Instituto Superior Técnico

Aplicações Distribuídas sobre a Internet

Projeto 2ªParte:

Sistema de Controlo de Acessos

Autores

Nome: Luís Coelho **Número:** 90127
Nome: Tiago Dias **Número:** 90196

Grupo 11
Prof. João Nuno De Oliveira e Silva

2021/2022 - 1º Período
15/11/2021

Introdução e Objetivos

Nste relatório será feita a descrição do sistema de controlo de acesso implementado na 2ª parte do projeto.

Este sistema tem como objetivo o da criação de um mecanismo de cancelas, em que um utilizador consegue a partir de um browser, por exemplo no smartphone, gerar uma chave (QRCode) em tempo real de modo a aceder à cancela. O utilizador poderá também consultar o seu histórico de acessos na aplicação.

Para implementar este projeto foram usadas diversos protocolos e linguagens programação, tendo sindo desenvolvidas 3 aplicações em web e 2 sistemas de bases de dados, todos eles controlados por um serviço central.

As 3 aplicações web seriam uma para os utilizadores da aplicação, outra para os administradores, que podem por exemplo registar novas cancelas ou ver o histórico dos acessos totais, e outra para as próprias cancelas onde, através de uma câmara, se poderiam ler e processar os QRCodes dos utilizadores.

Existiriam também 2 sistemas de bases de dados, um para utilizadores, que guardaria identificações e códigos de acesso, e outro para as cancelas, que guardaria informações relacionadas com a identificação, localização e segredos associados.

Para identificar os utilizadores deste sistema, e poder também fazer a divisão dos papéis dos utilizadores, entre administradores ou utilizadores simples, será usado o sistema de autenticação do Fenix.

1 Arquitetura

A arquitetura adotada para esta segunda parte do projeto do Sistema de Controlo de Acessos foi a seguinte:

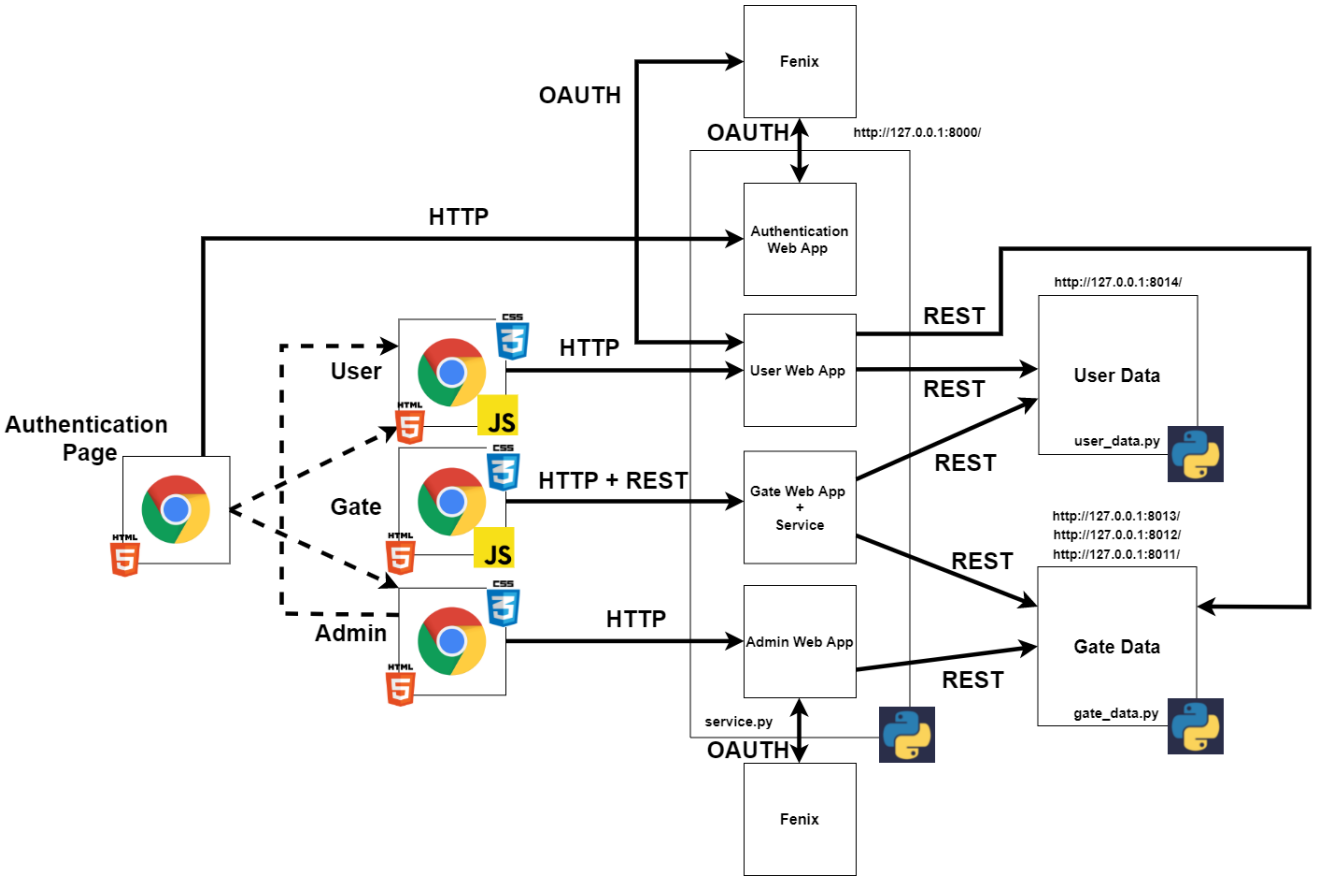

Figura 1: Estrutura do Sistema de Controlo de Acessos desenvolvido.

