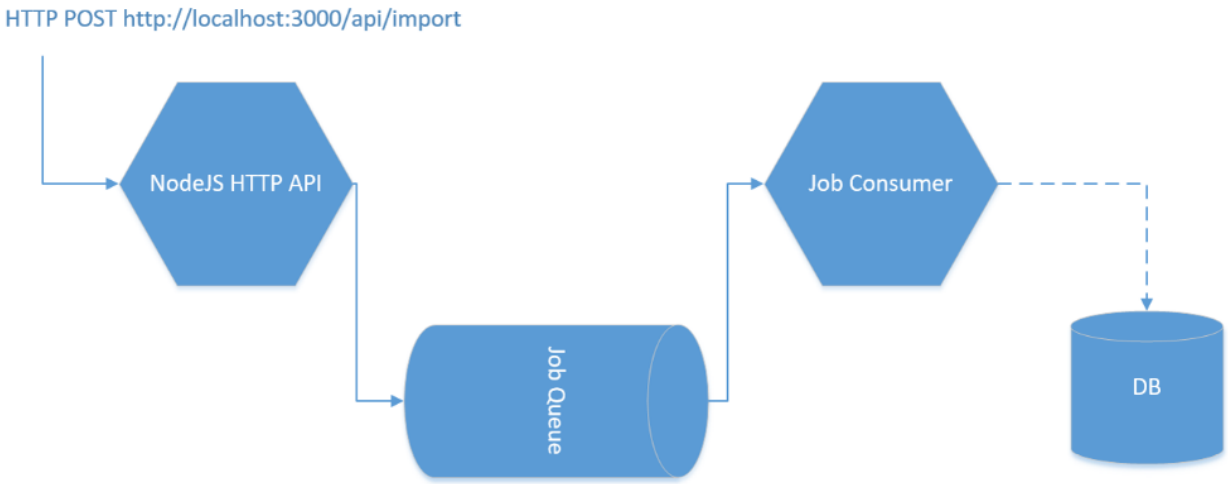
# Hiring Assignment : Hotel Reservation

**Objective:** Develop a simple HTTP API service with unit tests.

You can choose how complicated or simple the solution will be, how much time you want to spend on it. For example, you might log with console.log or a dedicated logging framework with an adapter for the console. In the simplest solution, for example, every part in the illustrative diagram below is an in-process in-memory representation.

**An illustrative diagram of the logical flow is attached**




**A scenario that should work:**

1. Accept (and log to console) incoming HTTP POST request to http://localhost:3000/api/import with the following JSON object in the message body

```
{
    "hotelId": "hotel1",
    "reservations": [
        { "reservationId": "reservation1", "name": "guest1" }
    ]
}
```

2. Place (and log to console) a job item to a queue (can be as simple as in memory array or Redis, RabbitMQ etc) that represents this message
3. Accept (and log to console) another incoming HTTP POST request to http://localhost:3000/api/import with the following JSON object in the message body

```
{
    "hotelId": "hotel1",
    "reservations": [
        { "reservationId": "reservation1", "name": "guest1" },
        { "reservationId": "reservation2", "name": "guest2" }
```

```
    ]
}
```

4. Place (and log to console) this message also to the same queue
5. There should now be 2 items in the job queue (unless first has already been consumed)
6. Another process or service consumes from the job queue on a simple FIFO basis. Can be implemented as periodically polling the queue (e.g. every 30 seconds)
7. Processing of every job item should take at least 10 seconds (can be just a sleep command)
    a. Log to console before starting to "process" the job item
    b. Log to console after finishing "processing" the job item
8. While a job is being processed the system should be able to accept new incoming messages and place them to the queue

The console log should look something like this (the exact order and formatting will likely be different):

```
[12:00:00] Received HTTP request - { "hotelId": "hotel1",
"reservations": [ { "reservationId": "reservation1", "name": "guest1"
} ] }
[12:00:01] Push job to queue - { "hotelId": "hotel1", "reservations":
[ { "reservationId": "reservation1", "name": "guest1" } ] }
[12:00:02] Received HTTP request - { "hotelId": "hotel1",
"reservations": [ { "reservationId": "reservation1", "name": "guest1"
}, { "reservationId": "reservation2", "name": "guest2" } ] }
[12:00:03] Push job to queue - { "hotelId": "hotel1", "reservations":
[ { "reservationId": "reservation1", "name": "guest1" }, {
"reservationId": "reservation2", "name": "guest2" } ] }
[12:00:04] Started processing job - { "hotelId": "hotel1",
"reservations": [ { "reservationId": "reservation1", "name": "guest1"
} ] }
[12:00:05] Started processing job - { "hotelId": "hotel1",
"reservations": [ { "reservationId": "reservation1", "name": "guest1"
}, { "reservationId": "reservation2", "name": "guest2" } ] }
[12:00:06] Received HTTP request - { "hotelId": "hotel2",
"reservations": [ { "reservationId": "reservation3", "name": "guest3"
} ] }
[12:00:07] Push job to queue - { "hotelId": "hotel2", "reservations":
[ { "reservationId": "reservation3", "name": "guest3" } ] }
[12:00:08] Started processing job - { "hotelId": "hotel2",
"reservations": [ { "reservationId": "reservation3", "name": "guest3"
} ] }
[12:00:14] Finished processing job in 10 seconds - { "hotelId":
"hotel1", "reservations": [ { "reservationId": "reservation1", "name":
"guest1" } ] }
[12:00:15] Finished processing job in 10 seconds - { "hotelId":
"hotel1", "reservations": [ { "reservationId": "reservation1", "name":
"guest1" }, { "reservationId": "reservation2", "name": "guest2" } ] }
[12:00:18] Finished processing job in 10 seconds - { "hotelId":
"hotel2", "reservations": [ { "reservationId": "reservation3", "name":
"guest3" } ] }
```

Mandatory requirements:

- Unit tests - done
- Readme - how to set up, build, run the system, how to run tests - done
- We should be able to follow the steps in the documentation and run the system. - done
- Use NodeJS -done
- Code with commit history available on git repository (can be local repo zipped and attached to reply email) -done

Optional bonus requirements:

- As step 9) store reservations in a database (file, Mongo, SQLite) - done
- HTTP message handling should be idempotent – if same data is sent multiple times, it should not result in duplicate database entries -done