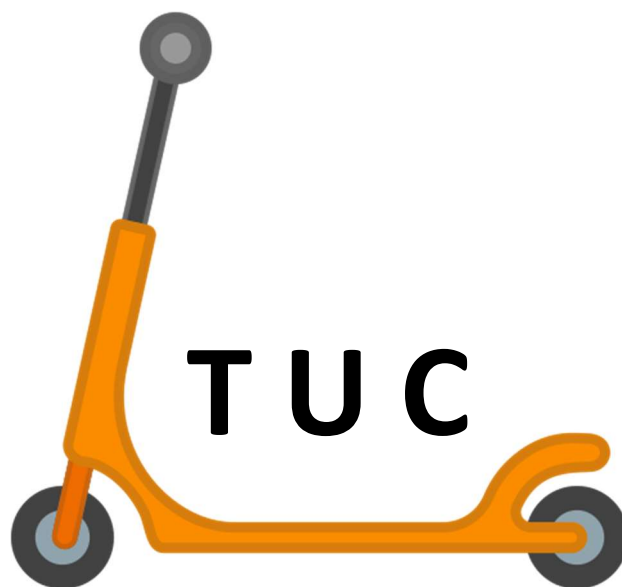




universidade
de aveiro



Trocinetes Universitárias Conectadas

49984 Projeto de Engenharia Informática

Metrado Integrado em Engenharia de Computadores e Telemática

2019-2020

Orientador

José Vieira

Grupo 3

Jacinto Lufilakio 89162

Joaquim Ramos 88812

Márcia Pires 88747

Rita Amante 89264

Tomás Martins 89286

Relatório de Projeto de Engenharia Informática do Mestrado Integrado em Engenharia de Computadores e Telemática da Universidade de Aveiro, realizado por Jacinto Lufilakio, Joaquim Ramos, Márcia Pires, Rita Amante e Tomás Martins sob a orientação de José Vieira, Pró-reitor para os Sistemas de Informação da Universidade de Aveiro.

Resumo

O projeto TUC é um sistema de partilha de trotinetes capaz de otimizar a deslocação dentro do Campus da Universidade de Aveiro, permitindo uma redução de $\frac{1}{4}$ do tempo nas deslocações, comparativamente a um percurso realizado a pé.

Este sistema permite ao utilizador, previamente autorizado e inserido na base de dados de utilizadores com acesso, a visualização das trotinetes e informação sobre a sua disponibilidade e bateria das mesmas, através do portal do utilizador. Após o utilizador localizar uma trotinete disponível, irá ao encontro da mesma e para usufruir do sistema, apenas tem de aproximar o cartão da Universidade de Aveiro ao leitor RFID, montado na trotinete. Depois do utilizador desbloquear a trotinete, este realiza o seu percurso e pelo caminho a trotinete envia para o servidor a sua localização e, caso o utilizador passe os limites geográficos definidos pelo gestor, a trotinete dispara o alarme para avisar o utilizador e bloqueia a roda motriz. Quando o utilizador regressar do seu destino, volta a aproximar o seu cartão ao leitor, para bloquear a trotinete.

O sistema foi implementado com sucesso, cumprindo com todos os requisitos pedidos.

Ao cliente será entregue, como produto final: todo o software desenvolvido, este documento que contém todas as instruções de instalação e o hardware que foi adquirido e montado.

Índice

INTRODUÇÃO	7
TUC – TROTINETES UNIVERSITÁRIAS CONECTADAS.....	8
1 Oportunidade de negócio.....	8
2 Cenários de utilização do sistema.....	9
3 Requisitos do sistema.....	10
3.1 Metodologia de recolha de requisitos.....	10
3.2 Requisitos funcionais.....	10
3.3 Requisitos não funcionais	10
SISTEMA TUC.....	11
1 Hardware	12
1.1 Escolha da trotinete.....	12
1.2 Componentes.....	12
1.3 Instalação dos componentes.....	12
2 Software.....	16
2.1 Scooter Client.....	16
2.2 API para a interação com a página	18
2.3 MQTT Server	19
2.4 Web Server	20
3 Interação com o utilizador	22
3.1 Desbloqueio da trotinete	22
3.2 Geofencing.....	23
3.3 Portal de Administração TUC	23
3.4 Portal do utilizador	29
RESULTADOS	30
CONCLUSÕES.....	31
1 Trabalho futuro.....	31
REFERÊNCIAS.....	32

Índice de Tabelas

Tabela 1: Tabela comparativa de trotinetes.	12
Tabela 2: Lista de componentes.....	30

Índice de Figuras

Figura 1: Visão geral do sistema.....	9
Figura 2: Arquitetura do sistema.	11
Figura 3: Diagrama de pinos do Raspberry Pi zero w.....	13
Figura 4 Exemplo de código de emparelhamento com a trotinete.	15
Figura 5: Sistema de desbloqueio da trotinete.....	22
Figura 6: Início de sessão do portal de administração.....	24
Figura 7: Funcionalidades do portal de administração TUC.	24
Figura 8: Lista de trotinetes.	25
Figura 9: Histórico de utilização da trotinete.....	25
Figura 10: Lista de utilizadores.....	26
Figura 11: Mapa de localização das trotinetes.....	26
Figura 12: Localização das trotinetes.	27
Figura 13: Geofencing.	27
Figura 14: Informação e localização das trotinetes.	29

INTRODUÇÃO

A Universidade de Aveiro possui no perímetro urbano da cidade de Aveiro dois campi designados por campus de Santiago e campus do Crasto. Um funcionário da Universidade de Aveiro que pretenda deslocar-se entre os pontos mais afastados do campus para a realização de uma pequena tarefa precisa de uma deslocação rápida e de um meio para o fazer. A deslocação pedonal entre esses dois pontos demora cerca de 24 minutos, o que implica que o funcionário gaste desnecessariamente aproximadamente 50 minutos para a realização da mesma. Se o funcionário recorresse a uma trotinete elétrica, a mesma deslocação levaria aproximadamente 15 minutos, uma vez que o uso da trotinete permite a redução do tempo de deslocação para cerca de $\frac{1}{4}$ do tempo.

Deste modo, surge o projeto TUC – Trotinetes Universitárias Conectadas – que permite a partilha de trotinetes dentro do Campus da UA.

O presente documento pretende dar a conhecer o projeto TUC. No seu decorrer serão abordados aspetos importantes, assim como: oportunidade do negócio, cenários de utilização, requisitos, hardware e software do sistema e interação do sistema com o utilizador.

TUC – TROTINETES UNIVERSITÁRIAS CONECTADAS

1 Oportunidade de negócio

Um funcionário da Universidade de Aveiro que pretenda deslocar-se entre os pontos mais afastados do campus para a realização de uma pequena tarefa precisa de uma deslocação rápida e de um meio para o fazer. Para uma deslocação rápida, surgiu a ideia do uso de uma trotinete que reduziria $\frac{1}{4}$ do tempo caso o funcionário efetua-se o percurso a pé.

Neste contexto, surgiu o projeto TUC que tem como objetivo criar um sistema de partilha de trotinetes capaz de otimizar a deslocação dentro do Campus da Universidade de Aveiro, através de trotinetes conectadas à Internet, com sistema de autenticação através do cartão da UA.

Enquanto que as soluções de partilha de trotinetes existentes no mercado estão mais vocacionadas para frotas com um grande número de trotinetes, no mínimo na ordem de centenas de unidades, tinham a necessidade de construção de docks e não apresentam soluções que integrassem o cartão da UA. Estas soluções costumam utilizar uma aplicação móvel para o desbloqueio das trotinetes ou o uso de um novo cartão. O sistema TUC oferece um sistema de partilha moderno e restrito ao campus e aos funcionários da Universidade de Aveiro através do cartão da UA e geofencing.

Sendo assim, o projeto TUC tem como finalidade facilitar a deslocação de um funcionário da UA, fornecendo a localização e informação das trotinetes, o que permite ao utilizador um acesso rápido ao local de destino.

A parte interessada envolvida ativamente no sistema é a Universidade de Aveiro, que deve ter a responsabilidade de permitir o fácil acesso às diversas infraestruturas no campus, o que possibilita a rápida resposta a um problema dentro do mesmo.

