

**Licenciatura em Engenharia Informática e Computadores**

**Projeto e Seminário - Semestre de Verão 2018/2019**

**Relatório de Final**

**Monitorização de qualidade de serviço em**

**comunicações ferroviárias**

João Vaz, Nº 41920 Luis Vasconcelos, Nº41556

41920@alunos.isel.pt 41556@alunos.isel.pt

961833140 910365046

**Orientado por:**

Nuno Cota, ncota@deetc.isel.pt

Ana Beire, anaritabeire@solvit.pt

**Índice**

[1. Introdução 1](#_Toc13577603)

[1.1. Enquadramento 1](#_Toc13577604)

[1.2. Objetivos 1](#_Toc13577605)

[1.3. Organização do Documento 1](#_Toc13577606)

[2. Descrição do sistema e Solução Proposta 3](#_Toc13577607)

[2.1. Arquitetura geral 3](#_Toc13577608)

[2.2. Configuração 4](#_Toc13577611)

[2.3. Plano de Testes 5](#_Toc13577612)

[2.4. Requisitos do sistema 5](#_Toc13577613)

[2.5. Solução Proposta 7](#_Toc13577614)

[2.5.1. Arquitetura do Sistema 7](#_Toc13577615)

[3. Implementação 9](#_Toc13577617)

[3.1. Base de Dados 9](#_Toc13577618)

[3.2. API 10](#_Toc13577620)

[3.2.1. Controladores 10](#_Toc13577621)

[3.2.2. Serviços 11](#_Toc13577622)

[3.2.3. Repositórios 12](#_Toc13577623)

[3.2.4. Interceptores 12](#_Toc13577624)

[3.3. Cliente 14](#_Toc13577625)

[3.3.1. Interceptores: 14](#_Toc13577626)

[3.3.2. Serviços 14](#_Toc13577627)

[4. Conclusão 19](#_Toc13577632)

[Referências 21](#_Toc13577634)

# Introdução

## Enquadramento

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. Nesse sentido, foram desenvolvidas unidades embarcadas em comboios (sondas) que realizam de forma autónoma e permanente a monitorização, ativa e passiva, aos sistemas de comunicações móveis ferroviários.

O conjunto de sondas embarcadas, designadas por OBU (*On Board Unit*), 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 como função igualmente servir de repositório de informação e permitir disponibilizar as interfaces necessárias à análise dos dados recolhidos.

## 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 será composto por repositório de dados e aplicação Web que permita a análise e o tratamento de informação proveniente das sondas de forma a permitir ao utilizador uma fácil e objetiva observação dos dados e administração do sistema.

## Organização do Documento

Este documento encontra-se dividido em seis 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 proposta. No quarto capítulo, descreve-se a implementação de cada componente do projeto. No quinto capítulo apresenta-se o planeamento de tarefas já realizadas e a realizar.

# 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.

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 em base de dados do operador, por 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.

## Arquitetura geral

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 e/ou PLMN [2].

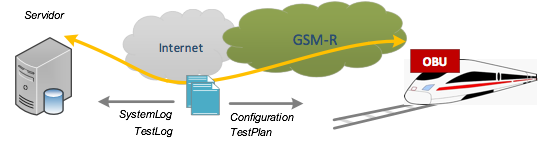


Figura 1 – Arquitetura geral do sistema.

A principal função 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 do disco local. Periodicamente, de acordo com a hora definida na configuração, serão enviados os resultados para o sistema central, para posterior registo em 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.

## 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.

## 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. Se não for despoletado por *geofence*, o plano de testes pode ter um padrão 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.

## Requisitos do sistema

Em termos de requisitos funcionais, foi definido um conjunto de aspetos funcionais a que o sistema deverá responder, os quais serão descritos de seguida.

Como requisitos obrigatórios temos:

* Aplicação Web Single-Page com integração de mapas;
* Gestão de OBUs (*On Board Unit*):

- 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 seja possível.

## 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 já atual- mente armazenada numa base de dados.

A informação resultante dos testes deverá ser apresentada de forma geo-referenciada, sobre mapas e através de gráficos. Para disponibilização do serviço de mapas e gráficos será utilizada a API Leaflet [3] e a API 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 deverá igualmente incluir toda a componente de gestão de utilizadores e perfis.

### 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 JavaScript com base na *framework* AngularJS [5] e a ferramenta de desenvolvimento Visual Studio Code [6]. 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 OBUs 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 [7]. A utilização da framework Spring MVC [8] permite simplificar e auxiliar o desenvolvimento da API.

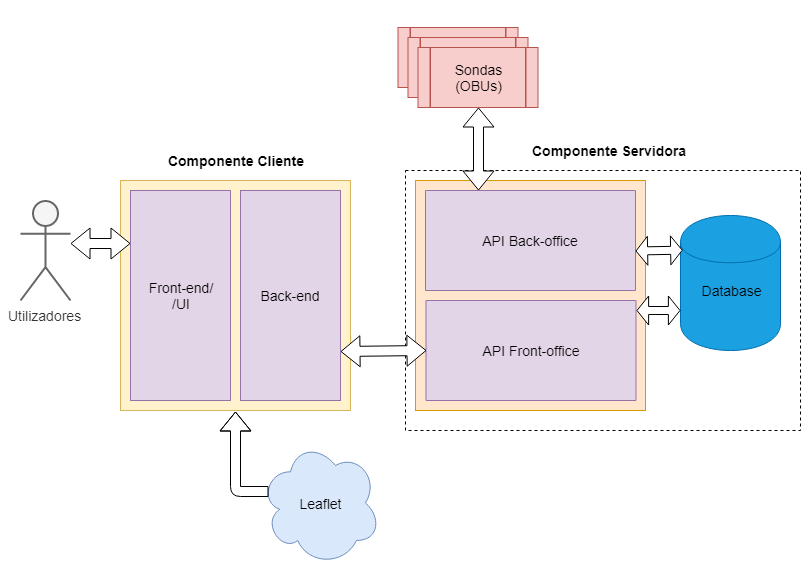


Figura 3 – Arquitetura da solução.

