**Software Requirements Specification**

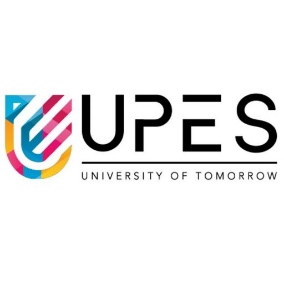
For

**Finance Management Application**

10/11/24

Prepared by

|  |  |  |
| --- | --- | --- |
| **Specialization** | **SAP ID** | **Name** |
| BCA AIML | 500110037 | Aviral Birthare |
| BCA AIML | 500106531 | Munendra Yaduvanshi |
| BCA AIML | 500110172 | Harsh Sharma |
| BCA AIML | 500112513 | A.Teja |



School of Computer Science

UNIVERSITY OF PETROLEUM & ENERGY STUDIES,

DEHRADUN- 248007. Uttarakhand

Table of Contents

|  |  |  |
| --- | --- | --- |
| **Topic** | | **Page No** |
| Table of Content | |  |
| 1 | Introduction | 3 |
|  | 1.1 Purpose of the Project | 3 |
|  | 1.2 Target Beneficiary | 3 |
|  | 1.3 Project Scope | 3 |
| 2 | Project Description | 4 |
|  | 2.1 Reference Algorithm | 5 |
|  | 2.2 Data/ Data structure | 5 |
|  | 2.3 SWOT Analysis | 6 |
|  | 2.4 Project Features | 6 |
|  | 2.5 User Classes and Characteristics | 6 |
|  | 2.6 Design and Implementation Constraints | 7 |
|  | 2.7 Design diagrams | 8 |
|  | 2.8 Assumption and Dependencies | 12 |
| 3 | System Requirements | 13 |
|  | 3.1 User Interface | 13 |
|  | 3.2 Software Interface | 14 |
|  | 3.3 Database Interface | 14 |
|  | 3.4 Protocols | 14 |
| 4 | Non-functional Requirements | 14 |
|  | 4.1 Performance requirements | 14 |
|  | 4.2 Security requirements | 14 |
|  | 4.3 Software Quality Attributes | 14 |
| 5 | Other Requirements | 15 |
| Appendix A: Glossary | | 15 |
| Appendix B: Issues List | | 15 |

* 1. INTRODUCTION
  2. **Purpose of the project**

The main and primary objective of this project is to create a user-friendly Finance Management App that simplify the personal finance management of the user by using comprehensive tools for effective management enabling decision-making and financial stability. The app will allow user to project focuses on building a fully functional online banking platform using modern web technologies. The platform will allow users to manage their finances efficiently and securely.

Through the use of latest modern technologies like React.js for the UI, Next.js for building the online banking platform with server-side rendering, Node.js for server-side technology used to build APIs and handle backend logic, and Plaid for connecting bank account and retrieve real bank data. The application not only guarantees seamless user experience but also ensure a robust backend capability of handling request efficiently. Key features include:

a) **Real-Time Transaction Displays**: Users can see their recent transactions and account balances instantly.

b) **Secure Money Transfers**: Transfer funds between accounts with enhanced security measures.

c) **Multi-Bank Integration**: Connect and manage multiple bank accounts from a single interface.

* 1. **Target Beneficiary**

The main beneficiary of the app is to help the individual who wish to track and Streamlined solution for their personal finances. This App focus is on users who wants to manage bank account and perform tasks such as fund transfers, checking account balance and tracking users’ transaction. Users often face difficulties in using data from various sources, and this application provides users a single platform to address those needs. This application can be also beneficiary for young professionals and students who need to manage their limited funds effectively. By offering a user-friendly interface to students and secure financial tools, the application can help users to plan their finances more effectively. This application does not initially target business users but remains open to future enhancements to support such use cases. The application is majorly beneficial for those users who prioritize real-time updates and secure access to their financial data. It is designed to ensure a consistent experience for users regardless of whether they access the app on a mobile device or desktop computer.

**1.3 Project Scope**

This project covers various features, such as bank integration through Plaid API, users’ money transaction monitoring, and a responsive design ensuring accessibility on multiple devices. The application will be built using modern technologies such as React.js and Next.js for the frontend, and Node.js and Appwrite for the backend. Additionally, Tailwind CSS will be utilized for an intuitive user interface, and Vercel will handle deployment for high availability.

This project mainly delivers a full functional application with a secured backend, a good-looking responsive UI that can be accessible in other platform, and a proper documentation for the users. This Finance Management App will be allowing users to:

* Track their daily, weekly, and monthly spending and transaction.
* Sending money to different bank account.
* Keeping track of all the connected bank cards.
* Real-Time Updates.

