Universidade de Aveiro

KIUA
Relatório Técnico

Alexandre Oliveira 93289, Gabriel Ribeiro 93036, João Gameiro 93097,

Marco Ramos 93388, Miguel Nogueira 93082, Pedro Abreu 93240

Professor Cláudio Teixeira

Projeto em Engenharia Informática

Mestrado Integrado em Engenharia de Computadores e Telemática

Departamento de Electrónica Telecomunicações e Informática

2021

Universidade de Aveiro

KIUA
Relatório de Projeto

Realizado por Alexandre Oliveira (93289) alexandretomas@ua.pt, Gabriel Ribeiro (93036) gabrielribeiro00@ua.pt, João Gameiro (93097) joao.gameiro@ua.pt, Marco Ramos (93388) msramos99@ua.pt, Miguel Nogueira (93082) miguel.nogueira@ua.pt, Pedro Abreu (93240) pedro.abreu@ua.pt

Orientado pelo Professor Cláudio Teixeira, Diretor de Serviços dos Serviços de Tecnologias da Informação e Comunicação da Universidade de Aveiro

Projeto em Engenharia Informática
Mestrado Integrado em Engenharia de Computadores e Telemática
Departamento de Electrónica Telecomunicações e Informática

2021

Resumo

O projeto Key Indicators Universidade de Aveiro (KIUA) tem como principais objetivos a recolha de Key Performance Indicators (KPIs) e geração de dashboards. Deste modo foi desenhado um sistema que permitisse a definição rápida e parametrizada de KPIs a recolher, assim como a geração de dashboards a partir das métricas recolhidas.

Foi construída uma arquitectura base para este sistema de modo a permitir o bom funcionamento da plataforma de gestão de métricas e o armazenamento de dados recolhidos.

Foi também desenvolvida uma página web de apresentação de dados em tempo real, sendo que para esta página teve de ser feita uma análise criteriosa de quais as KPIs que seriam importantes apresentar.

A principal ideia por detrás do desenvolvimento da plataforma de gestão de métricas foi que os utilizadores a usariam sem a necessidade de saber toda a complexidade envolvente (ex.: criação de dashboards intuitiva e simples sem ser necessário estarem preocupados com problemas como datasources ou querys a apresentar), e que pudessem monitorizar os seus serviços para tirar conclusões relativamente à sua performance.

Agradecimentos

Gostaríamos de agradecer ao orientador Cláudio Teixeira pelo apoio prestado no desenvolvimento do projeto, e também a todos os professores da unidade curricular de Projeto em Engenharia Informática do terceiro ano do curso Mestrado Integrado em Engenharia de Computadores e Telemática.

Conteúdo

Lista de Tabelas

Lista de Figuras

Glossário

API Application Programming Interface

CORS Cross-Origin Resource Sharing

DETI Departamento de Electrónica Telecomunicações e Informática

DHCP Dynamic Host Configuration Protocol

IP Internet Protocol

KIUA Key Indicators Universidade de Aveiro

KPIs Key Performance Indicators

SCOM System Center Operations Manager

WSO2 Web Services Oxygenated

HTML HyperText Markup Language

CSS Cascading Style Sheets

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

URL Uniform Resource Locator

UID Unique Identifier

DNS Domain Name System

UA Universidade de Aveiro

SQL Stuctured Query Language

VM Virtual Machine

SSH Secure Shell

Capítulo 1

Introdução

O presente relatório visa descrever o processo de implementação do projeto KIUA, desenvolvido no âmbito da unidade curricular de Projeto em Engenharia Informática do curso Engenharia de Computadores e Telemática.

Os principais objetivos deste projeto passam por estudar e recolher KPIs, desenvolver uma estrutura de suporte que permita o armazenamento destes dados, e a criação de duas aplicações web, uma para apresentação de dados em tempo real, e outra para gestão e definição das métricas a serem recolhidas que permita também a geração de dashboards.

Ao longo do relatório serão apresentados os vários aspectos e detalhes que permitem a melhor compreensão do projeto e de todo o processo de desenvolvimento do mesmo. Inicialmente será apresentado o contexto em que surgiu a ideia, assim como trabalhos e sistemas semelhantes. Posteriormente, no Capítulo 2 será apresentada uma análise e descrição dos requisitos identificados, assim como os casos de utilização e atores do sistema.

Após a análise, no Capítulo 3 serão descritos todos os detalhes da implementação e a utilização e interligação dos vários componentes do projeto.

No Capítulo 4, serão apresentados os resultados obtidos e será feita a sua análise e discussão. Finalmente, no Capítulo 5 é apresentada a conclusão do trabalho desenvolvido e a respectiva bibliografia.

1.1 Contexto

Atualmente a crescente digitalização de serviços implicou um aumento exponencial da quantidade de dados a circular. Deste modo, torna-se particularmente importante que cada organização tenha acesso aos dados em tempo real que estão a ser produzidos no âmbito das suas operações. A sua posterior análise permite tirar conclusões sobre o rumo atual do negócio, assim como a tomada de decisões administrativas para melhorar o mesmo.

Este tipo de dados chamam-se KPIs, pois permitem medir o desempenho de um negócio ou estratégia. Estes indicadores podem ser de alto nível ou de baixo nível, e tendo em conta a quantidade de dados a circular a sua escolha tem de ser criteriosa.

No caso da Universidade de Aveiro (UA) existe uma grande variedade de serviços disponibilizados à comunidade e por isso existe também uma grande diversidade em termos de KPIs a definir.

1.2 Estado da Arte

Com o objectivo de perceber melhor o resultado final a obter com este sistema foi feita uma pesquisa sobre sistemas e projetos similares.

Cluvio

Cluvio [1] é uma solução analítica baseada em nuvem que permite às empresas analisar dados através de painéis. A solução ajuda a executar consultas, filtrar resultados e exibir dados utilizando gráficos e tabelas para tal efeito.

Os principais recursos incluem um editor de Stuctured Query Language (SQL), scripts R personalizáveis, notificações push, patilha de painéis com clientes e muito mais. Os painéis do Cluvio também pode ser partilhados com os colegas através de um link que permite aos utilizadores visualizá-los sem fazer login. Os utilizadores podem dividir e separar dados, especificar intervalos de datas e definir configurações de agregação.

O Cluvio oferece integração com diversos tipos de bases de dados.

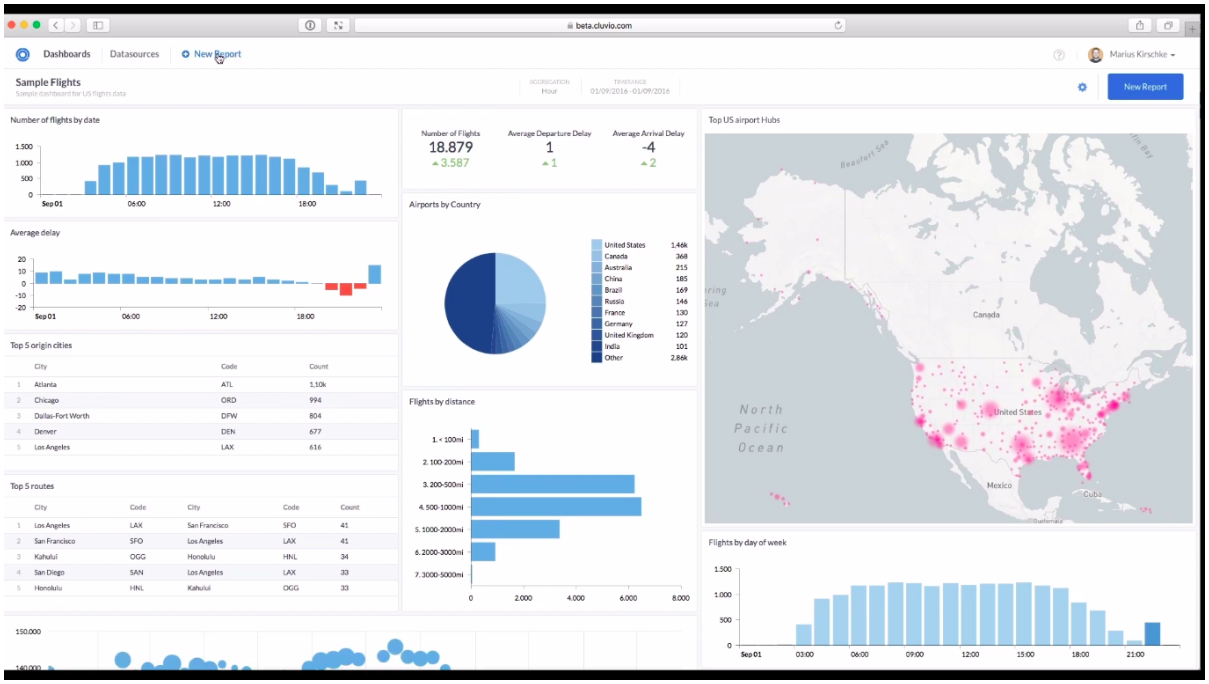

Figure 1.1: *Dashboard exemplo gerada no Cluvio*

Cluvio foi uma forte inspiração no resultado final e auxiliou também certas decisões do projeto (ex.: para seleccionar dados através do Cluvio é necessário um conhecimento avançado de SQL ou de R, condição que se quis evitar no nosso sistema, ou seja, os utilizadores devem ser capazes de escolher os dados a apresentar e criar dashboards sem terem de saber todos as características técnicas de como realmente o fazer).

Indicadores UA

O portal dos Indicadores UA [2] permite a consulta de dados importantes relativos à Universidade de Aveiro, apresentando por isso um conjunto de indicadores que dizem respeito às várias áreas de missão da Universidade de Aveiro (figura 1.2).

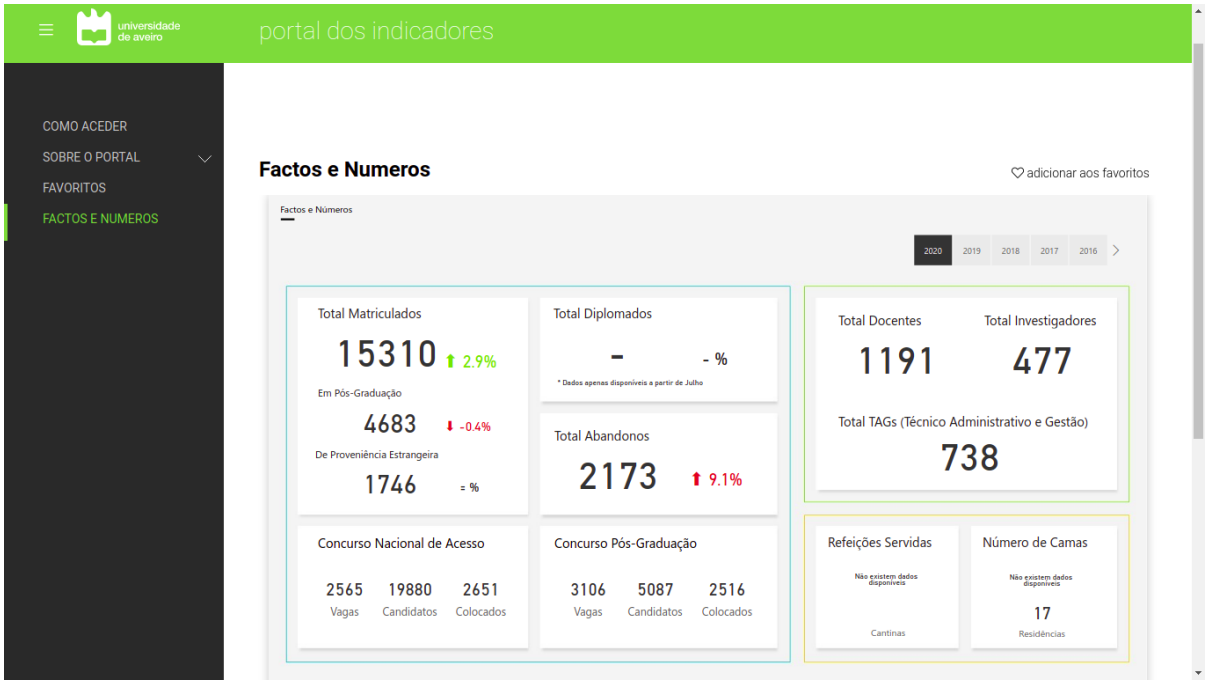

Figure 1.2: *Portal de Indicadores da UA*

Este portal serve principalmente como instrumento de apoio às operações de gestão e decisões administrativas a tomar por parte dos órgãos da universidade.

Neste website é permitida também a manipulação dos dados apresentados. Também é possível aceder diretamente às fontes de dados que alimentam este portal para utilizar em tecnologias como PowerBi ou Excel.

Capítulo 2

Análise do Sistema

De modo a tentar perceber qual a arquitectura do sistema, e quais os requisitos que o sistema deverá possuir ou quem serão os seus utilizadores foi feita uma análise do sistema.

Esta análise foi feita com recurso a constantes reuniões entre os elementos do grupo, onde se discutiram os aspectos importantes do projeto, e reuniões semanais com o orientador para esclarecimento de dúvidas e pontos de situação.

Os resultados da análise de requisitos são apresentadas ao logo das várias secções deste capítulo.

2.1 Actores do Sistema

Na Tabela 2.1, podemos observar todos os atores identificados para o sistema desenvolvido.

É importante salientar o papel do administrador, na medida em que gere métricas default para a plataforma. Estas métricas são importantes pois um utilizador pode querer experimentar a plataforma sem ter ou sem querer fornecer os seus endpoints. Deste modo, existe um conjunto de métricas que estão disponíveis no site, livres de serem usadas por qualquer utilizador para gerar dashboards e experimentar as funcionalidades da plataforma antes de fornecer os seus endpoints. Existem quatro métricas default que estão implementadas na aplicação e que nenhum administrador pode eliminar. Todas as outras são criadas por administradores e ficam disponíveis para todos os utilizadores.

Tabela 2.1: Atores do Sistema

Ator	**Papel do ator**
Utilizador	Pode criar, eliminar e definir visibiliade de dashboards. Pode fornecer endpoints para criar novas métricas.
Administrador	Tem acesso a todas as funcionalidade que o utilizador tem. Gere as métricas default do sistema.
IdpUa	Serviço que permite a autenticação de utilizadores da Universidade de Aveiro.

2.2 Casos de utilização

Na Figura 2.1 podemos observar os casos de utilização definidos para a plataforma de gestão de métricas. Os Casos de utilização foram divididos em dois pacotes, o do Backoffice e o das Dashboards, apesar das dashboards estarem embutidas no website.

Na Tabela 2.2 encontramos uma breve descrição do que cada caso de utilização representa.

Para a página de consumo em tempo real não foi elaborado nenhum diagrama de casos de utilização, pois ela apenas apresenta um caso que é a visualização de dados em tempo real. Não existe qualquer tipo de autenticação, nem funcionalidade extra para além da visualização dos dados apresentados em tempo-real.

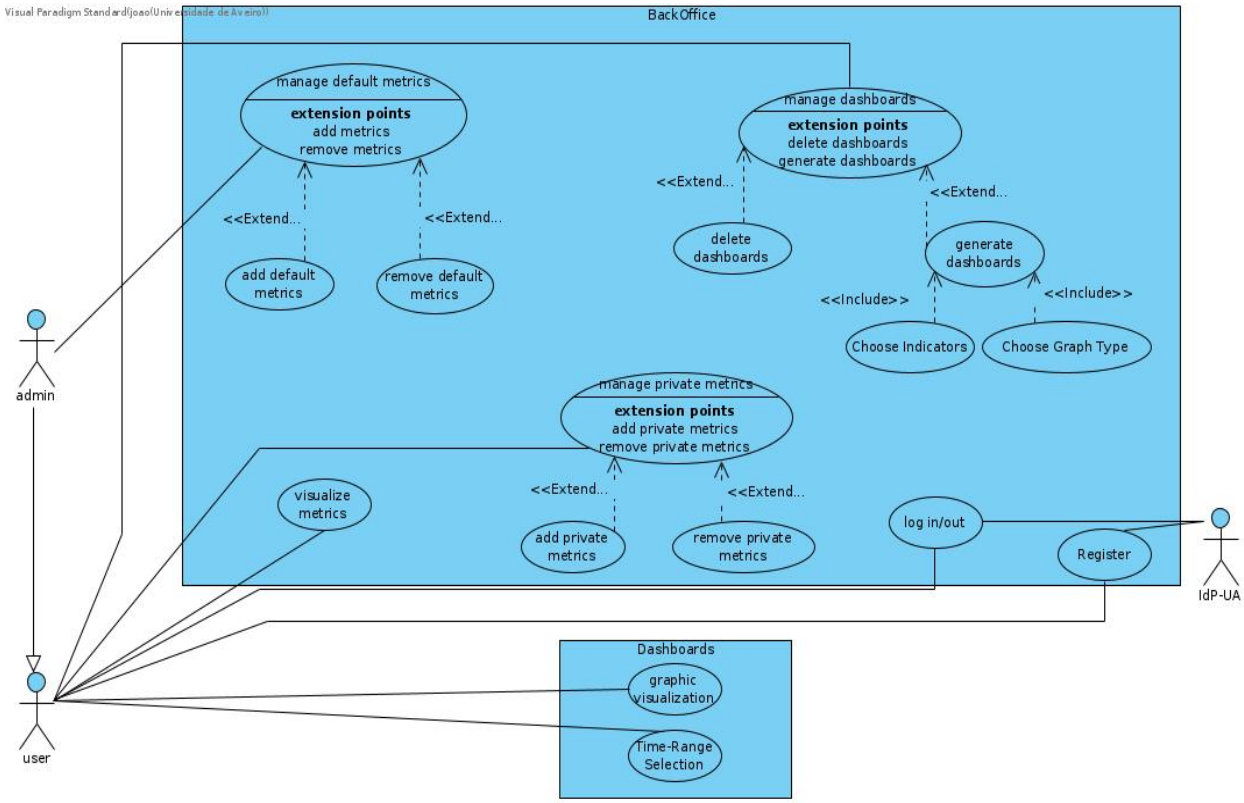

Figure 2.1: *Diagrama dos casos de utilização*

Caso de utilização	**Descrição**
Manage Default Metrics	Gestão das métricas de default (apenas disponível para administradores).
Add Default Metric	Criação de uma métrica de default que implica o preenchimento de um formulário aonde se indica o seu nome, endpoint, query frequency, campos a filtrar e uma pequena descrição da métrica.
Remove Default Metric	Remover uma métrica default da plataforma (apenas é necessário carregar no botão de remover).
Visualize Metrics	Ao aceder ao Backoffice qualquer utilizador irá ter na página My-Metrics uma lista com as suas métricas, assim como uma lista das métricas default na página Metrics.
Manage Private Metrics	Gestão de métricas privadas (disponível para cada utilizador).
Add Private Metric	Criação de uma métrica que implica o preenchimento de um formulário (Métrica fica apenas disponível para o próprio utilizador).
Remove Private Metric	Remover uma métrica privada (Apenas o utilizador que a criou o pode fazer).
Manage Dashboards	Gestão de dashboards (disponível para qualquer tipo de utilizador).
Delete Dashboards	Similarmente às métricas um utilizador pode facilmente apagar uma dashboard, se navegar até à página dashboards e carregar no botão de eliminar na respectiva dashboard.
Generate Dashboards	Geração de dashboards que implica o preenchimento de vários campos entre os quais o nome da dashboard, a escolha dos indicadores e painéis a aparecer.
Choose Indicators	Na página de criação de uma dashboard é apresentado ao utilizador um conjunto de métricas e os vários indicadores a a si associadas. O utilizador apenas precisa de selecionar as checkboxes para escolher quais os indicadores que pretende ver na sua dashboard.
Choose panel type	Para escolher o painel o utilizador especifica o seu nome e pode escolher o formato do gráfico aonde vai ser apresentada a informação especificada.
Graph Visualization	Na página de dashboards existe um link para cada uma das dashboards que se utilizador selecionar, será redireccionado para uma página com a respectiva dashboard aonde pode ver os vários gráficos constituintes da mesma.
Time-Range Selection	O utilizador pode especificar o time-range do gráfico se carregar com o rato em cima do painel em questão e fizer Ctrl-Z para fazer zoom out ou selecionar com o rato a área que quer fazer zoom in.

Tabela 2.2: Use Cases e a sua descrição

2.3 Análise de Requisitos

2.3.1 Levantamento de requisitos

Para apuramento dos requisitos necessários para o desenvolvimento do projecto foram realizadas reuniões semanais com o professor orientador do projecto, Cláudio Teixeira.

De forma a complementar as reuniões realizadas com o orientador foram também realizadas sessões semanais de brainstorm entre todos os elementos do grupo (mais que uma vez quando necessário).

2.3.2 Requisitos Funcionais

Dos requisitos obtidos durante a fase de apuramento foram definidas como funcionalidades essenciais para a aplicação web Backoffice os que estão representados na tabela 2.3. Também foram definidos requisitos para API que permite a gestão de métricas, visíveis na tabela 2.4.

Tabela 2.3: Requisitos funcionais do Backoffice

Requisito	**Descrição**
RFB-1	O sistema deve permitir aos utilizadores a criação, visualização e eliminação de dashboards de forma simples e intuitiva.
RFB-2	O sistema deve permitir a gestão da visibilidade de dashboards (públicas todos os utilizadores devem conseguir visualizar, privadas só o próprio utilizador).
RFB-3	Os utilizadores devem conseguir criar, utilizar e eliminar métricas de forma simples e intuitiva.

