NU Bank - Comprehensive Project Documentation

Modern Banking Platform

Team Members:

Abdelmawla Mohamed 231000150 – Malak Abdelhamid 231000786

Sara Mohamed 231001417 – Mariam Shahinn 231001745

PROJECT OVERVIEW

NU Bank is a modern digital banking platform that provides a secure and user-friendly environment for managing essential banking operations. The system allows users to create accounts, link multiple bank accounts, monitor transactions, and perform money transfers through a unified interface.

The project was developed as part of an academic software engineering course and demonstrates the application of core software engineering principles, including structured system design, clear workflows, comprehensive documentation, testing, and collaborative development.

CONTENT

Abstract

The rapid evolution of digital banking has increased the demand for secure, scalable, and user-centric financial platforms. This project presents **NU Bank**, a modern web-based banking system designed to provide users with real-time access to their financial data, seamless bank account integration, and secure money transfer services. The system is developed using **Next.js 15** and **TypeScript** for the frontend, ensuring high performance, responsiveness, and maintainability, while **Appwrite Cloud** is utilized as the backend platform to manage authentication, data storage, and real-time synchronization.

NU Bank integrates with third-party financial services to enhance functionality and reliability. **Plaid API** is used to securely connect users' bank accounts and retrieve financial data, while **Dwolla API** handles ACH-based money transfers between accounts. The system architecture follows a layered design that separates presentation, API, integration, and data layers, enabling scalability and ease of maintenance. Security is a core focus of the platform, employing **JWT-based authentication**, **password hashing**, **AES-256 encryption**, role-based access control, and HTTPS enforcement to protect sensitive user information.

The project also emphasizes software engineering best practices, including comprehensive documentation, database schema design, UML and sequence diagrams, and a structured testing strategy covering authentication, transactions, dashboard functionality, and security scenarios. Performance metrics demonstrate fast response times, real-time data updates, and high system availability. Additionally, a **CI/CD pipeline using GitHub Actions** is implemented to automate testing and ensure code quality on every update.

Overall, NU Bank demonstrates the practical application of modern web technologies, secure financial integrations, and systematic software engineering methodologies to deliver a reliable and extensible digital banking solution suitable for real-world deployment.

System Requirements Specification (SRS)

1. Functional Requirements:

FR-1: User Registration & Authentication

- Users must be able to register with their personal details (name, email, phone, password).
- Users must verify their accounts via email or OTP.
- Users must be able to log in and log out securely.
- Password recovery/reset functionality must be provided.

FR-2: User Profile Management

- Users can view and edit their personal information.
- Users can update security settings, such as passwords and 2FA.
- Users can upload a profile picture and manage contact preferences.

FR-3: Bank Account Linking

- Users can link multiple bank accounts securely.
- System validates account numbers and banking credentials before linking.
- Users can view linked accounts and account balances.

FR-4: Transaction Management

- Users can view transaction history with details (date, amount, type).
- Transactions can be filtered and sorted (by date, amount, type).
- Users can categorize transactions for budgeting purposes.

FR-5: Money Transfer

- Users can transfer money between their own accounts or to other users.
- Transfers require confirmation via PIN/OTP.
- Transfer status (pending, successful, failed) must be visible.

FR-6: Dashboard & Analytics

- Users can view a dashboard summarizing account balances and recent activity.
- System provides visual analytics (charts, graphs) for expenses, income, and account trends.

FR-7: Notifications & Alerts

- Users receive notifications for account activities (transactions, deposits, transfers).
- Alerts for unusual activities or security issues.
- Users can manage notification preferences.

FR-8: Role-Based Access Control (RBAC)

- Different roles (Admin, Bank Staff, Customer) have specific access permissions.
- Admins can manage users, accounts, and system settings.
- Bank staff can view and assist with user accounts based on assigned privileges.
- Customers have access only to their personal accounts and transactions.

2. Non-Functional Requirements

NFR-1: Security

- Data must be encrypted in transit (HTTPS) and at rest.
- Strong password policies and multi-factor authentication (MFA) are enforced.
- System must prevent unauthorized access, SQL injection, and other common attacks.

NFR-2: Performance

- API response time must be under 2 seconds for standard operations.
- Dashboard should load within 2 seconds even with multiple accounts linked.

NFR-3: Scalability

- System must support a growing number of users without degradation.
- Architecture should allow adding new features or modules easily.

NFR-4: Availability

- The system should be accessible 99.9% of the time.
- Proper error handling and fallback mechanisms should ensure minimal downtime.

NFR-5: Usability

- User interface must be intuitive, responsive, and mobile-friendly.
- Easy navigation for both technical and non-technical users.

NFR-6: Maintainability

- Code must follow clean coding standards and be well-documented.
- System updates should not disrupt existing functionality.

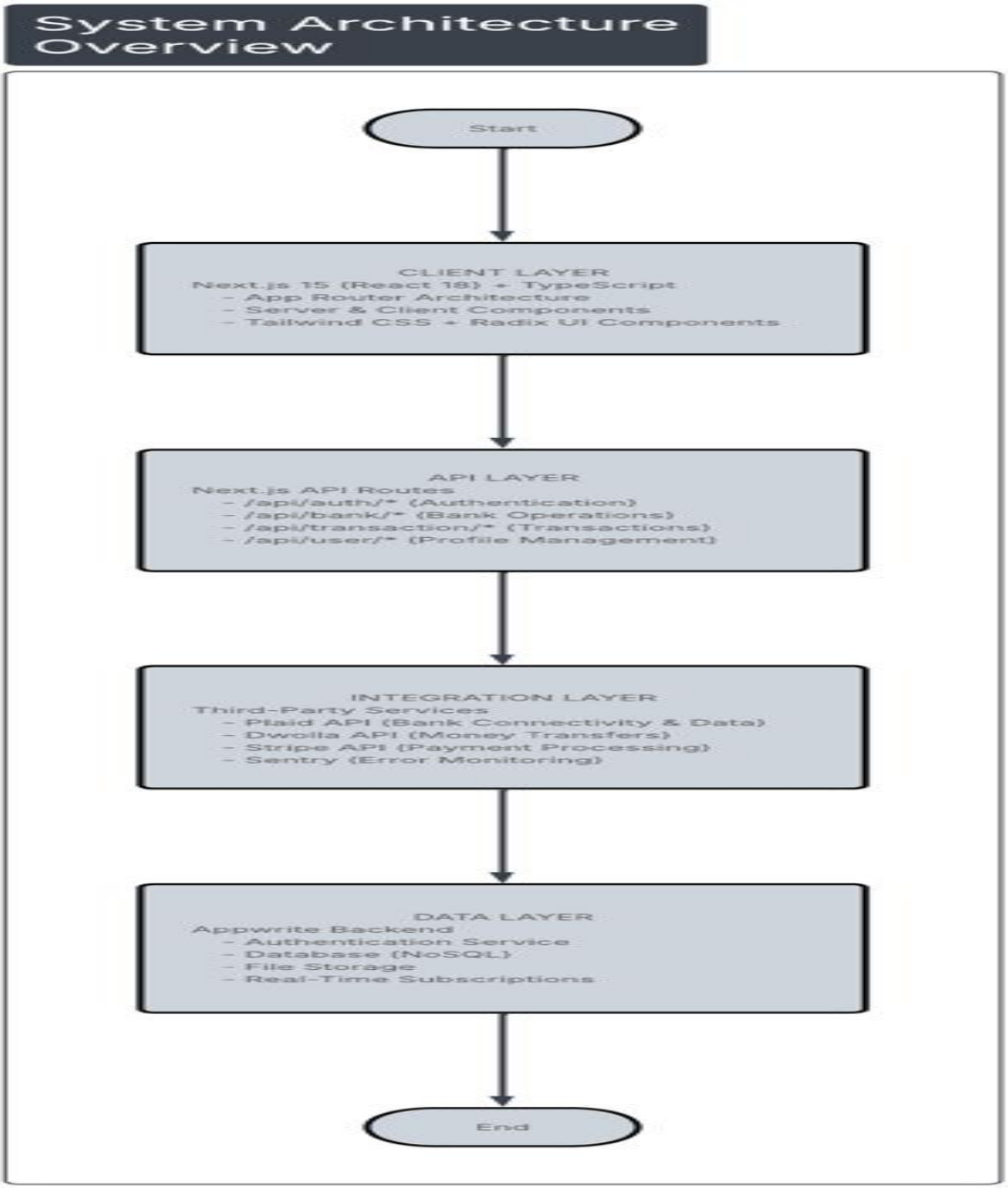
NFR-7: Reliability

- Transactions must be accurately processed and logged.
- The system must recover gracefully from failures or crashes.

SYSTEM ARCHITECTURE

The NU Bank system architecture is structured in layered components to ensure scalability, security, and maintainability. At the **client layer**, the platform leverages Next.js with React and TypeScript, offering a responsive, mobile-first interface enhanced with Tailwind CSS and Radix UI components. The API layer handles all internal requests, routing authentication, banking, transaction, and user profile operations through Next.js API routes. The integration layer manages connections with third-party services such as Plaid for bank data, Dwolla for money transfers, Stripe for payment processing, and Sentry for error monitoring. Finally, the **data layer** relies on Appwrite to provide authentication services, a NoSQL database, file storage, and real-time subscriptions. This architecture ensures smooth communication between layers, secure handling of sensitive information, and real-time updates for an optimal user experience.

For a detailed explanation and visual overview of the system, refer to the figure below.



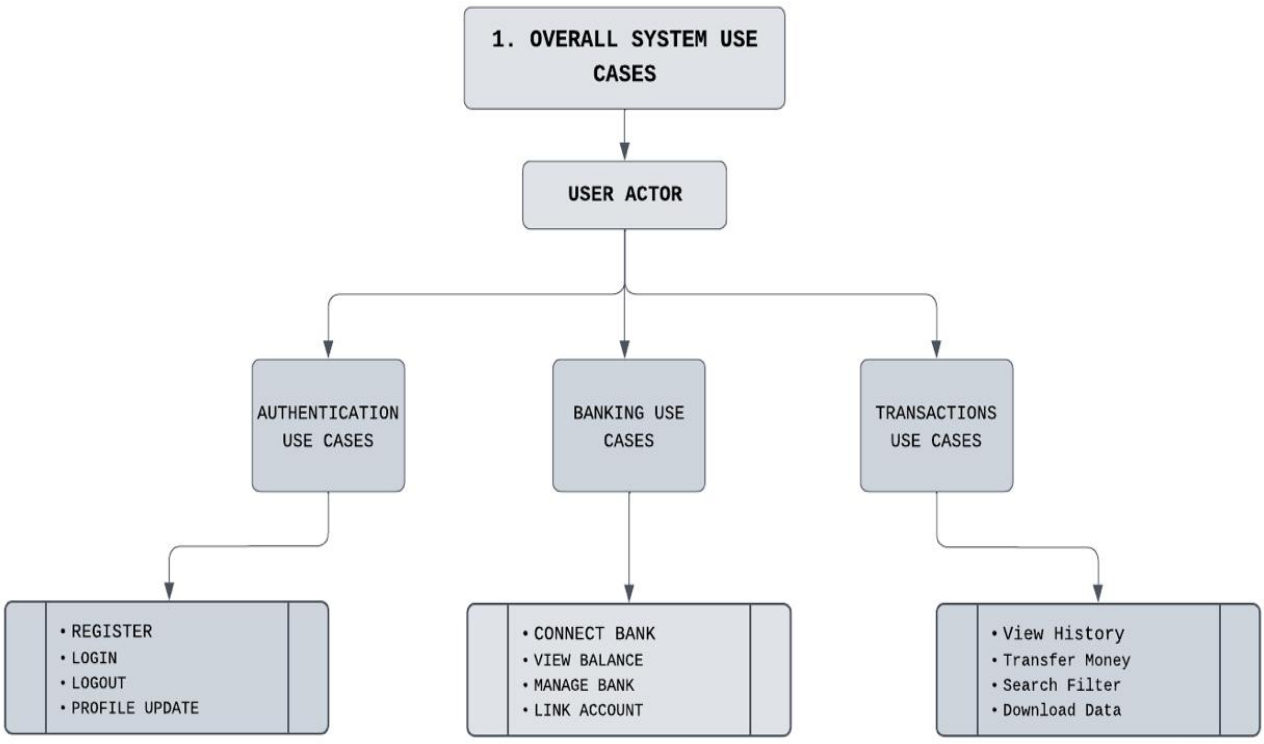
SYSTEM USE CASES & EXTERNAL INTEGRATIONS

