

Cryptocurrency Price Tracking REST API

Introduction

The primary objective of this project is to develop a RESTful API in JavaScript(node.js) for cryptocurrency price tracking, featuring user authentication with JWT tokens, CRUD operations, real-time updates, and thorough documentation. This report will detail the design choices, libraries used, and challenges faced during the development process.

CRUD Operations

User Management API:

The user management API comprises endpoints for creating users, retrieving user details, updating user information, deleting users, and user login. Token validation middleware secures certain routes, ensuring authenticated access.

User Favourites API:

For managing users' favourite cryptocurrencies, endpoints for adding, retrieving, updating, and deleting favourites were implemented. Token validation is integrated to secure these routes, ensuring only authorized users can perform these operations.

Real-Time Updates:

To provide real-time updates, the API is configured to refresh cryptocurrency prices every 5 minutes. This is achieved through scheduled tasks using set Intervals in the crypto prices update(cryptoPricesUpdate) module. While WebSocket implementation for real-time updates is considered a bonus, it enhances the user experience.

Cryptocurrencies symbols available in the API:

- etheur - Ethereum
- btcusdt - Bitcoin
- ethusdt - Ethereum
- xrpusdt - Ripple
- ltcusdt - Litecoin
- adausdt - Cardano
- bnbusdt - Binance Coin
- dogeusdt - Dogecoin
- dotusdt - Polkadot
- linkusdt - Chainlink

User Authentication:

User authentication is crucial for securing user data and ensuring authorized access. JWT (JSON Web Tokens) are employed for maintaining user sessions. The `token_validation.js` middleware verifies the token, granting access to protected routes only to authenticated users.

Libraries Used

Express.js:

Code: `'npm install express'`

Express.js is utilized for building the web application and handling HTTP requests. Its simplicity and flexibility make it an ideal choice for designing the API endpoints and managing routes.

MySQL2:

Code: `'npm install mysql2'`

MySQL2 is employed for efficient database interactions and queries. It enhances the API's ability to store and retrieve user and cryptocurrency data seamlessly.

JWT (JSON Web Token):

Code: `'npm install jsonwebtoken'`

JWT is chosen for secure user authentication. It allows the generation of tokens for authenticated users, facilitating secure communication between the client and server.

Bcrypt:

Code: `'npm install bcrypt'`

Bcrypt is used for password hashing and comparison, enhancing the security of user credentials stored in the database.

Dotenv:

Code: `'npm install dotenv'`

Dotenv is a zero-dependency module that loads environment variables from a `.env` file into `process.env`. It simplifies the management of environment variables in Node.js applications.

Nodemon:

Code: `'npm install --save-dev nodemon'`

Nodemon is a development utility that monitors changes in Node.js application files and automatically restarts the server when changes are detected. It's particularly useful during development to streamline the development workflow.

Ws(Web Shoket):

Code: 'npm install ws'

Ws is a simple and lightweight WebSocket library for Node.js. It allows implementing WebSocket communication in applications for real-time, bidirectional communication between clients and servers.

Node-cron:

Code: 'npm install node-cron'

Node-cron is a cron-like job scheduler for Node.js. It enables to schedule and automate tasks at specified intervals, making it useful for jobs such as periodic data updates, notifications, or other time-based operations.

Jest:

Code: 'npm install --save-dev jest'

Jest is a JavaScript testing framework for Node.js applications. Jest provides a testing environment, assertion libraries, and utilities for running unit tests, making it easier to ensure the correctness of code.

Challenges Faced**Database Interactions:**

Challenge: Establishing a reliable connection to the MySQL database and handling database queries efficiently.

Authentication and Authorization:

Challenge: Implementing secure token-based authentication and authorization for user routes.

Security and Environment Variables:

Challenge: Managing sensitive information, such as database details and API keys

Database Schema Validation:

Challenge: Ensuring that the data in the database follows a predefined schema to prevent data inconsistencies.ion secrets, securely.

