

**Student : Zein Khamis**

**Course : EECE430L**

**Project : Final Project Currency Exchange**

**Project Summary**

The Currency Exchange & P2P Wallet Platform is a Flask-based backend system designed to facilitate multi-currency transactions, peer-to-peer (P2P) trading, currency forecasting, and automated exchange triggers. It integrates secure user authentication using Firebase, supports real-time currency and gold rate updates from external APIs, and includes advanced features such as forecasting using Holt-Winters models and currency recognition using image classification.

The system provides:

* A secure wallet system allowing deposits and balances in multiple currencies.
* A P2P trading system with escrow logic and rating mechanisms.
* Real-time and historical exchange rates through API integration.
* Predictive analytics to suggest "BUY", "SELL", or "HOLD" strategies.
* Alert-based triggers to automate currency trades based on thresholds.
* Optional currency recognition through image uploads using a dummy classification model.
* Full documentation and testing flow using Postman and OpenAPI Swagger UI.

Each route has been secured, validated, and tested through Postman, with screenshots provided to demonstrate correct usage and expected responses.

**Implemented Features from the Proposal:**

1. **World Currency API & Gold Tracker**
   * Real-time exchange rates fetched from the [Frankfurter API](https://www.frankfurter.app/) and gold prices via the yahoo finance.
   * Routes like /api/live-rates, /api/gold-price, and /api/dashboard-rates provide current data.
   * Historical data available via /historical-df and /api/historical/<currency>.
2. **Prediction Model Based on Transaction History**
   * Implemented using Holt-Winters forecasting (statsmodels) to suggest future exchange trends.
   * Route /predict-df returns a forecasted dataset and buy/sell/hold suggestion.
   * /predict-plot returns a visual plot of the forecast.
3. **Currency Image Recognition (Simplified)**
   * A dummy TensorFlow model was created and integrated to classify sample currency images using /recognize-currency.
   * Real Roboflow implementation was explored but replaced due to API issues.
4. **Cloud Synchronization with Firebase**
   * Firebase Authentication is used across protected routes with JWT verification.
   * Transactions are synced to Firebase Firestore via sync\_transaction\_to\_firestore().
5. **Live Location-Based Currency Detection**
   * The backend detects the user’s country based on IP and suggests a default currency using the /detect-country route.
6. **Peer-to-Peer Exchange with Wallet System**
   * Users can deposit funds into multi-currency wallets.
   * P2P orders can be created, accepted with escrow, and released securely.
   * Routes include /api/orders, /api/wallet, /api/escrow/<id>/release, and /api/rating.
7. **Auto Exchange Triggers**
   * Users can create and check exchange triggers based on rate conditions.
   * /check-triggers and /check-live-triggers manage threshold logic and execution**.**

**Security features Implemented**

**Authentication**

* Implemented using Firebase Authentication.
* All sensitive routes are protected with the custom @firebase\_token\_required decorator, which uses verify\_id\_token to validate Firebase JWTs.
* Routes that require authentication include:
  + POST /transaction
  + GET /api/wallet, POST /api/wallet/deposit
  + POST /api/orders, POST /api/orders/{id}/accept
  + POST /api/escrow/{id}/release
  + POST /api/rating
* The Firebase token is passed via the Authorization header as a Bearer token.

**Authorization**

* Role-based and ownership checks are enforced in each route.
* Example: In /api/orders/{id}/accept, the route ensures that:
  + The accepting user has sufficient balance.
  + Only the correct buyer/seller can take action on an order.
* In /api/escrow/{id}/release, only the original seller is allowed to release funds.
* This ensures that users can only manipulate their own data or permitted operations.

**Rate Limiting**

* Flask-Limiter is used to prevent abuse and DDoS attacks.
* Applied per user (via Firebase token):
  + /transaction is limited to 10 requests per minute.
  + /transactions is limited to 5 requests per minute.
* This is configured using decorators like @limiter.limit("10/minute").

**CORS Protection**

* Implemented using the flask\_cors module.
* CORS(app) enables secure cross-origin access from frontend clients.
* Ensures that only approved origins can make requests to your backend, especially important when deployed with frontend apps.

**Input Validation**

* Every POST route validates input data with request.get\_json() and explicit checks.
* Examples:
  + /check-triggers: Ensures threshold is a number, operator is valid.
  + /api/wallet/deposit: Checks for positive amount, valid currency.
  + /api/orders: Verifies fields like type, base, target, and numeric values.
* Helps prevent invalid or malicious input from breaking or manipulating your backend.

**Database Constraints**

* SQLAlchemy models are used with strict schema constraints:
  + Fields like user\_id, amount, target\_currency, etc. are marked with nullable=False.
  + Example in the Wallet, Transaction, and Order models.
* This ensures backend logic isn’t the only layer enforcing correctness—data integrity is preserved at the database level.

