



ISEL – INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA
ADEETC – ÁREA DEPARTAMENTAL DE ENGENHARIA DE
ELECTRÓNICA E TELECOMUNICAÇÕES E DE COMPUTADORES

LEIM

LICENCIATURA EM ENGENHARIA INFORMÁTICA E MULTIMÉDIA

UNIDADE CURRICULAR DE PROJETO

Sistemas de Bases de Dados de Navegação Sea2Future



Pedro Nunes (46322)

Samuel Ventura (46339)

Orientador(es)

Professor Carlos Gonçalves (Instituto Superior de Engenharia de Lisboa)

Professor Mário Assunção (Escola Superior Náutica Infante D. Henrique)

julho, 2021

Resumo

Este trabalho enquadra-se no projeto *Sea2Future* da Escola Náutica Infante D. Henrique, tendo sido desenvolvido com o intuito de ser usado em junção com a embarcação autónoma desenvolvida pelos mesmos, que produz um conjunto de dados que necessitam de ser processados e armazenados localmente e temporariamente.

O processamento é aplicado sobre dados que são extraídos de tópicos publicados pela plataforma *Robot Operating System (ROS)*, que é responsável pelo controlo da embarcação em si, que se encontra em funcionamento no *Raspberry Pi* pertencente à embarcação. Estes dados consistem em várias informações relevantes à embarcação captadas pelos vários sensores da mesma.

Situa-se no seguimento de dois projetos prévios que implementaram o controlo da embarcação de modo remoto. Inicialmente, através de uma aplicação *desktop*. Posteriormente este conceito foi estendido noutra iteração de modo a permitir que o controlo fosse efetuado através de uma aplicação *Android*.

Durante uma viagem autónoma realizada pela embarcação existe um conjunto de dados de telemetria que devem ser guardados pela nossa aplicação a desenvolver. Estes dados devem ser primeiro armazenados localmente e posteriormente enviados para o servidor principal remoto para serem analisados e apresentados.

Consequentemente, este trabalho surge devido ao problema de armazenamento dos dados que são registados pela embarcação num ambiente onde existe armazenamento limitado. Adicionalmente deve, também, ser contemplada uma solução de armazenamento local e *online*, com a devida transmissão de dados quando possível para permitir a gestão de espaço limitado.

Esta aplicação *web* apresenta todos os dados das várias viagens da embarcação de uma forma mais *user-friendly* para facilitar a percepção do significado dos dados e o percurso realizado pela embarcação em si.

Keywords: Bases de Dados, Embarcação Remota, Python, MongoDB

Abstract

This module is part of the *Sea2Future* project from Escola Náutica Infante D. Henrique, and was developed to be used in conjunction with the autonomous craft developed by them that produces a set of data that needs to be processed and stored locally and temporarily.

The processing is applied on data that is extracted from topics published by the platform *Robot Operating System* (**ROS**), which is responsible for controlling the craft itself, that is running on the **Raspberry Pi** belonging to the craft. This data consists of various information relevant to the craft captured by the various sensors on the craft.

It follows on from two previous projects that implemented remote control of the vessel. Initially, through a desktop application. Later, this concept was extended in another iteration to allow the control to be carried out through an *Android* application.

During an autonomous trip made by the vessel there is a set of telemetry data that must be stored by our application to be developed. This data must first be stored locally and then sent to the remote main server to be analysed and displayed.

Consequently, this work arises due to the problem of storing the data that is recorded by the vessel in an environment where there is limited storage. Additionally a local and online storage solution should also be contemplated, with appropriate data transmission where possible to allow for limited space management.

This web application presents all the data of the various trips of the vessel in a more user-friendly way to facilitate the understanding of the meaning of the data and the route taken by the vessel itself.

Keywords: Database, Remote Vessel, Python, MongoDB

Agradecimentos

Aos orientadores, Prof. Carlos Gonçalves e Prof. Mário Assunção, por nos terem guiado e ajudado durante todo o processo desafiante que foi este projeto e por estarem sempre disponíveis. Agradecemos também a oportunidade de poder trabalhar no projeto *Sea2Future*, em colaboração com a Escola Superior Náutica Infante D. Henrique, e ajudar no seu desenvolvimento.

Aos nossos colegas Diogo Ribeiro, Davidson D'Apresentação, Rafael Gonçalves e Rodrigo Matela, gostaríamos de agradecer por sempre estarem presentes e caminharem ao nosso lado ao longo deste percurso que todos realizámos.

Eu, Pedro Nunes, agradeço aos meus pais, Luciano e Paula por serem compreensivos e apoiarem-me sempre ao longo do meu percurso académico e ao meu irmão, Bruno, por me ajudar a adaptar a este novo desafio que agora concluo. Ao meu melhor amigo, Pedro Ferreira, que sempre me apoiou e animou quando precisei.

Eu, Samuel Ventura, gostaria de agradecer aos meus pais, Sérgio e Isabel, por me terem apoiado durante o meu percurso académico. Ao meu irmão, Tiago, por ter estado sempre disposto a ouvir as minhas frustrações, mesmo sabendo que pouco poderia fazer para as resolver. Por fim, aos meus amigos mais próximos, por serem compreensivos e prestáveis durante estes últimos anos.

Índice de Conteúdos

Resumo	i
Abstract	iii
Agradecimentos	v
Índice de Conteúdos	vii
Índice de Tabelas	ix
Índice de Figuras	xi
Índice de Códigos	xiii
Índice de Acrónimos	xvi
1 Introdução	1
2 Trabalho Relacionado	3
2.1 Aplicações Anteriores	3
2.1.1 Desktop BoatCom	4
2.1.2 Android BoatCom	4
2.2 Robot Operating System	5
2.3 OpenSeaMaps	6
3 Modelo Proposto	7
3.1 Requisitos do Projeto	7
3.1.1 Requisitos Funcionais	8
3.1.2 Requisitos não Funcionais	9

3.2	Atributos do Sistema	10
3.3	Casos de Utilização	11
3.4	Abordagem	12
3.5	Interface Gráfica	14
4	Implementação do Modelo	17
4.1	Scripts Python	17
4.1.1	boat_subscribe	17
4.1.2	send_to_database	20
4.2	Base de Dados	23
4.2.1	Funcionalidade	23
4.2.2	Escolha da Base de Dados	23
4.3	Aplicação Web	24
4.3.1	Página Principal	25
4.3.2	Garantir a Autenticação	27
4.3.3	Listagem da Viagens	28
4.3.4	Listagem dos Registos	30
4.3.5	Detalhes de um Registo	41
5	Validação e Testes	45
6	Conclusões e Trabalho Futuro	53

Índice de Tabelas

3.1 Requisitos funcionais associados à embarcação.	8
3.2 Requisitos funcionais associados à aplicação.	9
3.3 Requisitos não funcionais associados à embarcação.	9
3.4 Requisitos não funcionais associados à aplicação.	10
3.5 Tabela de atributos relativos ao projeto.	11

Índice de Figuras

2.1	Estrutura do projeto <i>Desktop BoatCom</i>	4
2.2	Estrutura do projeto <i>Android BoatCom</i>	5
2.3	Diagrama de Funcionamento do ROS [4].	5
3.1	Casos de Utilização da aplicação.	12
3.2	Tipo de dados com o espaço necessário correspondente.	13
3.3	Frequências de registo baseado em prioridades.	13
3.4	Diferentes cenários de funcionamento da embarcação.	14
3.5	Espaço ocupado pelos dados em diferentes períodos.	14
3.6	Mockup da aplicação <i>web</i> a desenvolver.	15
4.1	Identificadores dos diferentes cenários.	19
4.2	Exemplo de documentos na base de dados.	24
4.3	Mensagem de cenário inválido.	25
4.4	Demonstração do formulário de definições de criação do gráfico.	41
5.1	Ecrã da página inicial num cenário de autenticação inválida. .	49
5.2	Ecrã que lista as duas viagens simuladas.	49
5.3	Página de detalhes de uma missão.	50
5.4	<i>Popup</i> para a confirmação da remoção da coleção.	51
5.5	Ecrã que agora lista apenas uma viagens simulada.	51
5.6	Detalhes de um registo e respetivo gráfico.	52
5.7	Gráfico transferido a partir da aplicação <i>web</i>	52

Índice de Códigos

1	Função subscritora do tópico fix (GPS).	18
2	Avaliação de estado da embarcação.	19
3	Excerto de código de armazenamento de dados.	20
4	Função de armazenamento local.	21
5	Função de upload de dados para a MongoDB.	22
6	Verificação da Sessão do Utilizador	25
7	Instância do cliente com credenciais da sessão.	26
8	Teste às credenciais do utilizador e tratamento dos resultados.	26
9	Verificação à sessão para garantir a autenticação	27
10	Acesso à base de dados e recolha das missões existentes.	28
11	Recolha e tratamento da informação de cada missão.	29
12	Recolha da coleção da base de dados através do nome.	30
13	Ciclo <i>foreach</i> realizado a todos os documentos da coleção.	31
14	Porção HTML de cada entrada da tabela.	31
15	Função para converter o formato <i>Decimal</i> em DMS.	32
16	Importação da biblioteca OpenLayers.	34
17	Método onde se instancia a camada do mapa	35
18	Método que instancia os marcadores e a sua camada.	36
19	Pesquisa por <i>Geocode</i> inversa para obter uma localização.	37
20	Criação de um ficheiro em formato <i>Comma-Separated Values</i>	38
21	Envio de um ficheiro por cabeçalho HTTP.	39
22	Método que averigua se o <i>input</i> coincide com a chave.	40
23	Remoção da coleção especificada da base de dados.	40
24	Botão para a transferência da imagem.	43
25	Exemplo de dados que serão <i>uploaded</i>	45
26	Função de armazenamento local.	46
27	Função de armazenamento remoto.	46

28	Documento JSON de exemplo de uma missão.	47
29	Documento CSV resultante do <i>download</i> dos dados de teste. . .	50

Índice de Acrónimos

API *Application Programming Interface.*

BSON *Binary JSON.*

CSV *Comma-Separated Values.*

DMS *Degree Minute Seconds.*

GPS *Global Positioning System.*

HTML *Hypertext Markup Language.*

HTTP *Hypertext Transfer Protocol.*

IMU *Inertial Measurement Unit.*

JSON *JavaScript Object Notation.*

NoSQL *Not Only SQL.*

OSM *OpenSeaMaps.*

PHP *Hypertext Preprocessor.*

ROS *Robot Operating System.*

SQL *Structured Query Language.*

TCP/IP *Transmission Control Protocol/Internet Protocol.*

URL *Uniform Resource Locator.*

Wi-Fi *Wireless Fidelity.*

ENIDH Escola Náutica Infante D. Henrique.

ISEL Instituto Superior de Engenharia de Lisboa.

LEIM Licenciatura em Engenharia Informática e Multimédia.

USV *Unmanned Surface Vehicle.*

Capítulo 1

Introdução

Ao longo dos anos, em paralelo com o desenvolvimento tecnológico do homem, tem surgido o conceito de automação que visa aumentar a produtividade e eficiência, reduzir riscos, entre outros. A área da navegação marítima não é exceção a esta evolução e, consequentemente, também têm surgido grandes evoluções nesta área em relação à automação de sistemas de navegação e, inclusive, embarcações autónomas.

Uma embarcação autónoma pode servir vários propósitos como reduzir o risco associado a expedições marinhas, caso não necessite de uma tripulação, maior possibilidade de exploração de recursos marinhos de maneira remota ou análise de ambientes marítimos semelhantemente de maneira remota.

Com isto em mente, apresenta-se a embarcação USV-*enautica1* (ou *Unmanned Surface Vehicle* - *enautica1*), desenvolvida pela Escola Náutica Infantil D. Henrique (ENIDH), que procura expandir e aprofundar as suas explorações através desta embarcação.

Esta embarcação é constituída por vários módulos e sensores que permitem o funcionamento do sistema e recolha de dados, respetivamente. A embarcação possui sete módulos que inibem várias funções, sendo estes: Sistema de Navegação e Orientação, Sistema de Comunicação, Sistema de Controlo Remoto, Sistema de Detecção de Colisões, Sistema de Detecção de Propulsão, Sistema de Controlo Dinâmico e Sistema de Detecção de energia, sendo todos estes controlados e coordenados por um computador principal.

Este projeto foca-se não necessariamente num dos módulos, mas sim nos sensores de recolha de informação mencionados previamente. Pretende-se implementar um sistema que recolha e armazene os dados para posterior análise que até agora apenas estavam a ser lidos por parte da embarcação, no entanto, não eram armazenados. Em adição visa-se que a solução de armazenamento seja eficiente com o uso da memória disponível internamente na embarcação de forma a que o sistema seja viável numa situação onde se realizam expedições em alto mar de longo prazo.

Estes dados serão extraídos da embarcação através de um *script* Python a correr no dispositivo **Raspberry Pi 3**[13] que, através do *Robot Operating System (ROS)*, irá processar estes dados de maneira a que outro *script* possa simplesmente realizar o seu envio para armazenamento. Por sua vez, este armazenamento pode ser efetuado localmente, tal como foi referido, ou então poderá ser efetuado um *upload* dos dados para uma base de dados remota, de maneira a melhor otimizar o uso de memória limitada e reduzindo assim, a sua carga.

