Table of Contents

Introduction:

Crypto Market is a web-based application, which is designed to enhance user experience in the cryptocurrency market, which is now a top trending market in the world. This project, developed for the A-level non-exam assessment (NEA), its aim is to advance software development skills by addressing a complex problem into, the real-world problem. In its development, Crypto Market follows the Software Development Life Cycle (SDLC). This structured approach is planning, analysis, design, development, testing, implementation, and maintenance, ensuring thoroughness and quality in the software development process. According to the requirements, our focus is too prototyping, and regular feedback which is according to agile methodologies. [1]

Analysis:

Problem statement:

In the world of cryptocurrency trading, the main difficulty faced by traders, and investors is the fragmented access to crucial market data. Traders who are business investors often find themselves sifting through multiple platforms to track the real-time prices of various cryptocurrencies, understand market trends, and compare exchange rates. This approach to information gathering is not just cumbersome but also highly inefficient, significantly impacting decision-making in a market, where every second counts because every second market changes, so the traders and users need such a system. The Crypto Market project is designed to handle this very issue by introducing a centralized, user-friendly platform that consolidates real-time data of different digital currencies from these cryptocurrency exchanges. This solution aims to streamline the process of market analysis, making it more accessible to every crypto user and actionable, for users. By providing an integrated view of the market, Crypto Market empowers users with the tools they need to make informed, timely trading decisions, effectively addressing, the challenges of navigating, the fast-paced and volatile cryptocurrency landscape.

Background details:

The application's architecture is purposefully designed to cater to essential demands:

User Account Management:

The user account management is basically creating an account for the user of the application. This Module also benefits users for accessing the application securely. User account management also facilitates the user to reset his password.

- Dashboard of our system:

We use the CCXT library for fetching the data, which is mainly displayed on the screen. Here we show news related to cryptocurrency on the dashboard. This dashboard comes after the login page.

- Data Retrieval:

We use the CCXT library for fetching the real time data of different crypto currencies. These crypto currencies are displayed on the main screen, this takes some time for fetching the data. It eases integration.

- User Management:

User Management handles the login and user registration which is basically a session management. We use Flask and SQLite. The Flask provides us simplicity and flexibility. SQLite is used for storing data in databases.

- Data Presentation: Displaying data in a user-friendly format.

Technology Justification:

- Flask: Flask is a python web framework which provides facility for user to use web on python language. Its use is very easy and simple. Its development is rapid as compared to other frameworks.
- SQLite: It is a Database mainly used for storing the data, in our application we use for storing the data of user. It is also called a lightweight database.
- bcrypt: It is used mostly for encryption means as a security purpose. Its use in passwords security and it serves as a strong capability.
- CCXT Library: It is the library used for the fetching of real-time data. Its coverage is very vast for cryptocurrency exchanges. [2]

Problem Solved:

- Users (Traders, Investors, Cryptocurrency Enthusiasts): These are the primary users of the platform.
- Developers: This group includes individuals and teams responsible for building, testing, and maintaining the platform.

- Exchanges (Platforms Providing Cryptocurrency Price Data): These are secondary stakeholders in the project.
- Casual Investors: Individuals that have a moderate interest in the cryptocurrency market.
- Serious Traders: Those traders who rely on up-to-the-minute data for day trading.
- Cryptocurrency Enthusiasts: Those Individuals interested in market trends and news.

Research methodology:

- User Interviews and Feedback:

We conduct different interviews and gather feedback, in this the google forms use for gathering the feedback. The feedback provides a comprehensive solution to our problem. We share the google forms in different groups, then we gather feedback.

- Analysis of Existing Platforms:

Our team thoroughly reviewed existing cryptocurrency tracking platforms. We looked at their features, ease of use, and how they presented data. For example, we noticed that a popular platform had comprehensive data but a very complex interface, which was intimidating for new users.

Technology Evaluation:

We evaluated various technologies for their suitability in our project. This included a deep dive into the Flask framework for its simplicity and flexibility, and SQLite for its lightweight nature. We also assessed the CCXT library for its extensive coverage of cryptocurrency exchanges.

Users Requirements:

- Real-Time Price Data (FR-04):

One of the primary requirements identified by most users was the need for real-time price updates of various cryptocurrencies, the reason is that this market changes in seconds so the need a real time data of currencies. [3]

- Comprehensive Price Comparison (FR-01):

Users expressed the need to view and compare prices from multiple exchanges in one place. This was particularly important for users looking to engage in arbitrage trading.

- User-Friendly Interface (NFR 01):

Newcomers to the cryptocurrency market, such as John, highlighted the need for an intuitive and easy-to-navigate interface. This led to the requirement for a user-friendly design that simplifies the complexity of cryptocurrency data.

Mostly, highlights the importance of authentication like login and registration.

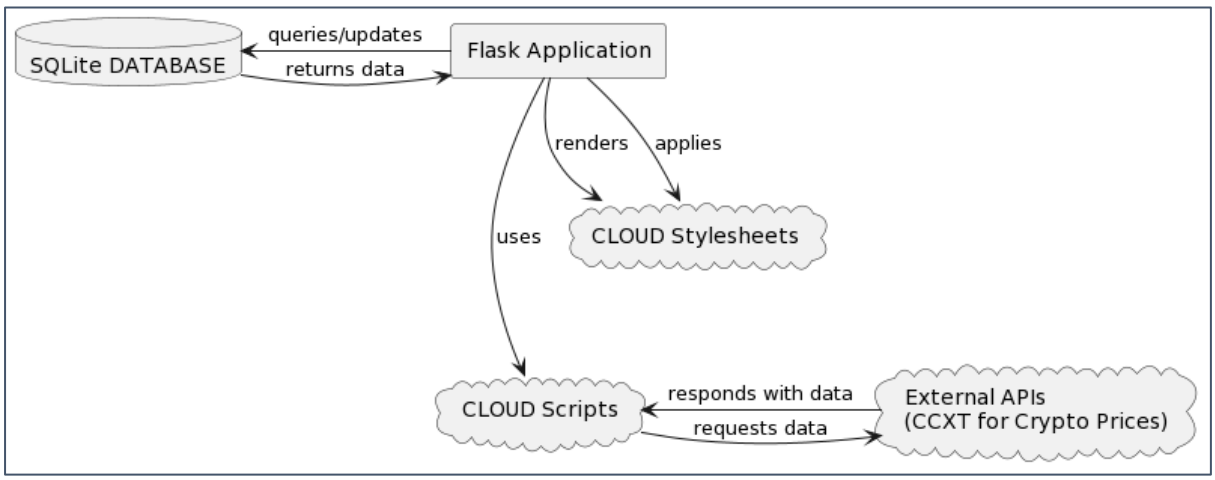
Objectives lists:

This system aims to:

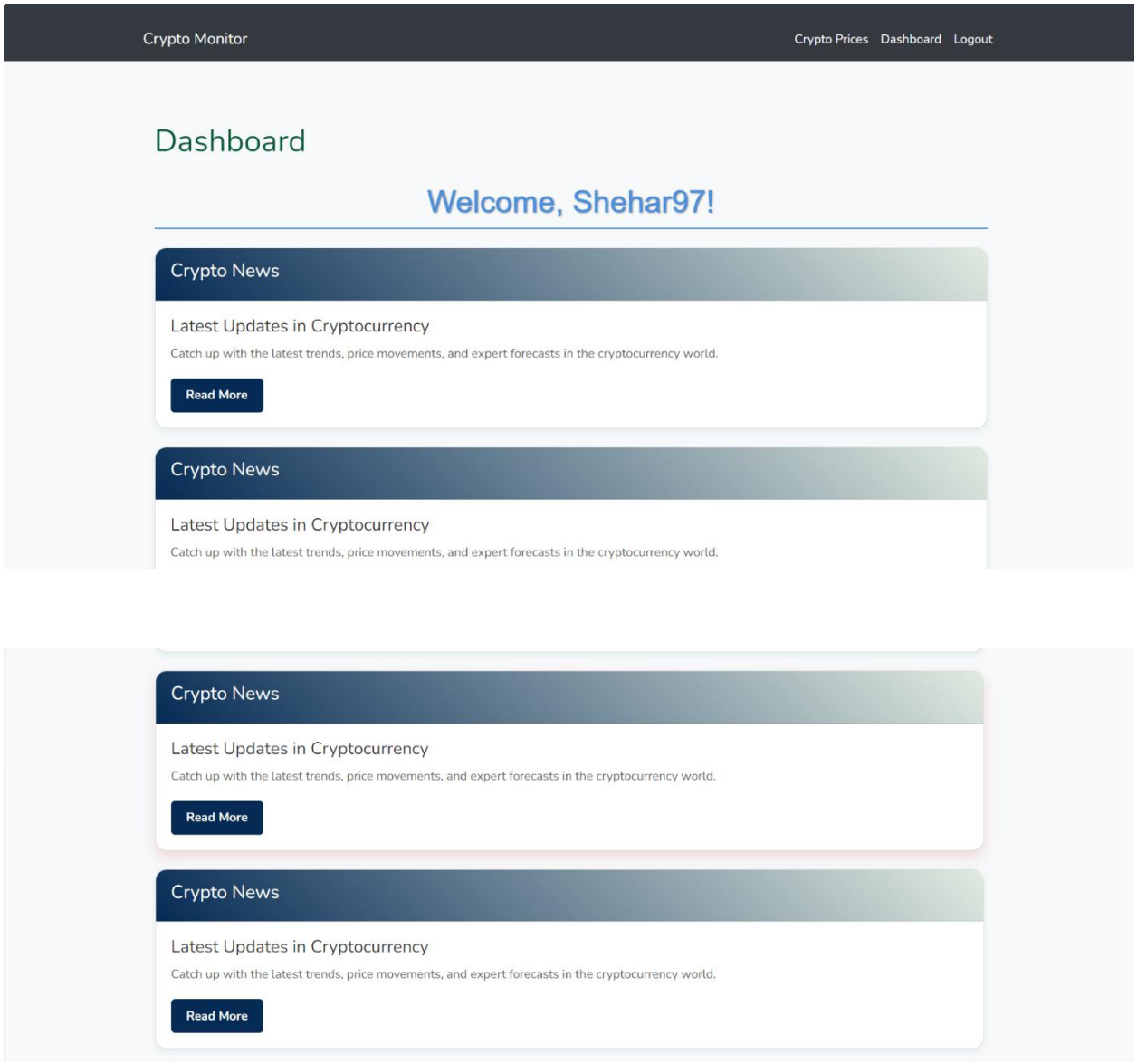
- User-friendly platform: Develop a user-friendly platform with robust functionalities, including user registration, login systems and personalized user experiences.
- Data accuracy: Ensure data accuracy and reliability by integrating robust backend technologies that fetch and display real-time data.
- User Engagement: Measured by active user count and session duration, aiming for a steady increase post-launch.
- Accuracy of Information: Regular checks against exchange data to ensure price accuracy. The target is a high accuracy rate greater than 90%.
- User Satisfaction: Regular user surveys post-launch to gauge satisfaction, aiming for high satisfaction scores.

Problem modelling:

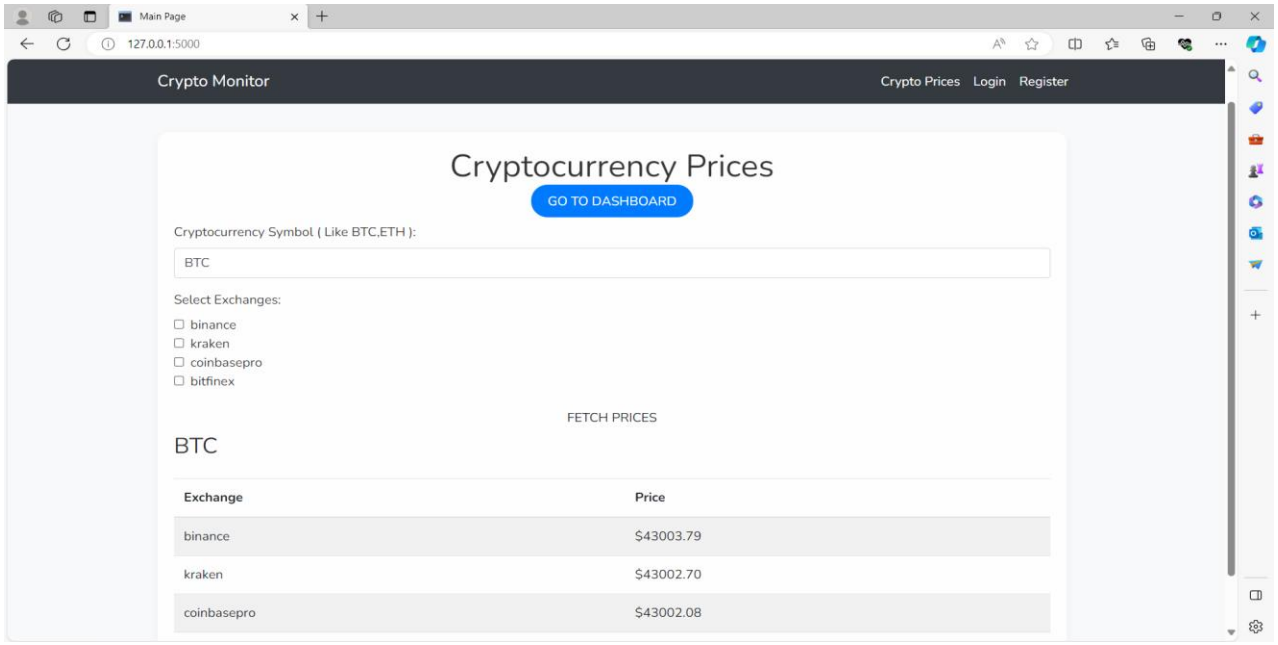
- Architectural Design:



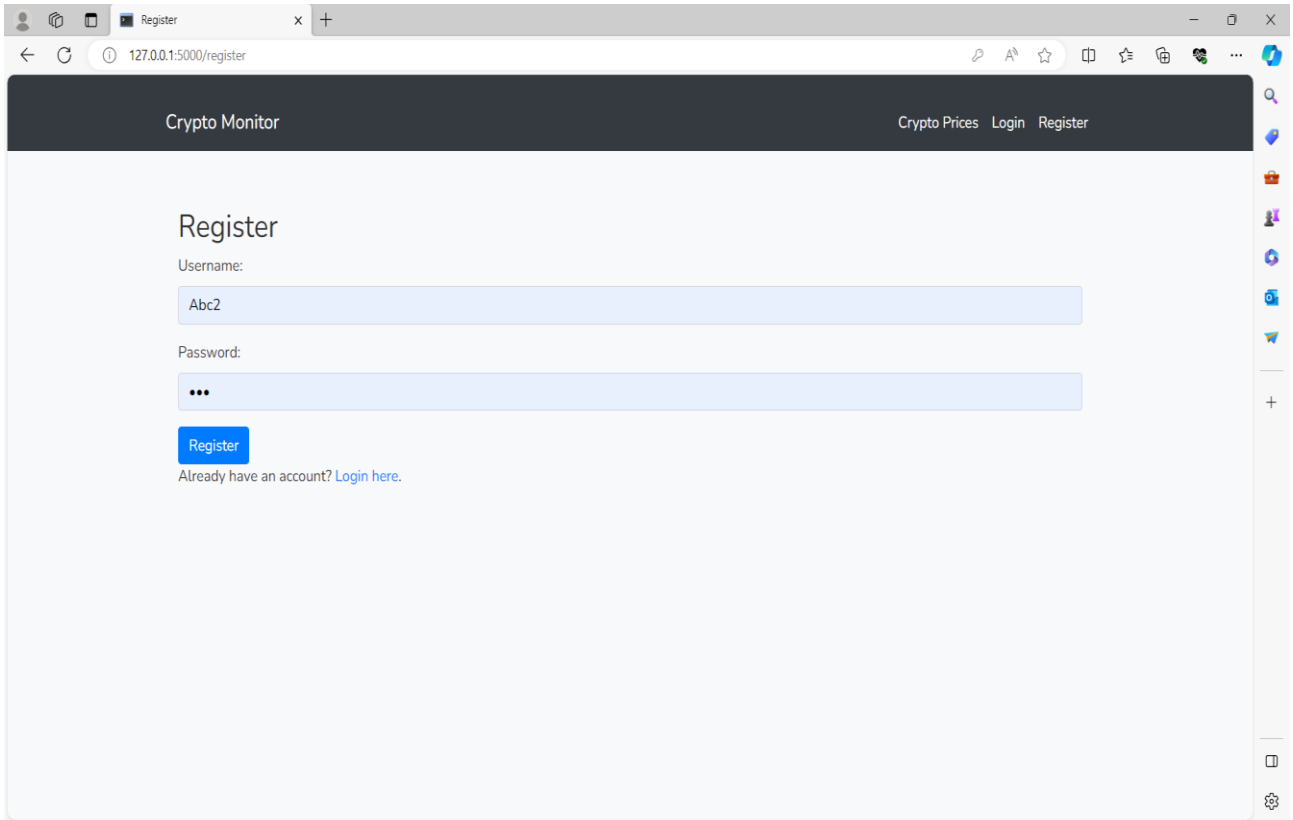
- User Interface Design:

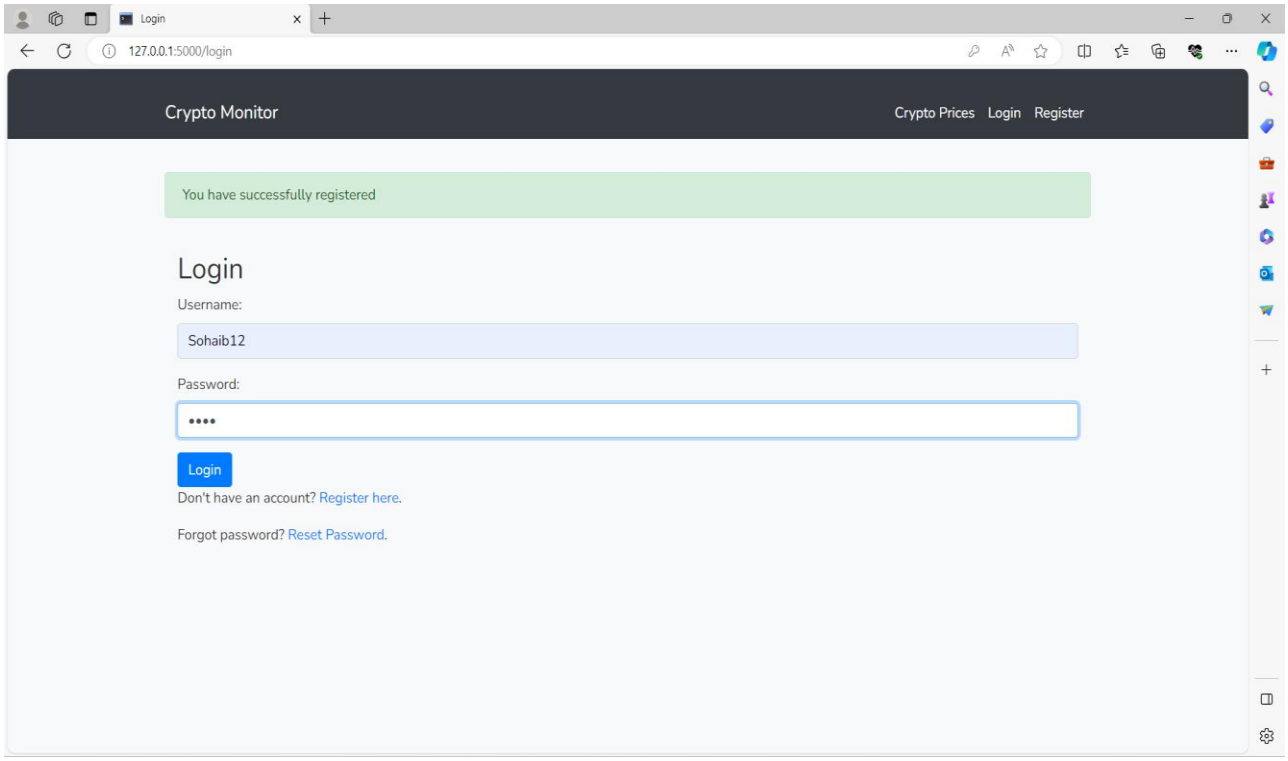

Crypto Monitor
Crypto Prices
Dashboard
Logout
Dashboard
Welcome, Shehar97!
Crypto News
Latest Updates in Cryptocurrency
Catch up with the latest trends, price movements, and expert forecasts in the cryptocurrency world.
Read More
Crypto News
Latest Updates in Cryptocurrency
Catch up with the latest trends, price movements, and expert forecasts in the cryptocurrency world.

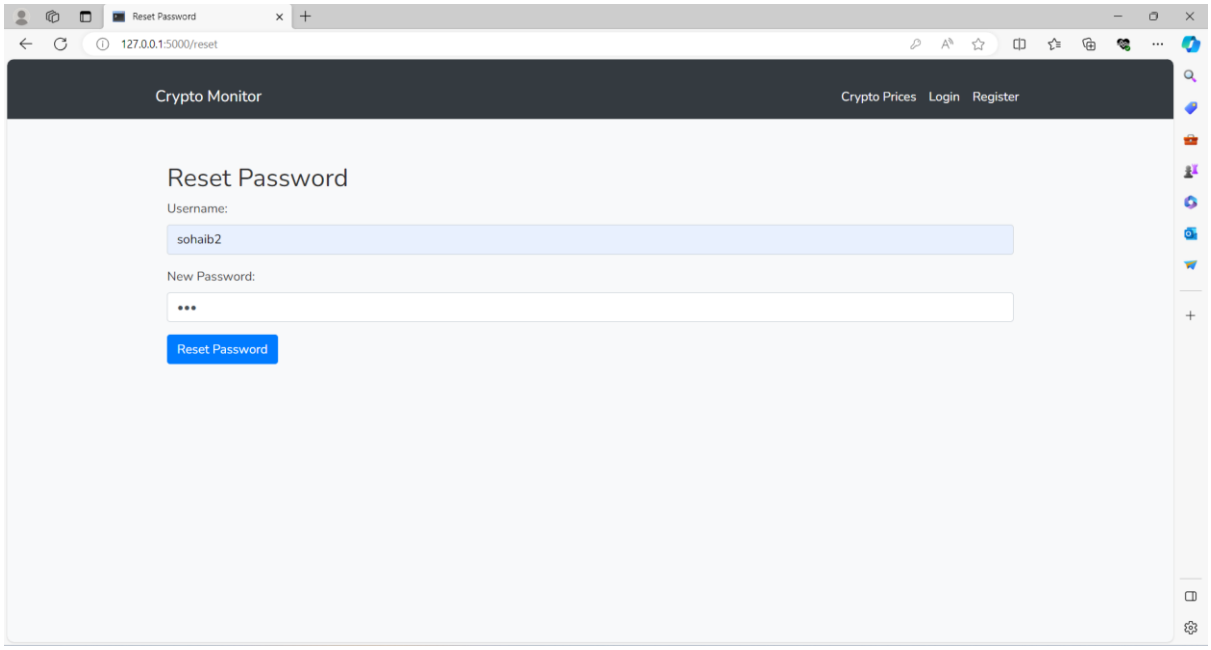
Crypto News
Latest Updates in Cryptocurrency
Catch up with the latest trends, price movements, and expert forecasts in the cryptocurrency world.
Read More
Crypto News
Latest Updates in Cryptocurrency
Catch up with the latest trends, price movements, and expert forecasts in the cryptocurrency world.
Read More


Main Page
127.0.0.1:5000
Crypto Monitor
Crypto Prices Login Register
Cryptocurrency Prices
GO TO DASHBOARD
Cryptocurrency Symbol (Like BTC,ETH):
BTC
Select Exchanges:
binance
kraken
coinbasepro
bitfinex
FETCH PRICES
BTC
Exchange
Price
binance
$43003.79
kraken
$43002.70
coinbasepro
$43002.08

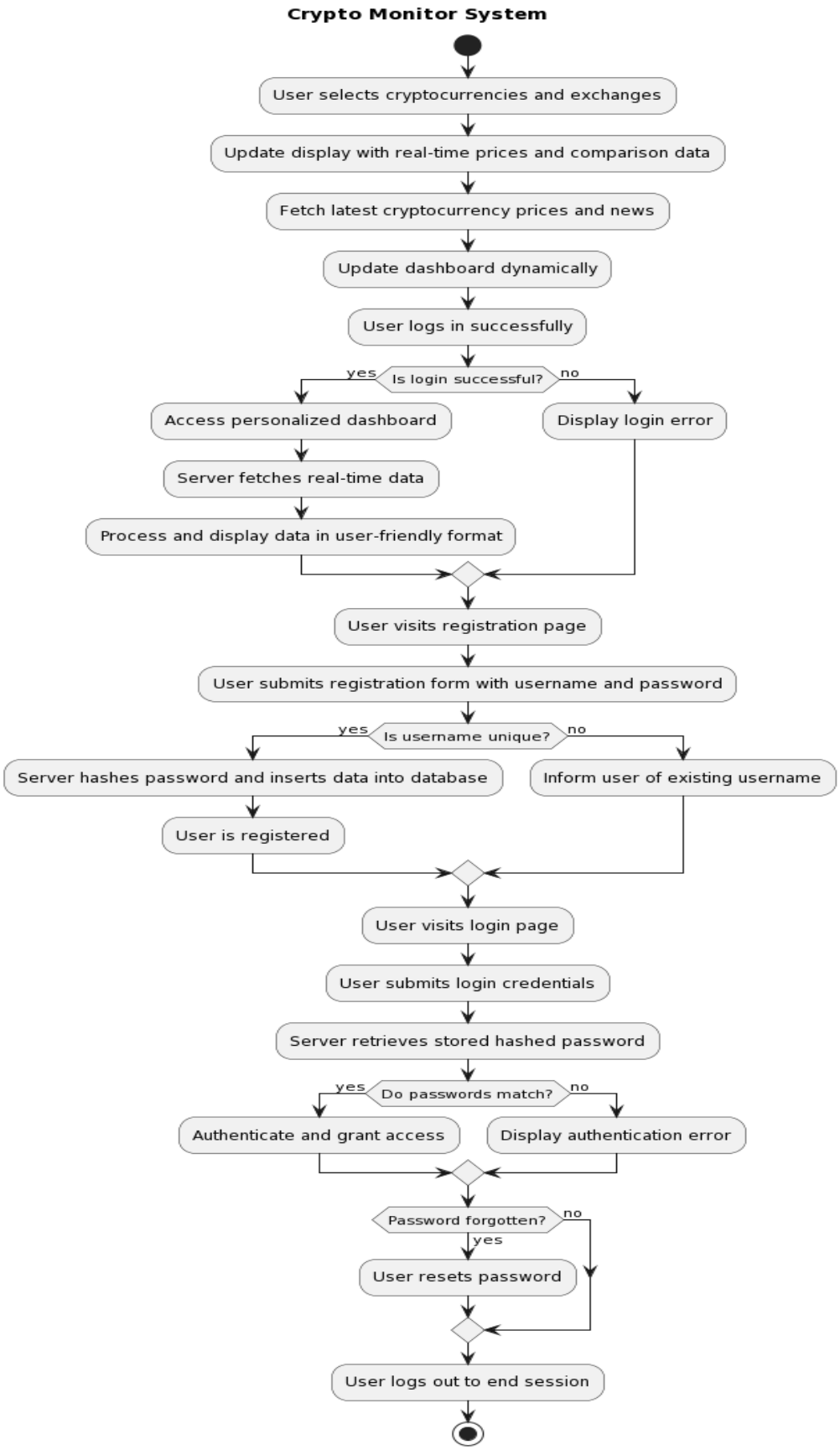

binance
$43003.79
kraken
$43002.70
coinbasepro
$43002.08
bitfinex
$43009.00


Register
127.0.0.1:5000/register
Crypto Monitor
Crypto Prices Login Register
Register
Username:
Abc2
Password:
Register
Already have an account? Login here.





- Activity Diagram:


Crypto Monitor System
User selects cryptocurrencies and exchanges
Update display with real-time prices and comparison data
Fetch latest cryptocurrency prices and news
Update dashboard dynamically
User logs in successfully
Is login successful?
yes
no
Access personalized dashboard
Display login error
Server fetches real-time data
Process and display data in user-friendly format
User visits registration page
User submits registration form with username and password
Is username unique?
yes
no
Server hashes password and inserts data into database
Inform user of existing username
User is registered
User visits login page
User submits login credentials
Server retrieves stored hashed password
Do passwords match?
yes
no
Authenticate and grant access
Display authentication error
Password forgotten?
no
yes
User resets password
User logs out to end session

Problem solvable methods:

- Frontend: HTML, CSS (with Bootstrap), and JavaScript. This stack provides a user-friendly interface and ensures cross-platform compatibility.
- Backend: Flask (a lightweight Python web framework), handling server-side logic, routing, and interactions with the database.
- Database: SQLite, a lightweight database solution, suitable for small to medium-scale applications and used for storing user data and session management.
- Password Security: Bcrypt for hashing and securing passwords, enhancing user security.
- Data Fetching and Processing: The **CCXT** library for fetching real-time cryptocurrency data from various exchanges.

Stakeholder interests:

1. Experienced Cryptocurrency Traders:

Interest in the Solution: They require a platform for quick, reliable access to real-time market data across various exchanges.

How our system meets needs: Our system Crypto Market provides a comprehensive dashboard to all traders with live price updates and exchange comparisons, enabling these traders to make informed, timely decisions.

2. Newcomers to Cryptocurrency Trading:

Interest in the Solution: New traders often struggle with the complexity of existing platforms and the volatility of the market.

How our system meets needs: Our application offers a simplified, user-friendly interface with clear, concise data presentation and educational resources to help them understand market trends.

3. Casual Investors:

Interest in the Solution: Casual investors need a straightforward way to monitor their investments without the detailed analysis required by day traders.

How our system meet needs The Crypto Market includes a personalized watchlist feature and notifications, allowing these users to stay updated on specific cryptocurrencies of interest with minimal effort.

4. Cryptocurrency Analysts:

Interest in the Solution: This group seeks in-depth market analysis and trends for research and knowledge purposes.

Meeting Their Needs: Our platform offers advanced data visualization tools, historical price trends, and market news, catering to their need for detailed market insights.

5. Solution Features:

- Coin Selection: The system allows users to select and view prices of various cryptocurrencies (such as Bitcoin, Ethereum, etc.). [4]
- Exchange Comparison: A core functionality where the system compares cryptocurrency prices across major exchanges. This feature provides a consolidated view of the market, helping users to make informative decisions.
- User Registration and Login: This feature can register Users. When Account is created user can easily login to the system.

Solution Limitations:

- Focus on Major Exchanges: The platform will primarily track and compare prices from well-known and widely used cryptocurrency exchanges. This focus ensures reliability and relevance of the data but may exclude smaller or niche exchanges.
- Basic User Management: The system will include fundamental user management features such as registration, login, and password reset.
- Scalability: While SQLite is sufficient for small to medium-scale applications, it might not scale well for very large user bases or extremely high transaction volumes.
- Exchange Coverage: The system primarily focuses on major exchanges, potentially overlooking emerging or niche exchanges which might offer unique opportunities.
- Data Latency: Depending on the frequency of data updates from the CCXT library and exchanges, there might be a minimal lag in price information, which is crucial for day traders.

Requirements Analysis:

A. Functional Requirements:

- Coin Selection:

Requirement ID	Requirement Description
FR-01	The system should allow users to select cryptocurrencies.
FR-02	Users should be able to add or remove cryptocurrencies from their watchlist.
FR-03	The system must provide a search feature to find specific cryptocurrencies.
FR-04	The system should display real-time price updates for selected cryptocurrencies.

- Exchange Comparison:

Requirement ID	Requirement Description
FR-01	The system should compare prices of selected cryptocurrencies across major exchanges.

FR-02	Users should have the ability to select specific exchanges for comparison.
FR-03	The system should provide historical price data for trend analysis.

- User Registration:

Requirement ID	Requirement Description
FR-01	The system should provide a simple registration form for new users.
FR-02	The system should validate username during registration.
FR-03	The system should ensure strong password requirements for account security.

- User Login:

Requirement ID	Requirement Description
FR-01	The system should offer a secure login mechanism with username and password.
FR-02	The system should provide a 'forgot password' feature for password recovery.
FR-03	The system should allow multiple attempts for login.
FR-04	The system should offer seamless integration with the user dashboard upon successful login.

B. Non-Functional Requirements:

- Performance:

Requirement ID	Requirement Description
NFR-01	The system should ensure a responsive UI design for optimal user experience across various devices.
NFR-02	The system should facilitate quick data retrieval for real-time display of cryptocurrency prices.
NFR-03	The system should optimize backend processes for efficient data handling and minimal latency.

- Security:

Requirement ID	Requirement Description
NFR-01	The system should employ robust data encryption methods to safeguard user data both in transit and at rest.
NFR-02	The system should implement secure authentication protocols to prevent unauthorized access.

NFR-03	The system should include regular security audits and updates to maintain the integrity and safety of user data.

- Usability:

Requirement ID	Requirement Description
NFR-01	The system should offer an intuitive user interface that is easy to navigate for users of all technical backgrounds.
NFR-02	The system should provide clear instructions and guidance for new users.
NFR-03	The system should ensure consistency in design and functionality across all modules for a cohesive user experience.

Design

System Overview:

- Frontend: HTML, CSS (with Bootstrap), and JavaScript. This stack provides a user-friendly interface and ensures cross-platform compatibility.
- Backend: Flask (a lightweight Python web framework), handling server-side logic, routing, and interactions with the database.
- Database: SQLite, a lightweight database solution, suitable for small to medium-scale applications and used for storing user data and session management.
- Password Security: Bcrypt for hashing and securing passwords, enhancing user security.
- Data Fetching and Processing: The CCXT library for fetching real-time cryptocurrency data from various exchanges.

Components interactions:

Component	Technologies	Functionality	Interactions
User Interface	HTML, CSS, JavaScript	User interactions, data presentation, responsive design	Sends user requests to the server; displays data from server

Server-Side Processing	Flask	Request processing, business logic, routing	Processes user requests; interacts with database and external APIs
Database	SQLite	User data storage, watchlist settings, session management	Stores and retrieves user-related and session data
Data Fetching	CCXT Library	Real-time cryptocurrency data fetching from exchanges	Fetches data requested by server; returns data to server
Security	bcrypt	Password hashing and security	Used by server to secure user passwords

Algorithms:

1) User Registration Algorithm:

```
Algorithm RegisterUser(username, email, password):
  If username is empty or email is empty or password is empty:
    Display "All fields are required"
    Stop

  If username exists in database:
    Display "Username already exists"
    Stop

  If email format is not valid:
    Display "Invalid email format"
    Stop

  hashedPassword = hash(password using bcrypt)
  Store username, email, hashedPassword in database
  Send verification email to user
  Display "Registration successful, please verify your email"
```

2) User Login Algorithm:

```
Algorithm LoginUser(username, password):
  If username or password is empty:
    Display "Username and password are required"
    Stop
```

```
  userRecord = fetch user data from database using username
  If userRecord is not found:
    Display "Invalid username or password"
    Stop

  If not checkPassword(password, userRecord.hashedPassword):
    Display "Invalid username or password"
    Stop

  Create user session
  Redirect to dashboard
```

3) Fetching Cryptocurrency Data:

```
Algorithm FetchCryptoData():
  exchanges = get list of supported exchanges
  cryptoData = empty list

  For each exchange in exchanges:
    try:
      data = fetch data from exchange using CCXT
      add data to cryptoData
    except connection or API error:
      continue to next exchange

  If cryptoData is empty:
    Display "Unable to fetch data"
  Else:
    Display cryptoData on dashboard
```

Problems Decomposition:

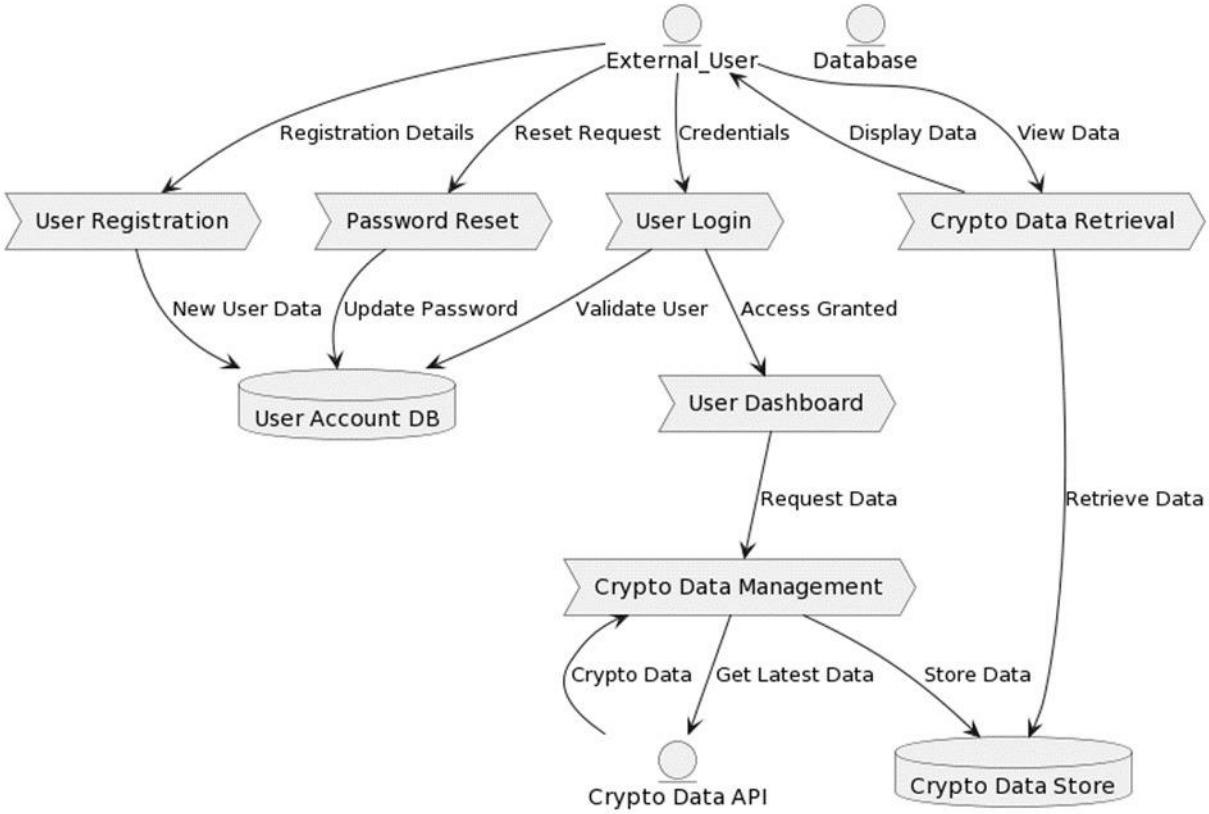
- Need for Real-Time Cryptocurrency Data: The volatility and rapid price changes in the cryptocurrency market make it essential for users to have access to the latest data.
- Comparison Across Various Exchanges: Users need a way to compare cryptocurrency prices across different exchanges to make informed trading decisions.
- User-Friendly Interface for Cryptocurrency Trading: The complexity of the cryptocurrency market can be daunting, especially for new users or those not technically savvy.
- Secure and Efficient User Management: Managing user accounts securely and efficiently is crucial, particularly in applications dealing with financial data.

