



Licenciatura em Engenharia Informática e Computadores
Projeto e Seminário - Semestre de Verão 2018/2019

Relatório Final

Monitorização de qualidade de serviço em
comunicações ferroviárias

João Vaz, Nº 41920
41920@alunos.isel.pt

Luis Vasconcelos, Nº41556
41556@alunos.isel.pt

Orientado por:

Nuno Cota, ncota@deetc.isel.pt
Ana Beire, anaritabeire@solvit.pt

Agradecimentos

A elaboração do presente Relatório Final não seria possível sem o apoio de alguns intervenientes. Assim sendo, pretendemos agradecer a todos os que sempre nos apoiaram e contribuíram para a realização e concretização desta etapa final na nossa formação, a Licenciatura em Engenharia Informática e Computadores.

Deste modo, agradecemos:

Às nossas famílias, pois tudo isto foi possível graças ao esforço e dedicação que sempre tiveram.

Aos Orientadores Nuno Cota e Ana Rita Beire pela sua disponibilidade e compreensão, orientando e guiando o desenrolar do nosso projeto, manifestando sempre as suas opiniões enriquecedoras para o crescimento deste Relatório Final e enriquecimento da nossa formação.

A todos os docentes que contribuíram para a nossa formação ao longo da licenciatura, por todos os conhecimentos, dedicação e contributo para o nosso crescimento pessoal e educacional.

Aos colegas que sempre estiveram presentes, com todo o apoio, carinho e compreensão.

Resumo

O presente Relatório Final, desenvolvido no âmbito de Projeto e Seminário, do plano de estudos da Licenciatura em Engenharia Informática e Computadores, abrange quatro partes distintas.

As redes de comunicações ferroviárias são consideradas redes críticas devido à sua importância na gestão da exploração ferroviária. Uma forma eficaz de monitorização de qualidade de serviço é baseada em sondas colocadas a bordo dos comboios que realizam a monitorização aos sistemas de comunicações móveis ferroviários. O projeto consiste no desenvolvimento de um sistema de informação que permita realizar a administração de sondas e análise de dados proveniente das sondas embarcadas nos comboios.

O conjunto de sondas embarcadas tem a sua administração e monitorização realizada de forma centralizada por um sistema de informação. O sistema é constituído por um conjunto de unidades móveis de recolha de medidas e pelo sistema central que envolve todas as aplicações e ferramentas necessárias à operação e gestão das sondas e do sistema global. A solução proposta passa pelo desenvolvimento de uma Web API para acesso aos dados reportados pelo conjunto de sondas.

A implementação do sistema foi dividida em componente cliente e componente servidora. A implementação da componente servidora teve por base a componente realizada posteriormente pelos orientadores. Na Base de Dados foi feito o modelo conceptual, restrições de integridade e lógica de negócio. Na API foi feita uma análise da informação que seria necessário disponibilizar de acordo com as funcionalidades que se iriam implementar. Este componente contém a interface gráfica apresentada ao utilizador. A componente cliente foi desenvolvida com o objetivo de disponibilizar uma experiência simples e objetiva ao utilizador na gestão de sondas e utilizadores e na apresentação de dados.

O trabalho desenvolvido procurou responder aos diversos desafios, tendo em conta a abrangência de tecnologias envolvidas no projeto. Considera-se que foram atingidos globalmente os objetivos propostos à exceção da pesquisa por local e o requisito opcional. Numa futura continuação do projeto faria sentido terminar os módulos não concluídos e implementar outras funcionalidades.

Índice

Agradecimentos.....	i
Resumo.....	iii
Lista de Figuras.....	viii
Lista de Tabelas.....	viii
Lista de Acrónimos	Erro! Marcador não definido.x
1. Introdução	1
1.1. Enquadramento.....	1
1.2. Objetivos.....	1
1.3. Organização do Documento	2
2. Descrição do sistema e Solução Proposta.....	3
2.1. Caracterização do sistema.....	3
2.1.1. Arquitetura	3
2.1.2. Configuração	4
2.1.3. Plano de Testes	5
2.1.4. Requisitos do sistema.....	5
2.2. Solução Proposta.....	6
2.2.1. Arquitetura do Sistema	6
3. Implementação	9
3.1. Base de Dados	9
3.1.1. Modelo Concetual.....	9
3.1.2. Restrições de Integridade e Lógica de Negócio	11
3.2. API	13
3.2.1. Controladores.....	13
3.2.2. Serviços	14
3.2.3. Repositórios.....	15
3.2.4. Intercetores	16
3.2.5. Filtro.....	16

3.2.6.	<i>Paths</i> disponibilizados	16
3.3.	Aplicação Cliente	19
3.3.1.	Intercetores:	20
3.3.2.	Serviços	20
3.3.3.	Componentes	21
3.3.4.	Permissões de Utilizadores Erro! Marcador não definido.	
4.	Conclusões	33
4.1.	Trabalho desenvolvido	33
4.2.	Conclusões Finais	34
4.3.	Trabalho futuro	34
5.	Referências	35

Lista de Figuras

Figura 1 – Arquitetura geral do sistema.	3
Figura 2 – Exemplo de uma OBU.	4
Figura 3 – Arquitetura da solução.	7
Figura 4 – Modelo EA.	9
Figura 5 – Interface de login da aplicação.	22
Figura 6 – Página principal da aplicação.	23
Figura 7 – Exemplo de visualização da rota da OBU num determinado período de tempo no mapa	24
Figura 8 – Exemplo de visualização da velocidade das OBUs no gráfico	24
Figura 9 – Menu lateral (na página inicial)	25
Figura 10 – Página ‘All Users’	26
Figura 11 – Página ‘User Details’	27
Figura 12 – Página ‘All OBUs’	28
Figura 13 – Página ‘Test Plan Details’	29
Figura 14 – Página ‘Setup Create’	30
Figura 15 – Página de ‘Serverlogs’	31

Lista de Tabelas

Tabela 1 – Permissões utilizadas no sistema.	12
Tabela 2 – Rotas disponibilizados pela API	17

Lista de Acrónimos

API: <i>Application Programming Interface</i>	passim
BA: <i>Basic Authentication</i>	13
CORS: <i>Cross Origin Resource Sharing</i>	16
EA: Entidade Associação.....	9, 11
GPS: <i>Global Positioning System</i>	5
GSM-R: <i>Global System for Mobile Communications – Railway</i>	3, 5
ID: Identificador.....	11
OBU: <i>On Board Unit</i>	passim
PLMN: <i>Public Land Mobile Network</i>	3
RI: Restrições de Integridade.....	11
Spring MVC: <i>Spring Model View Controller</i>	7
SQL: <i>Structured Query Language</i>	16
URI: <i>Uniform Resource Identifier</i>	11
URL: <i>Uniform Resource Locator</i>	13, 14, 16

1. Introdução

1.1. Enquadramento

Os comboios são, desde sempre, grandes utilizadores de sistemas de comunicações móveis, devido à necessidade permanente de troca de informação entre os maquinistas e restante pessoal ligado à operação ferroviária. As redes de comunicações ferroviárias são consideradas redes críticas devido à sua importância na gestão da exploração ferroviária. Para além de constituírem uma ajuda à exploração, transportam atualmente informação de sinalização que permite efetuar o comando de controlo de circulação de uma forma mais eficaz e eficiente, assegurando uma elevada fiabilidade das infraestruturas de transporte.

Assim, a monitorização permanente da qualidade de serviço em redes de comunicações móveis ferroviárias é fundamental para a deteção de alterações que coloquem em causa o serviço de comunicações de apoio à exploração. Uma das formas mais eficazes de monitorização de qualidade de serviço é baseada em equipamentos colocados a bordo dos comboios (sondas) que realizam de forma autónoma e permanente a monitorização, ativa e passiva, aos sistemas de comunicações móveis ferroviários.

A gestão das sondas terá de ser assegurada por sistemas de informação que permitam disponibilizar um conjunto de interfaces aos administradores do sistema e gestores de infraestrutura, permitindo a monitorização permanente da qualidade de serviço e a configuração e administração de todo o conjunto de sondas instaladas nos comboios.

1.2. Objetivos

O projeto consiste no desenvolvimento de um sistema de informação que permita realizar a administração de sondas e análise de dados proveniente das sondas embarcadas nos comboios. Este sistema e aplicação Web, deverá permitir a análise e o tratamento de informação proveniente das sondas de forma a permitir ao utilizador uma observação fácil e objetiva dos dados e administração do sistema.

Para além dos dados enviados pela sonda, o sistema deverá igualmente assegurar a gestão do equipamento, receção de alarmística e gestão de configurações, permitindo efetuar de forma permanente a monitorização das próprias sondas.

1.3. Organização do Documento

Este documento encontra-se dividido em cinco capítulos. O primeiro capítulo enquadra o problema, descreve as motivações e apresenta os objetivos do projeto. No segundo capítulo, apresenta-se a abordagem ao presente projeto assim como a descrição da arquitetura da solução proposta. No terceiro capítulo, descreve-se a implementação de cada componente do projeto. No quarto capítulo, apresenta-se o planeamento de tarefas já realizadas e a realizar. No quinto e último capítulo, encontram-se referências relativas ao conteúdo presente neste relatório.

2. Descrição do sistema e Solução Proposta

De forma a compreender a solução desenvolvida, é fundamental a apresentação do sistema alvo do projeto, particularmente os aspetos envolvidos no desenho e especificação da solução criada.

2.1. Caracterização do sistema

O conjunto de sondas embarcadas, designadas por OBU, tem a sua administração e monitorização realizada de forma centralizada por um sistema de informação. Este sistema, além de efetuar a administração das sondas, terá igualmente como função servir de repositório de informação e disponibilizar as interfaces necessárias à análise dos dados recolhidos.

De um modo geral, a arquitetura e caracterização técnica do sistema automático de monitorização de qualidade de serviço assenta nos seguintes pressupostos:

- Existência de módulos autónomos a bordo de comboios para efetuar a recolha automática de medidas de desempenho de qualidade de serviço;
- As medidas incidirão sobre serviços de voz e de dados;
- Os resultados das medidas serão georreferenciados e guardados numa base de dados do operador, de forma a possibilitar uma análise posterior;
- A rede alvo será o GSM-R [1], mas também poderão ser outras redes;
- Existência de um sistema central, que alojará um repositório contendo todas as informações recolhidas bem como os algoritmos de análise de informação.

