UNIVERSIDADE D
COIMBRA

Flávio Amaral Fernandes

Desenvolvimento e implantação das aplicações e plataforma NanoSen-AQM

Relatório de estágio no âmbito do Mestrado em Engenharia Informática, com especialização em Engenharia de Software orientado pelo Professor Doutor Filipe João Boavida Mendonça Machado Araújo e pela Professora Doutora Catarina Helena Branco Simões da Silva e apresentado ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Setembro de 2021

Esta página foi intencionalmente deixada em branco.

Resumo

Com o desenvolvimento da tecnologia, da medicina e o aumento da poluição a nível mundial, foi começando a aparecer cada vez mais o interesse pela medição da qualidade do ar. Por esse motivo foram desenvolvidas várias plataformas que nos mostram informação sobre a qualidade do ar em vários locais do mundo, no entanto, a maior parte das vezes os sensores que registam estas medições são demasiado caros para cobrir uma área geográfica considerável o que faz com que zonas mais remotas não tenham acesso a essas medições. Este projeto tem como base o desenvolvimento de uma plataforma de monitorização da qualidade do ar, mas ao invés de usar sensores demasiadamente caros, utilizada sensores de baixo custo de forma a que seja possível cobrir a maior área possível com custos mais reduzidos.
A plataforma inicial apresentou algumas falhas que serão abordadas neste documento e cabe agora ao estagiário integrado neste projeto, implementar novas funcionalidades na plataforma e fazer a manutenção e melhorias nos seus serviços para que esta esteja nas condições ideais para ser disponibilizada ao público. Neste documento é apresentado um estudo sobre as várias tecnologias que foram analisadas como potenciais soluções para os problemas identificados e, depois de escolhidas aquelas que acrescentavam mais ao projeto, foi feito um planeamento tendo em conta a implementação das soluções apresentadas garantindo sempre que os requisitos, a arquitetura e os atributos de qualidade sejam cumpridos.
A plataforma NanoSen-AQM faz parte de um projeto conjunto com outras entidades e neste momento a mesma existe em versão web e foram também disponibilizadas versões para dispositivos móveis.

Palavras-Chave

Desenvolvimento, *Web*, Multi-plataforma, Qualidade do Ar, *iOS*, *Android*

Esta página foi intencionalmente deixada em branco.

Abstract

With the advancement of technology and medicine, and the increase in pollution worldwide, interest in measuring air quality began to appear more. For this reason, several platforms have been developed to show us air quality information in various locations around the world. However, most of the time, these sensors are too expensive to cover a large area, which means that more remote areas do not have access to these measurements. This project is based on the development of an air quality monitoring platform. However, instead of using overly expensive sensors, low-cost sensors are used so that it is possible to cover the largest possible area with the lowest costs.
The initial platform exhibited several shortcomings that are addressed in this work, and it is now up to the intern integrated into this project to maintain and improve the platform's services so that it is in the ideal conditions to be made available to the public. This document presents a study on the various technologies that were analyzed as potential solutions to the identified problems and, after choosing those that added more to the project, a plan was made taking into account the implementation of the solutions presented, always ensuring that the requirements, architecture and quality attributes are met.
The NanoSen-AQM platform is part of a joint project with other entities and at the moment it exists in a web version and versions for mobile devices were also made available.

Keywords

Development, Web, Multiplataform, Air Quality, *iOS*, *Android*

Esta página foi intencionalmente deixada em branco.

Agradecimentos

Gostaria de agradecer ao Professor Filipe Araújo e à Professora Catarina Silva por todos os conselhos e por toda a ajuda que me deram ao longo do estágio. Gostaria também de agradecer a todos os envolvidos no projeto NanoSen-AQM em especial à Professora Bernardete Ribeiro e ao Professor Alberto Cardoso pelas sugestões e ajuda que iam dando nas reuniões semanais do projeto, e também ao Professor Pedro Salgueiro da Universidade de Évora pela disponibilidade demonstrada. Um agradecimento especial ao Pedro Lucas, antigo estagiário deste projeto, pela enorme ajuda que me deu numa fase inicial e pela disponibilidade que mostrou para discutir alguns problemas se eu necessitasse.

Por último, agradecer à minha família, aos meus colegas de curso e aos meus companheiros de casa por todo o apoio e amizade que me deram ao longo dos últimos tempos.

Este trabalho foi realizado no âmbito do projeto NanoSen-AQM, que foi financiado pelo Programa Interreg-Sudoe da União Europeia ao abrigo da convenção de subvenção nº SOE2/P1/E0569.

Acrónimos

API *Application Programming Interface*. 18

AQI *Air Quality Index*. 2

CAQI *Common Air Quality Index*. x, 4, 5

CD Continuous Deployment. 11, 16, 20, 21

CI Continuous Integration. 11, 16, 20, 21

CISUC Centro de Informática e de Sistemas da Universidade de Coimbra. 1

CSV *Comma-separated values*. 28

DEI Departamento de Engenharia Informática. 22

DEI-UC Departamento de Engenharia Informática da Universidade de Coimbra. 21, 25

EAQI *European Air Quality Index*. x, 4, 5

FCTUC Faculdade de Ciências e Tecnologia da Universidade de Coimbra. 1

MR Merge Request. 11, 16

SO Sistemas Operativos. 7, 12, 18, 28

UEV Universidade de Évora. 17, 25

Conteúdo

Lista de Figuras

Lista de Tabelas

Esta página foi intencionalmente deixada em branco.

Capítulo 1

Introdução

Este documento representa o projeto associado à Tese do Mestrado em Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra (FCTUC) no ano letivo 2020/2021. Este projeto foi realizado no Centro de Informática e de Sistemas da Universidade de Coimbra (CISUC).

1.1 Contexto

Apesar dos vários esforços que têm sido feitos ao longo dos anos, as concentrações de poluentes atmosféricos permanecem demasiado elevadas [1] e os problemas que daí resultam são sérios e continuam a existir [2]. A poluição causada pelos vários poluentes (O_3, NO_2, etc.) continua a colocar em causa a saúde da população mundial originado, alguns deles, doenças oncológicas.

A par das doenças causadas, a poluição resultante de fábricas, veículos a combustão, entre outros, também o meio ambiente é bastante afetado por todas estas complicações.

De forma a analisar a qualidade do ar, o interesse na sua monitorização tem vindo a crescer para alertar a população dos riscos aos quais a sua saúde está exposta. Várias cidades têm os seus próprios sistemas de análise do ar mas estão dependentes de sensores de elevado custo que limitam a quantidade de sensores que se colocam em diferentes espaços[4].

A poluição do ar é cada vez mais um fator com uma enorme influência na saúde dos seres vivos e do planeta como um todo. Plataformas como a do projeto NanoSen-AQM [5] ajudam, de uma forma muito direta, as pessoas a observar que a poluição é real e que afeta claramente a qualidade do ar. Sendo importante cada vez mais mostrar realmente estes valores, é necessário ocupar a maior área possível com sensores para o efeito, o que não é de todo fácil pois estes equipamentos são muito dispendiosos. Assim sendo, o projeto NanoSen-AQM [6], procura fornecer o mesmo serviço que outras plataformas de monitorização da qualidade do ar mas utilizando sensores de baixo custo e que sejam de igual forma confiáveis.

1.2 Motivação

A utilização de sensores de baixo custo permitirá que se aumente a área de monitorização da qualidade do ar e, desta forma, com o mesmo ou até menor investimento

conseguiria-se uma recolha mais alargada deste tipo de informações. Ainda assim, estes sensores registam as medições com menor grau de fiabilidade em comparação com sensores mais dispendiosos.

A ideia será reunir a maior quantidade de medições possível visto que maior parte dos países tem as suas medições isoladas dos outros, o que implica ter de aceder a diferentes *websites* para conseguir visualizar toda essa informação. Nos dias de hoje, os utilizadores esperam encontrar informação sem ter de sair do mesmo sistema.

A plataforma NanoSen-AQM [5] é, atualmente, uma página *web* onde é possível verificar os índices da qualidade do ar em alguns países da Europa, nomeadamente, Portugal, Espanha e França. Na plataforma é possível ver o *Air Quality Index* (AQI) de vários *clusters* e, selecionando um deles, pode observar-se a evolução de determinados componentes químicos associados a esse *cluster*, ao longo do tempo. Ainda dentro das informações de cada *cluster* podem ver-se previsões meteorológicas e, caso este seja móvel, é possível ver as localizações onde foram realizadas as medições dos vários sensores a ele associados.

Este projeto é apoiado pelo programa *Interreg Sudoe* [3] que é financiado por fundos da União Europeia. Os objetivos do projeto estão exemplificados na Figura 1.1, sendo as medições captadas por nano-sensores desenvolvidos no âmbito de projeto que serão posteriormente ajustadas e calibradas para que se apresentem medições mais fiáveis.

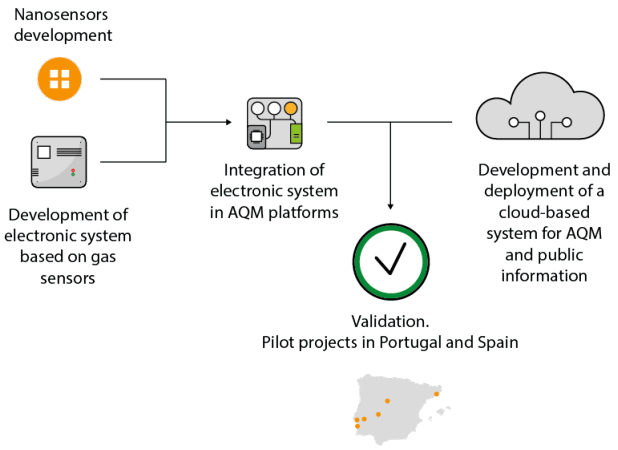

Figura 1.1: Fases do projeto NanoSen-AQM

1.3 Objetivos

O principal objetivo deste estágio é automatizar o processo de *deployment* da plataforma NanoSen-AQM [5], fazendo com que o que separe a implementação do código e o seu *deploy* seja o mais próximo possível de apenas um clique ou um comando num terminal. Em muitos projetos de desenvolvimento de software existe uma quantidade significativa de passos e caminhos a seguir entre a produção de código e o *deploy* desse mesmo artefacto.

Visto que a plataforma já está num estado avançado no que toca a desenvolvimento de funcionalidades, é necessário agora automatizar todo o processo de novas adições ou alterações ao que está feito assim como garantir que o serviço funciona e está sempre disponível quando o mesmo for preciso. Para além de todas as estas adições, é pedido

ao estagiário que implemente novas funcionalidades caso estas sejam requisitadas pelos parceiros do projeto. Assim sendo, ao longo do estágio irão ser utilizadas tecnologias que permitam reduzir o tempo, diminuir as configurações (instalação de *libraries*, instalação das tecnologias para correr os vários serviços, etc...) necessárias para novos elementos e fazer com que o processo de *deploy* seja tão simples quanto colocar uma nova funcionalidade num repositório e passado uns minutos vê-la na página web. Para além de automatizar todo este processo, é expectável que ao longo deste estágio seja disponibilizada uma aplicação para dispositivos móveis.

Assim, neste trabalho, pretende-se que o estagiário seja responsável por procurar possíveis tecnologias, definir os requisitos para o desenvolvimento do projeto e implementar as soluções propostas. No fim deste período é expectável que o projeto esteja a funcionar e disponível para todos os utilizadores.

1.4 Estrutura do Documento

Este deste documento encontra-se dividido nos seguintes capítulos:

- O **Capítulo 2** é dedicado aos conhecimentos prévios e ao estado da arte. Começa por dar uma breve introdução ao tema de monitorização da qualidade do ar, sendo depois abordadas várias tecnologias e *frameworks* que poderão ser relevantes para o desenvolvimento deste projeto. Por fim, existe uma reflexão acerca daqueles dos mais compatíveis com o que se pretende.
- No **Capítulo 3** é apresentada a proposta de desenvolvimento para o estágio.
- No **Capítulo 4** é apresentado o planeamento seguido no primeiro e segundo semestres. São apresentadas as ferramentas de apoio ao desenvolvimento do projeto e é feita uma análise aos possíveis riscos que podem aparecer ao longo do tempo.
- O **Capítulo 5** contém a descrição de todas as funcionalidades desenvolvidas ao longo do estágio.
- O **Capítulo 6** apresenta uma reflexão sobre o trabalho realizado ao longo do estágio, abordando também dificuldades encontradas ao longo do tempo.

Capítulo 2

Fundamentos e Estado da Arte

Neste capítulo será apresentado um estudo acerca dos temas e tecnologias relacionadas com o projeto. Ao longo do capítulo irão ser abordados alguns conceitos sobre a monitorização da qualidade do ar e, posteriormente, será feita uma abordagem às tecnologias a serem utilizadas.

2.1 Monitorização da qualidade do ar

Relativamente à monitorização da qualidade do ar na plataforma NanoSen-AQM [5] existem conceitos que interessa serem abordados:

- **Poluição atmosférica:** Qualquer forma de matéria que possa tornar o ar impróprio ou nocivo para a saúde.
- **Sensor:** Dispositivo responsável por medir os valores de um determinado parâmetro (CO_2, NO_2, etc...) ao longo do tempo.
- ***Cluster*:** Conjunto de sensores num determinado espaço. Este mecanismo agrega vários sensores que podem ou não fazer medições do mesmo componente químico.
- **AQI** (ou IQA em português): Significa *Air Quality Index* (*Índice da Qualidade do Ar*). É o índice padrão para medir o nível de poluição atmosférica. Conjunto de seis níveis de medição distinguidos por cores que simbolizam o quão poluída está uma determinada zona [7].

Os valores AQI variam de país para país e nos países europeus usam-se sobretudo dois desses índices, o *Common Air Quality Index* (CAQI) e o *European Air Quality Index* (EAQI).

Para o CAQI existe uma escala de 0 a 100 em que os valores mais próximos de zero significam uma melhor qualidade do ar e, pelo contrário, quando mais próximo de cem esse valor estiver, pior é a qualidade do ar nesse local. Para calcular esse índice, é necessário registar valores dos diferentes componentes químicos hora a hora ou então a cada 24 horas e o valor que será registado irá ser o pior que for medido. Na figura 2.1 pode ver-se como funciona a tabela que nos indica esses valores.