Na arquitetura podem ser visualizados todos os processos, browsers e aplicações usados para a implementação deste sistema, assim como as componentes de código que correm em cada nó.

Foram tomadas algumas decisões face à arquitetura presente no enunciado deste projeto, sendo a principal delas a adição de uma nova página web (**Authentication Page**) que tem como propósito a autenticação dos utilizadores, usando o serviço do Fénix. Esta página comunica por HTTP com o ficheiro **service.py**, mais concretamente com a secção do **Authentication Web App**, baseado no ficheiro **Fenixauth.py** fornecido como apoio ao projeto, que possui toda a lógica por detrás do processo de autenticação do utilizador (obtenção do url de autenticação, obtenção do token, etc.).

Na arquitectura podem ser visualizadas dois tipos de setas de fluxo: com preenchimento sólido ou a tracejado. As setas preenchidas correspondem a ligações entre nós através de um certo protocolo, podendo este ser HTTP, REST ou OAUTH. Por outro lado, as setas a tracejado correspondem ao fluxo das páginas web.

Quando um utilizador se conecta ao serviço, este é redirecionado para a **Authentication Page**, onde efetua o processo de autenticação. Deste processo de autenticação, um dos dados retirados é o id do Fenix do utilizador. Se este id constar numa lista com todos os números de identificação dos administradores, então isto significa que o utilizador é um administrador e, como tal, é redireciondao para a **Admin Web App**. Por outro lado, se não for administrador, o utilizador é reencaminhado para o **User Web App**. Para que um administrador consiga também aceder a

gates, optou-se por dar a opção a um administrador de aceder ao **User Web App**, através da **Admin Web App**.

Na arquitetura podem também ser visualizados os endereços em que corre cada nó. O ficheiro **service.py** correrá no endereço **https://127.0.0.1:8000/**, o ficheiro **user_data.py** correrá no endereço **https://127.0.0.1:8014/** e o **gate_data.py** poderá correr em 3 endereços diferentes: **https://127.0.0.1:8013/**, **https://127.0.0.1:8012/** e **https://127.0.0.1:8011/**.

Por fim, pode-se também verificar que existe uma ligação REST entre o **User Web App** e o **Gate Data**, algo que não constava na arquitectura proposta no enunciado. Como se poderá ver na secção dos **Modelos de Dados**, esta ligação resultou do facto de somente se ter implementado uma tabela dedicada ao armazenamento do historial de acesso às gates, tabela essa que ficou guardada na **Gate Data**.

2 Modelos de Dados

No ficheiro **gate_data.py** e no ficheiro **user_data.py**, estão presentes as bases de dados que serviram de modelo de dados para este projeto. A tabela **User** tem como objetivo guardar as informações dos utilizadores, a tabela **Gate**, guarda os dados das gates presentes na rede e a tabela **GateAccessLog** o registo de tentativas de acesso às gates por parte dos utilizadores.

2.1 User

Para o **User** são guardados 3 atributos: um número de identificação do utilizador (**userID**), o código associado a um utilizador (**userCode**), com 1 minuto de validade, e a data de criação do código (**codeTime**), que servirá para aferir se um userCode gerado expirou, ou ainda pode ser utilizado. A estrutura da base de dados **User** pode ser encontrada na Tabela 1.

Tabela 1: Estrutura da Tabela **User**

Table Name	**User**		
Column Names	userID	userCode	codeTime
Data Type	String	String	DateTime
Primary Key	True	False	False
Nullable	False	False	False
Unique	False	True	False

O userID é o ID dado ao utilizador depois de se autenticar pelo sistema FÉNIX. Este id terá o formato "ist1XXXXX".

2.2 Gate

A tabela **Gate**, relativa às gates, tem quatro atributos associados: o número de identificação da gate (**gateID**), a localização da gate (**gateLocation**), o código secreto associado a uma gate (**Gateecret**) e o número de acessos a uma dada gate (**gateOpens**). Esta estrutura pode ser vista na Tabela 2.

Tabela 2: Estrutura da Tabela **Gate**

Table Name	**Gate**			
Column Names	gateID	Gateecret	gateLocation	gateOpens
Data Type	Integer	String	DateTime	Integer
Primary Key	True	False	False	False
Nullable	False	False	False	False
Unique	True	False	False	False

2.3 Logs

A tabela **GateAccessLog**, relativa aos registos dos acessos às gates, está presente no ficheiro **gate_data.py** e tem quatro atributos associados: o número de identificação da gate (**gateID**), a identificação do user (**userID**), o tempo de quando a gate foi acessada (**accessTime**) e o estado do acesso a essa gate (**successfulAccess**), que poderá ser "Access Granted", no caso de sucesso, ou "Access Denied", no caso de insucesso. Esta estrutura pode ser visualizada na Tabela 3.

