



Mestrado em Engenharia Electrotécnica e de Computadores

## Arquitectura de Sistemas de Internet

1º Semestre 2020/2021

VQA – Video Question & Answer system

Relatório do Projeto

Grupo 9

Paulo [REDACTED]  
Ricardo [REDACTED] Barreto



# Conteúdo

<b>1</b>	<b>Introdução . . . . .</b>	<b>1</b>
<b>2</b>	<b>Arquitetura do Sistema . . . . .</b>	<b>1</b>
<b>3</b>	<b>Documentação dos Endpoints Implementados . . . . .</b>	<b>2</b>
3.1	Proxy . . . . .	2
3.2	Vídeo . . . . .	4
3.3	Utilizador . . . . .	4
3.4	QA . . . . .	5
3.5	Logs . . . . .	6
<b>4</b>	<b>Tecnologias e Bibliotecas Utilizadas . . . . .</b>	<b>6</b>
<b>5</b>	<b>User Interface do Sistema . . . . .</b>	<b>6</b>
<b>6</b>	<b>Integração com o FENIX . . . . .</b>	<b>8</b>
<b>7</b>	<b>Extensibilidade . . . . .</b>	<b>8</b>
<b>8</b>	<b>Tolerância a Falhas e Escalabilidade . . . . .</b>	<b>8</b>
<b>9</b>	<b>Conclusão . . . . .</b>	<b>9</b>

## 1 Introdução

O objetivo deste trabalho é desenvolver um sistema Video Question & Answer (VQA) que permite utilizadores registados a partir do FENIX visualizarem vídeos armazenados no sistema e fazerem questões sobre estes ou responderem às questões. Os utilizadores registados podem também ser administradores, onde os IDs destes estão armazenados numa lista no proxy, podendo ver estatísticas sobre todos os utilizadores e os logs de tudo o que ocorre no sistema, nomeadamente comunicações entre os utilizadores e o proxy e armazenamento de novos dados.

O *front-end* do sistema corre no browser e foi desenvolvido utilizando HTML e JavaScript, sendo as páginas do website fornecidas ao utilizador pelo proxy. O *back-end* foi desenvolvido utilizando Python, juntamente com outras bibliotecas que irão ser apresentadas na Secção 4.

## 2 Arquitectura do Sistema

O funcionamento e arquitetura do sistema encontram-se ilustrados na Figura 1.

O utilizador, interagindo com o browser, comunica diretamente com o proxy. O proxy é responsável por comunicar com os restantes componentes do serviço, escondendo-os do utilizador, de modo a que este não tenha acesso direto aos componentes. Estes componentes são:

- **Vídeo** - contém todos os recursos relativamente a vídeos, como estatísticas e acesso a vídeos existentes na base de dados;
- **Utilizador** - abrange toda a informação e estatísticas relativamente aos utilizadores;
- **QA** - inclui todas as questões e respostas feitas sobre os vídeos, juntamente com as suas estatísticas;
- **Logs** - armazena e devolve logs sobre todas as comunicações entre o utilizador e o proxy e sobre novos armazenamentos de dados, como um vídeo ou uma questão.