Index class	Grid	Traffic						City Background							
		core pollutants			pollutants			core pollutants				pollutants			
		NO2	PM10		PM2.5		CO	NO2	PM10		O3	PM2.5		CO	SO_2
			1-h.	24-h.	1-h.	24-h.			1-h.	24-h.		1-h.	24-h.		
Very low	0	0	0	0	*0*	*0*	0	0	0	0	0	*0*	*0*	0	0
	25	50	25	*15*	*15*	*10*	5000	50	25	*15*	60	*15*	*10*	5000	50
Low	25	50	25	*15*	*15*	*10*	5000	50	26	*15*	60	*15*	*10*	5000	50
	50	100	50	*30*	*30*	*20*	7500	100	50	*30*	120	*30*	*20*	7500	100
Medium	50	100	50	*30*	*30*	*20*	7500	100	50	*30*	120	*30*	*20*	7500	100
	75	200	90	50	*55*	*30*	10000	200	90	50	180	*55*	*30*	10000	*350*
High	75	200	90	50	*55*	*30*	10000	200	90	50	180	*55*	*30*	10000	*350*
	100	400	180	100	*110*	*60*	20000	400	180	100	240	*110*	*60*	20000	500
Very High*	> 100	> 400	>180	>100	*> 110*	*>60*	>20000	> 400	>180	>100	>240	*> 110*	*>60*	>20000	>500

NO_2, O_3, SO_2: hourly value / maximum hourly value in µg/m^3
CO 8 hours moving average / maximum 8 hours moving average in µg/m^3
PM_{10} hourly value / daily value in µg/m^3

* An index value above 100 is not calculated but reported as " > 100"

Figura 2.1: Tabela de cálculo do CAQI [11]

Para a medição do EAQI temos uma escala de cores acompanhada por uma escala de 1 a 5 onde, mais uma vez, os valores mais baixos significam uma boa qualidade do ar e os mais elevados uma qualidade do ar pior. Para registar este índice, observam-se os valores das várias partículas e componentes químicos e regista-se aquele que for pior. na tabela abaixo pode-se verificar como se aplica o índice por este método.

Pollutant	Index level (based on pollutant concentrations in µg/m3)					
	Good	Fair	Moderate	Poor	Very poor	Extremely poor
Particles less than 2.5 µm ($PM_{2.5}$)	0-10	10-20	20-25	25-50	50-75	75-800
Particles less than 10 µm (PM_{10})	0-20	20-40	40-50	50-100	100-150	150-1200
Nitrogen dioxide (NO_2)	0-40	40-90	90-120	120-230	230-340	340-1000
Ozone (O_3)	0-50	50-100	100-130	130-240	240-380	380-800
Sulphur dioxide (SO_2)	0-100	100-200	200-350	350-500	500-750	750-1250

Figura 2.2: Tabela de valores do EAQI [12]

2.2 Aplicações híbridas

As aplicações híbridas são *apps* que têm a mesma base de código, mas que funcionam em diferentes dispositivos independentemente do seu sistema operativo.

Esta capacidade é conseguida através do uso de tecnologias *Web*, como HTML e JavaScript, e são desenvolvidas através do recurso a *frameworks* específicas. O *Apache Cordova* tem um papel fundamental nesta integração pois atua como a plataforma que oferece APIs JavaScript para desenvolvimento mobile, fazendo passar-se por um *website* onde se usam tecnologias *web* para estruturar a interface gráfica [21].

2.3 *Frameworks* de desenvolvimento

Nesta secção irão ser abordadas várias *frameworks* utilizadas para o desenvolvimento da plataforma, tanto no *backend* como no *frontend*.

2.3.1 Django

Django é uma *framework* de código aberto, escrita em *Python* destinada a desenvolvimento Web [13]. Uma *framework* é um conjunto de bibliotecas utilizadas para simplificar e acelerar o desenvolvimento deste tipo de aplicações. O *Django* fornece várias funções já pré-definidas que tornam o processo de desenvolvimento muito simples, fácil de implementar e de compreender.

Esta *framework* usa o padrão *Model, Template, View* em que o *Model* é onde acontece toda a interação com a base de dados, os *Templates* são a renderização dos dados obtidos numa forma mais intuitiva para o utilizar e as *Views* é a lógica sobre os dados, isto é, onde estes são processados e onde se fazem operações sobre os mesmos.

Numa pesquisa pela internet [14] foram encontrados alguns prós e contras que estão colocados na Tabela 2.3.

Django Prós e Contras	
Prós	Contras
Desenvolvimento rápido e simples	Não aconselhável para projetos pequenos
Framework muito flexível	
Suporte para APIs com a REST framework	

Tabela 2.1: Prós e contras da *framework Django*

Reflexão

Esta *framework* é uma opção bastante viável para o desenvolvimento de do *backend* da aplicação. Para além de ser uma solução bastante madura e com boas referências no desenvolvimento de aplicações, é uma *framework* escrita em *Python* que é também uma linguagem simples de aprender. Analisando os prós e contras que oferece, esta solução enquadra-se bastante bem naquilo que é esperado para este estágio.

2.3.2 Ionic

Ionic [17] é uma *framework open-source* para desenvolvimento de aplicações híbridas. Uma aplicação híbrida é um produto cujo código é igual para as várias plataformas de desenvolvimento fornecendo dessa forma uma manutenção simples e comum a todas as plataformas.

O ionic é baseado em tecnologias já utilizadas em desenvolvimento web, tal como o HTML5, CSS e JavaScript, no entanto é a sua base em Apache Cordova que lhe dá todo

o benefício de compatibilidade em todos os dispositivos *android* e *iOS*. Para além destes dois Sistemas Operativos (SO) é suportado em todos que tenham compatibilidade com o *Cordova* (*Windows Phone*, *Blackberry*, etc). Graças a esta compatibilidade a *framework* está a crescer ao longo do tempo e a tornar-se uma alternativa importante neste tipo de desenvolvimento. Na seguinte tabela é possível verificar as suas vantagens e desvantagens.

Ionic Prós e Contras	
Prós	Contras
Boa documentação	Tamanho da aplicação elevado
Desenvolvimento Web	Falta de suporte em versões mais antigas
Vários componentes de UI pré-definidos	Baixa performance em alta renderização

Tabela 2.2: Prós e contras da *framework Ionic*

Reflexão

Uma boa solução para o que se pretende desenvolver na plataforma NanoSen-AQM [5]. Oferece soluções híbridas o que facilita o desenvolvimento tanto para aplicações *web* como móveis. visto que já tem alguns elementos visuais previamente criados facilita, de certa forma, o desenvolvimento do *frontend* da plataforma.

2.3.3 Angular

Angular é uma *framework* [9] de desenvolvimento web para construir aplicações *single-page* e utiliza HTML e TypeScript como base para a programação. Esta *framework* é uma versão melhorada do AngularJS também desenvolvida pela mesma equipa. Esta nova versão assenta na base dos componentes, em que cada um deles é uma classe que contém os dados da aplicação e a sua lógica, associado com um ficheiro HTML.

Através desta integração e através dos módulos, é possível alterar com Angular, uma página HTML antes desta ser renderizada. Alguns prós e contras [10] podem ser analisados na seguinte tabela:

Angular Prós e Contras	
Prós	Contras
Ligação de dados bi-direcional	*Performance*
Boa comunidade de programadores	Curva de aprendizagem alta

Tabela 2.3: Prós e contras da *framework Angular*

Reflexão

Apesar de oferecer uma curva de aprendizagem alta o estagiário já tem experiência com *frameworks* semelhantes o que facilita a aprendizagem desta em questão. Esta *framework* pode ser facilmente integrada com *Ionic* sendo este um grande ponto a favor.

2.4 Mecanismos de monitorização de plataformas

Nesta secção irão ser abordadas tecnologias com potencial para serem utilizadas na monitorização do estado da plataforma. Foi feito um estudo sobre todas elas e uma análise sobre quais se enquadravam melhor no contexto do estágio e da plataforma.

2.4.1 Grafana

Grafana [16] é uma aplicação *open-source* que nos permite observar e monitorizar o estado de plataformas, analisando vários fatores e organizando-os de forma muito intuitiva. Esta aplicação permite verificar que erros os utilizadores encontraram enquanto utilizavam a nossa plataforma, quais os países em que foram encontrados de forma mais frequente, em período temporal é que os mesmos ocorreram, etc.

É também possível a Grafana com vários *plugins* que nos fornecem outras análises que podem ser relevantes para quem está a usar. Esta aplicação permite também prever a probabilidade de algum erro ou quebra no sistema, acontecer novamente.

Os utilizadores podem criar painéis ao seu critério com as mais variadas análises e informações. É de uso livre e gratuito.

Figura 2.3: Exemplo de uma *dashboard* do Grafana [16]

2.4.2 Nagios

Nagios [15] é um software para monitorização de servidores e redes. Permite monitorizar todas as partes de uma infraestrutura, como aplicações, serviços, redes, etc. É possível integrar este *software* com *plugins* por ele fornecidos ou por terceiros.
O Nagios possui uma interface web com painéis de visualização muito semelhantes ao Grafana. Verifica constantemente a disponibilidade do serviço em questão e permite alertar os desenvolvedores por e-mail ou telemóvel em caso de alguma falha ocorrer. Este *software* fornece também a possibilidade de obter relatórios sobre a disponibilidade do sistema e configurar ações a serem executadas quando alguma falha for encontrada.

2.4.3 Comparação

As duas opções fornecem uma solução muito parecida mostrando as avaliações ao sistema em forma de gráficos sobre vários serviços de uma plataforma. Ainda assim, a *Grafana* têm mais apoio da comunidade visto ser uma aplicação *open-source* e é uma plataforma gratuita ao contrário do *Nagios*.

2.5 Ferramentas de auxílio ao desenvolvimento

Nesta secção irão ser abordadas eventuais tecnologias a serem utilizadas para a automatização do processo de desenvolvimento e *deploy* da plataforma. Foi feito um estudo sobre todas elas e uma análise sobre quais é que se enquadravam melhor no contexto do estágio e da plataforma.

2.5.1 Docker

Docker é uma ferramenta *open-source* que permite ter um implementação e execução de aplicações através do uso de *containers*. Um *container* permite aos programadores reunir todas as dependências necessárias e agrupá-las num único pacote para que se tenha a certeza que a aplicação que está a ser desenvolvida poderá ser executada em qualquer máquina, diferente daquela que foi usava para a desenvolver, independentemente da configuração que a mesma possa ter [23].

2.5.2 Docker Swarm

O Docker Swarm é uma ferramenta de osquestração de *containers*. Esta tecnologia permite distribuir *containers* por diferentes máquinas e fazer a sua reativação caso algum falhe inesperadamente.

A sua configuração requer a existência de, pelo menos, uma máquina *manager* que é a responsável pelas operações de distribuição de *containers*. Depois da configuração desta máquina, outras denominadas por *workers* podem ser adicionadas ao grupo. O *manager* também pode funcionar como um *worker* e executar *containers*.

Uma das grandes vantagens desta ferramenta é que os serviços podem ser reconfigurados, por exemplo, alterar o número de réplicas, sem ser necessário para e reiniciar o serviço. A inicialização dos serviços pode ser configurada através de um ficheiro que descreva os serviços a criar com as respetivas imagens *docker* associadas. Em relação a desvantagens é uma tecnologia que nunca foi muito explorada no mercado o que faz com que não exista tanto suporte em caso de complicações no processo [29].

2.5.3 Kubernetes

Kubernetes é uma plataforma open-source aberta pela Google. Está assente sobre um sistema de containers e com a premissa de ajudar na automação e gerenciamento de serviços e/ou aplicações.

Antes de chegarmos ao desenvolvimento por contentores, passamos por duas fases nesse processo. A primeira de desenvolvimento tradicional baseava-se na execução de

várias aplicações assentes na mesma máquina física e mesmo sistema operativo. Ora, isto levantava problemas de sobrecarga pois não havia controlo da memória que cada uma delas poderia alocar.

O próximo passo, na busca da solução para este problema, foi passar para um sistema de virtualização, isto é, auxiliado por máquinas virtual, que apesar de utilizarem o mesmo hardware, conseguiam isolar cada aplicação num sistema operativo diferente. Desta forma, era então mais fácil controlar a memória que cada aplicação usaria para além de que fornecia uma maior segurança, visto que agora cada aplicação estava isolada impossibilitando de uma outra aceder às suas informações.

Ainda assim, este método não era suficientemente eficaz, pois replicava sistemas operativos que em muitos casos eram iguais de aplicação para aplicação. Posto isto, apareceram os containers que trabalham na camada acima do sistema operativo, mas ainda assim, conseguem ter o seu próprio sistema de arquivos, memória, etc, tal como uma máquina virtual. Este sistema de contentores permite controlar a carga de trabalho que cada aplicação tem, sendo possível adicionar ou remover containers de acordo com a necessidade de cada serviço. Num sistema destes tem-se ainda a vantagem de conseguir ter um ambiente consistente, seja em desenvolvimento local ou em produção na *cloud* [19] [20].

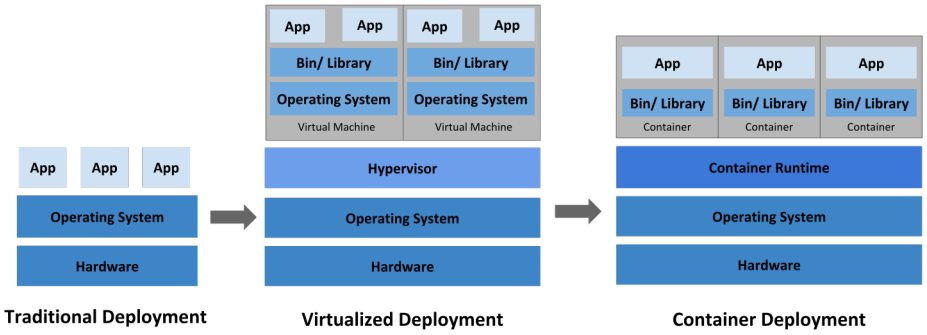

Figura 2.4: Fases de desenvolvimentos ao longo do tempo [19]

2.5.4 Jenkins

Jenkins é uma ferramenta de Countinuous Integration / Continuos Delivery (CI/CD) que nos permite verificar se um *commit* não quebra a integridade do código. Com o Jenkins e usando este mecanismo fica fácil perceber qual foi o pedaço de código que provocou o erro. Quando todos os testes ao novo *commit* forem positivos a pipeline avança para a próxima fase, que à partida será fazer o *build* do código mais recente. Posto isto, podem ser feitos testes automáticos à plataforma ou aplicação utilizando frameworks já existentes para esse propósito.

