The user should be granted access to the web application and should see the user dashboard.
Likely Problems/Bugs Revealed	N/A

Test ID	TC0.9_ Setting_Goals_Test
Purpose of Test	To ensure that users can successfully create, manage, and achieve their financial objectives
Test Environment	Scenarios of realistic saving goals and invalid entries done on a Windows 11
Test Steps	1. Click “Add New Goal” 2. Input valid data 3. Save the goal
Test Input	Goal name, target amount, and target date
Expected Result	The new goal is successfully added and displayed in the goals list with correct details
Likely Problems/Bugs Revealed	Duplicated goal name or missing notifications for approaching deadlines

Test ID	TC1.1
Purpose of Test	Validate that the user can add an expense with valid inputs, ensure the expense is categorized correctly, and ensure the budget updates as required.
Test Environment	Dashboard, where Add expenses, are The platform is windows 11
Test Steps	1. Open dashboard 2. Navigate to the “add expense” tab 3. Input valid values a. Expense name: Groceries b. Value: 100 c. Category: Food 4. Submit

Test Input	a. Expense name: Groceries b. Value: 100 c. Category: Food
Expected results	The expense is added as Groceries labeled with the food category
Likely problems/bugs revealed	Expense not added to the correct category.

Test ID	TC1.2
Purpose of Test	Validate that invalid inputs are rejected.
Test Environment	Dashboard, where Add expenses, are The platform is windows 11
Test Steps	1. Open dashboard 2. Navigate to the “add expense” tab 3. Input valid values a. Expense name: (empty string) b. Value: -100 c. Category: (empty string) 4. Submit
Test Input	a. Expense name: (empty string) b. Value: -100 c. Category: (empty string)
Expected results	Error for invalid input Expense is not added

Likely problems/bugs revealed	The system allows the invalid inputs No error message given

Test ID	TC2.1
Purpose of Test	Validate if the user can delete a single expense.
Test Environment	Dashboard, where expenses, are The platform is windows 11
Test Steps	1. Open dashboard 2. Navigate to the “Expenses” tab 3. Select the expense “Grocercies: $100” 4. Click the delete/trashcan icon button
Test Input	Selected Expense “Groceries: $100”
Expected results	The expense is removed from the list
Likely problems/bugs revealed	Expense not deleted

Bug Table:

Bug ID	Test uncovered	Description	Action taken
Bug-001	TC1.1	Expense not added to the correct category.	Debugged and fixed category assignment logic.

Bug-002	TC1.2	The system allowed a blank expense name.	Added validation checks for blank fields.
Bug-003	TC2.1	The budget needs to be updated after expense deletion.	Updated logic to recalculate the budget.

AI Insight testing:

Test ID	TC_AI_1
Purpose of test	Verify AI insights generation with valid user data as per Requirement
Test Environment	Windows 11, Selenium WebDriver, Chrome Browser, Dashboard where AI insight is located
Test steps	1. Open browser and navigate to login page. 2. Enter valid credentials and log in. 3. Navigate to AI Insights page. 4. Enter userID 123 and submit. 5. Verify displayed insight and status.
Test input	Login: Username: user1 Password: pass123 AI Insights Request: userID: 123
Expected results	Result: Insight: "You can save $200 this month by reducing dining expenses by 15%." Status: "success"
Likely Problems/bugs revealed	Insight generation fails or displays incorrect information.

Test ID	TC_AI_2
Purpose of test	Verify handling of requests for users with no financial data as per Requirement
Test Environment	Windows 11, Selenium WebDriver, Chrome Browser, Dashboard where AI insight is located
Test steps	1. Open browser and navigate to login page. 2. Enter valid credentials and log in. 3. Navigate to AI Insights page. 4. Enter `userID` 999 and submit. 5. Verify displayed error message.
Test input	**Login:**

	Username: `user1` Password: `pass123` **AI Insights Request:** `userID`: 999
Expected results	**Result:** Error: "No financial data found for this user."
Likely Problems/bugs revealed	System does not handle absence of data gracefully.

Test ID	TC_AI_3
Purpose of test	Verify response to invalid or expired JWT tokens as per Requirement
Test Environment	Windows 11, Selenium WebDriver, Chrome Browser, Dashboard where AI insight is located
Test steps	1. Open browser and navigate to login page. 2. Attempt to log in with invalid/expired JWT token (simulate by manipulating cookies/local storage). 3. Attempt to access AI Insights page. 4. Verify unauthorized access message.
Test input	**Login:** Invalid/Expired JWT Token **AI Insights Request:** `userID`: 123
Expected results	**Result:** Error: "Unauthorized access."
Likely Problems/bugs revealed	Unauthorized access is not properly restricted.

