

**Relatório do Trabalho Prático**

**Desenvolvimento de um Sistema Distribuído**

**1ª Fase - Planeamento**

Arquitetura e Integração de Sistemas

**Docente:** Bruno Lima

**Trabalho realizado por:**

14829 José Gomes

15708 Pedro Carvalho

15709 José Carreira

Ano letivo 2023/2024

Mestrado em Engenharia Informática

Barcelos, novembro 2023

Índice

[Índice 2](#_Toc153628674)

[1. Introdução 4](#_Toc153628675)

[2. Tema escolhido 5](#_Toc153628676)

[2.1. Problema Identificado 5](#_Toc153628677)

[2.2. Análise do problema 5](#_Toc153628678)

[2.3. Abordagem/Estratégia 6](#_Toc153628679)

[3. Requisitos 7](#_Toc153628680)

[3.1. Requisitos Funcionais 7](#_Toc153628681)

[3.2. Requisitos não funcionais 8](#_Toc153628682)

[4. Arquitetura do Sistema 9](#_Toc153628683)

[5. Tecnologias a utilizar 11](#_Toc153628684)

[5.1. .NET Core 7 11](#_Toc153628685)

[5.2. JWT (JSON Web Tokens) 11](#_Toc153628686)

[5.3. MongoDB Atlas 12](#_Toc153628687)

[5.4. ELK Stack 12](#_Toc153628688)

[5.5. Docker 13](#_Toc153628689)

[5.6. Postman 14](#_Toc153628690)

[5.7. Especificação Open API 15](#_Toc153628691)

[5.8. Polly 16](#_Toc153628692)

[5.9. xUnit – Teste unitários 16](#_Toc153628693)

[5.10. BCrypt 16](#_Toc153628694)

[5.11. Ocelote Gateway 17](#_Toc153628695)

[6. Serviços externos utilizados 17](#_Toc153628696)

[6.1. Open Meteo (Weather API) 17](#_Toc153628697)

[6.2. API-Football 18](#_Toc153628698)

[6.3. Google Maps 18](#_Toc153628699)

[7. Armazenamento de dados 19](#_Toc153628700)

[7.1. MongoDB Atlas 19](#_Toc153628701)

[7.2. ELK Stack 20](#_Toc153628702)

[8. Microservices 21](#_Toc153628703)

[8.1. MS Autorização 21](#_Toc153628704)

[8.2. MS Jogos 22](#_Toc153628705)

[8.3. MS Direções 22](#_Toc153628706)

[8.4. MS Lazer 22](#_Toc153628707)

[8.5. MS Estacionamento 22](#_Toc153628708)

[8.6. MS Gateway 22](#_Toc153628709)

[9. Pontos a salientar 23](#_Toc153628710)

[10. Conclusão 24](#_Toc153628711)

# Introdução

Este relatório é apresentado no âmbito do trabalho prático da unidade curricular de Arquitetura e Integração de Sistemas, do Mestrado de Engenharia Informática. Esta UC tem como objetivo dar a conhecer os fundamentos de arquiteturas de sistemas, baseadas em *microservices.*

Este trabalho foi divido em duas fases de desenvolvimento distintas. A primeira fase consistiu na identificação do problema a desenvolver e no planeamento da solução. Pretende-se efetuar o desenvolvimento de um sistema distribuído que permita gerir e agilizar um determinado processo de negócio, utilizando uma arquitetura de sistema baseada em *microservices,* permitindo independência entre eles*,* tornando o sistema escalável, com flexibilidade na utilização de tecnologias diferentes e agilidade no processo de desenvolvimento.

A segunda fase partiu para o desenvolvimento deste sistema.

# Tema escolhido

A aplicação a ser desenvolvida tem como objetivo agilizar a ida a eventos desportivos. Disponibilizando informação útil que torne uma ida a este tipo de eventos mais confortável para o utilizador da aplicação.

## Problema Identificado

Os adeptos de futebol normalmente gostam de assistir a jogos, sejam eles da equipa que apoiam ou outro qualquer jogo.

O problema identificado foi a complexidade na preparação de uma ida a um evento desportivo. A indisponibilidade demonstrada, desconhecimento de possibilidades e falta de tempo que as pessoas têm atualmente torna difícil responder ás seguintes questões:

* Onde estacionar;
* Onde almoçar/jantar;
* Como chegar lá;
* Condições meteorológicas.

Por estes motivos, uma experiência que deveria ser agradável, com momentos de alegria, entusiasmo e diversão, por vezes acaba por ser motivo de stress e desagradável.

## Análise do problema

Pretende-se desenvolver uma aplicação que ajude a resolver este problema, fornecendo informação ao utilizador sobre diferentes jogos, locais de estacionamento, estado da meteorologia, restaurantes/bares, direções até ao estádio, de forma rápida e assertiva.

Um utilizador pode verificar eventos desportivos do seu clube ou outros eventos que existam perto dele, dentro de um determinado período temporal e, para cada evento, obter um plano com informação relevante (direções, meteorologia, tempo de viagem, restaurantes/bares perto do recinto desportivo e locais de estacionamento).

## Abordagem/Estratégia

A estratégia assumida passa pela criação de um *microservice* para cada funcionalidade que se pretende disponibilizar na aplicação:

* Jogos;
* Direções;
* Lazer;
* Tempo;
* Estacionamento.

Cada *microservice* tem a sua própria arquitetura e é independente dos restantes, podendo comunicar com *API’s* externas e/ou base de dados. Também será utilizado o *microservice* de autenticação (que funcionará com alojamento na *Cloud* e hospedado num ambiente *Kubernetes*) que será desenvolvido para o trabalho da unidade curricular Sistemas de Computação na *Cloud*, esse *microservice* será desenvolvido tendo em consideração a articulação com o restante sistema, garantindo a segurança e gestão de acessos.

Salienta-se que foi também desenvolvido uma User interface (*microservice*) para demonstrar as funcionalidades desenvolvidas.

