Assessment for Backend [SDE - Intern].

Dear Candidate,

Thank you for your interest. As part of our hiring process, you shall be required to submit the assessment below. Please note that the deadline to submit your responses is in the next **48 Hours.**

For any questions, please feel free to reach out.

Problem Statement
You will be tasked with crafting a Credit Service designed to facilitate efficient lending by a Loan provider to users. This Credit Service should encompass the following key functionalities:

- Disbursement (Lend money to the user against his/her credit card so that he/she can pay off the debt again the credit card)
- REPAYMENT (Receiving Payments from the user, which can either clear the minimum due of a user or reduce the principal amount if the payment exceeds the minimum due).
- BILLING, for each user billing needs to be done each month till the time the account is not closed for that user, and an Account can be closed only when the user has a principal balance of $0 and there is no interest amount remaining against that user's loan.
    - Keep the billing date 30 days after the user's account is created

MIN DUE FORMULA: (Principal Balance*3%) + (APR accrued on a daily basis for that billing cycle)
Min due represents the amount that the user needs to pay for a billing cycle, it's the same as the EMI amount for each month.
Daily apr accrued formula = round(apr / Decimal('365')**,** 3)

APR represents the interest rate assigned to a loan

Billing for a user also needs to be done, for each billing cycle you need to map the PAYMENT received against that BILL for a user. The BILLING cycle duration is a month, after each month BILLING needs to be done again for a user.
Rules to be followed for BILLING:

- Interest needs to be accrued on a daily basis. On the billing day, the interest accrued for each day will be used to calculate the min_due amount.
- The due date will be 15 days from the billing date, for example, if the billing date is the 2nd of each month then the due date will be the 17th of that month.

Rules to be followed for PAYMENT:

- If the user doesn't pay the full min_due of past billing cycles maintain the past due also for that user. At the time of PAYMENT, 1st past **min_due** amount needs to be paid off then after that new **min_due** amount needs to be paid off.
- There shouldn't be any race conditions while moving the money for the user. There shouldn't be any lost updates in the DB.

To address this challenge, you are required to create a Django application with the specified functionalities and features. This entails designing and implementing pertinent models, views, APIs, workflows, and administrative integrations.
You will have to logically persist the data required inflows and fetch it via Django ORM where needed. While writing Django ORM Queries make sure that there is no query causing N+1 Problem. [Read here about the N+1 Problem in Django]

Your assessment will be evaluated on various criteria, including but not limited to model creation, authentication management, API development, serialization and deserialization, workflow handling, adherence to object-oriented programming principles, code clarity, project execution, Async task handling, or Celery Task handling.

Kindly assign a name to your project. Be aware that strict plagiarism checks will be enforced, and any instances of copying will result in disqualification.

Additionally, a .csv file containing details of user savings account transactions will be furnished with the application.

Field Definitions:

- AADHAR ID: A unique user identifier
- Date: The date of the transaction
- Amount: The transaction amount
- Transaction_type: This can be DEBIT or CREDIT, where CREDIT represents an amount added to the account and vice versa. Please note that sample data is included, and you can supplement it to validate your processes.

Required Functionalities:

- Implement a POST API View for registering a user who is availing the loan.
- Endpoint: **/api/register-user/**
- Response Codes: 200 (Success), 400 (Error)
- View: Create a user with the following details:
  - Name
  - Email
  - Annual Income
- Initiate an asynchronous Celery task through this API to calculate the user's credit score.
- The credit score calculation should be conducted using the transactions from the provided CSV file.
- The credit score should fall within a range of 300 to 900; it cannot be fractional.

Fractional.
In calculating the credit score, the total account balance, which is determined as the sum of (CREDIT - DEBIT) across all transactions for a specific user, follows these rules:

- If the value is greater than or equal to 1,000,000 (Rs. 10,00,000), the credit score reaches its maximum value of 900.
- Conversely, if the value is less than or equal to 10,000 (Rs. 1,00,000), the credit score assumes its minimum value of 300.
- For any intermediate account balance value, the credit score adjusts by 10 points for every Rs. 15,000 change in the account balance.
- As an example, for an account balance of Rs. 5,68,450, the corresponding credit score would be 670.