O Jenkins permite também, elaborar resultados aos testes feitos, elaborando relatórios de erro, informar o administrador sempre que seja detetado algum erro em toda a pipeline, entre outros. De forma a cobrir o maior números de testes possível, é possível fazê-los para os mais variados inputs em diferentes pontos da aplicação.

2.5.5 Gitlab CI/CD

A *Continuous Integration (CI)* é a integração de código desenvolvido por vários membros de uma equipa, para um mesmo repositório. Cada membro dessa equipa faz um *Merge Request (MR)* e são corridos testes e validações ao código a cada novo MR.

Depois de todos os testes serem validados, pode existir uma nova fase designada de *Continuous Deployment (CD)* que através de *pipelines* implementadas pelos programadores, consegue colocar o novo código, com as novas alterações, em produção.

A junção destas duas definições traz mais eficiência e rapidez em todo o processo de *deploy* código.

O GitLab como repositório de código, fornece este auxílio aos desenvolvedores, sendo possível configurar os ficheiros que contêm as *pipelines* ou então utilizar ficheiros já configurados para algumas tecnologias e *frameworks*. O desenvolvedor apenas tem de se preocupar com a configuração destes ficheiros e dos testes que quer correr e tudo o resto é tratado pela plataforma. Desta forma, a qualquer novo MR todos os testes contidos no ficheiro que foi configurado são automaticamente executados e o código só é aceite caso passe corretamente em todos eles.

2.5.6 Considerações finais

Para o *deploy* da plataforma e a para a manutenção dos serviços da mesma, optou-se por usar o GitLab CI/CD, tanto por oferecer um repositório de código *online* como também por oferecer mecanismos de integração com a metodologias de CI/CD o que torna o processo de *commit* mais simples.

Inicialmente pensou-se em utilizar *Kubernetes* para fazer a gestão dos *containers* dos serviços presentes na plataforma, no entanto, através de conversas com outras pessoas com experiência nessa tecnologia e também com outros parceiros do projeto, chegou-se à conclusão de que seria algo que ia consumir muito tempo pois era necessário fazer as configurações das máquinas do projeto para suportarem a utilização de *Kubernetes*. Caso o objetivo fosse usar máquinas na *cloud* esta solução era extremamente aliciante pois existem várias empresas a oferecer este tipo de serviços *web*. No entanto, os parceiros envolvidos no projeto NanoSen-AQM têm as suas próprias máquinas e não seria de todo prático estar a migrar todos os serviços para outra plataforma na *cloud*.

Assim sendo, foi escolhido o *Docker Swarm* para fazer a gestão dos *containers* por ser uma solução que se enquadra melhor no tipo de sistema que já estava desenvolvido e é mais amigável na sua configuração. Uma das vantagens de usar esta tecnologia é o facto de já existirem alguns serviços de outros parceiros a usar o *docker-compose* e os ficheiros de configuração podem ser reutilizá-los alterando algumas das configuração descritas nesses documentos.

A tecnologia *Jenkins* não foi escolhida para o desenvolvimento deste projeto pois a solução que é por esta oferecida também está disponível no GitLab que reúne outras ferramentas mais interessantes para o decorrer deste estágio.

Capítulo 3

Abordagem Proposta

Neste capítulo irão ser propostas as alterações e adições que poderão ser implementadas na plataforma e de que forma serão desenvolvidas. Inicialmente será apresentada uma visão geral do que é proposto e, de seguida, serão apresentadas as soluções para o problema.

3.1 Proposta de trabalho

O objetivo do trabalho a ser realizado é minimizar as preocupações que um novo desenvolvedor irá ter quando chegar ao projeto e pretenda adicionar novas funcionalidades à plataforma.

O ponto principal é fazer com que todo o processo de *deploy* seja feito em apenas uma iteração, seja ela um clique ou uma linha no terminal. Aliada a esta facilidade no envio de código para produção, será interessante adicionar mecanismos de *containers* ao processo de desenvolvimento para que se consiga obter um funcionamento uniforme do código e dos serviços seja qual for a máquina onde este estará a correr. Esta última abordagem irá oferecer, por consequência, uma maior facilidade de configuração do ambiente de desenvolvimento sempre que um novo desenvolvedor se juntar à equipa e pretenda ter a plataforma a correr no seu dispositivo.

Uma outra vertente é garantir a disponibilidade da plataforma. Sendo que está previsto fornecer a plataforma a outras entidades, é crítico que os serviços não falhem e que sejam consistentes. Assim sendo, desenvolver esta solução em *docker swarm* é um caminho bastante viável pois conseguimos garantir as premissas atrás identificadas e apresentar desse modo um sistema mais robusto.

Aproveitando o potencial da plataforma até então desenvolvida, faz sentido com que esta sirva de apoio a entidades que produzam ou possuem sensores e sintam necessidade de ter uma página que analise e demonstre os dados obtidos pelos mesmos. Deste modo, creio que seria uma mais valia fazer com que esses grupos tivessem a possibilidade de enviar os dados para esta plataforma e usufruir de todas as funcionalidades que ela tem. Assim, resolvíamos um problema com que estas entidades se deparam e por outros, fazíamos com que a plataforma fosse utilizada com a finalidade pretendida.

Por último e de forma a fornecer um serviço portátil, existe a necessidade de fornecer soluções móveis para que os dados possam ser vistos a qualquer momento. Assim sendo, é uma prioridade desenvolver aplicações móveis para os dois SO mais utilizados, iOS e

Android.

3.2 Requisitos do Sistema

3.2.1 Requisitos Funcionais

Nesta secção serão desenvolvidos os requisitos necessários para o cumprimento das propostas apresentadas na secção anterior. Os requisitos servem como uma forma de expor melhor de que forma a solução terá de ser abordada e quais os critérios para o seu desenvolvimento.

Os requisitos estão dispostos por ordem de prioridade, sendo o REQ01 o mais prioritário.

REQ01 - Serviços distribuídos por *containers*	
Nível	Clam
Ator	Programador
Objetivo	Ter um *container* para cada serviço
Pré-Condições	• Arquitetura bem definida • Docker
Pós-Condições	• Cada serviço corre num *container* independente dos outros
Cenário Principal	1. O ator executa o ficheiro de configuração *docker* e os serviços iniciam automaticamente
Cenário Alternativo	

REQ02 - Ter os *containers* dos serviços da plataforma a correr num *Docker Swarm*	
Nível	Cloud
Ator	Programador
Objetivo	Conseguir ter maior disponibilidade no sistema
Pré-Condições	• *Containers* dos serviços • Máquinas configuradas com o *Swarm*
Pós-Condições	• O *manager* do *swarm* gere todos os serviços nas diferentes máquinas
Cenário Principal	1. Um novo serviço é adicionado ao *swarm* 2. O *manager* aloca espaço numa das máquinas disponíveis 3. O serviço é criado e fica disponível
Cenário Alternativo	3a. O serviço falha inesperadamente 3a.1. O *manager* cria um novo serviço idêntico ao que falhou

REQ03 - Automatizar os testes para o *Backend* da aplicação	
Nível	Clam
Ator	Programador
Objetivo	O código da aplicação deve ser testado automaticamente
Pré-Condições	• Adição de novas funcionalidades
Pós-Condições	• Colocar os testes no repositório para que os *commits* sejam testados
Cenário Principal	1. O programador desenvolve código 2. Os testes são executados no código desenvolvido e passam com sucesso
Cenário Alternativo	2a. O testes executaram com erros 2a.1. O programador tem de corrigir os erros e repetir o processo

REQ04 - Testar e validar o código a cada *commit*	
Nível	Clam
Ator	Programador
Objetivo	Uma nova funcionalidade deve ser automaticamente testada e colocada em produção
Pré-Condições	• Ter ficheiros de testes criados • Ter um repositório no GitLab
Pós-Condições	• Todos os novos *commits* devem ser testados e apenas aceites caso passem nos teste • O novo código deve ser compilado e colocado em produção
Cenário Principal	1. O programador implementa uma funcionalidade 2. Cria um *commit* com o código adicionado 3. Os testes são executados com sucesso e o código aceite 4. A nova funcionalidade é colocada em produção
Cenário Alternativo	3a. O código continha erros e foi recusado 3a.1. O programador tem de corrigir os erros e repetir o processo

REQ05 - Disponibilizar a aplicação para dispositivos móveis	
Nível	Cloud
Ator	Programador
Objetivo	Produzir as aplicações cliente
Pré-Condições	• Plataforma web desenvolvida • Plataforma com código híbrido
Pós-Condições	• As aplicação são disponibilizadas aos clientes
Cenário Principal	1. O ator usa o código da plataforma web para gerar as aplicações móveis 2. A aplicação é disponibilizada na respetiva loja de aplicações
Cenário Alternativo	1a. O código totalmente compatível com o sistema operativo do dispositivo móvel 1a.1. O ator tem de resolver os conflitos e programar a funcionalidade com código compatível 2a. A aplicação não é aceite nas lojas *online*

REQ06 - Monitorizar o estado da plataforma	
Nível	Cloud
Ator	Cliente
Objetivo	Analisar a *performance* dos serviços prestados
Pré-Condições	• Plataforma *online*
Pós-Condições	• Uma página *web* com gráficos e/ou tabelas com os dados de *performance* da plataforma
Cenário Principal	1. O ator acede à plataforma 2. O ator analisa a informação que os dados lhe transmitem
Cenário Alternativo	2a. Os gráficos não contêm nenhuns dados

3.2.2 Atributos de qualidade

Nesta secção serão abordados os atributos de qualidade também conhecidos por requisitos não-funcionais. Cada atributo terá uma breve descrição e o seu impacto na arquitetura da plataforma (H - *High*/Elevado, M - Médio, L - *low*/Baixo).

- **Disponibilidade (H):** Se uma falha que impede o bom funcionamento do sistema ocorrer, esta deve ser resolver até 24 horas depois da deteção do erro.

  - Como avaliar: Se durante o período do estágio ocorrer alguma falha no sistema que seja crítica para o funcionamento da plataforma, a mesma deve ser corrigida do espaço temporal definido.

- **Portabilidade (M):** A plataforma deve ser compatível com as plataformas móveis *Android* e *iOS*. Deve também suportada pelos principais *browsers*.

  - Como avaliar: Quando finalizada, a plataforma deverá estar disponível para os sistemas operativos móveis acima identificados e deverá ser possível ser utilizada nos *browsers* atualmente disponíveis.

- **Segurança e Autenticação (M):** O sistema não deve permitir que utilizadores sem autenticação acedam a informação que apenas é permitido aceder quando se está autenticado.

  - *Como avaliar:* Testar com uma sessão sem utilizador registado se é possível aceder a informação só disponível para utilizadores registados.

- **Usabilidade (L):** A plataforma deve fornecer ao utilizador as funcionalidades principais em menos de 4 cliques.

  - *Como avaliar:* Testar as principais funcionalidades da plataforma manualmente e garantir que são alcançáveis em menos de 4 cliques.

3.3 Soluções identificadas

Nesta secção irão ser abordadas as soluções escolhidas para desenvolver as funcionalidades anteriormente propostas.

3.3.1 Gitlab CI/CD

Visto que a automação do *deploy* depende desta fase, este será o primeiro tópico a ser executado. Para tal, é necessário utilizar um repositório, que será o *GitLab*, pois o mesmo já oferece a integração com as técnicas de CI/CD tornando o processo mais simples e rápido. Esta funcionalidade requer dois passos fundamentais que serão apresentados nas seções seguinte.

Ficheiro de Configuração

Este ficheiro irá conter as *pipelines* que irão ser executadas para um novo MR ser aceite no repositório. Para este projeto, terá que ser dividido em duas fases distintas: a de testes e a de *deploy*. Na primeira, serão executados testes, previamente definidos, ao código de toda a aplicação. Só no caso de todos esses testes serem executados com sucesso é que a *pipeline* avança para a próxima fase. Como o próprio nome indica, na fase de *deploy*, compila-se todo o código para que este esteja pronto para ser colocado em produção. Caso algum erro ocorra nesta fase, toda a *pipeline* é anulada e o MR é recusado.

Definir Testes

Na secção anterior, falou-se de testar o código a cada MR, mas para isso é necessário que existam testes que possam ser atribuídos a esse código. Para tal, iremos dividir os testes em duas componentes: os testes para *Backend* e testes para *Frontend*.

Visto que o *Backend* está desenvolvido em *Python* (*Django*), a *framework* utilizada oferece utilidades que permitem correr os testes definidos com apenas uma linha num terminal. Posto isto, apenas é necessário criar um ou mais ficheiros com código em *Python* que permita cobrir e testar todas as funcionalidades da aplicação.

Por fim, o *Frontend* não pode ser testado da mesma forma que o código anterior pois é uma parte mais interativa e que requer a utilização da plataforma por parte de um utilizador. Ainda assim, podem ser utilizadas *frameworks* que permitem criar *bots* que irão

"caminhar"pela plataforma atuando como um humano. Dessa forma, podem ser definidos testes que contemplem os caminhos que um utilizador humano poderá fazer para descobrir eventuais falhas a esse nível.

3.3.2 Docker Swarm

Em primeiro lugar é necessário fazer a criação dos *containers* com todos os serviços da aplicação. Neste momento, já existem alguns que estão criados, nomeadamente os serviços de base de dados, mas pode haver a possibilidade de serem feitas alterações e até criar novos caso seja necessário. Estes *containers* serão criados utilizando *Docker*. Tendo em conta que o *swarm* irá replicar as imagens criadas por outras máquinas que não o *manager*, é necessário que estas sejam públicas pois esta tecnologia não transfere as imagens entre as diferentes máquinas. Assim sendo, as imagens terão de ser criadas numa fase inicial para serem alojadas no *DockerHub* para depois no ficheiro de configuração do *swarm* a referência ser feita ao repositório onde está guardada. A cada nova alteração, a imagem tem de ser construída novamente.

