Cryptfolio - A crypto portfolio tracker REST API

IT325 Web Services Final Project

by

Rayen Latrech

January 2024

BA/IT Senior Student

Tunis Business School

Ben Arous,TUNISIA.

2023-2024

Declaration of Academic Ethics

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I declare that I have properly and accurately acknowledged all sources used in the production of this report.

I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Date: January 21st, 2024 Rayen Latrech

Abstract

In the rapidly evolving landscape of digital technologies, the adoption of innovative solutions has become a cornerstone for progress. This is particularly evident in Tunisia, where the digital economy has emerged as a formidable contributor to the country's gross domestic product.

In response to this digital gap, the proposed API endeavors to enhance the cryptocurrency portfolio tracking experience. As the digital economy continues to reshape various facets of society, managing cryptocurrency investments remains a complex task. The API seeks to address this challenge by providing users with an intuitive and feature-rich platform, empowering them to monitor and analyze their cryptocurrency portfolios effectively. By leveraging real-time data, the API aims to contribute to the evolution of digital financial management tools.

Keywords - Digital Economy, Tunisia, Cryptocurrency, Portfolio Tracking, API Development.

Contents

Chapter 1

Introduction

My IT325 Web Services project culminates in a Flask-based RESTful API tailored for cryptocurrency portfolio management. Flask, known for its simplicity and flexibility, forms the backbone of this project. The API's core functionalities encompass user authentication, portfolio updates, and real-time cryptocurrency value retrieval through the CoinGecko API. The integration of Flask, SQLite for database management, and external API usage harmoniously aligns to provide a seamless and secure cryptocurrency portfolio management experience.

Going beyond the conventional Flask features, this API extends its capabilities with SQLite for efficient data handling and external API requests for dynamic cryptocurrency data. The resulting synergy presents a robust solution for users seeking an intuitive and secure platform for managing their cryptocurrency portfolios. This introduction lays the foundation for a deeper exploration of the API's intricacies, highlighting the symbiotic integration of Flask, SQLite, and external APIs in delivering a comprehensive cryptocurrency portfolio management tool.

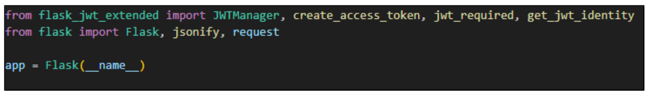
Chapter 2

Explanation of the work carried out

2.1 Flask Contribution

Flask is a small and lightweight Python web framework that provides useful tools and features that make creating web applications in Python easier. It gives developers flexibility and is a more accessible framework for new developers since you can build a web application quickly using only a single Python file. [1]

Implementation code :

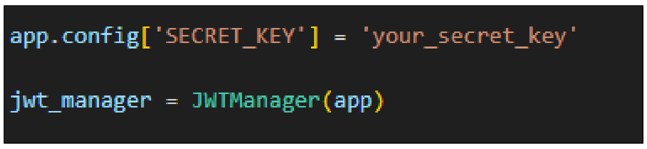


```
from flask_jwt_extended import JWTManager, create_access_token, jwt_required, get_jwt_identity
from flask import Flask, jsonify, request

app = Flask(__name__)
```

2.2 Flask JWT Extended

Flask JWT Extended is an extension for Flask that enhances its capabilities by adding JSON Web Token (JWT) support. It facilitates secure user authentication and authorization processes in Flask applications, enabling the implementation of token-based authentication mechanisms. [2]

I have used JSON Web Token authentication to secure the API access. Implementation code :



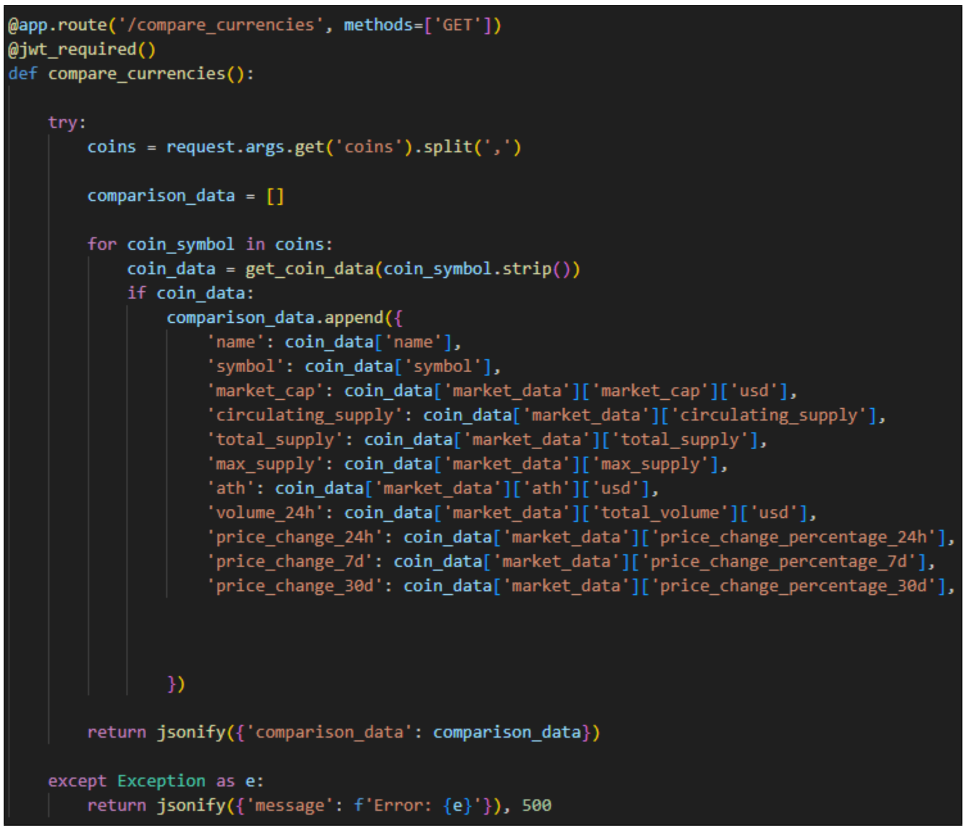
```python
app.config['SECRET_KEY'] = 'your_secret_key'

jwt_manager = JWTManager(app)
```