1. PROJECT DESCRIPTION
   1. **Reference Algorithim**

The project will use many different algorithms for authentication, data encryption, and transaction management. Queue-based algorithms will be used for handling asynchronous tasks like processing transactions in real time, ensuring users receive updates without delays. The use of JWT (JSON Web Tokens) for authentication ensures that user sessions are secure, while OAuth integration with third-party APIs like Plaid allows for secure access to financial data. Data structure that we are going to use for this project are as mentioned below:

1. Arrays/Lists:

* Used to store multiple transactions, account balances, or bank accounts.
* Example: A list of recent transactions or connected bank accounts.

1. **Dictionaries/Maps:**

* Used to store key-value pairs, such as user details (name, email), account information (account number, balance), or transaction metadata (transaction ID, amount, date).
* Example: {“account\_id": 123, "balance": 5000, "currency": "USD”}

1. Queues:

* Used for handling asynchronous tasks like processing transactions or real-time updates. Queues help manage the order of tasks efficiently.
* Example: A queue for pending financial transactions waiting for server validation.

1. Trees:

* Used in hierarchical structures like categories of expenses or financial goals (e.g., Budget → Savings → Investment).
* Example: A tree structure can categorize transactions (e.g., Food → Groceries → Supermarket).

1. **Graphs:**

* Used for managing connections between different accounts or relationships between entities like users and banks.
* Example: A graph could represent user relationships with multiple bank accounts and their respective balances.

1. Stacks:

* Used in undo/redo functionalities, especially for user actions such as editing transactions or updating financial goals.
* Example: Stack structure for tracking recent user actions.

1. Hash Tables:

* Used for quick lookups of user data, transaction history, or account details.
* Example: A hash table to quickly access transaction details using transaction IDs.

Most of these data structures are abstracted away by libraries and frameworks, so in most cases, we will be using higher-level abstractions like arrays and objective.

* React.js will handle much of the UI state using virtual DOM, arrays, and objects.
* Node.js will manage server-side data using arrays, objects, and sets for managing users, requests, and session data.
  1. **Characteristics Of Data**

The dataset that we will be handling, we will include transaction data such as income, expenses are user-generated data, account balances, and users credentials the primary data source is real-time bank data that we will be fetching via APIs like Plaid, while the secondary data sources may include some manually entered budgets or user’s custom financial goals, that is stored securely in the backend database that will be managed by Appwrite and Supbase.

As the data is highly sensitive and must be used/processed and stored with the highest security. Features like real-time updates primarily rely on the sampling techniques from the bank APIs. ensuring minimal latency. Statistical methods may also be applied to aggregate data, that will provide users with the meaningful insights, such as monthly expenditure reports.

* 1. **SWOT Analysis**

Strengths:

1. **User-Friendly Interface:** Easy-to-navigate design ensures a positive user experience.
2. **Real-Time Updates:** Instant access to transaction history and account balances across multiple accounts.
3. **Security:** Strong security measures like authentication and secure data transfer.

Weaknesses:

1. **Complex Backend Integration**: Managing multiple bank accounts requires intricate backend setup and third-party service integrations (e.g., Plaid).
2. **High Competition**: Many existing finance apps, making it hard to differentiate.

Opportunities:

1. **Financial Literacy Tools**: The app can integrate features to educate users on budgeting and saving.
2. **Expansion to Business Accounts**: Opportunity to extend services to small business owners.
3. **AI-Powered Insights:** Add features like AI-driven financial recommendations to improve decision-making in future

Threats:

1. **Data Breaches**: Constant threat of cyber-attacks on sensitive financial data.
2. **Regulatory Compliance**: Need to adhere to various banking and data privacy laws globally.
3. **Technological Changes:** Rapid changes in technology may require continuous updates and maintenance.
   1. **Project Features**

The Finance Management Application offers many of features that are designed to simplify and enhance the user’s financial management experience. These features collectively address the users’ needs and allow the users to do these tasks with convenience, security, and financial empowerment, making the application a comprehensive and secure financial management tool for the users. Features that the project has are mentioned below:

* **Real-Time Transaction Tracking**: Users can view recent transactions and account balances in real time, allowing for instant updates and decision-making.
* **Secure Fund Transfers**: The application ensures secure transactions between accounts with robust authentication and encryption protocols.
* **Multi-Bank Integration**: Users can connect and manage multiple bank accounts within a single platform, eliminating the need for multiple apps.
* **Personalized Insights**: The application provides spending and budgeting insights, helping users analyze their financial habits.
* **User Authentication and Security**: Multi-factor authentication ensures secure access, while encryption safeguards sensitive data.
  1. **User Classes and Characteristics:**

This application of Finance management has two primary user classes, each with a specific needs and characteristics:

* **Users:** Individuals who managing their personal finances. These users range from beginners, who need a simple and intuitive interface, to advanced users, who prefer detailed financial insights and seamless integration with multiple bank accounts. Their motivations include tracking transactions in real time, managing daily expenses, and achieving financial goals securely and efficiently. These users value data privacy, responsive interfaces, and minimal learning curves, making the application's ease of use and strong security features critical.
* **Administrators or Support Staff:** They are the backend users who is responsible for the system's smooth operation and user support. They are used to give a system a maintenance, to give users support, and do performance monitoring. They are proficient in application architecture, to do system debugging, and to manage database.
* **External System (Bank APIs):** The external system is the integrated system that are in the application to provide real-time financial data. They act as a bridge between the user who is using the application and their bank account. They are used to secure communication and to retrieve data accurately.
  1. **Design and Implementation Constraints**

Design and implementation of the Finance Management Application are monitor by several constraints to ensure the system that operates efficiently, securely, and under the defined scope. These constraints include hardware requirements, software interface, various tools and technologies, and security protocols standards:

* **Hardware Constraints:** Application should perform optimally on any device that has limited computational power, such as the old version of smartphones or low-end desktops.
* **Software Constraints:** This system relies on external APIs, like Plaid in our project, for bank integrations. If there are any changes or outages occurring in this APIs can affect the application functionality.
* **Development Tools:** This application will use tools like React.js, Node.js, and Supabase. These modern-day technologies help to impose their own constraints, such as language-specific limitations and dependency management.
  1. **Design Diagrams**
* **Use-Case Diagram:** Illustrates interactions between the user and the application's primary features, such as User Authentication, generating Financial Insights, Multi-Bank Integration, Transfer Fund, Track Transactions in Real-time, and View Account. This diagram ensures all user actions and Administrator action are accounted for and aligned with system functionalities.

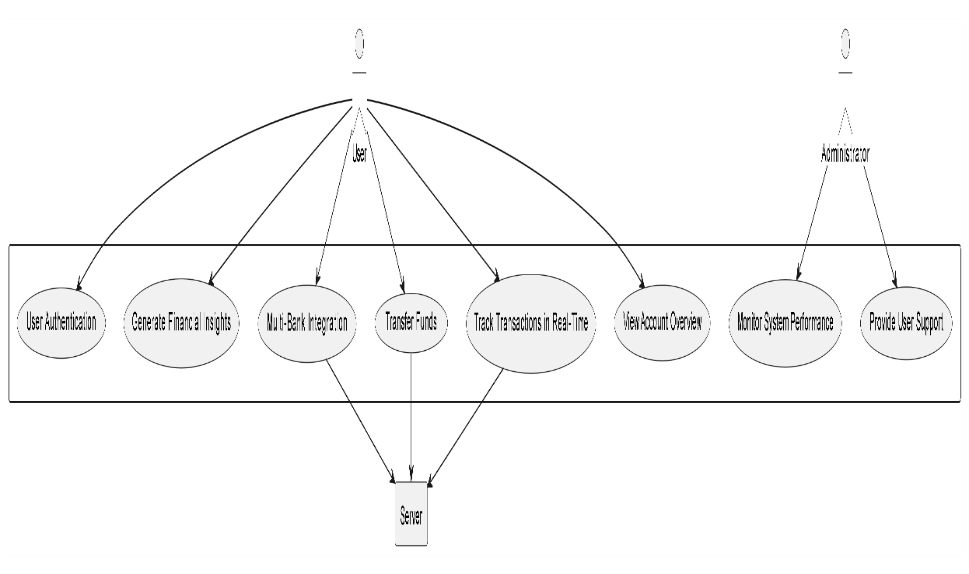
****

Figure 1: Use Case Diagram

* **Class Diagram:** Represents the relationships between core components of the system, including User, Account, application, and Transaction entities. Each class includes attributes and methods that define its behavior, such as view Balance () for the Account class or initiate Transaction () for the Transaction class.