Depois desta primeira etapa estar concluída é necessário fazer a configuração do *swarm*. Para tal, é necessário ter uma ou mais máquinas disponíveis em que uma delas será o *manager*, onde estará todo o código desenvolvido e onde todas as operações de manutenção serão realizadas. Depois de configurada esta máquina, será fornecido um *token* de acesso ao swarm. Com esse *token* teremos de, nas outras máquinas, conectar ao *manager* através dessa credencial e do endereço IP da máquina inicial. Essas máquinas serão denominadas de *workers* pois terão a função de executar o trabalho que o *manager* irá delegar.

Concluída a configuração das máquinas será necessário configurar um ficheiro que descreva todos os serviços *docker* a serem executados no *swarm* e, no *manager*, fazer *deploy* desses serviços para que sejam distribuídos pelas máquinas pertencentes a esse grupo.

Para a implementação deste sistema, será necessária a cooperação com a Universidade de Évora (UEV) pois é nos servidores dessa instituição que o *backend* da plataforma está a ser alojado e a manutenção desses serviços é realizada pelos mesmos.

3.3.3 *Dashboards* de monitorização

Visto que a plataforma poderá estar a servir várias entidades, é necessário garantir que esta não sofre qualquer tipo de falha que coloque o sistema numa situação crítica. Para tal, está a ser idealizada uma nova página com vários gráficos e informações sobre a análise aos diversos sistemas da plataforma.

Seria interessante adicionar esta funcionalidade visto que iremos ter os Kubernetes (Secção 2.5.3) implementados de forma a que o sistema seja reativado automaticamente em caso de falha e assim com esta *dashboard* conseguiríamos analisar até que ponto é que esses sistemas de recuperação estavam a ser suficientes para manter a plataforma a funcionar sem falhas.

Visto que existe uma restrição temporal limitada pela entrega final da tese de mestrado, esta funcionalidade só será implementada caso as propostas anteriormente especificadas sejam concluídas antes do tempo previsto.

3.3.4 Desenvolvimento das aplicações cliente

Sendo que a plataforma foi desenvolvida com intuito de estar disponível para todos os dispositivos e visto que já está disponível para *web*, falta agora colocar à disposição do público as respetivas aplicações móveis, tanto para *iOS* como para *Android*. Sendo assim, e visto que no desenvolvimento da aplicação foi usado *Ionic* 2.3.2, basta reutilizar esse código e apenas fazer pequenas alterações que sejam necessárias para obter a compatibilidade com os SO acima referidos.

As aplicações serão posteriormente disponibilizadas nas lojas nativas dos respetivos SO e estarão disponíveis para ser utilizadas por todos os utilizadores.

3.3.5 *Multi-tenancy*

Depois de garantir a automação de todo o processo de desenvolvimento e *deploy* da plataforma, irá ser desenvolvida uma nova funcionalidade que permitirá a entidades externas utilizar a plataforma que está a ser desenvolvida, para analisar os dados dos seus próprios sensores. A ideia chave é fornecer uma plataforma independente de qualquer entidade ou empresa, para que qualquer uma delas possa usufruir dos dados fornecidos pelos seus sensores sem terem de se preocupar com um desenvolvimento de uma aplicação para mostrar os seus valores.

O *multi-tenancy* a ser implementado na nossa plataforma irá permitir com que as entidades que a escolham, reduzam custos de desenvolvimento, pois não precisam de criar uma nova plataforma, não precisam de realizar trabalhos de manutenção e apenas se têm de preocupar em enviar os dados para a base de dados da plataforma.

Posto isto, o nosso trabalho será em primeiro lugar, realizar as alterações necessárias à base de dados, para que seja permitida a inserção de dados por outras entidades que não a atual, a *Application Programming Interface* (API) tem de ser alterada para que cada pedido transporte a informação sobre qual a entidade que está a fazer o pedido e por último, o *frontend* terá de sofrer alterações no que toca à comunicação com o *backend*.

3.4 Reflexão final

Ao longo deste capítulo foi proposta uma implementação a aplicar na plataforma no segundo semestre. As decisões foram feitas com base naquilo que a plataforma está a precisar para ter uma estrutura de desenvolvimento bem definida e amigável tanto para o programador como para os possíveis utilizadores. As decisões aqui tomadas foram abordadas com os outros membros do projeto para que pudessem, também por eles, serem validadas.

Capítulo 4

Planeamento

Neste capítulo serão abordados os planeamentos definidos para os dois semestres correspondentes ao período deste estágio. Será também feita uma apresentação das ferramentas utilizadas para algumas tarefas realizadas assim como uma análise dos riscos que podem ocorrer. Por fim, serão descritos dois momentos de divulgação da plataforma.

4.1 Trabalho realizado no primeiro semestre

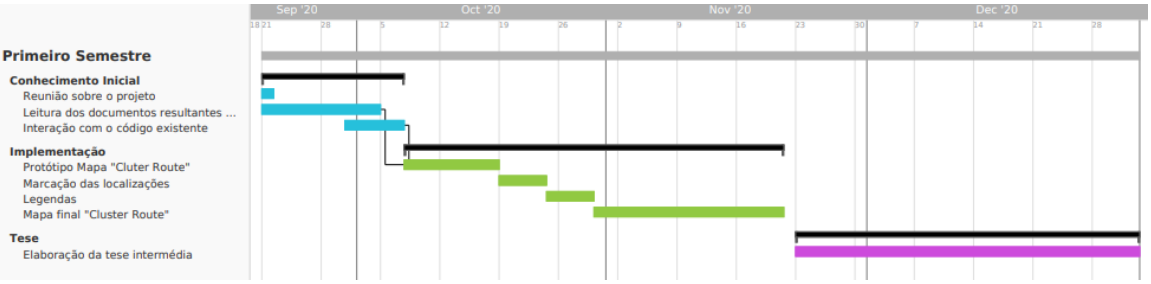

Figura 4.1: Diagrama de Gantt para o primeiro semestre

Para delinear o trabalho a ser realizado no primeiro semestre foi feito um diagrama de *Gantt* para auxiliar na decisão sobre que tarefas se iriam realizar e por qual ordem iriam ser executadas.

No início do semestre foi necessário fazer um estudo sobre todo o trabalho que tinha sido realizado por membros que estiveram no projeto em anos transatos, deste modo, comecei por ler os documentos resultantes do trabalho realizado por dois ex-alunos para posteriormente começar a investigar e a analisar o código existente para me integrar no projeto de forma definitiva.

Passada esta primeira fase de integração, passei a implementar funcionalidades para o frontend do projeto que eram pedidas por outros parceiros do NanoSen-AQM. Todas as alterações e adições que eram feitas à plataforma eram depois discutidas semanalmente com vários membros associados ao projeto, que validavam e/ou sugeriam alterações àquilo que era apresentado. A funcionalidade que foi pedida foi a de mostrar num mapa os pontos por onde um determinado cluster de sensores tinha passado, caso este fosse móvel. Surgiram várias alterações ao longo deste processo que como foi dito anteriormente eram sugeridas por vários membros ligados à plataforma e ao projeto.

Assim que foi terminado o mapa e validado por outros parceiros, restou o tempo para elaborar este documento.

4.2 Trabalho planeado para o segundo semestre

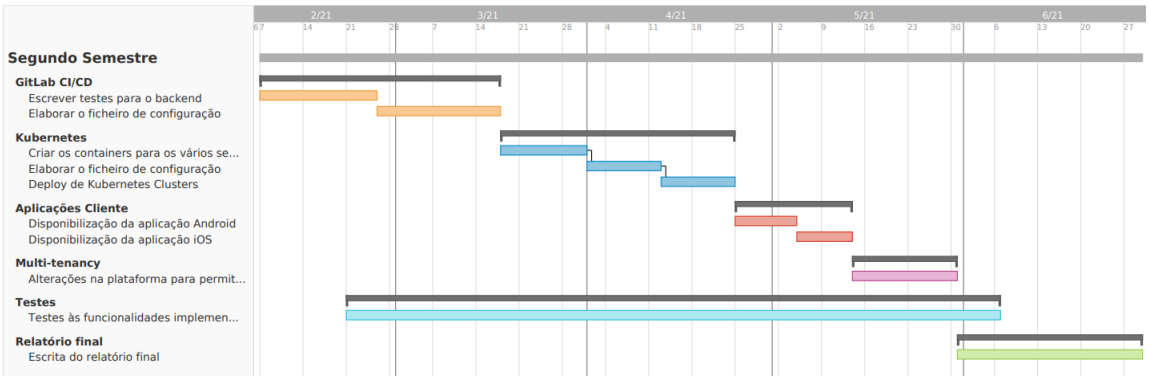

Figura 4.2: Diagrama de Gantt para o segundo semestre

Tendo em conta a abordagem proposta para o trabalho a realizar no segundo semestre e os requisitos previamente definidos é necessário perceber a ordem pela qual o trabalho irá ser realizado e quando as tarefas serão executadas.

Assim sendo, foi elaborado um diagrama de *Gantt* que nos dá uma ideia mais clara de como esse trabalho será feito. Tal como representado na figura 4.2, o primeiro foco será implementar o CI/CD pois irá afetar todo o desenvolvimento de código da plataforma. Depois disso, o trabalho passará por colocar todo o *backend* em *Kubernetes* o que implica criar *containers* para os atuais serviços utilizados neste projeto. Estando a parte de automatização da plataforma terminada, serão disponibilizadas duas aplicações móveis, uma para iOS e outra para Android.

Por fim, e caso se verifique que é necessário, será também desenvolvida uma *dashboard* de monitorização da *performance* da plataforma.

4.3 Ferramentas de Apoio

De forma a ter um processo de trabalho o mais organizado possível foram utilizadas algumas ferramentas *online* que ajudam a gerir tarefas e o código desenvolvido.

4.3.1 Trello

O *trello* é uma ferramenta de gestão de projetos que oferece informação sobre as tarefas que se têm de realizar, as que estão a ser implementadas e aquelas que já estão concluídas. A plataforma baseia-se em *cards* que contém as informações de cada tarefa e cada uma destas cartas está colocada num grupo delas. Esta ferramenta está a ser utilizada para controlar o trabalho que está a ser feito e para organizar o trabalho futuro de uma maneira mais simples.

4.3.2 GitLab

O Gitlab é um repositório de código baseado numa plataforma *web* que permite a equipas partilharem, armazenarem e editarem código através de mecanismos de controlo de versões do mesmo. Este repositório foi escolhido pois oferece mecanismos de integração com CI/CD assim como também com *Kubernetes*. Foi utilizado para armazenar todo o código desenvolvido ao longo do projeto, tanto do *backend* como do *frontend*.

4.3.3 TeamGantt

TeamGantt [25] é uma plataforma *web* para agendar tarefas ao longo de um período de tempo, através de gráficos de *Gantt*. Esta plataforma foi usada gratuitamente pois oferece um projeto sem qualquer custo. Permite criar dependências entre cada tarefa, adicionar uma percentagem de conclusão a cada tarefa e, se necessário, exportar o gráfico resultante. Esta ferramenta está a ser utilizada para perceber a ordem sobre a qual as tarefas devem ser realizadas ao longo dos semestres.

4.4 Metodologia de Desenvolvimento

Quando se trata do desenvolvimento de um projeto é necessário definir todas as suas etapas para que se obtenha um fluxo de trabalho o mais organizado possível.

Posto isto, quando foi realizada a primeira reunião com os orientadores deste estágio, foi acordado que nos iríamos reunir a cada duas semanas para apresentar o trabalho realizado até então e debater acerca daquele que era esperado fazer entre aquela reunião e a seguinte. Para além destes encontros a cada duas semanas, existe também uma outra reunião com todos os membros do Departamento de Engenharia Informática da Universidade de Coimbra (DEI-UC) que fazem parte do projeto NanoSen-AQM. Esses encontros eram realizados também a cada duas semanas mas na semana em que não existia a reunião com os orientadores do estágio. Nestas reuniões era apresentado o trabalho que ia desenvolvendo para a plataforma e eram apresentadas sugestões pelos vários parceiros do projeto.

Em suma, entre reuniões com a equipa do projeto e com os orientadores do estágio, existiam encontros todas as semanas.

4.4.1 Scrum-based

Scrum [26] é uma framework de desenvolvimento ágil que parte de um conjunto de conceitos e técnicas que ajudam a delinear de uma forma mais organizada o trabalho a fazer ao longo do desenvolvimento de um produto.

Este processo tem um fluxo definido [26] em que para começar necessitamos de ter todos os requisitos definidos à partida e depois dividí-los por *sprints*. Um *sprint* é uma meta normalmente entre duas a quatro semanas em que no ínicio é decidido o trabalho a fazer durante esse tempo e que no fim tem de estar concluído. No fundo, cada *sprint* tem os seus próprios requisitos. Os requisitos de cada uma dessas metas provêm maioritariamente daquilo que se conseguiu ou não fazer na meta anterior pois é nessa reunião entre o fim de um *sprint* e o ínicio de outro, que é analisado o trabalho realizado e se decide que requisitos irão fazer parte da nova meta.

A primeira fase deste método está referente ao primeiro semestre do estágio em que foram levantados os requisitos, fizeram-se decisões sobre o que se iria acrescentar à plataforma e de que forma iria ser realizado esse processo.

Para o segundo semestre entramos na fase de desenvolvimento do produto, existindo *sprints* ao longo do tempo e reuniões semanais para dar e obter *feedback* sobre o que está a ser feito. Ainda assim, alguns requisitos foram sendo alterados ao longo do processo.

4.5 Riscos

Ainda que sejam seguidos todos os passos previamente idealizados, podem sempre ocorrer riscos que influenciem o sucesso de todo este processo. Assim sendo, nesta secção irão ser abordados algum fatores que podem interferir negativamente no desenvolvimento da plataforma.

Fatores de sucesso

De forma a obter sucesso neste estágio, é esperado que se cumpram os seguintes objetivos:

- Terminar o trabalho planeado dentro do período temporal do estágio.
- As funcionalidades definidas ao longo deste documento devem ser implementadas e testadas.
- Plataforma disponível em versão *web* e *mobile*

Análise de Riscos

Os riscos que podem ocorrer ao longo deste processo são:

- **Novos pedidos por parte dos parceiros:** apesar do estágio ser no Departamento de Engenharia Informática (DEI), a plataforma faz parte de um projeto internacional. Como tal, parceiros de outras instituições podem pedir novas funcionalidades que irão depois alterar o planeamento idealizado.