2.1.1. Arquitetura

Na Figura 1 é representada a arquitetura geral do sistema proposto. O sistema é constituído por um conjunto de unidades móveis de recolha de medidas, apresentado na Figura 2, e pelo sistema central que envolve todas as aplicações e ferramentas necessárias à operação e gestão das OBUs e do sistema global. As medidas efetuadas, bem como toda a comunicação com o sistema central, serão suportadas pela rede móvel GSM-R [1] e/ou PLMN [2] .

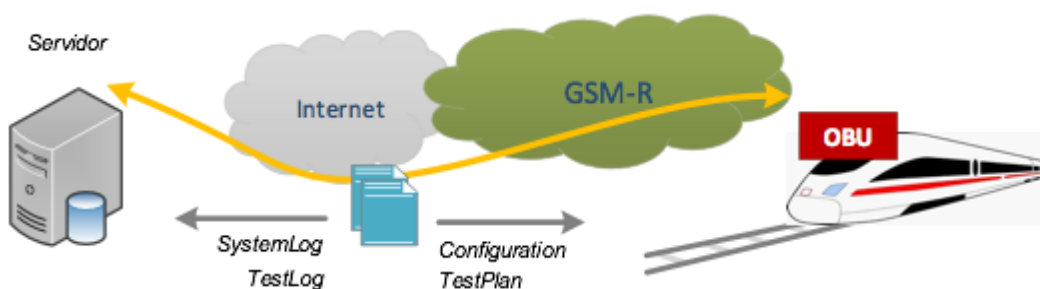


Figura 1 – Arquitetura geral do sistema.

A função principal será a recolha de medidas para estimar a qualidade de serviço da rede. Estas medidas incidirão sobre os serviços de voz e de dados. Todos os resultados serão associados a uma referência temporal e geográfica e guardados temporariamente no disco local. Periodicamente, de acordo com a hora definida na configuração, serão enviados os resultados para o sistema central, para posterior registo na base de dados.

O envio de informação de, e para as sondas é efetuado através de ficheiros, alojados num servidor central. Os ficheiros serão divididos nos seguintes tipos:

- Configuração: ficheiro recebido pela OBU que contém toda a informação de configuração da sonda;
- Plano de Testes: ficheiro recebido pela OBU que contém um plano de testes e toda a informação de configuração dos equipamentos durante o mesmo;
- Log do Sistema: ficheiro enviado pela OBU para o servidor que contém um registo do funcionamento da OBU;
- Log de Teste: ficheiro enviado pela OBU para o servidor que contém o registo das informações recolhidas durante a execução dos testes.



Figura 2 – Exemplo de uma OBU.

2.1.2. Configuração

A OBU contém uma configuração, que contém todos os parâmetros do software que podem ser parametrizáveis, tais como:

- Configuração da ligação à rede da sonda;
- Configuração das *control connections*;
- Configuração do GPS;
- Configuração do *scanning*;
- Configuração dos testes;
- Configuração do serviço de voz e de dados;

- Configuração de *upload* e *download* de ficheiros;
- Configuração do arquivo do sistema;

Uma configuração tem uma data de ativação, sendo descontinuada por uma com uma data de ativação superior. Na sequência de uma *control connection* pode ser pedido à sonda para fazer *download* de novas configurações.

2.1.3. Plano de Testes

Um Plano de Testes especifica o conjunto de testes que uma OBU deverá realizar sobre a rede GSM-R. Um plano de testes contém a seguinte parametrização dos testes a serem executados:

- Configuração de data de início e data de fim;
- Configuração do período;
- Configuração de *setups*;

Apesar de terem uma data de início, no caso ferroviário, em que a deslocação é feita em linha, pode haver necessidade de forçar que os testes apenas sejam executados dentro de uma determinada área, assim, podem ser definidas 2 posições delimitadoras da execução dos testes, designado por *geofence*. Se não for despoletado por *geofence*, o plano de testes pode ter um padrão temporal de repetição associado.

Um plano de testes é constituído por vários *Setups* (um por cada tipo de modem) e cada um contém parametrização de *scanning*. Na sequência de uma *control connection* pode ser pedido à sonda para fazer *download* de planos de teste.

2.1.4. Requisitos do sistema

Em termos de requisitos funcionais, foi definido um conjunto de aspetos funcionais a que o sistema deverá responder.

Em termos de requisitos obrigatórios temos:

- Aplicação Web *Single-Page* com integração de mapas;
- Gestão de OBUs:
 - Gestão de grupos
 - Gestão de hardware
 - Gestão de configurações

- Gestão de planos de teste
- Gestão de ligações
- Apresentação dos dados:
 - Mapas
 - Gráficos
 - Logs
- Gestão de utilizadores
 - Registo de utilizadores
 - Autenticação de utilizadores
 - Gestão de perfis de utilizadores
- Pesquisa de dados por nome, data, local, etc.

Para além dos requisitos funcionais mínimos definidos anteriormente, foi definida a possibilidade de efetuar a monitorização de sondas em tempo real, caso possível.

2.2. Solução Proposta

A solução proposta passa pelo desenvolvimento de uma Web API para acesso aos dados reportados pelo conjunto de sondas existente e apresentação de resultados de medidas e testes aos utilizadores, bem como possibilitar a administração das mesmas. A informação proveniente das sondas é atualmente armazenada numa base de dados.

A informação resultante dos testes deverá ser apresentada de forma georreferenciada, sobre mapas e através de gráficos. Para disponibilização do serviço de mapas e gráficos serão utilizadas as APIs Leaflet [3] e Chart.js [4], respetivamente.

De forma a que o cliente tenha acesso aos dados e informações terá de estar registado no sistema de informação, pelo que a solução inclui igualmente toda a componente de gestão de utilizadores e perfis.

2.2.1. Arquitetura do Sistema

Para a realização desta aplicação é necessário desenvolver os componentes da arquitetura, apresentados na Figura 3.

- **Componente Cliente** - A componente cliente é acedida pelo utilizador através do *browser* e vai ser responsável pela apresentação e gestão dos dados - vai ser desenvolvida utilizando a

linguagem de programação TypeScript [5] com base na *framework* AngularJS [6] e a ferramenta de desenvolvimento Visual Studio Code [7]. Esta componente vai disponibilizar a informação já devidamente tratada usando a informação oferecida pela componente servidora.

- **Componente Servidora** - A componente servidora constitui o meio utilizado pela aplicação cliente para obter, publicar e atualizar dados (*front-office*) e como meio de comunicação das OBU's com a base de dados (*back-office*). Estas operações são realizadas através da interação com a base de dados relacional.

Quanto à API, vai ser desenvolvida utilizando a linguagem de programação Java [8] com base na *framework* Spring MVC [9] que permite simplificar e auxiliar o desenvolvimento da API.

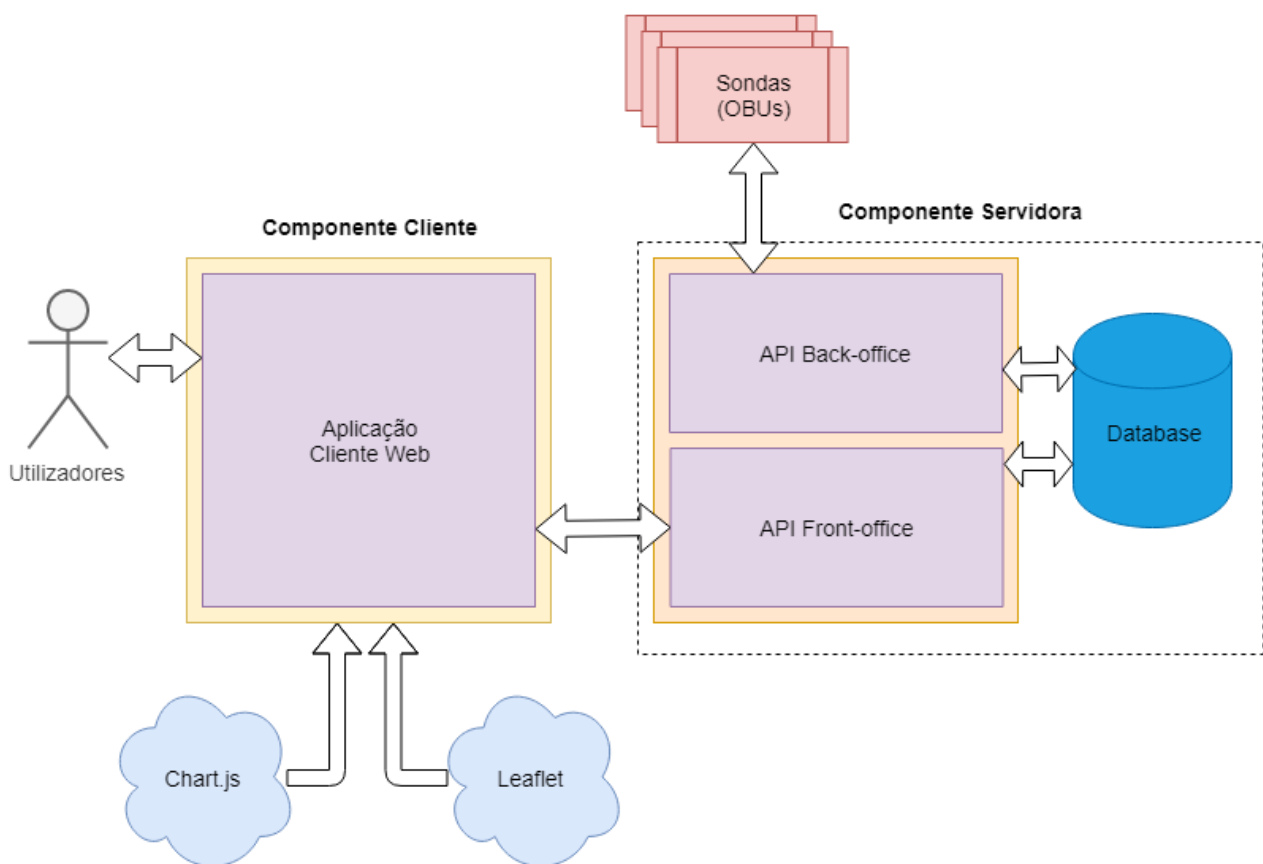


Figura 3 – Arquitetura da solução.