**Error Handling**

* Every route includes try-except blocks and clear error messages.
* Examples:
  + If a user’s token is missing or invalid, the system returns "Missing token" or "Invalid token".
  + If a deposit amount is zero or negative, it returns "Invalid amount".
  + In order processing, errors like "Insufficient balance" or "Order already accepted" are explicitly handled and returned in the response.
* These responses help both frontend developers and end users understand what went wrong.

**Escrow Logic**

* Secure logic for holding and releasing funds:
  + In /api/orders/{id}/accept, the buyer’s balance is deducted and moved to escrow.
  + The escrowed amount is held in a separate state until the seller calls /api/escrow/{id}/release.
  + Only the original seller can release the escrow.
* Prevents fraud and ensures both parties act fairly in a P2P exchange.

**1. User Sign Up**

* **Method: POST**
* **Firebase URL :**[**https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=AIzaSyBkfwfvBZNXMuHCbqAabEsyTxe6nWfuqTA**](https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=AIzaSyBkfwfvBZNXMuHCbqAabEsyTxe6nWfuqTA)

**^ This URL is exact .**

* **Body (JSON):**

**{**

**"email": "user@example.com",**

**"password": "123456",**

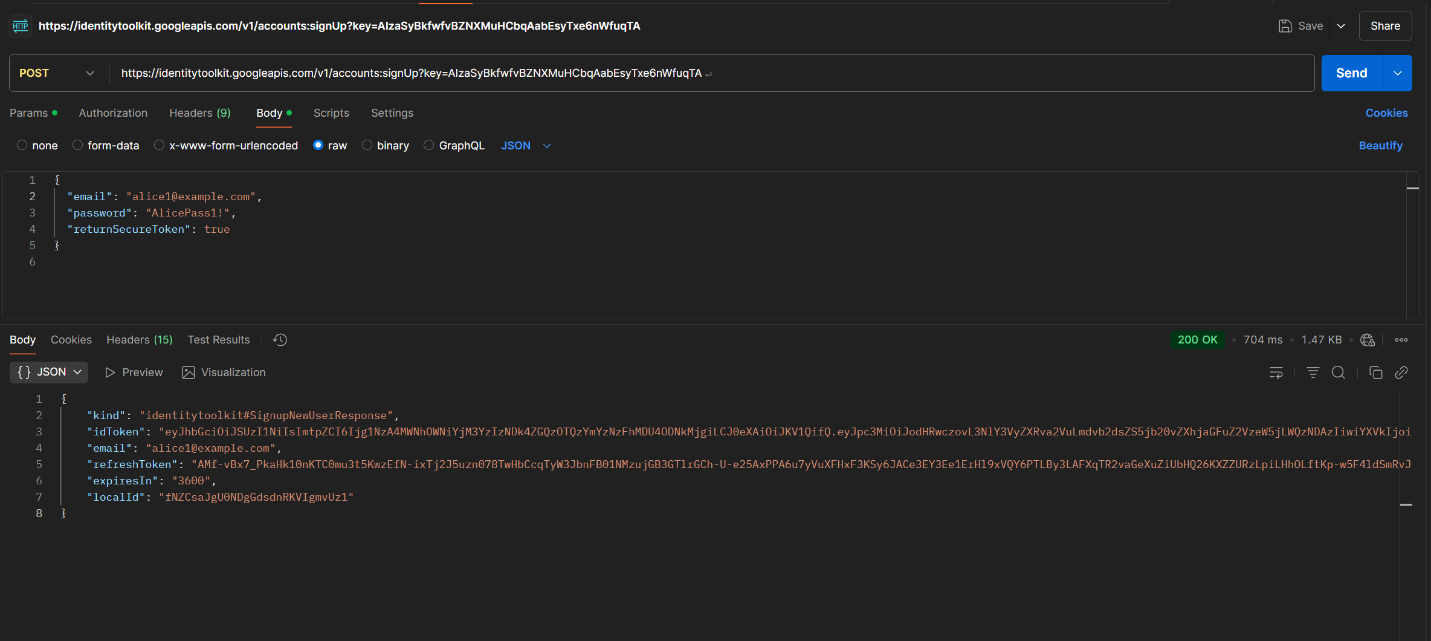
**"returnSecureToken": true**

**}**

**Headers :**

**Content-Type : application/json**

* **Returns: Firebase ID token (used for all secured routes)**

****

**2. User Login**

* **Method: POST**
* **Firebase URL:**[**https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=AIzaSyBkfwfvBZNXMuHCbqAabEsyTxe6nWfuqTA**](https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=AIzaSyBkfwfvBZNXMuHCbqAabEsyTxe6nWfuqTA)
* **Body (JSON):**

