1. Software architecture

Provide an overview of your system. Specifically:

- Identify and describe the major software components and their functionality at a conceptual level.
- Specify the interfaces between components.
- Describe in detail what data your system stores, and how. If it uses a database, give the high level database schema. If not, describe how you are storing the data and its organization.
- If there are particular assumptions underpinning your chosen architecture, identify and describe them.
- For each of two decisions pertaining to your software architecture, identify and briefly describe an alternative. For each of the two alternatives, discuss its pros and cons compared to your choice.

1.1 Overview

The Personal Finance Analyzer is a web application designed to help users track and analyze their spending habits, manage subscriptions, and set budgets. The application follows a layered architecture with a Front-End (UI/UX), Back-End (APIs and business logic), and a Database layer for secure financial data storage. We rely on the Plaid API for retrieving financial transactions from various financial institutions.

1.2 Major Software Components

1. Front-End (Client)
    - Technologies: HTML, CSS, JavaScript
    - Functionality
        - Displays user dashboards, budget tools
        - Sends requests to the Back-End for data
        - Offers interactive visualization
2. Back-End (Server)
    - Technologies: Node.js, Express
    - Functionality:
        - Receives requests from Front-End and processes business logic
        - Integrates with Plaid API to retrieve and normalize financial data
        - Interacts with the database layer for secure data retrieval and updates
3. Database Layer:
    - Technologies: MySQL?
    - Functionality:
        - Stores users, account information, transaction history, budget info
        - Ensures data consistency, security, and integrity through relational schema constraints
4. External API
    - Functionality:

    - Provides secure access to a user's bank account and transactions

1.3 Interfaces Between Components

Front-End and Back-End: RESTful API calls.
Back-End and Database: MySQL queries.
Back-End and Plaid API: Secure API calls for transaction retrieval.

1.4 Data Storage Details

User Data: UserID, authentication credentials (hashed), account settings.
Transactions: Date, amount, category, merchant.
Budgets: Category, allocated amount, actual spending.
Subscriptions: Recurring payments identified via transaction patterns.

1.5 Assumptions

- Users will link their bank accounts via Plaid.
- MySQL is scalable to compensate for user growth.
- All sensitive data will be encrypted.

1.6 Alternative Architectural Decisions

Alternative 1: Using MongoDB Instead of MySQL

- **Pros**: Easier to manage, built-in authentication.
- **Cons**: Complex queries can be slower, requires different indexing strategies.

Alternative 2: Using GraphQL Instead of REST

- **Pros**: Flexible schema, better scalability for unstructured data.
- **Cons**: More complex implementation, potential over-fetching issues.

2. Software design

Front-End

- **Dashboard.js**: Displays financial summary.
- **Budget.js**: Allows users to set and adjust budgets.
- **Transactions.js**: Lists user transactions with filtering.

Back-End

- server.js: Initializes Express app.
- transactions.js: Handles transaction-related API calls.
- plaidService.js: Fetches data from Plaid.

Database Tables

- users (id, email, passwordHash, settings)
- transactions (id, userID, date, amount, category, merchant)
- budgets (id, userID, category, limit, spent)

3. Coding guideline

JavaScript (Node.js, Front-End): https://github.com/airbnb/javascript

SQL (MySQL): https://www.sqlstyle.guide/

We chose these guidelines because they are widely accepted industry standards, ensuring code readability, maintainability, and consistency across the project. To enforce them, we will do continuous checks each week on the new pushes to the GitHub repo and make sure they follow the guidelines.

4. Process description

i. Risk Assessment

1. API Downtime
    - Likelihood: Medium
    - Impact: High
    - Evidence: Plaid API limitations
    - Mitigation: Implement caching, fallback strategies

2. Security Breach
    - Likelihood: Low
    - Impact: High
    - Evidence: Sensitive financial data
    - Mitigation: Use AES-256 encryption, HTTPS

3. Database Performance Issues
    - Likelihood: Medium
    - Impact: Medium
    - Evidence: High data volume
    - Mitigation: Indexing, query optimization

4. Team Coordination Issues
    - Likelihood: Medium
    - Impact: Medium
    - Evidence: Remote team, different schedules
    - Mitigation: Clear communication

5. Feature Integration Conflicts
    - Likelihood: Medium
    - Impact: Medium
    - Evidence: Multiple developers working on different components
    - Mitigation: Use feature branches, thorough code reviews, automated integration testing

ii. Project Schedule

- Project Setup: Repo, frameworks (1 week)
- Plaid API Integration: Auth, data retrieval (2 weeks)
- Backend Development: Routes, services (3 weeks)
- Frontend Development: UI Components (3 weeks)
- Testing & Debugging: Unit & Integration tests (2 weeks)
- Deployment: Cloud setup (1 week)

iii. Team Structure

- America Pacheco - Front-End Developer: UI design, HTML, CSS, JavaScript components
- Bryan Partida - Front-End Developer: JavaScript logic, UI interactivity
- Bailey Bounnam - Back-End Developer: API integration, authentication
- Ross Henderson - Back-End Developer: Feature implementation
- Brian Fang - Database Manager: Schema design, MySQL optimization

iv. Test Plan & Bug Tracking

- Unit Testing: Jest for JavaScript, Mocha for Node.js.
- Integration Testing: Postman for API validation.
- Usability Testing: User feedback sessions.
- Bug Tracking: GitHub Issues.

v. Documentation Plan (User Guide)

1. Introduction

- Overview of the Personal Finance Analyzer.
- Key features: Dashboard, Budgeting, Subscription Tracking, Spending Reports.
- How it helps users manage their finances.

2. Getting Started

- System Requirements: Browser compatibility, internet connection.
- Account Creation: Signing up, logging in, password recovery.
- Linking Bank Accounts: Using Plaid API for secure integration.

3. Using the Dashboard

- Overview of financial summary.
- Navigating different sections (transactions, budgets, reports).
- Customizing the dashboard.

4. Managing Transactions

- Viewing and filtering transactions.
- Categorizing expenses.
- Identifying recurring subscriptions.

5. Budgeting Tools

- Setting up budget categories.
- Tracking spending vs. budget.
- Adjusting budget limits.

6. Generating Reports

- Viewing spending trends.
- Exporting financial summaries.
- Customizing reports.

7. Security & Privacy

- Data encryption and protection measures.
- Managing account security settings.
- Disconnecting bank accounts safely.