# Requisitos

## Requisitos Funcionais

Efetuou-se o levantamento de requisitos funcionais para o *microservice* de autenticação:

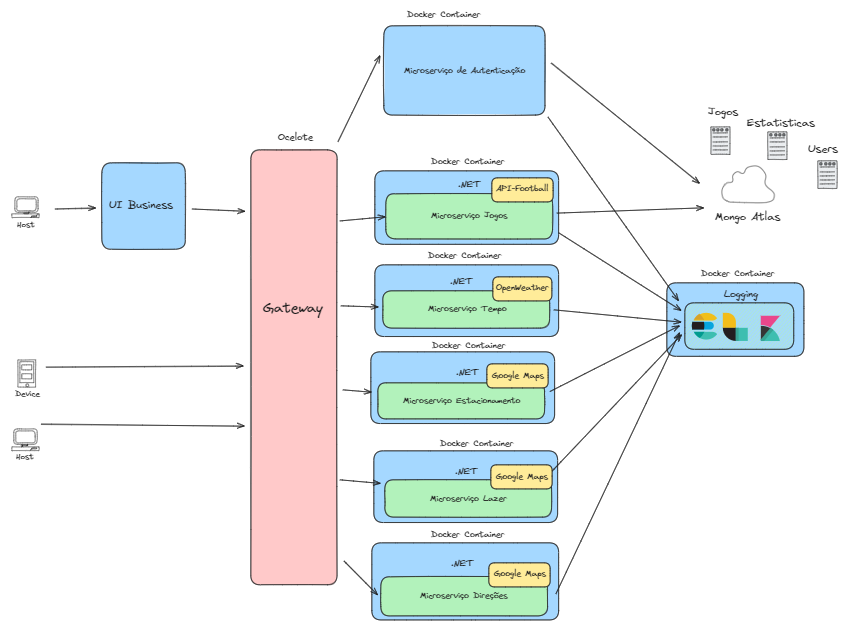
|  |  |  |  |
| --- | --- | --- | --- |
| **Requisitos Funcionais** | | | |
| **ID** | **Requisito** | **Descrição** | **Prioridade** |
| RF01.1 | Gestão de Utilizadores | O sistema tem de permitir registar novos utilizadores. Campos obrigatórios:   * Email; * Password; | MUST |
| RF01.2 | O sistema deverá considerar os seguintes Tipos de Perfil:   * Utilizador; * Administrador; | MUST |
| RF01.3 | O sistema poderá permitir inativar utilizadores. | SHOULD |
| RF01.4 | O sistema poderá permitir atualizar a informação do próprio utilizador:   * Password   Alterações possíveis enquanto o utilizador tem a sessão ativa. | MUST |
| RF01.5 | O login deverá ser possível através de:   * Registo de utilizador, apresentando os campos:   + Email;   + Password; | MUST |
| RF01.6 | O sistema poderá disponibilizar a opção de recuperação de password por Email. | COULD |
| RF01.7 | O sistema poderá controlar o nº de tentativas de passwords erradas, 3 erros no mesmo dia e fica com o login bloqueado durante 1h. | COULD |
| RF01.8 | O sistema deverá ter "pré-registado" um Administrador, cuja login seja gerida pelo proprietário. | MUST |
| RF01.9 | O sistema poderá permitir associar/retirar funcionalidades do perfil. | COULD |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Requisitos Não Funcionais** | | | | |
| **ID** | **Requisito** | **Descrição** | **Prioridade** |
| RNF01 | Desempenho | O sistema deverá apresentar bom desempenho: respostas rápidas após clique e a carregar conteúdos. | MUST |
| RNF04 | Interoperabilidade | O sistema deverá comunicar com *API’s, MongoDB* *Atlas Cloud* e sistema externo de *logs*. | MUST |
| RNF05 | Confiabilidade | O sistema deverá correr sem problemas 99% do tempo, o serviço online é fundamental. | SHOULD |
| RNF06 | Idiomas | O sistema deverá suportar o idioma: Português. | MUST |
| RNF07 | Segurança e Privacidade | O sistema terá de garantir a segurança e privacidade dos dados dos utilizadores, pesquisas e ações efetuadas. | MUST |
| RNF08 | Compatibilidade | O sistema deverá ser capaz de correr nos browsers Chrome, IE e Mozilla, e dispositivos móveis. | SHOULD |
| RFN09 |  | O sistema deverá suportar a comunicação com *cloud* para consumir o *microservice* de autenticação. | SHOULD |
| RFN10 | Integração | O sistema deverá suportar a integração com as *API’s*:   * *Google Maps* para direções, lazer e estacionamento; * *API-FOOTBALL* para informações de jogos, equipas e estatísticas; * *Weather-API* para informação relativa a meteorologia. | MUST |

## Requisitos não funcionais

# Arquitetura do Sistema

A figura acima representa a arquitetura definida para suportar os *microservices* desenvolvidos.



A arquitetura tem como ponto de acesso uma *gateway,* quetem como objetivo:

* Servir de ponto de acesso único aos diferentes *microservices* desenvolvidos;
* Encaminhar pedidos para os *microservices* de forma correta.

Apenas o *microservice* de jogos utiliza uma base de dados (MongoDB Atlas Cloud) de forma a poder guardar informação relativa a jogos já decorridos assim como informação que não é atualizada constantemente (ex: moradas de estádios).

Todos os *microservices* irão correr dentro de containers, permitindo que a sua execução ocorra de forma isolada.

Os restantes *microservices* utilizam API’s para obter informação atualizada de acordo com o pedido do utilizador.

São utilizadas 3 API’s diferentes para disponibilizar informação aos serviços

* *API Google Maps* – Para fornecer informação relativa a direções até determinado estádio, informação sobre restaurantes/bares/cafés nas proximidades do estádio e estacionamento para o utilizador.
* *API Football* – Para fornecer detalhes das equipas, estatísticas e informação sobre o jogo a ver.
* *Weather API* – Fornece dados meteorológicos informativos ao utilizador relativamente ao momento atual e uma janela temporal até *x* horas depois.

# Tecnologias a utilizar

Este seção do relatório tem como objetivo mencionar as tecnologias que foram utilizadas durante o desenvolvimento do *microservice* de autenticação, assim como referir todos os detalhes e funções adicionais que podem ser de interesse.

## .NET Core 7

Foi optado por desenvolver esta solução utilizando *.NET Core 7*.

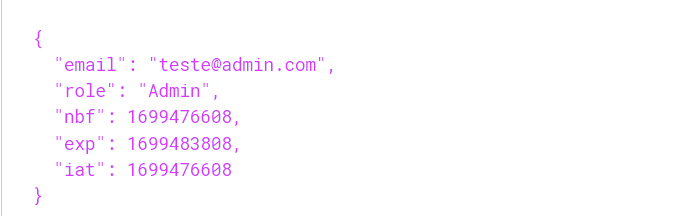
Esta versão do *.NET Core* trouxe bastantes melhorias em termos de performance, facilidade de uso, e ao mesmo tempo introduziu novas ferramentas que foram utilizadas ao desenvolver este projeto, como por exemplo as novas opções de documentação para *Open APIs*.