Finalmente, estes dados serão possíveis visualizar numa aplicação *web*, que irá apresentar todos os dados das várias expedições que tenham sido realizadas de uma forma mais organizada e simples, para permitir uma melhor análise do percurso da embarcação e dos seus dados.

Relativamente à organização do relatório, este apresenta-se seccionado em seis capítulos: a Secção 1 apresenta uma breve introdução contextualizando o projeto e explicando-o de forma sucinta, a Secção 2 retrata os trabalhos precedentes e outras tecnologias que serviram como referência para o desenvolvimento do projeto, a Secção 3 establece o planeamento e estruturação realizado previamente ao início da concretização do projeto, a Secção 4 foca-se na concretização do projeto em si, a Secção 5 apresenta os testes realizados para validar o correto funcionamento do projeto e, finalmente, na Secção 6 são apresentadas as conclusões finais acerca do projeto, assim como o trabalho futuro de expansão sobre o mesmo.

Todo o código desenvolvido associado ao projeto está presente num repositório na plataforma de hospedagem de código, **GitHub**, ao qual se pode ter acesso através do seguinte [link](#).

Capítulo 2

Trabalho Relacionado

Este projeto surge na continuidade dos projetos anteriores, *Desktop BoatCom* e *Android BoatCom*, de modo a permitir uma maior usabilidade e utilidade para o utilizador do projeto *Sea2Future* como um todo. Neste capítulo descrevem-se as principais funcionalidades dos trabalhos desenvolvidos anteriormente que se relacionam com o projeto desenvolvido, sendo abordado em vários capítulos as partes mais relevantes ao nosso projeto especificamente, bem como os que se enquadram diretamente no projeto mas numa maior escala. Na Secção 2.1 descrevem-se as aplicações desenvolvidas anteriormente e, na Secção 2.2 apresenta-se de modo simplificado a arquitetura do *Robot Operating System* (ROS).

2.1 Aplicações Anteriores

O projeto de Sistemas de Base de Dados de Navegação surge no seguimento dos projetos desenvolvidos previamente por outros colegas da LEIM em conjunto com a ENIDH. Estes consistiam em possibilitar a comunicação com a embarcação, funcionalidade esta que é essencial ao funcionamento do projeto atual, e, através desta mesma funcionalidade, mover a embarcação. Nas secções 2.1.1 e 2.1.2 apresentam-se, respetivamente, de forma resumida os projeto anteriores, *Desktop BoatCom* e *Android BoatCom*.

A 1^a iteração destes projetos foi a *Desktop BoatCom* que inibia a comunicação e movimentação da embarcação através de uma aplicação em contexto de *desktop* em conjunto com um *joystick*. Posteriormente, este projeto foi melhorado na forma da conceção de uma aplicação *Android*, providenci-

ando, assim, uma maior acessibilidade e portabilidade.

Comparativamente, este projeto irá tomar partido de tecnologias implementadas nestes projetos anteriores, nomeadamente a comunicação para permitir que sejam extraídos os dados essenciais da embarcação durante as suas viagens.

2.1.1 Desktop BoatCom

Este projeto[2] consistiu no desenvolvimento de um módulo que permite o controlo da embarcação de maneira remota. Isto era possibilitado através dumha aplicação *desktop*, que através dum *joystick* comunica os *inputs* para a embarcação via *Robot Operating System* (ROS), possibilitando a sua movimentação. A estrutura que este projeto seguiu encontra-se visível na Figura 2.1

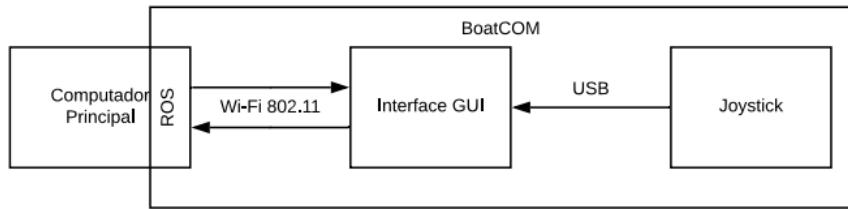


Figura 2.1: Estrutura do projeto *Desktop BoatCom*.

2.1.2 Android BoatCom

Este projeto[1] surgiu no seguimento do *Desktop BoatCom* e com o intuito de fornecer uma iteração do projeto anterior. Em termos de funcionamento apresenta a mesma funcionalidade que o seu predecessor. No entanto, este é de mais fácil acesso em termos de controlo da embarcação, visto que é feito por meio de uma aplicação *Android*, sendo esta solução, em adição, mais portátil que a anterior, dado que o único equipamento necessário é um dispositivo capaz de correr o sistema operativo *Android*.

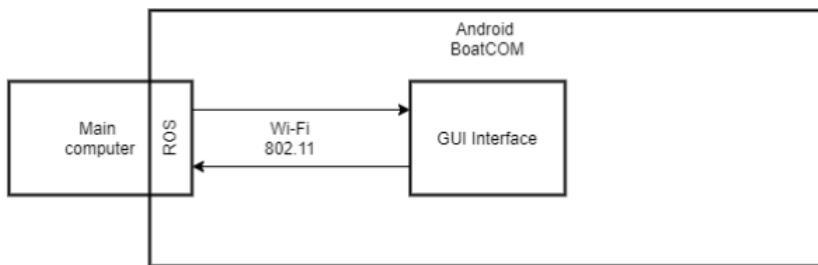


Figura 2.2: Estrutura do projeto *Android BoatCom*.

2.2 Robot Operating System

O *Robot Operating System* (**ROS**) é uma *framework* flexível constituída por várias bibliotecas que facilitam a criação de aplicações para sistemas robóticos para manipular os mesmos.

Mais concretamente aplicado ao projeto, é utilizado para permitir a comunicação com a embarcação, sendo esta possibilitada pelo **ROS** através da arquitetura que este fornece para a troca de mensagens segundo o protocolo TCP/IP. Esta arquitetura funciona através da publicação (escrita) de dados para nós que, para que seja possível a recolha dos dados publicados, necessitam que esteja a ser feita uma subscrição (leitura) por parte de outros nós. Isto pode-se observar através da Figura 2.3.

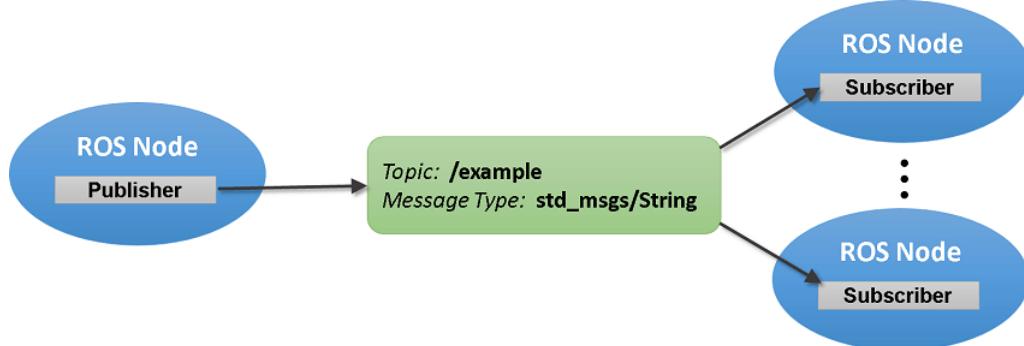


Figura 2.3: Diagrama de Funcionamento do **ROS** [4].

Estes tópicos são de nome único, não podendo haver dois tópicos com o mesmo nome pertencentes à mesma rede **ROS**, têm tipos associados às suas mensagens, semelhantemente aos tipos de variáveis, para especificar que tipo de dados são transmitidos nesse tópico.

Todos os nós pertencentes à mesma rede ROS estão sob o mesmo nó principal, denominado de **ROS master**, que garante a unicidade de nomes nos tópicos publicados e as suas localizações respetivas. Esta localização é guardada para ser fornecida a outros nós que desejam subscrever um determinado tópico pertencente à rede.

2.3 OpenSeaMaps

O *OpenSeaMaps* (**OSM**) é um projeto *open-source* de recolha de dados náuticos de utilização livre. Estes dados podem ser utilizados de modo a criar cartas náuticas que podem ser utilizadas em diversas aplicações por parte de múltiplas *frameworks* distintas.

”OpenSeaMap is an open source, worldwide project to create a free nautical chart. There is a great need for freely accessible maps for navigation purposes... The goal of OpenSeaMap is to record interesting and useful nautical information for the sailor which is then incorporated into a free map of the world.” [9]

No âmbito do projeto, este *software* é utilizado para apresentar num mapa interativo os diversos registos efetuados pela embarcação numa determinada viagem. Dado que a aplicação a utilizar é desenvolvida com um *browser* em mente, a *framework* aplicada para disponibilizar interatividade às cartas náuticas foi a *OpenLayers*[5]. Esta biblioteca JavaScript, também *open-source*, oferece uma *Application Programming Interface* (API) extensiva capaz de criar um ambiente geográfico interativo. Este ambiente tem a capacidade de desenhar sobre várias camadas as cartas náuticas, disponibilizadas pelo *OpenSeaMaps*, assim como outras componentes fornecidas pela biblioteca, como marcadores de pontos de interesse.

Capítulo 3

Modelo Proposto

Ao longo deste capítulo será abordado o planeamento realizado para estruturar e guiar o projeto.

Este capítulo será subdividido em cinco partes. Na Secção 3.1 serão abordados ambos os requisitos funcionais e não funcionais relativos ao projeto; a Secção 3.2 elabora sobre os atributos do sistema que caracterizam o mesmo. Posteriormente, na Secção 3.3 fala-se brevemente sobre os casos de utilização do projeto, que permite ter uma melhor noção da função e utilidade deste.

Na Secção 3.4 serão explicados os cálculos realizados para o dimensionamento da solução de armazenamento e as suposições associadas a esta. Finalmente, na Secção 3.5 serão apresentados os *mockups* criados para iniciar a prototipagem da aplicação *web* associada ao projeto.

3.1 Requisitos do Projeto

Os requisitos funcionais e não funcionais representam o conjunto das funcionalidades que o projeto deverá implementar. Os requisitos funcionais definem o conjunto de funcionalidades que o sistema desempenha, cada um representado com a sua respetiva referência. Consoante as suas funções, estes encontram-se categorizados em três categorias: evidente, invisível ou adornos. Os requisitos evidentes são aqueles cuja sua utilização no projeto é visível para o utilizador, contrastando com os requisitos invisíveis que têm o propósito contrário. Ambas as categorias são, no entanto, de implementação obrigatória, ao contrário da categoria dos adornos cuja implementação é opcional dado que a sua existência resume-se a funcionalidades menos signifi-

cantes. Os requisitos não funcionais representam a maneira como o sistema realiza cada uma das suas funcionalidades. Assim como os requisitos funcionais, estes podem ser agrupados nas mesmas categorias: evidente, invisível e adornos.

Na Secção 3.1.1 demonstra-se o conjunto de requisitos funcionais associados ao projeto, enquanto na Secção 3.1.2 o conjunto de requisitos não funcionais.

3.1.1 Requisitos Funcionais

A tabela 3.1 dispõem do conjunto de requisitos funcionais aplicados à embarcação e a tabela 3.2 os requisitos associados à aplicação gráfica a implementar.

Tabela 3.1: Conjunto de requisitos funcionais associados aos componentes da embarcação.

Ref.#	Função	Categoria
R1.1	Registo e processamento dos dados fornecidos pelo veículo.	Invisível
R1.2	Armazenar dados registados numa base de dados local ou remota.	Invisível
R1.3	Criar canal de comunicação veículo-servidor para envio de dados.	Invisível
R2.1	Histórico de viagens com respetivos dados registados e rota realizada.	Invisível

O sistema a implementar tem de ser capaz de processar os dados obtidos (RF1.1) e registá-los numa base de dados local à embarcação, ou numa base de dados remota num servidor (RF1.2), caso exista a possibilidade de se ligar à *internet*. Desta forma, é necessário contemplar também a existência de uma canal de comunicação entre a embarcação e o servidor (RF1.3) de modo a transferir os dados. A existência de várias viagens cria a necessidade de organizar os dados em estruturas ordenadas cronologicamente. Desta forma, a estruturação dos dados adquiridos devem de ser estruturados de forma a originar um histórico para os mesmos (RF2.1).

Tabela 3.2: Conjunto de requisitos funcionais associados as componentes da aplicação.

Ref.#	Função	Categoria
R3.1	Listar os dados da viagem na aplicação <i>web</i> utilizando tabelas ou gráficos.	Evidente
R3.2	Registar rota num mapa interativo com base nos dados do GPS.	Evidente
R4.1	Autenticação de utilizador (<i>User/Password</i>).	Invisível

Os dados obtidos do histórico de viagens são listados na aplicação gráfica (RF3.1), juntamente com as coordenadas *Global Positioning System (GPS)* da rota realizada, que é desenhada num mapa interativo (RF3.2). Esta aplicação necessita de um sistema de autenticação de utilizador, através da introdução de um nome e palavra-chave. Isto possibilita a autenticação do utilizador de forma a verificar se este é, ou não, válido para o visionamento dos dados registados (RF4.1).