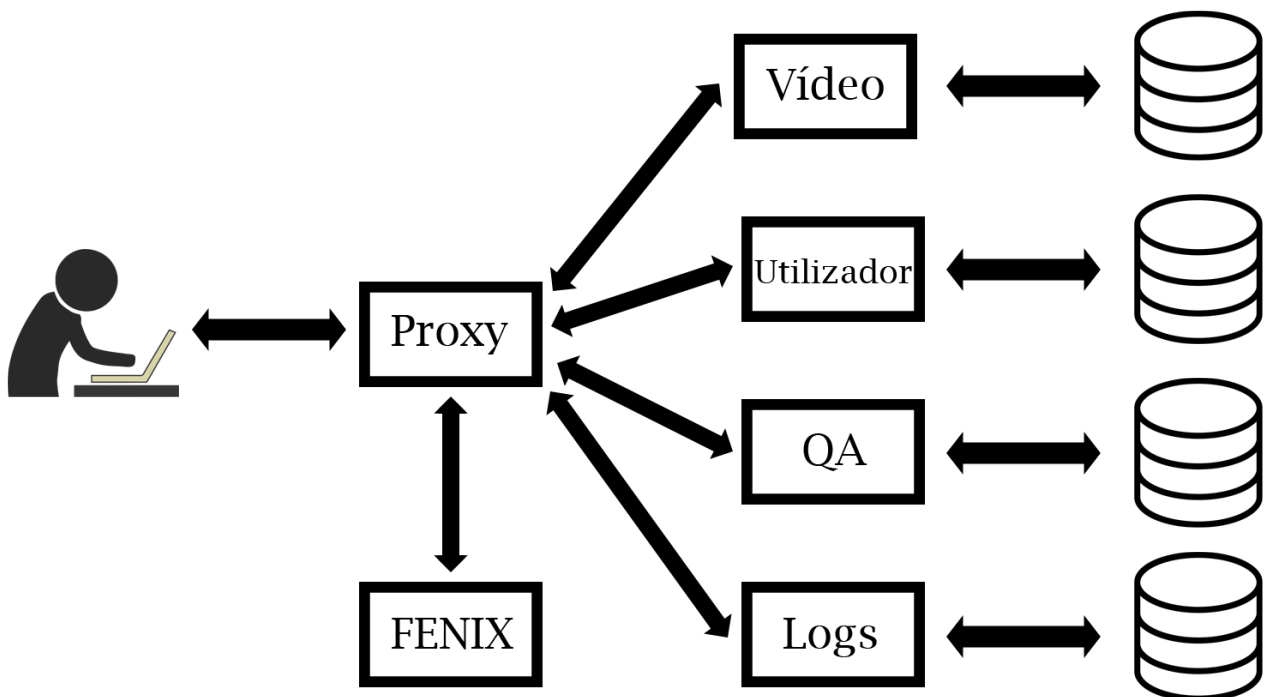


Figura 1: Arquitectura do sistema.

A cada um destes componentes está unicamente associada uma base de dados, onde esta apenas pode ser acedida por esse componente. Os componentes funcionam individualmente e em portas diferentes.

A integração com o FENIX está implementada no proxy. Este comunica diretamente com a API do FENIX para obter a informação relativamente ao utilizador e, mediante esta, permitir que o utilizador comunique com os endpoints existentes no proxy. Os restantes componentes não têm acesso direto à API do FENIX, apenas recebem informação sobre o utilizador a partir do proxy.

Os logs são criados diretamente no proxy, sempre que o utilizador interage com o sistema. Este é um potencial problema do sistema, dado que como são criados antes de obter a resposta do endpoint que verifica o correto funcionamento das funções, como adicionar uma nova resposta a uma questão de um vídeo, pode criar logs de algo que acaba por não se realizar.

### 3 Documentação dos Endpoints Implementados

Como ilustrado na Secção 2, o proxy comunica e interage com os restantes componentes do sistema, nomeadamente vídeo, utilizador, QA e logs. Todos estes componentes implementam endpoints, sendo estes mesmos espelhados no proxy, de modo a que o utilizador nunca aceda diretamente a estes componentes, estando limitado a comunicar com o proxy.

Todos os endpoints onde nenhum método é explicitamente indicado o default utilizado é o GET, sendo o GET apenas indicado explicitamente para diferenciar endpoints semelhantes mas com métodos diferentes.

#### 3.1 Proxy

O proxy é responsável pela comunicação entre o browser, operado pelo utilizador, e os restantes serviços, nomeadamente vídeo, utilizador, QA e logs.

Quando o utilizador entra na página inicial do website, este tem a opção de fazer login, adicionando um novo utilizador à base de dados caso este ainda não esteja presente nesta.

```
@app.route('/')
```

Sempre que este queira fazer logout, a sua conta é desconectada e este é redirecionado para a homepage novamente, onde pode fazer login.

```
@app.route('/logout')
```

Em todos os endpoints, com a exceção destes últimos dois, verifica-se se o utilizador já fez login e, caso não o tenha feito, é redirecionado para o primeiro endpoint, de modo a que este faça login.

