# Software Requirements Specification

# for

# 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

## Revision History

| **Name** | **Date** | **Reason For Changes** | **Version** |
|---|---|---|---|
| Kelly, Katie, Kody, Tristan | 9/1/24 | Original version | 1.0 |

# 1. Introduction

## 1.1 Purpose

This document provides a comprehensive overview of the requirements for the development of a financial management application. The document outlines the features, functionalities, and constraints of the budget application we are creating, serving as a guide for developers, project managers, testers, documentation writers, and end-users involved in the development and implementation of the product.

## 1.2 Project Scope

The budget application is designed to provide users with a secure and intuitive platform for managing their personal finances. The software allows users to register an account, securely store personal data, and manage their financial information through various features. Key functionalities include secure login with two-factor authentication, the ability to connect with the Plaid API to integrate bank account data, transaction tracking, budgeting tools, and personalized financial insights.

This application aims to empower users to manage their finances effectively by setting and monitoring savings goals, tracking debts, visualizing spending patterns through charts, and receiving timely reminders for upcoming financial obligations. By offering a flexible, user-friendly interface, the software aligns with corporate goals of enhancing personal financial literacy and providing a reliable tool for financial management.

# 2. Overall Description

## 2.1 Product Perspective

Our budget application will be a brand new product. Personal finance is an area many younger people struggle with, as according to the TIAA Institute, "Nearly four times as many young millennials demonstrated very low financial literacy than those who demonstrated a high level (31% vs. 8%); the corresponding result for older millennials is 24% vs. 13%. The 31% of young millennials with very low financial literacy is twice that among Baby boomers (15%)" [1].

As a result, our budgeting app will provide strategies and resources to help people of all ages with budgeting, but will have targeted resources specifically towards younger people (such as millennials and zoomers) who may be struggling with their finances.

The environment will be smartphones for maximum convenience. A lot of young people use their smartphone for financial decisions, as according to the Chase Digital Banking Attitudes Study, "87% of survey respondents say they use their banking app at least once per month or more," and "93% of millennials say they prefer to manage their banking in one place, followed by Gen X (90%) Gen Z (89%) and Boomers (84%)" [2]. Additionally, according to the TIAA Institute, "Millennials live a technology-enabled existence. More than 90% own a smartphone, which offers ready access to money management capabilities." [1]. So, because of all this, we are going to focus our efforts on making a mobile app (and, through using a framework like Flutter, make it accessible via both IOS and Android phones).

Our budget app will have an option to connect with a user's bank account, an option for a user to register and log into an account, a database server to store user data, a two-factor authentication system, and will integrate with Plaid's API for banking data.

**Considerations for DB:** *Firebase, Postgres, MySQL*

## 2.2 User Classes and Characteristics

We anticipate there being two user classes: general users (everyday individuals who use the app to manage their personal finances) and administrative users (all the group members of the project, any employees we may have, etc). We want to focus on general users, as they're our primary audience, and we want them to be able to update and delete their personal budget information at their convenience. Then, with administrative users, we want to make it so they have elevated privileges, but should only intervene due to a configuration issue or concern that a general user may have, making them primarily an asset for maintaining the app and keeping it running properly.

## 2.3 Operating Environment

The software will operate cross-platform. We are aiming to make a mobile app, so the platforms we want to utilize will primarily be Android and IOS. For the geographical location of users, we expect the userbase to be global (i.e, anyone in the world can use the app), but for an actual deployment, we would want to target data centers in millennial and zoomer residential hubs (since they're the most likely users we will be having). Infrastructure work for the application will be determined by programming language Dart, and our framework Flutter.

## 2.4 Design and Implementation Constraints

Some constraints we anticipate involve the bank account integration, as well as gathering necessary APIs. We anticipate that platform integration with Plaid API especially will be a design constraint, as we will have to follow the rules and policies of a third party as to what we could do. Also, obviously banks want to secure their information, so that gives us at least some constraints as to what we can do with transaction information as well.

