Arhitectural Design Document

Table of Contents

1. Introduction

An online banking platform digital service provided by a financial institution that allows its customers to perform various banking transactions and activities through the internet.

The primary purpose of an online banking platform is to provide convenience and accessibility to customers, allowing them to access their account information, make payments, transfer funds, and manage their finances from anywhere with an internet connection

1.1 System Purpose

The system purpose is to provide customers with convenient and secure access to their banking accounts and services via the internet.

It enables customers to perform various banking transactions and activities from their own devices, such as smartphones, tablets, or computers, without having to physically visit a bank branch.

1.3 Definitions, Acronyms

HTTPS - Hypertext Transfer Protocol Secure (https) combination of the Hypertext Transfer Protocol (HTTP) with the Secure Socket Layer (SSL)/Transport Layer Security (TLS) protocol. TLS authentication and security protocol widely implemented in browsers and Web servers.

JWT TOKEN - An open industry standard used to share information between two entities, usually a client (like your app's frontend) and a server (your app's backend). They contain JSON objects which have the

information that needs to be shared.

2FA - Two-factor authentication identity and access management security method that requires two forms of identification to access resources and data.

REST API - API that conforms to the design principles of the REST, or representational state transfer architectural style.

EC2 - web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers.

JDTB - application programming interface (API) for the Java programming language, which defines how a client may access a database.

HIBERNATE - open source object relational mapping (ORM) tool that provides a framework to map object-oriented domain models to relational databases for web applications. Object relational mapping is based on the containerization of objects and the abstraction that provides that capacity.

AWS - comprehensive, evolving cloud computing platform provided by Amazon that includes a mixture of infrastructure-as-a-service , platform-as-a-service and packaged-software-as-a-service offerings.

RDS INSTANCE - General purpose instances offer a combination of computing, memory, and networking resources. Memory-optimized instances are helpful for efficient performance in the case of workloads that handle huge data sets in memory. Burstable performance instances offer a baseline amount of CPU usage with the flexibility to burst CPU usage above the baseline level.

BCRYPT - password-hashing function.

DNS - Domain Name System is the phonebook of the Internet. When users type domain names such as 'google.com' or 'nytimes.com' into web browsers, DNS is responsible for finding the correct IP address for those sites.

LOAD BALANCER - device that acts as a reverse proxy and distributes network or application traffic across a number of servers. Load balancers are used to increase capacity (concurrent users) and reliability of applications.

ECS CLUSTER - highly scalable, high performance container management service that supports Docker containers and allows you to easily run applications on a managed cluster of Amazon Elastic Compute Cloud (Amazon EC2) instances.

SMTP SERVER - application that's primary purpose is to send, receive, and/or relay outgoing mail between email senders and receivers.

MySQL Server - relational database management system (RDBMS) that supports a wide variety of transaction processing, business intelligence and analytics applications in corporate IT environments.

1.4 Reference Documents

- Requirements Specification Document

2. Design Objectives

1. Availability
    - The system will be available 24/7, 365 days a year.
    - The system will be hosted on AWS
    - The database will be hosted on an RDS instance.
    - The system will be deployed on multiple availability zones, and will be load balanced.
    - The system will be monitored using CloudWatch, and will be automatically scaled based on the load.
2. Security
    - The system will be secured using HTTPS.
    - The system will be secured using JWT tokens.
    - The system will be secured using 2FA.
    - The system will be secured using role based access control.
    - The system will be secured using password hashing.
3. Usability
    - The system will be easy to use.
    - The system will be responsive.
    - The system will be accessible from any device.
4. Maintainability
    - The system will be easy to maintain.
    - The system will be easy to extend.
    - The system will be easy to deploy.

3. Proposed Architecture

3.1 General Overview of the System Architecture

1. **User subsystem:**:Responsible for managing the interactions between the bank and its customers who use the online platform. The subsystem typically consists of a user interface and a set of functionalities to provide the customer with a convenient, secure and efficient banking experience.

    Authentication:

    - users need to authenticate themselves to access their accounts, typically through a username and password and through a AWD token(2FA).
    - once the authentication is complete the user will get access to his account.

    Home:

    - the user will be able to select one of the following options: accounts, transactions and balance.
    - The “home” section practically represents a menu where the user.

    Accounts:

    - the user will be able to access the accounts that he has created, for instance: savings account, pay services account, shopping account.

