NucleUS

Group 5

41009 - Engenharia e Gestão de Serviços

Universidade de Aveiro

2023/2024

Bárbara Moreira 104056, David Bicho 93215, Diogo Correia 90327, Lara Rodrigues 93427

DETI - Departamento de Eletrónica, Telecomunicações e Informática

Contents

List of Figures

Introduction

1.1 Motivation

This project arises from the need to revamp how University Associative Groups, or 'Nucleus,' promote the events and workshops they manage. Mainly, we introduce a concept of gamification to those events, that contributes to keeping students engaged, even on pedagogic activities that would, otherwise, for the most part, have a lower attendance rate.

As students ourselves, we understand the struggle to both keep track of future activities and to find the motivation, mostly on busy weeks, to attend to some.

1.2 Objectives

The goal is to have a system that allows students to find the latest events from the Associative Groups they follow and earn Points for attendance, which can later be exchanged for prizes, whose management is out of the scope of this project. With this in mind, the following depicts what we will tackle along the way:

- Elaborate a mockup of our web application
- Work on each third party service separately
- Work on the application's frontend and backend API
- Use Docker to compose the deployment of our web application and required services
- Use Kubernetes to finally deploy it to a server and under a domain provided by the subject's professor

Architecture

This project is composed by a Web Application with a backend API that manages the communication with three third party services. The Web Application as a whole is implemented using Python's framework Django. We chose Django for this module because of the possibility to manage several apps and its versatility to develop both frontend and backend. The frontend - the pages rendered to the user - never interact directly with the Services.

Every request made on the client side, is always done through the backend API. This API exposes convenient endpoints to process user login, through the Authentication Service, events creation and listing, through the Events Service, and user points management, through the Points System Service.

All the three Services were developed using Python's framework Flask. They all implement an API that handles HTTP requests and manage a MySQL database.

The Authentication Service, other than implementing a standard password-based personal authentication, also allows for the usage of Universidade de Aveiro's Identity Provider (former for associate groups, latter for students). Because, for the purpose of this project, we were given a Development Access Token to the UA IDP, the authentication is only possible in the context of our local machine. To tackle this issue, we created a proxy that runs locally and redirects the user authentication data back to the Authentication Service (this is only valid as a proof of concept).

