Backend Mini Project Documentation

Overview:

The aim of this project is to design and write a simple micro service to mimic a digital lending platform such that one can request a loan offer and he/she is presented with one or more loan offers based on the customer loan maximum qualification. From the offers presented, a customer can choose and accept one of them.

Tools and Technologies Used

- Java
- Spring Boot
- Maven (Build tool)
- PostgreSQL (Database)
- Springdoc open-api (API documentation)

Steps to Reproduce the Project:

- Clone the project from the repository.
- Make sure you have at least Java 11 installed, as this project was compiled using Java 11.
- Set up a new database locally and change the values stored in the application.yml file to match the database user, password and url.
- Build the project, and Maven should import the dependencies specified in the pom.xml file of the project.
- Run the main class, *LendingPlatformApplication* to start the server.
- After a successful build, navigate to *http://localhost:8080/swagger-ui/index.html/* to access the documentation for the project.

User Walk Through of the Application:

Users sign up on the app using their first name, last name, email, phone number and password. Upon successful registration, an account is automatically created which is linked to their phone number. Once signed in, users can then access their account which has a maximum credit limit of 10,000 hardcoded while signing up and an initial balance of 0.

Users can then check to see which of the loan offers they are eligible for. Whatever is sent back as a response to the user, is dependent on their credit limit. This implies that users are eligible for only loan offers with a maximum allowable amount less than or equal to the credit limit of their account.

When a loan offer, which the user is eligible for, has been selected, the amount associated with that loan product is then calculated and then added to the current amount in the user's account. If the total sum is greater than the maximum credit limit, an error is thrown to the user to inform them that their credit limit has been exceeded.

However, if the total sum is under the maximum credit limit, it is then added to the amount in the user's account.

Design Choices:

The application uses Postgresql for the database, although any other SQL databases could be used. The choice of using an SQL database was to access relationships between tables using foreign keys. A NoSQL database could be used too, although the work around that would be to model the tables differently because of how you would need to access them.

The database comprises 4 tables, namely:

- **Customer**
  This table stores all personal information related to a user. It contains information such as the first and last name, email address, phone number etc.

- **Account**
  This table stores all account information such as the current amount, maximum credit limit etc related to a user. It is a One-to-one mapping with the User table.

- **ProductLoan**
  This table stores all loan products available on the platform. As per the requirements, only 2 are available and so, these were manually seeded into the database.

- **ProductRequest**
  This table keeps track of loan requests which have been made by users over time. It stores the id of the account that made the request, the id of the loan product in question and the timestamp of when the request was made.

Limitations:

- Users can not have more than one account, because an account is tied to their phone number which is inherently unique to each account.

- Also, for every user account created, a maximum loan value of 10,000 was hard-coded.

- Users could also request for loans as many times as possible, so far their credit limit has not been exceeded.

Possible Solutions/Improvements:

- In traditional bank accounts and real world applications, a user can have multiple accounts. We could also embrace that method in our application by removing the constraint where an

account is tied to a user's phone number, and instead extend it such that multiple accounts with the same phone number can exist, since the phone number belongs to a single user afterall.

- We could also improve the application by keeping track of their credit rating. This works such that whenever a user pays back a loan before the due date, their credit rating is improved, which also significantly increases their maximum credit limit, and makes them eligible for higher loans next time.

- We could also prevent accounts from spamming the app with loan requests by limiting the number of loan requests you can make within a given period of time.