

# ARQUITETURA SISTEMAS DE INTERNET

---

## Messages Dissemination Web App

---

### Grupo N.º 28

*Autores:*

Diogo Rodrigues 84030  
Leandro Almeida 84112  
Pedro Moreira 85228

*Professor:*

João Nuno De  
Oliveira e Silva

21 de Maio de 2020

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>2</b>
1.1	Tecnologias Utilizadas . . . . .	2
<b>2</b>	<b>Arquitetura do Sistema</b>	<b>2</b>
2.1	Função de cada componente . . . . .	3
<b>3</b>	<b>Autenticação</b>	<b>4</b>
3.1	Autenticação do utilizador . . . . .	4
3.2	Autenticação do administrador e Bot . . . . .	5
<b>4</b>	<b>REST API</b>	<b>5</b>
<b>5</b>	<b>Base de dados</b>	<b>6</b>
5.1	Sistema de gerenciamento da base de dados relacional (RDMS) . . . . .	6
5.2	Arquitetura da base de dados . . . . .	7
5.2.1	Utilizadores . . . . .	7
5.2.2	Bots . . . . .	8
5.2.3	Edifícios . . . . .	8
5.2.4	Movimentos . . . . .	8
5.2.5	Mensagens . . . . .	8
<b>6</b>	<b>Funcionalidades</b>	<b>9</b>
<b>7</b>	<b>Interfaces</b>	<b>10</b>
7.1	Utilizadores . . . . .	10
7.2	Administrador . . . . .	11
7.3	Bot . . . . .	12
<b>8</b>	<b>Implementação do sistema na Google Cloud Platform</b>	<b>13</b>
<b>9</b>	<b>Discussão</b>	<b>14</b>

# 1 Introdução

Este projeto consiste numa aplicação web utilizada para gerir a troca de mensagens entre utilizadores a trabalharem em locais de trabalho particulares. Numa organização, como o exemplo do IST, muitos edifícios são acedidos por múltiplos utilizadores. Assim, em caso de emergência, algumas notificações devem ser reencaminhadas para os utilizadores que estão a frequentar determinado edifício. O sistema a construir opera através de três diferentes classes: administradores, utilizadores e *bots*.

- O administrador acede às atividades dos logs, e gere informações estatísticas (os edifícios disponíveis, por exemplo);
- Utilizadores podem enviar mensagens entre si se estiverem separados por um certo *range* ou se estiverem no mesmo edifício. Para isto, os utilizadores enviam as suas coordenadas para o servidor;
- *Bot* são programas que estão atribuídos a certo edifício e que periodicamente enviam mensagens a quem está nesse mesmo edifício.

A aplicação pode ser vista em <https://asint-227820.appspot.com/app/>.

## 1.1 Tecnologias Utilizadas

Para este projeto, o **Back-End** foi realizado utilizando Python3 e a *web framework* Django. O **Front-End** foi realizado com HTML e JavaScript.

# 2 Arquitetura do Sistema

A arquitetura do sistema encontra-se ilustrada na figura 1.