Solutions Structure:

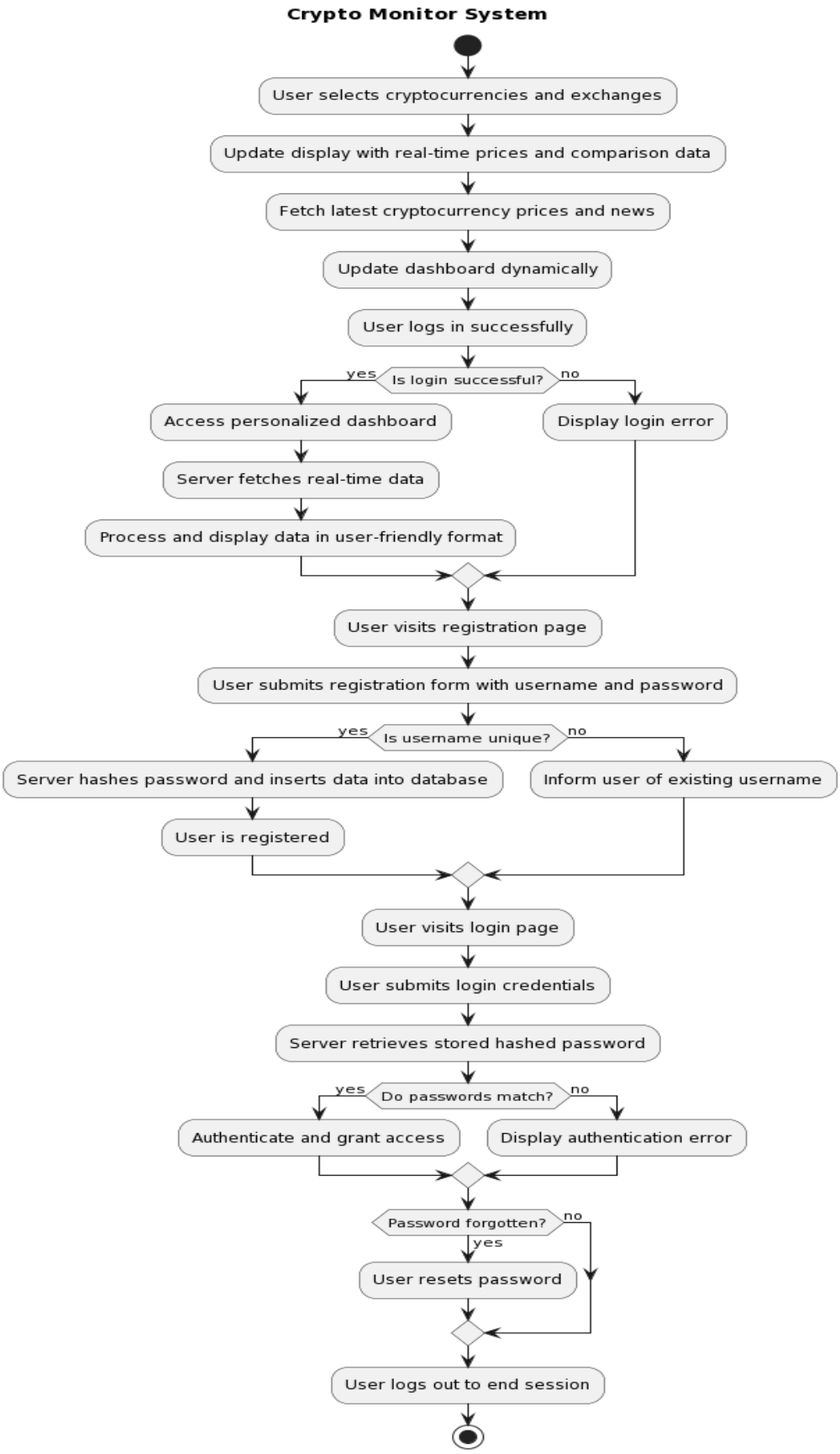
- Modelling the Problem and Breaking It Down: Your system addresses these problems by employing a structured approach, using the Software Development Life Cycle (SDLC) and agile methodologies. This helps in breaking down the complex problem into manageable parts.
- Real-Time Data Fetching and Display: Using the CCXT library for fetching real-time data, your system provides up-to-date cryptocurrency prices and market trends, solving the issue of needing current market data.
- Exchange Comparison Feature: The system compares prices from different exchanges, providing a consolidated market view and helping users in decision-making.
- User Interface Design: The focus on a user-friendly interface with features like charts and comparison tools caters to the need for an accessible platform for all users.
- Permanent Data Storage with SQLite: Choosing SQLite addresses the need for reliable and permanent data storage, suitable for small to medium-scale applications.
- Use of Flask and bcrypt for User Management: Flask facilitates user management, while bcrypt provides secure password encryption, addressing the need for secure user account management.
- Test Data for Iterative Development: The selection of test data, particularly for user authentication, ensures the system is robust and functions as intended.

Modelling the problem:

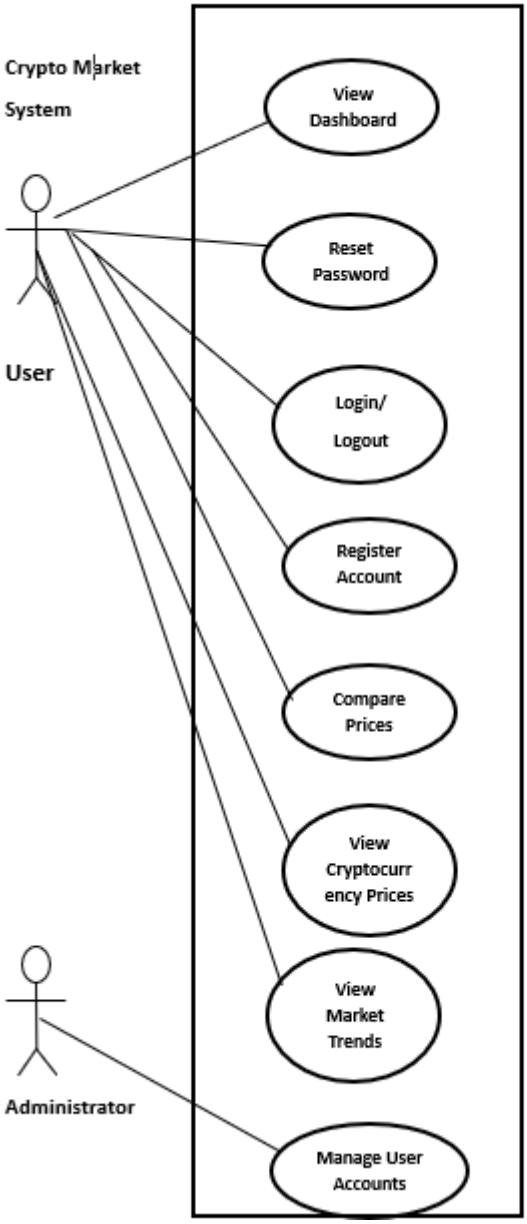
- DFD:



- Activity Diagram:



- Use Case Diagram:



Solution using algorithms:

- Flask: Flask is a python web framework which provides facility for user to use web on python language. Its use is very easy and simple. Its development is rapid as compared to other frameworks.
- SQLite: It is a Database mainly use for storing the data, in our application we use for storing the data of user. It is also called a lightweight database.
- bcrypt: It is used in mostly for encryption means as a security purpose. Its use in passwords security and it serves as a strong capability.
- CCXT Library: It is the library used for the fetching of real-time data. Its coverage is very vast for cryptocurrency exchanges.

Usability features:

- Dashboard of our system:

We CCXT library for fetching the data, which are mainly display on the screen. Here we show news of related to cryptocurrency on the dashboard. This dashboard comes after the login page.

- Real-time price update:
  The system should display real-time price updates for selected cryptocurrencies.
- Login system:

Develop a user-friendly platform with robust functionalities, including user registration, login systems and personalized user experiences.

Functions:

1. init_db(): This function initializes the SQLite3 database by creating a table named users if it does not already exist. This table is used to store user information such as their username and hashed password.

```
def init_db():
    with sqlite3.connect('users.db') as db:
        cursor = db.cursor()
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS users (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                username TEXT UNIQUE NOT NULL,
                hashed_password TEXT NOT NULL
            );
        ''')
        db.commit()
```

2. register (): This function handles the user registration process. It receives data from a registration form, including the username and password. It hashes the password using bcrypt before storing it in the database. It also checks if the username already exists in the database to prevent duplicate registrations.

```python
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password'].encode('utf-8')
        hashed = bcrypt.hashpw(password, bcrypt.gensalt())

        try:
            with sqlite3.connect('users.db') as db:
                cursor = db.cursor()
                cursor.execute('INSERT INTO users (username, hashed_password) VALUES (?, ?)', (username, hashed))
                db.commit()
            flash('You have successfully registered', 'success')
            return redirect(url_for('login'))
        except sqlite3.IntegrityError:
            flash('Username already exists', 'danger')
    return render_template('register.html')
```

3. login (): This function handles user login. It verifies the username and password provided by the user against the data stored in the database. If the credentials are correct, it sets session variables to indicate that the user is logged in.

```python
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password'].encode('utf-8')

        with sqlite3.connect('users.db') as db:
            cursor = db.cursor()
            cursor.execute('SELECT id, username, hashed_password FROM users WHERE username = ?', (username,))
            user = cursor.fetchone()

            if user and bcrypt.checkpw(password, user[2]):
                session['loggedin'] = True
                session['user_id'] = user[0]
                session['username'] = user[1]
                return redirect(url_for('dashboard'))
            else:
                flash('Incorrect username/password', 'danger')
    return render_template('login.html')
```

4. logout (): This function logs the user out by removing their session variables.

```python
def logout():
    session.pop('loggedin', None)
    session.pop('user_id', None)
    session.pop('username', None)
    return redirect(url_for('login'))
```

5. dashboard (): This function renders the dashboard template if the user is logged in. Otherwise, it redirects the user to the login page.

```python
def dashboard():
    if 'loggedin' not in session:
        return redirect(url_for('login'))
    return render_template('dashboard.html', username=session['username'])
```

6. reset_password (): This function allows users to reset their passwords. It takes a username and a new password, hashes the new password, and updates the corresponding user's password in the database.

```python
def reset_password():
    if request.method == 'POST':
        username = request.form['username']
        new_password = request.form['new_password'].encode('utf-8')
        hashed = bcrypt.hashpw(new_password, bcrypt.gensalt())

        with sqlite3.connect('users.db') as db:
            cursor = db.cursor()
            try:
                cursor.execute('SELECT id FROM users WHERE username = ?', (username,))
                if cursor.fetchone() is not None:
                    cursor.execute('UPDATE users SET hashed_password = ? WHERE username = ?', (hashed, username))
                    db.commit()
                    flash('Password successfully updated', 'success')
                else:
                    flash('Username not found', 'danger')
            except sqlite3.Error as error:
                flash('Error while updating password', 'danger')

        return redirect(url_for('login'))

    return render_template('reset.html')
```

7. main_page (): This function serves the main page of the application. It handles both GET and POST requests. For POST requests, it retrieves selected cryptocurrency symbols and exchanges from the form, fetches their prices using fetch_crypto_prices(), and renders the main template with the fetched data.

```python
def main_page():
    selected_exchanges = ['binance', 'kraken', 'coinbasepro', 'bitfinex']
    crypto_symbol = 'BTC'

    if request.method == 'POST':
        selected_exchanges = request.form.getlist('exchanges')
        crypto_symbols_input = request.form.get('crypto_symbol', default='')
        crypto_symbols = [symbol.strip().upper() for symbol in crypto_symbols_input.split(',')]

        all_prices = {symbol: fetch_crypto_prices(selected_exchanges, symbol) for symbol in crypto_symbols}
    else:
        all_prices = {crypto_symbol: fetch_crypto_prices(selected_exchanges, crypto_symbol)}

    return render_template('main.html', exchanges=selected_exchanges, prices=all_prices, crypto_symbols=CRYPTO_SYMBOL
```

8. fetch_crypto_prices (): This function fetches cryptocurrency prices from different exchanges. It takes a list of exchange IDs and a cryptocurrency symbol as input. It iterates over the specified exchanges, fetches the price for the given symbol, and returns a dictionary containing the prices for each exchange.

```python
def fetch_crypto_prices(exchanges, symbol):
    prices = {}
    for exchange_id in exchanges:
        exchange_class = getattr(ccxt, exchange_id)()
        try:
            exchange_class.load_markets()
            market_pair = symbol + '/USDT'
            price = exchange_class.fetch_ticker(market_pair)['last']
            prices[exchange_id] = f'${price:.2f}'
        except Exception as e:
            print(f"Error fetching price for {symbol} on {exchange_id}: {e}")
            prices[exchange_id] = '---'
    return prices
```

9. prices (): This function serves AJAX requests for fetching cryptocurrency prices. It takes exchange IDs and cryptocurrency symbols as query parameters and returns JSON data containing the prices fetched using fetch_crypto_prices().

```python
def prices():
    selected_exchanges = request.args.getlist('exchanges')
    crypto_symbols = request.args.get('crypto_symbol').split(',')
    all_prices = {}

    for symbol in crypto_symbols:
        symbol_prices = fetch_crypto_prices(selected_exchanges, symbol.strip().upper())
        all_prices[symbol] = symbol_prices

    return jsonify(all_prices)

if __name__ == '__main__':
    init_db()
    app.run(debug=True)
```

Permanent data storage:

- Database: SQLite, a lightweight database solution, suitable for small to medium-scale applications and used for storing user data and session management.
- SQLite: It is a Database mainly used for storing the data, in our application we use for storing the data of user. It is also called a lightweight database.

Variables and Validations:

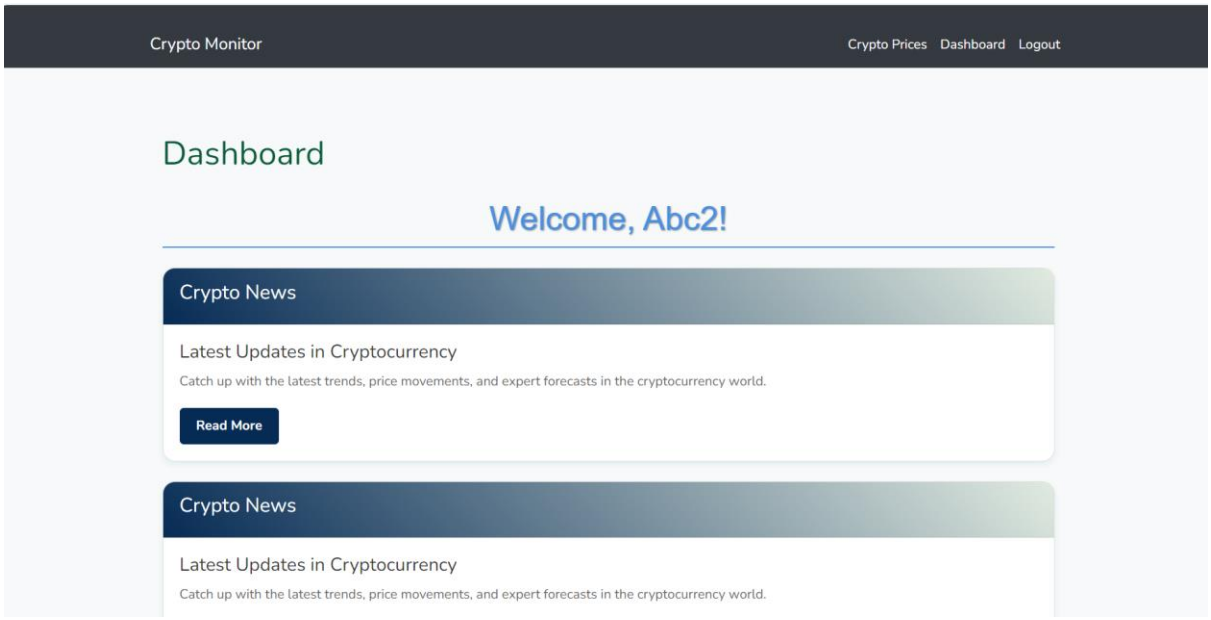
- Session Management: Implemented using Flask's session utilities, it's critical for maintaining state between HTTP requests, enabling a persistent user experience without re-authentication on every page.
- SQLite Database: We use the SQLite database which is also called a lightweight database. We use this for storing the information of users in a database. Based on this, the authentication performs.
- CCXT Library: The CCTX library use for fetching the real time cryptocurrency data. It interacts with various exchange APIs. It plays a base role for fetching the data.
- Crypto Data Structure: A data structure which use for encapsulates the crypto currency data, its major use is to hold prices, which is essential for the display of the dashboard.

Testing:

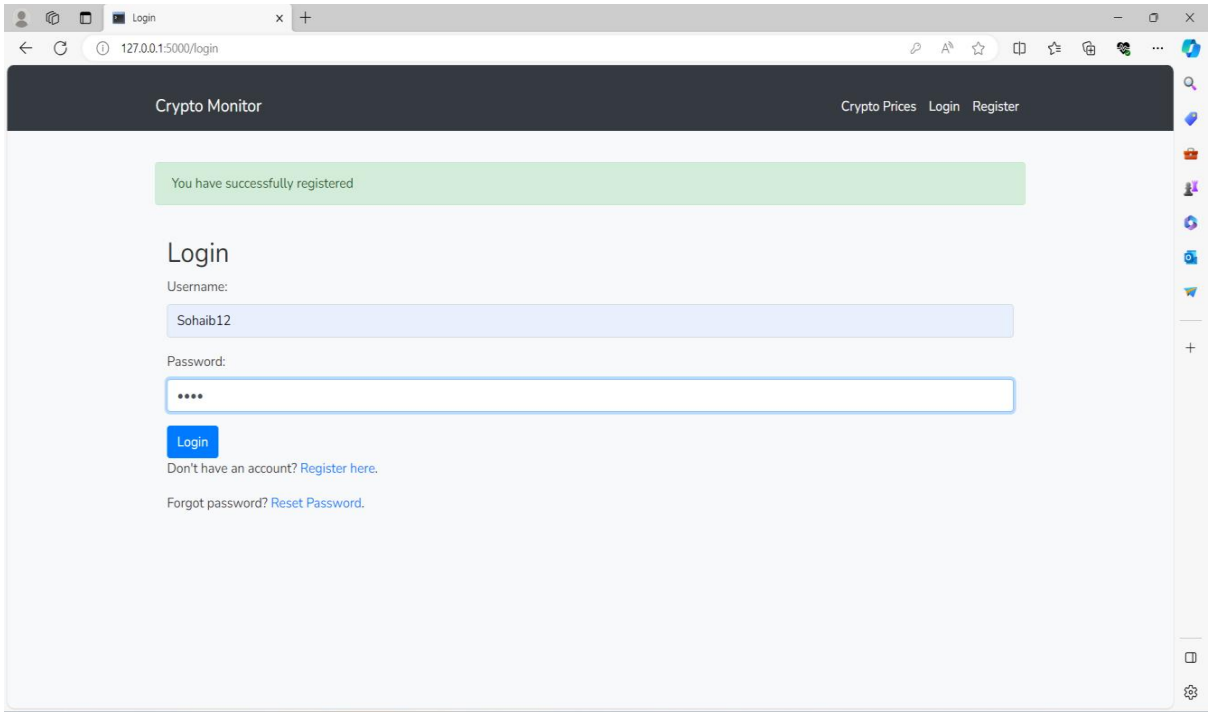
Unit Testing:

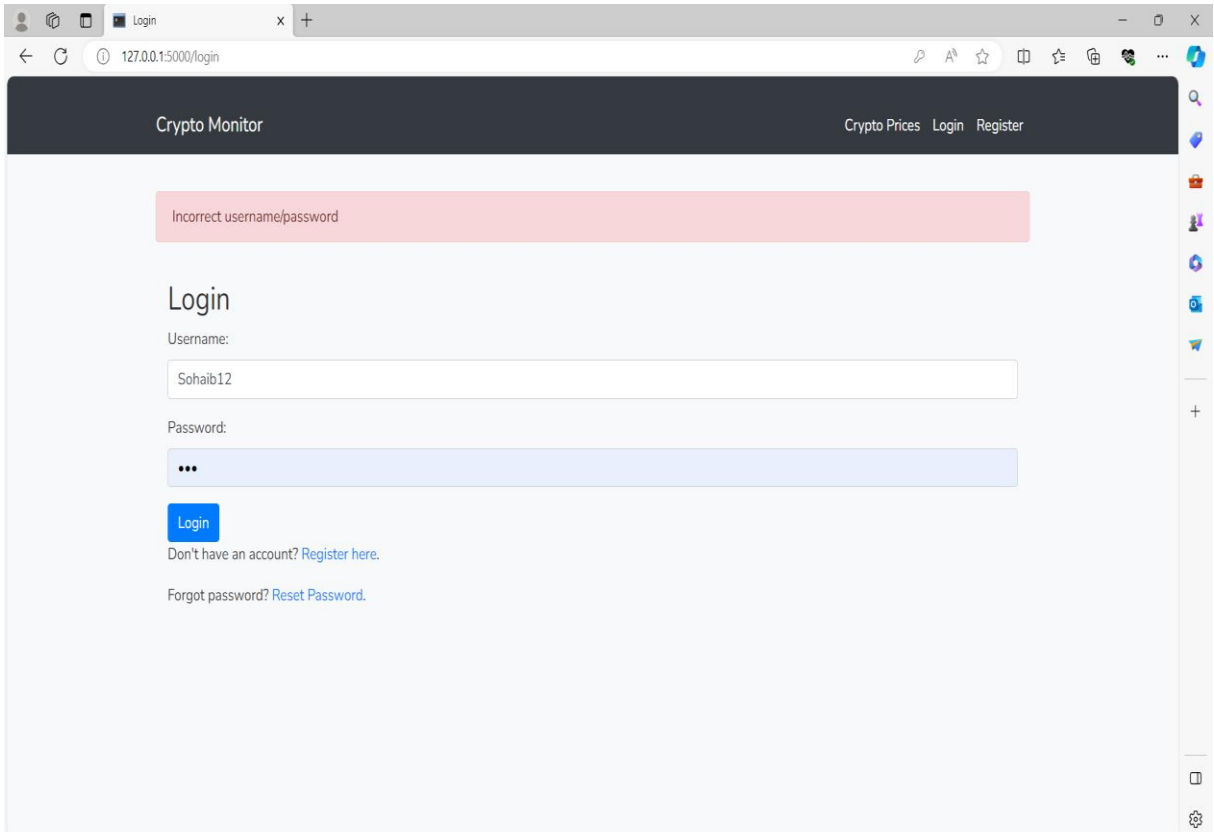
We Perform Unit Testing on these modules which are as given below.

- User Authentication Function:
- Objective: To test the authenticate_user function which verifies if the provided username and password match the stored credentials.
- Test Cases:
- Valid Credentials: Pass a known username and password and expect a successful authentication. The Login successfully and it redirects to dashboard.

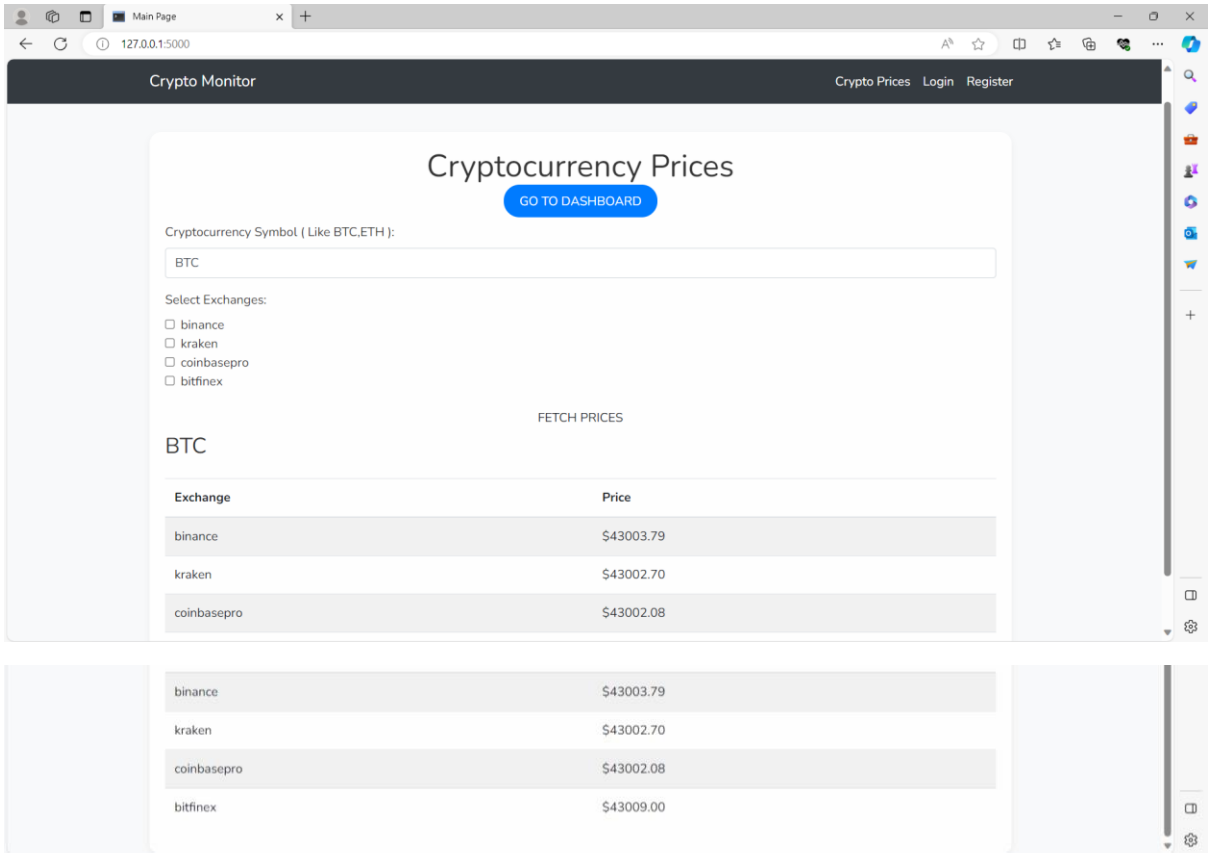


- Invalid Password: I Provided a correct username but an incorrect password and expect authentication to fail which shows our output result is correct as we expected.

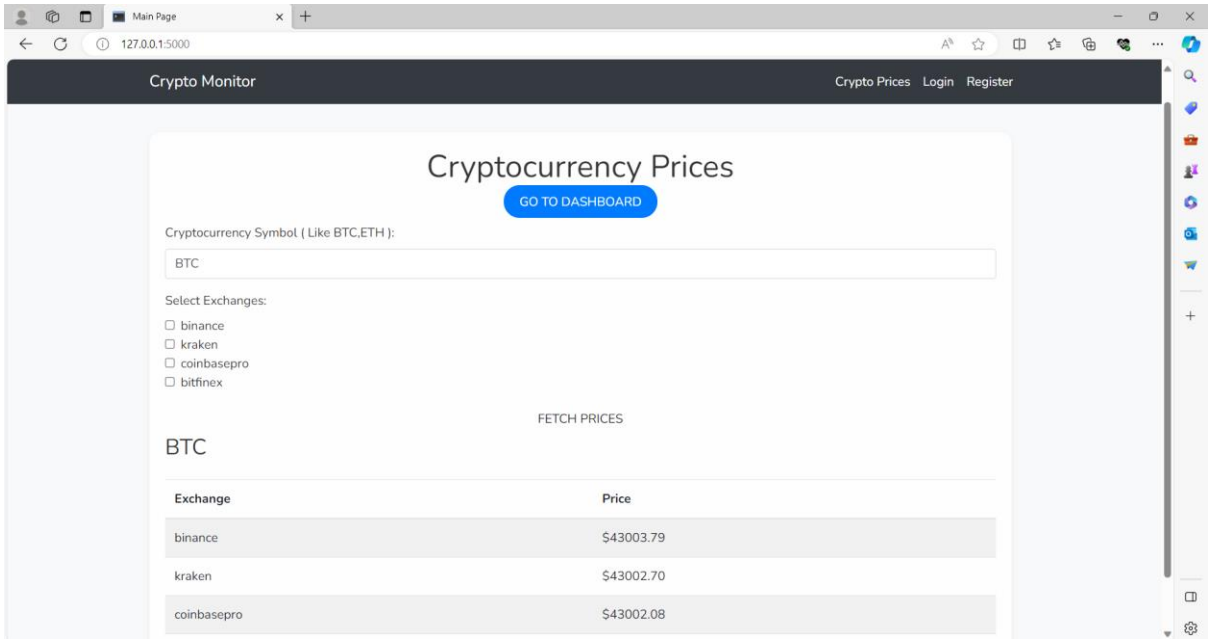




- Cryptocurrency Data Fetching:
- Objective: To test the fetch_crypto_data function that retrieves real-time cryptocurrency data using the CCXT library.
- Test Cases:
- Successful Data Fetch: Provide a valid symbol for a cryptocurrency and expect the function to return the latest data.



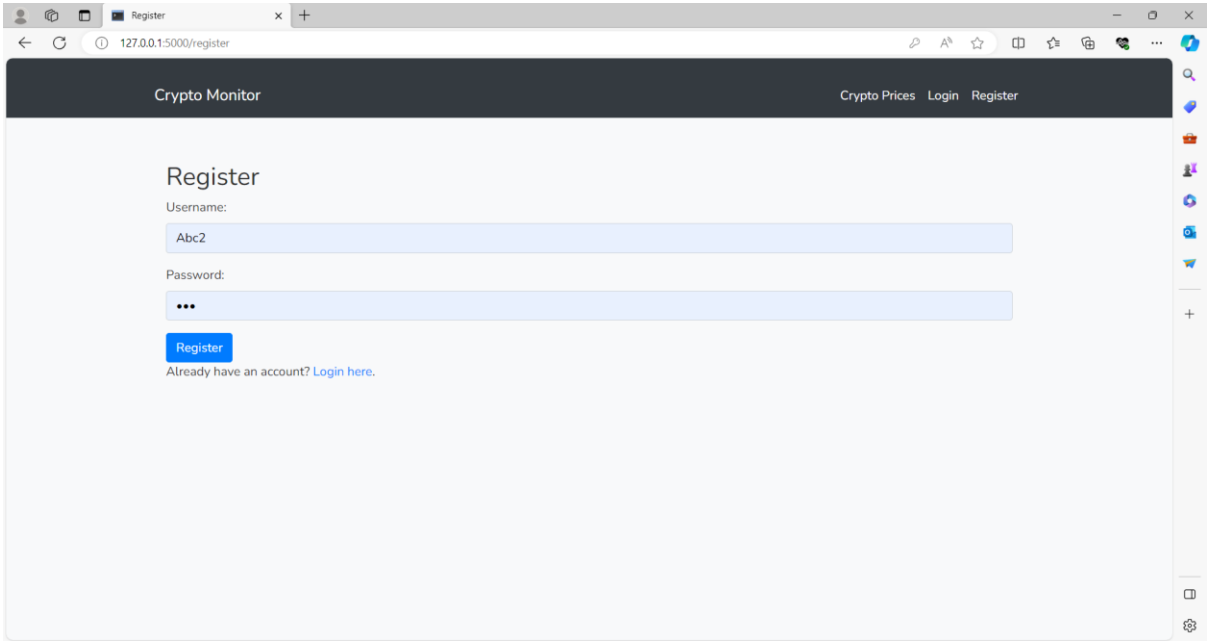
- Invalid Symbol: We Passes an invalid or unsupported symbol (BTCC) and this function to handle this gracefully, either by returning an error or an empty response.



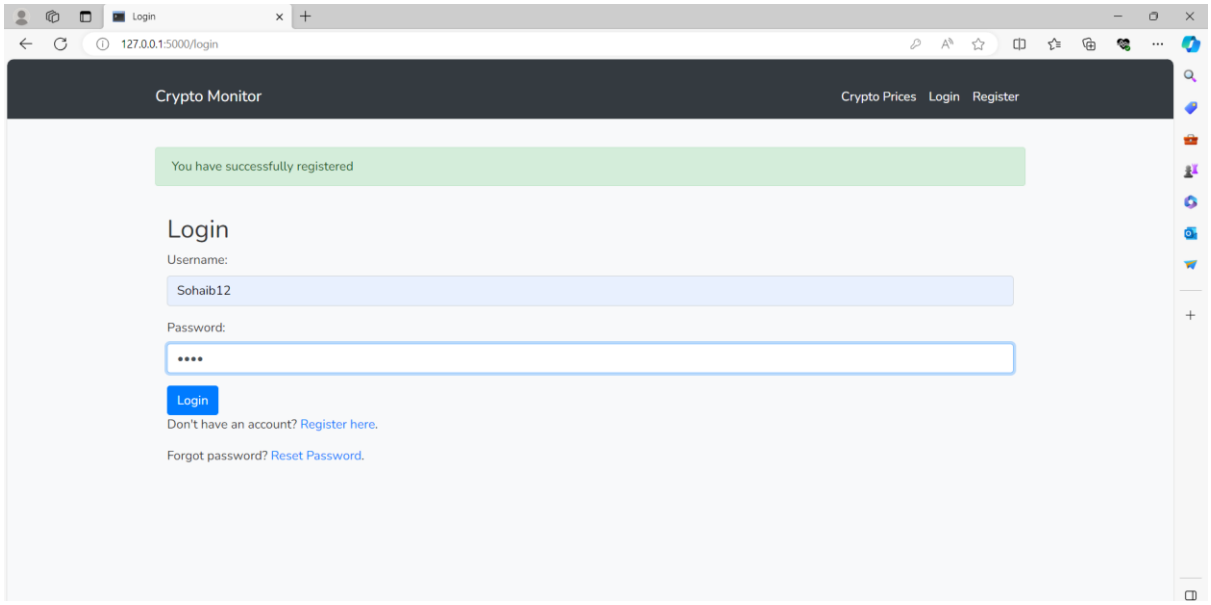
```
Error fetching price for BTCC on binance: binance GET https://api.binance.com/api/v3/exchangeInfo
Error fetching price for BTCC on kraken: kraken GET https://api.kraken.com/0/public/Assets
Error fetching price for BTCC on coinbasepro: coinbasepro does not have market symbol BTCC/USDT
Error fetching price for BTCC on bitfinex: bitfinex does not have market symbol BTCC/USDT
127.0.0.1 - - [02/Feb/2024 23:06:02] "POST / HTTP/1.1" 200 -
```

- User Registration Process:

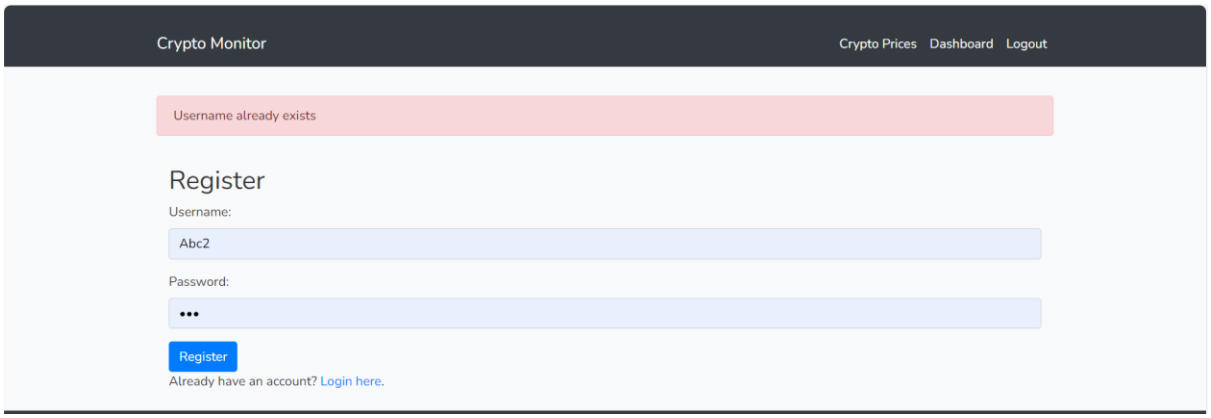
- Objective: To test the register_user function responsible for user registration.
- Test Cases:
- Successful Registration: Attempt to register a new user with a unique username and expect successful registration.



