Credimate Loan Management System

Name : Aarushi Verma

1. Overview

Credimate is a web-based Loan Management System developed using Django. The system is designed to help a loan service company efficiently manage the entire loan process, from user registration and loan applications to the calculation of credit scores and payment tracking.

The application allows users to apply for various types of loans, such as personal, home, education, and car loans, based on their credit score and annual income. The credit score is computed asynchronously using user transaction history provided in a .csv file. Once the user is registered, they can apply for a loan, and the system will calculate the EMIs (Equated Monthly Installments) and track payments until the loan is fully repaid.

This documentation outlines the structure, setup, and functionality of the system.

2. Technologies Used

The system has been developed using the following technologies:

- **Django 5.1.7**: The primary web framework.
- **Celery**: Used for handling asynchronous tasks, specifically for credit score calculation.
- **Redis**: Broker for Celery to manage background tasks.
- **SQLite**: Default database for storing user and loan data.
- **Django REST Framework (DRF)**: For building REST APIs for communication between the client and server.
- **Python 3.x**: The programming language used for backend development.
- **CSV**: Transactional data for calculating credit scores.

3. Project Structure

The project structure is organized to separate concerns across different features such as user management, loan applications, payments, and loan statements. Below is an overview of the directory structure:

credimate/

- **credimate/**
    - __init__.py
    - celery.py
    - settings.py
    - urls.py
    - asgi.py
    - wsgi.py
- **users/**

  - models.py
  - views.py
  - serializers.py
  - tasks.py
  - urls.py
- **loanapp/**
  - models.py
  - views.py
  - urls.py
- **payments/**
  - models.py
  - views.py
  - urls.py
- **loanstatement/**
  - models.py
  - views.py
  - urls.py
- Transactions.csv
- db.sqlite3

4. Application Flow and Features

4.1 User Registration and Credit Score Calculation

- **Endpoint**: POST /api/register-user/
- **Purpose**: To register a new user and trigger an asynchronous task to calculate the user's credit score based on transaction data from a .csv file.
- **Request Fields**:
  - aadhar_id: Unique Aadhar ID (provided in the .csv file).
  - name: Full name of the user.
  - email_id: User's email address.
  - annual_income: User's annual income in INR.
- **Response Fields**:
  - error: None if successful. Otherwise, an error message is returned indicating the reason for failure.

    - unique_user_id: A unique UUID generated for the user.

The **credit score calculation** is triggered asynchronously via Celery. The system reads the user's transaction history from the .csv file and computes the total balance (sum of CREDIT - DEBIT transactions). The credit score is assigned based on the following logic:

- **Total balance ≥ ₹1,000,000** → Credit score = **900**
- **Total balance ≤ ₹100,000** → Credit score = **300**
- **Intermediate balance** → Credit score increases by 10 points for every ₹15,000 increase in balance.

Example:

- For a balance of ₹5,68,450, the credit score would be calculated as 670.

4.2 Loan Application

- **Endpoint**: POST /api/apply-loan/
- **Purpose**: To apply for a loan after the user has been registered.
- **Request Fields**:
    - unique_user_id: Unique user identifier (UUID).
    - loan_type: Type of loan being applied for (e.g., "Car", "Home", "Education", "Personal").
    - loan_amount: Amount of loan requested in INR.
    - interest_rate: Interest rate for the loan.
    - term_period: Repayment term in months.
    - disbursement_date: Date on which the loan is disbursed.
- **Loan Conditions**:
    - User's credit score must be ≥ 450.
    - User's annual income must be ≥ ₹1,50,000.
    - Loan amount must be within the specific limits for each loan type.

EMI Calculation:

- The loan is disbursed with interest starting the next day.
- EMIs are calculated based on the loan amount, interest rate, and tenure. The EMI amount is capped at 60% of the user's monthly income.
- Interest rate must be at least 14%.
- All EMIs are due on the 1st of each month, with the last EMI settling all remaining dues.

4.3 Payment Registration

- **Endpoint**: POST /api/make-payment/
- **Purpose**: To record a payment made by the user towards an EMI.
- **Request Fields**:
    - loan_id: Unique loan identifier.

    - amount: Amount paid towards the EMI.
- **Validation**:
    - Payments will be rejected if the EMI has already been paid for that month.
    - Payments will be rejected if earlier EMIs are still pending.
    - EMI amounts will be recalculated if the payment amount is different from the expected due amount.

4.4 Loan Statement and Future Dues

- **Endpoint**: GET /api/get-statement/
- **Purpose**: To retrieve the statement of all payments made and upcoming dues.
- **Request Fields**:
    - loan_id: Unique loan identifier.
- **Response Fields**:
    - error: None if successful. Otherwise, an error message.
    - past_transactions: A list of past transactions, each containing:
        - date: The date of payment.
        - principal: The principal amount for that transaction.
        - interest: The interest on the principal for that transaction.
        - amount_paid: The total amount paid for that transaction.
    - upcoming_transactions: A list of upcoming EMI dates and the corresponding amounts due.

