SYSTEM REQUIREMENTS DOCUMENT

PROJECT DETAILS

PROJECT NAME		
YouBank		
CREATOR		
SZ TECH LLC		
DOCUMENT NO.	DATE	VERSION NO.
N/A	11/8/2023	0.1

1. EXECUTIVE SUMMARY SNAPSHOT

The Bank Sharing Application is an innovative financial technology solution designed to empower users with easy and secure access to banking services, enabling them to manage their accounts, conduct transactions, and stay informed about their financial activities. This application serves as a digital bridge between users, their accounts, and the Federal Reserve Bank (FRB) network. This document outlines the key technical requirements for the development and implementation of this application.

Objective:
The primary goal of the Bank Sharing Application is to provide a comprehensive and user-friendly digital platform for users to interact with their bank accounts, facilitate transactions, and access essential information about their financial institution. This application also includes an interface for bank administrators to manage user accounts, transactions, and FRB information.

Key Features:

1. **User Management:**

User registration with unique usernames and email addresses.
User authentication and login capabilities.
User profile management for updating personal information.

2. **Account Management:**

Creation and management of various types of bank accounts (e.g., savings, checking).
Real-time access to account balances, transaction history, and account statements.
Secure transaction processing, including deposits, withdrawals, and transfers.

3. **Transaction Processing:**

Handling deposits, withdrawals, and inter-account transfers.
Ensuring transaction security, integrity, and accuracy.
Transaction history and transaction receipts for users.

4. **Notifications:**

Sending notifications to users for important account events.

Supporting various notification channels (e.g., email, SMS, in-app notifications).

5. **Admin Panel:**

Admin interface for bank employees to manage user accounts and transactions.
Comprehensive audit trails for admin actions and financial oversight.

6. **FRB Integration:**

Storage and retrieval of FRB information, including routing numbers and contact details.
Admin capabilities to add, modify, and delete FRB data.

7. **Security and Compliance:**

Implementation of robust security measures, including data encryption and multi-factor authentication.
Compliance with financial regulations such as Know Your Customer (KYC) and Anti-Money Laundering (AML) requirements.
Protection of user data and privacy in accordance with data protection laws.

2. UML DIAGRAM

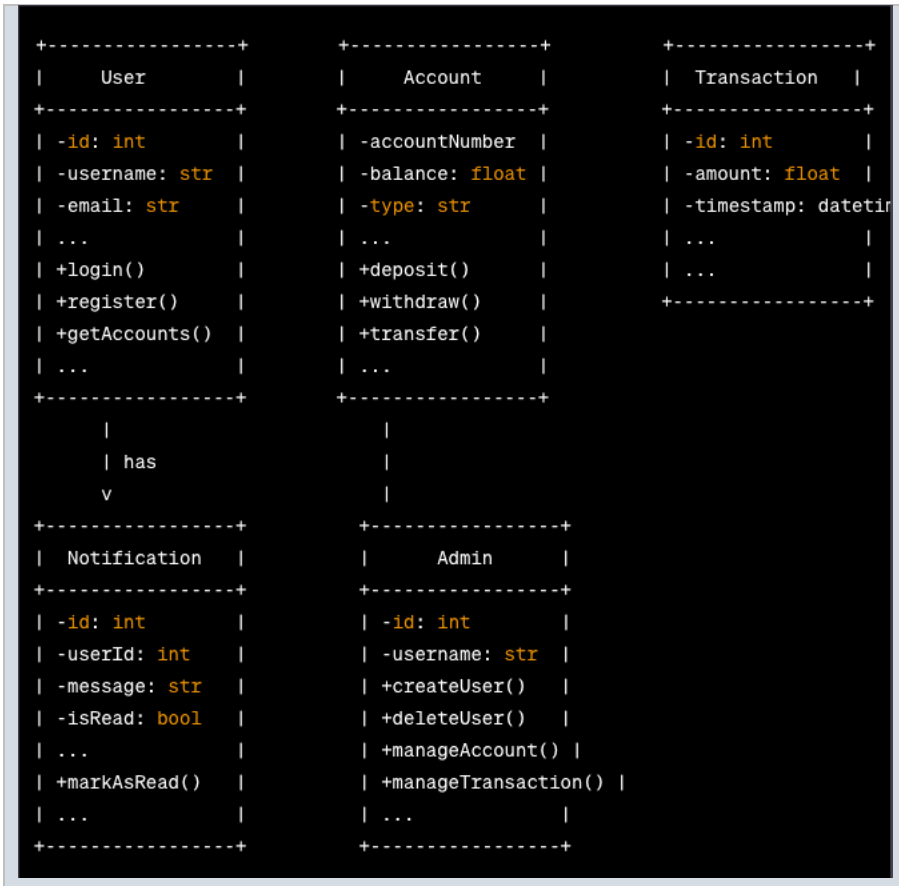
Creating a full UML (Unified Modeling Language) diagram and a backend API schema for a bank sharing application is a complex task, and it would be extensive to provide a complete diagram here. However, I can provide you with a simplified example to get you started. In this example, I'll focus on the key components, classes, and API endpoints.

