****

**Final Team Project**

**CSC 4022-470 Secure Software Engineering**

**FinApp**

**Financial Insights Web Application**

****

***Turing Tribe: Grant McPheeters, Mitchell Phelps, Aaron Garrison, Thea Banis,***

***Professor Rhodes***

***[insert your name]***

**Table of Contents**

[Project Overview 3](#_Toc147054538)

[Project Goals 3](#_Toc147054539)

[Project Features 3](#_Toc147054540)

[Target Audience 3](#_Toc147054541)

[Project Risks 4](#_Toc147054542)

[Design and Secure Architecture 5](#_Toc147054543)

# Project Overview

This project aims to create a user-friendly web application for financial insights, which will allow users to securely view and visualize their financial data from multiple different sources, including bank accounts, stocks, and liquid assets. The primary data source will be Plaid API, with possible integrations of other APIs for stocks and cryptocurrency data.

# Project Goals

1. Provide users with a centralized platform to monitor their financial accounts and investments.
2. Deliver intuitive data visualization through interactive charts and graphs.
3. Ensure real-time accuracy and security through Plaid API integration.
   * Consider additional APIs for stock market and cryptocurrency data.
4. Create a user-friendly and efficient interface for a smooth user experience.

# Project Features

The project scope outlines the specific areas of focus and responsibilities:

* **User Authentication:** Users will have the option to create accounts and securely log in to the platform.
* **Financial Overview:** Users will be presented with an intuitive dashboard that provides an overview of their financial assets, including bank account balances, stock portfolios, and liquid assets.
* **Data Visualization:** This web application will consist of interactive charts and graphs to help users visualize their financial data effectively.
* **Plaid API Integration:** This application’s functionality will rely on the Plaid API to fetch and update user financial data.
* **Stock and Cryptocurrency Data:** Additional APIs will be considered for integrating stock market and cryptocurrency data, offering users a comprehensive financial snapshot.
* **User-Friendly Interface:** The user interface will be designed with ease of use in mind, ensuring a smooth user experience.

# Target Audience

The target audience for this application focuses on casual investors looking to monitor basic transactions and funds on all connected accounts and people who wish to see a nice roundup of all of their financials in one application. This application is not anticipated to show advanced analytical data, so any advanced features would be done on the appropriate service.

Project Requirements

Functional Requirements

* + User authentication before full website access and maintained throughout the lifetime.
  + Secure any received data from Plaid/financial APIs. Data needs to be encrypted and securely deleted when it's not needed.
  + Modular, self-contained elements for each financial application connected by the user. These containers will show all relevant data, such as invested funds, transactions, and graphic elements. Data needs to be masked between all containers.
  + Historical data should be polled from plaid/APIs whenever available instead of caching to reduce data exposure.
  + GUI consists of various device resolutions and browsers.
  + Support HTTPS and restrict the use of HTTP protocols.
  + Create modular graphics and charts for data display.

Non-Functional Requirements

* + Physical security on balances and transactions. The app will only display numeric values when prompted by the user to prevent people from viewing users’ financials ‘over-the-shoulder'.

Project Team

The project team will consist of the following roles sourced from Turing Tribe:

* Project Manager
* Systems Administrator (Backend Developer)
* Frontend Developer/ UX/UI Designer
* Quality Assurance Tester

The diversity of this team will bring a range of skills and knowledge necessary for the successful development and deployment of this financial insights’ web application.

# Project Risks

Preliminary Threat Assessment - While our primary focus is on providing valuable financial insights, we acknowledge the importance of security. Below is a list of our preliminary security considerations:

* **Data Security**
  + Stored data will be encrypted. This includes users’ usernames, passwords, and API keys.
  + Physical security by means of hiding information in the app unless requested by the user.
  + Implement proper account permission on databases such as read-write access
  + Implement proper data sanitation on all User input received from the program
    - SQL injection attacks
    - Some mitigation on cross-site-scripting attacks
* **API Security**
  + Proper use of API keys (both private and public).
  + Confirmation from users on the use of Plaid API.
  + Proper verification of requests from the Plaid API.
  + User setting to save/or not save API authentication with their accounts
* **User Authentication**
  + Users will be prompted to make a secure password. At least 12 characters long, at least one uppercase and one lowercase letter, at least 1 number, and at least one special character.
  + After password confirmation, the User will be prompted with their dual-factor authentication such as email/text.
* **Updates**
  + Updates throughout planning addressing any security risks or flaws.

