Round 1 — API Integration

Owner	Ⓒ Chetan
Tags	
Created time	@November 14, 2023 2:51 PM

We strive to maintain a hiring process that is closely aligned with the practical challenges we encounter in our work, and the problem statement at hand is something that we work with, and would love to see your take on it, whilst trying to understand your problem-solving abilities and skills. Please go through the entire doc before attempting to build the final solution.
All the best!

Scenario

We are on-boarding a new API partner, and you've been assigned to do the Integration of their Inventory with our system

Below is the new partner's endpoint that you've been assigned to integrate with our system.

```
curl --location 'https://leap-api.tickete.co/api/v1/inventory/<productId>?date=<date>' \
--header 'x-api-key: <your-api-key>'
```

```
<productId> => Product ID
<date> => Date for which you're requesting inventory.(YYYY-MM-DD)
<your-api-key> => API key enclosed in the communication email.
```

Product IDs and their details

1. **14 (**https://leap-api.tickete.co/api/v1/inventory/14**)**
    a. Inventory is available Monday, Tuesday, and Wednesday.
    b. Multi-Time Slot (User can select one of the time slot for making bookings)
    c. Multi Pax (2 types of pax supported: Adult, Child)
2. **15 (**https://leap-api.tickete.co/api/v1/inventory/15**)**
    a. Inventory is available only on Sunday
    b. Single Time Slot
    c. Multi Pax (3 types of pax supported: Adult, Child, Infant)

What are you building?

Build a NodeJS App (ideally using NestJS Framework) and Prisma (or Similar ORM), Fetch inventory from the Dummy API above for the next two months and store it as normalized as you can imagine in your DB (avoid duplication of data as much as possible)

Once synced with your DB, you need to create two `GET` endpoints, as detailed below for consuming the data that you synced earlier.

1. **Slots API**
   `/api/v1/experience/<id>/slots?date=<date>`
   An API to return all slots available for a given `date` and product `id`
   Response type definition is given below
2. **Dates API**
   `/api/v1/experience/<id>/dates/`

Endpoint to return all available dates (for 2 months)

Type Definition for Output API.

```
// Slots API
type Slots = Array<Slot>

type PaxAvailability = {
  type: string;
  name?: string;
  description?: string;
  price: Price;
  min?: number;
  max?: number;
  remaining: number;
};

type Price = {
  finalPrice: number;
  currencyCode: string;
  originalPrice: number;
};

type Slot = {
  startTime: string;
  startDate: string;
  price: Price;
  remaining: number;
  paxAvailability: Array<PaxAvailability>;
};


// Dates API
type DateAvailability = {
  date: string;
  price: Price;
};

type DateInventory = {
  dates: Array<DateAvailability>;
};
```

Output Example for reference :
https://holdup-api.tickete.co/api/v1/experience/56/slots

https://holdup-api.tickete.co/api/v1/experience/56/dates

* **Fetching Period**

```
Every 1 day fetch availability for the next 30 days
Every 4 hours fetch availability for the next 7 days
Every 15 minutes fetch availability for today
```

Note: Output API need not be exactly the same as the type defined above, or need not be in the standard format we use at Tickete. They are just references if you believe you can improve the structure you can go ahead and implement it in an improved format. (as long as you're able to give sufficient reasoning for following a different format)

Constraints

1. **Rate Limit**

   API has a rate limit of 30rpm (requests per minute), try to handle this constraint in your fetching handler

2. **Database**
   Use a SQL DB (Postgres, MySQL, SQLite)
   💡If you're planning to skip deployment, use SQLite.

3. **Stale Data**
   You should ensure you refetch inventory for days as outlined on ***fetching period** above, you're free to choose how you'll ensure this runs periodically.
   Be cautious about the Rate Limit, for this assignment you're only working with two products but on a scaled up setup, you will have thousands of products so you should have a mechanism defined to ensure you dont breach rate limit.

Deployment (⭐Recommended)

Create a Free account on Render (https://render.com/) [or Vercel] both allow for hosting free Postgres DB and host a NodeJS App.

If you already have a different infrastructure at your disposal you may use it.

💡 if for any reason you're not able to deploy on these platforms or any you've previously worked with, you can stick to only submitting Github Repo (you may build this using **SQLite DB** in this case, and include SQLite DB file download link in your repo readme/share via mail.)

Priority / Evaluation Criteria

Attribute	**Evaluation Weight**
Code & DB Quality	30%
Fetching Period (Inventory Sync) Built a handler for syncing inventory for defined fetching period Brownie points for exposing APIs for pausing and resuming Inventory Sync.	25%
Rate Limit Built a handler to ensure the rate limit never breaches regardless of how many products require sync. (for this assignment up to 450 products)	20%
Deployment Deployed the Server, and are able to share an endpoint to access the given data.	15%
Optimizations Any optimizations done to make the app scale-able, there's no need to actually implement optimizations, you may document optimizations on your repo's Readme file	10%