You are provided with a set of three projects. From these, **select any one project and based** on your selection, create database **schemas** that align with the project requirements and the scope.

**Please note:** The project and feature descriptions are given only as **reference**. Your task is limited to creating the **database schemas**.

This activity is designed to test your **database schema creation skills** and will be evaluated before the **Capstone Project** is rolled out.

Table format:

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |

You have to specify the primary keys /foreign keys wherever required.

## **Project 1: Credit Card Management Portal**

Your team has been tasked with building a **Credit Card Management Portal** for a bank that issues credit cards to users and allows them to manage their cards and monitor transactions. The bank also offers Buy Now Pay Later (BNPL) plans for eligible users.

As part of the engineering team, your goal is to create an application that allows customers to:

* View their credit cards and details
* Apply for a new credit card
* Activate or block cards
* Increase or decrease card limits
* Simulate transactions
* View past transaction reports

### **Module 1: View Credit Cards + Activate/Block**

**Backend**

* Create GET /cards/{userId} to return list of cards
* Create PUT /cards/{cardId}/status to update status (active, blocked)
* Define Card entity and repository

**Database**

* Design user and cards table

### **Module 2: Apply for Credit Card + View Requests**

**Backend**

* Create POST /cards/apply to submit the application
* Create GET /cards/applications/{userId} to view status
* Define the CardApplication entity and repository

**Database**

* Create card\_applications table

### **Module 3: Manage Card Limit**

**Backend**

* Create PUT /cards/{cardId}/limit to update limit
* Validate the limit change within the allowed range

**Database**

* Ensure a field for limit exists in cards table

### **Module 4: Simulate & View Transactions**

**Backend**

* Create POST /transactions to simulate transaction
* Create GET /transactions/{cardId} for reports
* Define Transaction entity and service

**Database**

* Create transactions table

### **Module 5: Customer Profile Management**

**Backend**

* Create/update UserProfile table
* REST APIs for:  
  + GET /api/profile/{userId}
  + PUT /api/profile/{userId}
* Add simple validation (e.g., not empty, valid phone/email)

**Database**

* Create user\_profiles table

## **Project 2: Personal Loan Application System**

Your team has been tasked to **develop a full-stack personal loan management** application that supports loan lifecycle management — from customer onboarding and loan applications to approvals, EMI calculations, and support handling. The application should be modular, maintainable, and easy to deploy using containerization technologies.

As part of the engineering team, your goal is to create an application that allows customers to:

* Register and securely log in to their accounts
* Submit personal loan applications
* View the status of their loan applications
* Use an EMI calculator to estimate repayment amounts
* Raise support tickets to address any issues or inquiries related to their loan or account, and track resolution status.

In parallel, the application should provide admin capabilities that allow authorized staff to:

* Review submitted loan applications and approve or reject them based on predefined criteria.
* Monitor and manage support tickets raised by customers
* Access dashboards to view application statistics, repayment performance, and system health.

### **Module 1: User Management and Authentication**

This module deals with secure user registration, login, and role-based access control for both customers and admins.

#### **Backend:**

* Create User entity and repository
* Build REST APIs for:  
  + User registration
  + User login with JWT-based authentication
  + Role assignment (USER, ADMIN)

### **Module 2: Loan Application Submission and Status Tracking**

This module enables users to submit loan applications and track the current status.

#### **Backend:**

* Build RESTful APIs for:  
  + POST /loans/apply – Submit a new loan application
  + GET /loans/user/{userId} – View loan applications for a user
  + GET /loans/{loanId} – View detailed status of a single application

#### **Business Rules (Validation)**

* Income must be above a threshold (e.g., > ₹25,000)
* Credit score must be present and within a valid range (e.g., 300–900)
* Duplicate loan requests within 24 hours are not allowed

#### **Database**

* **CreateTable: loan\_applications**

### **Module 3: Admin Dashboard and Loan Processing Engine**

This module enables the admin to review, approve, or reject loan applications.