5. Installation and Setup

5.1 Requirements

To set up the project, ensure the following software is installed:

- Python 3
- Django 5.1.7
- Celery
- Redis
- SQLite
- Docker
- CSV
- Pandas

5.2 Setup Instructions

1. Clone the project repository.
2. Install the required dependencies:

```
pip install -r requirements.txt
```

3. Set up Redis as the Celery broker:

```
redis-server
```

4. Run the migrations to set up the database:

```
python manage.py migrate
```

5. Start the Celery worker in a separate terminal:

```
celery -A credimate worker --loglevel=info --pool=solo
```

6. Run the Django development server:

```
python manage.py runserver
```

6. Component: users App

6.1 Purpose

The users app is responsible for managing the registration of users who want to apply for a loan. Upon registration, an asynchronous Celery task is triggered to compute the user's credit score using a pre-provided transaction history (Transactions.csv). This score is later used to determine loan eligibility.

6.2 Data Models

UserProfile

This model represents a user applying for a loan. It includes basic user details along with financial information.

- id: A unique UUID automatically assigned to each user.
- aadhar_id: Acts as a foreign key reference to transactions and ensures linkage with the CSV.
- name: Full name of the user.
- email: User’s email; must be unique.
- annual_income: Annual income in INR (positive integer).
- credit_score: Nullable integer field populated by an asynchronous task after registration.

Transaction

This model is linked to a user via a foreign key and holds their individual transaction entries.

- user: Foreign key linked to the registered user (UserProfile).
- date: Date of transaction.

- amount: Amount involved in the transaction (positive/negative based on type).
- transaction_type: Indicates whether the transaction was a CREDIT (deposit) or DEBIT (withdrawal).

6.3 Serializers

UserRegistrationSerializer

This serializer is used to validate and transform data for user registration. It ensures required fields are provided and match the schema expected by the UserProfile model. It does not include the credit_score field as that is computed asynchronously.

6.4 Views

RegisterUserAPIView

This API view handles the POST request to register a user. Upon successful validation:

- A new user is created.
- The calculate_credit_score_task is dispatched asynchronously using Celery, passing the user's Aadhar ID.
- A UUID is returned in the response for future reference.

In case of invalid input, it returns detailed error messages with a 400 status code.

- Endpoint: /api/register-user/
- Method: POST
- Request: Aadhar ID, name, email, annual income
- Response:
    - unique_user_id: UUID of the user
    - error: None on success or validation error details

6.5 Background Task (Celery)

calculate_credit_score_task

This Celery task is responsible for computing a user's credit score based on their transaction history from the Transactions.csv file. It follows the rules defined in the problem statement:

- Score = 900 if total balance ≥ ₹10,00,000
- Score = 300 if balance ≤ ₹1,00,000
- For values in between:
  Score = 300 + (Rs. over 1L / 15,000) × 10, capped at 900

Steps performed:

1. Reads Transactions.csv using Pandas.

2. Filters rows based on Aadhar ID.
3. Computes balance by subtracting debits from credits.
4. Applies the scoring formula.
5. Updates the credit_score field of the user.

This task is run in the background to offload processing from the main thread and improve scalability.

6.6 URL Configuration

This app exposes a single endpoint through the following URL pattern:

- path('register-user/', RegisterUserAPIView.as_view())

7. Component: loanapp App

7.1 Purpose

The loanapp module is responsible for managing the loan application process. It handles eligibility checks, validates loan terms, computes EMI schedules based on loan parameters, and saves the loan data into the system if all conditions are met.

7.2 Data Models

Loan

This model stores all key attributes of a user's loan application.

- id: A unique UUID to identify each loan.
- user: Foreign key linking to the user applying for the loan.
- loan_type: Type of loan — must be one of the supported types (Car, Home, Education, Personal).
- loan_amount: Requested loan amount.
- interest_rate: Rate of interest in percentage.
- term_period: Duration of repayment in months.
- disbursement_date: Date when the loan is officially issued.
- emi_schedule: A JSON field storing monthly EMI breakdown (amount due and due dates).
- is_closed: A boolean indicating whether the loan is closed (fully repaid or terminated).

Payment

Though implemented here, this model is for capturing payment instances toward a loan. This is likely used by the payments app.

- loan: Foreign key to the associated loan.
- date: Auto-filled field capturing the payment date.
- amount: Amount paid in the transaction.

7.3 Serializer

LoanApplicationSerializer

A serializer used to validate and parse input data when applying for a loan. It includes all fields of the Loan model except:

- emi_schedule – which is auto-calculated.
- is_closed – defaults to False and is managed internally.

7.4 EMI Generation Utility

Function: generate_emi_schedule()

This utility method is the core of EMI computation. It ensures all business rules for EMI calculation are met:

- Converts the annual interest rate to monthly.
- Computes fixed EMI using standard amortization formula.
- Ensures EMI does not exceed 60% of the user's monthly income.
- Ensures the total interest earned by the lender is at least ₹10,000.
- Constructs a JSON schedule of EMIs:
  - Equal payments for all months except the last.
  - EMI due on the 1st of every month, starting one month post disbursement.