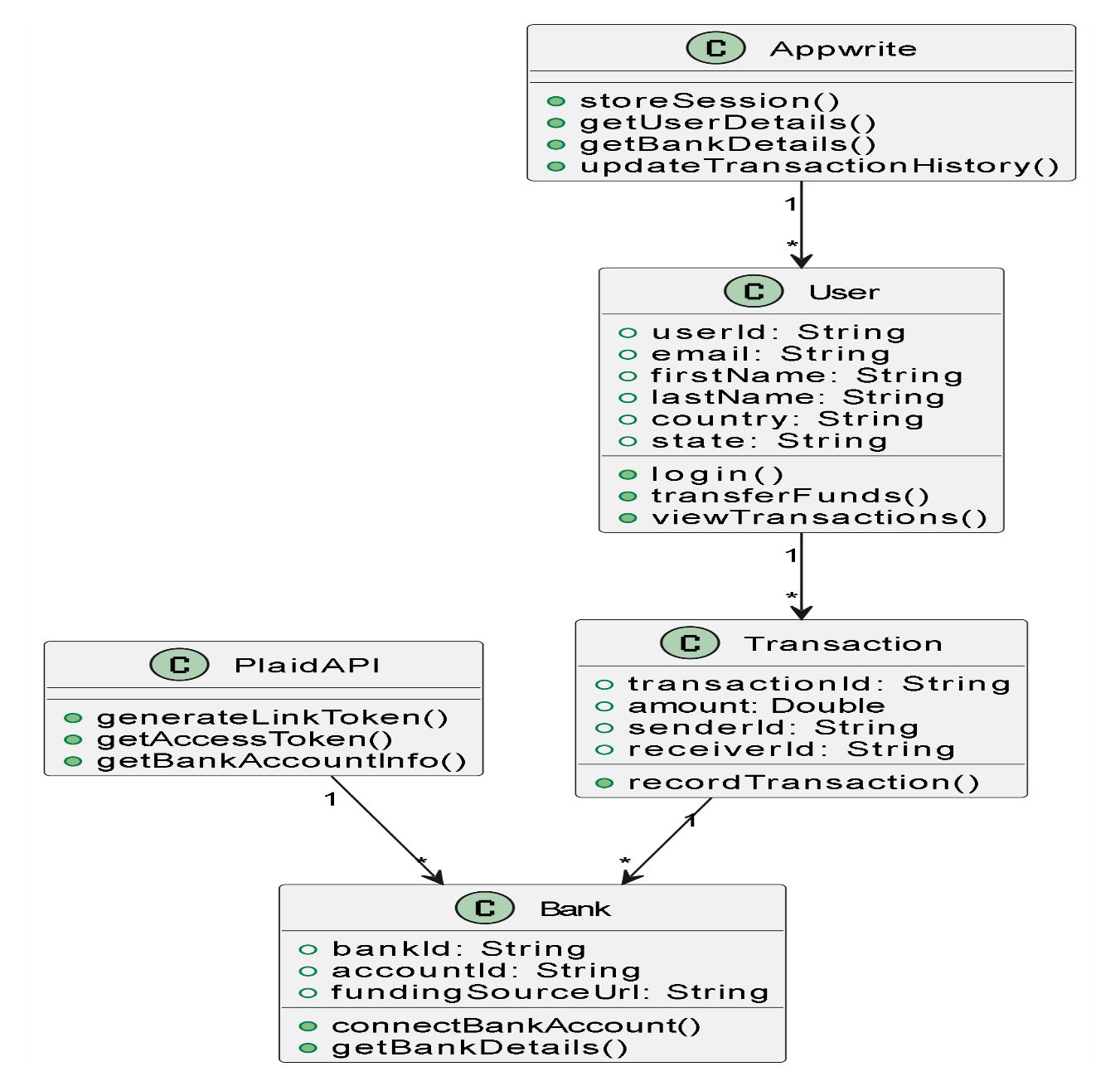


Figure 2: Class Diagram

* **Activity Diagram:** Maps out workflows like logging in, authenticating users, and executing fund transfers. It provides a step-by-step view of how users interact with the application.