Tabela 3: Estrutura da Tabela **GateAccessLog**

Table Name	**GateAccessLog**			
Column Names	gateID	userID	accessTime	successfulAccess
Data Type	Integer	String	DateTime	String
Primary Key	True	False	True	False
Nullable	False	True	False	False
Unique	False	False	False	False

Sempre que se tenta aceder a uma gate, é feito o registo da tentativa. No caso de um acesso ter sido efetuado com sucesso, o id do utilizador é guarddao. Caso um acesso tenha falhado, então não é guardado o id do utilizador, sendo colocado -1"na coluna **userID**.

3 Endpoints REST API e Web

Nesta secção irão ser listados os endpoints de cada nó presente na arquitetura do projeto, sendo referidos os métodos associados, inputs, outputs e códigos de erro. Também será dada uma breve descrição do funcionamento de cada um dos endpoints.

3.1 Interface Web do Authentication Web App

- / : este endpoint reencaminha o utilizador para o processo de autenticação, sendo obtidos tanto o url de autorização como o estado da ligação. Em caso de sucesso na obtenção destes termos, o utilizador é redirecionado para o url de autenticação obtido, caso contrário, é redirecionado para uma página web (**index_error.html**) com uma mensagem de erro associada. Nesta página de erro o utilizador pode tentar autenticar-se de novo no Fenix. Este endpoint não aceita qualquer tipo de inputs;

- **/callback** : este endpoint (correspondente ao redirect_uri da aplicação) é para onde o utilizador é redirecionado após efetuar o log-in no Fenix. Neste endpoint, é guardado o token do utilizador associado à sessão. Caso esta operação seja bem sucedida, o utilizador é, de seguida, redirecionado para outro endpoint (**/profile**). No entanto, caso não seja possível obter um token, o utilizador é de novo redirecionado para uma página web (**index_error.html**), com uma mensagem de erro associada, onde poderá tentar autenticar-se de novo. Este endpoint não possui qualquer tipo de inputs;
- **/profile** : este endpoint, quando é acedido, começa por retirar da sessão OAUTH o username do utilizador, acedendo ao endereço ('https://fenix.tecnico.ulisboa.pt/api/fenix/v1/person') da API do Fenix. Depois de ter o username do utilizador, este é comparado com os usernames que constam numa lista com os usernames de todos os administradores. Se o utilizador constar nessa lista, então é redirecionado para a página principal (**index_admin.html**) do **Admin Web App**, caso contrário, é redirecionado para a página principal (**index_user.html**) do **User Web App**. Em caso de erro na obtenção do username, o utilizador é redirecionado para a página web **index_error.html** juntamente com uma mensagem de erro associada, onde o utilizador pode tentar repetir o processo de autenticação na aplicação. Este endpoint não possui quaisquer inputs;
- **/error** : este endpoint é acedido a partir da página web **index_error.html**, clicando-se na opção "Try Again". O objetivo deste endpoint é tentar re-autenticar o utilizador após um processo de autenticação anterior ter falhado. Para tal, é efetuada uma tentativa da obtenção do estado da ligação e do url de autenticação. Se esta operação tenha for feita com sucesso, o utilizador é redirecionado para o url de autenticação, caso contrário, o utilizador é reencaminhado para a página web **index_error.html**, juntamente com uma mensagem de erro. Este endpoint não possui inputs;

3.2 Interface Web do User Web App

- **/user_web_app**: este endpoint não recebe nenhum input. Quando acedido, o endpoint renderiza a página principal do **User Web App** (**index_user.html**) no browser do utilizador;
- **/qrcode**: este endpoint também não recebe nenhum input. Quando é acedido, o endpoint recolhe o user-id (username) do utilizador a partir do token de sessão do utilizador. De seguida, o endpoint envia um pedido ao **User Data** pelo user-code do utilizador em questão, de modo a gerar uma string ("user-id user-code") que será transformada num QRCode. Se esta operação de geração da string for bem sucedida, o endpoint irá renderizar a página **qrcode_user.html** juntamente com a string criada. Será nesta página que a string é convertida para QRCode, através da biblioteca externa QRious para JavaScript. Caso se verifique algum erro durante a geração da string, o endpoint irá redirecionar o utilizador para a página inicial da **User Web App** (**index_user.html**), com uma mensagem de erro;
- **/userLog**: quando é acedido, este endpoint começa por recolher o id do utilizador através do seu token de sessão. Depois, com o user-id obtido, o endpoint faz um pedido à **Gate Data**, para que esta lhe devolva a lista dos acessos feitos pelo utilizador, presentes na tabela **GateAccessLog**. Caso esta operação seja efetuada com sucesso, o endpoint retorna uma página web (**listUserLog_user.html**) para o browser do utilizador juntamente com o resultado da

query, que será colocado na página sob a forma de uma tabela. No entanto, caso se verifique algum erro durante esta operação, o endpoint redireciona o utilizador para a página inicial (**index_user.html**) da **User Web App**, com uma mensagem de erro;

