CAPSTONE PROJECT

ON

INFINITE WALLET APP

AT

INFINITE COMPUTER SOLUTIONS

Submitted by

Dandamudi Naren

Sai Nikhil kothakonda

Yashaswini N V

Subhash Nalla

Sindhu Peddi

Introduction

The Wallet App project is a comprehensive Payment Wallet Application developed in Java using Spring Boot, with a MySQL database backend. The primary goal of this project is to provide a secure and efficient platform for customers to make payments using digital wallets during their purchases. The application includes a Graphical User Interface (GUI) built with React, ensuring a user-friendly experience. This documentation provides an overview of the key features and technologies employed in the Wallet App.

1.1 Digital Wallets in the Modern Landscape

In the contemporary financial landscape, digital wallets have emerged as transformative tools, reshaping the way individuals manage their finances. The Wallet App project is strategically positioned to align with and capitalize on the dynamic trends and demands within this modern ecosystem.

1.2The Evolution of Digital Wallets

Digital wallets, once considered a novel concept, have evolved into indispensable assets for individuals seeking convenience, security, and flexibility in their financial transactions. The advent of mobile technology has played a pivotal role in this evolution, turning smartphones into powerful hubs for managing diverse aspects of one's financial life.

1.3 Convenience at Your Fingertips

The Wallet App embraces the essence of modernity by placing financial capabilities directly into the hands of users. Through its mobile-centric approach, users can carry out transactions, check account balances, and manage investments from the convenience of their smartphones. This level of accessibility redefines the user experience, making financial interactions more immediate and user-friendly.

1.4 Security in the Digital Age

Security concerns have been at the forefront of digital finance discussions, and the Wallet App addresses these concerns with robust measures. By securely storing payment information and passwords, the app ensures that users can engage in financial transactions with confidence. Multi-layered authentication, including CAPTCHA, adds an extra level of protection against unauthorized access, reflecting the commitment to user data security.

2 Objectives

The Wallet App project is designed with a comprehensive set of objectives to meet the diverse needs of users in the ever-evolving digital financial landscape. Each objective contributes to creating a secure, user-friendly, and feature-rich environment for managing financial transactions seamlessly.

2.1 Seamless Transactions

The primary goal is to provide users with a seamless and efficient means of conducting transactions using digital wallets. The application aims to minimize friction in the payment process, making it intuitive for users to make purchases, transfer funds, and engage in various financial activities without complications.

2.2 User-Friendly Interface

The development of a Graphical User Interface (GUI) using React emphasizes the importance of user experience. The objective is to create an interface that is not only aesthetically pleasing but also intuitive, ensuring that users can navigate through the application effortlessly. By leveraging Redact's capabilities, the app strives to deliver a modern and responsive interface across various devices.

2.3. Financial Empowerment

The Wallet App goes beyond transactional functionalities, aiming to empower users with valuable financial information. By providing real-time updates on daily gold rates and insights into selected Mutual Funds and Stocks, users can make informed decisions about their investments and stay updated on market trends.

2.4. Security and Authentication

Security is a top priority, and the project implements robust authentication mechanisms to safeguard user data. Integration of CAPTCHA enhances the security posture by adding

an additional layer of user verification. These measures ensure that user information and transactions are protected from unauthorized access.

2.5. Comprehensive User Management

The application facilitates comprehensive user management features to enhance the overall user experience. Users can register their Debit/Credit cards, providing a convenient and secure way to manage their financial instruments. Personalized account information allows users to tailor their profiles according to their preferences and needs.

2.6. Record-Keeping and Transparency

Transparent financial dealings are crucial, and the Wallet App addresses this by providing users with detailed transaction reports. Users can access a comprehensive overview of their past transactions, promoting financial transparency and aiding in effective record-keeping.

2.7. Flexibility and Convenience

Recognizing the importance of convenience in modern financial applications, the project integrates features such as bill payments through a payment gateway. This functionality streamlines the process of handling utility payments within the application, offering users a centralized platform for managing various financial activities.