2.3 All the HTTP Methods used

2.3.1 GET Requests

GET /compare_currencies

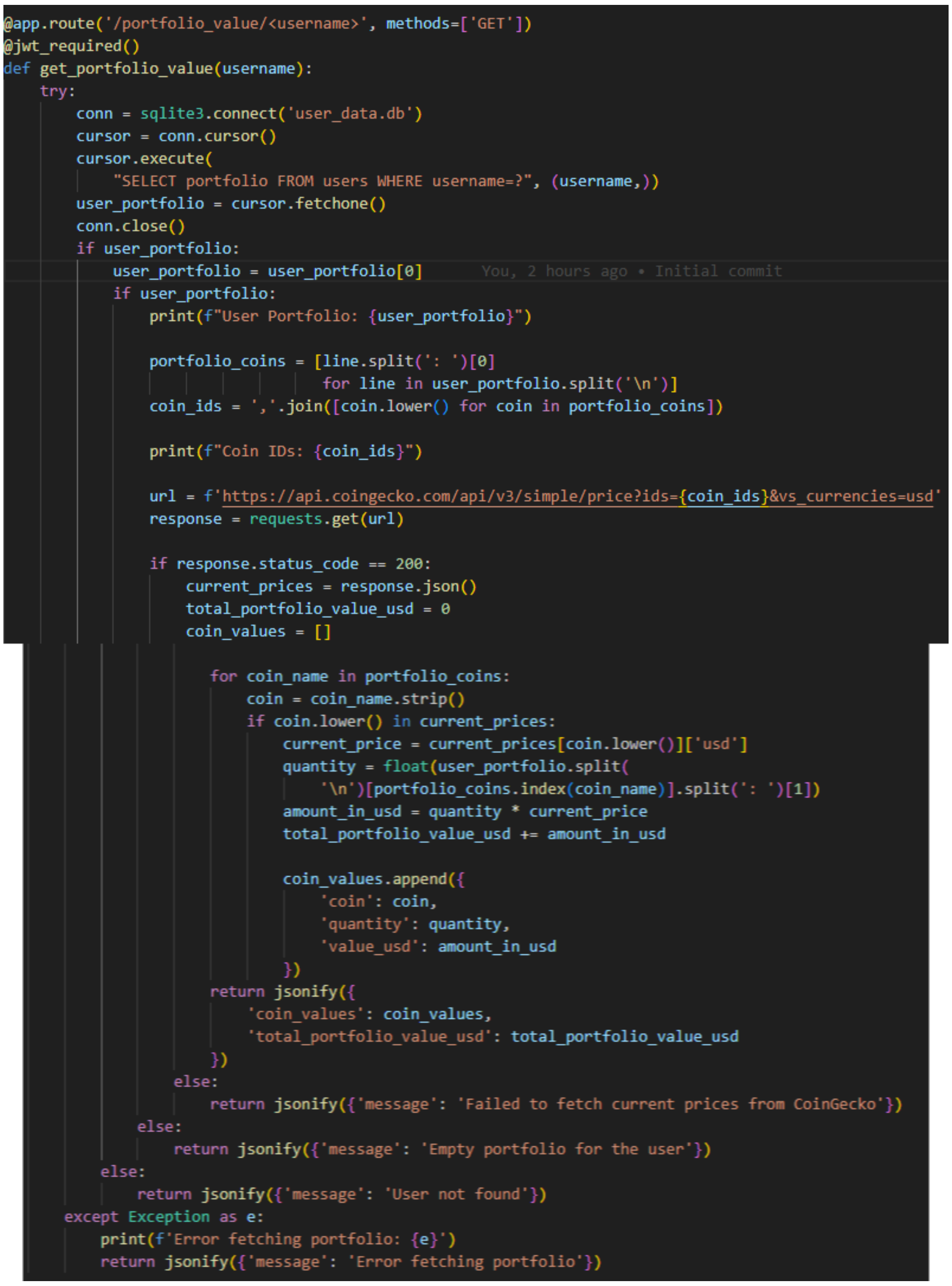
Retrieves and compares specified coins data to help provide insights on future decisions.



```python
@app.route('/compare_currencies', methods=['GET'])
@jwt_required()
def compare_currencies():

    try:
        coins = request.args.get('coins').split(',')

        comparison_data = []

        for coin_symbol in coins:
            coin_data = get_coin_data(coin_symbol.strip())
            if coin_data:
                comparison_data.append({
                    'name': coin_data['name'],
                    'symbol': coin_data['symbol'],
                    'market_cap': coin_data['market_data']['market_cap']['usd'],
                    'circulating_supply': coin_data['market_data']['circulating_supply'],
                    'total_supply': coin_data['market_data']['total_supply'],
                    'max_supply': coin_data['market_data']['max_supply'],
                    'ath': coin_data['market_data']['ath']['usd'],
                    'volume_24h': coin_data['market_data']['total_volume']['usd'],
                    'price_change_24h': coin_data['market_data']['price_change_percentage_24h'],
                    'price_change_7d': coin_data['market_data']['price_change_percentage_7d'],
                    'price_change_30d': coin_data['market_data']['price_change_percentage_30d'],



                })

        return jsonify({'comparison_data': comparison_data})

    except Exception as e:
        return jsonify({'message': f'Error: {e}'}), 500
```

