SDEV200 Java Project Proposal – Austin Martin

**Purpose of the App**

For my final project, I’m planning to build a web application that simulates basic banking features. The goal is to create a platform where users can manage their finances by performing actions such as checking their balance, making deposits and withdrawals, and viewing their transaction history. The app will let users create accounts, log in securely, and get a clear overview of their financial status. This project aims to provide a simple yet effective tool for personal finance management while also giving me hands-on experience with Java and Spring Boot.

**How the App Will Look to the User**

The user interface will be straightforward and intuitive, consisting of several main screens:

1. **Login Screen**: This is where users will enter their credentials to access their account.
2. **Dashboard**: This screen will display the user’s account balance and recent transactions in an easy-to-read format.
3. **Transaction Screen**: Users will use this screen to deposit or withdraw money, with forms to enter transaction details.
4. **Account Management Screen**: This will allow users to view and update their personal information.
5. **Error and Notification Screen**: Any errors or important notifications (like successful transactions) will be displayed here.

The design will focus on simplicity and ease of navigation, ensuring that users can easily interact with all features of the app.

**Classes Expected for the App**

1. **User**
   * **Fields**: userId, username, password, email, account (linking to an Account object)
   * **Methods**: getUserId(), getUsername(), setUsername(), getPassword(), setPassword(), getEmail(), setEmail(), getAccount(), setAccount()
2. **Account**
   * **Fields**: accountNumber, balance, accountHolder (linking to a User object), transactionHistory (list of Transaction objects)
   * **Methods**: getAccountNumber(), getBalance(), setBalance(), getAccountHolder(), setAccountHolder(), getTransactionHistory(), addTransaction(Transaction transaction)
3. **Transaction**
   * **Fields**: transactionId, amount, transactionType (deposit or withdrawal), date, associatedAccount (linking to an Account object)
   * **Methods**: getTransactionId(), getAmount(), setAmount(), getTransactionType(), setTransactionType(), getDate(), setDate(), getAssociatedAccount(), setAssociatedAccount()

**Relationships Between Classes**

* **User** and **Account**: Each User has exactly one Account, so there’s a one-to-one relationship between them.
* **Account** and **Transaction**: An Account can have multiple Transactions, which means there’s a one-to-many relationship between Account and Transaction.

**Fields in the Classes**

1. **User**: userId, username, password, email, account
2. **Account**: accountNumber, balance, accountHolder, transactionHistory
3. **Transaction**: transactionId, amount, transactionType, date, associatedAccount

This setup ensures that the app can manage user information, track account activities, and handle transactions efficiently. It also gives a clear structure for how the application will function and interact with users.

**Conclusion**

Using Spring Boot, I plan to build the backend of this application, which will help me learn more about building secure and scalable web apps. The project will focus on both functionality and user experience, making sure it’s easy to use while also handling all the necessary financial operations.