Tabela 2.4: Requisitos funcionais da API de Gestão de métricas

Requisito	**Descrição**
RFA-1	A Application Programming Interface (API) deve ter métodos que permitam criar e eliminar métricas
RFA-2	A API deve possuir métodos que permitam o controlo da recolha de dados associada a uma métrica

Para a página web de consumo em tempo real apenas foi definido o requisito funcional que determina que deve ser possível a visualização de dados em tempo real, pois essa é a sua única finalidade.

2.3.3 Requisitos não Funcionais

Ao nível de requisitos não funcionais, estes foram divididos em requisitos de documentação visíveis na tabela 2.5, requisitos de usabilidade aplicados ao backoffice ilustrados na tabela 2.6 e requisitos de segurança e integridade dos dados, apresentados na tabela 2.7.

Tabela 2.5: Requisitos não funcionais da documentação do sistema

Requisito	**Descrição**
RNFD-1	Documentação deve ser clara e sucinta
RNFD-2	Deve ser desenvolvido um manual de utilização para o backoffice
RNFD-3	API deve também possuir uma documentação clara e sucinta

Tabela 2.6: Requisitos não funcionais da usabilidade do backoffice

Requisito	**Descrição**
RNFU-1	O utilizador deve conseguir dominiar todas as funcionalidades do sistema em pouco tempo de utilização
RNFU-2	Criação de dashboards e de métricas devem ser simples e intuitivas
RNFU-3	Sistema deve ser compatível com qualquer tipo de browser

Tabela 2.7: Requisitos de segurança e integridade dos dados

Requisito	**Descrição**
RNFS-1	Dados recolhidas das fontes privadas de dados definidas nas métricas devem apenas estar visiveis e ser acedidos pelos próprios utilizadores que os definiram.
RNFS-2	Dashboards apenas podem ser alteradas pelos próprios utilizadores que as definiram.
RNFS-3	Dashboards privadas apenas podem estar vísiveis para o utilizador que a definiu.
RNFS-4	Após a eliminação de uma métrica todos os dados recolhidos associados à mesma devem ser eliminados.
RNFS-5	Criação de dashboards de um utilizador não deve afetar a de outro (dois utilizadores podem ter uma dashboard com o mesmo nome, mas um utilizador não pode ter duas dashboards com o mesmo nome).

2.4 Arquitetura do Sistema

2.4.1 Arquitetura Geral do Sistema

Nesta secção é apresentada a arquitetura geral do sistema (Figura 2.2). Serão apenas apresentados os módulos base que compõem a arquitetura do sistema, não especificando ainda a tecnologia escolhida para os mesmos nem entrando em muitos detalhes sobre a sua função.

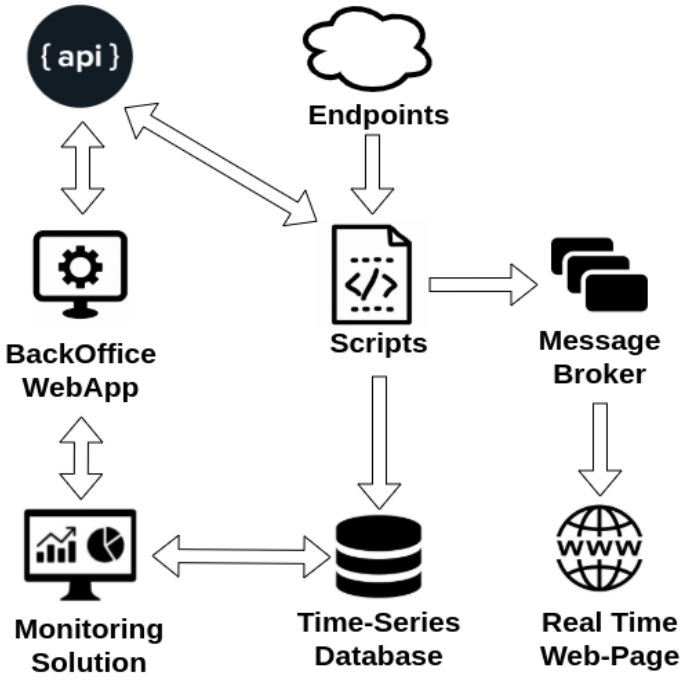

Figure 2.2: *Arquitetura Geral do Sistema*

Nesta figura estão presentes vários elementos, entre os quais:

- **Endpoints** que fornecem os dados a ser utilizados no projeto. Este elemento é geral, ou seja representa a fonte de dados e no projeto essa fonte de dados tanto pode ser os serviços da UA como os fornecidos pelos utilizadores.
- **Scripts** que foram desenvolvidos para recolher dados dos endpoints. Estes scripts não só têm o papel de recolha de dados, como de distribuição dos mesmos por outros componentes do sistema, nomeadamente para a base de dados e para o message broker.
- **Base de Dados (Time-Series Database)** que armazena todos os dados do projeto recolhidos dos mais diversos endpoints.
- **Message Broker** que vai armazenar dados para um objetivo especifico, a apresentação dos mesmos em tempo-real.
- **Aplicação Web de Consumo em Tempo Real (Real Time Web-Page)** para a respectiva apresentação dos dados presentes no message broker.
- **Ferramenta de Monitorização (Monitoring Solution)** para gerar dashboards a partir dos dados que estão presentes na base de dados.
- **Aplicação Web Backoffice (Backoffice WebApp)** que permite a definição parametrizada de métricas e a geração automática de dashboards. A geração automática de dashboards é feita através desta plataforma, e é resultado da interação com o módulo de monitorização. A criação de métricas era feita através de chamadas aos endpoints de uma API que permitisse a gestão de métricas.

- **API para gestão de métricas** Como é necessário a criação parametrizada de métricas, foi decidido que todo esse processo seria da responsabilidade de uma API. Essa API iria possuir endpoints que permitissem a criação e eliminação de métricas, o que implica também a recolha e gestão de dados associados a essas métricas.

2.4.2 Modelo de Tecnologia

No que diz respeito ao modelo de tecnologia do sistema (Figura 2.3), foram identificadas as seguintes tecnologias:

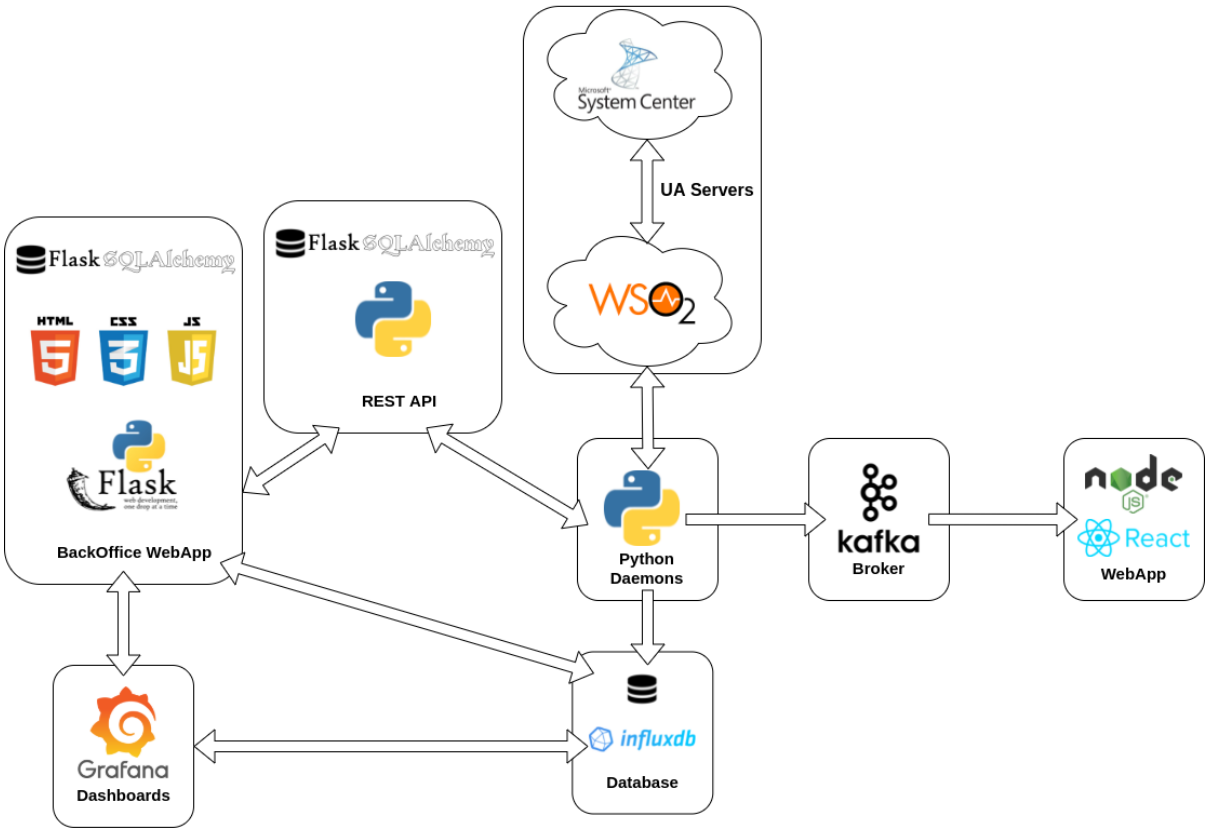

Figure 2.3: *Modelo de Tecnologia*

- Servidores da UA que fornecem os dados utilizados no projeto. Os dados provêm de uma API fornecida pelo System Center Operations Manager (SCOM), cujo acesso é mediado pelo Web Services Oxygenated (WSO2), que atua como proxy, limitando o acesso à API do SCOM.

- Apache Kafka para o message broker, devido ao facto de ser uma plataforma que permite o armazenamento e consumo de informação num modelo produtor-consumidor, por ser escalável, robusta e segura, assim como fácil de utilizar e de aprender [3].

- Como para este projeto era necessário o armazenamento de Time Series Data (coleção de observações obtidas através da repetição de amostras ao longo do tempo), foi escolhida

a tecnologia InfluxDB, por ser uma base de dados especificamente optimizada para o armazenamento de grandes quantidades de dados do tipo time series, assim como para a sua consulta e gestão.

- Servidor NodeJS e frontend em React para a WebApp que apresenta dados em tempo real. NodeJS foi escolhido por ter uma curva de aprendizagem pequena, por ser robusto e escalável (facilmente se adicionam funcionalidades extra ao servidor). Tal como o NodeJS, React foi escolhido por ter uma curva de aprendizagem pequena, assim como pela sua versatilidade (reutilização de componentes) e performance.

- Grafana que permite geração de dashboards tendo como datasource a base de dados InfluxDB. Para esta escolha inicialmente foi posta a opção de escolher a tecnologia PowerBi, no entanto após uma pesquisa foi decidido que a melhor opção seria Grafana pois é o mais adequado para a visualização de dados guardados em Time-Series Databases e oferece uma maior versatilidade em termos de filtragem de dados a apresentar.

- HyperText Markup Language (HTML), Cascading Style Sheets (CSS) e Javascript para o frontend da plataforma Web do Backoffice pelo facto do grupo já ter alguma experiência com estas tecnologias e querer agilizar o processo de desenvolvimento do frontend desta plataforma.

- Flask para o Backoffice, por ser um microFramework versátil, fácil de aprender e poderoso. Para a escolha da tecnologia do Backoffice foi considerada a opção Django, no entanto o framework que foi escolhido foi Flask pelo facto de vários membros do grupo já terem contactado com o mesmo em cadeiras passadas (o que agilizou bastante o processo de desenvolvimento do Backoffice). Foi desenvolvida também uma pequena base de dados de suporte ao backoffice com a biblioteca SQLAlchemy.

- Finalmente para a API desenvolvida foi também usada a linguagem de programação Python, assim como o framework Flask e a biblioteca SQLAlchemy para uma pequena base de dados de suporte.

2.4.3 Modelo de Domínio

O modelo de domínio (Figura 2.4) descreve o fluxo de informação dos vários intervenientes e recursos do sistema.

Analisando a figura constatamos que um utilizador pode definir, se quiser, métricas que representam fontes de dados que vão ser recolhidos e dashboards. Cada métrica é caracterizada por um id, nome, descrição, endpoint (de onde são recolhidos os dados que representam essa métrica), query_frequency (que representa a frequência com que vão ser feitos os pedidos ao endpoint fornecido) e um conjunto de fields, caso o utilizador queira filtrar os campos do dos dados provenientes do endpoint fornecido.

A cada métrica está associado um ou mais indicadores e o mesmo indicador pode estar associado a métricas diferentes (ex.: número total de resoluções Domain Name System (DNS) no último mês, e número total de dispositivos ligados à rede da UA no último mês).

Finalmente, as dashboards criadas pelo utilizador, têm a si associadas um id único (não controlado pelo utilizador), um nome definido pelo utilizador, uma visibilidade (pública ou privada) e um conjunto de painéis. Cada painel possui um gráfico de um determinado tipo

(Figura 3.19) e um ou mais indicadores associados. O conteúdo dos painéis é totalmente controlado e definido pelo utilizador, assim como a visibilidade da dashboard.

É importante referir também que as dashboards estão associadas num sistema de pastas no Grafana em que cada pasta pertence ao utilizador.

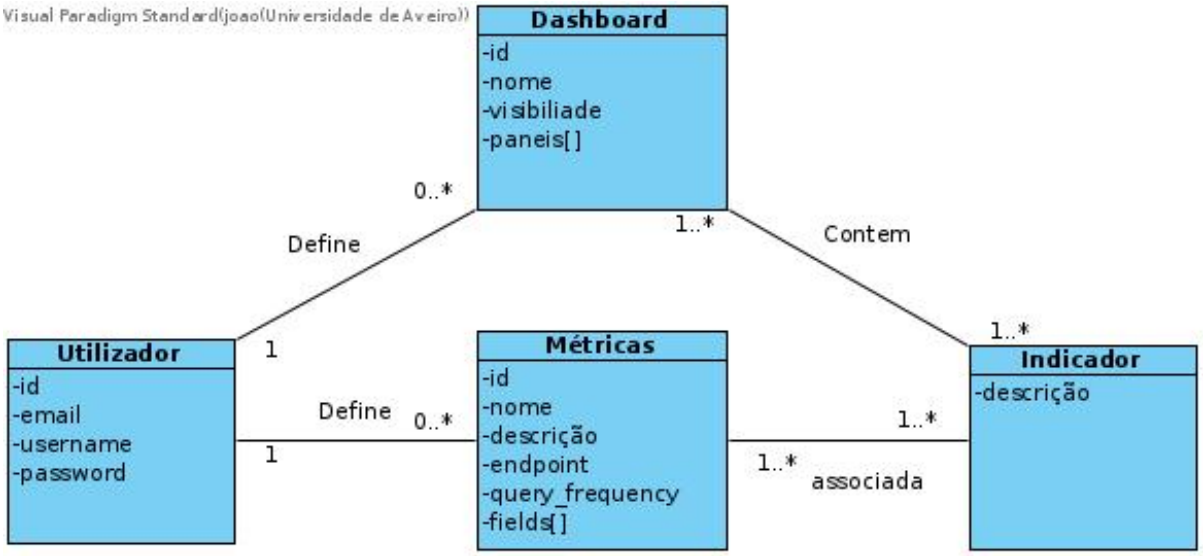

Figure 2.4: *Modelo de Domínio*

2.4.4 Modelo de instalação

Na Figura 2.5 é possível observar o diagrama de instalação desenvolvido para o projeto.

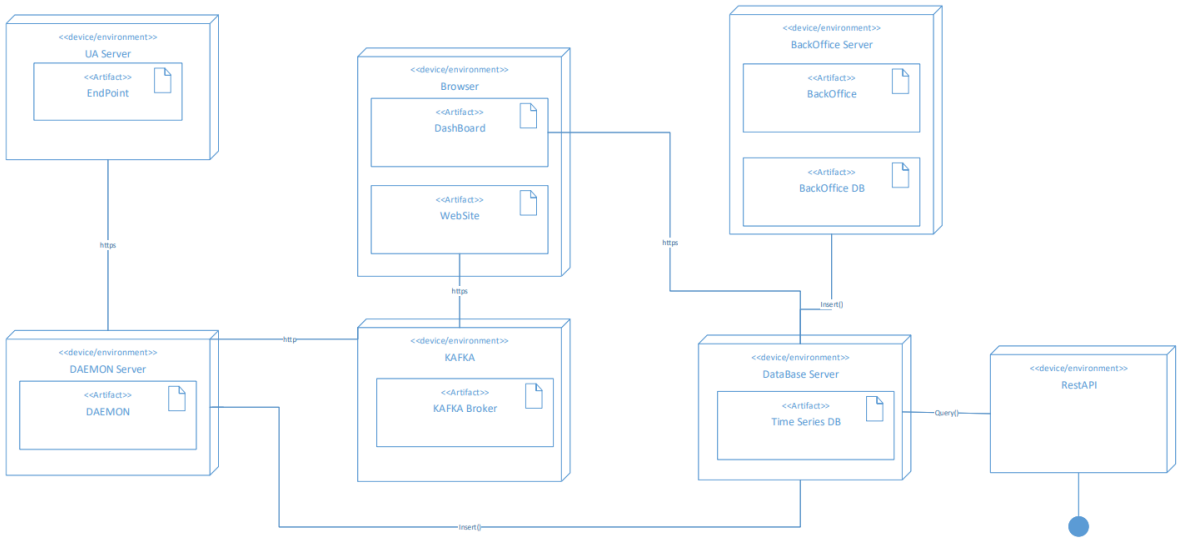

Figure 2.5: *Modelo de Instalação*

Capítulo 3

Implementação

3.1 Repositórios

Para o desenvolvimento, armazenamento e partilha do código foi criada uma organização na plataforma Github intitulada KIUA-PEI [4]. As motivações foram centralizar todos os repositórios associados ao projecto agilizando processos de permissão de acesso para a equipa de alunos que desenvolveu. Com esta organização fica também mais fácil de divulgar o trabalho e guardar para referências futuras.

Repositório	**Descrição**
KIUA_microsite	Contem o código do micro site do projeto
KIUA_WebApp	Contém código do servidor e da página web de apresentação de dados em tempo real
KIUA_Backoffice	Contém o código da plataforma Web de gestão de métricas e dashboards intitulada Backoffice
Daemon	Scripts Python para fazer requests e parse de dados.
DAEMONS	Repositorio para consumo da API de KPIS e encaminhamento para a influxDB
UnitFiles	Ficheiros .service implementados para automatizar o Deamon e o Kafka Broker nas máquinas virtuais

Tabela 3.1: Repositórios do Projeto

3.2 Kafka Broker

Kafka é uma plataforma distribuída de streaming orientada a serviços em tempo real, tais como: monitorização e recolha de métricas ou de eventos em websites, análise em tempo real, captura e analise de informação provinda de sensores, etc. Kafka tem ainda uma alta escalabilidade, é estável e apresenta um desempenho excelente.

No caso deste projecto, foi implementado apenas um broker que está a funcionar numa máquina virtual Ubuntu 18.04 associado à Azure.

Neste broker, configuramos os Internet Protocol (IP) e as portas da máquina através de uma conexão Secure Shell (SSH) disponiblizada pela Azure à Virtual Machine (VM). O Kafka funciona por tópicos e buffers de informação, criados dinamicamente ou na configuração do broker. As plataformas (dispositivos produtores) que geram os eventos, colocam informação nos tópicos, enquanto que outras plataformas (dispositivos consumidores), que subscrevem os mesmos e retiram a informação dos tópicos.

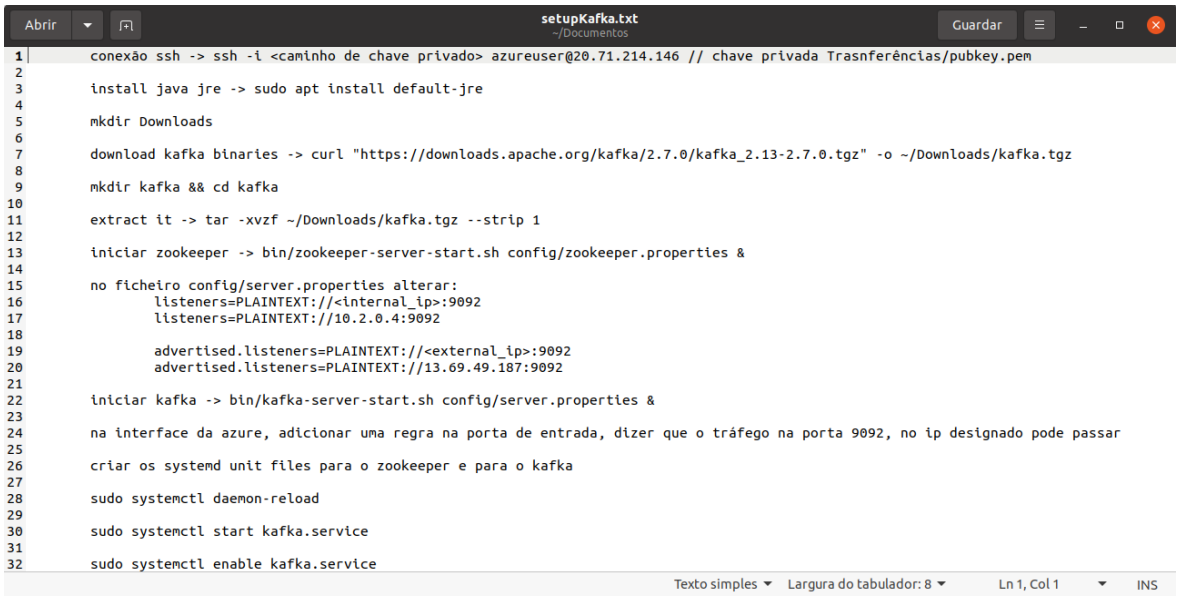

Figure 3.1: *Passos para configurar o broker kafka da nossa aplicação numa máquina*

Para este sistema foram criados 2 tópicos, parking e wifiusr, que disponibilizam informação em tempo real do estacionamento e do número de dispositivos ligados à rede wireless eduroam.