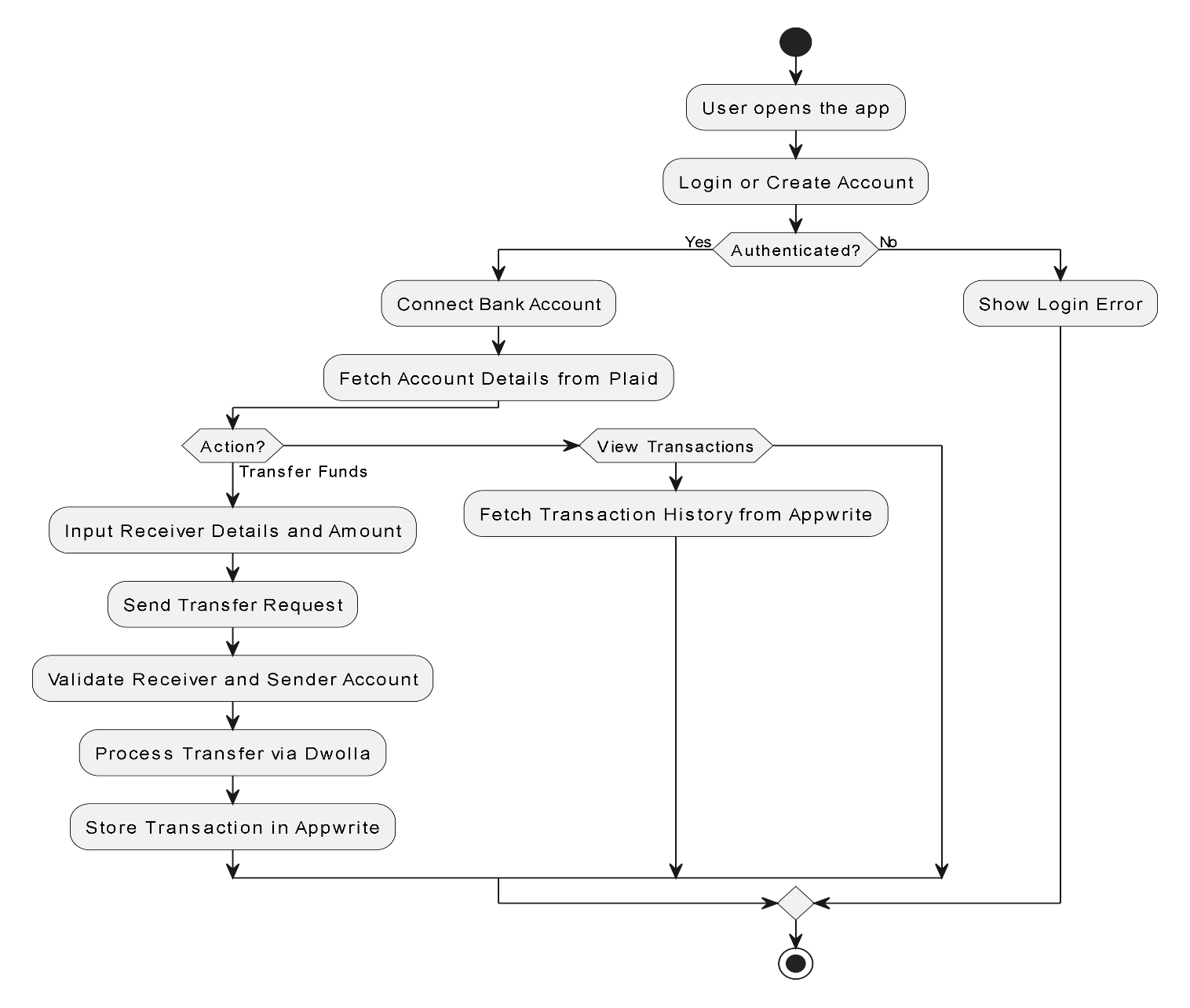


Figure 3: Activity Diagram

* **Sequence Diagram:** The sequence of interactions happing between the user interface, backend, and external APIs during a transaction process, ensuring synchronization across components.

