

Project Overview

The Stock Market Application is a platform designed to allow users to track, analyze, and manage their stock portfolio seamlessly. It offers real-time stock data, enabling users to make informed decisions for buying and selling stocks. The application integrates RESTful web services to provide backend functionalities, such as managing users, stock data, and transactions, ensuring a smooth interaction between the frontend and backend.

The platform offers features like real-time stock price tracking, volatility analysis, and transaction history, making it an ideal tool for both novice and experienced stock traders.

Objective

The primary goal of this platform is to offer users a real-time experience for managing their stock portfolios, with functionalities for buying, selling, and tracking stock prices. On the technical side, it aims to utilize RESTful web services to handle all backend operations efficiently. The use of a RESTful architecture ensures modularity, scalability, and maintainability.

Technology Stack

The application is built using a modern technology stack that ensures performance and a rich user experience:

- **NetBeans**: Used as the primary IDE for developing the web services in Java.
- **Java DB (Apache Derby)**: Used as the database for storing user data, stock information, and transaction records.
- **RESTful Web Services**: Backend services built with RESTful architecture handle CRUD operations for user and stock data.
- **Python**: AI models for stock predictions and recommendation systems.
- **Marketstack API**: Used to fetch real-time stock prices and data for market analysis.
- **Spring Boot**: Backend framework for integrating web services and handling user authentication and stock transactions.
- **Flask**: Used for AI model deployment for stock recommendations.
- **HTML/CSS/JS**: Frontend development to provide a user-friendly interface for interacting with stock data.

Key Features

Stock Price Analysis

- **Real-time Data**: Users can view live updates of stock prices and historical data.
- **Volatility Analysis**: A chart visualizes the stock's price fluctuations throughout the trading day.

Transaction Management

- **Buy/Sell Stocks**: Users can buy and sell stocks, with the system keeping track of each transaction.
- **Profit/Loss Calculation**: Displays profit/loss after buying or selling any stock.

Transaction History: Users can view a dashboard of all previous transactions with details.

User Management

- **User Authentication:** Users can register, log in, and manage their profiles.
- **Admin Dashboard:** Admins can view and manage user information, including transaction histories.

AI Recommendations

- **Stock Recommendations:** An AI-powered system recommends stocks based on market trends and user portfolio data.
- **Risk Analysis:** The system provides insights into stock volatility and potential risks.

RESTful Web Services

The backend is powered by RESTful web services, allowing for efficient data management and interaction. The following CRUD operations are handled:

- **Create:** User registration, adding new stock to the portfolio, adding transactions.
- **Read:** Fetching user details, stock data, transaction history, and stock price charts.
- **Update:** Updating user profiles, stock prices, and transaction statuses.
- **Delete:** Removing stocks from a user's portfolio, deleting user accounts.

Email Notifications

Upon buying or selling a stock, users receive a confirmation email, providing a summary of the transaction. This system is implemented using RESTful web services to enhance communication and improve user experience.

Future Expansion Ideas

- **Mobile Application:** Developing a mobile app for easy stock tracking on the go.
- **Advanced AI Models:** Integrating more sophisticated AI models for predicting stock trends.
- **Enhanced Security:** Implementing features like two-factor authentication to ensure data security.
- **Community Features:** A forum where users can discuss stocks, trends, and market predictions.

SWOT Analysis for Stock Market Application

Strengths

- **Real-Time Stock Data:** The platform provides up-to-date stock price data, allowing users to make timely decisions for buying or selling stocks.
- **RESTful Architecture:** The use of RESTful web services ensures that the backend is modular, scalable, and easy to maintain, providing efficient CRUD operations.
- **AI-Powered Stock Recommendations:** The inclusion of AI-based recommendations enhances the user experience by helping users choose better stocks based on market trends.

- **Transaction History & Profit/Loss Calculation:** The system tracks all transactions and provides users with a detailed profit/loss analysis for better financial decision making.
- **User-Friendly Interface:** A clean and intuitive UI design makes it easy for both novice and experienced traders to navigate the platform.
- **Security:** The platform offers basic authentication features to secure user data and transactions.