O cliente alvo deste sistema são os funcionários da Universidade de Aveiro, como por exemplo, os funcionários dos sTIC. O facto do utilizador não ter de se deslocar a pé de um ponto do campus para outro, faz com que poupe tempo do seu trabalho e acima de tudo reduza o tempo gasto na execução das tarefas.

Este sistema permite ao gestor recolher dados sobre as trotinetes, como a localização, o estado da bateria e o histórico de utilização da mesma. Permite ainda delimitar a área de utilização evitando assim usos indevidos.

Da parte do utilizador, este sistema permite a visualização da localização e do estado das trotinetes e a requisição das mesmas.

2 Cenários de utilização do sistema

Este sistema tem como atores os membros da Universidade de Aveiro que podem ser funcionários e/ou alunos autorizados da Universidade de Aveiro com um cartão da UA válido e um gestor que gere toda a informação do sistema.

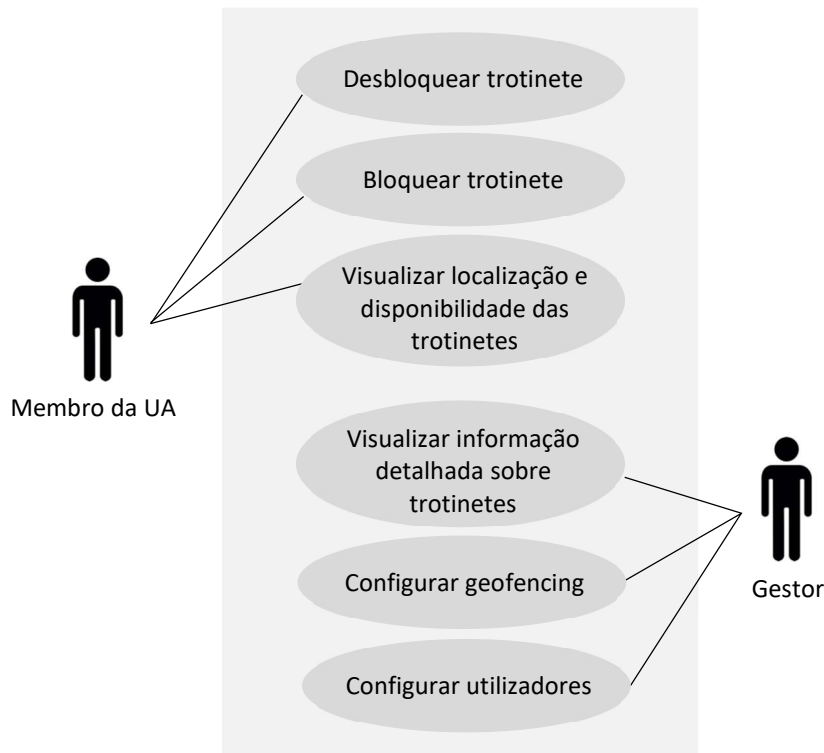


Figura 1: Visão geral do sistema.

Ao utilizador do sistema é permitido:

- **Visualizar a localização e disponibilidade das trotinetes:** o utilizador pode visualizar o mapa com a localização das trotinetes. E, se passar o cursor por cima de uma trotinete, pode visualizar também a disponibilidade e a bateria da mesma;
- **Desbloquear a trotinete:** o utilizador precisa de aproximar o cartão da UA ao leitor RFID montado na trotinete para se autenticar e desbloqueá-la;
- **Bloquear a trotinete:** após a utilização da trotinete, é necessário que o utilizador volte a passar o cartão no leitor RFID, o que por sua vez, irá bloqueá-la.

Ao gestor do sistema é permitido:

- **Visualizar informações sobre as trotinetes:** o gestor pode visualizar as informações das trotinetes, como por exemplo a disponibilidade, bateria e histórico de utilização e localização;
- **Controlar o acesso dos utilizadores:** o gestor pode configurar utilizadores, ou seja, adicionar e eliminar utilizadores, ou seja, permitir acessos ao sistema;
- **Delimitar a área de circulação das trotinetes:** o gestor pode configurar a área de uso do sistema.

3 Requisitos do sistema

3.1 Metodologia de recolha de requisitos

A fim de fazer o levantamento de requisitos, a equipa foi unânime na escolha das estratégias a utilizar: questionários. Deste modo, foram feitos questionários a diferentes docentes, para recolher informações sobre os limites das tecnologias em hipótese para a utilização no projeto.

3.2 Requisitos funcionais

No sistema TUC verificam-se alguns requisitos funcionais indispensáveis ao seu funcionamento, a saber:

- O utilizador deve ter na sua posse o cartão da UA válido para a utilização da trotinete;
- As trotinetes devem estar carregadas;
- As trotinetes devem estar dentro do alcance da rede móvel;
- O utilizador para aceder à página web deve ter acesso à internet.

3.3 Requisitos não funcionais

Relativamente aos requisitos não funcionais, o sistema TUC apresenta:

- Um LED que indica o estado de utilização da trotinete;
- A informação das trotinetes deve ser atualizada em tempo real;
- É necessário garantir que a comunicação entre o cartão e o servidor é feito em menos de 10 segundos.

SISTEMA TUC

Tendo em consideração as funcionalidades do sistema, é necessário haver estruturas para recolher, transmitir, armazenar e gerir a informação necessária ao funcionamento do mesmo. Deste modo, existem oito elementos fundamentais para o correto funcionamento do sistema: trotinete, Raspberry Pi, GPS, Bluetooth, Internet, RFID, servidor e a página web (hospedada no servidor).

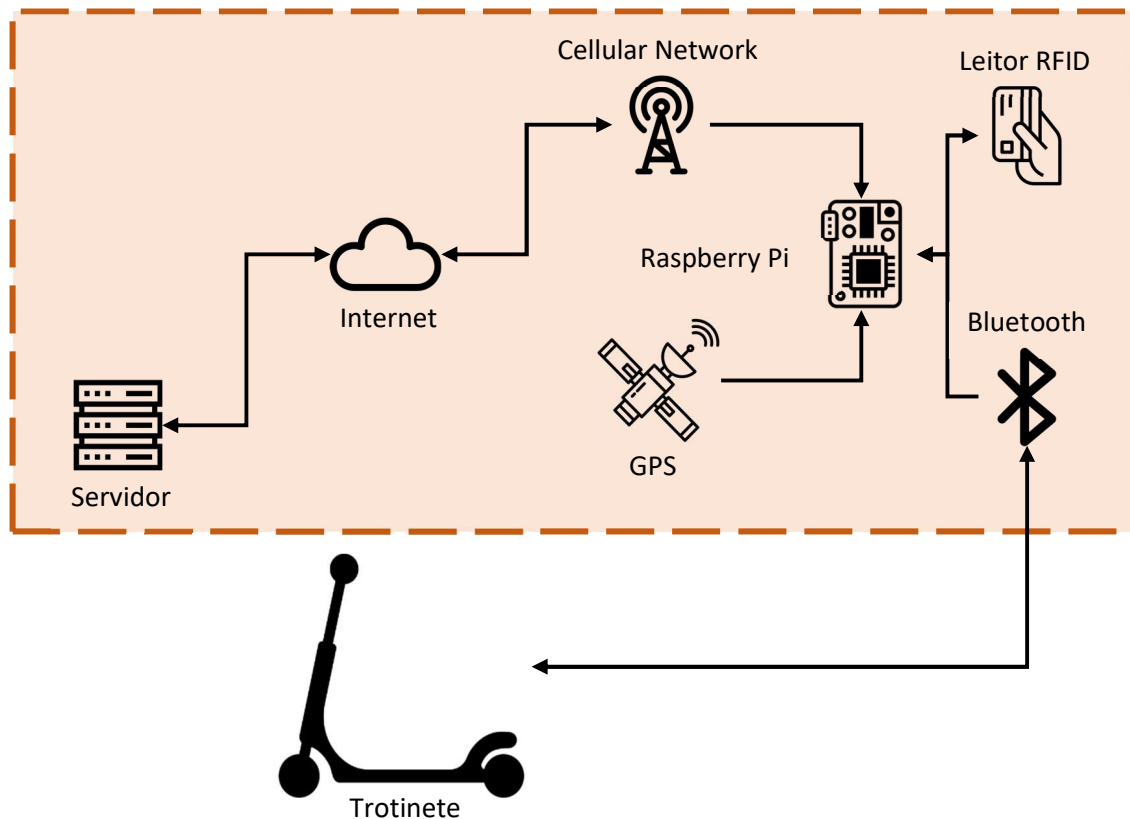


Figura 2: Arquitetura do sistema.