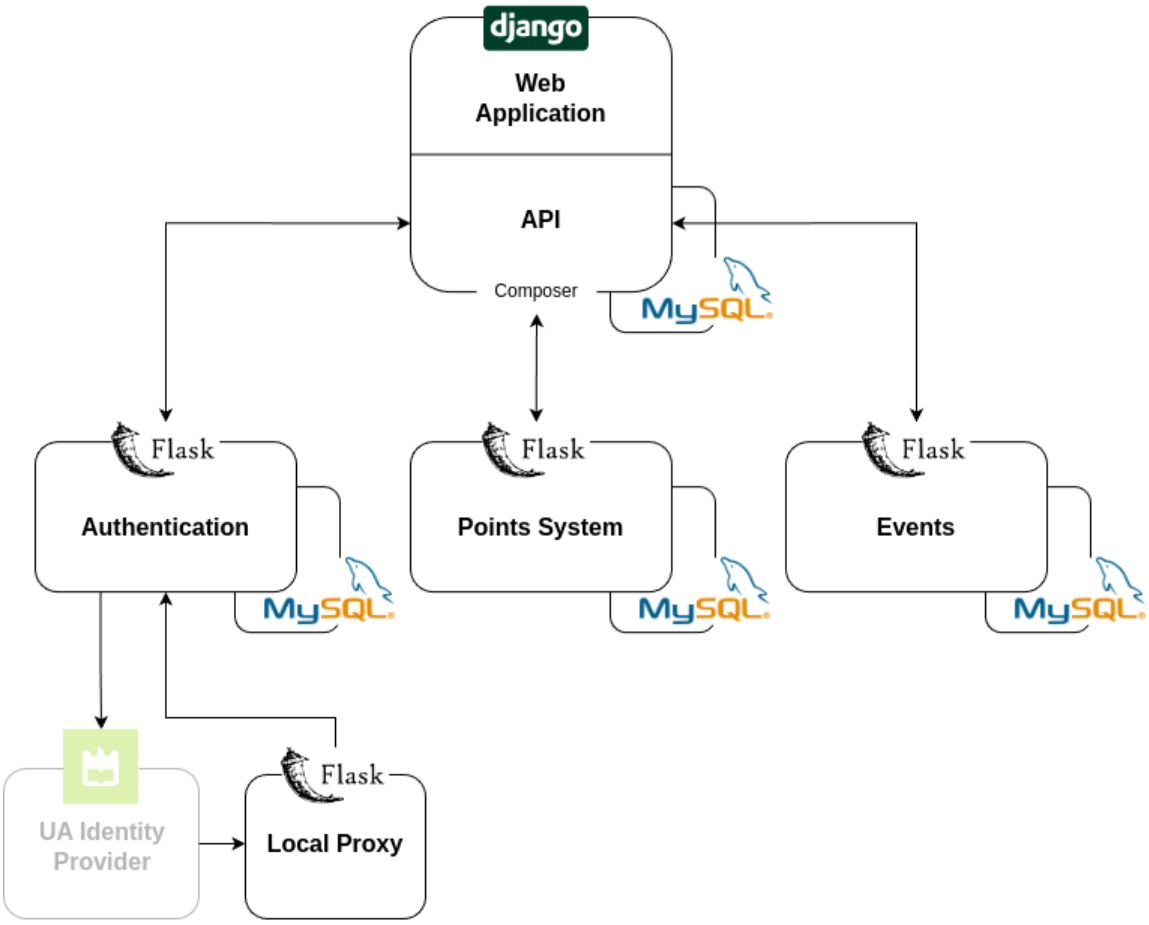

Figure 2.1: Project Structure

Services

3.1 Authentication Service

This service is responsible for the authentication of users(students) and nucleus. The authentication of users is done using UA IDP. There is also a feature of registration for new Nucleus and for users first time accessing the WebApp so they can choose the nucleo that they belong to.With the authentication of UA IDP is possible to retrieve the email, access token, refresh token and other info, that is then saved and used for inner processes.

3.1.1 Endpoints

\ - The redirect of the UA IDP, if successfully logged in sets up the cookies

`\v1\idp` - Hnadles the callback from the IDP after authentication. Updated the database and redirects to CheckUser.

`\v1\signin` - If there is a POST to this endpoint it means its a nucleo account that is trying to logIn and its activated the process to authenticate the nucleo. Otherwise it redirects to the UA IDP authorization URL for authentication of the user(student).

`\v1\register` - Handles user registration, associates the user(student) to a specific nucleo.

`\v1\check` - Checks, if the user(student) is logged in, has a nucleo associated. If the user has it is redirected to the homePage, otherwise it is redirected to the register page so he can choose a nucleo.

`\v1\user` - Retrieves user information

`\v1\nucleus` - Returns all nucleus in the database

`\v1\students` - Retrieves users belonging to a specific nucleo

3.1.2 Integration with Composer

All endpoints are working and integrated in the composer.

3.2 Points-System Service

To gamify our system and make it more adaptive for students, we implemented a **points system**. This system is based on the attribution, addition, and removal of points to an entity, object, or type, which in our application translates, respectively, to an user, event and students representative group. The service still provides the possibility to view standings and point allocation history.

3.2.1 Endpoints

`PATCH /points/entity/<entity_id>?object=object_id` - Add or remove points to an entity

`GET /points/entity/<entity_id>/history` - History of points transactions

`POST /points/type` - Add a new points type

`DELETE /points/type` - Remove type of points

`GET /points/standings` - Get general standings of all entities- Filter events by date

`GET /points/standings?type=<points_type>` - Get standings of a points type

3.2.2 Integration with Composer

The service was fully functional and tested individually. When integrating with the webapp we implemented the following functionalities:

- **Standings**: User can see the standings of all users logged in the platform.
- **Attribution Of Points**: The number of points for the user can be updated. This is not fully integrated with the frontend, but functional using Postman. User can also visualize the current points they have.

3.3 Events Service

The points service was developed to allow an organisation to create an event by entering the name, date, type of event, description, value and number of points it would give. This service also allows users to search by name, date and type of event.