3.1.2 Requisitos não Funcionais

A tabela 3.3 apresenta os requisitos não funcionais que dizem respeito às componentes relativas à embarcação, enquanto a tabela 3.4 representa os requisitos não funcionais relativos ao lado da aplicação a complementar o projeto.

Tabela 3.3: Conjunto de requisitos não funcionais associados à embarcação.

Ref.#	Função	Categoria
R1	Utilização de um script Python para o processamento e envio dos dados para uma base de dados NoSQL.	Invisível
R2	A comunicação deve ser feita com Wi-Fi e a ligação feita diretamente ao ROS.	Invisível

É vital a utilização de um *script* capaz de adquirir os dados de telemetria provenientes dos sensores da embarcação, de modo a processá-los para o

posterior envio para uma base de dados que não recorre a *Structured Query Language (SQL)* (RNF1). Dada a necessidade de se enviar os dados para uma base de dados, é preciso realizar uma ligação direta através do Wi-Fi, entre o ROS presente na embarcação e a base de dados que será armazenada num determinado servidor (RNF2).

Tabela 3.4: Conjunto de requisitos não funcionais associados à aplicação.

Ref.#	Função	Categoría
R3	Estruturação da base de dados de modo a que cada conjunto de documentos siga uma estrutura cronológica de viagens para recolha de dados.	Invisível
R4	Aplicação Web à qual o utilizador se autentica para o visionamento da informação de cada viagem presente na base de dados remota.	Evidente

A estruturação da base de dados é importante pois necessita de apresentar uma estrutura que permite a disposição dos dados recolhidos de forma cronológica e organizada, de modo a facilitar a criação, por exemplo, do histórico de navegação (RNF3). O visionamento destes dados é efetuado através a aplicação web à qual o utilizador válido acede para visualizar, de forma intuitiva, os dados registados pela embarcação (RNF4).

3.2 Atributos do Sistema

Os atributos definem as características ou dimensões de um sistema. Os atributos dispõem também de um conjunto de classificações que depende do tipo de característica que desempanham no projeto, podendo ser classificados como: "Plataforma", para qualidades a nível das ferramentas a utilizar; "Interação homem-máquina", para propriedades relativas à utilização das ferramentas por parte do utilizador; "Facilidade de Utilização", como o nome indica, características que facilitam a utilização do sistema; ou "Tolerância a Falhas", para características que permitem fortalecer o sistema. Para além disso, estes podem ser categorizados como "Obrigatórios" se tiverem de ser contemplados ou "Desejável" para características alcançaveis mas que não

necessitam de ser obrigatoriamente atingidas.

A tabela 3.5 define o conjunto de atributos a aplicar a este projeto.

Tabela 3.5: Tabela de atributos relativos ao projeto.

Atributo	Detalhe / Restrição Fronteira	Categoria
Plataforma	Microcontrolador para processar, publicar e subscrever tópicos ROS.	Obrigatório
Interação homem-máquina	Aplicação Web que permite demonstrar a informação registada na base de dados.	Obrigatório
Interação homem-máquina	Gráficos e diagramas na app para visualização.	Desejável
Facilidade de Utilização	Base de dados em NoSQL para armazenar dados.	Obrigatório
Facilidade de Utilização	Serviço de mapas para registo de rotas no mapa da interface gráfica.	Desejável
Tolerância a Falhas	Módulos externos de armazenamento adicional para evitar potenciais lotações de memória.	Desejável

3.3 Casos de Utilização

Os casos de utilização descrevem, de forma visual, o funcionamento de determinados elementos do sistema a desenvolver. São caracterizados pela existência de um ou mais atores (entidades intervenientes) que desempenham um determinado conjunto de funcionalidades na utilização do sistema.

No contexto do projeto, o sistema vai apresentar como atores a embarcação, a plataforma gráfica e um utilizador. A embarcação, que contém o *script*, é capaz de recolher e processar os dados publicados, saber armazenar os dados localmente e enviar os dados para um servidor externo. A aplicação gráfica utiliza os dados do servidor para os apresentar, de forma gráfica e intuitiva. Consequentemente, o utilizador é capaz de aceder à aplicação de modo a visualizar os dados registados pela embarcação.

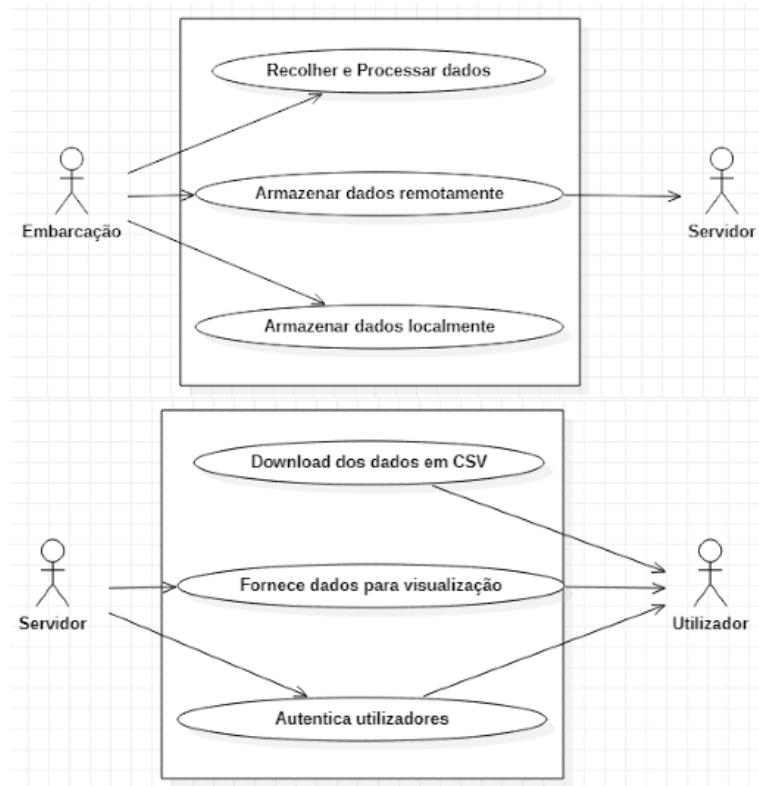


Figura 3.1: Casos de Utilização da aplicação.

3.4 Abordagem

Nesta secção aborda-se o tipo de formulações que se efetuaram de maneira a ter uma melhor noção do seguimento a tomar no desenvolvimento do/a modelo/estrutura proposto/a, permitindo adicionalmente, que seja possível ter uma análise mais crítica das decisões tomadas pelo grupo no seguimento do desenvolvimento do projeto.

De modo a guiar as opções de armazenamento, foi realizado um estudo inicial para obter uma estimativa da capacidade de armazenamento necessária. Para tal, elaborou-se uma folha de cálculo onde foram colocadas todas as variáveis que representam os tópicos gerados pela telemetria da embarcação, bem como o espaço (em *bytes*) ocupado por cada variável (Figura 3.2).

Dados	Tipo de Variável	Tipo de Variável (MongoDB)	Espaço ocupado (bytes)
GPS	string	string	40
IMU	float3	number	8
AIS	string	string	40
Corrente e Tensão Elétrica	float3	number	8
Água no Casco	float3	number	8

Figura 3.2: Tipo de dados com o espaço necessário correspondente.

De seguida, foram considerados vários cenários de funcionamento da embarcação que descreviam a frequência de amostragem das diversas variáveis. Estas frequências de amostragem foram establecidas por patamares de prioridade de armazenamento, como visível na Figura 3.3

Prioridade de registo	Registo/s
Very High	5
High	15
Medium	45
Low	120

Figura 3.3: Frequências de registo baseado em prioridades.

Posto isto, foram criados cinco cenários de funcionamento relativo ao estado da embarcação para que sejam apenas armazenados com maior frequência os dados mais relevantes, otimizando ao máximo o gasto de memória. Estes cinco cenários são os que se podem observar na Figura 3.4.

Nesta figura podem ser observados os níveis de prioridades de registo establecidos para a telemetria dependendo do estado da embarcação, assim como o consumo de cada um dos dados por minuto e a soma total destes num período de um minuto.

Para o resto dos cálculos que serão efetuados de forma a determinar os requisitos de memória será utilizado o cenário negativista, que implica leituras constantes de cada um dos dados presentes. Desta forma, permite-nos dimensionar a nossa solução de armazenamento de maneira mais conservadora. Neste cenário estima-se que irá ser necessário 1248 *bytes* de memória por cada minuto de funcionamento da embarcação.

De seguida, seguindo as condições do cenário negativista, extrapolou-se o consumo de memória em vários períodos de tempo, nomeadamente, uma hora, dia, mês e ano. Estes resultados estão presentes na Figura 3.5

Ocasião	Descrição Exemplo	Dados	Prioridade de registo	Consumo/min	Sum
Cenário 1	Movimento do Veículo	GPS	Very High	480	580
		IMU	Medium	10,7	
		AIS	Medium	53,3	
		Corrente e Tensão Elétrica	High	32	
		Água no Casco	Low	4	
Cenário 2	Movimento Lento/Parado	GPS	Low	20	52
		IMU	Low	4	
		AIS	Low	20	
		Corrente e Tensão Elétrica	Low	4	
		Água no Casco	Low	4	
Cenário 3	Arranque/Aceleração ou Paragens rápidas	GPS	High	160	409,3333333
		IMU	Very High	96	
		AIS	Medium	53,3	
		Corrente e Tensão Elétrica	Very High	96	
		Água no Casco	Low	4	
Cenário 4	Deriva do Veículo	GPS	Very High	480	744
		IMU	Very High	96	
		AIS	High	160	
		Corrente e Tensão Elétrica	Low	4	
		Água no Casco	Low	4	
Cenário Negativista	Tudo a registrar o mais frequentemente possível	GPS	Very High	480	1248
		IMU	Very High	96	
		AIS	Very High	480	
		Corrente e Tensão Elétrica	Very High	96	
		Água no Casco	Very High	96	

Figura 3.4: Diferentes cenários de funcionamento da embarcação.

Por hora	Por dia	Por semana	Por mês	Por ano	
74880	1797120	12579840	53913600	646963200	Bytes
0,00007	0,002	0,01	0,05	0,60	GB

Figura 3.5: Espaço ocupado pelos dados em diferentes períodos.

3.5 Interface Gráfica

Atendendo à necessidade de desenvolver uma aplicação *web*, foi inevitável a realização de um planeamento da mesma. Desta forma, concebeu-se um *mockup* através da ferramenta **Figma**[7]. Um *mockup* representa um modelo de um conceito de grande escala que ainda não foi concebido, de modo a demonstrar a maneira como se vai apresentar assim como as suas operações.

Na Figura 3.6, encontra-se alguns dos ecrãs realizados para os *mockups*. O primeiro ecrã apresentado dispõe de um formulário para o utilizador colocar as credenciais da sua conta. Adicionalmente, contém também um pequeno resumo do projeto assim como links auxiliares para mais informação. Ao iniciar sessão, o utilizador é redirecionado para o segundo ecrã que apresenta uma lista de missões realizadas. Cada missão dispõe de um pequeno mapa, assim como outras informações que identificam a viagem realizada, como por

exemplo a data de realização. A cada missão é possível visualizar os seus detalhes, sendo o utilizador redirecionado para o terceiro ecrã que dispõem uma lista com as múltiplas posições registadas em cada instante de tempo. Cada uma destas posições da tabela é interativa, conduzido o utilizador a uma página que apresenta as informações desse instante assim como a capacidade de gerar um gráfico com base nos dados registados para facilitar a visualização e a evolução dos mesmos a cada instante.