Na altura do planeamento da arquitetura do projeto foi posta em consideração se o Kafka Broker realmente era preciso ou não (os dados recolhidos iam diretamente para o website em tempo real em vez de ficarem armazenados em tópicos). Após pesquisa e discussão chegou-se à conclusão de que o Kafka era importante por uma questão de crescimento do sistema e de escalabilidade. Caso os dados ficassem armazenados em tópicos, muito facilmente se adiciona uma nova página web com um consumir embutido para subscrever do mesmo tópico sem ser necessário alterações no sistema. Concluindo, a utilização do Kafka é importante pois deixa espaço para que o sistema possa crescer em segurança.

Para automatizar o Kafka na máquina virtual, optamos por usar Systemd. Assim, para ativar o serviço, apenas tínhamos que ligar a máquina, pois o mesmo iria ser inicializado durante o arranque.

Figure 3.2: *Systemd Unit File for Kafka broker*

Uma vantagem não explorada no projecto foi a escalabilidade da plataforma, pois o Kafka pode ter múltiplos brokers para não só permitir redundância e robustez, como para servir múltiplos serviços em diferentes pontos geográficos. O processo de sincronização da informação é feito automaticamente pelo serviço Kafka, tirando essa responsabilidade ao utilizador.

Foi perdido algum tempo a registar todos os passos da configuração e do setup do Kafka para facilitar mudar, ou não, o broker para outra máquina virtual em caso de falha alheia ou de ficarmos sem créditos na Azure.

3.3 Systemd

O Systemd é um gestor de serviços de um sistema, presente na maioria das distribuições Linux. É possível configurar unit files para automatizar tarefas comuns implementadas em máquinas e servidores como deploy de serviços e lançamento de daemons.

Esta foi a metodologia escolhida para automatizar o servidor Kafka on boot e correr o daemon que recolhe e envia a informação para o Influx e para o Kafka.

3.4 InfluxDB e Grafana

O InfluxDB é uma *open-source time series database* e foi aqui que encontramos a base de dados perfeita para armazenar os dados que seriam recolhidos pelos nossos *daemons*. Como é descrito frequentemente, InfluxDB é otimizado para lidar com grandes taxas de escrita e de *queries* - precisamente o que era necessário para este projeto. Já o Grafana é uma aplicação *open source* que permite a criação e visualização de gráficos interativos.

Neste projeto a base de dados InfluxDB e o servidor de Grafana foram ambos instalados numa máquina virtual Ubuntu 18.04 hospedada nos serviços da Azure e com a comunicação a ser feita pelas portas 3000 e 8086 respetivamente. No início tinha sido configuradada uma máquina Windows que facilitava o acesso à máquina , por possuir uma interface gráfica, mas rapidamente se abandonou a ideia devido à elevada despesa. Desta forma, é dificultada a monitorização dos acessos para escrita\queries no terminal do InfluxDB, porém resulta num custo muito inferior.

Nesta base de dados foram guardados os dados recolhidos dos endpoints fornecidos (quer sejam os default quer sejam os do utilizador). Devido a ser uma base de dados de séries temporais, é armazenado em conjunto com os dados, um instante temporal que representa o instante em que a métricas foi recolhida.

O acesso à base de dados é apenas feito pelo Grafana conseguir gerar dashboards com gráficos que explicitam a informação que está na base de dados, ou para inserção e remoção de dados. A remoção é feita através das opções especificadas no Backoffice.

Na Figura 3.3 é possível vizualizar um exemplo do tipo de dados armazenados nesta base de dados.

```
> SELECT * FROM wifiusr WHERE time > now() -4h
name: wifiusr
time                           Nome       wifiCount
----                           ----       ---------
2021-06-25T10:47:40.714868Z aauav      1
2021-06-25T10:47:40.714868Z biblioteca 286
2021-06-25T10:47:40.714868Z cicfano    53
2021-06-25T10:47:40.714868Z cpct       51
2021-06-25T10:47:40.714868Z dao        31
2021-06-25T10:47:40.714868Z dbio       121
2021-06-25T10:47:40.714868Z dcspt      52
2021-06-25T10:47:40.714868Z deca       95
2021-06-25T10:47:40.714868Z decivil    40
2021-06-25T10:47:40.714868Z degeit     63
2021-06-25T10:47:40.714868Z dem        48
2021-06-25T10:47:40.714868Z dep        65
2021-06-25T10:47:40.714868Z deti       103
2021-06-25T10:47:40.714868Z dlc        47
2021-06-25T10:47:40.714868Z dmat       52
2021-06-25T10:47:40.714868Z dq         102
2021-06-25T10:47:40.714868Z essua      112
2021-06-25T10:47:40.714868Z fis        131
2021-06-25T10:47:40.714868Z geo        31
2021-06-25T10:47:40.714868Z ietta      0
2021-06-25T10:47:40.714868Z isca       57
2021-06-25T10:47:40.714868Z it         32
```

Figure 3.3: *Dados relativos ao número de dispositivos ligados à WIFI em cada departamento*

3.5 Página Web de Consumo em Tempo Real

Esta plataforma foi implementada com recurso a um servidor NodeJS e um frontend escrito em React e, tal como foi referido anteriormente, tem como principal objetivo a apresentação de dados em tempo real.

Todos os dados apresentados nesta página provêm de tópicos Kafka armazenados no message broker. Para a implementação da visualização de dados em tempo real decidimos adoptar uma lógica cliente-servidor em que os dados são enviados do servidor para o cliente, sendo que o cliente estaria implementado no código frontend e o servidor no backend. Ao estudar o problema percebemos que seria necessário usar uma metodologia de comunicação por eventos, ou seja, sempre que são recebidos novos dados (um novo evento), estes têm de ser apresentados na página. Por este motivo decidimos utilizar um socket que se adequava ao problema que se estava a tentar resolver. Após a decisão de utilizar um socket a escolha final recaía apenas em que tecnologia usar. Durante a pesquisa foram encontradas duas, *Websocket* e *Socket.IO*. A escolha final foi de *Socket.IO* pois esta biblioteca apresenta uma maior fiabilidade que a *Websocket*, assim como restabelecimento da sessão automático em caso de falha. [5]

3.5.1 Backend

Para a implementação do servidor utilizámos a framework do NodeJS intitulada Express que oferece um conjunto de ferramentas que auxiliam no desenvolvimento de aplicações web e mobile.

O servidor foi configurado para apresentar o frontend da página desenvolvido com react no seu caminho raiz ('/'). Este servidor possui também um socket à escuta no porto 4001 para envio da informação relativa aos dados em tempo real.

Na Figura 3.4 é possível ver a configuração do servidor, assim como a criação do socket. É importante referir que para a implementação deste servidor foi necessário ter atenção às políticas de Cross-Origin Resource Sharing (CORS) que são usadas para controlar pedidos a recursos de origens diferentes do servidor. Por isso, foi configurado tanto no servidor como no socket para garantir que é possível o acesso a quaisquer recursos de qualquer origem (neste caso era necessário ultrapassar esta política de segurança pois os dados provinham de um servidor diferente daquele donde está a correr a aplicação). [6]

```
/* Server Code */
const PORT = process.env.PORT || 4001;
var cors = require('cors');
const express = require('express');
const APPINDEX = 'build/index.html';

const server = express()
  .use(cors())
  .use(express.static('build'))
  .get('/', (req, res) => res.sendFile(APPINDEX, { root: __dirname }))
  .listen(PORT, () => console.log(`Listening on ${PORT}`));

const io = require('socket.io')(server, {
  cors: {
    origin: '*',
  }
});
```

Figure 3.4: *Configuração do servidor e do Socket*

Após a implementação do servidor, foi necessário construir o consumidor e a função para receção dos dados. Na Figura 3.5 podemos observar dois blocos de código em que o primeiro representa a criação do consumidor que recolhe dados de dois tópicos e o segundo representa a função de consumo (receção de mensagens).

No caso da criação do consumidor é possível verificar que a conexão ao broker instalado numa máquina virtual é feito com recurso à classe KafkaClient. Na inicialização do cliente é especificado o endereço da máquina virtual para que a ligação seja feita ao broker pretendido. Posteriormente, é instanciado um consumidor que tem como argumentos o cliente criado, os tópicos a subscrever e outras configurações. Após esta inicialização o consumidor fica oficialmente pronto para receber mensagens dos dois tópicos lá presentes.

No segundo bloco de código temos evidenciado o processo de receção de uma mensagem. Dentro desta função faz-se a verificação de qual o tópico de onde veio a mensagem e envia-se através do socket instanciado para o 'evento' correto (evento 'PARKING' se a informação vier do tópico parking e evento 'ROUTERS' se a informação vier do outro tópico).

```
index.js
index.js > kafkaHost
 9  //Creating the consumer
10  Consumer = kafka.Consumer,
11  client = new kafka.KafkaClient({kafkaHost: "13.69.49.187:9092"}),
12  consumer = new Consumer(
13    client,
14    [{ topic: 'parking', fromBeginning: true }, { topic: 'wifiusr', fromBeginning: true }],
15    { autoCommit: false },
16    { connectionTimeout: 20000000},
17  );
18
19
20  //Consume messages handler
21  const consume = async () => {
22    await consumer.connect()
23
24    consumer.on('message', function (message) {
25      const data = JSON.parse(new Buffer.from(message.value).toString());
26      if(message.topic == 'parking'){
27        console.log("PARKING");
28        parkData = data;
29        io.emit('Parking', data);
30      } else {
31        console.log("ROUTERS");
32        wifiData = data;
33        io.emit('Router', data);
34      }
35      //console.log(data);
36    });
37  }
38
```

Figure 3.5: *Funções de criação e consumo do consumidor Kafka*

3.5.2 Frontend

No que diz respeito ao frontend da aplicação o desenvolvimento foi dividido por quatro componentes para agilizar e simplificar o código.

No componente InfoSection existe um ficheiro RealTime.js que contém o código de apresentação dos dados em tempo-real. No topo desse ficheiro existe uma função 'UseEffect' que é responsável por fazer a conexão ao socket que está à escuta do lado do servidor e processar quaisquer novos eventos de receção de informação.

Tal como podemos verificar na Figura 3.6, assim que são recebidos os novos dados, é feito o parse aos mesmos e construído um objeto JavaScript Object Notation (JSON) resultado. O conteúdo desse objeto é posteriormente apresentado na página frontend. Analogamente ao que acontecia no servidor, a receção de informação está divida em vários eventos e é realizado um processamento diferente de acordo com o evento de onde se recebe a informação, isto é, se a informação for recebida no evento 'Parking' então o JSON construído é diferente do caso do evento 'Router'.

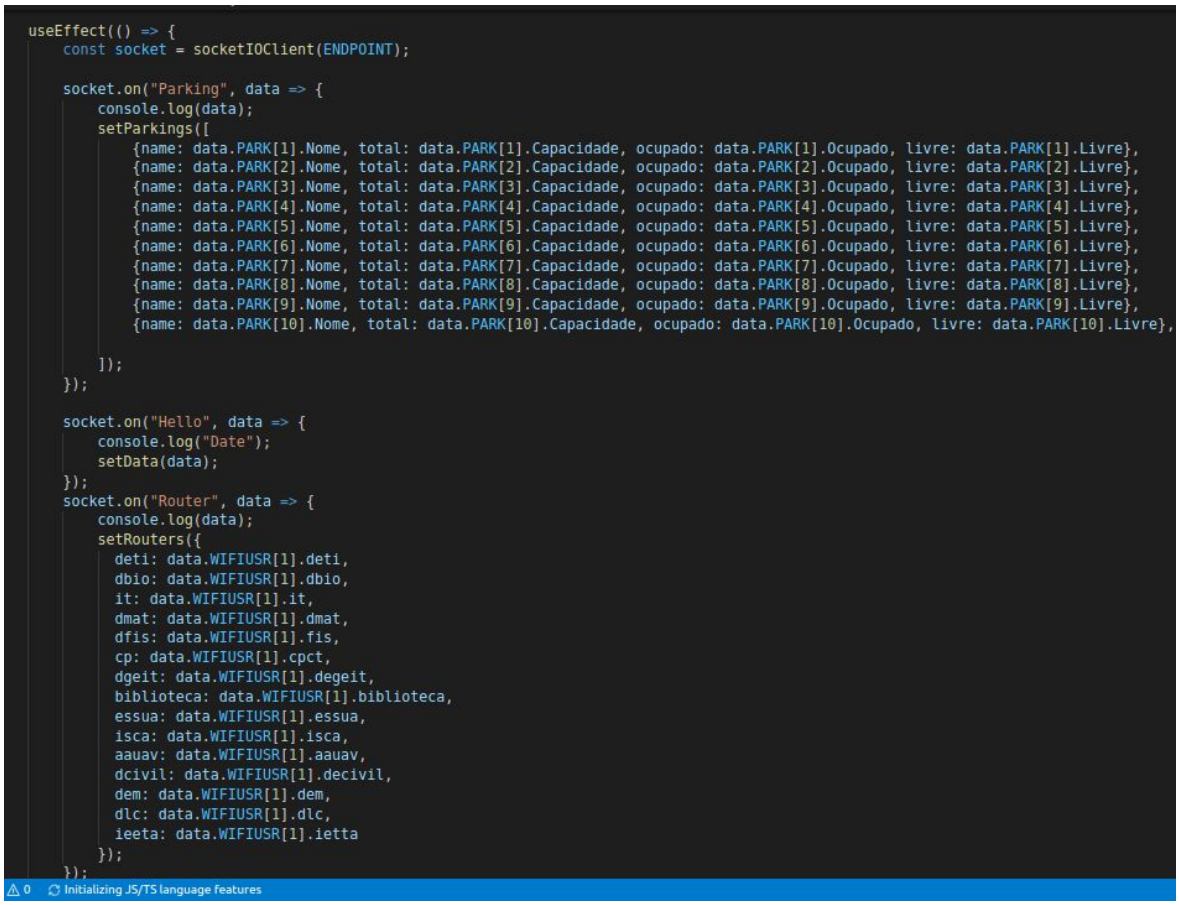

Figure 3.6: *Código da receção de informação no cliente*

O objeto JSON construído é guardado numa variável para ambos os casos, que por sua vez é chamada no código HTML para apresentar todos os dados de maneira organizada. Dando um exemplo em concreto, na Figura 3.6 foi criada uma variável de estado *routers* que recebe o valor do *data* através do *setRouters*. Após a mudança de estado, o conteúdo da variável *routers* passa a ser o JSON recebido que contém a informação dos departamentos. Finalmente resta apenas colocar no código HTML algo como:

```
<h2>{routers.deti} Users</h2>
```

para o caso específíco do número de dispositivos ligados aos pontos de acesso do Departamento de Electrónica Telecomunicações e Informática (DETI).

Na Figura 3.7 é possível observar um dos exemplos de apresentação de dados em tempo real. Este exemplo por sua vez resulta de um objeto JSON em que a cada departamento corresponde o número de dispositivos ligados aos seus pontos de acesso num determinado momento.

Nas Figuras 3.8 e 3.9 já podemos observar outro tipo de dados relativos à ocupação dos parques de estacionamento da universidade.

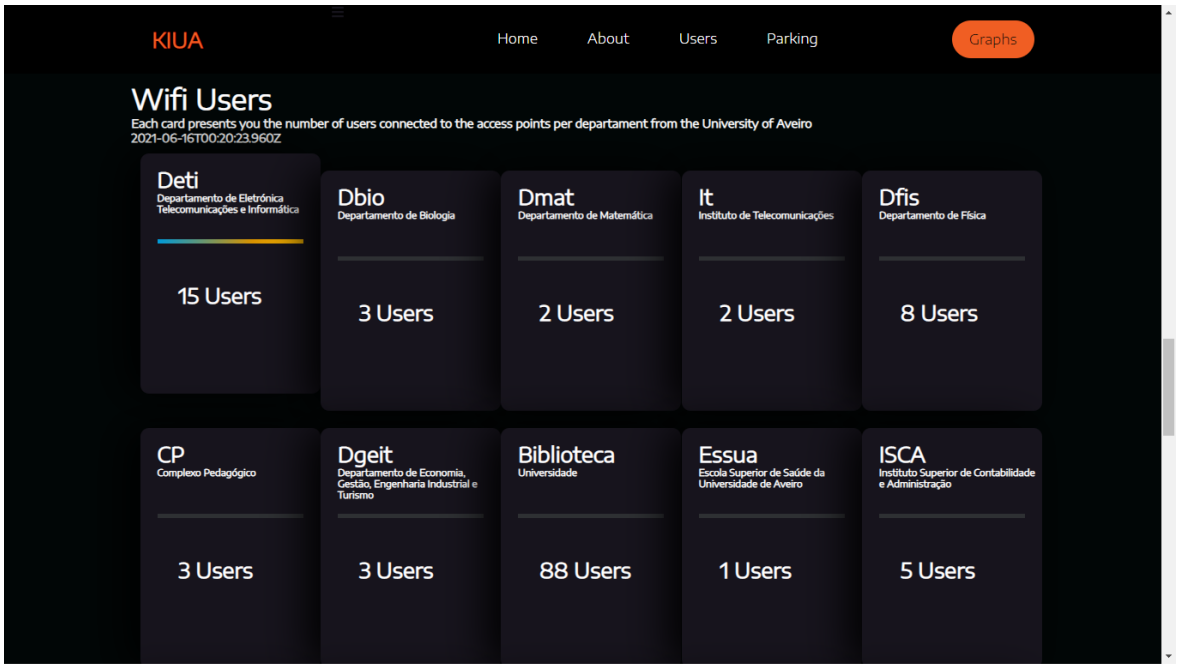

Figure 3.7: *Número total de dispositivos ligados aos access points de cada departamento em tempo real*

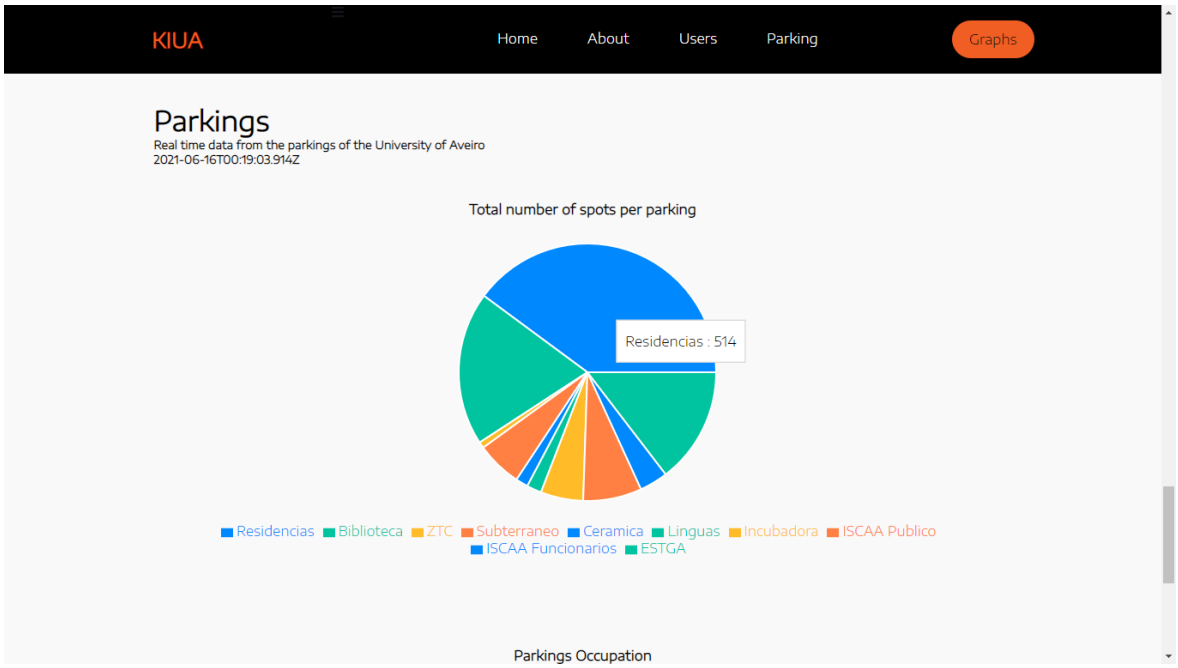

Figure 3.8: *Número total de lugares por estacionamento*

Figure 3.9: *Ocupação em tempo real dos parques de estacionamento*

Para fazer *deployment* do *website* utilizámos a solução *app-service* oferecida pela Azure e, por isso, o site encontra-se disponível em: https://peiwebapp.azurewebsites.net/.[7]

3.6 Automatização dos requests

A linguagem escolhida para recolher informação dos endpoints disponibilizados e para automatizar a recolha da informação foi Python pela sintaxe simples e fácil de aprender, e por termos descoberto duas bibliotecas, que permitiam produzir e consumir dados para um Kafka broker, [8] e para enviar dados para a base de dados influxDB, [9]. Apesar de open-source, tinham suporte para as necessidades que o projecto exigia nas trocas de informação entre os diferentes componentes do sistema.