## JWT (JSON Web Tokens)

É um protocolo de comunicação compacto, normalmente utilizado para autenticação e transferência de informação *client/server*. O token contém informação sobre o utilizador, que é depois utilizada para validar os pedidos. A estrutura do *token* é constituída por *header, body* e *key*, cada um com a sua própria função e a junção de todos os elementos garante a validação de segurança necessária.

Quando autenticado com sucesso é gerado um *token* e de seguida enviado de volta para o cliente, sendo incluído em todos os pedidos. Desta forma é garantida a segurança na informação transmitida pelos vários serviços do sistema.

Neste projeto foram utilizados *JWT* com *claims* específicas para validar o acesso aos *endpoints*. Nas imagens seguintes é possível ver uma estrutura básica de *JWT* e a validação que fazemos por *Roles*.



## MongoDB Atlas

Para a gestão dos utilizadores da aplicação foi utilizado o *MongoDB Atlas*.

A versatilidade de bases de dados não relacionais permitiu ao grupo iniciar o desenvolvimento do serviço de autenticação sem definir uma estrutura de dados concreta para o utilizador.

## ELK Stack

É um conjunto de ferramentas que permite a análise de dados em tempo real e utilizada para gestão de *logs*. É constituída por três ferramentas principais:

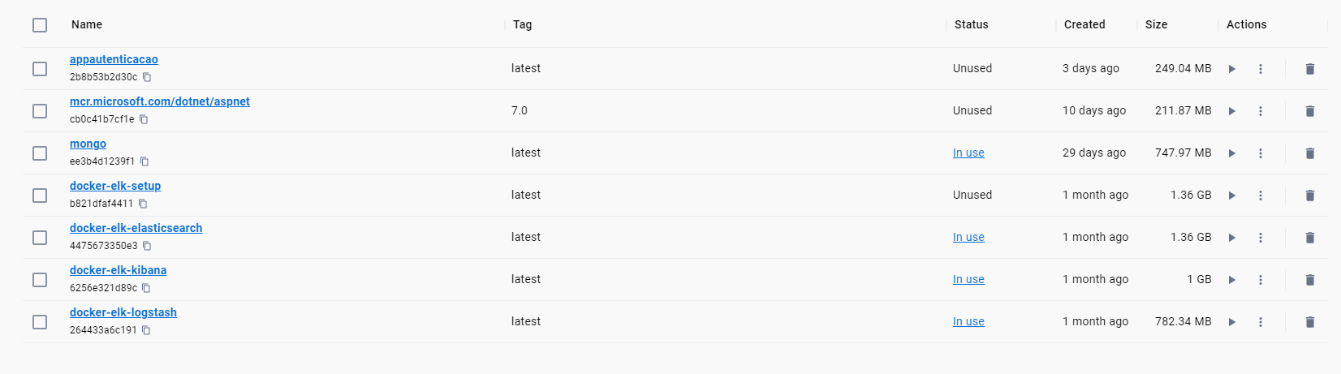
* *Elastic Search*, permite armazenar, procurar e analisar grandes volumes de dados de forma rápida e em tempo real;
* *Logstash*, é uma ferramenta que permite obter, processar, enviar para armazenamento ou analisar, dados de várias fontes. Funciona como um intermediário que valida e garante que os dados estão formatados corretamente antes de serem enviados para armazenamento ou análise nas outras ferramentas *ELK*;
* *Kibana*, é um *UI* que permite visualizar e analisar os dados armazenados no *Elasticsearch*. Tem disponíveis vários *dashboards* e gráficos que facilitam a compreensão dos dados, análise de padrões e monitorização efetiva.

Durante a execução dos nossos serviços vamos registando eventos para o *stack* com diferentes níveis de interesse:

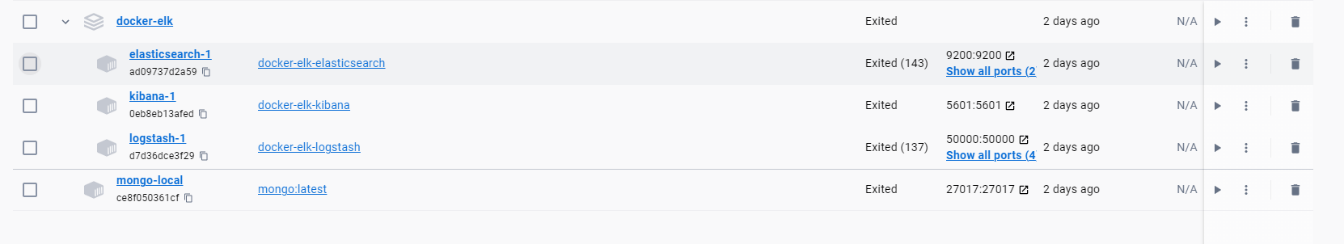
* Informação;
* Aviso;
* Perigo;
* Erro.

## Docker

É uma plataforma de software que permite simplificar o processo de desenvolvimento, testes, *build*, *deploy* e execução de aplicações em containers.

O *Docker* permite a criação de imagens (pacotes com tudo o que a aplicação necessita para correr). Essas imagens são utilizadas para criar *containers*, que funcionam como instâncias de execução dessas imagens.

Os *containers* permitem criar ambientes em que a compatibilidade deixa de ser problema, assegurando todas as dependências (código, bibliotecas, ferramentas e/ou outras dependências.) para que a aplicação corra de forma consistente em qualquer ambiente, seja ele de testes ou produtivo. Na imagem abaixo podemos ver alguns *containers* que utilizamos no projeto.



Para gestão de *containers* utilizamos o *Docker Compose,* esta ferramenta permite definir e/ou executar vários *containers* como um único serviço. Através de um ficheiro de configuração - *docker-compose.yml* – é possível especificar dependências, sequências de arranque de serviços e outras configurações. Tornando uma gestão que poderia ser extremamente complexa e com uma margem de erro enorme, numa tarefa relativamente simples e coordenada.

