Backend Coding Challenge

Intro

This is a coding challenge for engineering candidates at Foundation. It's crypto-related and makes use of tools we use daily.

This challenge should be completed in your own time, and should take roughly 3 hours. There's no expectation for you to go beyond that, and an incomplete version is okay as long as there is documentation (see below). We are looking for fully coded solution though and not pseudo code/architecture focused solutions.

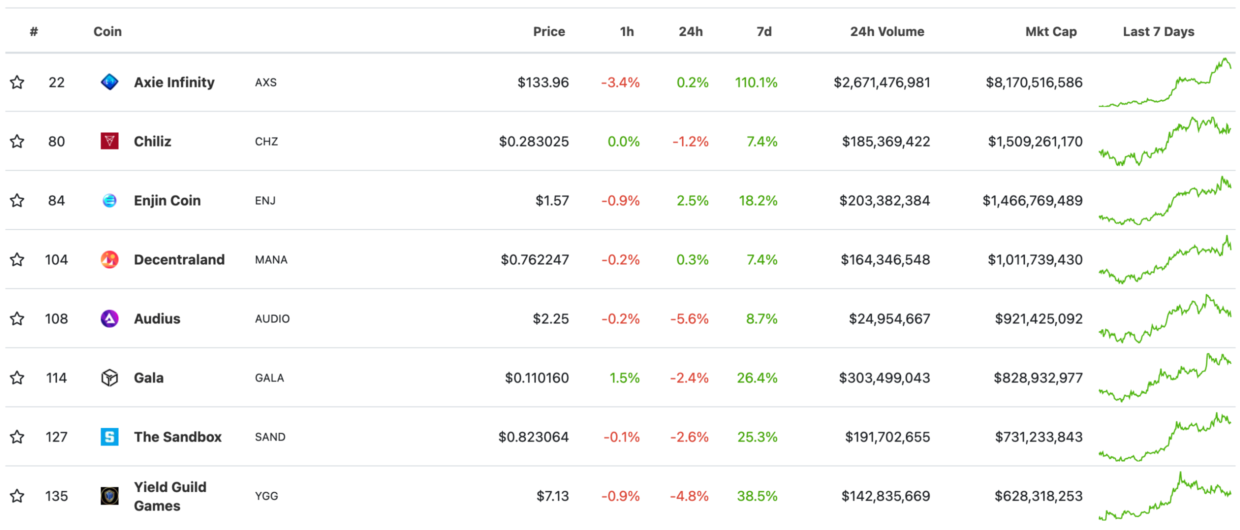
You'll be evaluated on:

- Problem solving
- Code quality
- Communication
- Testing

Background

A subgraph is a powerful Web3 tool that extracts data from the blockchain, processes it, and stores it for easy querying via GraphQL. Lots of Web3 platforms utilize subgraphs to interact and compose with other platforms. At Foundation, we host a subgraph to allow external developers to easily read our smart contract primitives such as `artworks, buy nows, offers, etc.`

For this coding challenge, you'll be tasked with creating a service that queries pricing data for various tokens via Uniswap's V3 open subgraph. With this service we'll hopefully be able to power a chart similar to below.



	#	Coin		Price	1h	24h	7d	24h Volume	Mkt Cap	Last 7 Days
☆	22	Axie Infinity	AXS	$133.96	-3.4%	0.2%	110.1%	$2,671,476,981	$8,170,516,586	
☆	80	Chiliz	CHZ	$0.283025	0.0%	-1.2%	7.4%	$185,369,422	$1,509,261,170	
☆	84	Enjin Coin	ENJ	$1.57	-0.9%	2.5%	18.2%	$203,382,384	$1,466,769,489	
☆	104	Decentraland	MANA	$0.762247	-0.2%	0.3%	7.4%	$164,346,548	$1,011,739,430	
☆	108	Audius	AUDIO	$2.25	-0.2%	-5.6%	8.7%	$24,954,667	$921,425,092	
☆	114	Gala	GALA	$0.110160	1.5%	-2.4%	26.4%	$303,499,043	$828,932,977	
☆	127	The Sandbox	SAND	$0.823064	-0.1%	-2.6%	25.3%	$191,702,655	$731,233,843	
☆	135	Yield Guild Games	YGG	$7.13	-0.9%	-4.8%	38.5%	$142,835,669	$628,318,253	

Example price charting UI your API will power.

