Senior Backend Engineer CONFIRMED - Take-Home Challenge.

E-Commerce is really easy, in theory. You have x stock for a given product, and ideally, you only sell as much as you have, not more, not less. As a company, we just made the decision to externalise the logic for managing, reserving and ultimately selling inventory into its own micro service, and it's your task to create a proof of concept of said service.

Solution architecture has provided a rough outline on how the service is supposed to be working:

* For each product (do not consider individual sizes, we just assume for simplicity that a product only ever has one size), there's three stock levels: IN_STOCK, RESERVED and SOLD. Stock starts being IN_STOCK, it can be reserved, and once reserved, moved to SOLD. Stock in RESERVED should at one point be automatically freed up, but that's out of scope for this PoC.

Your task is to build a system that consists of the following endpoints:

- `PATCH /product/:id/stock`, with a payload consisting of a JSON Object that is formed as follows:

```
{"stock": 123}
```

This endpoint sets the stock that is available to be sold (i.e. IN_STOCK), overwriting any existing values for IN_STOCK. If a record for the product does not exist, a new one is created.

- `GET /product/:id`

This endpoint returns a JSON Object returning the stock level for the product given in the following form:

```
{"IN_STOCK": 123, "RESERVED": 4, "SOLD": 12}
```

If there is no record for the given Product ID, the request returns a status code of 404 with an arbitrary response body that must be ignored, otherwise the status code is a 200.

- `POST /product/:id/reserve`, without any payload. This call reserves one item of stock. On success, a 200 status code is returned, with a response JSON that is having the following form

```
{"reservationToken": "22489339-5462-458b-b184-fc1f55eedab5"}
```

This call moves stock from IN_STOCK to reserved. Should the IN_STOCK column have a value lower than 1, this call returns a status code of 400 (feel free to suggest a semantically more fitting one) with an arbitrary response body that must be ignored.

The Reservation Token is unique for one item of stock and is required to later mark an item of stock as sold – without this token, both the unreserve (i.e. the inverse operation) and the final sold move cannot be performed. It is the callers responsibility to keep track of this reservationToken.

- `POST /product/:id/unreserve`, with a JSON payload like this:

```
{"reservationToken": "22489339-5462-458b-b184-fc1f55eedab5"}
```

This endpoint returns a 200 status code and moves one item of stock from RESERVED to IN_STOCK if, and only if, the reservation token is recognised and belonging to the product ID indicated. After the stock item has been moved to IN_STOCK again, it is not possible to use the reservation token again, it can be destroyed by the caller.

- POST /product/:id/sold, with a JSON payload of

```
{"reservationToken": "22489339-5462-458b-b184-fc1f55eedab5"}
```

This endpoint moves a stock unit from RESERVED to the state of SOLD, and only if the reservation token is valid for this product. If successful, a 200 status code is returned, otherwise the response is 400. After this operation is complete, the reservation token can no longer be used and can be destroyed by the caller.

This API is a minimal set of functions that are needed by our web frontend to make stock reservations, show available stock and convert units to being sold.

A special focus in the implementation of the proof of concept should be put on two aspects: The data should be held in a database that is external to the application - like MongoDB, Postgres or Redis, but you choose the technology you want to apply. Secondly, this service will need to scale significantly and still perform robustly, with a special regard for correctness of reservations. What that means? You need to ensure that even if the reserve endpoint is called twice at exactly the same time, and only one unit of stock is left, only one reservation is returned – there should be robustness in regards to concurrency, with no race conditions or double reservations happening.

Code Challenge Remarks

Delivery

Please provide an archive of your code, or alternatively upload the code on your favourite public code hosting platform, like GitHub, BitBucket or Gitlab. Let your TA contact know once you have the code ready, by either sharing the archive or link. Ideally, the code should be runnable (feel free to attach a README on how to do that). Our stack is Mac OS, so it would be great if we can get it to run there. We have Docker as well, so if that's your thing, feel free to provide a Dockerfile or a docker-compose.

Walkthrough

The key part is a walkthrough in interview form where you take us through the solution you've built. This helps us understand how you approach complex solutions, what your concerns are and how you entertain a group-based discussion about technological issues. As with most complex solutions, there's no wrong or right – as long as you can make a good case for your decisions, we're happy with everything.

Tools, Technologies, Stack

While we're using a TypeScript/Node-Stack, feel free to complete the challenge in another language if you don't have prior experience in JS/TS. Ideally however a delivery in TS or JS is preferred.

I don't have time to complete the challenge

We're aware that completing this challenge comes with a significant time investment, and if you're unable to invest this time for whatever reasons, we are also happy to accept a walkthrough on how you would build this solution – system design, data modelling, key processes and so forth. Please reach out to TA if this is the case so we can accommodate for this.

I have a question or need clarification

Please send your question to TA, we'll get back to you as soon as possible.