Transactions:

- Users can initiate and manage bill payments, set up recurring payments, and make payments to other users or third-party accounts.
- Fill the form with the required data (type of transaction, amount, destination account, etc.)
- The user is redirected to the transactions page where he can see the new transaction.

Balance:

- check amount left in any type of account.

2. **Administrator subsystem**: As a privileged user, the administrator should ensure that the platform functions smoothly and securely, being able to operate in the following components: Authentication, Administration and Dashboard.

   Authentication:

   - the adminsitrator should authenticate using its credentials.
   - after a successful login, the administrator can view its account.

   Administration:

   - The administrator should review and administer user requests , such as password reset requests, account activation requests, or any other requests that require administrative approval.
   - It has the right to change, modify or delete information from the system.

   Dashboard:

   - The administrator should be able to view its accont info.

3. **Backend Client**: Abstraction layer between the User and the Administrator used for data exchange between the client and the server applications.

   Rest API:

   - set of rules or constraints.
   - uses HTTP methods to define the type of operation that needs to be performed on a resource(eg. GET, POST, DELETE etc).
   - the resource is presented to the requester in a json format

4. **Bussiness Logic**: The business logic layer is the core of the application. It contains the business logic of the application and is responsible for processing the data and returning the results to the presentation layer. Itlso responsible for communicating with the data access layer to retrieve and store data.

   Authorization:

   - Receives the request from the client and checks if the useruthorized to perform the requested action.
   - Receives credentials from the client and checks if the user can be authenticated.
   - Receives the JWT token from the client and checks if the token is valid.

Data Validation:

- Receives data from the client and checks if the data is valid.
- Redirects the request to the component that will process the data.

Transaction:

- Check the balance of the user's account and if the user has enough money to make the transaction.
- Update the balance of the user's account in the database.

User Data:

- Receives updates on the user's data from the client and updates the user's data in the database.
- User data includes managing of the user's profile, the user's accounts, etc.

Bank Data:

- Receives updates on the bank's data from the client and updates the bank's data in the database.
- Bank data includes managing of the savings accounts, the credit cards, etc.

Data Access:

- Receives requests from the business logic layer and retrieves the data from the database.
- Updates the database with the data received from the business logic layer.
- Logs the data access operations.

5. **SMTP**

   - Gets the email address of the user from the database.
   - Gets request from the business logic layer.
   - Responsible for sending emails to the user.