Error Handling and Logging:

Challenge: Building a robust error-handling mechanism to gracefully handle errors and log relevant information for debugging.

Testing Complex Scenarios:

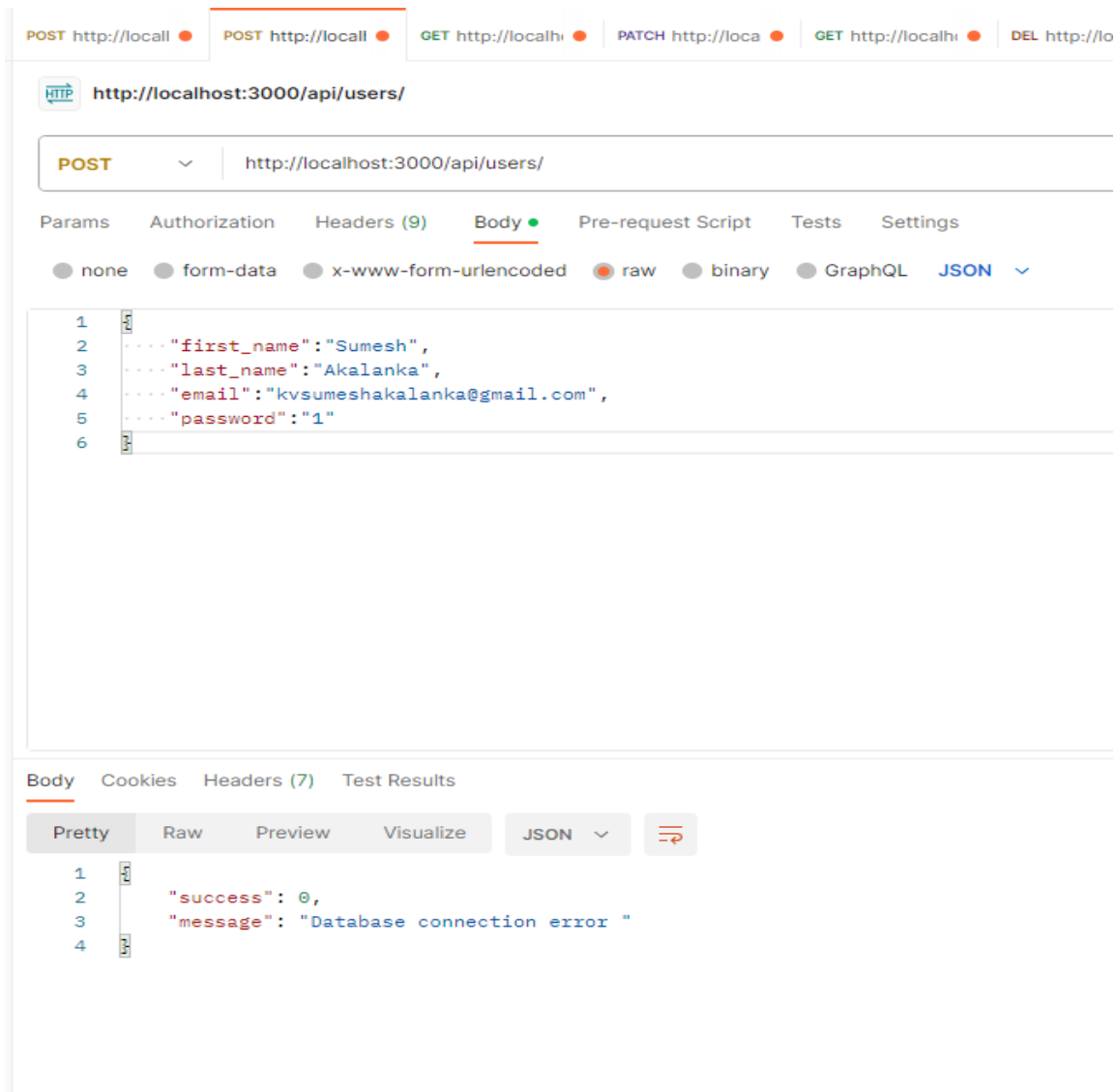
Challenge: Writing unit tests for complex scenarios, such as testing code, database interactions, or real-time features.

