



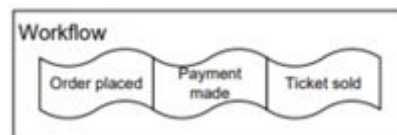
Ticket booking system

Cloud Native microsystem service

Group 18 | Service based software architecture | 23/04/2020

## Introduction

The goal of this project is to create a Ticket Booking system. It is composed of 3 microservices that are linked and work together through the following workflow:

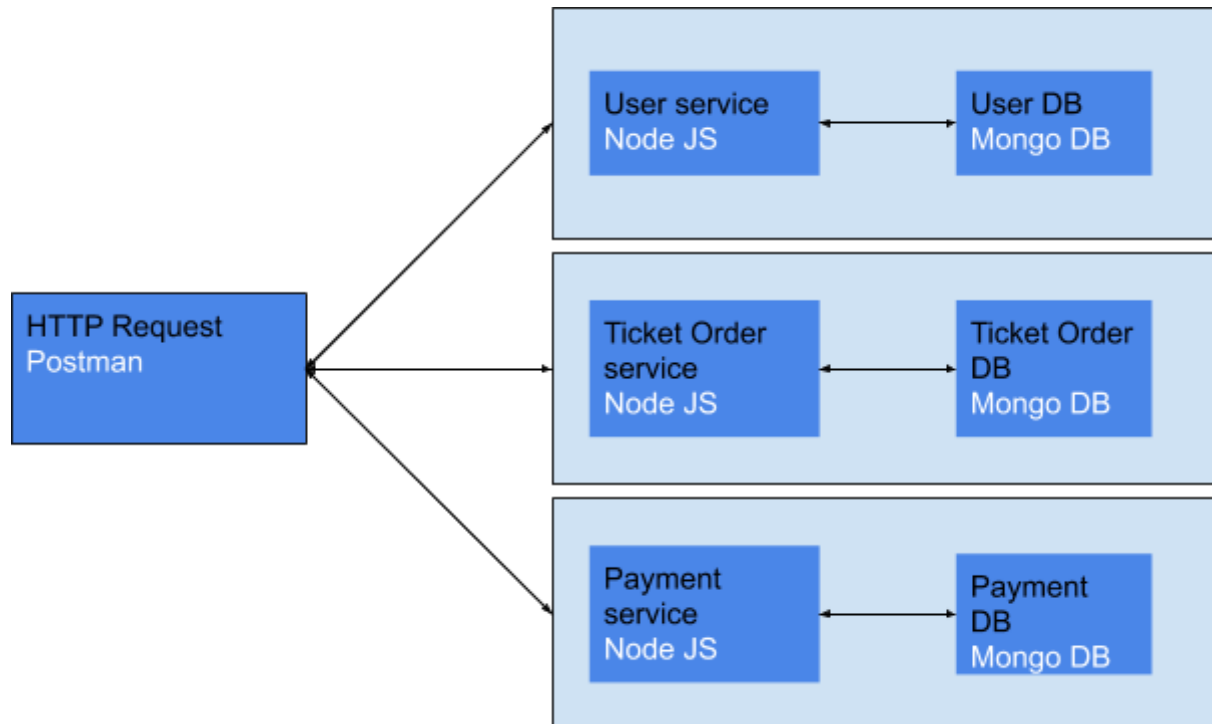


The first step is that the client should be able to manage his account, by creating or deleting them from the service. The second step is to be able to book the tickets and create an order for the user. The final step is payment upon the order.

## Architecture and design pattern

The project is encapsulated with Docker, each microservice is its individual container. By this way each microservice is independent between each other, and has its own database. The benefit of this approach is that the project is more scalable than a classic monolithic app. To

build the project we use Docker and create a docker container for each of these services with their own database.



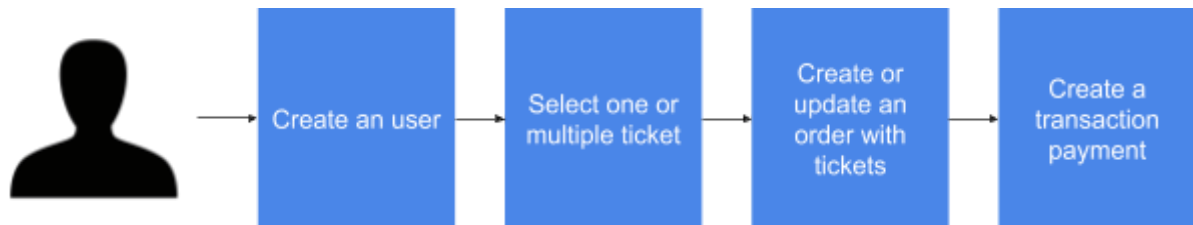
This architecture allows us to be able to deploy the application as one package even though the different parts are initially separated.

