Software Requirements and Design Document

For

Group 5

Version 1.0

Authors:

Trevor Fagan

Dorothy Palmer

Chris Que

Wifredo Huertas

Jose Miguel Sarenas

1. Overview (5 points)

Create a personal budget web app that tracks income and spending information. Give users the opportunity to connect with their personal bank through the Plaid API to get real time updates of their spending habits. The features will have an admin panel/administrative role that has user credentials, account type, and user count. The user will have the features of the budget, spending, income, and transaction history. There will also be login authentication, category tracking, and the option to add manual transactions.

2. Functional Requirements (10 points)

1. SQL functionality where an example user's information could be accessed from the database and displayed on the screen. Medium.

2. Auth 0 API allowed the user to sign in with a username and password, or just sign in with Google. This feature makes it easier for a user to login, rather than having them make a completely new account, they can use their Gmail. Medium.

3. Ability to connect to your bank(just sandbox mode for now, so real bank account is not being used). This allows users to update their budget live, and provide access to their bank accounts. High Priority.

4. Database with the user table is functioning, but we have not populated any data to it yet. This allows for accurate representation of information from the user. Medium.

5. Added charts with the information regarding income and expenses. Low.

6. Buttons for navigation to different pages. Low.

7. Logout feature. This feature provides a safe way for the user to keep their information

protected from other users. Low

8. Navigation to twitter, Instagram, and other social medias from the dashboard page. Low.

9. Ability to populate information regarding your transactions. Medium.

10. Dashboard functionality takes you back to the main page. This allows for easy navigation from the user from page to page. Low.

11. Ability to manually enter a transaction by clicking the "+" button. Low.

12. Ability to navigate to the other pages by clicking the boxes on the dashboard. Low.

3. Non-functional Requirements (10 points)

1. Users of the Mojito app system shall authenticate themselves using their gmail account. Derived by the authentication system using gmail.

2. Mojito users shall be able to use the system at all hours of the day.

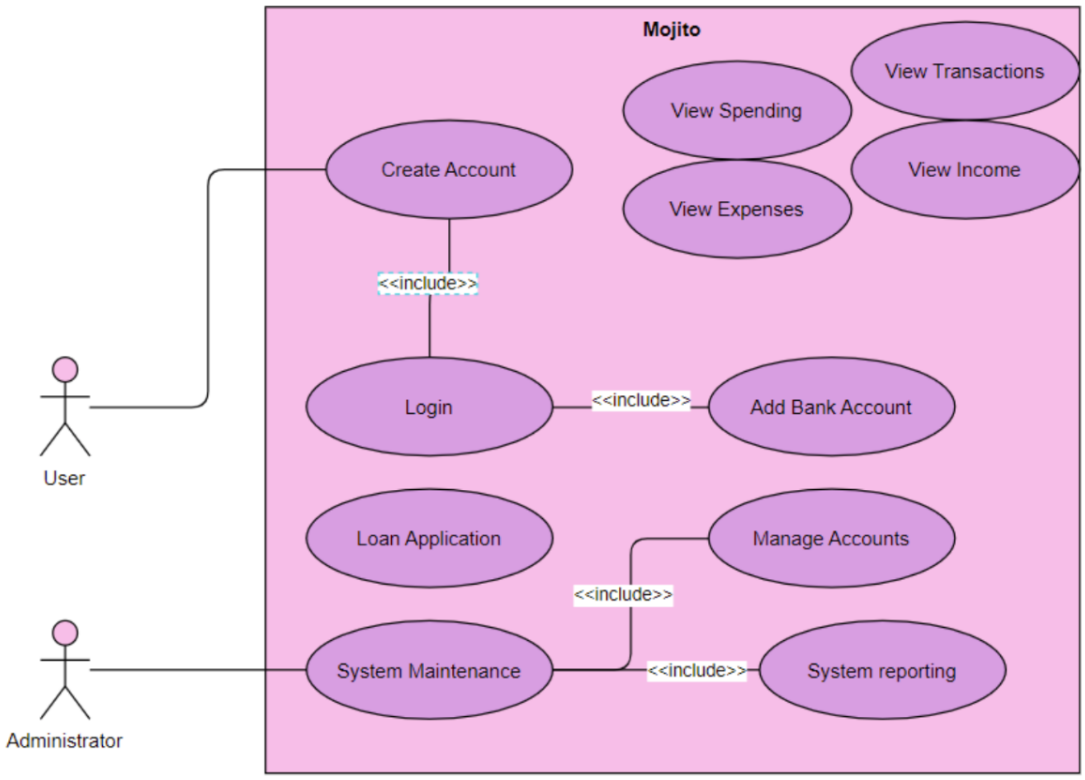
3. Mojito users' bank account information shall not be accessed by a user that is not them. This means that only the person who is tied to the bank account can see the bank information/budget. Derived by the need to keep bank account information private.