WebSocket's Implementation:

Challenge: Implementing and managing WebSocket connections for real-time updates.

Screenshots:

Create user:





http://localhost:3000/api/users/

POST

http://localhost:3000/api/users/

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1 {
2   ... "first_name": "Sumesh",
3   ... "last_name": "Akalanka",
4   ... "email": "kvsumeshakalanka@gmail.com",
5   ... "password": "1"
6 }
```

Body Cookies Headers (7) Test Results

Pretty

Raw

Preview

Visualize

JSON



```
1 {
2   "success": 1,
3   "message": "Creted new user",
4   "data": {
5     "fieldCount": 0,
6     "affectedRows": 1,
7     "insertId": 29,
8     "info": "",
9     "serverStatus": 2,
10    "warningStatus": 0,
11    "changedRows": 0
12  }
13 }
```

Login:

The screenshot displays a REST client interface with the following components:

- URL Bar:** Shows the URL `http://localhost:3000/api/users/login/` with an HTTP icon on the left.
- Method and URL:** A dropdown menu shows `POST` and the URL `http://localhost:3000/api/users/login/`.
- Tab Bar:** Includes tabs for `Params`, `Authorization`, `Headers (9)`, `Body` (selected), `Pre-request Script`, `Tests`, and `Settings`.
- Body Type Selection:** Radio buttons for `none`, `form-data`, `x-www-form-urlencoded`, `raw` (selected), `binary`, and `GraphQL`. A `JSON` button with a dropdown arrow is also present.
- Request Body:** A text area containing the JSON payload:

```
1 {
2   ... "email": "1",
3   ... "password": "1"
4 }
```
- Response Section:** Includes tabs for `Body` (selected), `Cookies`, `Headers (7)`, and `Test Results`.
- Response Body Type:** Buttons for `Pretty` (selected), `Raw`, `Preview`, and `Visualize`. A `JSON` button with a dropdown arrow and a refresh icon are also visible.
- Response Body:** A text area showing the JSON response:

```
1 {
2   "success": 0,
3   "data": "Invalid Email or password or Invalid input format"
4 }
```



http://localhost:3000/api/users/login/

POST

http://localhost:3000/api/users/login/

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1  {
2    "email": "kvsumeshakalanka@gmail.com",
3    "password": "1"
4  }
```

Body Cookies Headers (7) Test Results

Status: 200 O

Pretty

Raw

Preview


Visualize

JSON



```
1  {
2    "success": 1,
3    "message": "Login Successfully",
4    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOiJlM0M0NDU2N30.CSZ8j4Qhxmqqh6r1Ftns4yOdHZPqi_E4JDfe0rIxx3g"
5  }
```

Get all user:

 <http://localhost:3000/api/users/>

GET

⌵

http://localhost:3000/api/users/

ParamsAuthorizationHeaders (7)BodyPre-request ScriptTestsSettings

Query Params

	Key	Value
	Key	Value

BodyCookiesHeaders (7)Test Results

PrettyRawPreviewVisualizeJSON⌵⌵

```
1  {
2    "success": 1,
3    "data": [
4      {
5        "id": 29,
6        "firstName": "Sumesh",
7        "lastName": "Akalanka",
8        "email": "kvsumeshakalanka@gmail.com"
9      },
10     {
11       "id": 31,
12       "firstName": "Sudesh",
13       "lastName": "Anuragha",
14       "email": "kvsudeshanuradha@gmail.com"
15     }
16   ]
17 }
```


Get user by Id:

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:3000/api/users/29`
- Method:** `GET`
- Authorization:** Bearer Token
- Response Body:**

```
1  {
2    "success": 1,
3    "data": {
4      "id": 29,
5      "firstName": "Sumesh",
6      "lastName": "Akalanka",
7      "email": "kvsumeshakalanka@gmail.com"
8    }
9  }
```



http://localhost:3000/api/users/29

GET



http://localhost:3000/api/users/29

Params

Authorization ●

Headers (8)

Body

Pre-request Script

Tests

Settings



none



form-data



x-www-form-urlencoded



raw



binary



GraphQL

Body

Cookies

Headers (7)

Test Results

Pretty

Raw

Preview

Visualize

JSON



```
1 {  
2   "success": 0,  
3   "message": "Invalid token"  
4 }
```



http://localhost:3000/api/users/28

GET



http://localhost:3000/api/users/28

Params

Authorization ●

Headers (8)

Body

Pre-request Script

Tests

Settings

Type

Bearer Token



The authorization header will be automatically generated when you send the request. Learn more about [authorization](#)



Heads up! These parameters hold sensitive variables

Token

Body

Cookies

Headers (7)

Test Results

Pretty

Raw

Preview

Visualize

JSON



```
1 {
2   "success": 0,
3   "message": "Record not found"
4 }
```

Update user:

The screenshot shows a REST client interface with the following components:

- URL Bar:** Displays `http://localhost:3000/api/users/` with an HTTP icon on the left.
- Method and URL:** A dropdown menu shows `PATCH` and the URL `http://localhost:3000/api/users/`.
- Navigation Tabs:** Includes `Params`, `Authorization` (with a green dot), `Headers (10)`, `Body` (with a green dot and underline), `Pre-request Script`, `Tests`, and `Settings`.
- Body Type Selection:** A row of radio buttons for `none`, `form-data`, `x-www-form-urlencoded`, `raw` (selected with a red dot), `binary`, and `GraphQL`. A `JSON` button with a dropdown arrow is also present.
- Request Body:** A text area containing a JSON object:

```
1 {
2   "first_name": "Sumesh",
3   "last_name": "Akalanka",
4   "email": "new@gmail.com",
5   "password": "1",
6   "id": 29
7 }
```
- Response Section:** Tabs for `Body` (underlined), `Cookies`, `Headers (7)`, and `Test Results`. Below these are buttons for `Pretty` (selected), `Raw`, `Preview`, and `Visualize`. To the right are `JSON` and `Copy` buttons.
- Response Body:** A text area showing the response JSON:

```
1 {
2   "success": 1,
3   "message": "Updated successfully"
4 }
```



http://localhost:3000/api/users/

PATCH

http://localhost:3000/api/users/

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1  {
2    ... "first_name": "Sumesh",
3    ... "last_name": "Akalanka",
4    ... "email": "kvsudeshanuradha@gmail.com",
5    ... "password": "1",
6    ... "id": 29
7  }
```

Body Cookies Headers (7) Test Results

Pretty

Raw

Preview

Visualize

JSON



```
1  {
2    "success": 0,
3    "message": "Invalid input or Server error"
4  }
```

Delete user:

The screenshot displays a REST client interface for a DELETE request. The URL is `http://localhost:3000/api/users/29`. The request method is **DELETE**. The Authorization tab is active, showing a **Bearer Token** type. A warning message states: "Heads up! These parameters hold sensitive data. To k variables". The Token field contains the value `eyJh`. The response body is shown in the **Body** tab, displaying a JSON object: `{"success": 1, "message": "User deleted successfully"}`.

HTTP `http://localhost:3000/api/users/29`

DELETE `http://localhost:3000/api/users/29`

Params **Authorization** Headers (8) Body Pre-request Script Tests Settings

Type **Bearer Token**

The authorization header will be automatically generated when you send the request. Learn more about [authorization](#)

Heads up! These parameters hold sensitive data. To k variables

Token `eyJh`

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize **JSON**