1 Hardware

1.1 Escolha da trotinete

Para a resolução deste problema, primeiramente foi necessário fazer uma pesquisa para encontrar a trotinete que melhor se adaptava ao nosso sistema.

	XIAOMI M365	SEGWAY ES1	XIAOMI M365 Pro
AUTONOMIA	30 km	25km	45km
CUSTO	399€	449€	499€
BLUETOOTH	Sim	Sim	Sim
TRAVÃO	Disco c/ ABS	S/ Disco	Disco c/ ABS
PNEUS	Pneumático	Rígidos	Pneumático

Tabela 1: Tabela comparativa de trotinetes.

Após uma análise detalhada, a trotinete escolhida foi a Xiaomi M365, uma vez que, comparativamente com as trotinetes acima apresentadas, esta oferece:

- Bluetooth para se estabelecer a comunicação;
- Travão de disco com ABS, para uma travagem mais curta e segura;
- Equilíbrio entre preço e autonomia.

1.2 Componentes

O sistema TUC é composto pelos seguintes componentes:

- O **Raspberry Pi** é o cérebro da operação que interconecta todos os módulos;
- O **módulo GPS** está conectado ao Raspberry Pi, por uma comunicação série, e permite obter a localização da trotinete;
- O **módulo Bluetooth** está integrado no Raspberry Pi e permite a comunicação com a trotinete, para enviar ordens e recolher informação;
- A **Modem 3G USB** está conectada ao Raspberry Pi e permite que este esteja conectado à Internet para enviar todas as informações importantes (localização, bateria, etc.) da trotinete através de MQTT;
- O **leitor RFID** está conectado ao Raspberry Pi, por SPI, e permite obter as informações do cartão do utilizador para verificar a sua identidade.

1.3 Instalação dos componentes

Esta secção serve para descrever como foram instalados os componentes necessários ao funcionamento do sistema TUC no Raspberry Pi. Está descrita a instalação do hardware e dos pacotes e bibliotecas necessárias ao funcionamento do hardware. Também já enunciado aqui as bibliotecas e pacotes de software necessários ao funcionamento do programa Scooter Client.

Para esta solução foram ligados 3 componentes ao Raspberry Pi (o Bluetooth vem incorporado):

1. Neo 6M GPS
2. RFID RC522
3. Modem 3G USB Huawei E3131

A comunicação entre estes componentes e o microcontrolador é efetuada através de uma ligação SPI e, para isso, foi necessário ativá-las, seguindo os seguintes passos:

1. Abrir o terminal;
2. Escrever “sudo raspi-config”;
3. Selecionar a opção “Interfacing Options”;
4. Selecionar a opção “SPI”;
5. Escrever no terminal “sudo reboot”.

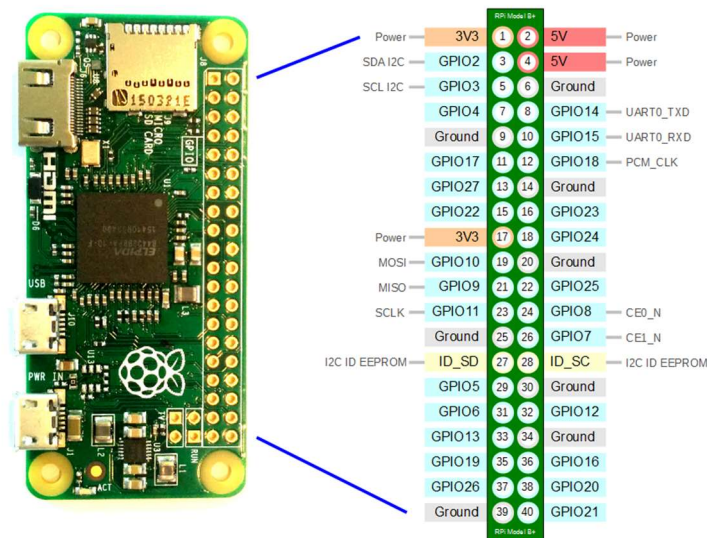


Figura 3: Diagrama de pinos do Raspberry Pi zero w.

O módulo RFID tem 8 pinos. O pino de interrupção não vai ser utilizado, pois, o sistema operativo que corre no Raspberry é derivado do Linux e trabalhar com as interrupções pode gerar problemas no sistema.

- SDA (Serial Data Signal) conectado ao pino 24;
- SCK (Serial Clock) conectado ao pino 23;
- MOSI (Master Out Slave In) conectado ao pino 19;
- MISO (Interrupt Request) conectado ao pino 21;
- GND (Ground Power) conectado ao pino 6;
- RST (Reset-Circuit) conectado ao pino 22;
- 3.3V conectado ao pino 17.

O modem 3G foi ligado ao Raspberry Pi através da porta micro USB com um adaptador USB tipo A (entrada) para micro USB tipo B (saída). O modem 3G quando se liga ao Raspberry, liga-se como uma pen de armazenamento, por isso é necessário algumas configurações no Raspberry para usar o modem como acesso à internet:

1. Abrir terminal;
2. Executar este comando “sudo apt-get install”;
3. Executar este comando “sudo apt-get ppp usb-modeswitch wvdial”.

Após a execução destes comandos agora o modem 3G deve-se ligar ao Raspberry como um ponto de acesso à internet, mas só pode ser usado com o WiFi desligado. Também é necessário ir à página do modem através do browser, aceder ao endereço ‘<http://192.168.1.1>’ e colocar o código pin do cartão SIM no local que a página pedir.

O módulo GPS tem 4 pinos, o pino de alimentação VCC que se liga ao pino 1 (3.3V) do Raspberry, o pino TX que se liga ao pino 15 (RX), o pino RX que se liga ao pino 14 (TX) e o pino GND que se liga ao pino 6.

Por definição, utiliza a UART em vez da *Serial Console* e no Raspberry a UART está designada à consola do Linux. Tendo isto em conta, além de ligar a os módulos e ativar o SPI foi necessário fazer algumas configurações no Raspberry como desligar a serial console, designar o Bluetooth para a mini UART e designar o GPS para a UART primária:

1. Abrir o terminal;
2. Escrever “sudo nano /boot/config.txt”;
3. Escrever no final do ficheiro “dtparam=spi=ondtoverlay=pi3-miniuart-btcore_freq=250enable_uart=1force_turbo=1”;
4. Pressionar ctr+x, escrever y e pressionar enter (guardar o ficheiro);
5. Escrever “sudo nano /boot/cmdline.txt”;
6. Apagar “console=ttyAMA0,115200”;
7. Pressionar ctr+x, escrever y e pressionar enter (guardar o ficheiro);
8. Escrever “sudo reboot”.

Depois de feita a preparação foi necessário instalar as bibliotecas para utilizar os componentes. Para tal, digitámos no terminal os seguintes comandos:

1. git clone <https://github.com/lthiery/SPI-Py.git>;
2. pip3 install pynmea2;

A biblioteca do RFID teve de ser alterada pois não suporta python3 e não era totalmente compatível com a biblioteca SPI, podendo ser encontrada no nosso repositório: <https://github.com/Grupo-3-PEI/tuc/blob/master/Scooter%20Software/MFRC522.py>.

É necessário instalar a biblioteca m365py para permitir a comunicação Bluetooth entre o Raspberry Pi e a trotinete para isso é necessário executar as seguintes instruções:

1. “sudo apt install libglib2.0-dev python3-pip git vim -y”;
2. “sudo pip3 install bluepy”;
3. “sudo pip3 install
git+https://github.com/AntonHakansson/m365py.git#egg=m365py”;

Ainda é necessário obter o MAC Address Bluetooth da trotinete para isso é necessário ligar a trotinete e executar o seguinte comando:

1. “sudo python3 -m m365py”.