**{**

**"email": "user@example.com",**

**"password": "123456",**

**"returnSecureToken": true**

**}**

* **Returns: Firebase ID token to use as Bearer <token>**

**When return token you can use that token in Auth section you click on Auth type you go to Bearer token and put that token that you can use across the requests .**

**{**

**"email": "alice@example.com",**

**"password": "AlicePass1!",**

**"returnSecureToken": true**

**}**

**This is an example of a user I created .**

**A black background with white lines

AI-generated content may be incorrect.**

**3. Deposit Funds**

* **URL:** [**http://localhost:5000/api/wallet/deposit**](http://localhost:5000/api/wallet/deposit)
* **Method: POST**
* **Authorization: Yes (Firebase Bearer Token)**
* **Headers:  
  Authorization: Bearer <Firebase\_ID\_Token>**
* **Body (JSON)**

**{**

**"currency": "USD",**

**"amount": 1000**

**}**

* **Use case: User A deposits money**

**A black screen with white text

AI-generated content may be incorrect.**

**👛 4. Get Wallet Balance**

* **URL:** [**http://localhost:5000/api/wallet**](http://localhost:5000/api/wallet)
* **Method: GET**
* **Authorization: Yes**
* **Headers:  
  Authorization: Bearer <Firebase\_ID\_Token>**

**A screenshot of a computer

AI-generated content may be incorrect.**

**5. Create Order**

* **URL:** [**http://localhost:5000/api/orders**](http://localhost:5000/api/orders)
* **Method: POST**
* **Authorization: Yes**
* **Headers:  
  Authorization: Bearer <Firebase\_ID\_Token>**
* **Body (JSON):**

