Backend Technical Interview

Context

You are a Software Engineer at a company that sells datasets to Quantitative Traders so that they can backtest their strategies on historical pricing data. You are responsible for building the Backend for the **Entitlements** part of the system (i.e., providing customers with programmatic access to datasets).

The 2 main users of this part of the system are internal Operations team members (**Ops**) and external Quantitative Trader customers (**Quant**). The typical workflow consists of a Quant browsing the datasets available on the platform, and requesting access to one or more datasets; then based on the validity of the request, it is either approved/rejected automatically, or it notifies an Ops who triggers the approval or refusal of the request; finally a Quant is notified of the decision, and - if approved - is able to fetch the content of the dataset for a given frequency.

To get the product in the hands of users and get feedback early, you decide to build an MVP. This means your implementation should work end-to-end, but be pragmatic in terms of the functionalities implemented, while still considering what would be needed for a production-ready system at scale.

Scope

Given the time constraints and MVP nature of the exercise, the following functionalities will have a **limited scope** or be **out of scope**:

- **Datasets**: Each dataset will be for 1 crypto asset, and will have a **name** and a **symbol**, and be available in various **frequencies** (see table below). The datasets info will be publically available, and the pricing data will come from this CoinCap API endpoint.

Only seed the datasets below

Name	Symbol	Frequencies
Bitcoin	BTC	1 hour + 1 day + 1 month
Ethereum	ETH	1 day

- **Users**: Users should only be represented by an API Key with a role attached to it (either seeded or created via API).
- **Payment**: The dataset access request approvals/refusals are at the discretion of the Ops and not dependent on a Quant buying a package/subscription.
- **Notification**: Some functionalities will be asynchronous by nature (e.g., decision notification) and can be implemented using the pattern that you wish (e.g., background job, polling, websocket).

Exercise 1: System Design

Using a digital white-boarding tool of your choice:

1. Draw out the technical components of the system and their interactions
2. Highlight the underlying cloud services you would use to deploy it to a production environment
3. Break out problem-space into subdomain (in a Domain Driven Design fashion), which can be implemented as microservices
4. Define the database modelling

Exercise 2: Code Implementation

Think of the following requirements as **Behavioural Driven Development test-cases** - they all need to pass for the product to be considered working. The details of implementation are up to you and will be discussed during your presentation.

Functional

Mandatory

- [ ] A Quant can view the metadata (i.e., name & symbol) and available frequencies for on all datasets on the platform (i.e., BTC & ETH)
- [ ] A Quant can request access to 1 or more frequencies for a given dataset
- [ ] An Ops can view all pending dataset access requests
- [ ] An Ops can approve or refuse a request, give access (or not) to a dataset for a frequency to the Quant
- [ ] A Quant can view the pricing data (in USD) of the datasets/frequencies they have access to

Optional

- [ ] Backfill the market-cap (in USD) of each dataset every day at 8pm UTC (using this CoinCap API endpoint)
- [ ] Implement a way for Ops to provide access to datasets/frequencies for a limited amount of time (e.g., trial) and revoque accesses

Non-functional

Mandatory

- [ ] Use NodeJS with Typescript
- [ ] Use a Postgres database
- [ ] Use REST for the API layer
- [ ] Dockerise the application and run it locally using docker-compose
- [ ] Write tests to cover the critical path

Optional

- [ ] Use the NestJS framework
- [ ] Use the Sequelize ORM
- [ ] Implement at least 2 micro-services