Date: 6th January, 2023

Project Name: Locker Management System

(Gaurav Malik)

Abstract: Bank Locker Management System is a web-based application that deals with bank lockers that store the valuables things of bank customers. There are different types/Size of locker available, customer can select as per there need or space.

All details of lockers are saved in the database. Bank Locker Management System project is developed using Node JS, Express JS, MongoDB, PostgreSQL with Micro-Service architecture.

Problem Definition: For high level security, we use bank lockers to secure our important documents, expensive jeweler, or cash etc, Hence it has become a very important part for every common human being. To suffer in this world and for a continuous development. The banking sector needs to accommodate a very hinge rise security.

Here in this Micro-service architecture, I am using total 4 Services. Each service has a different task.

1. User
2. Admin
3. Assigned Locker
4. Locker Type

User API :

Here User is act as a Customer, In this API - I am using **MongoDB** as Database with 3 attributes **{ username, email, password }.** If user already in database that means user/customer already have an account otherwise user needs to be Signup first and then canonly be login. Signup means user opening new account in the bank.

Every Sign-in one token will generate and we have to use it as a middleware to access other API's.

Here I am only Focusing Locker Management so, Kept only required information for users i.e., (username, email, password).

Admin API :

Here Admin is act as a Bank Employee/Manager. Using MongoDB as Database in this API with same attributes **{ username, email, password }.** First we have to manually create one admin in the database because nobody can directly signup as a admin.

Only previously existing admin can create/signup for a new admin.

Admin authentication will required as a middleware for creatingnew role as a admin.

Assigned Locker:

In this API there are some roles assigned to Admin and for user I have assigned two roles.

User Roles:

1. GET only own Locker Details
2. POST request to admin to change or update the data **Admin**

Roles:

1. GET all locker details
2. GET locker details with the help of locked-id
3. GET locker details with the help of User-id
4. GET locker details with the help of Locker Number
5. GET locker details with the help of User Name and Email
6. POST locker details as per user request
7. UPDATE locker details as per user request
8. DELETE locker details as per user request

Here Admin can Perform all operations for all Users but, User can only play with own data not others. I am using User and Admin authentication as a middleware to access this API. Without Authentication no one can perform any action on this API.

Locker Type:

This API is only for Admin not for User. In this API all the details regarding lockers will be store. And this is totally depend on **Assigned-Locker** API. If Admin will Post a locker the count of respective locker will be increase by one and if Admin will Delete a locker the count of respective locker will be decrease by one.

For this Logic I have used Fetch API inside Assigned-Locker API.<u>**Admin Roles**</u>**:**

1. Get Locker Type Details
2. POST new Locker Type
3. UPDATE Locker Type

<u>**Fetch Roles**</u>**:**

1. UPDATE increase count of locker by 1
2. UPDATE decrease count of locker by 1

<u>**Security Alert**</u>**:**

For security purpose an alert is created with the help of node mailer, in this as soon as the user tries to Sign-in/Sign-up the individual user will get an alert on this Gmail account that he has signed in successfully, in the same way if any person tries to access someone else account or enters wrong credentials again the individual user will receive a mail from the bank's server.

<u>**Software requirement:**</u> Postman, PostgreSQL, MongoDB, Docker,VS-Code

<u>**Technology:**</u> Node JS, Express JS, JSON web Token, Morgan for Logs, Swagger for Documentation, API Gateway, Mocha & Chai for Unit Testing, Docker for Deployment.