Project Overview:

Model Definitions:

For the Mini Banking Application, we will define three main entities: User, Account, and Transaction.

User Model

Fields:

- id (UUID): A unique identifier for the user.
- username (String): The user's chosen username.
- password (String): The user's encrypted password.
- email (String): The user's email address.
- createdAt (LocalDateTime): The user’s create date.
- updatedAt (LocalDateTime): The user’s update date.

Users can have more than one account.

Account Model

Fields:

- id (UUID): A unique identifier for the account.
- number (String): A unique account number.
- name(String): A unique account name.
- balance (BigDecimal): The current balance of the account.
- createdAt (LocalDateTime): The account’s create date.
- updatedAt (LocalDateTime): The account’s update date.

Accounts belong to only one User.

Transaction Model

Fields:

- id (Long): A unique identifier for the transaction.
- from (Account): The account ID from which money is being transferred.
- to (Account): The account ID to which money is being transferred.
- amount (BigDecimal): The amount of money being transferred.
- transactionDate (LocalDateTime): The date and time when the transaction was initiated.
- status (Enum): The status of the transaction (e.g., "SUCCESS", "FAILED").

RESTful Endpoints (Spring Boot):

The application will expose several RESTful endpoints to support user registration, authentication, account management, and transaction operations. RESTful API should support Swagger. Here's a breakdown of the endpoints:

User Management

1. Register a New User
    - POST /api/users/register
    - Registers a new user with username, password, and email.

2. User Login
    - POST /api/users/login
    - Authenticates a user and returns a JWT for accessing protected endpoints.

Account Operations

1. Create Account
    - POST /api/accounts
    - Creates a new account for the authenticated user.

2. Search Accounts
    - POST /api/accounts
    - Search accounts for the authenticated user.
    - Accounts should be filterable on number and name.

3. Update Account
    - PUT /api/accounts/{id}
    - Updates the selected account for the authenticated user.

4. Delete Account
    - DELETE /api/accounts/{id}
    - Deletes the selected account for the authenticated user.

5. View Account Details
    - GET /api/accounts/{id}
    - Retrieves details of a specific account, including the balance. Access is restricted to the account owner.

Transaction Operations:

1. Initiate Money Transfer
    - POST /api/transactions/transfer

- Transfers money from one account to another.
- Transfers can occur simultaneously.

2. View Transaction History
    - GET /api/transactions/account/{accountId}
    - Retrieves the transaction history for a specified account. Access is restricted to the account owner.

Frontend (React):

User Authentication:

- Implement login and registration forms.
- Use JWT tokens for handling user authentication and maintaining session state.

Banking Features:

- Account creation and management. (CRUD)
- Search should be done using query parameters on the frontend.
- Displaying account balance and details.
- Transferring money between accounts, including form validation and feedback on transaction success or failure.
- Viewing transaction history.

API Integration:

- Use Axios or Fetch API to interact with the backend, ensuring secure transmission of data, especially for authentication and money transfer operations.

State Management:

- Apply Context API or Redux or Zustand etc. for managing application state, focusing on user authentication status, account information, and transaction history.

Routing and Navigation:

Utilize React Router for navigating between different pages like login, account details, transfer money, and transaction history.

Submission Guidelines:

1. Your code should be submitted via a GitHub repository.
2. Include a README file with setup instructions and any necessary documentation.
3. Ensure code quality by following best practices and including meaningful comments.