#### **Backend:**

**APIs:** Admin APIs to:

* GET /admin/loans/pending – Retrieve all pending loan applications
* PUT /admin/loans/{loanId}/status – Approve or reject a loan (with optional remarks)

#### **Entity Update: LoanApplication**

Add fields according to the module requirements

### **Module 4: EMI Calculator**

This module adds the feature to calculate EMIs based on approved loans, generate repayment schedules, and allow users to view their upcoming payments.

#### **Backend:**

### Extend LoanApplication to include the repayment plan.

### Implement an **EMI calculator service** using business logic (e.g., formula for EMI).

#### **EMI Calculator Logic Formula** EMI = [P × R × (1+R)^N] / [(1+R)^N – 1]

### Where:

* P = Principal
* R = Monthly Interest Rate
* N = Tenure in months

#### **APIs**

* GET /loans/{loanId}/schedule - Get repayment plan for an approved loan
* POST /emi/preview - Simulate EMI value for given input (amount, rate, tenure)

#### **Database**

**Create Table**: RepaymentSchedule

### **Module 5: Customer Support Ticketing System**

This module allow users to raise support requests mimicking a real-world customer service flow.

#### **Backend:**

#### **APIs**

* POST /support – Create support ticket
* GET /support/user/{userId} – View tickets submitted by user

#### **Database**

* **Create Table:** SupportTicket

## **Project 3: Online Investment & Portfolio Tracker**

Build a secure, user-friendly **Online Investment & Portfolio Tracker** platform that allows users to manage their investments, track the performance of various assets (stocks, mutual funds, bonds, etc.), and gain insights into their portfolio health. The system must also support personalized dashboards and portfolio analytics.

### **Module 1: User Registration & Authentication**

#### **Backend APIs**

* POST /auth/register – Register new user (role: USER)
* POST /auth/login – Login and receive a JWT token
* GET /user/profile – Get user profile details
* GET /admin/users – (Admin only) List all users

#### **Business Rules**

* Only admins can access /admin prefixed routes

#### **Database**

* **Create User Table**

### **Module 2: Investment Product Listing & Selection**

#### **Backend APIs**

* GET /investments – List available investment products
* POST /admin/investments – Add new investment (Admin only)
* PUT /admin/investments/{id} – Edit or deactivate a product

#### **Business Rules**

* Each product must have type (e.g., mutual fund, stock), risk level, return rate, and current NAV
* Validation Rules:
  + expectedReturnRate should be a positive number.
  + riskLevel must be one of ["low", "medium", "high"] or defined constants.

#### **Database**

* **Create InvestmentProduct Table**

### **Module 3: Portfolio Management**

#### **Backend APIs**

* POST /portfolio/buy – Simulate investment in a selected product
* POST /portfolio/sell – Sell owned units
* GET /portfolio – View current holdings
* GET /portfolio/transactions – View past transactions

#### **Business Rules**

* Portfolio value = Sum of (units × current NAV) per product
* Selling is not allowed if units < requested
* Investment amount (units × NAV) meets the minInvestment (10000)
* On buying, the average purchase price should be recalculated
* Every buy/sell is recorded as a transaction

#### **Database**

* **Create Portfolio table**
* **Create Transaction table**

### **Module 4: Portfolio Analytics & Insights**

#### **Backend APIs**

* GET /portfolio/summary – Total investment value and performance
* GET /portfolio/allocation – Asset type breakdown
* GET /portfolio/gains – Gain/loss analysis for each asset

#### **Business Rules**

* Calculate absolute return: (current value - invested amount)
* Annualized return approximation using transaction history
* Show allocation % based on asset class (e.g., equity, debt)

#### **Database**

* Use existing Portfolio, Transaction, and InvestmentProduct tables to derive analytics

### **Module 5: Support & Helpdesk System**

#### **Backend APIs**

* POST /support – Raise a support ticket
* GET /support/user – View tickets by user
* GET /admin/support – View all tickets
* PUT /support/{ticketId}/respond – Admin (mock) responds to ticket

#### **Business Rules**

* Each ticket must be linked to a user and optionally an investment
* Status values: OPEN, RESPONDED, CLOSED
* Only unresolved tickets can be updated
* Support Ticket priority levels: LOW, MEDIUM, HIGH – default to MEDIUM if not specified

#### **Database**

* **Create SupportTicket table**