# Design and Secure Architecture

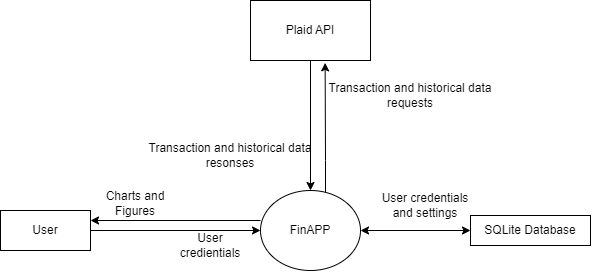
* **Secure Design Principles**
  + *Least Privilege*
    - The concept of least privilege is that a user is given the minimum access needed to perform their functions on the FinApp application. This concept can be seen in the Access Control Mechanisms section.
  + *Defense in Depth*
    - The concept of defense in depth is using multiple security practices to ensure the applications is safeguarded in multiple ways from multiple attacks or exploits. This concept is demonstrated throughout the rest of this section.
  + *Fail-Safe Defaults*
    - The concept of fail-safe defaults is that all users – including developers – are given default system privileges with security in mind. For FinApp, developers will not have plaintext access to users’ data by default and will only have access to specific functions for development purposes. For example, the front-end engineer does not need access to the actual database, only a test database by default.
* **Access Control Mechanisms**

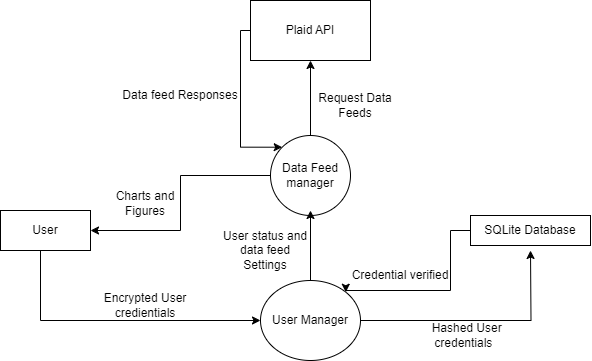
Access control mechanisms involve the privilege at which specific users or groups can access data. Specific users and groups will be defined below that have specific privileges. These privileges should be unchanging and only allow the user to access the data needed to perform their intended actions in the app. This also complies with the least privilege idea.

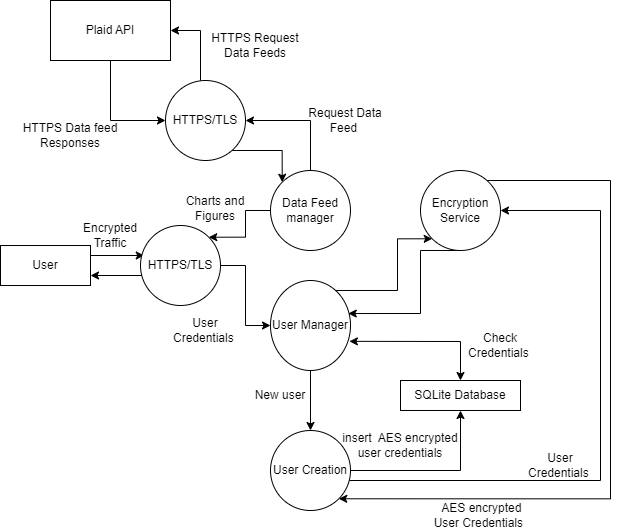
* + *User Groups*
    - Account Users
      * Will have access to their own financial data, username, password, and API keys. Under no condition should an account user be able to access the data of other users which will include usernames, passwords, API keys, or any financial data.
    - Test Users
      * Will have access to test data to test out the functionality of the app to see if they would want to create an account. Similar to account users, test users should only be able to access the test data on the app, excluding any test API keys, passwords, and usernames of test accounts. This will also be used as the main development account group for developers.
    - Data Admin
      * Will have access to the database, which will include hashed passwords and usernames. They will not have access under any circumstances to plain text passwords, plain text API keys, or any financial data of the user.
* **Encryption Strategies**

Encryption strategies are used to keep data safe, both in transit and in storage.

* + *Encryption*
    - The encryption standard of the industry is AES. FinApp will use AES encryption for all data including; API keys, passwords, personal information, biometrics, routing numbers, transactions, and investments will be encrypted on storage.
  + *Hashing*
    - Hashing is the process of running plaintext through a hashing algorithm to produce a seemingly random set of characters. The server side of FinApp will use hashing and salting to verify passwords.
    - Django uses the PBKDF2 algorithm with a SHA256 hash for password stretching and the encryption data in this framework will have a special key that will only be known by the application server.
* **Secure Communication Protocols**
  + Being a web application, FinApp will use HTTPS protocol. With the HTTPS protocol, servers and browsers must establish communication parameters before data transfers. This is especially important due to the data that FinApp will contain. While it does not allow users to make transactions or deal with money directly, it does have data that is extremely sensitive and could cause people to become potential targets if their data is acquired. Hashing and encryption will assist in secure communication protocols to ensure that FinApp is not vulnerable to man-in-the-middle attacks and spoofing.
* **Threat Modeling**
  + In the development of FinApp, our primary focus is on providing valuable financial sights while ensuring the highest levels of security. Threat modeling is a critical step in identifying potential vulnerabilities and addressing them proactively. Below are the key points related to threat modeling for this project:
    - Data Security – Being a financial-based insight app, we consider data security as paramount. All stored information, including usernames, passwords, and API Keys, will be fully encrypted at rest and transport. The physical security of this sensitive data will be ensured by concealing it within the app unless requested by the user in a different format.
    - Access Control – Proper access control mechanisms will be implemented to ensure that users and groups have least privilege necessary. Users will only be able to access data required for their intended actions within the app.
* **Protection Mechanisms**
  + To safeguard FinApp from potential threats, we will employ various protection mechanisms, adhering to standard secure design principles. These mechanisms include the following:
    - Privilege – Users will me granted the minimum amount of access necessary for their functions within the application. Thus ensuring that no user can access data beyond their needs and that administrative accounts are adequately configured.
    - Defense in Depth – We will incorporate security at many levels within the application; from application to physical layers, we can protect the application from various attacks and exploits.
    - Access Control Mechanisms – Specific user groups will be defined with distinct privileges. No one user will have the capability to execute executive functions without the confirmation of another. This will ensure that users can only access data required for their intended actions.
    - Encryption – Strong encryption such as AES will be used for sensitive data in both transit and storage. This will include API keys, passwords, phone numbers, addresses, and names. At no point will any of this information be stored in plaintext.
    - Threat Modeling – Regular threat assessments and analysis will guide the continuous improvement of implemented security protocols and practices within the FinApp.
* **Data Flow and Storage Security**
  + See attached images or “DFD Documents and Images” folder in the repository to see level 0-2 DFD.
  + Storage Security is focused on keeping the database encrypted using AES and verifying each users identity on all requests.