3. Implementação

A implementação do sistema foi dividida nos componentes identificados anteriormente.

3.1. Base de Dados

A implementação da Base de Dados foi feita em PostgreSQL [10]. O modelo de dados teve por base o sistema de informação existente, sendo que foi necessária uma adaptação ao sistema a desenvolver, adicionando-se informação.

3.1.1. Modelo Concetual

Na figura 4 é apresentado o modelo de Entidade-Associação (EA).

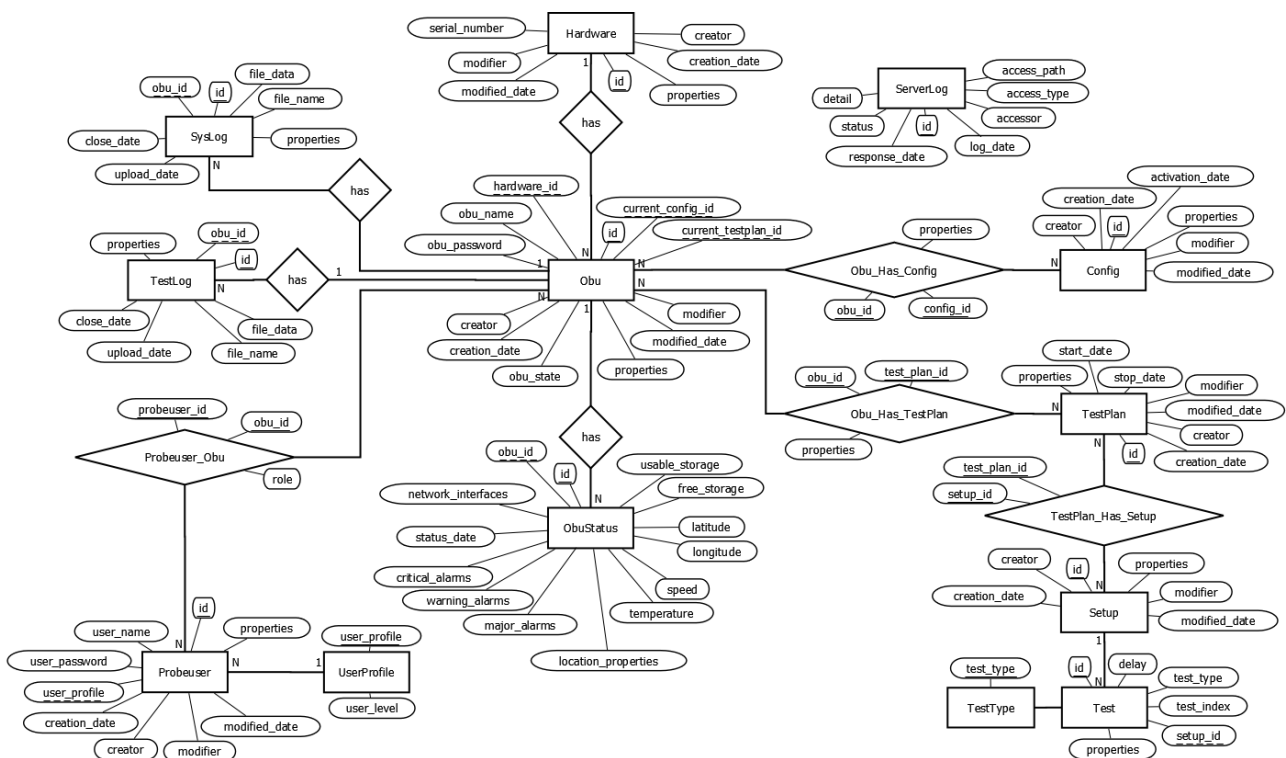


Figura 4 – Modelo EA.

Uma das componentes adicionadas ao sistema já existente está relacionada com a gestão de utilizadores para acesso à aplicação desenvolvida.

A tabela Probeuser representa um utilizador e toda a sua informação correspondente, desde o seu *username* ao seu criador, passando pelo seu nível de utilizador (*profile*), que está associado á tabela UserProfile, que concede, ou não, acesso a informações e ações mais específicas no contexto da aplicação.

A tabela ProbeUser_Obu representa a associação entre um utilizador e uma OBU. Esta tabela contém o campo *role* que estabelece se o utilizador é *viewer* ou *editor* de uma OBU específica.

A tabela Obu representa uma sonda e tem como objetivo principal guardar a informação referente ao seu estado, configuração atual e plano de testes atual. Todas as outras informações pertinentes relativamente a uma sonda encontram-se nas restantes tabelas associadas a esta:

- Hardware

Esta tabela representa o hardware sobre o qual a sonda está a operar. Guarda informações referentes aos seus componentes tais como tipo de componente, serial, modelo e fabricante.

- Config

Esta tabela representa uma configuração e tem como objetivo principal guardar informações referentes às diferentes opções de configuração sendo estas já referidas no tópico Configuração no Capítulo 2.

A tabela Obu_Has_Config representa a associação entre uma OBU e uma configuração.

- TestPlan

Esta tabela representa um plano de testes e tem como objetivo principal guardar informações referentes às diferentes opções de plano de teste sendo estas já referidas no tópico Plano de Testes no Capítulo 2.

A tabela Obu_Has_TestPlan representa a associação entre uma OBU e um plano de testes.

- ObuStatus

Esta tabela representa um conjunto de dados “de estado” reportado pela OBU. Estes dados são referentes a uma determinada data e representam informações tais como latitude, longitude, velocidade, temperatura, alarmes e armazenamento.

- SysLog e TestLog

Estas tabelas representam ficheiros com informações referentes a *logs* reportados pelas OBUs, respetivamente, sobre o estado do sistema e sobre os testes que foram efetuados.

A tabela Setup guarda informações, relativas ao tipo de modem e ao *scanning*, a serem usadas num plano de testes.

A tabela TestPlan_Has_Setup representa a associação entre um plano de testes e um Setup.

A tabela *ServerLog* guarda dados relativos aos logs da API, tais como, caminho de acesso (URI [11]), tipo de acesso (utilizador ou OBU), data do log, data da resposta e *status code* [12].

Foi também adicionado às tabelas que permitem edições por parte dos utilizadores os campos *Modifier* e *Modified_Date* para que fique guardado o último utilizador que as modificou assim como a sua última data de modificação.

Foram criadas algumas vistas na Base de Dados que auxiliaram no desenvolvimento da mesma. Mesmo assim só uma delas é usada pela API: *view_Probeuser_Obu* que é usada para retornar todas as associações entre utilizadores e OBUs com os *IDs* e os nomes do utilizador e da OBU.

Nesta componente também foram desenvolvidos três *triggers*, mas com as regras de negócio impostas depois de finalizada esta componente, dois destes encontram-se inúteis. Todavia, deve-se destacar o terceiro *trigger*, que é acionado quando o nome de um utilizador é atualizado, das seguintes formas:

- Os campos *creator* e *modifier* não se encontram relacionados com nenhum utilizador (não são *Foreign Keys*) aquando de um *update* ao nome do utilizador não são alterados. Como foi decidido manter inalterado o máximo possível do modelo de dados não foram inseridas estas *constraints*, daí termos achado necessário implementar este *trigger*.
- Na população inicial da base de dados ou uma inserção “à mão/direta” estes campos não verificam se o nome do *creator* ou *modifier* existem e pertencem a algum utilizador, logo se o nome de um utilizador for atualizado para o nome que está nesses campos, estes irão passar “a ser dele”.

Pois toda a lógica de negócio é feita na API.

3.1.2. Restrições de Integridade e Lógica de Negócio

Como restrições de integridade o *serial_number* da tabela *Hardware*, o *user_name* da tabela *Probeuser* e o *user_profile* da tabela *UserProfile* são *unique*. Existem também três enumerados: *OBUState*, *PO_Role* e *AccessType*, que não se encontram no modelo EA, pois foi pensado serem apenas tratados como RIs.

Certas páginas, como por exemplo, as páginas das listas de utilizadores ou lista de hardwares, não podem ser acedidas por utilizadores cujo nível não seja *Admin*, já outras, como a *Home Page*, podem ser acedidas por qualquer utilizador. Embora haja páginas que todos os utilizadores podem ver, algumas ações só estão disponíveis a utilizadores com maiores privilégios.

De seguida apresenta-se uma tabela com as diferentes ações que cada *profile* de utilizador pode realizar:

Tabela 1 – Permissões utilizadas no sistema.

		Normal	Super	Admin
Server Log	List	×	×	✓
User	List/View	×	×	✓
	Create	×	×	✓
	Delete	×	×	✓
Hardware	List/View	×	×	✓
	Create	×	×	✓
	Edit	×	×	✓
	Delete	×	×	✓
OBU	List/View (inclui Status, Configs, TestLogs e SysLogs)	✓	✓	✓
	Create	×	×	✓
	Edit	×	×	✓
	Delete	×	×	✓
	Associate/Remove/Cancel Config or Test Plan	×	×	✓
	Associate to User	×	×	✓
Config	List/View	✓	✓	✓
	Create	×	✓	✓
	Edit	×	✓	✓
	Delete	×	✓	✓
Test Plan	List/View	✓	✓	✓
	Create	×	✓	✓
	Edit	×	✓	✓
	Delete	×	✓	✓
Setups	List/View	✓	✓	✓
	Create	×	✓	✓
	Edit	×	✓	✓
	Delete	×	✓	✓

✓	Tem acesso
✓	Tem acesso apenas se estiver associado à OBU
✓	Tem acesso apenas se for o criador

Na associação entre OBU e utilizador, foi decidido manter o *role* que o utilizador tem sobre a OBU. Atualmente, este campo não está a ser utilizado, uma vez que depois de aplicadas as regras de negócio (referidas na tabela acima) sobre os *profiles* que podem ou não fazer alterações a uma OBU (só Admins) deixou de ter propósito, mas a decisão de o manter na tabela deve-se ao facto das regras de negócio poderem ser alteradas no futuro. Devido a esta decisão, de momento, não há diferença entre *viewers* e *editors*. Ainda assim para associar um utilizador a uma OBU é necessário indicar o *role*.