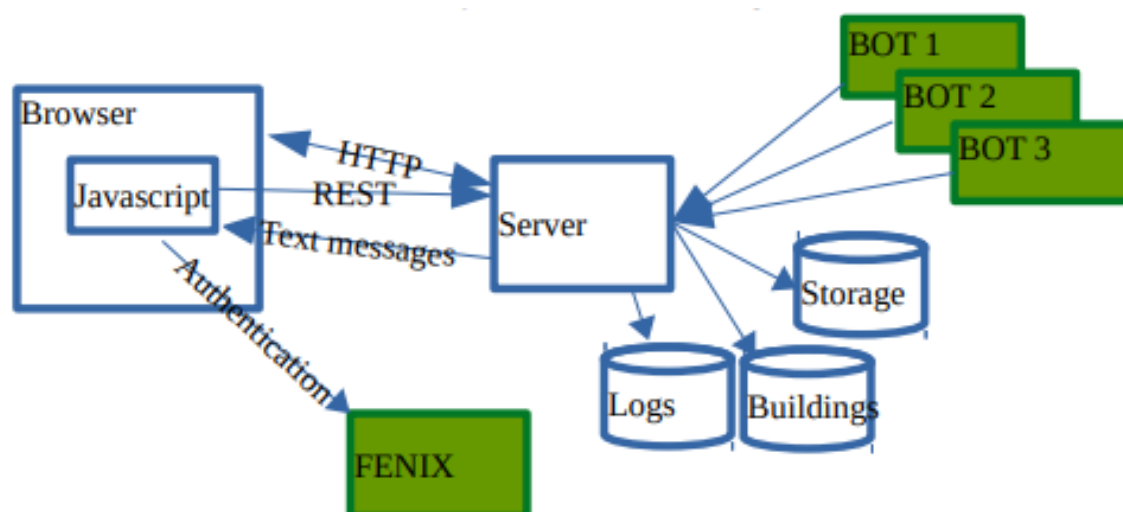


Figura 1: Arquitetura do Sistema

## 2.1 Função de cada componente

O utilizador através de uma interface realizada em **Javascript** e **HTML** comunica com o servidor, fazendo-lhe pedidos **REST**, e recebendo respostas por **HTTP**. O utilizador envia periodicamente as suas coordenadas para o servidor e envia também as mensagens que pretende enviar para os outros utilizadores. Realça-se que para um utilizador conseguir utilizar a aplicação, tem de se registar no FÉNIX (ilustrado na figura 1 através da seta de autenticação). O administrador é uma aplicação **Python** que comunica com o servidor por **REST**. Este atualiza os edifícios, ou vê todos os movimentos. O administrador tem de se autenticar por uma aplicação local.

No servidor é armazenado numa base de dados todos os *Logs*, todos os edifícios, assim como todos os utilizadores autenticados e as mensagens trocadas entre eles. Periodicamente o servidor recebe as coordenadas de cada utilizador e atualiza a sua base de dados com as novas indicações. O servidor responde a todos os pedidos do utilizador, bem como do administrador e Bots.

Os Bots são uma aplicação **Python** que estão associados a um certo edifício e que têm de fazer autenticação fornecida pelo administrador, para conseguirem enviar mensagens periódicas para os utilizadores que se encontram no respetivo edifício. A autenticação de um Bot é semelhante à do administrador.

### 3 Autenticação

Para uma pessoa se autenticar tem de ser aluno ou docente do IST, pelo que se utiliza o sistema de autenticação Técnico ID, baseado em OAuth 2.0. A autenticação do administrador é local à aplicação.

#### 3.1 Autenticação do utilizador

A figura 2 mostra o mecanismo de autenticação utilizado para os utilizadores.

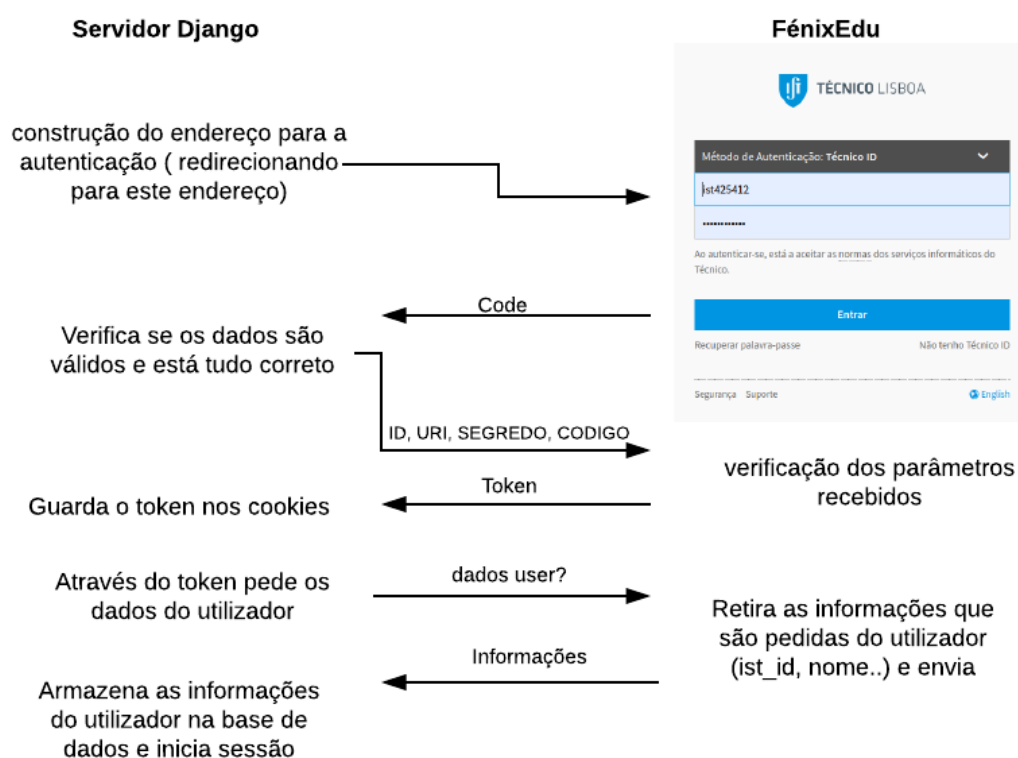


Figura 2: Esquema da Autenticação

Para se utilizar o sistema de autenticação tem de se registar a aplicação no Fénix. Neste registo, tem de se definir os parâmetros que a aplicação tem acesso (por exemplo informações do aluno), assim como o *redirect URI* que é o endereço para o qual o Técnico ID retorna o código de autenticação (para se obter os parâmetros informativos do utilizador, como nome e número de aluno).