3.3 Interface Web do Gate Web App

- **/gate_web_app** : este endpoint, quando acedido, devolve ao utilizador o ficheiro static (**index_gate.html**), correspondente à página inicial da **Gate Web App**. Todas as funcionalidades da **Gate Web App**, como a leitura do QRCode ou a introdução dos dados relativos a uma gate (gate id e gate secret), estão incluídas neste ficheiro. O endpoint em questão não possui qualquer tipo de inputs;

3.4 Interface Web do Admin Web App

- **/admin_web_app** : este endpoint não recebe qualquer input. Quando acedido, o endpoint renderiza a página principal do **Admin Web App** (**index_admin.html**) no browser do administrador;
- **/listGate** : quando acedido, este endpoint lista no browser do administrador todas as gates existentes na **Gate Data**. Deste modo, o output do endpoint consiste na renderização da página **listGate_admin.html** com o resultado obtido da query ou, em caso de erro na query ao **Gate Data**, da renderização da página inicial (**index_admin.html**), com uma mensagem de erro;
- **/newGate** : quando acedido, este endpoint renderiza no browser do administrador a página **newGate_admin.html**, que permite a criação de uma nova gates a ser adicionada à **Gate Data**, com id e localização à escolha do utilizador. Esta página que o endpoint gera como output também possui um botão que, quando clicado, redireciona o programa para o endpoint **/createGate**;
- **/createGate** : este endpoint, acedido a partir da página **newGate_admin.html**, permite a inserção de uma nova gate na tabela **Gate** da **Gate Data**. Este endpoint pode redirecionar o administrador para a página principal **index_admin.html** com uma mensagem apropriada, em caso de erro no acesso à base de dados ou com a introdução incorreta de pârametros (por exemplo id introduzido ser uma string de caracteres). Em caso de sucesso, o endpoint redireciona o administrador para a página **showGate_admin.html**, onde se pode visualizar o id, localização, número de acessos e segredo associados à gate criada. Caso o id da gate já se encontre na tabela **Gate**, este endpoint retorna a página **showGate_admin.html** com a informação da gate já existente;
- **/listGateLog** : quando acedido, o endpoint realiza uma query de modo a obter o conteúdo da tabela **Gate Access Logs** da **GateData**. Em caso de sucesso, o endpoint renderiza no browser do administrador a página **listGateLog_admin.html**, com o conteúdo da query, permitindo a visualização do histórico dos acessos às gates, nomeadamente do id da gate, da hora do acesso e se o acesso foi, ou não, bem sucedido. No caso da query não poder ser obtida, o endpoint redireciona o administrador para a página **index_admin.html**, com uma mensagem de erro;

3.5 Interface REST API do Serviço

- **/API/checkGateExists** (GET e POST) : este endpoint tem como objetivo verificar se uma dada gate existe na tabela **Gate**. Para o fazer, este endpoint recebe como input o id e o segredo da gate, introduzidos na inicialização da **Gate Web App**, e compára-os com as gates guardadas na tabela **Gate**. Em caso de sucesso, o endpoint retorna o estado da validação ("True") ou ("False") num json object e um código de estado 200. Caso não se tenha conseguido aceder à base de dados, então é retornada uma mensagem de erro num json object e um código de estado 404;
- **/API/checkUserCodeExists** (GET e POST) : acedido quando o utilizador passa o seu QRCode pela câmara da **Gate Web App**, este endpoint recebe como inputs o id e o código do utilizador e o id da gate associada ao scanner, de modo a processar a tentativa de acesso do utilizador à gate. São no total efetuadas 3 queries à base de dados: verifica-se se o código do utilizador presente no QRCode é válido e, caso seja, incrementa-se o número de acessos da gate associada ao scanner através do seu user-id, com outra query. Por fim, é registado na tabela **GateAccessLog**, com uma terceira query, a tentativa de acesso, bem ou mal sucedida, do utilizador de id user-id à gate dada pelo gate-id. Caso não seja possível aceder à **Gate Data** e/ou ao **User Data**, o endpoint retorna um json com uma mensagem de erro e o código de estado 404, caso contrário, retorna uma resposta em json com código de estado 200 e com o estado da validação do acesso à gate ("True") ou ("False");