Test ID	TC_AI_4
Purpose of test	Verify response when JWT token is missing as per Requirement
Test Environment	Windows 11, Selenium WebDriver, Chrome Browser, Dashboard where AI insight is located
Test steps	1. Open browser and navigate to AI Insights page without logging in. 2. Attempt to submit AI Insights request. 3. Verify authentication error message.
Test input	**AI Insights Request:**

	userID: 123
Expected results	**Result:** Error: "Authentication token is missing."
Likely Problems/bugs revealed	System allows access without authentication.

Test ID	TC_AI_5
Purpose of test	Verify handling of non-existent user IDs as per Requirement
Test Environment	Windows 11, Selenium WebDriver, Chrome Browser, Dashboard where AI insight is located
Test steps	1. Open browser and navigate to login page. 2. Enter valid credentials and log in. 3. Navigate to AI Insights page. 4. Enter non-existent userID 456 and submit. 5. Verify displayed error message.
Test input	**Login:** Username: user1 Password: pass123 **AI Insights Request:** userID: 456
Expected results	**Result:** Error: "User not found."
Likely Problems/bugs revealed	System does not properly identify non-existent users

Test ID	TC_AI_6
Purpose of test	Verify AI insights generation with large financial data as per Requirement
Test Environment	Windows 11, Selenium WebDriver, Chrome Browser, Dashboard where AI insight is located
Test steps	1. Open browser and navigate to login page. 2. Enter valid credentials and log in. 3. Navigate to AI Insights page. 4. Enter userID 789 and submit. 5. Verify displayed insight and status.
Test input	**Login:** Username: user1 Password: pass123 **AI Insights Request:** userID: 789
Expected results	**Result:**

	Insight: "Your average monthly spending on dining is $500. Consider using a dining rewards program to save $50 per month." Status: "success"
Likely Problems/bugs revealed	System fails or slows down with large data sets.

Test ID	TC_AI_7
Purpose of test	Verify system response during server errors as per Requirement
Test Environment	Windows 11, Selenium WebDriver, Chrome Browser, Dashboard where AI insight is located
Test steps	1. Simulate server failure (e.g., stop backend service). 2. Open browser and navigate to AI Insights page. 3. Enter `userID` 123 and submit. 4. Verify displayed error message.
Test input	**AI Insights Request:** `userID`: 123
Expected results	**Result:** Error: "Internal server error. Please try again later."
Likely Problems/bugs revealed	System does not handle server errors gracefully.

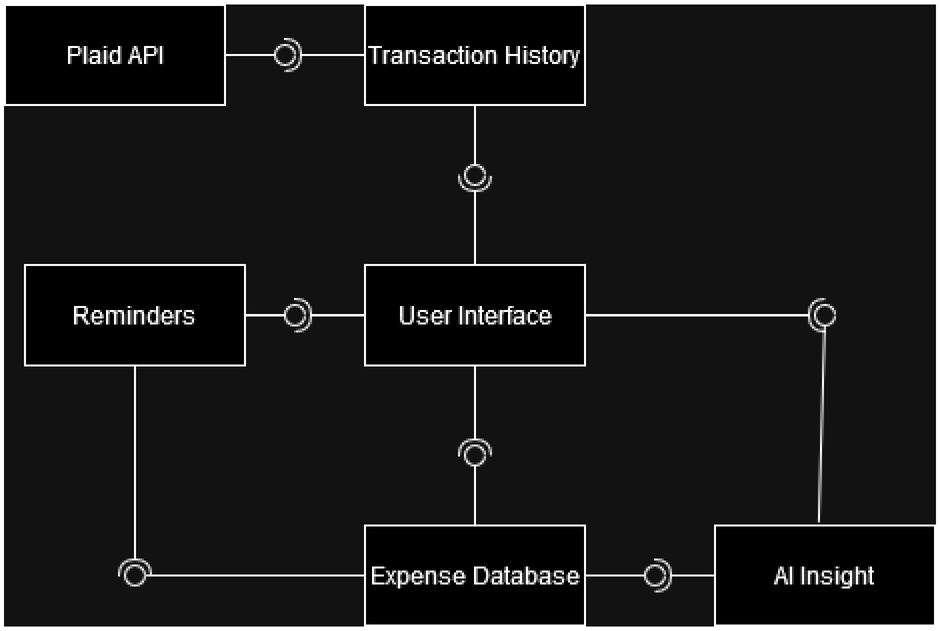
Bug Table:

Bug ID	Test uncovered	Description	Action taken
Bug-AI-001	TC_AI_2	API returns `200 OK` instead of `404 Not Found` when user has no financial data.	Fixed the service layer to check for data existence and return the appropriate HTTP status. Updated the controller to handle the `null` response correctly.