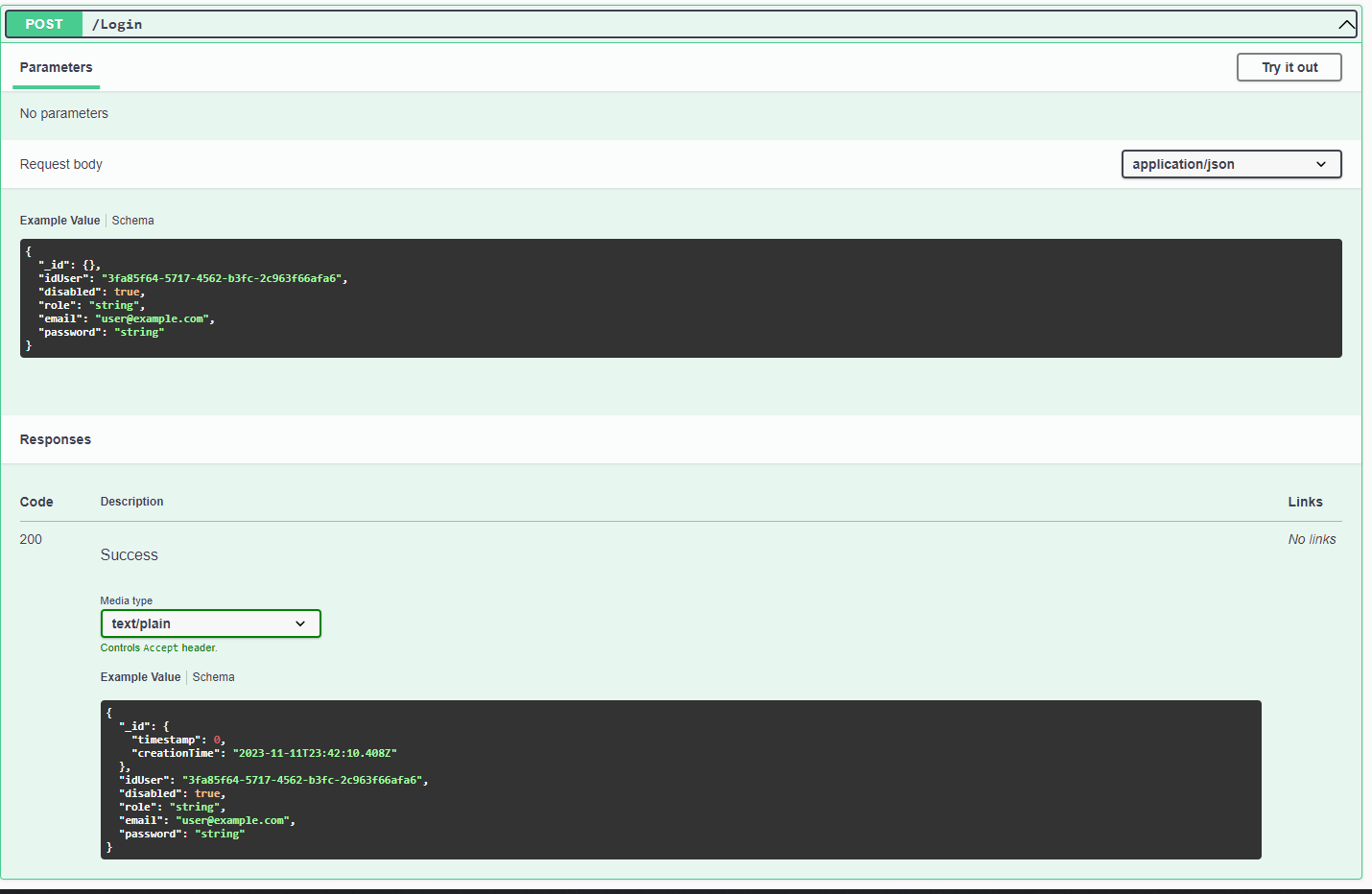


## Postman

O *Postman* é uma ferramenta utilizada para testar e automatizar pedidos a *APIs*. Durante o desenvolvimento foi utilizado extensivamente para verificar o bom funcionamento de todos os *endpoints* que foram criados, assim como verificar se as repostas devolvidas estavam corretas.

## Especificação Open API

Uma *Open API* define um conjunto de padrões e regras para a interação com o *software*, incluindo a estrutura de solicitações e respostas, métodos de autenticação e autorização, tipos de dados suportados e endpoints disponíveis.

Na definição da estrutura dos serviços teve-se atenção para documentar cada *endpoint* de modo a conseguir obter um documento final que respeitasse os padrões colocados. Utilizou-se o *Swagger* para ajudar a representar de forma visual os endpoints disponíveis para cada serviço. A figura abaixo mostra uma das funções disponíveis dos serviços através do *swagger* e a documentação gerada que indica o tipo de resposta esperada.

## Polly

A biblioteca *Polly* disponibiliza um conjunto de ferramentas para ajudar a implementar estratégias de resiliência e tolerância de falhas. Foi utilizada com o intuito de ajudar a comunicação do serviço com a base de dados em *cloud*, permitindo definir um numero de tentativas adicionais a fazer assim como o intervalo entre elas e em que condições devem ser feitas. Permite também definir uma estratégia de *fallback* caso não seja possível concluir a comunicação, sendo possível devolver um valor padrão ou fornecer dados *cache*.

Top of Form

## xUnit – Teste unitários

Foram utilizados para validar individualmente partes do *microservice* de autenticação. O objetivo é isolar e testar individualmente cada componente, de forma independente para garantir que cada parte do sistema funciona de acordo com o esperado.

## BCrypt

É uma biblioteca que permite a encriptação de palavras-passe, utilizando um algoritmo *hash*. Com o objetivo de encapsular a informação, tornando os dados indecifráveis.

Uma característica muito importante no *BCrypt* é o *Salt*, é um mecanismo que acrescenta sequencias de caracteres aleatórias, criando resultados altamente complexos e aumentando a segurança e dificultando ataques de força bruta.

## Ocelote Gateway

Facilita a criação de *APIs* consistentes e seguras, permitindo uma gestão mais eficiente de vários pedidos e respostas entre os vários *microservices*. A inclusão de uma camada adicional entre o serviço e o cliente permite a implementação e validações adicionais, controlo de acessos e ao mesmo tempo o cliente nunca comunica diretamente com os *endpoints*, facilitando assim a administração e segurança na comunicação entre diferentes micro serviços. Na figura abaixo podemos ver um exemplo de configuração da *gateway*, em que é definido o pedido que está a chegar (*upstream*) e para onde é encaminhado (*downsteam*).

# Serviços externos utilizados

## Open Meteo (Weather API)

É uma API pública que permite aceder a informação meteorológica para um determinado período temporal, baseado em coordenadas geográficas (latitude, longitude). O detalhe da informação disponibilizada pela API permite receber informação hora-a-hora de:

* Temperatura (Cº)
* Código de Tempo (*WMO*)

*World Meteorological Organization* (*WMO*) são códigos compostos de um conjunto de valores representados em tabelas que definem eventos meteorológicos. Para este trabalho considerou-se os seguintes eventos:

* Céu limpo
* Pouco nublado
* Nublado
* Muito nublado
* Chuva

## API-Football

É uma API *Freemium* que disponibiliza informação relativa a jogos de futebol. A partir desta API, de forma a dar resposta à nossa necessidade de negócio, obteve-se informação relacionada com a Primeira Liga e Taça de Portugal. Nomeadamente:

* Jogos;
* Ligas;
* Estádios;
* Equipas;
* Jogadores;
* Principais estatísticas (melhores marcadores, assistências, etc.).