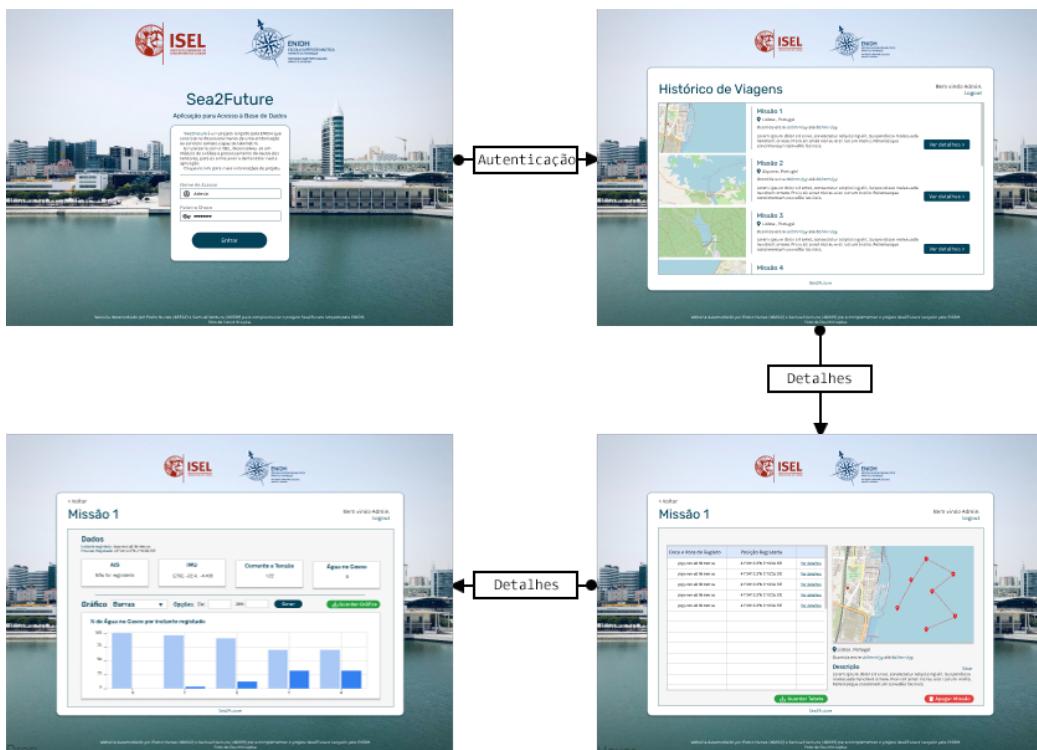


Figura 3.6: Mockup da aplicação web a desenvolver.

Capítulo 4

Implementação do Modelo

Ao longo deste capítulo será abordada a metodologia e implementação por trás de cada fase do projeto, assim como o raciocínio que resultou nas escolhas tomadas para a sua realização.

Este capítulo será subdividido em três partes. A Secção 4.1 elabora sobre o desenvolvimento dos *scripts Python*[12] que interagem com o ROS e realizam a extração e armazenamento de dados. Posteriormente, na Secção 4.2 será tratado a escolha de base de dados de suporte ao projeto e o seu papel no mesmo. Finalmente, na Secção 4.3 será expandido o desenvolvimento da aplicação *web* de suporte visual ao utilizador, assim como a sua ligação à base de dados.

4.1 Scripts Python

Nesta secção, tal como mencionado, será abordado o desenvolvimento dos dois *scripts* de Python que realizam a extração da informação do ROS da embarcação (Secção 4.1.1), assim como o armazenamento dos dados correspondentes numa base de dados local ou remota (Secção 4.1.2).

4.1.1 boat_subscribe

O primeiro *script* a ser desenvolvido, denominado `boat_subscribe.py`, na sua essência, implementa vários subscritores ROS, através da biblioteca `rospy`[14], que fornece as funcionalidades de cliente necessárias para o Python comunicar com o ROS. Estes subscritores correspondem aos tópicos que estão a ser

publicados pela embarcação, relativos aos dados dos vários sensores que esta possui.

Nestas funções subscritoras é realizada constantemente a leitura das informações publicadas, no tópico que lhe corresponde, ao ritmo que estas são publicadas. No entanto, estas não são armazenadas a este ritmo, apenas são consumidas, visto que se pretende ter um melhor controlo sobre o ritmo de armazenamento, neste ponto temporário, destas variáveis.

Posto isto, foram implementados *timestamps* de controlo para este armazenamento temporário inerente ao *script* que será realizado através de um dicionário. Estes *timestamps* (Figura 3.3) são flexíveis, baseando-se nos cenários de funcionamento da embarcação (Figura 3.4) que foram estabelecidos no capítulo anterior. Estes são responsáveis, tal como foi referido, por condicionar o armazenamento dos dados num dicionário antes do envio dos mesmos para armazenamento permanente, local ou remoto.

Código 1: Função subscritora do tópico fix (GPS).

```

1 def fix_callback(data):
2     global data_dict
3     global fix_timestamp
4     global prev_GPS
5     global curr_GPS
6     prev_GPS = curr_GPS
7     rospy.loginfo(rospy.get_caller_id() + "Latitude: %9.6f"
8             ↪, data.latitude)
9     rospy.loginfo(rospy.get_caller_id() + "Longitude: %9.6f"
10            ↪", data.longitude)
11     curr_GPS = (data.latitude, data.longitude)
12     if time() - fix_timestamp >= fix_threshold:
13         fix_timestamp = time()
14         data_dict["fix"] = (data.latitude, data.longitude)

```

Para obter os valores dos *timestamps* de controlo, deve ser avaliado o estado atual da embarcação. Como tal, foi criada uma função que avalia o estado atual da embarcação através dos dados que foram identificados como mais discerníveis do cenário atual. Estes dados são a corrente e posição GPS que, como se pode observar na figura 4.1, são utilizados de forma diferente, dependendo do cenário em questão. Devido à necessidade de comparar duas instâncias temporais distintas, em adição ao armazenamento em dicionário, armazena-se também as instâncias atuais e anteriores da corrente e do GPS. Após ser determinado o cenário de funcionamento atual, procede-se à alteração dos *timestamps* e, posteriormente, à recolha da informação.

Cenário	Identificadores
Movimento	Diferença entre posições GPS > 2m
Parado	Corrente baixa (< 2A) & Diferença entre posições GPS <= 2m
Aceleração	Picos de corrente (> 5A)
Deriva	Corrente baixa & Diferença entre posições GPS > 2m

Figura 4.1: Identificadores dos diferentes cenários.

Código 2: Avaliação de estado da embarcação.

```

1 def check_boat_status():
2     global prev_A
3     global curr_A
4     global prev_GPS
5     global curr_GPS
6
7     if math.sqrt((curr_GPS[0] - prev_GPS[0])**2 + (curr_GPS
8         [1] - prev_GPS[1])**2) > 2:
9         change_scenario("scenario1")
10    elif curr_A <= 2 and math.sqrt((curr_GPS[0] - prev_GPS
11        [0])**2 + (curr_GPS[1] - prev_GPS[1])**2) <= 2:
12        change_scenario("scenario2")
13    elif abs(curr_A - prev_A) > 5:
14        change_scenario("scenario3")
15    elif curr_A <= 2 and math.sqrt((curr_GPS[0] - prev_GPS
16        [0])**2 + (curr_GPS[1] - prev_GPS[1])**2) > 2:
17        change_scenario("scenario4")

```

Tal como já foi referido, o armazenamento temporário dos dados no *script* antes destes serem enviados para uma das bases de dados, é realizado através de um dicionário Python. Este dicionário possui uma chave correspondente a cada tópico que deve ser armazenado e guarda sempre apenas uma instância de cada tópico em correspondência.

Finalmente, a entrega dos dados para o *script* responsável pelo envio e armazenamento de dados é efetuado na função subscritora do tópico *upload*, que em adição a permitir ao utilizador controlar se deseja que, caso seja possível, os dados sejam guardados na base de dados remota, ou no tópico *signal_quality* para garantir o armazenamento local da informação.

4.1.2 send_to_database

O segundo *script* a ser desenvolvido, denominado `send_to_database.py`, implementa as funcionalidades de criação de ficheiros locais para armazenamento de dados, armazenamento de dados nesses mesmos ficheiros e *upload* de dados para a base de dados remota quando possível.

Quando a função de envio dos dados para armazenamento é invocada pela primeira vez, é determinado primeiramente o número da missão atual para que seja criado o ficheiro local de armazenamento e que seja possível aceder à coleção correspondente na base de dados remota. Para isto é realizada uma procura na pasta que contém os ficheiros das missões passadas para verificar o número mais baixo disponível. Após isto procede-se à criação do ficheiro JSON para acomodar o dicionário de informações que irá receber.

Com a criação do ficheiro concluída, procede-se a verificar se o utilizador deseja que seja feito *upload* dos dados para a base de dados remota e se as condições atuais o permitem. Para isto analisa-se o valor do dicionário que contém a qualidade do sinal de conexão à internet. Este valor será entre 0 e 100 e, caso seja inferior a 30, deduz-se que a conexão não é forte o suficiente e como tal, podem existir quebras no envio (Código 3), optando-se então por realizar armazenamento local dos dados.

Código 3: Excerto de código de armazenamento de dados.

```

1 if upload_to_cloud and not bad_connection:
2     # armazenar remotamente
3     # recolher os dados que estejam guardados
        ↪localmente

```

```

4     if os.path.exists("tmp/mission" + str(mission_num)
5         ↪+ ".json"):
6         file_path = "tmp/mission" + str(mission_num) +
7             ↪".json"
8         with open(file_path) as json_file:
9             stored_data = json.load(json_file)
10            stored_data = stored_data["mission_details"]
11                ↪]
12            # envio dos dados locais
13            for entry in stored_data:
14                upload_data(entry, mission_num)
15            # envio dos novos dados
16            upload_data(data, mission_num, codename,
17                ↪description)
18            # esvaziar o ficheiro local para otimizar espaco
19            create_json("tmp/mission" + str(mission_num) + "."
20                ↪json)
21        else:
22            # armazenar localmente
23            file_path = "tmp/mission" + str(mission_num) + "."
24                ↪json"
25            update_json(data, file_path)

```

Na eventualidade que a conexão seja considerada instável como foi estabelecido previamente, mesmo que o utilizador tenha indicado que deseja fazer *upload* para a *cloud*, apenas lhe é permitido que seja realizado o armazenamento localmente. Neste processo é acrescentado ao final do ficheiro local os novos dados recebidos, fazendo parte do array JSON que irá conter os dados relativos a essa missão (Código 4)

Código 4: Função de armazenamento local.

```

1 def update_json(new_data, file_name):
2     with open(file_name, 'r+') as file:
3         file_data = json.load(file)
4         file_data["mission_details"].append(new_data)
5         file.seek(0)
6         json.dump(file_data, file, indent=4)

```

Caso contrário em que a conexão seja forte e tenha sido recebida a indicação que deve ser efetuado o *upload*. Realiza-se o envio de todos os dados que foram recolhidos que estavam em armazenamento local sendo, após

este concluir, realizado o envio dos novos dados. Para permitir estes envios realiza-se o estabelecimento de conexão à base de dados com o auxílio da biblioteca `pymongo`[11] que permite o acesso direto à coleção correspondente à nossa missão atual para que sejam depositados os dados.

Código 5: Função de upload de dados para a MongoDB.

```

1 def upload_data(data_to_upload, mission_num, codename=None,
2     ↪ description=None):
3     # ligacao ao MongoDB
4     client = MongoClient(
5         "mongodb+srv://admin:conchinha123@clusterisel.ksoyd
6             ↪.mongodb.net/myFirstDatabase?retryWrites=true&
7             ↪w=majority")
8
9     my_db = client["BoatTelemetryDB"]
10
11    if codename and description is not None:
12        # se forem os dados novos
13        data_to_send = {"Timestamp": "" + data_to_upload[""
14            ↪reg_time"], "GPS": data_to_upload["fix"],
15            "IMU": (data_to_upload["hdg"],
16                ↪data_to_upload["pitch"],
17                ↪data_to_upload["roll"]),
18            "AIS": data_to_upload["AIS"], "Corrente
19                ↪": data_to_upload["current"],
20            "Voltagem": data_to_upload["voltage"],
21            "Water": data_to_upload["humidity"],
22            "Codename": codename,
23            "Description": description}
24    else:
25        # se forem os dados locais
26        data_to_send = {"Timestamp": "" + data_to_upload[""
27            ↪reg_time"], "GPS": data_to_upload["fix"],
28            "IMU": (data_to_upload["hdg"],
29                ↪data_to_upload["pitch"],
30                ↪data_to_upload["roll"]),
31            "AIS": data_to_upload["AIS"], "
32                ↪Corrente": data_to_upload[""
33                    ↪current"], }
```

```
25         "Voltagem": data_to_upload["voltage"
26             ↘],
27         "Water": data_to_upload["humidity"
28             ↘]}
my_collection.insert_one(dataToSend)
```

4.2 Base de Dados

Nesta secção será tratada a funcionalidade, assim como a razão que fundamenta a escolha da base de dados, MongoDB[6], em relação às outras alternativas possíveis baseado no contexto do problema a abordar assim como as suas vantagens.

4.2.1 Funcionalidade

Neste projeto é essencial a existência de uma base de dados remota de maneira a fornecer uma solução de armazenamento escalável a longo prazo que não gaste os recursos limitados da embarcação.

Como tal, esta base de dados está sempre pronta a receber novas missões e dados correspondentes a cada instante, provinda da embarcação que irá efectuar esta comunicação de envio de dados através dos *scripts Python* criados (Secção 4.1).

Posteriormente, estes dados armazenados serão utilizados por parte da aplicação *web* (Secção 4.3) desenvolvida para que esta informação seja demonstrada de maneira mais tangível, simplificada e compilada, para que o utilizador consiga tirar o máximo proveito dos mesmos para análise.

4.2.2 Escolha da Base de Dados

Em termos de requisitos para a escolha da base de dados, visto que não existe necessidade de criar relações entre dados, ou entre tabelas, dado que para cada missão apenas existe uma tabela, a utilização de bases de dados *Structured Query Language (SQL)*, ou bases de dados relacionais, seria redundante.