GET /portfolio_value/<username>

Get the portfolio coin amounts of a given user and fetches the coins real time prices to give an accurate value.



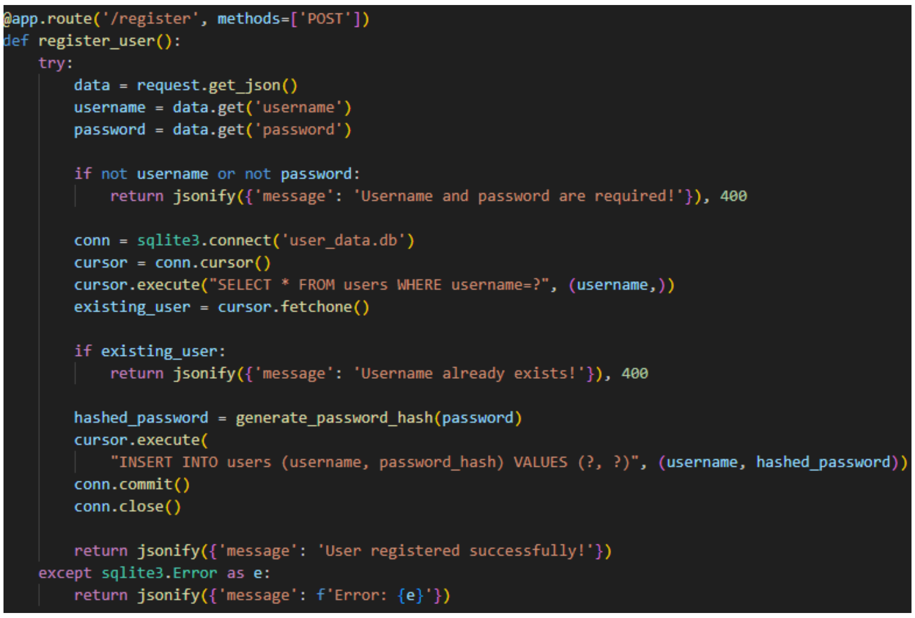
```python
@app.route('/portfolio_value/<username>', methods=['GET'])
@jwt_required()
def get_portfolio_value(username):
    try:
        conn = sqlite3.connect('user_data.db')
        cursor = conn.cursor()
        cursor.execute(
            "SELECT portfolio FROM users WHERE username=?", (username,))
        user_portfolio = cursor.fetchone()
        conn.close()
        if user_portfolio:
            user_portfolio = user_portfolio[0]        You, 2 hours ago • Initial commit
            if user_portfolio:
                print(f"User Portfolio: {user_portfolio}")

                portfolio_coins = [line.split(': ')[0]
                                   for line in user_portfolio.split('\n')]
                coin_ids = ','.join([coin.lower() for coin in portfolio_coins])

                print(f"Coin IDs: {coin_ids}")

                url = f'https://api.coingecko.com/api/v3/simple/price?ids={coin_ids}&vs_currencies=usd'
                response = requests.get(url)

                if response.status_code == 200:
                    current_prices = response.json()
                    total_portfolio_value_usd = 0
                    coin_values = []

                    for coin_name in portfolio_coins:
                        coin = coin_name.strip()
                        if coin.lower() in current_prices:
                            current_price = current_prices[coin.lower()]['usd']
                            quantity = float(user_portfolio.split(
                                '\n')[portfolio_coins.index(coin_name)].split(': ')[1])
                            amount_in_usd = quantity * current_price
                            total_portfolio_value_usd += amount_in_usd

                            coin_values.append({
                                'coin': coin,
                                'quantity': quantity,
                                'value_usd': amount_in_usd
                            })
                    return jsonify({
                        'coin_values': coin_values,
                        'total_portfolio_value_usd': total_portfolio_value_usd
                    })
                else:
                    return jsonify({'message': 'Failed to fetch current prices from CoinGecko'})
            else:
                return jsonify({'message': 'Empty portfolio for the user'})
        else:
            return jsonify({'message': 'User not found'})
    except Exception as e:
        print(f'Error fetching portfolio: {e}')
        return jsonify({'message': 'Error fetching portfolio'})
```

Figure 2.1: Illustration of the GET /portfolio_value/<username> request.

2.3.2 POST Requests

POST /register

Register in the API after specifying a valid username and password in the body of the request.