Esta API possui uma limitação no numero de pedidos para uso gratuito. Tendo em conta esta limitação optou-se por uma estratégia para minimizar o numero de chamadas à API que passou por importar os dados para a base de dados em Cloud. Desta forma o consumo dessa informação pelo sistema não é afetado por esta limitação na *API*.

De forma a garantir que os dados são atualizados com nova informação, foram criados endpoints (apenas acessíveis por *admin*) que permitem consumir a API de jogos e atualizar a informação pretendida. No capitulo 8.2 vamos entrar em mais detalhe sobre este processo.

## Google Maps

A API do Google Maps é uma API gratuita (disponibiliza um determinado valor de crédito para que um utilizador explore as suas funcionalidades). Possui vastas opções sobre localização, direções e sítios de interesse.

Utilizou-se esta *API* para dar resposta é necessidade de negócio complementando detalhes importantes para satisfazer as necessidades de um utilizador. A *API* tem inúmeros serviços disponíveis, dos quais utilizou-se

* *Places –* para obter informação (*price level*, *rating*, *availabilty, etc.*) sobre determinados locais;
* *Directions –* para obter informação sobre direções entre dois pontos;
* *Geolocation –* para obter as coordenadas de um local com base numa morada (utilizamos esta *API* para converter as moradas dos estádios em coordenadas de latitude e longitude de modo as serem utilizadas por outros serviços).

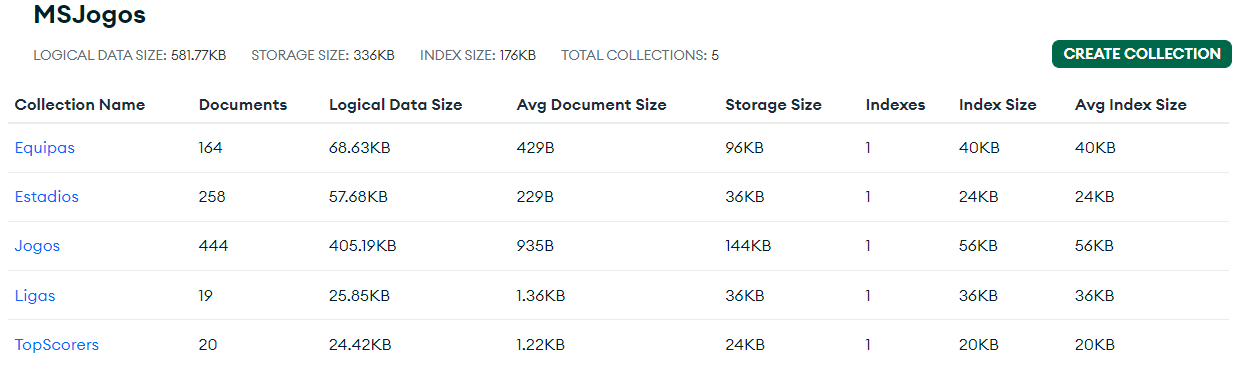
# Armazenamento de dados

## MongoDB Atlas

Optou-se pela base de dados não-relacional *MongoDB Atlas* hospedada na Cloud para armazenar determinados dados da aplicação. Foram criadas as seguintes bases de dados:

* Utilizadores – Guarda informação para o processo de autenticação dos utilizadores;
* Jogos – Guarda toda a informação relacionada com os jogos distribuída por várias *collections* (ligas, equipas, jogadores, estádios e estatísticas relevantes);
* Estatísticas – Esta base de dados possui *collections* com informação relativa à navegação de cada utilizador.

Na imagem seguinte é possível visualizar as collections disponíveis na base de dados Jogos, assim como vários detalhes importantes:



## ELK Stack

Utilizou-se *ELK Stack* para armazenamento de *logs* do tipo, informação, erros e avisos. A análise destes *logs* nos diversos *dashboards* disponíveis permitem-nos ter uma ideia do “estado de saúde” da nossa aplicação.

Na imagem abaixo representada temos um exemplo de um ecrã de *logs* no *Kibana* com diferentes níveis de criticidade.

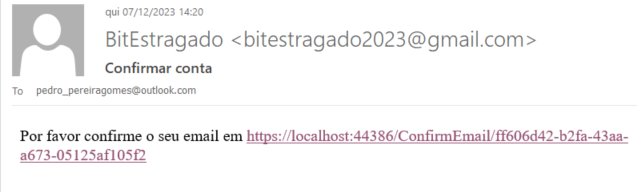


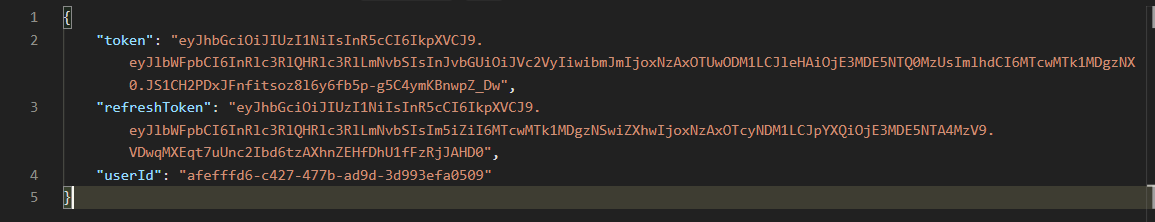
# Microservices

## MS Autorização

Serviço responsável por permitir gerir autenticação e perfis de acesso dos utilizadores na aplicação. Detalhes relevantes do microservice:

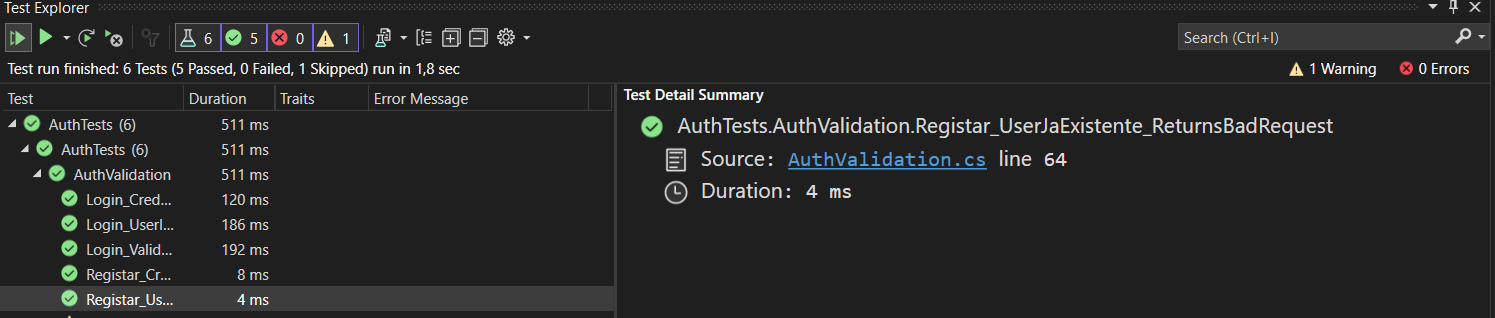
* Password encriptada (*Bcrypt*) para armazenamento seguro (*MongoDB Atlas*);
* Novo utilizador é criado como *disabled* e requer confirmação de conta através de um *link* recebido por email;



* Criação de *token* deacesso e *refresh* token utilizando JWT para autenticação segura. Os *tokens* possuem *claims* diferentes, sendo que o *token* de acesso possui as *claims*: *role* e *email*, e o *refresh* *token* apenas de *role;*

#### Testes unitários

Para validar a logica dos endpoints assim como detetar possíveis erros e melhorar a estrutura geral do código foi desenvolvido um conjunto de testes unitários utilizando a biblioteca xUnit em conjunto com a biblioteca Moq.



## MS Jogos

Este *microservice* é responsável por disponibilizar informação sobre jogos de futebol e toda a informação relevante relacionada com os eventos em causa.

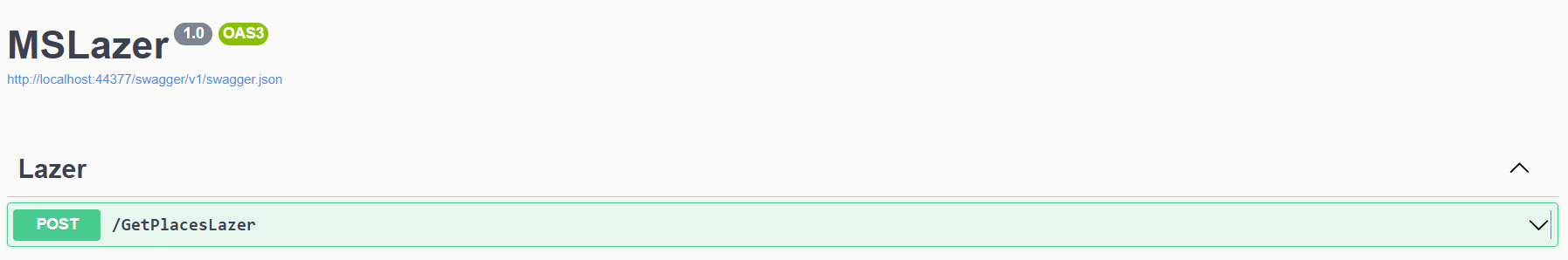
Possui dois conjuntos de endpoints (Jogos e AdminJogos) geridos pelos respetivos perfis de acesso da aplicação (Utilizador e Admin):

* AdminJogos, tem como objetivo permitir a um utilizador com a role de Admin atualizar os dados armazenados em *MongoDB* Atlas através de chamadas à API-Football garantindo que os dados permanecem atualizados. Também é possível aceder a informações estatísticas sobre a navegação de cada utilizador.
* Jogos, permitem a um utilizador (com token válido) obter informação sobre jogos futuros e o seu detalhe, assim como dados estatísticos de jogadores.

## MS Direções

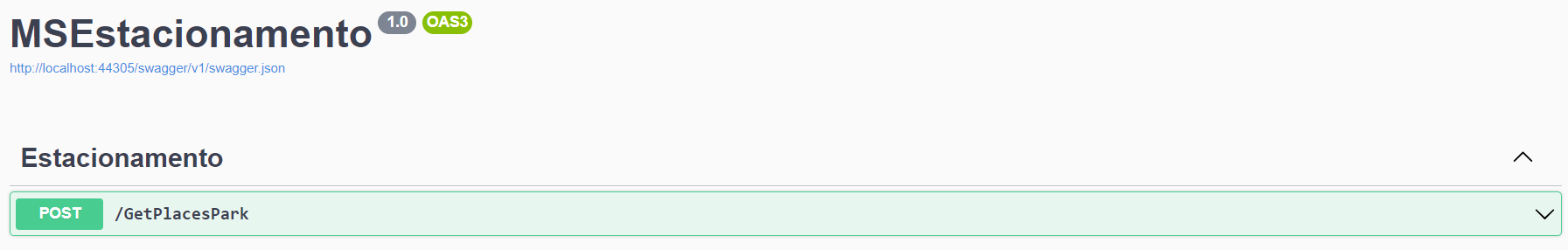
Este *microservice* é responsável por disponibilizar ao utilizador (com token válido) as direções entre dois pontos geográficos (indicando a latitude e longitude de cada ponto).

## MS Lazer



Este *microservice* é responsável por disponibilizar ao utilizador (com token válido) os locais de lazer (bares, restaurantes, cafés), tendo como referência as coordenadas geográficas do estádio correspondente ao jogo selecionado.

## MS Estacionamento



Este *microservice* é responsável por disponibilizar ao utilizador (com token válido) os locais de estacionamento nas proximidades do estádio, tendo como referência as coordenadas geográficas do estádio correspondente ao jogo selecionado.

## MS Gateway

Para implementar a *Gateway* foi utilizado o Ocelote. A implementação consistiu na configuração de um documento *JSON* onde é indicado o comportamento da *gateway* dependendo do pedido recebido, assim como para onde é encaminhado, e que tipos de pedidos são aceites.



**Routes** – É uma coleção com todas as regras que a *gateway* suporta. Cada elemento das *routes* corresponde a uma configuração de um pedido, assim como o que fazer com o mesmo.

**UpstreamPathTemplate** – Refere-se ao template do pedido recebido (upstream – rio acima). O exemplo acima representado indica que esta regra apenas tem efeito para pedidos que tenham o caminho /gateway/Register

**UpstreamHttpMethod** – Indica os verbos http suportados pelo gateway para determinado Path. Neste caso só são aceites pedidos POST.

**DownstreamPathTemplate** – Representa para onde vai ser encaminhado o pedido (downstream – rio abaixo). Quando mencionado em downstream refere-se á API que vai receber o pedido. O exemplo vai encaminhar o pedido para o endpoint /Register.