If any validation fails, it returns None along with an appropriate error message.

7.5 Views

ApplyLoanAPIView

Handles loan application requests. This view performs all the necessary checks and records the loan if approved:

- **Endpoint**: /api/apply-loan/
- **Method**: POST
- **Request Fields**:
  - unique_user_id: UUID of the registered user
  - loan_type: One of 'Car', 'Home', 'Education', 'Personal'
  - loan_amount: Requested amount in INR
  - interest_rate: Interest rate in percentage (minimum 14%)
  - term_period: Repayment period in months
  - disbursement_date: Loan disbursal date (used to schedule EMIs)

- **Validations**:
    - User must exist and have a valid credit score (≥ 200).
    - Annual income must be ≥ ₹1,50,000.
    - Loan type must be among supported ones.
    - Loan amount must not exceed category-specific limits:
        - Car: ₹7,50,000
        - Home: ₹85,00,000
        - Education: ₹50,00,000
        - Personal: ₹10,00,000
    - EMI must not exceed 60% of the monthly income.
    - Interest earned must be more than ₹10,000.
- **Response Fields**:
    - loan_id: Unique ID of the created loan
    - due_dates: List of due dates and EMI amounts
    - error: None on success or validation message on failure

If all validations pass, the EMI schedule is generated and saved along with the loan record.

7.6 URL Configuration

The following URL route is registered for this module:

It is served by the ApplyLoanAPIView view and is responsible for processing all loan applications.

8. Component: payments App

8.1 Purpose

The payments app is responsible for handling EMI payments for active loans. It verifies that users are paying on time and ensures that no payments are made out of sequence. It also dynamically adjusts the EMI schedule if the amount paid deviates from the expected value (either underpaid or overpaid).

8.2 Data Models

Payment

This model records individual payments made toward a specific loan.

- loan: A foreign key referencing the Loan object associated with this payment.
- date: The scheduled EMI date when the payment is considered due.
- amount_paid: The actual amount paid by the user.

Each entry in this model represents a single EMI transaction toward a loan.

8.3 Serializer

PaymentSerializer

This serializer is used to parse and validate data for processing a new payment.

- loan_id: UUID identifying the loan for which the payment is being made.
- amount: The amount paid by the user. This is checked against the expected EMI value.

The serializer ensures that both fields are present and correctly typed before further processing.

8.4 Views

MakePaymentAPIView

This view processes incoming payment requests and updates both the loan status and payment records based on the following logic:

- **Endpoint**: /api/make-payment/
- **Method**: POST

Request Fields

- loan_id: Unique identifier for the loan
- amount: EMI amount being paid

Core Logic

1. **Loan Validation**: Ensures the loan exists.
2. **Payment History Check**: Retrieves previous payments for the loan and determines which EMI is due next.
3. **Out-of-Order Check**:
    - Rejects payment if any previous EMI is unpaid.
    - Rejects if the current EMI is already paid.
4. **Payment Recording**: Creates a new payment record for the corresponding EMI date.
5. **Payment Adjustment Logic**:
    - If the paid amount differs from the scheduled EMI:
        - Calculates the new balance for the remaining term.
        - Redistributes remaining EMIs with updated equal amounts.
        - Ensures the new EMI values maintain the correct total repayment sum.

Response

- **Success**:

Failure: Returns appropriate error messages such as:

    - Loan not found
    - Previous EMIs unpaid
    - EMI already paid
    - Validation errors in request

8.5 URL Configuration

The following route is provided by the payments app:

It is handled by the MakePaymentAPIView and processes EMI payments submitted via POST requests.

9. Component: loanstatement App

9.1 Purpose

The loanstatement app provides a detailed breakdown of a user's loan repayment progress. It consolidates all past payments, calculates how each payment was split between principal and interest, and displays all upcoming EMI obligations. It serves as the backend for a user-visible loan statement dashboard.

9.2 Views

GetLoanStatementAPIView

This view handles POST requests to fetch a comprehensive statement for a specific loan.

- **Endpoint**: /api/get-statement/
- **Method**: POST

Request Fields

- loan_id: UUID of the loan for which the statement is requested.

Core Logic

1. **Loan Validation**:
    - Ensures the loan exists.
    - Rejects if the loan is already marked as closed.
2. **Data Retrieval**:
    - Fetches all past payments (Payment) for the loan.
    - Uses the stored emi_schedule field from the Loan model.
3. **Transaction Processing**:
    - For each paid EMI, calculates:
        - **Interest** = remaining principal × monthly interest rate
        - **Principal** = paid amount - interest
        - Updates remaining principal accordingly.

    - For unpaid EMIs, adds them to a separate list of upcoming transactions.

4. **Statement Summary**:
    - Current outstanding balance (remaining principal)
    - Remaining tenure (number of unpaid EMIs)

Error Scenarios

- Loan not found.
- Loan is already closed.

9.3 URL Configuration

The following route is exposed by this module:

This is managed by GetLoanStatementAPIView, which serves the loan statement logic.