



In association with



PROJECT TITLE- Capstone Project

Banking Application

SUBMITTED BY:

Date: .

Introduction:

The **Banking Application** is a sophisticated and user-centric platform designed to streamline and manage core banking services. From account creation to transaction processing and balance management, this application offers a comprehensive solution tailored to meet the needs of both users and administrators.

Developed as a capstone project, the application leverages modern technologies to provide seamless and intuitive user experience while maintaining a highly scalable and secure backend architecture. Its modular design ensures ease of maintenance, flexibility for future enhancements, and robust integration capabilities.

By focusing on key banking functionalities, this application demonstrates the perfect synergy of cutting-edge technology and practical financial solutions, making it a reliable tool for managing personal and administrative banking operations effectively.

Objectives:

- To design and develop a scalable banking application using **Spring Boot** (backend), **ReactJS** (frontend).
- To provide a modular structure with **at least three core modules** including user authentication.
- To demonstrate backend and frontend integration using RESTful APIs.
- To ensure data integrity with **MySQL database** and to document the application's architecture, features, and endpoints.

Technologies Used:

- **Frontend:** ReactJS, Bootstrap, Axios
- **Backend:** Spring Boot, REST APIs
- **Database:** MySQL
- **Version Control:** GitHub

Project Structure

Front-End:

- **Framework:** React.js
- **Structure Overview:**
 - src/pages/ - Contains all the React component files for different pages like Dashboard, AccountDetails, Transactions, etc.
 - src/index.js - Entry point for the React application.
 - public/ - Contains static files like index.html, images, and other assets.
 - Styles are applied via App.css and individual component-level CSS files.

Back-End:

- **Framework:** Spring Boot
- **Structure Overview:**
 - controller/ - Contains TransactionController, UserController for managing API endpoints.
 - model/ - Includes entity classes for Account, Transaction, User.
 - repository/ - Contains repository interfaces for database interactions.
 - service/ - Includes business logic classes like AccountService, TransactionService.
 - resources/ - Contains application.properties for configuration.

GITHUB LINK:

Features

- **User Module:**
 - User authentication and account management.
- **Account Module:**
 - Creating, freezing, and deleting accounts.
 - Managing account balances and statuses.
- **Transaction Module:**
 - Deposit, withdrawal, and viewing transaction history.

API Endpoints

Account Endpoints:

- GET /api/accounts/user/{userId} - Fetch accounts for a user.
- POST /api/accounts - Create a new account.
- PUT /api/accounts/{accountId}/status - Update account status.
- DELETE /api/accounts/{accountId} - Delete an account.

Transaction Endpoints:

- POST /api/add-transaction - Add a transaction.
- GET /api/transactions?userId={userId} - Get all transactions for a user.

FRONT END FOLDERS AND FILES

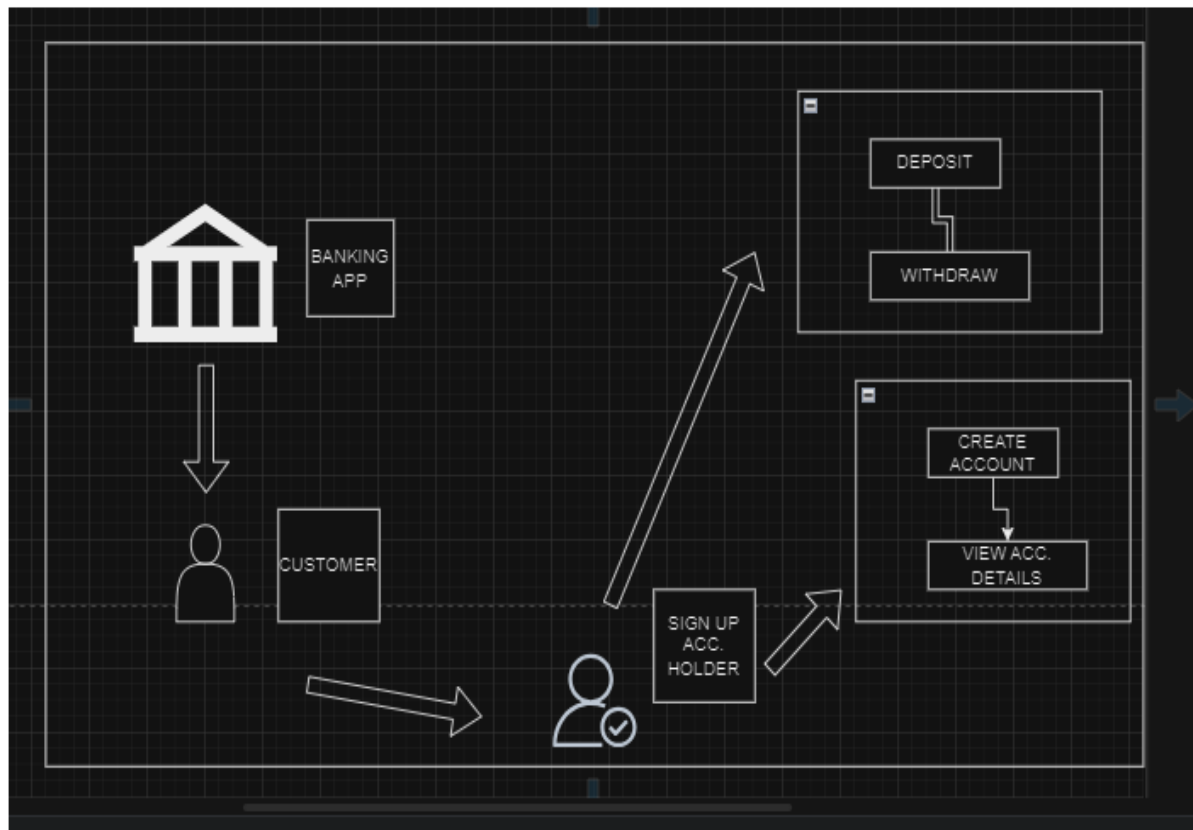
□ components/:

- Contains reusable UI components used across multiple pages of the application.
 - **Navbar.js**: Implements the navigation bar that is used throughout the app for routing between pages.
 - **Navbar.css**: Associated CSS file for styling the navigation bar.

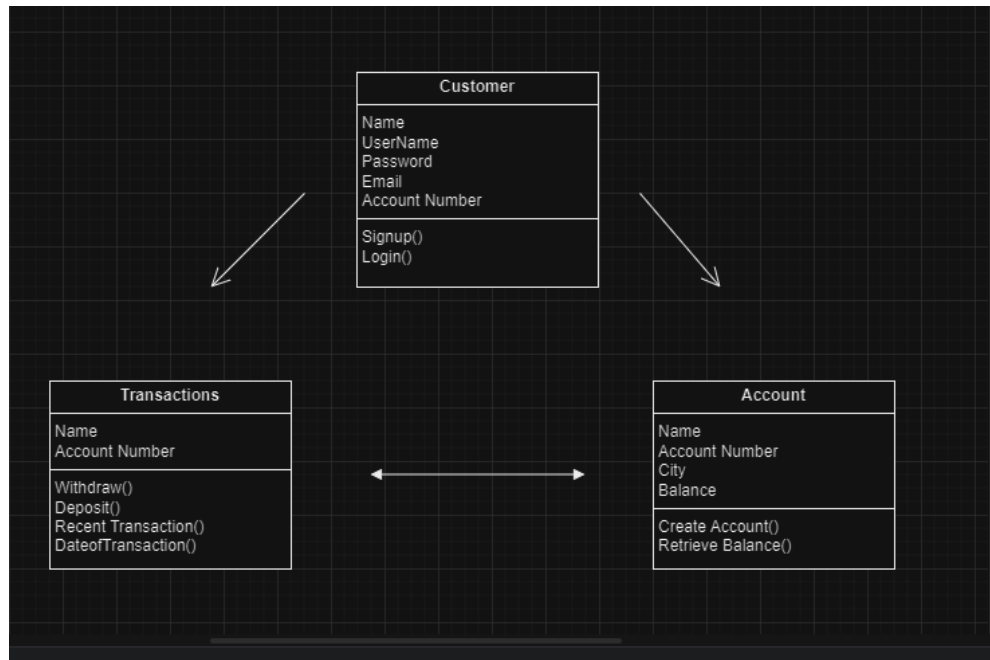
□ pages/:

- This folder holds individual pages of the application, with each file representing a specific route or functionality:
 - **About.js**: A page providing details about the banking app.
 - **AccountDetails.js**: Displays user account details.
 - **Dashboard.js**: The main dashboard for users after login, showing key banking features.
 - **Help.js**: A help page to guide users or provide support.
 - **Home.js**: The homepage of the application.
 - **Login.js**: The login page where users enter their credentials.
 - **SignUp.js**: A signup page to allow new users to register.
 - **Transactions.js**: A page showing user transactions and related financial data.
 - **CSS Files (About.css, Form.css)**: Styling files associated with respective pages for better UI.
 - **Images (customer support.jpg, innovative.jpg, etc.)**: Static assets used in the app's design and layout.

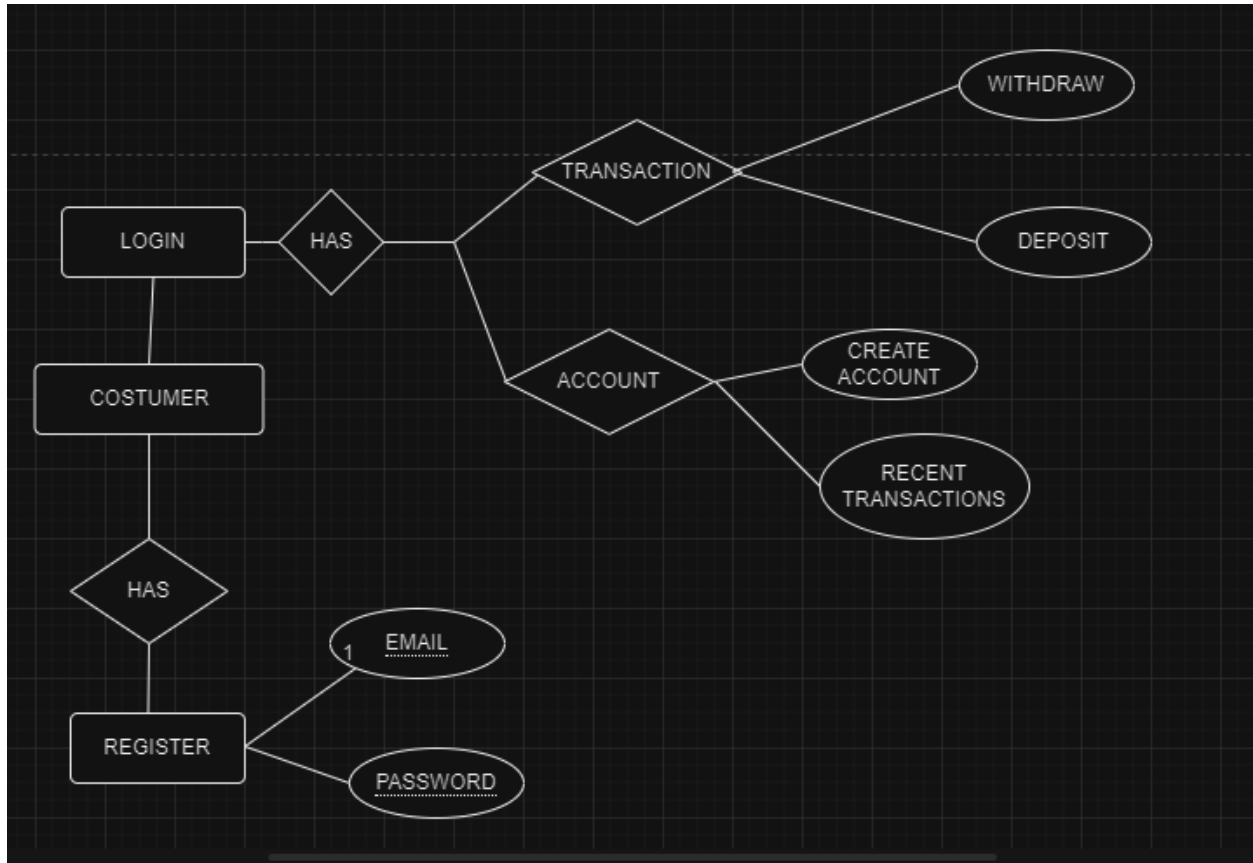
FLOW DIAGRAM OF THE APPLICATION



UML Diagram of the application:

