Backend Assessment - Flights

In this assessment, you will write a simple backend JSON API. If you notice something is not working (like any of the links in this document), please contact hello@hatchways.io.

This assessment will be evaluated based on the following criteria:

- Correctness: Is your solution complete and does it pass different test cases?
- Code Organization, Readability, & Maintainability: Is your code easy to read and well organized?
- Code Performance: Is your code efficient? Did you use appropriate data structures?
- Best Practices: Did you utilize good programming practices (write unit tests, avoid anti-patterns)? Did you show a good grasp of your language/framework of choice?
- Completion speed: A fast completion time comparable to the completeness of your solution. This is the least important criteria.

We use the following rubric to evaluate your submission. **Please note that if your submission does not attempt to complete all of the requirements, we will be unable to provide feedback on it.**

Resources

If you are not familiar with how to set up a basic API, we recommend the following resources:

- Python - Flask (Flask JSON API)
- Javascript - Node + Express
- Java - Spring Boot (JSON API)
- Ruby - Rails JSON API

You may use Javascript (Node.js), Python, Ruby, Java, Go, or Rust to complete this assessment. You may use any framework for your language of choice. Please document in a Readme how to start the application.

The Challenge

We want you to create a JSON API that has the following routes:

- **POST /api/tickets:** This route receives JSON data about new tickets being created on your system
- **GET /api/flights?startDate={startDate}&endDate={endDate}:** This route returns JSON data about the flights that are happening between start date and end date

You will be able to read more about the API specifications in this section of this document.

Ticket and Flight Data

In this challenge, you will be dealing with ticket and flight data. Here is some information about the data you will receive:

- Tickets
    - Has a unique ticket id (integer) - no two tickets ever have the same id
    - Has a seat number (alphanumeric) - no two tickets on the same flight have the same seat number
    - Has a cost (integer) in cents
- Flights
    - Has a unique flight number (alphanumeric) - no two flights ever have the same number
    - Has a flight date - the date when the flight is departing - this date can change, in which case the flight has been rescheduled

API Specifications

You will need to build the following API routes in your application:

Route: /api/tickets
Method: POST
Example Body (JSON):

```
{
    "event": {
      "ticketId": 1,
      "flightDate": "2021-11-01",
      "flightNumber": "AC1",
      "seatNumber": "1A",
      "ticketCost": 100000
   }
}
```

Each API request to this route represents a new ticket being created. Your application will be responsible for efficiently storing the ticket and flight data. **In order to store this information, you must store this data in memory (i.e. using variables in your program). To keep the challenge simple and easy to test, you are not allowed to use a database technology such as SQL technologies, NoSQL technologies or caching databases like Redis or Memcache.**

Note: The flight date for a specific flight number can change - this means that the flight's departure date has changed for all tickets on that flight. You can assume that API calls will come one at a time and that the most recent API request is the correct flight data.

Here are the possible response bodies for this API route:

Successful Response:
Return a status code of 200, with a JSON response body:

```
   {
      "status": "success"
   }
```

Unsuccessful Response:
If the API fails for any of the reasons below, do not update any data in your application (i.e. flight dates that change, or filling additional seats).

If the ticket id already exists, return a status code of 400, with a JSON response body:

```
{
  "status": "failed",
  "reason": "ticketId already exists"
}
```

If the seat number is taken for that flight (remember that flight numbers are unique), return a status code of 400 with a JSON response body:

```
{
  "status": "failed",
  "reason": "seatNumber already taken"
}
```

Route 2:

Request:

Route: /api/flights
Method: GET
Query Parameters*:

Field	Type	Description	Example
startDate	String (required)	A formatted date (YYYY-MM-DD) that represents the start date to gather flight information	2021-11-03
endDate**	String (required)	A formatted date (YYYY-MM-DD) that represents the end date to gather flight information	2021-11-05

*Notice that the parameter is a query parameter - you can read more about query parameters here.
**If startDate = endDate, show the results for that one date.

Successful Response:
This API returns aggregated flight data for all the dates in order between start date and end date. A successful response will return a status code of 200 and the data in the following format:

```
{
  "dates": [{
      "date": "2021-11-03",
      "flights": [{
          "flightNumber": "AC1",
          "revenue": 300000,
          "occupiedSeats": ["1A", "10A"]
        }]
    },
    {
      "date": "2021-11-04",
      "flights": []
    },
    {
      "date": "2021-11-05",
      "flights":[{
          "flightNumber": "AC2",
          "revenue": 250000,
          "occupiedSeats": ["7C", "11A"]
        },
        {
          "flightNumber": "AC10",
          "revenue": 270000,
          "occupiedSeats": ["12C", "5A"]
        }]
    }]
}
```

The response type is a list of objects sorted by date, where each object contains:

- The date
- An array of flights that depart on that day, each flight an object containing:
    - The flight number
    - The revenue for that flight (the sum of all ticket costs in that flight)
    - An array (unsorted) of all ticket seat numbers created for that flight (bonus: if you can find a logical way to sort the seats)

This API should be efficient - you should not be looping over all flights in your storage to solve this problem. You can assume that there are a lot of flights in your database, a lot of total tickets in your database, each flight has a relatively small number of occupied seats (<300 seats), and that this API will only fetch a small date range.