4. Mojito shall be able to store the bank account information.

4. Use Case Diagram (10 points)

*This section presents the **use case diagram** and the **textual descriptions** of the use cases for the system under development. The use case diagram should contain all the use cases and relationships between them needed to describe the functionality to be developed. If you discover new use cases between two increments, update the diagram for your future increments.*

***Textual descriptions of use cases**: For the first increment, the textual descriptions for the use cases are not required. However, the textual descriptions for all use cases discovered for your system are required for the second and third iterations.*



5. Class Diagram and/or Sequence Diagrams (15 points)

*This section presents a high-level overview of the anticipated system architecture using a **class diagram** and/or **sequence diagrams**.*

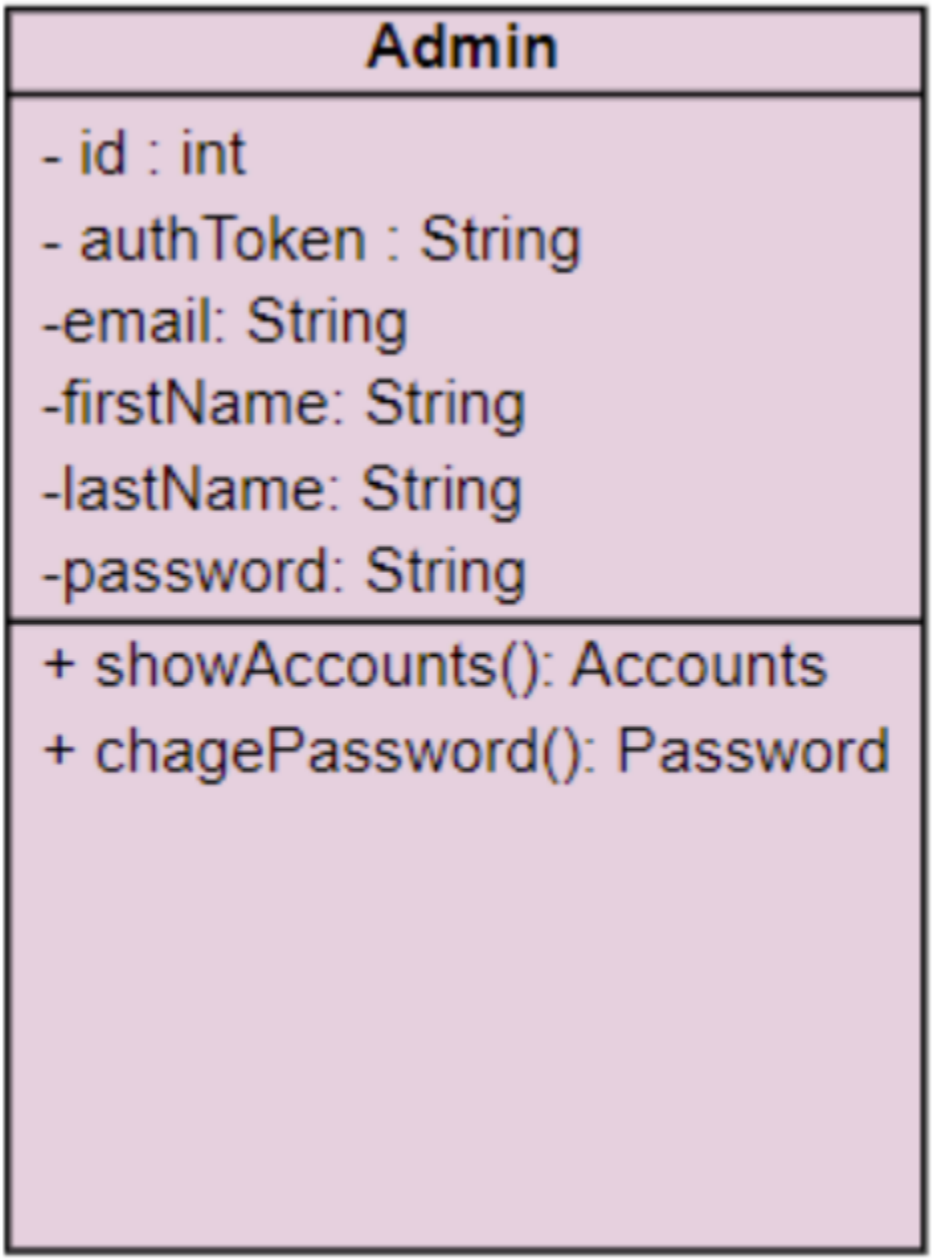
*If the main **paradigm** used in your project is **Object Oriented** (i.e., you have classes or something that acts similar to classes in your system), then draw the **Class Diagram of the entire system and Sequence Diagrams for the three (3) most important use cases in your system.***