O MAC Address que o comando imprime vai ser necessário para depois ser substituído na linha código que executa o método que permite o emparelhamento entre a trotinete e o Raspberry Pi. Por exemplo:

```
scooter_mac_address = 'D6:0E:DB:7B:EA:AB'
scooter = m365py.M365(scooter_mac_address, handle_message)
scooter.set_connected_callback(connected)
scooter.set_disconnected_callback(disconnected)
scooter.connect()
```

Figura 4 Exemplo de código de emparelhamento com a trotinete.

Para finalizar é necessário instalar os pacotes do protocolo MQTT para python3:

1. “pip3 install paho-mqtt”.

2 Software

Foi implementado o programa, no Raspberry Pi, que controla a trotinete por Bluetooth, recebe os dados do leitor de RFID, recebe e processa os dados do sensor GPS e faz o envio e processamento das mensagens para o servidor. Este programa é o “Scooter Client”.

Ao nível do software, foi implementada uma API que permite a comunicação entre o servidor e a dashboard. Servidor este, dividido em 2 programas: Um pelo protocolo MQTT e outro pelo protocolo HTTP.

Este processa e demonstra todos os dados da trotinete na dashboard, permite ao gestor configurar o geofencing, faz a autenticação do utilizador, hospeda um broker MQTT, um para permitir a troca de mensagens entre a trotinete e o servidor porque este é o agente que distribui as mensagens entre clientes, e a página da web, também é um cliente MQTT para obter os dados da trotinete. A página web está dividida em dois portais, um para o gestor do sistema e outro para o utilizador final.

2.1 Scooter Client

O Scooter Client é um programa em python que é executado no Raspberry Pi para interagir com leitor de cartões RFID, com o sensor de GPS e com a trotinete através de Bluetooth e enviar/receber mensagens do servidor através do protocolo MQTT, fazendo uso de várias bibliotecas que permitem controlar os vários módulos do hardware.

O protocolo MQTT faz uso do conceito de tópicos, isto é um cliente pode subscrever um tópico, e assim sempre que alguma publicação for feita nesse tópico o cliente vai recebê-la. O cliente também pode publicar nos tópicos que desejar. Para permitir a distribuição das mensagens usa-se um broker, que vai ser descrito mais à frente. O cliente deve-se conectar ao broker através da função “connect” da biblioteca “paho.mqtt.client” e deve ser passado o IP e porta do broker ou URL. O cliente está subscrito ao tópico “trotinete/#/cardanswer” (o cardinal representa o id da trotinete), neste tópico vem a resposta do servidor em relação à autenticação do cartão do utilizador que foi lido e também caso a trotinete esteja fora dos limites definidos de circulação é enviado o conteúdo “nok” que bloqueia a trotinete. Para desbloquear a trotinete é enviado o conteúdo “ok”. A função que processa a receção de mensagens é a função “on_message” que corre numa *thread* que é iniciada quando o cliente MQTT se conecta ao broker, que processa as mensagens escritas no tópico subscrito e estas mensagens só podem ter o conteúdo “nok” ou “ok”. Caso seja “nok” bloqueia a trotinete caso seja “ok” desbloqueia a trotinete. Este desbloqueio/bloqueio é a alteração de valor de uma variável booleana global “l” que guarda o estado de bloqueio da trotinete, se for True está bloqueada, se for False está desbloqueada. Para o envio de mensagens é usado o método “publish” da biblioteca “paho.mqtt.client”, por exemplo:

```
client.publish("trotinete/1/location", gps)
```

Figura 5: Exemplo de publish.

Para processar simultaneamente os dados vindos da conexão Bluetooth, do GPS e do leitor RFID são usadas *threads* para as seguintes funções:

- “rfid_read” que processa os dados do leitor;
- “gps” que processa os dados do gps;
- “scooter_control” que realiza a comunicação Bluetooth com a trotinete e envia estes dados também para o servidor;
- “keep_alive” que mantém a conexão com a trotinete.

Todas estas funções são executadas dentro das *threads* respetivas em ciclo infinito, só sendo interrompidas com a finalização do programa.

A função “rfid_read” realiza a leitura do id do cartão e o envio do mesmo para o servidor. Esta função utiliza a biblioteca MFRC522 que serve de interface para o módulo leitor de RFID. A função começa por verificar se há novos cartões que podem ser lidos. De seguida, se detetar um cartão, lê o identificador do cartão e envia para o nosso servidor, através do método *publish* no tópico “trotinete/#/cardrequest”, o cardinal é substituído pelo id da trotinete, e o conteúdo da mensagem é o id do cartão em *string*.

A função “gps” é responsável pela obtenção da localização da trotinete. A função começa por configurar a porta série para leitura da informação do GPS, em seguida lê a informação e faz o *parse* da mesma por forma a ser legível pelo servidor, para finalmente enviá-la através do método *publish* no tópico “trotinete/#/location”. O cardinal é substituído pelo id da trotinete, e o conteúdo da mensagem são as coordenadas “\$\$\$\$,****”, \$ é a longitude e * é a latitude, enviadas no formato *string*.

A função “scooter_control” é responsável pelo emparelhamento por Bluetooth com a trotinete, sendo possível após o emparelhamento realizado trocar ordens. Para as comunicações Bluetooth é usada a biblioteca m365py que contém métodos para a realização das comunicações. Na função “scooter_control” começamos por conectarmos o Raspberry Pi à trotinete através do método “connect” da biblioteca referida anteriormente, a partir deste momento a trotinete está emparelhada com o Raspberry Pi. De seguida fazemos o pedido para obter o valor da percentagem da bateria, através do método “request” da biblioteca referida anteriormente e com este método também se fazem pedidos para enviar ordens à trotinete. De seguida fazemos o pedido para ligar o cadeado da trotinete. Este cadeado é um modo especial da trotinete em que ela enquanto está ligada se a tentarem usar, ela bloqueia a roda da frente e liga um alarme sonoro enquanto pisca as luzes traseiras. Após estas instruções passamos para o ciclo infinito onde se começa por verificar a variável global “l” que já foi referida anteriormente (por predefinição está a True). Se estiver a igual a True fazemos o pedido para desligar a luz traseira e para ligar o cadeado, se estiver igual a False fazemos o pedido para desligar o cadeado e para ligar a luz traseira. Após o *if statement* realizamos o pedido para obter o valor da percentagem de bateria, os valores que a trotinete envia são escritos num ficheiro JSON, de forma automática pela biblioteca referida anteriormente, recolhemos esse valor do ficheiro e enviamo-lo para o servidor através do método “publish” no tópico “trotinete/#/battery” (o cardinal é substituído pelo id da trotinete) e o conteúdo da mensagem é o valor da percentagem de bateria em formato *string*.

A função “keep_alive” serve para manter a trotinete ligada evitando que esta entre em standby. Após realizarmos testes à ligação do Raspberry Pi com a trotinete descobrimos que se ela estiver com o cadeado e passarem três horas em inatividade ela desliga-se sendo então necessário voltar a emparelhar a trotinete e voltar a ligar o cadeado. Como a trotinete se desliga

de forma abrupta sem desemparelhar a biblioteca referida anteriormente gera um erro que interrompe a execução do programa, por isso decidimos criar esta função. De uma em uma hora verificamos o estado da variável global “l” e se for “True” o valor da variável é alterado para “False” durante um segundo voltando a “True” no final desse segundo. Ao fazermos isto, durante um segundo a trotinete desliga e liga o cadeado o que é o suficiente para realizar o reset ao tempo de inatividade do controlador interno da trotinete.