**{**

**"type": "SELL",**

**"base": "USD",**

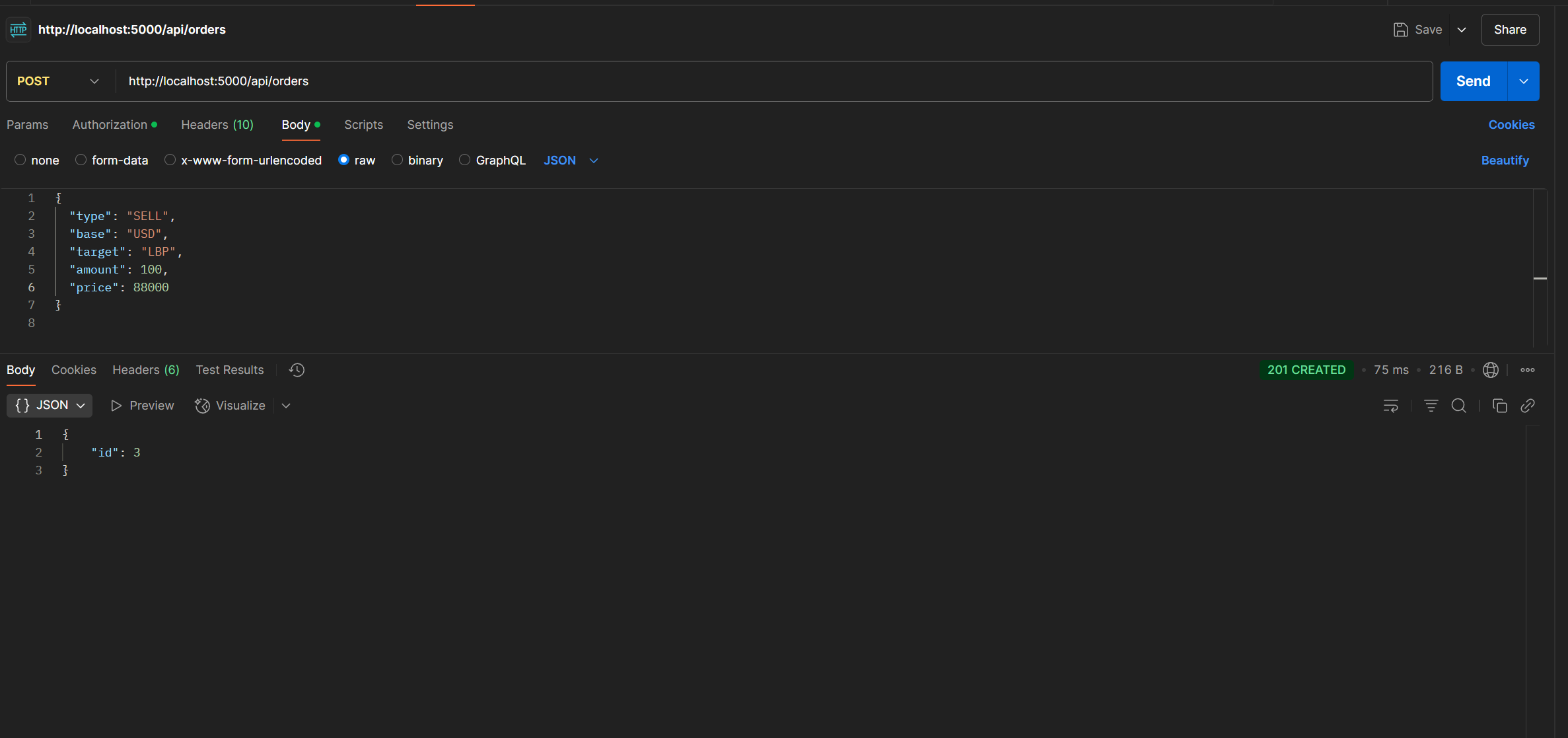
**"target": "LBP",**

**"amount": 100,**

**"price": 88000**

**}**

* **Use case: User A creates a sell order**

****

**6. Get All Orders**

**URL:** [**http://localhost:5000/api/orders**](http://localhost:5000/api/orders)

**Method: GET  
Authorization: Required (via token in headers)**

**Query Parameters:  
*None***

**A screenshot of a computer

AI-generated content may be incorrect.**

**This screenshot does not show the same order because I added this screenshot later**

**7. Accept Order**

* **URL:** [**http://localhost:5000/api/orders/<order\_id>/accept**](http://localhost:5000/api/orders/%3corder_id%3e/accept)
* **Method: POST**
* **Authorization: Yes**
* **Headers:  
  Authorization: Bearer <Firebase\_ID\_Token>**
* **Use case: User B accepts User A’s order. Escrow is created.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Make sure B has sufficient balance to accept the order . Deposit enough money in User B so he could accept the order . You have to sign in again with another user B deposit money and accept the order of User A and both have different tokens so in the Bearer auth here we put user B token .**

**8. Release Escrow**

* **URL:** [**http://localhost:5000/api/escrow/<escrow\_id>/release**](http://localhost:5000/api/escrow/%3cescrow_id%3e/release)
* **Method: POST**
* **Authorization: Yes**
* **Headers:  
  Authorization: Bearer <Firebase\_ID\_Token>**
* **Use case: User A (seller) releases escrowed funds to User B**

**A screen shot of a computer

AI-generated content may be incorrect.**

**Here we sign in again with user A and put their token to release the amount .**

**9. Submit Rating**

* **URL:** [**http://localhost:5000/api/rating**](http://localhost:5000/api/rating)
* **Method: POST**
* **Authorization: Yes**
* **Headers:  
  Authorization: Bearer <Firebase\_ID\_Token>**
* **Body (JSON)**

**{**

**"to\_user": "firebase\_user\_id",**

**"score": 5**

**}**

**A screenshot of a computer

AI-generated content may be incorrect.**

**10. Create Trigger**

* **URL:** [**http://localhost:5000/check-triggers**](http://localhost:5000/check-triggers)
* **Method: POST**
* **Body (JSON):**

**{**

**"base\_currency": "USD",**

**"target\_currency": "LBP",**

**"operator": ">=",**

**"threshold": 90000**

**}**

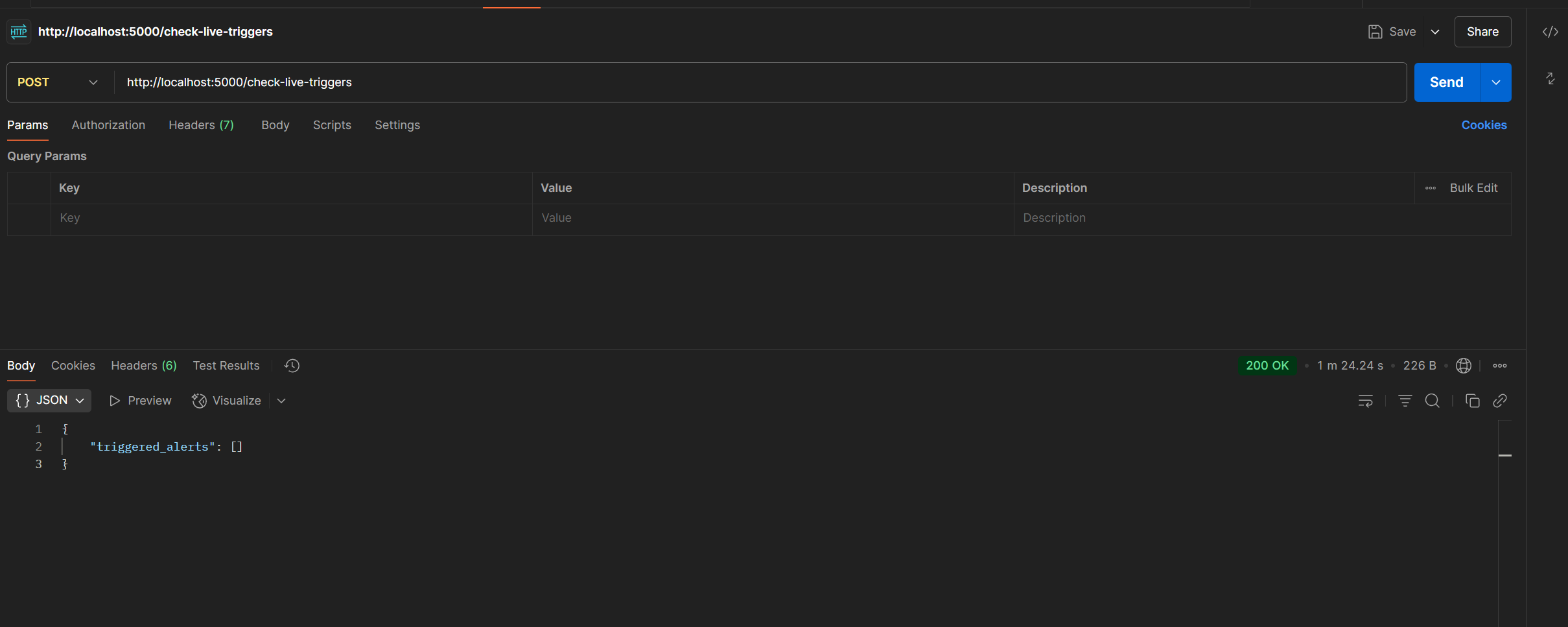
* **Use case: Set up auto-alert when condition is met**
* **Authorization: No**