Em adição, as bases de dados *Not Only SQL (NoSQL)* têm capacidade de suportar grandes quantidades de dados (devido a precisar de menos armaze-

namento para guardar a mesma quantidade de dados, devido à inexistência de objetos SQL, sendo assim mais facilmente escalável), apresenta custos mais baixos na eventual necessidade de escalar a solução de armazenamento e não necessita de qualquer tipo de modelação devido à sua natureza mais flexível em relação às estruturas de dados.

Com estes requisitos em mente, procedeu-se à escolha de uma base de dados NoSQL, sendo escolhida a base de dados MongoDB. Esta escolha deve-se em parte à sua popularidade, pois isto implica a existência de uma comunidade maior em termos de suporte, complementado por compatibilidade com os formatos JSON e BSON, com os quais o grupo já é familiar e permite uma fácil implementação em junção com Python e PHP, que são as duas línguagens de programação que suportam o funcionamento essencial do projeto.

Na figura 4.2 podemos observar um exemplo de uma coleção da base de dados MongoDB que contém dois documentos correspondentes a duas leituras de teste que foram submetidas para armazenamento.

```
_id: ObjectId("60f07a94392305dac4ace6c5")      _id: ObjectId("60f07b051319ba443fc0ad9")
Timestamp: "2021/07/15, 19:01:35"            Timestamp: "2021/07/15, 19:14:28"
  ↴ GPS: Array                                     ↴ GPS: Array
    0: 38.691498333333335                         0: 38.6915
    1: -9.298588333333333333                      1: -9.31
  ↴ IMU: Array                                     ↴ IMU: Array
    0: 358                                         0: 358
    1: -8.520272254943848                         1: -8.520272254943848
    2: 0                                           2: 0
  AIS: ""                                         AIS: ""
  Corrente: 32                                     Corrente: 32
  Voltagem: 123                                     Voltagem: 123
  Water: 2                                         Water: 2
```

Figura 4.2: Exemplo de documentos na base de dados.

4.3 Aplicação Web

Nesta secção, será relatado o desenvolvimento da aplicação *web* e das diversas páginas que a concretizam. Cada página será acompanhada por exemplos ou excertos de código que complementam a explicação da sua implementação.

4.3.1 Página Principal

A página principal é separada em duas componentes PHP distintas: o ficheiro `authentication.php` com a página HTTP a fornecer ao utilizador e o ficheiro `processAuthentication.php` que realiza o processamento da informação, introduzida pelo utilizador, no lado do servidor.

Página de Autenticação

Para a página principal, `authentication.php`, criou-se um formulário com dois *inputs* capazes de receber a informação relativa ao nome e à palavra-chave do utilizador. Ao submeter este formulário, é realizado um `POST` HTTP para a página `processAuthentication.php` do servidor, que dispõe da aplicação, de modo a processar a informação e comparar com a base de dados, de maneira a validar os dados introduzidos.

Se por acaso o utilizador já tiver uma sessão iniciada, este é automaticamente redirecionado para o processamento sem necessitar de realizar novamente o *login* com as suas credências. Este processo é realizado verificando se variável `$_SESSION` dispõem da informação que caracteriza o utilizador (Código 6).

Código 6: Verificação da Sessão do Utilizador

```
1 if ( isset($_SESSION["s2f-credentials"]) ) {  
2     header("Location: processAuthentication.php");  
3 }
```

Caso o utilizador tenha falhado a autenticação, ao retornar à página este é informado acerca da falha na introdução das suas credências, como demonstra a Figura 4.3.

O nome ou palavra-chave introduzidos estão incorretos.
Verifique se estes foram escritos corretamente.

Entrar

Figura 4.3: Mensagem demonstrada ao utilizador num cenário de autenticação inválida.

Dada a natureza do projeto, não é possível realizar o registo de utilizadores dado que o acesso à informação é apenas permitida aos profissionais que fazem parte do projeto *Sea2Future*. Desta forma, a introdução de novos utilizadores tem que ser realizada de forma manual para a base de dados, presente no *Atlas* da MongoDB.

Processamento da Autenticação

Para o processamento da informação, realizado através do ficheiro `process Authentication.php`, recorre-se à biblioteca MongoDB e a sua respetiva extensão para instanciar um cliente `MongoClient` capaz de receber uma *string* com a localização da base de dados no serviço *Atlas* da MongoDB (Código 7). Este objeto permite comunicar com a base de dados, hospedada no serviço, e analisar a informação armazenada nas diversas coleções e documentos que são criados pela embarcação no ato de *upload*.

Código 7: Criação da instância do cliente, fornecendo as credenciais armazenadas em sessão.

```

1 // import MongoDB library
2 require_once("libs/importMongo.php");
3 // instantiates a new client
4 $client = new \MongoDB\Client(
5     "mongodb+srv://". $username . ":" . $password . "
    ↪@cluster0sel.ksoyd.mongodb.net/myFirstDatabase
    ↪?retryWrites=true&w=majority");

```

De modo a realizar a autenticação do utilizador (Código 8), coloca-se no endereço as credenciais introduzidas e tenta-se realizar uma *query* básica para averiguar se o utilizador que se autenticou é válido para aceder à base de dados. Caso o utilizador não seja válido, este é retornado para a página principal através do *catch* da exceção `AuthenticationException`. Caso contrário, os seus dados são armazenados numa `$_SESSION` e este é redirecionado para a página seguinte.

Código 8: Teste às credenciais do utilizador para averiguar se são válidas e posterior tratamento dos resultados.

```

1 try {
2     $client->listDatabaseNames();

```

```

3   // starts a session with the user credentials
4   $_SESSION["s2f-credentials"] = array(
5     "username" => $username,
6     "password" => $password
7   );
8   $nextUrl = $sucessAuthURL;
9 } catch(MongoDB\Driver\AuthenticationException $e) {
10   echo "<br> $e";
11   $_SESSION["failed"] = true;
12   $nextUrl = $failedAuthURL;
13 }
```

4.3.2 Garantir a Autenticação

A necessidade de garantir que um utilizador se encontre autenticado para aceder às páginas privadas levou à origem do ficheiro `ensureAuthentication.php`. Este ficheiro averigua se existe um utilizador armazenado em sessão, analisando se esta dispõe do parâmetro `username` na sua sessão (indicativo de que existe uma sessão iniciada). Se este não existir, é gravado o endereço do local que o utilizador tentou aceder e este é redirecionado para a página de autenticação (Código 9). Ao autenticar-se com sucesso, o utilizador é redirecionado para o endereço pedido (e que foi guardado) em vez do endereço utilizado por omissão.

Através desta análise às variáveis de sessão, é possível garantir o acesso exclusivo às páginas privadas apenas aos utilizadores cujo acesso seja válido.

Código 9: Verificação à sessão do utilizador para garantir a sua autenticação.

```

1 // requested url
2 $uri = filter_input( INPUT_SERVER, 'REQUEST_URI',
3   FILTER_SANITIZE_URL, $flags );
4 // if the user is not set, redirect to the
5 // authentication
6 // and stores the location
7 if ( !isset( $_SESSION[ 'username' ] ) ) {
8   $_SESSION[ 'locationAfterAuth' ] = $uri;
9   header( "Location: authentication.php" );
  exit;
```

Dadas as capacidades modulares do PHP, de modo a usufruir desta garantia de autenticação, criada por este *script*, é necessário assegurar a sua inclusão (`require_once`) em todos os restantes ficheiros à qual queremos garantir a autenticação do utilizador, ou seja, em todas aquelas páginas que forem de visionamento privado.

4.3.3 Listagem da Viagens

Nesta página, dispõem-se uma lista com as múltiplas viagens realizadas pela embarcação. Como descrito anteriormente nos *mockups*, cada viagem é acompanhada por um conjunto de informação que a define. Esta informação é retirada dos documentos contídos na base de dados MongoDB, acedida através do cliente.

Como referido na Secção 4.1.2, cada viagem realizada pela embarcação constitui uma coleção e cada coleção contém um determinado número de documentos que dizem respeito a uma leitura realizada. Deste modo, para listar todas as viagens realizadas, é necessário listar todas as coleções existentes.

Desta forma, utilizando o cliente, cria-se uma instância `MongoDB\Database` da base de dados `BoatTelemetryDB`, sendo esta a base de dados que armazena a informação relativa ao projeto. Com esta instância, recolhe-se todas as coleções existentes na base dados sobre a forme de um `MongoDB\Model\CollectionInfoIterator`, através do método `listCollections()`. Como o nome indica, com este iterador somos capazes de iterar sobre as diversas coleções existentes na base de dados e recolher informações acerca das mesmas (Código 10).

Código 10: Acesso à base de dados e recolha das missões existentes.

```

1 // select a database
2 $db = $client->BoatTelemetryDB;
3 // get all collections (missions)
4 $collections = $db -> listCollections();
5 // for each mission in the collection list
6 foreach ( $collections as $mission ) { ... }
```

Estas informações (Código 11) são utilizadas para criar um excerto HTML com informação variável de cada viagem e inserido numa lista com *scrollbar* vertical. Esta estrutura HTML de uma entrada da tabela encontra-se num

ficheiro separado, denominado por `list_entry.php`, de modo a ser incluído e manipulado sempre que se desejar uma nova entrada. Desta forma, sabendo o nome de cada coleção, seleciona-se da base de dados a missão desejada e filtra-se os seus documentos por ordem crescente e decrescente de *timestamp* para se obter a data de início e de fim da missão, respetivamente. Ambas as datas são posteriormente formatadas para o formato dd/mm/yyyy e dispostas no respetivo local para serem apresentadas ao utilizador. Por fim, extrai-se o seu nome de código e descrição para serem também dispostas no excerto da missão.

Código 11: Recolha da informação de cada missão e tratamento da mesma para posterior visionamento pelo utilizador.

```

1 // gets the desired collection
2 $missionName = $mission->getName();
3 $collection = $db->$missionName;
4 // gets the earliest read
5 $earliestRead = $collection->findOne([], [ 'sort' => [
6     ↪Timestamp' => 1] ]);
7 $earliestDate = (new DateTime($earliestRead["Timestamp"]))
8     ↪->format('d/m/Y');
9 // gets the latest read
10 $latestRead = $collection->findOne([], [ 'sort' => [
11     ↪Timestamp' => -1] ]);
12 $latestDate = (new DateTime($latestRead["Timestamp"]))->
13     ↪format('d/m/Y');
14 // get GPS position based on earliest read
15 $osmZoom = 12;
16 $lat = $earliestRead["GPS"][0];
17 $lon = $earliestRead["GPS"][1];
18 // gets the document containing the codename (and
19     ↪description)
20 $data = $collection->findOne(["Codename" => array('$exists' ,
21     ↪=> true)], []);
22 $missionCodename = $data["Codename"];
23 $missionDescription = $data["Description"];

```

A implementação do mapa de uma entrada é semelhante à realizada para a página de detalhes da missão. Como tal, esta implementação encontra-se explicada em maior detalhe na Secção 4.3.4.

4.3.4 Listagem dos Registos

Esta página, `mission.php`, é acedida quando o utilizador deseja ver os registos efetuados pela embarcação durante uma determinada missão. De modo a identificar a missão a analisar, é transmitido, da página de listagem, o nome da missão através do método `GET` do `HTTP`. Sabendo o nome, realiza-se uma pesquisa na base de dados pela coleção à qual se pretende visualizar a informação contida nos seus múltiplos documentos.

Esta página é dividida em duas secções: a tabela de registos efetuados e a informações acerca da missão. Ambas as secções estão dependentes do conjunto de documentos que constitui a coleção e estes são adquiridos realizando-se uma busca à base de dados pela missão através do seu nome (Código 12).

Código 12: Recessão do nome da missão e recolha da coleção da base de dados.

```

1 // get the document given via GET
2 $INPUT_METHOD = INPUT_GET;
3 $flags [] = FILTER_NULL_ON_FAILURE;
4 $missionName = filter_input( $INPUT_METHOD, 'document',
5   ↪, FILTER_SANITIZE_STRING, $flags );
6
7 // get the given collection from the database
8 $db = $client->BoatTelemetryDB;
9 $collection = $db->$missionName;
```

Tabela de Registros da Embarcação

Para dispôr ao utilizador o conjunto de registos armazenados no servidor para uma missão, é elaborada uma tabela que introduz uma nova linha por cada documento da coleção. Para introduzir cada linha, recorre-se à coleção previamente adquirida da base de dados, sendo possível aceder aos documentos que a coleção apresenta e listar os mesmos na tabela.

Para esse efeito, realiza-se uma *query* de pesquisa geral para se obter todos os documentos existentes da coleção. Apresentando todos os documentos sobre a forma de um objeto `MongoDb\Driver\Cursor`, é possível de se efectuar um ciclo *foreach* a todos os documentos existentes e a cada iteração do

ciclo origina-se uma entrada na tabela com as informações que se pretende apresentar (Código 13).