UML Diagram:

1. Class Diagram:
   - User
   - Account
   - Transaction
   - Notification
   - Admin

2. Relationships:
   - User has one or more Accounts.
   - Account has one or more Transactions.
   - User can receive Notifications.
   - Admin manages Users, Accounts, and Transactions.

Textual representation of the UML class diagram:



Backend API Schema:

Here's a simplified representation of the API endpoints that correspond to the UML diagram:

1. User Management:
   - `POST /users`: Register a new user.
   - `POST /users/login`: User login.
   - `GET /users/{id}`: Get user details.
   - `GET /users/{id}/accounts`: Get user's accounts.
   - `GET /users/{id}/notifications`: Get user's notifications.
   - `PUT /users/{id}`: Update user information.
   - `DELETE /users/{id}`: Delete a user.

2. Account Management:
   - `POST /accounts`: Create a new account.
   - `GET /accounts/{accountNumber}`: Get account details.
   - `POST /accounts/{accountNumber}/deposit`: Make a deposit to an account.
   - `POST /accounts/{accountNumber}/withdraw`: Withdraw funds from an account.
   - `POST /accounts/{accountNumber}/transfer`: Transfer funds between accounts.
   - `GET /accounts/{accountNumber}/transactions`: Get account transactions.

3. Notification Management:
   - `GET /notifications`: Get all notifications.
   - `GET /notifications/{id}`: Get a specific notification.
   - `PUT /notifications/{id}`: Mark a notification as read.

4. Admin Actions:
   - `POST /admin/users`: Create a new user (admin).
   - `DELETE /admin/users/{id}`: Delete a user (admin).
   - `POST /admin/accounts`: Create a new account (admin).

- `POST /admin/transactions`: Create a new transaction (admin).
- `DELETE /admin/transactions/{id}`: Delete a transaction (admin).

Please note that this is a simplified representation. In a real-world application, you would need to consider more detailed API endpoints, data validation, security measures (authentication and authorization), request/response formats (e.g., JSON), and error handling. Additionally, you would need to document these endpoints and their functionalities comprehensively.