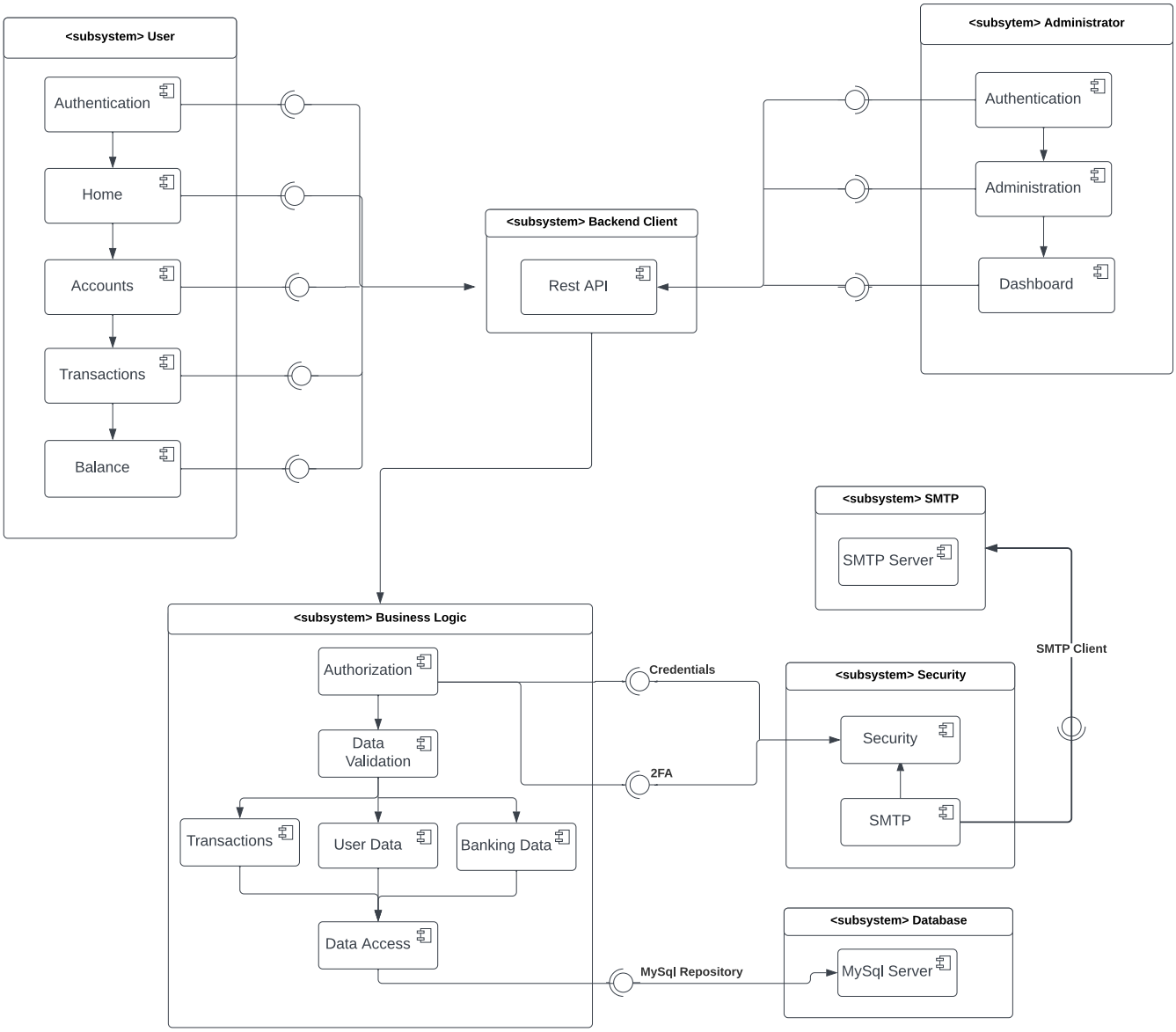
6. **Security**

   - Responsible for authentication.
   - Responsible for authorization.
   - Responsible for 2FA.
   - Responsible for password hashing.
   - Confirms to the backend client that the useruthorized to perform the requested action.