Optou-se pela opção de suspender um utilizador e não de o apagar pois o utilizador poderia ter criado algum tipo de configuração, plano de testes, etc., estando desta forma associado ao mesmo e no caso de um destes estar a ser utilizador por uma OBU ficaríamos com uma inconsistência.

3.2. API

A implementação da API teve por base uma API realizada posteriormente pelos orientadores para ser desenvolvida pelo grupo no decorrer deste projeto. Ainda assim, foi feita uma análise da informação que seria necessário disponibilizar para o exterior, de acordo com as funcionalidades que se iriam implementar e, tendo em atenção que o Cliente Web pode não ser o único a consumir essa informação.

A API dispõe de um sistema de versões, de forma a impedir que futuras alterações quebrem a compatibilidade com aplicações que estejam a usar versões mais antigas. Para suportar esta implementação foi adicionado ao URL base da API o número da versão.

O tipo de autenticação escolhido no desenvolvimento desta componente foi *Basic Authentication* que está especificado no RFC 7617 [13]. Este tipo de autenticação não oferece confidencialidade, logo a aplicação teria de ser *deployed* para um servidor que tenha um certificado SSL/TLS [14]. Como está especificado, para a implementação desta BA, cada utilizador possui um *token*, que é único a si, e a maneira de o obter é fazendo *login*. Após efetuado o *login* e obtido o *token*, todos os pedidos a recursos necessitam que o *token* esteja presente no cabeçalho *Authorization*, exceto, obviamente, o recurso para fazer *login*.

Esta componente garante que as respostas aos pedidos, que sejam completados com sucesso (não significando bem-sucedidos), cujos *status codes* indiquem um erro são do tipo (*Content-Type* [15]) *application/problem+json*. Este tipo de resposta está especificado no RFC 7807 [16].

Quando é feito um *request* à API os controladores apanham esse *request*. Estes controladores delegam trabalho aos serviços que fazem pedidos aos repositórios, estes, por sua vez, ligam-se e fazem consultas e atualizações à base de dados. Com estes três tipos de classes (*Controllers*, *Services* e *Repositories*) é possível organizar todos os passos, verificações e acessos a dados necessários para responder a um *request*.

Devido à similaridade destas classes e à metodologia que estas seguem, será efetuada uma breve descrição sobre as mesmas juntamente com os URLs suportados pela API.

3.2.1. Controladores

Os controladores são os últimos componentes que têm acesso ao *request* e verificam se as variáveis do *path* e qualquer *body* que venha no *request* são válidos. Também é aqui que a resposta é construída

e enviada (caso esta tenha *status code* 200/ok). Para efeitos de exemplo, de seguida encontra-se uma descrição detalhada das funções que compõem o UserController:

- getAllUsers - método *GET* - *path* “/users”
 - Esta função chama o UserService e a lista de utilizadores que este retorna é adicionada ao *body* da *response*.
- getUser - método *GET* - *path* “/user/{user-id}”
 - Esta função chama o UserService, com o user-id do *path*, e o utilizador que este retornar é adicionado ao *body* da *response*.
- createUser - método *POST* - *path* “/user”
 - Esta função faz a validação do *body* do *request*, chama o UserSevice para criar e para buscar o novo utilizador. Este novo utilizador é adicionado ao *body* da *response*.
- updateUser - método *PUT* - *path* “/user/{user-id}”
 - Esta função faz a validação do *body* do *request*, chama o UserService, com o user-id passado no *path* e o *body*, para procurar, fazer *update* e, novamente, para procurar o utilizador já *updated*. Este utilizador *updated* é adicionado ao *body* da *response*.
- suspendUser - método *PUT* - *path* “/user/{user-id}/suspend”
 - Esta função chama o UserService, com o user-id passado no *path* e o *body*, para procurar, fazer a suspensão e, novamente, para procurar o utilizador já suspenso. Este utilizador suspenso é adicionado ao *body* da *response*.

3.2.2. Serviços

Os serviços são responsáveis por verificações, neste sentido, e a título de exemplo, verifica se o utilizador que fez o *request* tem permissão/perfil alto o suficiente para aceder ao recurso. Estes serviços são também responsáveis por transformar os *Data Transfer Objects* em *Data Access Objects* e vice-versa para os repositórios e os controladores.

Para efeitos de exemplo, de seguida encontra-se uma descrição detalhada das funções que compõem o UserService:

- createUser
 - Inicialmente, esta função verifica se o nome do utilizador a inserir é *unique* em relação aos utilizadores já registados e se o utilizador que fez o *request* tem permissões/perfil

suficiente para criar um utilizador. Em seguida transforma o *body* em *User Object* e chama o *UserRepository* retornando o *id* do utilizador criado (que vem do repositório).

- *updateUser*
 - Inicialmente, esta função verifica se o utilizador que fez o *request* tem permissões/perfil suficiente para fazer *update* a um utilizador. Um utilizador só pode fazer *update* a outro, se o outro tiver perfil mais baixo que o utilizador que lhe quer fazer *update*. Finalmente transforma o *body* em *User Object* e chama o *UserRepository*.
- *getUser*
 - Inicialmente, esta função verifica se o utilizador que fez o *request* tem permissões/perfil suficiente para pesquisar um utilizador e, finalmente, chama o *UserRepository*, com o *id* que veio no *path*, retornando o utilizador.
- *getAllUsers*
 - Inicialmente, esta função verifica se o utilizador que fez o *request* tem permissões/perfil suficiente para procurar os utilizadores e, finalmente, chama o *UserRepository* retornando os utilizadores.
- *deleteUser*
 - Inicialmente, esta função verifica se o utilizador que fez o *request* tem permissões/perfil suficiente para apagar um utilizador e se o utilizador a apagar não é um *admin* (não é possível apagar *admins*). Finalmente, chama o *UserRepository*, com o *id* que veio no *path*.
- *suspendUser*
 - Inicialmente, tal como *updateUser*, esta função verifica se o utilizador que fez o *request* tem permissões/perfil suficiente para suspender um utilizador. Um utilizador só pode fazer suspender outro, se o outro tiver perfil mais baixo que o utilizador que o quer suspender. De seguida, a suspensão é *toggled* e o utilizador é *updated*.

3.2.3. Repositórios

Os repositórios contêm as *query strings* de SQL e são responsáveis por fazer as conexões, consultas e atualizações à base de dados.

3.2.4. Intercetores

Os intercetores são responsáveis por intercetar todos os *requests* cujos *paths* estão subscritos (por exemplo o *AuthInterceptor*, descrito abaixo, não está subscrito ao *path* do login, logo não o interceta). O objetivo dos intercetores é “processar” o *request* (pelo menos uma pequena parte) antes ou depois deste passar pelo controlador. À frente encontra-se a descrição dos dois intercetores desenvolvidos.

- **AuthInterceptor** - este intercetor é responsável pela verificação do *token* e do utilizador associado, em qualquer *request* feito (exceto o *post login*) antes deste ser processado. Aqui é verificado se o *token* está bem construído e pertence a algum utilizador válido e, caso pertença, se este utilizador está ou não suspenso.
- **LoggingInterceptor** - Este intercetor é responsável pelo *logging* que é gravado na componente servidora. Este faz inserções na tabela *ServerLog* onde guarda os detalhes de todos os *requests* e as suas respetivas respostas (ou seja, por exemplo, URLs e status codes). Ele usa o *request_date* que vem no *request* ou, caso não exista este cabeçalho, insere o tempo a que o *request* chegou à API.

3.2.5. Filtro

Foi desenvolvido um filtro para inserir os cabeçalhos do CORS.

3.2.6. Rotas disponibilizados

De seguida apresenta-se na tabela 2 os endereços específicos da API (*endpoints*) a disponibilizar.

O URL base é dado por: `http://{servidor:porto}/api/v1/frontoffice`

Tabela 2 – Rotas disponibilizados pela API

Método	Pedido HTTP	Descrição
POST	/login	Retorna o token para efectuar o login
GET	/login/user	Retorna o utilizador que está <i>logged in</i>
GET	/users	Retorna todos os utilizadores existentes
GET	/user/{id}	Retorna o utilizador referente ao id
POST	/users	Cria um utilizador
PUT	/user/{id}	Atualiza o utilizador referente ao id
PUT	/user/{id}/suspend	Suspende o utilizador referente ao id
GET	/obu-user/user/{id}	Retorna todas as associações utilizador (referente ao id) OBU
POST	/obu-user	Cria uma associação entre uma OBU e um utilizador
DEL	/obu/obu/{obu-id}/user/{user-id}	Apaga a associação utilizador OBU referente aos ids
GET	/hardware	Retorna todos os hardwares
GET	/hardware/{id}	Retorna o hardware referente ao id
POST	/hardware	Cria um hardware
PUT	/hardware/{id}	Atualiza o hardware referente ao id
DEL	/hardware/{id}	Apaga o hardware referente ao id
GET	/obu	Retorna todas as OBUs
GET	/obu/{id}	Retorna a OBU referente ao id
POST	/obu	Cria uma OBU
PUT	/obu/{id}	Atualiza a OBU referente ao id
DEL	/obu/{id}	Apaga a OBU referente ao id
GET	/obu/{obu-id}/config	Retorna as configs da OBU referente ao id
POST	/obu/{obu-id}/config{config-id}	Adiciona a config à OBU referentes aos ids

DEL	/obu/{obu-id}/config{config-id}	Tira a config à OBU referentes aos ids
GET	/obu/{obu-id}/test-plan	Retorna os test plans da OBU referente ao id
POST	/obu/{obu-id}/test-plan{test-plan-id}	Adiciona o test plan à OBU referentes aos ids
DEL	/obu/{obu-id}/test-plan{test-plan-id}	Tira o test plan à OBU referentes aos ids
GET	/obu/{id}/test-log ? order=«boolean» filename=«string» page=«integer» limit=«integer»	Retorna todos (ou uma página) os test logs da OBU referente ao id e o número total de logs existentes order: ascendente (true) ou descendente (false) (descendente por omissão) filename: nome do ficheiro (completo ou incompleto) page: número da página limit: número de itens por página page e limit devem têm de ser ambos passados para uma resposta paginada
GET	/obu/{id}/system-log ? order=«boolean» filename=«string» page=«integer» limit=«integer»	Retorna todos (ou uma página) os system logs da OBU referente ao id e o número total de logs existentes order: ascendente (true) ou descendente (false) (descendente por omissão) filename: nome do ficheiro (completo ou incompleto) page: número da página limit: número de itens por página page e limit devem têm de ser ambos passados para uma resposta paginada
GET	/obu/{id}/status	Retorna os status da OBU referente ao id
GET	/config	Retorna todas as configs
GET	/config/{id}	Retorna a config referente ao id
POST	/config	Cria uma config
PUT	/config/{id}	Atualiza a config referente ao id