3.3.1 Endpoints

`POST/v1/events` - Create an event

`GET/v1/events` - Return all of the events

`GET/v1/events/<int:event_id>` - Filter events by id
`GET/v1/events/date?<string:date>` - Filter events by date
`GET/v1/events/type?<string:type>` - Filter events by type
`GET/v1/events/location?<string:location>` - Filter events by location
`GET/v1/events/price?<float:price>` - Filter events by price

3.3.2 Integration with Composer

For lack of time and due to some errors that occurred in the implementation during the integration with the composer, only the creation of events and making them available to the user was functional.

`POST/v1/events` - Create an event
`GET/v1/events` - Return all of the events

3.4 Web Application

Our webapp has communication with all 4 built services, it integrates with these to authenticate users, fetch and update user points, and manage user standings, post and view nucleo's updates, and so on. The Figure 3.1 serves as an example to ilustrate the integration of the authentication service when login in as a students representative group.

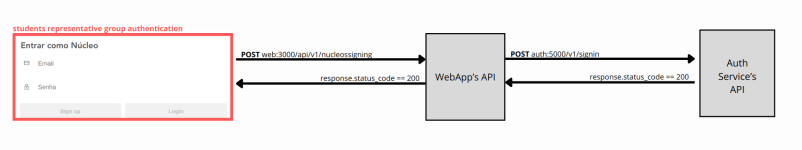

Figure 3.1: Login Messages

3.4.1 Endpoints

As previously mentioned in each service, only a few of the original service's endpoints are integrated in webapp's api.

`GET /v1/auth` - Redirects to the UA's idp service, through the Auth Service

`GET /v1/institutions` - Returns a list of all institutions

`POST /v1/register` - Calls the Auth service and associates a student to a students representative group

`GET/v1/events` - Calls the events service and returns a list of all events

`POST/PATCH/DELETE /v1/events/<int:event_id>` - Calls the events service and publishes/updates/deletes a given event

`GET/PATCH /v1/entity` - Calls the Points service and returns or updates the number of points for an user

`GET /v1/standings` - Calls the Points service and returns or the current point's standings

`GET /v1/nucleos` - Calls the Auth Service and returns a list of all the logged in students representative group

`POST /v1/nucleossignin` - Calls the Auth Service and loggsin a new students representative group

`GET /v1/user` - Calls the Auth Service and returns the users information (id, email and nucleo associated)

`GET /v1/students/<int:nucleo_id>/` - Calls the Auth Service and returns the users belonging to a nucleo

Deployment

The deployment of this project was divided between two platforms, Docker and Kubernetes.

The first approach was to use Docker Images and the Compose tool to aggregate all our services and easily deploy them on our machine. All four main modules - the three services and the web app - as well as the databases for each module and the reverse proxy Nginx were containerized.

The routing for each service is made with Nginx, which uses the container names to dynamically identify the hosts. The configuration is as follows:

```
events {
    worker_connections 1024;
}

http {
    server {
        listen 80;

  location /auth/ {
    proxy_pass http://auth:5000/;
            ...
  }

  location /events/ {
    proxy_pass http://events:5001/;
            ...
  }

  location /points/ {
```

```
    proxy_pass http://points:5002/;
            ...
  }

  # kubernetes deployment only (with docker-compose access
    with localhost:8090)
  location /adminer/ {
    proxy_pass http://adminer:8080/;
            ...
  }

  location / {
    proxy_pass http://web:3000/;
            ...
  }
    }
}
```

All services were now accessible through the port 80, on different endpoints. This is valid on the client side. On the server side, services still need to be accessed through their original host and port, as we can see on these environment variables for the Web App API:

```
API_URL=web:3000/api/v1
AUTH_SERVICE_URL=auth:5000/v1
EVENTS_SERVICE_URL=events:5001/v1
POINTS_SERVICE_URL=points:5002/v1
```

The next step was to deploy to the web. For this, we used Kubernetes that creates a cluster of pods, in which we find the containers from before, created with the same Docker images and Dockerfiles. Because we already had a compose.yaml describing the configurations needed for the containers, we followed that structure and migrated everything to a deployment.yaml that makes sense in the context of a Kubernetes cluster. For each module, database and Nginx, we added a Service and a Deployment component to manage routing and pod replicas. To handle external routing, we added an Ingress component with Traefik. The volumes for the databases were added to a storage.yaml file, with PersistentVolumeClaim components.