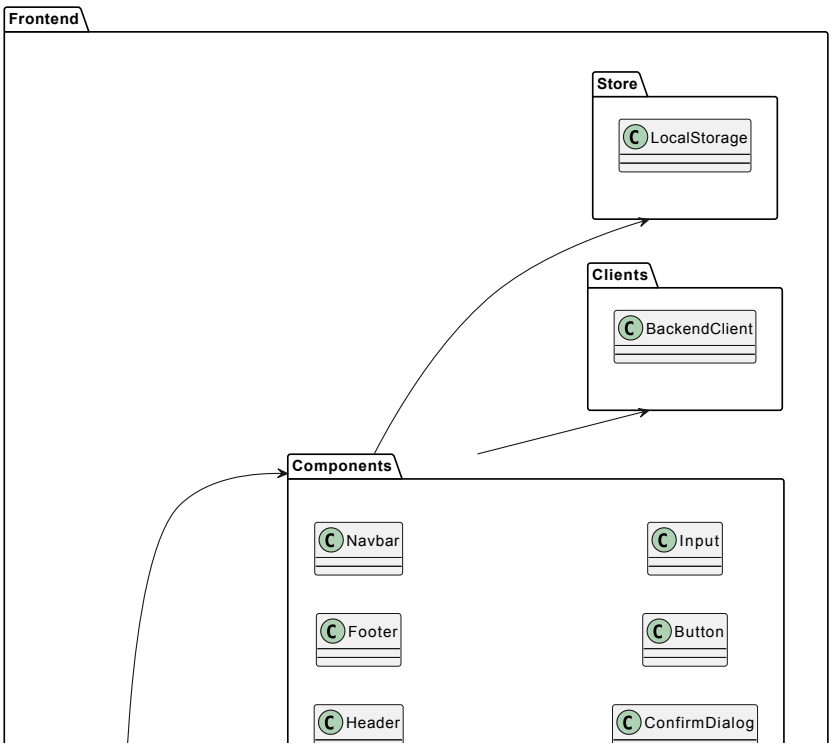
7. **Database**

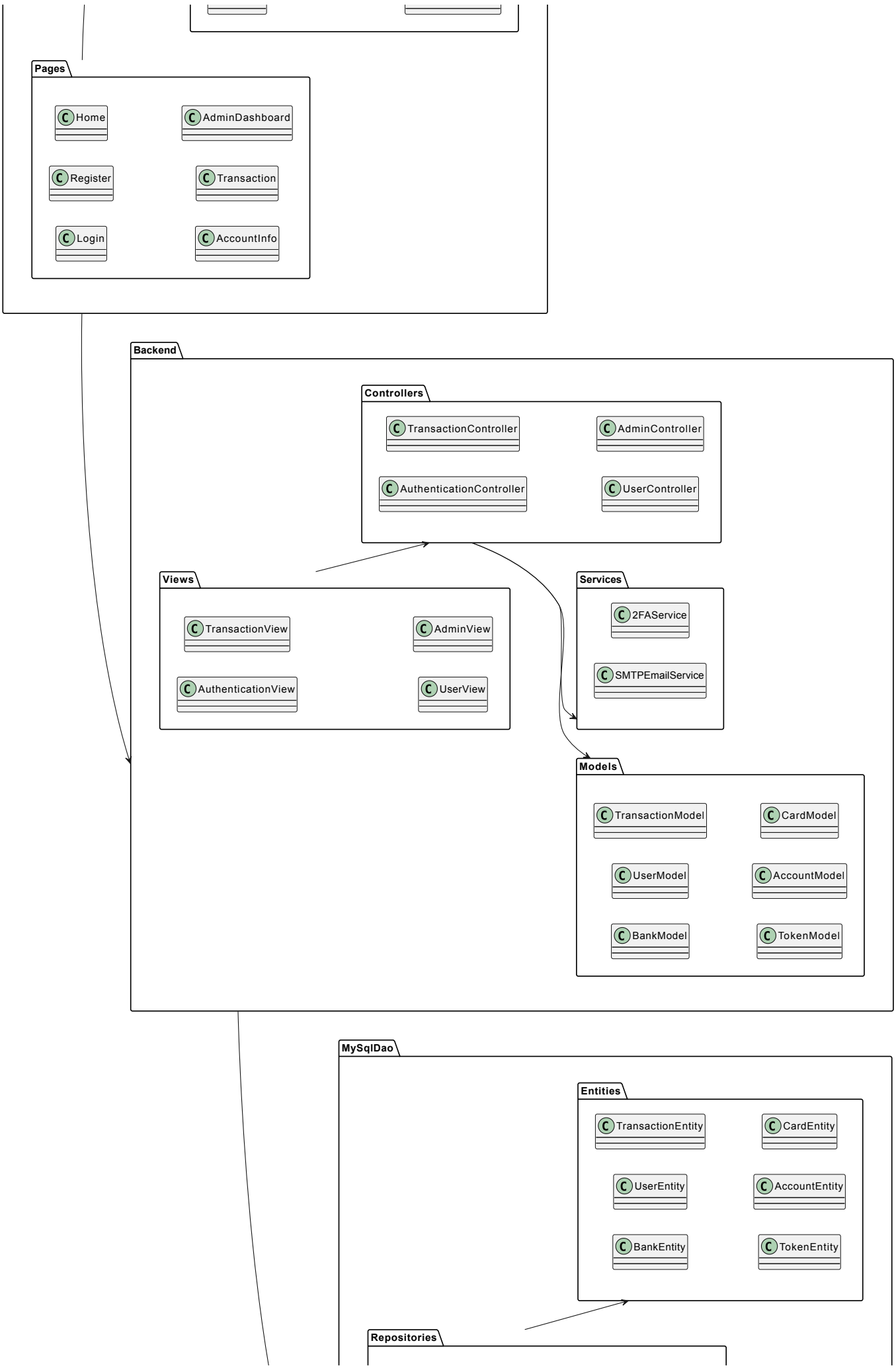
   - Responsible for storing the data.
   - Partially responsible for data validation.
   - Role based access control.

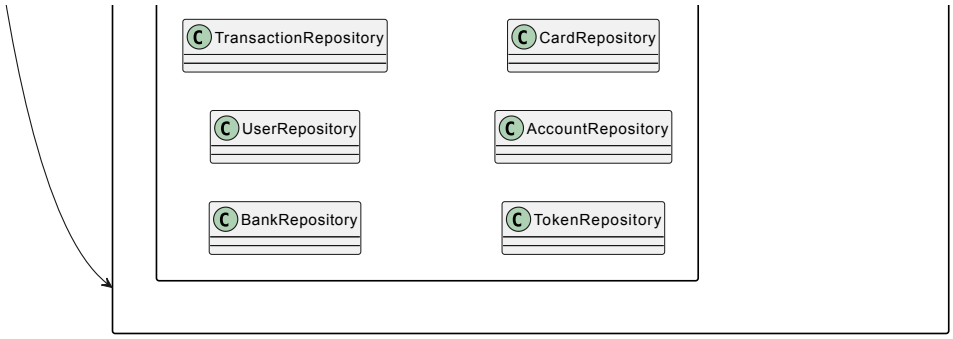
3.2 Subsystem Decomposition and Responsibilities