2.2 API para a interação com a página

A API funciona através da framework FLASK e através da mesma, é possível comunicarmos com o JavaScript e assim, interagir com o dashboard. Para a instalação do FLASK, basta seguirmos os seguintes passos:

- **Criar o ambiente virtual:**
 - > \$ sudo apt install python3-venv
 - > \$ mkdir my_flask_app
 - > \$ cd my_flask_app
 - > \$ python3 -m venv venv
- **Iniciar o ambiente virtual:**
 - > \$ source venv/bin/activate
- **Instalar o Flask no venv:**
 - > (venv) \$ pip install Flask
- **Desativar o ambiente virtual:**
 - > (venv) \$ deactivate

Documentação da API:

- ***index()***: retorna a página html inicial.
- ***check_credentials()***: Verifica se as credenciais inseridas no login do gestor estão corretas.
- ***login page()***: retorna a página html do login.
- ***menu()***: retorna a página html do menu.
- ***more_info()***: retorna a página html dos detalhes da trotinete.
- ***map1()***: Retorna o as coordenadas das trotinetes, lidas no ficheiro geojson.
- ***scooter_list()***: Retorna uma string com a lista de todas as trotinetes.
- ***user_list()*** : Retorna uma string com a lista de todos os utilizadores
- ***delete_user()***: Elimina um utilizador recebido pelo AJAX, do ficheiro dos utilizadores.
- ***add_user()*** : Adiciona um utilizador, ao ficheiros dos utilizadores.
- ***get_useTime()***: Retorna os detalhes de utilização de uma trotinete, cujo id é recebido pelo AJAX.
- ***geoFencing()***: Escreve as coordenadas dos pontos do geofencing num ficheiro.
- ***geomapping()***: Retorna o as coordenadas do geofencing, lidas no ficheiro geojson.

2.3 MQTT Server

O servidor MQTT utiliza o broker Mosquitto, dado que a sua instalação e utilização são bastante simples, é compatível com todo o tipo de máquina, incluindo as mais antigas e é bastante eficaz sem prejudicar a efetividade de um computador.

Para proceder à sua instalação basta seguir os seguintes passos:

- **Para instalar docker:**
 - > \$ curl -fsSL <https://get.docker.com> -o get-docker.sh
 - > \$ sudo sh get-docker.sh
- **Para instalar mosquitto:**
 - > \$ docker pull eclipse-mosquitto

De seguida, criar alguns diretórios que serão usados pelo Mosquitto. Estes serão usados para logs e data:

```
> $ mkdir -p /var/mosquitto/logs
> $ mkdir -p /var/mosquitto/data
> $ chmod ugo+w -R /var/mosquitto
```

Agora, precisamos de criar um ficheiro de configuração. Para tal, precisamos de criar um ficheiro com o nome: “mosquitto.conf”. No nosso caso, criámo-lo na localização /root/mosquitto.conf. Inserimos as seguintes configurações básicas no ficheiro acabado de criar:

```
> persistence true
> persistence_location /mosquitto/data/
> log_dest file /mosquitto/log/mosquitto.log
```

Estamos prontos para criar e executar o container. Executámos os seguintes comandos para iniciar o container, encaminhar para os portos 1883 e 9901, e montar os ficheiros e diretórios para a localização correta.

```
> $ docker run -d -it -p 1883:1883 -p 9901:9901 -v
/root/mosquitto.conf:/mosquitto/config/mosquitto.conf -v
/var/mosquitto/data:/mosquitto/data -v
/var/mosquitto/log:/mosquitto/log eclipse-mosquitto
```

Documentação do servidor MQTT:

- **Classe *Scooter***
 - ***class Scooter(id)***: Classe de uma trotinete. Recebe apenas o ID como argumento.
 - ***Scooter.get_state()***: Preenche os atributos de uma Scooter (state, battery, user, location, geofencing) ao ler a linha do ficheiro de texto das trotinetes, corresponde ao seu id.
 - ***Scooter.update_scooter()***: Atualiza o ficheiro o scooters.csv com as informações da trotinete.
- **Classe *Trotinete***
 - ***Check_id(id)***: verifica se o ID da trotinete existe no ficheiro.
 - ***Insert_scooter(id,status,battery,user,location,geofencing)***: Insere uma nova trotinete, no ficheiro.

- ***Update_scooters(id,status,battery,user,location,geofencing)***: Atualiza o ficheiro o scooters.csv com as informações passadas.
- ***Get_ids()***: Retorna uma lista com o id de todas as trotinetes.
- ***Get_last_ids()***: retorna o id da última trotinete.
- **Utilização**
 - ***Is_full()***: Verifica se uma trotinete já tem 20 entradas no seu ficheiro de utilização. Retorna True, se tal for verificado.
 - ***Save_unlock()***: Guarda no scooters_use, a data e hora que a trotinete foi desbloqueada
 - ***Save_lock()***: Guarda no scooters_use, a data e hora que a trotinete foi bloqueada
- **Utilizadores**
 - ***Insert_user(id,name,email)***: insere um utilizador no people.csv
 - ***Update_users(id,name,email)***: Atualiza o ficheiro o people.csv com as informações passadas.
 - ***Get_user_ids()***: Retorna uma lista com o id de cada utilizador
- **Geofencing**
 - ***Write_location()***: Escreve a localização das trotinetes recebidas pelo broker, no ficheiro de coordenadas geojson.
 - ***Is_inside_geofencing(location)***: Retorna true, se a trotinete estiver dentro dos limites do geofencing. Retorna falso caso contrário.
 - ***GetGeofencingPoints()***: Lê o ficheiro csv das coordenadas geofencing(geofencing.csv) e retorna-as.
- **MQTT**
 - ***On_connect()***: Subscrive os tópicos MQTT de todas as trotinetes.(battery,location,cardrequest).
 - ***On_message()***: Executa o programa, quando recebe uma mensagem num tópico ao qual subscreveu.

2.4 Web Server

Para o protocolo HTTP foi escolhido um servidor Apache. Para a sua instalação, basta seguir os seguintes passos:

- > \$ sudo apt update
- > \$ sudo apt install apache2

Se inserirmos o nosso endereço IP na barra de navegação é aberta a página de default do apache. Podemos então criar a pasta para o nosso website em /var/www e apelidá-lo de gci. Criamos então um ficheiro html:

- > \$ sudo mkdir /var/www/gci/
- > \$ cd /var/www/gci/
- > \$ nano index.html

Podemos então, configurar o virtualhost:

- > \$ cd /etc/apache2/sites-available/
- > \$ sudo cp 000-default.conf gci.conf
- > \$ sudo nano gci.conf

Adicionamos as seguintes linhas:

```
> ServerAdmin yourname@example.com //Opcional: Se quisermos ser contactados
pelos users
> DocumentRoot /var/www/gci/
> ServerName gci.example.com // Se tivermos um domínio
```

De seguida ativamos o VirtualHost:

```
> $ sudo a2ensite gci.conf
```

Sempre que quisermos interagir com o servidor, executamos os seguintes comandos:

```
> $ sudo service apache2 reload // Se quisermos recarregar o servidor
> $ sudo service apache2 stop // Se quisermos parar o servidor
> $ sudo service apache2 start // Se quisermos iniciar o servidor
```

3 Interação com o utilizador

Quando o utilizador, previamente autorizado e inserido na base de dados, aproxima o cartão no leitor RFID de uma trotinete disponível, o sistema TUC desbloqueia a trotinete. Depois de ter desbloqueado a trotinete, o utilizador realiza o seu percurso e, quando regressar do seu destino, volta a aproximar o seu cartão ao leitor, para bloquear a trotinete.

3.1 Desbloqueio da trotinete

O utilizador, que queira usufruir do sistema TUC, tem duas formas de saber se uma trotinete está disponível: ou acede pela internet ao portal do utilizador e através do mapa verifica a localização e a disponibilidade da trotinete ou verifica na trotinete se o LED traseiro está desligado.

Após saber a disponibilidade da trotinete e esta se verificar disponível, o utilizador passa o cartão da UA no leitor RFID. Se o cartão for validado, a trotinete emite um som e ligará a luz traseira, passando a estar em utilização. Para cartões não validados, se passado 10 segundos do cartão estar encostado não acontecer nada, quer dizer que o utilizador não tem autorização ou que a trotinete está fora do limite de circulação (geofence).

Para a validação do cartão é lido o ID do cartão, que será enviado para o servidor através do protocolo MQTT. Quando chega uma mensagem, este verifica se o cartão está ou não autorizado a desbloquear a trotinete, e depois verifica se a trotinete está em uso.