3.6 Interface REST API da Gate Data

- **/API/gates** (GET): este endpoint tem o propósito de retornar uma lista com todas as gates presentes na tabela **Gate**, sob a forma de uma string json. Em caso de sucesso, esta string é retornada com o código de estado 200. Em caso de insucesso, é retornado outro elemento json com uma mensagem de erro apropriada e um código de erro 503. Este endpoint não recebe qualquer tipo de input;
- **/API/gate/<path:gate_id>** (GET): este endpoint procura selecionar uma gate específica com ID = "gate_id"na tabela **Gate**. Em caso de sucesso, o endpoint retorna a informação da gate selecionada num objeto json, com um código de estado 200. Em caso de insucesso, o endpoint retorna outro objeto json com uma mensagem de erro e um código de estado de 404. Este endpoint recebe o id da gate como input;
- **/API/gates** (POST): este endpoint insere uma nova gate na tabela **Gate**. Para tal, o endpoint recebe o id e a localização da gate como input. Caso a query de inserção seja feita com sucesso, é devolvido um objeto json com o conteúdo da gate criada e com código de estado 200. Caso contrário, é enviado outro objeto json com uma mensagem de erro e um código de estado 503;
- **/API/gates/<path:gate_id>/<path:gate_secret>** (GET) : este endpoint compara o segredo introduzido na inicialização da **Gate Web App** com o segredo guardado na tabela **Gate**, na linha cujo id = gate_id, verificando se o segredo introduzido corresponde ao segredo da gate em questão. Caso a query à linha seja efetuada com sucesso, o endpoint retorna um objeto json com um código de estado 200, indicando se os segredos são iguais ou diferentes entre si. Caso não se tenha conseguido efetuar a query, porque não se encontrou uma gate

com tal id, então é retornado outro objeto json com uma mensagem de erro e com o código de estado 404;

- **/API/gateOpens/<path:gate_id>** (PUT) : este endpoint recebe como input o id de uma dada gate. Caso a gate recebida exista na tabela **Gate**, o endpoint incrementa o número de acessos dessa gate e retorna um objeto json com uma mensagem de sucesso e um código de estado 200. Caso não exista uma gate com tal id, então é retornado outro objeto json com uma mensagem de erro e um código de estado 404;
- **/API/gateAccessLog** (GET) : este endpoint faz uma query à tabela **GateAccessLog**, de modo a devolver um objeto json com os gateID, accessTime e sucessfulAccess de todos os seus registos. Caso se consiga realizar esta query, o objeto json é retornado juntamente com um código de estado 200. Caso contrário, então é retornado outro objeto json com uma mensagem de erro e um código de estado 503;
- **/API/userAccessLog/<path:current_user>** (GET) : este endpoint recebe como input o id de um dado utilizador e faz uma query à tabela **GateAccessLog**, de modo a devolver um objeto json com os gateID, userID e accessTime de todos os seus registos, para que o utilizador consiga consultar o seu histórico de acessos na aplicação. Caso se consiga realizar esta query, o objeto json é devolvido com um código de estado 200. Caso contrário, é devolvido outro objeto json com uma mensagem de erro e um código de estado 503;
- **/API/gateAccess/<path:state>** (POST) : este endpoint recebe como input o estado de um acesso ("1"se um acesso for feito com sucesso e "0", caso contrário), assim como o id da gate e o id do utilizador. Com estes inputs, o endpoint faz uma query à tabela **GateAccessLog**, de modo a adicionar um novo gate access. Caso esta operação seja efetuada com sucesso, o endopoint retorna um json object com o acesso registado e um código de estado 200. Caso contrário, é devolvido um objeto json com uma mensagem de erro e com um código de estado 503;

3.7 Interface REST API do User Data

- **/API/users/<path:user_id>/user_code** (GET) : este endpoint recebe como input o id de um utilizador, e tem como objetivo o de gerar um código válido para este utilizador e, de seguida, registar o utilizador na tabela **User**, com este código gerado. Caso esta operação seja efetuada com sucesso, é retornado um objeto json com o user code gerado e com um código de estado 200. Caso contrário, é retornado um objeto json com uma mensagem de erro e com o código de estado 503;
- **/API/user_code/<path:user_code>** (GET) : este endpoint recebe como input o código do utilizador e verifica se este código existe na tabela **User**. Caso exista, também verifica se este código está ou não expirado. Se for possível executar estas operações, então o estado de validade do código é retornado num objeto json juntamente com um código de estado 200. Caso contrário, é retornado um objeto json com uma mensagem de erro e com um código de estado 404;

4 Funcionalidades

Nesta secção, será descrita a implementação de algumas das funcionalidades do Sistema de Controlo de Acessos. Para tal, serão listadas algumas das operações que o utilizador tem de fazer, que código é executado, por quais endpoints este passa e quais algumas das mensagens que o utilizador pode visualizar durante a interação.

4.1 Login e Autenticação no Fenix

Ao aceder ao endereço onde está a correr o Serviço (**https://127.0.0.1:8000/**), o código presente no endpoint / é executado e, como tal, o utilizador inicia o seu processo de login no Fenix. Primeiro a sessão é aberta, e são gerados um estado de sessão e um authorization_url. Se este processo for bem sucedido, o utilizador é redirecionado para uma página de autenticação do Fenix. Se o log-in efetuado no Fenix for válido, o utilizador é de seguida redirecionado para o endpoint (**/callback**), devido ao redirect_uri definido. Neste endpoint, obtém-se o token de autenticação do utilizador e redireciona-se o utilizador para o endpoint **/profile**.

No endpoint **/profile**, é retirado a partir do endereço web do API do Fenix, presente em **https://fenix.tecnico.ulisboa.pt/api/fenix/v1/person**, o username do utilizador, que servirá como o seu id no sistema. Caso este id conste na lista de user-id's de administradores (definida por nós), então o utilizador é redirecionado para a **Admin Web App**, caso contrário, o utilizador é redirecionado para a **User Web App**.