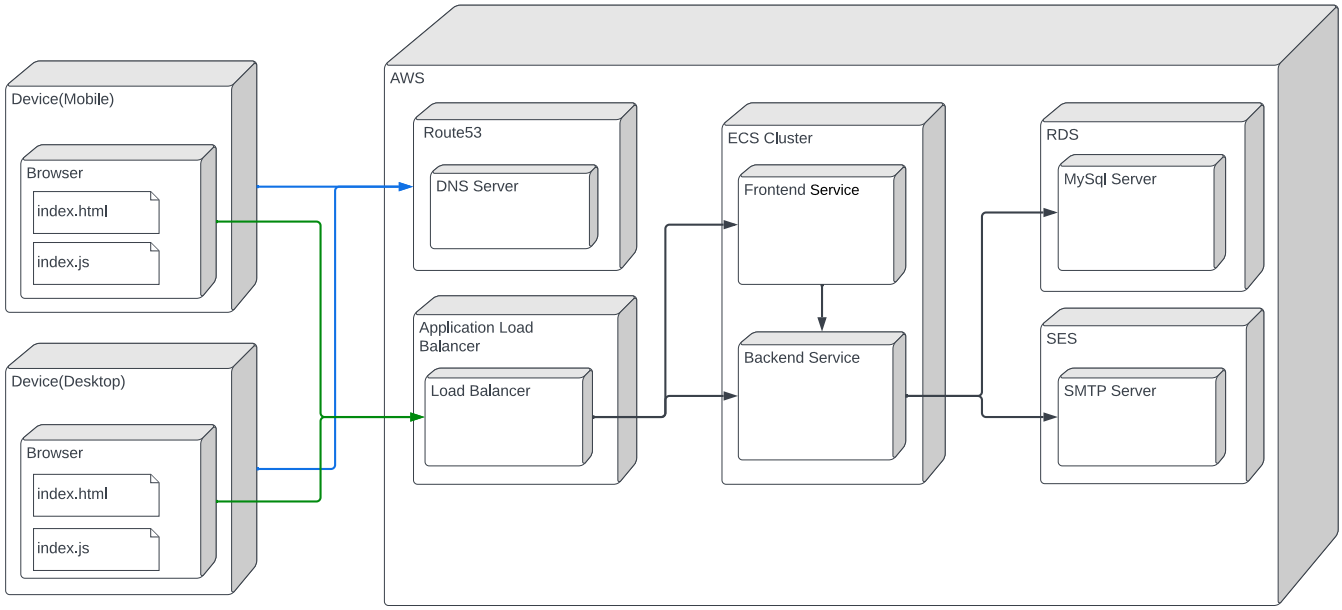
3.3 Packaging




Pages
Home
AdminDashboard
Register
Transaction
Login
AccountInfo
Backend
Controllers
TransactionController
AdminController
AuthenticationController
UserController
Views
TransactionView
AdminView
AuthenticationView
UserView
Services
2FAService
SMTPEmailService
Models
TransactionModel
CardModel
UserModel
AccountModel
BankModel
TokenModel
MySqlDao
Entities
TransactionEntity
CardEntity
UserEntity
AccountEntity
BankEntity
TokenEntity
Repositories