Este endpoint apenas pode ser acedido por um utilizador registado. Consiste num menu onde o utilizador pode aceder a uma lista de vídeos ou, caso este seja administrador do sistema, as estatísticas de todos os utilizadores ou uma lista de logs.

```
@app.route('/private/')
```

As listas e estatísticas previamente referidas podem ser acessadas, nomeadamente, por estes endpoints.

```
@app.route('/videolist/')
@app.route("/userslist/")
@app.route('/loglist/')
```

Quando o utilizador acede à lista de vídeos, o endpoint seguinte é utilizado, retornando uma lista de JSON com todos os vídeos presentes na base de dados.

```
@app.route('/API/videos/')
```

O utilizador pode adicionar um novo vídeo, incrementando posteriormente o número de vídeos submetidos por este, utilizando respetivamente os seguintes endpoints.

```
@app.route("/API/videos/", methods=['POST'])
@app.route("/API/users/<string:id>/videos/", methods=['PUT'])
```

Este, ao seleccionar um vídeo, é redireccionado para um template HTML com toda a informação respetiva deste, incrementando o número de visualizações do vídeo.

```
@app.route("/showvideo/<int:id>/")
@app.route("/API/users/<string:id>/views/", methods=['PUT'])
```

O template HTML utiliza este endpoint para obter a informação correspondente ao vídeo pedido.

```
@app.route("/API/videos/<int:id>/")
```

O utilizador consegue visualizar questões sobre o vídeo e adicionar uma nova questão sobre este, incrementando o número de questões do vídeo e feitas pelo utilizador. Estas funcionalidades correspondem respetivamente a estes endpoints.

```
@app.route("/API/qa/questions/<int:id>/")
@app.route("/API/qa/questions/", methods=['POST'])
@app.route("/API/videos/<int:id>/questions/", methods=['PUT'])
@app.route("/API/users/<string:id>/questions/", methods=['PUT'])
```

Este, ao seleccionar uma questão em específico, obtém todas as respostas referentes a essa questão, podendo também adicionar uma nova resposta, incrementando o número de respostas dadas pelo utilizador.

```
@app.route("/API/qa/answers/<int:id>/")
@app.route("/API/qa/answers/", methods=['POST'])
@app.route("/API/users/<string:id>/answers/", methods=['PUT'])
```

Passando para outro serviço, no caso de o utilizador, sendo este administrador, querer obter as estatísticas de todos os utilizadores, a tabela com estas estatísticas é criada dinamicamente, adicionando individualmente cada utilizador, utilizando respetivamente os seguintes endpoints.

```
@app.route("/API/users/")
@app.route("/API/users/<string:id>/")
```

De modo análogo, este pode visualizar os logs de tudo o que ocorre na comunicação entre o proxy e os restantes serviços e sempre que alguma informação é adicionada às bases de dados, como um novo utilizador, vídeo, pergunta ou resposta.

```
@app.route("/API/logs/", methods=['GET'])
```

Estes logs são armazenados numa base de dados utilizando um endpoint acedido diretamente pelo proxy.

```
@app.route("/API/logs/", methods=['POST'])
```

## 3.2 Vídeo

A aplicação **vídeoapp.py** é responsável por executar todas as operações, redirecionadas pelo proxy, relacionadas com os vídeos.

Nesta parte do projecto é usada uma tabela relacional chamada YTVideo com os atributos id (identificador do vídeo), description, url e questions (número de questões).

O seguinte endpoint imprime no web browser o template da página onde é possível visualizar o vídeo escolhido, juntamente com a tabela das perguntas/respostas do respectivo vídeo/pergunta. Há ainda a opção de adicionar novas perguntas/respostas. É recebido o id do vídeo (proveniente do valor presente do endpoint) e o nome e id do utilizador (método GET) que são, posteriormente, enviados para o template.

```
@appvideo.route("/showvideo/<int:id>/")
```

Do mesmo modo, o seguinte endereço imprime no web browser o template da página que faz a listagem dos vídeos e o formulário que permite adicionar um novo vídeo. É recebido e passado para o template o id do utilizador (id é recebido pelo método GET).

```
@appvideo.route("/videolist/")
```

Retorna a informação guardada na base de dados de todos os vídeos no formato lista de JSON.

```
@appvideo.route("/API/videos/", methods=['GET'])
```

Retorna a informação guardada na base de dados de um vídeo específico no formato JSON. Recebe o id do vídeo pelo valor presente no endpoint.

```
@appvideo.route("/API/videos/<int:id>/")
```

Endpoint para criar um novo vídeo (guarda a informação do novo vídeo na base de dados). Recebe os atributos do vídeo pelo método POST.

```
@appvideo.route("/API/videos/", methods=['POST'])
```

Incrementa o contador de número de questões de um vídeo específico. Recebe o id do vídeo pelo valor presente no endpoint.

```
@appvideo.route("/API/videos/<int:id>/questions/", methods=['PUT'])
```

## 3.3 Utilizador

A aplicação **userapp.py** é responsável por executar todas as operações, redirecionadas pelo proxy, relacionadas com utilizadores.