Primeiro é gerado um endereço de autenticação do Técnico ID, através do ID da aplicação bem como o *redirect URI* da aplicação. O utilizador é redirecionado para este endereço e em seguida coloca os seus dados. Após a autenticação ser bem sucedida, é retornado do Técnico ID um código. A aplicação confirma que a autenticação foi bem executada e faz um pedido (enviando o ID, o segredo, o endereço de retorno e o tipo de autorização) para obter os *tokens* de autenticação e renovação. Se estiver tudo correto,

o Técnico ID retorna os tokens, que é guardado na *cookie*. De esta forma, se se estiver a aceder à aplicação através da mesma conta por dispositivos diferentes, consegue-se garantir que são utilizados dois *tokens* diferentes, não havendo assim conflito entre as duas diferentes sessões. A aplicação, por fim, faz um pedido REST ao Técnico ID para obter as informações do utilizador e inicia sessão.

### 3.2 Autenticação do administrador e Bot

O processo de autenticação do administrador é bastante simples, consiste em verificar os dados de *username* e de *password* enviados pela aplicação. Caso estes parâmetros estejam corretos é então gerado aleatoriamente um segredo (onde se garante que o mesmo nunca é igual a um já existente) que é retornado para aplicação para que a mesma sempre que faça um pedido, envie este segredo de forma confirmar a correta autenticação. Posteriormente, do lado do servidor, este segredo é guardado na *cache* para sempre que algum pedido é feito verificar se o segredo enviado já existe e se está correto. De notar que quando o segredo é colocado na cache, é colocado com um *timeout* para que o segredo não fique sempre na cache, não permitindo assim ações depois de algum período de tempo sem efetuar a autenticação.

Relativamente aos *Bots*, quando um administrador regista um bot num determinado edifício é atribuída uma *password* ao *Bot* que está a ser criado. Desta forma, quando alguém quiser correr um *Bot* para um determinado edifício (por exemplo, um funcionário do IST) terá que ter a *password* dada pelo administrador. Assim sendo, para correr um *Bot* não é necessário ser administrador mas é preciso ter a autorização do mesmo para ter acesso as *passwords* dos *Bots*.

## 4 REST API

Elaborou-se uma api para permitir a utilização da base de dados desenvolvida em novas aplicações que outros utilizadores possam querer realizar. Esta api baseia-se no estilo arquitetural REST. Os *endpoints* começados por */app* são para os utilizadores regulares (para a aplicação em si), os começados em */admin* para as operações do administrador, e só o administrador tem acesso. Por último os começados em */bot* destinam-se aos Bots. Listam-se de seguida os *endpoints* relevantes utilizados:

#### UTILIZADOR

- **endpoint:** */app/range*, **Método** POST - Permite o utilizador definir o seu *range*. Envia um *json* com o *range*.
- **endpoint:** */app/message*, **Método** POST - Permite o utilizador enviar mensagens. Envia um *json* com o conteúdo da mensagem, e o tipo de mensagem.
- **endpoint:** */app/users/range*, **Método** GET - Permite ao utilizador obter todos os utilizadores que estão no seu *range*. É retornado um *json* com os identificadores de cada utilizador que cumpre a condição.
- **endpoint:** */app/users/building*, **Método** GET - Permite ao utilizador obter todos os utilizadores que estão no seu edifício. É retornado um *json* com os identificadores dos utilizadores.

- **endpoint:** /app/building, **Método** GET - Permite ao utilizador saber o seu edifício. Retornado um *json* com o identificador e nome do edifício.
- **endpoint:** /app/location, **Método** POST - Permite ao utilizador alterar as suas coordenadas (latitude e longitude). Envia um *json* com a latitude e longitude.

ADMINISTRADOR (os métodos utilizados serão sempre POST, sendo que isto acontece devido ao facto de ser sempre necessário enviar o segredo para verificar a autenticação)

- **endpoint:** /admin/buildings, **Método** POST - Permite ao administrador definir os edifícios e as suas localizações (latitude, longitude).
- **endpoint:** /admin/users, **Método** POST - Permite ao administrador listar todos os utilizadores que estão com *login* efetuado no sistema.
- **endpoint:** /admin/building/users, **Método** POST - Permite ao administrador listar todos os utilizadores que estão dentro de um determinado edifício.
- **endpoint's:** /admin/logs/movements/user e /admin/logs/movements/building | /admin/logs/messages/user e /admin/logs/messages/building, **Método** POST - Permite ao administrador listar o histórico de todos os movimentos ou mensagens trocadas, fazendo a seleção por utilizador ou edifício.
- **endpoint:** /admin/bots, **Método** POST - Permite ao administrador registar um bot para um determinado edifício.