Código 13: Ciclo *foreach* realizado a todos os documentos da coleção.

```

1 <?php
2 // for every document in the collection/mission
3 $documents = $collection->find([], []);
4 foreach ($documents as $document) {
5 $documentId = $document["_id"];
6 ?>
7 // HTML Code...
8 <?php } ?>
```

De cada documento, extrai-se as informações que se pretendem utilizar para a criação de uma entrada na tabela HTML (Código 14), sendo estas: A data e hora de registo e a latitude e longitude no formato *Degree Minute Seconds (DMS)*. Após serem filtradas do documento, as informações são posteriormente processadas de modo a apresentarem a estrutura correta para serem demonstradas em cada coluna.

Código 14: Porção HTML de cada entrada da tabela.

```

1 <tr>
2     <td id="remove-margin">
3         <?php echo (new DateTime($document["Timestamp"]))->format('d/m/Y H:i:s'); ?>
4     </td>
5     <td id="remove-margin">
6         <?php echo lat_lng($document["GPS"])[0], $document["GPS"][1]; ?>
7     </td>
8     <td id="remove-margin-button">
9         <form action="detail.php?
10            document=<?php echo $missionName ?>&
11            documentId=<?php echo $documentId ?>" method="POST">
12             <button id="link-btn" type="submit"
13                class="btn btn-link btn-md">Ver detalhes</
14                 button>
15             </form>
16         </td>
17     </tr>
```

A data resulta da formatação de uma instância de um objeto `DateTime`, à qual é fornecida a *string*, referenciada como `Timestamp` no documento. A formatação deste objeto permite reescrever a data num formato capaz de indicar o dia, mês e ano, seguido da hora, minuto e segundo à qual foram registados os dados.

Para a posição, foi necessário um processamento diferente. Os dados armazenados no documento encontram-se num formato *Decimal* que é diferente do formato *Degree Minute Seconds* pretendido. Para resolver este dilema, utiliza-se os métodos criados que constam no script `decDmsFormat.php`. A *string* formatada da localização em formato DMS é obtida fornecendo a latitude e a longitude ao método `lat_lng($lat, $lng)`. Dentro desta função, é utilizado o método `dec_dms($dec)` (Código 15) que converte uma determinada latitude ou longitude no formato *Decimal* para o formato DMS, para ser colocada posteriormente na *string*, no formato desejado.

Código 15: Função que permite converter uma coordenada no formato *Decimal* para o formato DMS.

```

1 function dec_dms($dec) {
2     $vars = explode(".", $dec);
3     $deg = $vars[0];
4     $tempma = '0.' . $vars[1];
5
6     $tempma = $tempma * 3600;
7     $min = floor($tempma / 60);
8     $sec = $tempma - ($min * 60);
9
10    return array('deg' => $deg, 'min' => $min, 'sec' => $sec);

```

Secção da Informação da Missão

A segunda secção que compõe a página dos registo efetuados demonstra um conjunto de informações que permitem detalhar a missão realizada pelo veículo.

A descrição é uma das componentes desta secção e apresenta um breve resumo que permite descrever a viagem realizada. Esta é introduzida pelo programador no ato da publicação dos registo para a base de dados. Junto da descrição, está disposto o período à qual foi efetuada a viagem. Estas

datas são adquiridas analisando o **Timestamp** registado pelo primeiro e último documento, estabelecendo assim um espaço temporal que permite identificar quando é que esta foi realizada.

A componente desta secção que se destaca é a localização da missão com auxílio a um mapa interativo. Este mapa demonstra a região onde foram efetuados os registo, colocando diversas marcas interativas nas diferentes posições **GPS** documentadas. Ao interagir com cada marca, o utilizador é redirecionado para a página de detalhe dessa marca em específico, demonstrando todos os detalhes do documento selecionado. Esta representação recorre ao projeto *open source OpenSeaMaps* para a criação dos *layers* do mapa, utilizando posteriormente a *framework OpenLayers* para lhe atribuir a interatividade. A implementação desta componente é referida na Secção 4.3.4 em maior detalhe.

A concluir esta secção, existem dois botões que servem propósitos distintos.

O botão ”Guardar Coleção” permite, como o nome indica, descarregar os dados contidos nos documentos da coleção num ficheiro em formato *Comma-Separated Values (CSV)*. Esta funcionalidade possibilita aos utilizadores aceder com facilidade à informação que foi armazenada na base de dados e que se encontra num formato estático e conhecido. A medida tomada promove a acessibilidade dos dados e a flexibilidade de utilização dos mesmos, dado que estes podem ser utilizados por aplicações externas recorrendo apenas aos ficheiros descarregados. A implementação desta funcionalidade é relatada na Secção 4.3.4.

Por fim, o botão ”Apagar missão” remove por completo a coleção da base de dados, incluindo todos os seus documentos. Esta decisão não é imediata, pois a interação com este mecanismo faz surgir um *modal* com um campo de *input* à qual o utilizador necessita de escrever ”DELETE” para prosseguir com a decisão. Esta implementação prevê situações onde o utilizador poderá apagar accidentalmente uma coleção importante, tornando-se apenas uma funcionalidade que é tomada com a plena consciência do utilizador. A Secção 4.3.4 detalha o procedimento efetuado para a implementação desta funcionalidade assim como a medida de segurança associada.

Mapa Interativo com Auxílio a Software Externo

O mapa interativo é a componente mais ambiciosa da implementação proposta para a página dos detalhes da missão. Esta aplicação geográfica é responsável por demonstrar o local onde a viagem foi realizada, assim como demonstrar no mapa os diversos pontos geográficos onde foram efetuados registos por parte da embarcação, através de marcadores.

Para o desenvolvimento desta funcionalidade, optou-se pela utilização de *software open-source*. Como tal, foi escolhido o *OpenSeaMaps* para fornecer os dados capazes de desenvolver as cartas náuticas, que podem ser posteriormente utilizadas numa *framework* de modo a introduzir interatividade. No contexto do projeto, é utilizada a *framework OpenLayers* para servir esse propósito, principalmente porque esta *framework* já dispõe de um conjunto de funcionalidades que possibilita a comunicação direta com a API.

Foi necessário descarregar a biblioteca JavaScript *OpenLayers* e coloca-la junto aos restante ficheiros da aplicação *web*, de modo a que o projeto seja capaz de usufruir das funcionalidades disponibilizadas por esta *framework*. Esta biblioteca é capaz de realizar a comunicação com a API do *OpenSeaMaps* através do objeto *OSM()*, que descarrega as cartas náuticas necessárias para construir o mapa da localização que se pretende visualizar.

Desenvolveu-se assim o ficheiro *OSMOpenLayer.php* para implementar esta biblioteca no projeto, com o intuito de apresentar as viagens da embarcação. Este ficheiro importa a biblioteca (Código 16) e cria um conjunto de *scripts* JavaScript que são manipulados no lado do servidor através do PHP, consonte os dados dispostos na base de dados.

Código 16: Importação da biblioteca OpenLayers.

```

1 <!-- bring in the OpenLayers library -->
2 <script src="libs/OpenLayers-2.13.1/OpenLayers.js"></script
    ↩>
```

É implementado o método *loadOpenLayersMap()* (Código 17) que, através da biblioteca previamente importada, instancia um objeto *Map()* para ser colocado na estrutura HTML com identificador *id* igual a ”map-[nome_da_missão]”. Este objeto constitui a aplicação que vai albergar as diversas *layers* que compõem o mapa interativo, sendo uma dessas *layers* as telhas do mapa *OpenSeaMaps*. Quando se conclui a construção do mapa, este é colocado no

elemento HTML referido no construtor do mapa, para posterior interação com o utilizador na página *web*.

Para instanciar a *layer* do mapa OSM, cria-se um objeto `OSM()` da biblioteca. Este objeto permite a comunicação direta com a API, como referido anteriormente, através do conjunto de *links* que se encontram no construtor do objeto.

Para representar qualquer coordenada no mapa interativo, é necessário instanciar duas projeções para garantir que os pontos instanciados encontram-se dentro da formatação suportada pelo mapa. De acordo com a documentação, é necessário uma projeção para transformar do formato *World Geodetic System*[3] e outra para converter esse formato para *Spherical Mercator Projection*[8]. Estas transformações são aplicadas a uma coordenada, criada a partir da sua longitude e latitude, através do método `transform(from, to)`.

Dispondo de todas estas estruturas, é necessário adicionar a *layer* OSM ao objeto `Map` e centrar o mapa numa determinada longitude e latitude, transformada de acordo com as projeções criadas. É também possível estabelecer o fator de *zoom* por omissão.

Código 17: Parte inicial do método onde se instancia a camada do mapa e componentes relacionadas.

```

1 // loads the map focused on the lat and lon of the first
   ↵registered point.
2 function loadOpenLayersMap() {
3     map = new OpenLayers.Map("map-<?php echo $missionName
   ↵?>");
4     var mapnik = new OpenLayers.Layer.OSM();
5     var fromProjection = new OpenLayers.Projection("EPSG
   ↵:4326");    // Transform from WGS 1984
6     var toProjection = new OpenLayers.Projection("EPSG
   ↵:900913");    // to Spherical Mercator Projection
7     var position = new OpenLayers.LonLat(<?php echo $lon
   ↵?>,<?php echo $lat ?>).transform(fromProjection,
   ↵toProjection);
8     var zoom = <?php echo $osmZoom ?>;
9
10    map.addLayer(mapnik);
11    map.setCenter(position, zoom);
12    ...
13 }
```

A colocação dos marcadores no mapa necessita de aceder à base de dados **MongoDb** para se obter a latitude e longitude dos diversos registo efetuados. Atendendo às funcionalidades da linguagem PHP, como este ficheiro é incluído na página principal podemos garantir que terá acesso à variável `$collection` que armazena a coleção da missão selecionada. Desta forma, conseguimos extrair desta coleção todos os documentos nela contidos (cada um dispondo um registo efetuado pela embarcação num determinado instante de tempo).

Dispondo de cada documento, itera-se sobre cada um e extrai-se a sua latitude e longitude de modo a instanciar-se uma posição que é também transformada de acordo com as projeções. Cada uma destas marcas é adicionada à *layer markers* para agrupar estas marcadas numa só camada e demonstra-la por cima da *layer* do mapa. Adicionalmente, acrescenta-se um evento quando uma marca é clicada, de modo a redirecionar o utilizador para a página dos detalhes do documento que pretende visualizar. Cada um destes tratamentos realizados a cada documento é efetuado com auxílio à linguagem PHP para introduzir na função as linhas necessárias para o tratamento de cada marca. (Código 18).

Código 18: Parte final do método onde se instancia os marcadores e a sua camada.

```

1 // loads the map focused on the lat and lon of the first
   ↪registered point.
2 function loadOpenLayersMap() {
3     ...
4     // add markers
5     var markers = new OpenLayers.Layer.Markers( "Markers" )
   ↪;
6     map.addLayer(markers);
7
8     <?php
9     $documents = $collection->find( [ ] , [ ] );
10    foreach ($documents as $document) {
11        $doclat = $document[ "GPS" ][ 0 ];
12        $doclon = $document[ "GPS" ][ 1 ];
13        echo "var position = new OpenLayers.LonLat(" .
   ↪$doclon . "," . $doclat . ").transform(
   ↪fromProjection , toProjection );\n";
14        echo "var mark = new OpenLayers.Marker(position);\n
   ↪";
```

```

15 echo "markers.addMarker(mark);\\n";
16 echo "mark.events.register(\"click\", mark,
17   →function(e){ window.location.href=\"detail.php
18   →?document=$missionName&documentId=$documentId
19   →\"; });
  }?
}
  
```

Por fim, de acordo com o modelo proposto, é necessário especificar a cidade e o país onde foram efetuados os registos. Para esta funcionalidade, recorreu-se novamente à API do *OpenSeaMaps* onde foi necessário realizar uma pesquisa por *Geocode* inversa, ou seja, uma pesquisa através da latitude e longitude de maneira a obter os dados de uma determinada posição geográfica. O método `reverseGeocode` acede à API, utilizando o método *built-in JavaScript* `fetch()`, através de um endereço à qual é especificando a longitude e latitude da posição que se pretende analisar. A resposta é entregue ao cliente no formato JSON e desta resposta acede-se à morada da posição (*address*), que dispõe do distrito (*county*) e país (*country*), informação necessária para preencher a localização do registo. (Código 19);

Código 19: Pesquisa por *Geocode* inversa para obter uma localização.

```

1 // reverse geocodes and gets the address from the location
2   →specified on the mission.
3 function reverseGeocode() {
4   fetch('https://nominatim.openstreetmap.org/reverse?
5     →format=json&lon=' + <?php echo $lon ?> + '&lat=' +
6     → <?php echo $lat ?>
7     .then(function(response) {
8       return response.json();
9     }).then(function(json) {
10    // console.log(json);           // debug
11    if (json["address"]["county"] != null && json[
12      →"address"]["county"] != null)
13      document.getElementById("location-name-<?php
14        →echo $missionName ?>").innerHTML = json[
15          →"address"]["county"] + ", " + json["address
16          →"]["country"];
17    else
18      document.getElementById("location-name-<?php
  
```

```

12 }      →echo $missionName ?> ).innerHTML = "N.A";
13 }
```

Gravação dos Ficheiros CSV

Para a gravação da informação da coleção, foi concebido o *script* `processSaveCollection.php`. Este cria um documento temporário e introduz uma primeira linha de cabeçalho, servindo como uma legenda para cada linha inserida. Dispondo da coleção e todos os seus documentos, itera-se sobre todos eles e constrói-se uma *string*, introduzindo cada parâmetro pela ordem estabelecida pela linha que serve de legenda. Cada iteração coloca no documento uma nova entrada, repetindo-se o processo até serem registadas as informações de todos os documentos (Código 20).