Additionally,  incorporating two factor authentication may also be a constraint and cause issues, due to it being a little more complex than having a single factor for authentication.

Database integration may also be a possible design constraint as well, especially at scale. This, of course, depends primarily on the userbase of the app – if we have a lot of users, it may put strain on our database system to host all that data. We are not anticipating having a large enough userbase to cause this, but… it is a constraint regardless on the storage capabilities of the app.

## 2.5 Assumptions and Dependencies

**Some assumptions we are making going forward are:**

- That Plaid API will integrate with our app, and that it will be available, reliable, and continue to provide the necessary financial data for users to connect their bank accounts
- That 2FA will integrate with our app
- That we will use Flutter and Dart throughout development
- That the users will have reliable internet connections for data synchronization, API calls, and other network related features
- That the app will perform consistently across both iOS and Android platforms
- That the app will comply with current data privacy regulations, and there will be no additional regulations added while the app is in the development phase

**Some dependencies we will have include:**

- The Plaid API, for connecting and retrieving data from users' financial accounts
- A third party service for two factor authentication
- A cloud storage provider for storing data
- Developmental tools (such as IDEs like Visual Studio Code, Intellij, etc)
- The Flutter framework, and any design constraints within it

# 3. System Features

## 3.1 Account Creation

### 3.1.1 Description and Priority

The ability to create, maintain, personalize and recover accounts through an easy to utilize interface. This is a high priority, as it is a main feature of our app.

### 3.1.2 Stimulus/Response Sequences

*Main page:*

- *Account creation button*
- *Account recovery button*
- *Existing account login button*

*Secondary pages:*

- *Settings button for profile*
- *Switch profile button*
- *Log out button*

### 3.1.3 Functional Requirements

Account Creation:

Username, Password, Primary and Secondary Emails, 2 form authentication, bank account linking integrations

Account Recovery:
Performs 2 form authentication check, and utilizes primary and/or secondary emails for verification, will send username and/or password information.

Existing Account Login:

Takes username and password, performs verification of items. Successful login calls 2FA to authorize. Upon authorization login success. Negative authorization returns to login up to 5 times until lockout.

Settings button:
Moves to user profile to change settings such as primary/secondary emails, 2FA, bank account linking. All changes will require password and 2FA authorization.

Switch Profile button:

Will log out of the current user's account, and enter the Existing Account Login loop.

## 3.2 2 Factor Authentication (2FA)

### 3.2.1 Description

The ability to utilize email and a program such as Google Authenticator for protection of user information. This is a high priority because we want to use user accounts to be able to store each user's data efficiently.

### 3.2.2 Stimulus/Response Sequences

Authentication Page:
Dialog should form after successful username/password authentication to complete login through 2FA. Should have key information, messages and/or colors, to notify if successful/unsuccessful or timeout.

### 3.2.3 Functional Requirements

Connection to Third Party:
Has the ability for communicate with Google's 2FA service

## 3.3 Bank Account Linking

### 3.3.1 Description

The integration with Plaid's API will be the primary means with which a user will be able to transfer banking information to their profile. This is also a high priority (as having transaction information can help a lot with analytics), and can consist of:

***Credits** – any categorized financial income*
***Debits** – any categorized financial expenditure*
***Transfers** – any categorized financial transfer between users own/other banking accounts*

#### 3.3.2 Stimulus/Response Sequences

Connect to Bank Account Section:
Dialog that will allow a user to generate a Plaid service request to give permission for application to import all relevant banking information.

#### 3.3.3 Functional Requirements

Integration with Plaid API:
2 way communication to Plaid services to allow constant banking information updates. The information shall then be saved to the user's profile to later manipulate, show and analyze banking habits.

### 3.4 Budgeting Tools and Displays

#### 3.4.1 Description

The ability to display, organize and format banking habits into a number of ways. This is  high priority because it is the main purpose for the app.