3. Features

3.1. User Login and Registration

- The Login Component is responsible for handling user login. It includes a form with fields for email and password, and it communicates with a server to authenticate the user

State

- **email**: State variable to store the user's email.
- **password**: State variable to store the user's password.
- **Is Logged In**: State variable to track the user's login status.
- **error**: State variable to store and display any error messages.

Styling

- The component uses Bootstrap for styling.

3.2 Registration:

- User registration is fortified with additional security measures, offering a seamless yet secure onboarding experience.

State

- **First name**: State variable to store the user's first name.
- **Last name**: State variable to store the user's last name.
- **email**: State variable to store the user's email.
- **password**: State variable to store the user's password.
- **dob**: State variable to store the user's date of birth.

- **Mobile no**: State variable to store the user's mobile number.
- **address**: State variable to store the user's address.
- **state**: State variable to store the user's state.
- **Pin code**: State variable to store the user's pin code.

Methods

- **save(event)**

Function to handle the form submission. It sends a POST request to the server for user registration. Upon successful registration, it navigates the user to the login page.

Login Navi(event)

Function to handle the "Go to Login" button click. It prevents the default form submission and navigates the user to the login page.

3.3 CHATBOX:

The **My Chatbot** component is a React-based chatbot implemented using the **react-simple-chatbot** library. It is designed to assist users in addressing various issues related to transactions, payments, fixed deposits, gold deposits, credit cards, and debit cards. Users can also choose to describe their issues in their own words.

- **Endpoints**

The chatbot provides information and assistance related to different services:

- **transactions**
- **payments**

- **fixed deposit**
- **gold deposits**
- **credit card**
- **Debit Card**
- **Write issues**

Each endpoint provides additional information based on the user's selection.

3. Card Management

- The **Card Details** component is a React-based component that allows users to input and visualize credit card details. It utilizes the "react-credit-cards" library to display a credit card preview and captures the card number, cardholder name, card expiry, and CVC (Card Verification Code) from user input.

- Users can effortlessly register their Debit/Credit cards, streamlining the transaction process and providing a convenient payment method within the application.

3.1 Component Structure

The component consists of the following elements:

- **Cards Component**: Renders a credit card preview based on the input card details.
- **Form**: Allows users to input credit card details, including card number, cardholder name, expiry date (MM/YY), and CVC.

3.2 State

The component uses the **use State** hook to manage the following states:

- **number**: State variable to store the credit card number.
- **name**: State variable to store the cardholder's name.

- **expiry**: State variable to store the card expiry date.
- **cvc**: State variable to store the Card Verification Code (CVC).
- **focus**: State variable to track the currently focused input field.

3.3 User Interaction

Users can interact with the component by entering their credit card details into the input fields. The credit card preview updates dynamically based on the entered information.

3.4 Form Validation

- The CVC input field has a pattern set to only accept three digits.
- The CVC input field has the "required" attribute set to true.

4. Wallet Transactions

The **Transactions** component is a React-based component that handles bill payments and transactions. Users can input details such as the type of transaction, amount, username, and bill number. The component sends this information to the server for processing.

Efficient Deposit and Withdrawal Functions:
The wallet functionality is designed for efficiency, allowing users to seamlessly deposit funds into their wallets or withdraw as needed, providing flexibility in managing financial resources.

4.1 Transaction History:

- A comprehensive transaction history log is maintained, enabling users to track and review their financial activities over time.

4.2 State

- The component uses the **use State** hook to manage the following states:
- **Type of Trans**: State variable to store the type of transaction.
- **amount**: State variable to store the transaction amount.
- **username**: State variable to store the username.
- **bill_ num**: State variable to store the bill number.

4.3 Form

The component includes a form with the following input fields:

- **bill_ num**: Input field for the bill number.
- **amount**: Input field for the transaction amount.
- **username**: Input field for the username.

