Maverick Bank

Problem statement:

The Maverick Bank is a Banking and Financial System is a comprehensive application designed to manage the financial transactions and accounts of customers in a banking. This system caters to both individual and business customers, offering services such as account management, funds transfer, loans, investments, and financial reporting.

Scope:

1. **Customer Account Management:** Provide customers with the ability to open and manage various types of accounts (savings, checking, business, etc.).
2. **Transaction Processing:** Enable customers to perform a wide range of financial transactions, including deposits, withdrawals, transfers.
3. **Loan Management:** Offer loan products and manage loan applications, approvals, disbursements, and repayments.
4. **Financial Reporting:** Generate account statements, transaction history, and financial reports for customers.

Technologies:

- Frontend: React.js / Angular Js.
- Backend: Java, Spring Boot/C#, .Net / Python Djnago for API development.
- Database: MySql / Sql Server.
- Authentication: JSON Web Tokens (JWT) for secure user authentication.

Reference Link: https://www.sbi.com/

Use case Diagram:

Actor: Customer

- Use Case: Register Account
- Use Case: Log In
- Use Case: Manage Accounts
    - Open Account
    - View Account Details
- Use Case: Perform Transactions
    - Deposit Funds
    - Withdraw Funds
    - Transfer Funds
- Use Case: Apply for a Loan
- Use Case: Generate Reports
    - Account Statement
    - Transaction History
- Use Case: Log Out

Actor: Bank Employee:

- Use Case: Log In
- Use Case: Manage Customer Accounts
    - Approve new account creation.
    - View Account details.
    - View Transaction details.
    - Close old account by customer request.
- Use Case: Manage Loans
    - Review Loan Applications
    - Approve/Reject Loans
    - Disburse Loans
- Use Case: Generate Reports
    - Financial Performance Reports
    - Regulatory Reports
- Use Case: Log Out

Actor: Administrators:

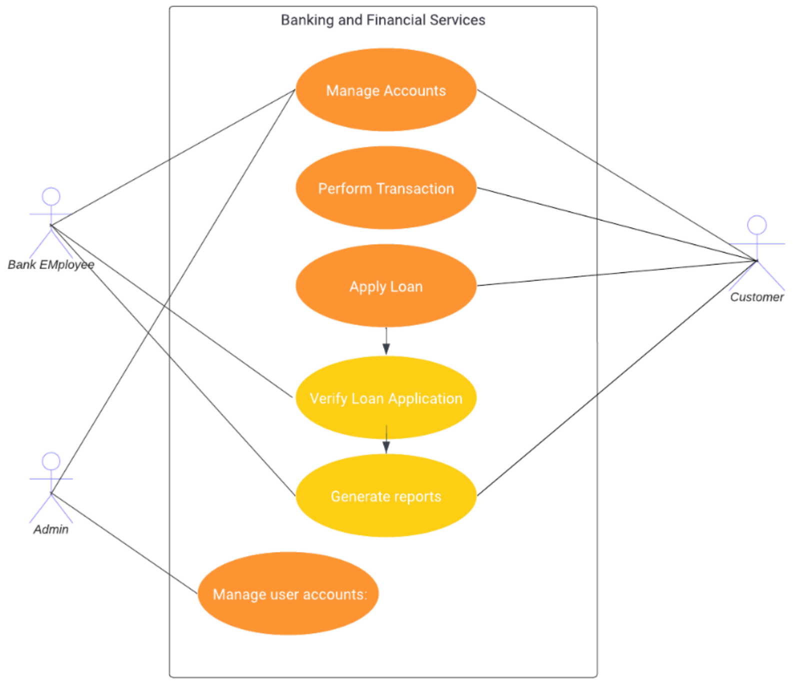
- Use Case: Manage User Accounts and bank employees

System: Security and Authentication

- Use Case: Authenticate User

Associations:

- Customer performs Account Management (Open Account, Close Account, Manage Account).
- Customer initiates Transaction Processing (includes: Deposit, Withdrawal, Transfer).
- Customer applies for a Loan. Bank Employee reviews and approves account creation and closing application and Loan Applications



Development Process:

1. **User Registration and Login:**
    - Customers can create accounts, providing personal details (name, gender, contact number, address, etc.)
    - The system validates the information and creates user profiles.
    - Users log in using their credentials (username/email and password).
2. **Customer Dashboard and Account Management:**
    - Customers can log in to profile and access the account management section. It should show all accounts holded by the user.
    - By selecting the account, customers can view account information (name, account number, IFC code, Branch name, address), account balances, transaction history (must have option to view last 10 transactions, last month transactions and view transaction between two date).
    - Customer can open new accounts by selecting the account type and providing necessary information (Name, Address, Date of birth, aadhar number, age should be calculated from DOB, PAN number).
    - Customer can add beneficiary details like account name, number, bank name, branch and IFSC code.
    - To add beneficiary details, customers should enter the bank name and it should display all branches on selecting branch it should display IFSC code.
    - Customer can close account by account close request and it should be closed by bank employee.
3. **Customer Dashboard and Transactions:**
    - Customers can perform financial transactions, including deposits, withdrawals, transfers.