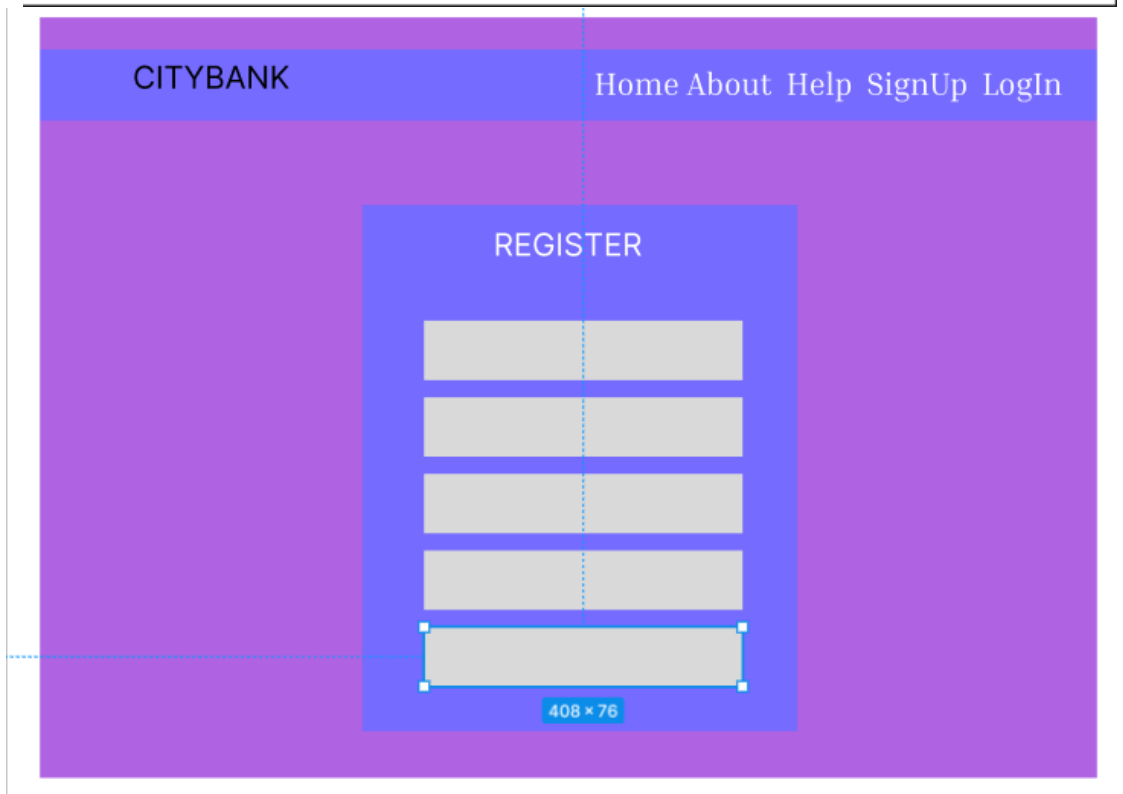
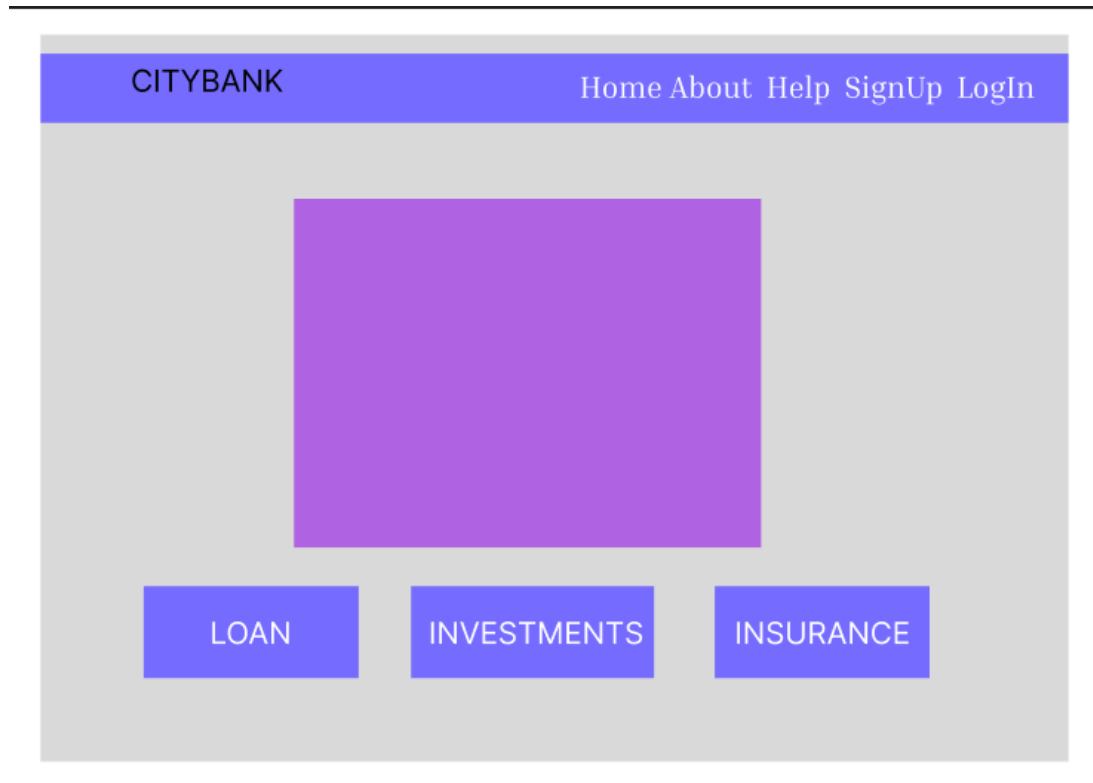


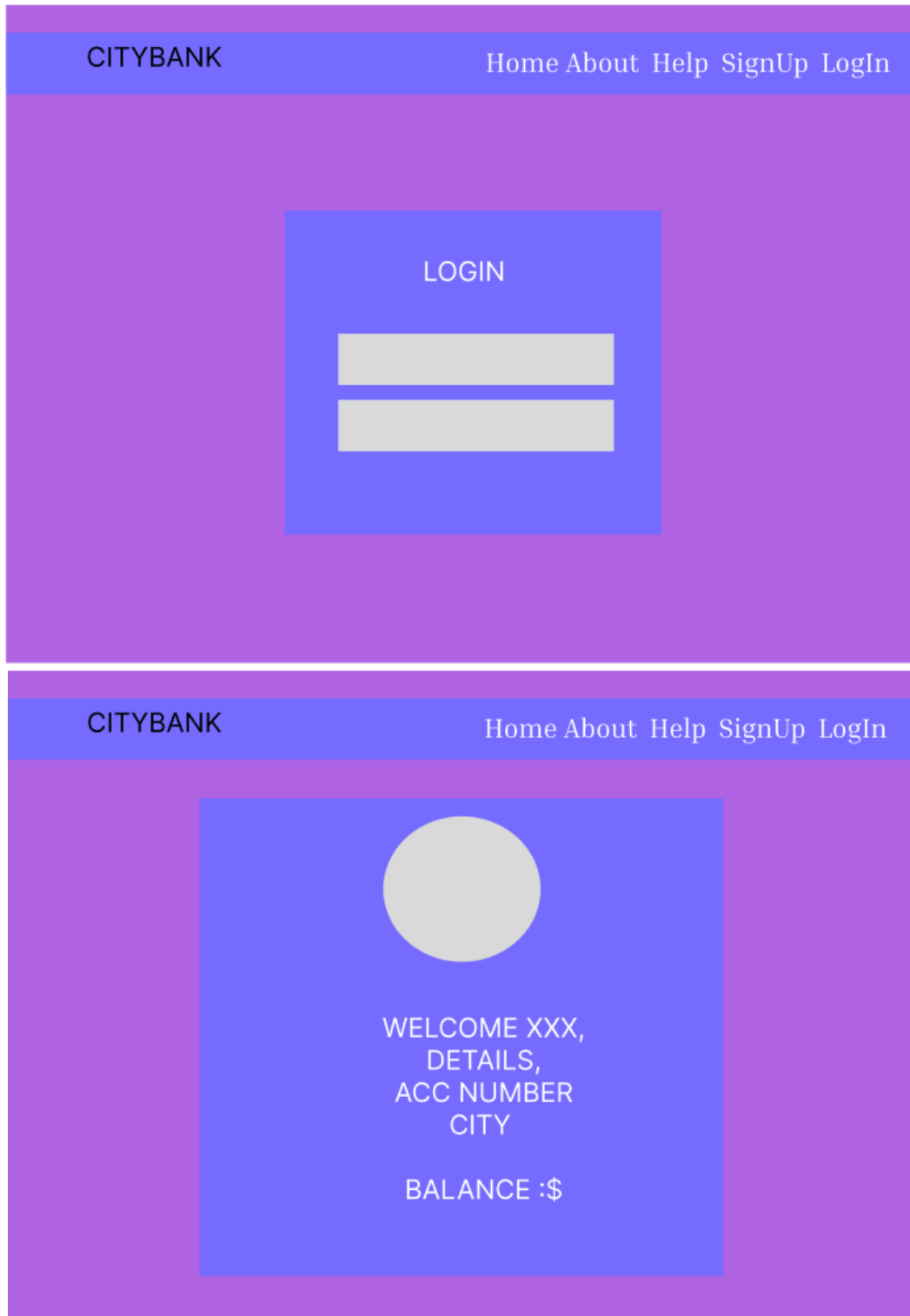
ER Diagram of the Banking Application:

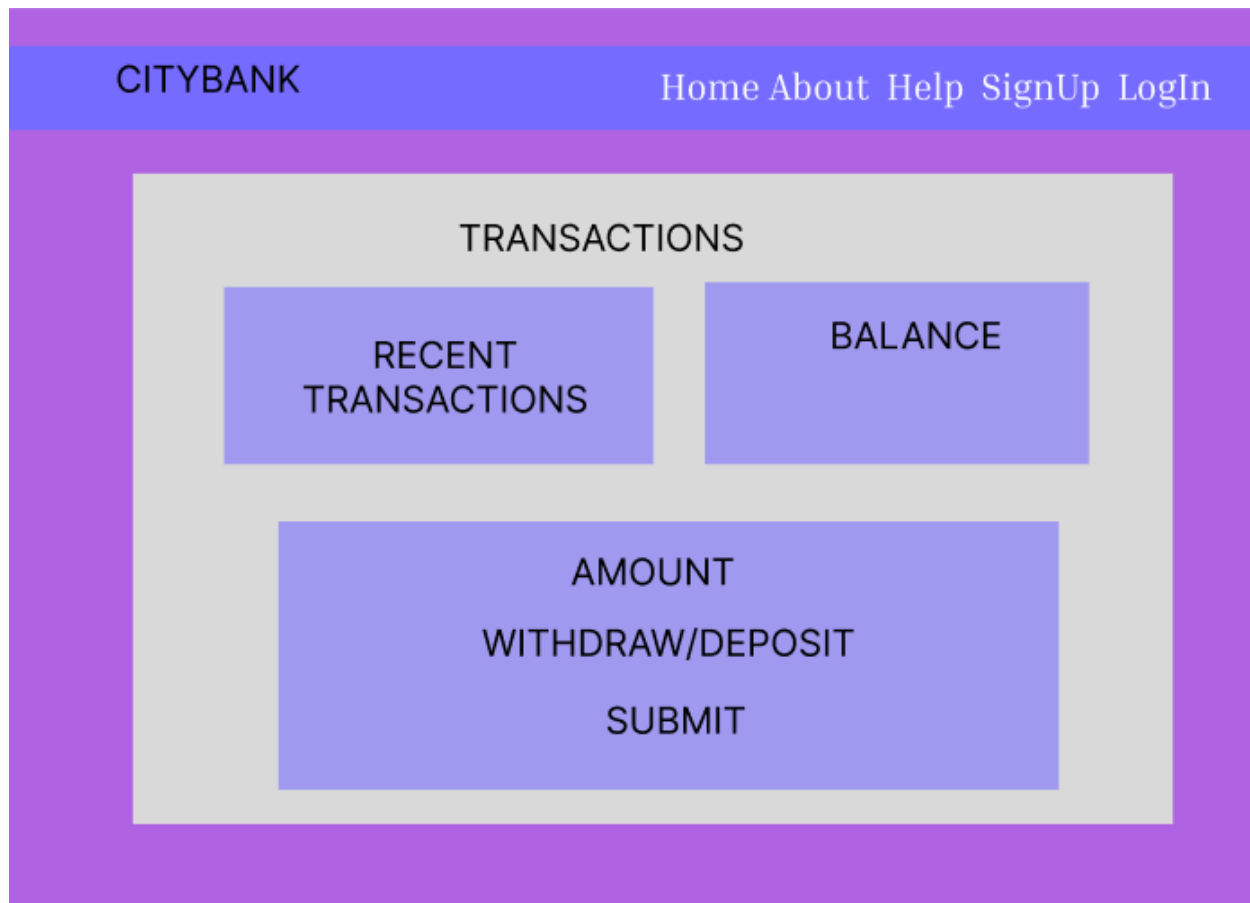


This is an ERD of the banking application. It illustrates the relationship between a **customer**, who can **Register** with their details and **Login** (email and password) to their account. The customer can manage an **Account**, which allows them to **Create Accounts**, view **Recent Transactions**, and perform **Transactions** such as **Withdraw** and **Deposit**. The relationships highlight the flow and interaction between different entities within the system.


WIRE FRAME







CITY BANK
HOME
ABOUT US
HELP
SIGN UP
LOGIN






WELCOME TO CITY BANK

INSURANCE
LOANS
NEW INVESTMENTS

About CITYBANK

Welcome to **City Bank**, where we are redefining modern banking. Our mission is to deliver innovative solutions, unparalleled customer support, and a secure financial platform to help you achieve your financial goals.

With our cutting-edge technology, personalized services, and a customer-first approach, we ensure your banking experience is seamless, efficient, and secure.

CITY BANK
HOME
ABOUT US
HELP
SIGN UP
LOGIN

Need Help?

Our customer support team is here to assist you with any issues or inquiries. Feel free to reach out to us at any time! We're committed to ensuring your experience with City Bank is seamless and enjoyable.

Contact Support

Frequently Asked Questions

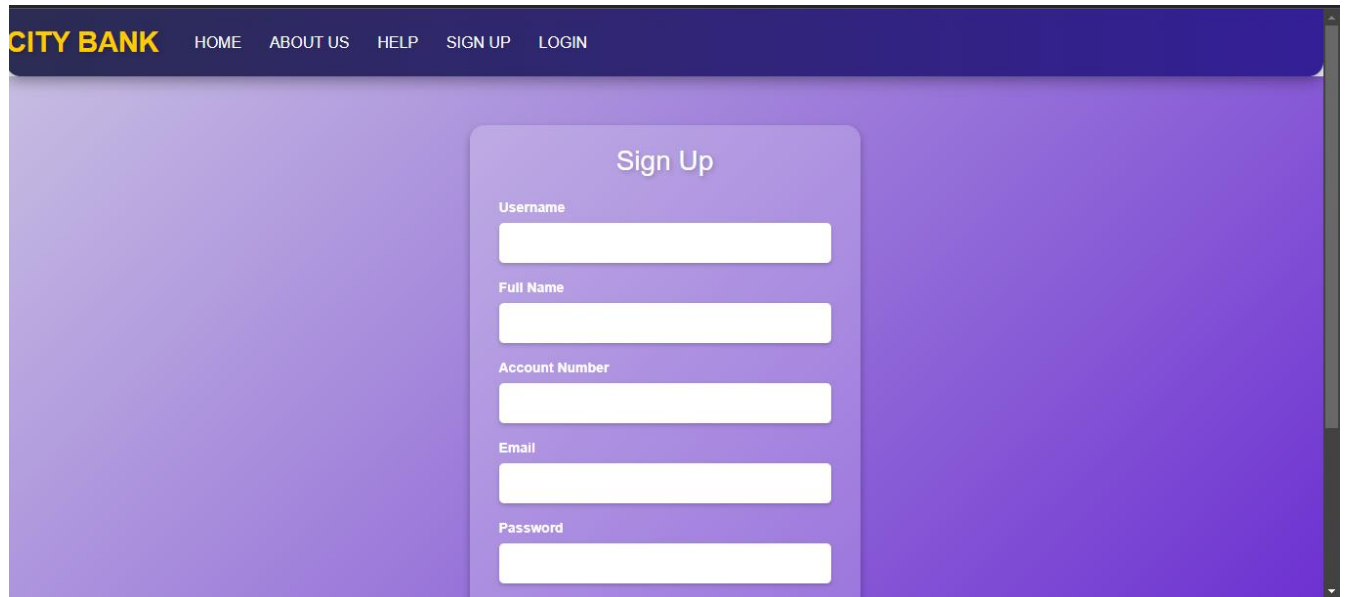
Q: How do I reset my password?