# Implementação

A implementação do sistema irá ser dividida nos componentes identificados anteriormente.

## Base de Dados

A implementação da Base de Dados foi feita em SQL PostGres [9]. O modelo de dados teve por base o sistema de informação existente, sendo que foi necessária a sua adaptação ao sistema a desenvolver, adicionando-se informação. A figura 4 apresenta o modelo de dados desenvolvido.

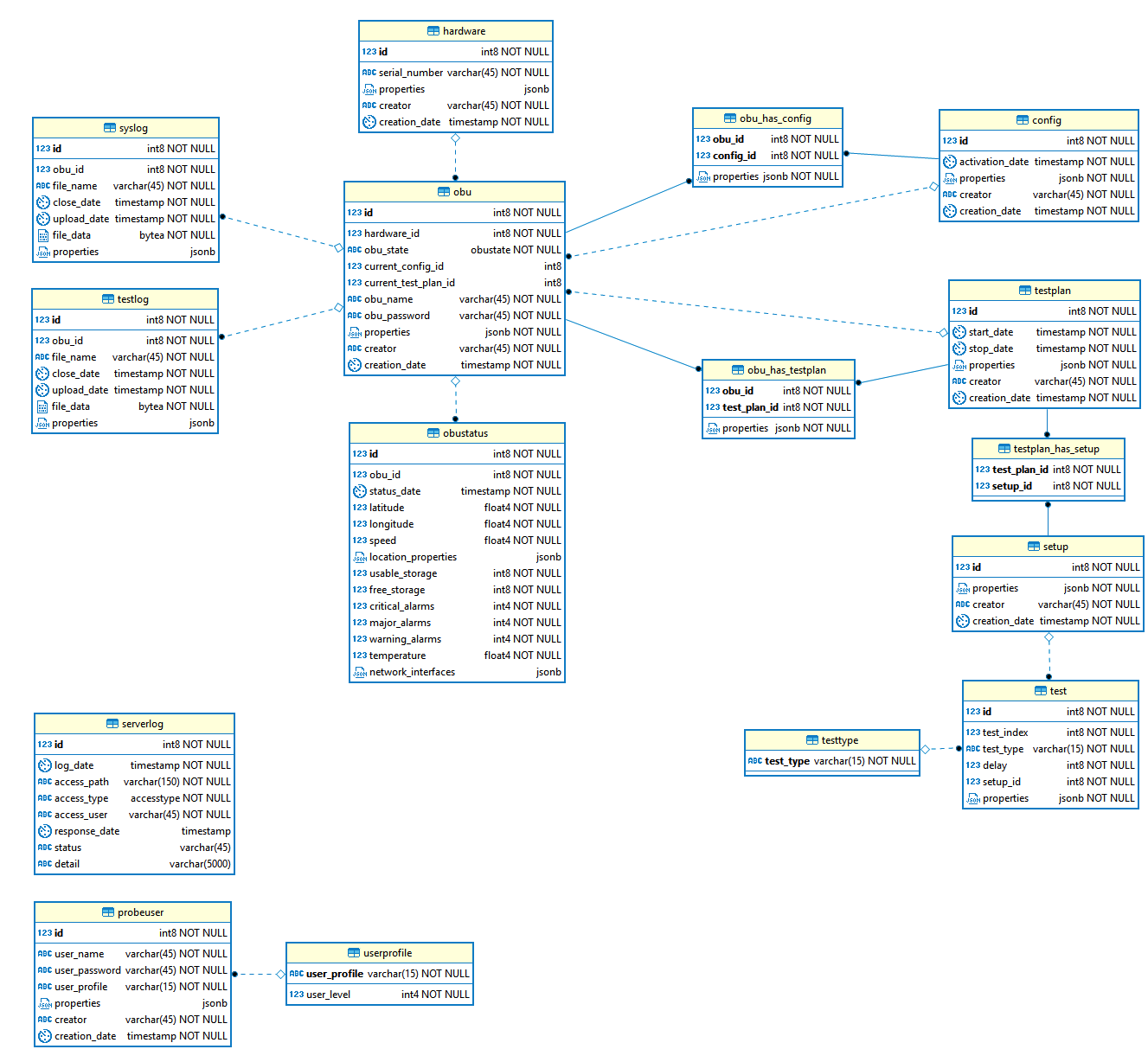


Figura 4 - Modelo de dados.

Uma das componentes adicionadas 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, que está associado á tabela UserProfile, que lhe permite o acesso ou não a informações mais especificas.

A tabela ProbeUser\_Obu representa a associação entre um utilizador e uma OBU.

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.

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 sob o qual a sonda irá estar funcional. Guarda informações referentes aos seus componentes tais como tipo de componente, serial, modelo e fabricante.

* Config

Esta tabela representa uma configuração. 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. 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.

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.

* ObuStatus

Esta tabela representa um conjunto de dados 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 dados sobre ficheiros com informações referentes a logs [10] reportados pela OBU.

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

## API

Para a implementação da API foi feita uma análise da informação que é necessária disponibilizar para o exterior, de acordo com as funcionalidades que se irá implementar e, tendo em atenção que o Web Cliente pode não ser o único consumir essa informação.

A API vai dispor, de início, 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.

Como tipo de autenticação foi escolhido *Authorization Basic*.Para aceder aos recursos da API é necessário passar o *token* neste cabeçalho *Authorization*. Para obter este *token* é necessário fazer *login*. Como tipo do cabeçalho *Authorization* que a API está pronta a receber é *Authorization Basic*, logo o *token* não é encriptado, mas sim codificado em Base64 [7].

Os controladores mapeiam um *request* numa resposta. Quando é feito um *request* à API os controladores apanham esse *request*. Estes fazem pedidos aos serviços que por sua vez fazem pedidos aos repositórios que fazem pedidos 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 ao *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.

### Controladores

Os controladores são os únicos 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 buscar, fazer update e, novamente, para buscar 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 buscar, fazer a suspensão e, novamente, para buscar o utilizador já suspendido. Este utilizador suspendido é adicionado ao *body* da *response*.

