BANK MANAGEMENT SYSTEM

OBJECTIVE:

- **Bank management system for managing customer data**

Users of the System:

- **Admin**
- **Account holder**

Functional Requirements:

- **The Traditional way of maintaining the details of a user in a bank was to enter the details and record them.**

- **Every time the user needs to perform some transactions he has to go to the bank and perform the necessary actions, which may not be so feasible all the time.**

- **It may be a hard-hitting task for the users and the bankers too. The project gives a real-life understanding of the Online Banking System and activities performed by various roles in the supply chain. Here, we provide automation for the banking system through the Internet.**

- **The Online Banking System project captures activities performed by different roles in real-life banking which provides enhanced techniques for maintaining the required information up-to-date, which results in efficiency.**

- **The primary aim of this “Bank Account Management System” is to provide an improved design methodology, which envisages future expansion, and modification, which is necessary for a core sector like banking.**

Output/ Post Condition:

- **Records Persisted in Success &amp; Failure Collections**
- **Standalone application to manage customer's bank accounts**

Non-Functional Requirements:

- **Secure App Platform –User AccountNumber/Password-Based Credentials.**
- **Sensitive data has to be categorized and stored in a secure manner.**
- **Secure connection for transmission of any data.**

Logging & Auditing:

- **The system should support logging(app/web/DB) auditing at all levels Monitoring.**
- **Should be able to monitor via as-is enterprise monitoring tools Cloud.**
- **The Solution should be made Cloud-ready and should have a minimum impact**
- **When moving away to Cloud infrastructure.**

Browser Compatible:

- **All latest browsers with secure protection.**

Technology Stack:

- **Front End Angular 10+/ React 16+**
- **Material Design Bootstrap / Bulma**
- **Server Side Spring Boot / .Net WebAPI/ Node Js**
- **Database MySQL or Oracle or MSSQL**

Key points to remember:

- **The id (for frontend) and attributes(backend) mentioned in the SRS should not be modified at any cost. Failing to do may fail the test**
- **Remember to check the screenshots provided with the SRS. Strictly adhere to id mapping and attribute mapping. Failing to do may fail test cases.**
- **Strictly adhere to the proper project scaffolding (Folder structure), coding conventions, method definitions, and return types. Adhere strictly to the endpoints given below.**

Validations:

- **Basic email validation should be performed.**
- **Basic mobile validation should be performed.**
- **Password validations should be performed.**

Application assumptions:

- **The login page should be the first page rendered when the application loads.**

- **Manual routing should be restricted by using AuthGuard by implementing the can activate the interface.**

- **Unless logged into the system, the user cannot navigate to any other pages.**

- **Logging out must again redirect to the login page.**

- **To navigate to the admin side, you can store a user type as admin in the database with a username and password as admin.**

- **Use admin/admin as the username and password to navigate to the admin account dashboard.**

Project Tasks:

API Endpoints:

USER

Action URL Method Response
Login /login POST true/false
Signup /signup POST true/false
Get All Banking Details– Home /home GET Array of Details
New payment/home/{id} POST
Add payee /statement/{id} GET Array of statements
Cancel payment/transfer/delete POST payment cancel
Payment confirm/save transaction POST transfer bill added to the statement
list/transactions POST Array of transfer
Transfer to account directly /account transfer POST Place transaction to statements directly

ADMIN

Action URL Method Response
All accountpayee /admin GET Array of payee
Add Payee /admin/addPayee POST-Payee added
Delete Payee/admin/delete/{id} GET Payee deleted

Account Edit /admin/productEdit/{id} GET Get All details of customer id
Account details Edit /admin/p ayeeEdit/{id} POST Save the Changes
Get All transfers/admin/orders GET Array of transactions

Frontend:
Customer:

- **Auth: Design an auth component (Name the component as auth for an angular app whereas Auth for reacting app. Once the component is created in react app, name the jsx file as same as component name i.e Auth. jsx file) where the customer can authenticate login and signup credentials**
- **Signup: Design a signup page component (Name the component as signup for an angular app whereas Signup for respond app. Once the component is created in react app, name the jsx file as same as component name i.e Signup. jsx file)where the new consumer has the option to sign up by furnishing their basic details.**

a. **Ids:**
   - **username**
   - **mobile number**
   - **password**
   - **confirm password**
   - **submit button**
   - **signLINK**
   - **signup box**

b. **API endpoint Url**

c. **Output screenshot**

Dashboard / Home: **Design a home page component named (Name the component as homepage for angular app whereas HomePage for reacting app. Once the**
component is created in react app, name the jsx file as same as component name i.e HomePage.jsx file) that has the navigation bar and lists all the available products as grid elements with appropriate filter options.

a. **Ids:**
   1. **userNavbar**
   2. **productHomeButton**

3. **Make payment button**
4. **Add payee button**
5. **Balance Button**
6. **logoutButton**
7. **transactionsHomeBody**

b. API endpoint Url: http://localhost:8000/home

c. Screenshot

Transaction and statements: **Design a transfer component**

Named statement where we can see the transactions

a. Ids

  i. Transaction Body

  ii. Statement Body

b. API endpoint Url: http://localhost:8000/cart

c. API endpoint Url: http://localhost:8000/orders

d. Screenshot

Admin:

Admin Dashboard:

1. adminNavbar

2. adminProductButton

3. adminOrderButton

4. logoutButton

b. Add Payee: Design an add payee component (Name the component as addpayee for angular app whereas AddPayee for react app. Once the component is created in react app, name the jsx file as same as component name i.e AddProduct.jsx file) in which the admin can add new products to the inventory.

i.Ids:

1.addpayeeBody

2.payeeName

4.accountDescription

ii.API endpoint Url: http://localhost:8000/addProduct

c.Screenshot

c.Viewstatement: Create a view component (Name the component as viewtransaction statements for
angular app whereas Viewhistoryfor react app. Once the component is created in react app, name the jsx file as same as component name i.e Viewhistorys.jsx
file) where the admin can look into the new and old orders.
i.Ids:
1.adminOrderBody
ii.API endpoint Url: http://localhost:8000/admin/orders
iii.Screenshot

Backend:
Class and Method description:

Model Layer:

1. UserModel: This class stores the user type (admin or the customer) and all user information.
a. Attributes:

- **i. email: String**
- **ii. password: String**
- **iii. username: String**
- **iv. mobileNumber: String**
- **v. active: Boolean**
- **vi. role: String**

2. LoginModel: This class contains the email and password of the user.
a. Attributes:

- **i. email: String**
- **ii. password: String**

4. transferModel: This class stores the cart items.
a. Attributes:

- **i. customerID: String**
- **ii. userId: UserModel**

- **iii. payeeName: String**
- **iv. amount: int**

v. remark: String

5. statementModel: This class stores the transfer details.
a. Attributes:

i. transferId: String
ii. userId: String
iii. payeeName: String
Iv. amount: int
v. remark: String
Vi: account balance:int

Controller Layer:
6. SignupController: This class control the user signup
a. Methods:
i. saveUser(UserModel user): This method helps to store users in the database and return true or false based on the database transaction.
7. LoginController: This class controls the user login.
a. Methods:
i. checkUser(LoginModel data): This method helps the user to sign up for the application and must return true or false.
8. PayeeController: This class controls the add/edit/update/view products.
a. Methods:
i. List&lt;Payee accounts&gt; getPayee(): This method helps the admin to fetch
all transactions from the database.
iv. accountEditSave: This method helps to edit a
Account and save it to the database.
v. add account : This method helps to add a new
product to the database.

9. Statement Controller: This class helps with the orders such as save transactions
a. Methods:
i. List&lt;transferTemp&gt; getUseraccount(String id): This method helps to list
the orders based on the user id.
ii. savestatement(String id): This method helps to save the transfer history as
An statement.

iii. Amount transfer: This method helps to place statement by the customer.