5. Bill Payments

The commented-out code appears to be a React component for handling payments using Stripe. The component sets up a form that utilizes Stripe Checkout to process payments. It includes hidden input fields for the amount and uses the Stripe.js library to create a payment button.

5.1 Integration with Payment Gateway:

- Seamless integration with a trusted payment gateway empowers users to settle power and telephone bills directly within the application, eliminating the need for multiple platforms.

- **Stripe.js Script**

The component includes the Stripe.js script (**checkout.stripe.com/checkout.js**) with several data attributes:

- **data-key**: Your Stripe public key.
- **data-amount**: The amount to charge.
- **data-currency**: The currency for the payment.
- **data-name**: Name of your store or service.
- **data-description**: Description of the checkout.
- **data-image**: URL of an image representing your store or service.
- **data-locale**: Language/locale for the Checkout.
- **data-zip-code**: Indicates whether to collect the customer's zip code.

6. Gold Rate Information

Daily Gold Rate Updates:

- The application provides users with up-to-date information on gold rates for both 22 and 24 karats, enabling them to stay informed about market trends and make informed decisions.
- In an effort to provide users with immediate financial insights, the Wallet App's login page features a static display of the current gold rates for both 22 and 24 karats. This feature offers users a quick snapshot of valuable financial information, fostering financial awareness right from the moment they log in.

7. Financial Information

7.1 Mutual Funds Overview:

1. **Fund Performance Metrics:**
   - The application provides users with key performance metrics for selected Mutual Funds, including historical returns, expense ratios, and risk indicators. This comprehensive overview assists users in evaluating the past performance of funds.
2. **Asset Allocation Breakdown:**
   - Users gain insights into the asset allocation strategy of each Mutual Fund, understanding the distribution of investments across asset classes. This breakdown aids in assessing the diversification and risk profile of the funds.
3. **Fund Manager Information:**
   - The application features information about the fund manager, including their professional background and experience. Users can assess the expertise of the fund manager, contributing to their confidence in the fund's management.

7.2 Stock Market Insights:

The **Portfolio** component renders a table displaying portfolio data for various stocks. The data includes stock details such as stock ID, stock name, shares bought, values of shares, latest trading price (LTP), returns since May, and returns between January and April.

Users can access insights into various stocks, including performance history and market trends, facilitating informed decisions in the stock market.

7.3 Data

The **portfolio Data** array contains an array of objects, each representing information about a specific stock. The stock details include the stock ID (**id**), stock name (**STOCK**), shares bought (**SHARES_BOUGHT**), values of shares (**VALUES_OF_SHARES**),

latest trading price (**LTP**), returns since May (**RETURNS_SINCE_MAY**), and returns between January and April (**RETURNS_BETWEEN_JAN_APR**).

7.4 Component Structure

- The component renders a table with headers for each stock detail.
- Each row in the table corresponds to a stock, and data is dynamically populated from the **portfolio Data** array.
- Alternating row colors are used for better visibility.

7. User Profile Management

7.1 Personalized User Profiles:

- Users have the ability to view and personalize their profiles, tailoring the application to their preferences and ensuring a unique and satisfying user experience.

7.2 Profile Security:

- Profile information is securely stored, and users can update their personal details with confidence, knowing that their data is protected.

8. Transaction Reports

The **Transactions** component represents a form for making bill payments. Users can input information such as the type of transaction, amount, username, and bill number. The component also includes functionality for navigating to the login page and submitting the payment details.

8.1 Component Structure

- The component includes a form with input fields for the type of transaction (**type of Trans**), amount (**amount**), username (**username**), and bill number (**bill_ num**).
- The form also includes a button to submit the payment details.
- User input is managed using the **use State** hook.
- The component uses the **use Navigate** hook from 'react-router-dom' to handle navigation.

8.2 State Variables

- **Type of Trans**: Represents the type of transaction (e.g., bill payment).
- **amount**: Represents the amount of the bill payment.
- **username**: Represents the username associated with the bill payment.
- **bill_ num**: Represents the bill number for the payment.