Request Fields

- Aadhar ID: Unique User Identifier already generated and the same is given in csv. ● name
- email_id
- annual_income

Response Fields

- Error: None in case of success. Error string in case ingestion was not successful stating failure reason
- unique_user_id (UUID value) generated for the given User

POST API View to apply & initiate a Loan again Credit Card to pay off the debt of a credit card

Endpoint: /api/apply-loan/

- Response Codes: 200 (Success), 400 (Error)
- View: Create a Credit Card Loan Application for a user with all necessary details, adhering to the conditions outlined below. Please note that user registration should precede the loan application.
- Currently, the system only supports Credit Card loans.
- The loan application will only be processed if the user's credit score is equal to or greater than 450. Loans will not be disbursed if the user's credit score is not found in the system.
- The user's annual income must be equal to or greater than Rs. 1,50,000.
- The requested loan amount must fall within specified limits, specifically, it must not exceed Rs. 5000. Applications requesting more than Rs. 5000 in loan amount should be declined.

EMIs calculation

- Interest begins accruing from the day following the loan disbursal.
- All due amounts must be repaid within the specified tenure.
- All Equated Monthly Installments (EMIs) should have identical amounts, except for the final EMI, which may be less than the others.
- It's important to note that the last EMI covers all outstanding dues.
- Each EMI includes a portion of the principal amount due, calculated according to the formula provided for the minimum due.
- Interest is calculated on the principal amount for the respective month.
- The EMI should not exceed 20% of the user's monthly income.
- The interest rate must be equal to or greater than 12%.
- The cumulative interest earned for a month should surpass Rs.50.
- All EMIs will be due on scheduled dates.

Request Fields

- unique_user_id: A distinctive User Identifier (UUID)
- loan_type: The type of loan, as previously mentioned
- loan_amount: The loan amount in rupees
- interest_rate: The interest rate as a percentage
- term_period: The duration for repayment in months
- disbursement_date: The date of loan disbursal

Response Fields

- Error: None in the event of a successful operation. In case of an unsuccessful operation, an error string will be provided, indicating the appropriate reason.
- Loan_id: A unique identifier for identifying the initiated loan.
- Due_dates: A list comprising objects specifying EMI dates and corresponding amounts. Each object includes:
    - Date: The EMI date
    - Amount_due: The amount due on that date.

POSTAPI to register payment made

- Endpoint: /api/make-payment/
- Response Codes:** 200 (Success), 400 (Error)
- View: Receive and process data pertaining to payments made towards an Equated Monthly Installment (EMI).
- Payment Rejection Criteria:
    - Reject payment if it has already been recorded for the specified date.
    - Reject payment if previous EMIs remain unpaid.
    - If the amount paid by the user is less or more than the due installment amount, recalculate the EMI amount.
- Request Fields:
    - Loan_id: A unique identifier used to identify the loan.
    - Amount: The payment amount, varies according to the user.
- Response:
    - Error: Null (None) in the absence of errors. An error string will be provided in the case of a failure.

GET API to fetch the transaction statements and future dues

- Endpoint: /api/get-statement/
- Response Codes:** 200 (Success), 400 (Error)
- View:
    - If the loan does not exist or is closed, an error will be returned.
    - For a valid loan, the response will include:
    - A statement detailing the amount paid, with the following specifics. For simplicity, assume all transactions (including interest calculations and payments) are scheduled on the 1st of each month:
    - Principal due
    - Interest on the principal due for that month
    - Repayment of principal
    - A statement listing all upcoming EMIs.

Request fields

- Loan_id: Loan for which details are to be fetched

Response fields

- Error: None if no error. Error String in case of failure
- Past_transactions: Empty list for no past transactions. For valid transactions, each object contains:

 - Date

- Principal

- Interest
- Amount_paid

- Upcoming_transactions: List containing objects of EMI dates and amount. Each object contains:
  - Date: EMI date
  - Amount_due: Amount due on that date.

Cron Job to initiate the billing for a user:
Run a job on each day, and fetch the users for whom billing needs to be done. Once the list of users is available, start the billing process.

Billing Process is as follows:

- A billing cycle will be 30 days
- The difference between the billing date and the due date will be 15 days, which means the due date will be 15 days in the future from the billing date.
- By the due date, the user should pay off the min due amount.
- Min due amount formula has been explained above.
- Create a table to store Billing Details and DuePayments details.
- **NOTE**: Account Balance movement should be atomic, free from race conditions, and there shouldn't be any lost updates.

All The Best