Nesta parte do projecto é usada uma tabela relacional chamada User com os atributos user\_id, name, admin, views (número de vídeos vistos), videos (número de vídeos adicionados), questions (número de questões) e answers (número de perguntas).

O seguinte endpoint imprime no browser o template estático da página onde é possível visualizar as estatísticas referentes aos utilizadores registados.

```
@appuser.route("/userslist/")
```

Sempre que um utilizador faz o login no website, é chamado o seguinte endpoint onde é adicionado o novo utilizador à base de dados. Caso não seja a primeira vez que o utilizador faça o login é emitido o erro 409. Recebe os atributos do novo utilizador pelo método POST.

```
@appuser.route("/API/users/", methods=['POST'])
```

Retorna a informação guardada na base de dados de todos os utilizadores no formato lista de JSON.

```
@appuser.route("/API/users/", methods=['GET'])
```

Retorna a informação guardada na base de dados de um utilizador específico no formato JSON. Recebe o id do utilizador pelo valor presente no endpoint.

```
@appuser.route("/API/users/<string:id>/")
```

Os últimos 4 endpoints, incrementam os contadores das estatísticas do utilizador actual, respectivamente, questões feitas, respostas a questões, número de vídeos vistos e número de vídeos submetidos. Todos os 4 endpoints recebem o id do utilizador pelo valor presente no endpoint.

```
@appuser.route("/API/users/<string:id>/questions/", methods=['PUT'])
@appuser.route("/API/users/<string:id>/answers/", methods=['PUT'])
@appuser.route("/API/users/<string:id>/views/", methods=['PUT'])
@appuser.route("/API/users/<string:id>/videos/", methods=['PUT'])
```

### 3.4 QA

A aplicação **qaapp.py** é responsável por executar todas as operações, redirecionadas pelo proxy, relacionadas com a criação e a listagem de perguntas e respostas.

Nesta parte do projecto são usadas duas tabelas relacionais, uma chamada Questions com os atributos question\_id, video\_id, question\_time, user\_id, user\_name e text e outra chamada Answers com os atributos answer\_id, question\_id, user\_id, user\_name e text.

Endpoint para criar uma nova questão no vídeo seleccionado (guarda a informação da nova questão na tabela Questions). Recebe os atributos da pergunta pelo método POST.

```
@appqa.route("/API/qa/questions/", methods=['POST'])
```

Endpoint para criar uma nova resposta à questão seleccionada (guarda a informação da nova resposta na tabela Answers). Recebe os atributos da resposta pelo método POST.

```
@appqa.route("/API/qa/answers/", methods=['POST'])
```

Retorna a informação guardada na base de dados de todas as perguntas do vídeo seleccionado no formato lista de JSON. Recebe o id do vídeo pelo valor presente no endpoint.

```
@appqa.route("/API/qa/questions/<int:id>/", methods=['GET'])
```

Retorna a informação guardada na base de dados de todas as respostas da questão seleccionada no formato lista de JSON. Recebe o id da pergunta pelo valor presente no endpoint.

```
@appqa.route("/API/qa/answers/<int:id>/", methods=['GET'])
```

### 3.5 Logs

A aplicação **logapp.py** é responsável por executar todas as operações, redirecionadas pelo proxy, relacionadas com o registo de um novo log e a obtenção de todos os logs guardados em memória.

Nesta parte do projecto é usada uma tabela relacional chamada Logs com os atributos `log_id`, `ip`, `endpoint`, `method`, `timestamp` e `description`.

