# Challenge: Enhanced Loan Application Processing Service

## Overview

Develop a microservice that manages loan applications, including submission, status checks, and admin management. Optionally, add authentication and authorization to secure the service and manage user roles.

## Technical Stack

- **Backend**: NestJS or Express.js with Typescript (mandatory)
- **Database**: PostgreSQL
- **Containerization**: Docker

## Core Functional Requirements

- **Loan Application Management**:
  - `POST /applications`: Submit a new loan application.
  - `GET /applications/{id}`: Retrieve the status of a specific application.
  - `GET /applications`: (Admin only) Retrieve all loan applications.
- **Database Schema**:
  - Design tables for applications and, optionally, users and roles.
- **Docker Integration**:
  - Provide a `Dockerfile` and optionally a `docker-compose.yml` to containerize the application and database.

## Optional Security Features

If you choose to implement authentication and authorization:

- **User Management Endpoints**:
  - `POST /auth/register`: Register a new user account.
  - `POST /auth/login`: Login to receive a JWT token.
- **Role-Based Access Control (RBAC)**:
  - Two roles: `Admin` and `Applicant`.
  - Secure application endpoints based on user roles.

## Deliverables

- **Source Code**: Including backend logic, authentication, and authorization if implemented.

- **Dockerfile**: To containerize the NestJS or Express.js application.
- **docker-compose.yml**: Optional, for setting up the application and PostgreSQL services together.
- **Database Schema**: With migration and seed scripts. Include user and role tables if implementing security features.
- **README**: Setup instructions, API usage, and details on running the service with Docker. Include authentication details if applicable.

# Documentation Guidelines

- **Setup Instructions**: Step-by-step guide on how to run the application using Docker, including environment setup and database initialization.
- **API Endpoints**: Documentation for each endpoint, specifying method, URL, request body, response, and any required headers (e.g., Authorization header for secured endpoints).
- **Authentication Guide**: If implemented, explain the registration and login process, how to obtain a JWT token, and how to use it to access secured endpoints.

# Evaluation Criteria

- **Functionality**: The application meets the core requirements, with optional security features implemented as per the candidate's choice.
- **Code Quality**: Clean, readable, and maintainable code. Best practices in NestJS or Express.js and TypeScript.
- **Database Design**: Logical and efficient schema design.
- **Security Implementation**: If applicable, secure handling of authentication and authorization, including JWT usage and password management.
- **Documentation**: Comprehensive and clear, making it easy for others to set up and use the application.