A: To reset your password, click on the "Forgot Password" link on the login page and follow the instructions.

Q: How can I contact customer support?

A: You can reach us via email at support@citybank.com or by calling our 24/7 hotline at +1-800-123-4567.

MODULE -1 USER AUTHENTICATION



The image shows a web browser displaying the City Bank Sign Up page. The page has a dark blue header with the City Bank logo and navigation links: HOME, ABOUT US, HELP, SIGN UP, and LOGIN. The main content area has a purple gradient background. In the center, there is a white rounded rectangle containing the 'Sign Up' form. The form fields are: Username, Full Name, Account Number, Email, and Password. Each field has a white input box with a label above it.

CITY BANK HOME ABOUT US HELP SIGN UP LOGIN

Sign Up

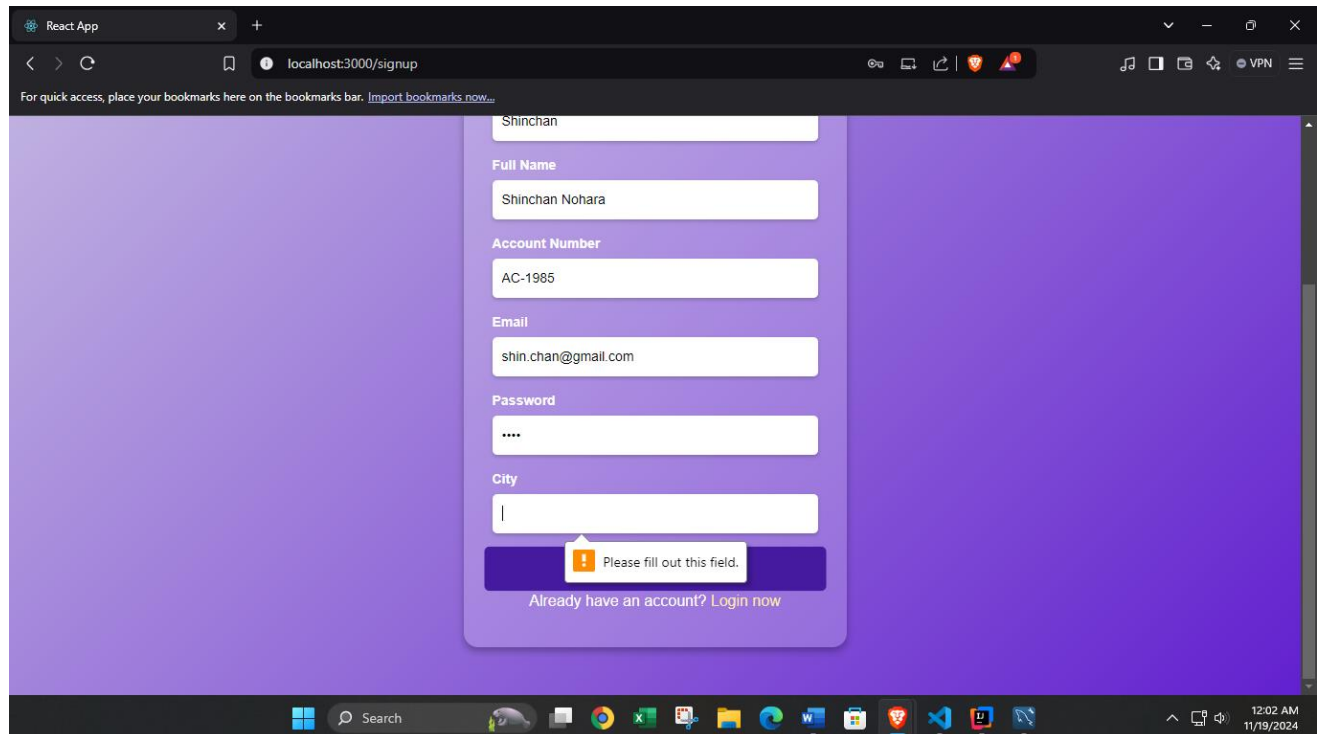
Username

Full Name

Account Number

Email

Password



The image shows the same City Bank Sign Up page, but now the form fields are filled with data. The data entered is: Username: Shinchan, Full Name: Shinchan Nohara, Account Number: AC-1985, Email: shin.chan@gmail.com, Password: ****, and City: (empty). A red error message 'Please fill out this field.' is displayed below the City input field. At the bottom of the form, there is a link: 'Already have an account? Login now'.

React App x +

localhost:3000/signup

For quick access, place your bookmarks here on the bookmarks bar. [Import bookmarks now...](#)

Shinchan

Full Name
Shinchan Nohara

Account Number
AC-1985

Email
shin.chan@gmail.com

Password

City

Please fill out this field.

Already have an account? [Login now](#)