We have decided to use Node.js and MongoDB to code the different parts of the project, because the different members were more at ease with it, and these languages allowed us to fit with the requirements. It was then possible to create

Every microservice is composed of the following parts and components:

- The docker related components, which allow the communication with others microservices.
- The database containing the data of the system's component.
- The app which takes care of all the internal operations related to the function of the microservice.

## User story



Firstly, the user creates a user object.

Then he can order a ticket, this action will add the ticket to the ticket list of the current order of the user and creates the order if it does not exist.

Finally the user can proceed to the payment, it will update the payment status of the order and a transaction will be created so that it will get stored in the database to get the history of orders.

## Source code

The project's source code is available on the following link :

<https://github.com/bengrc/TicketBooking>

## Conclusion

The point of the project was mainly to create a stable and useful architecture to support the communication between microservices. The security and stability of the communications were crucial in order to make the project work. Microservices are communicating with protocols, and the data circulates in the different databases.

## Deployment & Usage

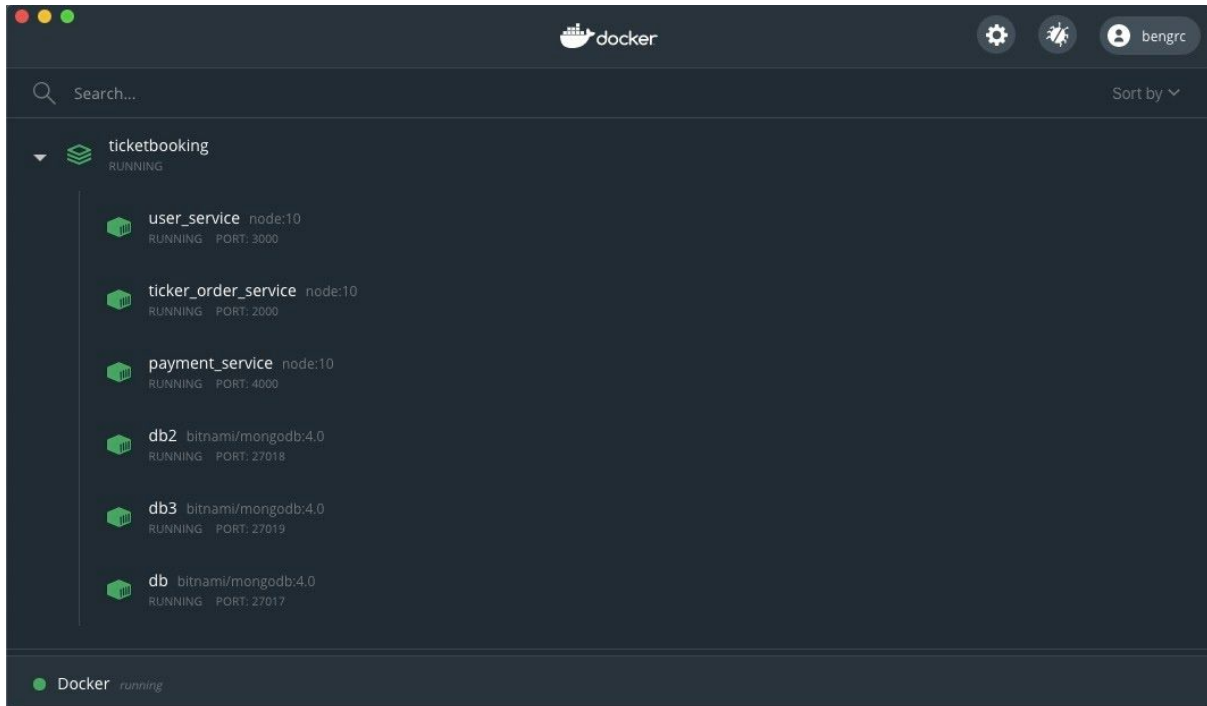
1 - Clone the .git repository available [here](#).

```
git clone https://github.com/bengrc/TicketBooking
```

