Take-home Assignment

⚠️ Time Limit: 1 hour

General Guidelines

1. Please read this whole document before getting started.
2. Please limit your time spent on this to 1 HOUR (NOT INCLUDING reading the instructions). If you don't finish, that's also okay — we're more looking for your thought process than finishing the tasks!
3. It's okay if you don't get through all the requirements. Once you get something to work, you can think about refactoring/cleaning up the code, etc.
4. Please leave some comments (can be anything) if you want to communicate what you're thinking with a block of code, etc.
5. Requirements are intentionally open-ended. Assume this is a production application, so if you're going to make trade-offs or do something EXTREMELY hacky, have a justification for doing so during our call.
6. We will assess you on the following (in priority order) —

- Do the features work? (Most important)
- Is the code organization clean?
- Is the code written in an easy-to-understand, maintainable, extensible way?

7. Feel free to use any library, any methodology, GPT, Cursor, Google, copy, paste, steal code from anywhere if you need to.

Submission Instructions

At the end of the exercise, please wrap your code up in a ZIP file

1. If you're not sure how to create a ZIP file, please consult Wikihow. Please do not send a public Github repo.
2. Please do not include the node_modules
3. Outside of the time limit, please include a `README.md` file that includes:

- Setup and run instructions: Clear, step-by-step instructions on how to get your service running
- API usage: Instructions on how to call your endpoint(s)
- Reflection: Write a brief paragraph on how you spent the time and what you would do if you had more time (either Vietnamese or English is fine).

Assignment: Build a spaceship booking system

Welcome to SpaceRyder, the future of intercity space travel! With our fleet of three high-speed spaceships, traveling between major cities has never been faster. Our spaceships, each with a

unique and fun name, zip across the skies at 1,000 mph, offering an unparalleled travel experience.

Requirements

Your mission is to develop the backend of the SpaceRyder booking system using TypeScript. You'll be writing an API to handle the logic for requesting rides, canceling trips, and checking the status of our spaceships.

Request Trip

This API takes in two airport codes (start and end), and an ISO datetime representing the desired departure time. It should return a TripStatus object as follows —

- `departureLocationCode`: This is the starting location of the trip.
- `destinationLocationCode`: This is the destination location of the trip.
- `departureAt`: The departure time of the spaceship in ISO-8601
- `arrivalAt`: The arrival time of the spaceship in ISO-8601
- `spaceshipId`: The ID of the spaceship assigned to the trip
- `tripId`: The ID assigned to the trip

If no spaceship is available at the requested time (aka they are all en-route), the system should inform the user of the earliest available boarding time on the earliest available ship. For the first iteration of this, you don't have to make this algorithm super optimized, but after getting something working, you should put some thought into how to handle edge cases for the application logic.

Cancel Trip

This API allows users to cancel a trip. It requires the tripId to be canceled.

Status

This API, given a tripId, returns a TripStatus (e.g. either what was returned in the ride request method OR the current location of the spaceship if the trip is in progress).

Spaceship Fleet

You have three spaceships at your disposal, each with a unique ID. They are all available in the initial state of the world (e.g. SS-001 is initially located in JFK).

```
id,name,location
SS-001,Galactic Voyager,JFK
SS-002,Star Hopper,JFK
```

```
SS-003,Cosmic Cruiser,SFO
```

Locations

For the launch of this star-liner, we only have 3 possible departure and destination points.

```
location,latitude,longitude
JFK,40.6413,-73.7781
SFO,37.6213,-122.3790
LAX,33.9416,-118.4085
```

Current Time

Depending on your implementation, you may need to assume a current time at the initial state. If so, assume 2025-01-01. If not useful, ignore this requirement.

```
2025-01-01T00:00:00
```

Assumptions and Simplifications

For the sake of this exercise, you can simplify the calculation of travel times and distances between airports (hint: you should ask GPT to generate this).

1. Assume straight-line travel (or great circle distance) travel distances at 1,000 mph.
2. Einstein in shambles: Acceleration and deceleration is instantaneous. Spaceships do not have to account for elevation (e.g., they go in a straight line and not a parabolic curve).
3. No traffic: Spaceships instantly take off at the departure time and do not encounter “traffic”.
4. Only one-passenger per spaceship: If a spaceship is departing from a location, it cannot take additional passengers. Reservations are processed in a FIFO queue. If reservation requests are received simultaneously, we can randomly assign the lock to either request.
5. Assume UTC: You can assume all times are communicated in UTC and you do not have to account for timezones.
6. Non-optimal routing: Don’t feel obligated to come up with the optimal routing algorithm for this since it becomes quite complex. It’s OK for spaceships to have downtime or for passengers to wait “longer”. We can improve on this later. We will not ding you for utilizing the fleet inefficiently (anti-matter fuel is free in the future).
7. Feel free to make any assumptions unless otherwise stated.

Tech Stack

- We strongly recommend Typescript for this takehome. If you are not familiar with Typescript, feel free to pick any language of your choosing.
- You must build these interactions as REST, websocket, or graphQL API endpoints (we may want to build a frontend on this later, so do not implement this as a CLI).
- You can use any data store to keep track of the spaceships' statuses and trips (Postgres, etc.).

What We're Looking For

- Since there is no starter code, a big part of this project is being able to leverage resources available to use (Cursor/Claude) to complete the exercise. Feel free to copy/paste code from any resources available to you. As long as you're able to explain the reasoning, we do not care. The end state should be a working backend, but the specific technology choices don't matter too much except for what's listed above in "Tech Stack".
- This is a lot to implement in one hour! Make sure to use all the tools available to you, so that you're not caught up in implementing boilerplate for the one hour.
- Cleanly defining abstraction layers is extremely important. Being able to reason about why you decided on certain assumptions and not others is key.
- Surprise us! Want us to add some bells and whistles? If you're looking for ideas on what to implement if you finish early.