We will have a suite of  tools that will be able to display various information through:

- Bar graphs
- Pie charts

Additionally, we will also have a set of tools to set budgets:

- ○ **Fixed Expenses** - *anything that is recurring and a requirement to be paid, usually missing a payment will incur a penalty such as Credit Score penalty or late fees.*

- ○ **Recurring payments** - *recurring payments not categorized as a fixed expense, can flag as things such as Netflix, Xbox etc.*

- ○ **Remaining Monthly Funds -** *The difference between the total amount of money the user has from their expenses and payments.*

- ○ **Entertainment -** *Pulls from categorization of entertainment purchases*

- ○ **Groceries -** *Pulls from categorization of grocery purchases*

- ○ **Custom Budget Tool -** *allows user to create own budget with categories:*
    - ***Name -*** *create custom name*
    - ***Time -*** *daily, monthly, annual*
    - ***Monetary amount -*** *ability to specify an amount by day, month, year*

### 3.4.2 Stimulus/Response Sequences

Buttons:

Interact with buttons that will navigate the user between different displays and tool features of the budgeting suite.

### 3.4.3 Functional Requirements

Integration with Plaid API:

Ability to communicate with the Plaid API, and scrape real time information such as vendor merchant codes to categorize expenses and credits.

Graphs and Other Visuals:

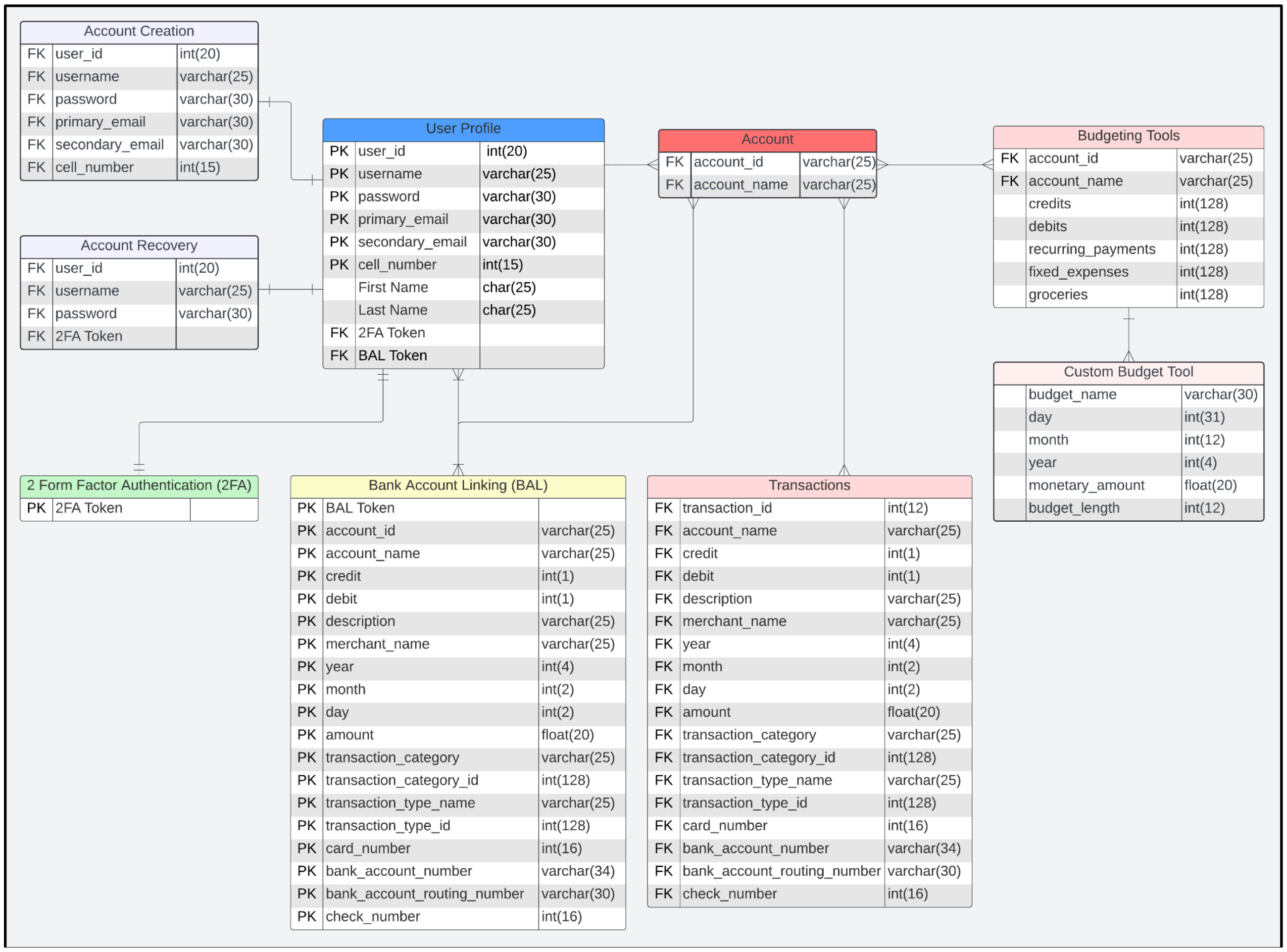
Ability to visualize information categories to the user