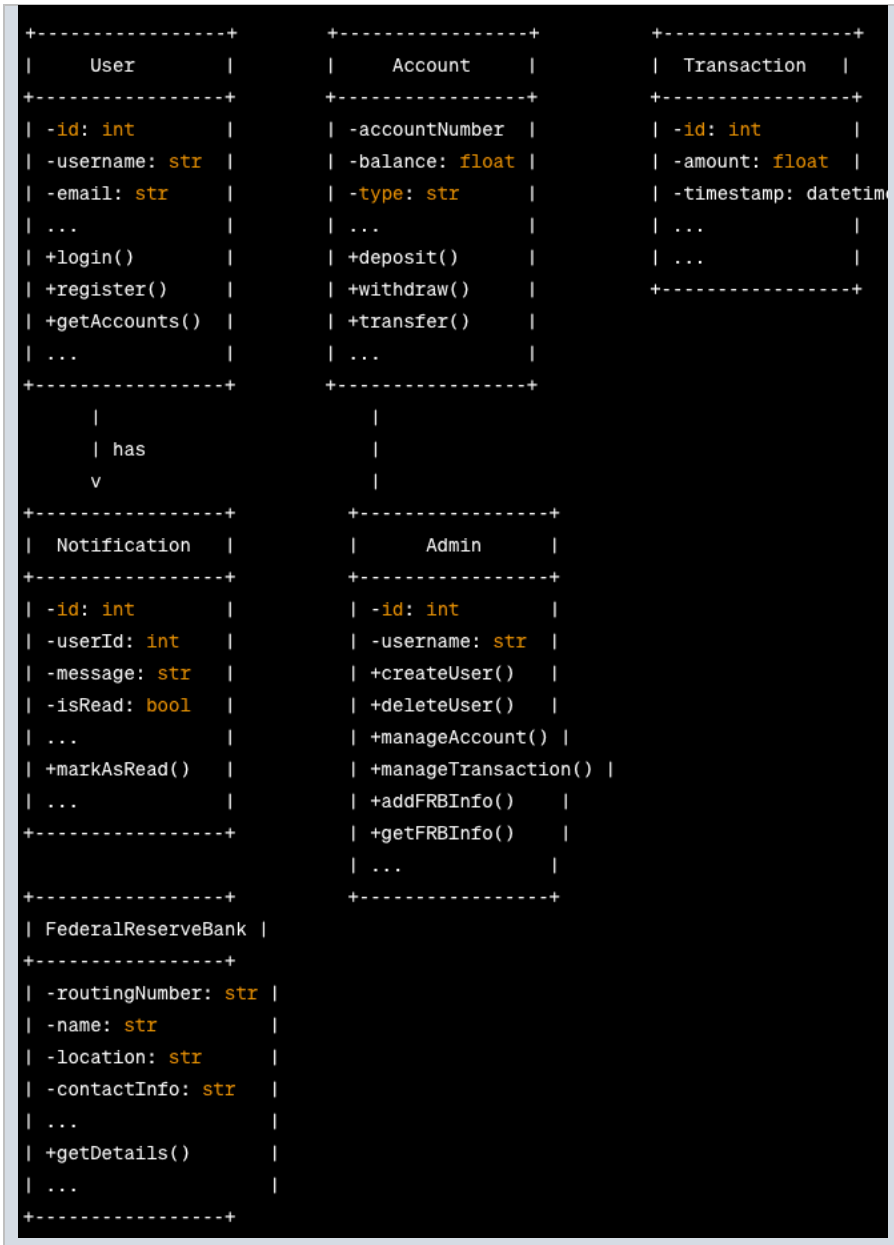
The following is Federal Reserve Bank (FRB) information into the bank sharing application that is needed to expand the backend API schema and database structure to include relevant FRB-related data.

UML Diagram (Extension):

1. Class Diagram:
   - User
   - Account
   - Transaction
   - Notification
   - Admin
   - FederalReserveBank

2. Relationships:
   - User has one or more Accounts.
   - Account has one or more Transactions.
   - User can receive Notifications.
   - Admin manages Users, Accounts, Transactions, and Federal Reserve Bank Information.

Textual representation of the extended UML class diagram:



Backend API Schema (Extension):

Extension of the existing API schema to include FRB-related information:

1. Federal Reserve Bank (FRB) Information:
   - `GET /frb`: Get a list of all Federal Reserve Banks.
   - `GET /frb/{routingNumber}`: Get details of a specific Federal Reserve Bank.

2. Admin Actions (Extension):
   - `POST /admin/frb`: Add FRB information (admin).
   - `GET /admin/frb/{routingNumber}`: Get details of an FRB (admin).
   - `PUT /admin/frb/{routingNumber}`: Update FRB information (admin).
   - `DELETE /admin/frb/{routingNumber}`: Delete FRB information (admin).