8.3 Functions

- **loginNavi**: Navigates to the login page when called. It is triggered by the button click event.
- **save**: Sends a POST request to the specified API endpoint (**http://localhost:8080/api/transactions/save**) with the payment details. It catches errors and displays an alert if there is an issue.

8.4 Form Inputs

- **bill_num**: Numeric input for the bill number.
- **password**: Password input (Note: It seems like a copy-paste error. If it is not needed, it can be removed or replaced with the correct input).

9. Account Closure

User-Initiated Account Closure:

- Users have the autonomy to initiate the closure of their accounts within the application, providing a straightforward and user-friendly process.

Account Closure Confirmation:

- A secure confirmation process ensures that users are informed about the consequences of closing their accounts and have the opportunity to confirm their decision.

10. Terms and Conditions

Clear and Concise Documentation:

- The terms and conditions section provides users with a clear and concise understanding of the rules and guidelines governing the use of the Wallet App.

Accessibility and Acknowledgment:

- Users are prompted to acknowledge and agree to the terms and conditions before proceeding, fostering transparency and setting clear expectations.

Technology Stack

- **Front End: React**

Utilizes React for building a responsive and dynamic user interface.

- **Database: MySQL**

MySQL is employed as the backend database to store user information and transaction data securely.

- **Programming: Spring Boot with Layered Architecture**
- The backend is developed using Spring Boot, following a layered architecture to ensure modularity and maintainability.
- **IDE Used: Eclipse/STS**
- Developed using Eclipse or Spring Tool Suite (STS) for efficient coding and project management.

Project Structure

The project follows a modular and layered architecture:

- **Presentation Layer (React):**
- Contains React components responsible for the user interface.
- **Business Logic Layer (Spring Boot):**
- Implements the core business logic, including transaction processing and user management.
- **Data Access Layer (MySQL):**
- Manages the interaction with the MySQL database for storing and retrieving data.

Getting Started

Prerequisites

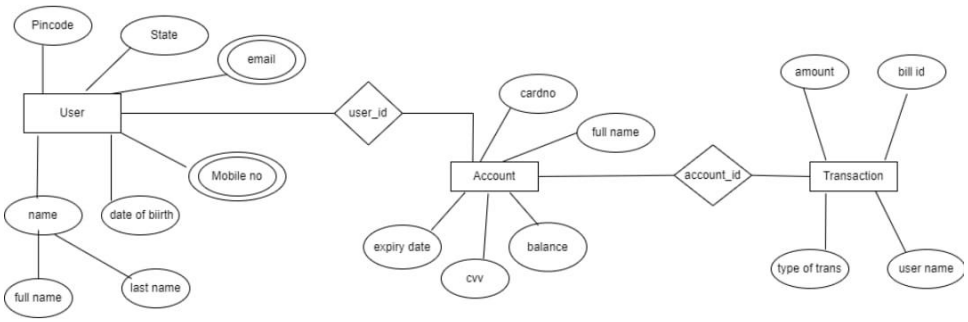
- Ensure you have Node.js and npm installed for React development.
- Set up a MySQL database instance for backend data storage.
- Install Eclipse or Spring Tool Suite (STS) for Java development.

Installation Steps

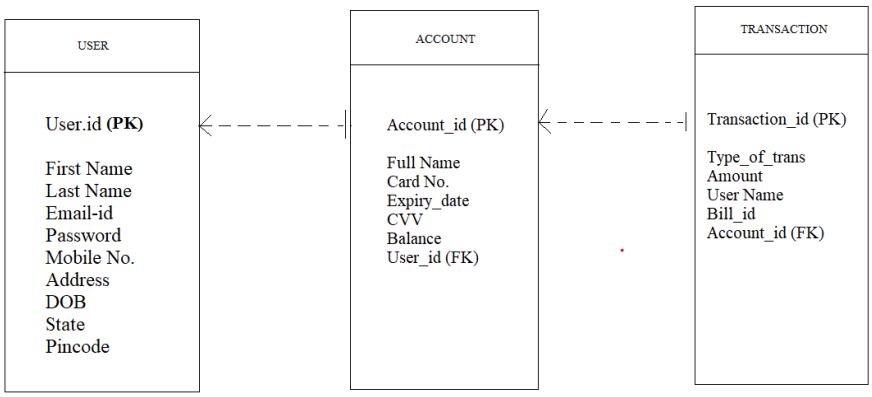
- Clone the project repository.
- Set up the MySQL database with the provided schema.
- Configure the application properties for database connectivity.
- Run the Spring Boot application.
- Navigate to the React frontend directory and run the application.