### Serviços

Os serviços são responsáveis por verificações, por exemplo, se o utilizador que fez o *request* tem permissão/perfil alto o suficiente para aceder ao recurso, e por transformar os objetos 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 buscar 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 buscar 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*.

### Repositórios

Os repositórios contêm as *query strings* de SQL e são responsáveis por fazer a injeção de variáveis passadas pelos serviços nestas mesmas *strings*. Também cabe aos repositórios fazer a conexão e pedidos à base de dados.

### Interceptores

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 intercepta). 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.

* **AuthInterceptor** - este intercetor é responsável pela verificação do *token* e do utilizador associado, em qualquer *request* feito (excepto 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 não está 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.

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

De seguida apresenta-se uma tabela com os endereços específicos da API (*endpoints)* a disponibilizar:

|  |  |  |
| --- | --- | --- |
| Método | Pedido HTTP | Descrição |
| GET | users | Retorna todos os utilizadores existentes. |
| GET | user/{param} | Retorna o utilizador referente ao param, podendo este ser id ou nome de utilizador. |
| PUT | user/{id}/suspend | Suspende o utilizador referente ao id. |
| POST | users | Cria um novo utilizador. |
| POST | login | Retorna o token para efectuar o login. |
| GET | obu | Retorna todas as OBUs existentes. |
| GET | obu/{id} | Retorna a OBU referente ao id. |
| PUT | obu/{id} | Retorna a OBU referente ao id com os campos atualizados. |
| POST | obu | Cria uma nova OBU. |
| GET | hardware | Retorna todos os hardwares. |
| GET | hardware/{id} | Retorna o hardware referente ao id. |
| PUT | hardware/{id} | Retorna o hardware referente ao id com os campos atualizados. |
| POST | hardware | Cria um novo hardware. |
| GET | config | Retorna todas as configurações existentes. |
| GET | config/{id} | Retorna a configuração referente ao id. |
| GET | test-plan | Retorna todos os planos de teste. |

## Cliente

Este componente contém a interface gráfica apresentada ao utilizador. O *Web Client* ainda vai sofrer algumas modificações em termos de UI/*frontend* para disponibilizar uma melhor experiência ao utilizador.

Os dados apresentados na interface são obtidos através da comunicação com a componente servidora, mais propriamente a API.

### Interceptores:

* **AuthGuard** - Na implementação da componente cliente foi implementado um AuthGuard que implementa a interface CanActivate [8]. 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 suspendido. 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 [9]. Assim esta classe intercepta 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* específico ao utilizador, mas também é usado para error handling de certos erros (ainda em desenvolvimento!!!), tal como o de o utilizador estar suspenso.

### 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[XXX] 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 final e a única maneira de o utilizador ser logged out por estar suspenso é se este fizer 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 getPositionFromOBU que faz um pedido relativo á informação das posições 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/Configuração, OBU/Plano de Teste e Plano de Teste/Setups, respectivamente. São somente pedidos *get*, *create* e *delete*, desta forma 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*.

### 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*[XXX] e um .*css*[XXX] que são responsáveis pela apresentação e um ficheiro .spec.ts e um .ts que são responsáveis pelas ações que podem ser efetuadas na página.

* ***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 ou com sessão expirada resulta no redireccionamento a esta página. No caso da tentativa de login com credenciais erradas ou campos em falta é emitida uma mensagem de alerta (Figura 5) avisando assim o utilizador.

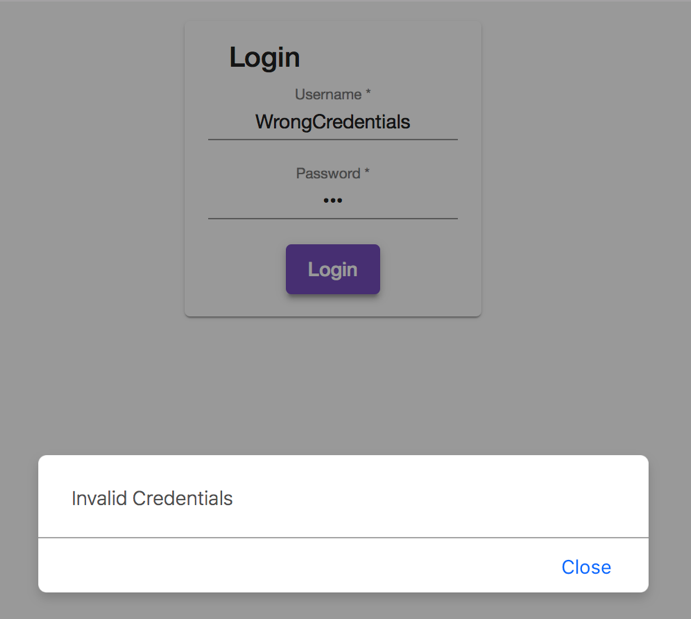
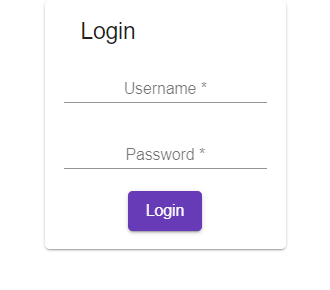


Figura 5 – Login na página inicial da aplicação e tentativa de login com mensagem de alerta

* ***HomeMap e NavMenu*** – 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.