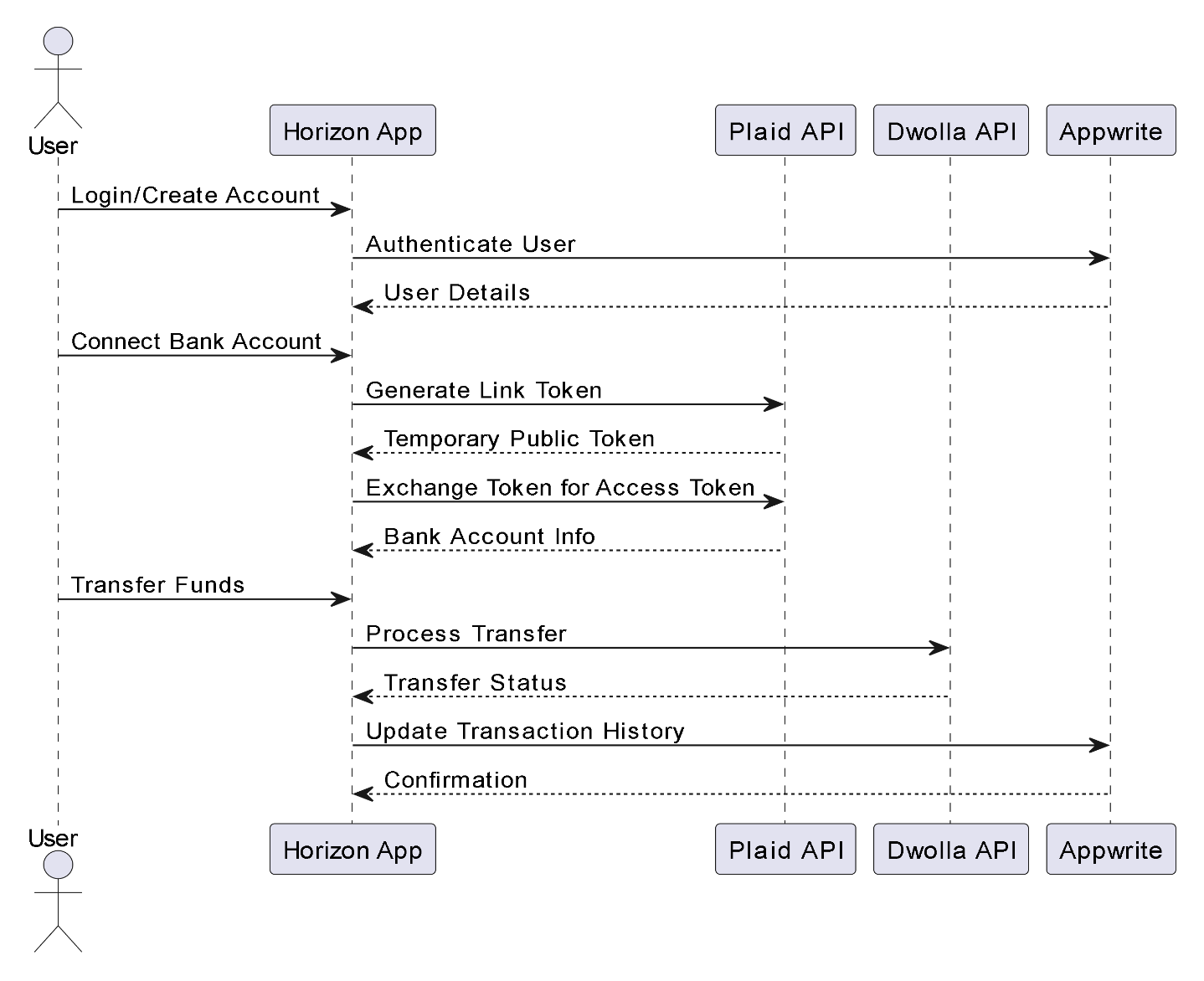


Figure 4: Sequence Diagram

* **Data Flow Diagram (DFD):** Demonstrates the movement of data, starting from user input (e.g., fund transfer request) to data storage in the database and retrieval via APIs.

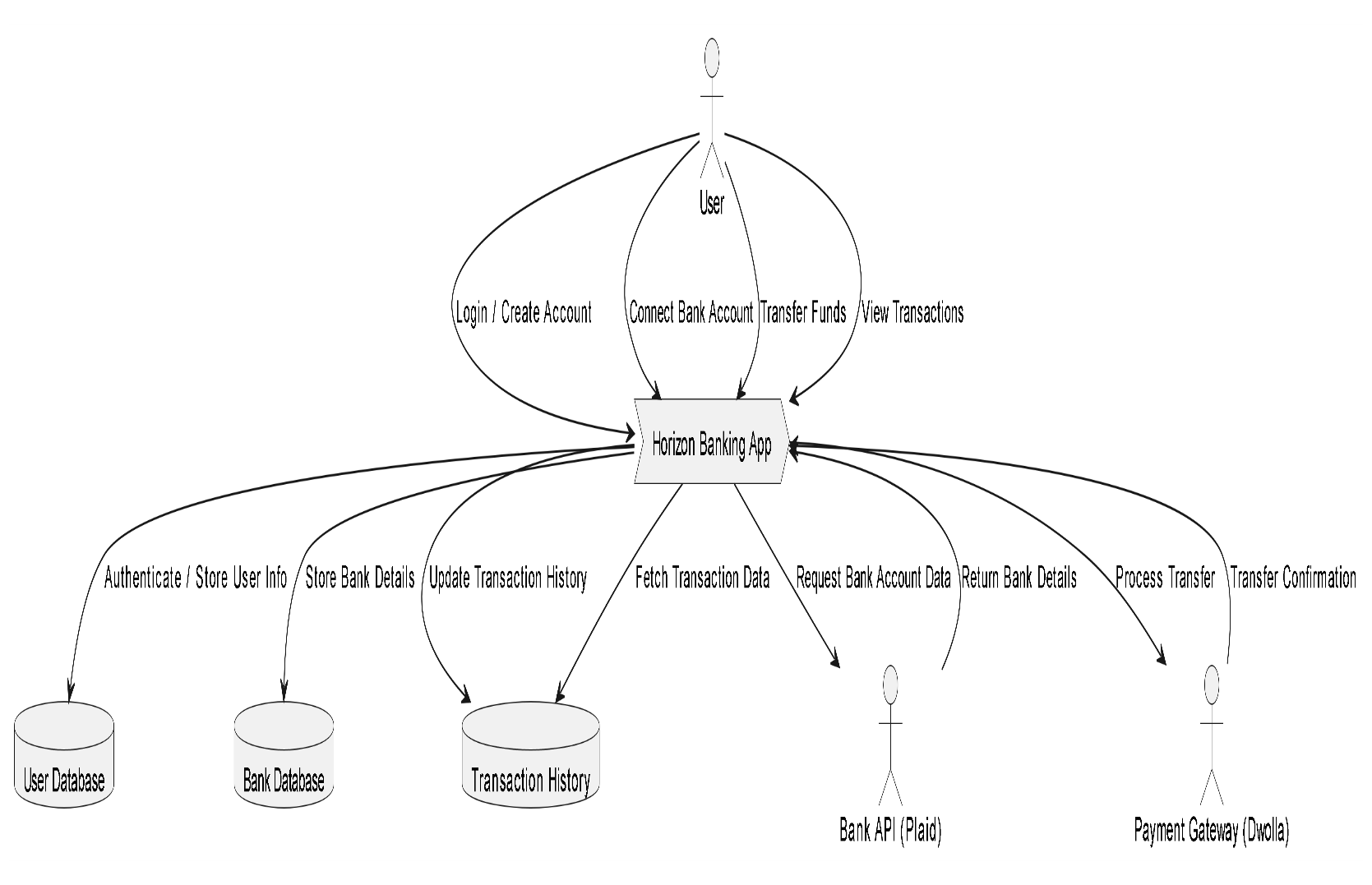


Figure 5: Data Flow Diagram

* **State Diagram:** Shows the various states of a transaction, when the user initiation to processing, and eventual completion or failure, ensuring that the system handles all possible scenarios.,