3.6.1 Metrics API

Para implementar a lógica de criação e gestão de métricas no backoffice, foi tomada a decisão de criar uma Rest API. Esta API disponibiliza vários métodos para gerir cada uma das métricas (e a recolha de dados a si associados), criadas a partir do Backoffice. O deploy desta API foi feito numa VM da Azure com Nginx e Gunicorn e foi usada uma chave privada para limitar o acesso à mesma.

Para adicionar uma métrica é necessário fornecer o ID, Uniform Resource Locator (URL), período e os campos a recolher sendo que por default recolhe todos. Este último detalhe é usado para o caso em que o utilizador sabe o formato dos dados que o seu endpoint fornece e quer filtrar, escolhendo apenas alguns campos. Para o bom funcionamento desta API foi criada uma base de dados que armazena os dados relativos às várias métricas criadas pelos utilizadores.

Tipos de Autenticação

Esta API permite recolher informação de APIs com os seguintes tipos de autenticação:

- **Open API** URL sem necessidade de autenticar.
- **Key Authentication** URL com autenticação através de uma chave fixa.
- **Http Authentication** URL com autenticação através de um username e password.
- **Token Authentication** URL com autenticação através de um token, sendo que é necessário fornecer o TokenURL, UserSecret, Author, ContentType.

Operações

Para cada um dos tipos de autenticação é possível realizar as seguintes operações:

- /**Add** Método usado para adicionar uma nova métrica.
- /**Remove**/<**ID**> Método usado para remover uma métrica dado um ID.
- /**Start**/<**ID**> Método usado para recomeçar iniciar a recolha dado um ID.
- /**Pause**/<**ID**> Método usado para pausar a recolha de dados de uma métrica dado um ID.
- /**Change**/<**ID**> Método usado para alterar um dos campos de uma Métrica.

Estes métodos são chamados, de acordo com as operações realizadas pelo utilizador através do Backoffice, de forma a permitir uma gestão das métricas ou correcção de algum dos seus campos caso tenha ocorrido um erro na recolha.

Alguns destes métodos não são usados na atual implementação do Backoffice (ex.: se um utilizador eliminar uma métrica é feita a chamada ao método */Remove/<ID>*), no entanto esta API foi construída a pensar no futuro, ou seja, de modo a que sempre que possível sejam adicionadas novas funcionalidades ao Backoffice e essas novas funcionalidades impliquem a chamada aos mesmos.

3.6.2 Base de Dados

A base de dados é constituída por 4 tabelas (uma para cada tipo de autenticação) sem relações entre si, sendo que em cada uma destas tabelas são armazenados os campos das métricas de acordo com o seu tipo de autenticação. O seu esquema encontra-se representado na Figura 3.10.

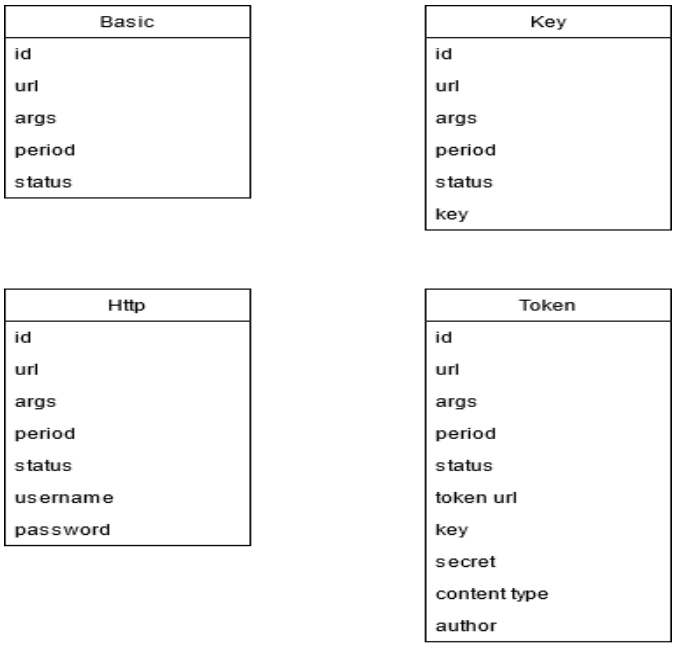

Figure 3.10: *Base de Dados da API*

3.6.3 Requests

Para recolher toda a informação usada para formar os diferentes KPI's do utilizador foi implementado um módulo que, usando os diferentes Schedulers, vai fazer pedidos (de acordo com o tipo de autenticação) ao endpoint fornecido pelo utilizador. Esse pedido tem um timeout de 40 segundos de forma a não interferir com nenhumas outras métricas que estão a ser recolhidas pelo mesmo Scheduler. Caso ocorresse algum erro na recolha, a ideia seria enviar uma mensagem de erro para o backoffice para que o utilizador o possa corrigir, mas não chegámos a implementar esta parte.

Estes pedidos têm algumas diferenças dependendo do tipo de autenticação, por isso, vão ser apresentados ao longo das próximas secções, incluindo vários blocos de pseudo-código que permitem a melhor compreensão de todo o processo.

Open Authentication

Para este tipo de pedidos apenas era necessário fazer um pedido por URL, e se houver alguma métrica com URL repetido usamos o mesmo formato JSON (pois a resposta de dois URLs iguais é um objeto JSON com o mesmo formato) para obter a informação aplicando os diferentes filtros necessários em cada métrica. No entanto, isto só é aplicável em APIs sem autenticação pois todos os pedidos são iguais. Esta situação não acontece em APIs com autenticação, pois diferentes utilizadores podem ter respostas diferentes.

```
#Pseudo-Código da recolha de dados em APIs sem autenticação
for distinct urls in database:
    request(url)
    for args in metrics with url:
```

```
        filter(args)
        format influx
        send to influx
```

Key/Http

Devido ao uso de autenticação a recolha tem de ser efectuada de forma independente, pois diferentes utilizadores podem receber diferentes respostas.

```
#Pseudo-Código da recolha de dados nas APIs com autenticação do tipo Key/HTTP
for urls in database:
    request(url,authentication)
    filter(args)
    format influx
    send to influx
```

Token Authentication

Para autenticação com recurso a tokens o processo é diferente. Estes tokens vão sendo actualizados pelo servidor, sendo assim necessário pedir um de cada vez que for feito um pedido ao servidor.

Em geral os tokens costumam ser actualizados mais ou menos de hora em hora, por isso nas métricas com períodos menores (5,15,30 minutos) foi decidido que seria melhor guardar os tokens usados de modo a evitar a necessidade de pedir um novo token cada vez que se faz um pedido. Isto pode evitar dezenas de pedidos no caso de métricas com uma frequência de 5 em 5 minutos.

```
#Pseudo-Código que representa a recolha de dados de APIs com Token Authentication
for urls in database:

    if id in tokens:
        token = tokens[id]
    else:
        token = request_token
        if period < 60:
            tokens[id] = token

    request(url,token)
    if authentication_error and period < 60:
        token = request_token
        tokens[id] = token
        request(url,token)

    filter(args)
    format influx
    send to influx
```

3.6.4 Scheduler

Scheduler é uma biblioteca em Python que permite iniciar processos em segundo plano com um determinado período. No contexto do projeto os Schedulers foram usados para recolher a informação necessária para criar as KPI's e enviar para a base de dados, pelo que vamos lançar um scheduler por cada métrica e período. Como existem 4 tipos de métricas (uma por cada tipo de autenticação) e 5 períodos diferentes (5 em 5 minutos, 30 em 30 minutos, hora a hora e dia a dia) vão existir 20 Schedulers.

No entanto, podem ocorrer alguns problemas a fazer os pedidos aos endpoints pois cada um destes pedidos tem um timeout de 40 segundos, e se a resposta associada a cada uma das métricas demorar o tempo máximo, mais o tempo de formatar o ficheiro JSON e enviar para a influxDB, o período do Scheduler pode ser facilmente ultrapassado.

Para evitar esse problema decidimos permitir que cada Scheduler possa lançar um novo Scheuduler quando ultrapassa o limite do número de métricas por minuto de forma a conseguirmos fazer todos os pedidos sem qualquer problema, isto é, um Scheduler pode recolher cinco métricas cuja periodicidade de recolha seja de 5 em 5 minutos (15 se a periodicidade for de 15 em 15 minutos, etc). Caso seja adicionada uma nova métrica com recolha de 5 em 5 minutos, o próprio Scheduler lança um novo para ficar responsável pelas próximas 5 métricas.

Mesmo com a possibilidade de criar novos Schedulers, ainda podem ocorrer problemas pois a máquina tem um limite de Schedulers. Por isso, num ambiente mais realista onde teríamos de um número bastante elevado de métricas a recolher, seria ideal ter a possibilidade de expandir o número de máquinas virtuais a recolher as métricas e dividir de acordo com a sua capacidade.

```
#Lançar Schedueleres para iniciar a recolha das KPI's
schedueleres = {'5':1,'15':1,'30':1,'60':1,'1440':1}

schedueler = new Schedueler()
schedueler.add(start_requests,minutes=5)
schedueler.add(start_requests,minutes=15)
schedueler.add(start_requests,minutes=30)
schedueler.add(start_requests,minutes=60)
schedueler.add(start_requests,minutes=1440)

def start_requests(id,period):
    count = 0
    for url in database with period==period:
        count += 1
        if count >= period*(request_id-1) and count < period*id:
            make_request
        elif id+1 > schedueleres[str(period)] and count+1 > period*id:
            new schedueler(id+1,period)
    if count == 0:
        remove schedueler(id)
```

É importante referir que um utilizador pode remover as suas métricas ou pode ocorrer um

erro na recolha de dados. Logo, o sistema deve reagir a esses casos, por isso, caso um Scheduler não tenha nenhuma métrica para recolher deve terminar o seu processo com a possibilidade de recomeçar, caso seja necessário.

3.6.5 Filtragem

De forma a dar possibilidade ao utilizador de filtrar os diferentes campos dos ficheiros JSON que provêm dos seus endpoints, foi necessário criar um algoritmo que percorra todo o ficheiro e seleccione os campos indicados pelo utilizador.

Como o ficheiro pode ter diferentes formatos, alguns mais complexos que outros, o algoritmo criado é recursivo para juntar os diferentes campos numa lista em que todos os elementos estão no mesmo formato para assim conseguirmos enviar para a base de dados.

Figure 3.11: *Ficheiro JSON*

No exemplo da Figura 3.11 é possível observar um ficheiro com quatro campos, sendo que três deles são inteiros e um é uma lista de dicionários. O objectivo do algoritmo é pegar em todos os objectos que estão fora da lista e juntá-los com os elementos da lista. Por exemplo, se o utilizador quiser o "first","last"e todos os "clientCount"o resultado será uma lista de dicionários com o formato {"first":0,"last":0,"clientCount":0}.

O algoritmo (Figura 3.12) vai percorrer todos os campos do ficheiro e adicionar a:

- **Fields** são os campos do objecto JSON que vão ser recolhidos para juntar com as listas que possam vir a aparecer no ficheiro. Caso o ficheiro não tenha nenhuma lista a função

retorna os Fields. Os campos que vão ser recolhidos são todos os objectos de um tipo primitivo ou dicionários que estejam contidos numa key (formato "key":). Neste último caso vai ser chamada a função recursivamente, e se a função retornar um elemento é adicionado aos Fields, se retornar uma lista com vários elementos o resultado vai ser adicionado às Entrys e é tratado como uma lista de elementos.

- **Entrys** representa a lista de todos os elementos contidos em listas ou dicionários caso o ficheiro seja uma lista de dicionários, antes de retornar a lista de todos os elementos recolhidos são adicionados os Fields a todos os elementos da lista juntado os elementos dos dois grupos.

Para auxiliar o algoritmo de filtragem, foi desenvolvida uma função *merge_entrys* (Figura 3.13) que junta todos os elementos de um dicionário fields a todos os elementos da lista entrys. Esta função só é chamada no fim da recolha para impedir que a ordem possa afectar o resultado final.

```python
def merge_filter(data,args):
    entrys = []
    fields = {}
    for field in [field for field in data]:
        if isinstance(field,str):
            if isinstance(data[field],dict):
                aux = merge_filter(data[field],args)
                if len(aux) == 1:
                    fields.update(aux[0])
                elif aux:
                    entrys += aux
            elif not isinstance(data[field],str) and isinstance(data[field],list):
                aux = []
                for val in data[field]:
                    aux += merge_filter(val,args)
                entrys += merge_entrys(aux,fields)

            elif args == 1:
                fields[field] = data[field]
            elif field in args:
                fields[field] = data[field]

        elif isinstance(field,dict):
            entrys += merge_filter(field,args)

        elif isinstance(field,list):
            aux = []
            for val in field:
                aux += merge_filter(val,args)
            entrys += aux

    return merge_entrys(entrys,fields)  if entrys else [fields]
```

Figure 3.12: *Algoritmo para filtrar o ficheiro JSON*

```python
def merge_entrys(entrys,fields):
    for entry in entrys:
        entry.update(fields)
    return entrys
```

Figure 3.13: *Função merge_ entrys*

Resultados

Na Figura 3.14 é possível ver o resultado de uma filtragem a um objeto JSON (cujo conteúdo vinha no formato apresentado na Figura 3.11) em que foi especificado que apenas se queria o *macAddress*, a *location*, o *clientCount*, *first* e *last*.

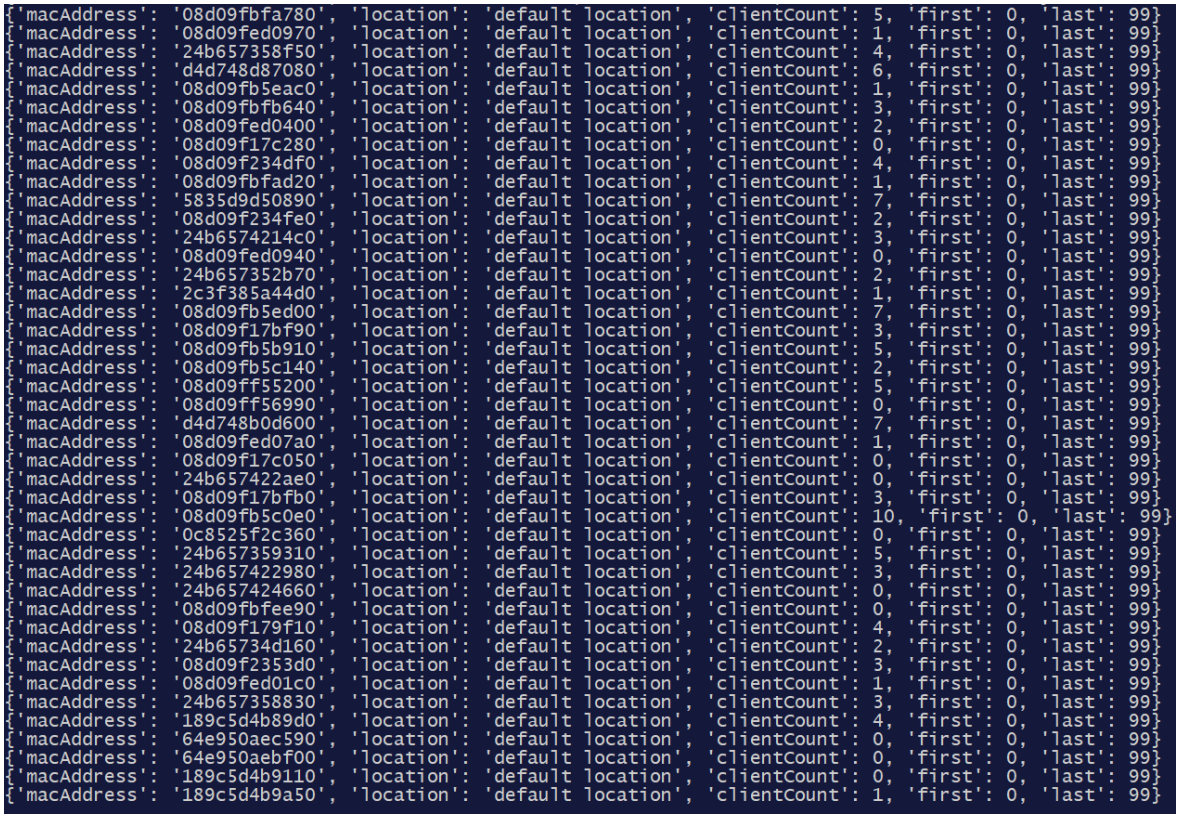

Figure 3.14: *Resultados da função de filtragem a um objeto JSON*

O resultado vai ser a junção dos campos filtrados na lista de "accessPoints"com os campos recolhidos fora da lista que vão ser adicionados a todos os elementos da lista.

Erros

Devido à natureza do problema, é possível que ocorram alguns erros a formatar o ficheiro pois não conseguimos adaptá-lo a todos os tipos de ficheiros JSON.

Alguns dos problemas que podem surgir são:

- **Mais de que uma lista** No caso de aparecerem várias listas no ficheiro, o ficheiro é filtrado sem problemas mas isto pode vir a causar problemas na organização da base de dados e levar a alguns conflitos na criação de dashboards se as listas tiverem keys com o mesmo nome mas com significados diferentes.

- **Listas de dicionários com formatos diferentes** Se o ficheiro tiver dicionários com formatos diferentes pode fazer com que o resultado final contenha algumas inconsistências, como elementos com keys a menos, como vai ser referido no próximo exemplo.

Um exemplo de um mau formato do ficheiro JSON é o ficheiro enviado pela API dos parques de estacionamento (Figura 3.15).


```
{'Timestamp': 1625415219}
{'ID': 'P1|4', 'Nome': 'Residencias', 'Latitude': 40.63155, 'Longitude': -8.656537, 'Capacidade': 514, 'Ocupado': 6, 'Livre': 508}
{'ID': 'P5', 'Nome': 'Biblioteca', 'Latitude': 40.630195, 'Longitude': -8.659275, 'Capacidade': 249, 'Ocupado': 6, 'Livre': 243}
{'ID': 'P6', 'Nome': 'ZTC', 'Latitude': 40.63041, 'Longitude': -8.659082, 'Capacidade': 10, 'Ocupado': 0, 'Livre': 10}
{'ID': 'P8', 'Nome': 'Subterraneo', 'Latitude': 40.631876, 'Longitude': -8.657819, 'Capacidade': 65, 'Ocupado': 0, 'Livre': 65}
{'ID': 'P9', 'Nome': 'Ceramica', 'Latitude': 40.635008, 'Longitude': -8.658727, 'Capacidade': 20, 'Ocupado': 1, 'Livre': 19}
{'ID': 'P10', 'Nome': 'Linguas', 'Latitude': 40.635584, 'Longitude': -8.657832, 'Capacidade': 24, 'Ocupado': 1, 'Livre': 23}
{'ID': 'P11', 'Nome': 'Incubadora', 'Latitude': 40.636305, 'Longitude': -8.657178, 'Capacidade': 69, 'Ocupado': -1, 'Livre': 70}
{'ID': 'P13', 'Nome': 'ISCAA Publico', 'Latitude': 40.63096, 'Longitude': -8.652294, 'Capacidade': 95, 'Ocupado': 1, 'Livre': 94}
{'ID': 'P14', 'Nome': 'ISCAA Funcionarios', 'Latitude': 40.630261, 'Longitude': -8.652237, 'Capacidade': 46, 'Ocupado': 0, 'Livre': 46}
{'ID': 'P15', 'Nome': 'ESTGA', 'Latitude': 40.574671, 'Longitude': -8.443438, 'Capacidade': 188, 'Ocupado': -1, 'Livre': 189}
```


Figure 3.15: *Formato da informação da API dos parques de estacionamento*

Este ficheiro contém uma lista de dicionários em que o primeiro elemento contém a Timestamp e os restantes a informação dos parques de estacionamento. Uma possível correcção seria por exemplo ter a Timestamp com uma key "Timestamp"e a informação dentro de uma lista com a key "Parques"de forma a permitir distinguir os 2, mas como estão todos dentro da mesma lista leva a que o resultado da filtragem também tenha um elemento só com a Timestamp.


```
{'Timestamp': 1625503940}
{'Nome': 'Residencias', 'Capacidade': 514, 'Ocupado': 1, 'Livre': 513}
{'Nome': 'Biblioteca', 'Capacidade': 249, 'Ocupado': 54, 'Livre': 195}
{'Nome': 'ZTC', 'Capacidade': 10, 'Ocupado': 0, 'Livre': 10}
{'Nome': 'Subterraneo', 'Capacidade': 65, 'Ocupado': 0, 'Livre': 65}
{'Nome': 'Ceramica', 'Capacidade': 20, 'Ocupado': 7, 'Livre': 13}
{'Nome': 'Linguas', 'Capacidade': 24, 'Ocupado': 13, 'Livre': 11}
{'Nome': 'Incubadora', 'Capacidade': 69, 'Ocupado': 26, 'Livre': 43}
{'Nome': 'ISCAA Publico', 'Capacidade': 95, 'Ocupado': 3, 'Livre': 92}
{'Nome': 'ISCAA Funcionarios', 'Capacidade': 46, 'Ocupado': 2, 'Livre': 44}
{'Nome': 'ESTGA', 'Capacidade': 188, 'Ocupado': 22, 'Livre': 166}
```


Figure 3.16: *Resultado da filtragem aos dados da API dos parques de estacionamento com os argumentos "Timestamp","Capacidade","Livre","Ocupado","Nome"*

3.6.6 Format Influx

Antes de enviar a informação para a influxDB é necessário especificar a tabela, timestamp, tags e fields. A informação que vai ser formatada é o resultado de um ficheiro json após a sua filtragem. O algoritmo de formatação dos dados para estarem prontos a ser enviados para a base de dados influxDB está representado na Figura 3.17.