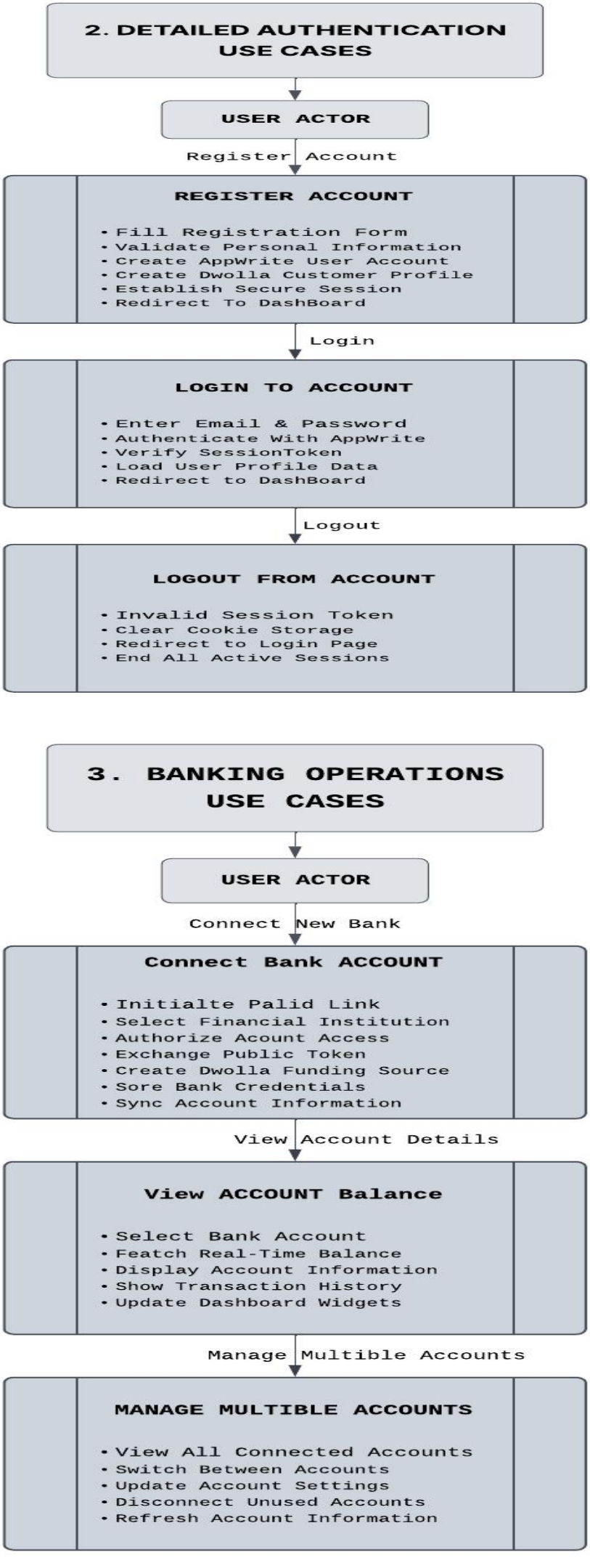
The system supports a set of core use cases that reflect the primary interactions between users and the application, ensuring that business requirements are met efficiently and securely. These use cases include user authentication and authorization, role-based access to system features, data creation, retrieval, updating, and deletion, as well as administrative approval and management workflows where applicable. Each use case is designed to align with the system's architecture and clearly define how different user roles interact with the system's services. In addition to internal functionality, the system integrates with external components such as databases and third-party tools or services (e.g., APIs for data exchange, authentication services, or notification mechanisms) to enhance functionality and ensure data consistency. These integrations are handled through well-defined interfaces to maintain modularity, scalability, and ease of future extension, while minimizing coupling between the core system and external dependencies

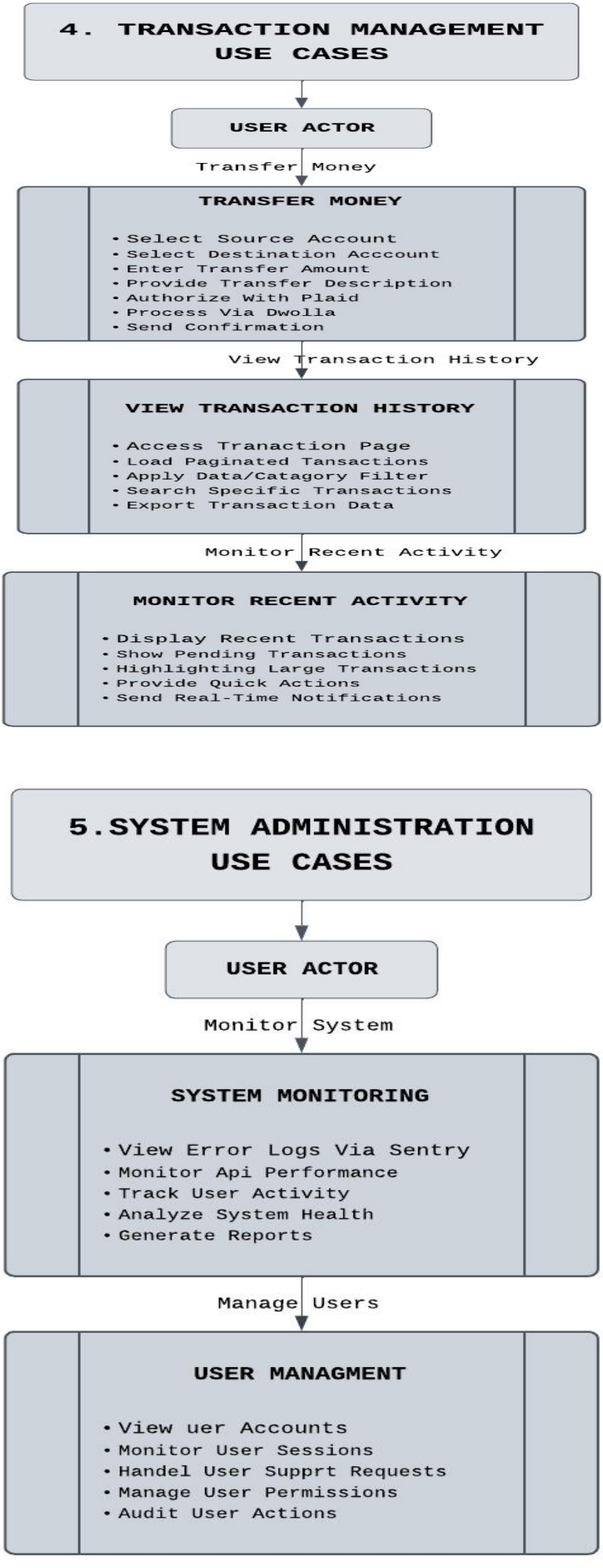
Further details and visual representation of the system are illustrated in the figures below..

Key Features:

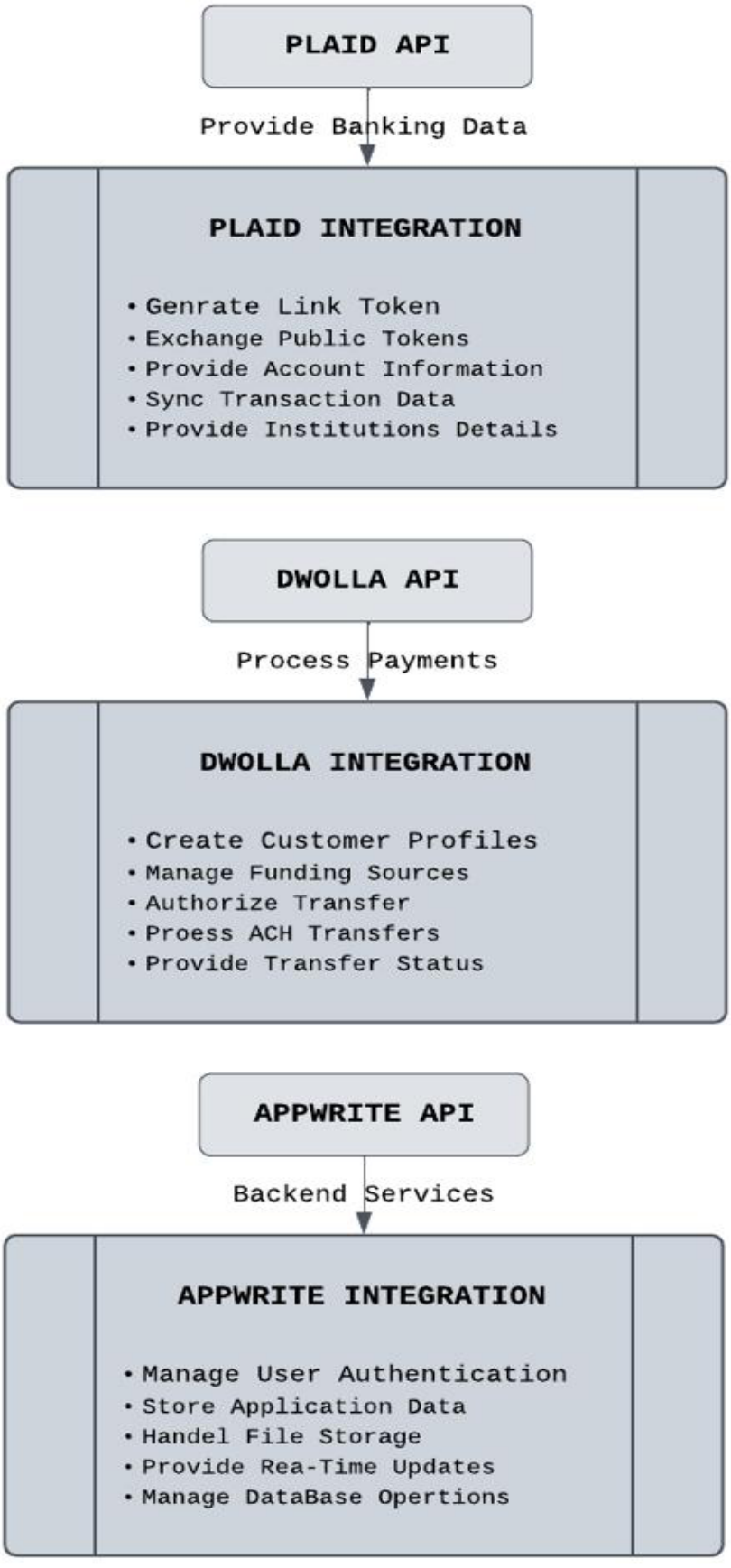
- User Authentication & Registration
- Multiple Bank Account Connections
- Transaction Management & History
- Money Transfer Between Accounts
- Interactive Dashboard with Charts & Spending Analysis
- Responsive Mobile-First Design
- Secure Banking Integrations
- Real-Time Fraud Detection & Alerts




2. DETAILED AUTHENTICATION USE CASES
USER ACTOR
Register Account
REGISTER ACCOUNT
• Fill Registration Form
• Validate Personal Information
• Create AppWrite User Account
• Create Dwolla Customer Profile
• Establish Secure Session
• Redirect To DashBoard
Login
LOGIN TO ACCOUNT
• Enter Email & Password
• Authenticate With AppWrite
• Verify SessionToken
• Load User Profile Data
• Redirect to DashBoard
Logout
LOGOUT FROM ACCOUNT
• Invalid Session Token
• Clear Cookie Storage
• Redirect to Login Page
• End All Active Sessions
3. BANKING OPERATIONS USE CASES
USER ACTOR
Connect New Bank
Connect Bank ACCOUNT
• Initialte Palid Link
• Select Financial Institution
• Authorize Acount Access
• Exchange Public Token
• Create Dwolla Funding Source
• Sore Bank Credentials
• Sync Account Information
View Account Details
View ACCOUNT Balance
• Select Bank Account
• Featch Real-Time Balance
• Display Account Information
• Show Transaction History
• Update Dashboard Widgets
Manage Multible Accounts
MANAGE MULTIBLE ACCOUNTS
• View All Connected Accounts
• Switch Between Accounts
• Update Account Settings
• Disconnect Unused Accounts
• Refresh Account Information


4. TRANSACTION MANAGEMENT USE CASES
USER ACTOR
Transfer Money
TRANSFER MONEY
• Select Source Account
• Select Destination Acccount
• Enter Transfer Amount
• Provide Transfer Description
• Authorize With Plaid
• Process Via Dwolla
• Send Confirmation
View Transaction History
VIEW TRANSACTION HISTORY
• Access Tranaction Page
• Load Paginated Tansactions
• Apply Data/Catagory Filter
• Search Specific Transactions
• Export Transaction Data
Monitor Recent Activity
MONITOR RECENT ACTIVITY
• Display Recent Transactions
• Show Pending Transactions
• Highlighting Large Transactions
• Provide Quick Actions
• Send Real-Time Notifications
5.SYSTEM ADMINISTRATION USE CASES
USER ACTOR
Monitor System
SYSTEM MONITORING
• View Error Logs Via Sentry
• Monitor Api Performance
• Track User Activity
• Analyze System Health
• Generate Reports
Manage Users
USER MANAGMENT
• View uer Accounts
• Monitor User Sessions
• Handel User Supprt Requests
• Manage User Permissions
• Audit User Actions

6.EXTERNAL SERVICE INTEGRATIONS



DATABASE SCHEMA & IMPORTANT Diagrams

Database Schema

The NU Bank database schema is designed to efficiently manage users, their linked bank accounts, and transaction records while ensuring security and relational integrity. The Users Collection serves as the core entity, storing essential personal information such as names, address, date of birth, and encrypted sensitive data like Social Security Numbers, alongside identifiers linking to authentication and Dwolla profiles. Each user can have multiple bank accounts stored in the Banks Collection, which includes account-specific details, encrypted access tokens for Plaid integration, and Dwolla funding sources to facilitate transactions. The Transactions Collection records all financial activities between accounts, capturing details such as transaction amount, category, source, and timestamps, while referencing both sender and receiver accounts. This schema provides a structured, secure, and scalable foundation for handling user data, bank integrations, and transaction management across the platform.

More detailed information and visual representations can be found in the figures below.