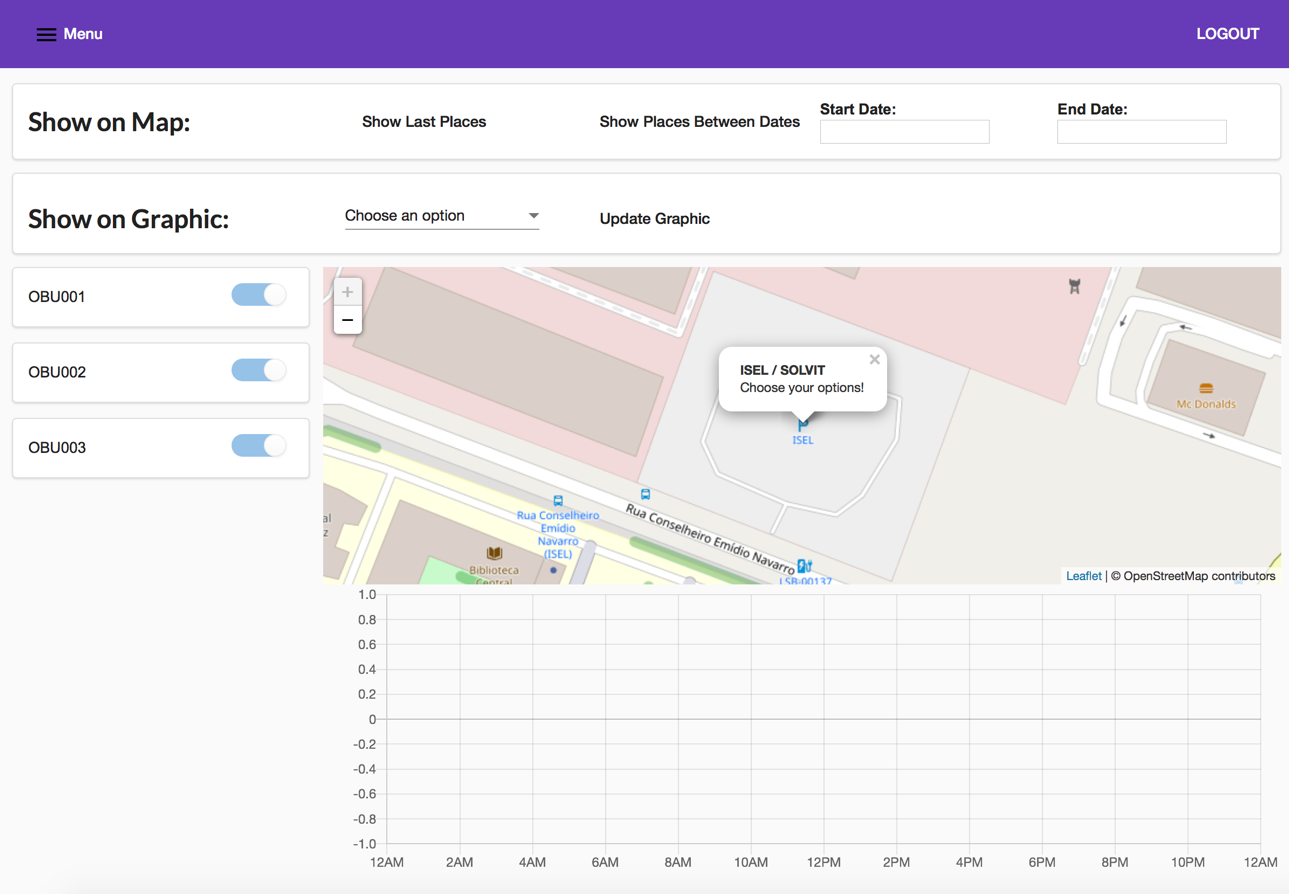


Figura 6 - Página principal da aplicação.

No mapa é possível verificar o ultimo sitio onde cada OBU esteve (botão - ‘Show Last Places’) assim como verificar onde esteve cada OBU num determinado espaço de tempo indicando uma data de inicio 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 getPositionFromOBU do serviço OBUService, referido anteriormente, tendo como parâmetros as respetivas datas permitidas assim que a filtragem seja feita. 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 verificar 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 ás 09:54 horas até ás 11:54 horas desse mesmo dia. Neste caso especifico existem três OBUs associadas ao utilizador corrente e apenas uma é visível no mapa, isto porque não existem dados relativos ás restantes no período de tempo especificado.

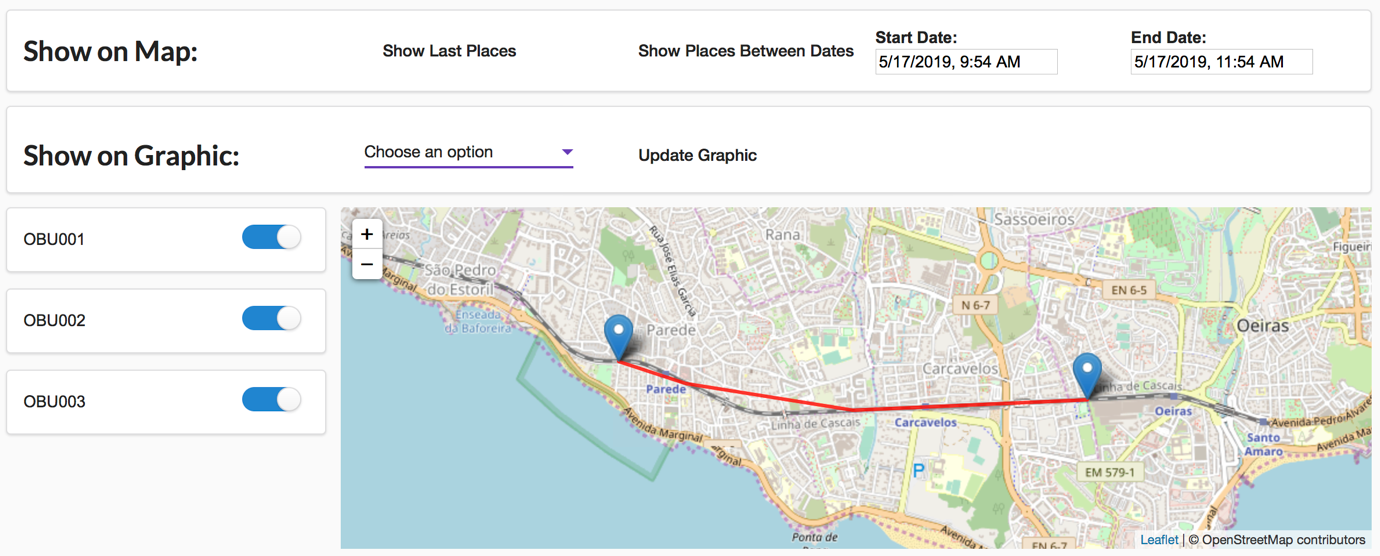


Figura 7 – Exemplo de visualização da rota da OBU num determinado período de tempo no mapa

No gráfico 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.

