|  |  |  |
| --- | --- | --- |
|  | **NAAN MUDHALVAN** |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |

**Stocker: Cloud-Based Stock Trading Platform With Flask On Aws Ec2 And Rds**

**Project Created by:**

**Project Created Date: 21/Nov/2024**

**College Code: 1106**

**College Name: Indira institute of engineering and technology**

**Team Name:**

***BONAFIDE CERTIFICATE***

Certified that this Naan Mudhalvan project report **“Stocker: Cloud-Based Stock Trading Platform With Flask On Aws Ec2 And Rds”** is the Bonafide work of \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ who carried out the project work under my supervision.

**SIGNATURE SIGNATURE**

**Project Coordinator SPoC**

**Naan Mudhalvan Naan Mudhalvan**

INTERNAL EXAMINER EXTERNAL EXAMINER

**Objective**

The **Stocker Application** aims to provide users with a secure and user-friendly platform to:

1. **Register and manage user accounts.**
2. **Authenticate users securely using encrypted credentials.**
3. **Facilitate stock transactions, including buying and selling stocks.**The project leverages a robust backend and a responsive frontend to ensure scalability and reliability for user operations.
4. **Buying and selling stocks.**The project leverages a robust backend and a responsive frontend to ensure scalability and reliability for user operations.

**Scope**

1. Enable users to **register and log in** using a simple interface while ensuring robust security through password hashing and input validation.

1. Provide a system for users to **execute stock trades**, including validation for stock symbols, quantities, and prices.
2. **Store user data and transactions** in a secure relational database, enabling future expansion to more complex trading functionalities.
3. Deliver a responsive and accessible **user interface**, adaptable to both desktop and mobile platforms.

**Technology Stack**

**Frontend:**

* **HTML5**: For semantic content structuring.
* **CSS3**: For responsive and visually appealing styling.
* **JavaScript**: Adds interactivity, including real-time feedback for users.

**Backend:**

* **Flask**: A lightweight Python web framework for routing, handling requests, and API creation.
* **Flask-SQLAlchemy**: An ORM (Object-Relational Mapper) for database management.
* **Flask-Bcrypt**: Ensures secure password storage through hashing.
* **Flask-CORS**: Allows secure cross-origin requests between the frontend and backend.

**Database:**

* **MySQL (hosted on AWS RDS)**: Relational database for storing user data and stock transactions.

**Environment Management:**

* **dotenv**: Manages sensitive configuration data, such as database credentials, through environment variables.

**System Features**

**1. User Management**

* **Registration**:
  + New users can register with a username and password.
  + Ensures password strength with a minimum length of 8 characters.
  + Encrypts passwords using **bcrypt** to protect sensitive data.
  + Handles duplicate usernames gracefully with database constraints.
* **Login**:
  + Authenticates users by verifying credentials against the database.
  + Uses **bcrypt** to compare hashed passwords securely.
  + Provides feedback on invalid credentials.

**2. Stock Transactions**

* Enables users to perform **buy** and **sell** operations on stocks.
* Validates stock symbol, quantity, and price before committing transactions.
* Ensures secure and consistent transaction recording in the database.

**3. Responsive UI**

* Dynamic forms and buttons provide real-time feedback during interactions.
* Uses CSS for clean and visually appealing layouts.
* Responsive design ensures compatibility across devices (desktop, tablet, mobile).

**4. Security**

* **Password Hashing**:
  + Implements hashing with **bcrypt** for secure password storage.
* **Cross-Origin Requests**:
  + Uses Flask-CORS to enable secure communication between the frontend and backend.
* **Environment Variables**:
  + Manages sensitive credentials securely using .env.

**Implementation Details**

**Frontend**

The frontend consists of an HTML and CSS-based interface enhanced with JavaScript. It includes two primary sections:

1. **Authentication Section**: Allows users to log in or register.
2. **Trading Section**: Displays stock trading options after successful authentication.