DEL	/config/{id}	Apaga a config referente ao id
GET	/test-plan	Retorna todos os test plans
GET	/test-plan/{id}	Retorna o test plan referente ao id
POST	/test-plan	Cria um test plan
PUT	/test-plan/{id}	Atualiza o test plan referente ao id
DEL	/test-plan/{id}	Apaga o test plan referente ao id
GET	/test-plan/{test-plan-id}/setup	Retorna todos os setups (vista reduzida) que estão associados ao test plan (referente ao id)
POST	/test-plan/{test-plan-id}/setup/{test-plan-id}	Adiciona o setup ao test plan referentes aos ids
DEL	/test-plan/{test-plan-id}/setup/{test-plan-id}	Tira o setup ao test plan referente aos ids
GET	/setup	Retorna todos os setups
GET	/setup/{id}	Retorna o setup referente ao id
POST	/setup	Cria um setup
PUT	/setup/{id}	Atualiza o setup referente ao id
DEL	/setup/{id}	Apaga o setup referente ao id
GET	/server-log ? order=«boolean» accessType=«string» accessor=«string» page=«integer» limit=«integer»	Retorna todos (ou uma página) os server logs e o número total de logs existentes order: ascendente (true) ou descendente (false) (descendente por omissão) accessType: tipo de acesso (USER ou OBU) accessor: nome de quem acede (completo ou incompleto) page: número da página limit: número de itens por página page e limit devem têm de ser ambos passados para uma resposta paginada

3.3. Aplicação Cliente

Este componente contém a interface gráfica apresentada ao utilizador. A Aplicação Cliente Web foi desenvolvida com o objetivo de disponibilizar uma experiência simples e objetiva ao utilizador.

Como esta componente está separada da componente servidora foi necessário implementar restrições para os utilizadores não conseguirem fazer ações que resultariam em “erro” devido, por exemplo, a falta de permissões por parte do utilizador, garantindo assim uma experiência intuitiva.

Como dito anteriormente, os dados apresentados na interface são obtidos através da comunicação com a componente servidora.

3.3.1. Intercetores:

- **AuthGuard** - Na implementação da componente cliente foi desenvolvido um AuthGuard que implementa a interface CanActivate [17]. Todos os componentes que sejam necessários estar “guardados” precisam de especificar que usam a classe AuthGuard. Esta classe vai verificar se o utilizador: está *logged in*, se tem permissões/perfil/estatuto (*user_profile*) para ver a página pedida e se este não se encontra suspenso. Caso o utilizador não esteja *logged in* ou esteja suspenso este será, automaticamente, *logged out* e redirecionado para a página de login. Caso o utilizador não tenha permissões para ver a página este será redirecionado para a página *home*.
- **HttpRequestInterceptor**. Esta classe implementa a interface HttpInterceptor [18]. Assim esta classe interceta qualquer chamada feita à API. Este foi criado com o propósito de adicionar, ao *request*, os cabeçalhos necessários que a API requer, tal como o cabeçalho *Authorization* com o *token* do utilizador, mas também é usado para error handling de certos erros, tal como o de o utilizador estar suspenso ou de ter introduzido credenciais erradas no ato de *login*.

3.3.2. Serviços

Os serviços são responsáveis pela gestão de recursos e *requests* a fazer à API.

- **Auth** - Este serviço é responsável por fazer *requests* à API que estejam relacionados com a autenticação do utilizador. Funções, tais como, fazer o login e o logout e ver se o perfil do utilizador é suficientemente alto, são aqui implementadas.
- **LocalStorage** - Este serviço é responsável por gerir a LocalStorage [19] do *website*.
- **Background** - Este serviço é responsável por fazer *requests* à API periodicamente para, por exemplo, saber se o utilizador está suspenso. Foi decidido não ser usado, pelo menos nesta fase, devido ao número de utilizadores não o justificar e a única maneira de o utilizador ser

logged out por estar suspenso é se este fizer uma ação que provoque um *request* à API ou se mudar de página.

- **User** - Este serviço é responsável por fazer *requests* à API que estejam relacionados com os utilizadores, tais como, *getUsers*, *getUserById*, *updateUser*, *createUser* ou *suspendUser*.
- **Hardware, OBU, Config, TestPlan e Setup** - Estes serviços são responsáveis por fazer *requests* à API que estejam relacionados com os componentes acima referidos e todos eles seguem a mesma ordem de ideias. Estes *requests* são *getComponent*, *getComponentById*, *createComponent*, *updateComponent* e *deleteComponent*, onde *Component* representa o componente em questão. No caso do serviço da OBU temos também disponível uma função *getStatusFromOBU* que faz um pedido relativo à informação/*Status* da OBU.
- **UserHasOBU, OBUHasConfig, OBUHasTestPlan e TestPlanHasSetups** – Estes serviços são responsáveis por fazer *requests* à API que estejam relacionados com as associações Utilizador/OBU, OBU/Configuration, OBU/Test Plan e Test Plan/Setups, respetivamente. São pedidos *get*, *create* e *delete*, e como tal não é permitido ao utilizador editar associações. Neste caso o utilizador tem de desassociar e voltar a associar o que pretender modificar.
- **ServerLog e OBULogs** - Estes serviços são responsáveis por fazer *requests* à API que estejam relacionados com Logs, sendo por este mesmo motivo apenas pedidos *get*. O *ServerLog* é para logs relacionados com a API enquanto o *OBULogs* é para os testlogs e systemlogs associados a cada OBU.

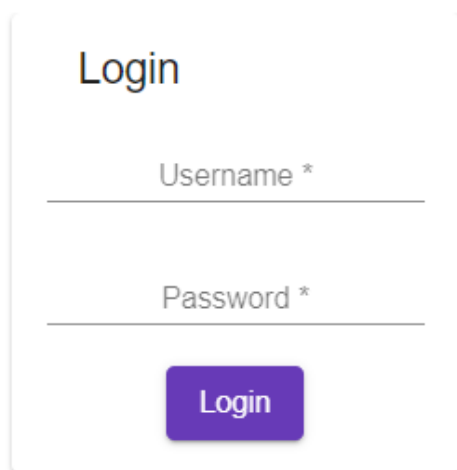
3.3.3. Componentes

Um componente *angular* fornece metadados de configuração que determinam como o componente deve ser processado, instanciado e usado em tempo de execução. Cada componente representa uma página *Web* e tem associado a si quatro ficheiros, um ficheiro *.html* [20] e um *.css* [21] que são responsáveis pela apresentação, um ficheiro *.spec.ts* e um *.ts* que são responsáveis pelas ações que podem ser efetuadas na página.

De seguida apresenta-se uma lista dos componentes e exemplos dos mesmos:

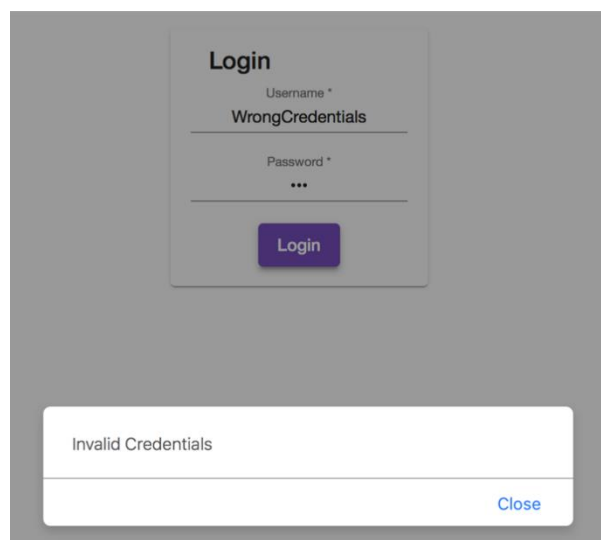
- **Login** – Este componente é responsável pela página inicial da aplicação. A página inicial da aplicação, Figura 5, permite ao utilizador iniciar sessão com o seu nome de utilizador e palavra-passe correspondente. Qualquer tentativa de acesso a outra página sem início de sessão, com sessão expirada ou se o utilizador se encontrar suspenso resulta no redirecionamento a esta página. No caso da tentativa de login com credenciais erradas, campos em falta ou o

utilizador se encontrar suspenso é emitida uma mensagem de alerta que avisa, assim, o utilizador (caso esteja suspenso a mensagem é “User is suspended”).



The image shows a clean, white login form with a light blue border. At the top, the word "Login" is displayed in a large, dark blue font. Below it, there are two input fields: "Username *" and "Password *", both with light gray placeholder text. A purple "Login" button is positioned at the bottom center of the form.

a) Login na página inicial da aplicação



The image shows the same login form as in (a), but it is overlaid on a dark gray background. The form itself is slightly dimmed. Below the form, a white rectangular box contains the text "Invalid Credentials" in a dark gray font. A blue "Close" link is located at the bottom right of this box. The "Login" button on the form is still visible and appears to be a dark purple color.

b) Tentativa de login com mensagem de alerta

Figura 5 – Interface de login da aplicação.

- **HomeMap** – Após um *login* bem sucedido, o utilizador é redirecionado para a *Home Page*, Figura 6, onde, pode escolher os parâmetros e definições que quer visualizar, tanto no mapa como no gráfico.