USERS COLLECTION		
Field	Type	Description
id	string	Primary Key
email	string	Unique
userId	string	Appwrite User ID
firstName	string	User First Name
lastName	string	User Last Name
address1	string	Address Line 1
city	string	City
state	string	State/Province
postalCode	string	ZIP/Postal Code
dateOfBirth	string	DOB
ssn	string	Encrypted Social Security Number
dwollaCustomerUrl	string	Dwolla Customer Resource URL
dwollaCustomerId	string	Dwolla Customer ID

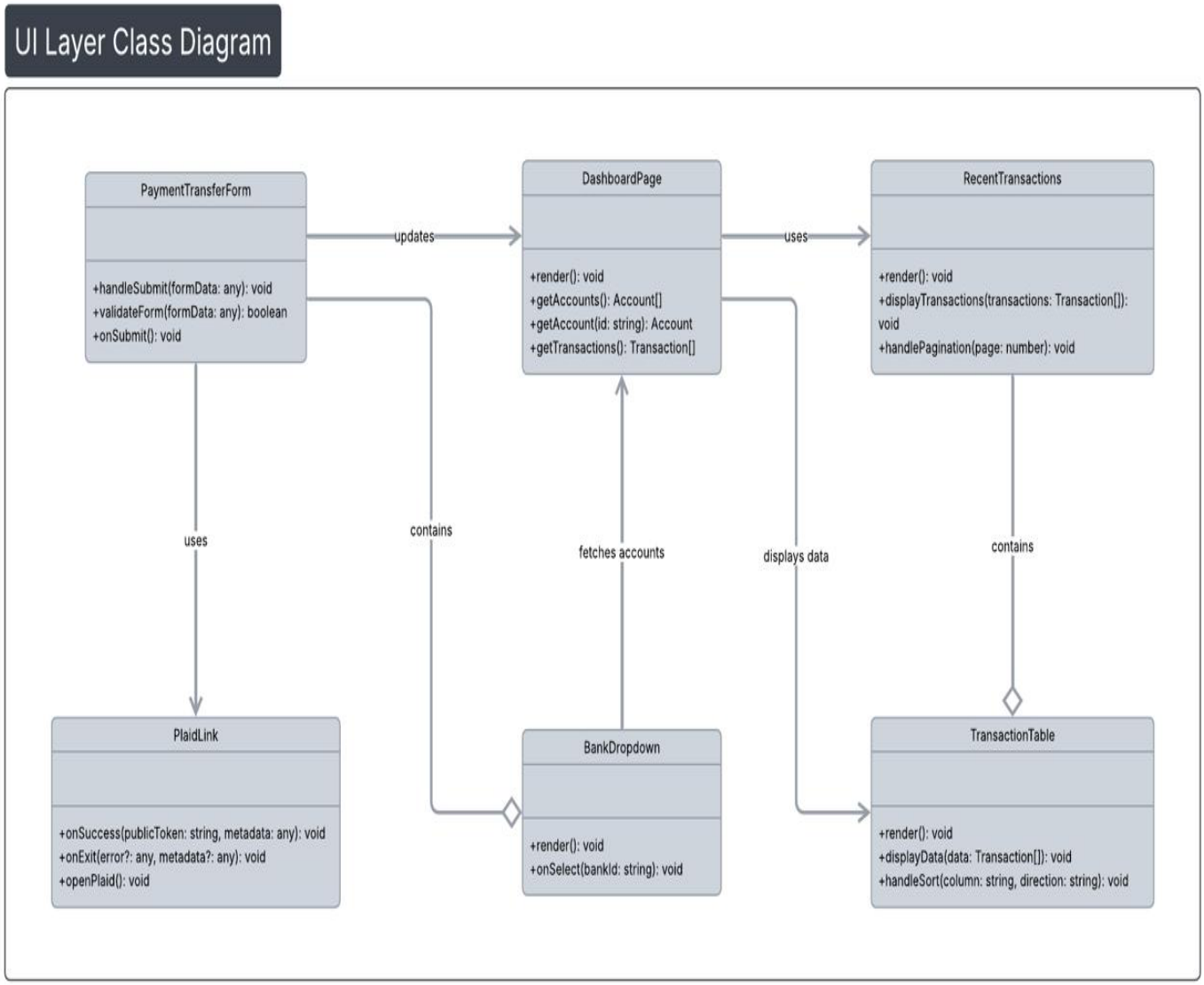
BANKS COLLECTION		
Field	**Type**	**Description**
id	string	Primary Key
userId	string	Foreign Key → Users
accountId	string	Bank Account ID
bankId	string	Plaid Bank Identifier
accessToken	string	Encrypted Plaid Token
fundingSourceUrl	string	Dwolla Funding Source URL
sharableId	string	Public Identifier

TRANSACTIONS COLLECTION		
Field	**Type**	**Description**
id	string	Primary Key
name	string	Transaction Name
amount	number	Transaction Amount
channel	string	Source (Bank/Card)
category	string	Category (Auto/Manual)
senderBankId	string	Foreign Key → Banks
receiverBankId	string	Foreign Key → Banks
$createdAt	timestamp	Record Creation Date

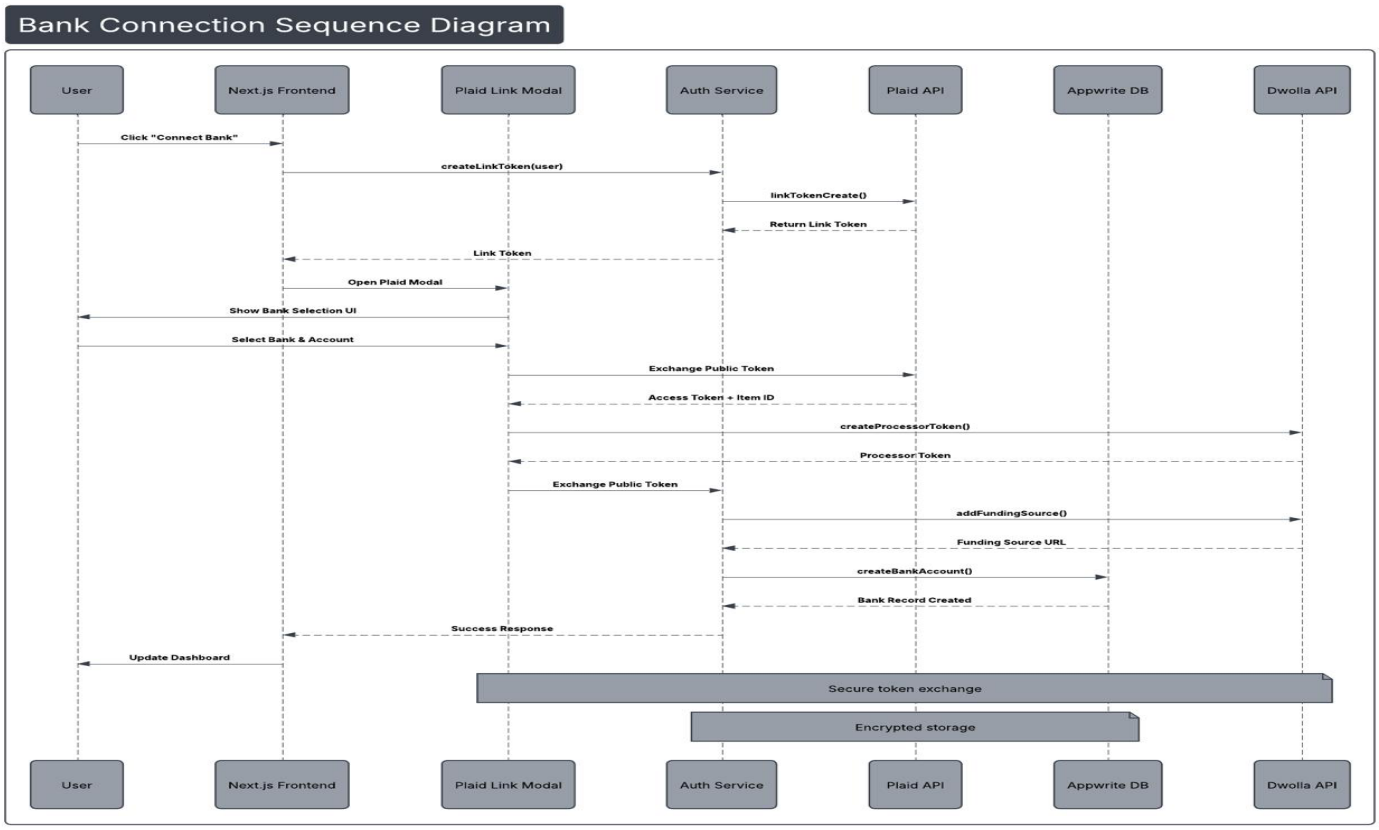
Class Diagrams & Relations

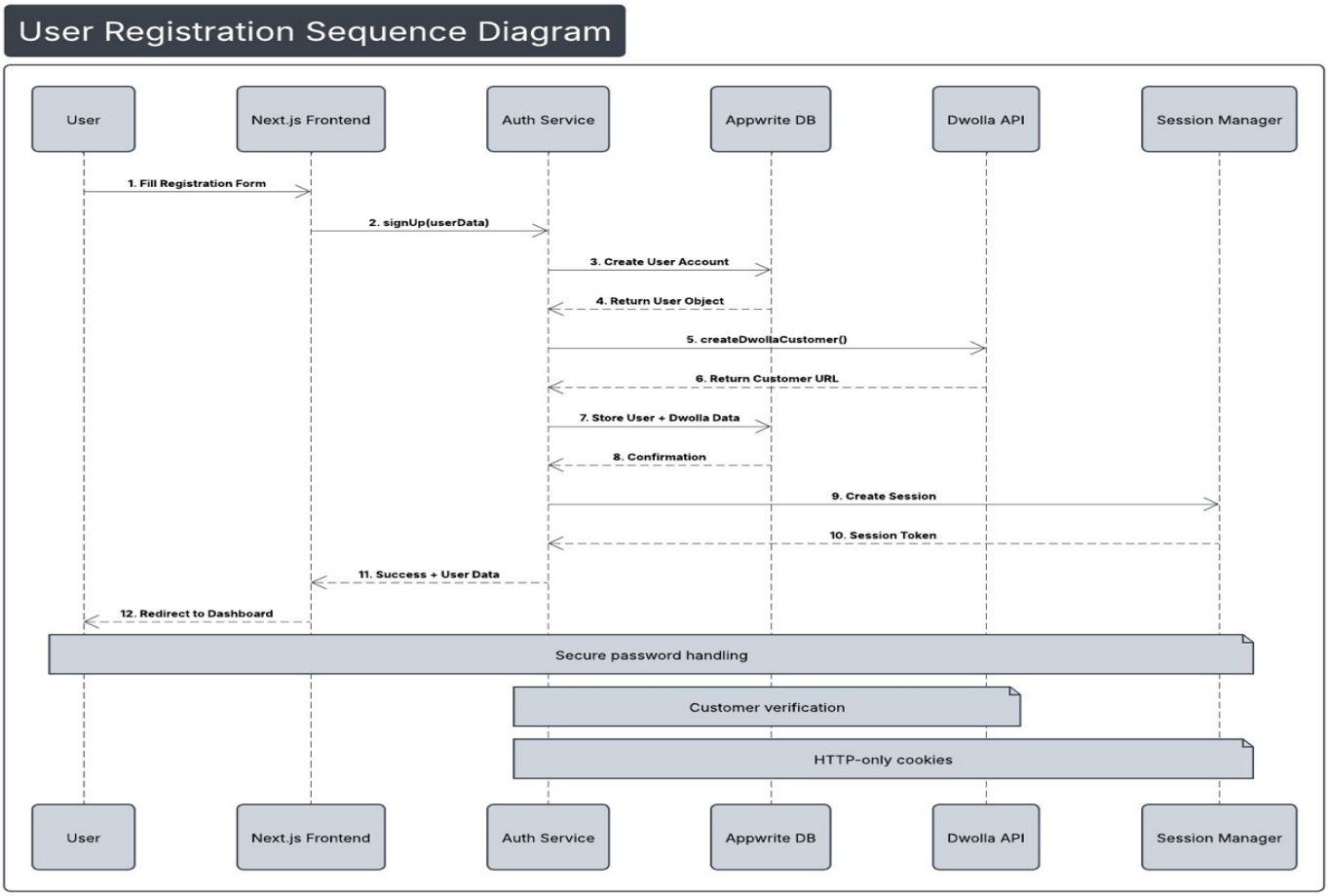
The following diagrams provide a visual representation of the system's structure and behavior. The use case diagram illustrates the interactions between users and the system, highlighting the system's main functionalities. The UML and sequence diagrams further describe the internal design, showing class relationships and the flow of interactions between system components over time. Together, these diagrams help clarify the system architecture and support a better understanding of how the system operates.

