Software Architecture Project 2023/24

Flight tickets

Author: Mattia Dei Rossi

Matricolation: 885768

Year: 2023/24


VENETIARVM VNIVERSITAS
IN DOMO FOSCARI

Index

Topic

Implement an IT system to manage tickets of flights.

Approach

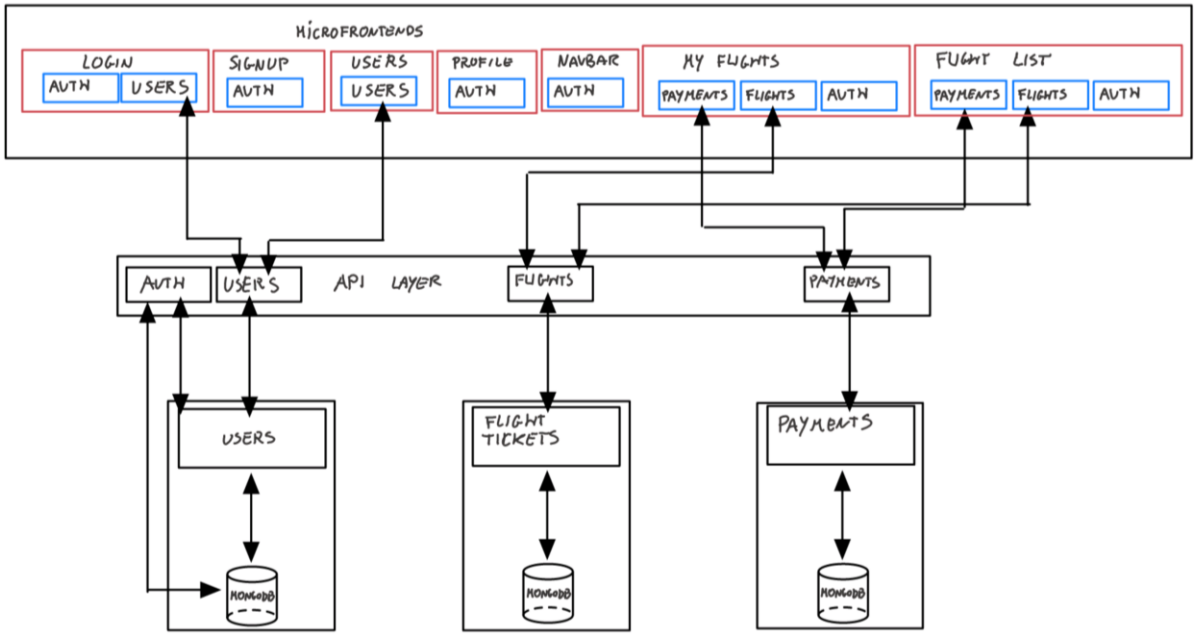
To implement this type of IT system has been decided to use a microservices architecture with microfrontends.

The frameworks chosen are:

- Angular for microfontends.
- Nodejs for microservices.
- MongoDB for databases.
- Docker compose to manage entire IT platform.

Each microfrontend component interact with backend using asynchronous observables components via synchronous communications through REST APIs.

Software architecture schema



Microservices

Microservices are:

- API Layer Gateway: it redirects the requests to the correct services. It manages the Authentication of users, in particular sign up, sign in and auth functions.
- Users: it communicates with a database to manage the users on the IT platform.
- Flights: it communicates with a database to manage the flights on the IT platform.
- Payments: it communicates with a database to manage the payments on the IT platform.

Microfrontends

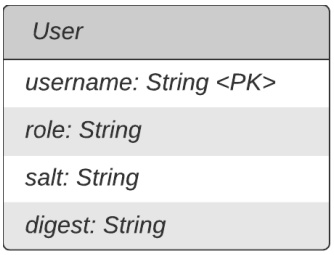
Microfronteds are:

- Components:
  - Flight list: it shows available flights, search and buy flights.
  - Login: it manages the login phase
  - My-flights: it shows the flights ticket purchased by user logged in
  - Navbar: it manages the menu of the entire system
  - Profile: it shows information about the user logged in
  - Signup: it manages the register of new users
  - Users: it is accessible only via “admin” user. It permits to delete users.
- Services:
  - Auth-guard: guarantee the correct authorization in the angular routing system.
  - Auth: it manages the token of user logged in
  - Flights: it communicates with flight service
  - Payments: it communicates with payment service
  - Users: it communicates with users service