The tokens we'll want to track:

- **$WBTC -** a token used to wrap bitcoin so you can use bitcoin as an ERC20
  ERC20 token address:
  `0x2260fac5e5542a773aa44fbcfedf7c193bc2c599`
- **$SHIB** - a meme token turned into a decentralized ecosystem
  ERC20 token address:
  `0x95ad61b0a150d79219dcf64e1e6cc01f0b64c4ce`
- **$GNO** - Gnosis platform's standard token for rewarding the platform's users
  ERC20 token address:
  `0x6810e776880c02933d47db1b9fc05908e5386b96`

The Task

Part 1: Getting the data

For this task we'll pull the data from Uniswap's V3 subgraph. You can find more details on their subgraph implementation here. Similar to Uniswap, we have a subgraph that allows developers to interact with our data.

With Uniswap's Subgraph, the `TokenHourData` object, and the `Token` object we can fetch everything we'll need for this assignment. In terms of field storage we want to store `open, close, high, low, and priceUSD` from `TokenHourData`. From `Token` we will store only the latest `name, symbol, totalSupply, volumeUSD, and decimals`. We'll also need to access the past 7 days of pricing data.

A few caveats:

1. The Graph's GraphQL language can't filter on nested columns.
2. The limit per page when querying is 100 records.
3. The Graph's subscription support is limited. We recommend using their query API instead.
4. Due to a quirk on The Graph, you'll need to downcase the address in your queries.
5. The data for a given token can be sparse. Ex. $AUPUNK may have no transactions at a certain hour.

Part 2: Ingesting the data

Now that we can access the data we need to store it. We will want to ingest the data in order starting from 7 days ago and continue polling after it has caught up.

Remember we'll only want to store the above 3 tokens. At Foundation, we use Postgres but you're free to store it in any SQL-like storage. You can also store it in whatever format or schema that you like but keep performance in mind, and remember to tell us why you chose it.

Part 3: Serving the data

Next we will power the following API endpoint for our charting and analytics service with the data we've ingested. You can use any web server, framework, or protocol you like but please explain why you chose it.

`getChartData(tokenSymbol, timeUnitInHours)` - Given the token symbol and a `timeUnitInHours` (ex. 4 means four hour increments), this should return the token metadata and a 3D array of time, price type (`open, close, high, low, priceUSD`), and the value. Time should be formatted as `2021-01-01T03:28:30` and it should include all data you've ingested. Also sort in ascending order of the time column. If no data exists within that time interval it should return a 0 for `open, close, high, low, priceUSD`.

Example call: `getChartData("SLP", 2)`

Example for 3D array.

```
[
    [
        ["2021-01-01T03:28:30", "open", 23.0], ["2021-01-01T05:28:30", "open", 25.0], ["2021-01-01T07:28:30", "open", 27.0]
    ],
    [
        ["2021-01-01T03:28:30", "close", 33.0], ["2021-01-01T05:28:30", "close", 35.0], ["2021-01-01T07:28:30", "close", 37.0]
    ],
    [
        ["2021-01-01T03:28:30", "high", 43.0], ["2021-01-01T05:28:30", "high", 45.0], ["2021-01-01T07:28:30", "high", 47.0]
    ],
    [
        ["2021-01-01T03:28:30", "low", 53.0], ["2021-01-01T05:28:30", "low", 55.0], ["2021-01-01T07:28:30", "low", 57.0]
    ],
    [
        ["2021-01-01T03:28:30", "priceUSD", 63.0], ["2021-01-01T05:28:30", "priceUSD", 65.0], ["2021-01-01T07:28:30", "priceUSD", 67.0]
    ]
]
```

Requirements

- You may use any programming language you like (although we prefer Python or Typescript).
- For persistence, please store the data in any SQL-like storage.
- If you have time, please include some sort of automated test infrastructure. You can choose any library you like.
- Your solution should have a single entry point that starts both the API service and the ingestion and runs them simultaneously. Once the ingestion has caught up to the current time it should make the API available so we never have partial data. While serving, the ingestion should continue to poll.

Documentation

Please document your code and/or write a README file. Include any instructions on how to run your solution. We'd love to get as much insight into your decisions as possible. High-level notes will help us understand your general thinking. Code comments help contextualize your decisions.

Questions you should to cover:

- What went well?
- What could have gone better?
- Is there anything specific you'd like to come back and improve if you had time? Why?

Submission

When you're finished, please create a GitHub repo with your submission. Then invite the following GitHub usernames.

- `fujiin`

- `hsuhanooi`
- `reggieag`
- `matbhz`
- `philbirt`
- `mirajp12`