## BOTS

- **endpoint:** /bots, **Método** POST - Permite enviar mensagens periódicas para os utilizadores num determinado edifício.

## 5 Base de dados

Uma vez necessário que toda a informação indispensável para as funcionalidades do programa fosse guardada de forma sólida e segura, tornou-se óbvio que os dados teriam de ser armazenados em memória persistente (em disco). Para esta causa fez-se uso de uma base de dados baseada na linguagem SQL.

### 5.1 Sistema de gerenciamento da base de dados relacional (RDMS)

O sistema de administração utilizado para interagir com a base de dados foi o MySQL. Inicialmente, apenas para efeitos locais, começou por utilizar-se o sistema de administração SQLite, no entanto, quando chegou a altura de escalar o programa (para uma cloud, como será explicado mais à frente), houve a necessidade de correr a uma base de dados alojada num sistema operativo focado para a mesma, método que não é suportado pelo SQLite. Já o MySQL, não só suporta este método, como facilmente se conseguiu pô-lo a correr numa máquina Linux. O MySQL tem apenas a nuance de não permitir um número significativo de funcionalidades da linguagem SQL, no entanto, abrange todos os procedimentos que foram necessários para o programa interagir com a base de

dados, pelo que este tópico não foi problemático. Há ainda que referir que usando o SQLite ter-se-iam perdido várias capacidades como consistência imediata, métodos de replicação, etc.

## 5.2 Arquitetura da base de dados

Vai-se agora de encontro ao desenho da base de dados em si, isto é, aos dados que armazena, como os armazena, e os seus significados.

A nível da decisão das tabelas a definir na base de dados, criou-se uma para cada tipo de "utilizadores" do sistema - utilizadores e Bots -, uma tabela para definir os edifícios existentes no programa, e tabelas para descrever os logs - mensagens e movimentos.

Nas próximas subsecções vão ser descritos cada atributo de cada tabela, para que se entenda os objetivos das mesmas. Não obstante, para uma ideia mais visual da base de dados, está expresso na Figura 3, uma imagem que expressa o modelo da mesma.

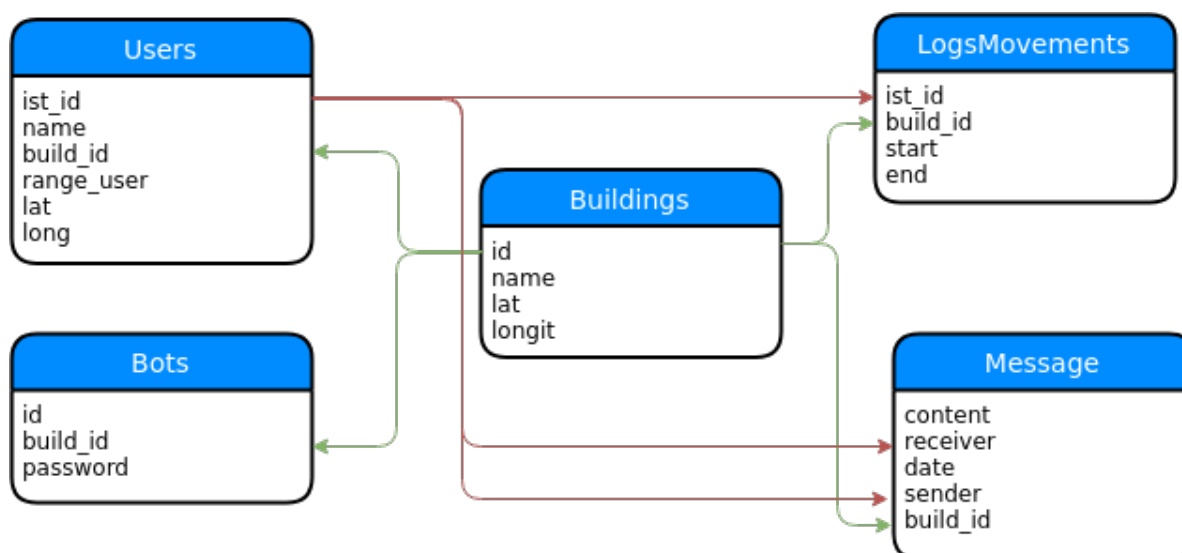


Figura 3: —

### 5.2.1 Utilizadores

Aberga toda a informação intrínseca a cada utilizador. O nome desta tabela no código é "Users", e os seus atributos são:

- `ist_id` - identificador descritivo do utilizador (*username* no *fenix*);
- `name` - nome do utilizador;
- `range_user` - raio em metros para o qual o utilizador pretende mandar mensagens;
- `lat` - última latitude recebida do utilizador;
- `long` - última longitude recebida do utilizador.