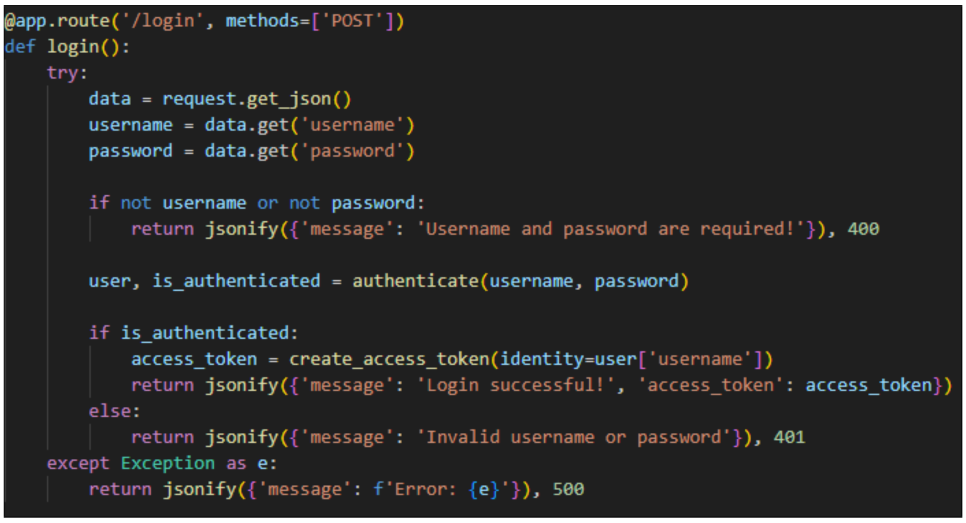


```python
@app.route('/register', methods=['POST'])
def register_user():
    try:
        data = request.get_json()
        username = data.get('username')
        password = data.get('password')

        if not username or not password:
            return jsonify({'message': 'Username and password are required!'}), 400

        conn = sqlite3.connect('user_data.db')
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM users WHERE username=?", (username,))
        existing_user = cursor.fetchone()

        if existing_user:
            return jsonify({'message': 'Username already exists!'}), 400

        hashed_password = generate_password_hash(password)
        cursor.execute(
            "INSERT INTO users (username, password_hash) VALUES (?, ?)", (username, hashed_password))
        conn.commit()
        conn.close()

        return jsonify({'message': 'User registered successfully!'})
    except sqlite3.Error as e:
        return jsonify({'message': f'Error: {e}'})
```

Figure 2.2: Screenshot demonstrating the POST /register request.

POST /login

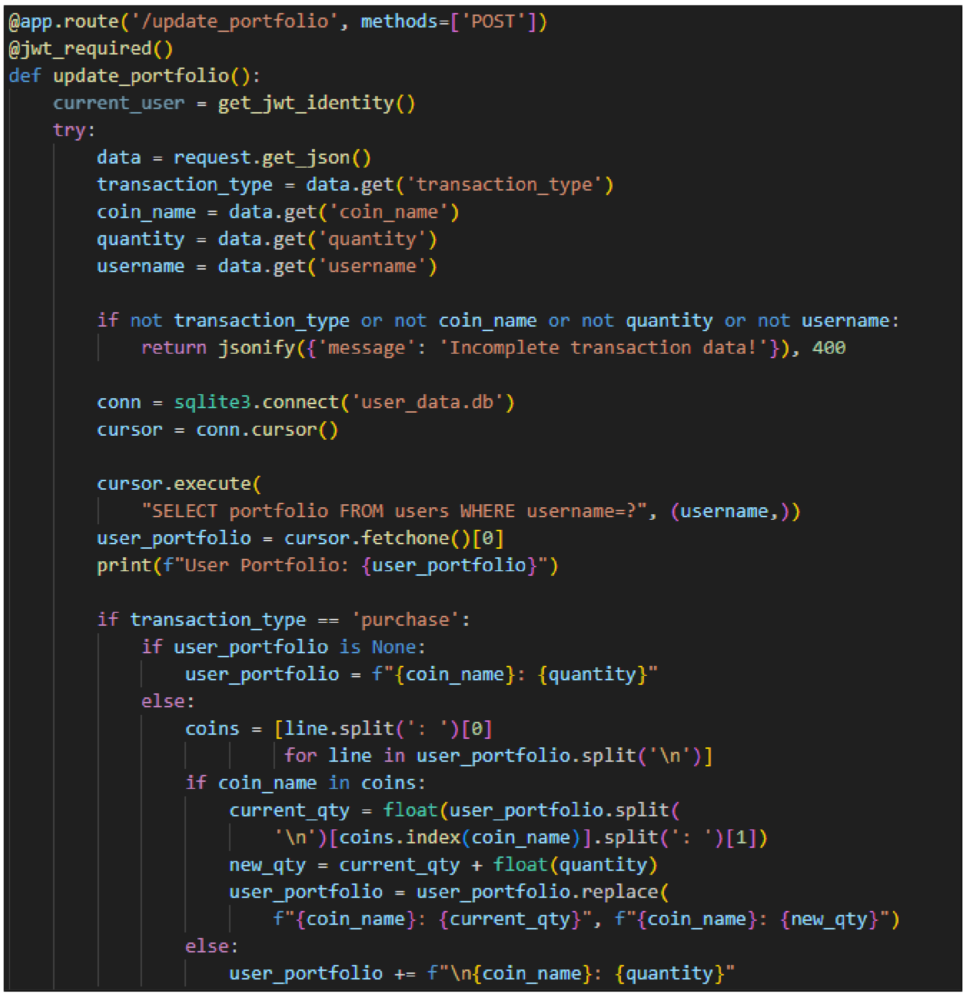
User authentication endpoint validating username and password to provide a JWT for secure access.