The `FederalReserveBank` class is introduced to store information about individual FRBs. This information can include details like routing numbers, bank names, locations, and contact information.

3. DATA MODEL

The following is fake data model for the bank sharing app backend API schema involves defining sample data for the various components in your application.

This is a simplified example with fictional data. In a real application, we would use a database to store and retrieve this data. The sample data includes users, accounts, transactions, notifications, and even some Federal Reserve Bank (FRB) information.

Json


```
{
  "users": [
    {
      "id": 1,
      "username": "user1",
      "email": "user1@email.com"
    },
    {
      "id": 2,
      "username": "user2",
      "email": "user2@email.com"
    }
  ],
  "accounts": [
    {
      "accountNumber": "1234567890",
      "userId": 1,
      "balance": 1000.00,
      "type": "Savings"
    },
    {
      "accountNumber": "9876543210",
      "userId": 1,
      "balance": 5000.00,
      "type": "Checking"
    },
    {
      "accountNumber": "5555555555",
      "userId": 2,
      "balance": 2500.00,
      "type": "Savings"
    }
  ],
  "transactions": [
    {
      "id": 1,
      "accountId": "1234567890",
      "amount": 100.00,
      "timestamp": "2023-11-08T10:00:00Z"
    },
    {
      "id": 2,
      "accountId": "9876543210",
      "amount": -50.00,
      "timestamp": "2023-11-08T11:00:00Z"
    },
    {
```

```
      "id": 3,
      "accountId": "5555555555",
      "amount": 200.00,
      "timestamp": "2023-11-08T12:00:00Z"
    }
  ],
  "notifications": [
    {
      "id": 1,
      "userId": 1,
      "message": "Your savings account balance has increased by $100.00.",
      "isRead": false
    },
    {
      "id": 2,
      "userId": 1,
      "message": "A withdrawal of $50.00 was made from your checking account.",
      "isRead": false
    }
  ],
  "admin": [
    {
      "username": "admin1"
    }
  ],
  "frb": [
    {
      "routingNumber": "123456",
      "name": "Fake Bank of Example",
      "location": "Sample City, USA",
      "contactInfo": "contact@fakebank.example"
    },
    {
      "routingNumber": "789012",
      "name": "Fictional National Bank",
      "location": "Imaginary Town, USA",
      "contactInfo": "info@fictionalbank.com"
    }
  ]
}
```


4. OPENAPI SPEC

This is comprehensive example that includes various endpoints for users, accounts, transactions, notifications, and Federal Reserve Bank (FRB) information. Please note that this is a simplified example.