A. Class Diagrams & Relations

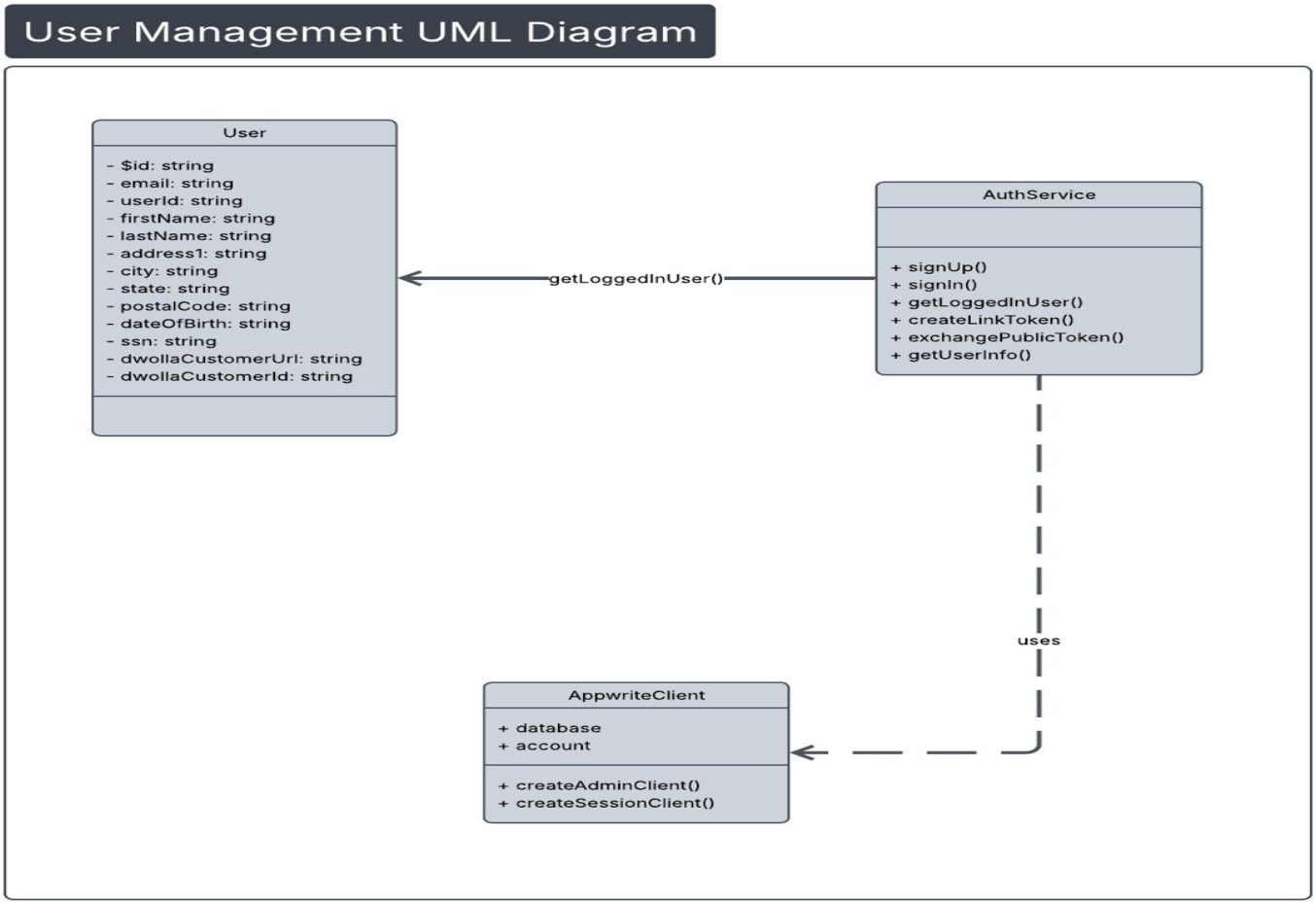


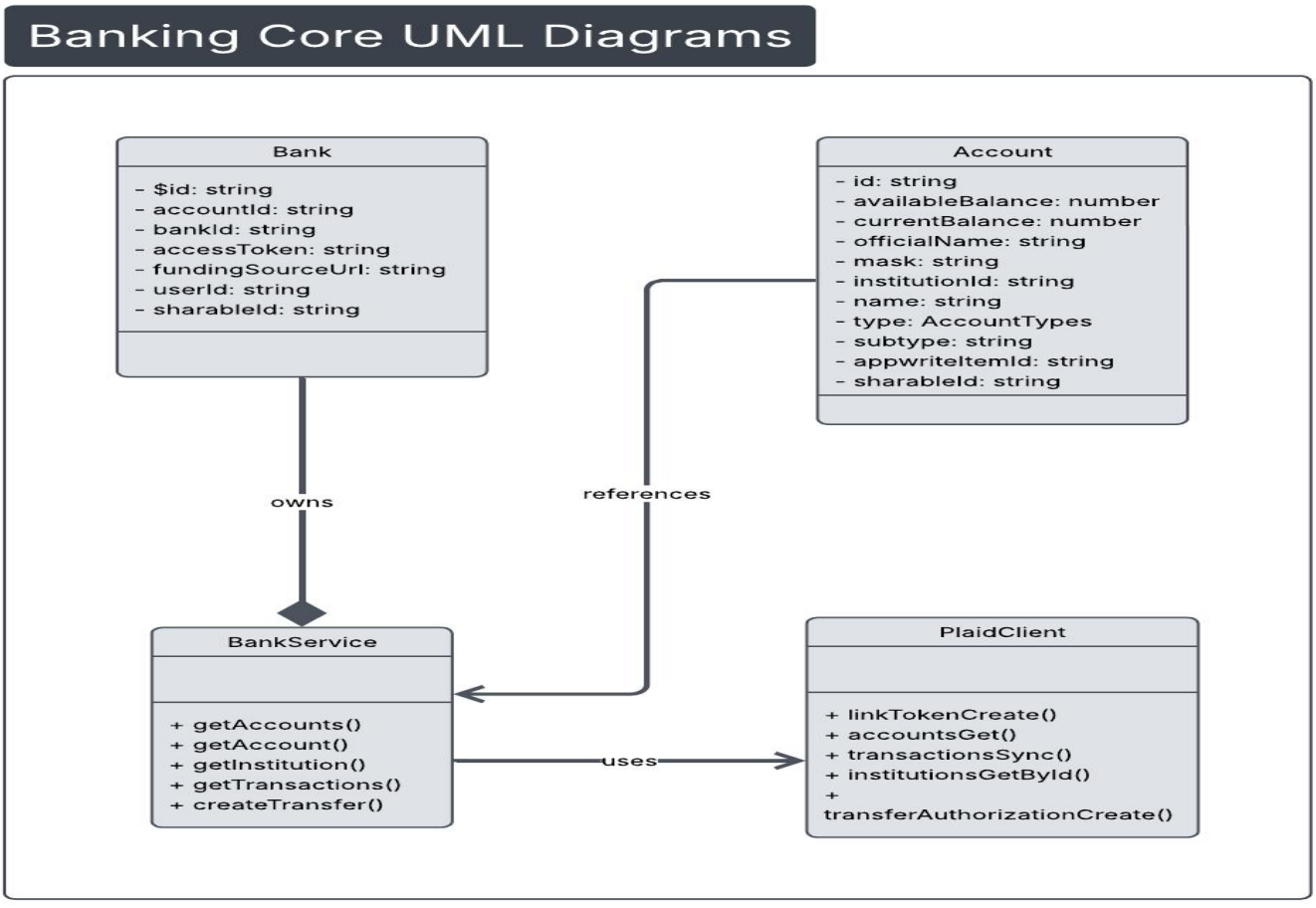
B. Sequense Diagrams



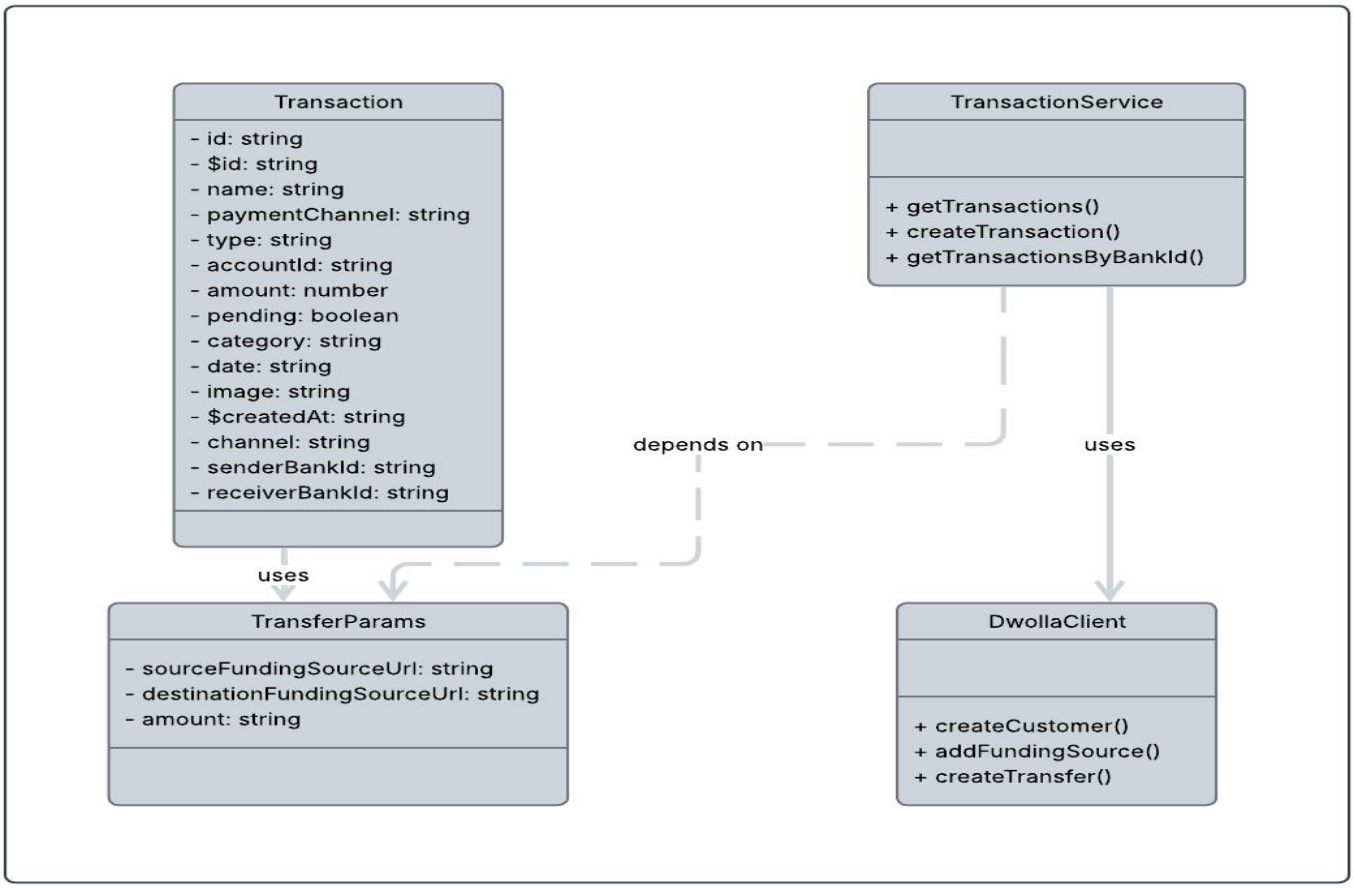


C. UML Diagrams

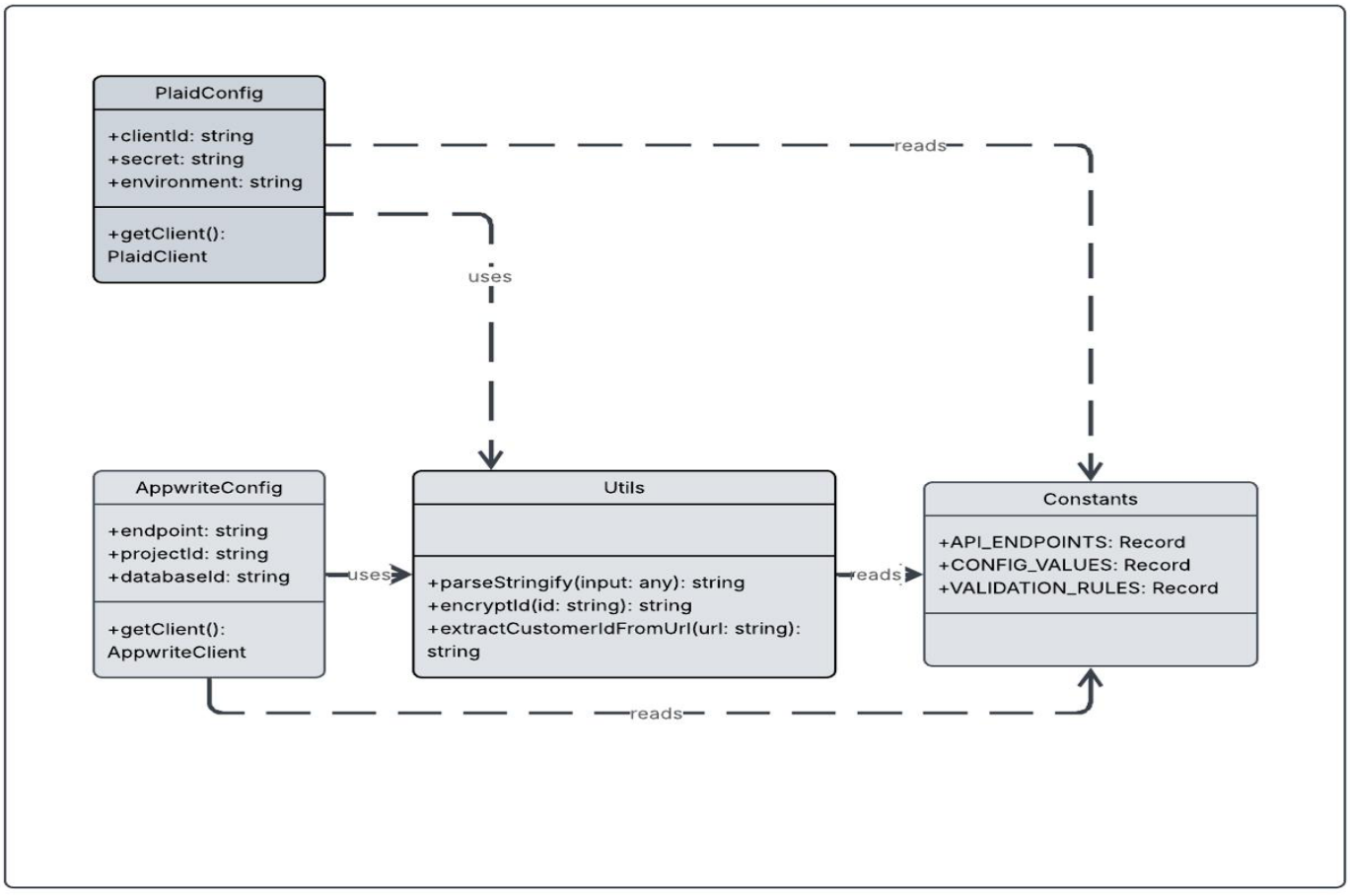




Transaction Management UML Diagram



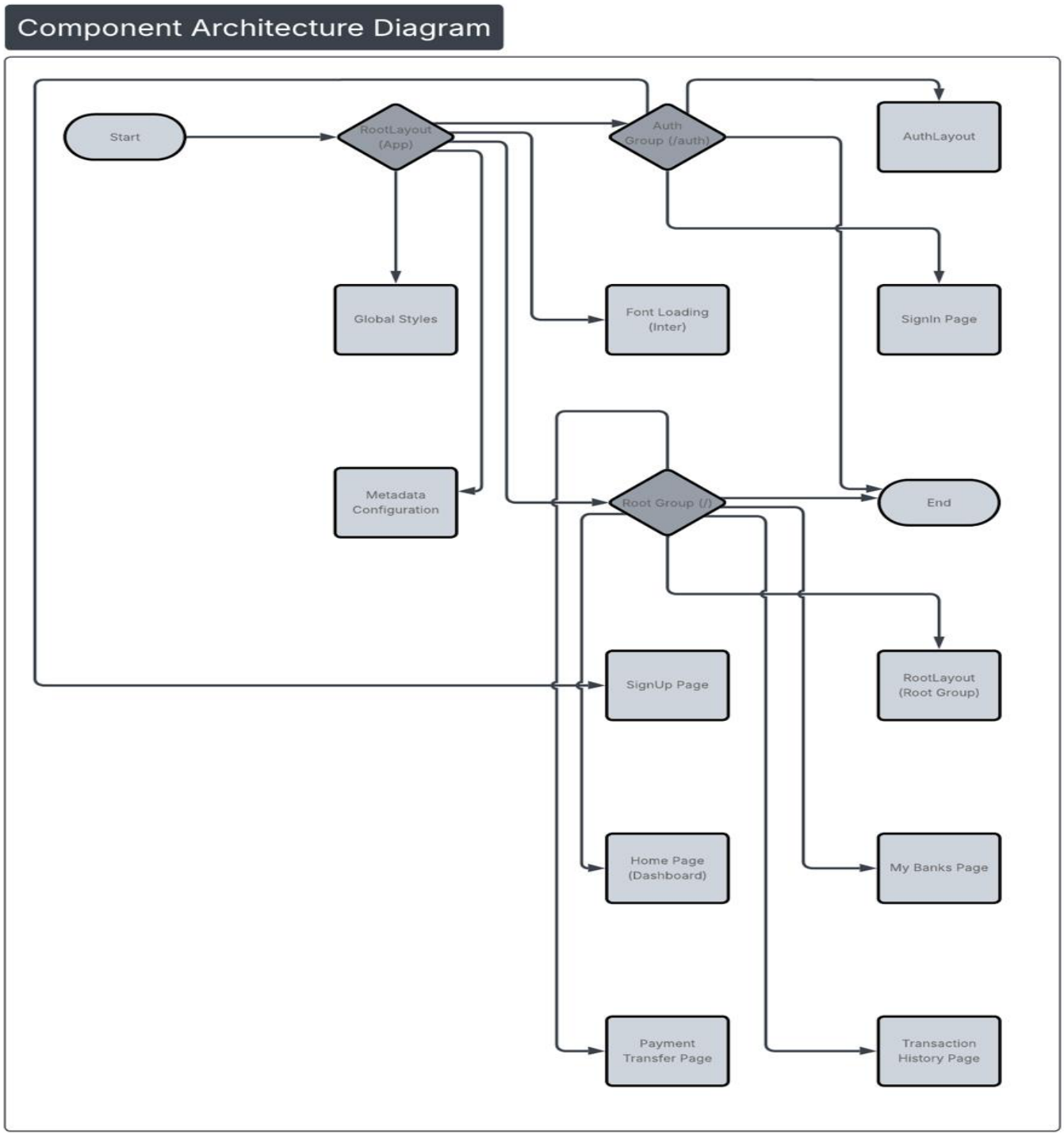
Configuration & Utilities UML Diagram



COMPONENT ARCHITECTURE

The **App Structure** of NU Bank is organized to provide a clear separation between authentication-related pages and the main application functionality. At the root level, the RootLayout handles global concerns such as styling, font loading (using the Inter font), and metadata configuration, ensuring a consistent look and feel across the app. The Auth Group (`/auth`) contains its own layout and pages dedicated to user authentication, including the SignIn and SignUp pages, which manage the login and registration processes. The Root Group (`/`) represents the main user interface after authentication, featuring a Home Dashboard for an overview of accounts and activity, a My Banks Page to manage connected bank accounts, a Payment Transfer Page for initiating transactions, and a Transaction History Page to review past transactions. This modular structure ensures separation of concerns, easier maintenance, and a scalable architecture for future features.

The figure below presents a detailed visualization of the system, offering further clarification of the described components and interactions.