A figura 8 demonstra um exemplo onde se apresenta a informação relativa á velocidade ao longo de vários dias. Constata-se facilmente que a velocidade mais alta foi atingida pela OBU002 e era cerca de 90km/hora. Também é possível verificar que a velocidade dessa mesma OBU no dia 14 de junho de 2019 pelas 09:14 horas era de 51km/hora.

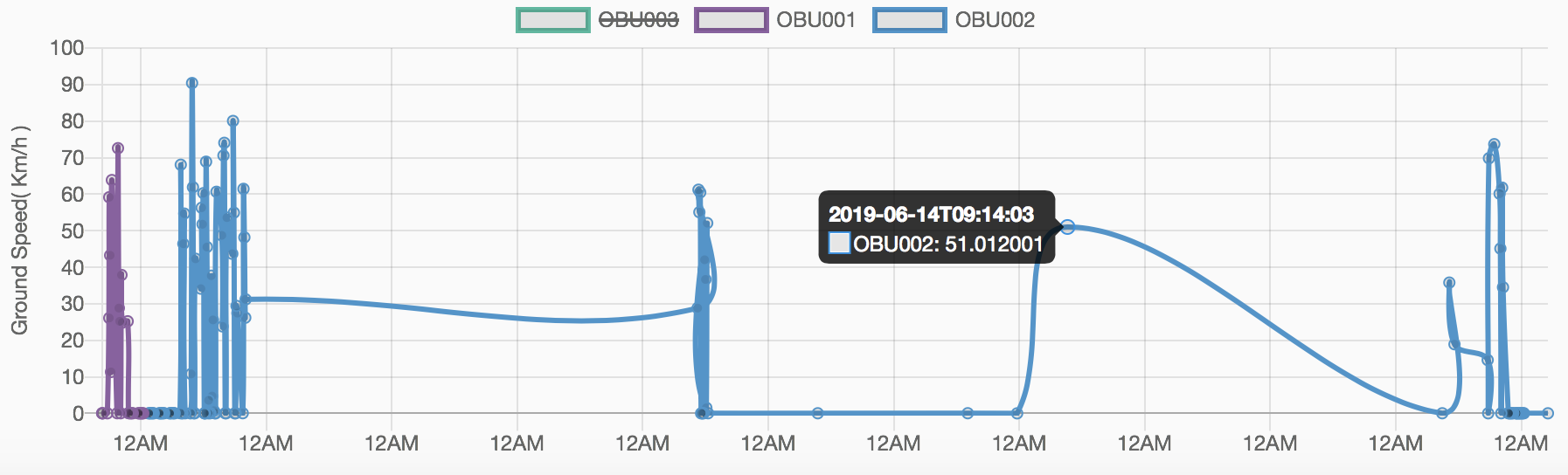


Figura 8 – Exemplo de visualização da velocidade das OBUs no gráfico

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.