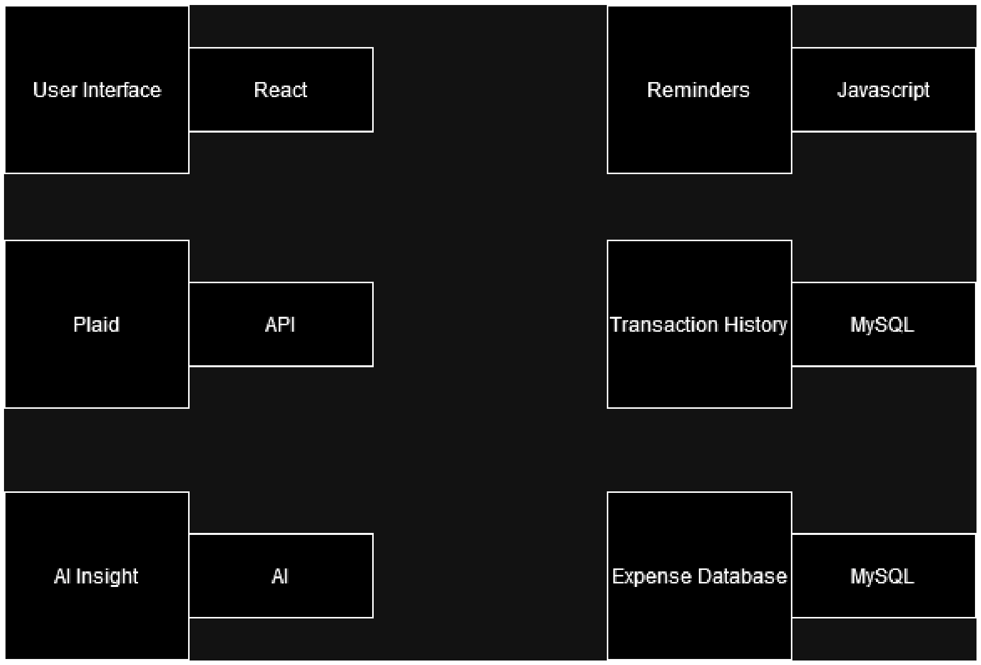
Bug-AI-002	TC_AI_7	AI Insights page does not display proper error message during server failure; instead, it shows a generic error or crashes.	Implemented global exception handling on the frontend to capture errors and display user-friendly messages. Ensured the backend sends structured error responses.