For detailed installation and configuration steps, refer to the Installation Guide section in the complete documentation.

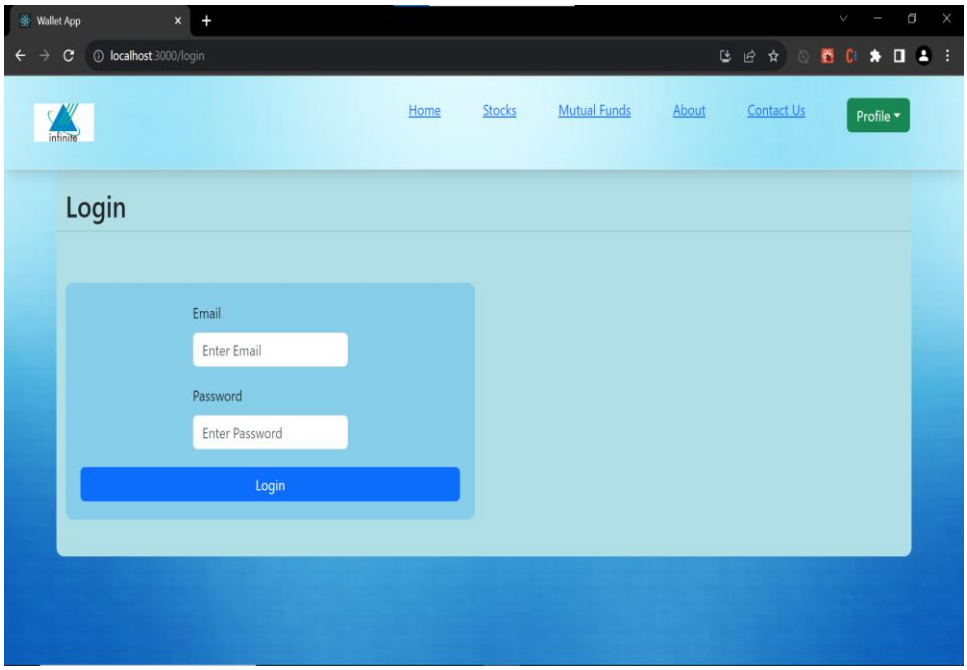