- **timestamp** Este campo não é necessário para enviar para a base de dados mas é importante para uma correta apresentação dos gráficos no Grafana, por isso, permitimos ao utilizador gerar a sua própria timestamp que deve ser enviada e incluída nos campos dos objetos json provenientes da recolha de dados associada à métrica. Caso não

exista um campo timestamp, é adicionada a timestamp Lisbon/London da altura que a informação foi recolhida.

- **measurement** Usado para especificar a tabela na base de dados. Como cada métrica vai ter a sua tabela específica, o nome da sua tabela vai ser o ID da métrica.
- **fields** Neste campo serão adicionados os valores usados para gerar KPI's. Para isto, apenas os campos numéricos serão usados (ex.: neste campo, considerando a API dos parques de estacionamento constaria informação como o número de lugar livres, ou de ocupados, etc).
- **tags** todos os campos que não sejam inteiros poderão dar erros na geração de dashboards, por isso serão usados como tags. Estas tags serão usadas como uma espécie de índice para identificar cada entrada na influxDB, também podem ser usadas para filtrar os valores nos gráficos gerados no Grafana (ex.: no caso da API dos parques neste campo viria algo como o nome do parque de estacionamento, para que no gráfico fosse apresentado, por exemplo, o número de lugares livres do parque com este nome).

```python
def format_influx(metric_id,data):
    result = []
    for entry in [entry for entry in data if entry]:
        add_entry = {"measurement":metric_id,"tags":{'id':metric_id}}

        if 'Timestamp' in entry:
            add_entry['time'] = dt.fromtimestamp(entry['Timestamp']).isoformat()
            del entry['Timestamp']
        else:
            add_entry['time'] = str(get_timestamp())

        for key in [key for key in entry if isinstance(entry[key],str)]:
            add_entry['tags'][key] = entry[key]
            del entry[key]

        if entry:
            add_entry['fields'] = entry
            result.append(add_entry)

    return result
```

Figure 3.17: *Função format_influx*

3.7 Plataforma Web Backoffice

A solução de Backoffice foi a tarefa mais challenging do projeto. Embora simplista, foi possível a construção das funcionalidades mais importantes, que dão nome ao projecto, recolher e automatizar métricas a partir de fontes de dados do utilizador, e gerar, dinamicamente, dashboards a partir das métricas disponíveis.

O Backoffice foi desenvolvido com Flask. Flask é considerado um microframework Python por ter um conjunto core de funcionalidades, mas é poderoso por ser extensível, possuir módulos de base de dados, autenticação, gestão de erros, templating, recursos que estão muitas vezes associados a projectos Flask. [10]

3.7.1 Prototipagem

No sentido de colocar todos os elementos do grupo no mesmo contexto daquilo que iria ser o Backoffice e para agilizar o desenvolvimento da interface foi elaborado um protótipo de baixa fidelidade. O protótipo foi feito utilizando diagrams.net. Este acabou por se revelar muito útil, especialmente nas primeiras semanas de desenvolvimento, e ficou muito perto da solução final.

O protótipo de baixa fidelidade encontra-se representado na Figura 3.18, e para uma análise mais detalhada (devido ao tamanho elevado da imagem) sugere-se a ida ao link: https://kiua-pei.github.io/KIUA_microsite/doc/backoffice_prototype.png.

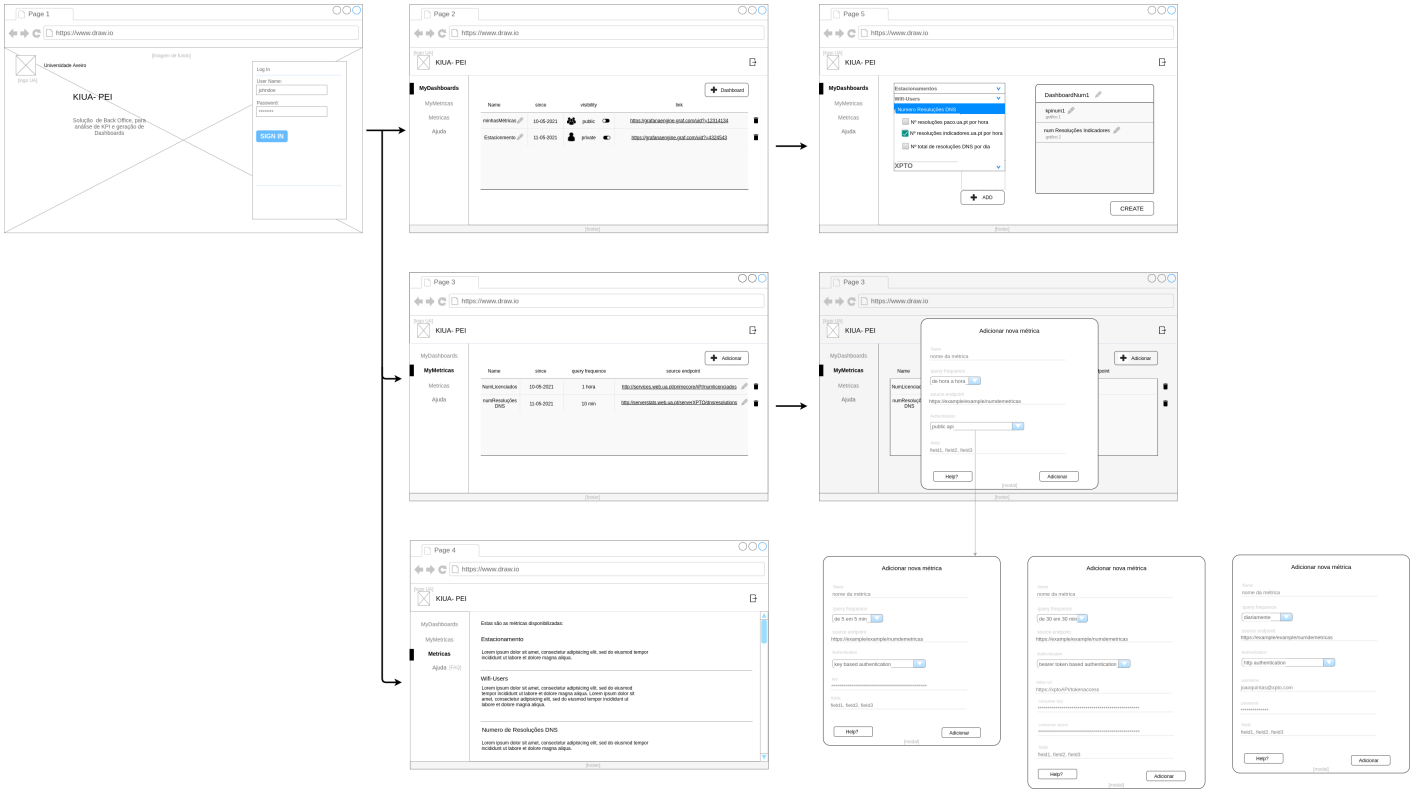

Figure 3.18: *Protótipo baixa fidelidade do Backoffice*

3.7.2 Geração automática de dashboards

Para criação e manipulação de painéis interativos no Grafana foi desenvolvida em Python a classe Dashboard que foi integrada com a plataforma Backoffice para permitir a geração automática de dashboards.

Esta classe possui métodos que permitem a manipulação de objetos JSON que representam a dashboard segundo as directrizes definidas pelo Grafana. Assim sendo, utilizamos como ferramenta auxiliar para construção desses objectos JSON a pasta json_files que é constituída por 8 ficheiros, em que um deles tem uma estrutura que representa uma parte específica de uma dashboard.

Os vários ficheiros do diretório json_files são:

- **dashboard.json** - este ficheiro é lido durante a fase de inicialização do objecto, pelo facto de guardar a estrutura base para a criação de uma dashboard onde são carregados os restantes componentes json que compõem os painéis.
- **target.json** - corresponde à estrutura onde estão inseridos os querys da base de dados, que permitem especificar que conteúdo da base de dados vai ser apresentado num painel. A estrutura do target define apenas um único query, mas um painel pose ser constituído um ou mais targets (ou até mesmo nenhum), logo quando o utilizador especifica mais do que um query num painel são criados vários targets.

Para além do ficheiro que especifica a estrutura base, existem mais 6 que permitem indicar qual o tipo de painel a ser criado, e onde mais tarde são inseridos os targets/queries associados. O Grafana oferece uma grande variedade de painéis para a visualização dos dados, porém o grupo apenas escolheu implementar 6 modelos por corresponderem a estruturas de visualização em timeseries, que são adequadas para apresentação dos dados do projecto. Os seis tipos de gráficos escolhidos estão todos visíveis na Figura 3.19 e são:

Figure 3.19: *Tipos de gráficos disponíveis*

- **Graph descrito em graph.json** estrutura para representação de dados em formato de linear.

- **HeatMap descrito em heatmap.json** Estrutura para representação de dados em formato de histograma temporal.
- **Pie Chart descrito em piechart.json** Estrutura para representação de dados em formato de gráfico circular. Este tipo de gráfico é adequado para comparação rápida de pequenos conjuntos de valores.
- **Bar Gauge descrito em bargauge.json** Estrutura para representação de dados em formato de gráfico de barras.
- **Gauge descrito em gauge.json** Estrutura para representação de dados em formato de valor único apresentado num medidor.
- **Stat descrito em stat.json** Estrutura para representação de dados mais estatísticos que pode ser composto por vários mini gráficos onde cada um apresenta um valor instantâneo da estatística.

Por fim, quando o objecto JSON que representa dashboard já está elaborado, este é enviado via a API do Grafana através de um métoto Hypertext Transfer Protocol (HTTP) POST. No servidor da Azure aonde foi realizada a instalação da base de dados InfluxDB também foi instalado o serviço Grafana. Esta decisão de pôr ambos os serviços na mesma máquina devu-se devido ao facto de eles terem de estar constantemente a comunicar pois os dados apresentados nas dashboards provêm da base de dados.

No Grafana instalado no servidor optou-se criar apenas uma conta e todas as dashboards eram guardadas nessa conta. Para ser possível a exportação de gráficos do Grafana para os incluir no website foi necessário alterar o ficheiro de configuração */etc/grafana/grafana.ini*.

```
#Alterações feitas ao ficheiro grafana.ini
# para permitir a exportação de gráficos
allow_embedding = true
#em auth.anonymous
enabled = true
org_name = Main Org.
org_role = Viewer
```

Estando todas as permissões e configurações feitas faltava apenas decidir como se iriam organizar as dashboards no servidor. Foi então definido que iria ser seguida uma lógica de pastas em que para cada utilizador existe uma pasta associada no servidor Grafana.

Quando é criada uma conta no website, é simultaneamente criada uma pasta associada a cada utilizador no Grafana e a partir desse momento sempre que um utilizador criar uma dashboard esta é colocada na sua pasta. O nome dessa pasta é o id que é associado ao utilizador no momento da criação da conta.

Esta opção permitiu uma melhor gestão das dashboards e dos seus nomes dados por cada utilizador (dois utilizadores podem ter uma dashboard com o mesmo nome). Este aspecto é, no entanto, totalmente invisível para o utilizador na medida em que toda a interação que ele tem com o sistema acontece através da execução de tarefas na interface do Backoffice, tarefas essas que são traduzidas em pedidos à API do Grafana, através da chamada dos métodos da classe Dashboard com as opções especificadas pelo utilizador.

Na Figura 3.20 é possível ver a divisão por pastas das dashboards pertencentes a vários utilizadores que já criaram conta.

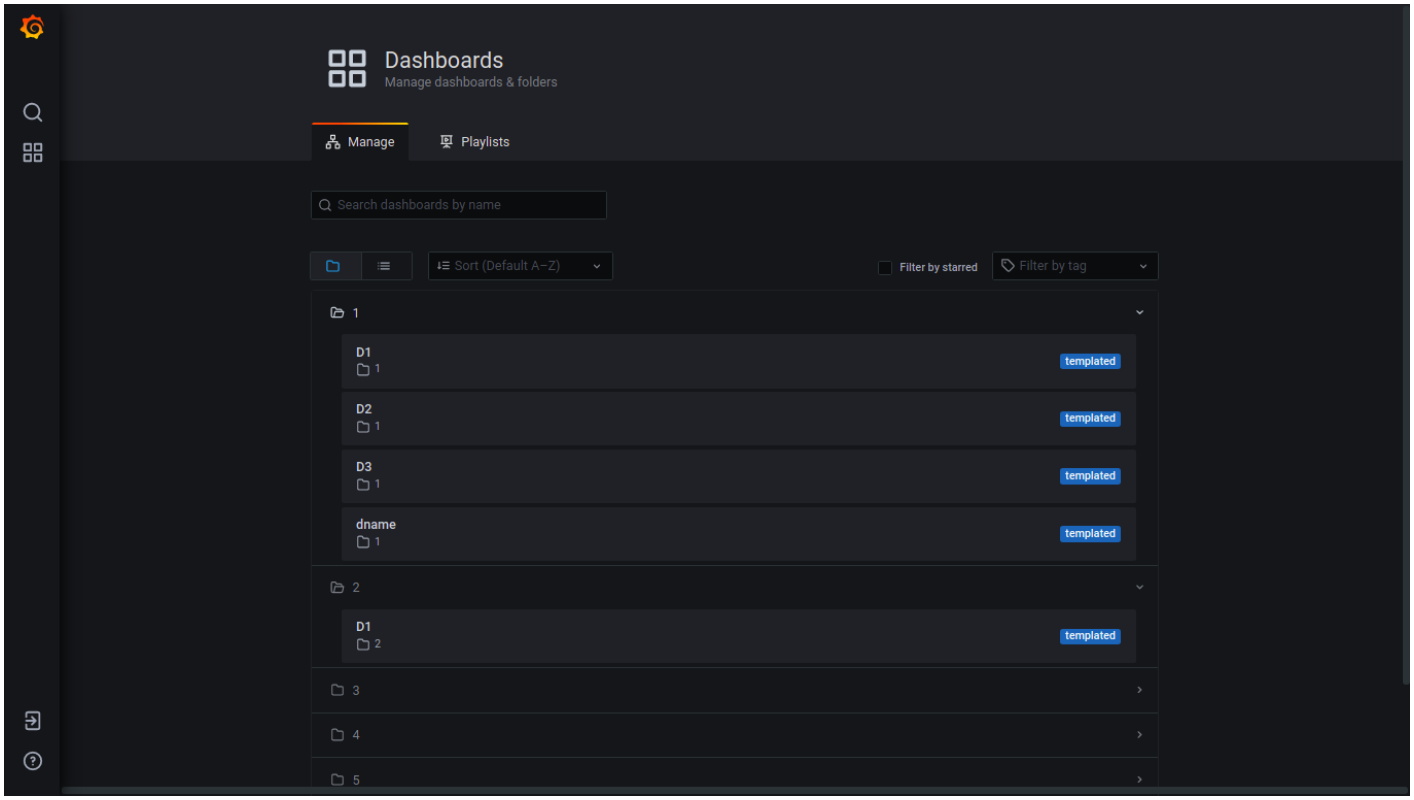

Figure 3.20: *Divisão das dashboards no servidor Grafana por pastas*

Na tabela 3.2 estão descritos de modo resumido todos os métodos da classe Dashboard.

Método	Descrição
get_panels	Devolve uma lista de strings que contem o nome de todos os painéis de um objecto Dashboard
del_dash	Elimina dashboard do servidor Grafana utilizando a API atravês de um método HTTP request delete
send_dash	Enviar via API do Grafana uma estrutura JSON que representa uma Dashborad, através de um método HTTP POST
dash_set_time	Permite definir a janela temporal apresentada nos gráficos de uma dashboard (default: últimas 6h)
add_panel	Adiciona uma estrutura JSON ao objeto JSON que representa a dashboard. Esta estrutura a ser adicionada representa um tipo de painel (graph, gauge, bargauge, heatmap, stat, piechart), que é lido de um ficheiro template
del_panel	Elimina um parte especifica da estrutura JSON que representa uma Dashboard que corresponde ao painel que se pretende apagar
add_query	Adiciona um query a uma dashboard. A adição é feita, integrando o query num ficheiro template target.json, e colocando o resultado deste processo na estrutura JSON que compõe a Dasboard
create_folder	Criação de uma pasta no servidor Grafana. É enviada através da API do grafana uma estrutura JSON que representa uma pasta através de um método http post

Tabela 3.2: Métodos da classe Dashboard

3.7.3 Base de Dados

Foi preciso desenvolver uma base de Dados associada ao Backoffice para guardar as contas dos utilizadores, as suas dashboards, métricas e ainda as métricas default do website. Esta base de dados tem também um propósito adicional que é gerir o que cada utilizador pode ou não ver. Por exemplo dashboards públicas têm o campo visibilidade a 1, e quando este campo está a 1 todos os utilizadores podem ver a dashboard. De forma similar todas as métricas presentes na tabela *DefaultMetrics* estão acessíveis para todos os utilizadores, enquanto que as presentes na tabela *MyMetrics* apenas estão presentes para o utilizador que as criou.

Escolhemos usar SQLite, formada a partir do framework SQLAlchemy.

Na Figura 3.21 está representado, o diagrama Entidade-Relação desta base de dados.

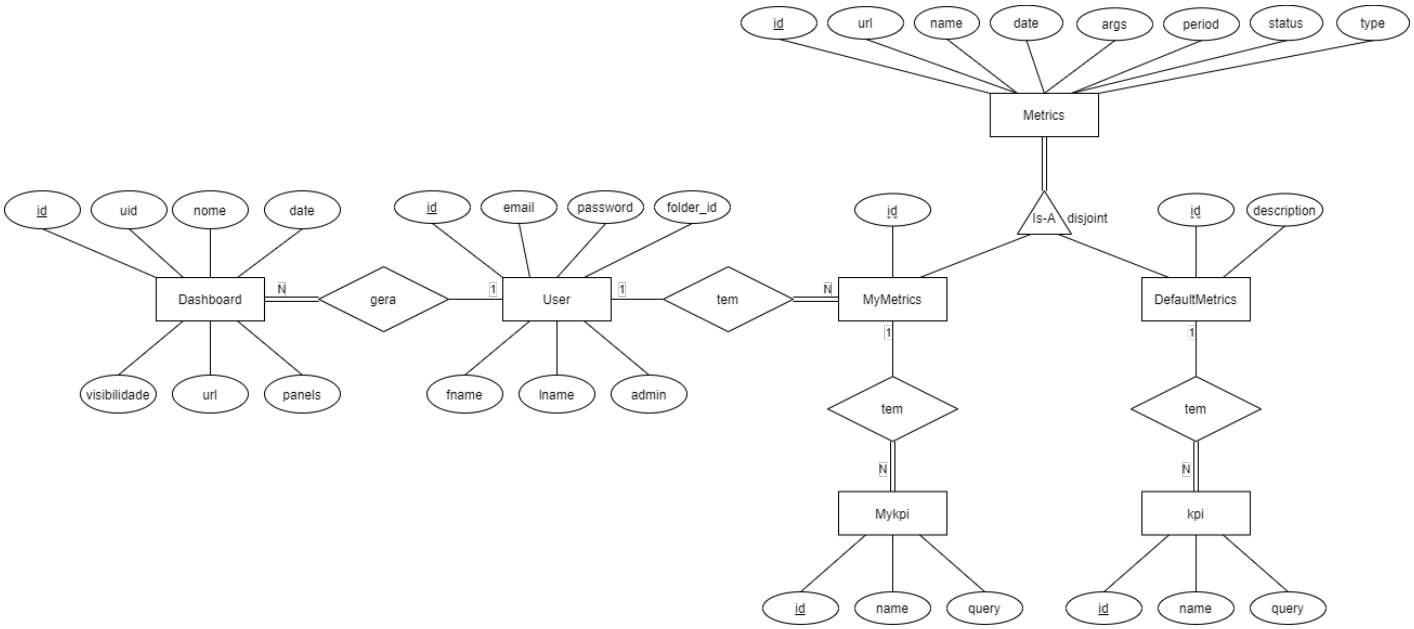

Figure 3.21: *Diagrama Entidade-Relação da Base de Dados do Backoffice*

3.7.4 Geração automática de indicadores

Como uma das principais motivações por detrás do desenvolvimento desta plataforma foi a de que a sua utilização não deveria implicar conhecimentos de bases de dados, SQL ou grafana, foi tomada a decisão de que após a criação de uma métrica, os indicadores a si associados seriam gerados automaticamente. Por este motivo foi criada uma função intitulada *get_ querys* que receberia como parâmetro de entrada o nome da tabela.

Para contextualizar, a cada métrica está associada uma tabela na base de dados *InfluxDB* que contém os dados recolhidos. Deste modo, a função recebe o nome da tabela e recolhe as Tags da tabela, os valores associados às Tags e as Keys. Após todos estes dados serem recolhidos, é gerada uma lista de indicadores em que cada indicador é representado por um tuplo, contendo o query gerado e uma string (para apresentação do indicador no website).