Se o utilizador nunca tiver acedido ao programa com o seu Fénix, então tem que aceitar o acesso deste às informações pedidas.

Se ocorrer um erro na autenticação, o utilizador é reencaminhado para uma janela de erro (endpoint **/error**), onde lhe será dada a opção de voltar a tentar autenticar-se e demonstrada uma mensagem de erro.

4.2 Pedido de um Utilizador por um Novo Código

Estando na página inicial do **User Web App** (ficheiro **index_user.html**), o utilizador pode clicar para gerar um novo QRCode. Ao clicar no botão "Generate New QR Code", é reencaminhado para o endpoint **/qrcode**. Este endpoint começa por enviar um pedido ao **User Data** para obter um userCode (endpoint **/API/users/<path:user_id>/user_code**).

É então do lado do **User Data** que é gerado o código. No momento da criação do código, também é guardada a informação do utilizador na base de dados (tabela **User**) em conjunto com o seu userCode. Ao todo são guardados o id do utilizador, o seu código e o tempo em que o código foi gerado, de modo a se poder futuramente determinar se este se encontra expirado. Se já existir o User na base de dados, então é feita a atualização do userCode, e também do codeTime, para ter o novo código válido durante 1 minuto.

O userCode é então recebido pelo Service e é feita a imagem do QRCode em HTML+JavaScript.

Caso se registe algum erro durante este processo, o utilizador é reencaminhado de novo para a página inicial do **User Web App**, onde poderá também ver uma mensagem de erro.

4.3 Acesso do Utilizador à Lista de Acessos

Estando na página inicial do **User Web App**, o utilizador pode clicar para aceder ao historial de acessos do User. Ao clicar no botão "User History", é reencaminhado para o endpoint **/userLog**.

Então, é feito um pedido da lista de acessos do utilizador, com o userID incluído, para se obter a lista de todos os acessos feitos apenas por aquele utilizador. O pedido é feito à tabela **GateAccessLog** do **Gate Data**, onde são guardados os registos dos acessos às gates (endpoint **/API/userAccessLog/<path:current_user>**). Este endpoint responde com um objeto json contendo a lista de todos os acessos a gates feito pelo utilizador.

Então, depois de recebida e processada, esta lista é passada para HTML sob a forma de uma tabela com a informação vinda da base de dados.

Novamente, caso se registe algum erro durante este processo, o utilizador é reencaminhado de novo para a página inicial do **User Web App**, onde poderá também ver uma mensagem de erro.

4.4 Validação do Código pela Gate

Para validar um dado userCode, é necessário aceder à **Gate Web App**, que se encontra no endereço (**https://127.0.0.1:8000/gate_web_app/**) (endpoint **gate_web_app**), e ler o QRCode gerado pelo utilizador. Para aceder à gate é necessário preencher os dados da gate pretendida (gateID e gateSecret).

Ao preencher os dados e clicar em "Log-in", do lado do browser, é feito um pedido por AJAX ao Service para verificar se as credenciais da gate estão corretas (endpoint **/API/checkGateExists**). Para o Service verificar isto, envia um pedido para a **Gate Data** (endpoint **/API/gates/<path:gate_id>/<path:gate_secret>**). Esta verificação passa por fazer uma query à tabela **Gate**, de modo a obter a linha correspondente ao gateID em questão, e a depois comparar o gateSecret da query com o gateSecret introduzido pelo utilizador. Caso os secrets correspondam, um objeto json com esta informação é retornado em cada endpoint acedido até retornar ao Javascript de onde se iniciou o pedido. Se por acaso os segredos não correspondessem, aparecia na página da **Gate Web App** uma mensagem de erro.

Considerando que o acesso à gate foi obtido, ficou-se agora com acesso à câmara (QRCode scanner). O utilizador deve apontar a câmara para o QRCode que pretende que seja descodificado. Ao fazê-lo, é feita a leitura do QRCode, e é então necessário conectar o utilizador à gate.

Para o fazer, enviam-se por AJAX os dados para o serviço (endpoint **API/checkUserCodeExists**) e verifica-se se os dados do QRCode correspondem a um utilizador existente. Para isso, é feito um pedido à **User Data** para que verifique a existência do userCode na base de dados. Este pedido é para o endpoint **API/usercode/<path:user_code>**, que verifica se o código existe na tabela **User** e, caso exista, se está expirado, comparando o tempo atual com o tempo de criação do código existente na tabela. O **User Data** devolverá depois um json object com a validade do código lido.

Ainda no serviço, depois de este receber o objeto json com a validade, se o QRCode tiver um userCode válido, então é enviado um pedido à **Gate Data**, mais concretamente para o endpoint **/API/gateOpens/<path:gate_id>** para atualizar o número de acessos feitos à gate. Esta atualização é feita fazendo uma query pela linha da tabela **Gate** cujo id correponde ao da gate desejada,

e depois incrementar o valor presente na coluna gateOpens.