**A screenshot of a computer

AI-generated content may be incorrect.**

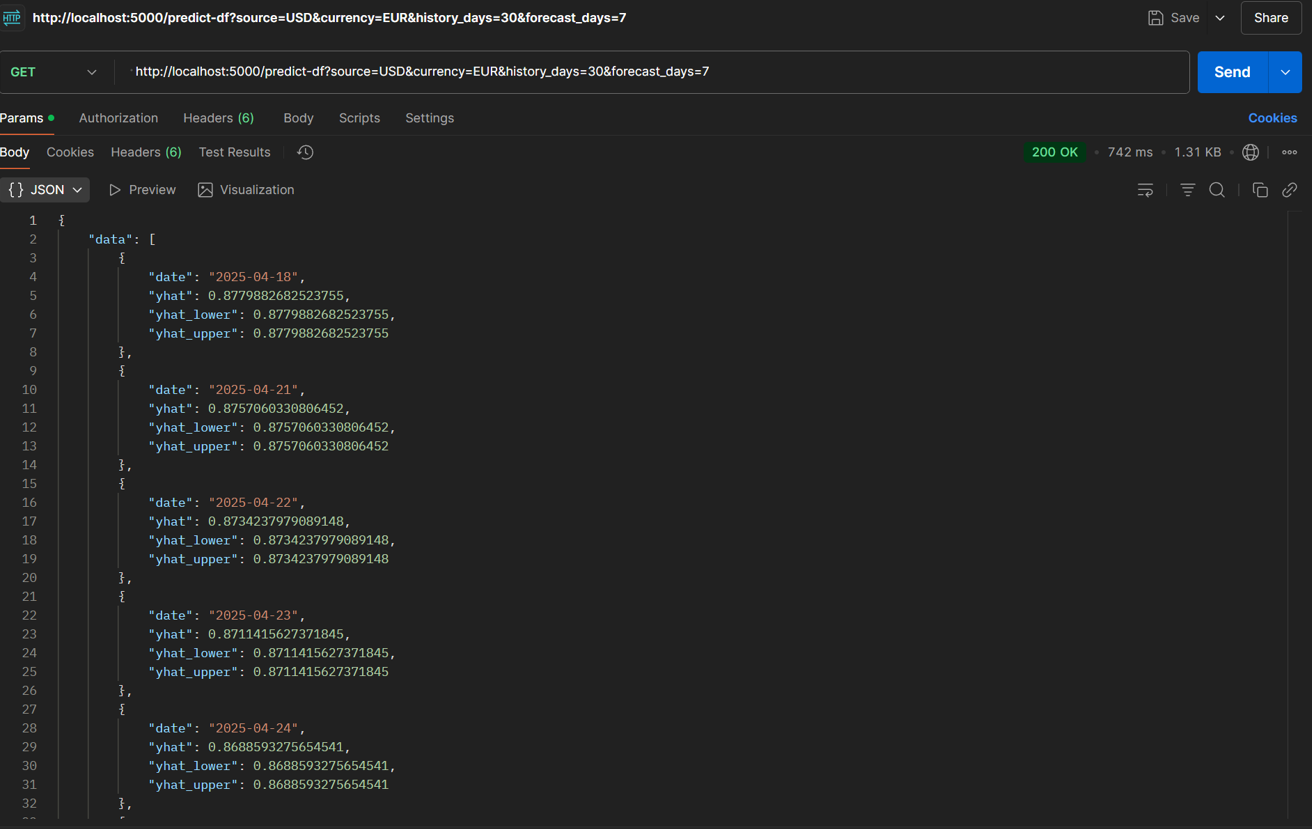
**11. Check Live Triggers**

* **URL:** [**http://localhost:5000/check-live-triggers**](http://localhost:5000/check-live-triggers)
* **Method: POST**
* **Use case: Returns all matching triggers**
* **Authorization: No**