Para voltar a bloquear a trotinete o processo é o mesmo que o desbloquear, assumindo que o cartão que está a ser passado no leitor é válido no nosso sistema e que a trotinete foi desbloqueada pelo mesmo. O cartão é passado no leitor, é enviado o ID, o servidor recebe a mensagem. Como ela já está em uso e verifica se o cartão que foi enviado é o mesmo que a desbloqueou. Se os cartões coincidirem, envia uma mensagem para bloquear, caso contrário, mantém a trotinete desbloqueada.

O processo de desbloqueio da trotinete encontra-se ilustrado na figura seguinte.

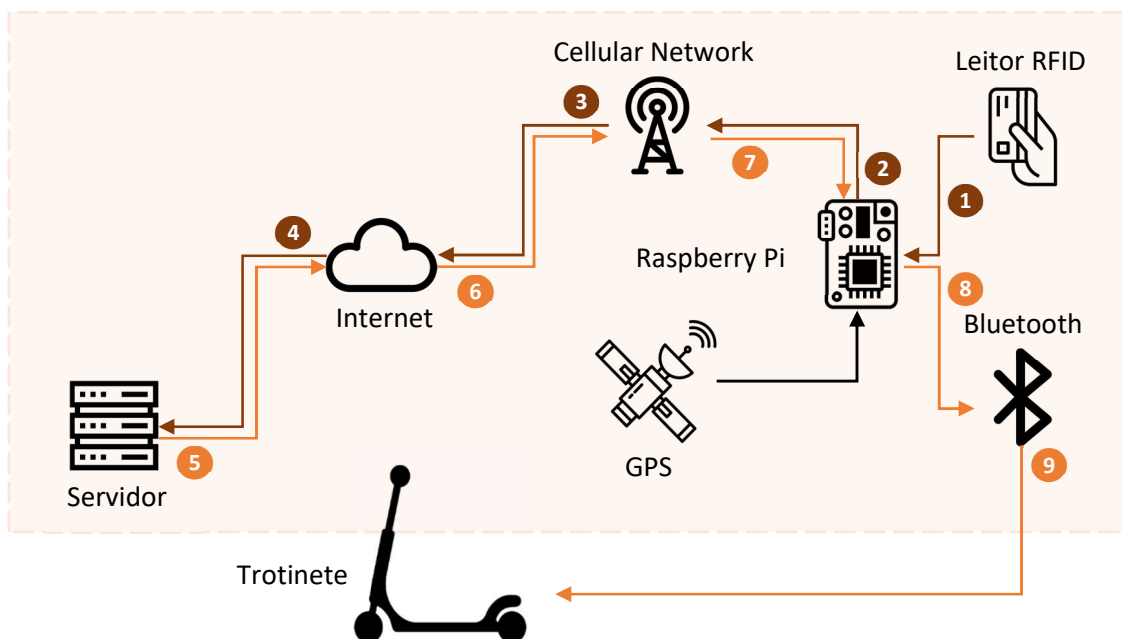


Figura 5: Sistema de desbloqueio da trotinete.

3.2 Geofencing

Cabe ao gestor do sistema definir os limites geográficos em que a trotinete pode circular livremente. Para a definição dos limites de circulação, o gestor tem acesso a uma secção no portal de administração para definir e modificar a área de circulação, como será abordado mais à frente.

Após delimitar a área de circulação, o servidor guarda os dados do polígono desenhado num ficheiro CSV (geofencing.csv) e, ao mesmo tempo que recebe a localização das trotinetes, compara as coordenadas e verifica se a trotinete está dentro ou fora da geofence.

No momento em que a trotinete ultrapassa a geofence, esta entra em modo lock. Se nesse momento a trotinete estiver a uma velocidade alta, apesar de já ter a instrução de lock guardada, não aplica logo a travagem, só é aplicada quando estiver abaixo de uma determinada velocidade, travando não só a roda da frente, como também dispara o alarme e as luzes de trás entram num estado intermitente. Para voltar a desbloquear a trotinete é necessário que esta volte a estar dentro dos limites definidos pelo utilizador.

3.3 Portal de Administração TUC

Para a criação da dashboard foram utilizadas as três tecnologias: a linguagem de marcação **HTML**, que fornece a estrutura básica dos websites; a linguagem de folhas de estilo **CSS**, que é usada para definir a apresentação, disposição e formatação do mesmo; e a linguagem de programação **JavaScript**, responsável pela dinâmica de comportamento dos diferentes elementos. Além destas tecnologias, recorreu-se ao uso da biblioteca mais popular de JavaScript: **jQuery**. Esta biblioteca permite uma navegação mais simples no documento HTML, comunicar mais facilmente com a API e permite a utilização da tecnologia **AJAX**, facilitando a interação com o utilizador.

Cada bloco de dados é atualizado independentemente e assíncronamente, utilizando o método ***setTimeout(function, miliseconds)***. Dado que o AJAX é assíncrono, os tempos definidos para o método `setTimeout()` nas diferentes funções, têm que ser distintos.

Para aceder a este portal, o gestor tem de inserir as suas credenciais de acesso, como ilustrado abaixo (Figura 7).

De momento, as credenciais são “admin” “admin”, email e password respetivamente. Esta informação é enviada e processada através da API entre o servidor e a dashboard.

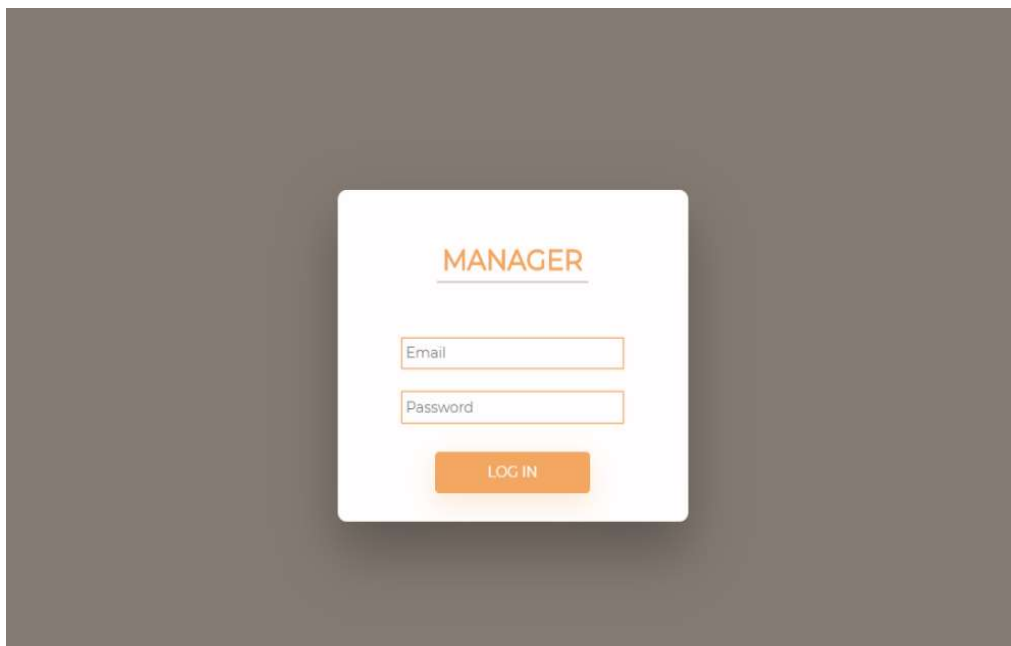


Figura 6: Início de sessão do portal de administração.

No portal de administração é fornecida informação sobre as trotinetes, utilizadores, localização das trotinetes e o *geofencing*, como é ilustrado na figura abaixo.



Figura 7: Funcionalidades do portal de administração TUC.

A informação das trotinetes encontra-se numa lista com informação sobre o ID, estado e bateria e se a respetiva trotinete se encontra dentro dos limites definidos, como ilustrado na Figura 8. Esta informação é guardada num ficheiro CSV(scooters.csv) que é constantemente atualizado.




ID	STATUS	BATTERY	GEOFENCING
1	IN USE	75	INSIDE
2	AVAILABLE	100	INSIDE
3	AVAILABLE	100	INSIDE
4	AVAILABLE	100	OUTSIDE

Figura 8: Lista de trotinetes.