### 5.2.2 Bots

Aberga a informação necessária para aceder ao bot, e o edifício onde irá atuar. O nome desta tabela no código é "Bots", e os seus atributos são:

- id - identificador descritivo do bot (para efeitos credenciais para aceder-lhe);
- build\_id - identificador descritivo do edifício onde irá atuar;
- password - password necessária para aceder ao bot sujeito.

### 5.2.3 Edifícios

Aberga a informação geográfica e a necessária à identificação/descrição de cada edifício. O nome desta tabela no código é "Buildings", e os seus atributos são:

- id - identificador descritivo do edifício;
- name - Nome do edifício;
- lat - latitude do centro do edifício;
- long - longitude do centro do edifício.

### 5.2.4 Movimentos

Aberga a informação de cada transição entre edifício por parte de cada utilizador. O nome desta tabela no código é "LogsMovements", e os seus atributos são:

- ist\_id - identificador descritivo do utilizador sujeito;
- build\_id - identificador descritivo do edifício sujeito;
- start - data de entrada no edifício;
- end - data de saída do edifício.

### 5.2.5 Mensagens

O nome desta tabela no código é "Messages", e os seus atributos são:

- content - conteúdo da mensagem;
- receiver - identificador descritivo do utilizador que enviou a mensagem;
- sender - identificador descritivo do utilizador que recebeu a mensagem;
- build\_id - identificador descritivo do edifício onde a mensagem foi trocada (se é que foi trocada num edifício).

## 6 Funcionalidades

- **Cookie:** O *Token* utilizado por cada utilizador numa sessão é guardado numa *Cookie* e circula entre o utilizador e a aplicação por HTTP. Como já referido, se a mesma conta estiver a ser utilizada em dois dispositivos diferentes, serão gerados dois tokens (um para cada sessão). Garante-se assim que os dados do utilizador não circulem entre a comunicação entre os dois componentes, circulando apenas o token. No entanto, foi definido um tempo ao fim do qual o token expira (sai da *cookie*). Ao fim deste tempo, tem de se voltar a fazer a autenticação.
- **Cache:** Na cache é guardada a informação referente ao utilizador: nome e *ist\_id*. A chave utilizada para obter estes dados é o *Token* passado por HTTP. Realça-se que passado 1 hora os dados guardados em cache expiram, tendo que o utilizador voltar a iniciar sessão.  
Na cache também é guardado o segredo gerado durante o processo de login do administrador, sendo que o mesmo também irá expirar passado um curto período de tempo, pelo que o administrador terá que voltar a fazer a autenticação.

Assim sendo o mecanismo geral utilizado quando é realizado um pedido REST por parte do utilizador é: A aplicação obtém o *Token*, que é passado por HTTP no pedido efetuado pelo utilizador. Em seguida a aplicação vai à cache (através da chave *token* e obtém os dados nela guardados: identificador do utilizador e, se for necessário (dependendo da ação), o seu nome. Depois de saber qual o identificador do utilizador que fez o pedido, consegue satisfazer o seu pedido (indo por exemplo à base de dados filtrando pelo seu identificador).

- **Troca de mensagens:** Uma das principais funcionalidades que esta plataforma pretende oferecer é a de troca de mensagens em tempo real enviadas pelo administrador para qualquer utilizador ou para todos os utilizadores que se encontrem dentro de uma sala. Para o **envio de mensagens** por parte do utilizador (tanto para quem está no seu edifício, ou para quem está dentro do seu *range*, criou-se um *form* com o corpo da mensagem e a opção de submissão. Ao se submeter o *form* é efetuado um POST para a o *endpoint* `"/app/message"`. Neste *endpoint* a função associada está pronta para a criação de mensagens. É criada uma entrada por cada mensagem na tabela das mensagens.  
No que toca a **receber mensagens** em tempo real utilizou-se uma função em JavaScript a correr com um intervalo de tempo na ordem dos milissegundos entre cada chamada à função. Esta função utiliza *Ajax*, para fazer pedidos GET ao servidor. Nestes pedidos são retornadas as novas mensagens pertencentes ao utilizador. Para a função da parte da aplicação retornar as novas mensagens, o que faz é, através do conteúdo passado nas *Cookies* obter o *Token* e com isso aceder à cache para obter o identificador do utilizador. Após saber o identificador do utilizador, procura na base de dados de Mensagens aquelas que têm como destinatário o utilizador em questão.
- **Atualização das coordenadas e do edifício:** Através de uma função do JavaScript, a correr com um intervalo de tempo entre cada chamada à função, são obtidas as coordenadas (latitude e longitude) do utilizador e desde logo

representadas na sua interface, como se vai observar em seguida. Esta função também utiliza **Ajax** que realiza **POSTs** para o endpoint `"/app/updateLocation/"`. Mais uma vez, este *endpoint* através do conteúdo passado nas **Cookies** obtém o *Token* e com isso acede à cache para obter o identificador do utilizador. Sabendo o identificador, edita as entradas da tabela deste utilizador referentes à sua latitude, longitude e o edifício onde este se encontra. Para saber o edifício onde este se encontra, é efetuada uma conta, por comparação das suas coordenadas com as dos edifícios armazenados na base de dados.