The user Register Successfully, when add a new user.



- Duplicate Username: Try registering a user with an already existing username and expect the process to fail or return an error.



Integration Testing:

Test Scenarios and Results

- User Account:

Tested the entire flow from account creation, data storage in the SQLite database, to user login verification.

Result: Successfully passed, with all user data correctly handled and authenticated.

- Real-Time Data Display:

Tested the retrieval of cryptocurrency data through the CCXT API and its display on the user dashboard.

Result: Passed with minor issues in handling API rate limits, which were subsequently fixed.

- Data Consistency Across Modules:

Ensured that data displayed on the user interface was consistent with the data stored in the backend database.

Result: Successfully passed, with all modules showing consistent data.

- Error Handling and Failover Mechanisms:

Tested the system's response to simulated failures like server downtime and database access issues.

Result: The system correctly handled errors, but improvements were recommended for failover mechanisms.

User Acceptance Testing (UAT):

- User Registration and Login:

Participants registered new accounts and logged in.

Feedback: Users found the registration process straightforward. However, a few users suggested an improved password recovery feature for enhanced usability.

- Real-time Data Display and Exchange Comparison:

Users navigated through the dashboard to check real-time cryptocurrency data and compared prices across different exchanges.

Feedback: The real-time data update was well-received for its accuracy and speed. Some users requested additional filter options for more refined data analysis.

- Usability and Interface:

Participants interacted with various UI elements, testing the intuitiveness and responsiveness of the interface.

Feedback: The interface was praised for its clarity and ease of navigation. A suggestion was made to include a tutorial or help section for new users unfamiliar with cryptocurrency trading.

- Performance Under Load:

The system was tested under high data load conditions.

Feedback: The system maintained a good performance level, but there were minor delays in data refresh during peak load, indicating a need for optimization.

- Security and Data Handling:

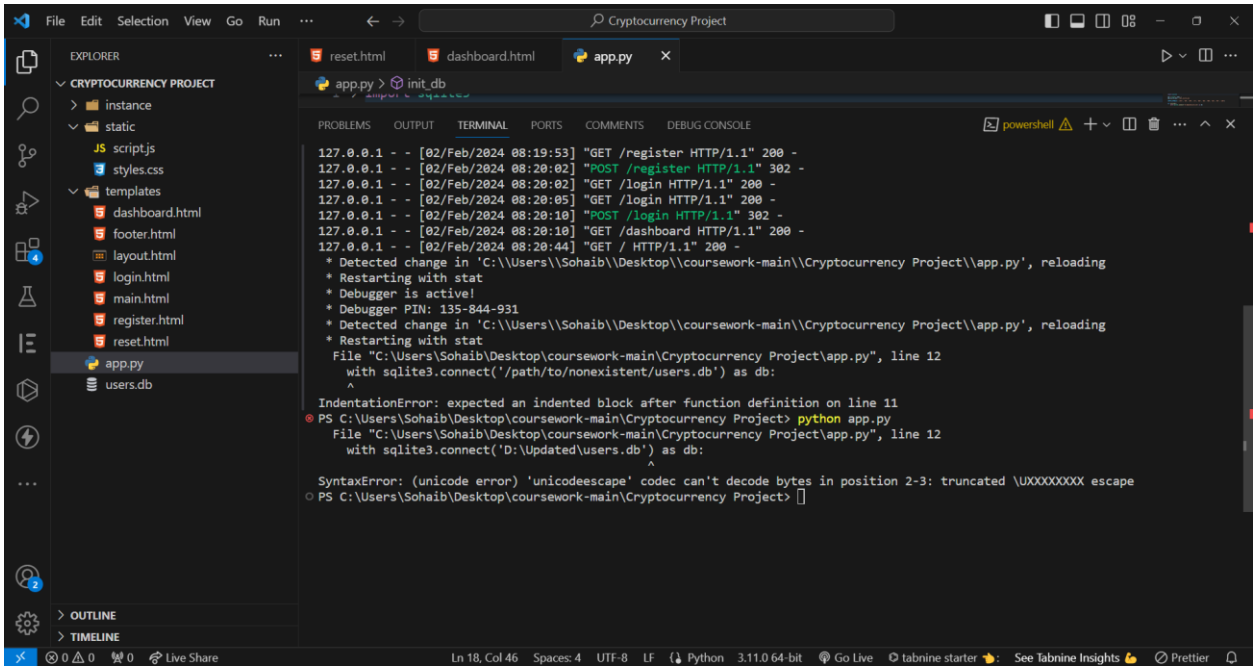
Participants were asked to verify the security measures in place, especially during login and financial data processing.

Feedback: The security measures were deemed robust, giving users confidence in the system's ability to protect sensitive information.

Testing Evidence:

Errors occur during implementation:

1) Database Connection Error:



- **Error Introduced:**

```
def init_db():
    with sqlite3.connect('D:\Updated\users.db') as db:
        cursor = db.cursor()
        cursor.execute('''
```

```
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT UNIQUE NOT NULL,
            hashed_password TEXT NOT NULL
        );

    ''')

    db.commit()
```

- **Code for how I solve this error:**

```
def init_db():
    with sqlite3.connect('D:\Updated\users.db') as db:
        cursor = db.cursor()
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS users (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                username TEXT UNIQUE NOT NULL,
                hashed_password TEXT NOT NULL
            );
```

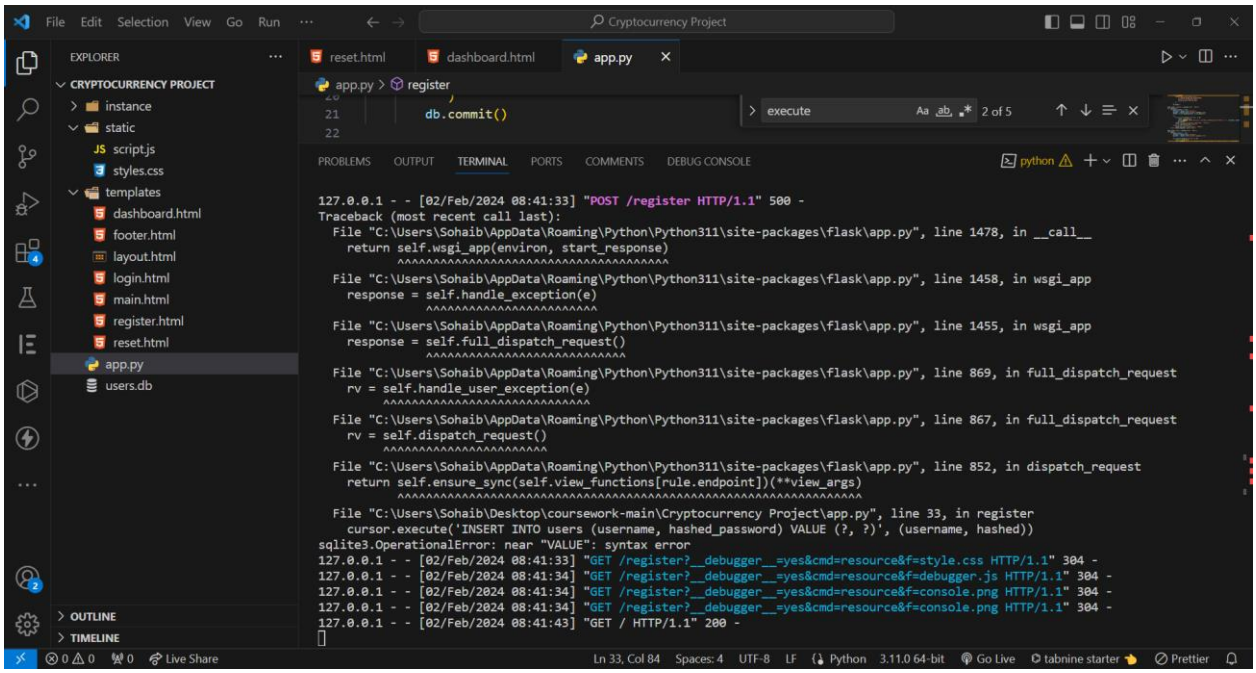
```
''')

db.commit()
```

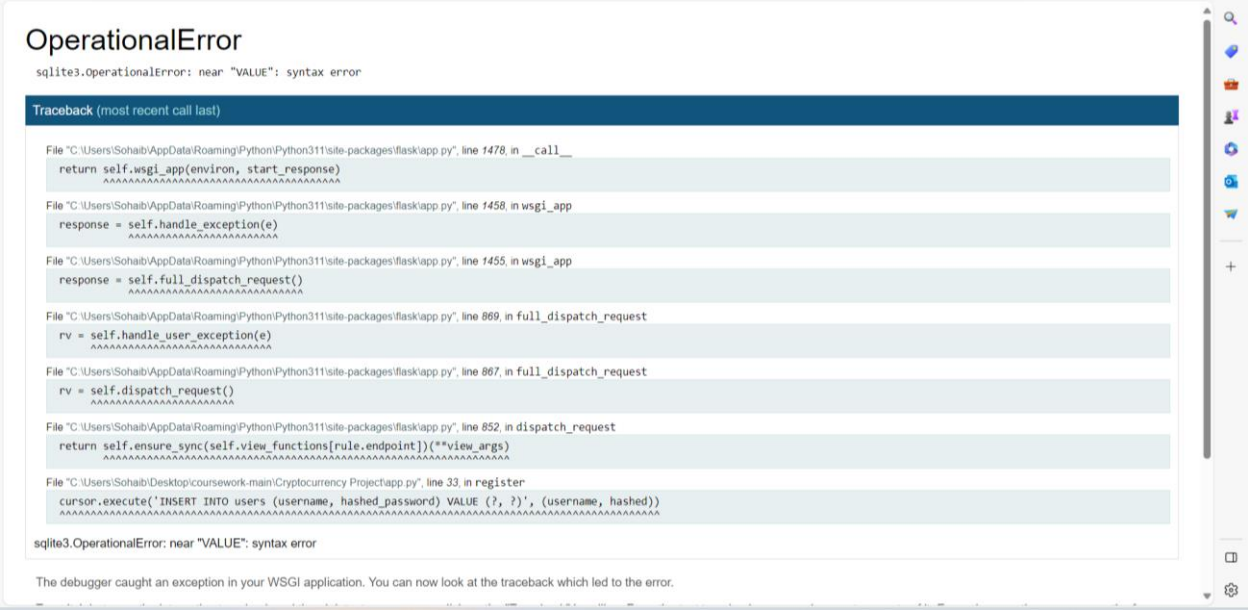
- **My Solution:**

I realized that our error is a database connection which is due to file path for SQlite database file. I placed a correct path of users.db which solves my error.

2) Error Of SQL Query:

Error On VScode:






```python
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password'].encode('utf-8')
        hashed = bcrypt.hashpw(password, bcrypt.gensalt())

        try:
            with sqlite3.connect('users.db') as db:
                cursor = db.cursor()
                cursor.execute('INSERT INTO users (username, hashed_password) VALUE (?, ?)', (username, ha
                db.commit()
            flash('You have successfully registered', 'success')
            return redirect(url_for('login'))
        except sqlite3.IntegrityError:
            flash('Username already exists', 'danger')
    return render_template('register.html')
```

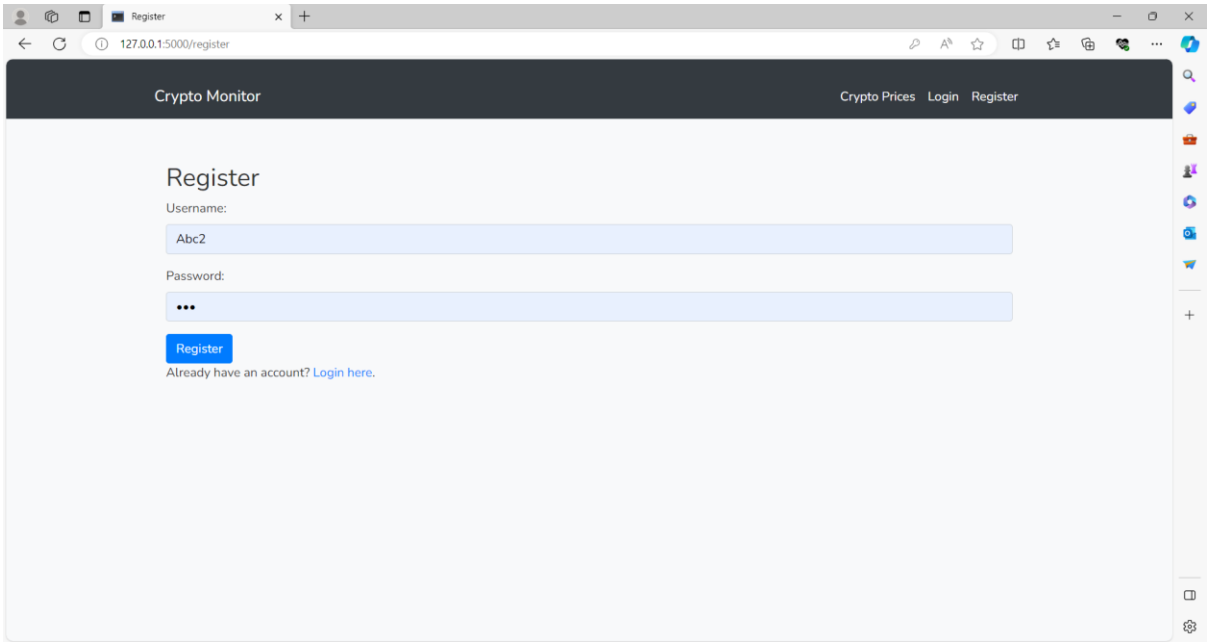
- **How I Solve this Error:**

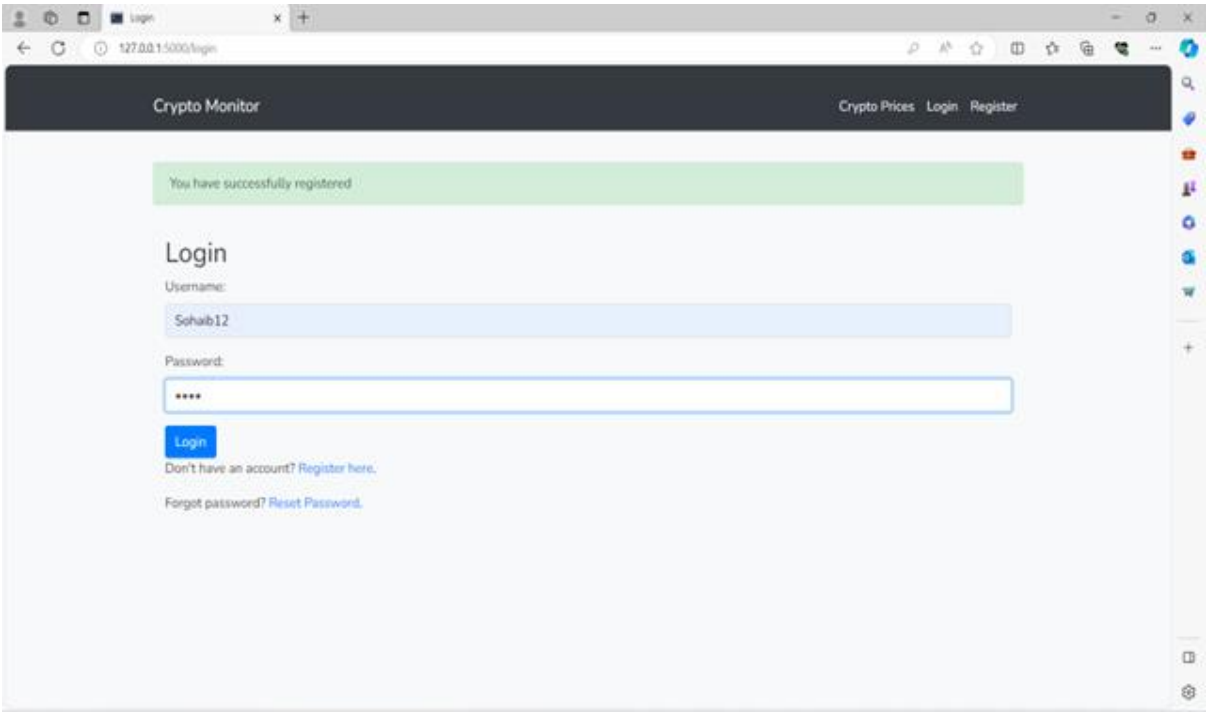
This error is most common which I face, the error is due to the VALUE. This is a typo and most common error of SQL query. By use VALUES instead of VALUE, So this error was easily remove.

```
try:
        with sqlite3.connect('users.db') as db:
            cursor = db.cursor()
            cursor.execute('INSERT INTO users (username, hashed_password) VALUES (?, ?)',
(username, hashed))
            db.commit()
        flash('You have successfully registered', 'success')
        return redirect(url_for('login'))
```

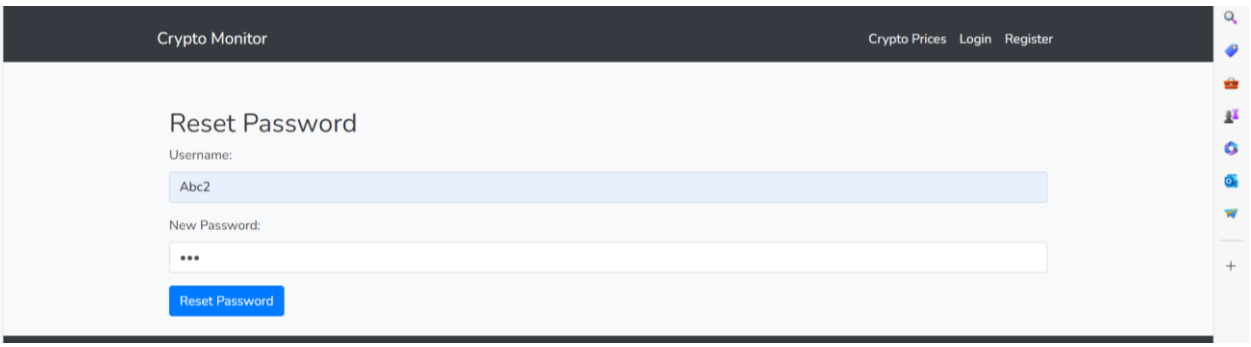
```
        except sqlite3.IntegrityError:
            flash('Username already exists', 'danger')
    return render_template('register.html')
```

After Solving this Error, the registers successfully and record in a database (users.db):

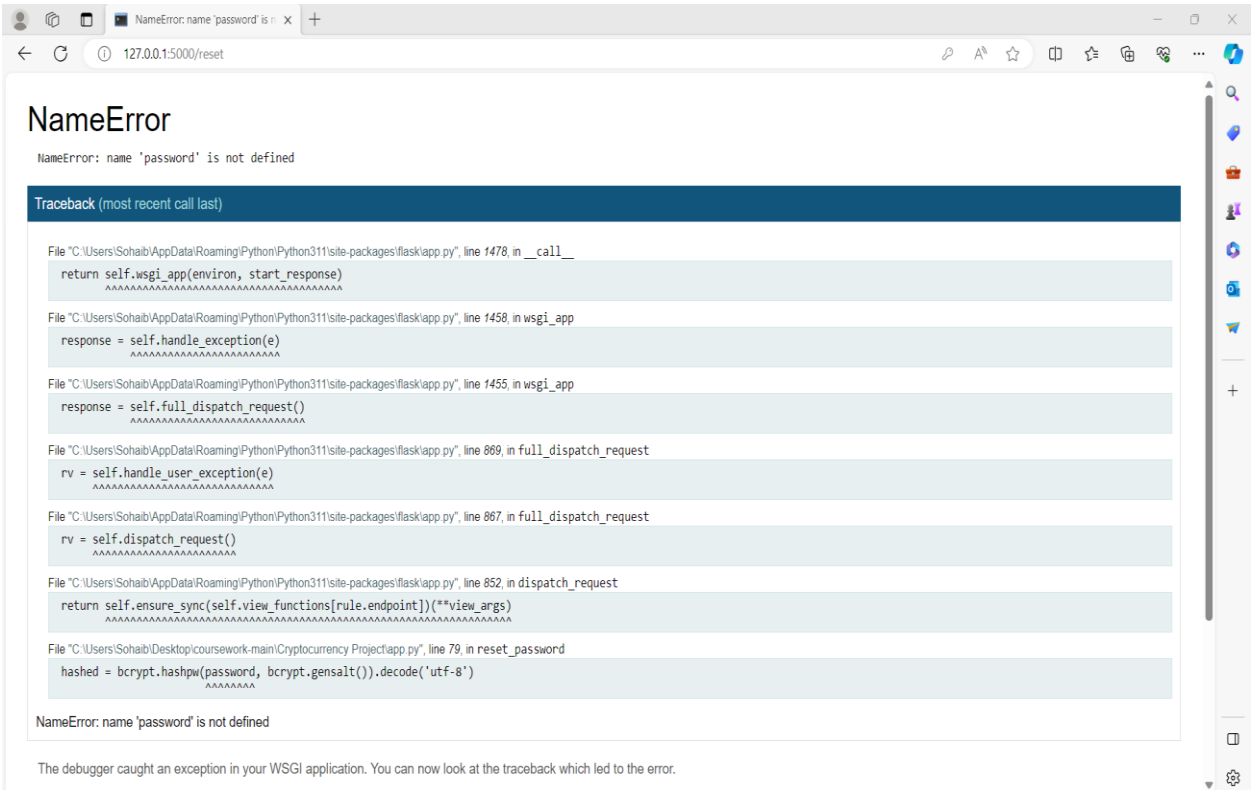


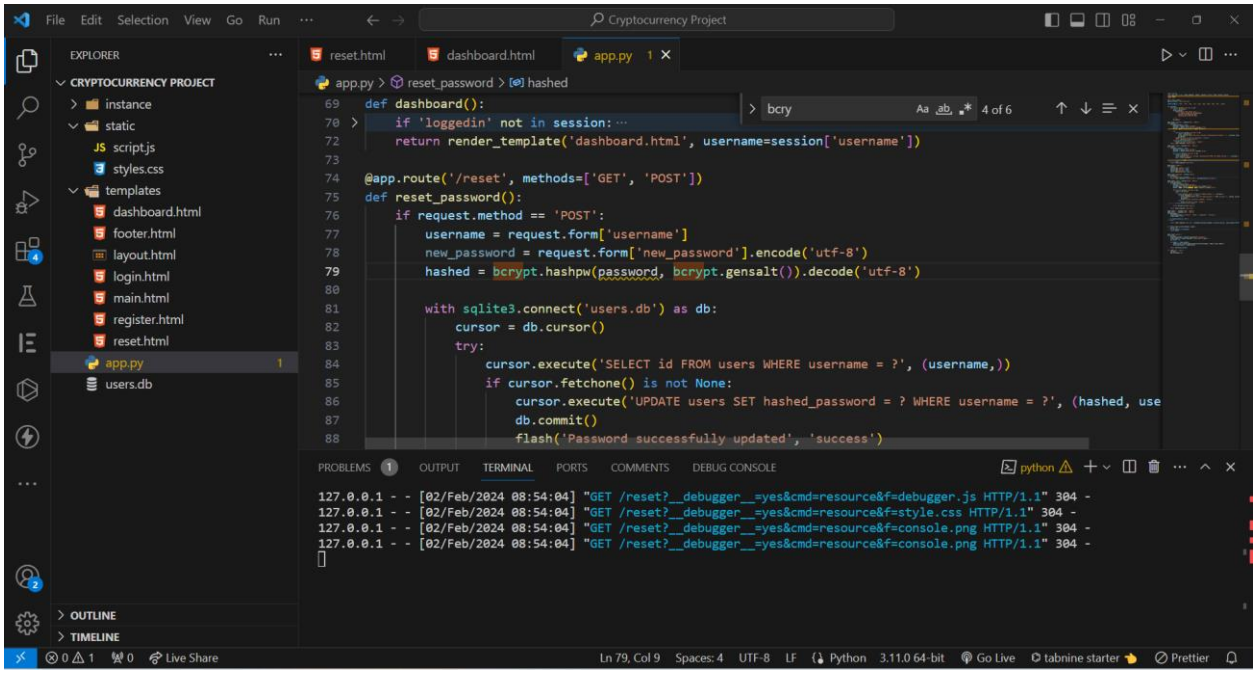


3) Logic Error In Password Hashing:



- **When I click on Reset Password:**



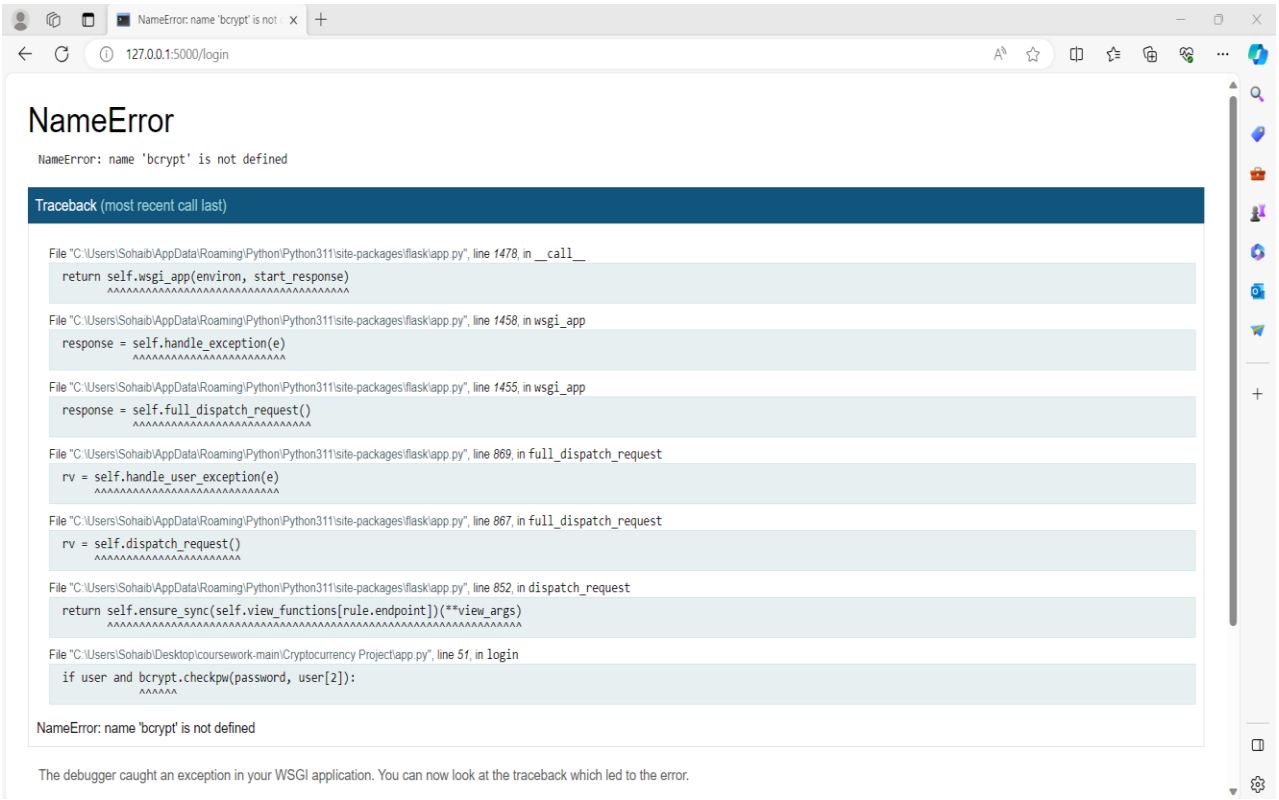


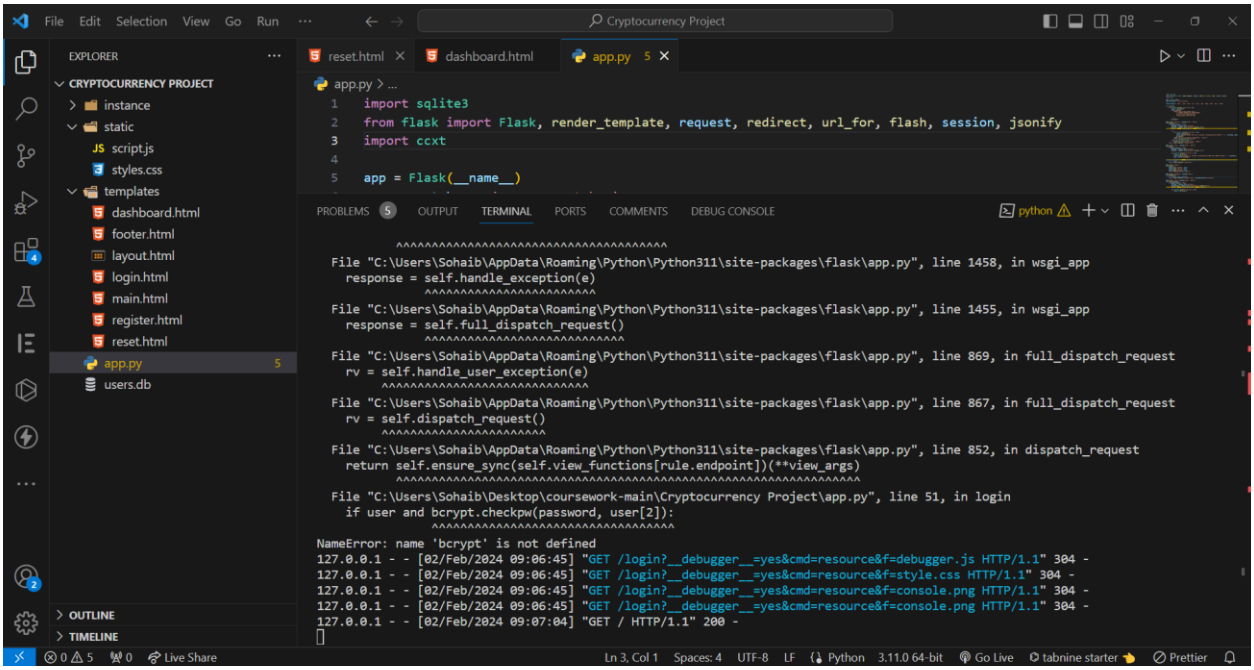
- **How I Solve This Error:**

```python
@app.route('/reset', methods=['GET', 'POST'])
def reset_password():
    if request.method == 'POST':
        username = request.form['username']
        new_password = request.form['new_password'].encode('utf-8')
        hashed = bcrypt.hashpw(new_password, bcrypt.gensalt())
```

This is a logical error which I solve by removing the decode('utf-8') and ensure the hased password.

4) Bcrypt Error:

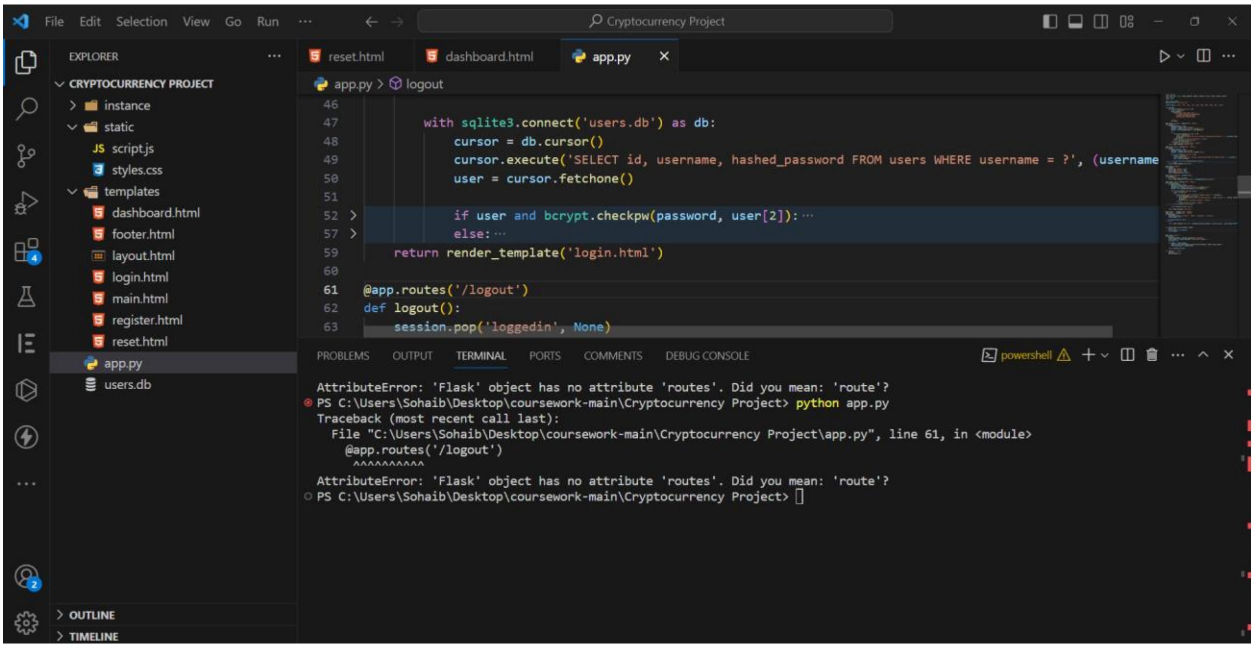




- **My Solution:**

This Error I face is due to the missing of import statement for bcrypt library. This error occurs when bcrypt is mentioned in app.py. By simple adding it, it easily removes.