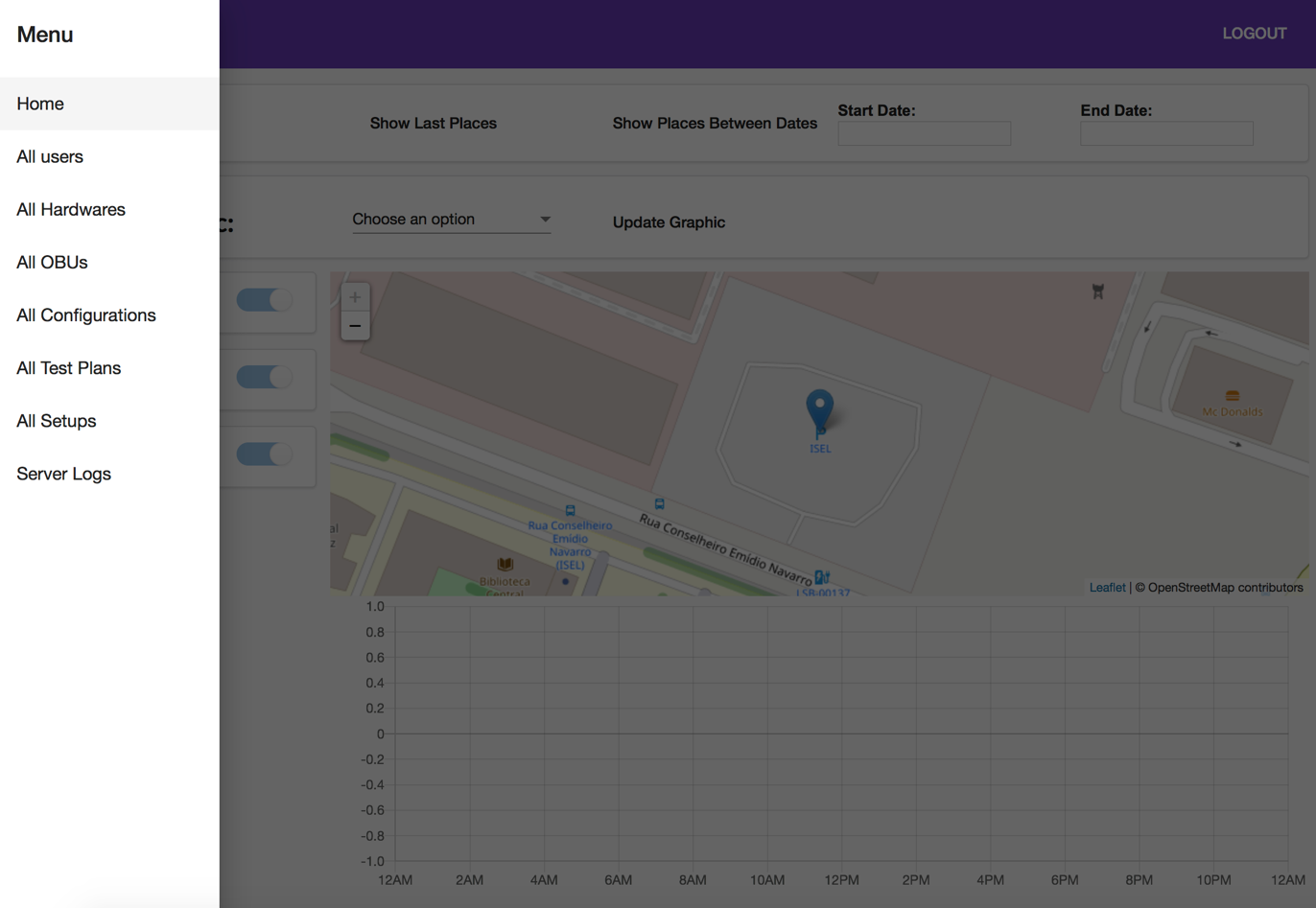


Figura 8 – Menu lateral da página inicial

O NavMenu é o componente responsável pela barra superior da página inicial, é possível ser visto na Figura 6. Permite ao utilizador ter acesso ao *Menu* que o pode redirecionar para outras páginas da aplicação ou fazer *logout*.

* ***User e User-Detail***
* ***Hardware, OBU, Configuration, TestPlan e Setup***
* ***Hardware-Detail, OBU-Detail, Configuration-Detail, TestPlan-Detail e Setup-Detail***
* ***Hardware-Create, OBU-Create, Configuraion-Create, TestPlan-Create e Setup-Create***
* ***ServerLog***

De seguida é redirecionado para a *Home Page* (Figura 6), onde, através do menu lateral (Figura 7), pode escolher uma das várias opções fornecidas com a finalidade de aceder a informações sobre utilizadores, OBUs, etc.

Nas páginas de informações é apresentada uma lista com os detalhes mais importantes sobre, por exemplo, os utilizadores; detalhes como o nome e o seu nível (se é *Admin*, *SuperUser* ou *NormalUser*) (Figura 8).

Certas páginas como as páginas das listas de utilizadores e obus não podem ser acedidas por utilizadores cujo nível seja *NormalUser*, embora outras como a *Home Page* possa ser acedida por qualquer utilizador.

Futuramente os utilizadores suspendidos não poderão aceder a qualquer página e serão redirecionados para uma página que dirá que estão suspensos, ou simplesmente no ato de início de sessão um *popup* indicará ao utilizador que está suspenso e que não pode iniciar sessão.

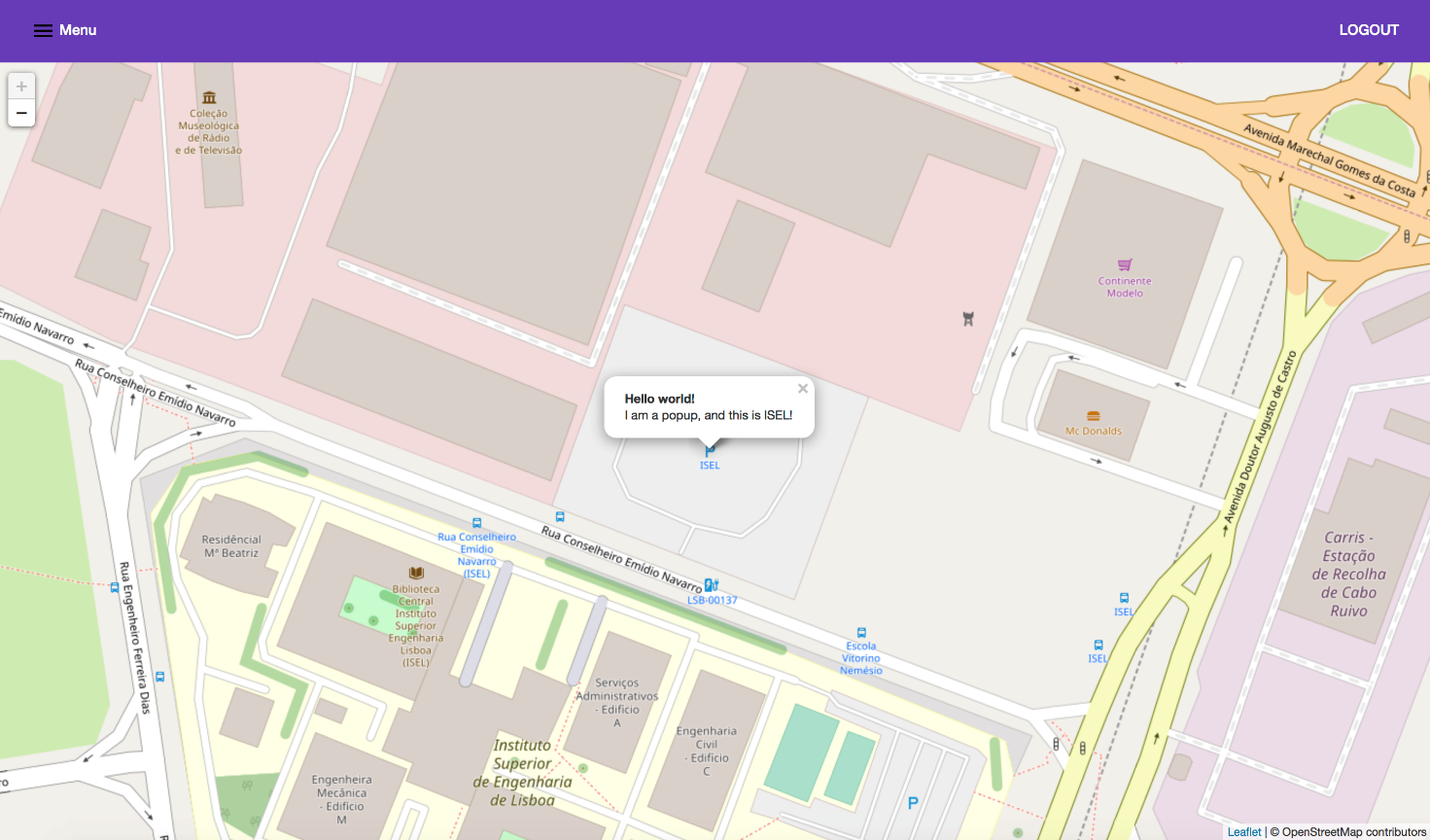


Figura 6 -Página principal da aplicação.