****

**12. Predict Future Exchange Rates**

* **URL:**[**http://localhost:5000/predict-df?source=USD&currency=EUR&history\_days=30&forecast\_days=7**](http://localhost:5000/predict-df?source=USD&currency=EUR&history_days=30&forecast_days=7)
* **Method: GET**
* **Authorization: No**
* **Returns: List of forecasts + suggestion (BUY/SELL/HOLD)**

****

**13. Forecast Plot Image**

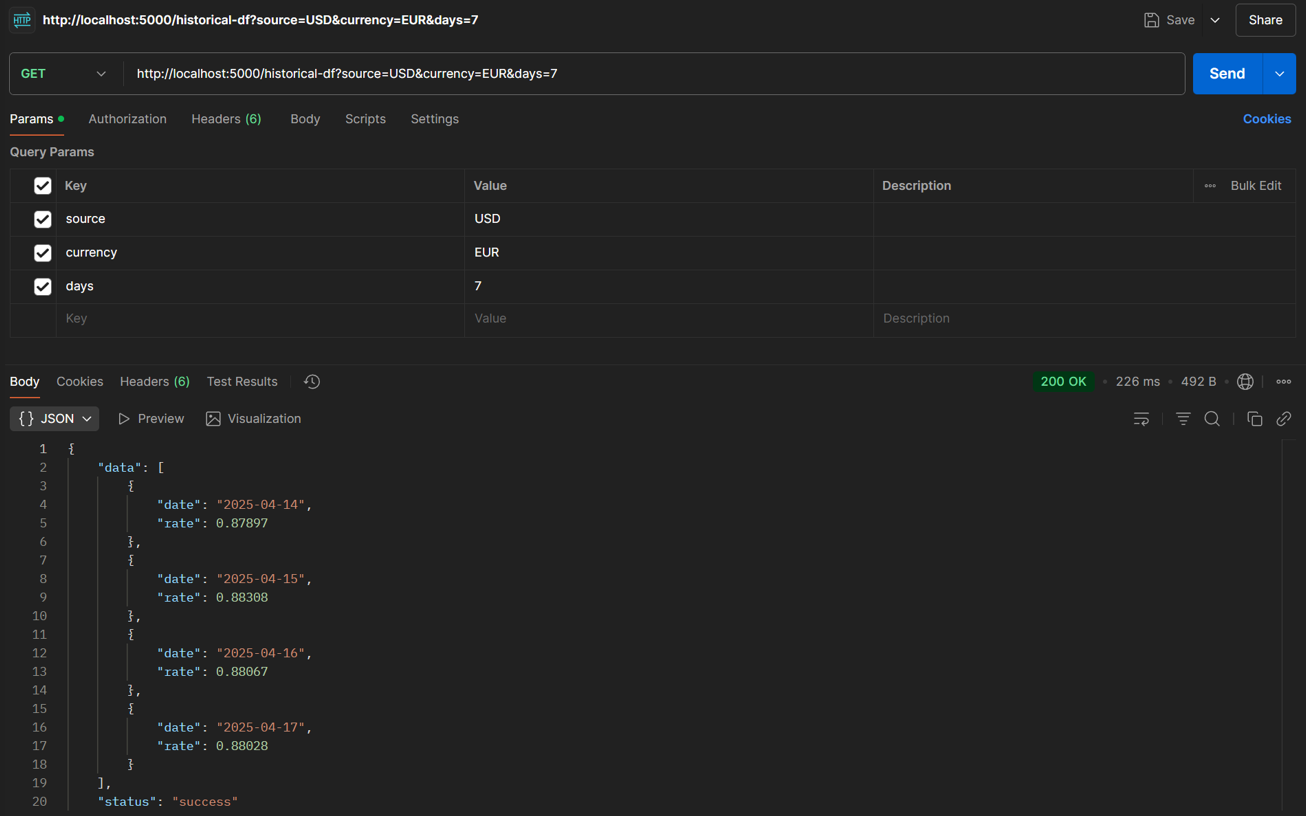
* **URL:**[**http://localhost:5000/predict-plot?source=USD&currency=EUR&history\_days=30&forecast\_days=7**](http://localhost:5000/predict-plot?source=USD&currency=EUR&history_days=30&forecast_days=7)
* **Method: GET**
* **Returns: PNG image chart**