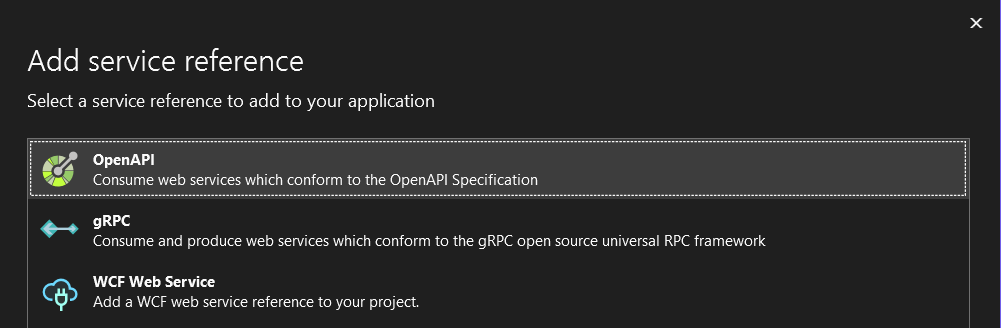
**DownstreamHostsAndPorts** – Contém a informação sobre a API que vai responder ao pedido.

A configuração acima tem a seguinte representação:

# Business Class

#### 9.1.1 Service Reference - OpenAPI

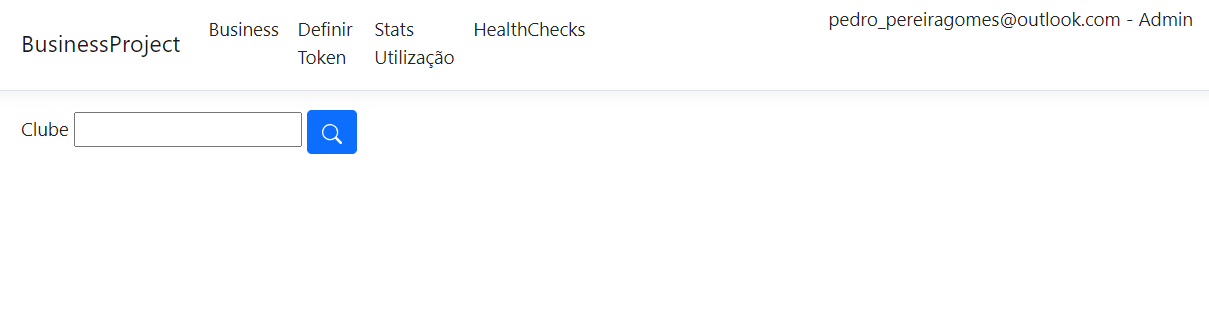
Uma correta documentação dos endpoints desenvolvidos em conjunto com *swagger* permitiu utilizar uma funcionalidade disponível no *Visual Studio 2022* que possibilitou a importação direta de uma especificação *OpenAPI* e automaticamente criou um cliente capaz de consumir os métodos desenvolvidos nos microserviços.



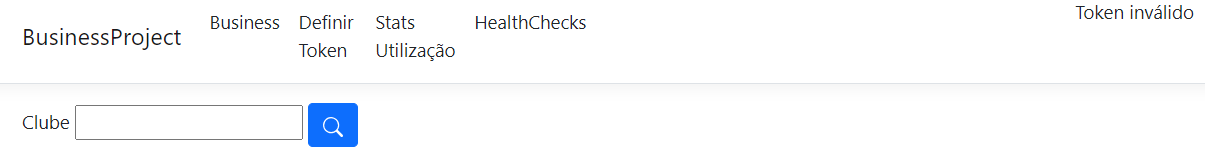
#### 9.1.2 Análise a Tokens

Para utilizar as funcionalidades da aplicação é necessário apresentar um token de acesso válido. Na business class é permitido ao utilizador inserir um token para ser utilizado nos diferentes microservices, sendo de seguida validados os valores contidos no token.

No exemplo abaixo representado é possível verificar um token válido com a role de Admin.

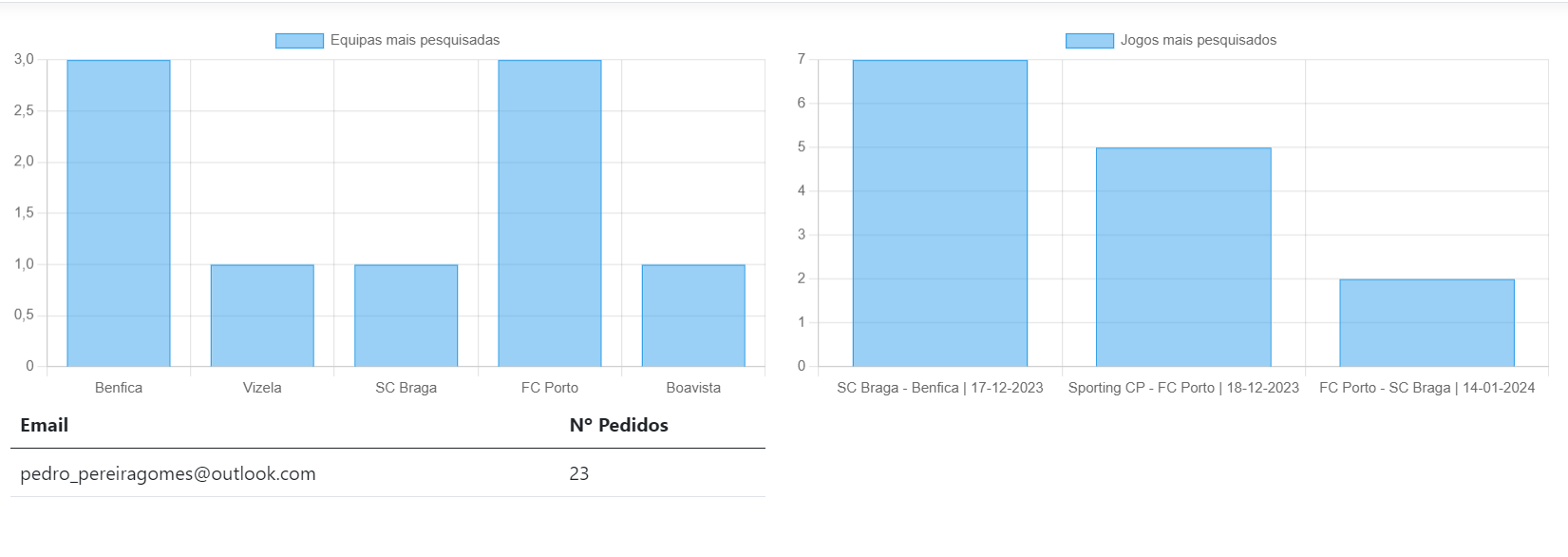


No exemplo abaixo representado é possível verificar um token inválido.