O seguinte endpoint imprime no web browser o template estático da página onde é possível visualizar os logs registados pelo programa.

```
@applog.route("/loglist/")
```

Retorna a informação guardada na base de dados de todos os logs no formato lista de JSON.

```
@applog.route("/API/logs/", methods=['GET'])
```

Endpoint para guardar a informação de um novo pedido (guarda a informação do novo log na tabela Logs). Recebe os atributos do novo registo (log) pelo método POST.

```
@applog.route("/API/logs/", methods=['POST'])
```

## 4 Tecnologias e Bibliotecas Utilizadas

Como já foi referido anteriormente, a arquitectura do sistema VQA é um conjunto de um proxy mais 4 aplicações que correm em simultâneo e que necessitam a qualquer momento comunicar para solicitar informação requerida pelo utilizador.

Para evitar fazer programação ao nível de sockets é utilizado o middleware *flask*. As funções *redirect*, *url\_for* e *requests* são também usadas para auxiliar as comunicações e a interação com a aplicação web.

Ao nível das bases de dados, para criar, preencher, ler ou alterar o valor de uma célula da tabela é usado a biblioteca *sqlalchemy* do *Python*.

De forma a que os utilizadores se possam autenticar pelo FENIX é usada a função *OAuth2ConsumerBlueprint*.

Para chamar os templates das páginas web é usada a função *render\_template* e o método *send\_static\_file*. Nos templates, é utilizado *HTML*, *CSS*, *JavaScript*, *JQuery* e, ainda, a tecnologia *AJAX* que permite actualizar páginas web de forma assíncrona.

Na aplicação, é ainda usado o formato de dados JSON para guardar vários atributos sobre um objecto e o método *datetime.now* para obter o instante actual.

## 5 User Interface do Sistema

O utilizador, ao entrar no site pelo endereço 127.0.0.1:5000, vai-lhe ser pedido para se autenticar pelo FENIX. Ao fazê-lo, o browser vai apresentar a página web da figura 2, onde o utilizador pode fazer o logout ou prosseguir para o site principal.

Escolhendo a opção de continuar, o utilizador vai visualizar o menu principal apresentado na figura 3, onde existe a opção de visualizar a lista de vídeos guardados na base de dados, os logs do sistema ou as estatísticas do utilizador (estas últimas duas opções necessitam de privilégios de administrador).





Figura 2: Página do utilizador autenticado que permite que o utilizador seja redirecionado para o site principal.



Figura 3: Menu principal com as diversas opções que os utilizadores podem escolher.

Ao seleccionar a primeira opção, é impresso no browser um template, presente na figura 4, com uma tabela com a descrição e o número de perguntas de todos os vídeos submetidos e um formulário para adicionar um novo vídeo ao sistema.

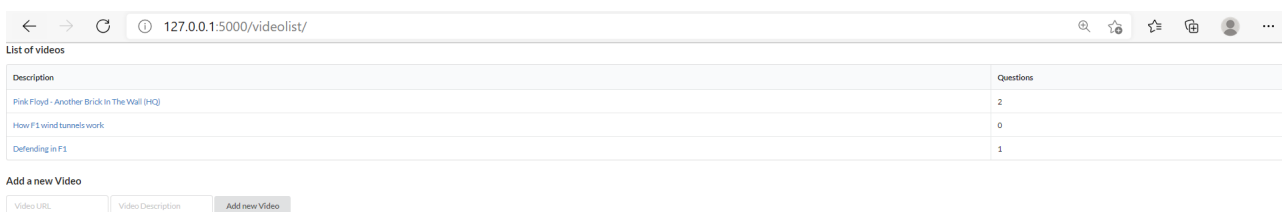


Figura 4: Listagem dos vídeos armazenados no sistema.

Selecionando o primeiro vídeo, o utilizador depara-se com a página presente na figura 5, onde é possível visualizar o vídeo, as respetivas perguntas e, caso seleccione uma pergunta, as respetivas respostas.