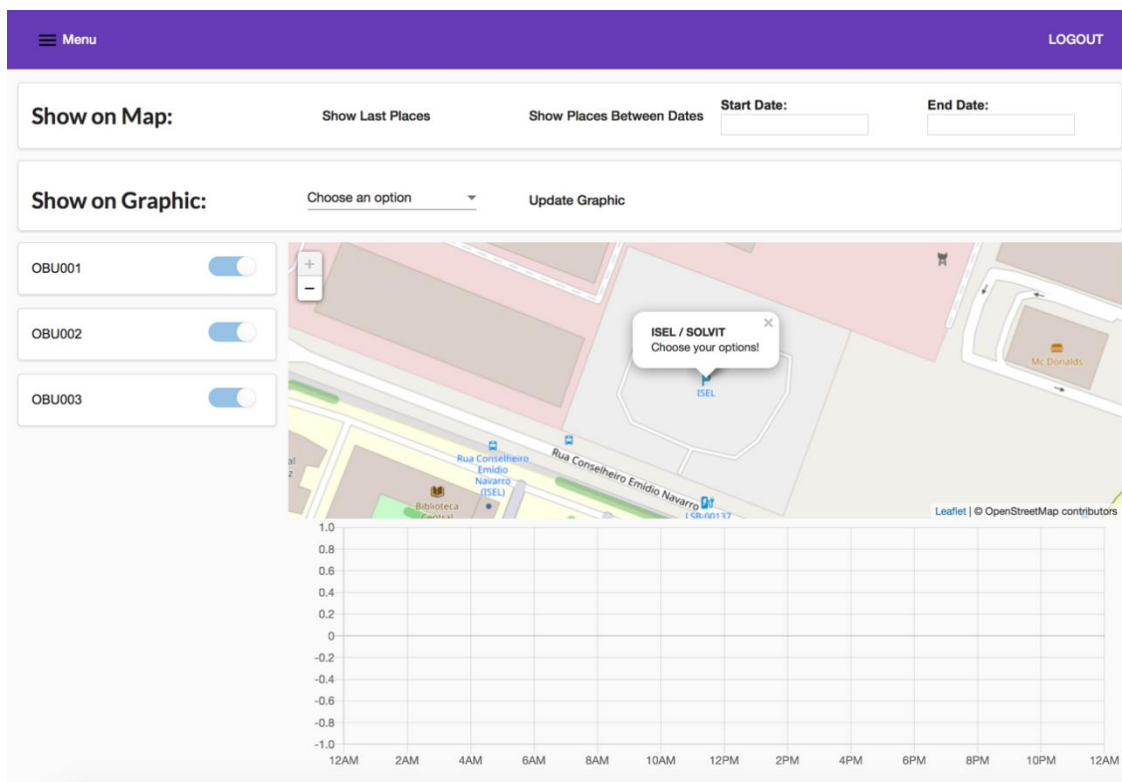


Figura 6 – Página principal da aplicação.

No mapa é possível verificar o último local onde cada OBU esteve (botão - ‘Show Last Places’) assim como verificar onde esteve cada OBU num determinado espaço de tempo indicando uma data de início e uma data de fim (botão – ‘Show Places Between Dates’), no caso de nenhuma das datas estar preenchida aquando do *click* no botão o gráfico irá mostrar toda a informação sem ser filtrada. Esta funcionalidade é realizada através da chamada ao método `getStatusFromOBU` do serviço `OBUService`, referido anteriormente, tendo como parâmetros as respetivas datas de início e fim.

Quanto ao que é apresentado no mapa, para além de ser permitido filtrar por data também é permitido filtrar por OBU o que permite ao utilizador visualizar apenas as OBUs que pretende.

A figura 7 demonstra um exemplo onde se apresenta a rota de uma OBU desde o dia 17 de maio de 2019 às 09:54 horas até às 11:54 horas desse mesmo dia. Neste caso específico existem três OBUs associadas ao utilizador corrente e apenas uma é visível no mapa, isto porque não existem dados relativos às restantes no período de tempo especificado.

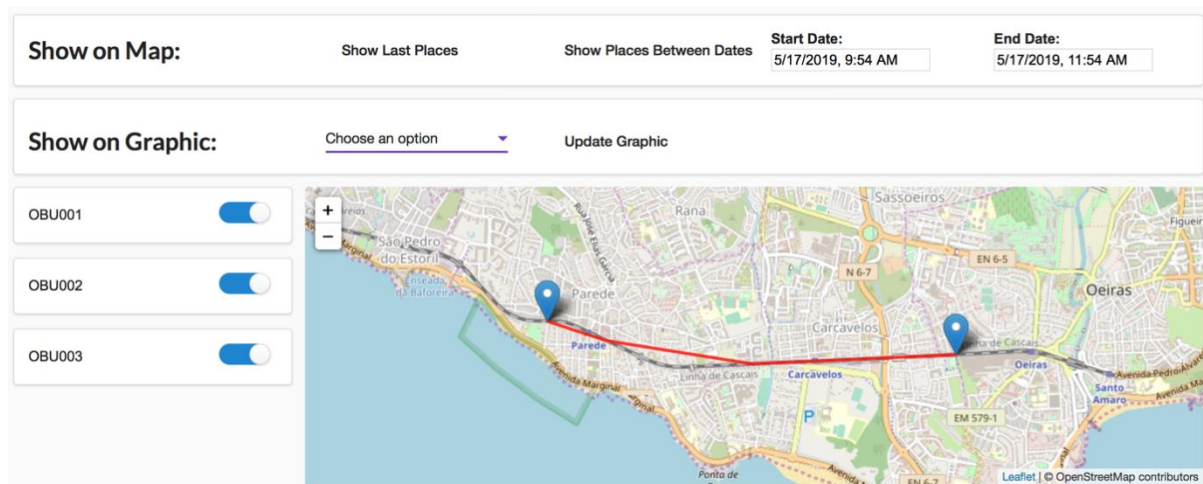


Figura 7 – Exemplo de visualização da rota da OBU num determinado período de tempo no mapa

Na Figura 8 é possível verificar informações tais como velocidade, alarmes (avisos, críticos ou principais), armazenamento (livre ou utilizável) e temperatura selecionando a opção pretendida. Os dados necessários ao preenchimento do gráfico são guardados assim que a página é inicialmente carregada.

Na seguinte figura é demonstrado um exemplo onde se apresenta a informação relativa à velocidade ao longo de vários dias. Consta-se facilmente que a velocidade mais alta foi atingida pela OBU com nome OBU002 e era cerca de 90km/hora. Também é possível verificar que a velocidade dessa mesma OBU no dia 14 de junho de 2019 pela 09:14 horas era de 51km/hora.

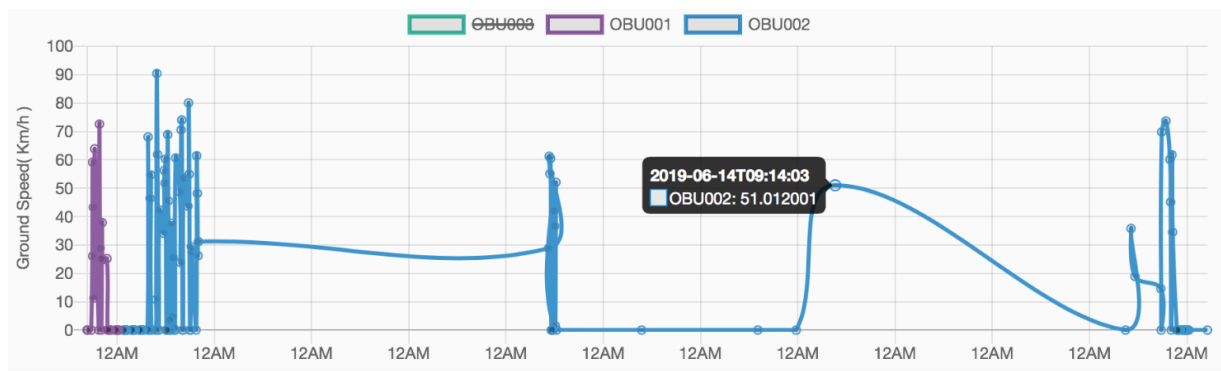


Figura 8 – Exemplo de visualização da velocidade das OBUs no gráfico

Ainda na página principal, através do menu lateral (Figura 9) é possível escolher uma das várias opções fornecidas com a finalidade de aceder a informações sobre utilizadores, Hardwares, OBUs, etc.

- **NavMenu** – Este componente acompanha o utilizador em toda a aplicação. Este é responsável pela barra de navegação e pelo menu lateral (Figura 9), que permite ao utilizador navegar para outras páginas da aplicação ou fazer *logout*, voltando, desta forma, à página de *login*. Em relação ao menu lateral, é possível escolher uma das várias opções fornecidas com a finalidade de aceder a informações sobre utilizadores, Hardwares, OBU's, etc. Estas opções podem desaparecer e não se encontrar disponíveis dependendo do nível de perfil (*Profile*) do utilizador.