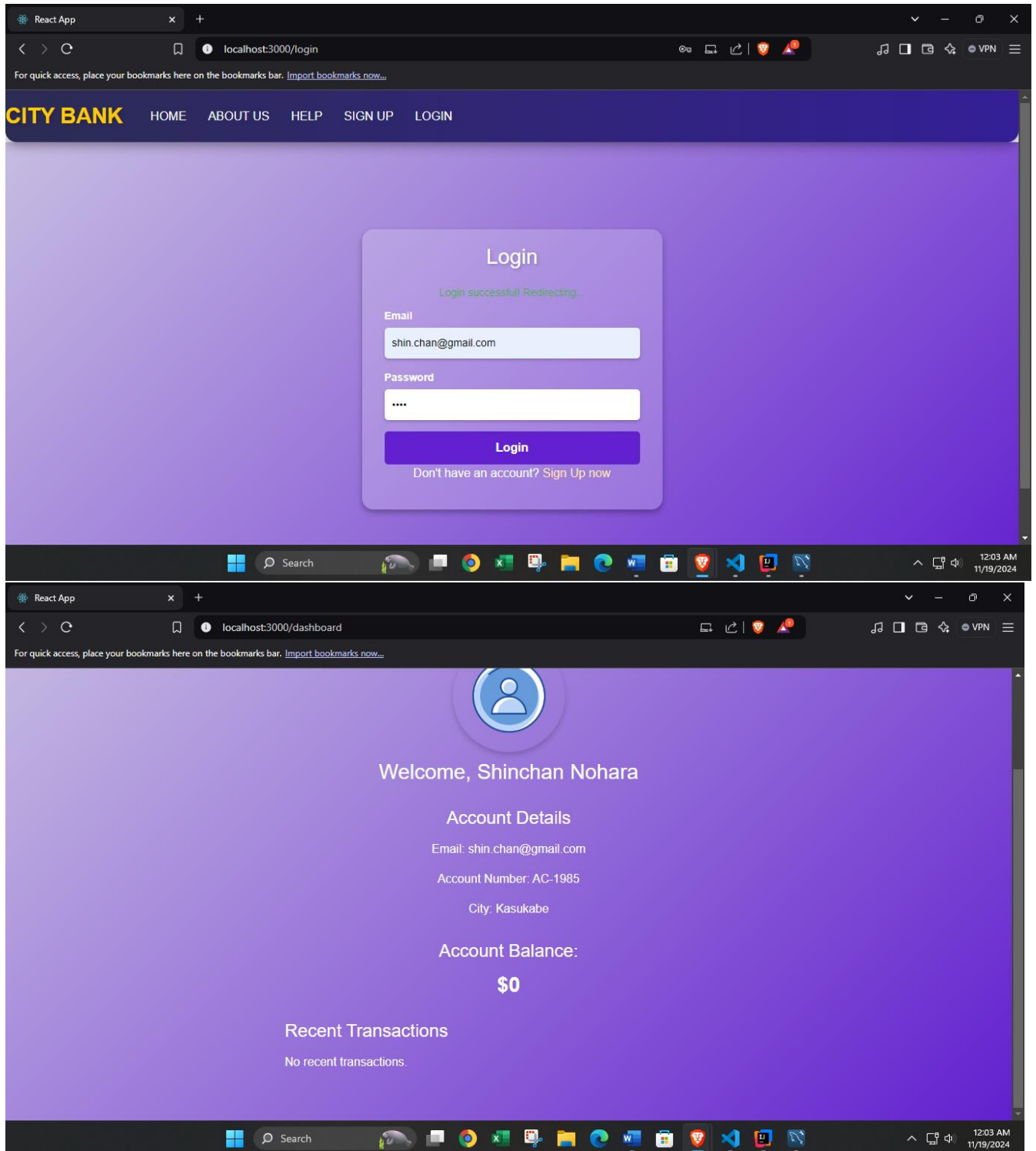
Search

12:02 AM 11/19/2024

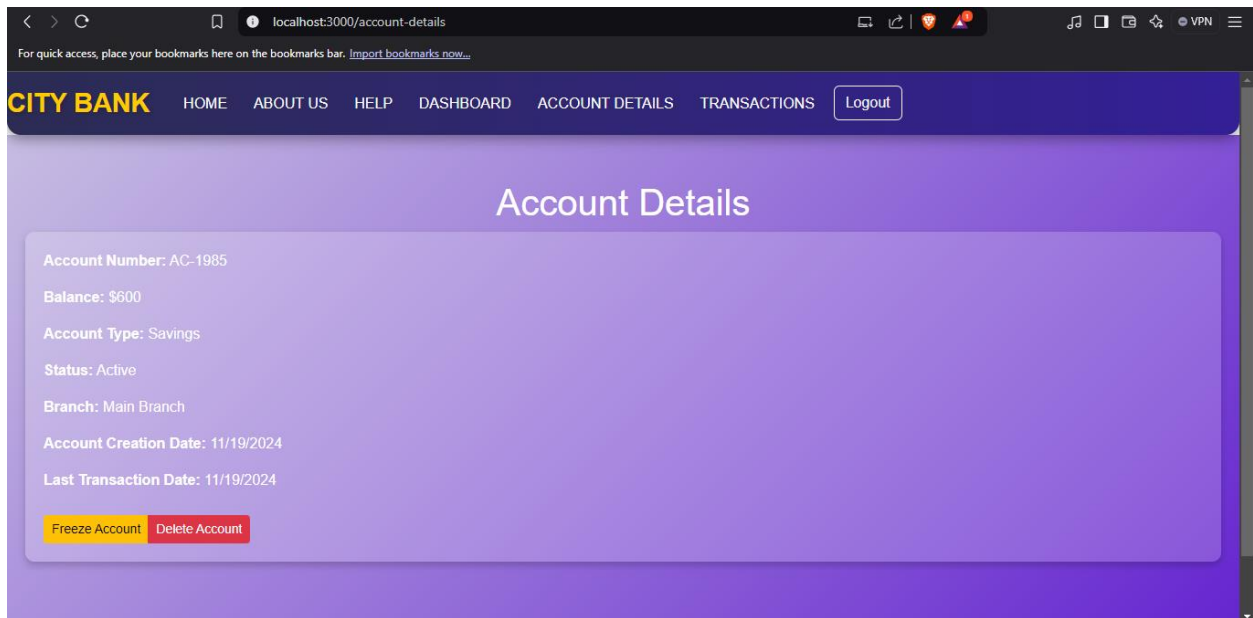
The **User Module** is the foundation of the application, enabling secure access and effective account management for users.

Key Features:

- **User Authentication:**
 - Secure login and registration processes ensure that only authorized users can access their accounts.
 - Credentials are securely stored and managed using modern encryption techniques.
 - Includes session management with auto-logout for inactive sessions to enhance security.
- **Profile Management:**
 - Users can view and update personal information such as name, email, and contact details.
 - Allows users to change their passwords with validation for enhanced security.
- **Accessibility and Personalization:**
 - Ensures a smooth onboarding process for new users with an intuitive registration flow.
 - Includes account recovery features like password reset for a user-friendly experience.
- Safeguards user data with robust security mechanisms.
- Provides a seamless and personalized experience for users to manage their profiles.
- Facilitates a secure gateway for accessing other banking features.



MODULE 2- ACCOUNT MODULE



The **Account Module** focuses on managing user accounts, ensuring that users and administrators have full control over account-related activities.

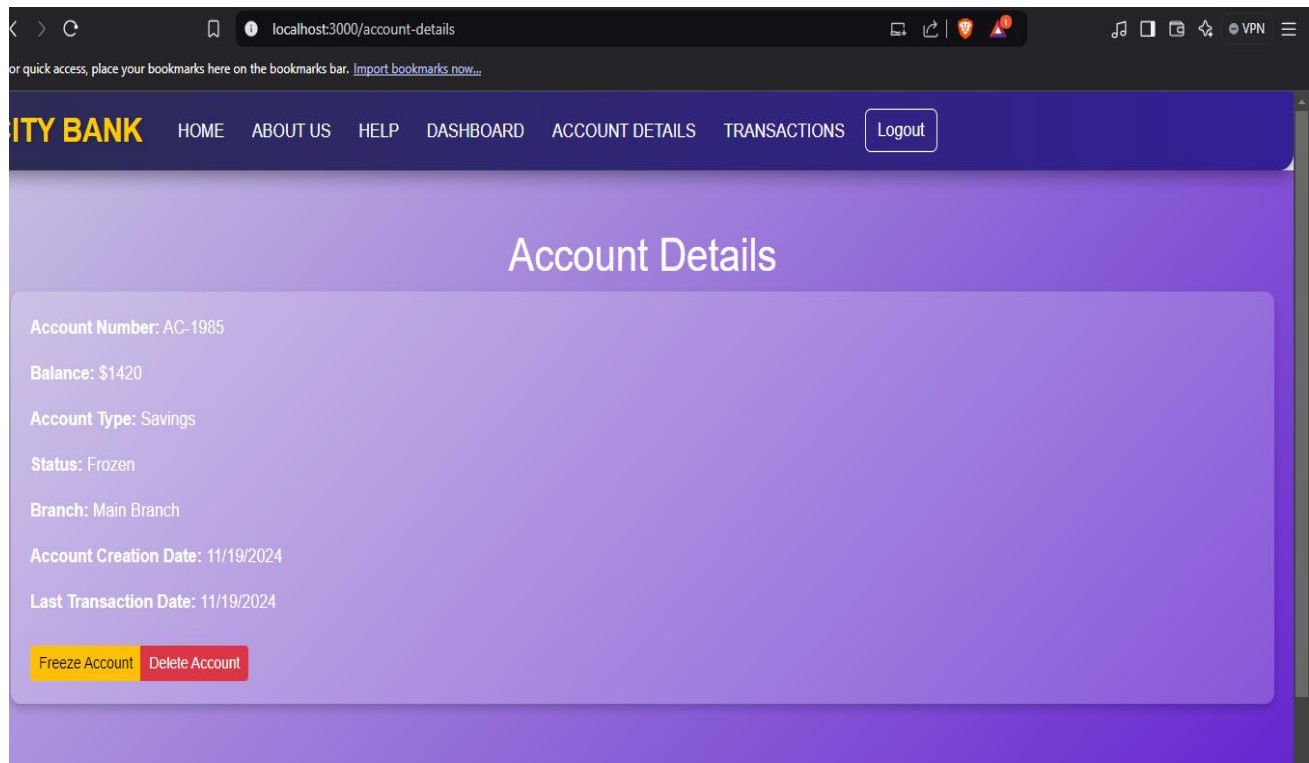
Key Features:

- **Account Creation:**
 - Simplifies the process of opening new accounts for users.
 - Allows administrators to approve or reject account requests based on predefined criteria.
- **Account Freezing and Deletion:**
 - Provides administrators the ability to freeze accounts in case of suspicious activity.
 - Offers a secure mechanism for account closure, ensuring proper verification before deletion.
- **Balance and Status Management:**
 - Real-time balance updates based on transactions (deposits and withdrawals).