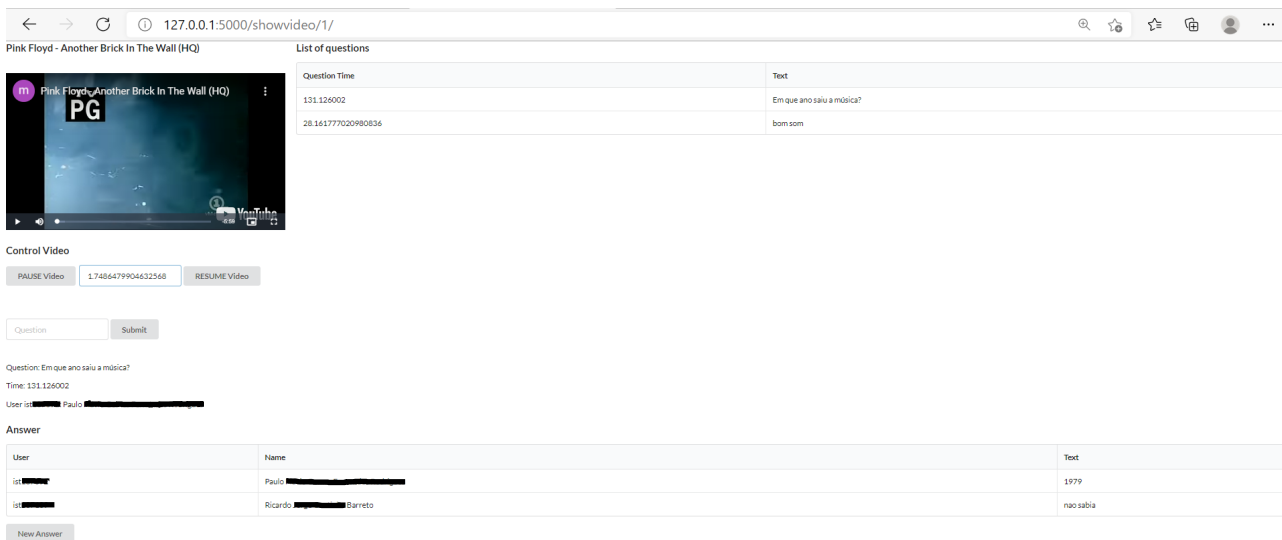
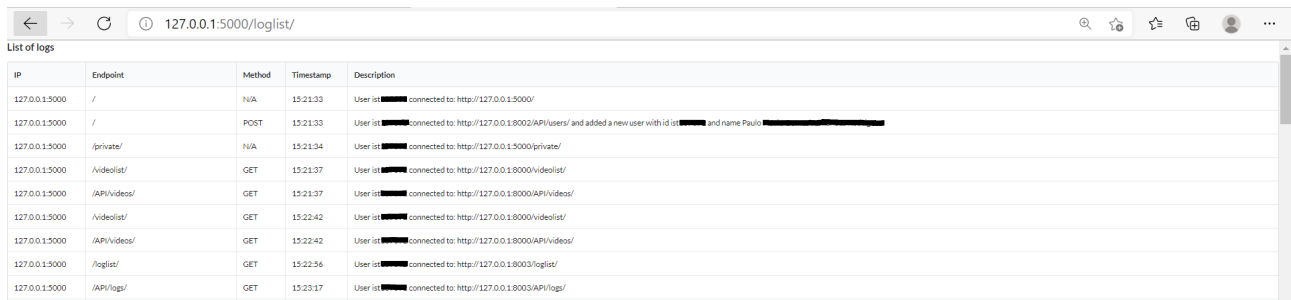


Figura 5: Página do vídeo 1 com as respetivas perguntas e com a pergunta 1 seleccionada.

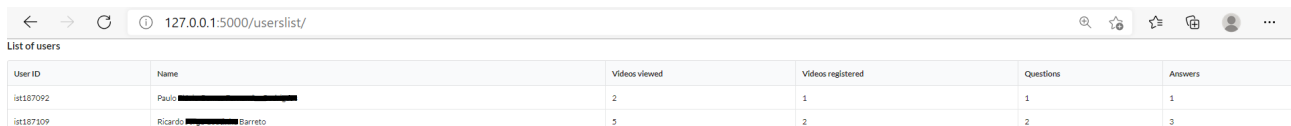
Voltando ao menu principal (figura 3), e seleccionando a segunda opção é possível visualizar os registos (logs) do programa. Esta página é apresentada na figura 6.

Finalmente, se o utilizador pretender visualizar as estatísticas do utilizador, basta escolher a terceira opção no menu principal. Esta operação vai mostrar uma nova página com uma tabela onde é possível verificar o número de vídeos vistos, o número de vídeos registados, o número de perguntas feitas e o número de respostas a perguntas que cada utilizador efetuou (figura 7).