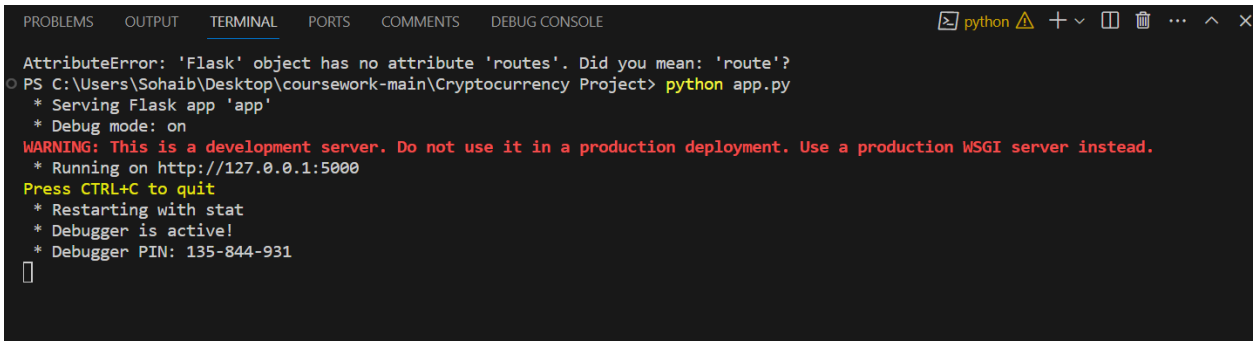
5) Error on route:

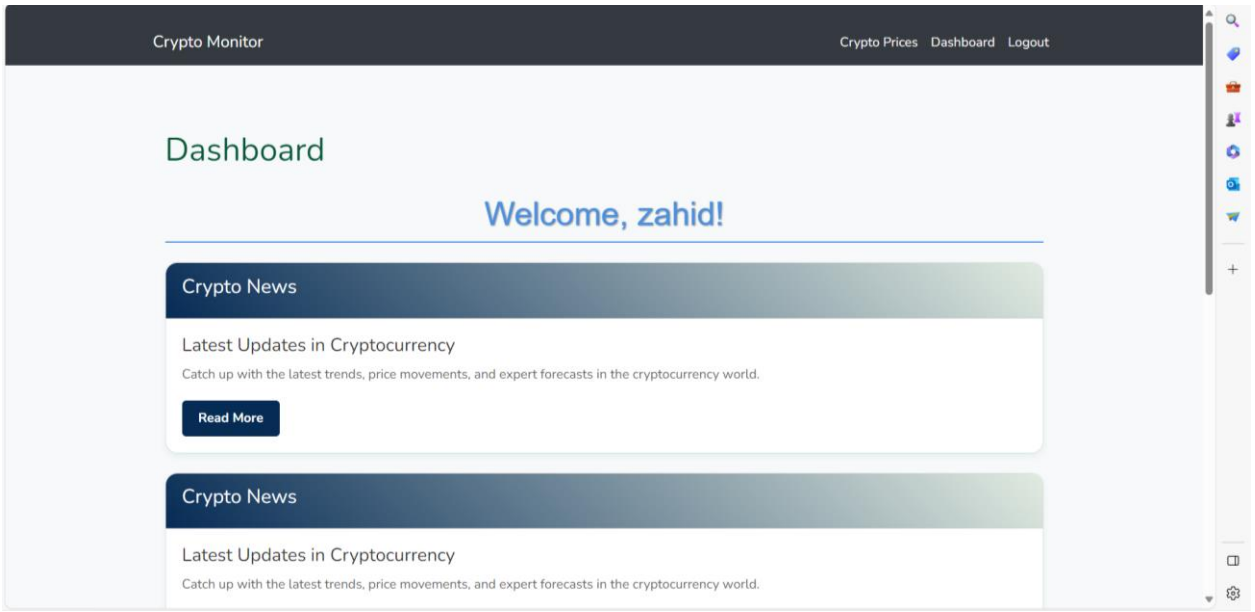


- **How I Solve this Error:**

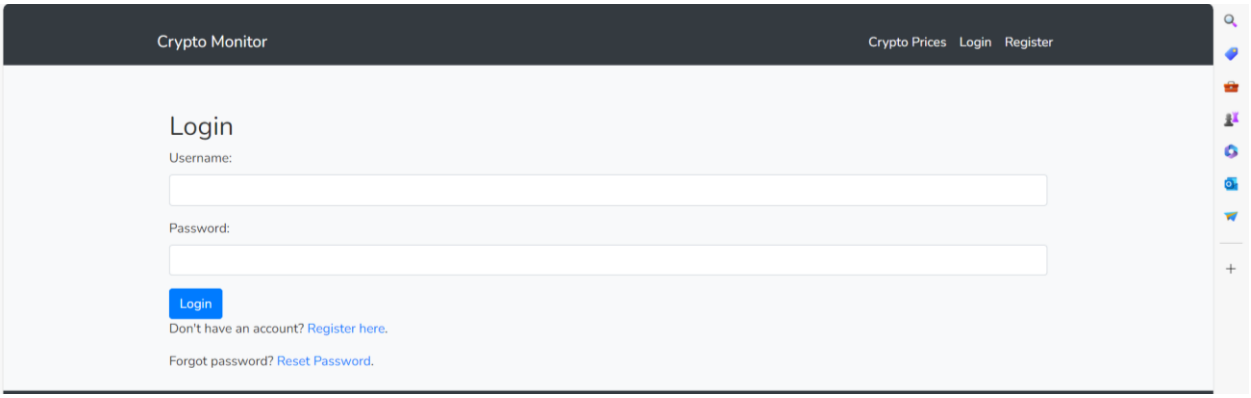
```python
@app.route('/logout')
def logout():
    session.pop('loggedin', None)
    session.pop('user_id', None)
    session.pop('username', None)
    return redirect(url_for('login'))
```

Our App Works perfectly, after Removing error:





When I click on logout, It Perfectly Works:



- **How I Solve This Error:**

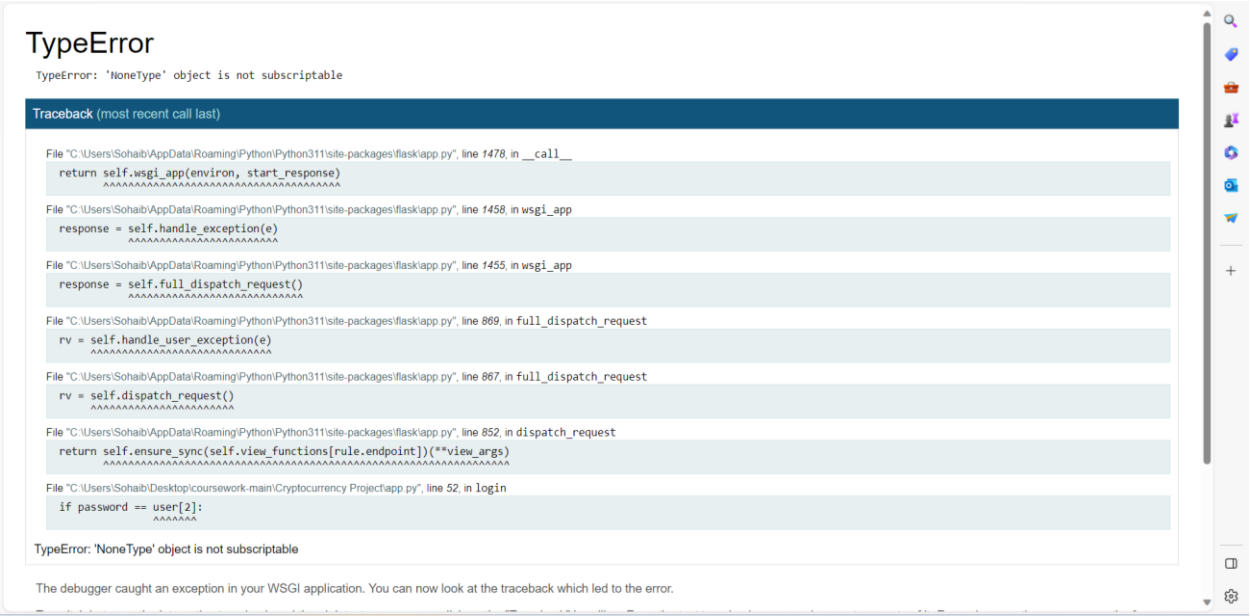
To fix this, I changed it to the correct Flask decorator @app.route. instead of @app.routes.

After correcting code is like this:

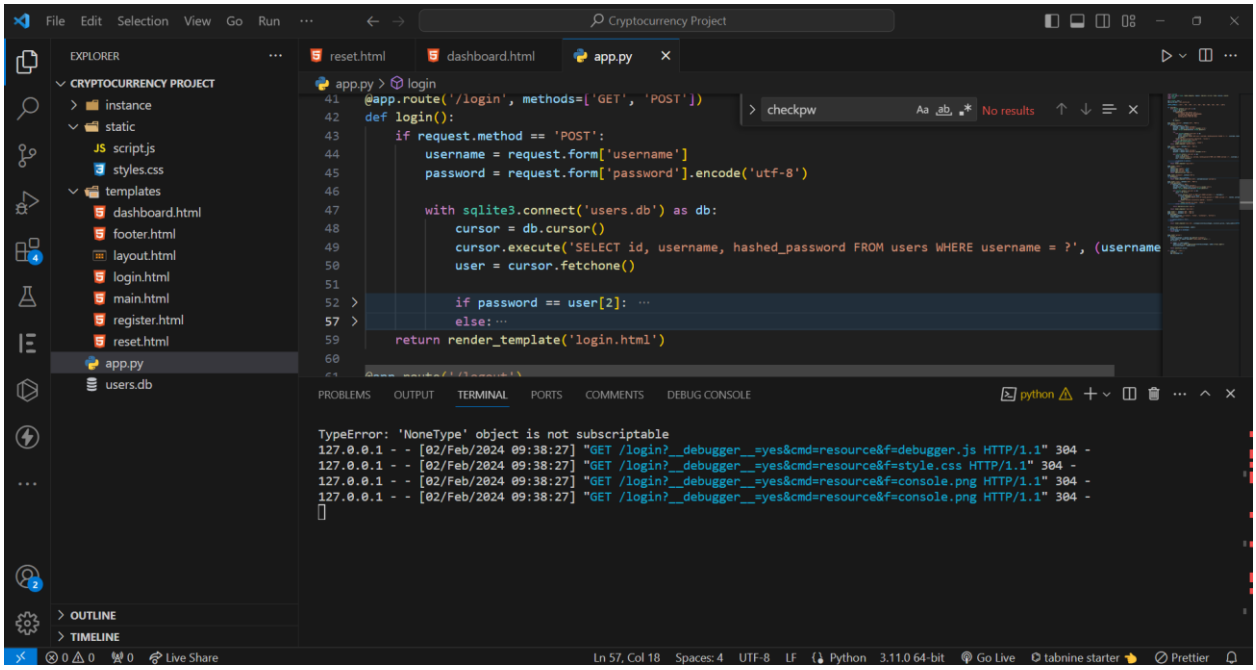
@app.route('/logout')

def logout():

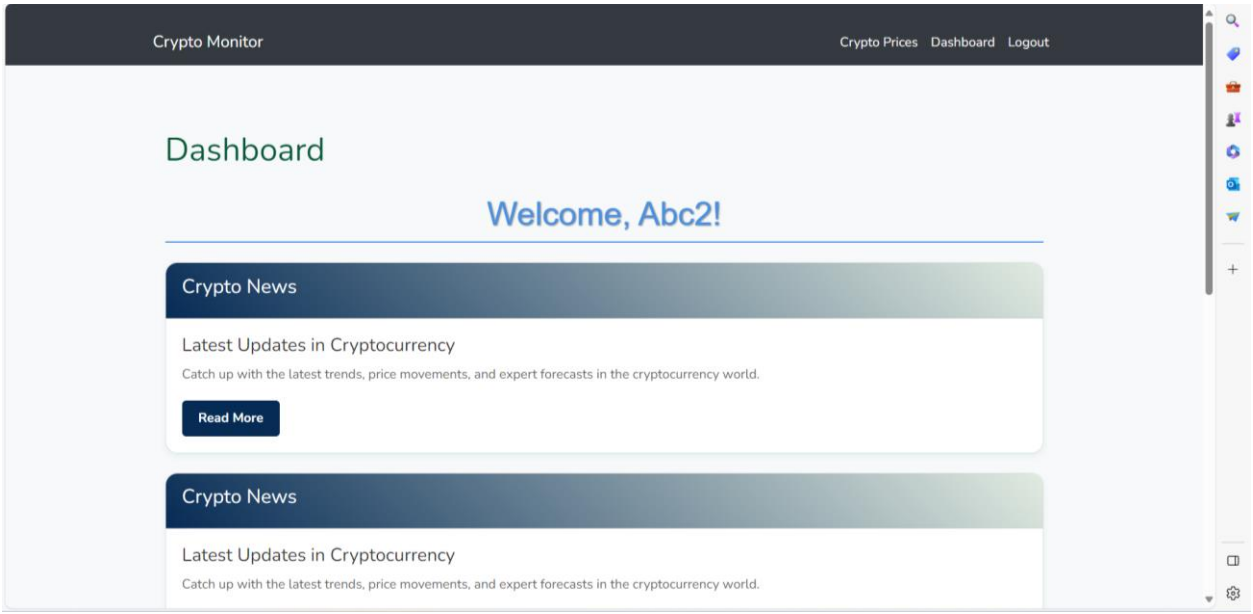
6) Error occurs when I Login:



7) Error on VS Code:



After Solve this Error, It Successfully Login:



- **How I Solve this Error:**

The correct approach involves using **bcrypt.checkpw** to compare the hashed version , of the input password with the hashed password from the database which is users.db , ensuring secure authentication.

```
        with sqlite3.connect('users.db') as db:
            cursor = db.cursor()
            cursor.execute('SELECT id, username, hashed_password FROM users WHERE username = ?', (username
            user = cursor.fetchone()

            if user and bcrypt.checkpw(password, user[2]):···
            else:···
    return render_template('login.html')

@app.route('/logout')
```

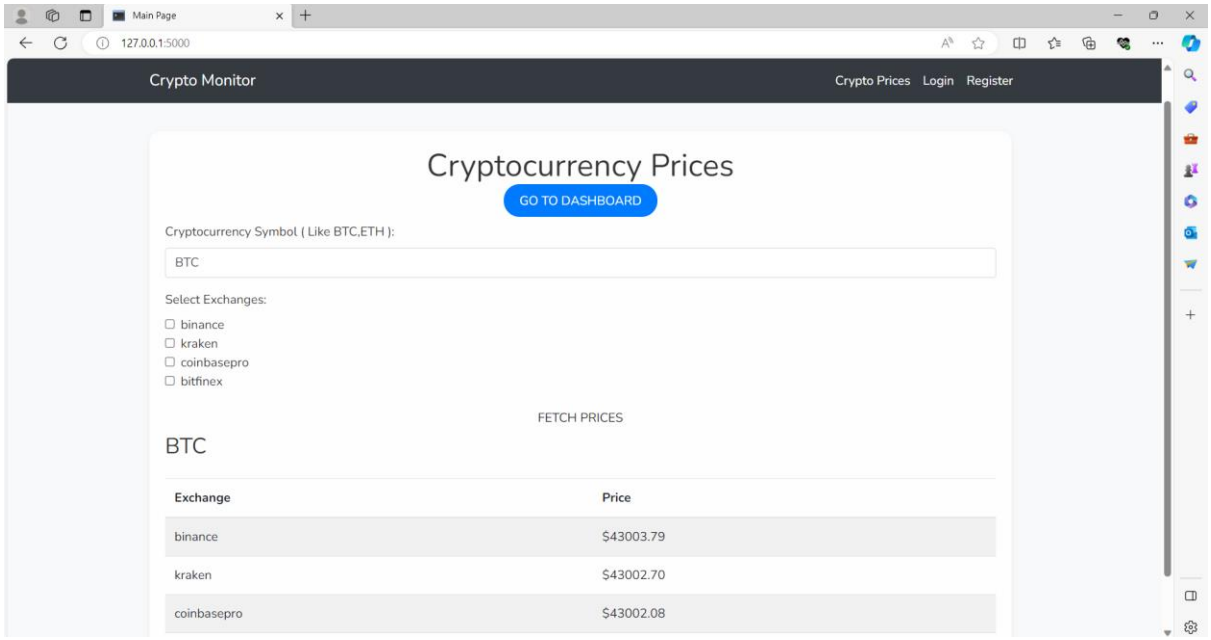
- **Error Fetching Prices:**

```
136
137         price = exchange.fetch_ticker(symbol + '/USDT')['last']
138         symbol_prices = fetch_crypto_prices(selected_exchanges, symbol.strip().upper())
139         all_prices[symbol] = symbol_prices
```

PROBLEMS 5 OUTPUT TERMINAL PORTS COMMENTS DEBUG CONSOLE powershell

```
 * Debugger is active!
 * Debugger PIN: 135-844-931
127.0.0.1 - - [02/Feb/2024 09:41:23] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [02/Feb/2024 09:41:30] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [02/Feb/2024 09:41:37] "POST /login HTTP/1.1" 302 -
127.0.0.1 - - [02/Feb/2024 09:41:37] "GET /dashboard HTTP/1.1" 200 -
 * Detected change in 'C:\\Users\\Sohaib\\Desktop\\coursework-main\\Cryptocurrency Project\\app.py', reloading
 * Restarting with stat
  File "C:\Users\Sohaib\Desktop\coursework-main\Cryptocurrency Project\app.py", line 138
    symbol_prices = fetch_crypto_prices(selected_exchanges, symbol.strip().upper())
IndentationError: unexpected indent
PS C:\Users\Sohaib\Desktop\coursework-main\Cryptocurrency Project>
```

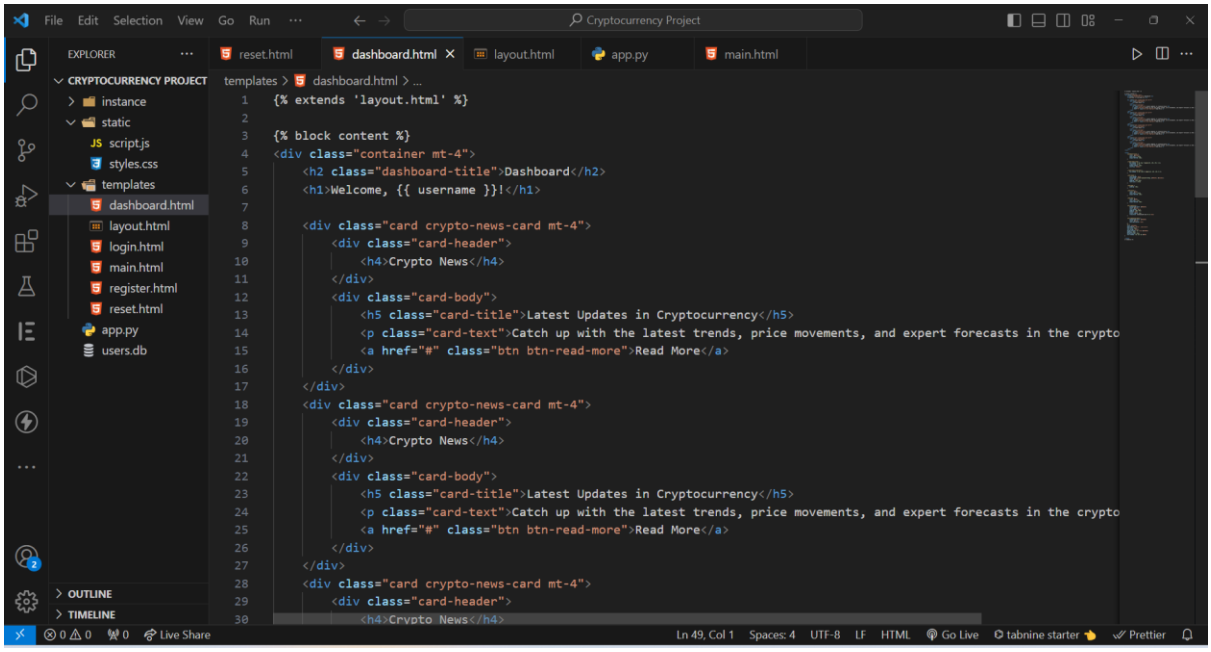
After Solving this Error Fetching:



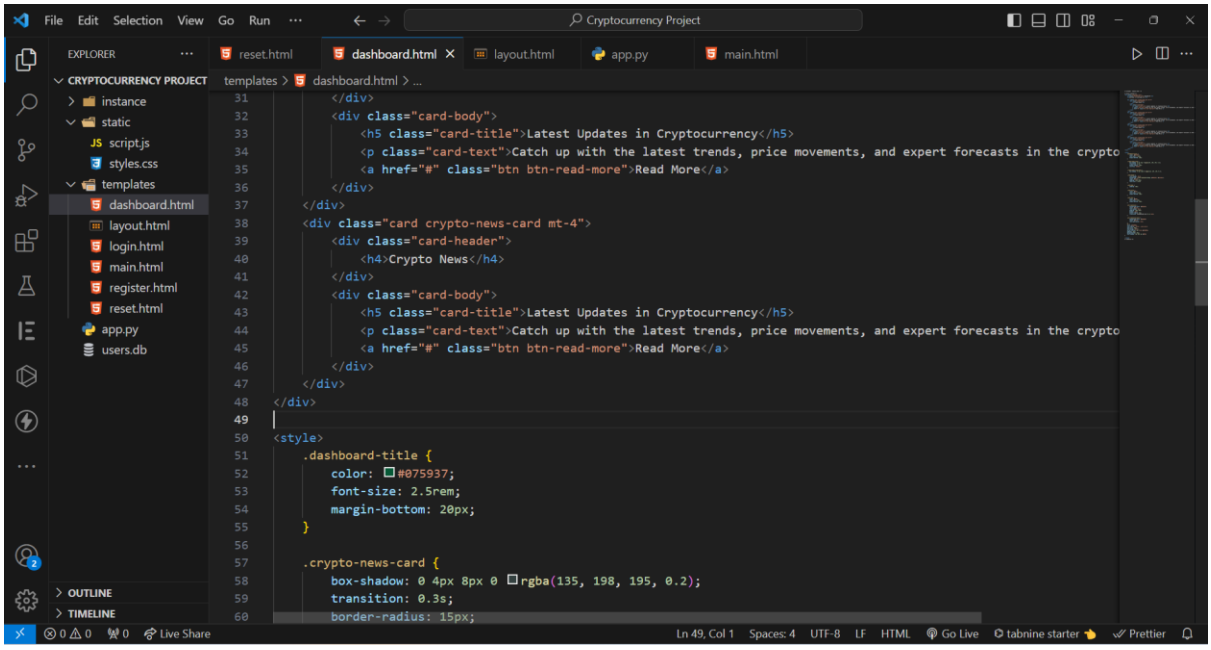
Technical solution:

Code Documentation:

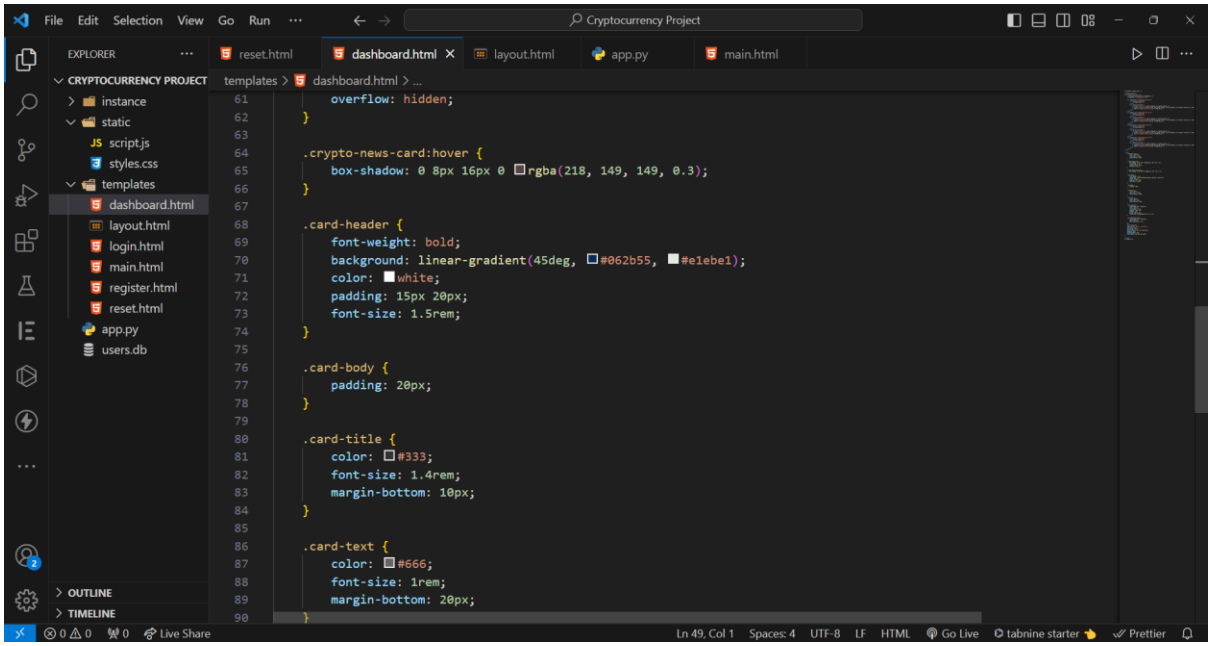
I. Dashboard.html:

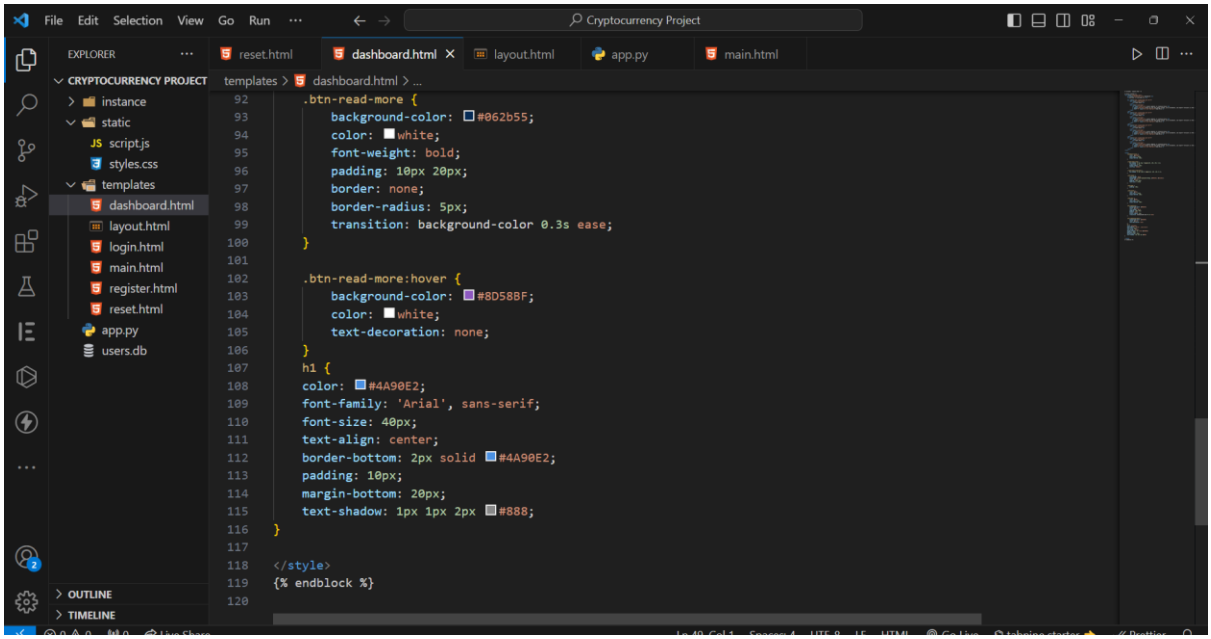


```
{% extends 'layout.html' %}

{% block content %}
<div class="container mt-4">
    <h2 class="dashboard-title">Dashboard</h2>
    <h1>Welcome, {{ username }}!</h1>

    <div class="card crypto-news-card mt-4">
        <div class="card-header">
            <h4>Crypto News</h4>
        </div>
        <div class="card-body">
            <h5 class="card-title">Latest Updates in Cryptocurrency</h5>
            <p class="card-text">Catch up with the latest trends, price movements, and expert forecasts in the crypto
            <a href="#" class="btn btn-read-more">Read More</a>
        </div>
    </div>
    <div class="card crypto-news-card mt-4">
        <div class="card-header">
            <h4>Crypto News</h4>
        </div>
        <div class="card-body">
            <h5 class="card-title">Latest Updates in Cryptocurrency</h5>
            <p class="card-text">Catch up with the latest trends, price movements, and expert forecasts in the crypto
            <a href="#" class="btn btn-read-more">Read More</a>
        </div>
    </div>
    <div class="card crypto-news-card mt-4">
        <div class="card-header">
            <h4>Crypto News</h4>
```

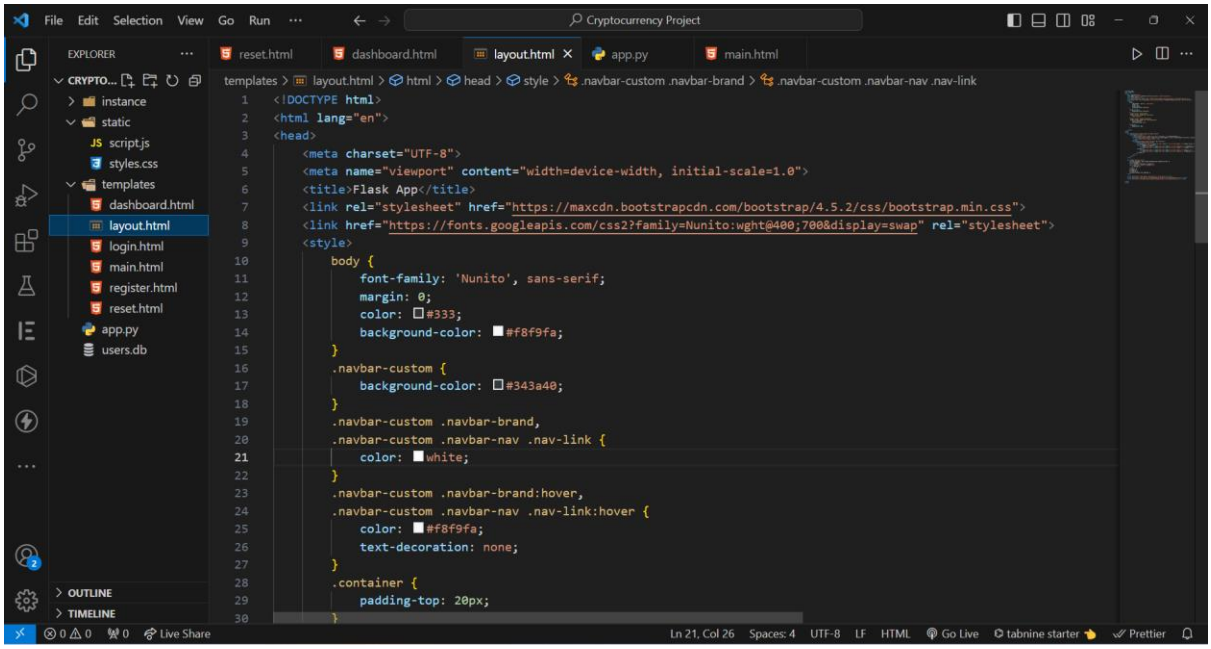


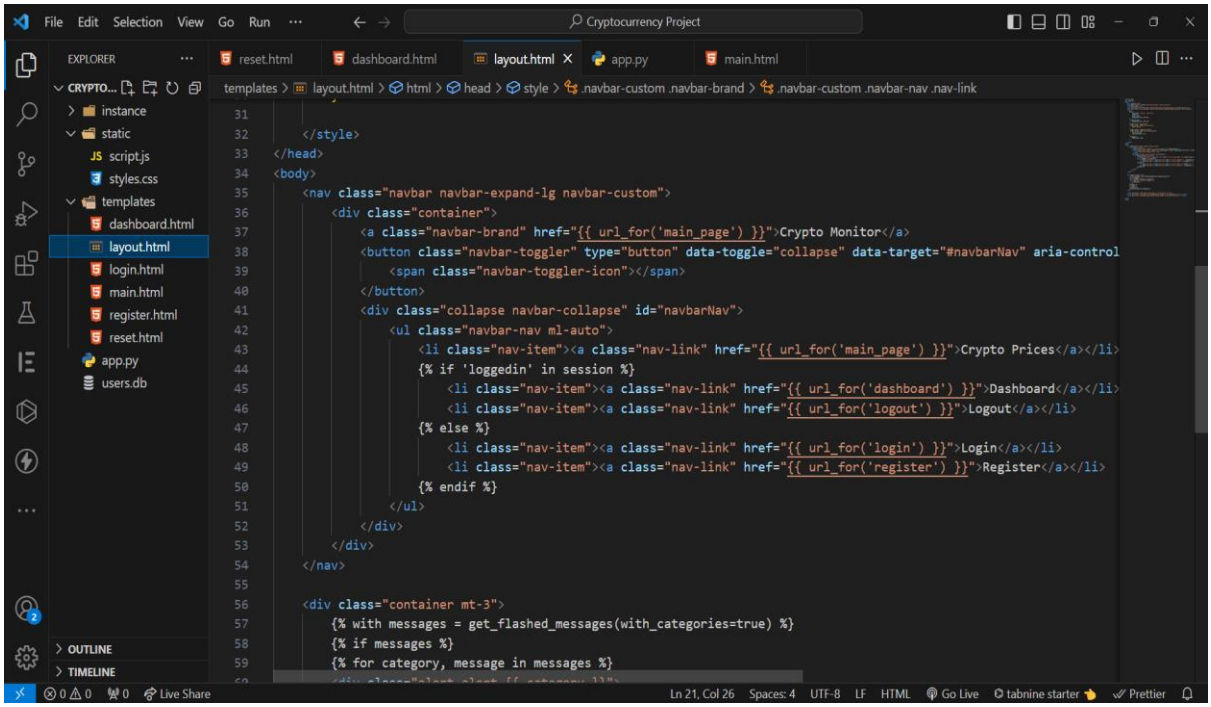
```
        </div>
        <div class="card-body">
            <h5 class="card-title">Latest Updates in Cryptocurrency</h5>
            <p class="card-text">Catch up with the latest trends, price movements, and expert forecasts in the crypto
            <a href="#" class="btn btn-read-more">Read More</a>
        </div>
    </div>
    <div class="card crypto-news-card mt-4">
        <div class="card-header">
            <h4>Crypto News</h4>
        </div>
        <div class="card-body">
            <h5 class="card-title">Latest Updates in Cryptocurrency</h5>
            <p class="card-text">Catch up with the latest trends, price movements, and expert forecasts in the crypto
            <a href="#" class="btn btn-read-more">Read More</a>
        </div>
    </div>
</div>

<style>
    .dashboard-title {
        color: #075937;
        font-size: 2.5rem;
        margin-bottom: 20px;
    }

    .crypto-news-card {
        box-shadow: 0 4px 8px 0 rgba(135, 198, 195, 0.2);
        transition: 0.3s;
        border-radius: 15px;
```





II. Layout.html:


File Edit Selection View Go Run
Cryptocurrency Project
EXPLORER
reset.html
dashboard.html
layout.html
app.py
main.html
templates > layout.html > html > head > style > .navbar-custom .navbar-brand > .navbar-custom .navbar-nav .nav-link
CRYPTO...
instance
static
script.js
styles.css
templates
dashboard.html
layout.html
login.html
main.html
register.html
reset.html
app.py
users.db
OUTLINE
TIMELINE
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Flask App</title>
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
<link href="https://fonts.googleapis.com/css2?family=Nunito:wght@400;700&display=swap" rel="stylesheet">
<style>
body {
font-family: 'Nunito', sans-serif;
margin: 0;
color: #333;
background-color: #f8f9fa;
}
.navbar-custom {
background-color: #343a40;
}
.navbar-custom .navbar-brand,
.navbar-custom .navbar-nav .nav-link {
color: white;
}
.navbar-custom .navbar-brand:hover,
.navbar-custom .navbar-nav .nav-link:hover {
color: #f8f9fa;
text-decoration: none;
}
.container {
padding-top: 20px;
}
Ln 21, Col 26 Spaces: 4 UTF-8 LF HTML Go Live tabnine starter Prettier
Live Share


File Edit Selection View Go Run
Cryptocurrency Project
EXPLORER
reset.html
dashboard.html
layout.html
app.py
main.html
templates > layout.html > html > head > style > .navbar-custom .navbar-brand > .navbar-custom .navbar-nav .nav-link
CRYPTO...
instance
static
script.js
styles.css
templates
dashboard.html
layout.html
login.html
main.html
register.html
reset.html
app.py
users.db
OUTLINE
TIMELINE
</style>
</head>
<body>
<nav class="navbar navbar-expand-lg navbar-custom">