- **Administador/Bots:** O administrador acede ao sistema através de uma aplicação no seu próprio computador (uma aplicação **Python**) que interage com o servidor usando uma API muito bem definida. Basicamente, o administrador tem acesso as atividades dos logs (movimentos e mensagens) e gere toda informação estática (nomeadamente, os edifícios que existem). Na sua interface tem as várias funcionalidades descritas anteriormente (secção 4), acedendo as mesmas por pedidos **REST** e recebendo respostas **HTTP**.

Relativamente aos *Bots*, estes são programas locais (aplicação **Python** também) que enviam mensagens para os utilizadores num determinado edifício. Este edifício, é especificado no momento em que o administrador regista um *Bot* para um determinado edifício, garantido que todas as mensagens enviados por um *Bot* em específico serão apenas entregues aos utilizadores no edifício correspondente a esse *Bot*. Estes programas, numa instituição como o IST, poderiam ser utilizados por um funcionário, sendo que o mesmo escolheria um *Bot* para o edifício que pretende (teria que ter a *password* do mesmo para poder ter acesso) e quando corresse a aplicação iria definir qual o conteúdo da mensagem, o número de mensagens que pretende enviar e qual a periodicidade das mesmas.

## 7 Interfaces

### 7.1 Utilizadores

A página inicial do utilizador representa-se em 4. Ao se carregar em *Login*, é redire-

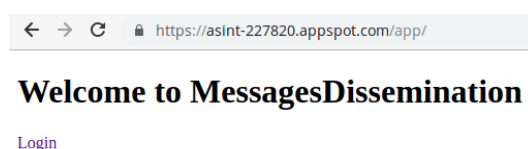


Figura 4: Página Inicial

cionado para o sistema de autenticação já referenciado e após a autenticação para a interface do utilizador que se ilustra na figura 5.

## Welcome to Messages Dissemination

IST ID: ist425412

Name: Leandro José Pereira de Almeida

Building: Torre Norte

My current coordinates

Latitude: 38.8187119

Longitude: -9.103044599999999

Only to Check exchanges

latitude  longitude

Range

Who is nearby me on: [myBuilding](#) [myRange](#)

Insert a Message to users inside my Range:

Insert a Message to users in my Building:

Figura 5: Interface do utilizador

Primeiramente pode-se observar que os dados recolhidos pela aplicação ao Técnico ID são representados na interface do utilizador (são parâmetros estáticos), assim como o edifício onde este se encontra, que pode estar sempre a ser atualizado. Se o utilizador carregar em pesquisar as pessoas que estão no seu edifício, aparecem todos os identificadores dos utilizadores que se encontram no seu edifício, e se pressionar por *range* aparecem todos os que se encontram dentro do *range* por si definido. Existe a opção do utilizador alterar o seu *range*.

De modo a se conseguir testar os movimentos de um utilizador, adicionou-se um campo para colocar "à mão" a latitude e a longitude. Desta forma, estando por exemplo na Torre Norte, pode colocar as coordenadas da Torre Sul, observando-se assim da parte do administrador que este utilizador se movimentou da torre Norte para a Torre Sul. Este facto não tem qualquer utilidade prática, sendo apenas apresentado para efeitos de teste.

O utilizador pode inserir uma mensagem para todas as pessoas que se encontram no seu edifício ou no seu *range*. Em ambos os casos, é feito um **POST** para a aplicação com o conteúdo da mensagem. Este **POST** é realizado pelo **Ajax** para um *endpoint* dedicado mesmo para estes pedidos, que, de seguida, atualiza a base de dados de mensagens dos respetivos destinatários.

As mensagens recebidas pelo utilizador também são visualizadas na interface, como representado em 6.