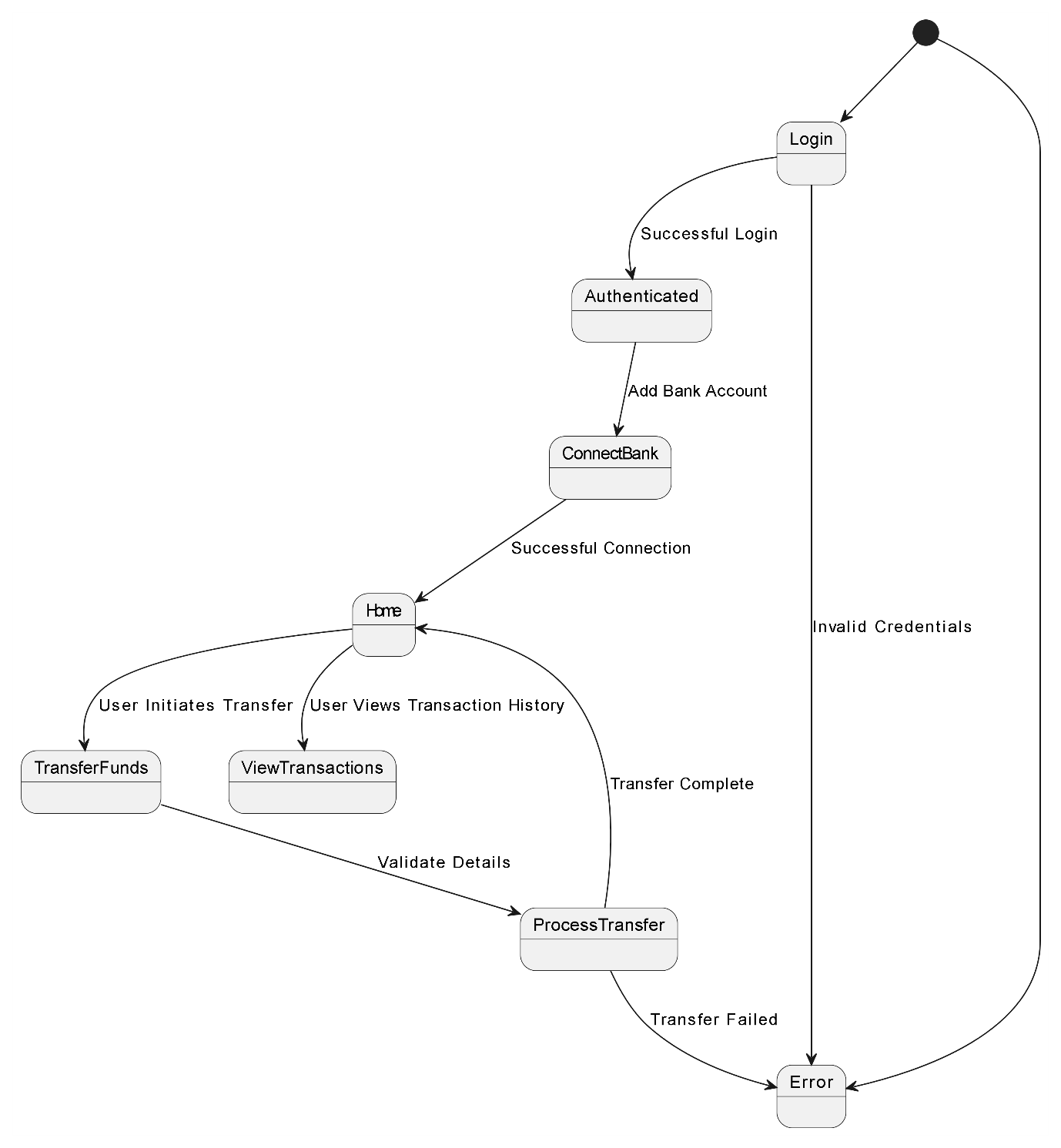


Figure 6: State Diagram

* 1. **Assumptions and Dependencies**

Assumptions:

* Users will have a stable access to the internet, stable internet access is very crucial for real-time updates and secure transaction.
* APIs that are connecting to the bank like Plaid, will provide a reliable service with giving a minimal sever downtime.
* Users using the application will follow basic security protocols, like not sharing login details or setting a weak password.