```
1 {
2   "success": 1,
3   "message": "User deleted successfully"
4 }
```

Create favourite cryptocurrencies for specific user:


The screenshot displays a REST client interface with the following components:



- URL Bar:** Shows the endpoint `http://localhost:3000/api/user_favorites`.
- Method:** A dropdown menu is set to `POST`.
- Body Tab:** The request body is defined in JSON format:


```
1 {  
2   "userId": 31,  
3   "crypto_symbols": "etheur,btcusdt"  
4 }
```
- Response Section:** Includes tabs for `Body`, `Cookies`, `Headers (7)`, and `Test Results`. The `Body` tab is active, showing the response in a `Pretty` JSON format:

```
1 {  
2   "success": 1,  
3   "data": {  
4     "fieldCount": 0,  
5     "affectedRows": 1,  
6     "insertId": 9,  
7     "info": "",  
8     "serverStatus": 2,  
9     "warningStatus": 0,  
10    "changedRows": 0  
11  }  
12 }
```

 http://localhost:3000/api/user_favorites


POST  http://localhost:3000/api/user_favorites

Params Authorization  Headers (10) Body  Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** 

```
1 {
2   ... "userId": 31, ...
3   ... "crypto_symbols": "etheur,btcusdt" ...
4 }
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON 

```
1 {
2   "success": 0,
3   "message": "Dublicate entry or Database connection error"
4 }
```


Get favourite cryptocurrencies for specific user:

HTTP `http://localhost:3000/api/user_favorites/31`

GET `http://localhost:3000/api/user_favorites/31`

Params **Authorization** Headers (8) Body Pre-request Script Tests Settings

Type **Bearer Token**

The authorization header will be automatically generated when you send the request. Learn more about [authorization](#)

Heads up! These parameters hold sensitive data. To keep them secure, we've masked them with variables.

Token `eyJhbGciOiJI`

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "success": 1,
3   "data": [
4     {
5       "userId": 31,
6       "cryptoSymbols": "etheur,btcusdt"
7     }
8   ]
9 }
```

Update favourite cryptocurrencies for specific user:

The screenshot displays a REST client interface with the following components:

- URL Bar:** Shows the URL `http://localhost:3000/api/user_favorites/` with an HTTP icon on the left.
- Method and URL:** A dropdown menu is set to `PATCH`, followed by the URL `http://localhost:3000/api/user_favorites/`.
- Request Body:**
 - Navigation tabs: Params, Authorization, Headers (10), **Body** (selected), Pre-request Script, Tests, Settings.
 - Content type selection: none, form-data, x-www-form-urlencoded, **raw** (selected), binary, GraphQL, **JSON** (dropdown).
 - Request body content (lines 1-6):

```
1 {}
2 {
3   "userId": 31,
4   "crypto_symbols": "etheur,btcusdt,adausdt"
5 }
6
```
- Response Body:**
 - Navigation tabs: **Body** (selected), Cookies, Headers (7), Test Results.
 - View options: Pretty (selected), Raw, Preview, Visualize, JSON (dropdown), and a refresh icon.
 - Response body content (lines 1-4):

```
1 {}
2 {
3   "success": 1,
4   "message": "Updated successfully"
5 }
```

Delete favourite cryptocurrencies for specific user:

HTTP **DELETE** http://localhost:3000/api/user_favorites/31

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Type **Bearer Token**

The authorization header will be automatically generated when you send the request. Learn more about [authorization](#)

Heads up! These parameters hold sensitive data. To keep this c
[variables](#)

Token eyJhbGciOiJI

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "success": 1,
3   "message": "Favourite cryptocurrrencies deleted successfully"
4 }
```

Unit Test Result:

```
Node.js v18.17.1
PS C:\Users\Sumesh\Desktop\crypto price tracker> npm test

> crypto-price-tracker@1.0.0 test
> jest

PASS  __tests__/_user_favorites.service.test.js
PASS  __tests__/_user.service.test.js

Test Suites: 2 passed, 2 total
Tests:       10 passed, 10 total
Snapshots:   0 total
Time:        0.432 s, estimated 1 s
Ran all test suites.
```