  - Plano de mitigação: Tentar definir numa fase precoce do desenvolvimento as tarefas que devem ser realizadas e, no caso de aparecerem tarefas a meio do processo, definir prioridades para cada uma delas.

- **Cooperação com outras instituições:** sendo que o projeto é realizado em conjunto com a Universidade de Évora e esta detém grande parte do *backend* da plataforma (área onde este estágio se foca), podem acontecer atrasos ou até mesmo a recusa da implementação de algumas funcionalidades caso haja divergências na disponibilidade de ambas as partes.

  - Plano de mitigação: Reunir com ambas as partes antes da implementação de algum aspeto que envolva a cooperação das diferentes entidades. Acordar o que pode/deve ser feito e delinear o que compete a cada instituição.

- **Incumprimento do planeamento definido:** se por algum motivo uma funcionalidade demorar mais tempo do o esperado, pode depois existir um efeito "bola de neve"que irá afetar consequentemente todas as tarefas seguintes.

- Plano de mitigação: Cumprir o planeamento definido sem sobrepor tarefas ou executar tarefas fora do prazo.

- **Trabalho em tempo de pandemia:** tendo em conta o período pandémico em que vivemos no neste momento, torna-se difícil prever o que poderá acontecer num futuro próximo. Para lá desta incerteza, a comunicação com outros membros do projeto pode ser mais difícil do que o esperado.

  - Plano de mitigação: Definir canais de comunicação a usar para comunicar de forma eficaz. Marcar reuniões com antecedência e de forma frequente para haver partilha e discussão do trabalho que está a ser realizado.

- **Serviços na *cloud* que não suportem as tecnologias escolhidas:** tendo em conta as várias tecnologias que serão utilizadas neste estágio e visto que a plataforma terá de estar a funcionar na *cloud*, poderá existir o risco das plataformas onde este projeto está alojado não suportarem algumas das tecnologias escolhidas.

  - Plano de mitigação: Estudar as tecnologias antes de partir para o desenvolvimento para garantir que se suportam mutuamente.

4.6 Divulgação de plataforma

Nesta secção serão abordados momentos de divulgação da plataforma junto de outras entidades que decorreram ao longo deste estágio.

4.6.1 Universidade da Extremadura

Já numa fase final do projeto um dos estudantes nele envolvido através da Universidade da Extremadura deslocou-se até Coimbra para registar dados com um dos sensores móveis por eles calibrados. Durante a sua estadia houve um encontro nas instalações do DEI com a minha presença e a de um outro aluno de Coimbra para a discussão de outras formas de calibrar outros tipos de sensores que serão futuramente integrados na plataforma.

Esta reunião serviu também para dar conta do trabalho que cada uma das instituições estava a desenvolver e partilhar conhecimentos e sugestões sobre melhorias ou adições ao trabalho que já estava realizado.

4.6.2 Universidade da Madeira e *Startups* Madeirenses

Concluída a implementação da plataforma e das respetivas aplicações móveis era necessário divulgá-la a possíveis parceiros que poderiam usá-la para analisar a qualidade do ar na sua zona ou até armazenar os seus dados para obter uma melhor visualização. Esta visita ocorreu no sentido de finalizar um ciclo que foi iniciado há cerca de 2 anos atrás numa visita aos mesmos locais para dar conta do início deste projeto.

A visita deste ano teve ações de divulgação durante o dia 18 de Junho onde na parte da manhã tivemos encontros com duas startups madeirenses para troca de conhecimentos e ideias em relação ao funcionamento dos sensores e sobre as várias formas de como poderíamos analisar e utilizar os dados por eles gerados. Este encontro ocorreu na StartupMadeira localizada no Tecnopolo da Universidade da Madeira e contou com a presença de representantes da Universidade de Coimbra, Universidade da Madeira, representantes das empresas DTWay e BigWave e com o Presidente da StartupMadeira.

Depois destas reuniões seguimos para o edifício da Direção Regional do Ambiente e Alterações Climáticas onde nos encontramos com o Diretor Regional do Ambiente e Alterações Climáticas no sentido de divulgar a plataforma e explorar a possibilidade de se tornarem parceiros em futuros passos para a colheita de dados sobre a qualidade do ar naquela região autónoma. Tivemos uma grande abertura e disponibilidade para futuras parcerias e ficou em aberto a possibilidade de utilizarmos alguns dos seus dispositivos para fazermos essas medições.

Da parte da tarde foi tempo de mostrar a plataforma a funcionar. Num primeiro momento foi dada uma palestra na Universidade da Madeira onde participaram: o Professor Alberto Cardoso, a Professora Bernardete Ribeiro e o Professor Filipe Araújo, onde explicaram o projeto onde esta plataforma está envolvida entre outros aspetos teóricos e técnicos e que terminou com uma apresentação minha mostrando a plataforma web a funcionar. Terminada esta apresentação seguimos para a Ordem dos Engenheiros da Madeira onde mais uma vez repetimos a apresentação anteriormente realizada na Universidade da Madeira. Nos dois encontros existiu um tempo para debate muito construtivos de onde surgiram várias sugestões e também disponibilidade de lá voltar para continuar a desenvolver e a melhorar a plataforma do projeto NanoSen-AQM.

Capítulo 5

Desenvolvimento

Neste capítulo serão abordados os passos efetuados no desenvolvimento das soluções propostas. Irão ser explicadas as decisões tomadas, assim como problemas que foram surgindo ao longo do processo e a forma como foram contornados.

5.1 Estado atual do projeto

O projeto NanoSen-AQM começou em 2018. Ao longos dos últimos 2 anos várias funcionalidades têm vindo a ser implementadas por outros alunos de mestrado que têm também usado esta plataforma como tema do seu trabalho.

Quanto à plataforma está dividida em dois componentes principais: o *frontend* e o *backend*. O *backend* está a correr num servidor localizado na UEV enquanto que o *frontend* encontra-se no DEI-UC [6].

Foi feito um estudo de todo o código desenvolvido ao longo dos últimos anos e em relação a funcionalidades implementadas já está num estado muito avançado tendo uma base de código consideravelmente grande. Neste momento, o *backend* já está praticamente finalizado e no *frontend* irão ser adicionadas novas utilidades para tornar a plataforma mais completa. Já é possível visualizar *clusters*, ver os seus detalhes, listar por países e distritos, analisar a evolução das medições dos vários componentes ao longo do tempo, etc.

Relativamente às tecnologias que estavam a ser usadas no desenvolvimento da plataforma verifiquei que no *backend* estava a ser utilizado Django (uma framework web desenvolvida em Python) e no *frontend* usava-se Ionic / Angular.

5.1.1 Passagem de Testemunho

Visto que o projeto já está em desenvolvimento há algum tempo, como novo membro do grupo de trabalho, necessitei de auxílio a entender toda a arquitetura do projeto pois, tal como já referido, tinha uma complexidade grande e vários pormenores que de outra forma não conseguiria ter acesso.

Para tal, tive várias reuniões com o antigo aluno que esteve presente no desenvolvimento todo da plataforma, o Pedro Lucas. Nessas reuniões abordamos inicialmente a parte de acesso aos dados no *backend*, mais especificamente a API e a base de dados e ao longo das outros encontros fomo-nos direcionando para o *frontend*. Foi feita a configuração

das tecnologias utilizadas no meu computador e como era feito o *deploy* da aplicação para produção.

5.1.2 Mapa de localização das medições

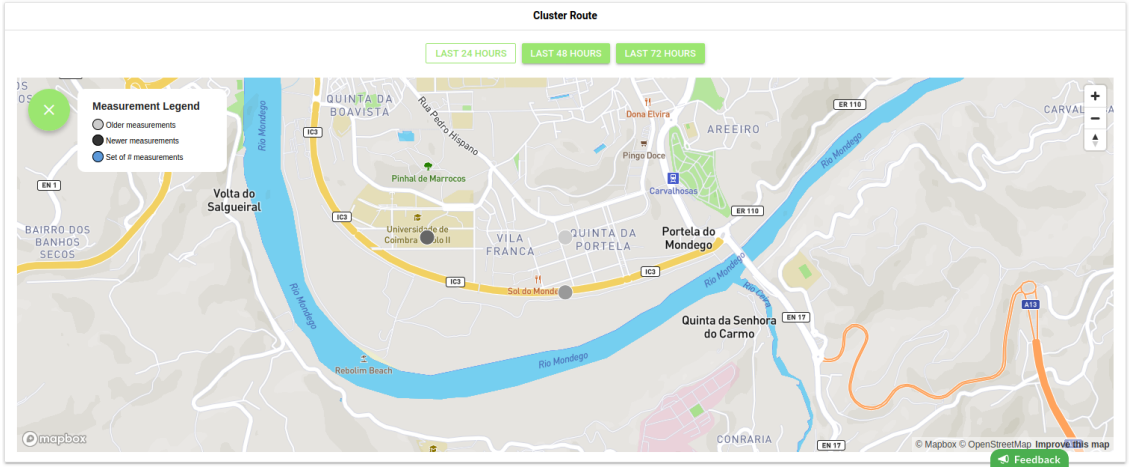

Figura 5.1: Estado do protótipo para a funcionalidade implementada

Sabendo que a melhor forma de aprender é praticar comecei por implementar uma funcionalidade que foi pedida por um dos parceiros do projeto.

Para contextualizar, existem dois tipos de clusters: os fixos (aqueles que estão sempre num local, seja em postes, em habitações, etc.) e os móveis (que podem alterar a localização ao longo do tempo, podendo estar colocados, por exemplo, em bicicletas e/ou carros).

Posto isto, chegou-se à conclusão de que seria interessante observar o percurso destes conjuntos de sensores num mapa. Para tal, comecei por estudar a API e verifiquei que já existia um endpoint que me fornecia as informações para popular o mapa. De forma a ficar um mapa completo iria necessitar de:

- Coordenadas GPS das medições;
- Registo temporal das mesmas;
- Valor, parâmetro e unidade de medição registada;
- ID do sensor, pois um cluster é um conjunto de vários sensores.

Conseguindo estas informações através da API faltava agora colocá-las de forma perceptível num mapa. Comecei por estudar possibilidades de libraries que me ajudassem nesta implementação quando encontrei, já no projeto, uma API chamada MapBox que fornecia os serviços de display de mapas que eu procurava. Após isto, comecei por estudar a versão JavaScript desta library (Mapbox GL JS) de forma a programar aquilo que nos tinha sido proposto.

Depois de reunir toda esta informação comecei a estudar a melhor forma de colocar o mapa na página de detalhes já existente. Como era necessário fazer uma multi-plataforma utilizei os templates das divisões já existentes de forma a ser auto-ajustável para cada uma delas. Esses templates chamam-se ion-card [18].

Assim que o mapa estava construído tive de pensar de que forma iria organizar os vários pontos e como iria dar a informação das várias medições. Ficou então decidido que iriam ser marcados pontos no mapa em tons cinza, em que os mais claros representam as medições mais antigas e os mais escuros as mais recentes. Quando se faz zoom-out as medições são agregadas em conjuntos para permitir uma melhor visualização e ter a informação mais organizada. De forma a que o mapa tivesse outra utilidade que não apenas a visualização das localização, cada ponto representado no mapa pode ser selecionado para que a informação dessa medição específica seja visualizada.

Ao longo do tempo esta funcionalidade sofreu várias modificações que iam sendo propostas nas reuniões quinzenais com todos os membros envolvidos no projeto [5].

5.2 Previsão dos valores dos poluentes

Uma das funcionalidades principais da plataforma era ter a previsão dos valores dos sensores para as próximas 24 horas, para tal, usando funções de previsão desenvolvidas por outros alunos presentes no projeto, todos os sensores apresentam a previsão dos valores de todos os poluentes.

De forma a isto ser possível, o código desenvolvido para calcular a previsão é utilizado dentro de uma instância do OpenWhisk que recebe os valores anteriores medidos para um determinado poluente e usa-o como entrada no código pretendido e devolve um Json com os próximos 24 valores horários. Por padrão a função utilizada é a Moving Average mas os donos de sensores podem criar a sua própria e fazer upload do seu código para a plataforma e posteriormente utilizá-lo para calcular as previsões. Todos os sensores quando criados têm a função padrão anteriormente referida associada ao cálculo da previsão.

5.3 Upload de dados dos sensores

Apesar de a maior parte dos sensores enviarem as medições em tempo real para a plataforma era importante que existisse a possibilidade de carregar os dados para a mesma através de um ficheiro. Para tal, os dados têm de seguir um determinado formato csv com as colunas necessárias para a representação de uma medição. Com esse ficheiro os donos dos sensores podem fazer o upload dos dados para a plataforma a qualquer momento.

A solução para implementar esta funcionalidade foi modificar o código para converter o csv para um formato Json que por sua vez é enviado para o backend para as medições serem adicionadas à base de dados. A cada novo carregamento é verificado se as medições trazem as colunas corretas para a representar uma nova medição. Uns tempos depois verificou-se que esta solução não funcionava para ficheiros com mais de 5000 novas medições pois excedia o tamanho limite para o pedido que era enviado ao backend. O problema foi resolvido modificando o código para dividir o pedido original em outros pedidos com apenas 1000 medições e enviá-los separadamente.

5.4 Download de dados dos sensores

Todos os dados das medições efetuadas ao longo do tempo são guardadas na base de dados para visualizações futuras. Assim sendo, era necessário permitir aos utilizadores da plataforma que fizessem o *download* dos dados para poderem guardá-los e analisá-los.

A solução que estava implementada não estava a funcionar corretamente e teve de ser implementada uma nova solução. Desse modo, é construído no *backend* da plataforma um ficheiro no formato *Comma-separated values* (CSV) que contém a data em que a medição foi registada, o id do sensor que fez a medição, o valor registado, o parâmetro e a unidade de medida do valor registado e, por fim, as coordenadas geográficas onde a medição foi efetuada.

Para ter acesso a este ficheiro os utilizadores precisam de aceder aos detalhes de um determinado *cluster* e no gráfico das medições carregar no botão de *download* e escolher um período temporal sobre o qual querem que as medições estejam inseridas.