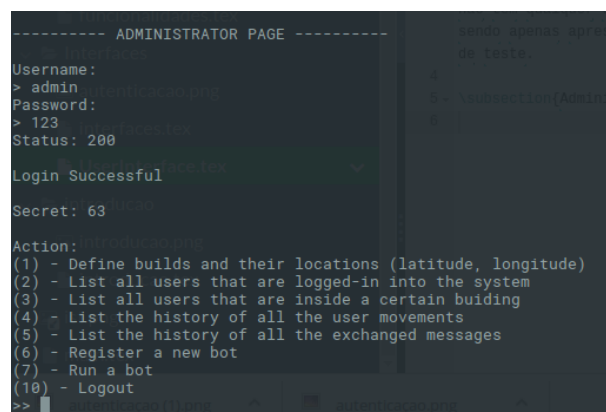
## 7.2 Administrador

O utilizador corre através de uma aplicação local realizada em **Python**. Para correr o administrador tem apenas de se executar `$Python3 administrator.py`. Inicialmente é pedido desde logo o *username* e *password* e depois de o Login ser bem conseguido, é apresentado um menu com as diversas funcionalidades que o administrador pode

Received messages		
Date	Sender Id	Content
2019-01-28T14:31:06.286Z	ist425412	Teste para mesmo building
2019-01-28T14:30:56.027Z	ist425412	Teste para range
2019-01-28T14:30:04.871Z	BOT : 34	oi oi oi oi oi
2019-01-28T14:30:02.849Z	BOT : 34	oi oi oi oi oi
2019-01-28T14:30:00.823Z	BOT : 34	oi oi oi oi oi

Figura 6: Mensagens Recebidas

executar. Para efetuar uma delas é só carregar na respetiva tecla. A figura 7 representa a interface do administrador.



```

----- ADMINISTRATOR PAGE -----
Interface:
Username:
> admin
Password:
> 123
Status: 200
Login Successful
Secret: 63
Action:
(1) - Define builds and their locations (latitude, longitude)
(2) - List all users that are logged-in into the system
(3) - List all users that are inside a certain building
(4) - List the history of all the user movements
(5) - List the history of all the exchanged messages
(6) - Register a new bot
(7) - Run a bot
(10) - Logout
  
```

Figura 7: Interface do administrador

### 7.3 Bot

Na figura 8 mostra-se como se regista um Bot no administrador, sendo que lhe é atribuída desde logo uma *password* e um identificador a utilizar na interface do Bot.

A figura 9 ilustra a interface utilizada para o Bot, sendo que este escolhe a mensagem a enviar e a sua prioridade, bem como o edifício associado. Realça-se que para ter acesso, necessita de colocar a *password* especificada em 8.

```
>> 6
Building ID:
> 2448131361091
Status: 200

Bot ID: 51
Building ID: 2448131361091
Password: wcygjmes
```

Figura 8: Administrador a registar um Bot, atribuindo assim um ID e uma password para esse bot

```
Bot ID:
> 51
Building ID:
> 2448131361091
Password:
> wcygjmes
Message:
> olá a todos
Number:
> 10
Periodicity:
> 100
```

Figura 9: Interface do Bot

## 8 Implementação do sistema na Google Cloud Platform

Depois do desenvolvimento parcial (já quase total) da aplicação, esta foi implantada na "Google Cloud", num dos serviços que esta oferece - "Google App Engine flexible environment".

Começou então por aceder-se à "Google Cloud Platform", a partir de onde se podem administrar todos os recursos alojados na cloud, e instanciar uma máquina virtual com o sistema operativo Linux (distribuição Debian). É de notar que neste passo se escolheu o serviço gratuito, pelo que grandes necessidades de memória ou processamento não serão satisfeitas com esta escolha, no entanto, há sempre a possibilidade de assinar pelos serviços pagos, não sendo necessário realojar nem reiniciar todos os componentes já instalados e funcionais.

De seguida, antes de correr a aplicação remotamente, foi necessário lançar um servidor para a base de dados utilizada. Uma vez que numa fase inicial (apenas local), se estava a utilizar a base de dados SQLite, que não suporta ser corrida num servidor com o propósito de manusear uma base e dados, alterou-se para MySQL. Criou-se então uma instância DBMS do tipo MySQL (Google Cloud SQL instance), e de seguida, não só para facilidade de desenvolvimento como para facilidade de testes e debug gerou-se um proxy para que fosse possível, através deste, correr a aplicação localmente, mas usando a base de dados remota instanciada na cloud da google.

Agora, na aplicação em si, é necessário alterar o programa, de modo a que este direcione a informação que pretende guardar de forma persistente na base de dados instanciada. Uma vez que esta base de dados é um servidor a correr de forma independente, acede-se a esta tal como a outra máquina qualquer, através de um IP e de um porto. Será também necessário, para um acesso válido, indicar o nome da base de dados e as credenciais de acesso. Todas estas alterações à aplicação são aplicadas no ficheiro

settings.py que inclui toda a informação de como e onde correr todas as vertentes do programa.