ER DIAGRAM



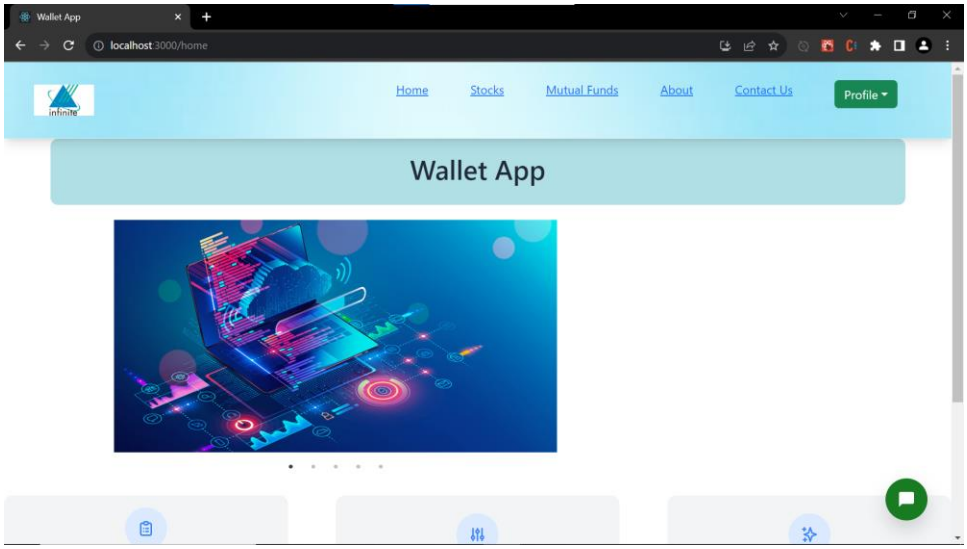
CLASS DIAGRAM



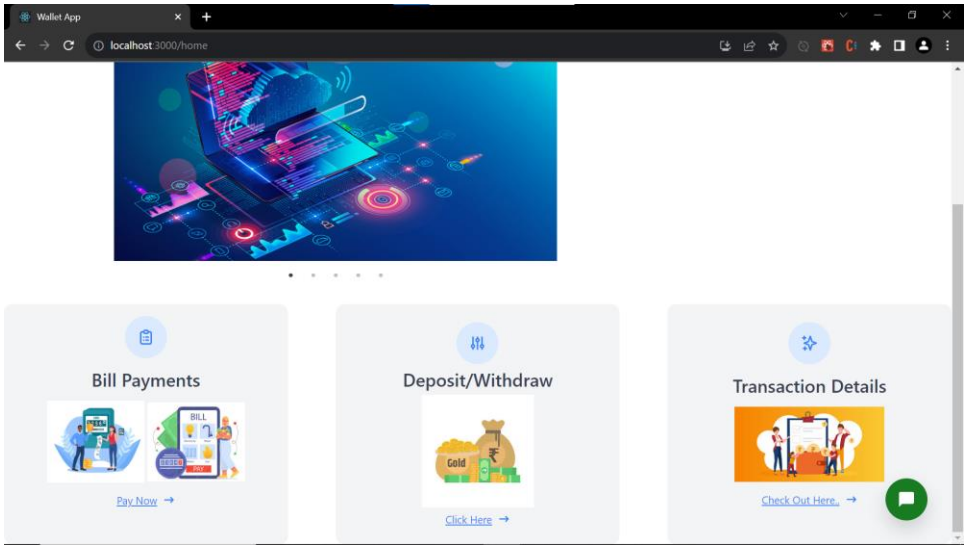
OUTPUT SCREENSHOTS:



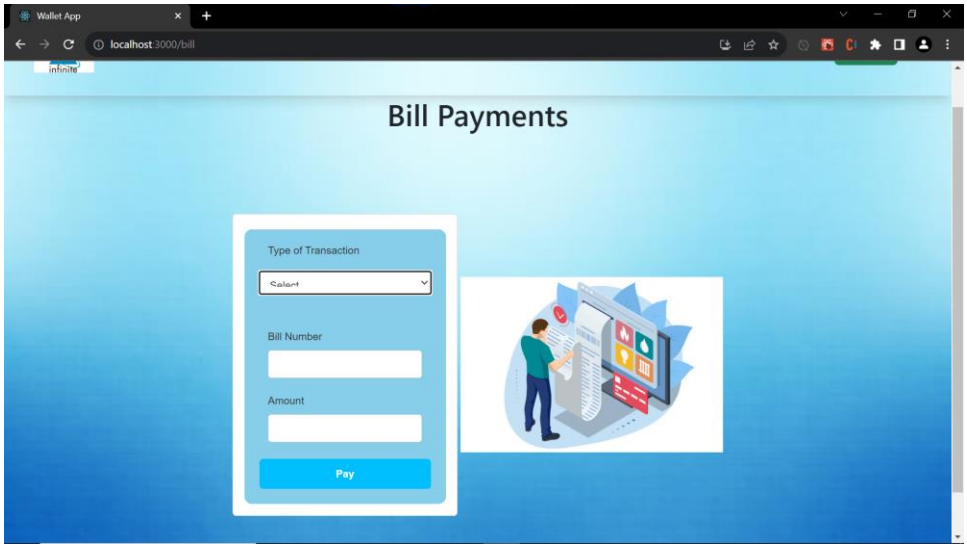
LOGIN PAGE OUTPUT



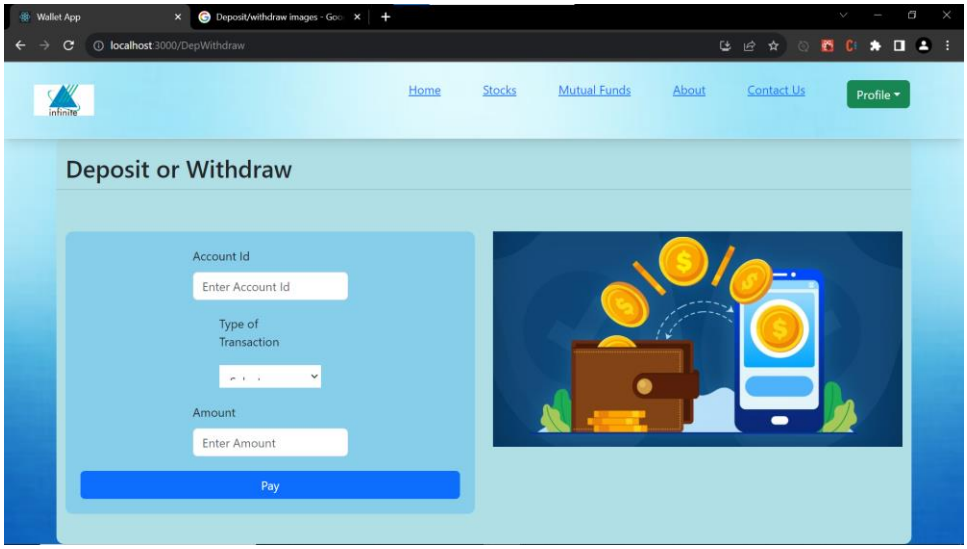
HOME PAGE OUTPUT 1



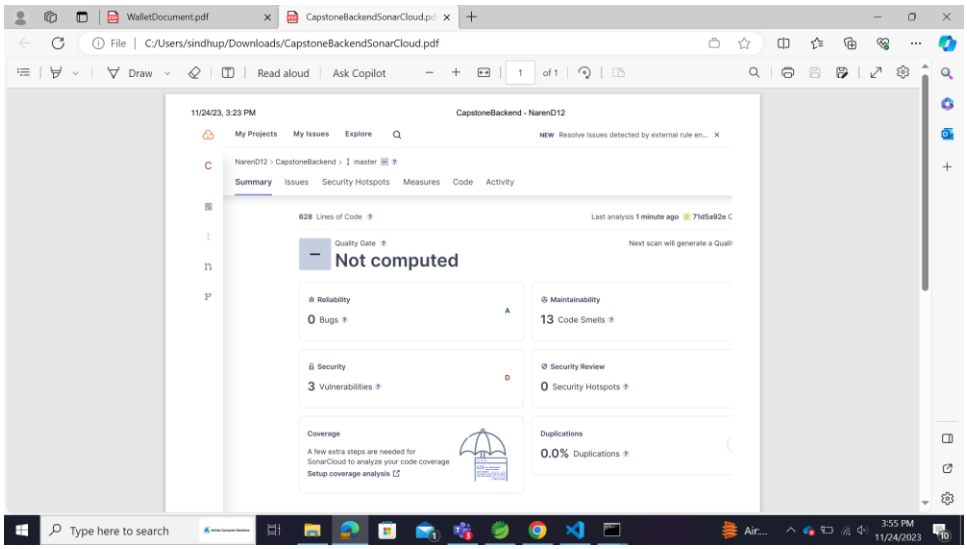
HOME PAGE OUTPUT 2



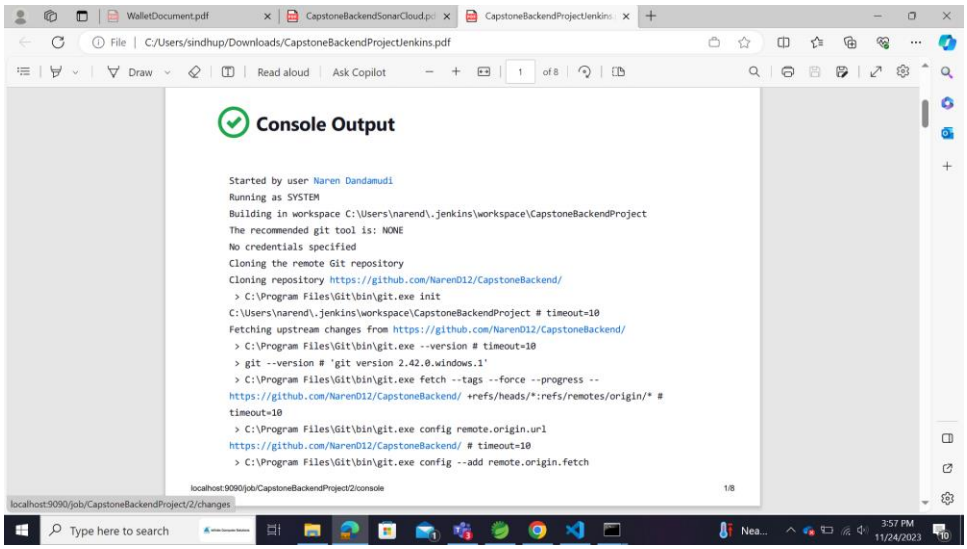
BILL PAYMENT PAGE



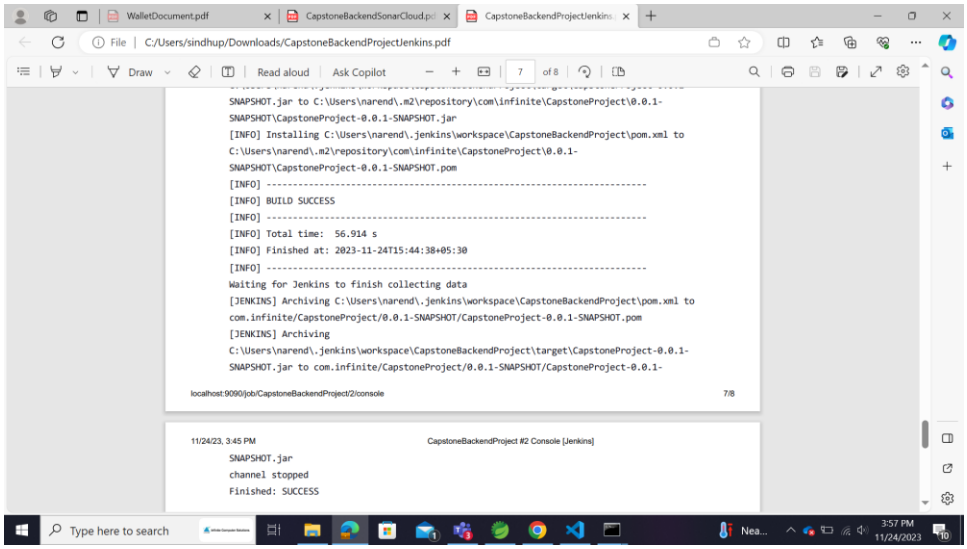
DEPOSIT OR WITHDRAW PAGE



SONARCLOUD



JENKINS



CONCLUTION:

The Wallet App project combines the power of React, Spring Boot, and MySQL to deliver a robust and user-friendly payment wallet application. This documentation provides a high-level overview, and detailed documentation is available for further reference.