Caso o gestor pretenda obter o histórico de utilização de uma trotinete, tem de seleccioná-la e será listada informação sobre o ID do cartão da UA e nome do utilizador, data e hora de desbloqueio e bloqueio, como ilustrado na figura 10, informação essa também guardada num ficheiro CSV (scooter_use.csv). Este ficheiro guarda as últimas 20 entradas de cada trotinete.

Tanto as informações sobre o estado, como a bateria, são enviadas pela trotinete para o servidor através de publicações nos tópicos “trotinete/#/cardrequest” e “trotinete/#/battery” respetivamente, sendo # o id da trotinete.



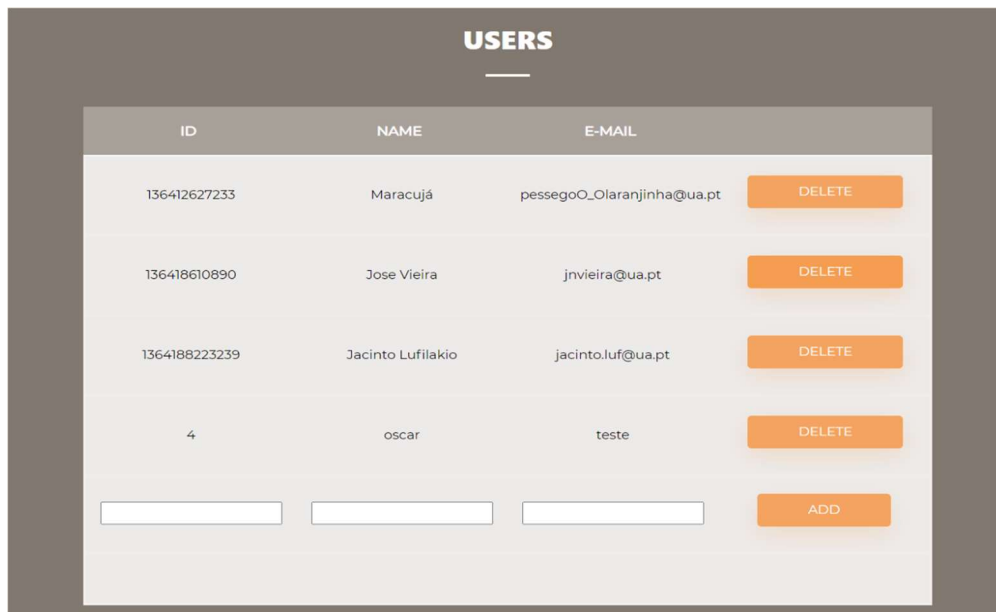
CARD ID	NAME	UNLOCK TIME	LOCK TIME
136412627233	Maracujá	01/06/2020 15:37	01/06/2020 15:38
136412627233	Maracujá	01/06/2020 18:44	01/06/2020 18:45
136412627233	Maracujá	01/06/2020 20:24	01/06/2020 20:37
136412627233	Maracujá	01/06/2020 20:30	01/06/2020 20:37
136412627233	Maracujá	02/06/2020 21:39	02/06/2020 21:39

Figura 9: Histórico de utilização da trotinete.

A informação dos utilizadores com permissão para utilizarem as TUC, encontra-se numa lista com informação sobre o ID do cartão da UA, nome e e-mail do utilizador. Esta informação é guardada num ficheiro CSV (people.csv) que é atualizada sempre que adicionamos ou removemos um utilizador.

Se o gestor pretender eliminar um utilizador, basta clicar no botão “Delete” e se pretender adicionar um novo utilizador tem de preencher os dados pedidos e clicar no botão “Add”, como ilustrado na figura 11.

Se o utilizador desconhecer o seu ID do cartão, é apenas necessário passá-lo no leitor de cartões de uma das trotinetes, que irá aparecer no terminal, o seu respetivo número do cartão.



ID	NAME	E-MAIL	
136412627233	Maracujá	pessegoO_Olaranjinha@ua.pt	DELETE
136418610890	Jose Vieira	jnvieira@ua.pt	DELETE
1364188223239	Jacinto Lufilakio	jacinto.luf@ua.pt	DELETE
4	oscar	teste	DELETE
<input type="text"/>	<input type="text"/>	<input type="text"/>	ADD

Figura 10: Lista de utilizadores.

Quanto à localização das trotinetes, é apresentada num mapa, em tempo real, com a localização das trotinetes. A exibição desse mapa é possível através da utilização da biblioteca de JavaScript, **OpenLayers**. Esta biblioteca, permite também a disposição de coordenadas geográficas, que representam então a localização das trotinetes. Esta informação é lida diretamente de um ficheiro geojson. Cada uma das trotinetes é representada com um círculo que pode ser verde, caso esteja disponível, caso contrário estará a vermelho.

Para uma vista mais simplificada, é possível saber a percentagem da bateria e o ID da trotinete, ao passar o cursor em cima de cada círculo. Para esta comunicação ser possível, a trotinete publica regularmente no tópico “trotinete/#/location” (sendo “#” o id da trotinete), com a sua localização.

A figura seguinte ilustra o mapa de localização das trotinetes.

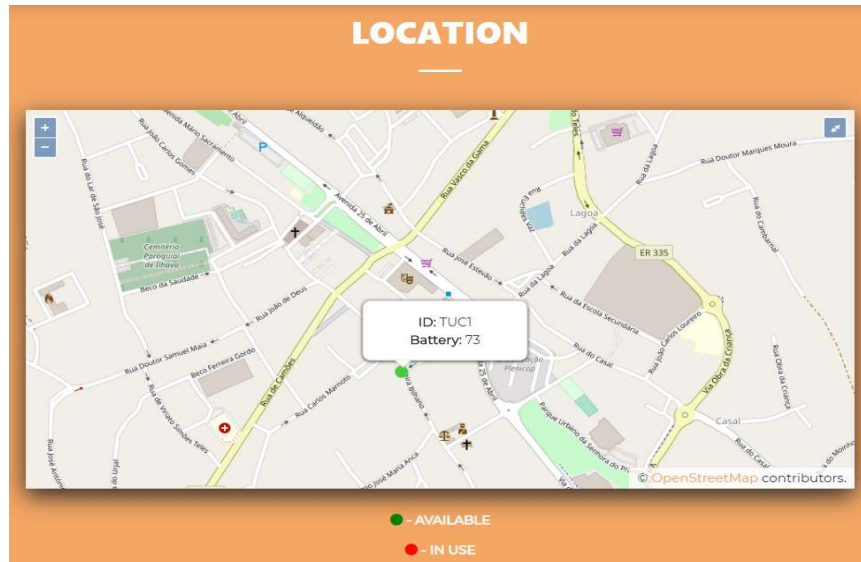


Figura 12: Localização das trotinetes.

Relativamente ao *geofencing* é apresentado um mapa onde é permitido ao gestor modificar a área de circulação, selecionando no mapa os pontos que este deseja que delimitem a área. Também pode modificar a área atual, através da opção “Modify” e arrastando os vértices/arestas da área já definida anteriormente, para a posição desejada. Para submeter as alterações, basta que o gestor selecione a opção de “Confirm”.

Tal como referido anteriormente, a tecnologia que possibilita a exibição do mapa, a inserção de pontos e o desenho do respetivo polígono no mesmo, é a biblioteca **OpenLayers**.

O gestor pode limitar a área com um número máximo ilimitado de pontos, ou seja, pode fazer um polígono com 5, 50 ou mais vértices, sendo que o número mínimo de vértices para ser considerado um polígono é 3.

Esta informação é escrita num ficheiro geojson e vai ser processada através da biblioteca Shapely em python, que dado um conjunto de pontos, consegue criar um polígono e verificar se um ponto está contido nesse polígono.

Para a **instalação do Shapely**, basta executar o seguinte comando:

```
> $ pip3 install shapely
```



Figura 13: Geofencing.