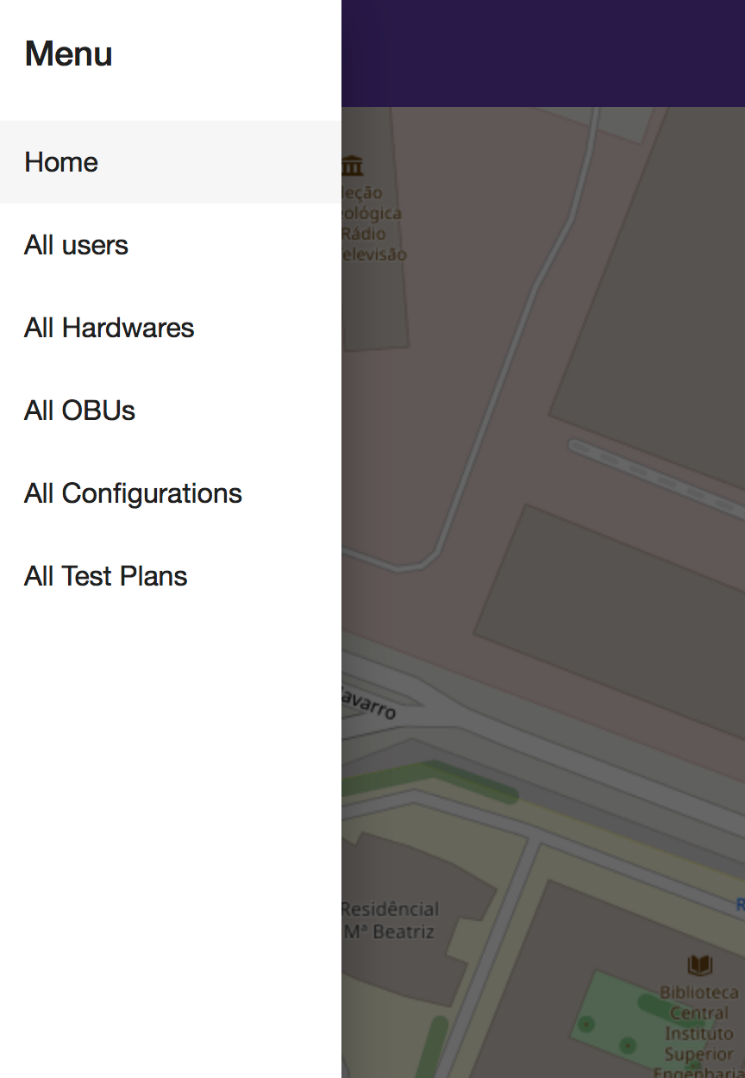


Figura 7 - Menu lateral.

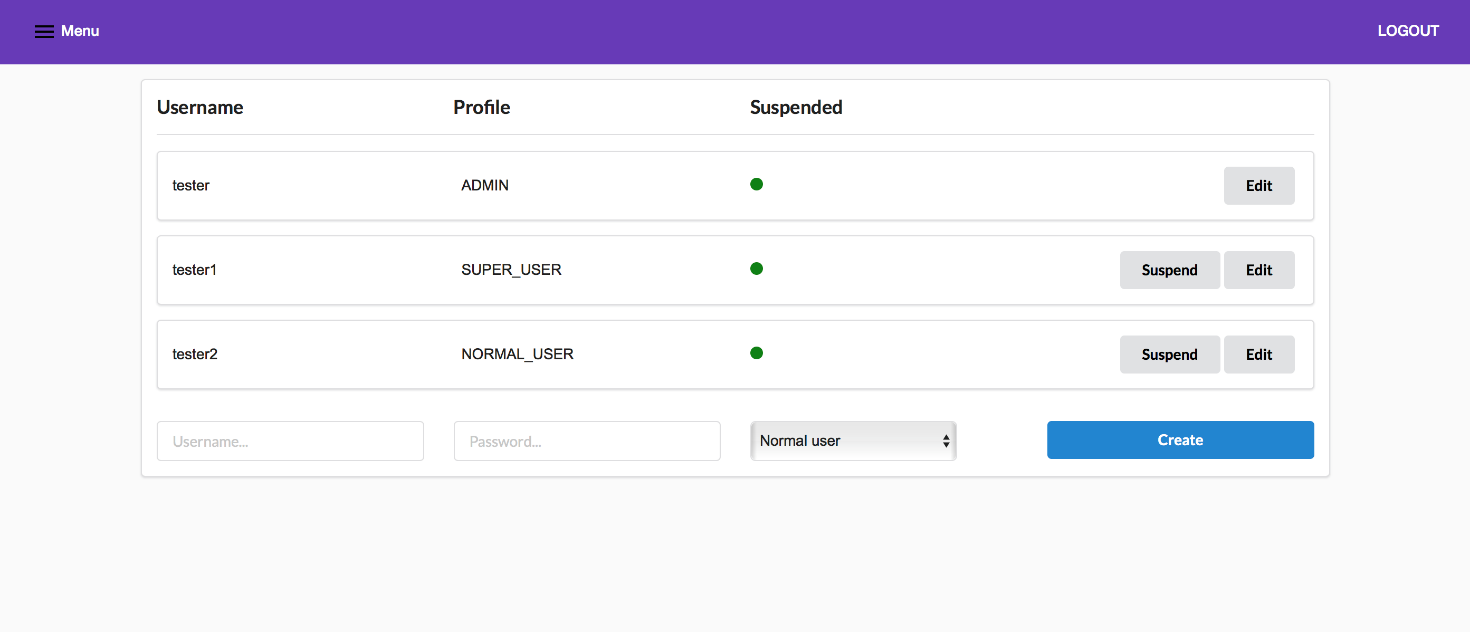


Figura 8 – Página de lista de utilizadores.