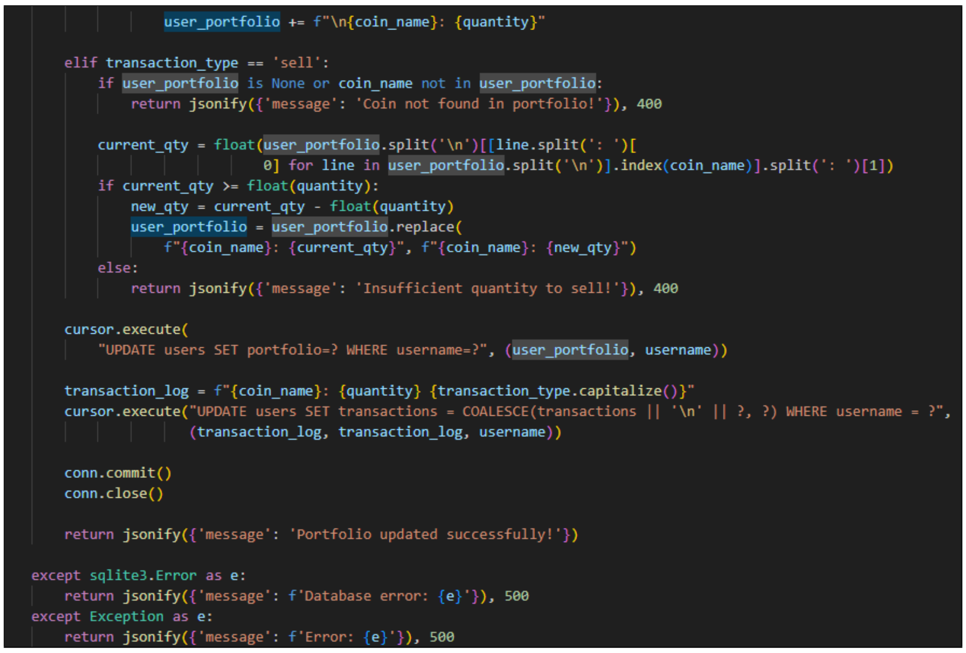
```python
@app.route('/login', methods=['POST'])
def login():
    try:
        data = request.get_json()
        username = data.get('username')
        password = data.get('password')

        if not username or not password:
            return jsonify({'message': 'Username and password are required!'}), 400

        user, is_authenticated = authenticate(username, password)

        if is_authenticated:
            access_token = create_access_token(identity=user['username'])
            return jsonify({'message': 'Login successful!', 'access_token': access_token})
        else:
            return jsonify({'message': 'Invalid username or password'}), 401
    except Exception as e:
        return jsonify({'message': f'Error: {e}'}), 500
```

POST /update_portfolio

Manages the user’s selling and purchasing requests and updating his portfolio the transaction history accordingly.



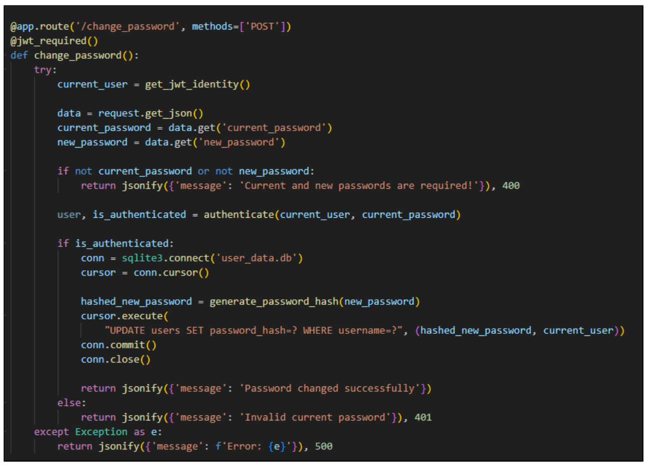
```python
@app.route('/update_portfolio', methods=['POST'])
@jwt_required()
def update_portfolio():
    current_user = get_jwt_identity()
    try:
        data = request.get_json()
        transaction_type = data.get('transaction_type')
        coin_name = data.get('coin_name')
        quantity = data.get('quantity')
        username = data.get('username')

        if not transaction_type or not coin_name or not quantity or not username:
            return jsonify({'message': 'Incomplete transaction data!'}), 400

        conn = sqlite3.connect('user_data.db')
        cursor = conn.cursor()

        cursor.execute(
            "SELECT portfolio FROM users WHERE username=?", (username,))
        user_portfolio = cursor.fetchone()[0]
        print(f"User Portfolio: {user_portfolio}")

        if transaction_type == 'purchase':
            if user_portfolio is None:
                user_portfolio = f"{coin_name}: {quantity}"
            else:
                coins = [line.split(': ')[0]
                         for line in user_portfolio.split('\n')]
                if coin_name in coins:
                    current_qty = float(user_portfolio.split(
                        '\n')[coins.index(coin_name)].split(': ')[1])
                    new_qty = current_qty + float(quantity)
                    user_portfolio = user_portfolio.replace(
                        f"{coin_name}: {current_qty}", f"{coin_name}: {new_qty}")
                else:
                    user_portfolio += f"\n{coin_name}: {quantity}"
```



```python
                    user_portfolio += f"\n{coin_name}: {quantity}"

        elif transaction_type == 'sell':
            if user_portfolio is None or coin_name not in user_portfolio:
                return jsonify({'message': 'Coin not found in portfolio!'}), 400

            current_qty = float(user_portfolio.split('\n')[[line.split(': ')[
                                0] for line in user_portfolio.split('\n')].index(coin_name)].split(': ')[1])
            if current_qty >= float(quantity):
                new_qty = current_qty - float(quantity)
                user_portfolio = user_portfolio.replace(
                    f"{coin_name}: {current_qty}", f"{coin_name}: {new_qty}")
            else:
                return jsonify({'message': 'Insufficient quantity to sell!'}), 400

        cursor.execute(
            "UPDATE users SET portfolio=? WHERE username=?", (user_portfolio, username))

        transaction_log = f"{coin_name}: {quantity} {transaction_type.capitalize()}"
        cursor.execute("UPDATE users SET transactions = COALESCE(transactions || '\n' || ?, ?) WHERE username = ?",
                       (transaction_log, transaction_log, username))

        conn.commit()
        conn.close()

        return jsonify({'message': 'Portfolio updated successfully!'})

    except sqlite3.Error as e:
        return jsonify({'message': f'Database error: {e}'}), 500
    except Exception as e:
        return jsonify({'message': f'Error: {e}'}), 500
```