User Input and Categorizations:

Ability to change categorizations and create visualizations with the user's inputted information

# 4. Data Requirements

## 4.1 Logical Data Model



Account Creation

| | | |
|---|---|---|
| FK | user_id | int(20) |
| FK | username | varchar(25) |
| FK | password | varchar(30) |
| FK | primary_email | varchar(30) |
| FK | secondary_email | varchar(30) |
| FK | cell_number | int(15) |

Account Recovery

| | | |
|---|---|---|
| FK | user_id | int(20) |
| FK | username | varchar(25) |
| FK | password | varchar(30) |
| FK | 2FA Token | |

User Profile

| | | |
|---|---|---|
| PK | user_id | int(20) |
| PK | username | varchar(25) |
| PK | password | varchar(30) |
| PK | primary_email | varchar(30) |
| PK | secondary_email | varchar(30) |
| PK | cell_number | int(15) |
| | First Name | char(25) |
| | Last Name | char(25) |
| FK | 2FA Token | |
| FK | BAL Token | |

Account

| | | |
|---|---|---|
| FK | account_id | varchar(25) |
| FK | account_name | varchar(25) |

Budgeting Tools

| | | |
|---|---|---|
| FK | account_id | varchar(25) |
| FK | account_name | varchar(25) |
| | credits | int(128) |
| | debits | int(128) |
| | recurring_payments | int(128) |
| | fixed_expenses | int(128) |
| | groceries | int(128) |

Custom Budget Tool

| | | |
|---|---|---|
| | budget_name | varchar(30) |
| | day | int(31) |
| | month | int(12) |
| | year | int(4) |
| | monetary_amount | float(20) |
| | budget_length | int(12) |

2 Form Factor Authentication (2FA)

| | | |
|---|---|---|
| PK | 2FA Token | |

Bank Account Linking (BAL)

| | | |
|---|---|---|
| PK | BAL Token | |
| PK | account_id | varchar(25) |
| PK | account_name | varchar(25) |
| PK | credit | int(1) |
| PK | debit | int(1) |
| PK | description | varchar(25) |
| PK | merchant_name | varchar(25) |
| PK | year | int(4) |
| PK | month | int(2) |
| PK | day | int(2) |
| PK | amount | float(20) |
| PK | transaction_category | varchar(25) |
| PK | transaction_category_id | int(128) |
| PK | transaction_type_name | varchar(25) |
| PK | transaction_type_id | int(128) |
| PK | card_number | int(16) |
| PK | bank_account_number | varchar(34) |
| PK | bank_account_routing_number | varchar(30) |
| PK | check_number | int(16) |