Weaknesses

- **Limited AI Models:** Currently, the AI functionality may be limited to basic stock recommendations. More advanced AI models could improve stock predictions.
- **Potential Performance Bottlenecks:** Depending on the volume of stock data processed in real-time, there could be performance issues that affect data retrieval speeds.
- **Basic Security Features:** While user authentication is implemented, the platform could benefit from enhanced security measures like two-factor authentication or data encryption.

Opportunities

- **Mobile Application:** Developing a mobile version would increase the platform's accessibility and engage users who prefer trading and tracking stocks on mobile devices.
- **Enhanced AI Models:** Expanding the AI models to include more advanced prediction algorithms and portfolio optimization could attract more experienced traders.
- **Integration with Financial Services:** Partnering with financial institutions or integrating third-party services (e.g., stock exchanges, banking services) could expand the platform's functionality.
- **Community-Driven Features:** Implementing features like a discussion forum or social trading capabilities could increase user retention and engagement.

Threats

- **Market Competition:** Many stock trading platforms offer similar features. Competitors with more resources or advanced AI capabilities could pose a threat to user retention.
- **Regulatory Issues:** As the platform grows, it may need to comply with financial regulations related to stock trading and user data privacy.
- **Cybersecurity Risks:** As the platform manages financial transactions and user data, it could become a target for cyberattacks. Continuous security updates would be necessary.
- **Stock Market Volatility:** Fluctuations in the stock market may affect user engagement, as significant downturns might discourage users from trading.

System Requirements

Functional Requirements

1. User Registration & Authentication

- Users should be able to register, log in, and log out securely.
- Admin users should have access to a dashboard to view and manage user data.
- 2. Real-Time Stock Data**
 - Fetch and display real-time stock prices using the Marketstack API.
 - Display stock volatility charts and price trends for each stock.
- 3. Stock Transactions**
 - Users should be able to buy and sell stocks.
 - Stock buying should be based on available real-time prices.
 - A transaction history should be available to users, including the ability to view profit or loss from each transaction.
- 4. AI-Powered Stock Recommendations**
 - Provide AI-driven stock recommendations based on market data and user preferences.
 - Analyze stock volatility and provide insights to users about potential risks.
- 5. Admin Management**
 - Admin users should have the ability to view, search, update, and delete user details.
 - Admins should be able to track stock transactions by users and manage stock data.
- 6. Email Notifications**
 - Send confirmation emails to users upon successful transactions.
 - Provide users with updates on their portfolio performance or stock recommendations via email.

Non-Functional Requirements

- 1. Scalability**
 - The platform should be scalable to handle increasing amounts of data and a growing number of users as it expands.
- 2. Performance**
 - Real-time stock data should be updated quickly to ensure users have the latest information when making trades.
 - The platform should handle high traffic efficiently without slowing down.
- 3. Security**
 - Ensure secure handling of user credentials, stock data, and transactions.
 - Implement secure API authentication for external integrations such as the Marketstack API.
- 4. Usability**
 - The platform should have an intuitive and user-friendly interface to ensure ease of use.
 - The system should provide clear navigation and easily accessible stock data and transaction history.
- 5. Reliability**
 - The system should ensure minimal downtime and consistent access to real time stock data.
 - It should offer reliable performance, even during high market volatility.
- 6. Maintainability**
 - The codebase should be modular, with clear separation of concerns between frontend, backend, and data layers.
 - RESTful services should follow best practices, making the platform easy to update and maintain.
- 7. Compliance**
 - Ensure that the platform complies with any legal and financial regulations

related to stock trading and data privacy.

Code:

Register.java

```
package com.WS;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.xml.bind.annotation.XmlRootElement;

/**
 *
 * @author pawan
 */
@Entity
@XmlRootElement
public class Register implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id fields are not set if
        (!(object instanceof Register)) {
            return false;
        }
        Register other = (Register) object;
        if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }
}
```

```
private String Name;
```

```
/**  
 * Get the value of Name  
 *  
 * @return the value of Name  
 */  
public String getName() {  
    return Name;  
}
```

```
/**  
 * Set the value of Name  
 *  
 * @param Name new value of Name  
 */  
public void setName(String Name) {  
    this.Name = Name;  
}
```

```
private String Password;
```

```
/**  
 * Get the value of Password  
 *  
 * @return the value of Password  
 */  
public String getPassword() {  
    return Password;  
}
```

```
/**  
 * Set the value of Password  
 *  
 * @param Password new value of Password  
 */  
public void setPassword(String Password) {  
    this.Password = Password;  
}
```

```
private String ConfirmPassword;
```

```
/**  
 * Get the value of ConfirmPassword  
 *  
 * @return the value of ConfirmPassword  
 */  
public String getConfirmPassword() {  
    return ConfirmPassword;  
}
```

```
/**  
 * Set the value of ConfirmPassword  
 *  
 * @param ConfirmPassword new value of ConfirmPassword  
 */  
public void setConfirmPassword(String ConfirmPassword) {
```

```
this.confirmPassword = confirmPassword;
}
```

```
private String Email;
```

```
/**
 * Get the value of Email
 *
 * @return the value of Email
 */
public String getEmail() {
    return Email;
}

/**
 * Set the value of Email
 *
 * @param Email new value of Email
 */
public void setEmail(String Email) {
    this.Email = Email;
}
```

```
@Override
public String toString() {
    return "com.WS.Register[ id=" + id + " ]";
}

}
```

RegisterFacade.java

```
/*
 * To change this license header, choose License Headers in Project
 * Properties. * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.WS;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.xml.bind.annotation.XmlRootElement;

/**
 *
 * @author pawan
 */
@Entity
@XmlRootElement
public class Register implements Serializable {
```

```

private static final long serialVersionUID = 1L;
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Long id;

public Long getId() {
return id;
}

public void setId(Long id) {
this.id = id;
}

@Override
public int hashCode() {
int hash = 0;
hash += (id != null ? id.hashCode() : 0);
return hash;
}

@Override
public boolean equals(Object object) {
// TODO: Warning - this method won't work in the case the id fields are not set if
(!object instanceof Register)) {
return false;
}
Register other = (Register) object;
if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {
return false;
}
return true;
}
private String Name;

/**
 * Get the value of Name
 *
 * @return the value of Name
 */
public String getName() {
return Name;
}

/**
 * Set the value of Name
 *
 * @param Name new value of Name
 */
public void setName(String Name) {
this.Name = Name;
}