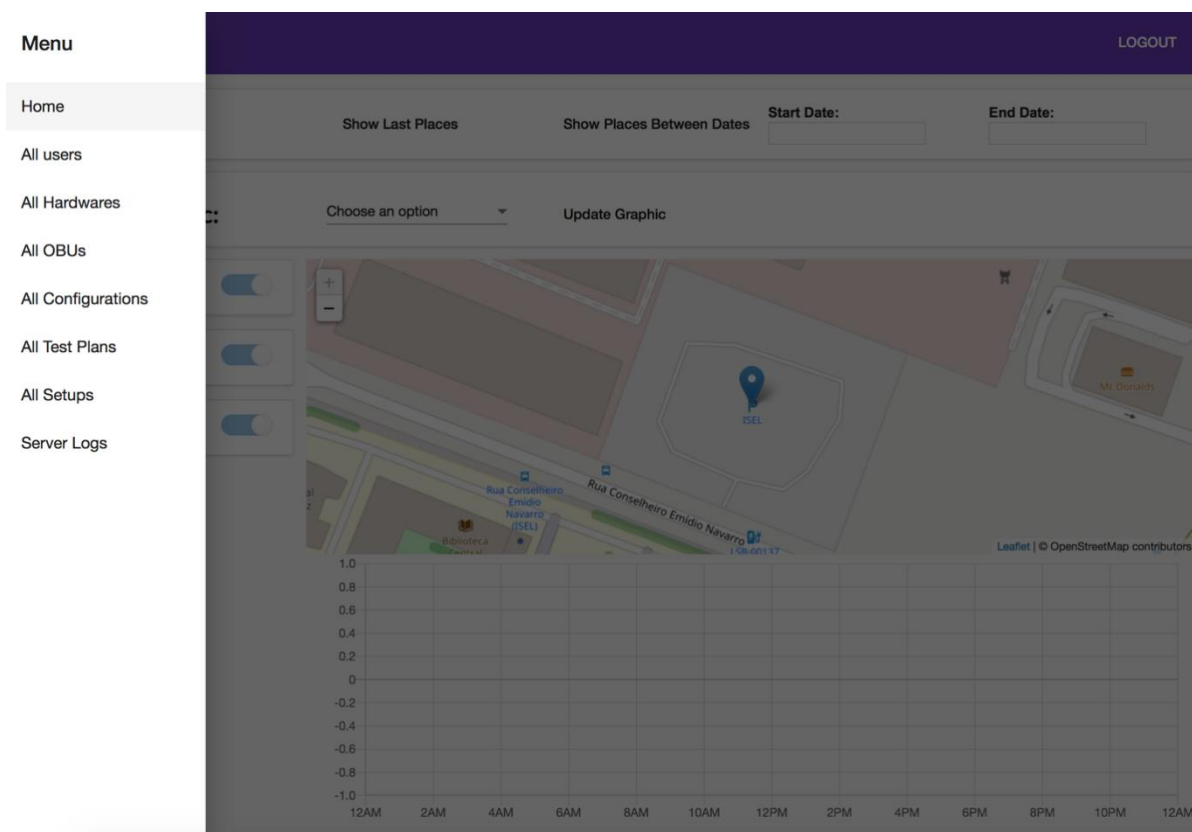
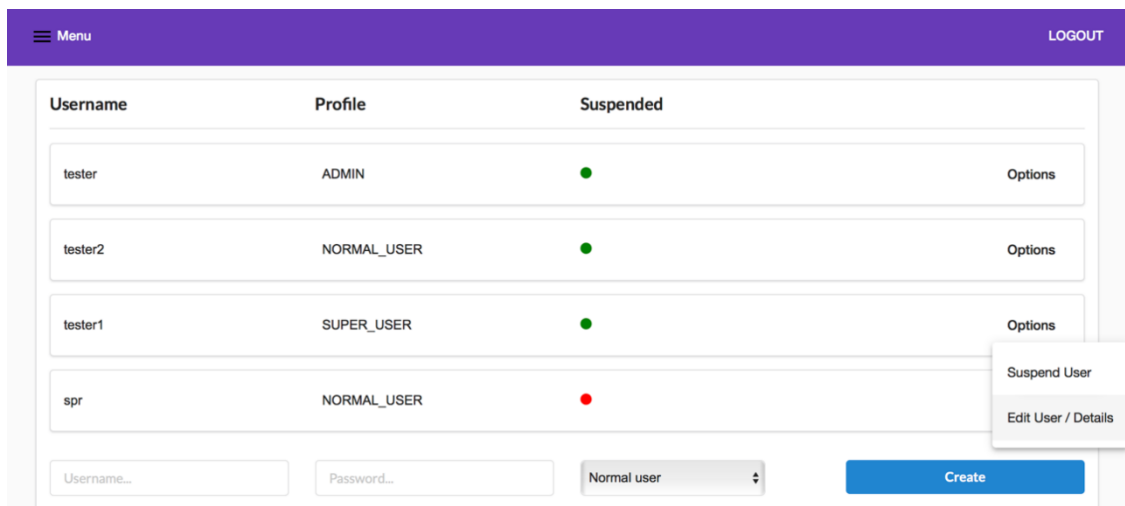


Figura 9 – Menu lateral (na página inicial)

- **User e User-Detail** – Estes componentes são responsáveis por toda a parte relacionada com os utilizadores. Ao seleccionar a opção ‘All Users’ no *menu* o utilizador é redireccionado para uma página que apresenta uma lista onde cada entrada dessa mesma lista representa um utilizador contendo o nome, tipo de perfil (*Admin*, *SuperUser* ou *NormalUser*) e informação sobre suspensões. Por cada entrada da tabela, ou seja, por cada utilizador são permitidas opções tais como suspender o utilizador, editar ou verificar informações mais detalhadas sobre o próprio. Ao suspender o utilizador enquanto este esteja a utilizar a aplicação, no momento em que o utilizador tenta aceder a uma nova página irá ser notificado e *logged out*.

Ainda nesta página, é possível criar um utilizador indicando o *username*, *password* e o seu tipo de perfil.

A Figura 10 é um exemplo do que foi acima referido e onde conseguimos também constatar que o utilizador ‘spr’ está suspenso ao contrário dos restantes assim como as diferentes opções para cada utilizador da lista (Suspend User e Edit User / Details).



Username	Profile	Suspended	
tester	ADMIN	●	Options
tester2	NORMAL_USER	●	Options
tester1	SUPER_USER	●	Options
spr	NORMAL_USER	●	Options Suspend User Edit User / Details

Username... Password... Normal user Create

Figura 10 – Página ‘All Users’

Ao aceder à página dos detalhes do utilizador, Figura 11, é possível editar campos tais como o *username*, perfil de utilizador e ainda associar ou desassociar uma OBU ao utilizador.

User Details

id: 2

User Name: tester1

User Profile: SUPER_USER

☐ **Suspended**

OBU Associations

OBU001	EDITOR	Delete
--------	--------	--------

OBUs: Choose OBU

Role: ☒ Viewer ☐ Editor

Add OBU

Creator: rita
 Creation Date: 2019-07-08 16:32:07
 Modifier: tester
 Modified Date: 2019-07-10 12:54:23

Go Back Save

Figura 11 – Página ‘User Details’

Nestes dois componentes todas as informações mostradas e funcionalidades tais como criar ou editar um utilizador foram efetuadas através de chamadas ao serviço *User*, com a exceção das funcionalidades acerca da associação Utilizador/OBU que foram efetuadas através do serviço *UserHasOBU*.

- ***Hardware, OBU, Configuration, TestPlan e Setup*** – Estes componentes são bastante semelhantes ao componente *User* já acima referido. Apresentam uma lista onde cada entrada representa um hardware, OBU, configuração, plano de teste ou setup dependendo da página escolhida com uma breve informação acerca do item. Tem também a lista de opções possíveis por cada item da lista sendo estas editar/verificar detalhes ou apagar e no final da página tem uma opção que redireciona o utilizador para uma outra página onde permite a inserção de dados para criação de um novo item.

Na figura 12 é possível ver como exemplo a página ‘All OBUs’ onde para este caso específico a breve informação acima referida é o nome, estado e configuração corrente da OBU.

OBU Name	OBU State	Current OBU Configuration	
OBU001	READY	No Configuration	Options
OBU002	READY	No Configuration	Options
OBU003	READY	No Configuration	Options
Create New OBU			

Figura 12 – Página ‘All OBUs’

Nestes componentes todas as informações mostradas e funcionalidades foram efetuadas através de chamadas ao serviço correspondente á página selecionada, no caso do exemplo da Figura 12 é o OBU *Service*.

- ***Hardware-Detail, OBU-Detail, Configuration-Detail, TestPlan-Detail e Setup-Detail*** - Estes componentes são bastante semelhantes ao componente *User-Detail* já acima referido. Apresentam todos os detalhes, possíveis de edição ou não, de um hardware, OBU, configuração, plano de testes ou setup consoante a página em questão. No caso da OBU-Details é possível também fazer as associações/desassociações entre OBU e configuração e entre OBU e planos de teste assim como no caso do TestPlan-Details onde é possível associar/desassociar setups ao plano de testes corrente.

Na Figura 13 é possível ver como exemplo a página ‘Test Plan Details’ onde se consegue observar com mais detalhe informações sobre o plano de testes com o nome TestPlan001. Observa-se também a tabela de associação entre o plano de testes e setup que neste caso apenas tem um setup associado.

Test Plan Details

id: 1
Creator: tester
Creation Date: 2019-07-08T16:32:07

TestPlan001
Test Plan Name: TestPlan001

Start Date: 12/14/2018, 6:55 PM
Stop Date: 12/16/2018, 8:00 PM

Period: PT24H

Test Plan Setups

Setup001	GSMR	Delete
----------	------	--------

Setups: Choose Setup ▼

Add Setup

Go Back Save

Figura 13 – Página ‘Test Plan Details’

Como as operações sobre as tabelas de associações são feitas através de serviços diferentes aos dos detalhes e, por conseguinte, seriam feitos dois pedidos distintos á API um para alterar a tabela de associação e um para alterar detalhes do componente, decidimos que qualquer alteração em tabelas associativas iria fazer esse pedido de forma imediata e independente á API guardando imediatamente as alterações efetuadas. Para alterações relacionadas com os detalhes do componente apenas quando o utilizador fizesse ‘Save’ é que as alterações iriam ser efetuadas, ou seja, um outro pedido á API, desta forma garantimos que a base de dados se mantém consistente.

- ***Hardware-Create, OBU-Create, Configuraion-Create, TestPlan-Create e Setup-Create*** – Estes componentes são responsáveis por tudo o que está relacionado com a criação de hardware, OBU, configuração, plano de teste e *setup*. É aqui que é pedido ao utilizador que introduza a informação relativa ao que vai criar.

Na figura 14 temos como exemplo a página ‘Setup Create’ onde permitimos que o utilizador insira como quer que o novo *setup* funcione. Antes do pedido ao serviço para ser criado um *setup*, é verificado se algum campo obrigatório, como é o caso do nome, se encontra vazio. No caso de se encontrar vazio é emitido um aviso ao utilizador e só depois de este se encontrar devidamente preenchido é que se prossegue com o pedido ao serviço.

