


Loaning-API specifications

Overview:

The lending platform microservice is designed to meet the business requirements and technical constraints using a well-defined design process and appropriate design choices. The microservice uses atomic transactions to ensure concurrency and atomicity, separation of concerns to improve maintainability and scalability, and seed data to simplify data population. Appropriate error handling mechanisms are used to ensure the smooth functioning of the service. The design process and choices are documented to enable future enhancements and modifications.

Instructions on running the app

Find the postman collection [here](#) . 

After running the service (as per README) ,follow the steps below to use the service 🌟:

- **GET:{{api}}/loans/{userId}/request_loan_products**: List all available loan products for selection
- **POST:{{api}}/loans/accept_loan_offer**: Within the body of this endpoint, it expects the **userId** of the loan recipient , **loanProductId** which should be available in the **request_loan_products** endpoint. It also expects an **accept** value which is a boolean value (**true/false**)
- **GET:{{api}}/loans/{userId}/user** : This endpoint is used to retrieve the current active loans associated with a particular user.
- **POST:{{api}}/loans/pay_loan**: This endpoint is used to make a payment towards a specific loan. Within the body of this endpoint, it expects the following parameters :**userId**: The ID of the user making the payment **amount** : The amount being paid towards the loan successful payment towards a loan will reduce the outstanding balance of the loan by the specified amount.

Design Choices:

Separation of Concern:

- a. The application logic is separated into three layers - presentation layer, service layer, and data access layer.
- b. Each layer is responsible for a specific concern, making it easier to test, maintain, and scale the application.

Atomic Transactions:

- a. Whenever a user accepts a loan, multiple records need to be updated - user wallet, company wallet, and loan entry.
- b. To ensure concurrency and atomicity, all updates will be performed in a single transaction.

Use of Seed Data:

- a. Assumed the existence of a company float, which is debited when loaning out money and credited back with profit upon loan repayment.
- b. Seed data is included in the database to populate the company float, user creation, and loan product population.

Choice of Technology Stack:

- a. The microservice will be developed using Node.js, Express.js, and PostgreSQL.
- b. These technologies are well-suited for building scalable and performant microservices.

Error Handling:

- a. Appropriate error messages and status codes will be returned to the client in case of failures.
- b. The logs will be used to capture any errors or exceptions for debugging purposes.

Assumptions:

A customer can only accept one loan offer at a time.

The loan amount cannot exceed the maximum allowable limit of the loan product.

The interest percentage and tenure for each loan product are fixed and cannot be modified.

Mobile wallet balances are stored in a separate system and will be queried using an API.

SMS or email notifications will be implemented using a logging mechanism.

Pitfalls and Improvements:

Pitfalls:

- a. The use of atomic transactions can impact performance if there are frequent updates to the database.
- b. The reliance on seed data can cause inconsistencies if the data becomes outdated or corrupted.
- c. The use of logging for notifications can result in delayed or unreliable delivery.

Improvements:

- Use of a message queue system for notifications to improve delivery and reliability.
- Use of caching mechanisms to reduce database queries and improve performance.
- Use of a scheduler to automate loan payment deductions on the due date to avoid manual intervention.