**Key Features:**

* **Dynamic Button State**: Buttons are disabled during API requests to prevent multiple submissions:

javascript

Code:

setButtonState(true, 'register-btn');

* **Input Validation**: Ensures all fields are populated and meet constraints before submitting forms:

javascript

Code:

if (!username || !password) {

alert('Username and password are required!');

return;

}

* **Blurred Background Effect**: Provides a modern look and keeps the focus on the main content:

css

Code:

body::before {

filter: blur(10px);

}

**Backend**

The backend is built using Flask, which provides the routing, authentication, and trading logic. Key highlights include:

1. **Database Models**:
   * **User Model**: Stores user credentials with unique constraints and hashed passwords.

python

Code:

class User(db.Model):

id = db.Column(db.Integer, primary\_key=True)

username = db.Column(db.String(80), unique=True, nullable=False)

password = db.Column(db.String(255), nullable=False)

* + **StockTransaction Model**: Tracks individual stock transactions for each user.

python

Code:

class StockTransaction(db.Model):

id = db.Column(db.Integer, primary\_key=True)

user\_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)

stock\_symbol = db.Column(db.String(10), nullable=False)

1. **Secure Password Hashing**: Passwords are hashed during registration and verified during login:

python

Code:

hashed\_password = bcrypt.generate\_password\_hash(data['password']).decode('utf-8')

1. **Transaction Handling**: Validates user inputs and processes stock transactions securely:

python

Code:

transaction = StockTransaction(

user\_id=int(data['user\_id']),

stock\_symbol=data['stock\_symbol'].upper(),

quantity=int(data['quantity']),

transaction\_type=data['transaction\_type'].lower(),

price=float(data['price'])

)

db.session.add(transaction)

db.session.commit()

1. **Error Handling**: Provides meaningful feedback on errors such as invalid inputs or database issues:

python

Code:

except IntegrityError:

db.session.rollback()

return jsonify({"message": "Username already exists"}), 400

**Database Design**

**Schema:**

1. **Users Table**:
   * **id**: Primary key.
   * **username**: Unique string for user identification.
   * **password**: Hashed password string.
2. **StockTransactions Table**:
   * **id**: Primary key.
   * **user\_id**: Foreign key referencing the Users table.
   * **stock\_symbol**: Stores the stock's ticker symbol.
   * **quantity**: Number of shares in the transaction.
   * **transaction\_type**: buy or sell.
   * **price**: Transaction price per share.

**Deployment**

**Hosting:**

* **Frontend**: Can be served via Flask or a static file host (e.g., GitHub Pages).
* **Backend**: Hosted on **AWS EC2** or **Heroku**.
* **Database**: MySQL database hosted on **AWS RDS**.

**Environment Management:**

* Sensitive configurations (e.g., database credentials) are stored in .env files.

**Challenges and Solutions**

1. **Handling Concurrent Requests**:
   * **Solution**: Ensured database transactions are atomic.
2. **Data Security**:
   * **Solution**: Used hashing for passwords and environment variables for sensitive data.
3. **Validation**:
   * **Solution**: Implemented both frontend and backend validation.

**Future Enhancements**

1. **JWT Authentication**:
   * Replace user ID-based session handling with JWT tokens.
2. **Real-Time Stock Data**:
   * Integrate APIs to fetch live stock prices and market data.
3. **User Dashboard**:
   * Display transaction history and account details.
4. **Email Notifications**:
   * Send alerts for successful trades or significant market changes.
5. **Mobile App Integration**:
   * Expand to native mobile platforms using frameworks like React Native or Flutter.

**Conclusion**

The **Stocker Application** successfully delivers a foundational platform that combines ease of use, security, and scalability. By integrating robust authentication mechanisms, secure password handling, and database-backed transaction management, it ensures both user data integrity and a seamless trading experience. The choice of Flask for backend operations, coupled with SQLAlchemy for database management, provides a solid framework for handling complex operations while maintaining simplicity in code structure. Similarly, the frontend prioritizes accessibility, responsiveness, and a user-friendly interface to cater to a broad audience.

This project highlights the importance of aligning security, functionality, and design to create a balanced application. The use of bcrypt for password hashing and JWT readiness ensures that security remains a top priority. At the same time, careful attention to UI/UX elements, such as dynamic button states and real-time feedback, enhances the overall user experience. The application is designed with future growth in mind, providing a clear path for additional features like real-time stock data integration, advanced analytics, and mobile app support.

By laying a robust foundation, the **Stocker Application** positions itself as a scalable solution for users looking to manage and trade stocks. While its current capabilities focus on core functionalities, the system is ready for modular expansion, making it suitable for both beginner and experienced users. The project reflects a blend of technical skill, thoughtful design, and strategic planning, serving as an example of how technology can simplify and enhance stock trading in today’s digital age. With continuous updates and the integration of additional features, this platform has the potential to evolve into a comprehensive trading tool for a diverse user base.