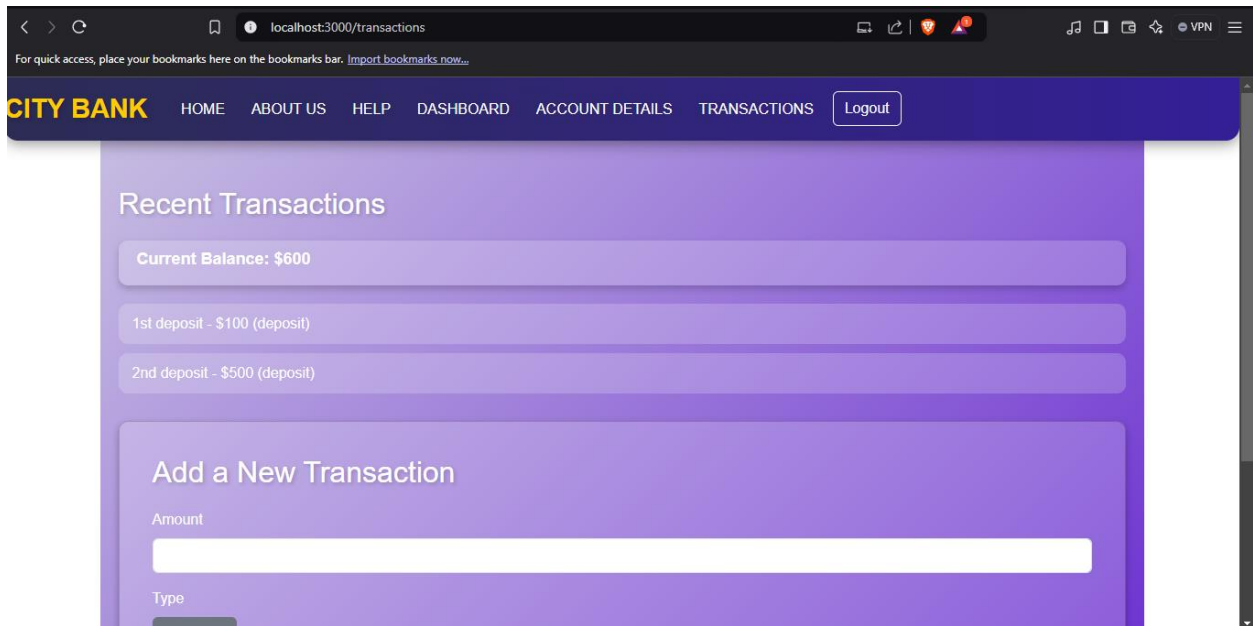
- Displays account status (active, frozen, or closed) for easy monitoring.
- Administrators can reset or update account statuses as required.

Benefits:

- Enhances operational efficiency by enabling dynamic account management.
- Empowers users with transparency over their account status and balances.
- Protects users and the institution by offering features like account freezing for security threats.



MODULE -3 TRANSACTION MODULE

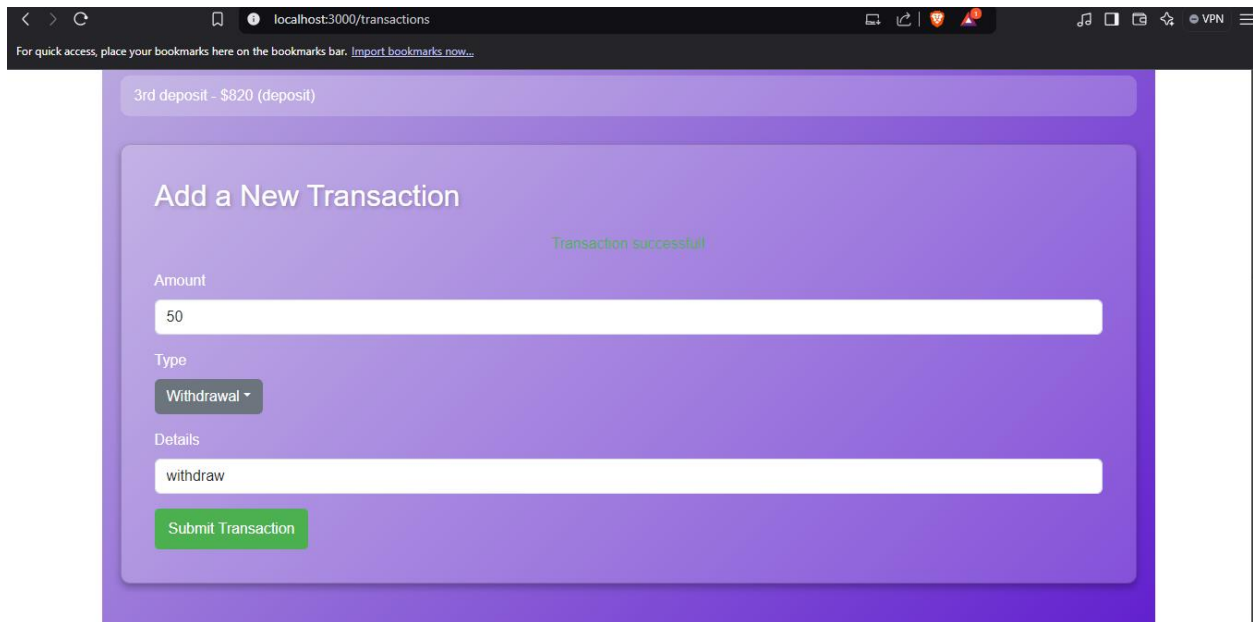


The **Transaction Module** serves as the financial backbone of the application, facilitating efficient and secure monetary operations.

Key Features:

- **Transaction Processing:**
 - Enables users to perform deposits and withdrawals seamlessly.
 - Real-time validation ensures sufficient balance for withdrawals.
- **Transaction History:**
 - Displays a detailed log of all user transactions, including date, amount, type (deposit/withdrawal), and a brief description.
 - Includes search and filter options to help users locate specific transactions quickly.
- **Balance Overview:**
 - Updates user account balances instantly after every transaction.
 - Provides a clear and concise view of account balance, enabling users to plan their finances better.
- Ensures financial transparency by providing detailed transaction logs.

- Facilitates secure and error-free financial operations.
- Keeps users informed of their financial status with real-time updates.



