Banking Management System

Problem Statement:

The Bank Management System is an application for maintaining a person's account in a bank. In this project, I tried to show the working of a banking account system and cover the basic functionality of a Bank Account Management System. It develops a project for resolving a customer's financial applications in a banking environment to meet the needs of an end banking user by providing multiple ways to complete banking chores. Additionally, this project is to provide additional features to the user's workspace that are not available in a traditional banking project. The Bank management system is built on cutting-edge technologies. This project's main goal is to create software for a bank account management system. This project was designed to make it simple and quick to complete previously impossible processes with manual systems which are now possible with this software.

Modules

- Login Module
- Customer Module
- Account Module
- Transactions Module
- Recipient Module
- Appointment Booking Module

Functionalities:

1. Functions of Admin:
    - Login
    - List all customers.
    - Update Bank details of customer.
    - View/ edit customer account details
    - View transaction details of customer
    - Approve or cancel transactions
    - Create or suspend accounts for customer
2. Functions of Customer:
    - Login
    - View profile details
    - Register with bank
    - View account details
    - Edit profile details
    - View transactions
    - Perform transactions like amount withdrawal, amount deposit etc.

Project Development Guidelines:

The project to be developed based on the below design considerations

Backend	• Use Rest APIs (Springboot/ASP.Net Core WebAPI to develop the services
Development	• Use Java/C# latest features • Use ORM with database • Use Swagger to invoke APIs • Implement API Versioning • Implement security to allow/disallow CRUD operations • Message input/output format should be in JSON (Read the values from the property/input files, wherever applicable). Input/output format can be designed as per the discretion of the participant. • Any error message or exception should be logged and should be user-readable (not technical) • Database connections and web service URLs should be configurable • Implement Unit Test Project for testing the API • Follow Coding Standards
Frontend Development	• Use Angular/React to develop the UI • Implement Forms, databinding, validations • Implement Routing and navigations • Use JavaScript to enhance functionalities • Implement External and Custom JavaScript files • Implement Typescript for Functions, Operators. • Any error message or exception should be logged and should be user-readable (and not technical) • Follow coding standards • Follow Standard project structure

Good to have implementation features

- Generate a SonarQube report and fix the required vulnerability
- Use the Moq framework as applicable
- Create a Docker image for the frontend and backend of the application
- Implement OAuth Security
- Implement Logging
- Implement design patterns
- Use JWT for authentication in SpringBoot/WebApi. A Token must be generated using JWT. Tokens must expire after a definite time interval, and authorization must be handled accordingly based on token expiry
- Deploy the docker image in AWS EC2 or Azure VM
- Build the application using the AWS/Azure CI/CD pipeline. Trigger a CI/CD pipeline when code is checked-in to GIT. The check-in process should trigger unit tests with mocked dependencies
- Use AWS RDS or Azure SQL DB to store the data