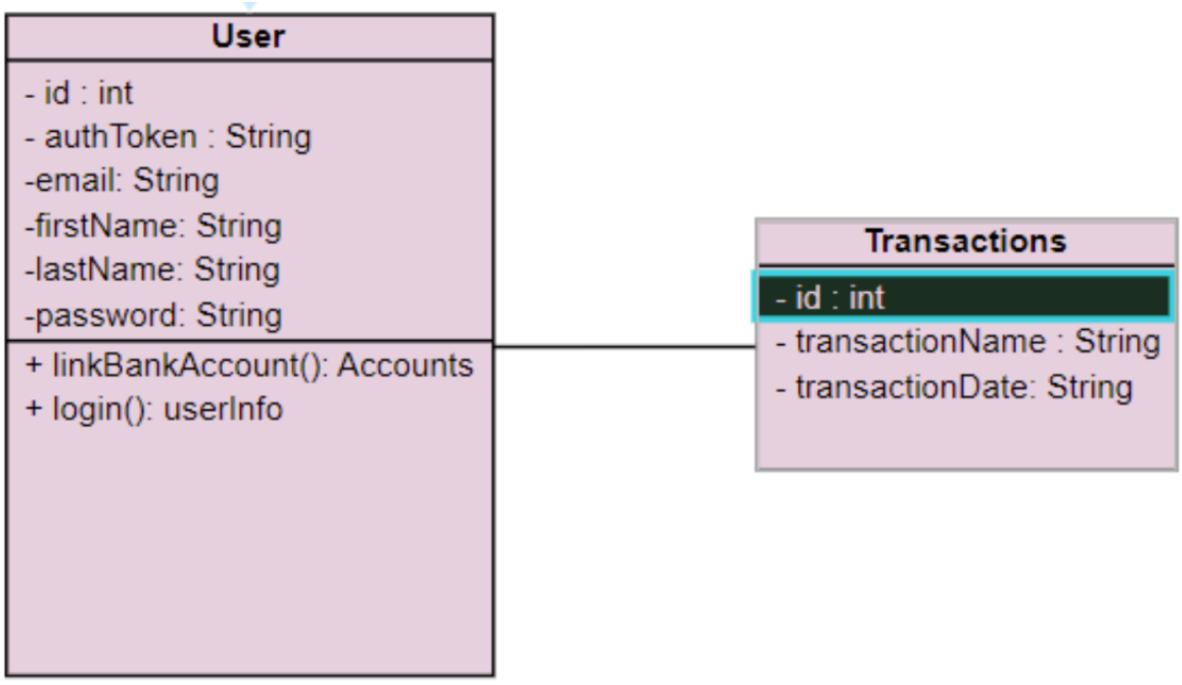
*If the main **paradigm** in your system is **not Object Oriented** (i.e., you **do not** have classes or anything similar to classes in your system) then only draw **Sequence Diagrams**, **but for all the use cases of your system.** In this case, we will use a modified version of Sequence Diagrams, where instead of objects, the lifelines will represent the functions in the system involved in the action sequence.*

Class Diagrams** show the **fundamental objects/classes** that must be modeled with the system to satisfy its requirements and **the relationships** between them. Each class rectangle on the diagram **must

also include the attributes and the methods of the class *(they can be refined between increments). All the* ***relationships between classes and their multiplicity*** *must be shown on the class diagram.*

A ***Sequence Diagram*** *simply depicts* ***interaction between objects*** *(or* ***functions -*** *in our case - for non-OOP systems) in a sequential order, i.e. the order in which these interactions take place. Sequence diagrams describe how and in what order the objects in a system function.*





6. Operating Environment (5 points)

● Hardware Platform: Amazon EC2 t2.large (2 vCPUs, 8GiB memory, 16 GB EBS Volume SSD), 64-bit (x86) ● Operating System: Linux kernel 4.14, Ubuntu 20.04 LTS ● Other: systemd 219, GCC 7.3, Glibc 2.26, Binutils 2.29.1 ● Database: Amazon RDS MySQL

7. Assumptions and Dependencies (5 points)

The Plaid API continues to allow sandbox development mode. We also assume that they will allow us to move forward to the development part of the API so that we can start to integrate our own data. If not, we will have to move forward with dummy data - The free SQL database we are using continues to be free and allows multiple tables such as we will be needing - The Twillio API works and has all of the API calls that we need in order to send text messages to our users Dependencies: - Vercel continues to allow React applications to be hosted on their platform so when the time comes for deployment, we can still make sure everything is integrated properly - Tailwind css continues to work alongside React and that any updates do not cause errors with other dependencies.