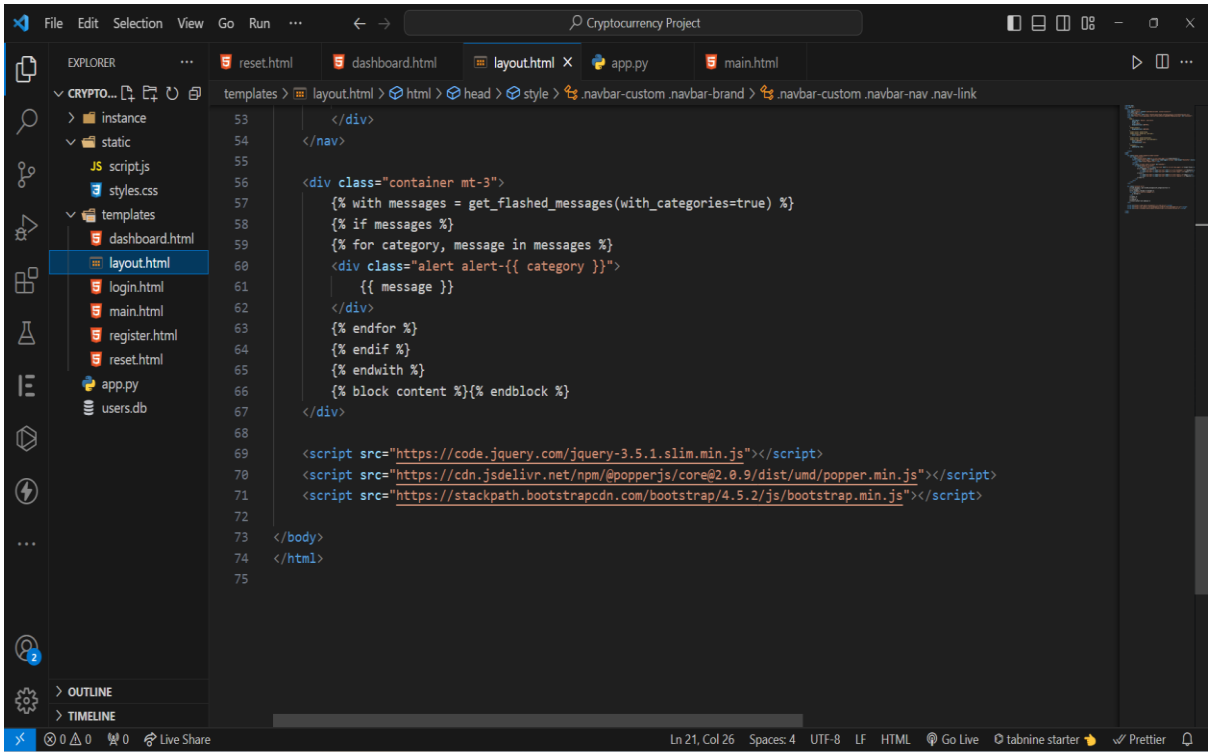
Crypto Monitor
<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-control

</button>

<ul class="navbar-nav ml-auto">
<li class="nav-item">Crypto Prices
{% if 'loggedin' in session %}
<li class="nav-item">Dashboard
<li class="nav-item">Logout
{% else %}
<li class="nav-item">Login
<li class="nav-item">Register
{% endif %}

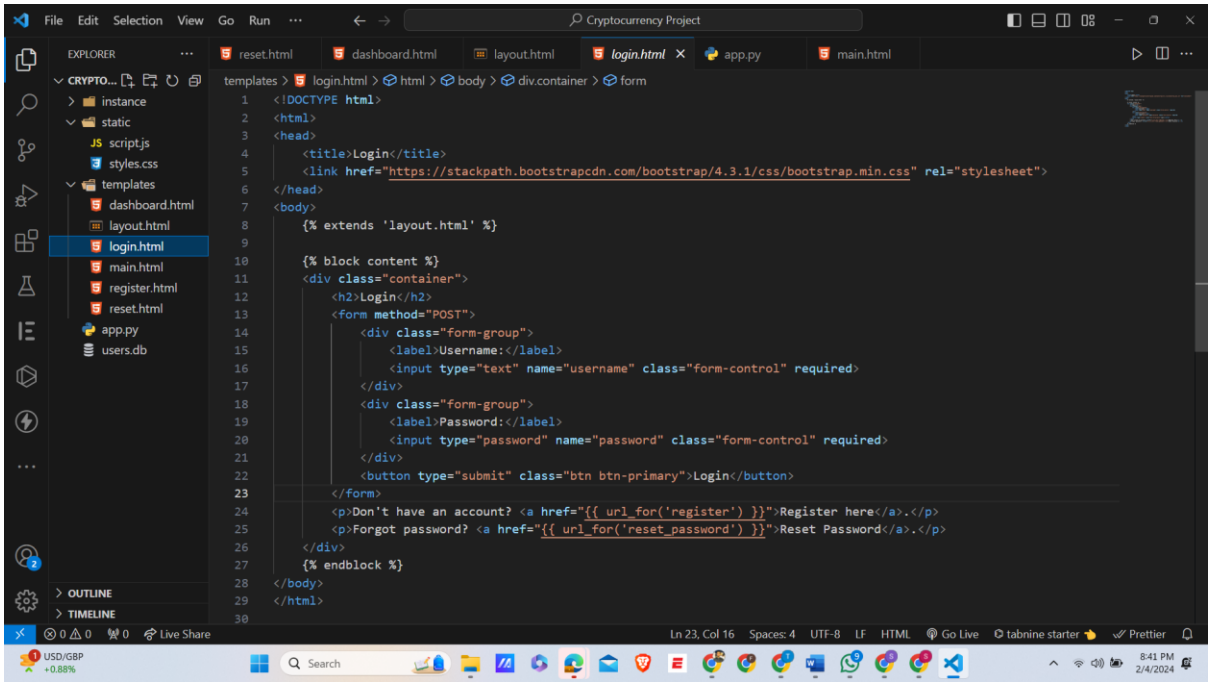
</nav>

{% with messages = get_flashed_messages(with_categories=true) %}
{% if messages %}
{% for category, message in messages %}
Ln 21, Col 26 Spaces: 4 UTF-8 LF HTML Go Live tabnine starter Prettier
Live Share



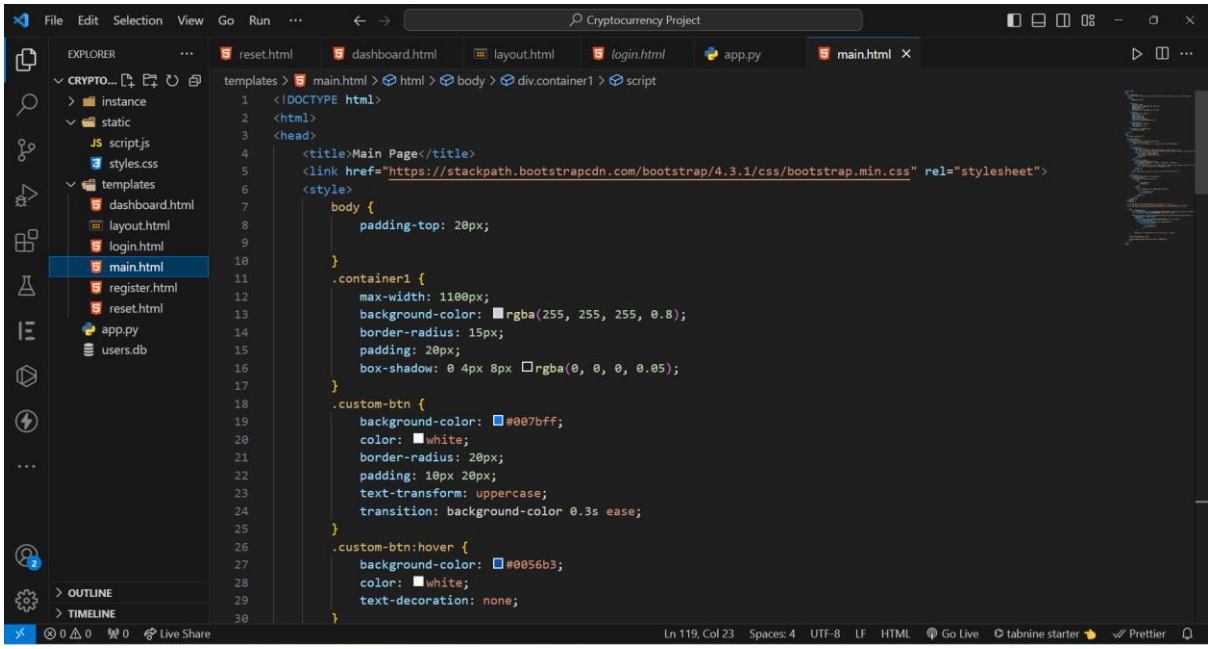

```
        </div>
    </nav>

    <div class="container mt-3">
        {% with messages = get_flashed_messages(with_categories=true) %}
        {% if messages %}
        {% for category, message in messages %}
        <div class="alert alert-{{ category }}">
            {{ message }}
        </div>
        {% endfor %}
        {% endif %}
        {% endwith %}
        {% block content %}{% endblock %}
    </div>

    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.0.9/dist/umd/popper.min.js"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>
</html>
```

III. Login.html:

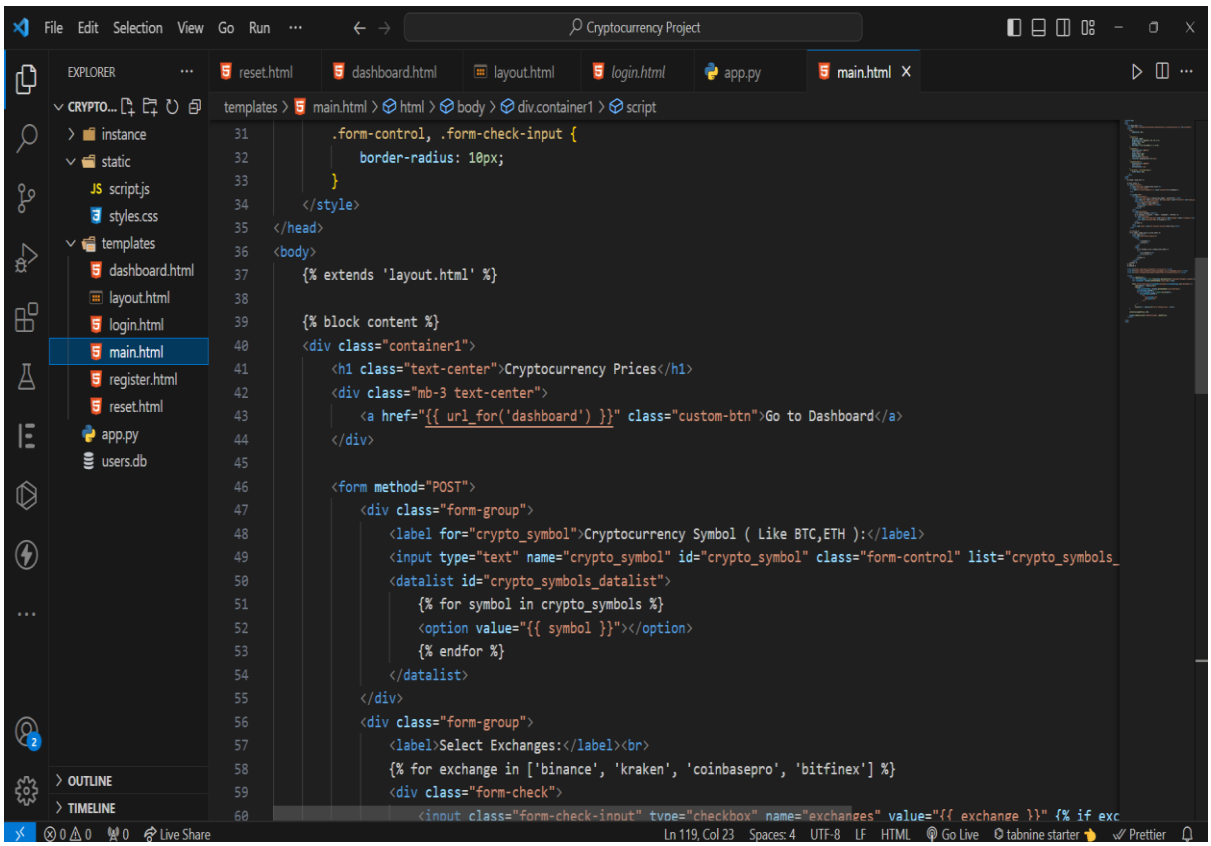



```
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
    <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
    {% extends 'layout.html' %}

    {% block content %}
    <div class="container">
        <h2>Login</h2>
        <form method="POST">
            <div class="form-group">
                <label>Username:</label>
                <input type="text" name="username" class="form-control" required>
            </div>
            <div class="form-group">
                <label>Password:</label>
                <input type="password" name="password" class="form-control" required>
            </div>
            <button type="submit" class="btn btn-primary">Login</button>
        </form>
        <p>Don't have an account? <a href="{{ url_for('register') }}">Register here</a>.</p>
        <p>Forgot password? <a href="{{ url_for('reset_password') }}">Reset Password</a>.</p>
    </div>
    {% endblock %}
</body>
</html>
```

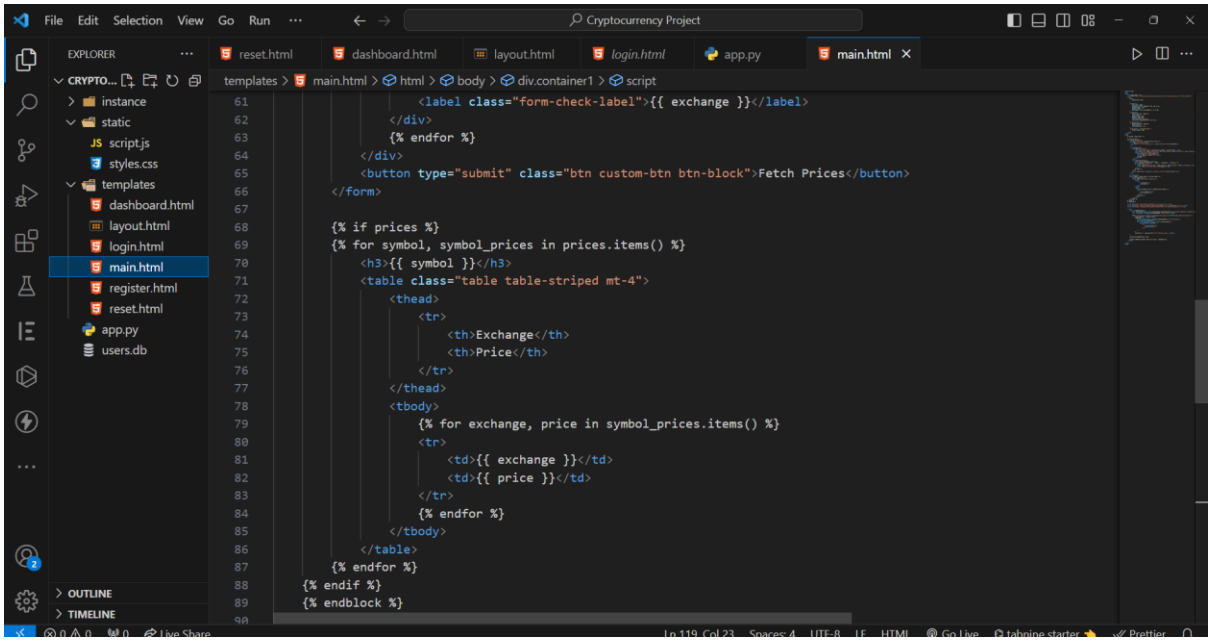
IV. Main.html:

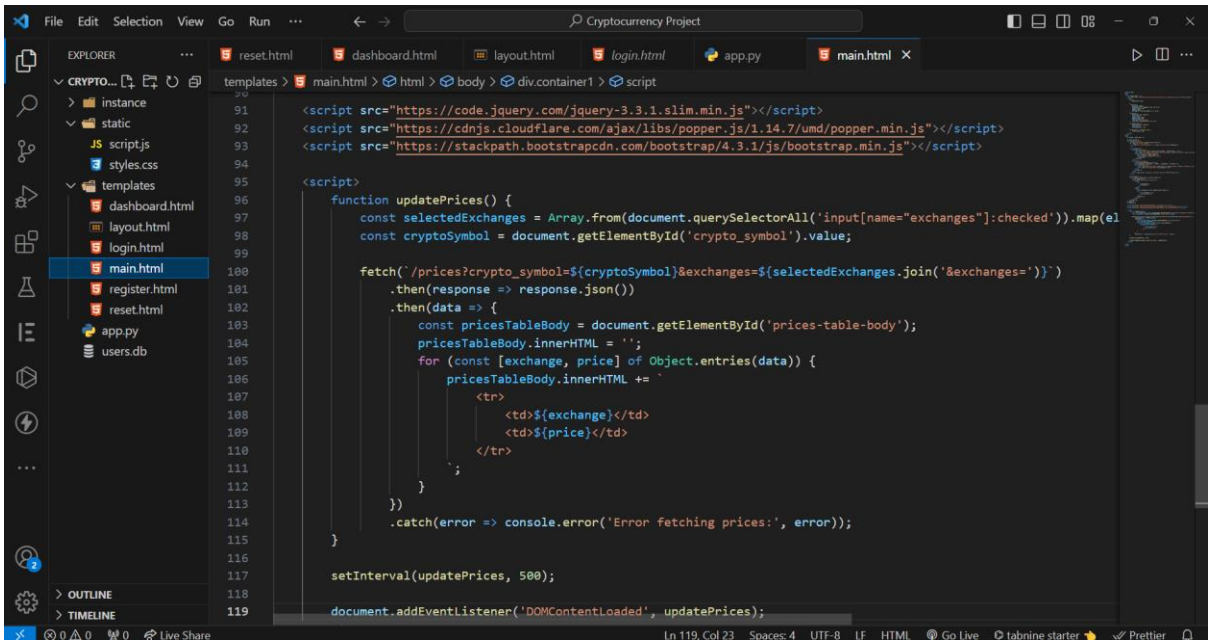



```
<!DOCTYPE html>
<html>
<head>
    <title>Main Page</title>
    <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" rel="stylesheet">
    <style>
        body {
            padding-top: 20px;

        }
        .container1 {
            max-width: 1100px;
            background-color: rgba(255, 255, 255, 0.8);
            border-radius: 15px;
            padding: 20px;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
        }
        .custom-btn {
            background-color: #007bff;
            color: white;
            border-radius: 20px;
            padding: 10px 20px;
            text-transform: uppercase;
            transition: background-color 0.3s ease;
        }
        .custom-btn:hover {
            background-color: #0056b3;
            color: white;
            text-decoration: none;
        }
```

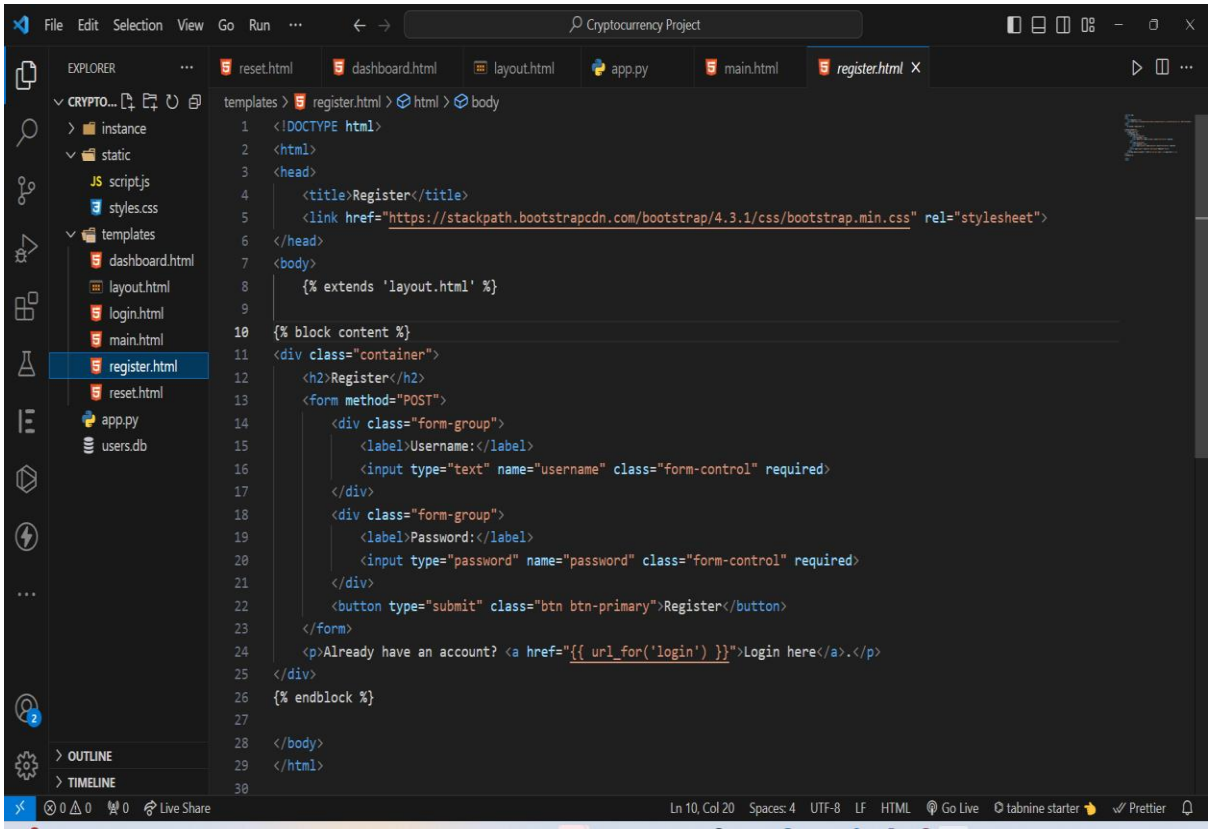



```
        .form-control, .form-check-input {
            border-radius: 10px;
        }
    </style>
</head>
<body>
    {% extends 'layout.html' %}

    {% block content %}
    <div class="container1">
        <h1 class="text-center">Cryptocurrency Prices</h1>
        <div class="mb-3 text-center">
            <a href="{{ url_for('dashboard') }}" class="custom-btn">Go to Dashboard</a>
        </div>

        <form method="POST">
            <div class="form-group">
                <label for="crypto_symbol">Cryptocurrency Symbol ( Like BTC,ETH ):</label>
                <input type="text" name="crypto_symbol" id="crypto_symbol" class="form-control" list="crypto_symbols_
                <datalist id="crypto_symbols_datalist">
                    {% for symbol in crypto_symbols %}
                    <option value="{{ symbol }}"></option>
                    {% endfor %}
                </datalist>
            </div>
            <div class="form-group">
                <label>Select Exchanges:</label><br>
                {% for exchange in ['binance', 'kraken', 'coinbasepro', 'bitfinex'] %}
                <div class="form-check">
                    <input class="form-check-input" type="checkbox" name="exchanges" value="{{ exchange }}" {% if exc
```



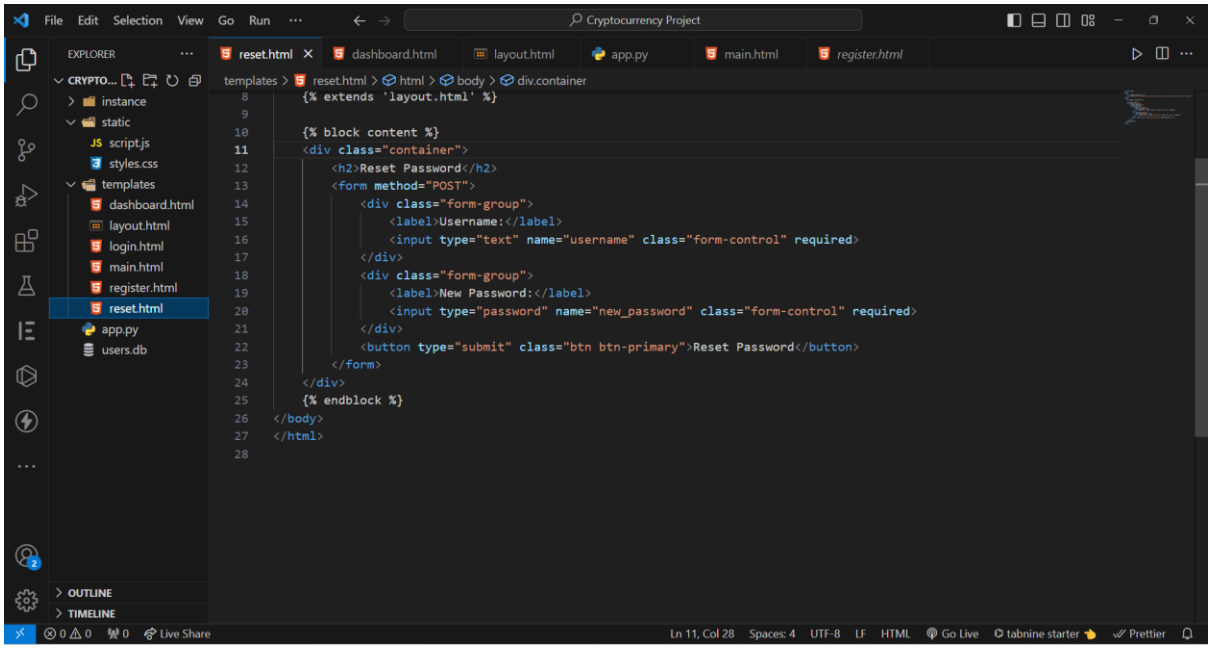


V. Register.html:



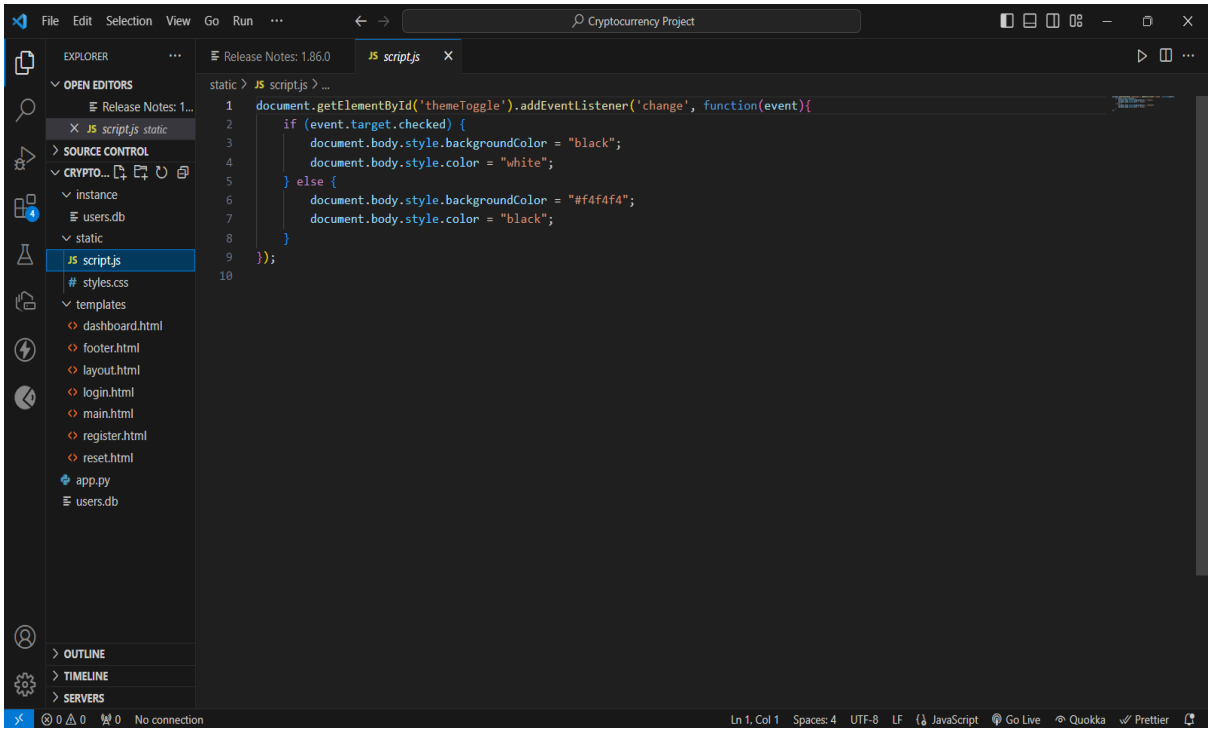

```
<!DOCTYPE html>
<html>
<head>
    <title>Register</title>
    <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
    {% extends 'layout.html' %}

{% block content %}
<div class="container">
    <h2>Register</h2>
    <form method="POST">
        <div class="form-group">
            <label>Username:</label>
            <input type="text" name="username" class="form-control" required>
        </div>
        <div class="form-group">
            <label>Password:</label>
            <input type="password" name="password" class="form-control" required>
        </div>
        <button type="submit" class="btn btn-primary">Register</button>
    </form>
    <p>Already have an account? <a href="{{ url_for('login') }}">Login here</a>.</p>
</div>
{% endblock %}

</body>
</html>
```

VI. Reset.html:

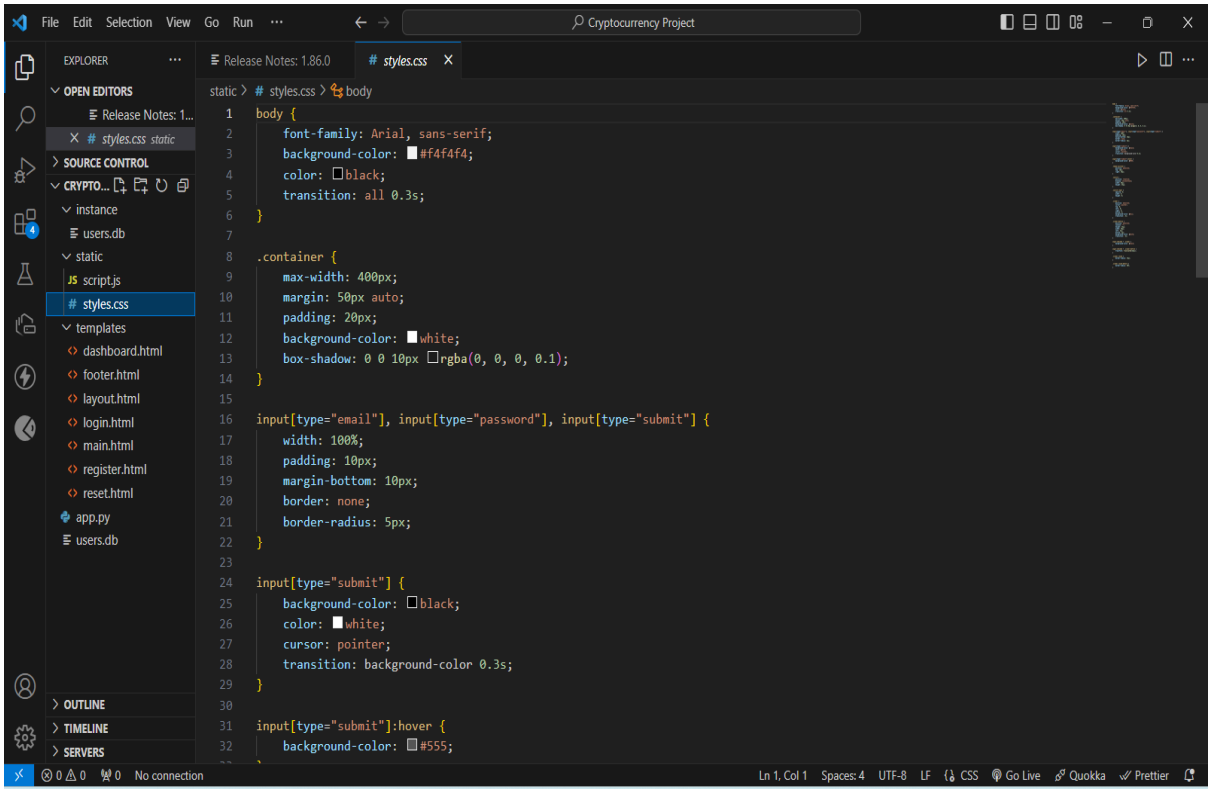


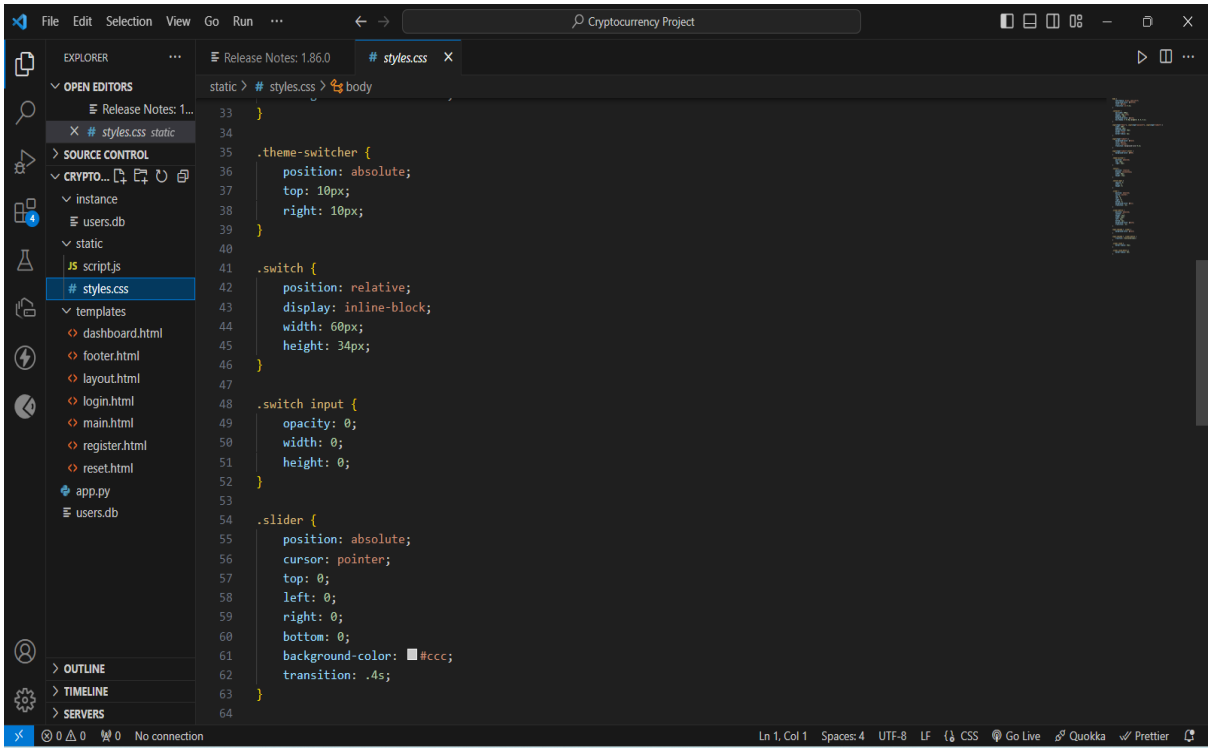

```
    {% extends 'layout.html' %}

    {% block content %}
    <div class="container">
        <h2>Reset Password</h2>
        <form method="POST">
            <div class="form-group">
                <label>Username:</label>
                <input type="text" name="username" class="form-control" required>
            </div>
            <div class="form-group">
                <label>New Password:</label>
                <input type="password" name="new_password" class="form-control" required>
            </div>
            <button type="submit" class="btn btn-primary">Reset Password</button>
        </form>
    </div>
    {% endblock %}
</body>
</html>
```

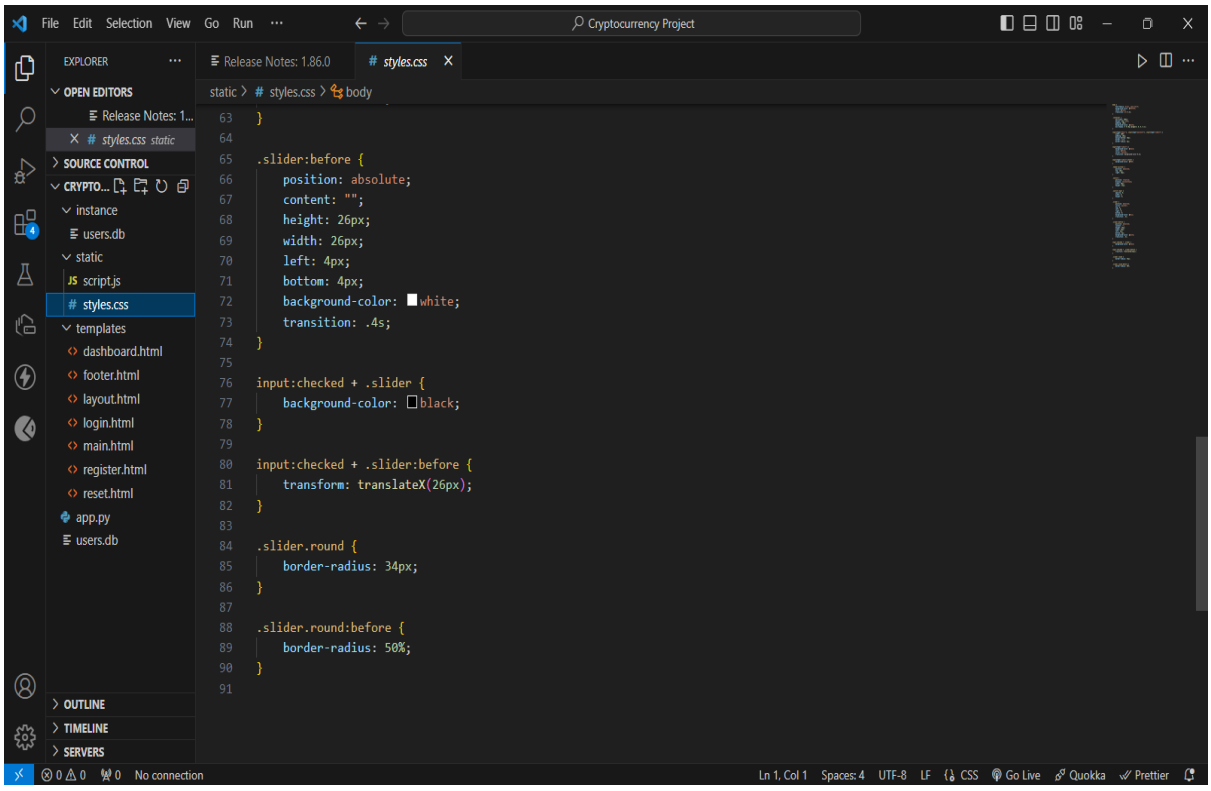
VII. Script.js:

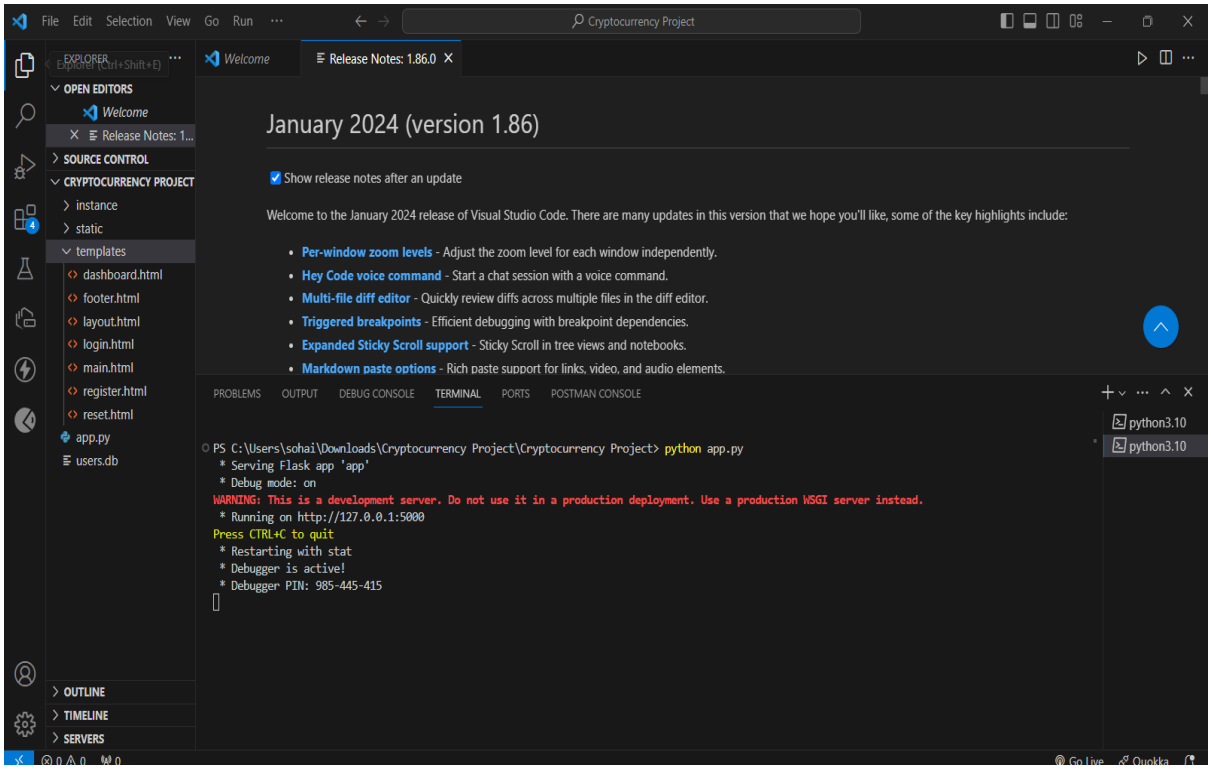


VIII. Style.css:

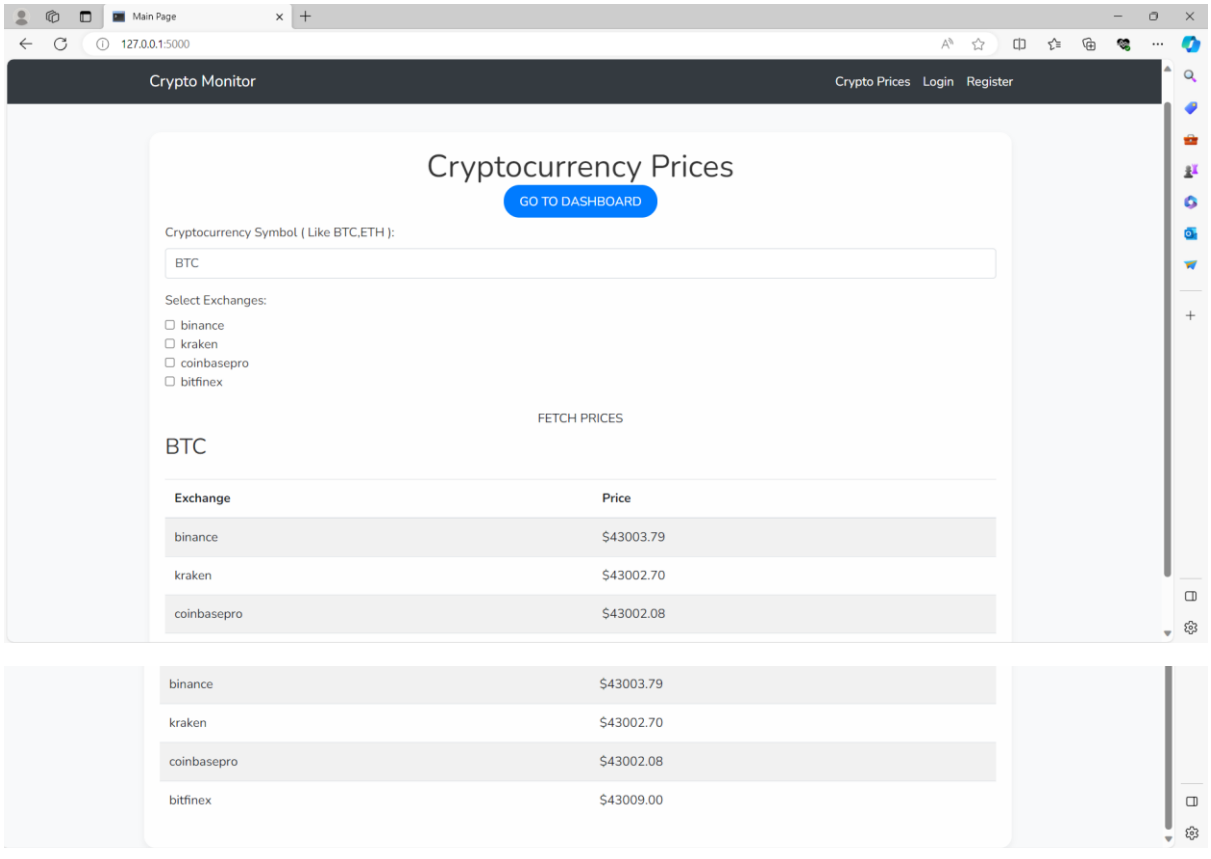


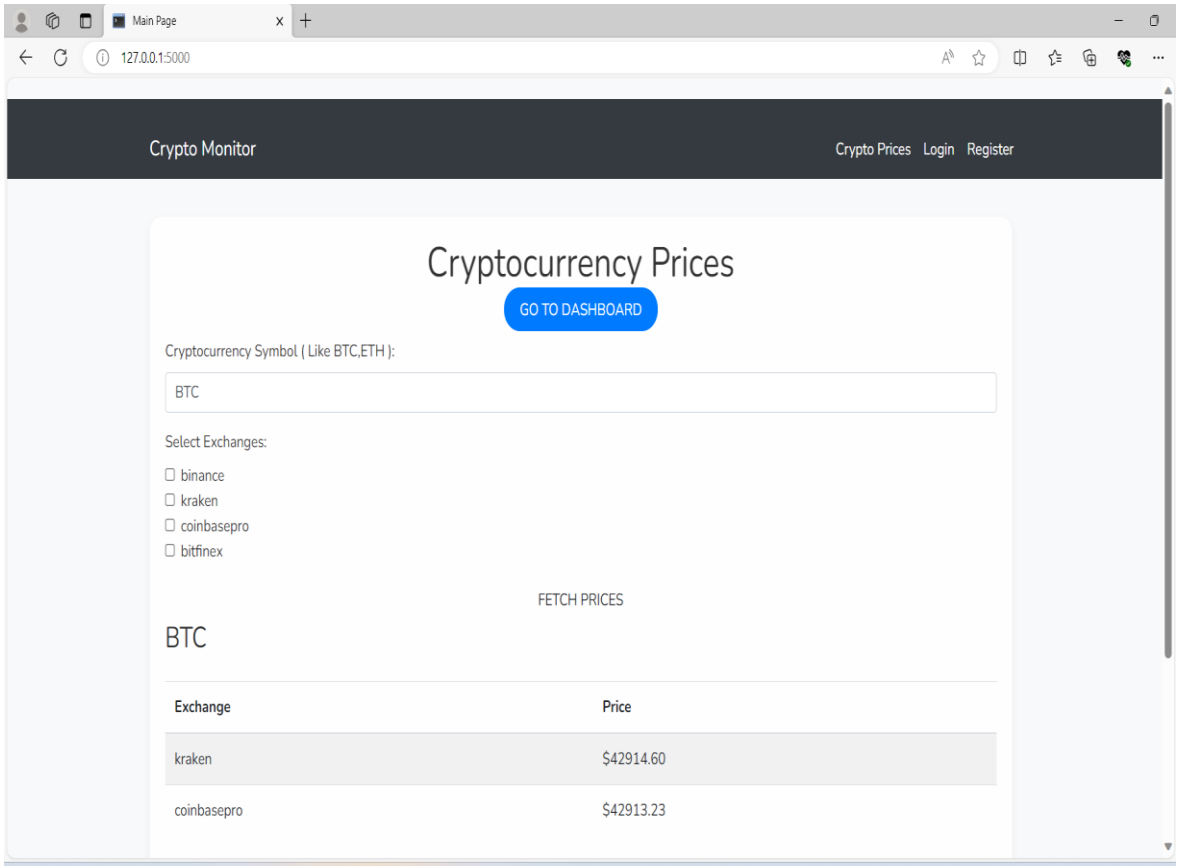

File Edit Selection View Go Run
Cryptocurrency Project
EXPLORER
OPEN EDITORS
Release Notes: 1...
styles.css static
SOURCE CONTROL
CRYPTO...
instance
users.db
static
script.js
styles.css
templates
dashboard.html
footer.html
layout.html
login.html
main.html
register.html
reset.html
app.py
users.db
OUTLINE
TIMELINE
SERVERS
Release Notes: 1.86.0
styles.css
static > styles.css > body
33 }
34
35 .theme-switcher {
36 position: absolute;
37 top: 10px;
38 right: 10px;
39 }
40
41 .switch {
42 position: relative;
43 display: inline-block;
44 width: 60px;
45 height: 34px;
46 }
47
48 .switch input {
49 opacity: 0;
50 width: 0;
51 height: 0;
52 }
53
54 .slider {
55 position: absolute;
56 cursor: pointer;
57 top: 0;
58 left: 0;
59 right: 0;
60 bottom: 0;
61 background-color: #ccc;
62 transition: .4s;
63 }
64
0 0 0 No connection
Ln 1, Col 1 Spaces: 4 UTF-8 LF CSS Go Live Quokka Prettier

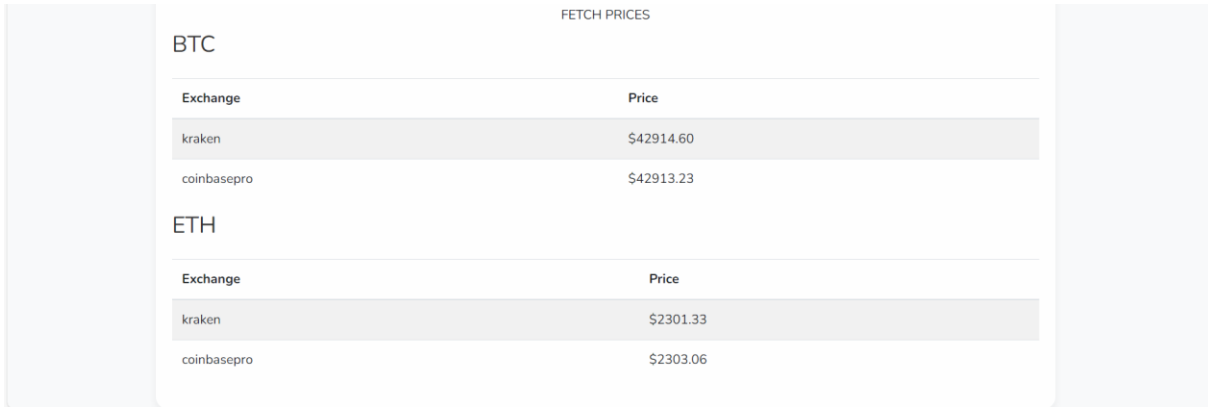

File Edit Selection View Go Run
Cryptocurrency Project
EXPLORER
OPEN EDITORS
Release Notes: 1...
styles.css static
SOURCE CONTROL
CRYPTO...
instance
users.db
static
script.js
styles.css
templates
dashboard.html
footer.html
layout.html
login.html
main.html
register.html
reset.html
app.py
users.db
OUTLINE
TIMELINE
SERVERS
Release Notes: 1.86.0
styles.css
static > styles.css > body
63 }
64
65 .slider:before {
66 position: absolute;
67 content: "";
68 height: 26px;
69 width: 26px;
70 left: 4px;
71 bottom: 4px;
72 background-color: white;
73 transition: .4s;
74 }
75
76 input:checked + .slider {
77 background-color: black;
78 }
79
80 input:checked + .slider:before {
81 transform: translateX(26px);
82 }
83
84 .slider.round {
85 border-radius: 34px;
86 }
87
88 .slider.round:before {
89 border-radius: 50%;
90 }
91
0 0 0 No connection
Ln 1, Col 1 Spaces: 4 UTF-8 LF CSS Go Live Quokka Prettier

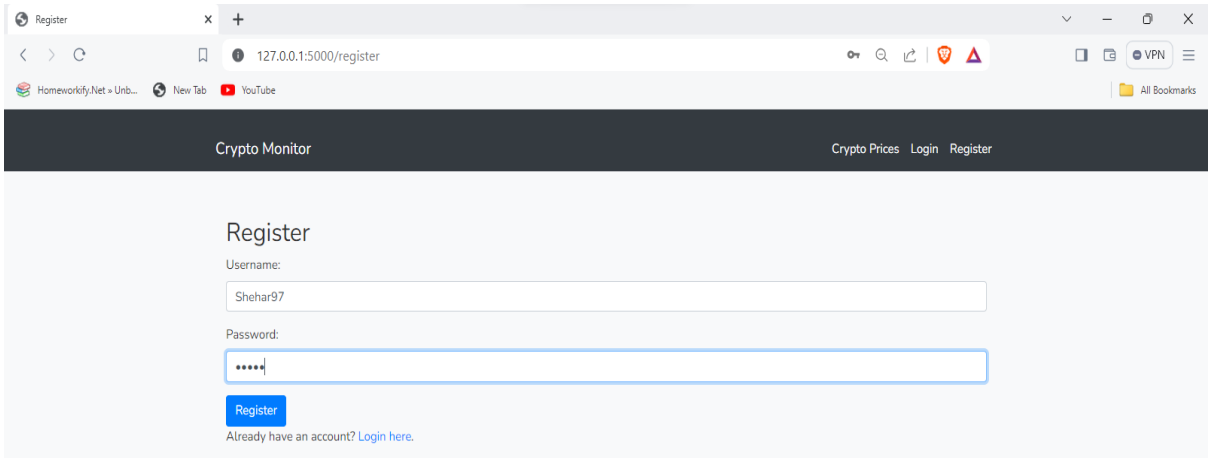


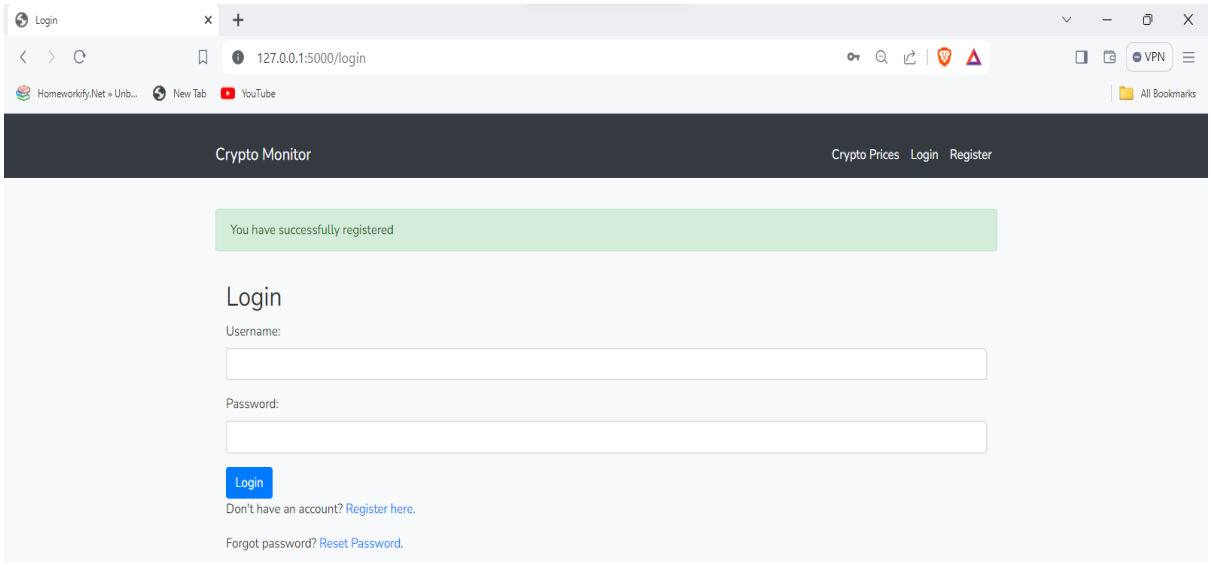
Execution of Our Crypto Monitor System:

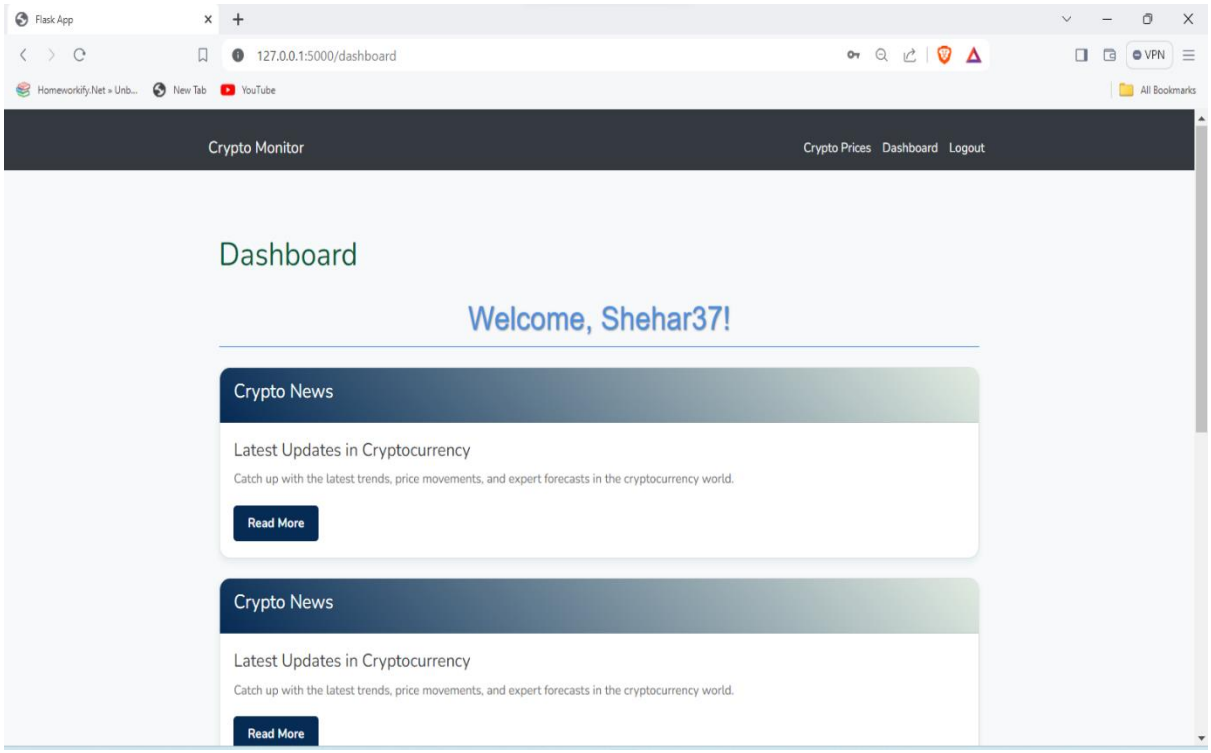


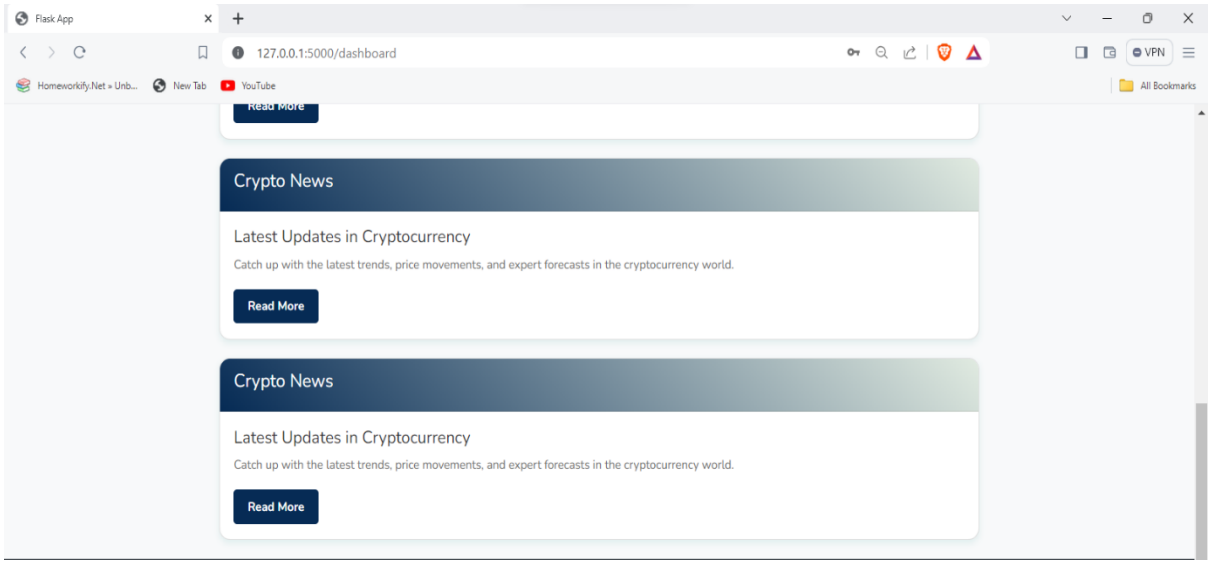

Main Page
127.0.0.1:5000
Crypto Monitor
Crypto Prices
Login
Register
Cryptocurrency Prices
GO TO DASHBOARD
Cryptocurrency Symbol (Like BTC,ETH):
BTC
Select Exchanges:
binance
kraken
coinbasepro
bitfinex
FETCH PRICES
BTC
Exchange
Price
kraken
$42914.60
coinbasepro
$42913.23


FETCH PRICES
BTC
Exchange
Price
kraken
$42914.60
coinbasepro
$42913.23
ETH
Exchange
Price
kraken
$2301.33
coinbasepro
$2303.06


Register
127.0.0.1:5000/register
Homeworkify.Net » Unb...
New Tab
YouTube
All Bookmarks
Crypto Monitor
Crypto Prices
Login
Register
Register
Username:
Shehar97
Password:
Register
Already have an account? Login here.


Login
127.0.0.1:5000/login
Homeworkify.Net » Unb...
New Tab
YouTube
All Bookmarks
Crypto Monitor
Crypto Prices
Login
Register
You have successfully registered
Login
Username:
Password:
Login
Don't have an account? Register here.
Forgot password? Reset Password.





Implementation:

Dashbord.html:

templates > dashboard.html > ...

```
{% extends 'layout.html' %}

{% block content %}
<div class="container mt-4">
    <h2 class="dashboard-title">Dashboard</h2>
    <h1>Welcome, {{ username }}!</h1>

    <div class="card crypto-news-card mt-4">
        <div class="card-header">
            <h4>Crypto News</h4>
        </div>
        <div class="card-body">
            <h5 class="card-title">Latest Updates in Cryptocurrency</h5>
            <p class="card-text">Catch up with the latest trends, price movements, and expert forecasts in the crypto
            <a href="#" class="btn btn-read-more">Read More</a>
        </div>
    </div>
    <div class="card crypto-news-card mt-4">
        <div class="card-header">
            <h4>Crypto News</h4>
        </div>
        <div class="card-body">
            <h5 class="card-title">Latest Updates in Cryptocurrency</h5>
            <p class="card-text">Catch up with the latest trends, price movements, and expert forecasts in the crypto
            <a href="#" class="btn btn-read-more">Read More</a>
        </div>
    </div>
    <div class="card crypto-news-card mt-4">
        <div class="card-header">
            <h4>Crypto News</h4>
```

Ln 49, Col 1 Spaces: 4 UTF-8 LF HTML Go Live tabnine starter Prettier

templates > dashboard.html > ...

```
        </div>
        <div class="card-body">
            <h5 class="card-title">Latest Updates in Cryptocurrency</h5>
            <p class="card-text">Catch up with the latest trends, price movements, and expert forecasts in the crypto
            <a href="#" class="btn btn-read-more">Read More</a>
        </div>
    </div>
    <div class="card crypto-news-card mt-4">
        <div class="card-header">
            <h4>Crypto News</h4>
        </div>
        <div class="card-body">
            <h5 class="card-title">Latest Updates in Cryptocurrency</h5>
            <p class="card-text">Catch up with the latest trends, price movements, and expert forecasts in the crypto
            <a href="#" class="btn btn-read-more">Read More</a>
        </div>
    </div>
</div>

<style>
    .dashboard-title {
        color: #075937;
        font-size: 2.5rem;
        margin-bottom: 20px;
    }

    .crypto-news-card {
        box-shadow: 0 4px 8px 0 rgba(135, 198, 195, 0.2);
        transition: 0.3s;
        border-radius: 15px;
```

Ln 49, Col 1 Spaces: 4 UTF-8 LF HTML Go Live tabnine starter Prettier

templates > dashboard.html > ...

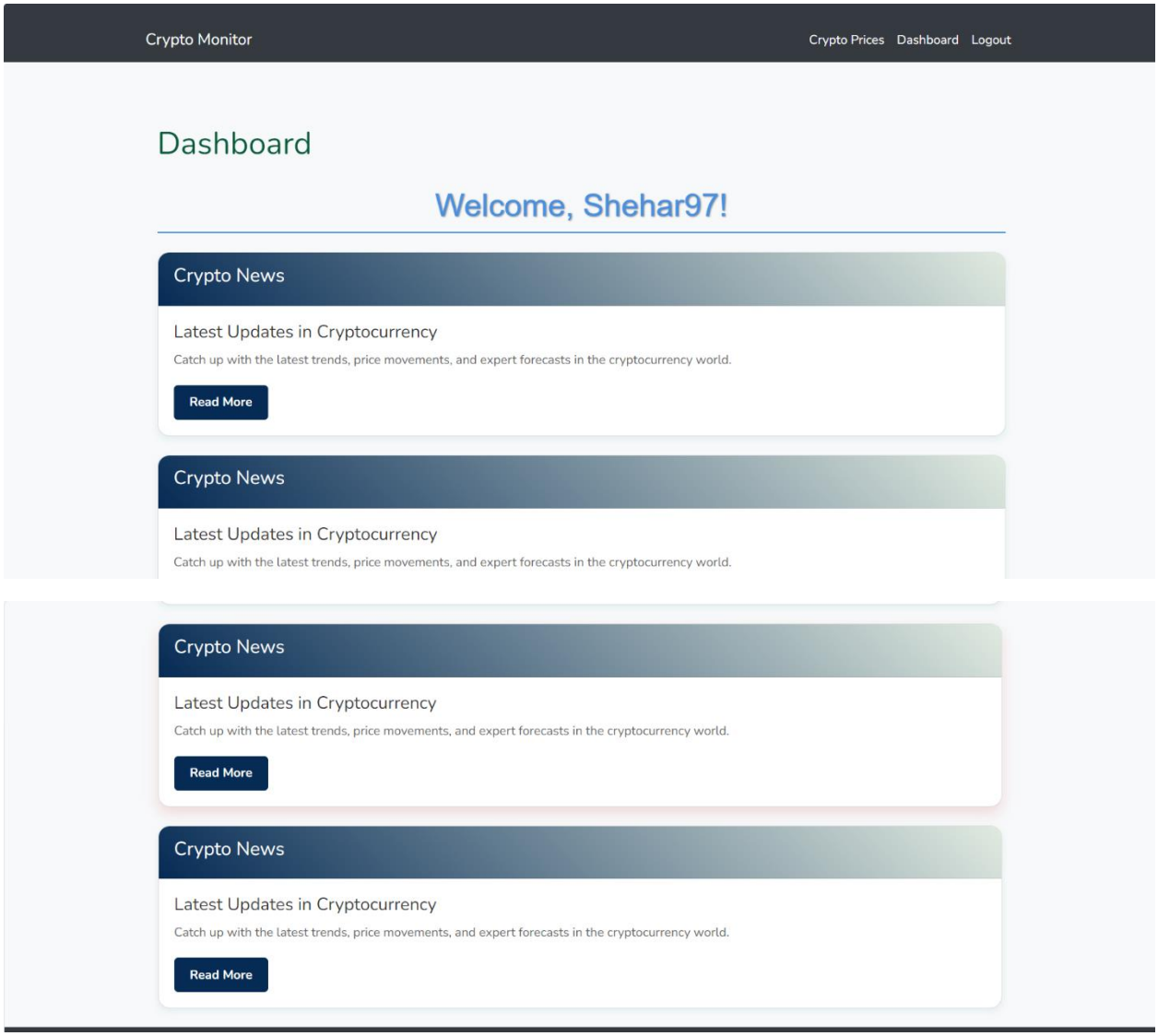
```
61      overflow: hidden;
62  }
63
64  .crypto-news-card:hover {
65      box-shadow: 0 8px 16px 0 rgba(218, 149, 149, 0.3);
66  }
67
68  .card-header {
69      font-weight: bold;
70      background: linear-gradient(45deg, #062b55, #e1ebe1);
71      color: white;
72      padding: 15px 20px;
73      font-size: 1.5rem;
74  }
75
76  .card-body {
77      padding: 20px;
78  }
79
80  .card-title {
81      color: #333;
82      font-size: 1.4rem;
83      margin-bottom: 10px;
84  }
85
86  .card-text {
87      color: #666;
88      font-size: 1rem;
89      margin-bottom: 20px;
90  }
```

Ln 49, Col 1 Spaces: 4 UTF-8 LF HTML Go Live tabnine starter Prettier

templates > dashboard.html > ...

```
92  .btn-read-more {
93      background-color: #062b55;
94      color: white;
95      font-weight: bold;
96      padding: 10px 20px;
97      border: none;
98      border-radius: 5px;
99      transition: background-color 0.3s ease;
100 }
101
102 .btn-read-more:hover {
103     background-color: #8D58BF;
104     color: white;
105     text-decoration: none;
106 }
107 h1 {
108 color: #4A90E2;
109 font-family: 'Arial', sans-serif;
110 font-size: 40px;
111 text-align: center;
112 border-bottom: 2px solid #4A90E2;
113 padding: 10px;
114 margin-bottom: 20px;
115 text-shadow: 1px 1px 2px #888;
116 }
117
118 </style>
119 {% endblock %}
120
```

Output Screenshot:



Explanation Of Code:

This code snippet represents a part of a web page designed for a cryptocurrency dashboard, using HTML enhanced with Jinja2 template syntax for dynamic content rendering. It is likely part of a larger web application, potentially built using a Python web framework like Flask. Here's a detailed breakdown of its components:

Jinja2 Template Inheritance and Content Blocks

1. Template Inheritance: {% extends 'layout.html' %} indicates that this page is extending a base template named layout.html. This means it inherits the structure and elements defined in layout.html and can override or add specific sections.
2. Content Block:

- {% block content %}...{% endblock %} defines a content block. This is where the specific content for this webpage is placed, overriding the corresponding block in the base template.

HTML Content

1. Main Container:
    - A with class container mt-4 serves as the main content area. The classes suggest the use of Bootstrap for styling, with mt-4 adding margin-top spacing.
2. Dashboard Title and User Welcome Message:
    - An <h2> tag for the dashboard title and an <h1> tag welcoming the user. The username is dynamically inserted using {{ username }}, displaying the current user's name.
3. Crypto News Cards:
    - Multiple ** ** elements with class **card crypto-news-card mt-4** create a series of cards displaying cryptocurrency news.
    - Each card has a header (** **) and a body (** **).

CSS Styling

- Custom CSS styles are defined within a **<style>** tag, targeting specific classes to enhance the visual appearance of the dashboard:
    - .dashboard-title styles the dashboard title.
    - .crypto-news-card applies styling to the news cards, including box shadow and hover effects.
    - .card-header and .card-body styles for the header and body of each news card.
    - .btn-read-more styles the 'Read More' buttons.
    - Additional styling for the <h1> tag to enhance the welcome message's appearance.

Summary

This code snippet is part of a web application's user interface, specifically a cryptocurrency dashboard. It uses a combination of HTML for structure, CSS for styling, and Jinja2 template syntax for dynamic content rendering. The page features a welcoming message for the user and a series of news cards related to cryptocurrency, each with a title, description, and a link

for more details. The use of a base template (layout.html) suggests a modular and maintainable approach to web design, common in modern web development.