```



```
}
```

```
private String Password;
```

```
/**
```

```
 * Get the value of Password
```

```
 *
```

```
 * @return the value of Password
```

```
 */
```

```
public String getPassword() {
```

```
    return Password;
```

```
}
```

```
/**
```

```
 * Set the value of Password
```

```
 *
```

```
 * @param Password new value of Password
```

```
 */
```

```
public void setPassword(String Password) {
```

```
    this.Password = Password;
```

```
}
```

```
private String ConfirmPassword;
```

```
/**
```

```
 * Get the value of ConfirmPassword
```

```
 *
```

```
 * @return the value of ConfirmPassword
```

```
 */
```

```
public String getConfirmPassword() {
```

```
    return ConfirmPassword;
```

```
}
```

```
/**
```

```
 * Set the value of ConfirmPassword
```

```
 *
```

```
 * @param ConfirmPassword new value of ConfirmPassword
```

```
 */
```

```
public void setConfirmPassword(String ConfirmPassword) {
```

```
    this.ConfirmPassword = ConfirmPassword;
```

```
}
```

```
private String Email;
```

```
/**
```

```
 * Get the value of Email
```

```
 *
```

```
 * @return the value of Email
```

```
 */
```

```
public String getEmail() {
```

```
    return Email;
```

```

}

/**
 * Set the value of Email
 *
 * @param Email new value of Email
 */
public void setEmail(String Email) {
    this.Email = Email;
}

@Override
public String toString() {
    return "com.WS.Register[ id=" + id + " ]";
}

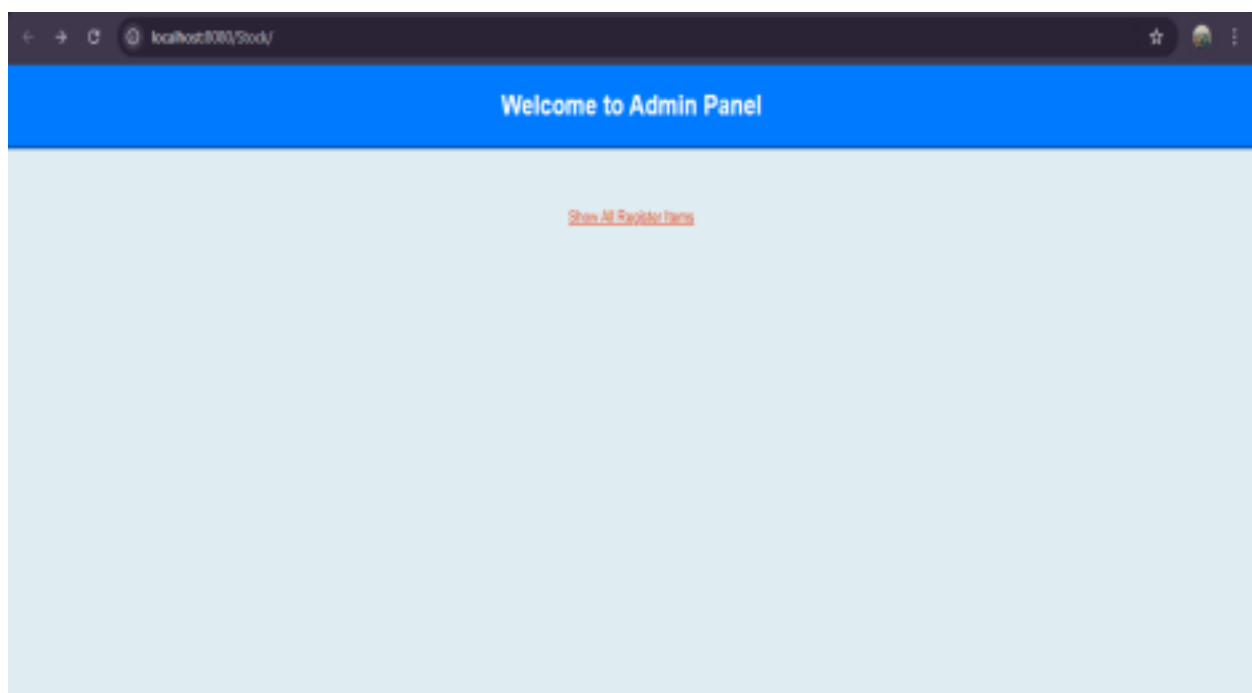
```

Conclusion

The Stock Market Application provides a comprehensive platform for stock trading, analysis, and management. By leveraging RESTful web services, it offers a scalable and robust backend, ensuring that users can efficiently manage their portfolios. The platform's future development will focus on enhancing AI capabilities and expanding its functionality to improve user experience.

Output:

Stock Web App



localhost:8080/Stock/faces/register/List.xhtml

Create New Register

Name:

Password:



Confirm Password:

Email:

[Register](#)

[Show All Registered Items](#)

