Software Design Document

Sp-26 Green Budget App

Version 1.0

CS 4850 - Section 01 – Fall 2024

September 1, 2024

Prepared by: *Kelly Erhunse, Kody Clark, Katie Carlson, Tristan Stocks*

Table of Contents

1. Introduction and Overview **3**

2. Design Considerations **3**
2.1. Assumptions and Dependencies 3
2.2. General Constraints 4
2.3. Development Methods

3. Architectural Strategies 4
3.1. Programming Language and Framework Selection 5
3.2. Database and Data Storage Strategy 6
3.3. Integration with Financial APIs 6
3.4. Security Strategies 6
3.5. Scalability 6

4. System Architecture **6**

5. Detailed System Design **7**
5.1. Classification 7
5.2. Definition 8
5.3. Constraints 8
5.4. Resources 9
5.5. Interface/Exports 10

1. Introduction and Overview

This software design document outlines the requirements for the development of a financial management application, detailing its features, functionalities, and constraints. It serves as a guide for developers, project managers, testers, documentation writers, and end-users throughout the development and implementation process.

The financial management application is intended to provide a secure and user-friendly platform for managing personal finances. It will enable users to create accounts, securely store personal data, and manage their financial information using a range of features. Core functionalities include secure login with two-factor authentication, integration with the Plaid API for syncing bank account data, transaction tracking, budgeting tools, and personalized financial insights. The application is designed to help users set and monitor savings goals, track debts, visualize spending patterns with charts, and receive reminders for upcoming financial commitments. With a focus on flexibility and ease of use, this software aims to enhance personal financial literacy and provide a reliable tool for effective financial management, in line with corporate objectives.

2. Design Considerations

2.1. Assumptions and Dependencies

A program of this nature will have to use multiple dependencies to work. We will have to integrate API's from multiple places such as, banking institution integration for transactional trend analysis, client server infrastructure utilizing a database to ensure access and back end security. Additionally, we will also have to have an integration of a two form factor authentication service for client authorization access.

Within the development of the software program, we will target web, Android, and IOS as the platforms that our clients will interact with. This would allow around the clock access to their profiles, budgets, and tools wherever they have an active internet connection.

Throughout the software development cycle, our team can run into multiple challenges. These challenges, as outlined as dependencies prior, could therefore shape the entire landscape of the development project. Therefore, we have set out secondary routes to accomplish our application if integration is not possible.

Firstly, if we are unable to integrate Plaid as our banking institution linking software, we will then utilize an alternative such as OmniConnect. In the event of a failure at this stage, our tertiary plan would be to create a custom entry widget which a user could interact with.

Secondly ,we have our integration of a hosted database, our team has committed to utilizing one that will support our client server model. We have final candidates for Amazon Web Service (AWS), IBM cloud Free Database, Firebase, and Microsoft Azure. Lastly, if we need a tertiary option, we can develop our own in-house SQL server if these integrations are incompatible.

Finally, we have two form factor authentication that we have interest in integrating– either Google Authenticator, or Firebase. If critical failures occur while trying to implement this portion of our design, our team would then move to one factor authentication as a last resort.

2.2. General Constraints

One of the major barriers to banking software of this scope is the ability to adhere to all rules and regulations between agencies such as FinCen and other regulating bodies in the United States alone. Looking at an international scale, there are regulatory agencies that sit between countries such as the United States to Canada. The EU has an entire regulatory body plus individual countries. A failure to adhere to any single one of these bodies will result in punishments such as fines, the termination of the app and all the way up to criminal charges that can be To summarize, it is a giant undertaking to ensure that our program is able to abide by many of these authorities.

Due to the security requirements established by regulators, such as FinCen, our security requirements would have to establish encryption techniques at the front end layer, on users devices and also through our back end at our database system. Other techniques that could mitigate this would be to perform routine audits and robust authorization/authentication techniques.

2.3. Development Methods

Due to the amount of tasks, modules and planning the team has brainstormed, a hybrid method will be utilized for our development methods. The methodology that will be utilized is a Feature-driven Development process that will allow our team to tackle our stated requirements in series or in tandem, dictated by how closely we can follow our milestones. This will allow us to work

on features, such as authentication, and the back-end concurrently instead of in series. The flexibility of tackling each feature as its own entity allows our team to prototype many modules and concepts, and will allow more efficient module validation techniques, thus lowering development time.

There were other approach candidates that our team had as contenders, one being the Waterfall method. The simplicity of this method would allow us to perform all of our needed tasks in succession, making it easier for each part of development. This unfortunately does not support the strategy of prototyping and redesigning portions of our software. The team so far has not fully locked in the feature set and though requirements have been set, it would be difficult to strictly adhere to the Waterfall model.

Scrum was the last model observed by the team, and though it allows for frequent iteration on ideas, it did not make the team's vision of development methodology. One of the biggest features of making Scrum a robust development tool is to have daily meetings. The time constraint required to have daily meetings ultimately was the deciding factor in not accepting Scrum as a model.

3. Architectural Strategies

The architecture of the financial management application is designed to ensure security, scalability, and user-friendliness, aligning with the core objectives of enhancing personal financial literacy and providing a reliable tool for financial management. Several key design decisions and strategies have been employed to shape the overall structure and organization of the system, including the adoption of a modular client-server architecture (which separates the user interface from the backend processing), the integration with third party APIs, the use of the Flutter framework, the use of a relational database for storing user data, and the implementation of security measures such as two-factor authentication.

3.1. Programming Language and Framework Selection

The application will be developed using the Dart programming language. We also decided to use the Flutter framework, due to its ability to create dynamic, responsive user interfaces, and because it is able to seamless integrate with web technologies.