Depois de já ter a base de dados alojada e com todas as funcionalidades desejadas (todos os testes à base de dados podem ser feitos localmente através do proxy), é ainda necessário escrever/gerar um ficheiro requirements.txt com a informação de quais os pacotes e dependências que serão necessários instalar na máquina virtual que está a correr na cloud. Uma vez que se trata também de um computador a funcionar de forma independente, para correr a aplicação que lhe será implantada, precisa de saber quais os pacotes que são necessários, tal como foi feito no desenvolvimento local, isto é, todos os "pip install ...", muniram a máquina local de certas ferramentas necessárias para correr o programa. A máquina alojada na cloud que também irá correr o mesmo programa, precisa também de possuir estas ferramentas. Por isso, quando uma aplicação é implantada, antes de lançá-la, a instância certifica-se de que todos os pacotes expressos no ficheiro requirements.txt estão instalados e ativos.

Agora sim, está-se em condições de implantar a aplicação no serviço da cloud instanciado. O ficheiro que ordena a implantação (deploy), é o ficheiro app.yaml. É necessário indicar neste qual será o interpretador (e a sua versão) a ser utilizado, neste caso é a versão mais recente de python. Inclui-se então neste ficheiro a linha "runtime: python37", e mais uns promenores não tão significativos.

Para fazer a implantação basta, com todo este background já preparado, correr o seguinte comando no terminal:

```
$ gcloud app deploy
```

A implantação leva alguns minutos, visto que se está a reiniciar um servidor e não apenas a correr um programa. Por esta razão torna-se bastante útil poder recorrer à base de dados através de um proxy pois fica-se com a liberdade de correr o programa localmente, que para correr desta forma é quase instantâneo. Uma vez finalizada a implantação, vê-se no terminal qual o link de acesso à aplicação por um browser, após o qual o processo lançado pelo comando `$ gcloud app deploy` se vê concluído.

No caso deste projeto, para aceder à aplicação, basta aceder ao link:

<https://asint-227820.appspot.com/app/>

Todas as informações de como implantar a aplicação, corrê-la localmente, e até ativar o proxy para a utilização da instância da base de dados numa tentativa local, encontram-se escritas no ficheiro "readme".

## 9 Discussão

Nesta secção serão abordadas as decisões feitas no que toca à implementação e desenvolvimento da aplicação e explicadas as vantagens vistas nestas decisões e nas tecnologias utilizadas.

Começando pelo ponto de vista mais superficial, a aplicação deste projeto foi concebida recorrendo ao forte uso de bibliotecas e *frameworks* programáticas pré-concebidas e já de muito alto nível. Optou-se por este método de desenvolvimento, pois não só foi possível poupar bastante tempo, uma vez que grande parte do trabalho já "vem feito" por

defeito e não é necessário estar a programar "à mão" todos os detalhes do esqueleto da aplicação, como para além disso há uma garantia de que não há erros na estrutura do programa, visto que se está a usar código já testado, aprovado e implementado por milhares de *developers* no mundo. Para além disto, há ainda uma maior garantia na segurança da aplicação, visto que toda a estrutura é bem definida e bastante sólida.

A aplicação desenvolvida, trabalhou todos os *endpoints* através do protocolo REST, que permitiu para este projeto duas funcionalidades bastante importantes e necessárias - não só permite que uma aplicação externa aceda a todos os métodos disponibilizados por uma interface web, como ainda permite que a troca de informação seja feita através de um tipo de dados que pode ser interpretado por todas as linguagens de programação utilizadas (e muitas mais).

A nível de acessos à aplicação, como já foi explicado anteriormente, foram criadas duas formas distintas de o fazer, onde no modo cliente/utilizador, se pôde fazer uso do sistema de autenticação disponibilizado pelo IST, sendo que desta forma se consegue requerir uma autenticação alheia sem que seja necessário o acesso a dados privilegiados, pois estar-se-ia nesse caso a proliferar vulnerabilidades de segurança. Para a autenticação do servidor, fez-se uso do sistema de autenticação disponibilizado pelo Django, onde aqui, mais uma vez, se verificam todas as vantagens enunciadas no segundo parágrafo desta secção.

Finalmente, o *deployment* da aplicação na google cloud, através do serviço disponibilizado pela Google, trouxe inúmeras vantagens e capacidades a esta, das quais se destacam as 3 que foram vistas como mais relevantes:

- Fiabilidade - alojar a aplicação numa cloud oferece maior fiabilidade do que uma estrutura "caseira".
- Escalabilidade - Alojando a aplicação numa cloud, pode escalar-se o sistema "indefinidamente", de modo a poder sempre satisfazer-se toda a procura sem que seja necessário reestruturar ou reiniciar os servidores nem as bases de dados.
- Flexibilidade - Alojando a aplicação numa cloud, há a possibilidade de fazer *upgrades* à aplicação e à base de dados sem que para isso seja necessário aos utilizadores fazerem o *download* ou instalação de pacotes desnecessariamente.