Código 20: Criação e escrita do ficheiro em formato *Comma-Separated Values* com base nos documentos presentes na coleção selecionada.

```

1 $csvFile = fopen("temp.txt", "w") or die("Erro na abertura!");
  ↵";
2 $txt = "Timestamp ; Latitude ; Longitude ; AIS ; AguaNoCasco" . ;
  ↵Corrente ; Voltagem ; Heading ; Pitch ; Roll\n";
3 fwrite($csvFile, $txt);
4 $documents = $collection->find([], []);
5 foreach($documents as $document) {
6     $txt = $document["Timestamp"] . ";" . $document["GPS"][0] . "
  ↵;" . $document["GPS"][1] . ";" . $document["AIS"] . ";" .
  ↵$document["Water"] . ";" . $document["Corrente"] . ";" .
  ↵$document["Voltagem"] . ";" . $document["IMU"][0] . ";" .
  ↵$document["IMU"][1] . ";" . $document["IMU"][2] . "\n";
7     fwrite($csvFile, $txt);
8 };
9 fclose($csvFile);
```

Dispondo do ficheiro a enviar, o próximo desafio é transferi-lo para o utilizador que o pediu. Deste modo, recorre-se às funcionalidades do cabeçalho HTTP, estabelecendo um conjunto de parâmetros que possibilitam a troca deste ficheiro (Código 21). Indica-se no parâmetro `Content-Disposition` que será enviado um ficheiro em anexo, assim como o nome que este irá apresentar. É importante não esquecer de indicar no parâmetro `Content-Type`

que este anexo encontra-se representado como uma *stream* de octetos que terá um determinado tamanho variável, sendo preciso recorrer ao parâmetro **Content-Length** para se saber a dimensão exata da *stream* enviada para posterior descodificação.

Código 21: Atribuição dos múltiplos cabeçalhos HTTP para o envio de ficheiro na resposta ao pedido efetuado.

```

1 // clean the mission name
2 $csvFileName = str_replace("_", "", $missionName);
3 header('Content-Description: CSV File Transfer');
4 header('Content-Type: application/octet-stream');
5 header('Content-Disposition: attachment; filename=csv-'.
6     ↪$csvFileName.'.txt');
7 header('Expires: 0');
8 header('Cache-Control: must-revalidate');
9 header('Pragma: public');
10 header('Content-Length: ' . filesize("temp.txt"));
11 readfile("temp.txt");
12 exit;

```

Remoção da Missão da Base de Dados

Esta funcionalidade é desempenhada pelos *scripts* `processDeleteCollection.php` e `deletePopup.php`, ambos com funcionalidades distintas.

Como já referido, ao interagir com o botão, o utilizador não realiza de imediato um pedido ao servidor. A interação resulta na abertura de um *modal popup*, cuja estrutura encontra-se em `deletePopup.php`. Este *modal* apresenta no seu corpo um formulário e é este que, ao ser submetido, realiza um pedido ao servidor para o *script* `processDeleteCollection.php`.

No entanto, este formulário apresenta um *onsubmit*, que interceta a submissão do formulário recorrendo ao método JavaScript `checkDelete()` (Código 22). Este método averigua se o utilizador introduziu corretamente a chave "DELETE" no campo com identificador "confirmDelete", de modo a autorizar a realização da ação de submissão do formulário.

Código 22: Método que averigua se o *input* do utilizador coincide com a chave.

```

1 // checks if user input was correct (=DELETE)
2 function checkDelete() {
3     if (document.getElementById("confirmDelete").value ==
4         →"DELETE") {
5         return true;
6     }
7 }
```

Ao submeter o pedido ao servidor, é iniciada a execução do ficheiro `processDeleteCollection.php` que vai realizar as alterações à base de dados de modo a remover a coleção. A remoção de uma coleção é facilmente efetuada recorrendo à biblioteca MongoDB, dado que esta disponibiliza diretamente o método `dropCollection($collection)` do objeto `Database`. Desta forma, para eliminar a coleção do sistema, é necessário instanciar a base de dados da aplicação a partir do cliente `MongoClient`. Para além da base de dados, é necessário apresentar uma referência para a coleção a eliminar. Esta coleção é aquela ao qual o utilizador se encontrava a visualizar os seus detalhes. Desta forma, é apenas necessário fornecer ao servidor, como parâmetro do pedido, o nome do documento que estava a ser analisado. Dispondo de todas as estruturas, somos capazes de realizar a remoção da missão da base de dados através do método `dropCollection($collection)` (Código 23), não esquecendo de redirecionar o utilizador de volta à página de listagem no final do processamento.

Código 23: Remoção da coleção especificada da base de dados.

```

1 // get the collection name given via GET
2 $missionName = filter_input( INPUT_GET, 'document',
3     →FILTER_SANITIZE_STRING, $flags);
3 ...
4 // get the given collection from the database
5 $db = $client->BoatTelemetryDB;
6 $db->dropCollection( $missionName );
```

4.3.5 Detalhes de um Registo

A última página implementada diz respeito aos detalhes de um registo efetuado durante a missão. Um utilizador é redirecionado para esta página através da tabela de registos dos detalhes da missão ou através da interação com um dos marcadores do mapa interativo que se encontra na mesma página.

Esta página dispõe de uma tabela flexível, capaz de apresentar todos os dados presentes num documento, mesmo que este seja posteriormente alterado face às potenciais modificações que poderão existir no futuro.

Para além disso, a página implementa um simples sistema de criação de gráficos. Através de um formulário, realiza-se um conjunto de escolhas que permitem transformar o gráfico que se pretende criar. É possível alterar o seu estilo, entre gráfico de barras ou gráfico de linhas, escolher a estrutura de dados a analisar, com base nos dados presentes no documento, e definir o intervalo de documentos que vão fazer parte do gráfico, limitando o inicio e o fim dos documentos representados. Este formulário encontra-se disposto na Figura 4.4.

The screenshot shows a user interface for creating a plot. At the top, there is a dropdown menu labeled "Estilo" with "Barras" selected. Below this, there are three input fields: "Parâmetro:" containing "IMU", "De:" containing "0", and "Até:" containing "2".

Figura 4.4: Demonstração do formulário de definições de criação do gráfico.

A alteração de cada um destes *inputs* realiza uma chamada ao método `updatePlot()` através do parâmetro `onchange`. Este método JavaScript permite atualizar um *Uniform Resource Locator* (URL) fornecendo os múltiplos valores dos *inputs* como parâmetros GET para serem processados no servidor. Este endereço URL é inserido como a *source* de um elemento de imagem do HTML. Quando atualizada a *source*, é realizada uma chamada ao endereço especificado que, neste caso, realiza um pedido ao servidor pelo ficheiro `processDetailPlot.php` de modo a fornecer uma imagem do gráfico desejado.

Para a criação dos gráficos do lado do servidor, utiliza-se a biblioteca `PHPPlot`[10] que fornece um conjunto de funcionalidades PHP que servem este

propósito. Ao ser recebido o pedido para o ficheiro `processDetailPlot.php`, são armazenados todos os parâmetros recebidos através do GET para posterior uso. Adicionalmente, instancia-se um objeto `PHPPlot`, que representa o *canvas* onde será desenhado o gráfico. As definições deste objeto são alteradas dependendo do tipo de gráfico a desenhar, escolha que se encontra armazenada no atributo recebido pelo pedido dos *inputs* do formulário do utilizador. Estas configurações estão armazenadas em dois métodos distintos do ficheiro `customPHPPlots.php`: para gráficos de barras é utilizado o método `plotBars($plot, $data)` enquanto que para um gráfico de linhas é utilizado o método `plotLines($plot, $data)`. Cada um dos métodos recebe o objeto `PHPPlot $plot` à qual vai efetuar as alterações e uma estrutura `$data` que coincide com os dados a serem representados no gráfico. Esta estrutura de dados é adquirida através do método `getDataFromCollection($args)` deste ficheiro que, através de um determinado conjunto de argumentos, instância uma estrutura com o formato adequado para ser utilizado nos gráficos da biblioteca.

Por fim, após configurado o objeto `$plot`, utilizando todas as configurações pedidas, é utilizado o método `DrawGraph()` da biblioteca, que permite criar uma imagem a partir do *canvas* do gráfico. Esta imagem que foi desenhada é o resultado que vai ser atribuído ao elemento de imagem que se encontra do lado do cliente, dado que é o *output* do pedido especificado pela *source* do mesmo. Desta forma, a imagem é sempre disposta ao utilizador e alterada após atualizado qualquer um dos *inputs*, isto desde que os parâmetros especificados sejam válidos.

Adicionalmente, adicionou-se um elemento de botão (Código 24) cujo papel é realizar a transferência da imagem do gráfico para o computador do cliente. Utilizando um botão, é possível realizar a transferência de qualquer conteúdo especificando o seu atributo `href` com a hiperligação do ficheiro que se pretende transferir. É importante indicar também uma *string* para o atributo `download`, de modo a indicar que o ficheiro especificado no endereço é para ser transferido e deve de utilizar a *string* especificada como nome do ficheiro transferido.

Código 24: Botão para a transferência da imagem.

```
1 <a id="download-plot" class="btn btn-sm btn-success btn-
  ↪outline text-white mt-3" href="processDetailPlot.php?
  ↪args=..." download="plot.png" style="float: right">
  2   <i class="fa fa-download fa-sm pr-2" aria-hidden="true"
    ↪></i> Descarregar Gr fico
  3 </a>
```


Capítulo 5

Validação e Testes

Dado que a embarcação USV-enautica1 ainda não se encontra concluída, não foi possível realizar testes de contexto real, ou seja, em alto mar, como seria esperável relativo à função da embarcação.

No entanto, para efeitos de teste e para determinar o correto funcionamento das tecnologias de armazenamento criadas pelo grupo, foram realizados testes utilizando dados extraídos dos sensores que se encontravam em funcionamento da embarcação através do *script* desenvolvido para este propósito (Secção 4.1.1).

Após ter sido realizada a extração destes dados, foi criado um *script* específico que, quando alimentado com dados, realiza o *upload* para a base de dados remota, MongoDB, de forma idêntica ao que o *script* de *upload* faria quando recebesse esta indicação (Secção 4.1.2).

No código presente no Código 25 pode-se verificar um exemplo dos dados que a função responsável pelo armazenamento de dados local ou remoto.

Código 25: Exemplo de dados que serão *uploaded*.

```
1 # exemplo de dados guardados localmente previamente
2 dados3 = {"Timestamp": (datetime.fromtimestamp(timestamp)).strftime("%Y/%m/%d, %H:%M:%S"), "GPS":
3     ↪(38.691498333333335, -9.29858833333333),
4     ↪"IMU": (358, -8.520272254943848, -0.0),
5     ↪"AIS": ' ', "Corrente": 32,
6     ↪"Voltagem": 123,
6     ↪"Water": 2}
7 # exemplo de novos dados para upload
8 dados4 = {"Timestamp": (datetime.fromtimestamp(timestamp)).
```

```

9   ↪strftime ("%Y/%m/%d , %H:%M:%S") , "GPS" : (38.6914983335 ,
10  ↪-9.2985883334) ,
11  "IMU" : (358 , -8.520272254943848 , -0.0) ,
12  "AIS" : ' ' , "Corrente" : 30 ,
13  "Voltagem" : 125 ,
14  "Water" : 3 ,
    "Codename" : "ENIDH" ,
    "Description" : "Descrição de teste ,
        ↪viagem realizada entre
        ↪localização X e Y" }
```

Posteriormente, dependendo do tipo de armazenamento a ser empregue, foram criadas duas funções idênticas às presentes no *script send_to_db.py*: uma para o ser realizado localmente (Código 26) e outra para realizar armazenamento remoto (Código 27). No entanto, a função de *upload* para a base de dados remota encontra-se simplificada para efeitos de teste.

Código 26: Função de armazenamento local.

```

1 def update_json(new_data , file_name=file_path) :
2     with open(file_name , 'r+') as file :
3         file_data = json.load(file)
4         file_data ["mission_details"].append(new_data)
5         file.seek(0)
6         json.dump(file_data , file , indent=4)
```

Código 27: Função de armazenamento remoto.

```

1 def upload_data(data_to_upload):
2     # ligacao ao MongoDB
3     client = MongoClient(
4         "mongodb+srv://admin:conchinha123@clusterisel.ksoyd
5             ↪.mongodb.net/myFirstDatabase?retryWrites=true&
6             ↪w=majority")
7
8     mydb = client ["BoatTelemetryDB"]
9
10    mycollection = mydb ["Mission1"]
11
12    mycollection.insert_one(data_to_upload)
```

Após o *upload* estar concluído, verificou-se a existência dos novos dados na base de dados remota com sucesso, como se pode constatar na Figura 4.2. Localmente está presente, como presente no Código 28, um ficheiro JSON de exemplo com os dados como esperado.