5.5 Aplicações *mobile*

Finalizada a plataforma era importante conseguir chegar também aos dispositivos móveis. Tendo usado a framework Ionic conseguimos, com o mesmo código, gerar as aplicações tanto para Android como para iOS. Apesar de a maior parte das funcionalidades funcionarem sem quaisquer alterações no código inicial, algumas necessitaram de ser alteradas, nomeadamente, aqueles que usam serviços de armazenamento e localização.

5.5.1 Android

Na versão Android foi necessário alterar a forma como estava a ser guardado o ficheiro que resultava do download dos dados das medições ou do histórico do AQI. Como estava a ser utilizada uma biblioteca que não funcionava para dispositivos móveis, foi usada a engine Cordova que já vêm integrada como Ionic e que traz algumas funções que ajudam no desenvolvimento de soluções híbridas.

Posto isto foi também necessário alterar o pedido das permissões de acesso tanto ao armazenamento (para o download e upload dos dados) e da localização (para encontrar os clusters mais próximos).

Uma das principais alterações realizadas para a versão *Android* que depois também foi reutilizada para a versão *iOS* foi a forma como o mapa dos *clusters* de sensores estava a ser carregada. Tendo em conta que o código que estava escrito era apenas utilizado na plataforma *web* o carregamento dos marcadores do mapa estava a ser gerado através da criação de um componente *HTML* para cada um dos marcadores. Ora, nas versões móveis, tendo em conta que a capacidade de processamento não é tão elevada, a criação do mapa e a sua utilização era extremamente desagradável para os utilizadores porque a navegação pelo mapa era demasiado lenta. Para resolver este problema, refiz e desenvolvi todo o código de carregamento do mapa de forma a utilizar a biblioteca do Mapbox GL JS [30] para criar o mapa e para usar os marcadores criados por essa biblioteca para preencher as informações do mapa. Desta forma, reduziu-se a lentidão anteriormente verificada e forneceram-se as mesmas funcionalidades.

Visto que a aplicação *android* foi produzida antes da versão *iOS*, grande parte das alterações feitas para este SO foram depois aplicadas em ambos. Uma das alterações efectuadas que se aplicaram dos dois sistemas foi o menu lateral responsivo. Como não ainda tinha sido desenvolvido código para tal, quando se gerou a primeira versão da aplicação *android*, o menu lateral ocupava grande parte do ecrã do telemóvel o que fazia com que o espaço disponível para a apresentação das informações fosse reduzido e tornava a aplicação

um pouco deselegante. Assim sendo, foi feita a alteração no componente *Ionic* que fornecia o menu lateral e o menu passou a ficar escondido em ecrãs mais pequenos com um botão para expandir o menu, caso fosse necessário.

5.5.2 iOS

Na versão da plataforma para dispositivos com o sistema operativo iOS foi necessário também fazer alterações para que algumas funcionalidades funcionassem e também foi preciso configurar as permissões dos serviços de localização e armazenamento que a aplicação necessita.

Inicialmente previa-se que as alterações realizadas para a versão Android (5.5.1) funcionassem também neste sistema operativo, no entanto, algumas das bibliotecas que a *engine* do Ionic utilizava não estavam a ser suportadas pelos dispositivos da Apple. De forma a resolver este problema foi necessário instalar novas versão de algumas bibliotecas.

Em relação a modificações no código desenvolvido aplicaram-se as alterações referidas na secção anterior (5.5.1) e também duas outras sugeridas por utilizadores que testaram a aplicação na sua primeira versão.

A primeira alteração foi feita na legenda dos gráficos das medições registadas pelos sensores na página de detalhes dos *clusters*. Antes da alteração a legenda estava confusa porque a linha que representava os valores registados era muitos semelhante à linha que representava a previsão para as próximas horas, o que poderia induzir em erro alguns utilizadores. Esta alteração foi depois utilizada na versão *Android* e na plataforma *web*.

A segunda modificação, que acabou depois por não ser utilizada, era implementar a possibilidade de fazer *zoom* também no gráfico das medições para permitir uma visualização mais focada num determinado período temporal. Acabou por não ser utilizada na aplicação pois esta modificação tinha conflitos com o *scroll* naquela página. Quando se navegava por essa página no telemóvel, ao chegar ao gráfico, a aplicação não conseguia perceber se o *scroll* estava a ser feito na página ou no gráfico (assumindo que se estava a fazer zoom). Visto que este conflito diminuía bastante a qualidade da utilização da plataforma achou-se, por bem, não publicar esta funcionalidade.

Quanto tudo parecia resolvido e foi feito o *upload* da aplicação para a Apple Store, recebemos uma notificação a informar que a tecnologia que transformava o código escrito com a *framework Ionic* em código que conseguia ser executado em iOS, já não era mais suportada pela Apple e que teria de ser alterada por uma mais recente. Essa alteração foi realizada com alguns problemas ao longo do processo pois algumas funcionalidades deixaram de funcionar até chegar a uma configuração que resultasse na aplicação sem erros e pronta para ser utilizada.

Depois de ultrapassados estes problemas a aplicação foi aceite na Apple Store e distribuída internamente pelos membros do projeto.

5.5.3 Validação e testes das aplicações

De forma a validarmos as soluções desenvolvidas para os dispositivos móveis decidimos dividir esta fase em três etapas. A primeira aconteceu ao longo do período de desenvolvimento, assim que se ia adicionando ou alterando algo nas aplicações testou-se de imediato essa alteração.

Depois desta fase e já com as aplicações próximas de uma versão parecida com a final, fizemos testes internos onde partilhamos a aplicação, para ambos os sistemas operativos móveis, por diversos membros do projeto. Durante este período fomos recebendo feedback sobre pontos a melhorar e sugestões para integrar na aplicação. Algumas alterações foram efetuadas e estão presentes na versão final das aplicações, enquanto outras que foram implementadas por acharmos que fariam sentido, quando colocadas em prática acrescentavam outros problemas e decidimos não avançar com essas modificações.

Por fim, na versão Android existiu ainda uma versão BETA que foi disponibilizada para 1000 utilizadores para poderem fazer download e testar a aplicação e, deste modo, obtermos um maior números de testers que por consequência nos daria uma melhor ideia da robustez da aplicação. Infelizmente, a divulgação desta fase foi ineficiente e não obtivemos o feedback que desejavamos.

5.6 Docker Swarm

Para termos o backend distribuído por várias máquinas e para garantir que quando um container parar este consiga voltar a operar automaticamente configuramos um swarm em três máquinas que irão suportar um ambiente de desenvolvimento. Para começar é necessário definir a máquina que irá controlar todos os containers construídos. Existem 3 máquinas denominadas por: nano1-dev, nano2-dev e nano3-dev.

Foi definida a máquina nano2-dev como manager. Será nesta máquina que serão executados todos os comandos de configuração e manutenção e será também responsável por distribuir os containers pelas outras máquinas. Quando executado o comando para definir como manager uma determinada máquina é-nos mostrado um token para depois associar outras máquinas (denominadas de workers) ao swarm. Para tal é necessário ir a cada uma dessas máquinas, neste caso a nano1-dev e a nano3-dev e executar o comando para se juntar ao conjunto de máquinas fornecendo o token que identifica a máquina que é o manager.

5.6.1 Deploy de Containers

De forma a tornar mais fácil o deploy dos containers docker foram definidos num ficheiro de configuração os serviços necessários para a execução do backend. Foram eles:

- O serviço da base de dados timescale
- Um serviço com o container da API
- Um serviço do NGINX para redirecionar os pedidos feitos ao IP da máquina para o container que estará a suportar o serviço da API.
- Um serviço para a cache da API suportado por uma imagem Redis.
- Um serviço para carregar o sistema com medições provenientes do OpenAQ.

No ficheiro de configuração são descritas as imagens que serão utilizadas para a criação dos containers. Na figura 5.2 pode ver-se a configuração utilizada para o *container* da API.

```
django:
    image: flaviofernandes/django:latest
    build:
      context: djangoserver
      dockerfile: docker/Dockerfile-prod
    user: django
    depends_on:
      - timescale
    command: /gunicorn.sh
    env_file: djangoserver/.env
    ports:
        - 5000:5000
    restart: unless-stopped
```

Figura 5.2: Configuração do *container* da API Django

Visto que o Docker Swarm replica os containers pelas máquinas associadas àquele conjunto de máquinas, surgiu um problema com as imagens Docker criadas localmente, nomeadamente a imagem da API e da Base de Dados. Como estas imagens foram criadas com código desenvolvido ao longo do projeto, ao serem construídas, apenas iam estar disponíveis na máquina onde esse processo foi efetuado, neste caso, no manager. Para contornar este problema utilizamos o repositório online para imagens Docker, o Docker Hub. Deste modo, sempre que existe uma nova alteração no código que implique construir de novo a imagem, a imagem é construída, é feito upload dessa imagem para o repositório e no ficheiro de configuração colocamos o nome do repositório e da imagem e assim já pode ser replicada pelas diferentes máquinas pois já não está restritamente armazenada naquele que é o manager.

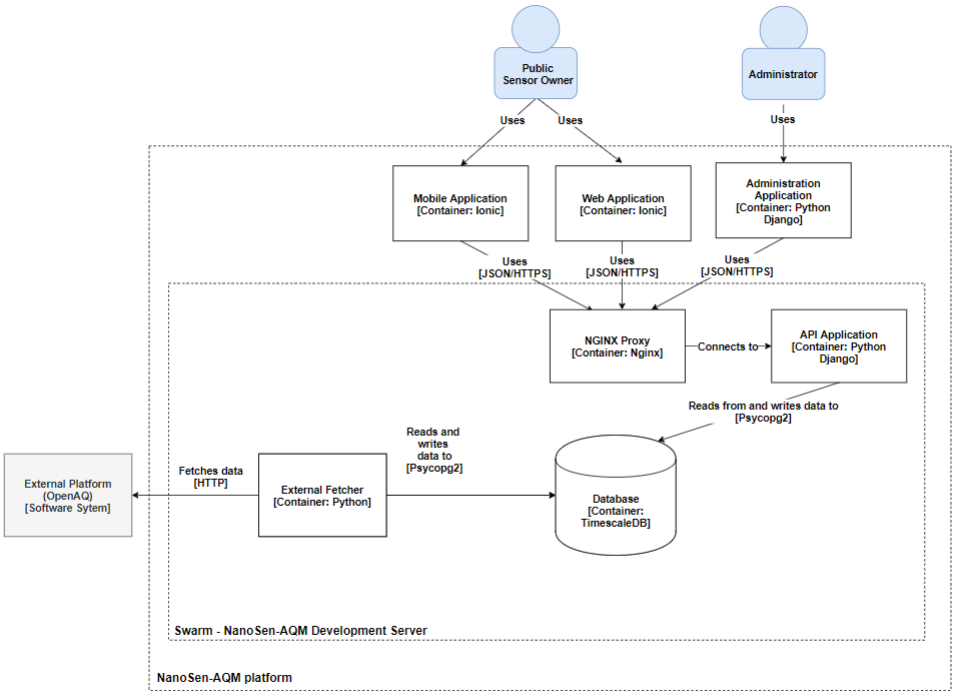

Figura 5.3: Arquitetura representativa dos serviços do Docker Swarm

Da implementação dos serviços resultou a arquitetura representada na Figura 5.3. Os *containers* pertencentes aos *swarm* estão dentro da mesma rede para que se consigam comunicar entre si. Estes serviços podem ser acedidos pelas aplicações *web* e *mobile* se as mesmas estiverem num ambiente de desenvolvimento, por exemplo. Também o *Django* oferece uma plataforma de administrador que pode ser acedida para executar ações diretamente na API.

O *container* representado por *External Fetcher* é o responsável por retirar as medições fornecidas pelo OpenAQ que depois de processadas e validades são enviadas para a base de dados para ficarem disponíveis e armazenadas.

5.7 Testes

Para testar o código desenvolvido em Python para a API Django escrevemos testes unitários para cobrir o maior código possível, para tentarmos executar todas as possibilidades e verificar se funcionam como é suposto. Ao longo do desenvolvimento do projeto o antigo estagiário foi escrevendo testes para as funcionalidades que desenvolveu e agora fomos adicionando mais testes, como demonstrado no apêndice A.1, para as novas funcionalidades e alterar aqueles cujo código a ser executado também sofreu alterações.

De forma a termos sempre uma cobertura elevada, a cada novo *endpoint* que se adicione à API, novos testes têm de ser escritos.

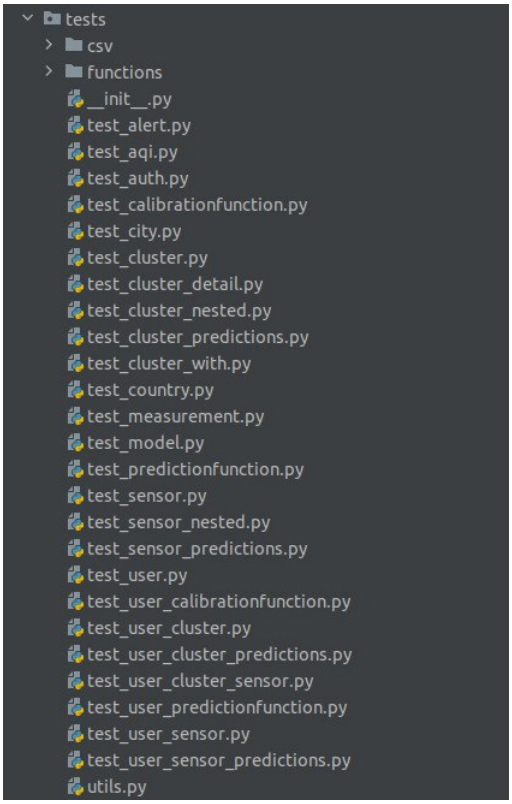

Figura 5.4: Lista de ficheiros de testes à API

Como observado na figura 5.4, os testes estão distribuídos por 25 ficheiros sendo, cada um desses ficheiros, referente a um determinado tópico que se quer testar. Os testes

aqui definidos podem ser executados em conjunto ou separadamente, caso seja necessário. Estes testes serão executados sempre que houver um novo *commit* para o repositório, como será explicado na secção 5.8. No total existem 738 testes para serem executados.