COMMON COMPONENTS

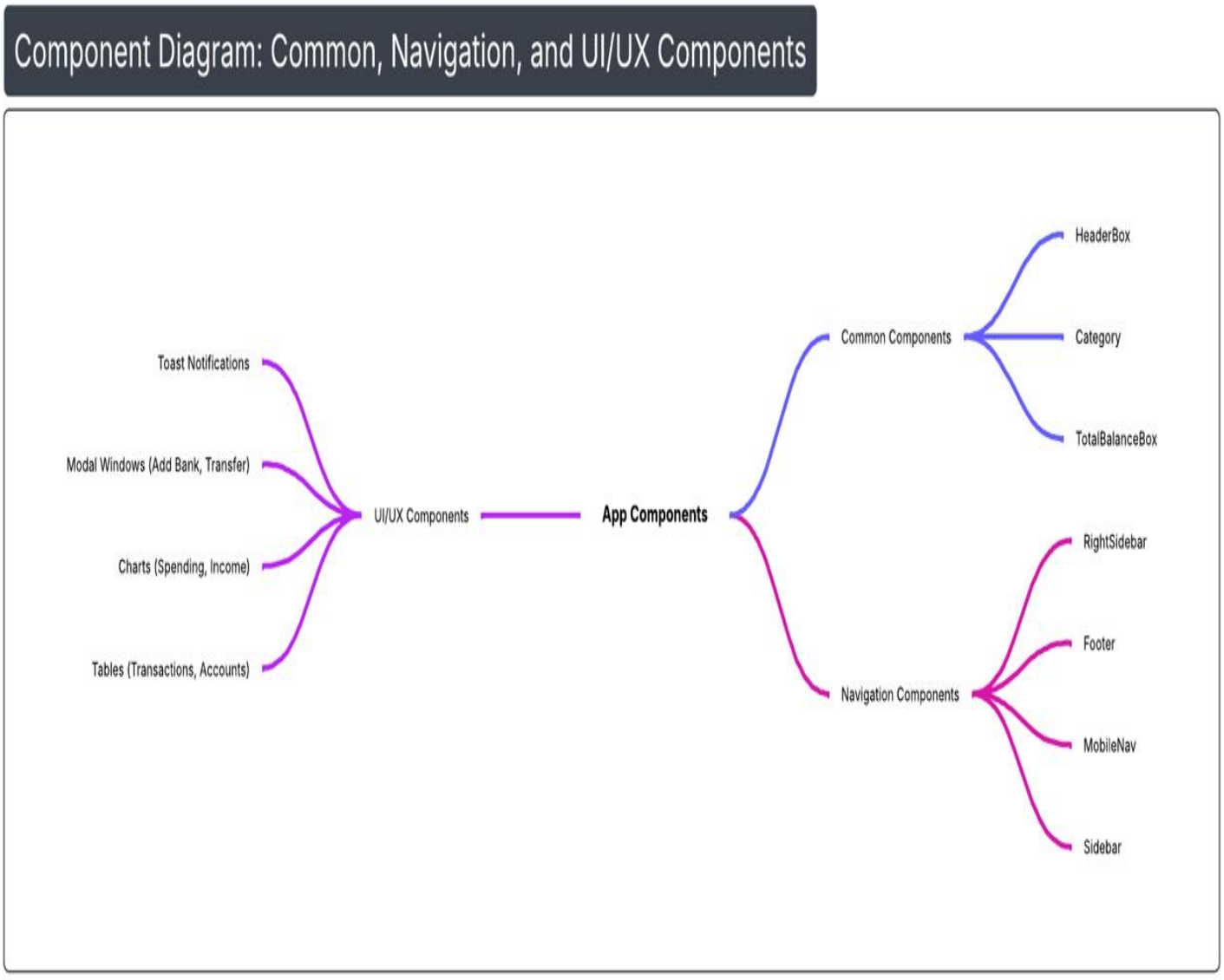
- **HeaderBox** – User header & notifications
- **TotalBalanceBox** – Displays total account balance
- **Category** – Transaction categories with icons

NAVIGATION COMPONENTS

- Sidebar
- MobileNav
- RightSidebar
- Footer

UI/UX COMPONENTS

- Modal windows (Add Bank, Transfer)
- Toast notifications
- Charts (Spending, Income)
- Tables (Transactions, Accounts)

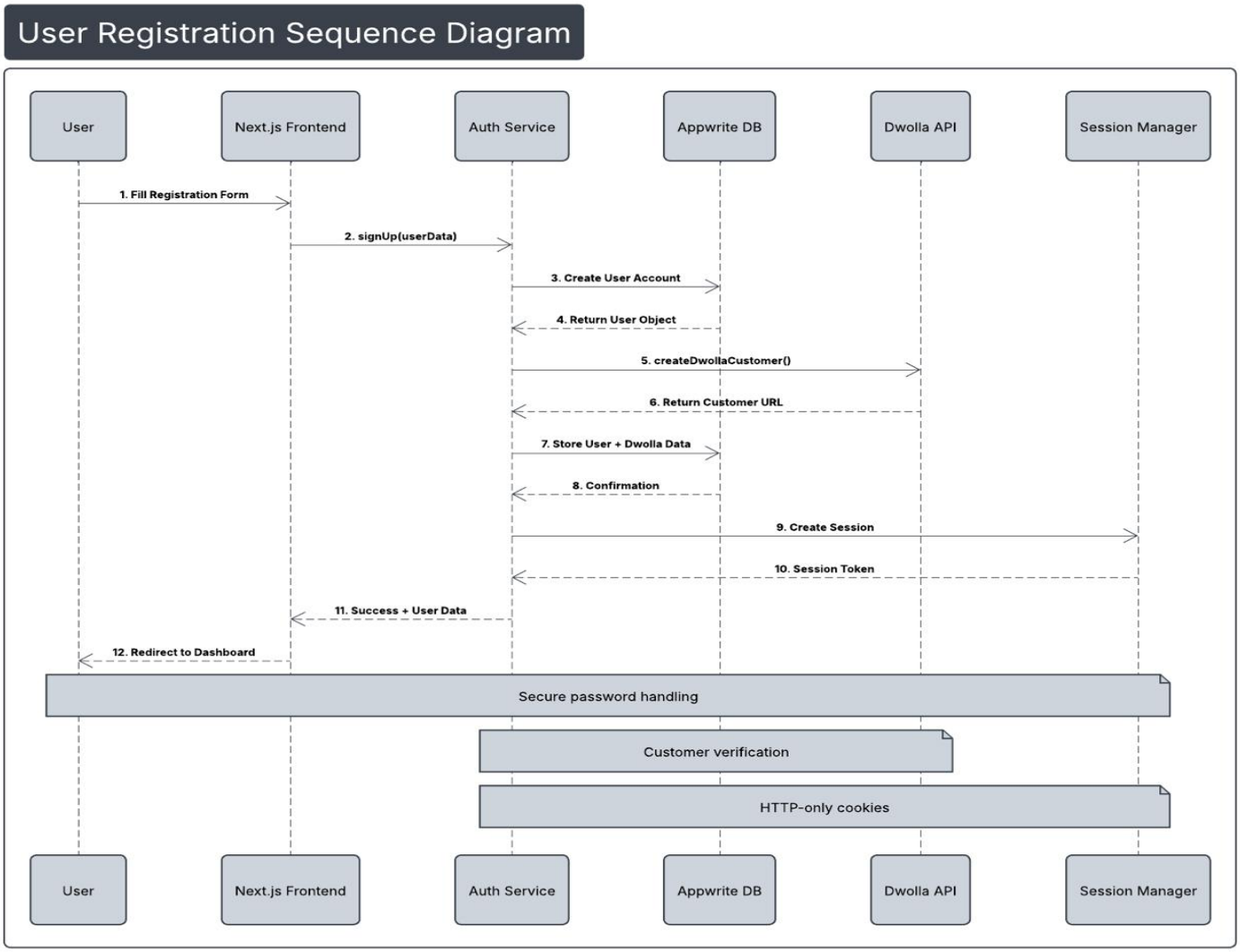


WORKFLOWS

The NU Bank workflows define a clear sequence of user interactions and system processes that ensure a seamless banking experience. The **user registration workflow** guides users from filling out personal information and verifying their email to creating secure accounts on Appwrite and Dwolla, culminating in a redirect to the dashboard. The bank connection workflow enables users to link their financial institutions through Plaid, authorize accounts, and establish Dwolla funding sources while securely storing credentials. For money transfers, the workflow covers account selection, amount entry, validation through PIN and ACH processing, and the subsequent update of balances and transaction history, followed by user notifications. Finally, the **real-time data synchronization workflow** maintains up-to-date dashboards through Appwrite subscriptions, Plaid API calls, transaction enrichment, and automatic categorization, ensuring that users always see accurate balances and analytics.