* **Error Handling and Logging**
  + *Errors*
    - Errors need to be handled securely when FinApp has access to users’ data. In FinApp, the first assurance needs to be that errors do not allow users to access data they are not supposed to. For example, in the off chance that an error allows a user to figure out the API key of another user or spoof it, that user would be able to access data that they are not allowed to. In this case, a proper way to deal with this error is to tie an API key to an account so that they would not be able to use the key.
    - During a critical error, the user sessions should be disconnected, and all data financial data should be deleted. Users should be alerted that an error has occurred and properly logged.
  + *Logging*
    - Logging is especially important to FinApp. Specific logs should include failed attempts at logging in, changes in passwords or usernames, and changes in API keys. Logging these specific events would allow developers and admins to deduce if a specific user’s account is under attack or potentially hacked – either through a fault of the app or a social attack on the user. All logs must avoid having any PII that could compromise the user's account.
* **Secure API Design**
  + FinApp is using the Plaid API. This allows us to connect to participating banks and other financial institutions. The Plaid API authenticates and has multiple functions to ensure the identity of the user. Some of the functions that we will use involve checking the address and phone number of a user to see if it matches with the bank account they are trying to connect to.
* **Third-Party Integration**
  + FinApp is currently set to use several third-party libraries. These libraries focus on Django for the web service and may use either PyCryptodome or Cryptography for security services. Django does come with SQLite, so it will likely be used for the database.
* **Scalability and Performance**
  + Django is known for its scalability and its ability to handle both large and smaller loads. As loads change, Django can move application states to different technologies. These customizations can include the database, session management, and file storage.
* **Documentation**
  + *User Groups*
    - There will be three user groups for FinApp with specific privileges and defaults. They are the Account Users, Test Users, and Data Admins. Their exact privileges are listed above in Access Control Mechanisms.
  + *Encryption and Hashing*
    - On storage, passwords, API keys, phone numbers, and addresses will be encrypted with AES.
    - Password verification will be done using hashing and salting to ensure that the server will not know the users’ sensitive data.
  + *Error Handling and Logging*
    - Errors need to be handled appropriately and not allow for vulnerabilities. Critical errors need to disconnect the user and delete all financial information.
    - Logs need to include login attempts, changed passwords and usernames, and changes in API keys.
  + *Secure API Design*
    - Use Plaid API’s inbuilt verification and authentication methods to ensure the security of the API calls.
* **Peer Reviews**
  + Peer review will be conducted by pull requests for all feature branches we build during development. These pull requests are focused on maintaining coding standards and verifying that code meets requirements.
* **Iterative Design**
  + As the Turing Teams learn new secure implementations and strategies, this document may be modified to implement more appropriate security protocols. Additionally, through checkups with our stakeholders, we may learn of changing requirements that apply to our secure design. Continuous threat assessment may also change the protocols of FinApp as we learn how to implement the Plaid API better and develop further. Feature development will be implemented around small sprints where individuals will be given small modules to build. These modules will be the basis of combining into larger systems that will be unit and integration tested.
* **Review with Stakeholders**
  + FinApp plans on having monthly meetings with their stakeholders to allow for development time of potential changes to functionality and new features. This will also include meetings with security professionals to allow for proper analysis throughout the development cycle. Finding a potential vulnerability early allows FinApp to fix it and be more secure on release.
* **Design Validation**
  + *Unit Tests*
    - Encryption Unit Test - Verify that encryption modules encrypt and decrypt data.
    - Database Unit test - Verify successful operations of the database. Tests need to add, modify, and delete of data in the database.
    - User creation unit test - Verify that a user can successfully be created and be able to log in.
    - API Unit Test - Run tests on used API functions.
      * Successful login.
      * Successful query.
      * Successful logout.
    - Visual Chart Unit Test
      * Successful creation of charts using fake data
      * Test for each offered chart.
  + *Integration Testing*
    - Finance Module Unit Test
      * Create a fake module that is created and used by each individual application.
        + Balances
        + Charts
        + Transactions
        + Data From API