Loan Origination & Approval System — Node.js + React Project

1. Project Overview

This project evaluates your ability to design and implement a full-stack **Loan Origination & Approval System** using **Node.js (Express)** for the backend, **MongoDB (Mongoose)** for the database, and **React** for the frontend.

Purpose: Create a digital loan management system that allows customers to apply for loans, calculates eligibility, and enables loan officers to review, approve, or reject applications.

Learning Goals:

- Design a modular backend using Express + Mongoose.
- Model complex relationships between Customers, Loans, and Officers using MongoDB references.
- Implement secure JWT authentication and role-based access control.
- Build a responsive React frontend that consumes REST APIs.

Key Challenge: Efficiently model the many-to-many relationships between customers, loan officers, and applications, while maintaining consistency in eligibility scoring and approval tracking.

2. Problem Statement

Build a platform where:

- Customers can register, apply for loans, and view their application status.
- Loan Officers can review loan applications, assess eligibility, and approve or reject them.
- The system automatically evaluates loan applications based on defined financial parameters (e.g., income, credit score, amount requested).

All actions should be accessible via **JWT-protected REST APIs**, consumed by a **React frontend**.

3. Actors

Customer — registers, applies for loans, tracks approval status.

Loan Officer — reviews and approves or rejects applications.

Platform (Admin/System) — manages authentication, stores data, calculates eligibility scores, and ensures business logic consistency.

4. System Modules

Auth Module — handles registration, login, and JWT issuance.

Customer Module — manages user details and loan applications.

Loan Module — handles application creation, eligibility scoring, and approval.

Officer Module — allows officers to view and manage pending loan requests.

Each module should have separate router and controller files. Use Mongoose models for schema definitions and data management.

5. Database Schema (Mongoose Models)

Use separate collections with references where appropriate.

5.1 User

```
const User = new Schema({

  name: { type: String, required: true },

  email: { type: String, required: true, unique: true },

  passwordHash: { type: String, required: true },

  role: { type: String, enum: ['CUSTOMER', 'OFFICER'], required: true }
}, { timestamps: true });
```

5.2 Customer

```
const Customer = new Schema({
  userId: { type: Schema.Types.ObjectId, ref: 'User', required: true },
  income: { type: Number },
  creditScore: { type: Number }
});
```

5.3 LoanOfficer

```
const LoanOfficer = new Schema({
  userId: { type: Schema.Types.ObjectId, ref: 'User', required: true },
  branch: { type: String }
});
```

5.4 LoanApplication

```
const LoanApplication = new Schema({
  customerId: { type: Schema.Types.ObjectId, ref: 'Customer', required: true },
  officerId: { type: Schema.Types.ObjectId, ref: 'LoanOfficer' },
  amountRequested: { type: Number, required: true },
  tenureMonths: { type: Number, required: true },
  interestRate: { type: Number },
  status: { type: String, enum: ['PENDING','APPROVED','REJECTED'], default: 'PENDING' },
  eligibilityScore: { type: Number }
}, { timestamps: true });
```

6. Loan Evaluation Logic (MongoDB Aggregation)

Goal: For each loan application, compute an eligibilityScore using income, creditScore, and requested amount.

Suggested Logic:

- Normalize income and creditScore to a 0–1 scale.
- Compute score = (0.6 * creditScoreNorm) + (0.4 * incomeNorm).
- Compare against loan amount threshold.
- If score ≥ threshold → status = APPROVED; else → status = REJECTED.

Implementation: Create a `loanService.evaluateLoan(applicationId)` function that:

1. Fetches application and linked customer data.
2. Computes score.
3. Updates eligibilityScore and status fields.

7. Backend API Requirements

All endpoints (except `/auth/*`) require a valid JWT in the header: `Authorization: Bearer <token>`.

7.1 Auth Module

POST /auth/register — Register as Customer or Officer.

```
{
  "name": "Ravi Kumar",
  "email": "ravi@example.com",
  "password": "P@ssw0rd",
  "role": "CUSTOMER"
}
```

Response: `{ "message": "User registered successfully", "userId": "<ObjectId>" }`

POST /auth/login — Login and receive JWT.

{ "email": "ravi@example.com", "password": "P@ssw0rd" }

Response: `{ "token": "<jwt>", "userId": "<ObjectId>", "role": "CUSTOMER" }`

7.2 Loan Module

POST /loans/apply — Create a new loan application.

```
{
  "customerId": "<ObjectId>",
  "amountRequested": 500000,
  "tenureMonths": 24
}
```

Response: `{ "loanId": "<ObjectId>", "message": "Loan application submitted." }`

GET /loans/:id/status — Fetch status and eligibility score.
Response: `{ "status": "APPROVED", "eligibilityScore": 0.82 }`

7.3 Officer Module

GET /officer/loans/pending — Get all pending loan applications.
POST /officer/loans/:id/review — Approve/Reject loan.

8. JWT Authentication

- Tokens are signed using a secret key stored in environment variables.

- Payload: `{ userId, role, iat }`.
- Middleware validates the token and attaches `req.user`.
- Role-based access ensures only officers can approve/reject loans.

9. React Frontend Requirements

Core Concepts:

- React functional components and hooks (useState, useEffect, useContext)
- Routing with `react-router-dom`
- Axios for API communication with JWT header
- Toast notifications (`react-toastify`)
- Context-based role management

Key Components:

- Login, Register
- Customer Dashboard: Apply Loan, Track Status
- Officer Dashboard: Review Loans, Approve/Reject
- Shared Components: Navbar, Loader, ToastContainer

10. Expected Outcome

Deliver:

- Modular Node.js backend with MongoDB.
- JWT-based authentication.

- Automatic loan eligibility scoring logic.
- React frontend integrated with backend APIs.

11. Submission Requirements

Repository Structure:

/backend

/frontend

Include in README:

- Setup instructions and environment variables
- MongoDB connection and seed data setup
- API documentation with examples
- How to run backend & frontend

Deliverables:

- GitHub repository (backend + frontend)
- README file with setup details
- 5–10 min walkthrough video (architecture, auth, eligibility logic, UI demo)

Good Luck!