5.7.1 Estrutura dos testes

Os testes seguem um esquema em que inicialmente é feita a preparação das informações que serão necessárias para, posteriormente, serem enviadas num pedido HTTP. Esse pedido é direcionado ao *endpoint* que queremos testar no momento. Feito o pedido, espera-se pelo código da resposta e verifica-se se este corresponde àquele que era esperado (Código 200 no caso de pedidos bem sucedidos). Depois é necessário verificar o conteúdo da resposta. Caso o resultado recebido seja igual àquele que que definimos como correto, o teste passa com sucesso, caso contrário, falha. No apêndice A.2 pode ver-se um exemplo de código escrito para um teste unitário.

5.7.2 Resultados

Atualmente todos os 738 testes que foram escritos passam com sucesso. Os testes foram efetuados aos diversos endpoints criados no backend da aplicação tendo-se obtido uma cobertura de 91% no ficheiro onde estes são descritos, no *views.py*, como se pode ver na Figura 5.5. Sendo assim e visto que grande parte do código foi testada consegue-se garantir um alto nível de funcionalidade no que corresponde à API.

Em relação ao resto dos ficheiros continua a haver uma elevada cobertura de testes. Em algumas situações essa percentagem é mais baixa porque ou é código que não é utilizado de momento ou então ainda não foram escritos testes para tal. Esse valor consegue facilmente ser superior escrevendo mais testes que cheguem a esse código.

Module	statements	missing	excluded	coverage ↓
api/exceptions.py	6	0	0	100%
api/urls.py	55	0	0	100%
api/factories.py	180	4	0	98%
api/admin.py	90	3	0	97%
api/signals.py	113	6	0	95%
api/views.py	776	67	0	91%
api/models.py	329	33	0	90%
api/permissions.py	112	11	0	90%
api/serializers.py	736	71	0	90%
api/openwhisk.py	70	8	0	89%
api/utils.py	181	28	0	85%
api/response.py	45	7	0	84%
api/cache.py	29	5	0	83%
api/mixins.py	29	7	0	76%
api/tokens.py	44	12	0	73%
api/filters.py	173	73	0	58%
api/renderers.py	65	27	0	58%

Figura 5.5: Percentagem de cobertura dos testes nos diversos ficheiros

Para lá destes testes, ao longo do desenvolvimento de novos pedaços de código foram sendo testadas as novas adições e analisou-se se o comportamento observado correspondia ao esperado.

5.8 CI/CD

Com o CI/CD do gitlab conseguimos testar a aplicação a cada novo commit e caso passe em todos os testes, o código será colocado em produção na máquina definida anteriormente como manager (6.5). Para testar o código desenvolvido serão utilizados testes desenvolvidos para a API e caso sejam todos executados com sucesso, será feita uma conexão SSH à máquina nano2-dev e serão executados os scripts para colocar o novo código disponível.

Para definirmos o processo do CI/CD é necessário especificar todos os passos num ficheiro de configuração no formato .YML. Nesse ficheiro começámos por fazer a instalação de todas as bibliotecas necessárias para a execução de código Python assim como um ambiente virtual para suportar o código da API. Faz-se também a instalação das bibliotecas SSH necessárias para o passo do deploy e adicionamos a chave privada do SSH da máquina nano2-dev aos hosts da máquina do GitLab para que possamos fazer sempre a ligação à máquina sem ser necessário inserir a password. Esta configuração pode ser visualizada na figura 5.6

```
.python_before: &python_before_template
  before_script:
    - apk update && apk upgrade
    - apk add postgresql-dev gcc musl-dev bash python3 python3-dev py3-
pip build-base
    - pip3 install virtualenv
    - virtualenv -p python3 venv
    - . venv/bin/activate


before_script:
    - apt-get install openssh-client
    - eval $(ssh-agent -s)
    - echo "$SSH_PRIVATE_KEY" | tr -d '\r' | ssh-add - > /dev/null
    - mkdir -p ~/.ssh
    - chmod 700 ~/.ssh
    - ssh-keyscan "$SSH_HOST_IP" >> ~/.ssh/known_hosts
    - chmod 644 ~/.ssh/known_hosts
```

Figura 5.6: Instalação das bibliotecas e configuração do SSH

Depois de feita esta configuração inicial é necessário instalar as bibliotecas requeridas para a execução do código desenvolvido e foi criada uma base de dados de testes para guardar os dados criados ao longo deste processo. A base de dados é criada através de um script desenvolvido anteriormente para este fim.

Por fim, entramos na última fase deste processo que é conectar à máquina que está responsável pelo deploy da API, retiramos os dados que estão no repositório do *GitLab* para termos a versão atualizada do código localmente, construímos a nova imagem *Docker* sobre o código da API e enviamos essa imagem para o *DockerHub*. Para terminar e colocar tudo em produção corremos o comando para distribuir os serviços pelo swarm, como descrito na figura 5.7, e tudo deverá ficar a funcionar.

```
deploy_staging:
  type: deploy
  environment:
    name: staging
  script:
    - ssh -o StrictHostKeyChecking=no -v ffernandes@nano2-
dev.xdi.uevora.pt "cd data-access/ && git status && git pull && echo
$SUDO_PWD | sudo -S docker ps && cd djangoserver && sudo docker build -f
docker/Dockerfile-prod -t flaviofernandes/django:latest . && sudo docker
push flaviofernandes/django:latest && sudo docker push
flaviofernandes/django:latest && cd .. && sudo docker stack deploy -c
docker-prod.yml data && exit"

  only:
    - master
```

Figura 5.7: Código a executar na máquina remota para *deploy* da nova configuração

Assim sendo, o processo conseguido segue a lógica representada na figura 5.8. Sempre que existir um novo *commit* para o repositório do *GitLab* onde é armazenado o código é executado o *script* configurado no ficheiro anteriormente referido. Num primeiro momento é executada a parte referente aos testes que foram desenvolvidos e aos quais a API é submetida para verificar se tudo está de acordo com o esperado. Caso todos os testes sejam submetidos com sucesso, o processo continua, caso contrário, a pipeline termina e o desenvolvedor é informado por e-mail que algo correu mal no processo.

Continuando o processo, entra na fase final onde o novo código, já testado, é colocado no *swarm manager* através de comandos executados por SSH. Mais uma vez o desenvolvedor é informado caso algo corra mal. Se tudo correr como o esperado, as novas funcionalidades são disponibilizadas na API em produção.

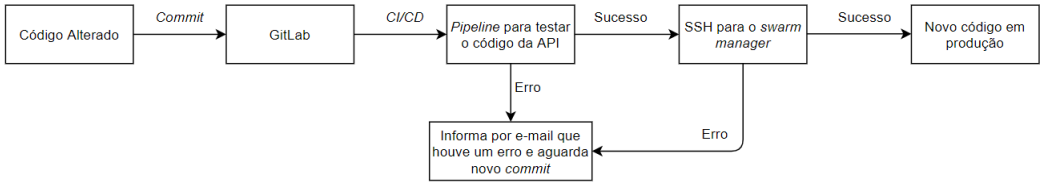

Figura 5.8: Esquema do processo desde um novo *commit* até à disponibilização das novas alterações na API

5.9 OpenAQ Fetcher

Como o número de sensores pertencentes ao projeto ainda é relativamente reduzido, a plataforma iria ter muito pouca informação para quem a visitasse nesta fase. Assim sendo, decidiu-se usar serviços externos para alimentar a aplicação com dados e, desta forma, torná-la mais completa.

A solução implementada utilizava a API do OpenAQ para fornecer os dados de medições horárias de sensores espalhados por Portugal, Espanha e França. Esta API, por sua vez, recolhia e agregava dados de outras entidades europeias e locais para uma maior cobertura geográfica.

Ao longo deste estágio a forma como a API enviava os dados sofreu alterações e foi necessário alterar o código de forma a ser compatível com o novo formato das respostas que o OpenAQ nos enviava. Depois de receber os dados através de um pedido HTTP estes eram organizados no formato que pretendíamos para depois enviar por Kafka para um consumer que está sempre a receber estas informações. Quando esse serviço recebe as mensagens anteriormente enviadas, verifica a sua integridade e escreve os dados na base de dados para estes estarem disponíveis.

5.10 Outros

Ao longo do percurso deste estágio foram aparecendo outros trabalhos que não estavam previstos e que não eram relacionados diretamente com o estágio mas que impactavam a disponibilidade da plataforma. Estes problemas ocuparam muito mais tempo do que aquilo que era expectável porque eram partes do projeto com as quais nunca tinha trabalhado e, como tal, não tinha conhecimento prévio do que estava feito e, então, tive

primeiro de estudar essas zonas, que não era inicialmente previsto que fosse necessário trabalhar com elas, para depois resolver os problemas identificados.

Inicialmente foi necessário dar acompanhamento a dois alunos que estavam a trabalhar também na plataforma através de uma bolsa de estudo. Estavam a desenvolver algoritmos para a previsão de valores dos poluentes e foram necessitando da minha ajuda para colocar os novos algoritmos na plataforma.

Também ao longo deste tempo foram aparecendo falhas na plataforma que tiveram de ser resolver para a manter operável. Existiu um episódio em que um dos parceiros do projeto teve de reiniciar uma das máquinas onde estava alojada a API e todos os serviços tiveram de ser reiniciados também por mim.

Para lá destes problemas, foram surgindo algumas sugestões para fazer alterações no *frontend* da plataforma que também foram ocupando algum tempo, pois era necessário programar essa parte e, algumas delas não foram escolhidas por não serem tão necessárias como era inicialmente previsto.

Capítulo 6

Conclusão

Nesta secção será feita uma análise sobre o estágio, o trabalho nele realizado, uma comparação entre aquilo que estava inicialmente planeado e o que realmente foi feito. Serão descritas também as dificuldades encontradas ao longo do tempo e trabalho que pode ser realizado futuramente.

6.1 Considerações finais

De uma forma global, os objetivos fundamentais foram cumpridos. Conforme foi referido ao longo deste documento foram disponibilizadas duas aplicações para dispositivos móveis, para *iOS* e *Android* que dão aos utilizadores um acesso mais rápido e mais concentrado aos dados fornecidos na plataforma do projeto. Foi também feita a simplificação do processo de *deploy* de novas funcionalidades do *backend* da plataforma utilizando tecnologias que permitem também uma maior recuperação dos diferentes serviços em caso de falha. Ainda que com uma tecnologia diferente daquela que estava inicialmente prevista (anteriormente *Kubernetes*, agora *Docker Swarm*) conseguiu-se chegar ao mesmo objetivo de conseguir a recuperação e disponibilidade dos serviços e também fazer a atualização do backend em produção e testar o código a cada novo *commit*.

Conseguiu-se também ir fazendo a manutenção aos serviços já existentes resolvendo algumas falhas que apareceram e adicionar novas funcionalidades tanto ao *backend* como ao *frontend*.

Ainda assim, ficaram alguns pontos por realizar. Ainda que tenha sido definido como opcional, as *dashboards* de monitorização não foram adicionadas. Outro aspeto abordado que não foi implementado foi o *multi-tenancy* que permitia a outras entidades usar a plataforma como meio de visualização dos seus dados. O facto de terem aparecido algumas tarefas que não estavam inicialmente previstas mas que eram essenciais para a plataforma, tirou bastante tempo que faltou depois para a implementação destes tópicos.

6.2 Planeamento esperado *vs* trabalho realizado

O planeamento inicialmente previsto para o segundo semestre (Figura 6.1 deste estágio sofreu ligeiras alterações como se pode ver na Figura 6.2.

Inicialmente começou-se por realizar tarefas que não estavam previstas quando se fez o primeiro planeamento pois eram alterações que tinham de ser feitas na plataforma que já estava em produção. Como eram tarefas relativamente rápidas optei por realizá-las numa fase inicial.

Visto que *Kubernetes* foi substituído pelo *Docker Swarm* também existe esse diferença no planeamento pois inicialmente comecei por fazer um curso *online* sobre *Kubernetes* mas, uns tempos depois, numa reunião com pessoas com mais experiência com a tecnologia verificou-se que não iria haver tempo para terminar tudo, daí a trocar por Docker Swarm. As aplicações cliente foram desenvolvidas a seguir ao *Docker Swarm* tal como já estava previsto.

A configuração do GitLab CI/CD passou para o final do semestre pois percebeu-se que era necessário ter as máquinas totalmente configuradas com o *Docker Swarm* para se conseguir testar a configuração que se iria implementar. Previa-se inicialmente que os testes ocupassem mais tempo mas como já existiam alguns testes escritos, a quantidade de novos testes foi inferior à que era prevista. Desta forma, os testes foram escritos em simultâneo com a configuração do CI/CD pois era neste momento que iam ser utilizados.

Ao longo deste período foram realizados também trabalhos de manutenção a soluções que já estavam implementadas mas que foram necessitando de intervenção da minha parte.