In YAML


```
openapi: 3.0.0
info:
  title: Bank Sharing App API
  description: API for a bank sharing application
  version: 1.0.0
paths:
  /users:
    get:
```

```
    summary: Get a list of all users
    responses:
      '200':
        description: A list of users
  post:
    summary: Register a new user
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              username:
                type: string
              email:
                type: string
    responses:
      '201':
        description: User registered successfully

/users/{id}:
  get:
    summary: Get user details by ID
    parameters:
      - name: id
        in: path
        required: true
        schema:
          type: integer
        description: User ID
    responses:
      '200':
        description: User details
  put:
    summary: Update user information
    parameters:
      - name: id
        in: path
        required: true
        schema:
          type: integer
        description: User ID
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              username:
                type: string
              email:
                type: string
    responses:
      '200':
        description: User information updated successfully
```

```
  delete:
    summary: Delete a user by ID
    parameters:
      - name: id
        in: path
        required: true
        schema:
          type: integer
        description: User ID
    responses:
      '204':
        description: User deleted successfully

/accounts:
  get:
    summary: Get a list of all accounts
    responses:
      '200':
        description: A list of accounts
  post:
    summary: Create a new account
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              userId:
                type: integer
              balance:
                type: number
              type:
                type: string
    responses:
      '201':
        description: Account created successfully

/accounts/{accountNumber}:
  get:
    summary: Get account details by account number
    parameters:
      - name: accountNumber
        in: path
        required: true
        schema:
          type: string
        description: Account number
    responses:
      '200':
        description: Account details
  put:
    summary: Update account information
    parameters:
      - name: accountNumber
        in: path
        required: true
```

```
        schema:
          type: string
        description: Account number
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              balance:
                type: number
    responses:
      '200':
        description: Account information updated successfully
  delete:
    summary: Close an account
    parameters:
      - name: accountNumber
        in: path
        required: true
        schema:
          type: string
        description: Account number
    responses:
      '204':
        description: Account closed successfully

/transactions:
  get:
    summary: Get a list of all transactions
    responses:
      '200':
        description: A list of transactions
  post:
    summary: Create a new transaction
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              accountId:
                type: string
              amount:
                type: number
    responses:
      '201':
        description: Transaction created successfully

/transactions/{id}:
  get:
    summary: Get transaction details by ID
    parameters:
      - name: id
        in: path
```

```
        required: true
        schema:
          type: integer
        description: Transaction ID
    responses:
      '200':
        description: Transaction details

/notifications:
  get:
    summary: Get a list of notifications for the current user
    responses:
      '200':
        description: A list of notifications

/notifications/{id}:
  get:
    summary: Get notification details by ID
    parameters:
      - name: id
        in: path
        required: true
        schema:
          type: integer
        description: Notification ID

/frb:
  get:
    summary: Get a list of all Federal Reserve Banks
    responses:
      '200':
        description: A list of Federal Reserve Banks

/frb/{routingNumber}:
  get:
    summary: Get details of a specific Federal Reserve Bank by routing number
    parameters:
      - name: routingNumber
        in: path
        required: true
        schema:
          type: string
        description: Routing number
    responses:
      '200':
        description: Federal Reserve Bank details
```

5. TECHNICAL REQUIREMENTS

These technical requirements form the foundation for building a secure, scalable, and compliant Bank Sharing Application.

Programming Language: Java
Why? Java is a mature and widely adopted language with strong support for building secure and scalable applications. It offers excellent performance and a vast ecosystem of libraries and tools.

Web Framework: Spring Boot (Java-based)
Why? Spring Boot is well-suited for building robust and scalable web applications. It provides a comprehensive framework for creating RESTful APIs and offers features like dependency injection, security, and integration with databases.

Database: PostgreSQL (Relational Database Management System)
Why? PostgreSQL is a robust, open-source RDBMS known for data security and ACID compliance. It supports complex data models and is suitable for financial applications with relational data requirements.

RESTful API:
The application should expose a RESTful API to facilitate communication between the frontend and backend. API endpoints should be designed according to industry best practices and follow a clear, consistent structure.

Security Measures:

1. **Authentication and Authorization:**

Implement user authentication using JWT (JSON Web Tokens) to securely manage user sessions.
Define user roles (customer, employee, admin) and assign appropriate permissions.

2. **Data Encryption:**

Use HTTPS to encrypt data in transit and protect sensitive information during communication between clients and the server.
Implement encryption for sensitive data at rest in the database.

3. **Rate Limiting:**

Apply rate limiting to API endpoints to prevent abuse or misuse.

4. **Multi-Factor Authentication (MFA):**

Offer MFA options to enhance user account security.

5. **OWASP Top Ten:**

Conduct regular security assessments to address vulnerabilities listed in the OWASP Top Ten.

6. **Security Compliance:**

Ensure compliance with financial regulations, such as Know Your Customer (KYC) and Anti-Money Laundering (AML) requirements.

7. **Logging and Monitoring:**

Implement robust logging to track user activities and system behavior.
Set up monitoring tools to detect and respond to security incidents.

Scalability:
Design the application to be horizontally scalable, allowing it to handle increased user loads and demand. Consider containerization (e.g., Docker) and orchestration (e.g., Kubernetes) for managing scalability.

Cloud Services:
AWS or AZURE

Data Protection and Privacy:
Adhere to data protection laws and regulations, such as GDPR, to safeguard user data and privacy.

APIs:
API documentation for OpenAPI to is above.