Layout.html

layout.html

templates > layout.html > html > head > style > footer a:hover

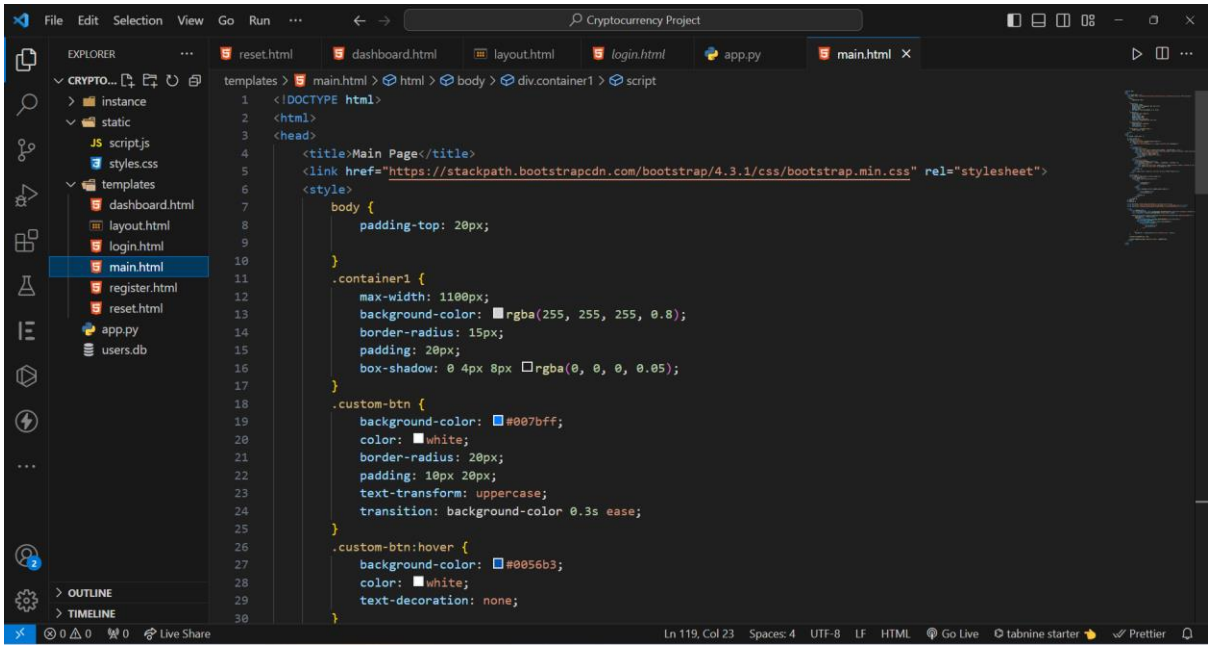
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Flask App</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <link href="https://fonts.googleapis.com/css2?family=Nunito:wght@400;700&display=swap" rel="stylesheet
    <style>
        body {
            font-family: 'Nunito', sans-serif;
            margin: 0;
            color: #333;
            background-color: #f8f9fa;
        }
        .navbar-custom {
            background-color: #343a40;
        }
        .navbar-custom .navbar-brand,
        .navbar-custom .navbar-nav .nav-link {
            color: white;
        }
        .navbar-custom .navbar-brand:hover,
        .navbar-custom .navbar-nav .nav-link:hover {
            color: #f8f9fa;
            text-decoration: none;
        }
        .container {
            padding-top: 20px;
        }
        footer {
            background-color: #343a40;
            color: white;
            padding: 20px 0;
            margin-top: 40px;
        }
        footer a {
            color: #f8f9fa;
        }
        footer a:hover {
            color: #ddd;
            text-decoration: none;
        }

    </style>
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-custom">
        <div class="container">
            <a class="navbar-brand" href="{{ url_for('main_page') }}">Crypto Monitor</a>
```

```
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" a
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarNav">
                <ul class="navbar-nav ml-auto">
                    <li class="nav-item"><a class="nav-link" href="{{ url_for('main_page') }}">Crypto Pric
                    {% if 'loggedin' in session %}
                        <li class="nav-item"><a class="nav-link" href="{{ url_for('dashboard') }}">Dashboa
                        <li class="nav-item"><a class="nav-link" href="{{ url_for('logout') }}">Logout</a>
                    {% else %}
                        <li class="nav-item"><a class="nav-link" href="{{ url_for('login') }}">Login</a></
                        <li class="nav-item"><a class="nav-link" href="{{ url_for('register') }}">Register
                    {% endif %}
                </ul>
            </div>
        </div>
    </nav>

    <div class="container mt-3">
        {% with messages = get_flashed_messages(with_categories=true) %}
        {% if messages %}
        {% for category, message in messages %}
        <div class="alert alert-{{ category }}">
            {{ message }}
        </div>
```

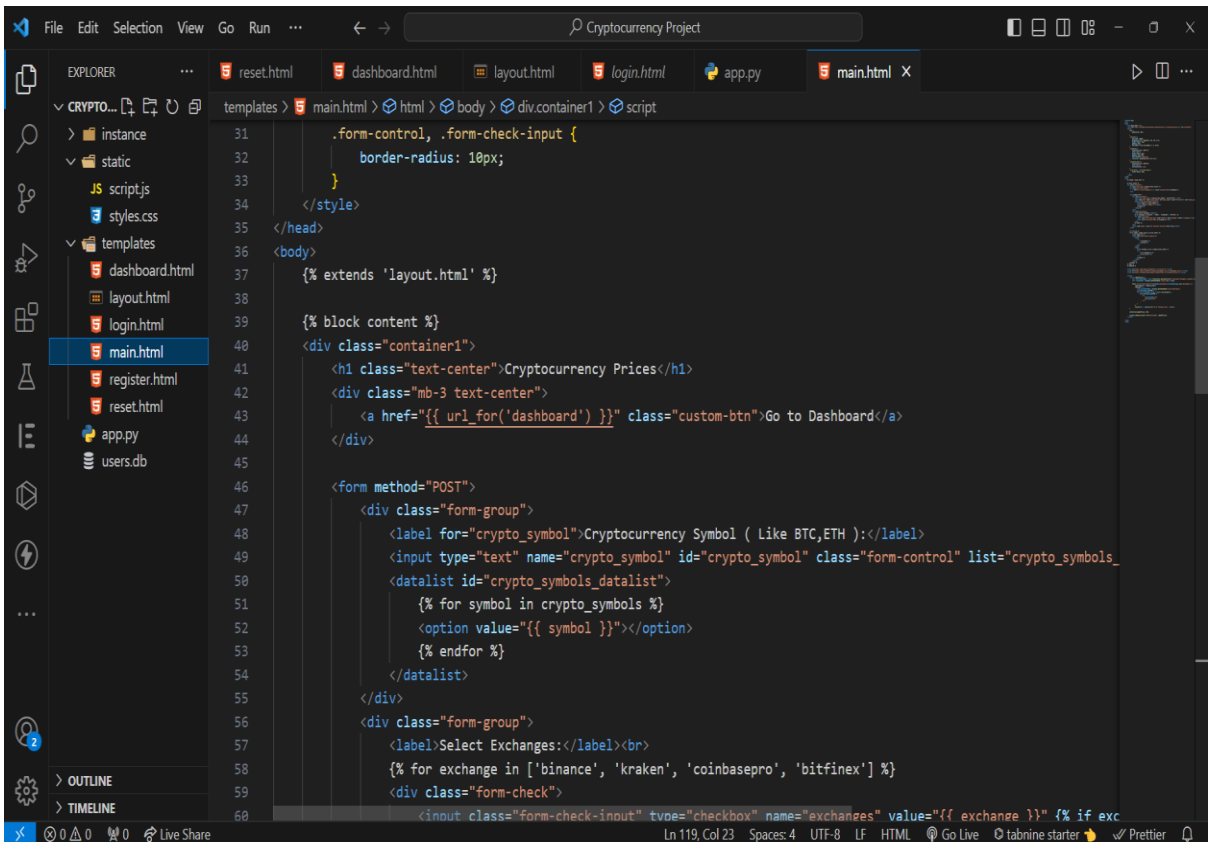
```
        {% endfor %}
        {% endif %}
        {% endwith %}
        {% block content %}{% endblock %}
    </div>

    <script src="                                    om/jquery-3.5.1.slim.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.0.9/dist/umd/popper.min.js"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

    {% include 'footer.html' %}
</body>
</html>
```

Follow link (ctrl + click)

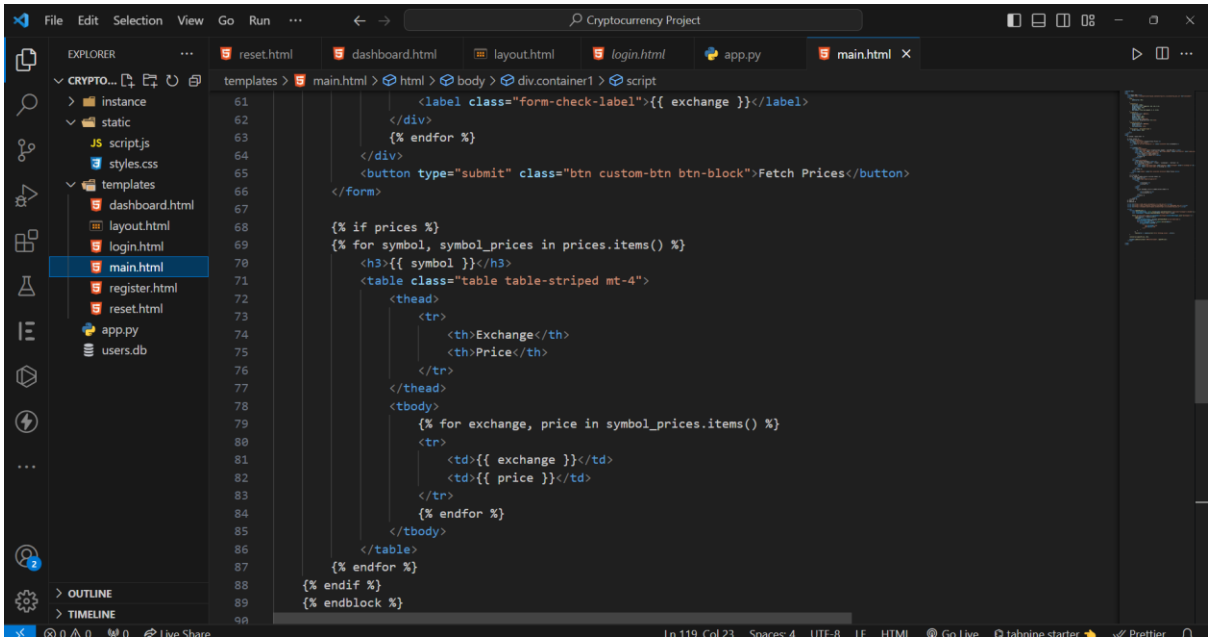
Main.html:

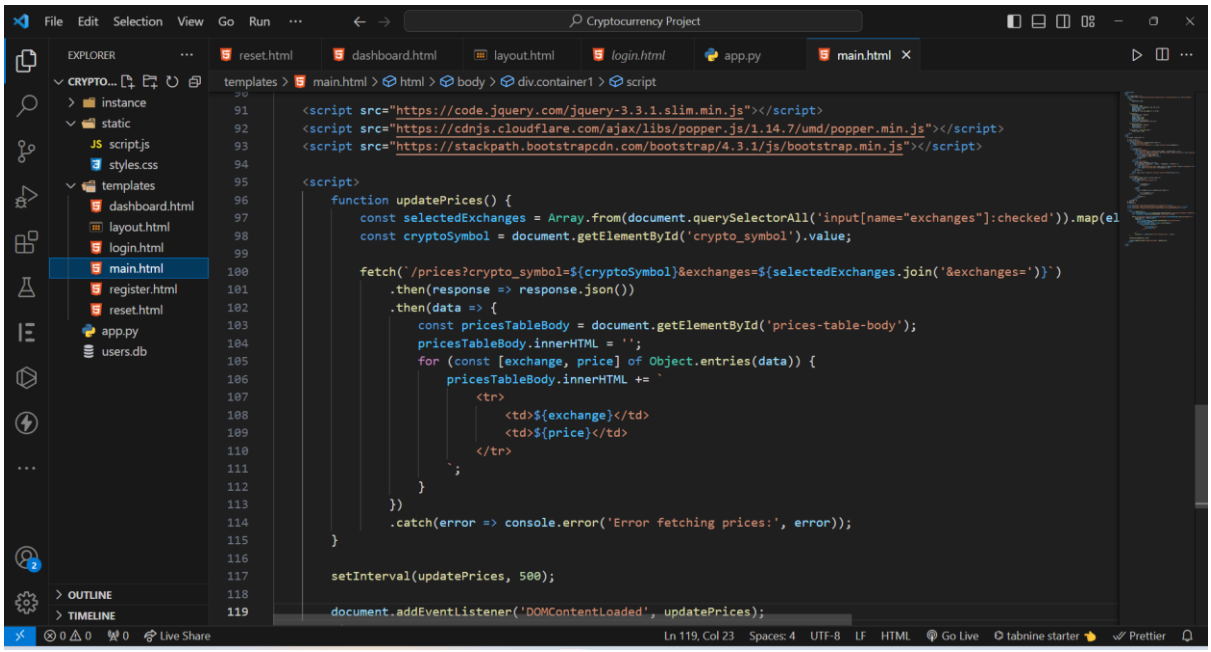



```
<!DOCTYPE html>
<html>
<head>
    <title>Main Page</title>
    <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" rel="stylesheet">
    <style>
        body {
            padding-top: 20px;

        }
        .container1 {
            max-width: 1100px;
            background-color: rgba(255, 255, 255, 0.8);
            border-radius: 15px;
            padding: 20px;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
        }
        .custom-btn {
            background-color: #007bff;
            color: white;
            border-radius: 20px;
            padding: 10px 20px;
            text-transform: uppercase;
            transition: background-color 0.3s ease;
        }
        .custom-btn:hover {
            background-color: #0056b3;
            color: white;
            text-decoration: none;
        }
```




```
        .form-control, .form-check-input {
            border-radius: 10px;
        }
    </style>
</head>
<body>
    {% extends 'layout.html' %}

    {% block content %}
    <div class="container1">
        <h1 class="text-center">Cryptocurrency Prices</h1>
        <div class="mb-3 text-center">
            <a href="{{ url_for('dashboard') }}" class="custom-btn">Go to Dashboard</a>
        </div>

        <form method="POST">
            <div class="form-group">
                <label for="crypto_symbol">Cryptocurrency Symbol ( Like BTC,ETH ):</label>
                <input type="text" name="crypto_symbol" id="crypto_symbol" class="form-control" list="crypto_symbols_
                <datalist id="crypto_symbols_datalist">
                    {% for symbol in crypto_symbols %}
                    <option value="{{ symbol }}"></option>
                    {% endfor %}
                </datalist>
            </div>
            <div class="form-group">
                <label>Select Exchanges:</label><br>
                {% for exchange in ['binance', 'kraken', 'coinbasepro', 'bitfinex'] %}
                <div class="form-check">
                    <input class="form-check-input" type="checkbox" name="exchanges" value="{{ exchange }}" {% if exc
```





This code represents a complete HTML document structured to display a webpage for tracking cryptocurrency prices. It is designed for integration with a backend server, likely using a Python web framework like Flask, as indicated by the Jinja2 template syntax (**{% ... %}**). Here's a breakdown of its various parts:

HTML Structure

1. Doctype and HTML Tags: The document begins with <!DOCTYPE html> indicating an HTML5 document, followed by the standard <html> tag.
2. Head Section:
    - <head> contains metadata and links to external resources.
    - <title> sets the page's title to "Main Page".
    - A link to Bootstrap's CSS for styling.
    - Custom CSS styles are defined within a <style> tag, styling the body, a container (container1), buttons (custom-btn), and form elements.
3. Body Section:
    - The <body> section contains the visible content of the webpage.
    - {% extends 'layout.html' %} suggests inheritance from a base layout (layout.html), a common practice in web frameworks for consistent design.
    - {% block content %}...{% endblock %} defines a section for specific content in child templates.

Content

1. Container for Cryptocurrency Prices:
    - A div with class container1 provides a styled area for content.
    - Inside this container, there's a heading (<h1>) for "Cryptocurrency Prices" and a button linking to a dashboard page.
2. Form for Selecting Cryptocurrencies and Exchanges:
    - The form uses POST method, sending data back to the server.
    - A dropdown (<datalist>) for selecting cryptocurrency symbols, populated dynamically using template looping over crypto_symbols.
    - Checkboxes for selecting exchanges, also populated dynamically.
3. Display of Cryptocurrency Prices:
    - This section displays if prices is available. It loops over prices, creating a table for each cryptocurrency symbol showing prices from different exchanges.

Scripts

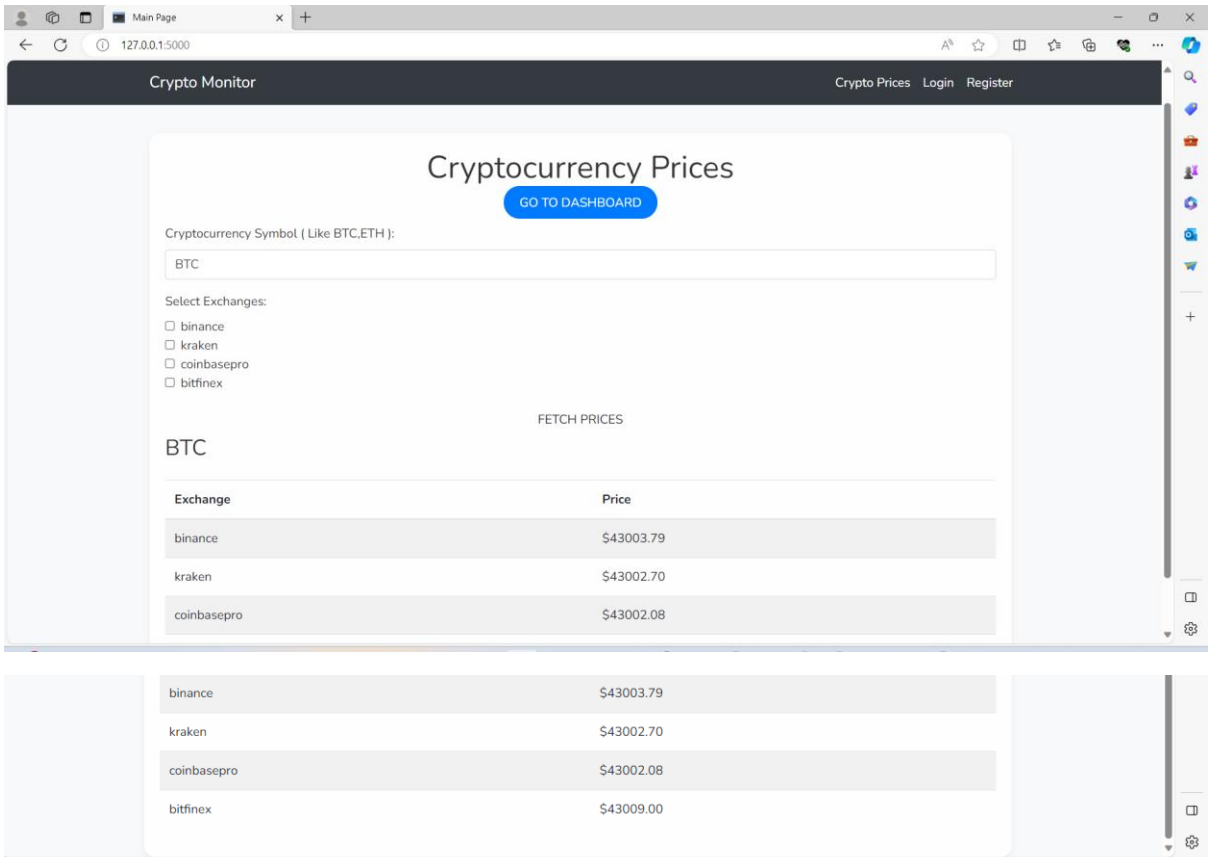
1. Bootstrap and jQuery Scripts: Links to jQuery and Bootstrap's JS files are included at the end of the body.
2. JavaScript for Updating Prices:
    - A script for fetching updated prices from the server asynchronously using fetch.
    - updatePrices function sends a request to /prices endpoint with selected exchanges and the cryptocurrency symbol, updating the page with new prices.
    - setInterval(updatePrices, 500) calls updatePrices every 500 milliseconds for real-time updates.
    - updatePrices is also called once the DOM content is fully loaded.

Jinja2 Template Syntax

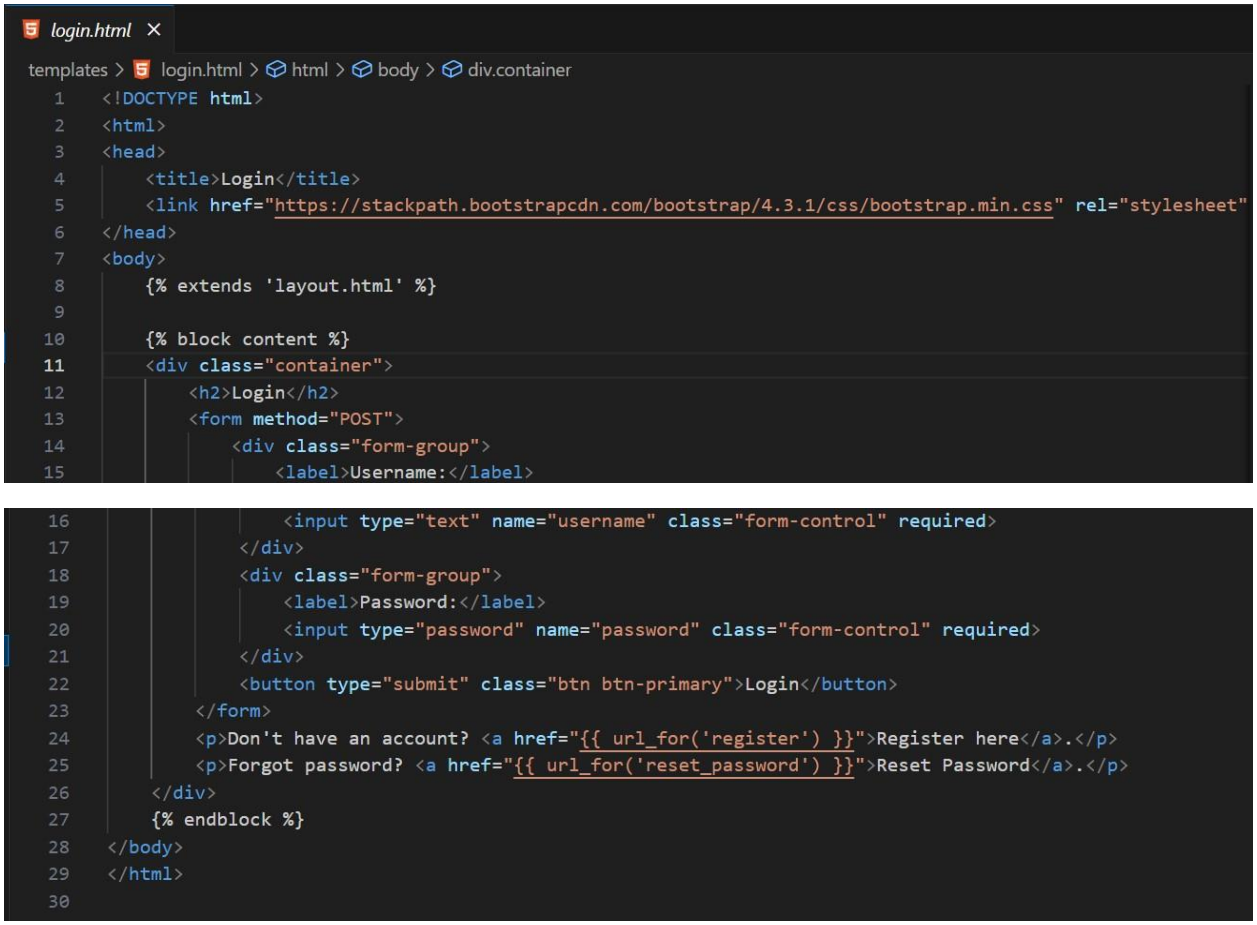
- {% ... %} and {{ ... }} indicate Jinja2, a template engine used in Python web frameworks like Flask for server-side rendering.

Summary

This code represents a dynamic webpage for displaying cryptocurrency prices, utilizing Bootstrap for styling and JavaScript for real-time updates. The page allows users to select different cryptocurrencies and exchanges to view up-to-date price information, serving as a tool for those interested in cryptocurrency markets.



Login.html:



```
login.html
templates > login.html > html > body > div.container
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
    <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" rel="stylesheet"
</head>
<body>
    {% extends 'layout.html' %}

    {% block content %}
    <div class="container">
        <h2>Login</h2>
        <form method="POST">
            <div class="form-group">
                <label>Username:</label>
                <input type="text" name="username" class="form-control" required>
            </div>
            <div class="form-group">
                <label>Password:</label>
                <input type="password" name="password" class="form-control" required>
            </div>
            <button type="submit" class="btn btn-primary">Login</button>
        </form>
        <p>Don't have an account? <a href="{{ url_for('register') }}">Register here</a>.</p>
        <p>Forgot password? <a href="{{ url_for('reset_password') }}">Reset Password</a>.</p>
    </div>
    {% endblock %}
</body>
</html>
```

This document outlines the structure and functionality of an HTML login page template designed for integration with web applications using a templating engine, for instance, Jinja2 in conjunction with the Flask framework. The template incorporates elements from the Bootstrap CSS framework to ensure a responsive and aesthetically pleasing user interface.

Overview of the Template Structure

Document Type Declaration

- The declaration <!DOCTYPE html> indicates that the document adheres to the HTML5 standard.

HTML Root Element

- Encapsulates the head and body sections of the document, structuring the webpage's foundational elements.

Head Section Elements

- Webpage Title: Defined within the <title> tag, this specifies the webpage's title as "Login", which is displayed in the browser's title bar or tab.

- External Stylesheet Reference: A <link> tag is employed to incorporate the Bootstrap 4.3.1 CSS framework from a CDN, enabling the utilization of predefined styles for UI components and ensuring responsiveness across various devices.

Body Section Composition

- Template Extension: Utilizing {% extends 'layout.html' %}, the template inherits from a base layout, layout.html, which likely includes common webpage elements like navigation bars.
- Dynamic Content Block: The {% block content %} ... {% endblock %} syntax denotes a dynamic section where specific content for each page is inserted, allowing for the customization of this segment within the inherited layout structure.
- Content Container: A div element with the container class wraps around the form and textual elements, leveraging Bootstrap's grid system for optimal spacing and alignment.

Login Form Details

- Form Submission Method: The form element specifies a method="POST" attribute, indicating the submission of form data via an HTTP POST request, encapsulating the user's credentials in the request body.
- Form Elements:
    - Username Input: A form group containing a label and a text input field for the username, marked with the form-control class for Bootstrap styling.
    - Password Input: Similar to the username input but for the password, with the input field's type set to "password" to obscure the input text.
    - Submission Button: A button tagged with class="btn btn-primary", signifying the primary action button styled by Bootstrap, is provided to submit the form data.

Auxiliary Links

- Registration and Password Reset:
    - Text links offer navigation to registration and password reset pages, employing Flask's {{ url_for('register') }} and {{ url_for('reset_password') }} for dynamic URL generation, facilitating user navigation for account creation or recovery.

This template ingeniously marries Flask's dynamic content rendering and URL management with Bootstrap's responsive design principles, ensuring a user-friendly login interface. It exemplifies modern web development best practices, such as accessibility considerations, template inheritance for code efficiency, and responsive design for cross-device compatibility.

Reset.html:

reset.html

templates > reset.html > html > body

```
<html>
<head>
    <title>Reset Password</title>
    <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" rel="stylesheet"
</head>
<body>
    {% extends 'layout.html' %}

    {% block content %}
    <div class="container">
        <h2>Reset Password</h2>
        <form method="POST">
            <div class="form-group">
                <label>Username:</label>
                <input type="text" name="username" class="form-control" required>
            </div>
            <div class="form-group">
                <label>New Password:</label>
                <input type="password" name="new_password" class="form-control" required>
            </div>
            <button type="submit" class="btn btn-primary">Reset Password</button>
        </form>
    </div>
    {% endblock %}
</body>
</html>
```

This document is an HTML template for a password reset page, intended for use in a web application that incorporates a template engine, such as Jinja2, commonly paired with the Flask Python framework. The template is structured to inherit from a base layout (layout.html) and leverages the Bootstrap CSS framework to ensure the page is visually appealing and responsive across a range of devices.

Document Structure Overview

HTML Document Declaration

- Begins with <!DOCTYPE html>, declaring the document type as HTML5, ensuring modern web standards are followed.

HTML Head Section

- Page Title: Set within the <title> element, indicating the purpose of the page as "Reset Password".
- External CSS Link: A <link> element imports the Bootstrap CSS framework (version 4.3.1) from a CDN, enabling standardized styling and responsive design features for the page's elements.

Body Section

- Template Inheritance: By using {% extends 'layout.html' %}, the template inherits common structural elements from a master layout, promoting consistency and reducing duplication across the web application.
- Content Block: Enclosed by {% block content %}...{% endblock %}, this section is designated for page-specific content, allowing for the insertion of the password reset form within the inherited layout.
- Container Element: A div with the container class encapsulates the form, leveraging Bootstrap's grid system to manage the layout effectively, ensuring the content is appropriately spaced and aligned.

Password Reset Form

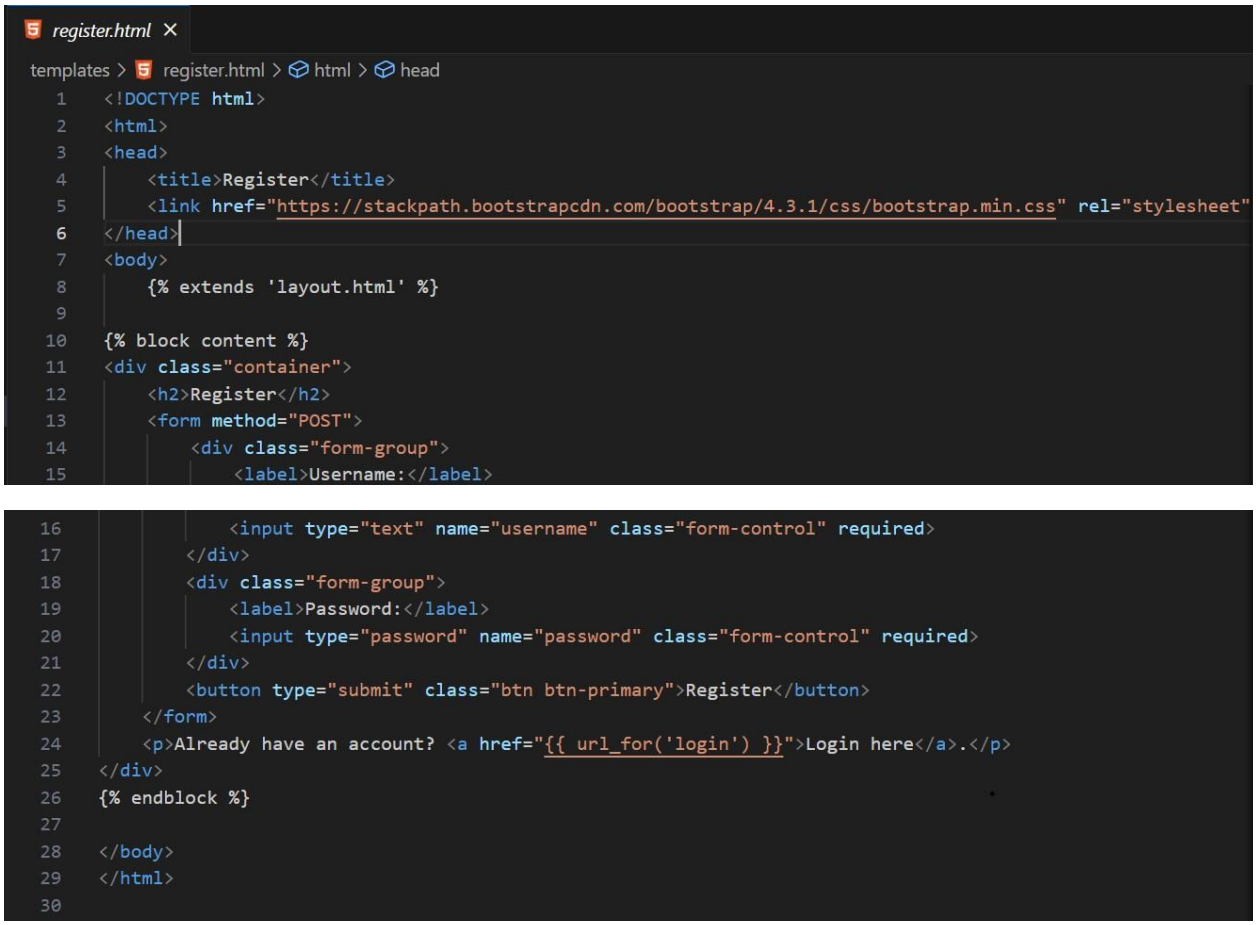
- Form Configuration: The form element is set to submit data via the POST method, a secure choice for transmitting sensitive information such as passwords.
- User Input Fields:
    - Username: A form group contains a label and a text input for the username, essential for identifying the user account in need of a password reset. The input is styled with Bootstrap's form-control class for consistency.
    - New Password: Similarly, this form group provides a label and a password input for the new password, ensuring the input is obscured for privacy. This input also receives the form-control styling.
- Submission Button: A submit button, styled with btn and btn-primary classes from Bootstrap, enables the user to submit the form, initiating the password reset process.

Functional and Stylistic Highlights

This template combines practical web development techniques with aesthetic considerations, ensuring a user-friendly interface for password reset requests. It demonstrates the effective use of template inheritance for maintaining a unified look and

feel across the web application, the strategic deployment of Bootstrap for responsive design, and the secure handling of sensitive information through form submission methods. The inclusion of essential user input fields, styled consistently and laid out for easy navigation, reflects a thoughtful approach to user experience design.

Register.html:



```
register.html
templates > register.html > html > head
<!DOCTYPE html>
<html>
<head>
    <title>Register</title>
    <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" rel="stylesheet"
</head>
<body>
    {% extends 'layout.html' %}

{% block content %}
<div class="container">
    <h2>Register</h2>
    <form method="POST">
        <div class="form-group">
            <label>Username:</label>
            <input type="text" name="username" class="form-control" required>
        </div>
        <div class="form-group">
            <label>Password:</label>
            <input type="password" name="password" class="form-control" required>
        </div>
        <button type="submit" class="btn btn-primary">Register</button>
    </form>
    <p>Already have an account? <a href="{{ url_for('login') }}">Login here</a>.</p>
</div>
{% endblock %}