Assim sendo, o query gerado automaticamente fica com o formato:

```
SELECT '<field>' AS '<value><field>' FROM '<tablename>' WHERE '<tag>' = '<value>';
```

Dando um exemplo em concreto para ajudar a perceber melhor como funciona esta geração automática, no caso da recolha de dados dos parques de estacionamento o JSON devolvido é o da Figura 3.22

E neste caso, na base de dados *InflluxDB* o Tag está configurado como sendo o campo Nome. Todos os outros campos (Capacidade, Ocupado e Livre) são keys. A iteração é realizada sobre cada tag e cada valor de tag (neste caso iria iterar sobre todos os nomes Residencias, Biblioteca, etc). Para cada tag é colocado na cláusula *SELECT* as *KEYS* que são renomeadas por uma questão de auxilio à visualização dos gráficos. Um exemplo de um query gerado para este caso é:

```
SELECT Ocupado AS ResidenciasOcupado FROM <tablename> WHERE Nome = Residencias;
```

De notar que a renomeação é uma concatenação do valor da tag com a key a ser usada na cláusula *SELECT*.


```
parking = [
    {"Timestamp":1423519992},
    {"Nome":"Residencias","Capacidade":663,"Ocupado":5,"Livre":658},
    {"Nome":"Biblioteca","Capacidade":249,"Ocupado":2,"Livre":247},
    {"Nome":"ZTC","Capacidade":10,"Ocupado":0,"Livre":10},
    {"Nome":"Subterraneo","Capacidade":74,"Ocupado":0,"Livre":74},
    {"Nome":"Ceramica","Capacidade":42,"Ocupado":4,"Livre":38},
    {"Nome":"Linguas","Capacidade":36,"Ocupado":2,"Livre":34},
    {"Nome":"Incubadora","Capacidade":74,"Ocupado":22,"Livre":52},
    {"Nome":"ISCAA Publico","Capacidade":95,"Ocupado":0,"Livre":95},
    {"Nome":"ISCAA Funcionarios","Capacidade":46,"Ocupado":1,"Livre":45},
    {"Nome":"ESTGA","Capacidade":188,"Ocupado":15,"Livre":173}
]
```


Figure 3.22: *Objeto JSON devolvido pela API dos parques de estacionamento*

Para cada query é gerado um indicador automaticamente que corresponde, basicamente, à concatenação de uma string entre o valor da tag e a key. O resultado da geração de query's automáticas dos dados da API dos parques de estacionamento está representado na Figura 3.23.


```
Generating Querys for parking

('SELECT Capacidade AS "BibliotecaCapacidade" FROM "parking" WHERE Nome = \'Biblioteca\';', 'Capacidade from Biblioteca')
('SELECT Capacidade AS "CeramicaCapacidade" FROM "parking" WHERE Nome = \'Ceramica\';', 'Capacidade from Ceramica')
('SELECT Capacidade AS "ESTGACapacidade" FROM "parking" WHERE Nome = \'ESTGA\';', 'Capacidade from ESTGA')
('SELECT Capacidade AS "ISCAAFuncionariosCapacidade" FROM "parking" WHERE Nome = \'ISCAA Funcionarios\';', 'Capacidade from ISCAA Funcionarios')
('SELECT Capacidade AS "ISCAAPublicoCapacidade" FROM "parking" WHERE Nome = \'ISCAA Publico\';', 'Capacidade from ISCAA Publico')
('SELECT Capacidade AS "IncubadoraCapacidade" FROM "parking" WHERE Nome = \'Incubadora\';', 'Capacidade from Incubadora')
('SELECT Capacidade AS "LinguasCapacidade" FROM "parking" WHERE Nome = \'Linguas\';', 'Capacidade from Linguas')
('SELECT Capacidade AS "ResidenciasCapacidade" FROM "parking" WHERE Nome = \'Residencias\';', 'Capacidade from Residencias')
('SELECT Capacidade AS "SubterraneoCapacidade" FROM "parking" WHERE Nome = \'Subterraneo\';', 'Capacidade from Subterraneo')
('SELECT Capacidade AS "ZTCCapacidade" FROM "parking" WHERE Nome = \'ZTC\';', 'Capacidade from ZTC')
('SELECT Livre AS "BibliotecaLivre" FROM "parking" WHERE Nome = \'Biblioteca\';', 'Livre from Biblioteca')
('SELECT Livre AS "CeramicaLivre" FROM "parking" WHERE Nome = \'Ceramica\';', 'Livre from Ceramica')
('SELECT Livre AS "ESTGALivre" FROM "parking" WHERE Nome = \'ESTGA\';', 'Livre from ESTGA')
('SELECT Livre AS "ISCAAFuncionariosLivre" FROM "parking" WHERE Nome = \'ISCAA Funcionarios\';', 'Livre from ISCAA Funcionarios')
('SELECT Livre AS "ISCAAPublicoLivre" FROM "parking" WHERE Nome = \'ISCAA Publico\';', 'Livre from ISCAA Publico')
('SELECT Livre AS "IncubadoraLivre" FROM "parking" WHERE Nome = \'Incubadora\';', 'Livre from Incubadora')
('SELECT Livre AS "LinguasLivre" FROM "parking" WHERE Nome = \'Linguas\';', 'Livre from Linguas')
('SELECT Livre AS "ResidenciasLivre" FROM "parking" WHERE Nome = \'Residencias\';', 'Livre from Residencias')
('SELECT Livre AS "SubterraneoLivre" FROM "parking" WHERE Nome = \'Subterraneo\';', 'Livre from Subterraneo')
('SELECT Livre AS "ZTCLivre" FROM "parking" WHERE Nome = \'ZTC\';', 'Livre from ZTC')
('SELECT Ocupado AS "BibliotecaOcupado" FROM "parking" WHERE Nome = \'Biblioteca\';', 'Ocupado from Biblioteca')
('SELECT Ocupado AS "CeramicaOcupado" FROM "parking" WHERE Nome = \'Ceramica\';', 'Ocupado from Ceramica')
('SELECT Ocupado AS "ESTGAOcupado" FROM "parking" WHERE Nome = \'ESTGA\';', 'Ocupado from ESTGA')
('SELECT Ocupado AS "ISCAAFuncionariosOcupado" FROM "parking" WHERE Nome = \'ISCAA Funcionarios\';', 'Ocupado from ISCAA Funcionarios')
('SELECT Ocupado AS "ISCAAPublicoOcupado" FROM "parking" WHERE Nome = \'ISCAA Publico\';', 'Ocupado from ISCAA Publico')
('SELECT Ocupado AS "IncubadoraOcupado" FROM "parking" WHERE Nome = \'Incubadora\';', 'Ocupado from Incubadora')
('SELECT Ocupado AS "LinguasOcupado" FROM "parking" WHERE Nome = \'Linguas\';', 'Ocupado from Linguas')
('SELECT Ocupado AS "ResidenciasOcupado" FROM "parking" WHERE Nome = \'Residencias\';', 'Ocupado from Residencias')
('SELECT Ocupado AS "SubterraneoOcupado" FROM "parking" WHERE Nome = \'Subterraneo\';', 'Ocupado from Subterraneo')
('SELECT Ocupado AS "ZTCOcupado" FROM "parking" WHERE Nome = \'ZTC\';', 'Ocupado from ZTC')
```


Figure 3.23: *Resultado da função de geração de querys sobre dados da API dos estacionamentos*

Esta solução foi escolhida pelo facto de oferecer simplicidade ao utilizador, ou seja, todo o processo é gerado automaticamente. Para que o utilizador indicasse especificamente, seria necessário um conhecimento base de no mínimo SQL e Grafana. É compreensível também que esta solução possa muitas vezes apresentar resultados ao utilizador que ele não pretende utilizar ou muitas vezes gerar ambiguidades (ter duas tags que identificam a mesma coisa mas que têm nomes diferentes), no entanto após discussão com os membros do projeto chegou-se à conclusão que este seria o preço a pagar por não ter de efectuar todo o processo de especificar a query.

É importante referir que este processo de gerar querys/indicadores associados a uma métrica apenas acontece uma vez para cada métrica, e após estar finalizado, os indicadores são todos guardados na base de dados do Backoffice para que não seja preciso voltar a chamar a função.

3.7.5 Aplicação Web

Ao longo desta secção irão ser descritas as várias páginas que constituem a aplicação web desenvolvida, assim como as ações e tarefas que é possível realizar. Para cada ação e tarefa será explicado o que acontece no background sem o utilizador saber e que permite o bom funcionamento de todo o sistema.

Para o deployment do Backoffice foi usado o mesmo serviço da Azure usado no deploy da página de visualização de dados em tempo-real [7]. O Backoffice está disponível no link:

http://kiuabackoffice.azurewebsites.net/

Página de Login

Na Figura 3.24 é possível observar a página de login da aplicação onde se pode de maneira bastante simples fazer a autenticação que vai permitir o acesso aos serviços. Para o fazer apenas é necessário preencher os campos (email e password) e carregar no botão de *Login*. Após o fazer, o utilizador, será imediatamente redireccionado para a página inicial da aplicação caso a autenticação tenha corrido bem, caso contrário será apresentada uma mensagem de erro.

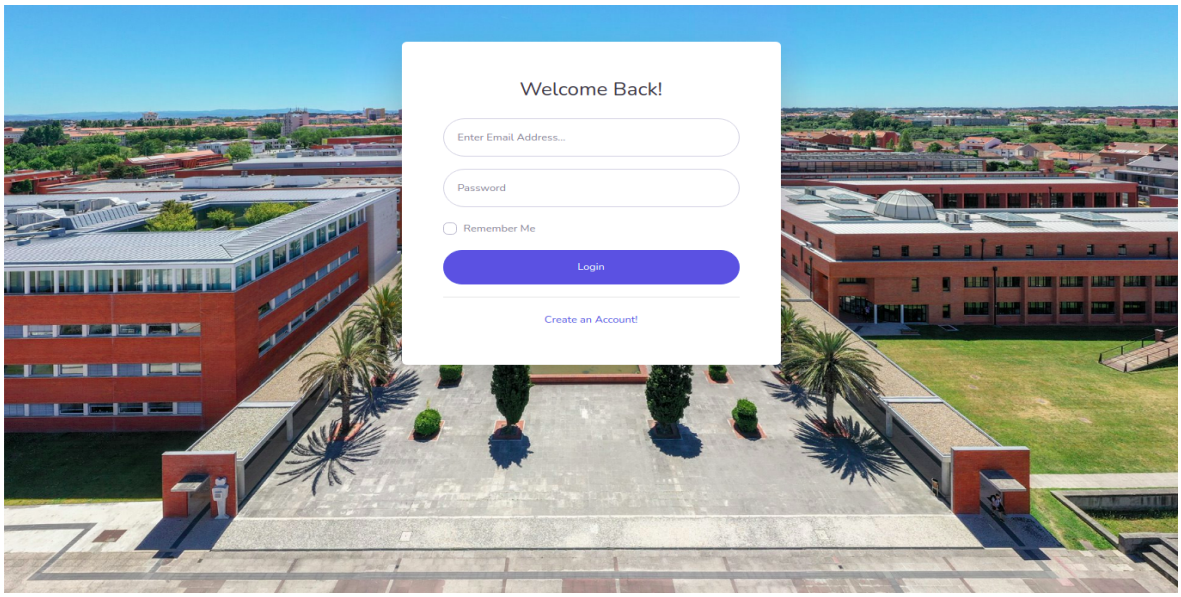

Figure 3.24: *Página de Login*

Caso ainda não possua uma conta, o utilizador apenas tem de carregar no link *Create an Account* e será redirecionado para a página de criação de conta.

Página de Register

Com um design bastante similar à página de *Login*, temos a página de Register (Figura 3.25) onde é possível a criação de contas. Para a criação apenas é necessário especificar o primeiro e último nome, um endereço de email e uma password (que deve ter pelo menos 7 caracteres).

O processo de criar uma conta envolve um conjunto de verificações entre as quais a validade do email e da password. Se o email especificado já existe na base de dados do Backoffice ou não (é feito um query para saber se o email já existe).

Assumindo que corre tudo bem na criação da conta é adicionado à base de dados um novo utilizador com os seus respetivos dados de conta, e no momento desta inserção é criado um identificador único que passa a estar associado a este utilizador.

No entanto, o processo de criar uma conta envolve mais um processo, que ocorre sem que o utilizador se aperceba, já referido anteriormente que é a chamada a um método da classe Dashboard (*create_folder*) que permite criar uma pasta no Grafana que foi instalado no servidor. Essa pasta vai ter como nome o identificador único associado ao utilizador que acabou de criar conta e serve para armazenar todas as dashboards do utilizador. A partir deste momento, sempre que for criada uma dashboard nesta conta é armazenada na pasta que foi criada no momento de registo.

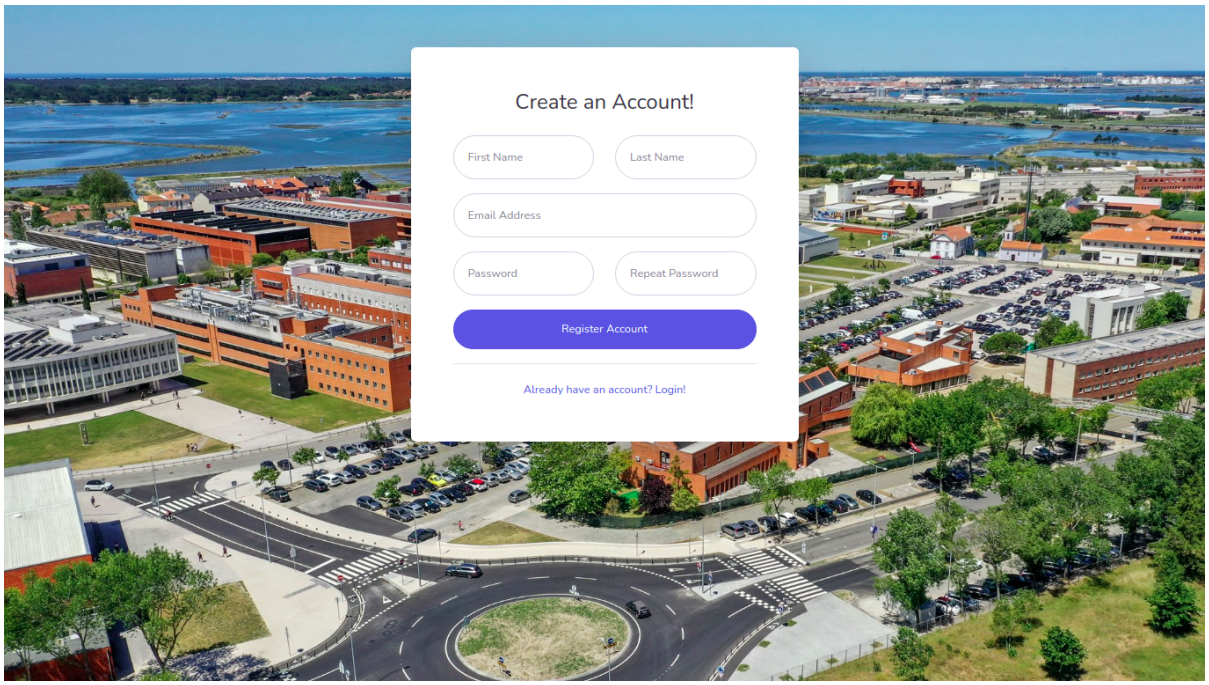

Figure 3.25: *Página de Register*

Página MyDashboards

A página *MyDashboards* (Figura 3.26) é a primeira que aparece após um *Login* ou *Register* efectuado com sucesso. Estando nela é possível observar e ter acesso tanto às dashboards criadas pelo próprio utilizador, como às públicas criadas por outros utilizadores.

Nas dashboards do próprio utilizador é apresentada uma tabela com o nome, data de criação, visibilidade, link e um botão que permite a eliminação da mesma. A visibilidade pode ser configurada como privada ou pública, sendo que para o fazer é apenas necessário carregar no botão de mudança de visibilidade (ex.: carregar no botão *public* se a dashboard for privada e se quiser alterar para pública). A alteração da visibilidade da dashboard implica apenas a modificação do campo visibilidade na base de dados do Backoffice que está a 1 se ela for pública e a 0 se a dashboard for privada. Tendo em conta o que acabou de ser referido, as dashboards que estão presentes na base de dados com o campo visibilidade a 1 estão visiveis para todos os utilizadores.

O botão de *delete* por sua vez apaga não só a *dashboard* da base de dados, mas também chama um método da classe Dashboard (*del_ dash*) que permite, através da API do Grafana, a eliminação da dashboard da pasta do utilizador presente no servidor.

Finalmente, em cada linha está presente um link que redireciona o utilizador para uma página do website onde é possível a visualização de todos os gráficos criados para a dashboard em questão.

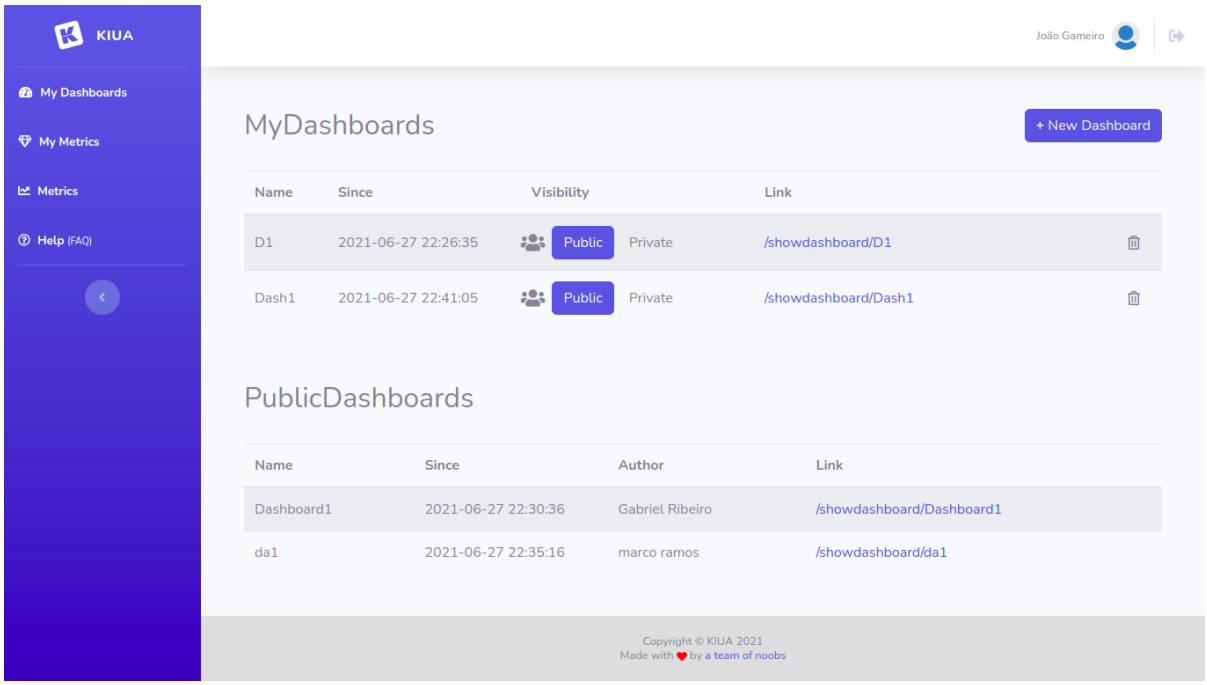

Figure 3.26: *Página MyDashboards*

Na secção *PublicDashboards* é possível observar dashboards que outros utilizadores configuraram como públicas. Não é possível alterá-las nem de qualquer modo eliminá-las, apenas é permitido a visualização de informação como o nome da dashboard, a data de criação o seu autor e o link para a visualização da mesma.

Finalmente, para criar dashboards é apenas necessário carregar no botão *+ New Dashboard*. Após carregar nesse botão, será imediatamente apresentado um pop-up (Figura 3.27) através do qual se especifica o nome da dashboard a criar.