3.2. Database and Data Storage Strategy

A relational database, such as Dart, is selected for its strong compliance, ensuring data integrity and security, which are crucial for financial transactions and personal data management. Additionally, Dart provides scalability and supports complex queries needed for generating financial insights and visualizations. The use of this database also allows for future expansion, as it supports a range of data types and indexing options.

3.3. Integration with Financial APIs

The application will integrate with the Plaid API to securely connect with users' bank accounts and retrieve real-time transaction data. This decision leverages a well-established, secure API to avoid developing custom banking integrations, thereby saving development time and reducing security risks. This strategy aligns with the goal of quickly bringing a reliable financial management tool to market while maintaining high-security standards.

3.4. Security Strategies

To protect sensitive user data, the application will implement robust security mechanisms, including two-factor authentication (2FA) for login, encryption of all data at rest and in transit, and regular security audits. These decisions are driven by the priority of ensuring user trust and compliance with data protection regulations. The use of industry-standard security libraries and frameworks is favored over custom solutions to reduce vulnerabilities and leverage well-tested technologies.

3.5. Scalability

The architecture is designed to be modular and scalable, allowing for future enhancements such as the addition of new financial tools or integration with other financial services. The use of microservices architecture for backend services is considered to ensure that different components can be developed, deployed, and scaled independently. This strategy also facilitates future migration to other platforms or incorporation of new technologies without a complete overhaul of the system.

4. System Architecture

The system architecture we will be using for our budgeting application is client-server. This means that we are going to organize the system so that different tasks are handled by two parties: clients (the computer/device the users interact with) and servers (the computer/device that provides the service, stores and processes the data, and listens for requests from clients). The

client side will serve as the front end, with an interactive user interface (UI) component, and a storage component (for non-sensitive and session data specifically, not for long term storage). The server side will serve as the back end, and will have an authentication subsystem, a data storage component (for user data like personal information, budget settings, transaction history, etc), a subsystem that integrates with the Plaid API (in order to pull financial data from bank accounts), a notification and reminder component, and an analytics component.

The way this will work in action is that when the user interacts with the app, their input is processed by the client-side UI component. Then, the component that manages the user's data will retrieve the data from the database, and display it on the UI for the user. Then, if the user opts to connect their bank account, the request goes through the integration system with Plaid's API, allowing the app to access transaction data from the user's bank account. This data is then processed and used by the analytics component, which updates the UI accordingly. Additionally, if the user sets any goals or reminders, the app will store the data the user inputs by storing this information in the server side's database. This makes it so the client side and server side are constantly communicating over a computer network, so that the server can share its resources with the clients.

5. Detailed System Design

5.1. Classification

Our application will have a 2-layer client-server architecture with a UI Layer and a Data Layer. The UI Layer which will be in charge of displaying application data to the user's screen which will be handled by the client. The Data Layer, which handles the business logic of the program, will be handled by the server, not on the client device, which will require a network connection.

UI Layer:

- *Widget module*
- *Action module*
- *Communication (API) module*

Data Layer:

- *Firebase module*
- *2FA authentication module*

- *Plaid API module*
- *Transaction module*
- *Account recovery module*

5.2. Definition

UI Layer:

- ***Widgets:*** *Flutter has several components(which it calls widgets) that it uses to build its UI.*
- ***Actions:*** *This is what will allow an interactive and responsive application, the user will be able to select navigation windows and enter information*
- ***Communications (APIs):*** *The UI Layer will have communication protocols with the Data Layer*

Data Layer:

- ***Firebase module:*** *This will store user account details and login information such as primary email, secondary email, and cell phone number*
- ***2FA authentication module:*** *This will interface with a two-factor authentication program over a network connection and be crucial to the security of our application*
- ***Plaid API module:*** *This will be responsible for interfacing with user banking information*
- ***Transaction module:*** *This will facilitate the logic between the transaction actions presented to users in the UI layer and communication with the Plaid API module.*
- ***Account recovery module:*** *This will verify user accounts with secondary email and phone number and prompt for password resetting and username recovery*

5.3. Constraints

UI Layer:

- ***Widgets:*** *Widgets cannot typically have any size they want*
- ***Actions:*** *Base widgets may not have desired functionality may have to author own widgets to create desired functionality*

- ***Communications (APIs):*** *Amount of traffic being processed at once can be limited by network transfer speed*

Data Layer:

- ***Firebase module:*** *Speed and downtime can be possible constraints for the database*
- ***2FA authentication module:*** *Availability due to authentication servers being down*
- ***Plaid API module:*** *Some institutions may not accept Plaid API calls*
- ***Transaction module:*** *Data integrity could be a constraint*
- ***Account recovery module:*** *Will only operate if correct secondary email and cell phone number is supplied*

5.4. Resources

UI Layer:

Memory and storage:

- *Widgets*
- *Actions*

Network connection:

- *Communications (APIs)*

Data Layer:

Network connection:

- *Firebase database*
- *2FA authentication calls*
- *Plaid API calls - Plaid API service*
- *Transaction calls - Bank account service*

5.5. Interface/Exports

UI Layer:

Flutter API:

- *Widgets*
- *Actions*
- *Communications (APIs)*
- *Transaction module*
- *Account recovery module*

Firebase API:

- *Communications*
- *Firebase module*
- *Account recovery module*

Data Layer:

Plaid API:

- *Communications*
- *2FA authentication module*
- *Plaid API module: Plaid API*
- *Transaction module*
- *Account recovery module*

Firebase API:

- *Communications*
- *Firebase module*
- *Account recovery module*

Google Authenticator API:

- *2FA authentication module*
- *Communications (APIs)*