**A screen shot of a graph

AI-generated content may be incorrect.**

**14. Historical Rates as DataFrame**

* **URL:**[**http://localhost:5000/historical-df?source=USD&currency=EUR&days=7**](http://localhost:5000/historical-df?source=USD&currency=EUR&days=7)
* **Method: GET**
* **Returns: JSON of past 7 days rates**

****

**15. Get Historical Currency Rates**

**URL:** [**http://localhost:5000/api/historical/EUR?days=7**](http://localhost:5000/api/historical/EUR?days=7) **Method: GET  
Authorization: No**

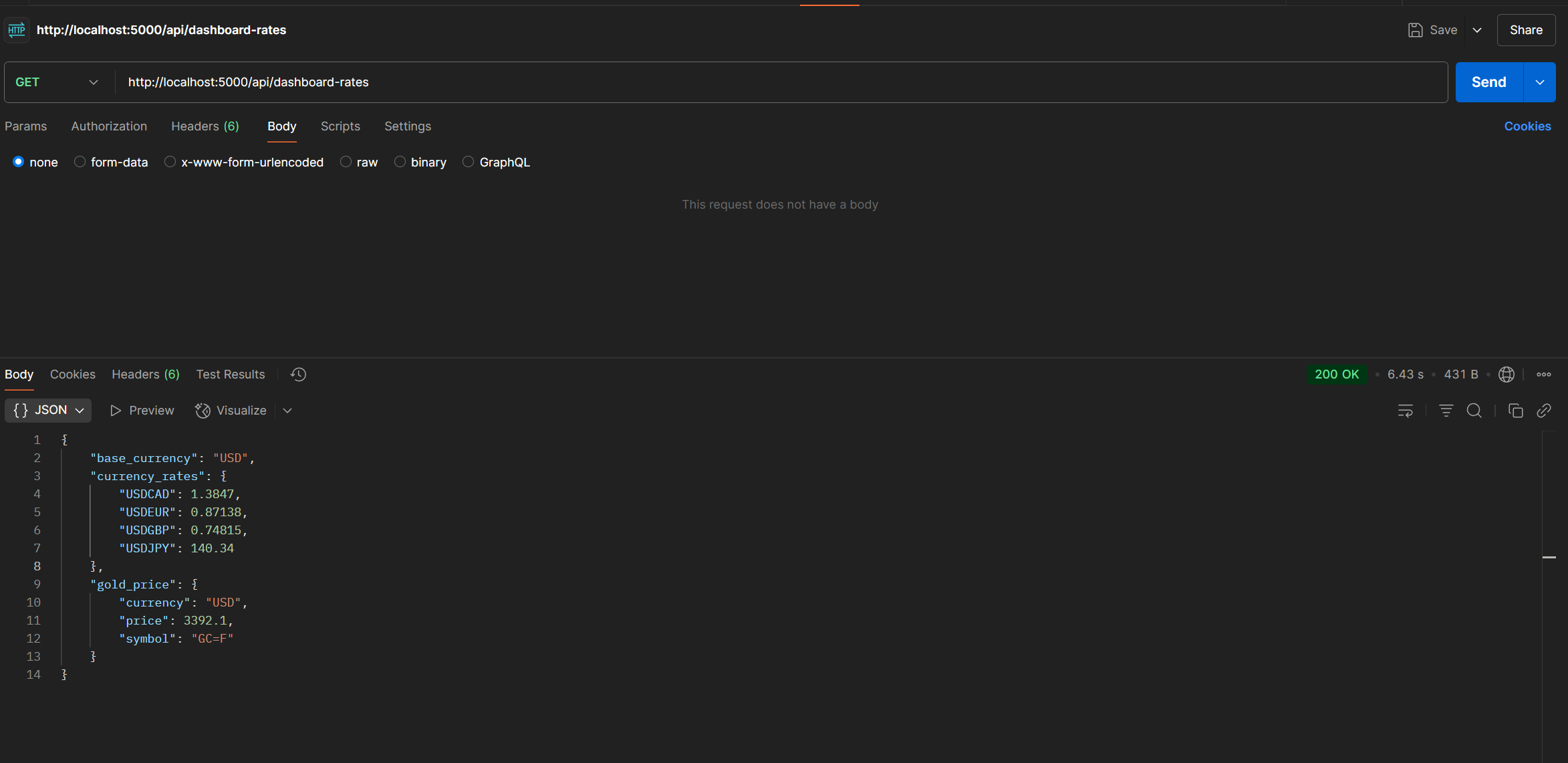
**Returns: Historical exchange rates from USD to EUR for the past X days.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**16. Live Currency & Gold Dashboard**

* **URL:** [**http://localhost:5000/api/dashboard-rates**](http://localhost:5000/api/dashboard-rates)
* **Method: GET**
* **Returns: Combined currency and gold data**