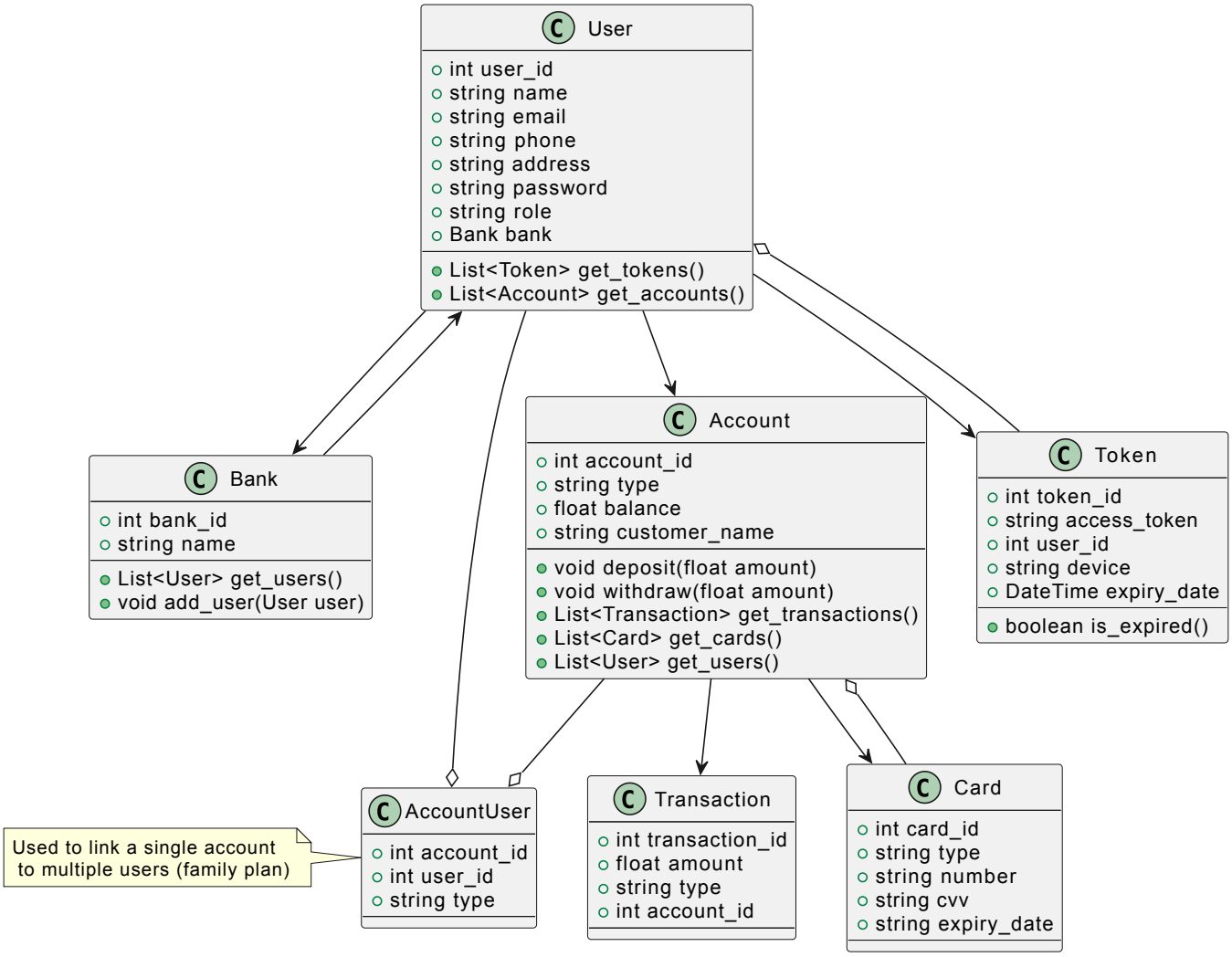
3.4 Subsystem Distribution on Hardware/Software Platforms



3.5 Persistent Data Management

For persistent data management, we will use a relational database, MySQL. It will be running on an EC2 instance, and will be accessed by the application through a JDBC driver. The backend application will access the database through the Hibernate ORM framework.

The database schema is defined in the following diagram:



3.6 User Access Control to the System

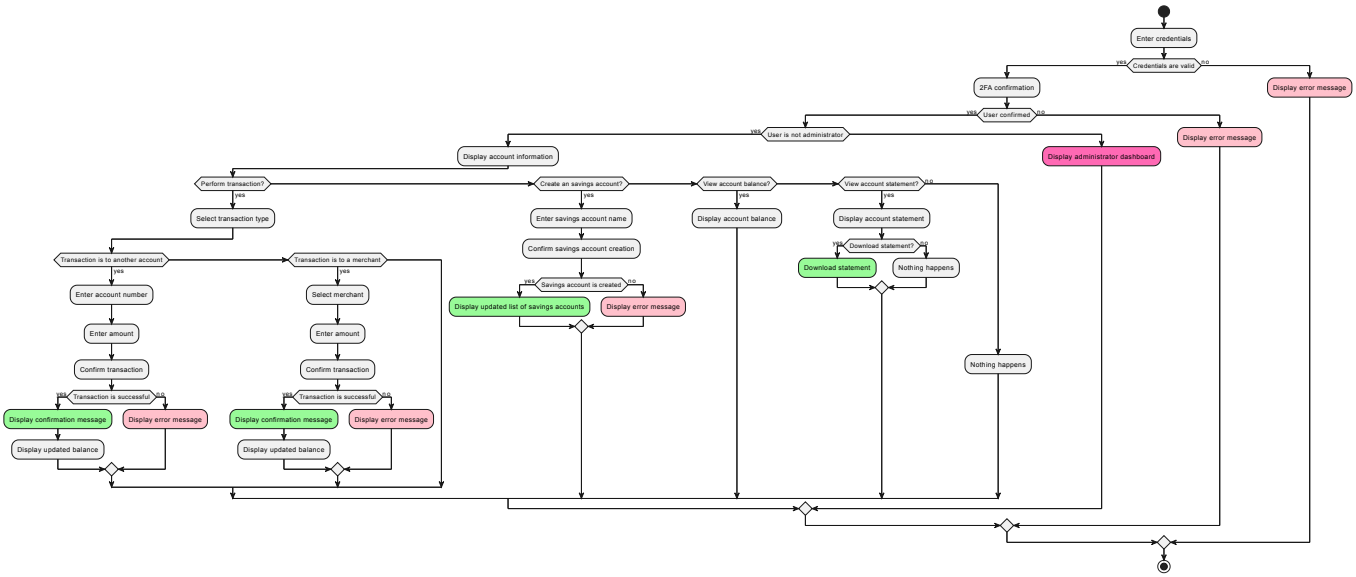
1. Authentication
    - authentication is performed in 2 steps (login and 2FA)
    - communication between the frontend and the backend is done through REST API
    - traffic is encrypted using HTTPS
2. Authorization
    - after the useruthenticated, the backend will generate a JWT token
    - JWT will be stored in the browser's local storage and send with every request
    - for an request to be authorized, the JWT token must be valid (not expired) and the user must have the required role
    - roles are stored in the database, and are assigned to users
    - JWT token expires after 72 hours (3 days)
3. User data management
    - user can only access their own data
    - only admins can access other user's data
    - admins can create, update and delete users
    - users can only update their own data

- any other user specific data (transactions, savings accounts, etc.) follows the same rules as above

4. Security

- passwords are hashed using BCrypt
- communication between the frontend and the backend is done through HTTPS
- JWT token has a limited lifetime
- JWT token stores data about the device and cannot be used on another device

3.7 Global Control Flow



3.8 Boundary Conditions

Global:

1. No internet connection

- The application will not be able to access the database, and will not be able to perform any operations with the server.
- React will display an error message, but will allow the user to continue using the application with the data thatlready loaded.

2. User is not logged in:

- The user will be redirected to the login page.

3. User is not allowed to perform an action:

- The user will be redirected to the home page.

Create a new user:

1. The user already exists:

- No new user will be created, and the user will be notified that the usernamelready taken.

2. Password is not strong enough:
    - Frontend won't allow the user to submit the form, and will display a message with the requirements for the password.
3. Passwords don't match:
    - Frontend won't allow the user to submit the form, and will display a message that the passwords don't match.
4. Email is not valid:
    - Frontend won't allow the user to submit the form, and will display a message that the email is not valid.

Authenticate a user:

1. User doesn't exist:
    - The user will be notified that the username or password is incorrect. (The user will not be notified that the username doesn't exist, to prevent brute force attacks.)
2. Password is incorrect:
    - The user will be notified that the username or password is incorrect.
3. User is not verified:
    - The user will be notified that the account is not verified, and will be redirected to the verification page.

Perform a transaction:

1. The user doesn't have enough money:
    - The transaction will not be performed, and the user will be notified that they don't have enough money.
2. The destination account doesn't exist (same if the user is not verified):
    - The transaction will not be performed, and the user will be notified that the destination account doesn't exist.