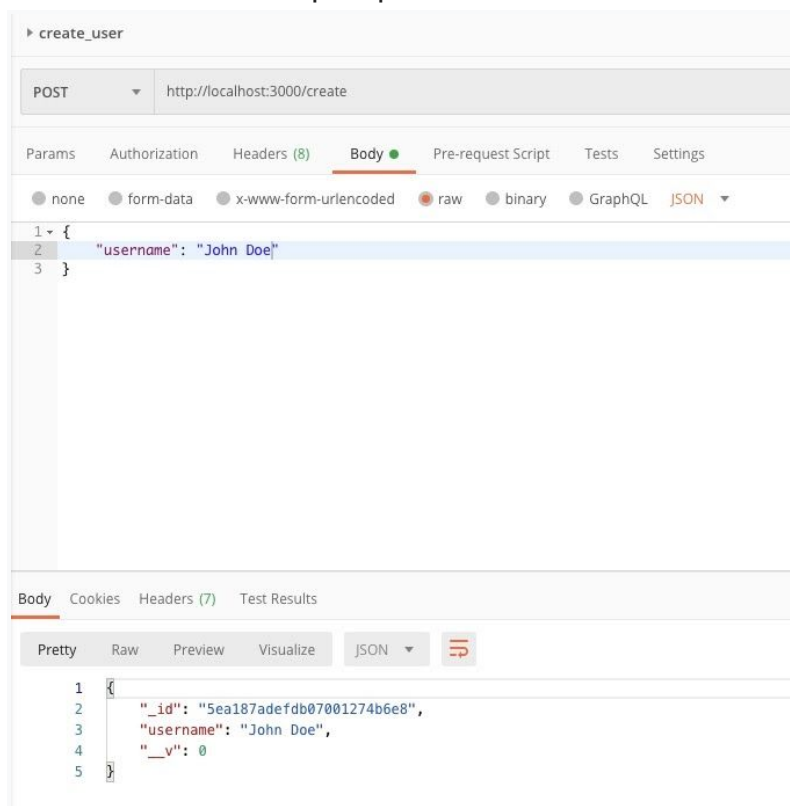
2 - Run the project using the following command :

```
npm install && npm start
```

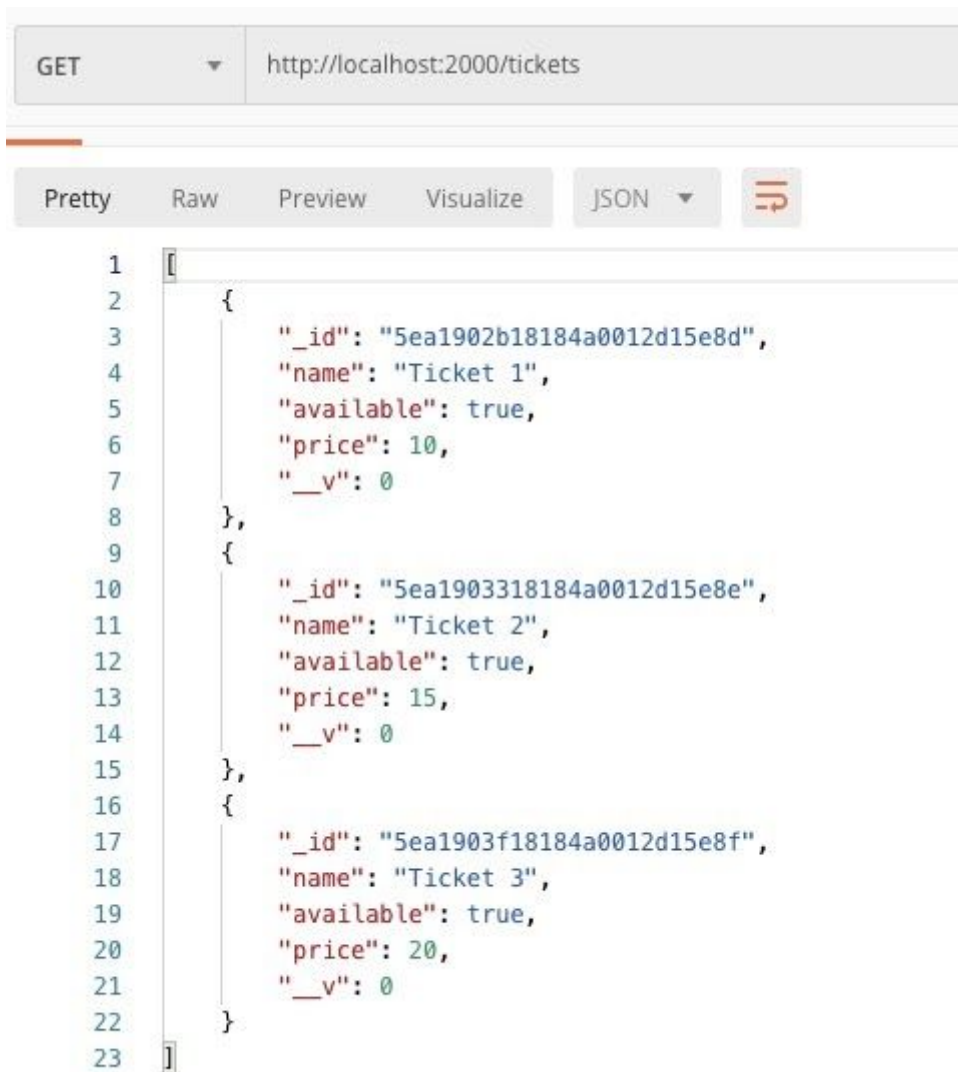
3 - After launching npm, we can notice (with Docker Desktop for example) that the different services are encapsulated and deployed in Docker containers.