****

**17. Real-time Currency API**

* **URL:** [**http://localhost:5000/api/live-rates**](http://localhost:5000/api/live-rates)
* **Method: GET**

**A screenshot of a computer

AI-generated content may be incorrect.**

**18. Real-time Gold Price API**

* **URL:** [**http://localhost:5000/api/gold-price**](http://localhost:5000/api/gold-price)
* **Method: GET**

**A black screen with white text

AI-generated content may be incorrect.**

**19. URL :** [**http://localhost:5000/api/historical/gold?days=7**](http://localhost:5000/api/historical/gold?days=7)

**Purpose: Fetch historical gold prices for the last specified number of days.  
Method: GET  
Query Params:**

* **days: Number of previous days to retrieve data for (e.g., 7**

**A screenshot of a computer

AI-generated content may be incorrect.**

**20. Currency Recognition (Dummy Model)**

* **URL:** [**http://localhost:5000/recognize-currency**](http://localhost:5000/recognize-currency)
* **Method: POST**
* **Body:  
  form-data  
  Key: image (File upload: .jpg/.png)**
* **Returns: Predicted label like "USD\_10"**

**A black screen with white text

AI-generated content may be incorrect.**

**21. Add Transaction**

**URL:** [**http://localhost:5000/transaction**](http://localhost:5000/transaction)

**Method: POST  
Authorization: Yes (Firebase Bearer Token required)**

**Headers:**

**Authorization: Bearer <Firebase\_ID\_Token>**

**Content-Type: application/json**

**Body (JSON):**

**{**

**"usd\_amount": 200,**

**"lbp\_amount": 300000,**

**"usd\_to\_lbp": true**

**}**

**A screenshot of a computer

AI-generated content may be incorrect.**

**22. Get All Transactions**

**URL:** [**http://localhost:5000/transactions**](http://localhost:5000/transactions)

**Method: GET  
Authorization: Yes (Firebase Bearer Token required)**

**Headers:**

**Authorization: Bearer <Firebase\_ID\_Token>**

**A screenshot of a computer

AI-generated content may be incorrect.**

**23. Get Latest Transaction**

**URL:** [**http://localhost:5000/latest**](http://localhost:5000/latest)

**Method: GET  
Authorization: ✅Yes (Firebase Bearer Token required)**

**Headers:**

**Authorization: Bearer <Firebase\_ID\_Token>**

**A black screen with white text

AI-generated content may be incorrect.**

**24. Get Margin Rate with Markup**

**URL:** [**http://localhost:5000/api/margin/EUR?percent=2**](http://localhost:5000/api/margin/EUR?percent=2)

**Method: GET  
Authorization: No**

**Query Parameters:**

| **Parameter** | **Type** | **Description** |
| --- | --- | --- |
| **percent** | **int** | **The markup percentage to apply (e.g., 2 for 2%)** |

**Use Case:  
Returns the official exchange rate from USD to a given currency, applies the specified markup percentage, and returns the platform rate. This helps simulate how a currency exchange service adds a profit margin.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**25. Detect Currency by IP**

**URL: http://localhost:5000/detect-currency  
Method: GET  
Authorization: Not required**

**Query Parameters:  
*None***

**Use Case:  
Automatically detects the user's country based on IP and returns:**

* **Their default currency (used for conversions),**
* **A list of nearby travel currencies for convenience (ideal for globetrotters),**
* **The detected IP and country.**

**This powers the Live Location-Based Currency Detection feature.**

**A black screen with white text

AI-generated content may be incorrect.**

**Included APIs by Category**

**Authentication (Firebase)**

1. **POST /signUp –**
2. **POST /signInWithPassword –**

**Wallet Operations**

1. **POST /api/wallet/deposit –**
2. **GET /api/wallet –**

**P2P Exchange & Escrow**

1. **POST /api/orders –**
2. **GET /api/orders –**
3. **POST /api/orders/<order\_id>/accept –**
4. **POST /api/escrow/<escrow\_id>/release –**
5. **POST /api/rating –**

**Auto Exchange Triggers**

1. **POST /check-triggers –**
2. **POST /check-live-triggers –**

**Forecast & Historical Data**

1. **GET /predict-df –**
2. **GET /predict-plot –**
3. **GET /historical-df –**
4. **GET /api/historical/<currency>?days=7 –**

**Real-Time Data**

1. **GET /api/dashboard-rates –**
2. **GET /api/live-rates –**
3. **GET /api/gold-price –**
4. **GET /api/historical/gold?days=7 –**

**Image Recognition**

1. **POST /recognize-currency –**

**Transaction Logging**

1. **POST /transaction –**
2. **GET /transactions –**
3. **GET /latest –**

**Location Detection**

1. **GET /detect-currency –**

**Margin Calculation**

1. **GET /api/margin/<currency>?percent=x –**