localhost:8080/Stock/faces/register/home.xhtml



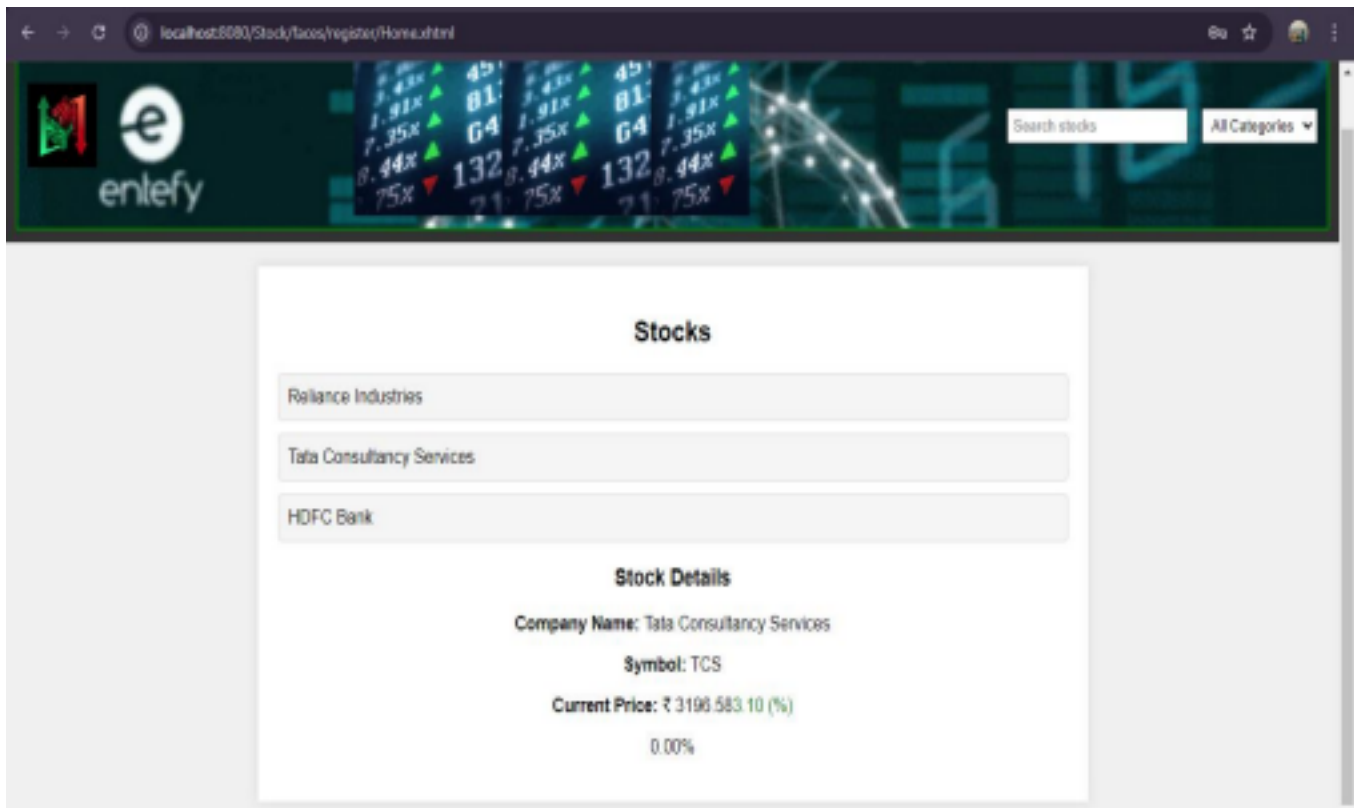
All Categories ▾

Stocks

Reliance Industries

Tata Consultancy Services

HDFC Bank



List

1.55

Id	Name	Password	Confirm Password	Email	
501	ram	111222	11222	raamu@gmail.com	View Edit Destroy
551	Ramu	121212	121212	raamu@gmail.com	View Edit Destroy
601	Raj	111111	111111	raj@gmail.com	View Edit Destroy
602	1122vew	434343	434343	raj@gmail.com	View Edit Destroy
703	ram	2121	2121	raamu@gmail.com	View Edit Destroy

[Create New Register](#)

[Index](#)

(CRUD Operation)

Create:

localhost:8080/Stock/Faces/register/List.xhtml

Create New Register

Name:

Password:

ConfirmPassword:

Email:

[Register](#)
[Show All Register Items](#)

Read

localhost:8080/Stock/Faces/register/View.xhtml

View

Id: 501
Name: nam
Password: 111222
ConfirmPassword: 111222
Email: namu@gmail.com

[Destroy](#)
[Edit](#)
[Create New Register](#)
[Show All Register Items](#)
[Index](#)

Update

Edit Register

Id:	<input type="text" value="501"/>
Name:	<input type="text" value="ram"/>
Password:	<input type="password"/>
ConfirmPassword:	<input type="password"/>
Email:	<input type="text" value="ramu@gmail.com"/>

[Save](#) [View](#) [Show All Register Items](#) [Index](#)

Delete

View

Register was successfully deleted.

Id: 602
Name: 1122view
Password: 434343
ConfirmPassword: 434343
Email: raj@gmail.com

[Delete](#)

[Edit](#)

[Create New Register](#)
[Show All Register Items](#)

[Index](#)