Dependencies:

* Database System: Supabase that we are using for this project for managing user data, including the transaction history of the users and their account detail.
* External APIs: Plaid that is being used for multi-bank connection and for real-time data fetching.
* Backend Frameworks: Node.js and Appwrite for handling authentication and database interaction.

1. **SYSTEM REQUIREMENTS**

**3.1 User Interface**

The user interface (UI) of the Finance Management Application is essential for facilitating intuitive and seamless user interactions. It includes:

1. **Account Management Interface**:

* Components for creating user accounts, linking existing accounts, and managing user profiles (name, email, etc.).
* Integration of Plaid Link to securely connect users' bank accounts.

1. **Bank Dashboard**:

* Displays bank account information retrieved through Plaid, including account balances, transaction history, and funding sources.

1. **Transaction Processing Page**:

* A form-based UI for initiating fund transfers, capturing sender/receiver details, and specifying amounts (with up to two decimal places).
* Displays transaction statuses and histories stored in the “Transaction” collection.

1. **Security Features**:

* Includes password management, session cookies for login sessions, and notifications for unauthorized access attempts.

**3.2 Software Interface**

The system interfaces with multiple components for secure data exchange and smooth processing.User inputs (e.g., login credentials, fund transfer details) are processed through Appwrite, validated via Plaid and Dwolla, and logged in the transaction collection.

1. **Connections Between Modules**:

* User authentication using Appwrite for creating email-password sessions and storing sessions in cookies.
* Integration of Dwolla for processing payments and creating funding sources linked to verified bank accounts via Plaid.

1. **Services Needed**:

* **Appwrite**: Manages user and bank collections, stores session cookies, and handles transactions.
* **Plaid**: Fetches bank account details and generates access tokens for verified accounts.
* **Dwolla**: Manages customer IDs, funding sources, and fund transfers between accounts.

1. **API Protocols**:

* RESTful APIs facilitate communication between the frontend and backend.
* API calls to Plaid for fetching user financial data, Dwolla for payment processing, and Appwrite for data storage and retrieval.

**3.3 Database Interface**

* **Database Management System**:
  + Appwrite is used as the backend service for managing collections of user information, bank details, and transaction histories.
* **Database Tables/Collections**:
  + **User Collection**: Stores user details like email, first name, last name, and state.
  + **Bank Collection**: Stores access tokens, account IDs, funding source URLs, and encrypted public bank account IDs.
  + **Transaction Collection**: Logs transaction details, including amounts, sender/receiver IDs, and associated bank IDs.
* **Data Access**:
  + Query-based data retrieval for generating transaction reports and updating bank account details.

**3.4 Protocols**

* **Authentication Protocols**:
  + OAuth 2.0 ensures secure authorization for banking API integrations.
  + Session cookies manage persistent user logins.
* **Encryption and Security**:
  + All data in transit is encrypted using TLS, while data at rest is secured using AES-256 encryption.
* **Synchronization Mechanisms**:
  + Tokens generated by Plaid are exchanged for access to user bank accounts, ensuring real-time synchronization of account balances and transaction statuses.

**4.NON-FUNCTIONAL REQUIREMENTS**

**4.1 Performance Requirements**

* **Latency**:
  + API calls to Plaid and Dwolla should return results within **1 second** for real-time updates.
* **Scalability**:
  + The system must handle up to **10,000 concurrent users** without performance degradation.
* **Transaction Processing**:
  + Fund transfers and associated updates to user and bank collections should occur within **5 seconds**.

**4.2 Security Requirements**

* **User Authentication**:
  + Multi-factor authentication (MFA) is implemented for critical actions like transferring funds.
* **Data Privacy**:
  + Compliance with GDPR to ensure the protection of user data.
* **Access Control**:
  + User roles and permissions to prevent unauthorized modifications to sensitive collections.

**4.3 Software Quality Attributes**

* **Adaptability**:
  + Designed to integrate new APIs or replace existing ones (e.g., Plaid or Dwolla) with minimal code changes.
* **Reliability**:
  + Maintains a system uptime of **99.9%** through redundancy and efficient error handling.
* **Usability**:
  + Features intuitive forms, responsive layouts, and detailed error messages for better user experience.

**5.OTHER REQUIREMENTS**

**Regulatory Compliance**:

* + Adherence to AML (Anti-Money Laundering) standards for transaction monitoring.

**Internationalization**:

* + Support for multiple currencies and regional settings based on user location.

**Appendix A: Glossary**

1. **Appwrite**: A backend-as-a-service platform used for user authentication, database management, and file storage.
2. **Dwolla**: A payment processing service for creating funding sources and handling secure money transfers.
3. **Plaid**: A financial data aggregator that connects user bank accounts to applications.

**Appendix B: Issues List**

**Pending API Limitations**: Some Dwolla and Plaid functionalities may require additional permissions.

1. **Scalability Testing**: Performance under extreme concurrent usage scenarios is under observation.