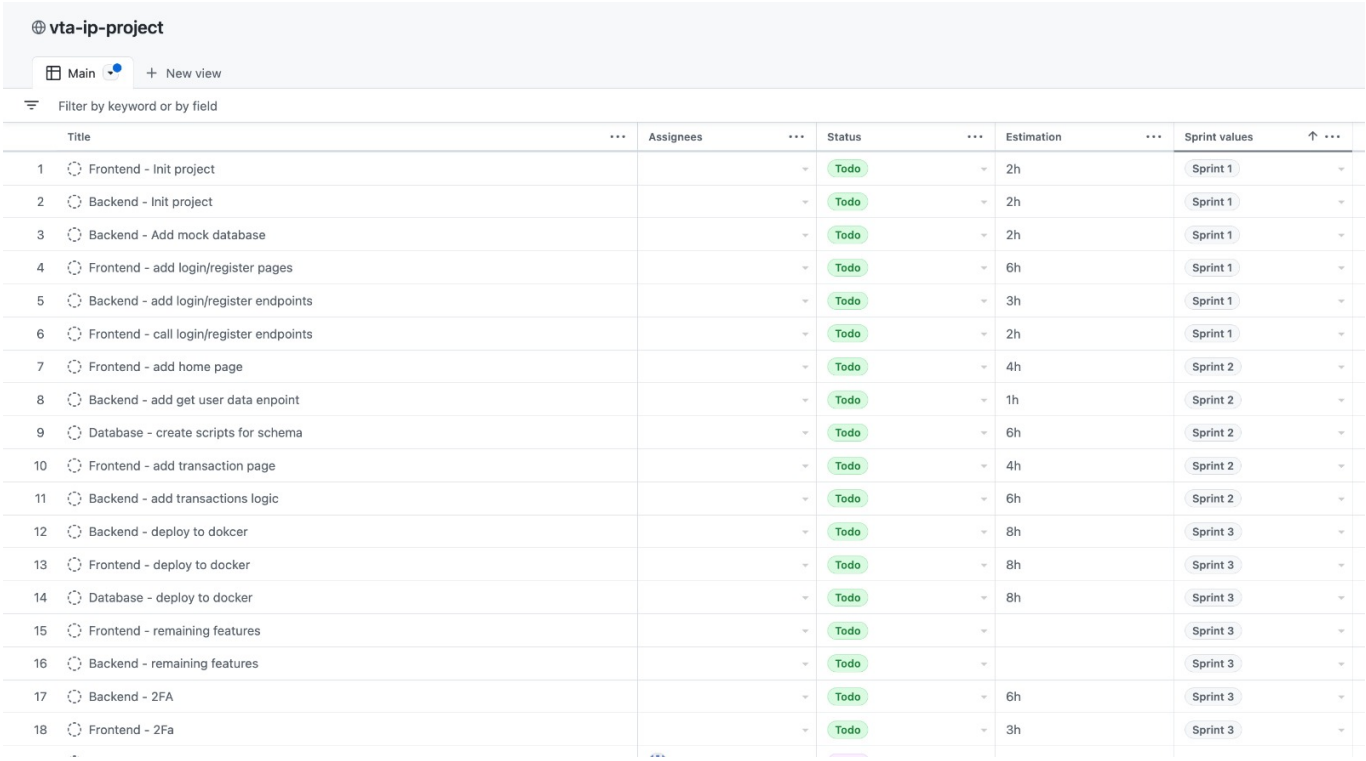
Create a savings account:

1. The user already has a savings account:
    - The savings account will not be created, and the user will be notified that they already have a savings account.

4. Planification

Dashboard: https://github.com/users/vladtf/projects/1/views/1?sortedBy%5Bdirection%5D=asc&sortedBy%5BcolumnId%5D=38728626

Table:



vta-ip-project

Main + New view

Filter by keyword or by field

	Title	Assignees	Status	Estimation	Sprint values
1	Frontend - Init project		Todo	2h	Sprint 1
2	Backend - Init project		Todo	2h	Sprint 1
3	Backend - Add mock database		Todo	2h	Sprint 1
4	Frontend - add login/register pages		Todo	6h	Sprint 1
5	Backend - add login/register endpoints		Todo	3h	Sprint 1
6	Frontend - call login/register endpoints		Todo	2h	Sprint 1
7	Frontend - add home page		Todo	4h	Sprint 2
8	Backend - add get user data enpoint		Todo	1h	Sprint 2
9	Database - create scripts for schema		Todo	6h	Sprint 2
10	Frontend - add transaction page		Todo	4h	Sprint 2
11	Backend - add transactions logic		Todo	6h	Sprint 2
12	Backend - deploy to dokcer		Todo	8h	Sprint 3
13	Frontend - deploy to docker		Todo	8h	Sprint 3
14	Database - deploy to docker		Todo	8h	Sprint 3
15	Frontend - remaining features		Todo		Sprint 3
16	Backend - remaining features		Todo		Sprint 3
17	Backend - 2FA		Todo	6h	Sprint 3
18	Frontend - 2Fa		Todo	3h	Sprint 3