POST /change_password

Enables the user to change his password while requiring the current password before change for security purposes.

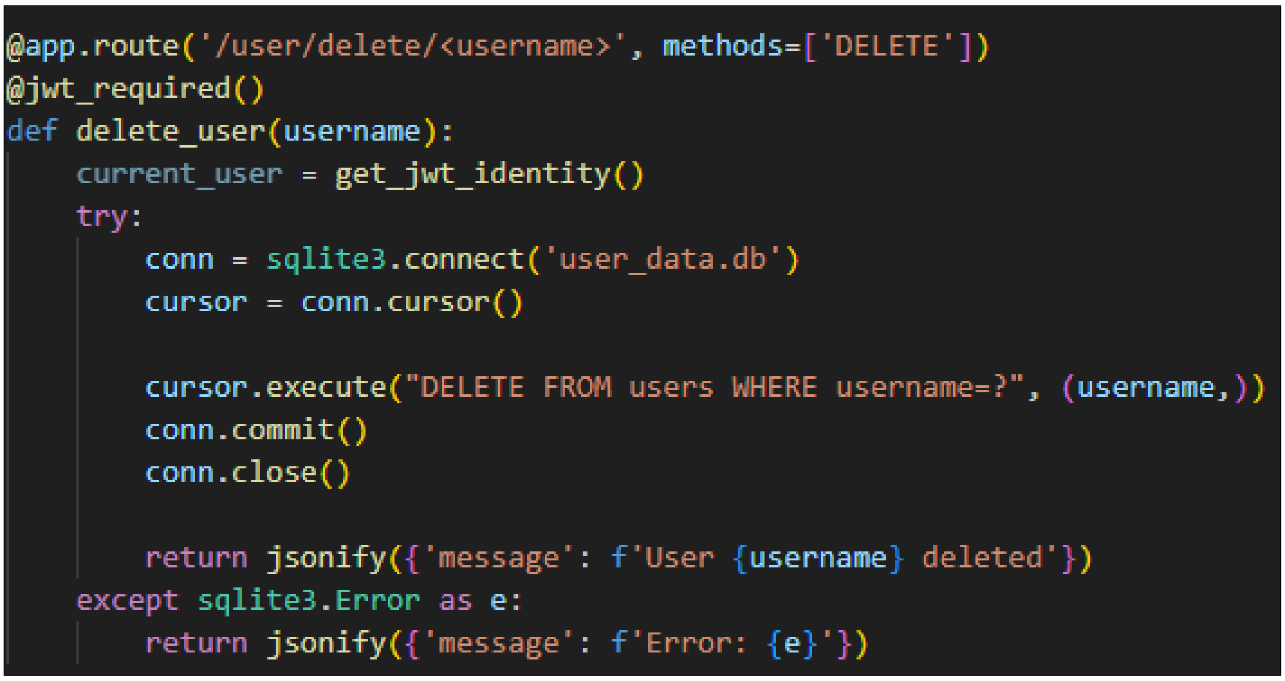


```python
@app.route('/change_password', methods=['POST'])
@jwt_required()
def change_password():
    try:
        current_user = get_jwt_identity()

        data = request.get_json()
        current_password = data.get('current_password')
        new_password = data.get('new_password')

        if not current_password or not new_password:
            return jsonify({'message': 'Current and new passwords are required!'}), 400

        user, is_authenticated = authenticate(current_user, current_password)

        if is_authenticated:
            conn = sqlite3.connect('user_data.db')
            cursor = conn.cursor()

            hashed_new_password = generate_password_hash(new_password)
            cursor.execute(
                "UPDATE users SET password_hash=? WHERE username=?", (hashed_new_password, current_user))
            conn.commit()
            conn.close()

            return jsonify({'message': 'Password changed successfully'})
        else:
            return jsonify({'message': 'Invalid current password'}), 401
    except Exception as e:
        return jsonify({'message': f'Error: {e}'}), 500
```

2.3.3 DELETE Requests

DELETE /user/delete/<username>

Allows authenticated users to delete their account.