Unsuccessful Response:
If startDate or endDate are not given by the request, return a status code of 400, with a JSON response body:

```
{
  "status": "failed",
  "reason": "startDate is empty"
}
```

If startDate or endDate are not properly formatted, return a status code of 400, with a JSON response body:

```
{
  "status": "failed",
  "reason": "startDate format is invalid"
}
```

If the endDate is before the startDate, return a status code of 400, with a JSON response body:

```
{
  "status": "failed",
  "reason": "endDate cannot be before startDate"
}
```

Example Test Case

Here are a few example test cases. Imagine these API calls were made one after each other:

API Call	Request Body	Expected Response Status Code	Expected Response Body
1. POST /api/tickets	JSON input	200	JSON output (success)
2. POST /api/tickets	JSON input	400	JSON output (invalid Id)
3. POST /api/tickets	JSON input	400	JSON output (invalid seat)
4. POST /api/tickets	JSON input	200	JSON output (success)

5. GET /api/flights?startDate=2021-11-01&endDate=2021-11-03		200	JSON output
6. POST /api/tickets *Notice the flight date changes	JSON input	200	JSON output (success)
7. POST /api/tickets	JSON input	200	JSON output (success)
8. GET /api/flights?startDate=2021-11-01&endDate=2021-11-03		200	JSON output

Testing

An important part of development is testing. In this step, we want to see tests written for your routes. We recommend in your tests to check your storage variables to see if your API is working as expected.

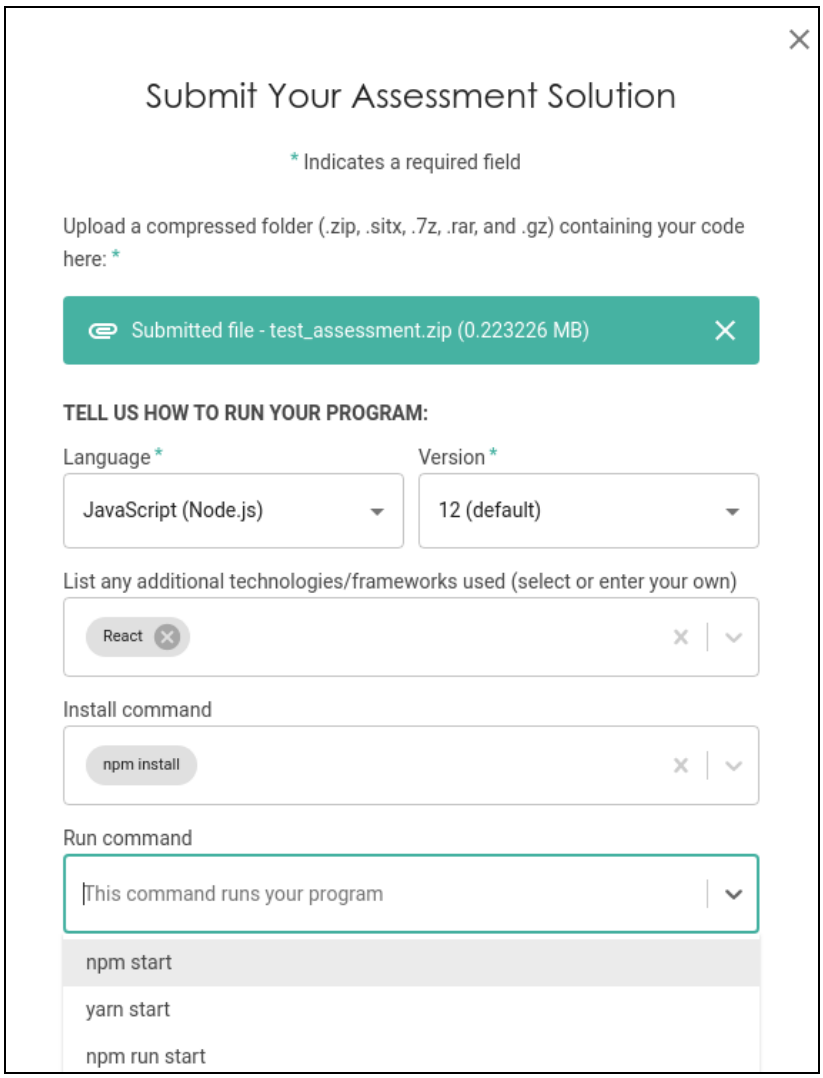
Submission Details

Please submit your code in a compressed folder on the Hatchways platform. The max submission size is 5MB.

Upon clicking the submission button, you will see a form as pictured below. We need this information to be able to test your application.

1. Choose which **language** and additional technologies you used to develop your solution. Be sure to select the appropriate version for the language you have used.
2. Provide us with the **install command**, the **run command**, and the **port** that you used to run your application.
3. If you cannot find your commands in our suggestions, simply type your own and select “Use command”.

Please note that these commands will be used to run automated tests, so filling in every relevant field and providing accurate commands will allow you to receive feedback more quickly on your submission. If you have any notes to provide about your submission, please put them in a README, not in the submission form. Additionally, note that the install and run commands will be run from the root level of your submission, so please organize your files accordingly.



Do not submit any built folders, since the compressed folder will be too large. **Do not submit your external dependencies (like the node_modules folder), since the compressed folder will be too large. We will be installing your dependencies before we run your code.**

If your submission is too big and you can't figure out how to compress, you are welcome to email your solution to hello@hatchways.io. Please include your name, and use the email you signed up with on the Hatchways platform. Use the subject line “Back-end Assessment Submission”.

Public Repositories

Do not post your solution to a public repository. We understand that you may want to share projects you have worked on, but many hours go into developing our tools so we can provide a fair skills evaluation.