Código 28: Documento JSON de exemplo de uma missão.

```

1  {
2      "mission_details": [
3          {
4              "Timestamp": "2021/07/17, 18:54:34",
5              "GPS": [
6                  38.691496,
7                  -9.29858
8              ],
9              "IMU": [
10                 358,
11                 -8.52025,
12                 -0.0
13             ],
14             "AIS": """",
15             "Corrente": 33,
16             "Voltagem": 125,
17             "Water": 3
18         },
19         {
20             "Timestamp": "2021/07/17, 18:55:36",
21             "GPS": [
22                 38.691498333333335,
23                 -9.29858833333333
24             ],
25             "IMU": [
26                 358,
27                 -8.520272254943848,
28                 -0.0
29             ],
30             "AIS": """",
31             "Corrente": 32,
32             "Voltagem": 123,
33             "Water": 2
34         },
35         {
36             "Timestamp": "2021/07/18, 14:23:55",

```

```

37     "GPS" : [
38         38.6914983335,
39         -9.2985883334
40     ],
41     "IMU" : [
42         358,
43         -8.520272254943848,
44         -0.0
45     ],
46     "AIS" : """",
47     "Corrente" : 30,
48     "Voltagem" : 125,
49     "Water" : 3,
50     "Codename" : "ENIDH",
51     "Description" : "Descri\u00e7\u00e3o de teste,
52                               \u2192 viagem realizada entre localiza\u00e7\u00e3o \
53                               \u2192 u00e3o X e Y"
54 }
```

Com isto, conclui-se que toda a parte relativa ao funcionamento, desde a recolha de dados da embarcação até ao *upload* para a base de dados, encontra-se em devido funcionamento.

Finalmente, no que diz respeito à aplicação *web* para a apresentação dos dados ao utilizador, utilizaram-se os dados colocados previamente na base de dados para simular a existência de várias missões com os respetivos dados. Para os testes à aplicação *web*, realizaram-se duas viagens simuladas com um número reduzido de registos efetuados.

Iniciou-se o exame à aplicação com a página inicial. Testou-se iniciar sessão com um utilizador não registado, resultando no aparecimento da mensagem de erro disposta prevista, tal como visualizado na Figura 5.1.

Ao ser autenticado, o utilizador é redirecionado para a página de listagem das missões existentes, que se encontra demonstrada na Figura 5.2. É possível identificar o registo das duas viagens que foram anteriormente simuladas e que se encontram na base de dados. Adicionalmente, podemos averiguar que se encontram a funcionar as componentes flexíveis, como as imagens ou os nomes de código de cada missão.

Ao interagir com o botão de ”Ver Detalhes” o utilizador é redirecionado



Figura 5.1: Ecrã da página inicial num cenário de autenticação inválida.



Figura 5.2: Ecrã que lista as duas viagens simuladas.

para a página que disponibiliza os detalhes acerca da missão selecionada, disposta na Figura 5.3. Verifica-se à direita a tabela que demonstra os dois registos efetuados e as suas respetivas datas, juntamente com as coordenadas do seu instante geográfico no formato DMS. À esquerda apresenta-se o conjunto de informações relativas à missão, que são capazes de a descrever, assim

como os comandos que possibilitam apagar a coleção ou realizar o *download* dos dados. Verifica-se a existência de um período temporal, que é estipulado de acordo com os documentos armazenados, assim como da descrição, que se encontra armazenada no servidor e que foi colocada pelo utilizador no ato de *upload* dos registos simulados. O pressionar do botão "Guardar coleção" despoleta a funcionalidade de *download* dos dados no formato CSV. O resultado desta funcionalidade utilizando os dados de teste encontram-se disponibilizados na Lista 29.

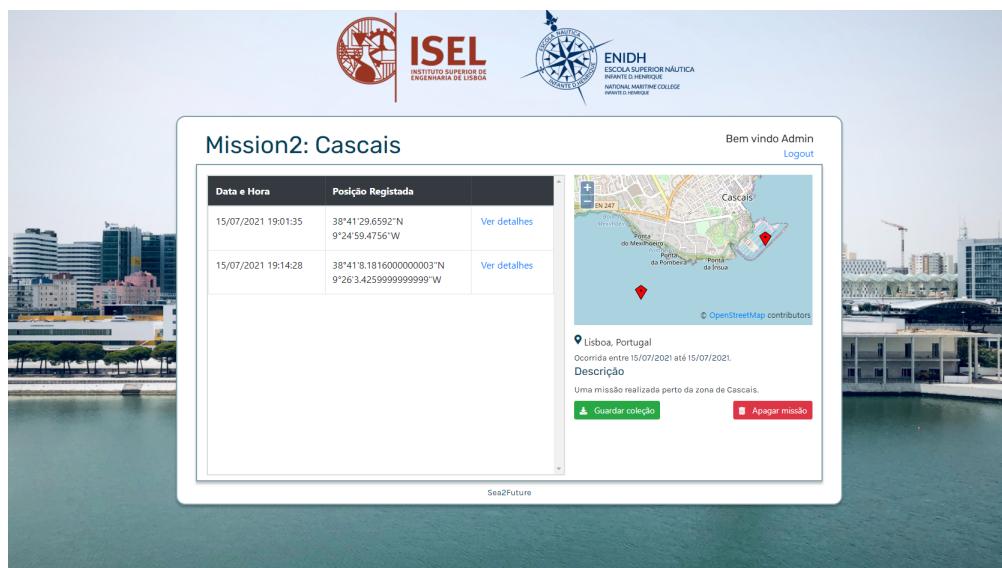


Figura 5.3: Página de detalhes de uma missão.

Código 29: Documento CSV resultante do *download* dos dados de teste.

```

1 Timestamp ; Latitude ; Longitude ; AIS ; AguaNoCasco ; Corrente ;
   ↪ Voltagem ; Heading ; Pitch ; Roll
2 2021/07/15, 19:01:35; 38.691572; -9.416521; ; 2; 32; 123;
   ↪ 358; -8.5202722549438; -
3 2021/07/15, 19:14:28; 38.685606; -9.434285; ; 2; 32; 123;
   ↪ 358; -8.5202722549438; -

```

Ao premir o botão de "Apagar missão", surge a janela de confirmação apresentada na Figura 5.4. Esta janela pergunta pela confirmação do utilizador no ato de apagar a missão da base de dados.

Quando inserida corretamente a chave de confirmação, é apagada a coleção



Figura 5.4: *Popup* para a confirmação da remoção da coleção.

selecionada. Este processo foi realizado para a missão dois que, dado o sucesso na remoção da coleção, já não se encontra visível na Figura 5.5.



Figura 5.5: Ecrã que agora lista apenas uma viagem simulada.

A última página a verificar diz respeito aos detalhes de um registo efectuado durante a missão. Seleccionou-se o primeiro documento da coleção **Mission1**, onde se foi redirecionado para a página disposta na Figura 5.6.

Nesta página, podemos averiguar à esquerda a tabela com todos os parâmetros que constam no documento. Confirma-se a flexibilidade da tabela, visto que apresenta a barra de *scroll* que permite descer nas entradas caso seja decidido implementar mais parâmetros a armazenar em cada registo.

À direita dispõe-se do formulário, que foi preenchido para apresentar um

gráfico de barras com os dados do *Inertial Measurement Unit* (IMU), desde o primeiro documento até ao terceiro documento.



Figura 5.6: Detalhes de um registo da missão um e o seu respetivo gráfico desenhado.

Realizou-se a transferência da imagem, para averiguar a implementação desta funcionalidade, obtendo-se o ficheiro disposto a Figura 5.7, que é idêntica ao gráfico disposto na interface gráfica e que foi pedido.

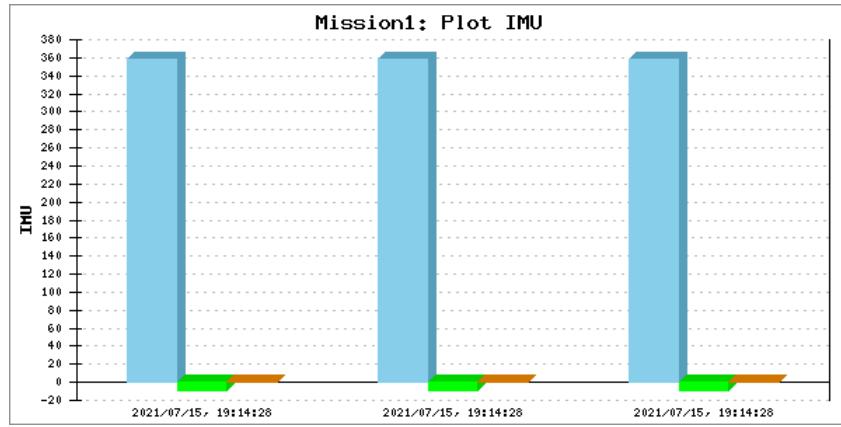


Figura 5.7: Gráfico transferido a partir da aplicação *web*.

Capítulo 6

Conclusões e Trabalho Futuro

Este projeto proporcionou ao grupo um desafio diferente dos que já tinha enfrentado ao longo do seu percurso, devido à natureza do mesmo. A utilização e necessidade de familiarização de novas tecnologias como o *Robot Operating System (ROS)*, *MongoDB* e *OpenSeaMaps (OSM)* surgiu como uma valorização de competências futuras, assim como coordenação entre os vários membros intervenientes no projeto que apresentam grande relevância num ambiente profissional.

Apesar do projeto planeado inicialmente ter sido concluído com sucesso com a adição extra da aplicação *web* por parte do grupo para complementar o projeto, a implementação total do mesmo, mais concretamente na parte que depende diretamente na embarcação, não foi possível realizar devido à embarcação ainda não se encontrar finalizada, devido a razões alheias a ambos a ENIDH e o ISEL.

A realização do projeto apresentou alguns desafios, no entanto, com ajuda dos orientadores não existiram grandes dificuldades durante o desenvolvimento.

Futuramente, de maneira a expandir o projeto, seria de interesse implementar algumas melhorias a nível dos *scripts* e a nível da aplicação *web*. De maneira a tornar o armazenamento de dados mais inteligente e autónomo, pretende-se que seja implementada a capacidade por parte do *Python* de analisar o espaço que ainda está disponível no cartão de memória do *Raspberry Pi* da embarcação e, através desta informação, adapta os valores dos intervalos correspondentes às prioridades de armazenamento dos dados, em vez da implementação atual fixa que implica a alteração manual dos ficheiros de

configuração. Em relação ao envio dos dados prentede-se que seja implementada a capacidade de saber se o envio dos dados foi realizado na sua integra, algo que atualmente não se verifica. Finalmente, para demonstrar os gráficos foi utilizada a biblioteca `PHPPlot`, que apesar de funcionar, não se enquadra como pretendido em termos estéticos, como tal, procura-se implementar esta funcionalidade através da API `CanvasJS`.

Bibliografia

- [1] Brás, N. e Viegas, C. (2020). Android boatcom. Technical report, Dept. de Eletrónica e Telecomunicações e de Computadores, Instituto Superior de Engenharia de Lisboa, Instituto Politécnico de Lisboa.
- [2] Cardoso, F. (2019). Boat communication - boatcom. Technical report, Dept. de Eletrónica e Telecomunicações e de Computadores, Instituto Superior de Engenharia de Lisboa, Instituto Politécnico de Lisboa.
- [3] GISGeography (2021). World geodetic system (wgs84). <https://gisgeography.com/wgs84-world-geodetic-system/>.
- [4] MathWorks (2021). Exchange data with ros publishers and subscribers. <https://www.mathworks.com/help//ros/ug/exchange-data-with-ros-publishers-and-subscribers.html>.
- [5] MetaCarta (2006). Openlayers. <https://openlayers.org/>.
- [6] MongoDb (2008). Welcome to the mongodb documentation. <https://docs.mongodb.com/>.
- [7] Nunes, P. e Ventura, S. (2021). Mockups da aplicação realizados em figma. <https://www.figma.com/proto/vXmVs4qZZVhjswjBQFusFu/Projeto-final?node-id=1%3A2&scaling=min-zoom&page-id=0%3A1&starting-point-node-id=1%3A2>.
- [8] OpenLayers (2012). Spherical mercator. http://docs.openlayers.org/library/spherical_mercator.html.
- [9] OpenSeaMaps (2009). Frequently asked questions. <https://www.openseamap.org/index.php?id=faq&L=1>.

- [10] PHPlot (2016). Phplot documentation. <https://github.com/AJRepo/PHPlot/tree/master/phplotdocs>.
- [11] PyMongo (2021). Pymongo documentation. <https://pymongo.readthedocs.io/en/stable/>.
- [12] Python3 (2012). Python programming language. <https://docs.python.org/3/>.
- [13] RaspberryPi (2012). Faqs. <https://www.raspberrypi.org/documentation/faqs/>.
- [14] ROS (2017). Rospy documentation. <http://wiki.ros.org/rospy>.