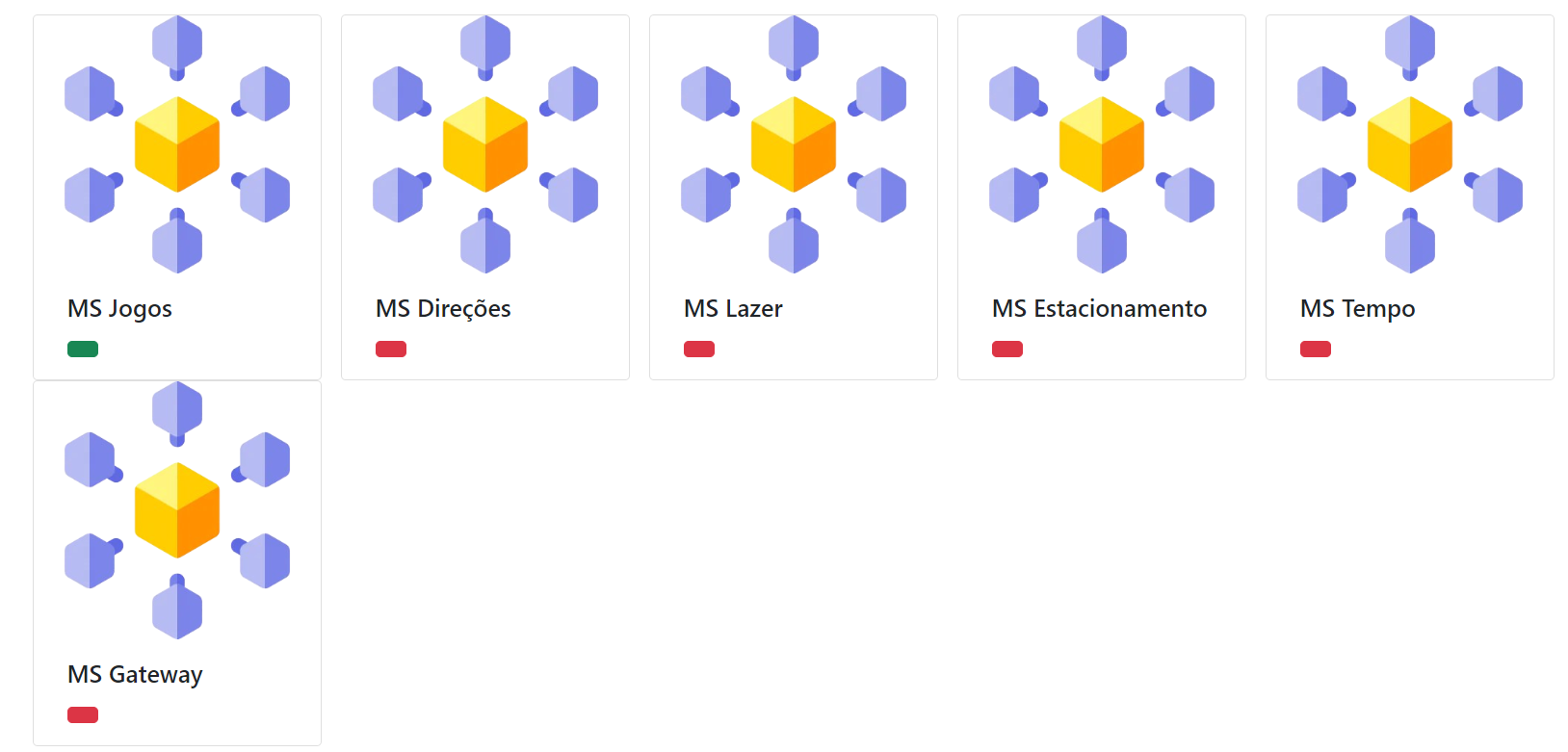
#### 9.1.3 Análise estatística á navegação de um utilizador

Durante a utilização da aplicação as equipas e jogos que o utilizador consulta são registados numa *collections* em MongoDB Atlas para que posteriormente possa ser feita a sua análise. Utilizando os dados armazenados relativos á navegação de um utilizador foi possível criar um conjunto de gráficos que representam:

* Equipas mais procuradas
* Jogos mais pesquisados
* Utilizadores com mais pesquisas

#### 9.1.4 Verificar o estado dos serviços

Para garantir que a consulta dos estados dos *microservice* era simples e visualmente percetível foi desenvolvida um sistema que verifica o estado de cada *microservice* disponível.



# Pontos a salientar

* Ao utilizar a API-Football para receber informação de equipas verificou-se a ausência de informação relativa ás coordenadas de latitude e longitude das mesmas (algo essencial ao funcionamento dos restantes *microservices)*. Face a este problema optou-se por utilizar a API de *Geolocalização* do Google Maps para popular as coordenadas de cada estádio.
* Todos os *microservices* validam se o *token* de acesso é valido para continuar o fluxo de execução, contudo para não existir uma dependência entre o *microserviço* de autenticação e os restantes foi replicada a lógica de validação de *token* para os restantes serviços, o que garante que não existe dependência entre *microservices.*
* Os pedidos de informação relativamente a jogos de futebol e equipas são guardados em *collections* para uso estatístico.

# Conclusão

A realização deste trabalho foi importante para conhecer diferentes formas de definir a arquitetura de um sistema, o que nos levou a explorar algumas possibilidades, ferramentas, bibliotecas e conceitos novos.

É um desafio interessante escolher a estrutura, de forma correta e seguindo os princípios fundamentais aprendidos em aula (arquitetura modular e escalável), que melhor se enquadra com o modelo de negócio que se pretende operacionalizar. Além disso, a autonomia na implementação e manutenção de cada serviço ofereceu uma flexibilidade percetível ao grupo, permitindo atualizações e correções específicas sem afetar a totalidade do sistema.

O desenvolvimento deste trabalho também possibilitou a utilização de ferramentas com as quais o grupo não estava familiarizado (exemplo: *Docker, Open APIs, ELK Stack,xUnit,Bcrypt,MongoDB Atlas*). A utilização destas ferramentas foi um desafio que acabou por aumentar o nosso conhecimento, permitiu-nos organizar melhor o trabalho, sermos mais assertivos na resolução de problemas e agilizou o nosso processo de desenvolvimento.