# Conclusão

Apresenta-se na Figura 9 o plano que está a ser seguido na realização deste projeto. O plano contém as tarefas a realizar bem como os respetivos períodos de tempo. Até ao momento está a ser cumprido o plano que foi proposto.

Encontram-se concluídos o desenvolvimento da arquitetura da solução, o desenho da interface com utilizador da Web API, implementação do modelo relacional e a apresentação de alguns dos dados na aplicação Cliente.

Foram já testados, com sucesso, vários sub-sistemas da aplicação, nomeadamente:

* Login de utilizadores, testado nos navegadores de internet Chrome e Safari, assim como nos sistemas operativos Windows e MacOS;
* Pedido à API e devolução de resultado em formato JSON;
* Listagem e edição de utilizadores, OBUs;
* *Display* de pontos, e conexões entre os mesmos, no *homemap*;
* *Logging* persistente na base de dados de todos os pedidos e respostas feitos à API;

Os próximos passos passam pela continuação da implementação do Website - gestão de utilizadores e apresentação de dados assim como a continuação da implementação da Web API.

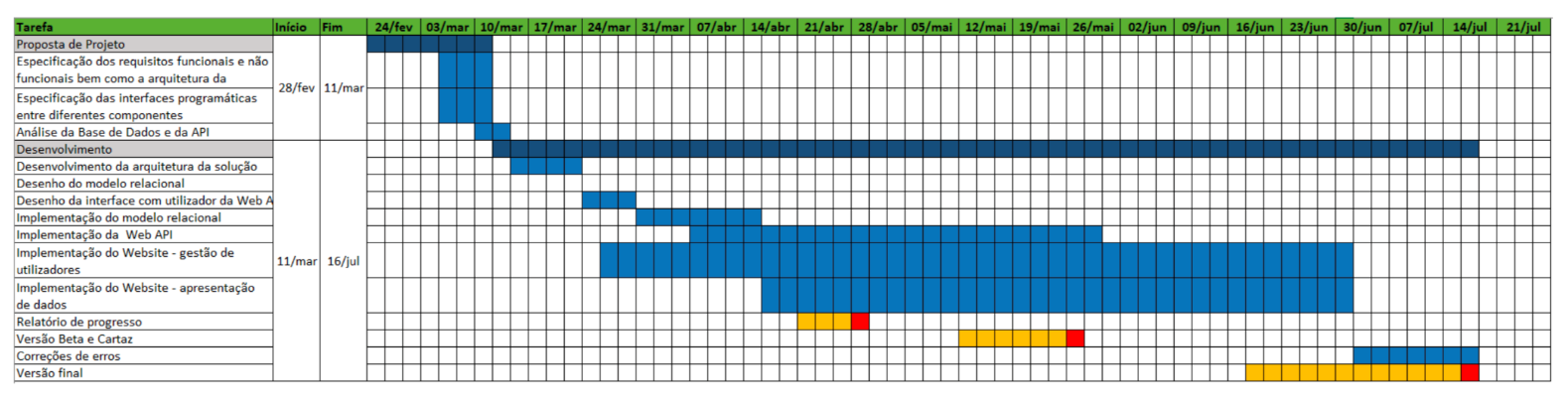


Figura 9 - Planificação temporal das atividade e tarefas associadas ao projeto.

# Referências

[1] “GSM-R,” [Online]. Available: https://pt.wikipedia.org/wiki/GSM-R [Acedido em 09 07 2019].

[2] “PLMN,” [Online]. Available: https://en.wikipedia.org/wiki/Public\_land\_mobile\_network [Acedido em 09 07 2019].

[3] “Leaflet,” [Online]. Available: https://www.chartjs.org/. [Acedido em 23 05 2019].

[4] “Chart.js,” [Online]. Available: https://leafletjs.com/. [Acedido em 23 05 2019].

[5] “AngularJS,” [Online]. Available: https://angularjs.org. [Acedido em 23 05 2019].

[6] “Visual Studio Code,” [Online]. Available: https://code.visualstudio.com. [Acedido em 23 05 2019].

[7] “Java,” [Online]. Available: https://www.java.com/en/. [Acedido em 23 05 2019].

[8] “Spring MVC,” [Online]. Available: https://spring.io/guides/gs/serving-web-content/. [Acedido em 23 05 2019].

[9] “PostGresql,” [Online]. Available: https://www.postgresql.org/. [Acedido em 09 07 2019].

[10] “Log de Dados,” [Online]. Available: https://pt.wikipedia.org/wiki/Log\_de\_dados

/. [Acedido em 09 07 2019].

[11] “Códigos de Estado,” [Online]. Available: https://pt.wikipedia.org/wiki/Lista\_de\_códigos\_de\_estado\_HTTP#Lista\_de\_códigos\_de\_status\_HTTP/. [Acedido em 09 07 2019].

[7] “Base 64 wiki,” [Online]. Available: https://pt.wikipedia.org/wiki/Base64. [Acedido em 23 05 2019].

[8] “Angular - CanActivate,” [Online]. Available: https://angular.io/api/router/CanActivate. [Acedido em 23 05 2019].

[9] “Angular - HttpInterceptor,” [Online]. Available: https://angular.io/api/common/http/HttpInterceptor. [Acedido em 23 05 2019].

//Cliente – verificar números no final

[XXX] “LocalStorage,” [Online]. Available: https://developer.mozilla.org/pt-BR/docs/Web/API/Window/Window.localStorage/. [Acedido em 09 07 2019].

[XXX] “Html,” [Online]. Available: https://pt.wikipedia.org/wiki/HTML/. [Acedido em 09 07 2019].

[XXX] “Css,” [Online]. Available: https://pt.wikipedia.org/wiki/Cascading\_Style\_Sheets/. [Acedido em 09 07 2019].