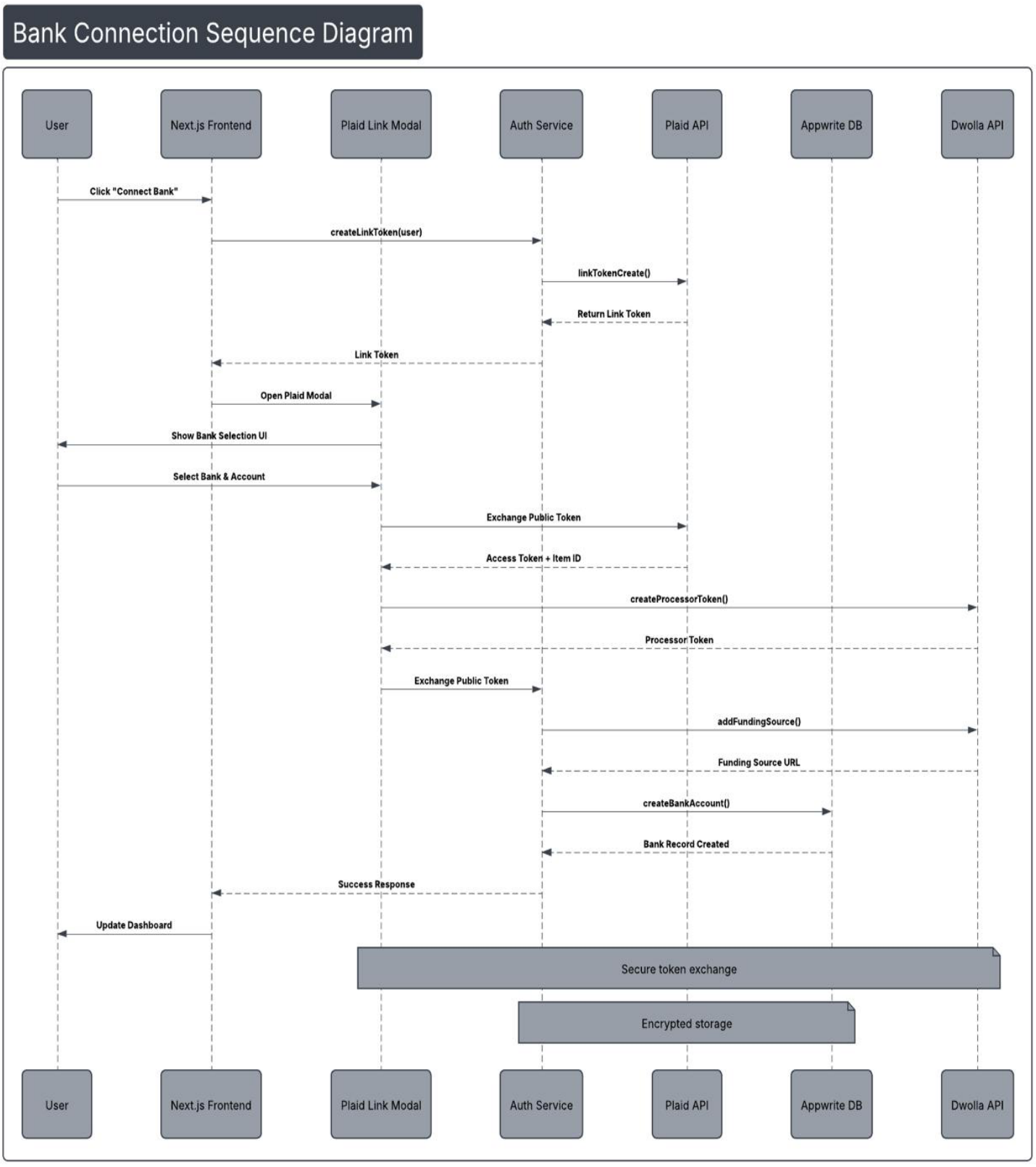
User Registration

1. Navigate to SignUp Page
2. Fill personal info and submit
3. Email verification
4. Create Appwrite user & Dwolla profile
5. Redirect to dashboard



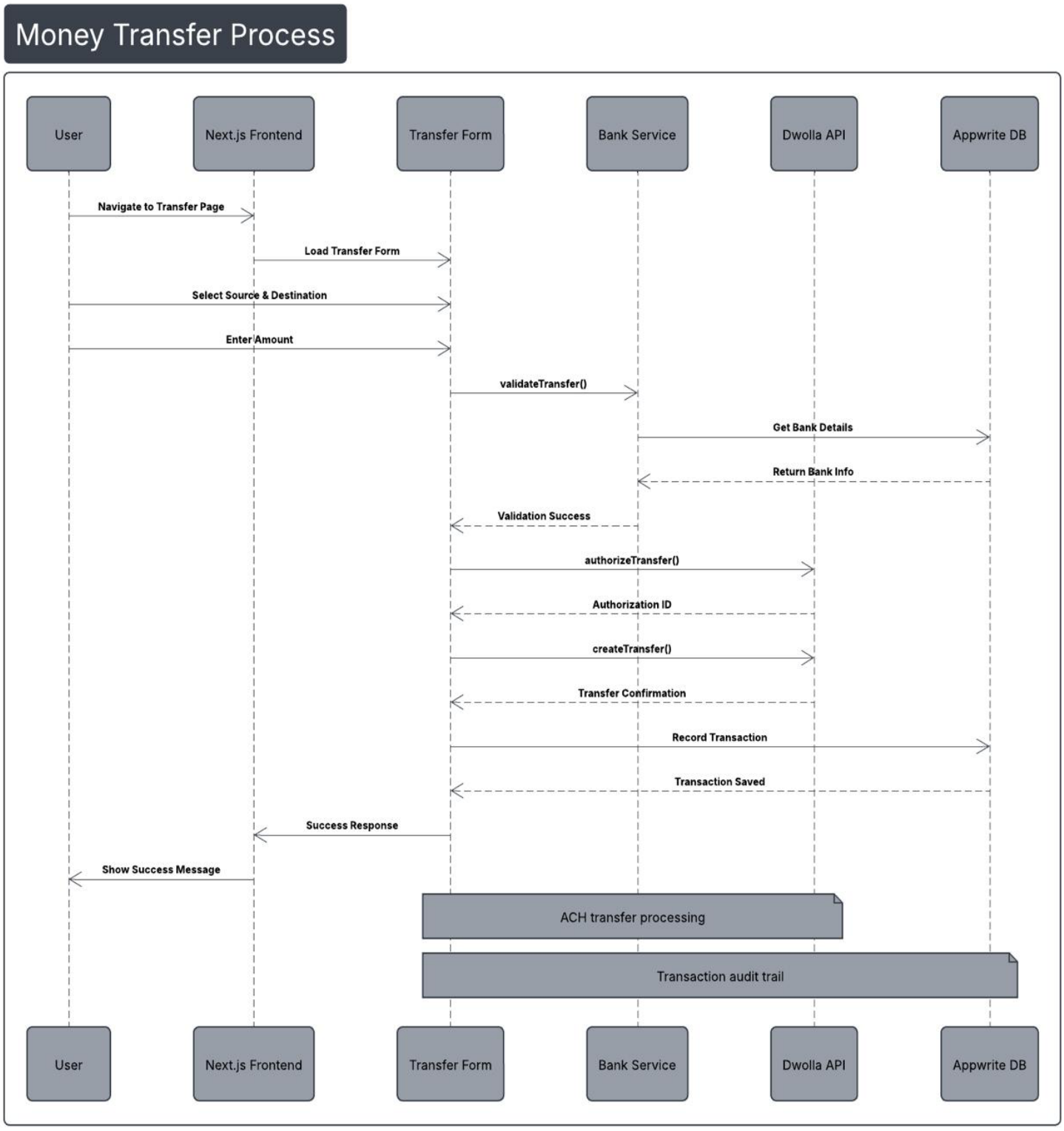
Bank Connection

1. Click "Connect Bank"
2. Select bank via Plaid Link
3. Authorize account
4. Create Dwolla funding source
5. Store encrypted credentials



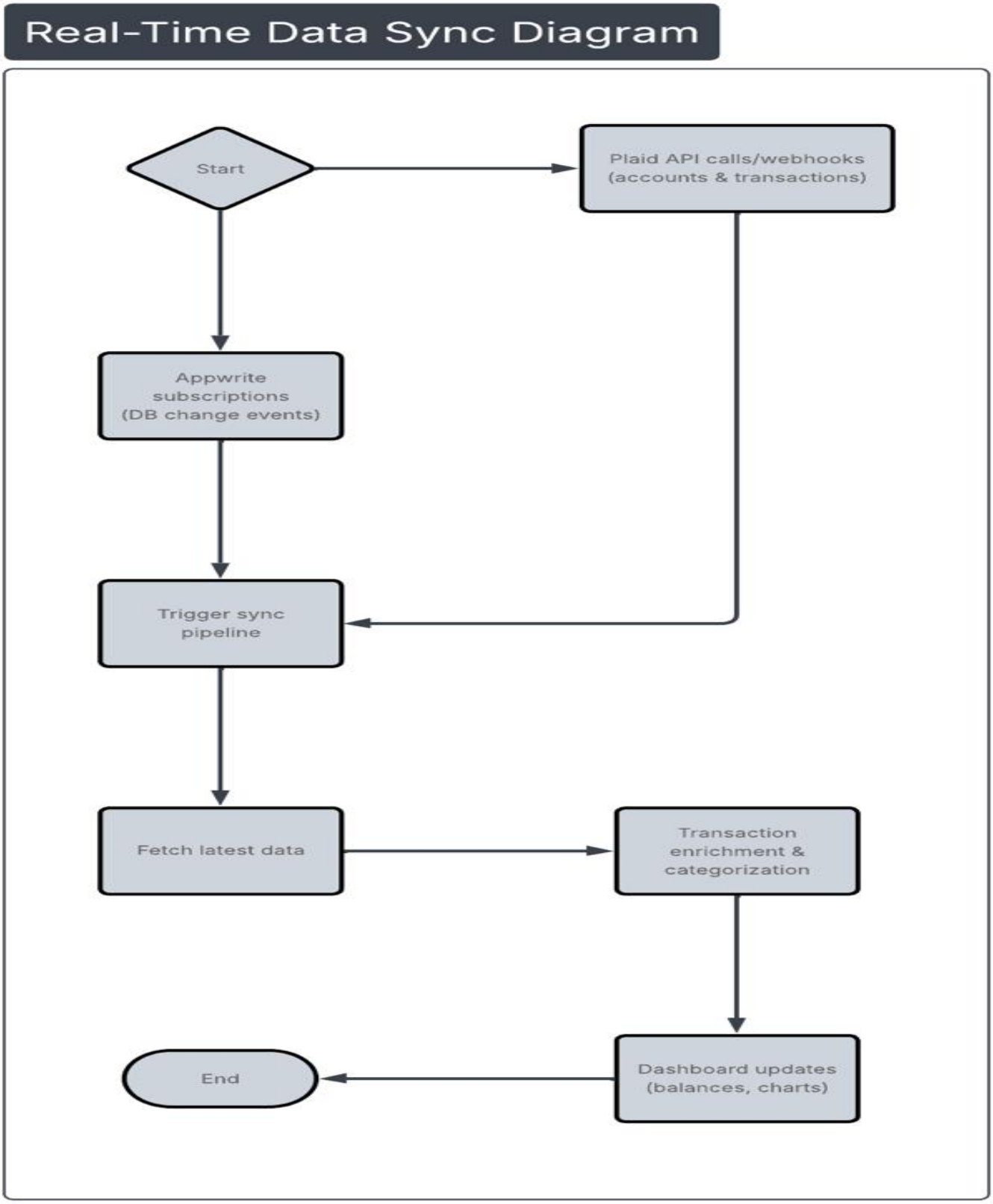
Money Transfer

1. Navigate to transfer page
2. Select accounts and amount
3. Enter description and PIN
4. Validate transfer & ACH processing
5. Update balances and transaction history
6. Notify users



Real-Time Data Sync

- Appwrite subscriptions
- Plaid API calls
- Transaction enrichment & categorization
- Dashboard updates (balances, charts)

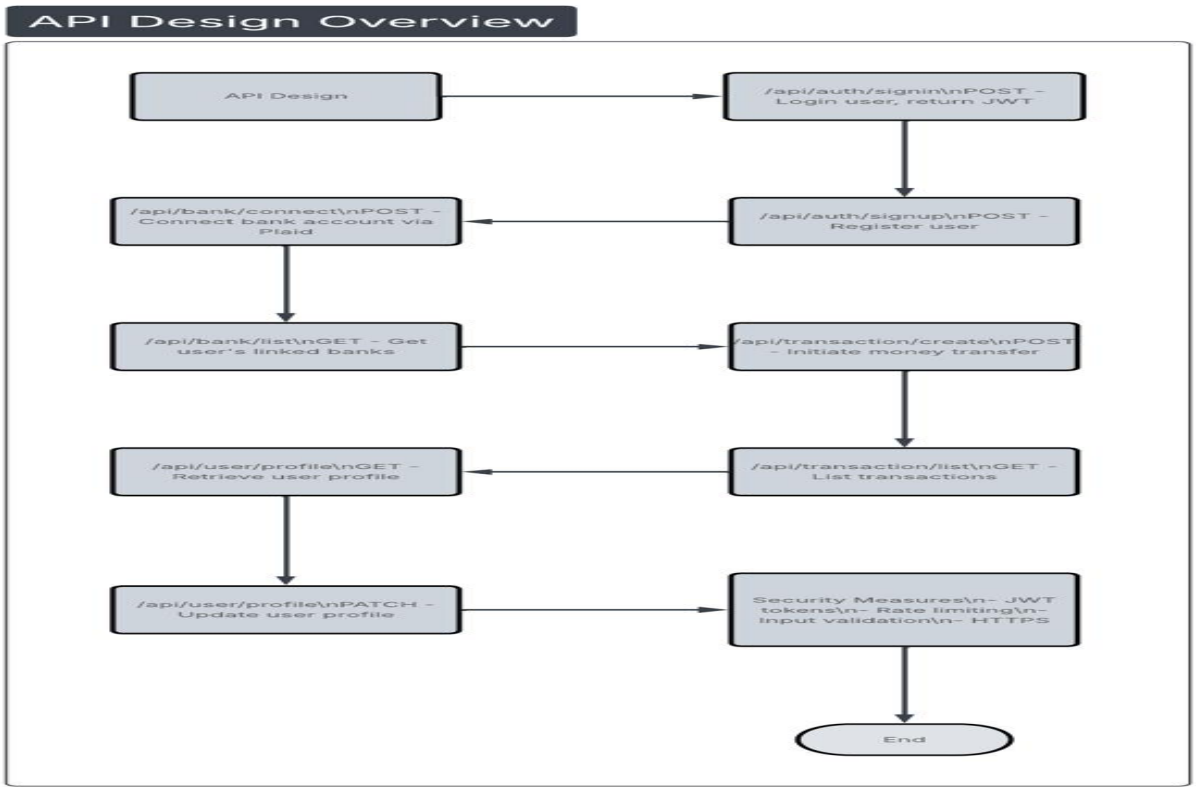


API DESIGN

The API design of NU Bank establishes a structured and secure interface for client-server communication, enabling seamless interaction between the frontend and backend services. Each endpoint serves a specific purpose: authentication endpoints (`/api/auth/signin` and `/api/auth/signup`) handle user login and registration with JWT-based sessions, while banking endpoints (`/api/bank/connect` and `/api/bank/list`) allow users to securely link and view their bank accounts via Plaid integration. Transaction endpoints (`/api/transaction/create` and `/api/transaction/list`) manage money transfers and transaction history, and user profile endpoints (`/api/user/profile`) facilitate retrieval and updates of personal information. Security is enforced through HTTPS, JWT authentication, input validation, and rate-limiting, ensuring data integrity, user privacy, and protection against unauthorized access.