1. **Security:** Modern encryption methods and robust authentication mechanisms protect user data and transactions.
2. **Scalability:** Modular design allows for seamless integration of new features in the future.
3. **User-Centric:** Intuitive interfaces and personalization ensure exceptional user experience.
4. **Administrative Control:** Administrators have access to comprehensive tools for managing accounts, transactions, and system security.

By combining these three modules, the **Banking Application** delivers a powerful platform that simplifies banking operations for both users and administrators, ensuring efficiency, reliability, and security.

Top Screenshot:

Method: POST, URL: http://localhost:8080/register, Status: 201 Created, Size: 133 Bytes, Time: 24 ms

Request Body (JSON):

```
1 {
2   "username": "romeo",
3   "name": "Romeo Cat",
4   "accountNumber": "506070",
5   "password": "rome",
6   "email": "romeo2@gmail.com",
7   "city": "Manchester"
8 }
9
```

Response (JSON):

```
1 {
2   "id": 13,
3   "name": "Romeo Cat",
4   "email": "romeo2@gmail.com",
5   "password": "rome",
6   "accountNumber": "506070",
7   "city": "Manchester",
8   "balance": null
9 }
```

Bottom Screenshot:

Method: POST, URL: http://localhost:8080/api/add-transaction?userId=13, Status: 201 Created, Size: 246 Bytes, Time: 71 ms

Request Body (JSON):

```
1 {
2   "amount": 5000,
3   "description": "Deposit for rent payment",
4   "type": "deposit",
5   "date": "2023-01-02"
6 }
7
```

Response (JSON):

```
1 {
2   "id": 15,
3   "amount": 5000.0,
4   "type": "deposit",
5   "details": null,
6   "transactionDate": "2024-11-20T19:46:18.3846416",
7   "user": {
8     "id": 13,
9     "name": "Romeo Cat",
10    "email": "romeo2@gmail.com",
11    "password": "rome",
12    "accountNumber": "506070",
13    "city": "Manchester",
14    "balance": null
15  }
16 }
```

The screenshot displays a REST client interface with two tabs. The top tab shows a POST request to `http://localhost:8080/login` with a JSON body containing user credentials. The response is a 200 OK status with a JSON object representing a user. The bottom tab shows a GET request to `http://localhost:8080/api/transactions?userId=13` with a JSON body containing transaction details. The response is a 200 OK status with a JSON array containing a single transaction object.

Request 1:

Method: POST
URL: `http://localhost:8080/login`
Status: 200 OK, Size: 133 Bytes, Time: 190 ms

Body:

```
{
  "email": "romeo2@gmail.com",
  "password": "rome"
}
```

Response:

```
{
  "id": 13,
  "name": "Romeo Cat",
  "email": "romeo2@gmail.com",
  "password": "rome",
  "accountNumber": "506070",
  "city": "Manchester",
  "balance": null
}
```

Request 2:

Method: GET
URL: `http://localhost:8080/api/transactions?userId=13`
Status: 200 OK, Size: 247 Bytes, Time: 39 ms

Body:

```
{
  "amount": 5000,
  "description": "Deposit for rent payment",
  "type": "deposit",
  "date": "2023-01-02"
}
```

Response:

```
[
  {
    "id": 15,
    "amount": 5000.0,
    "type": "deposit",
    "details": null,
    "transactionDate": "2024-11-20T19:46:18.384642",
    "user": {
      "id": 13,
      "name": "Romeo Cat",
      "email": "romeo2@gmail.com",
      "password": "rome",
      "accountNumber": "506070",
      "city": "Manchester",
      "balance": null
    }
  }
]
```

Endpoint	Method	Description
/api/accounts/user/{userId}	GET	Fetch all accounts associated with a user.
/api/accounts	POST	Create a new account.
/api/accounts/{accountId}/status	PUT	Update the status of an account (e.g., freeze or activate).
/api/accounts/{accountId}	DELETE	Delete an account permanently.
/api/add-transaction	POST	Add a new transaction (deposit or withdrawal).
/api/transactions?userId={userId}	GET	Retrieve all transactions for a specific user.

URLS:

1. **Fetch accounts for a user with ID 1:**
2. <http://localhost:8080/api/accounts/user/1>
3. **Create a new account:**
4. <http://localhost:8080/api/accounts>
5. **Update account status for account ID 123:**
6. <http://localhost:8080/api/accounts/123/status>
7. **Delete account with ID 123:**
8. <http://localhost:8080/api/accounts/123>
9. **Add a new transaction:**
10. <http://localhost:8080/api/add-transaction>
11. **Get all transactions for user with ID 1:**
12. <http://localhost:8080/api/transactions?userId=1>

FRONT-END URLS:

<http://localhost:3000/Home>
<http://localhost:3000/Help>
<http://localhost:3000/About>
<http://localhost:3000/Login>
<http://localhost:3000/SignUp>
<http://localhost:3000/Dashboard>
<http://localhost:3000/AccountDetails>
<http://localhost:3000/Transaction>

MySQL DATABASE

The screenshot shows the MySQL Workbench interface. The 'Query Editor' contains the following SQL script:

```
1 CREATE DATABASE bankingdb;
2 USE bankingdb;
3 SHOW TABLES;
4 SELECT * FROM USER;
5 SELECT * FROM transaction;
6 select * from account;
```

The 'Navigator' pane on the left shows the 'Schemas' section with 'account' and 'Columns' expanded. The 'Information' pane shows the 'Column: id' definition: 'id bigint AI PK'. The 'Result Grid' shows the output of the 'SHOW TABLES' query, listing 'account', 'transaction', and 'user'. The 'Output' pane shows the execution log with the message: '3 01:04:12 select * from account LIMIT 0, 1000 6 row(s) returned'.

The screenshot shows the MySQL Workbench interface. The 'Query Editor' contains the same SQL script as the previous screenshot. The 'Result Grid' shows the output of the 'SELECT * FROM account' query, listing 8 rows of data. The 'Output' pane shows the execution log with the message: '12 01:09:47 select * from account LIMIT 0, 1000 6 row(s) returned' and '13 01:11:40 SELECT * FROM USER LIMIT 0, 1000 11 row(s) returned'.

#	id	account_number	city	email	name	password	username
1	123456789	New York	john@example.com	John Doe	password123	john_doe	
2	123456	Bangalore	zaiba.sharief06@gmail.com	Z Sharief	123456	sharief_z	
3	1112223344	Bangalore	saman@gmail.com	Samanwitha Nanda	pass@saman	saman	
4	000000		aliya@gmail.com		00000000	aliya	
5	98804525	Hogwarts	harry.08@gmail.com	Harry Potter	1234		
6	8904462653	sheffield	zaiba.sharief06@gmail.com	Zaib Sharief	123456		
7	8904462653	Bangalore	zaiba.sharief06@gmail.com	Z Sharief	123456		
8	123456789	pune	alina@gmail.com	Alina Mansoor	010101		

The screenshot displays the MySQL Workbench environment. The 'Query' tab is selected, showing a SQL script with the following queries:

```

1 CREATE DATABASE bankingdb;
2 USE bankingdb;
3 SHOW TABLES;
4 SELECT * FROM USER;
5 SELECT * FROM transaction;
6 select * from account;

```

The 'Result Grid' shows the execution results, including a table with columns: id, account_creation_date, account_number, account_type, balance, branch, last_transaction_date, status, and user_id. The table contains 10 rows of data.

The 'Output' tab shows the execution log with the following messages:

```

14 01:12:02 SELECT * FROM transaction LIMIT 0, 1000
15 01:12:21 select * from account LIMIT 0, 1000

```