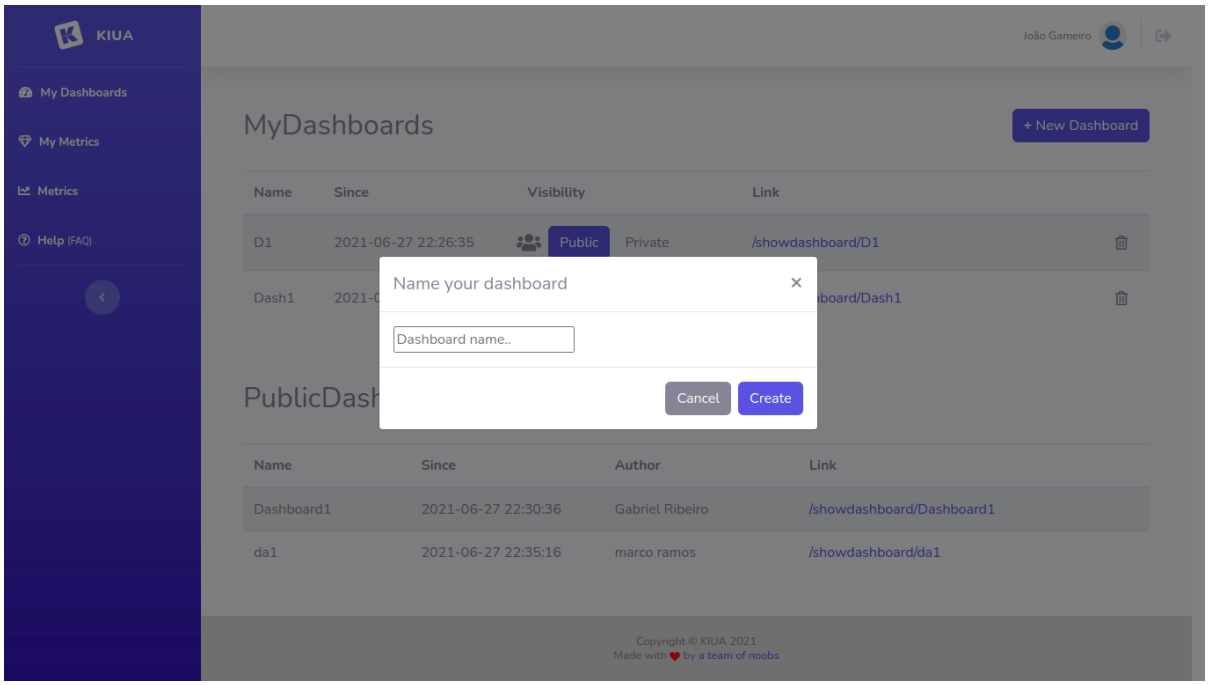

Figure 3.27: *Pop Up para dar nome à Dashboard*

Após a especificação do nome a dar à dashboard, é feito uma verificação na base de dados para garantir que este utilizador ainda não criou uma dashboard com o mesmo nome, pois caso tenha sido criada, não é permitido o avanço para a fase seguinte do processo de criação e é apresentada uma mensagem de erro. É importante referir que a divisão das dashboards no Grafana por pastas permite que múltiplos utilizadores possam ter uma dashboard com o mesmo nome, mas um utilizador não pode ter duas com o mesmo nome.

Página para Criação de Dashboards

Após carregar no botão de criação de uma nova dashboard e presumindo que o nome especificado é válido, o utilizador é redirecionado para a página representada na Figura 3.28.

Esta página está dividida em duas colunas, a do lado esquerdo onde se seleccionam todos os indicadores que se pretendem colocar na dashboard e a do lado direito que possui um painel onde são apresentados os gráficos adicionados até ao momento à dashboard (é atualizado sempre que o utilizador adiciona um novo gráfico).

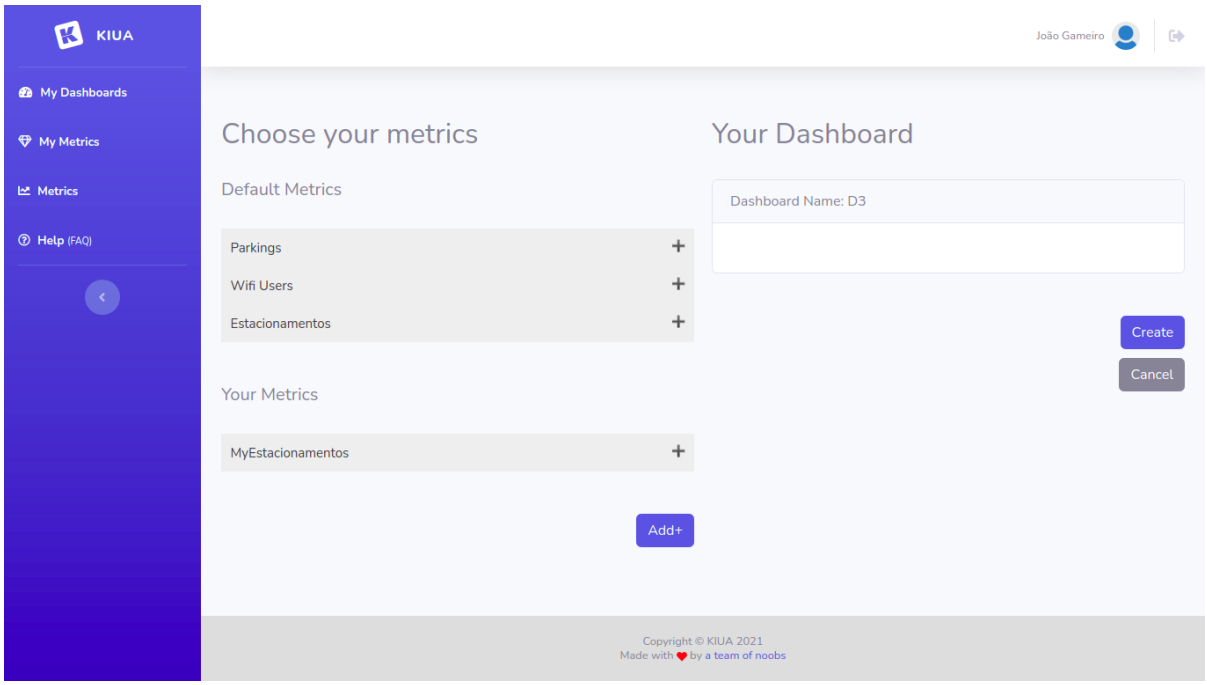

Figure 3.28: *Página para criar Dashboards*

A coluna das métricas está divida em métricas default e métricas do utilizador. Se o utilizador carregar numa métrica vai surgir um dropdown (Figura 3.29) com todos os indicadores associados a essa métrica. Para adicinonar um indicador à dashboard apenas é necessário escolher os indicadores pretendidos (selecionando as checkboxes) e carregar no botão *Add+* para tal efeito.

Todos os indicadores e métricas apresentados nesta página provêm da base de dados do backoffice (sendo que os indicadores são gerados automaticamente). Sempre que o utilizador abre esta página é feita uma verificação se existe uma métrica sem indicadores (pois se a query frequency de uma métrica for por exemplo 5 minutos, os indicadores da métrica só vão estar disponíveis 5 minutos depois da sua criação). Se por acaso existir uma métrica sem indicadores é feita uma chamada à função *get_ querys* para obter os querys e colocá-los na base de dados para que o utilizador os possa usar na sua dashboard. Se ainda não tiverem sido recolhidos dados para a métrica, a função *get_ querys* vai retornar uma lista vazia e não vai ser feita nenhuma inserção de indicadores na base de dados do Backoffice. Se uma métrica não tiver indicadores será apresentada a mensagem *Indicators not available yet, try again later.*

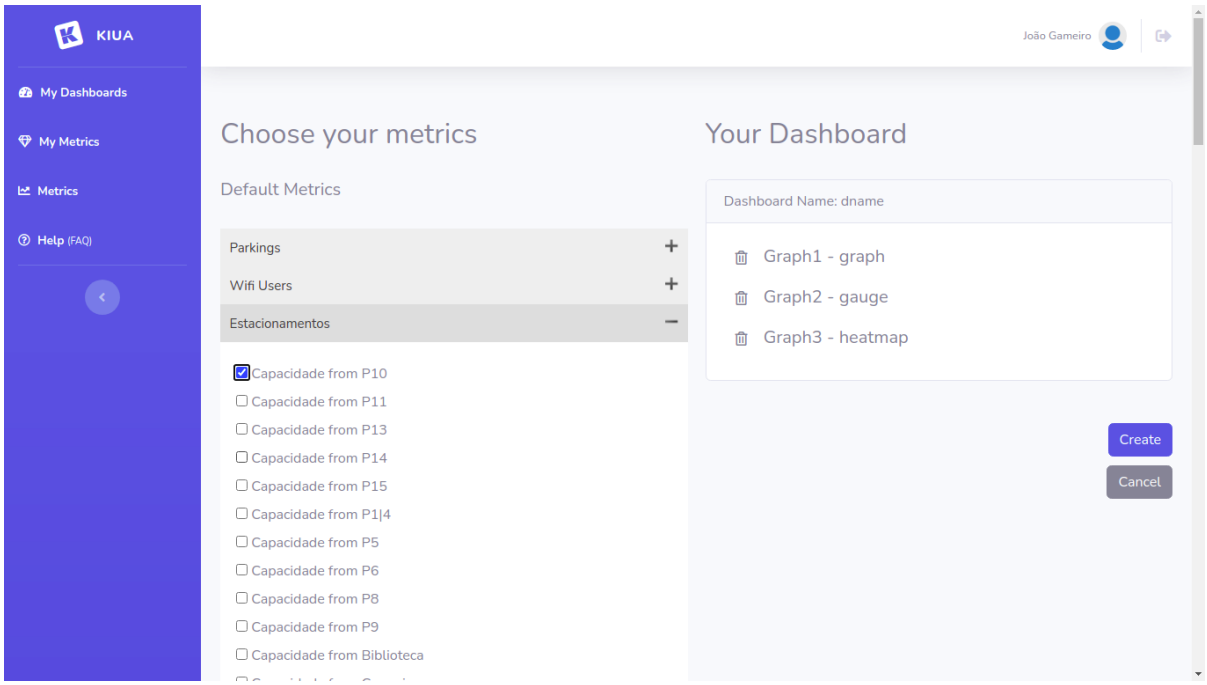

Figure 3.29: *Página para criar Dashboards com indicadores*

A ação de adicionar um gráfico a uma dashboard implica carregar no botão *Add+* que por sua vez vai abrir um pop-up para o utilizador escolher o formato de apresentação da informação selecionada e dar nome ao gráfico (Figura 3.30). A escolha do tipo de gráfico pode ser uma dentro das seis oferecidas, já referidas anteriormente na secção sobre geração automática de dashboards.

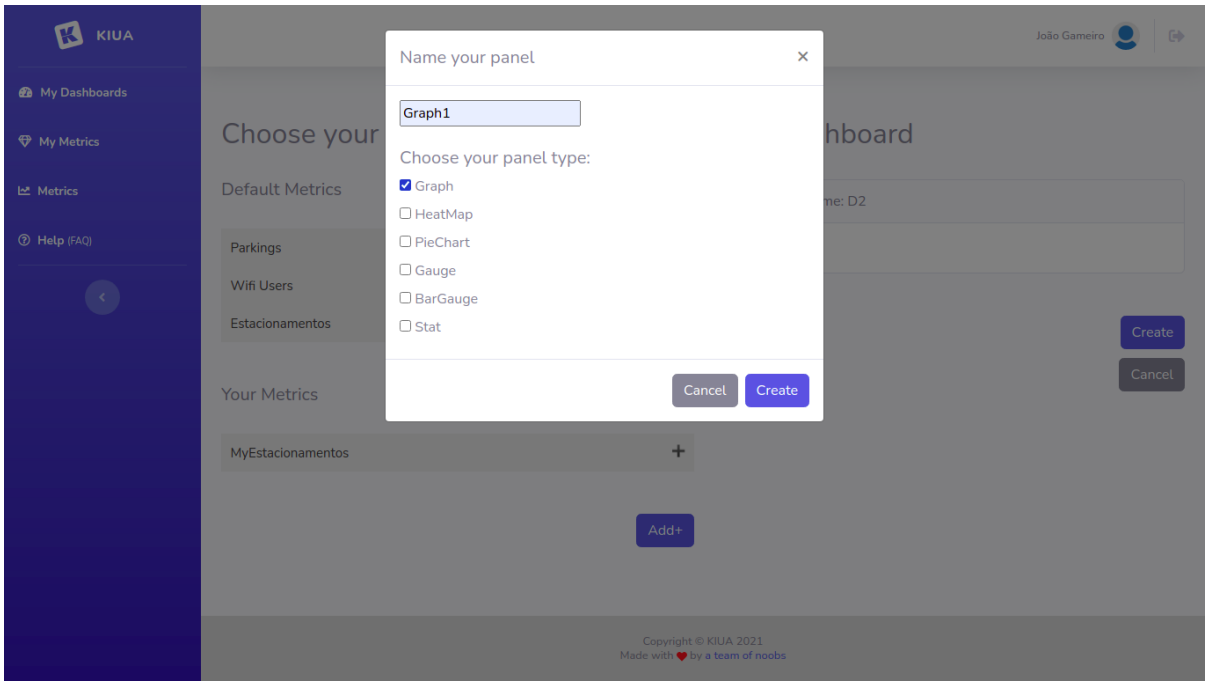

Figure 3.30: *Pop Up para adicionar uma métrica a uma dashboard*

Após a escolha de todos os gráficos a apresentar na dashboard é apenas necessário selecionar a opção *Create* (localizado na coluna do lado direito da página). Ao carregar nesta opção, é criado um objeto Dashboard da classe Dashboard, que gera um JSON automaticamente com toda a configuração especificada pelo utilizador. Este objeto é enviado para o servidor Grafana através da função *send_dash* e cria na pasta do utilizador uma dashboard associada, devolvendo em caso de sucesso o Unique Identifier (UID) da respectiva dashboard gerado pelo Grafana.

Este UID é posteriormente utilizado para adicionar à base base de dados do Backoffice a informação da dashboard criada para que o utilizador possa aceder e visualizar o que acabou de criar.

É também importante referir que a qualquer momento o utilizador pode cancelar todo este processo carregando no botão *Cancel*.

Página de visualização de uma Dashboard

Na página *MyDashboards* existem links para páginas onde é possível a visualização de dashboards.

Se o utilizador carregar num link é redirecionado para um URL no formato */showdashboard/<user-id> /<dashboard-nome>*. Os parâmetros do URL são utilizados para fazer um *query* à base de dados do backoffice para obter o UID da dashboard.

Através deste UID e com recurso também ao nome da dashboard é possível construir um URL no formato *http://<server-ip>:3000/d-solo/<uid>/<nome>?panelId=<panel-id>*. Neste url, *server-ip* representa o endereço IP do servidor aonde está instalado o grafana, *d-solo* indica que se pretende obter um painel em específico em vez da dashboard toda. Finalmente falta referir que é feito um ciclo for para iterar sobre todos os paineis pertencentes à dashboard

e substituir o campo *<panel-id>* pelo id do painel.

Após a construção dos URLs de todos os painéis é feito um ciclo for no código HTML para fazer a inserção em iframes (Figura 3.31).

```
{% for link in urls %}
<iframe src="{{ link }}" width="750px" height="320px" frameborder="0"></iframe>
{% endfor %}
```

Figure 3.31: *iframes que contêm os painéis de uma Dashboards*

O resultado da renderização é visível na Figura 3.32 onde está representada uma dashboard com três gráficos.

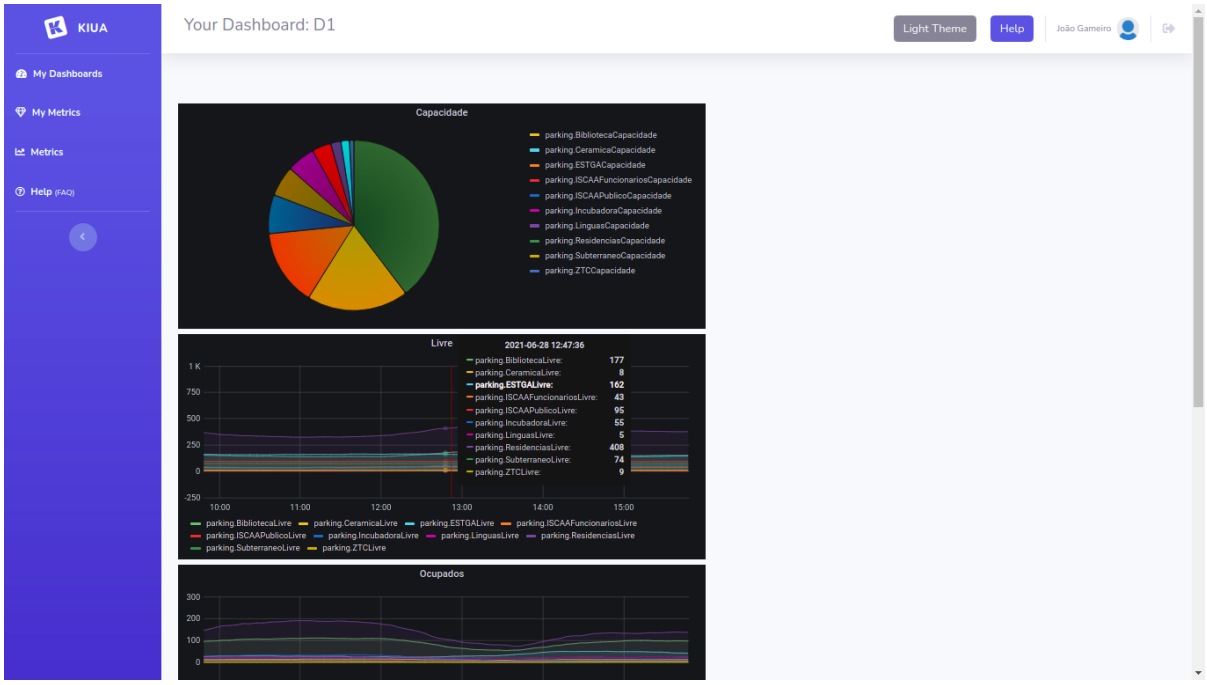

Figure 3.32: *Página de visualização de uma dashboard*

Página MyMetrics

Na página *MyMetrics* (Figura 3.33) é possível observar as métricas criadas pelo utilizador.

Com um design similar à página *MyDashboards* é apresentado numa tabela para cada métrica, o seu nome, data de criação, frequência de recolha dos dados associados à respectiva métrica e um botão para eliminar. Nesta tabela a única interação possível é a de eliminar a métrica, que envolve vários passos:

- Eliminação da métrica criada da base de dados de suporte à API através da chamada ao endpoint /Remove/<MetricType>/<MetricId> em que estes dois últimos parâmetros são respetivamento o tipo da métrica e o seu Id.
- Eliminação de todos os dados recolhidos associados a esta métrica da InfluxDB (feito

através da criação de um cliente *InfluxDBClient* e da eliminação da tabela associada à métrica).

- Finalmente, eliminação da métrica da base de dados de suporte ao Backoffice.

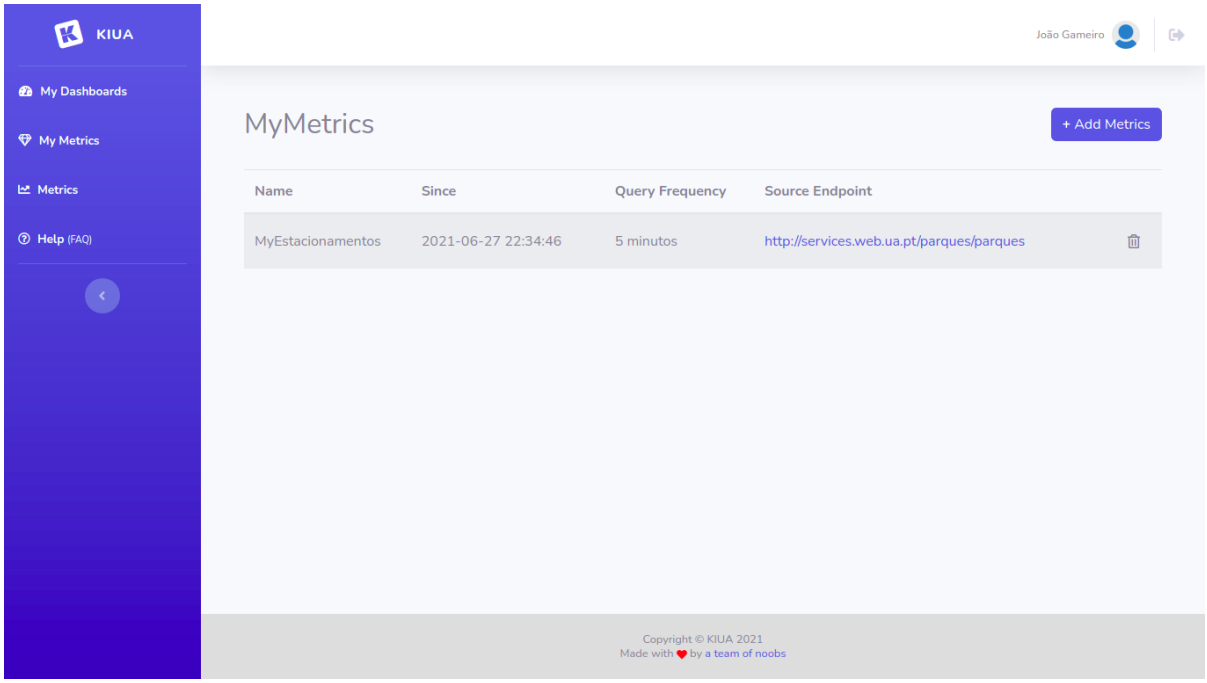

Figure 3.33: *Página MyMetrics*

Esta página possui também um botão *+ Add Metrics* que permite a criação de novas métricas. Após selecionar este botão, é aberto um pop-up (Figura 3.34) que contém um formulário para a criação de uma nova métrica onde é especificado:

- Nome da métrica
- Frequência de recolha de dados (pode ser 5 em 5 minutos, 30 em 30 minutos, hora a hora ou diariamente)
- endpoint para recolha dos dado
- Tipo de autenticação nesse endpoint (Public, Key Based, Bearer Token Based Authentication ou Http authentication)
- Caso o utilizador saiba o conteúdo dos dados que vão ser recebidos e queira filtrar, apenas é necessário indicar os campos que quer receber no parâmetro fields (este campo pode ir vazio e por default e assim são recolhidos todos os campos)

Após carregar no botão create é feita uma chamada à API, ao endpoint */Add/<MetricType>*, através de um HTTP Request em que vai especificado no cabeçalho toda a informação recolhida, resultante do preenchimento deste formulário. Se for devolvido um código HTTP 200 ou HTTP 201 (significando sucesso) é indicado ao utilizador o sucesso da operação de criação.