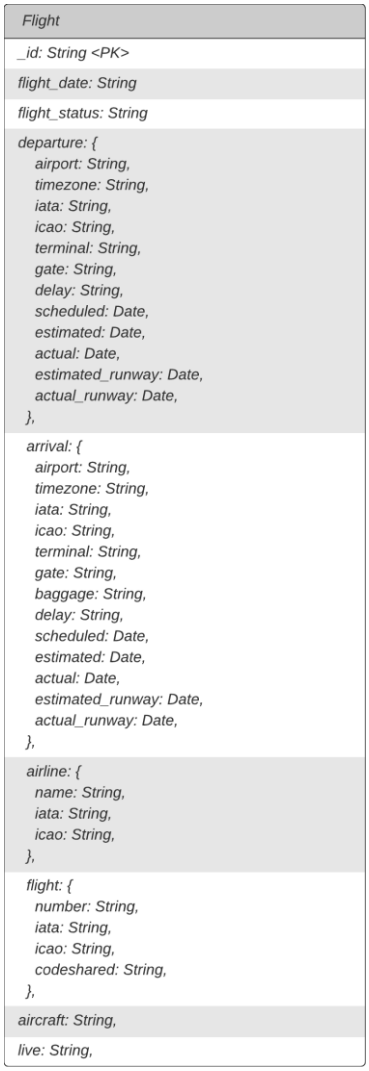
Databases

Databases are:

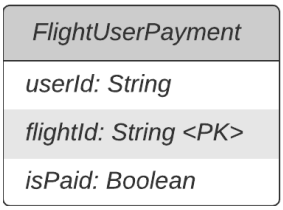
- MongoDB-users:



- MongoDB-flights:



- MongoDB-payments:

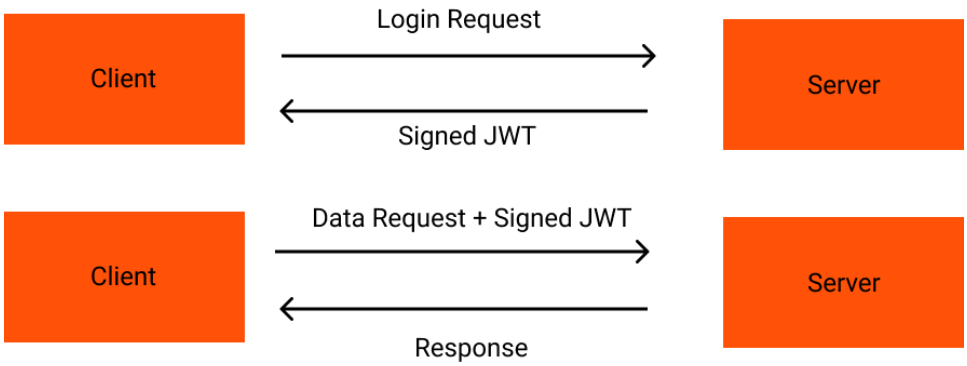


API

Please download the repo and open backend/apidoc/index.html with your favorite browser.

Authentication

Authentication is handled through a JSON Web Token (JWT) generated after logging in by user.



The operations performed for token generation are:

1. Executing a GET request to the /login endpoint with http header containing the field

   authorization: 'Basic ' + btoa(username + ':' + password).

   btoa() creates an ASCII string with Base64 encoding from a binary string.

2. Decoding the **authorization** field using the **passportHTTP.BasicStrategy()** function.
3. Searching the DB for the requested user.
4. Creation of token starting from username, role, id.
5. Signing token using the jsonwebtoken.sign() function that requires a token, a password

   let **token_signed** = jsonwebtoken.sign(tokendata, JWT_SECRET, { expiresIn: '1h' });

6. After the signed token is generated, it is sent for each future request in the header field

   authorization: 'Barer **token_signed**'.

CORS Policy

Cross-origin resource sharing (CORS) is an extension of the same-origin policy. You need it for authorized resource sharing with external third parties. The actual policy is described in the following schema.