Team Information

America Pacheco: Front-End Developer (HTML, CSS, UI design)
Bailey Bounnam: Back-End Developer (Node.js, Express, API integration)
Ross Henderson: Team Lead and GitHub Manager, Back-End Developer (coordinate tasks, oversee GitHub repository, as well as assisting with back-end integration)
Brian Fang: Database Manager (MySQL design, secure storage of financial data)
Bryan Partida: API Specialist (Plaid API integration, financial data processing)

Github Repository: https://github.com/henderos/CS362-Class-Project

Communication Methods

Our main form of communication will be Discord, where we will establish clear threads for tasks, announcements, and questions. We will try to respond within 17 hours of a message of discord. If no one responds, then politely give a reminder.

Product description

Personal Finance Analyzer

Abstract:

The Personal Finance Analyzer is a web application designed to simplify financial management for users, providing tools to track spending and manage subscriptions. With an intuitive user interface and analytics, it aims to empower users to make informed financial decisions effortlessly.

Goal:

This project will help users to manage and keep track of their personal finances, see where their money is going, and make decisions for future spending based on these facts.

Current practice:

Today, people commonly use spreadsheets, individual banking apps, or third-party apps to track their finances. These methods are often not automated, or if you have accounts in different

banks, then you have to individually log into each banks' account. With our web app, you will be able to link every account so that you can see everything in one spot with advanced features such as subscription tracking or personalized savings advice.

Novelty:

Our finance manager will not only allow users to see their finances, but also will tell users where certain "problem areas" may be, allowing them to see where they are likely able to improve.

Effects:

The people who will be using our web app are people who want to see all their financial information with a user-friendly interface. If we are successful in our app, our app will show more transparency with how our users' money is being handled. This will help reduce unnecessary expenses by identifying underused subscriptions and help users allocate their income effectively by highlighting spending patterns.

Technical approach:

Our project will employ a scalable and efficient system architecture. The front end will be built using HTML and CSS, with JavaScript to enhance interactivity and responsiveness. For the back end, we will use Node.js with Express to handle server-side functionality and API routing. Data storage will be managed using MySQL to securely handle user information, while the Plaid API will enable integration with users' bank accounts for retrieving financial data.

Risks:

One of the most significant challenges we face is ensuring effective and efficient communication within a large team. Miscommunication or delays in responding to inquiries could lead to misunderstandings, duplicated efforts, or even missed deadlines in some cases. Additionally, with multiple members handling interconnected tasks, it can be challenging to maintain a clear and cohesive understanding of project progress and priorities.

To mitigate this risk, we will establish clear communication expectations from the beginning. As mentioned above, team members will be required to regularly check Discord, our primary communication platform, and respond to messages within 17 hours. Tasks will be clearly assigned and tracked using GitHub Projects to provide transparency and accountability. Regular updates will be shared in designated channels to keep everyone informed of ongoing progress.

Major features we will implement:

- **Dashboard Overview:** Display consolidated financial data, including recent transactions, and monthly spending summaries.

- **Subscription Tracking**: Identify and track recurring subscriptions, flagging underused services.
- **Spending Categories:** Provide visual breakdowns of expenses by category (e.g., groceries, rent, entertainment).
- **Budgeting Tool:** Allow users to set budgets for categories and receive alerts when nearing limits.

Stretch goals we hope to implement:

- **Implementing AI suggestions and advice for users.**
- **Mobile support.**

1. Use Cases (Functional Requirements)

Use Case 1: Subscription Tracking

Actors:

- User
- System

Triggers:

- User navigates to the “Subscriptions” section of the app.

Preconditions:

- User has linked financial accounts.
- Transaction data is available.

Postconditions:

- The system identifies recurring transactions and displays them as subscriptions.

List of Steps:

- User selects the “Subscriptions” tab.
- System retrieves recurring transactions from the linked accounts.
- System categorizes these transactions and flags underused subscriptions.
- User reviews and optionally adjusts subscription tracking preferences.

Extensions/Variations:

- User can manually add subscriptions.
- User can flag a subscription as “not recurring.”

Exceptions:

- Errors in identifying recurring transactions.
- Incomplete transaction data.

Use Case 2: Budget Setting

Actors:

- User
- System

Triggers:

- User navigates to the “Budgeting Tool” section of the app.