For a detailed explanation and visual overview of the system, refer to the table below.

Endpoint	Method	Description
/api/auth/signin	POST	Login user, return JWT
/api/auth/signup	POST	Register user
/api/bank/connect	POST	Connect bank account via Plaid
/api/bank/list	GET	Get user's linked banks
/api/transaction/create	POST	Initiate money transfer
/api/transaction/list	GET	List transactions
/api/user/profile	GET	Retrieve user profile
/api/user/profile	PATCH	Update user profile



.SECURITY ARCHITECTURE

Authentication & Authorization

- JWT-based sessions
- Password hashing (bcrypt)
- Role-based access control (RBAC)
- Multi-factor authentication optional

Data Protection

- AES-256 encryption for sensitive data
- HTTPS enforced
- Masking sensitive fields (SSN, bank tokens)

Monitoring & Logging

- Security logs for login, transfers, and API calls
- Real-time fraud detection and alerts
- Anomaly detection and pattern recognition

Testing & Quality Assurance

Testing Strategy

The testing strategy for this system is designed to ensure reliability, correctness, and maintainability across all major components. A combination of manual **and** automated testing is used to balance thorough validation with development efficiency. Automated tests are applied where repeatability and regression prevention are critical, particularly in the backend logic and APIs, while manual testing is used to validate user experience and edge cases in the frontend. The testing scope covers frontend functionality, backend services, and integration between system components, ensuring that individual modules work correctly both in isolation and as part of the complete system. Tools such as Postman are used for API testing, while frontend and logic validation are supported by appropriate testing frameworks and manual test scenarios. This strategy fits the system well because it aligns with its layered architecture, reduces the risk of critical failures, and ensures that core functionalities behave as expected before deployment.

Test Cases

The figures below presents a detailed visualization of the system, offering further clarification of the described components.

Authentication / Login / Sign Up Test Cases				
Test ID	Feature	Test Scenario	Input	Expected Result
TC-01	Login Page	Login page loads properly	None	Email field, password field, and login button are visible
TC-02	Authentication	Login with valid credentials	Valid email & password	User is logged in and redirected to dashboard
TC-03	Authentication	Login with incorrect password	Valid email, wrong password	Error message “Invalid credentials” displayed
TC-04	Authentication	Login with empty fields	Empty form submission	Validation error messages shown
TC-05	Sign Up Page	Registration page loads properly	None	Name, email, password fields visible
TC-06	Authentication	Register with invalid email format	Invalid email	Email validation error shown
TC-07 (Added)	Authentication	Register with existing email	Email already registered	Error message “Email already exists”
TC-08 (Added)	Authentication	Password strength validation	Weak password	Password policy error displayed

Dashboard & UI Test Cases			
Test ID	Feature	Test Scenario	Expected Result
TC-09	Dashboard	Dashboard loads after login	Account summary, balances, and charts displayed
TC-10	Navigation	Sidebar expand and collapse	Sidebar toggles correctly
TC-11 (Added)	Dashboard	Empty state handling	“No accounts connected” message shown
TC-12 (Added)	UI	Responsive layout	UI adapts correctly on mobile and desktop

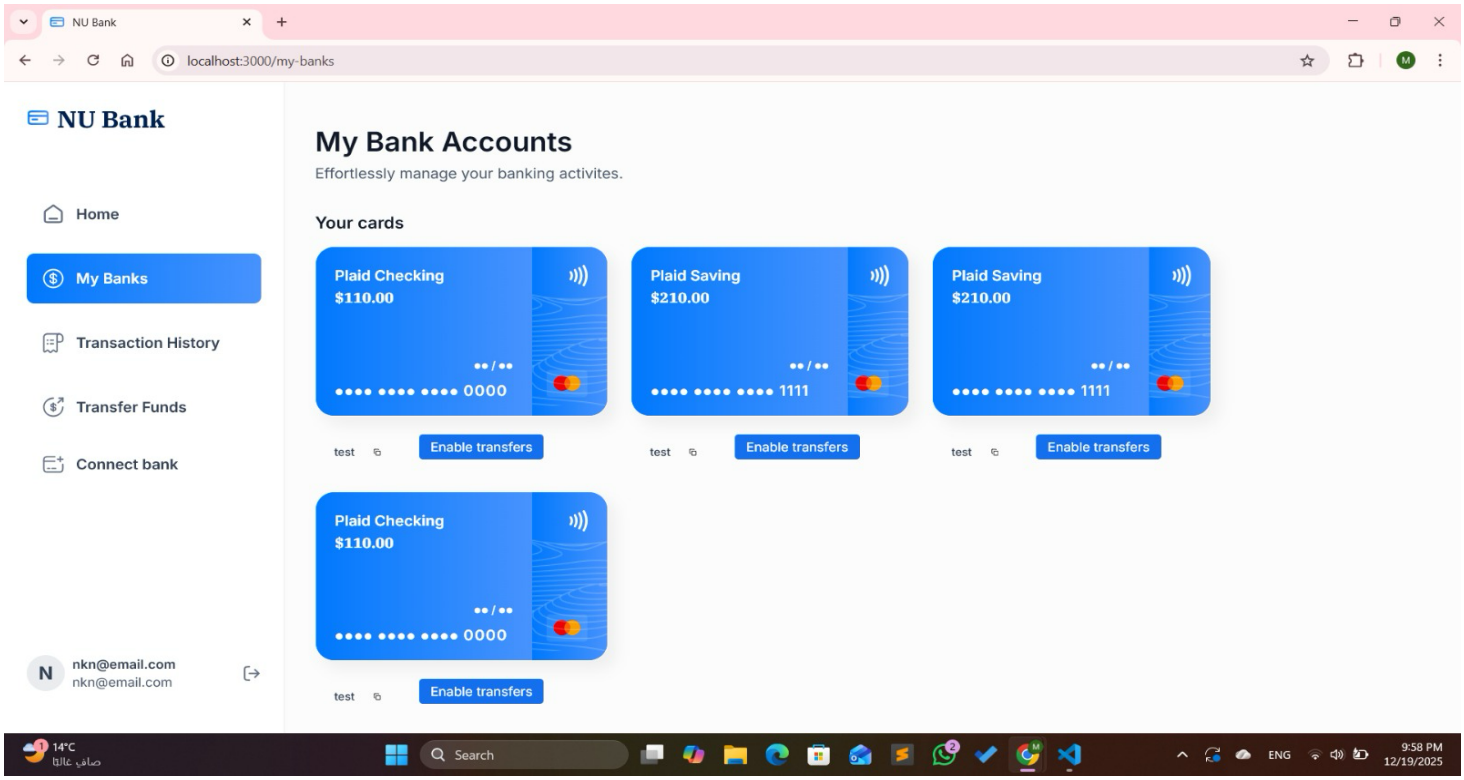
Bank Account Management Test Cases			
Test ID	Feature	Test Scenario	Expected Result
TC-13	Bank Connection	Connect multiple bank accounts	New bank appears in “My Banks” section
TC-14	Bank Connection	Connect bank with invalid information	Error or validation message displayed
TC-15 (Added)	Bank Connection	Reconnect existing bank	Bank connection refreshed successfully
TC-16 (Added)	Bank Connection	Plaid API failure	Graceful error message displayed

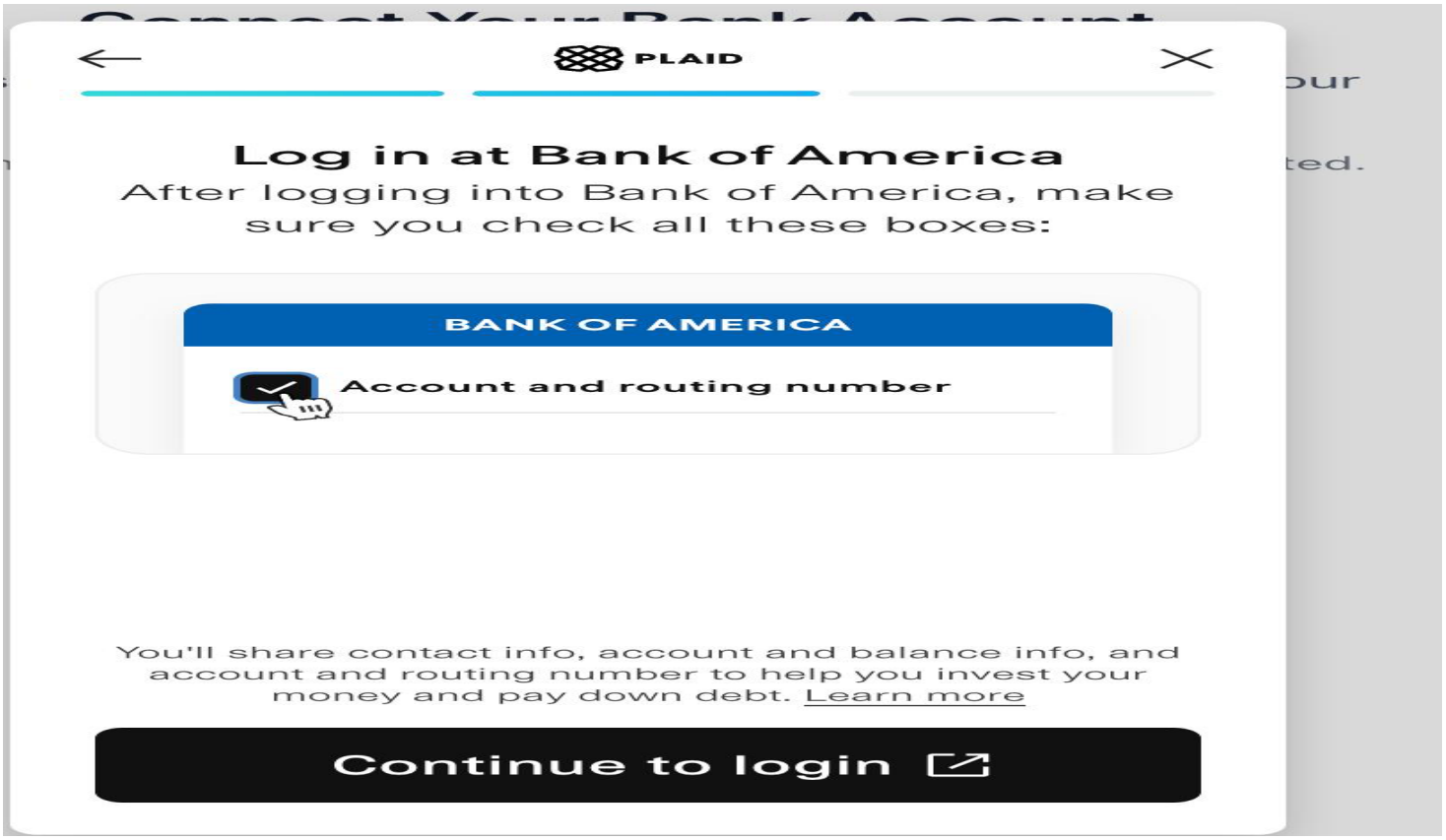
Transactions Test Cases			
Test ID	Feature	Test Scenario	Expected Result
TC-17	Transactions	Display recent transactions	Recent transactions displayed correctly
TC-18	Transactions	Transaction pagination	Next page of transactions displayed
TC-19 (Added)	Transactions	Filter by date or category	Filtered transactions displayed
TC-20 (Added)	Transactions	No transactions available	“No transactions found” message