Section 11: Architectural Modeling

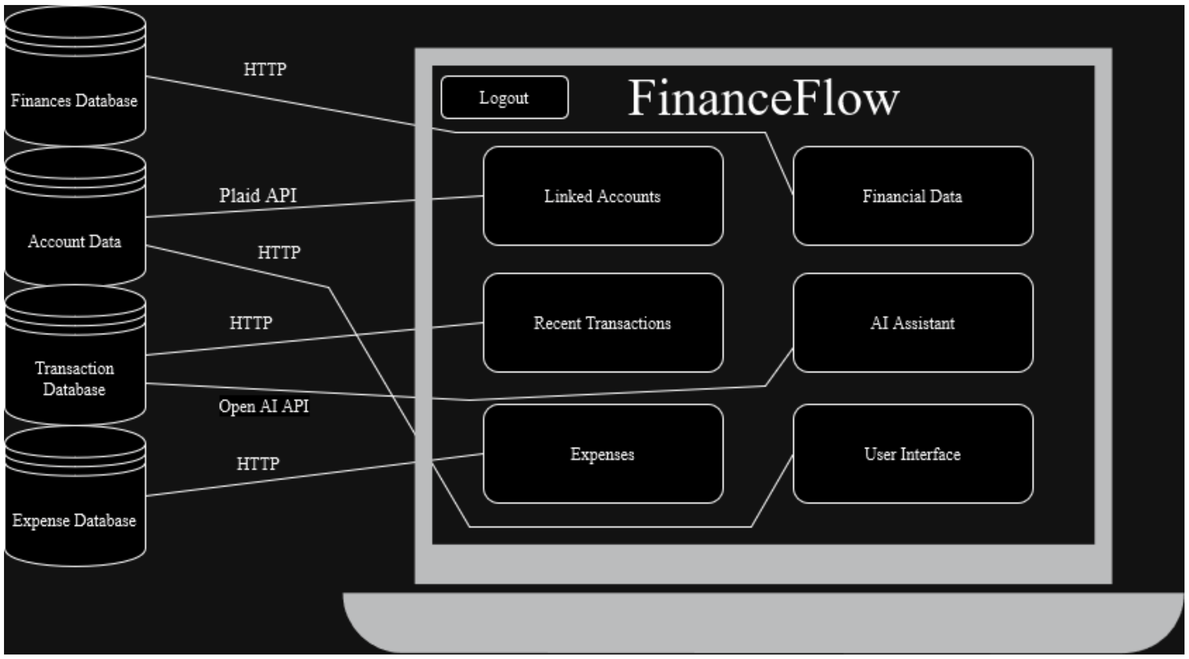
Logical View



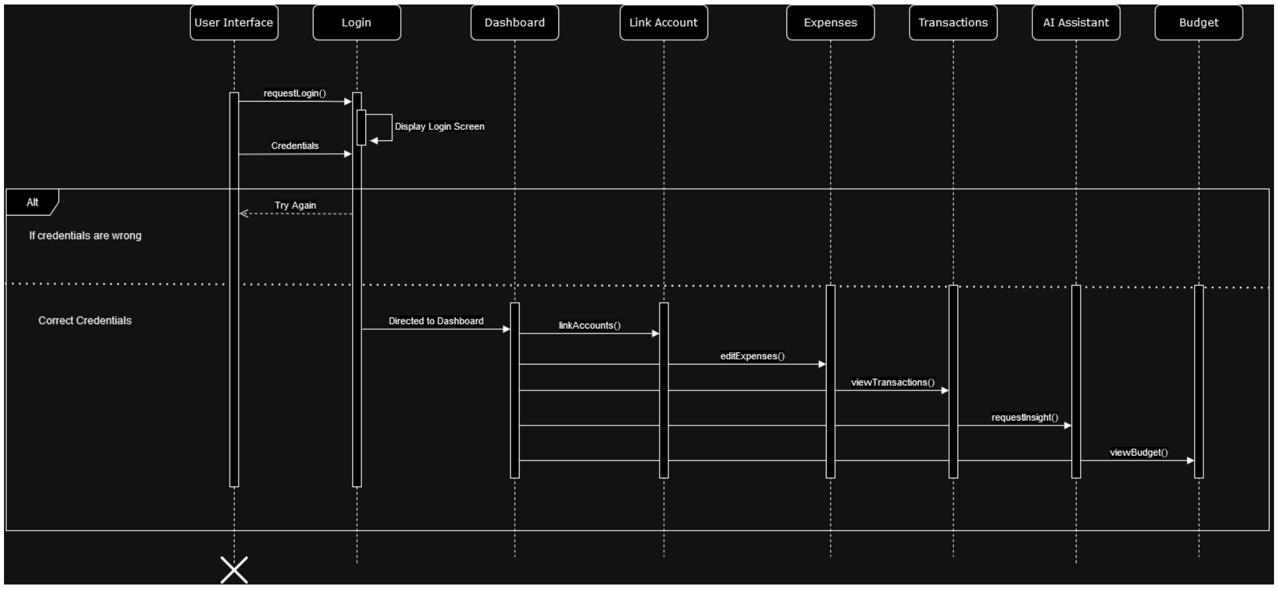
Development View:



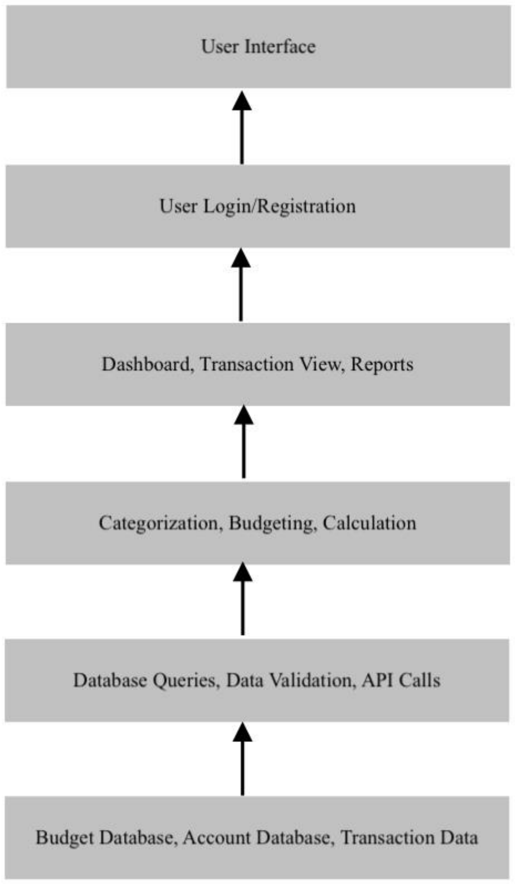
Physical View:



Process View:



Architectural Layered Model


User Interface
User Login/Registration
Dashboard, Transaction View, Reports
Categorization, Budgeting, Calculation
Database Queries, Data Validation, API Calls
Budget Database, Account Database, Transaction Data

References

- https://www.youtube.com/watch?app=desktop&v=0s02i6HQ0KQ
- https://www.youtube.com/watch?v=Rvjcolmg4d4