</body>
</html>
```

This HTML template is crafted for a registration page within a web application that utilizes a templating engine, such as Jinja2, which is often paired with Flask, a Python web framework. The template inherits from a base layout, **layout.html**, ensuring a consistent look and feel across the site, and uses Bootstrap for styling to create a user-friendly and responsive interface.

Document Structure and Content

HTML5 Declaration

- The <!DOCTYPE html> declaration specifies that this document conforms to HTML5 standards.

Head Section

- Page Title: The <title> tag sets the window or tab title to "Register", clearly indicating the page's purpose.
- Bootstrap CSS: Through a <link> element, the page includes Bootstrap 4.3.1 from a content delivery network (CDN), allowing for quick access to a vast library of styles that enhance the appearance and responsiveness of the registration form and other page elements.

Body and Layout

- Template Inheritance: The {% extends 'layout.html' %} directive signifies that this page is a child template, inheriting styles, scripts, and structural elements from a master layout file. This promotes reusability and ensures uniformity across different pages of the application.
- Content Block: Enclosed within **{% block content %}...{% endblock %}**, this section is designated for the content unique to the registration page, allowing for flexibility and customization within the predefined layout structure.
- Container: A div with the Bootstrap container class wraps the registration form and informational text, providing a visually appealing and structured layout that adapts to different screen sizes.

Registration Form

- Form Setup: The form uses the POST method to securely transmit user-entered data to the server. This method is preferred for submitting sensitive information, such as passwords, because it does not expose data in the URL.
- User Input Fields:
    - Username: Includes a form group with a label and a text input for the username. This field is essential for creating a unique identifier for each user. It is styled with Bootstrap's form-control for consistency and usability.
    - **Password**: Similar to the username field, this includes a label and a password input, ensuring user privacy by hiding input characters. It also uses the form-**control** class for styling.
- Submission Button: A button element styled with Bootstrap's btn and btn-primary classes is used for submitting the form, encouraging user interaction with a visually distinctive and attractive button.

Additional Navigation

- Link to Login Page: A paragraph offers a link for users who already have an account, directing them to the login page. This is facilitated by Flask's {{ url_for('login') }}, dynamically generating the appropriate URL, which enhances user experience by providing easy navigation between related pages.

This template exemplifies effective web development practices by combining Flask's server-side rendering capabilities with Bootstrap's client-side styling. It effectively creates a registration page that not only looks professional and is easy to use but also promotes a secure and efficient method for user data submission. The inclusion of dynamic URL generation for navigation further underscores the application's focus on a seamless user experience.

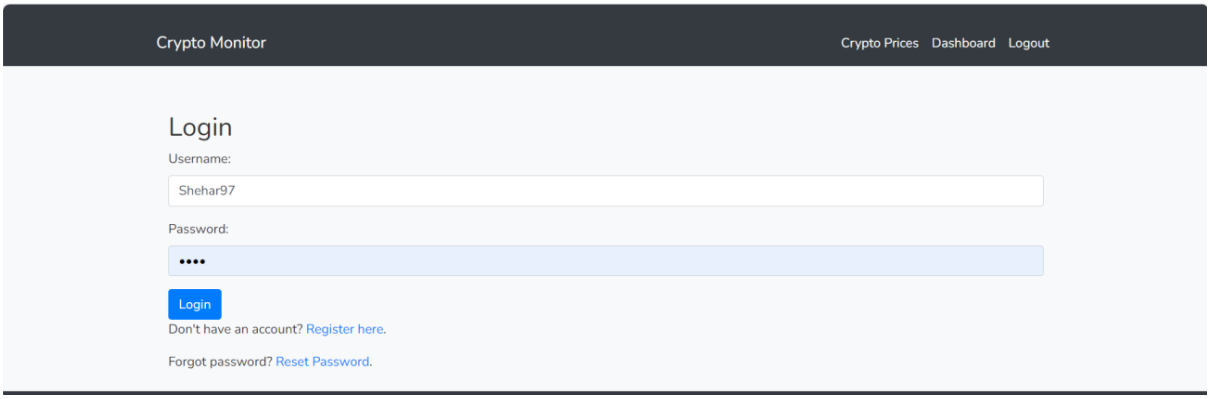
Evaluation:

Function and Robustness Testing

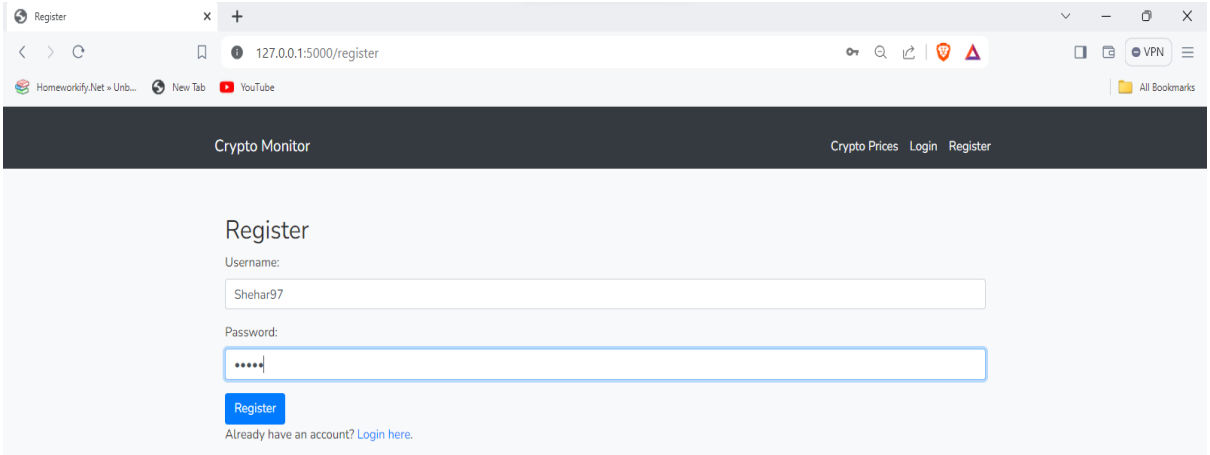
Test Evidence:

1. Login and Registration Workflow: The application successfully handles user registration, login, and session management. SQLite database integration ensures user credentials are securely managed.

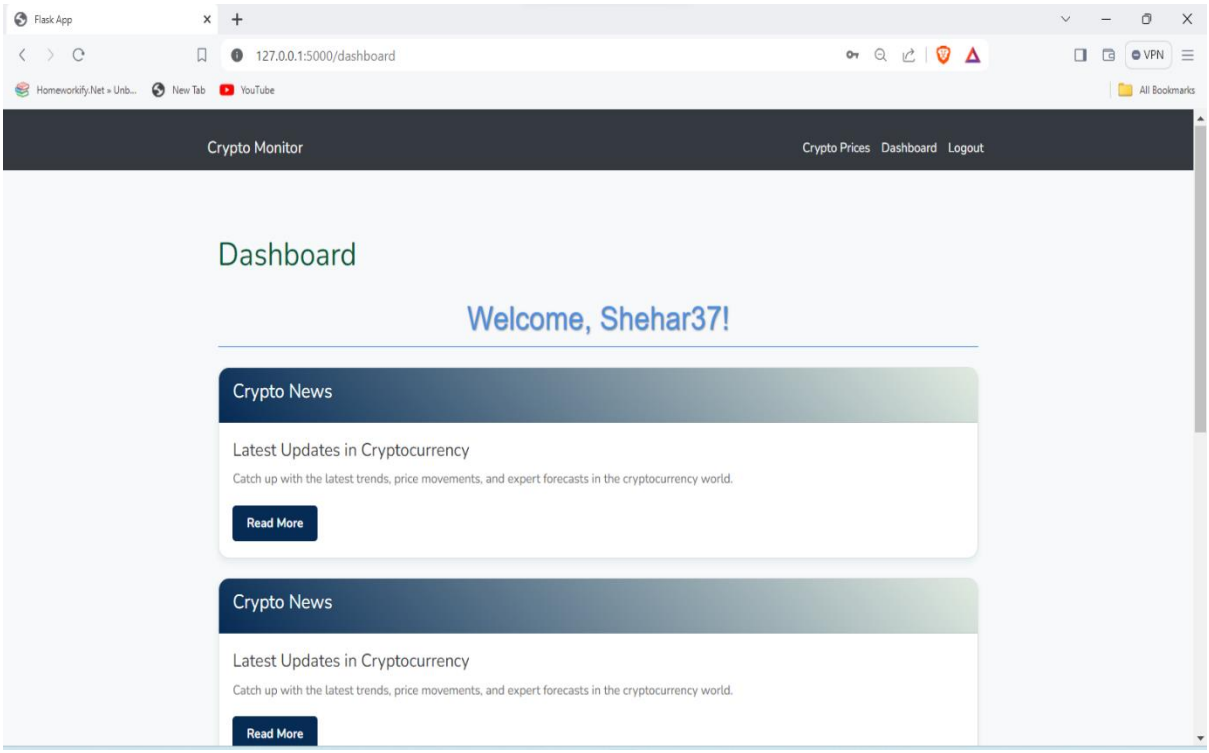
   Login form:

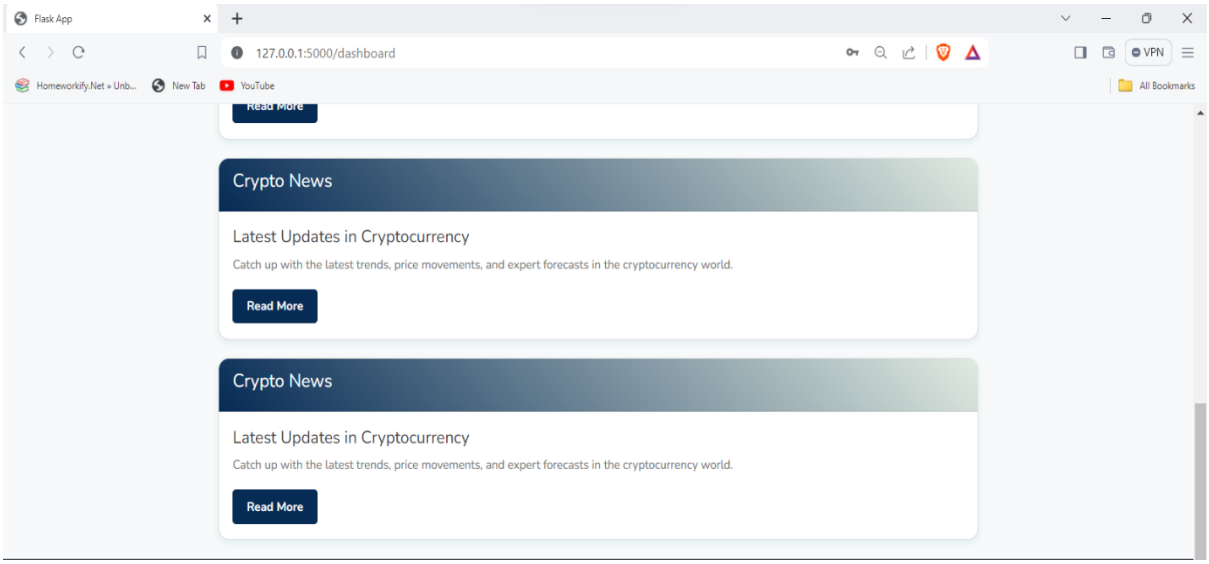


   Registration form:

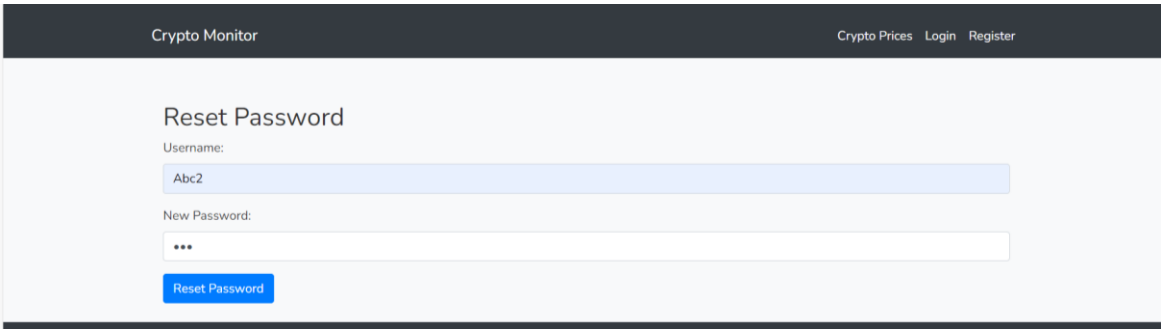


   It then leads to the dashboard.





2. Password Reset Functionality: This feature is robust, allowing users to securely update their credentials.



3. Dashboard Access and Display: Post-login, the dashboard displays cryptocurrency news and personalized greetings, functioning as expected.
4. Crypto Data Fetching and Display: The integration with **ccxt** library and selected exchanges (Binance, Kraken, etc.) works effectively, fetching real-time crypto prices.

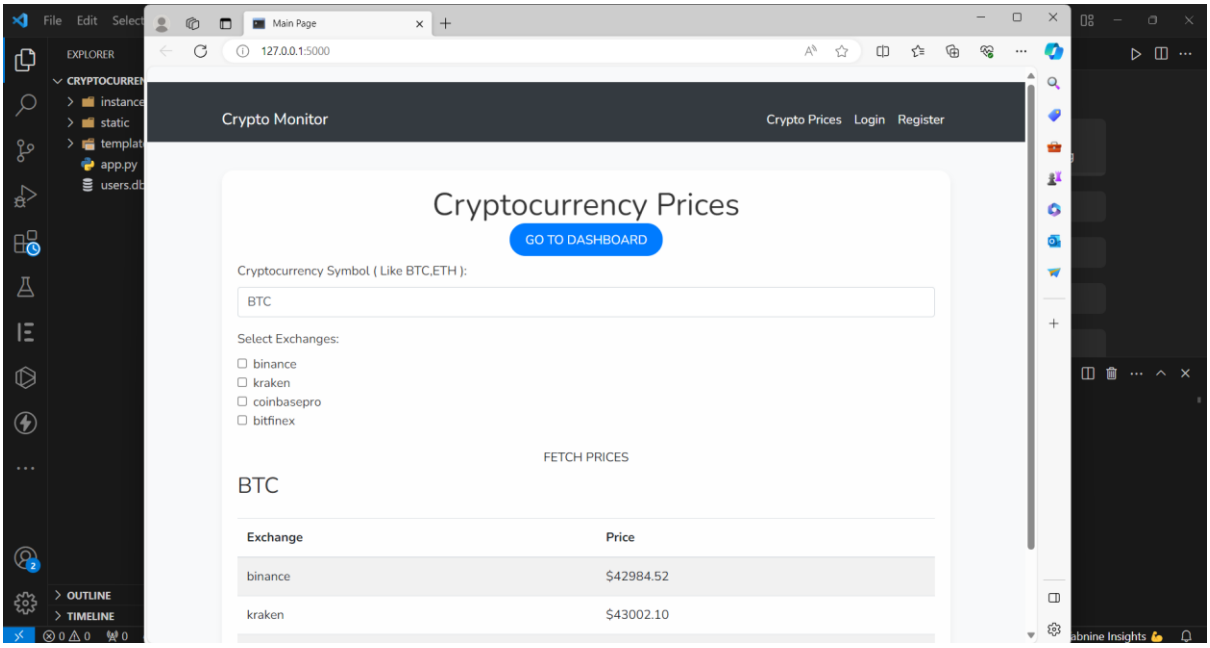
Robustness:

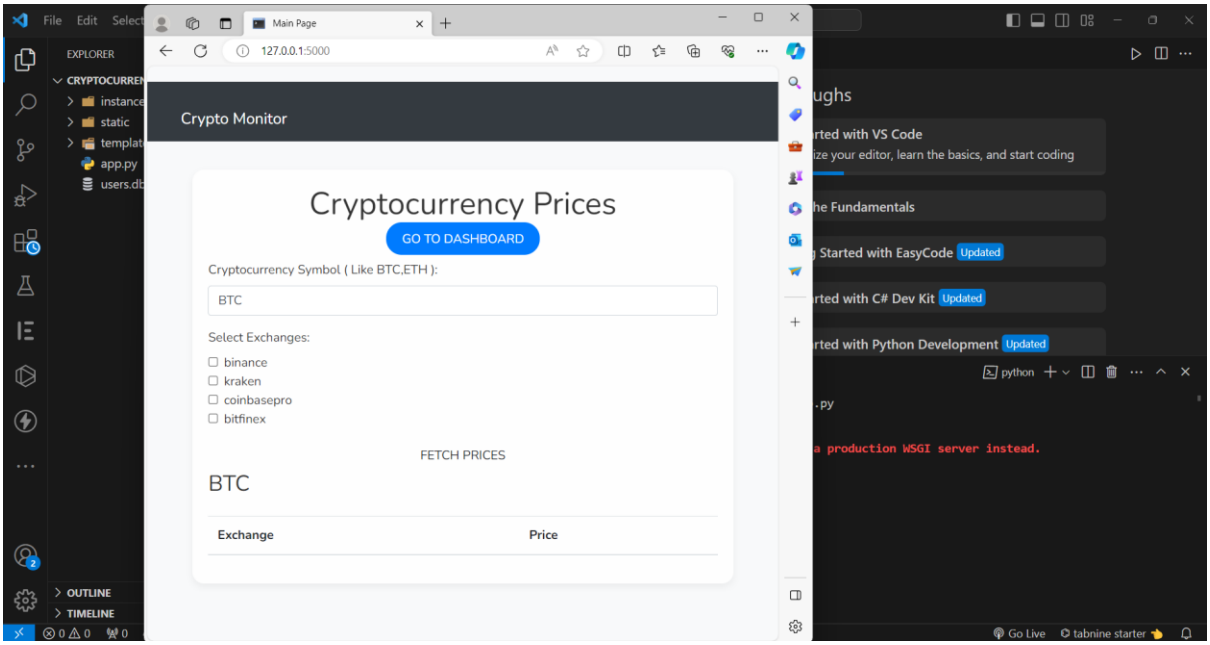
- The application handles errors gracefully, providing user feedback via flash messages.
- Database operations are encapsulated within try-except blocks to prevent runtime errors.

Usability Testing

Test Evidence:

1. Ease of Navigation: The layout is intuitive, with clear navigation options in the navbar.
2. Responsiveness: Bootstrap framework ensures the application is responsive across various devices.



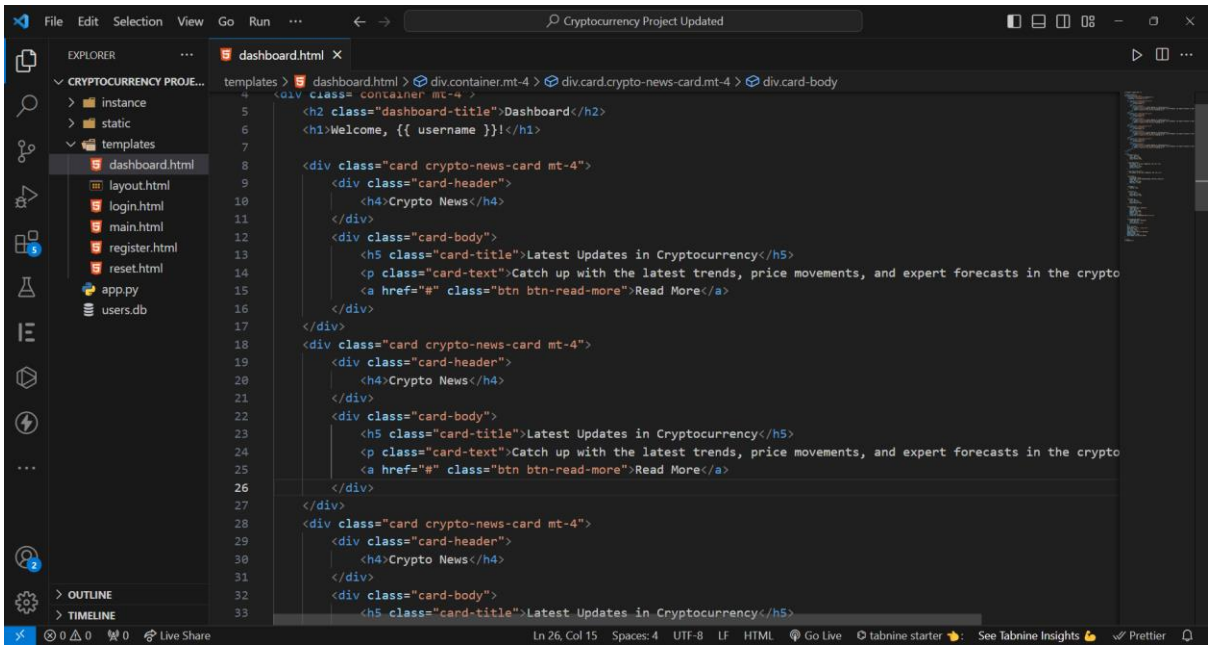


3. Readability and Aesthetics: Aesthetic elements like color schemes, font choices, and card layouts enhance user experience.

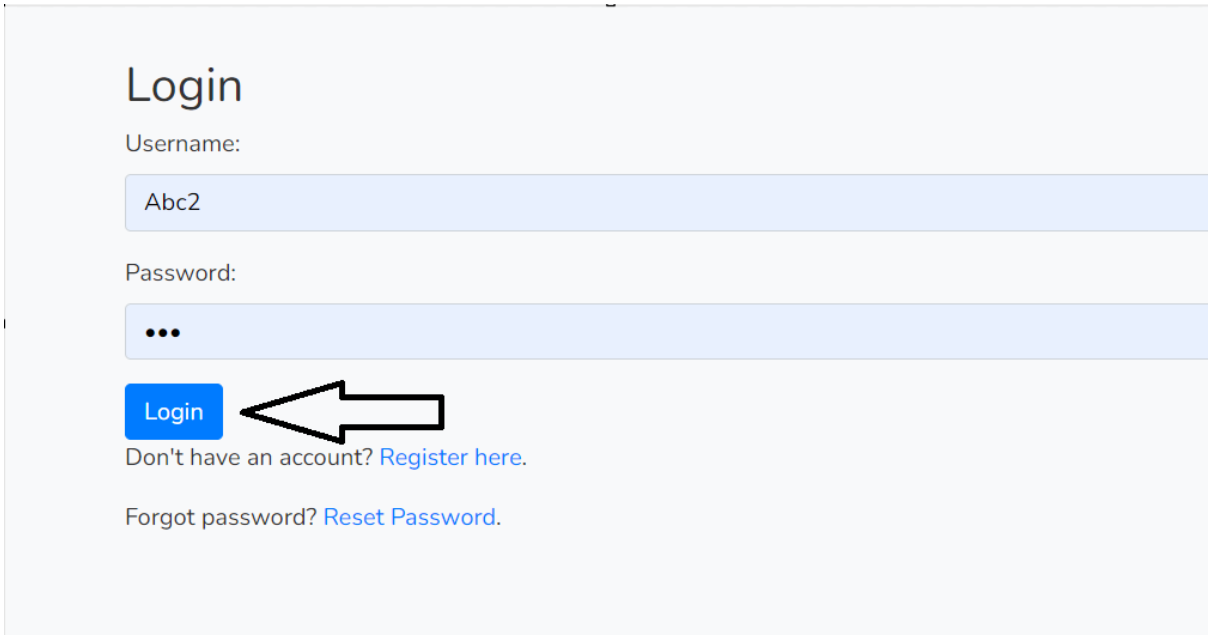
Usability Features Justification:

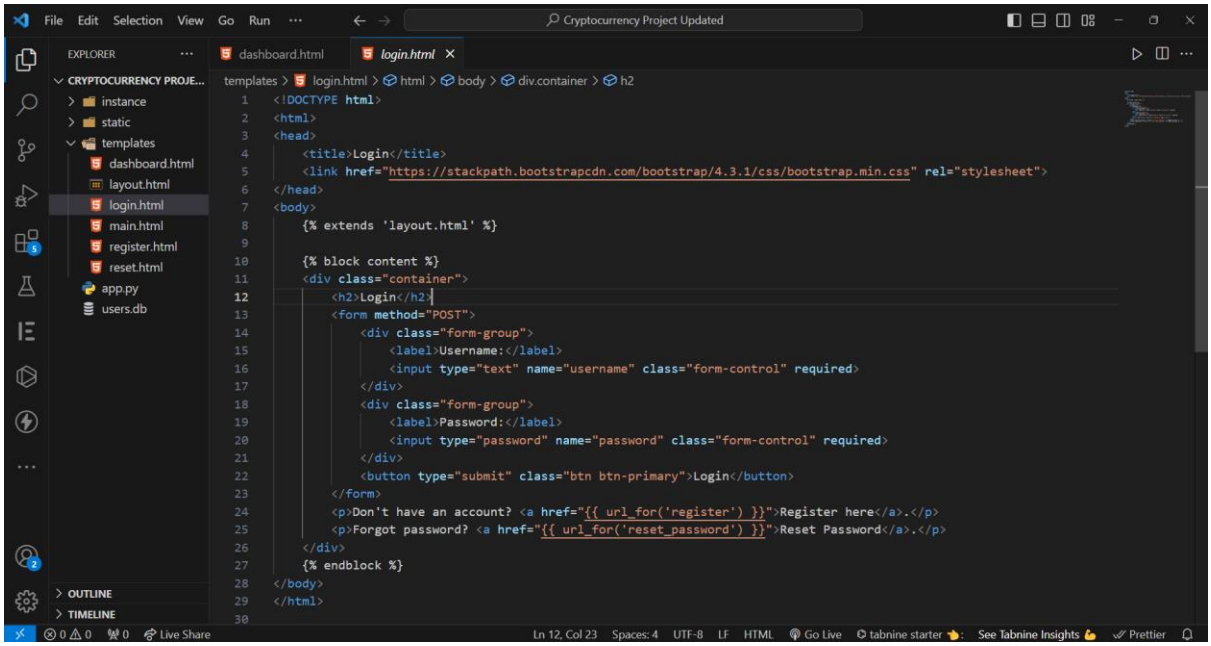
- Feature Name: Simplified User Interface
- Purpose: To make the application or website easy to use for everyone, including those not familiar with technology.
- Explanation: This design focuses on essential features, using clean lines, ample space, and a clear layout to avoid confusion and help users navigate easily. It aims to create a comfortable and straightforward environment for all users.

CODE:



- Feature Name: Interactive Elements with Immediate Feedback
- Purpose: To engage users and inform them about the results of their actions instantly.
- Explanation: Elements like buttons and forms are designed to respond immediately to user interactions. This feedback helps users understand their actions' impact, reducing confusion and enhancing the overall user experience.





Register

Username:

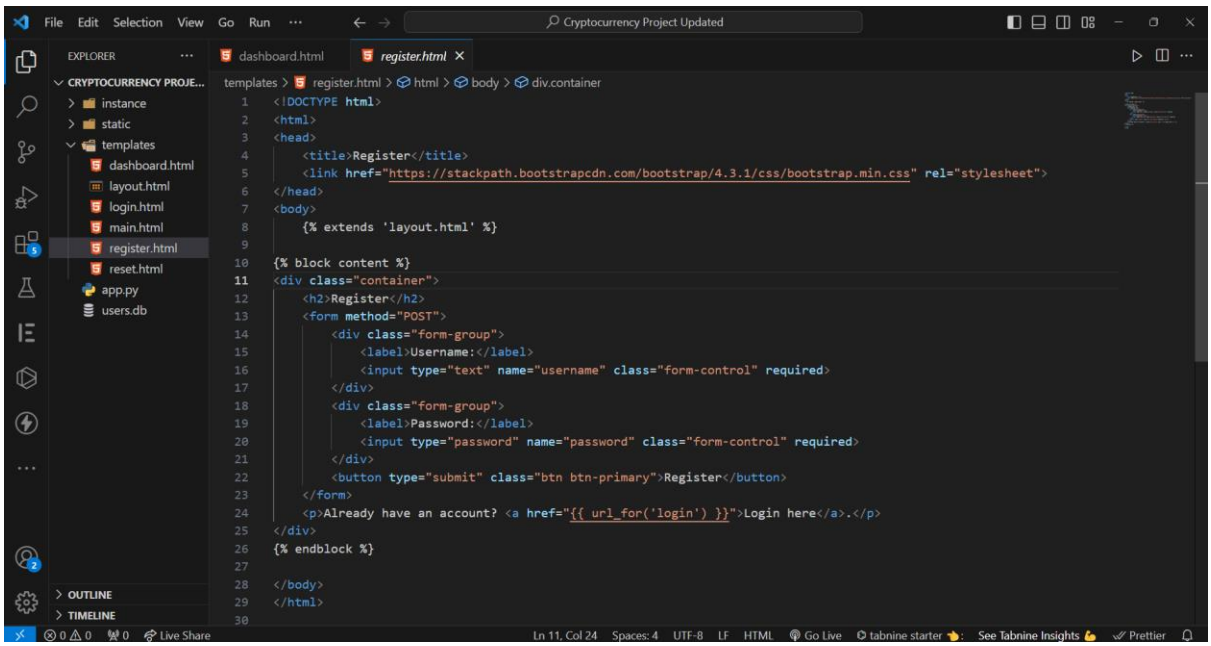
Abc2

Password:

•••

Register

Already have an account? Login here.

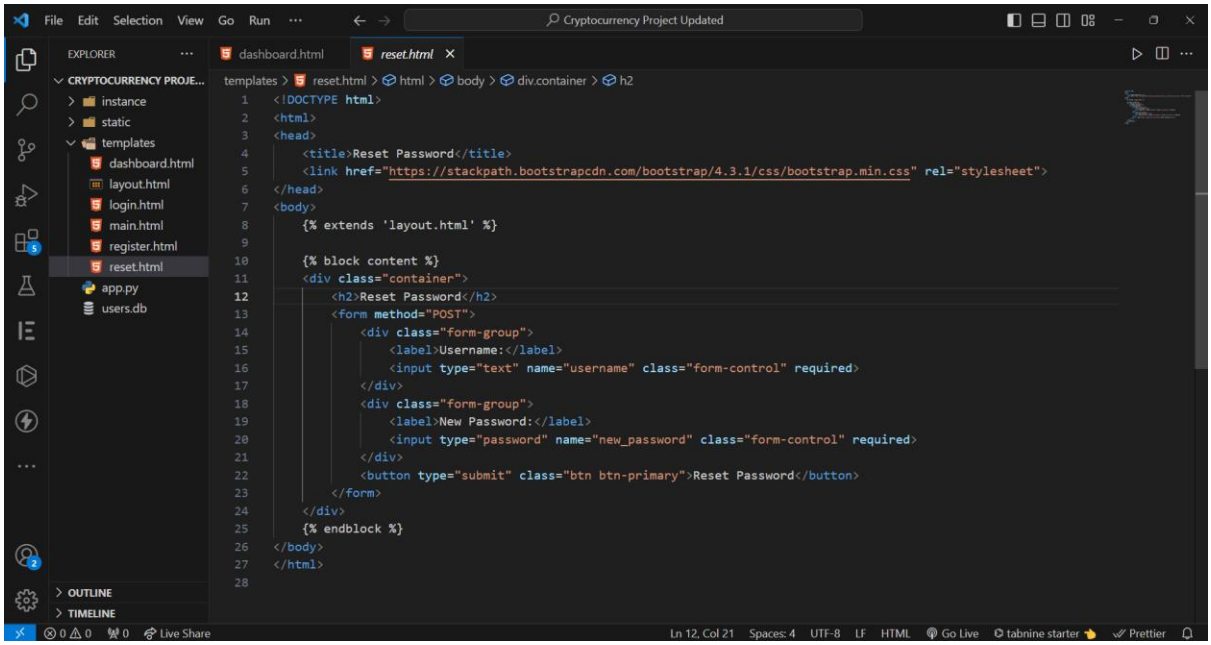


Reset Password

Username:

New Password:

Reset Password



Maintenance and Limitations

1. Maintenance Issues:

- Dependency on External APIs:

The application must frequently update to stay compatible with external APIs like the ccxt library and cryptocurrency exchanges. Changes in these services require prompt adjustments to maintain functionality.

- Database Scalability:

SQLite, used for its simplicity, may struggle with scalability for larger user bases or data volumes, potentially necessitating a migration to a more robust database system for improved performance and concurrency support.

2. Limitations:

- Lack of Advanced User Preferences or Customization:

The application does not offer extensive customization options, limiting users' ability to personalize their experience, which could hinder user engagement and satisfaction.

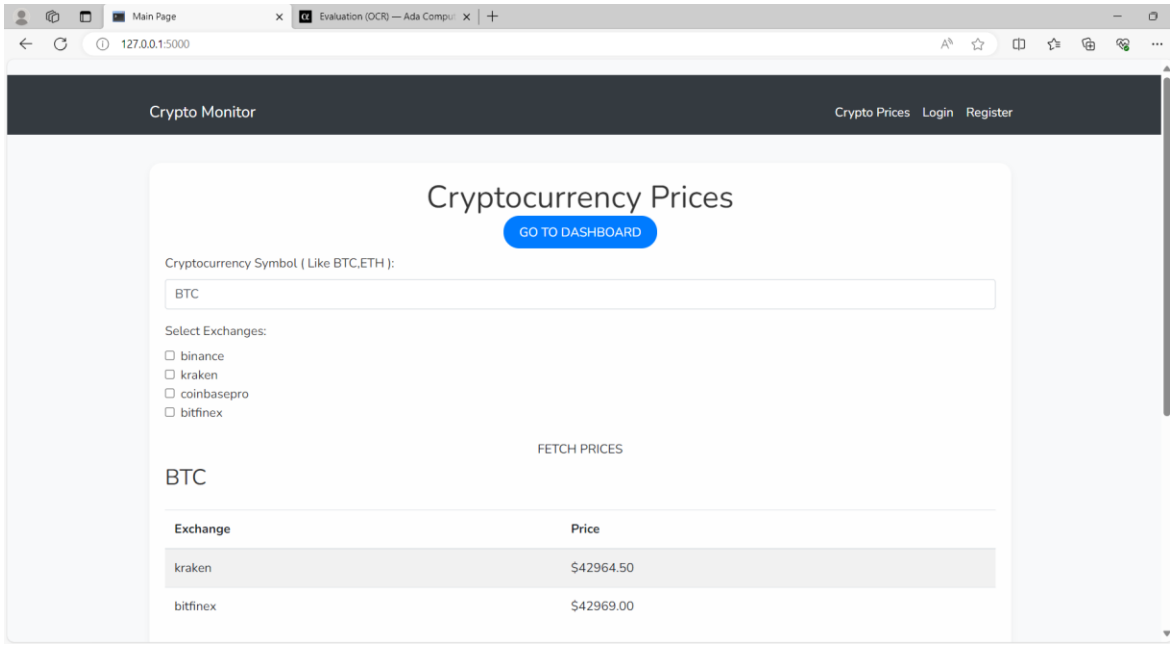
- Static Crypto News Section:

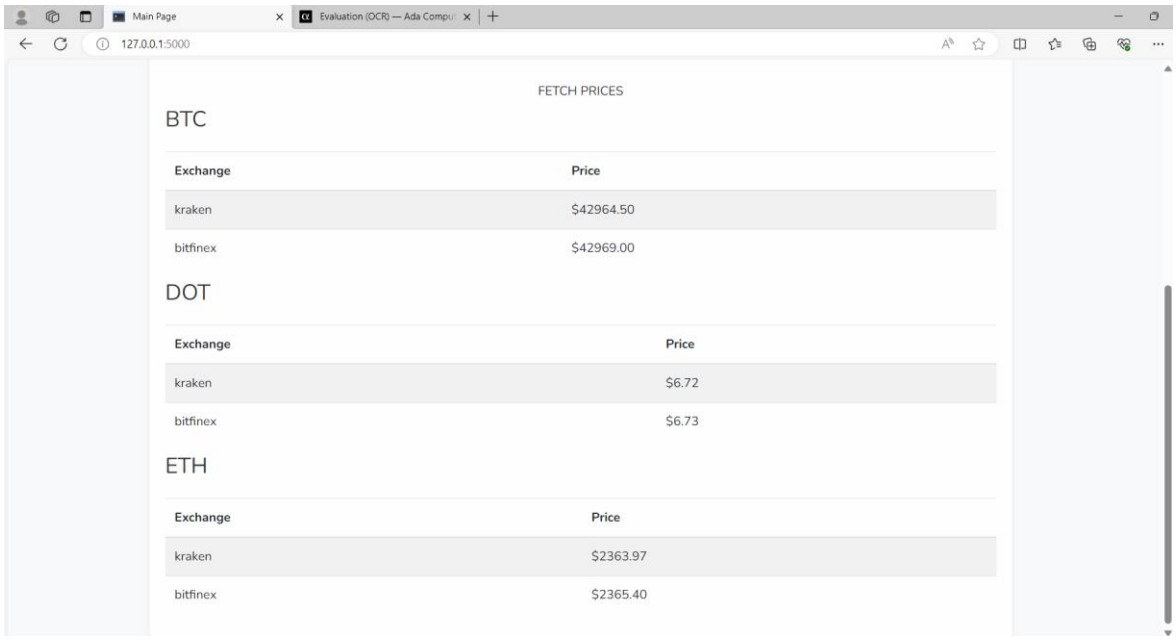
The crypto news section lacks dynamic updates, making it less useful for users seeking current information. Integrating a real-time news API could make this feature more relevant and enhance the overall user experience.

Independent Feedback

Feedback Summary:

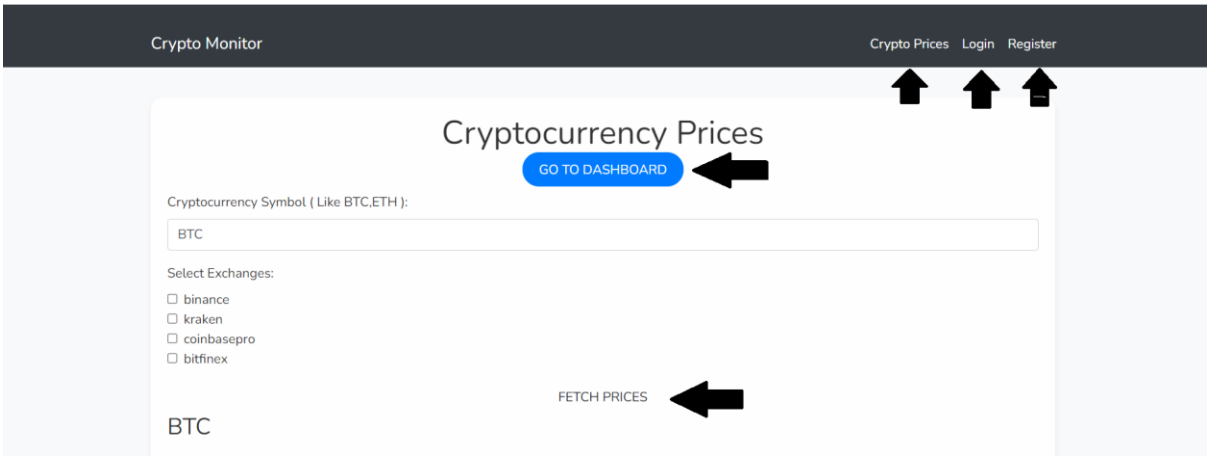
- Users appreciated the clean interface and straightforward functionality.
- Suggestions included implementing more interactive elements on the dashboard and providing more detailed crypto market analytics.





Improvement Considerations

1. Enhancing Dashboard Features: Including interactive charts and more comprehensive market analytics could significantly improve user engagement.



2. Scalability: Migrating to a more scalable database like PostgreSQL would improve performance.
3. User Customization: Allowing users to set preferences for news, alerts, and market updates would personalize the experience.
4. Security Enhancements: Implementing two-factor authentication for user accounts would enhance security.

Conclusion:

In conclusion, this Flask web application offers a streamlined and secure platform for users to interact with cryptocurrency data. The application's architecture efficiently handles user authentication, providing functionalities like registration, login, and password resetting. The interface of dashboard a user-friendly showcasing the latest trends, and prices in the cryptocurrency market, and it is fetching real-time data from external APIs. The backend, built with Flask, integrates seamlessly with the SQLite database, ensuring data integrity and security. Overall, this project successfully combines web development best practices with cryptocurrency market data to deliver a valuable resource for its users. [3]

References

[1] E. Times, "The Economic TimesET Markets," 2024.

[2] 2nd, Ed."CCTX Python," *CCTX.*

[3] H. Millier, "Cryptocurrency Prices And News: Bitcoin Price Holds $43,000 Amid Regional Bank Worries," *Cryptocurrency,* 21 December 2021.

[4] A. Aloosh, The Psychology of Cryptocurrency Prices, 2019.