Preconditions:

- User has linked financial accounts.
- User has access to spending categories.

Postconditions:

- The system stores the user's budget for each selected category

List of Steps:

- User selects "Budgeting Tool"
- User selects a spending category
- User enters budget amount

Extensions/Variations:

- If user is editing an existing budget

Exceptions:

- If the budget amount is too small/large

Use Case 3: Viewing Financial Data

Actors:

- User
- System

Triggers:

- User logs into the app

Preconditions:

- User has linked financial accounts.

Postconditions:

- The user sees an overview of recent transactions, monthly spending, and account balances

List of Steps:

- User logs in
- System retrieves financial data
- Dashboard shows financial information

Extensions/Variations:

- User can change between light and dark mode

Exceptions:

- If the system can't connect to the bank account

Use Case 4: Receiving Budget Alerts

Actors:

- User
- System

Triggers:

- User is almost over budget for category

Preconditions:

- The user has linked financial accounts
- The user has set budgets for categories
- The user has set alert preference (email, through app, etc.)

Postconditions:

- The system notifies the user through an alert

List of Steps:

- The user spends money in tracked category
- The system calculates total spending in category
- The system identifies that user is close to their set budget
- The system sends an alert to the user

Extensions/Variations:

- The alert preference is changed

Exceptions:

- Alert fails to send

Use Case 5: Generating Spending Reports

Actors:

- User
- System

Triggers:

- User selects “Generate Spending Report” on app

Preconditions:

- The user has linked financial accounts
- Financial data is available for time period

Postconditions:

- The system processes and generates a spending report for selected time period

List of Steps:

- The user selects “Generate Spending Report”
- The user selects a time period for report
- The system processes and generates a spending report

Extensions/Variations:

- The user wants to export report as a different file type

Exceptions:

- Insufficient data for selected time period

2. Non-functional Requirements

Scalability: The system should handle up to 10,000 concurrent users without performance degradation. This includes managing API calls to Plaid and retrieving data from the MySQL database efficiently.

Security and Privacy: User data must be encrypted in transit (HTTPS) and at rest (AES-256). Only authenticated and authorized users can access sensitive financial data.

Usability: The web application must be responsive and accessible on desktop devices and as a stretch goal, should also be accessible on mobile. All features should adhere to WCAG 2.1 accessibility guidelines.

3. External Requirements

1. The product must validate and handle user input errors gracefully.
2. Deployment must include a public URL accessible to end-users.
3. Comprehensive documentation must be provided, including installation instructions for developers and usage instructions for users.
4. The project scope must align with the team's resources and timeline.

4. Team process description

Brian Fang - Database Manager: Designs and maintains database, ensuring data integrity and secure storage.

Bryan Partida - API Specialist: Implements and integrates the Plaid API into the web app

America Pacheco - Front-End Developer: Designs and implements the UI/UX portions of the app, ensuring a visually appealing and functional app.

Ross Henderson - Back-End Developer: Develops server-side logic and manages the API routing.

Bailey Bounnam - Back-End Developer: Develops server-side logic and integrates with front-end.

Week	Task	Assigned Member(s)
1	Set up GitHub Repository and initial project structure	Ross Henderson
2	Complete front-end skeleton and basic navigation.	America Pacheco
3	Design database schema and set up MySQL database.	Brian Fang
4	Integrate Plaid API and finalize HTML	Bryan Partida, America Pacheco
5	Establish basic financial data retrieval	Bailey Bounnam, Ross Henderson
6	Implement budgeting tools	Bailey Bounnam, Brian Fang
7	Implement subscription tracking	Ross Henderson, Bryan Partida
8	Conduct usability testing and	Everyone

	refine UI/UX.	
9	Finalize project features and perform end-to-end testing.	Everyone
10	Finalize project documentation and deploy the application.	Everyone

Major Risks

API Downtime: Plaid API unavailability could disrupt data retrieval. Mitigation: Include fallback mechanisms.

Miscommunication: Poor coordination could delay tasks. Mitigation: Regular updates on Discord and GitHub Projects.

Security Vulnerabilities: Mishandling of financial data could lead to breaches. Mitigation: Adhere to security best practices.

External Feedback

Get feedback once the basic website is functional and plaid is integrated before working on major features. Probably around week 5 or 6.