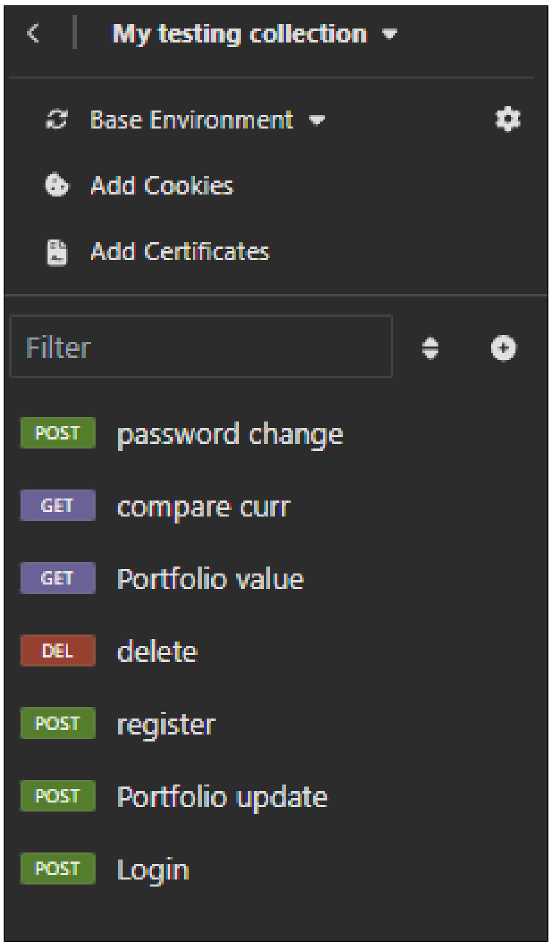


```python
@app.route('/user/delete/<username>', methods=['DELETE'])
@jwt_required()
def delete_user(username):
    current_user = get_jwt_identity()
    try:
        conn = sqlite3.connect('user_data.db')
        cursor = conn.cursor()

        cursor.execute("DELETE FROM users WHERE username=?", (username,))
        conn.commit()
        conn.close()

        return jsonify({'message': f'User {username} deleted'})
    except sqlite3.Error as e:
        return jsonify({'message': f'Error: {e}'})
```

2.4 Insomnia Contribution

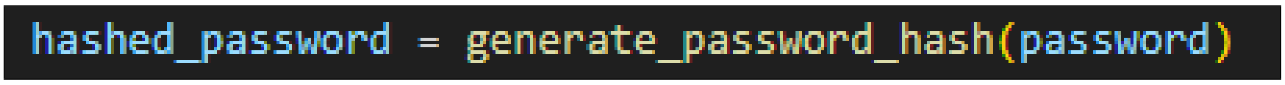
Insomnia is an application used for API testing. It is an HTTP client that tests HTTP requests, utilizing a graphical user interface, through which we obtain different types of responses that need to be subsequently validated. [?]

I have used Insomnia for the automatic testing of my API requests, as well as some snippets such as "Response Time less than 200ms","Status Code is 200"..etc.

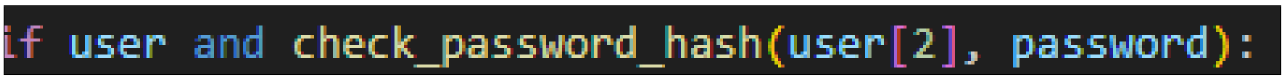


2.5 Werkzeug Contribution

Werkzeug is utilized for secure password management through its password hashing functionality, enhancing the security of user credentials in the application.. [?]



```
hashed_password = generate_password_hash(password)
```

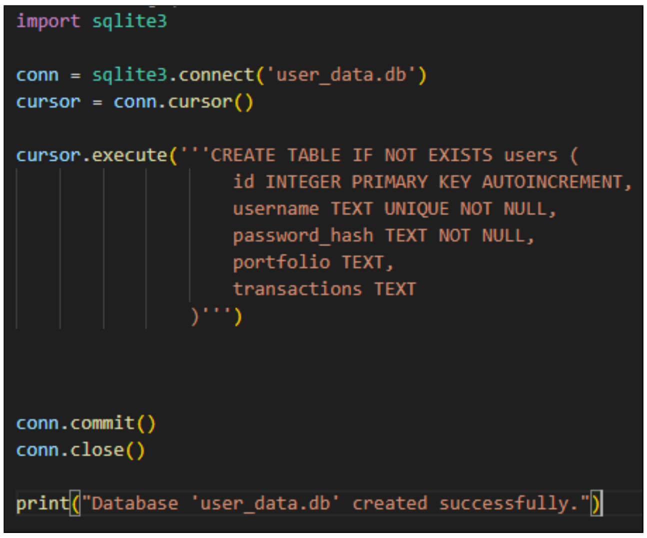


```python
if user and check_password_hash(user[2], password):
```

2.6 SQLite3 Contribution

SQLite3 is a lightweight, serverless, and self-contained relational database management system. [?]

SQLite3 was employed to manage user data, including user authentication details, portfolios, and transactions, providing a local database solution for my API.

