Introduction

This assessment is for candidates applying for the Backend Developer role at Asite 3D Repo. It is designed to take no longer than three hours, but you are welcome to spend as much or as little time as you wish. Please detail any additional work you would have done if you had more time, either as inline comments or supplementary documents. This helps us understand what your complete solution would look like.

You are expected to complete this task independently without assistance from other individuals (Googling is, of course, allowed).

A skeleton structure is provided to help you get started. This includes an Express JS service implementation, a test folder using the Jest framework initialised with an in-memory MongoDB for testing any implementations as required. Sample tests are included to demonstrate how to use this setup.

The provided code is intended to assist you in getting started, but it is not mandatory to use, you are also welcome to restructure anything as you see fit. If you are more comfortable with other libraries or languages, feel free to start from scratch. The solution does not need to be in JavaScript – we are primarily interested in the quality of your implementation rather than your proficiency with any particular language.

Please approach this assessment as if you are writing production-ready code. You will be evaluated on the robustness and maintainability of your solution. While we encourage Test-Driven Development (TDD), we understand that time is limited. Therefore, we ask that you include at least one full unit test and one full system test for your solution.

If you have any questions or concerns, please reach out to your point of contact, and we will be happy to assist you.

The Task

The objective is to create a set of RESTful API endpoints for an event ticketing system. The primary functionality should include:

- Adding a new event
- Recording a ticket transaction for an existing event
- Retrieving statistics around ticket sales

The expected inputs and outputs for these endpoints are described below. The design of the routes, database schema, and system architecture is left to your discretion. You may also include any additional libraries/frameworks as required.

Add a new event

You will create an endpoint to add events. The expected input data for this endpoint is as follows:

```
{
   "name": "Charity Auction",
   "date": "31/10/2024",
   "capacity": 100,
   "costPerTicket": 5
}
```

NOTE:

- Multiple events can have the same name.
- Only one event can occur on a given day.

Upon successful creation, the event should be stored in the database, and the service should return a response containing a unique identifier for the event.

Add ticket transaction

Once events are established, you'll create an endpoint to record ticket transactions. The expected input data for this endpoint is as follows:

```
{
  "event": <Event ID you would've returned from the previous endpoint>,
  "nTickets": 3
}
```

NOTES:

- The event provided must be a recognised (existing) event.
- If the event is sold out, or if the transaction would exceed the event's capacity, the system should return an error response.

Retrieve Ticket Sales Statistics

The final endpoint should generate statistics regarding ticket sales. Specifically, this endpoint will return monthly statistics for all events held within the past 12 months.

The output should look like this:

```
[
  {
    "year": 2024,
    "month": 9,
    "revenue": 10203,
    "nEvents": 10,
    "averageTicketsSold": 40
  },
 {
    "year": 2024,
    "month": 8,
    "revenue": 0,
    "nEvents": 0,
    "averageTicketsSold": 0
  },
  // …similar output for the July 2024 to October 2023.
]
```

NOTES:

- **revenue**: The total revenue from all tickets sold for events occurring within the month.
- **nEvents**: The number of events that took place within the month.
- **averageTicketSold**: The average percentage of tickets sold for events held during the month.