Transactions

| | | |
|---|---|---|
| FK | transaction_id | int(12) |
| FK | account_name | varchar(25) |
| FK | credit | int(1) |
| FK | debit | int(1) |
| FK | description | varchar(25) |
| FK | merchant_name | varchar(25) |
| FK | year | int(4) |
| FK | month | int(2) |
| FK | day | int(2) |
| FK | amount | float(20) |
| FK | transaction_category | varchar(25) |
| FK | transaction_category_id | int(128) |
| FK | transaction_type_name | varchar(25) |
| FK | transaction_type_id | int(128) |
| FK | card_number | int(16) |
| FK | bank_account_number | varchar(34) |
| FK | bank_account_routing_number | varchar(30) |
| FK | check_number | int(16) |

## 4.2 Reports

Reports that will be formulated from this program should be to the user in the form of banking information that has aggregated from their bank account linking, and their overall budgeting documents and goals.

## 4.3 Data Acquisition, Integrity, Retention, and Disposal

Data will be acquired through user entry for profile creation. Data for account information will be pulled from Plaid API linked to users banking institutions. All data points will be stored in a remote database to serve the client server model. Mirrored back ups would be established for maintenance times for 24/7 operation.

# 5. External Interface Requirements

## 5.1 User Interfaces

All interfaces will have a responsive design (meaning, it adapts across various screen sizes and orientations, from small mobile screens to larger tablets). We will do this by utilizing flexible and expanded widgets, and using MediaQuery to retrieve screen size and orientation (so that the UI can be adjusted accordingly).

Additionally, all interfaces will have the logo for the app displayed in the top right corner, a similar color scheme, and a similar style of buttons/notifications/etc. Also, for every part of the app, there will be a help icon that guides the user to a FAQ section (which will answer common questions about the app and its functionality).

### 5.1.1 Logical Characteristics for Each Individual Interface

#### User Registration and Login

- *There will be input fields for email, password, and any 2FA codes.*
- *There will be buttons for actions such as signing up, logging in, and for password recovery.*
- *There will be error messages that appear if the user inputs incorrect credentials.*

#### Dashboard

- *The page will be divided into sections, such as 'Budget Goals', 'Recent Transactions', and 'Reminders'.*
- *There will be a navigation bar to help navigate through the sections.*
- *There will be graphs and other visualizations displayed on screen, which will change depending on the user's transaction history.*
- *In case there is an issue with loading the data, the dashboard will display a 'load again' or 'retry' message.*

#### Transaction Input Section

- *There will be input fields for transaction data (split into details such as amount, category, date, etc).*
- *There will be a save button, so that the user can save the transaction they just inputted.*
- *There will be input validation, so in case the user types in something wrong, it will be detected and let the user know via a notification or error message.*

### Connect to Bank Account Section

- *There will be a button that asks if you want to allow the app to connect to your account. Upon being clicked, the app will use Plaid Link in order to handle authentication and authorization for financial institutions.*

### Budget and Savings Goals Section

- *There will be input fields for setting goals.*
- *There will be progress bars for tracking goals.*
- *There will be buttons for submitting or changing any data (so, buttons like 'Set Goal', 'Edit Goal', and 'Delete Goal').*
- *If there are issues with saving changes, the UI will display an error message.*

### Profile Section

- *There will be buttons for various actions, such as "Add Profile Picture", "Edit Profile," and "Save Changes."*
- *There will be input fields for personal (but non-sensitive) information, such as "First Name" and "Account Created on (Date)".*
- *If there are issues with saving changes, the UI will display an error message.*

### Settings Section

- *There will be buttons for various actions, such as "Change Password," "Change Email," "Logout," and "Save Changes."*
- *If there are issues with saving changes, the UI will display an error message.*

### 5.1.2 Software Components

***Authentication Module:*** *Handles registration, login, and password recovery.*

***Notifications Module:*** *Handles any notifications, alerts, reminders, error messages, etc.*

***Dashboard Module:*** *Handles navigation, summarizes the user's financial history, and does analytics via visualizations.*