4 - Now we can start having fun with our microservices. Let's start by creating a user using the associated service : User Service. To do this, we use Postman which allows us to make Http requests to our different microservices.



5 - Now we can have access to the different tickets on sale by requesting the ticket order service.



The screenshot shows a REST client interface. At the top, a GET request is made to the URL `http://localhost:2000/tickets`. Below the request bar, there are tabs for 'Pretty', 'Raw', 'Preview', and 'Visualize', with 'Pretty' selected. To the right of these tabs is a dropdown menu set to 'JSON' and a refresh icon. The response area displays a JSON array of three ticket objects, formatted with line numbers 1 through 23 on the left margin. The JSON data is as follows:

```
1  [
2    {
3      "_id": "5ea1902b18184a0012d15e8d",
4      "name": "Ticket 1",
5      "available": true,
6      "price": 10,
7      "__v": 0
8    },
9    {
10     "_id": "5ea1903318184a0012d15e8e",
11     "name": "Ticket 2",
12     "available": true,
13     "price": 15,
14     "__v": 0
15   },
16   {
17     "_id": "5ea1903f18184a0012d15e8f",
18     "name": "Ticket 3",
19     "available": true,
20     "price": 20,
21     "__v": 0
22   }
23 ]
```

6 - The user can now choose a product to add to his shopping cart, using the following query.

The screenshot displays a REST client interface. The top section shows a POST request to the URL `http://localhost:2000/ticket/book`. The 'Body' tab is selected, showing a JSON payload with the following structure:

```
1 {  
2   "ticketId": "5ea1903f18184a0012d15e8f",  
3   "userId": "5ea187adefdb07001274b6e8"  
4 }
```

Below the request, the 'Response' section is visible, showing the JSON response in 'Pretty' format:

```
1 {  
2   "_id": "5ea1927d18184a0012d15e91",  
3   "userId": "5ea187adefdb07001274b6e8",  
4   "ticketsBooked": [  
5     {  
6       "_id": "5ea1902b18184a0012d15e8d",  
7       "name": "Ticket 1",  
8       "available": true,  
9       "price": 10  
     }  
   ]  
}
```

7 - After that the user can choose to modify it shopping card, if the user wants to buy another ticket, for example . Here, we consider that the user will just pay, by doing that, he calls the payment service with the following request :

The screenshot displays a REST client interface. The top section shows a POST request to the URL `http://localhost:4000/transaction/create`. The 'Body' tab is selected, showing a JSON payload:

```

1 {
2   "orderId": "kyeu345zjhy73Hksoj83oks",
3   "userId": "okpokczpock5367jxozjxoizoi",
4   "ticketsBookedIds": ["pok456lpç3I0ioikdo3IJkjdnsz", "opkpokpokpokpokpkpsxqsok"],
5   "totalPrice": 10
6 }

```

Below the request, the 'Body' tab of the response is shown, displaying a JSON object with the following structure:

```

1 {
2   "ticketsBookedIds": [
3     "pok456lpç3I0ioikdo3IJkjdnsz",
4     "opkpokpokpokpokpkpsxqsok"
5   ],
6   "_id": "5ea1970482f48f00125b2695",
7   "orderId": "kyeu345zjhy73Hksoj83oks",
8   "userId": "okpokczpock5367jxozjxoizoi",
9   "totalPrice": 10,
10  "__v": 0
11 }

```

8 - Here, we can see that the payment is done and a trace of the transaction has been recorded on the database associated with the service.

#### ***Authors :***

Benjamin Gracia - 19129135

Maxime Boiteau -19129055

Abde Namir - 19129126

Matthew Livoti - 19129051