Por fim, ainda no serviço, caso se receba do endpoint **/API/gateOpens/<path:gate_id>** um json object com uma mensagem que não uma de erro, é então enviado um pedido para o endpoint do **Gate Data /API/gateAccess/<path:state>**. Mediante o valor da mensagem este endpoint permite a inserção na tabela **GateAccessLog** de um acesso a uma gate com ou sem sucesso. Se o state for "0", então é feito um registo de uma tentativa sem sucesso e se o state for "1", então é feito um registo de uma tentativa de acesso com sucesso. No caso de uma tentativa sem sucesso, o userID não é guardado na tabela, ficando definido a "−1". Se algum destes passos resultar num erro, será mostrado no browser uma mensagem de erro de validação do QRCode.

Considerando a não ocorrência de quaisquer erros, o AJAX na página Web irá receber um objeto json com a validade do código. Se o código for válido é representada a abertura da gate com um semáforo, que fica verde durante o tempo em que a gate está aberta, 5 segundos. Durante este tempo o scanner fica em pausa, de modo a evitar novos pedidos. Caso seja inválido, então o semáforo mantém-se vermelho, e o scanner fica pausado durante 2 segundos, por escolha própria.

4.5 Criação de uma Gate por um Administrador

Ao aceder à página inicial do **Admin Web App** (endpoint **/admin_web_app**), no menu há uma opção "Register New Gate", que lhe permite registar na **Gate Data** uma nova gate. Ao clicar na opção, pelo endpoint **/newGate**, o menu leva o administrador a uma página onde este pode escrever o gateID e a gateLocation desejados. Quando estiver satisfeito, o administrador pode carregar no botão "Submit"para guardar a gate na base de dados.

Ao fazê-lo, é reencaminhado para um novo endpoint **/createGate**. Este endpoint faz dois pedidos ao **Gate Data**.

Primeiro, é feito o pedido para criar uma nova gate com os dados introduzidos, através do endpoint **/API/gates - POST**. É verificada a existência de uma gate com esse gateID e, caso não exista, é gerado um gateSecret novo. A nova gate é guardada na tabela **Gate** e é retornada como objeto json.

Depois, em caso de sucesso do pedido anterior, é feito o pedido para obter a informação sobre a gate criada, através do endpoint **/API/gate/<path:gate_id>**, onde o gate_id é o id da gate criada. Este endpoint retorna para o serviço um json object com o conteúdo da linha da tabela **Gate** correspondente à gate desejada.

A resposta ao pedido é então apresentada ao administrador. Em caso de erro, o administrador é redirecionado para a página inicial da **Admin Web App**, e é-lhe demonstrada uma mensagem de erro.

4.6 Enumeração pelo Administrator de Acessos a Gates

Na página inicial do **Admin Web App**, o administrador para visualizar a lista de todas as tentativas de acesso às gates, clica no botão "List All Gate Accesses"(endpoint **/listGateLog**). Ao fazê-lo, é enviado um pedido à base de dados da Gate (endpoint **/API/gateAccessLog**).

A base de dados faz uma query para ir buscar à tabela **GateAccessLog** toda a informação guardada, retornando para o serviço o resultado da query num objeto json.

O Service recebe o objeto json e, em caso de sucesso, é apresentada a lista de acessos ao administrador,através de uma tabela em HTML.

Caso se verifique algum erro, o administrador é reencaminhado para a página principal do **Admin Web App**, onde irá visualizar uma mensagem de erro.

5 Validações

Para o programa correr como pretendido, é necessário fazer validações ao longo dos diversos passos, tais como:

- verificar se o utilizador se autenticou
- verificar se a gate existe
- verificar a validade do QRCode
- verificar se os pedidos são executados como pretendido

5.1 Log-in do utilizador

Ao iniciar a aplicação Web, é necessário fazer uma autenticação com a plataforma Fénix. Caso não se tenha já aceite, é também necessário aceitar o acesso às informações necessárias para o funcionamento do programa.

Feita esta autenticação, é atribuído ao utilizador um *userID* (como explicado na secção **Modelos de Dados**), e o utilizador terá então acesso ao programa.

Para um utilizador ter acesso de administrador, é necessário a conta de Fénix ter o ID na lista de acessos administrativos ("admin_list"do código **service.py**).

Para garantir que utilizadores não administradores não tenham acesso à **Admin Web App**, foi criado um Flask Decorator, entitulado de admin_access. Este decorator é aplicado a todos os endpoints da **Admin Web App** e tem como objetivo garantir que o utilizador logado tem permissão para aceder a tais páginas. Isto é feito com o token de autenticação da sessão, que pode ser usado de modo a obter-se o username do utilizador logado. Se este username corresponder ao de um administrador, então o endpoint do **Admin Web App** pode ser acedido, caso contrário, o utilizador será redirecionado para a **User Web App**.

De modo a garantir que todos os utilizadores se encontram sempre autenticados, na própria **User Web App** foi adicionado também um Flask Decorator. No entanto, este decorator não faz qualquer tipo de comparação de usernames, servindo só para garantir que o utilizador logado possui um token de autenticação válido.

Caso um utilizador a usar a programa não se encontre autenticado, estes decorators redirecionam o utilizador para a **Authentication Page**.