***Budgeting Module:*** *Handles the UI that helps users track their budget, and helps them set savings goals. This saves any goals they make to their account.*

***Plaid API Module:*** *Handles the integration with the Plaid API.*

***Settings Module:*** *Handles the UI that helps the users with settings related to their account (so, like changing passwords or logging out, for example).*

***Profile Module:*** *Handles the UI that helps the user set up a personal profile. This saves this data to their account.*

## 5.2 Software Interfaces

### *1. Plaid API*

Firstly, if given permission by the user, the budgeting app will interact with the Plaid API. This is done to connect user financial accounts to the app, allowing for retrieval of transaction data and account balances.This is done by the app sending HTTPS requests to Plaid's API endpoints. Plaid responds to these requests by returning JSON-formatted data (which contain bank account details, transaction histories, and account balances), and then the app will parse and map this data into the app's data structures.

### *2. Authentication Component*

Considering the app will identify users and their data based on their accounts, it is important to have an registration and authentication component. The authentication component will handle user registration, login, and two-factor authentication (2FA). It will do this by first having the user input their credentials (email, password) to whatever service we use for authentication (most likely Firebase's authentication service). The service will then return authentication tokens and user IDs, and when the app receives them, it will process and securely store that information (for the sake of using it during the session).

### *3. Database Component*

Since the app deals with financial data, it is important to have a database to store all that information. The database component (most likely Firebase's Firestore) will serve as a cloud database for storing user data (which includes personal information, budget goals, and transaction histories). This is done by the app sending HTTP requests to the database components to read/write data in the form of JSON objects. The database component will respond by doing this (and send a status code indicating an error if something goes wrong), and the app will in turn take this data and map to and from Dart objects within the app.

### *4. Flutter Libraries and Packages*

The framework that will be used for the budgeting app is Flutter. Flutter is going to be used because it allows developers to build the UI for multiple platforms within a single codebase. Due to this, the app will rely a lot on various Flutter libraries and packages for UI design, graph generation, HTTP requests, etc.

The way the data will be mapped and passed within this component is between various libraries and the app's core logic, and the data will take the form of Dart objects in order to do this.

***5. Entire Software Interface/All Components***

Finally, with each and every one of the components, it is ideal that all APIs and services respond within a few seconds, ensuring that the user has a quick and seamless user experience. Additionally, it is also important that all data exchanged between components is encrypted (especially since the app deals with financial data), and that sensitive user data is stored securely.

## 5.3 Hardware Interfaces

The hardware components of the budgeting app will largely depend on what kind of mobile device the user is using. However, the app will be multi-platform, allowing it to be accessible on both Android and IOS devices. Various hardware components for these type of devices include components such as a touchscreen (for user input), a storage component within the phone, and a network interface (to allow it to connect to the Internet).

***Touchscreen*** *– The touchscreen interface will be used for all user interactions within the app, such as entering data, navigating through the various sections, and interacting with visual elements like graphs. The app will be able to process these interactions because Flutter's gesture detection will allow it to capture user inputs (like taps and swipes). The inputs will be processed as touch events (which are detected as positional data), and the UI will either output a change in display (based on whatever the user inputted) or initiate an API call.*

*Ideally, this will all be done within milliseconds, as it is important to the user experience that the app responds with minimal latency.*

***Storage Interface*** *– The storage interface will be used for saving user data. The app will interact with the mobile device's storage through Flutter's file handling APIs, which will write input data to local storage (which includes JSON data for transactions, images, etc). The output data will be read from the device's storage (and will probably be used for data to related to things like the app's initialization, a cache for the app, etc). This data will also be stored in JSON format, as well as any other file types that the app supports (like PNGs for profile images, for example).*

*It is also ideal here that date access and processing times is optimized to be as quick as possible, in order to minimize any delays that could occur.*