IP	Endpoint	Method	Timestamp	Description
127.0.0.1:5000	/	N/A	15:21:33	User ist connected to: http://127.0.0.1:5000/
127.0.0.1:5000	/	POST	15:21:33	User ist connected to: http://127.0.0.1:8000/API/users/ and added a new user with id ist and name Paulo
127.0.0.1:5000	/private/	N/A	15:21:34	User ist connected to: http://127.0.0.1:5000/private/
127.0.0.1:5000	/videolist/	GET	15:21:37	User ist connected to: http://127.0.0.1:8000/videolist/
127.0.0.1:5000	/API/videos/	GET	15:21:37	User ist connected to: http://127.0.0.1:8000/API/videos/
127.0.0.1:5000	/videolist/	GET	15:22:42	User ist connected to: http://127.0.0.1:8000/videolist/
127.0.0.1:5000	/API/videos/	GET	15:22:42	User ist connected to: http://127.0.0.1:8000/API/videos/
127.0.0.1:5000	/loglist/	GET	15:22:56	User ist connected to: http://127.0.0.1:8000/loglist/
127.0.0.1:5000	/API/logs/	GET	15:23:17	User ist connected to: http://127.0.0.1:8000/API/logs/

Figura 6: Listagem dos logs do sistema.



User ID	Name	Videos viewed	Videos registered	Questions	Answers
ist187092	Paulo	2	1	1	1
ist187109	Ricardo Barreto	5	2	2	3

Figura 7: Listagem dos utilizadores que já fizeram o login na aplicação e as respectivas estatísticas.

## 6 Integração com o FENIX

Nesta aplicação, o FENIX é o sistema usado para os utilizadores se autenticarem.

Ao longo do programa é necessário obter certas informações sobre o utilizador. Deste modo, através de um *session.get* ao endpoint `"/api/fenix/v1/person/"`, é feito um pedido ao FENIX com o objectivo de obter os dados do utilizador autenticado. Esta informação é retornada em formato da classe *requests.models.Response*.

Independente de qual seja o endpoint, desde que seja válido, o proxy verifica sempre, através do objeto resultante da função *OAuth2ConsumerBlueprint*, se o utilizador está autenticado.

## 7 Extensibilidade

Todos os componentes do sistema funcionam individualmente e em portas diferentes. O sistema tem boa extensibilidade, dado que é fácil adicionar, remover ou modificar um componente, sendo apenas necessário implementar os endpoints relevantes nesse mesmo componente e, caso estes endpoints sejam novos, espelhar estes mesmos no proxy.

## 8 Tolerância a Falhas e Escalabilidade

Como já foi referido várias vezes ao longo do relatório, os componentes do sistema funcionam individualmente e em portas diferentes. Isto permite que no caso de um componente do sistema falhar, o resto do sistema continua a funcionar normalmente. Deste modo, o sistema tem uma boa tolerância a falhas.

Os componentes e as bases de dados estão ligados entre si, mas apenas unicamente - a cada componente corresponde uma base de dados diferente. Isto permite que cada componente, juntamente com a sua base de dados, consiga correr em máquinas separadas e individuais, sendo visto como um sistema distribuído, pois o utilizador apenas tem a perceção de o sistema ser uma única máquina. Deste modo, o sistema tem uma boa escalabilidade horizontal, pois pode aumentar ou diminuir o número de máquinas mediante o número de componentes do sistema, juntamente com uma boa escalabilidade vertical, dado que se pode melhorar o desempenho de cada máquina individual, adicionando mais RAM ou memória interna.

## 9 Conclusão

Todas as funcionalidades do sistema VQA foram implementadas corretamente. Como já foi referido na Secção 2, um dos aspetos a melhorar do sistema é apenas criar logs após os acontecimentos e não antes, de modo a evitar criar logs falsos. Outro aspeto a melhorar seria isolar mais as bases de dados dos componentes do sistema. Apesar de os componentes e as suas bases de dados poderem correr em máquinas individuais, ao separar as bases de dados destes, a carga de trabalho poderia ser distribuída por mais máquinas, onde cada base de dados poderia correr numa máquina separada, aumentando a disponibilidade do sistema, o que leva a uma melhor tolerância a falhas.