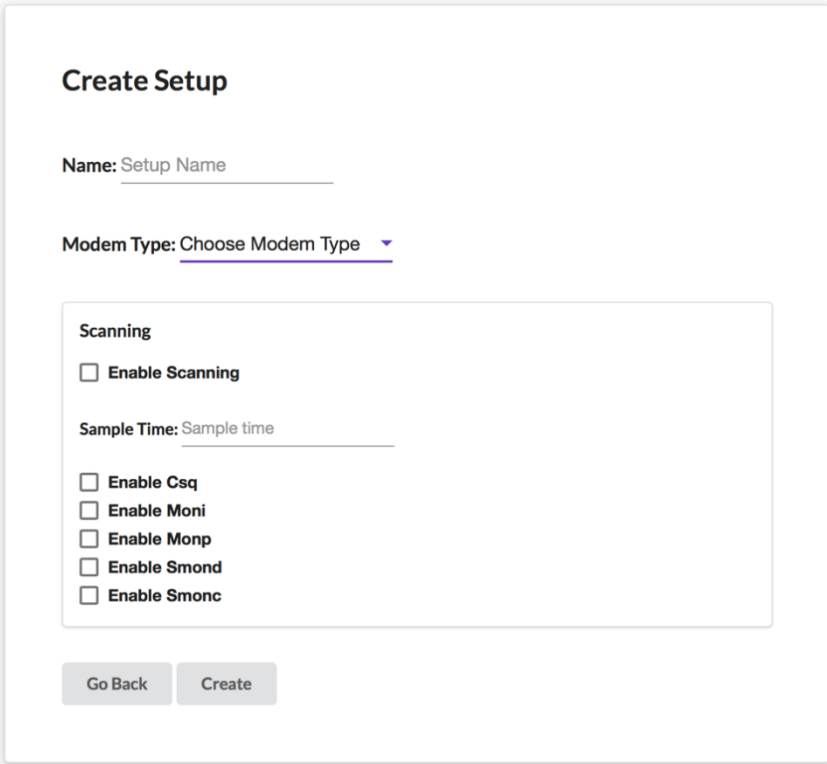
The image shows a web form titled "Create Setup". It contains a text input field for "Name" with the placeholder text "Setup Name". Below it is a dropdown menu for "Modem Type" with the placeholder text "Choose Modem Type". A section titled "Scanning" contains a checkbox for "Enable Scanning". Below this is a text input field for "Sample Time" with the placeholder text "Sample time". Another section contains five checkboxes: "Enable Csq", "Enable Moni", "Enable Monp", "Enable Smond", and "Enable Smonc". At the bottom of the form are two buttons: "Go Back" and "Create".

Figura 14 – Página ‘Setup Create’

- **ServerLog, TestLog e SystemLog** – Estes componentes são responsáveis pela listagem de *Logs*. Estes são listados por páginas de 20 elementos de cada vez quando os componentes carregam, sendo possível listar por 10, 20, 30, 40 ou 50 elementos por página. Para mudar de página basta carregar no botão *Next* ou *Previous Page* ou inserir o número de uma página específica e pressionar *enter* e será realizado um pedido à API para obter a página pretendida. Também é possível fazer filtragem sobre os *logs*. No caso do ServerLog é possível filtrar por tipo de acesso e por nome de utilizador ou OBU. No caso dos TestLog e SystemLog é possível filtrar por nome do ficheiro. E em ambos é possível organizar por data ascendente ou decendente, sendo decendente por omissão.

A figura 16 apresenta o *layout* da página de ServerLogs, sendo este *layout* bastante similar ao de TestLogs e SystemLogs.

Filters:						
Logs per Page: 20 ▾		Date order: Descending ▾	Access type: All ▾		User/OBU name: Search by User... Apply	
Request Date	Access Path	Access Type	Accessor Name	Response Date	Status	
2019-07-15T16:31:31.251781	GET : /api/v1/frontoffice/login/user	USER	tester	2019-07-15T16:31:31.291777	200 OK	
2019-07-15T16:31:29.00003	GET : /api/v1/frontoffice/obu/3/status	USER	tester	2019-07-15T16:31:29.171533	200 OK	
2019-07-15T16:31:28.99671	GET : /api/v1/frontoffice/obu/2/status	USER	tester	2019-07-15T16:31:29.458652	200 OK	
2019-07-15T16:31:28.991247	GET : /api/v1/frontoffice/obu/1/status	USER	tester	2019-07-15T16:31:29.25742	200 OK	
2019-07-15T16:31:28.725759	GET : /api/v1/frontoffice/obu	USER	tester	2019-07-15T16:31:28.859044	200 OK	
2019-07-15T16:31:28.662959	GET : /api/v1/frontoffice/login/user	USER	tester	2019-07-15T16:31:28.691139	200 OK	
2019-07-15T16:31:28.600856	GET : /api/v1/frontoffice/login/user	USER	tester	2019-07-15T16:31:28.631704	200 OK	
2019-07-15T16:31:28.297148	POST : /api/v1/frontoffice/login	USER	tester	2019-07-15T16:31:28.518484	200 OK	

Figura 15 – Página de ‘Serverlogs’

4. Conclusões

4.1. Trabalho desenvolvido

O objetivo do projeto aqui apresentado consistia no desenvolvimento de um sistema de informação que permita que tenha por funções:

- Realizar a administração de sondas, assegurando a gestão do equipamento, receção de alarmística e gestão de configurações, permitindo efetuar de forma permanente a monitorização das próprias OBU's;
- Análise e o tratamento de informação proveniente das OBU's de forma a permitir ao utilizador uma observação fácil e objetiva dos dados e administração do sistema

A abordagem ao problema apresentado anteriormente consistiu no desenvolvimento das seguintes tarefas:

1. Especificação dos requisitos funcionais e não funcionais e interfaces programáticas. Nesta fase foram, em conjunto com a entidade acolhedora do projeto, efetuados os levantamentos de todos os requisitos que poderiam influenciar o desenho e desenvolvimento do sistema. Para além dos requisitos funcionais, foram definidos todos os requisitos não funcionais, designadamente a arquitetura da solução e tecnologias utilizadas no desenvolvimento da mesma;
2. Análise da Base de Dados e da API. Nesta fase foram efetuadas a análise e propostas de alteração, em certas partes, de modo a melhorar estas duas componentes;
3. Desenvolvimento da arquitetura da solução. Nesta fase foi desenvolvida a base para a Web API inicialmente desenvolvida em linguagem *Kotlin* [22], que devido á disponibilização da API dos orientadores, foi descartada passando á linguagem *Java* [8];
4. Desenho da interface com o utilizador. Nesta fase foi efetuado um planeamento das páginas que iriam ser apresentadas ao utilizador por parte da aplicação cliente;
5. Implementação do modelo relacional e Web API. Nesta fase foi efetuada uma evolução do modelo relacional e Web API disponibilizados pelos orientadores. Foram também realizadas alterações aos mesmos e às restrições de integridade e lógica de negócio no decorrer da implementação;
6. Implementação do Website. Nesta fase foi desenvolvido a componente cliente, ou seja, todas as páginas de interação com o utilizador, serviços que servem estas páginas e restrições sobre as mesmas seguindo a lógica de negócio;

4.2. Conclusões Finais

Tendo em conta os objetivos apresentados, o trabalho desenvolvido procurou responder aos diversos desafios, tendo em conta a abrangência de tecnologias envolvidas no projeto. O resultado final foi devidamente testado e validado, o que envolveu os seguintes módulos:

- Aplicação Cliente Web, testada nos navegadores de internet Chrome e Safari;
- API e base de dados, testados nos sistemas operativos Windows e MacOS;

Assim, considera-se que foram atingidos globalmente os objetivos propostos, existindo, no entanto, alguns módulos que, devido a restrições temporais, não foram incluídos, designadamente:

- Pesquisa de dados por localização;
- Monitorização de OBUs em tempo real

4.3. Trabalho futuro

Numa futura continuação do projeto faria sentido terminar os módulos não concluídos e implementar funcionalidades como:

- Melhoria da interface gráfica;
- Melhoria no tratamento de erros (alertas ao utilizador sobre os errors);
- Melhoria entre os componentes da Aplicação Cliente Web, em termos de ter código mais genérico entre componentes mais similares, como as listas e as *views/edits* dos objetos (como por exemplo: Hardware, OBUs, mas maioritariamente os Logs)
- Fazer *deployment* do projeto;

5. Referências

- [1] “GSM-R,” [Online]. Available: <https://en.wikipedia.org/wiki/GSM-R>. [Acedido em 10 7 2019].
- [2] “PLMN,” [Online]. Available: https://en.wikipedia.org/wiki/Public_land_mobile_network. [Acedido em 10 7 2019].
- [3] “Leaflet,” [Online]. Available: <https://leafletjs.com/>. [Acedido em 23 05 2019].
- [4] “Chart.js,” [Online]. Available: <https://www.chartjs.org/>. [Acedido em 9 7 2019].
- [5] “TypeScript,” [Online]. Available: <https://www.typescriptlang.org/>. [Acedido em 11 7 2019].
- [6] “AngularJS,” [Online]. Available: <https://angularjs.org>. [Acedido em 23 05 2019].
- [7] “Visual Studio Code,” [Online]. Available: <https://code.visualstudio.com>. [Acedido em 23 05 2019].
- [8] “Java,” [Online]. Available: <https://www.java.com/en/>. [Acedido em 23 05 2019].
- [9] “Spring MVC,” [Online]. Available: <https://spring.io/guides/gs/serving-web-content/>. [Acedido em 23 05 2019].
- [10] “PostgreSQL,” [Online]. Available: <https://www.postgresql.org/>. [Acedido em 10 7 2019].
- [11] “URI,” [Online]. Available: <https://pt.wikipedia.org/wiki/URI>. [Acedido em 10 07 2019].
- [12] “Http Status Code,” [Online]. Available: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status>. [Acedido em 10 7 2019].
- [13] “RFC 7617,” [Online]. Available: <https://tools.ietf.org/html/rfc7617>. [Acedido em 11 7 2019].
- [14] “SSL/TLS,” [Online]. Available: <https://www.hostinger.com.br/tutoriais/o-que-e-ssl-tls-https/>. [Acedido em 10 7 2019].
- [15] “Content-Type,” [Online]. Available: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Headers/Content-Type>. [Acedido em 11 7 2019].
- [16] “Problem+Json,” [Online]. Available: <https://tools.ietf.org/html/rfc7807>. [Acedido em 11 7 2019].

- [17] “Angular - CanActivate,” [Online]. Available: <https://angular.io/api/router/CanActivate>.
[Acedido em 23 05 2019].
- [18] “Angular - HttpInterceptor,” [Online]. Available:
<https://angular.io/api/common/http/HttpInterceptor>. [Acedido em 23 05 2019].
- [19] “LocalStorage,” [Online]. Available:
https://www.w3schools.com/html/html5_webstorage.asp. [Acedido em 9 7 2019].
- [20] “html,” [Online]. Available: <https://www.w3schools.com/html/>. [Acedido em 9 7 2019].
- [21] “CSS,” [Online]. Available: <https://www.w3schools.com/css/>. [Acedido em 9 7 2019].
- [22] “Kotlin,” [Online]. Available: <https://kotlinlang.org/>. [Acedido em 11 7 2019].