***Network Interface** – The network interface will be greatly utilized by the app, because the app will have to use interfaces like Wi-Fi and cellular data to connect to APIs, and to sync data over the Internet. The app does this by using the device's network interface to make HTTP/HTTPS requests, as well as to manage the transferring of data. The inputs in this case will be any network requests (which includes API calls and data transmissions), and the outputs will be responses from the servers (and whatever data is retrieved from the APIs). The formats for this data will also be JSON files, and other similar data formats.*

*Like all the other components, it is important to optimize the network requests for latency, so that it can be ensured that the app will remain responsive during data transmission.*

## 5.4 Communications Interfaces

### 5.4.1 Email

The app is going to use emails for account-related notifications (so, things like password recovery, two-factor authentication codes, reminders, etc). These emails will not include a lot of information or any attachments (for the sake of preventing security vulnerabilties, and to ensure quick delivery), and they will most likely be sent using the TLS protocol, as that will ensure that the information within it is encrypted during transmission. For handshaking and synchronization, the SMTP protcol will be used (as it's purpose is to transmit emails over a network, and it can be used regardless of any computer or server's underlying hardware or software).

### 5.4.2 Network Protocols

Network protocols are going to be used for the sake of communication between the app, any external services (so, like Plaid's API, for example) and the app's backend server. HTTP/HTTPS will be used for all the API communications, and TLS/SSL will be used to encrypt all the data that's being transmitted over the network (to protect against things like eavesdropping and man-in-the-middle attacks). TCP handshakes will be used for message delivery, as it is a reliable way of transmitting data in a secure manner. When it comes to constraints, the app should be optimized to handle poor network connections (incorporating mechanisms such as a 'retry/load again' option, in case of lagging or delay).

#### 5.4.3 Electronic Forms

In the app, there are going to be electronic forms that are used for user input (for things such as budget goals, personal information, etc). The data will most likely be formatted in JSON, and will be transferred via HTTPS to the app's backend. HTTPS protocols will also be used for handshaking, and data will be encrypted during transmission (using TLS) to protect the user's sensitive information.

#### 5.4.4 Notifications

Notifications are going to be used by the app for the purpose of reminders, alerts, and error messages. The messages will be formatted in JSON once again (containing both the title and the message itself), and will occur when certain events are triggered by the user. They will appear seconds within being triggered, and will be small in size (in order to ensure quick delivery).  Users will be able to manage their notification preferences in the 'Settings' section of the app.

# 6. Quality Attributes

## 6.1 Usability

The software must be easy for users to learn and navigate, ensuring a user-friendly experience. New users should quickly understand how to register, connect accounts, and manage their finances with minimal guidance, supported by onboarding tips and help features. The interface should be consistent, with clear instructions and helpful error messages to guide users in correcting mistakes. Accessibility is important, so the app should meet web accessibility standards, making it usable for people with disabilities. Key tasks should be easily reachable within a few clicks, keeping interactions efficient and straightforward.

## 6.2 Security

Security is crucial due to the sensitivity of financial data. The software must include authentication for all logins to enhance account security. User data, including personal and financial information, should be encrypted both when stored and during transmission. Strict access controls should ensure that only authorized users can access sensitive data, and regular audits should check for any unauthorized access. Passwords should be strong, managed securely, and regularly updated, with a secure recovery process if users forget them. The system must comply with security standards like PCI-DSS and GDPR to protect user privacy.

### 6.3 Scalability

The software must be able to grow alongside its user base and data volume without losing performance. It should maintain speed and responsiveness even as more users connect their accounts and add transaction data. This ensures the system remains effective and efficient, regardless of how much it expands.

### 6.4 Modifiability

The software should be easy to update and maintain, allowing for new features or changes without major overhauls. This is achieved by using modular code that is well-documented, making it easier for developers to understand, modify, and expand the system over time.

## 7. References

1. https://www.tiaa.org/content/dam/tiaa/institute/pdf/summary/2018-11/tiaa-understanding-millennial-finances-r1o-10-30-18-final.pdf

2. https://media.chase.com/news/consumers-rely-more-and-more-on-mobile-banking