Na **Gate Web App** não foi adicionado qualquer tipo de decorator para autenticação.

5.2 Acesso à Gate

Na **Gate Web App** (**http://127.0.0.1:8000/gate_web_app**) é possível aceder-se a uma dada gate, através de uma página com dois campos para preencher, o gateID e o gateSecret.

Preenchidos os dois campos, estes são enviados por AJAX para o serviço, de modo a verificar a validade dos mesmos. Esta validade é obtida através da tabela **Gate** na **Gate Data**. O service pede por REST, através do endpoint **/API/gates/<path:gate_id>/<path:gate_secret>**. Este endpoint faz a procura na tabela **Gate** por uma gate cujo id seja igual ao id introduzido na **Gate Web App**. Caso encontre algum registo que corresponda, então faz seleciona esse registo e compara o valor guardado na coluna gateSecret (segredo real da gate) com o segredo introduzido na **Gate Web App**. Se estes segredos forem iguais, então o acesso à gate é validado e verificado. Caso contrário, o acesso é negado.

5.3 QRCode

Para ler o QRCode é necessário apontar a câmara do scanner presente na **Gate Web App** para o QRCode presente na **User Web App**.

O QRCode gerado pelo programa tem como informações o userID e o userCode, dispostos numa string com a forma ("userID userCode"). Quando um QRCode é lido, é feita a a separação do conteúdo da string, do lado da **Gate Web App**, de modo a que este possa ser enviado para o serviço.

Se o formato do QRCode estiver correto, então, no serviço, será feito um pedido à **User Data**, para que esta procure algum registo de um userCode igual ao userCode lido. Se existir algum registo igual na tabela **User**, então podemos estar perante um código válido. Para se ter a certeza tem de se verificar se o código está expirado.

Quando é gerado um userCode, é também guardado na tabela **User** o momento da sua criação (codeTime), para ser possível verificar se o código expirou.

Assim, ao tentar validar o código, é feita a comparação entre o momento da criação do código presente na tabela e o momento atual. Se tiver passado mais do que 1 minuto, ocorreu o timeout do código e este passa a ser inválido.

5.4 Pedidos

Para realizar pedidos aos servidores, é feito o recurso de lógica "try except"do Python. Quando é feito um pedido, numa ligação REST de tipo cliente-servidor, o servidor faz *try* do código que se pretende executar e, se o código for válido, apresenta a página seguinte, como esperado.

Se não for possível realizar o código, é considerado que ocorreu uma falha. Assim, é executado o código em *except*.

É neste *except* que são enviados os códigos de erro, que demonstram a falha do funcionamento do programa. Perante os erros também é apresentada uma página ou mensagem de erro. Consoante o erro ocorrido, a página de erro será diferente para fornecer mais informação ao utilizador.

6 Tolerância de Falhas

Para garantir o acesso à base de dados foi criado um sistema que permite correr várias réplicas do servidor, para que consigam fornecer a informação sobre as Gate que forem pedidas.

Ao executar o programa *"gate_data.py"* múltiplas vezes, os servidores criados são abertos em Portos diferentes, de modo a poder aceder qualquer um dos servidores.

Assim, se um servidor estiver em baixo, os pedidos são respondidos na mesma por outro servidor, garantindo que ao cair um servidor o sistema não falha por completo. Estes servidores acedem uma base de dados comum.

7 Libraries Usadas

Para fazer este programa, foram usadas diversas bibliotecas.

- flask: para implementar os servidores com endpoints REST e Web
- sqlalchemy: para implementar as bases de dados
- oauth: para aceder ao fénix e realizar as autenticações
- html5-qrcode: para implementar a leitura do QRCode em Javascript
- qrious: para criar o QRCode em Javascript

8 Diferenças Face ao Projeto Intermédio

Durante a realização do projeto final, tiveram que se fazer algumas alterações em relação ao projeto intermédio.

Como os acessos do utilizador e da gate no projeto final são feitos pela Web, os ficheiros python do gate app e user app existentes para a entrega intermédia foram eliminados, passando a se fazer todo o acesso pela Web.

Com esta alteração também houve endpoints que deixaram de existir por terem a antiga função de fazer a ligação entre estes ficheiros de python e o serviço, através do protocolo REST. Agora estas ligações são feitas por HTTP. Como tal, os endpoints **/API/receiveUserCode** e **/API/users/generateUserCode** deixaram de ter uso, face à passagem para a Web.

Para além disso, a base de dados foi separada em 2 ficheiros, em vez de estarem apenas no *gatedata.py* ficaram no *gate_data.py* e no *user_data.py*.

Também se alterou o tipo de dados do userID da tabela **User** para String. No projeto intermédio, como só era necessário fazer o sistema para um único utilizador, considerou-se o userID sempre "1". No projeto final, visto ser desenhado para múltiplos utilizadores, foi necessário alterar a lógica por detrás das queries que faziam inserts de valores de userID, de modo a poderem ser inseridos valores variáveis. Tendo em consideração que o id de autenticação do Fenix corresponde ao tipo "ist1XXXXX", optou-se por modificar o tipo de dados para string, de modo a poder tornar o id de autenticação no userID do sistema.