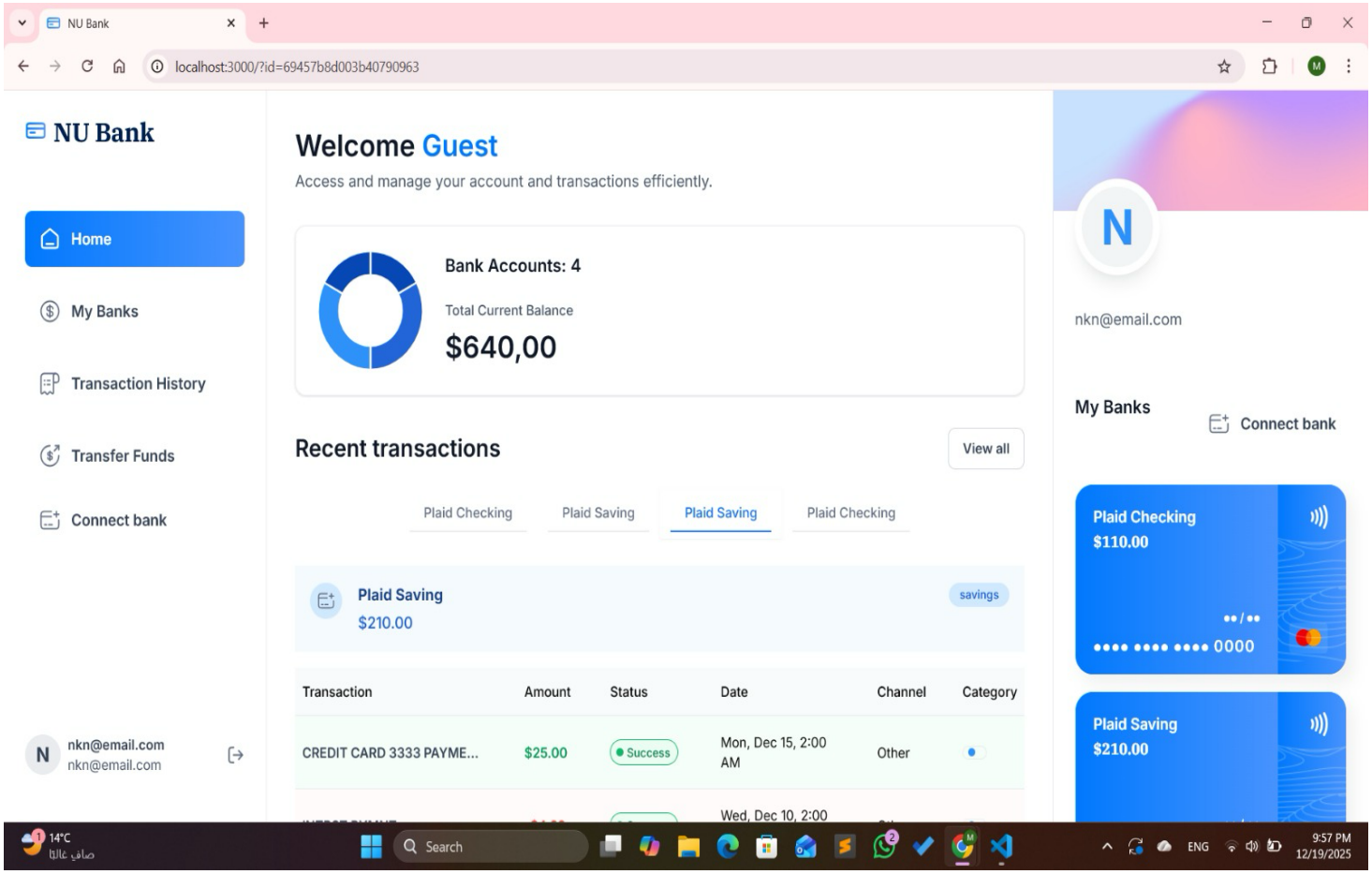
Transfer & Payment Test Cases			
Test ID	Feature	Test Scenario	Expected Result
TC-21	Payment	Transfer payment page loads	Transfer form fields visible
TC-22	Payment	Transfer with insufficient balance	Error message “Insufficient funds”
TC-23 (Added)	Payment	Successful transfer	Transfer completed and confirmation shown
TC-24 (Added)	Payment	Invalid transfer amount	Validation error shown
TC-25 (Added)	Payment	Duplicate transfer submission	Duplicate request prevented

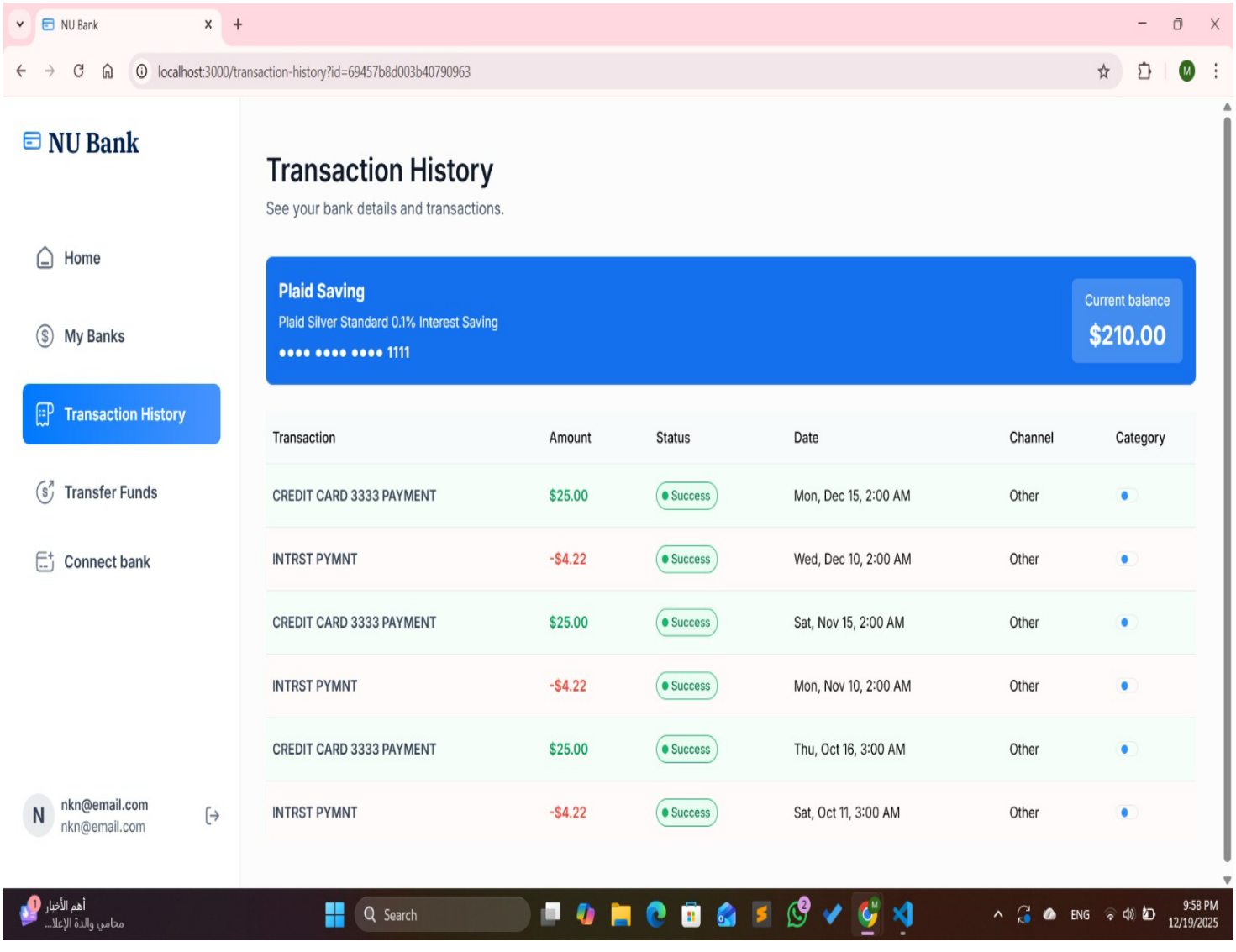
Data & Visualization Test Cases			
Test ID	Feature	Test Scenario	Expected Result
TC-26	History	Transaction history page loads	Historical transactions displayed
TC-27	Analytics	Spending categories display	Categories rendered correctly
TC-28 (Added)	Charts	Chart data accuracy	Chart matches transaction data
TC-29 (Added)	Charts	Real-time updates	Charts update after new transaction

Security & Session Management Test Cases			
Test ID	Feature	Test Scenario	Expected Result
TC-30	Security	Session expiration after logout	User redirected to login page
TC-31 (Added)	Security	Access protected route without login	Redirected to login page
TC-32 (Added)	Security	Session timeout after inactivity	User logged out automatically






NU Bank
localhost:3000/?id=69457b8d003b40790963
NU Bank
Home
My Banks
Transaction History
Transfer Funds
Connect bank
nkn@email.com
nkn@email.com
Welcome Guest
Access and manage your account and transactions efficiently.
Bank Accounts: 4
Total Current Balance
$640,00
Recent transactions
View all
Plaid Checking
Plaid Saving
Plaid Saving
Plaid Checking
Plaid Saving
$210.00
savings
Transaction
Amount
Status
Date
Channel
Category
CREDIT CARD 3333 PAYME...
$25.00
Success
Mon, Dec 15, 2:00 AM
Other
Wed, Dec 10, 2:00
N
nkn@email.com
My Banks
Connect bank
Plaid Checking
$110.00
•••• •••• •••• 0000
Plaid Saving
$210.00
14°C
صافي غالبًا
Search
ENG
9:57 PM
12/19/2025


NU Bank
localhost:3000/transaction-history?id=69457b8d003b40790963
NU Bank
Home
My Banks
Transaction History
Transfer Funds
Connect bank
nkn@email.com
nkn@email.com
Transaction History
See your bank details and transactions.
Plaid Saving
Plaid Silver Standard 0.1% Interest Saving
•••• •••• •••• 1111
Current balance
$210.00
Transaction
Amount
Status
Date
Channel
Category
CREDIT CARD 3333 PAYMENT
$25.00
Success
Mon, Dec 15, 2:00 AM
Other
INTRST PYMNT
-$4.22
Success
Wed, Dec 10, 2:00 AM
Other
CREDIT CARD 3333 PAYMENT
$25.00
Success
Sat, Nov 15, 2:00 AM
Other
INTRST PYMNT
-$4.22
Success
Mon, Nov 10, 2:00 AM
Other
CREDIT CARD 3333 PAYMENT
$25.00
Success
Thu, Oct 16, 3:00 AM
Other
INTRST PYMNT
-$4.22
Success
Sat, Oct 11, 3:00 AM
Other
أهم الأخبار
محامي والدة الإعلا...
Search
ENG
9:58 PM
12/19/2025

Email

Enter your email

Please enter a valid email

Password

Enter your password

Password must be at least 8 characters

Sign In

State (e.g., CA)

CA

Postal Code

123456

Postal code must be a valid US ZIP code

Date of Birth (YYYY-MM-DD)

22/2/2222

Date of Birth must be in YYYY-MM-DD format

SSN

123456789

Email

State (e.g., CA)

CA

Postal Code

12345

Date of Birth (YYYY-MM-DD)

1111-11-11

SSN

123456789

Email

PERFORMANCE METRICS

This section outlines the key performance indicators used to evaluate the efficiency, responsiveness, and security of the system. These metrics were defined to ensure a smooth user experience, reliable system

User Experience

The system is designed to provide a fast and intuitive user experience across all core workflows. User registration is completed within **3 to 5 minutes**, including validation and account setup. Connecting a bank account typically takes **2 to 3 minutes**, depending on the external banking provider. Payment or transfer initiation is processed in **less than 3 seconds**, while the main dashboard loads in **under 2 seconds**, ensuring quick access to essential financial information.

- Registration: 3–5 min
- Bank connection: 2–3 min
- Transfer initiation: < 3 sec
- Dashboard load: < 2 sec

System Performance

From a system-level perspective, performance is monitored through response times and data processing efficiency. External API calls maintain response times of **less than 2 seconds** for Plaid and **less than 3 seconds** for Dwolla. Database queries are optimized to execute within **200 milliseconds**, enabling fast data retrieval. Real-time updates propagate to the user interface in **under 1 second**, and the overall system maintains an availability target of **99.9% uptime**.

- API Response: < 2 sec (Plaid), < 3 sec (Dwolla)
- DB Queries: < 200 ms
- Real-time updates: < 1 sec
- System uptime: 99.9%

Security

Security-related performance metrics focus on rapid threat detection and data protection. Failed login attempts are detected in real time to prevent unauthorized access. Fraud detection mechanisms respond in **less than 1 second**, minimizing potential risk. All sensitive data is protected using **AES-256 encryption**, ensuring compliance with industry-standard security practices.

- Failed login detection: Real-time
- Fraud detection: < 1 sec
- Encryption: AES-256

DEPLOYMENT & CI/CD

The system is deployed using a cloud-based architecture to ensure high availability and ease of access. The frontend application is hosted on Vercel, providing optimized performance and seamless support for Next.js applications. The backend services and database are managed through Appwrite Cloud, which offers scalable server-side functionality and secure data storage using a NoSQL database. Additionally, the system integrates with external financial services such as Plaid, Dwolla, and Stripe to support bank

Deployment

- **Frontend:** Vercel (Next.js)
- **Backend:** Appwrite Cloud
- **Database:** Appwrite NoSQL
- **Third-Party Services:** Plaid, Dwolla, Stripe

CI/CD

A continuous integration and continuous deployment (CI/CD) pipeline is implemented to maintain code quality and ensure reliable deployments. GitHub Actions is used to automate the build and testing processes. On every pull request, unit and integration tests are executed to detect issues early. Once changes are approved, deployments are automatically triggered, ensuring that the latest stable version of

- GitHub Actions for build & test
- Automatic deployments to Vercel/Appwrite
- Unit & integration tests run on each pull request

MAINTENANCE & SCALABILITY

The system is designed with maintainability and scalability as core principles. A modular architecture is used to simplify future enhancements and reduce system complexity. Real-time data synchronization is supported to ensure that users always interact with up-to-date information. The backend infrastructure allows horizontal scaling to handle increased user load efficiently. Furthermore, automated monitoring and alert mechanisms are in place to track system performance, availability, and potential errors, enabling

- Modular API and component design
- Real-time subscriptions for instant sync
- Horizontal scaling for backend services
- Automated alerts and monitoring for uptime & errors

Conclusion

This project demonstrates the practical application of software engineering principles in designing and developing a complete, scalable, and secure system. Throughout the project, emphasis was placed on clear requirements analysis, modular system architecture, and well-defined documentation to ensure maintainability and future extensibility. The system successfully integrates multiple components, including user authentication, data management, transaction handling, and external service integrations, while maintaining strong performance and security standards.

In addition, the use of structured design models, comprehensive testing, and performance evaluation highlights the team’s commitment to building a reliable and user-centered solution. The project reflects a solid understanding of both theoretical concepts and real-world software development practices, providing a strong foundation for future enhancements and deployment in production environments.