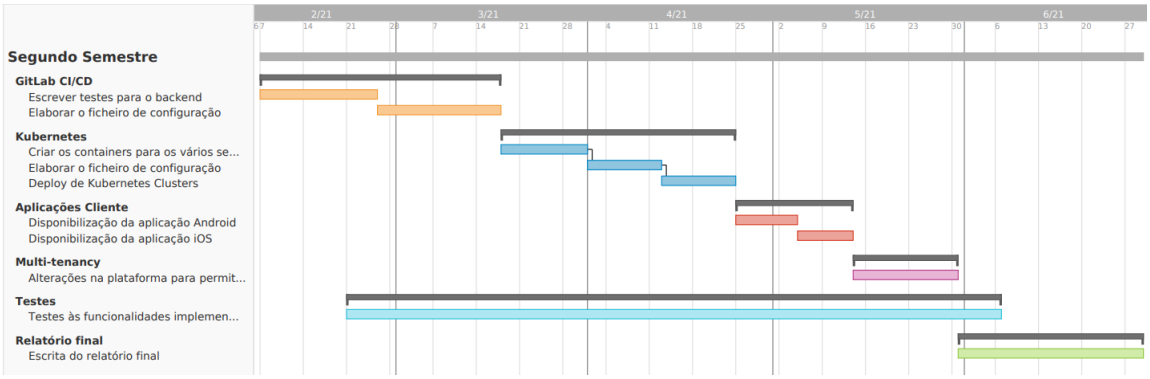

Figura 6.1: Planeamento original para o segundo semestre

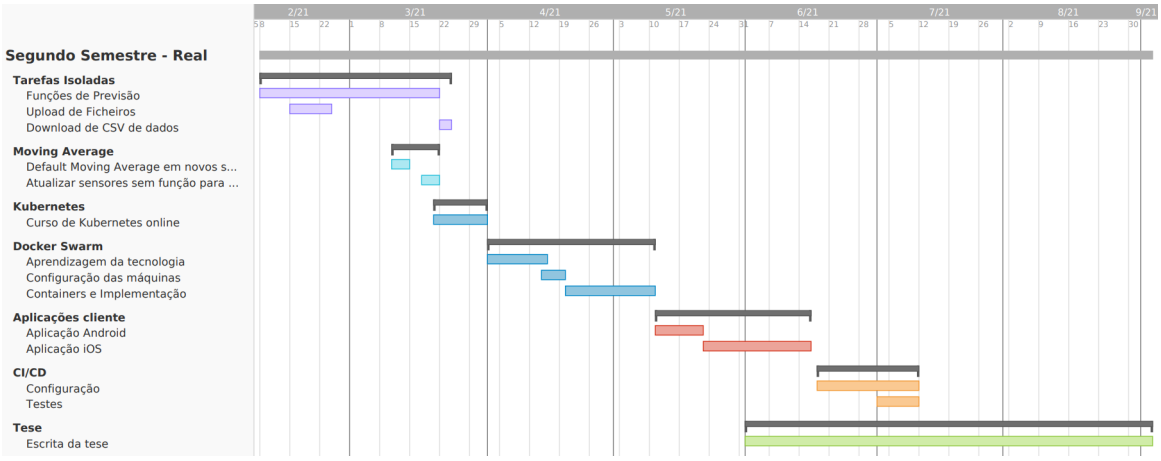

Figura 6.2: Planeamento real no segundo semestre

De um modo geral o planeamento foi seguido. As tarefas sofreram um pequeno atraso devido a novas tarefas que apareceram e por consequência todas as tarefas posteriores sofreram alterações nesse sentido. Também o facto de algumas tarefas ocuparem mais tempo do que aquilo que se esperava atrasou algum trabalho.

6.3 Dificuldades encontradas

Esta secção aborda dificuldades que foram aparecendo ao longo do estágio e que atrasaram o processo.

OpenWhisk

O *OpenWhisk* é uma ferramenta que está a ser utilizada para executar funções *Python* criadas pelos utilizadores para fazer a calibração e previsão das medições. Ao longo do projeto aconteceu este serviço ter algumas falhas e, visto que não era um serviço no qual não estava previsto que se fosse mexer, não tinha conhecimento de como funcionava e, qualquer alteração para corrigir algumas falhas ou até verificar *logs* para ver o que se estava a passar demorava algum tempo. Noutras situações em que outros alunos que estavam a desenvolver essas funções e estavam a dar erro, foi necessária a minha intervenção para conseguir perceber qual era o erro.

Aplicações *mobile*

A passagem do código da plataforma web para as aplicações *mobile* ocorreu relativamente bem, no entanto, existiram alguns problemas na versão *iOS* que ocuparam muito mais tempo e trabalho do que aquele que era esperado. Inicialmente o texto que deveria aparecer nos diferentes locais da aplicação aparecia com o nome da variável ao invés de aparecer o texto que estava definido. O mapa dos *clusters* não carregava, a aplicação não passava do *splash screen* e, nos detalhes do *cluster* as previsões meteorológicas também não carregavam. Alguns deste problemas corrigiram-se atualizando as bibliotecas que forneciam estas funcionalidades. No entanto, até chegar à solução de algumas destas falhas passaram-se semanas a tentar corrigir o problemas de formas que não se verificaram eficazes. Também o *upload* da aplicação para a Apple Store foi demorado porque havia componentes desatualizados que depois influenciaram noutros aspetos da aplicação.

Configuração do NGINX

A configuração do NGINX para utilizar com o *Docker Swarm* foi uma tarefa bastante dispendiosa em relação a tempo. Todo o processo de ter certificados válidos e configurar o serviço de forma a que ficasse operacional foi desgastante. Ainda assim, onde senti mais dificuldade foi em conectar este serviço ao serviço da API para redirecionar os pedidos para o *backend*. Tendo em conta que o serviço da API estava num *container docker* a correr dentro do *swarm* foi necessário colocar o NGINX na mesma rede que esse *container*. Também aqui o processo até chegar à solução correta passou por muitas tentativas de outras solução que não funcionaram.

CI/CD

Esta foi uma parte do estágio que também tirou mais tempo do que era esperado

porque nunca tinha tido experiência com esta tecnologia. A configuração do ficheiro para descrever os passos que deveriam ser feitos foi difícil nomeadamente na ligação por SSH à maquina onde era suposto fazer o *deploy*. Para entrar na máquina era necessário *password* no entanto, essa tinha de ser inserida manualmente e não por uma linha de código como teria de ser. A solução que se aplicou foi armazenar a chave pública no GitLab como variável de ambiente e fazer o *login* através dessa chave. Assim, a máquina reconhece essa chave e permite a entrada sem a *password*.

6.4 Trabalho futuro

Nesta secção será descrito o trabalho que poderá ser desenvolvido o futuro. Apesar do estágio ter terminado, o projeto irá continuar e seria interessante ir adicionado novas funcionalidades à plataforma.

***Dashboards* de monitorização**

Tal como já foi referido neste documento as *dashboards* de monitorização seriam uma adição interessante para se ter noção da saúde dos serviços que estão presentes na aplicação. Era uma ferramenta que iria valorizar a plataforma pois fornece informação que, neste momento, não se consegue obter.

Alteração do tipo de utilizador

Através da experiência enquanto utilizador/administrador da plataforma do projeto, considero que era necessário adicionar na plataforma os privilégios de administração que os administradores da plataforma têm. Um dos casos que aconteceu durante este estágio foi o facto de ter de dar permissões de *"Sensor Owner"* a uma conta que até àquele momento apenas era um utilizador normal. Para tal, essa funcionalidade só é permitida através do acesso à plataforma de administração do Django. Assim sendo, esta funcionalidade permitiria que todas estas vantagens estivessem concentradas na mesma plataforma.

Multi-tenancy

Para potenciar a utilização da plataforma seria interessante fazer com que outras entidades pudessem começar a inserir os seus dados na nossa plataforma. Isso faria com que a mesma ficasse mais completa pois reuniria uma maior quantidade de informação que de outra forma teria de ser acedidas individualmente. Para as outras entidades era também vantajoso pois retirava os custos de terem de desenvolver uma plataforma para o armazenamento e visualização dos seus dados.

Referências

[1] Europa.eu *https://op.europa.eu/webpub/eca/special-reports/air-quality-23-2018/pt/*. Consultado a 23/11/2020

[2] Europa.eu *https://www.eea.europa.eu/themes/air/intro*. Consultado a 23/11/2020

[3] interreg *https://www.interreg-sudoe.eu/gbr/programme/about-interreg-sudoe*. Consultado a 07/01/2021

[4] Science Direct. *https://www.sciencedirect.com/science/article/pii/S0160412016309989*. Consultado a 23/11/2020

[5] NanoSen-AQM DEI. *https://nanosenaqm.dei.uc.pt/*. Consultado a 25/11/2020

[6] NanoSen-AQM EU. *https://www.nanosenaqm.eu/*. Consultado a 25/11/2020

[7] Escala AQI. *https://aqicn.org/scale/pt/*. Consultado a 07/01/2021

[8] Tabela AQI. *https://forum.airnowtech.org/t/aqi-calculations-overview-ozone-pm2-5-and-pm10/168*. Consultado a 07/01/2021

[9] Angular. *https://angular.io/guide/architecture*. Consultado a 22/12/2020

[10] AltexSoft - Angular. *https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/*. Consultado a 22/12/2020

[11] CAQI Air quality index. *https://www.airqualitynow.eu/download/CITEAIR-Comparing_ Urban_ Air_ Quality_ across_ Borders.pdf*. Consultado a 16/01/2021

[12] EAQI Air quality index. *https://www.eea.europa.eu/themes/air/air-quality-index*. Consultado a 16/01/2021

[13] Django. *https://www.djangoproject.com/start/overview/*. Consultado a 21/12/2020

[14] Django *Pros and Cons*. *https://www.netguru.com/blog/django-pros-and-cons*. Consultado a 21/12/2020

[15] Nagios. *https://www.nagios.org/*. Consultado a 22/12/2020

[16] Grafana. *https://grafana.com/*. Consultado a 22/12/2020

[17] Ionic Framework. *https://ionicframework.com/*. Consultado a 21/12/2020

[18] Ionic Framework. *https://ionicframework.com/docs/api/card*. Consultado a 27/11/2020

[19] Kubernetes.io *https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/*. Consultado a 30/11/2020

[20] Ubuntu.com *https://ubuntu.com/kubernetes/what-is-kubernetes*. Consultado a 30/11/2020

[21] Medium. *https://medium.com/@waldyrfelix/6-aspectos-essenciais-para-decidir-entre-aplica%C3%A7%C3%B5es-mobile-h%C3%ADbridas-e-nativas-51bce0dace68*. Consultado a 23/12/2020

[22] Blommidea - Apps Híbridas. *https://bloomidea.com/blog/aplicacoes-nativas-vs-hibridas-qual-escolher-para-o-seu-projeto*. Consultado a 22/12/2020

[23] OpenSource - Docker. *https://opensource.com/resources/what-docker*. Consultado a 04/01/2021

[24] ReactJS. *https://reactjs.org/*. Consultado a 16/01/2021

[25] TeamGantt. *https://www.teamgantt.com/* Consultado a 24/11/2020

[26] Scrum Framework. *https://www.atlassian.com/agile/scrum* Consultado a 24/11/2020

[27] Tuleap - Scrum. *https://www.tuleap.org/agile/agile-scrum-in-10-minutes/*. Consultado a 23/12/2020

[28] Pedro Lucas (2019). *Development of the server for the NanoSen-AQM Project*, University of Coimbra

[29] Docker Swarm. *https://docs.docker.com/engine/swarm/key-concepts/*. Consultado a 29/07/2021

[30] Mapbox GL JS. *https://docs.mapbox.com/mapbox-gl-js/guides/*. Consultado a 24/02/2021

Appendices

Esta página foi intencionalmente deixada em branco.

Apêndice A

Testes

A.1 Resultados

Testes unitários à funcionalidade "*upload de ficheiros*"

Teste	Esperado	Obtido	Resultado
test_csv_insert_valid_non_auth	HTTP_401_UNAUTHORIZED	HTTP_401_UNAUTHORIZED	Pass
test_csv_insert_valid_public_user	HTTP_403_FORBIDDEN	HTTP_403_FORBIDDEN	Pass
test_csv_insert_valid_sensor_owner	HTTP_200_OK	HTTP_200_OK	Pass
test_csv_insert_valid_coordinates	HTTP_200_OK	HTTP_200_OK	Pass
test_csv_insert_invalid_no_time	HTTP_400_BAD_REQUEST	HTTP_400_BAD_REQUEST	Pass
test_csv_insert_invalid_no_sensor_id	HTTP_400_BAD_REQUEST	HTTP_400_BAD_REQUEST	Pass
test_csv_insert_invalid_no_value	HTTP_400_BAD_REQUEST	HTTP_400_BAD_REQUEST	Pass
test_csv_insert_invalid_no_latitude_and_longitude	HTTP_400_BAD_REQUEST	HTTP_400_BAD_REQUEST	Pass
test_csv_insert_invalid_not_found_sensor	HTTP_400_BAD_REQUEST	HTTP_400_BAD_REQUEST	Pass
test_csv_insert_invalid_not_owner_of_sensor	HTTP_403_FORBIDDEN	HTTP_403_FORBIDDEN	Pass
test_raw_csv_insert_valid_non_auth	HTTP_401_UNAUTHORIZED	HTTP_401_UNAUTHORIZED	Pass
test_raw_csv_insert_valid_public_user	HTTP_403_FORBIDDEN	HTTP_403_FORBIDDEN	Pass
test_raw_csv_insert_valid_sensor_owner	HTTP_200_OK	HTTP_200_OK	Pass
test_raw_csv_insert_valid_coordinates	HTTP_200_OK	HTTP_200_OK	Pass
test_raw_csv_insert_invalid_no_time	HTTP_400_BAD_REQUEST	HTTP_400_BAD_REQUEST	Pass
test_raw_csv_insert_invalid_no_sensor_id	HTTP_400_BAD_REQUEST	HTTP_400_BAD_REQUEST	Pass
test_raw_csv_insert_invalid_no_value	HTTP_400_BAD_REQUEST	HTTP_400_BAD_REQUEST	Pass
test_raw_csv_insert_invalid_no_latitude_and_longitude	HTTP_400_BAD_REQUEST	HTTP_400_BAD_REQUEST	Pass
test_raw_csv_insert_invalid_not_found_sensor	HTTP_400_BAD_REQUEST	HTTP_400_BAD_REQUEST	Pass
test_raw_csv_insert_invalid_not_owner_of_sensor	HTTP_403_FORBIDDEN	HTTP_403_FORBIDDEN	Pass

Figura A.1: Resultado esperado e obtido para os testes

A.2 Exemplo de código de um teste unitário

```python
def test_raw_csv_insert_invalid_not_owner_of_sensor(self):

        auth_client, user = authenticate_user_sensor_owner()

        u = UserFactory()
        c = ClusterFactory(owner=u)
        s = SensorFactory(cluster=c)
        s2 = SensorFactory(cluster=c)

        current_datetime = str(datetime.now().astimezone())

        data = [{"time": current_datetime, "sensor_id": int(s.id), "value": "1",
               "parameter": s.parameter.name, "unit": s.unit.name},
               {"time": current_datetime, "sensor_id": int(s2.id), "value": "2",
               "parameter": s2.parameter.name, "unit": s2.unit.name}]

        response = auth_client.put(reverse("user-cluster-rawmeasurements-csv",
                    args=[str(user.id), str(c.id)]), data.encode('utf-8'), data)

        self.assertEqual(response.status_code, status.HTTP_403_FORBIDDEN)
```

Figura A.2: Exemplo de código de um teste unitário