Assignment for Internship - Backend

Credit Approval System

In this assignment you will be working on creating a credit approval system based on past data as well as future transactions, the goal of this assignment is to assess the proficiency with the JavaScript/NodeJS stack, using background tasks as well handing operations on Databases.

1. Setup and Initialization

a) Setup

- For this assignment, please use NodeJS with Express Web framework package.
- There is no requirement to make a frontend for the application.
- You need to build appropriate data models for the application
- The entire application and all its dependencies should be dockerized.
- You must logically persist the data using MySQL DB / PostgreSQL DB and fetch it via using SQL.

b) Initialization

You are provided with a “customer_data.xlsx” which is a table of the existing customers with the following attributes

- customer_id
- first_name
- last_name
- phone_number
- monthly_salary
- approved_limit
- current_debt

Also, you are provided with “loan_data.xlsx” which is a table of past and existing loans by customers with the following attributes

- customer id
- loan id
- loan amount • tenure
- interest rate
- monthly repayment (emi)
- EMIs paid on time
- start date
- end date

Ingest the provided data into the initial system using appropriate background workers for the matching tech-stack

An async celery task must be triggered via this API to ingest the data.

2. API-

you need to build the following API endpoints with appropriate error handling for and status codes for each.
Use compound interest scheme for calculation of monthly interest.

• /register

Add a new customer to the customer table with approved limit based on salary using the following relation:

- approved_limit = 36 * monthly_salary (rounded to nearest lakh)

a) Request body

Field	Value
first_name	First Name of customer (string)

last_name	Last Name of customer (string)
age	Age of customer (int)
monthly_income	Monthly_income of individual (int)
phone_number	Phone number(int)

b) Response body

Field	Value
customer_id	Id of customer (int)
name	Name of customer (string)
age	Age of customer (int)
monthly_income	Monthly_income of individual (int)
approved_limit	Approved credit limit (int)
phone_number	Phone number (int)

- ## /check-eligibility

Check loan eligibility based on credit score of customers (out of 100) based on the historical loan data from “loan_data.xlsx”, consider the following components while assigning a credit score:

i. Past Loans paid on time
ii. No of loans taken in past
iii. Loan activity in current year
iv. Loan approved volume
v. If sum of current loans of customer > approved limit of customer, credit score = 0

You must come up with a working scheme assigns a credit score to each user based on above listed components.

Note: - Credit Score cannot be fractional

Based on the credit score of the customer, approve loans as per the following:

- If credit score > 50, approve loan
- If 50 > credit score > 30, approve loans with interest rate > 12%
- If 30> credit score > 10, approve loans with interest rate >16%
- If 10> credit score, don't approve any loans
- If sum of all current EMIs > 50% of monthly salary, don't approve any loans
- If the interest rate does not match as per credit limit, correct the interest rate in the response, i.e suppose credit score is calculated to be 20 for a particular loan and the interest_rate is 8%, send a corrected_interest_rate = 16% (lowest of slab) in the response body, corrected

a) Request body

Field	Value
customer_id	Id of customer (int)
loan_amount	Requested loan amount (float)
interest_rate	Interest rate on loan (float)
tenure	Tenure of loan (int)

b) Response Body

Field	Value
customer_id	Id of customer (int)
approval	can loan be approved (bool)
interest_rate	Interest rate on loan (float)

corrected_interest_rate	Corrected Interest Rate based on credit score, same as interest rate if the interest rate matches the slab (float)
tenure	Tenure of loan (int)
monthly_installment	Monthly installment to be paid as repayment (float)

- ## /create-loan

Process a new loan based on eligibility. a) Request body

Field	Value
customer_id	Id of customer (int)
loan_amount	Requested loan amount (float)
interest_rate	Interest rate on loan (float)
tenure	Tenure of loan (int)

b) Response Body

Field	Value
loan_id	Id of approved loan, null otherwise (int)
customer_id	Id of customer (int)
loan_approved	Is the loan approved (bool)
message	Appropriate message if loan is not approved (string)
monthly_installment	Monthly installment to be paid as repayment (float)

• /view-loan/loan_id

View loan details and customer details a)
Response Body

Field	Value
loan_id	Id of approved loan (int)
customer	JSON containing id, first_name , last_name, phone_number, age of customer (JSON)
loan_amount	Is the loan approved (bool)
interest_rate	Interest rate of the approved loan (float)
monthly_installment	Monthly installment to be paid as repayment (float)
tenure	Tenure of loan (int)

• /make-payment/customer_id/loan_id

- Make a payment towards an EMI
- EMI amount should be recalculated if the Amount being paid by the User is less/more than the due installment amount.
- Appropriate error handling should be done

• /view-statement/customer_id/loan_id

- View statement of a particular loan taken by the customer.
- Appropriate error handling should be done.
- Response Body (list of loan items, each loan item will have the following body)

Field	Value

customer_id	Id of the applicant of the loan
loan_id	Id of approved loan (int)
principal	Principal amount of the loan
interest_rate	Interest rate of the approved loan (float)
Amount_paid	Amount repaid by the applicant towards the loan
monthly_installment	Monthly installment to be paid as repayment (float)
repayments_left	No of EMIs left (int)

3. General Guidelines

- Candidate will be judged on ease of running the code and best REST API practices and use of proper status codes is necessary.
- Provide a name to the project. Strict plagiarism checks will be imposed, and any copy found will result in disqualification.
- Ensure code quality, organisation and segregation of responsibilities.
- Adding unit tests although is not necessary, will be considered for bonus points.
- The assignment should be submitted within 4 days.
- The entire application and all its dependencies like DB should be dockerized and should run from a single docker compose command.
- Please submit the GitHub link of the repository.