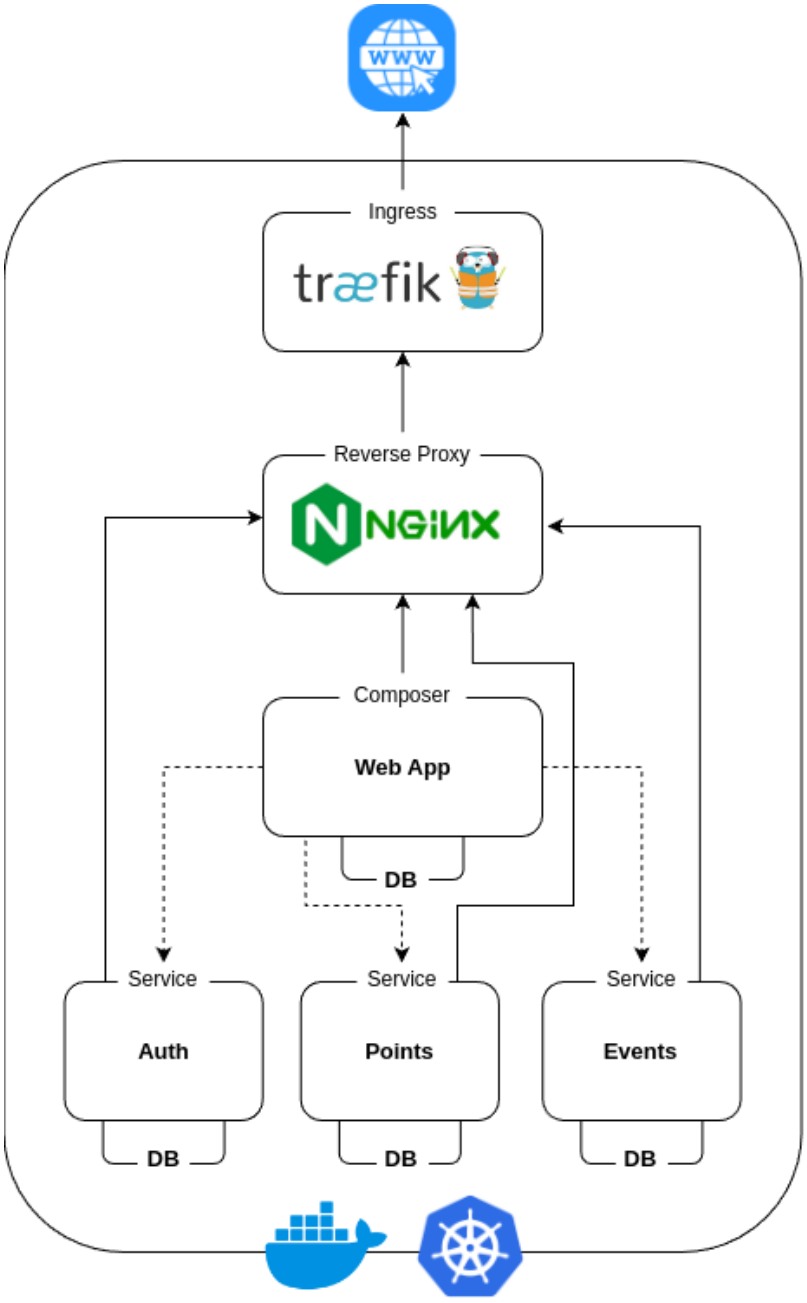

Figure 4.1: Deployment

Conclusion

Platforms like Docker and Kubernetes provide a great layer of abstraction when it comes to handling and managing several services. It is enough to provide good configurations and understand how containers communicate with each other to get a conglomerate of applications to work.

5.1 Work Division

The work in this project was fairly equally divided between the four members of the group: 25% workload each.

5.2 Self Evaluation

We believe our team worked as expected, the work was well divided, and we met the requirements. Therefore, we believe that our work can be evaluated with a grade of 17.

5.3 GitHub Organization

For the project, we created a GitHub Organization "NucleUS-EGS" divided in repositories, one per service. The link can be found here: https://github.com/orgs/NucleUS-EGS/repositories