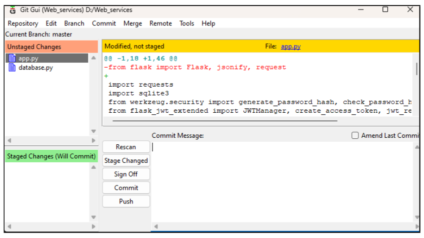


```python
import sqlite3

conn = sqlite3.connect('user_data.db')
cursor = conn.cursor()

cursor.execute('''CREATE TABLE IF NOT EXISTS users (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    username TEXT UNIQUE NOT NULL,
                    password_hash TEXT NOT NULL,
                    portfolio TEXT,
                    transactions TEXT
                )''')



conn.commit()
conn.close()

print("Database 'user_data.db' created successfully.")
```

2.7 Git Contribution

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. [3]

I have used Git to save changes and version control for the development of my API).



```
@@ -1,18 +1,46 @@
-from flask import Flask, jsonify, request
+
 import requests
 import sqlite3
 from werkzeug.security import generate_password_hash, check_password_h
 from flask_jwt_extended import JWTManager, create_access_token, jwt_re
```

2.8 CoinGecko API Contribution

CoinGecko API is a cryptocurrency data API that provides comprehensive information about various cryptocurrencies..

I used the CoinGecko API to fetch real-time cryptocurrency prices, market data, and other relevant details. This data is utilized to calculate and display information such as portfolio values and to compare different cryptocurrencies within the API.

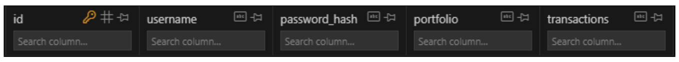
2.9 Database Structure

Source of my database : user_data.db.
The database consists of single table named "users".

Table "users"

containing 5 columns :

1. id : Unique identifier for each user.
2. username : Name of the user, stored as text and ensuring uniqueness.
3. password_hash : Securely hashed password for user authentication.
4. portfolio : Field to store user portfolio information related to cryptocurrency transactions.
5. transactions : Field capturing transaction logs, detailing user activities.



id	username	password_hash	portfolio	transactions
Search column...	Search column...	Search column...	Search column...	Search column...

Chapter 3

Conclusion

This API project serves as a practical and innovative solution, addressing the need for secure cryptocurrency portfolio management, and real-time data comparison. Passionate about the use of crypto in finance and a desire to contribute to societal and economic growth, the project fits into the bigger aim of promoting digital transformation.

It has been a pleasure working on this project. Despite the fact that I encountered multiple difficulties, I am happy that I had the opportunity to learn as much as I did and advance both intellectually and personally.

I am grateful to our professor who paved our learning path to lead us here, and I would like to thank him for his guidance.

Rayen Latrech

TUNIS BUSINESS SCHOOL
UNIVERSITY OF TUNIS

Appendix A

Response examples

A.1 GET Requests

A.1.1 GET /compare_currencies

```
{"comparison_data":[{"ath":69045,"circulating_supply":19600337.0,
"market_cap":844748653778,"max_supply":21000000.0,"name":"Bitcoin",
"price_change_24h":0.41499,"price_change_30d":2.36884,"price_change_7d"
"symbol":"btc","total_supply":21000000.0,"volume_24h":21798897867},
{"ath":410.26,"circulating_supply":74100145.7334713,
"market_cap":5135502538,"max_supply":84000000.0,
"name":"Litecoin","price_change_24h":-1.35354,"price_change_30d":-4.839
"price_change_7d":4.83494,"symbol":"ltc","total_supply":84000000.0,
"volume_24h":576150651},
{"ath":259.96,"circulating_supply":432695349.1424,"market_cap":41832590
"max_supply":null,"name":"Solana","price_change_24h":0.71043,
"price_change_30d":30.64585,"price_change_7d":-3.46362,"symbol":"sol",
"total_supply":567226197.657204,"volume_24h":1772694639}]}
```

A.1.2 GET /portfolio_value/<username>

```
{"coin_values":[{"coin":"bitcoin","quantity":0.05,"value_usd":2048.3}
,{"coin":"ethereum","quantity":0.1,"value_usd":245.639},
{"coin":"turbos-finance","quantity":10000.0,"value_usd":36.173300000000
"total_portfolio_value_usd":2330.1123000000002}
```

A.2 POST Requests

A.2.1 POST /register

```
{
  "message": "User registered successfully!"
}
```

A.2.2 POST /login

```
{"access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc
2UsImlhdCI6MTcwNTY4NTk5MSwianRpIjoiNGZmNzAzNmUtYmQzOC00M2ZlLWIxY2QtNDY2
ZjNmNDEyY2FlIiwidHlwZSI6ImFjY2VzcyIsInN1YiI6InJheWVuIiwibmJmIjoxNzA1Njg
1OTkxLCJjc3JmIjoiZTI0MTkxY2UtODJiOC00NjQ5LThlMzktNGU3NzJkYWMwNTY0IiwiZX
hwIjoxNzA1Njg2ODkxfQ.8RqxVwaYexBBckzbOzF91SniTfBZVtEuCADxXn-0qt0",
"message":"Login successful!"}
```

A.2.3 POST /update_portfolio

```
{"message":"Portfolio updated successfully!"}
```

A.2.4 POST /change_password

```
{"message":"Password changed successfully"}
```

A.3 DELETE Requests

A.3.1 DELETE /user/delete/<username>

```
{
  "message": "User rayen deleted"
}
```

Bibliography

[1] “flask.” https://en.wikipedia.org/wiki/Flask_(web_framework).

[2] “Jwt.” https://en.wikipedia.org/wiki/JSON_Web_Token.

[3] “Git.” https://git-scm.com/.