Caso tenha ocorrido algum erro no processo é apresentado também ao utilizador a respetiva mensagem de erro.

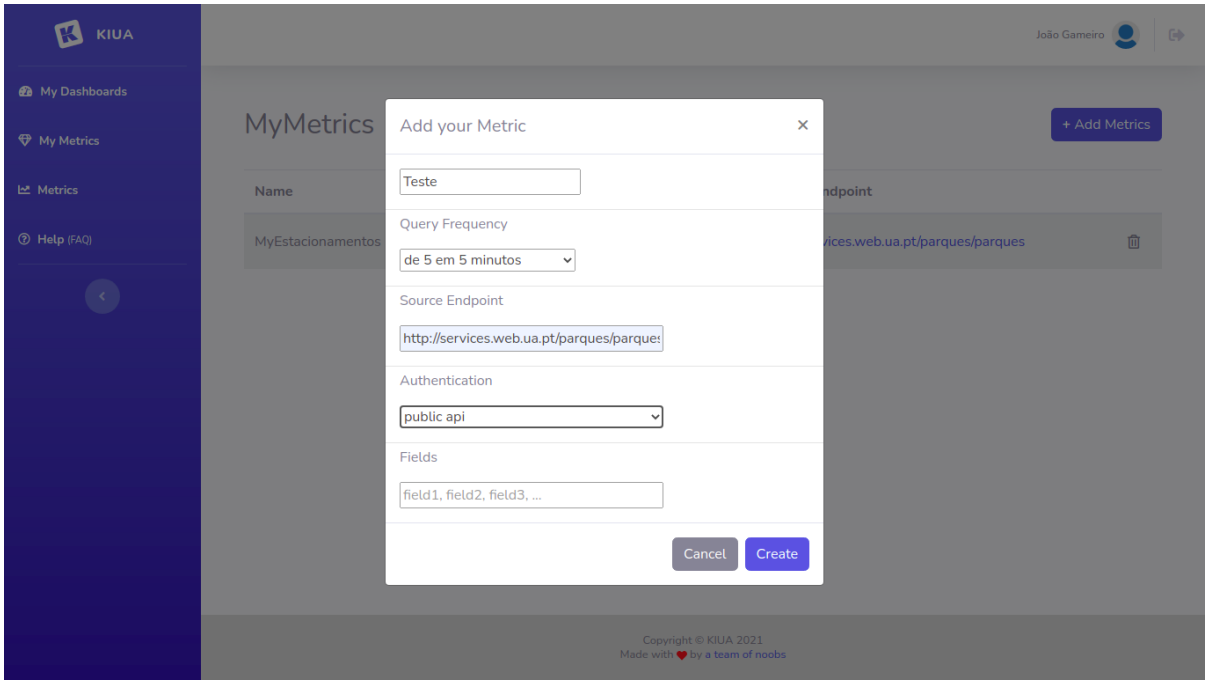

Figure 3.34: *PopUp para criação de métricas*

Página Metrics

Na página *Metrics* (Figura 3.35) é possível visualizar as métricas default do website assim como a sua descrição. As métricas apresentadas neste website são geridas pelos administradores que no momento de criação especificam uma pequena descrição associada à métrica para além de todos os campos obrigatórios a preencher.

Foi tomada uma decisão de projeto que deveriam existir quatro métricas default, para além das que o administrador gere. Ou seja, no momento em que o primeiro utilizador de todos faz login, é feita uma chamada a uma função *loadmetrics()* que por sua vez vai carregar na base de dados do Backoffice quatro métricas (métrica *Parkings*, métrica *Wifi Users*, métrica *DHCP Pool* e métrica *Node Storage*) e os seus indicadores associados (gerados com a função *get_ querys*), que podem ser utilizadas para gerar dashboards. Esta decisão deveu-se ao facto de se pretender que um utilizador possa utilizar o site sem especificar os seus endpoints e sem estar dependente de se um administrador criou ou não métricas default para ser usadas.

Na Figura 3.35 é possível verificar a existência das quatro métricas default do website e de mais uma que foi criada por um administrador (métrica *WheaterLisbon*). Sempre que um administrador criar uma métrica default ela vai aparecer nesta página com a respetiva descrição e vai estar disponível para todos os utilizadores usarem nas suas dashboards.

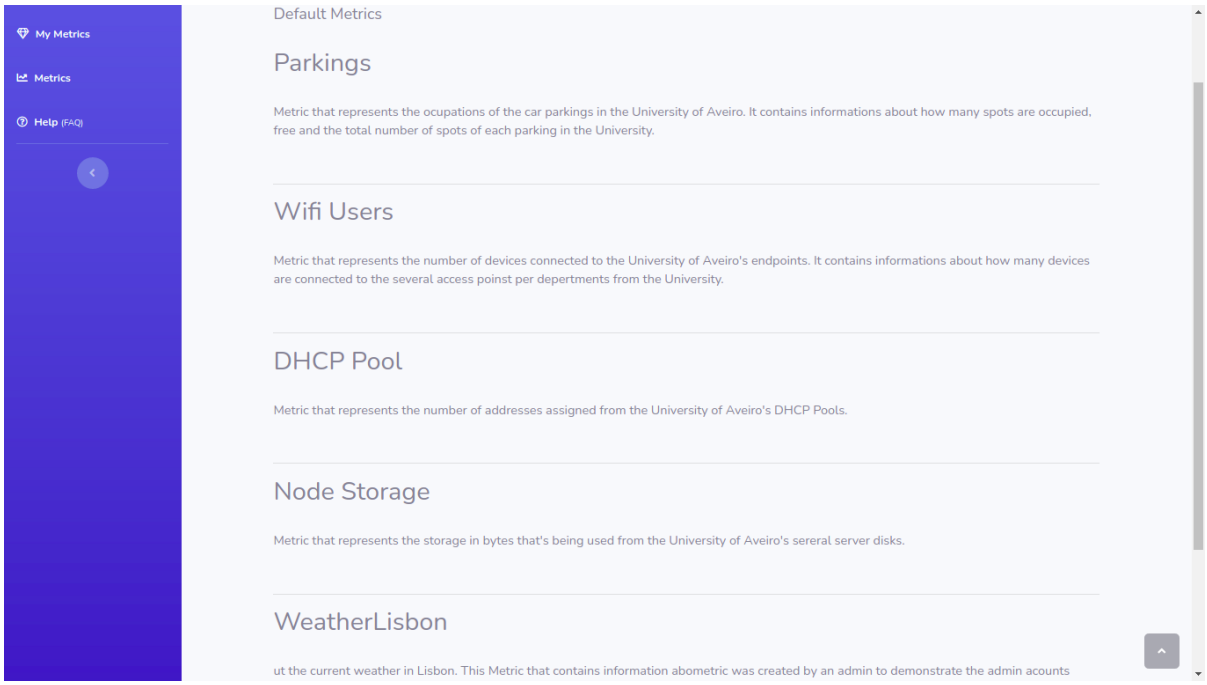

Figure 3.35: *Página Metrics*

Vista de Administrador

A vista de Administrador é em tudo igual à de utilizador normal. Tem acesso a todas as funcionalidades desde a criação de métricas privadas, até à criação de dashboards e alteração da sua visibilidade.

A única diferença entre as duas contas reside no facto de na *sidebar* de navegação existir uma opção a mais para os administradores (opção *Default Metrics* visível na Figura 3.36).

Através desta página os administradores fazem a gestão das métricas default, ou seja, todas as que criarem através desta página vão estar disponíveis para todos os utilizadores utilizarem nas suas dashboards. Os administradores são os únicos que podem apagar ou criar novas métricas default e o processo de criação é precisamente igual ao de criação de métricas normais.

Apenas existe uma diferença no processo todo (Figura 3.37), no formulário para criação de métricas default, existe um campo a mais que é a descrição. Este campo tem de ser preenchido obrigatoriamente e serve fundamentalmente para que, quando a métrica for criada, aparecer na página *Metrics* a sua respetiva descrição para que os outros utilizadores saibam o que representa.

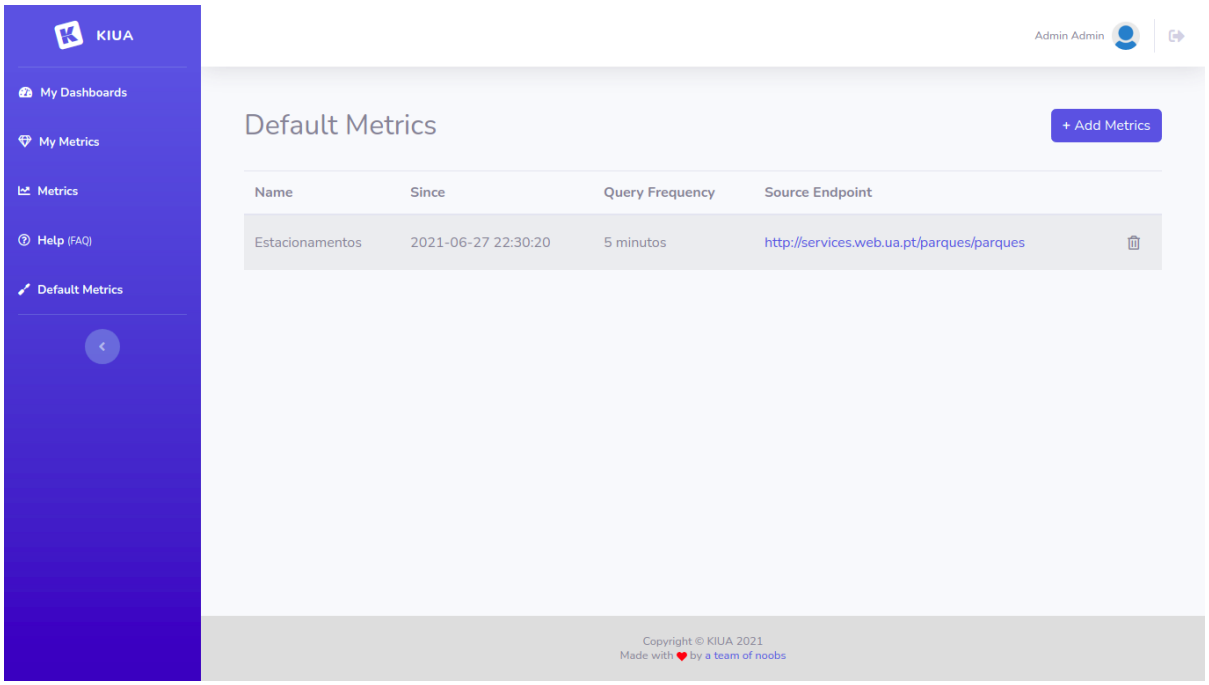

Figure 3.36: *Admin - Página DefaultMetrics*

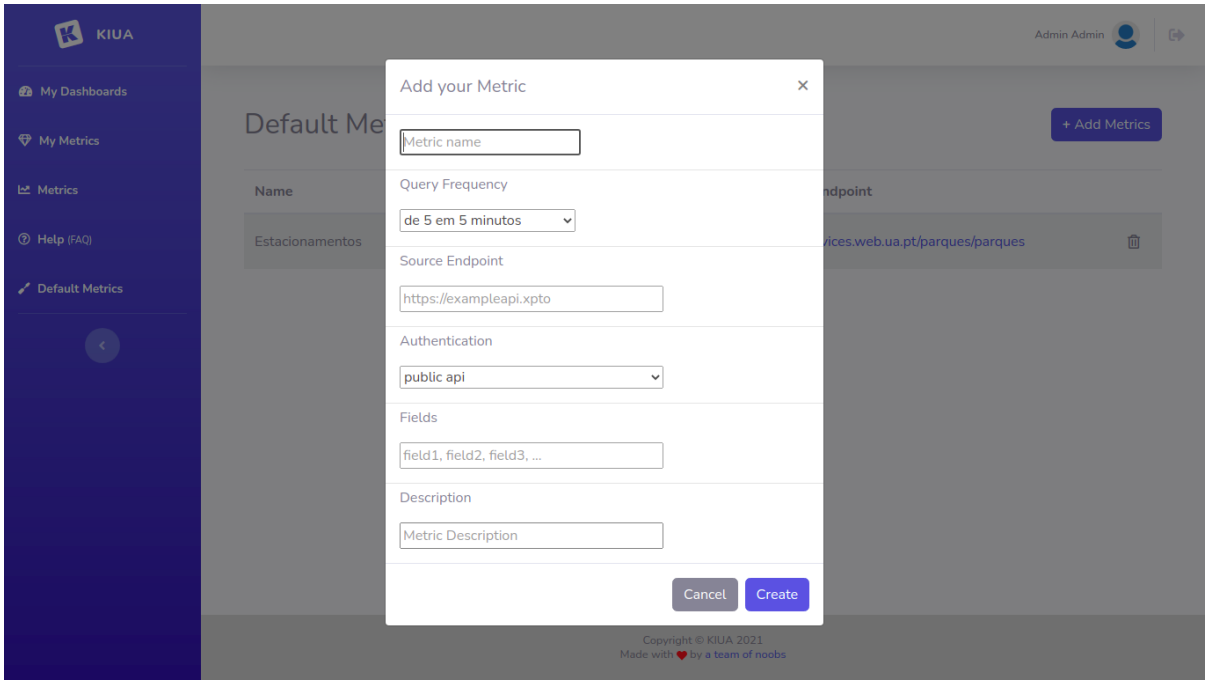

Figure 3.37: *Admin - Adição de uma métrica*

Capítulo 4

Resultados

4.1 Resultados

Como o título do projeto indica, o principal objetivo era dar a oportunidade a um utilizador de recolher KPI's e gerar dashboards a partir das mesmas. No nosso caso, estas KPIs estariam relacionadas com a Universidade de Aveiro e seriam disponibilizadas através de *endpoints*.

Mas como outro dos objetivos era a definição rápida e parametrizada de métricas, foi adicionada a funcionalidade de permitir a possibilidade do utilizador disponibilizar o seu próprio endpoint para que futuramente possa gerar dashboards utilizando os dados recolhidos.

Outra componente do projeto foi a página de consumo em tempo real. Neste website qualquer utilizador pode observar as métricas recolhidas em tempo real (número de estacionamentos livres/ocupados em cada parque, o número de dispositivos ligados aos pontos de acesso em cada departamento, entre outras).

Em termos de resultados é importante analisar o principal objetivo da plataforma web Backoffice. Através da definição de métricas e da geração de dashboards a partir dos dados recolhidos é possível efectuar a monitorização de serviços. A análise dos dados apresentados nas dashboards permite inferir conclusões sobre a performance do sistema e consequentemente a tomada de decisões administrativas que a afectem positivamente. Ou seja, o Backoffice acaba por ser uma solução para qualquer utilizador que queira monitorizar os seus serviços de forma simples, e com a vantagem de que oferece tudo isto sem obrigar o utilizador a saber SQL ou Grafana.

4.2 Objectivos Alcançados

No geral os objetivos propostos inicialmente para o projeto foram atingidos:

- Desenvolvimento de métodos para a recolha de métricas provenientes dos vários endpoints fornecidos.
- Construção de uma arquitetura base de suporte aos dados recolhidos (Base de dados InfluxDB, Message Broker).
- Criação de uma página web para apresentação de dados em tempo-real.

- Desenvolvimento de uma API que permite a gestão e recolha de dados a partir de endpoints fornecidos.
- Plataforma web Backoffice que permite a definição de métricas, assim como a geração automática de dashboards a partir dos dados disponíveis.
- Deployment de todos os serviços na Cloud para estarem disponíveis ao público.

O sistema desenvolvido apresenta em tempo real informação sobre as seguintes métricas:

- Número de lugares livres e ocupados dos parques de estacionamento da UA.
- Número de dispositivos ligados aos vários pontos de acesso da UA.
- Utilização em bytes dos vários discos virtuais da UA.
- Número de endereços atribuídos das várias pools de Dynamic Host Configuration Protocol (DHCP) da UA.

4.3 Objectivos Não Alcançados

Não chegou a ser possível efetuar a integração da autenticação da UA com o Backoffice, devido à falta de tempo e à entrega tardia da informação necessária para o fazer.

O número de métricas recolhidas relativamente aos serviços da Universidade de Aveiro acabou ficar abaixo do que se esperava inicialmente, no entanto é importante referir que tendo em conta a forma como foram fornecidos os dados, esta tarefa é uma tarefa repetitiva. Ou seja, o verdadeiro desafio era a construção de um conjunto inicial de métricas e todo este processo é um processo trivial que assenta no fundo no estudo dos dados recebidos e em como eles podem ser transformados em métricas.

4.4 Desafios

Os principais objetivos e momentos de maior dificuldade no decorrer deste projeto foram:

- O deployment dos websites na cloud assim como a inicialização das máquinas virtuais foram dos maiores desafios encontrados, resolvido com reuniões em grupo, para ter um maior número de pessoas a ajudar e a procurar erros.
- Integração da API desenvolvida com os restantes componentes do projeto.
- Aprendizagem de novas tecnologias como React, NodeJS, Kafka, InfluxDB e Grafana, que implicou uma certa elasticidade no que diz respeito à àrea de trabalho de certos membros do grupo.

4.5 Limitações

Uma das maiores limitações ao projeto foi a dificuldade em arranjar APIs dos vários tipos para conseguirmos testar todos os tipo de métricas e a sua consequente recolha de dados.

Capítulo 5

Conclusão

5.1 Conclusão Geral

No geral todos os objetivos inicialmente propostos para o projeto foram desenvolvidos, sendo possível concluir que esta foi uma experiência bastante enriquecedora. Desenvolvemos competências extremamente importantes na área de trabalho e organização em equipa, assim como a capacidade para analisar e resolver problemas sozinhos. O estudo de novas tecnologias por sozinho também foi uma importante competência que foi bastante desenvolvida.

Para uma melhor compreensão de todo o processo de desenvolvimento do sistema é recomendável o acesso ao micro-site do projeto aonde está presente um registo semanal e um diagrama de Gantt que apresenta uma divisão do trabalho e tarefas desenvolvidas. [11]

5.2 Trabalho futuro

Apesar das funcionalidades base de aplicação web Backoffice terem sido desenvolvidas com sucesso, é importante referir que a usabilidade do website precisa de melhorias. Por isso, propõe-se uma lista de tarefas para conseguir melhorar esta interface o melhor possível de modo a que torne toda a experiência de usar o Backoffice ainda mais intuitiva e agradável ao utilizador:

- Tanto a criação de métricas como a criação de dashboards implicam um quantidade bastante elevada de dados de entrada por parte do utilizador. Deste modo, uma das tarefas a fazer no futuro é o estudo de como todo este processo pode ser agilizado de modo a envolver menos *clicks* e menos preenchimento de formulários.
- Após a criação de dashboards e de métricas, implementar a possibilidade de alterar certos campos das mesmas, tal como o nome ou a query frequency (pois de momento apenas é possível a criação e eliminação das mesmas e não a sua alteração).
- Finalmente, fazer testes de usabilidade com potenciais utilizadores alvo e recolher mais informação para melhorar toda a interface.

No que diz respeito à recolha de KPIs, ao longo do desenvolvimento do projeto chegámos à conclusão de que a geração automática de indicadores muitas vezes pode não fornecer ao

utilizador o que ele pretende ou mesmo os indicadores gerados não serem adequados à métrica disponibilizada.

Por isso, seria interessante para trabalho futuro o desenvolvimento e estudo de algoritmos que permitissem uma geração de querys avançada (recebendo até se possível inputs do utilizador).

Em relação ao website de visualização de dados em tempo real, que acabou por ter menos métricas que o inicialmente planeado, era bastante interessante tornar a página num verdadeiro ponto de monitorização da Universidade de Aveiro, com a apresentação de dados dos mais variados tipos de serviços da Universidade. Ou seja, acrescentar mais métricas e pensar em novas formas criativas de as apresentar ao utilizador.

Bibliografia

[1] Cluvio. https://www.cluvio.com/.

[2] IndicadoresUA. Universidade de aveiro. https://indicadores.ua.pt/.

[3] Todd Palino Neha Narkhede, Gwen Shapira. *Kafka The Defenitive Guide*. O'Reilly Media, 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2017.

[4] KIUA. Repositórios do projeto no github. https://github.com/KIUA-PEI, 2021.

[5] Socket.IO. Socket.io website - introduction. https://socket.io/docs/v4/index.html#What-Socket-IO-is.

[6] Joseph Chege. How to use cors in node.js with express. https://www.section.io/engineering-education/how-to-use-cors-in-nodejs-with-express/, 2021.

[7] Azure Microsoft. App-service. https://azure.microsoft.com/pt-pt/services/app-service/#product-overview.

[8] David Arthur Dana Powers. Kafkaconsumer - usage. https://kafka-python.readthedocs.io/en/master/usage.html, 2016.

[9] docs.influxdata. Python client library. https://docs.influxdata.com/influxdb/cloud/tools/client-libraries/python/.

[10] FLASK. Flask documentation. https://flask.palletsprojects.com/en/2.0.x/, 2010.

[11] KIUA. Kiua microsite. https://kiua-pei.github.io/KIUA_microsite/, 2021.