- Customers enter transaction details, such as the amount and destination account. destination account details should be stored for future purposes to avoid typing incorrect account details as beneficiary details.
- The system validates the transaction and updates account balances. Transactions are securely processed and recorded.

4. **Customer Dashboard and Loan Application:**
    - Customers can apply for loans through the system. customers can view the list loan with details (loan amount, rate of interest and tenure or duration for which the loan is availed).
    - Customers can choose the loan and submit loan applications with required details (e.g., amount, purpose).
    - customers should view availed loans if not only list of loans.
5. **Bank employee Dashboard and Account and Transactions:**
    - Bank employees should validate the customer details and approve the account creation and account deletion.
    - Bank employees can view all transactions done by the account holders.
    - Bank employees can view individual account transactions and total inbound transactions and outbound transactions.
6. **Bank employee Dashboard and Transactions:**
    - Bank employees review loan applications, check creditworthiness (based on inbound and outbound transactions) and make loan decisions.
    - Approved loans are disbursed to the customer's account.
7. **Administrator Dashboard and Transactions:**
    - Administrators can create, modify, or deactivate user accounts, including customers and bank employees.

Security and Compliance:

- User authentication and authorization are enforced to ensure data privacy.

    1. JWT Authentication:

    JWT authentication involves generating a token upon successful user login and sending it to the client. The client includes this token in subsequent requests to authenticate the user.

    - User Login: Upon successful login (using valid credentials), generate a JWT token on the server.
    - Token Payload: The token typically contains user-related information (e.g., user ID, roles, expiration time).
    - Token Signing: Sign the token using a secret key known only to the server. This ensures that the token hasn't been tampered with.
    - Token Transmission: Send the signed token back to the client as a response to the login request.

- Client Storage: Store the token securely on the client side (e.g., in browser storage or cookies).

2. JWT Authorization:

JWT authorization involves checking the token on protected routes to ensure that the user has the required permissions.

- Protected Routes: Define routes that require authentication and authorization.
- Token Verification:
  1. Extract the token from the request header.
  2. Verify the token's signature using the server's secret key.
- Payload Verification:
  1. Decode the token and extract user information.
  2. Check user roles or permissions to determine access rights.
- Access Control: Grant or deny access based on the user's roles and permissions.

Logout:

- Logging out involves invalidating the JWT token on both the client and the server to prevent further unauthorized requests.

Project Development Guidelines

The project to be developed based on the below design considerations.

1	**Backend Development**	• Use Rest APIs (Springboot/ASP.Net Core WebAPI to develop the services • Use Java/C# latest features. • Use ORM with database. • perform backend data validation. • 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. • Implement JWT for Security. • Implement Logging.

		• Follow Coding Standards with proper project structure.

Frontend Constraints

1.	**Layout and Structure**	Create a clean and organized layout for your registration and login pages. You can use a responsive grid system (e.g., Bootstrap or Flexbox) to ensure your design looks good on various screen sizes.
2	**Visual Elements**	**Logo:** Place your application's logo at the top of the page to establish brand identity.
		Form Fields: Include input fields for email/username and password for both registration and login. For registration, include additional fields like name and possibly a password confirmation field.
		Buttons: Design attractive and easily distinguishable buttons for "Register," "Login," and "Forgot Password" (if applicable).
		Error Messages: Provide clear error messages for incorrect login attempts or registration errors.
		Background Image: Consider using a relevant background image to add visual appeal.
		Hover Effects: Change the appearance of buttons and links when users hover over them.
		Focus Styles: Apply focus styles to form fields when they are selected
3.	**Color Scheme and Typography**	Choose a color scheme that reflects your brand and creates a visually pleasing experience. Ensure good contrast between text and background colors for readability. Select a legible and consistent typography for headings and body text.
4.	**Registration Page/Add Bank Employee**	**Form Fields:** Include fields for users to enter their name, email, password, and any other relevant information. Use placeholders and labels to guide users.
		Validation: Implement real-time validation for fields (e.g., check email format) and provide immediate feedback for any errors. **Form Validation:** Implement client-side form validation to ensure required fields are filled out correctly before submission.
		Password Strength: Provide real-time feedback on password strength using indicators or text. **Password Requirements**: Clearly indicate password requirements (e.g., minimum length, special characters) to help users create strong passwords.
		Registration Success: Upon successful registration, redirect users to the login page.
5.	**Login Page: Customer/Bank Employee**	**Form Fields:** Provide fields for users to enter their email and password.
		Password Recovery: Include a "Forgot Password?" link that allows users to reset their password.
6.	**Common to React/Angular**	• Use Angular/React to develop the UI. • Implement Forms, data binding, validations, error message in required pages. • 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. • Design your pages to be responsive so they adapt well to different screen sizes, including mobile devices and tablets.

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 design patterns.
- 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.