Documentação da dashboard:

- ***setTimeout(function,milliseconds)***: executa uma função(function) ao fim de x milissegundos (milliseconds).
- ***loadMap()***: inicializa os mapas da dashboard – mapa da localização e mapa do geofencing.
- ***LoadLocation()***: cria o mapa da localização.
- ***LoadPointsLoop()***: carrega as localizações das trotinetes, lidas num ficheiro geojson com as respetivas coordenadas, para o mapa de localização. Este processo é executado sistematicamente.
- ***loadGeo()***: cria o mapa do geofencing e permite o desenho de polígonos sobre o mesmo.
- ***confirmGeo(source)***: recebe os pontos desenhados no mapa do geofencing e envia-os para o servidor, no URL '/geoFencing'.
- ***\$("#add").on('click', function())***: recebe informações dos campos de inserção do novo utilizador, adiciona-as à tabela e envia-as para o servidor, no url '/add_user'.
- ***\$("#trot tr").click(function())*** : recebe o id da linha correspondente à que o utilizador selecionou, guarda esse mesmo id num escopo global e abre a página html: *details.html*.
- ***\$(document).on('click', '.menos', function())***: recebe um elemento, descobre o seu ID, elimina-o da tabela e envia-o para o servidor no url '/delete_user' .
- ***(function updateDetails())***: envia o id correspondente à trotinete selecionada pelo utilizador para o servidor. Recebe então as informações sobre a utilização da trotinete e apresenta-as no ficheiro *details.html*.
- ***(function updateTrots())***: recebe do servidor, as informações das trotinetes e apresenta-as na tabela Scooter List.
- ***(function updateUsers())***: recebe do servidor, as informações dos utilizadores e apresenta-as na tabela User List.
- ***\$("#login").click(function())***: envia os dados de autenticação inseridos pelo utilizador para o servidor. Após receber uma resposta, pode seguir para a página de gestor ou retornar um alerta.

3.4 Portal do utilizador

No portal do utilizador é apenas fornecida informação sobre a localização das trotinetes, em tempo real, como é ilustrado na figura 14.

Tal como acontece no portal do gestor, a trotinete é representada com um círculo a verde se estiver disponível, caso contrário a vermelho.

O utilizador tem ainda acesso a algumas informações sobre as trotinetes, como a percentagem da bateria e o ID da trotinete, ao passar o cursor em cima de cada círculo.

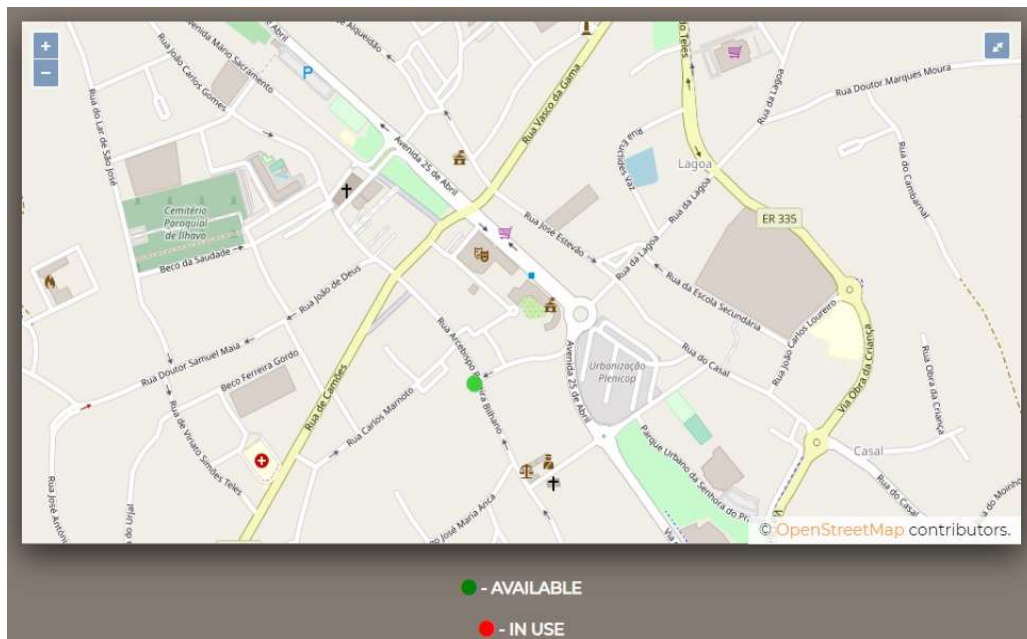


Figura 14: Informação e localização das trotinetes.

RESULTADOS

Para a implementação deste sistema foram comprados os seguintes componentes:

	PREÇO	LINK
Leitor RFID RC522	0.76€	https://pt.aliexpress.com/item/32803014822.html?src=google
GPS NEO-6m	25.70€	https://www.electrofun.pt/comunicacao/modulo-gps-gy-neo-6m-v2
Raspberry Pi zero W	10.44€	https://www.kubii.fr/les-cartes-raspberry-pi/1851-raspberry-pi-zero-w-kubii-3272496006997.html?src=raspberrypi
Huawei E3131	16.55€	https://pt.aliexpress.com/item/33003528502.html?src=google&gclid=CjwKCAjwztL2BRATEiwAvnALcjh8z8vNGAB57IAV1skil3PfsApDeA724PfrZhpSeGz1b0qM6UG1HhoC4uQQAvD_BwE
Trotinete Xiaomi M365	399€	-

Tabela 2: Lista de componentes.

Esta solução tem um custo total aproximadamente de 460€, por trotinete, excluindo contrato de operadora para dados móveis. Apresenta um consumo de, aproximadamente, 2.07W e a trotinete consegue aguentar 19 horas em standby sem ter de ser carregada.

Conseguimos realizar uma comunicação entre as trotinetes e o servidor/dashboard em espaços de tempo muito reduzidos, que não seriam possíveis se tivéssemos optado por utilizar a rede LORA como inicialmente tínhamos planeado.

A adição de utilizadores é feita de maneira simples e intuitiva para o gestor.

É possível observarmos a informação e a localização das trotinetes em tempo real, sem utilizador ser obrigado a atualizar a página. É também possível consultar as últimas utilizações de cada trotinete, bem com o respetivo utilizador.

CONCLUSÕES

O sistema foi implementado com sucesso, cumprindo com todos os requisitos pedidos. No decorrer do desenvolvimento do projeto, encontrámos alguns problemas, no entanto fomos capazes de os solucionar.

Ao cliente será entregue, como produto final:

- Todo o software desenvolvido, que pode ser encontrado no nosso repositório;
- Este documento que contém todas as instruções de instalação;
- O hardware que foi adquirido e montado.

1 Trabalho futuro

Para um trabalho futuro, pretende-se:

- Reduzir o consumo de energia do sistema TUC;
- Notificar o gestor, quando uma trotinete ultrapassa o limite definido;
- Ligar o sistema TUC à bateria da trotinete;
- Conceber uma caixa à prova de água para a trotinete;
- Testar o sistema com utilizadores para aperfeiçoar o sistema;
- Criar uma base de dados para suportar o aumento da escala da solução;
- Aumentar a escala da solução;
- Monetizar a solução.

REFERÊNCIAS

<https://pimylifeup.com/raspberry-pi-rfid-rc522/>

<https://raspberrypi.stackexchange.com/questions/83610/gpio-pinout-orientation-raspberypi-zero-w>

<https://github.com/mxgxw/MFRC522-python/issues/69>

<https://github.com/mxgxw/MFRC522-python>

<https://www.instructables.com/id/Raspberry-Pi-the-Neo-6M-GPS/>

<https://sparklers-the-makers.github.io/blog/robotics/use-neo-6m-module-with-raspberry-pi/>

<https://www.raspberrypi.org/documentation/configuration/uart.md>

<http://www.jamesrobertson.eu/blog/2014/jun/24/setting-up-a-huawei-e3131-to-work-with-a.html>

<https://www.thefanclub.co.za/how-to/how-setup-usb-3g-modem-raspberry-pi-using-usbmodeswitch-and-wvdial>

<https://github.com/AntonHakansson/m365py>

<https://www.ev3dev.org/docs/tutorials/sending-and-receiving-messages-with-mqtt/>

<https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/>

<https://selfhostedhome.com/using-two-mqtt-brokers-with-mqtt-broker-bridging/>

<https://docs.docker.com/engine/install/ubuntu/>

<https://openlayers.org/>