# Projeto - Relatório

Arquitectura de Sistemas de Internet

Grupo 37
Diogo Alves (86980)
Luís Crespo (87057)

Prof. João Nuno Silva

# 1 Arquitetura do Sistema

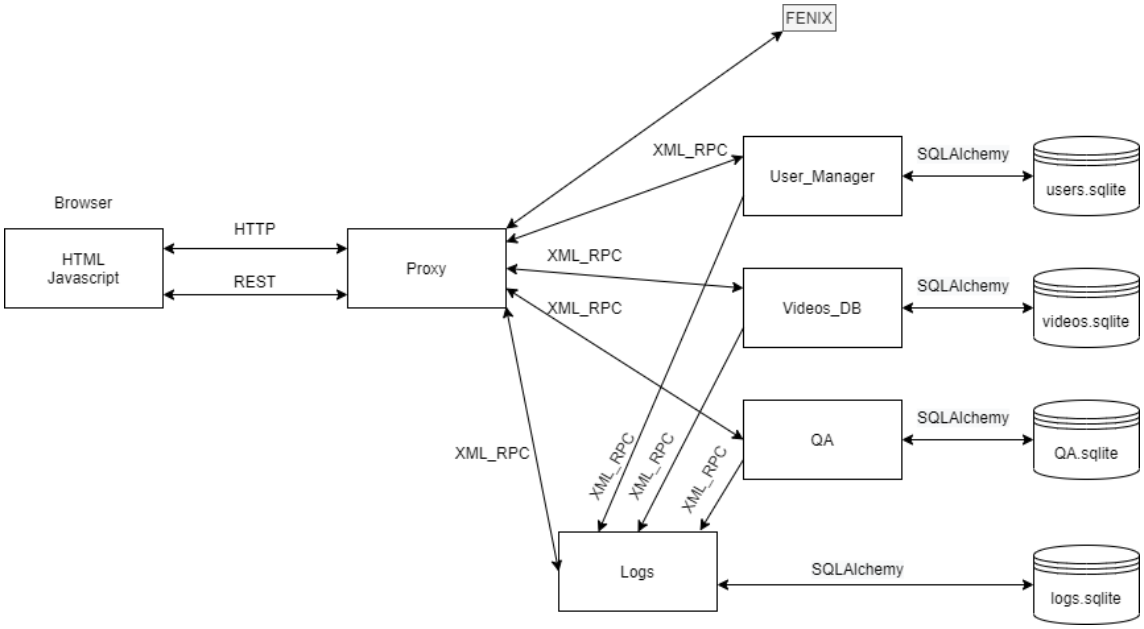


Figure 1: Arquitetura do Sistema

# 2 Endpoints REST

Foram definidos os seguintes *endpoints* de forma a implementar as funcionalidades de administrador:

- GET /API/admin/users/ – Retorna a lista de utilizadores.
- GET /API/admin/users/<string:user_id>/ – Retorna as estatísticas referentes ao ID de utilizador especificado.
- GET /API/admin/listLog/ – Retorna a lista de todos os *logs*, *requests* e criação de dados (*endpoint* não utilizado).
- GET /API/admin/listRequests/ – Retorna a lista de *logs* referentes a *requests*.
- GET /API/admin/listDataCreationEvents/ – Retorna a lista de *logs* referentes a criação de dados.

Relativamente às funcionalidades de utilizador, foram definidos os seguintes *endpoints*:

- GET /API/private/videos/ – Retorna a lista de vídeos.
- POST /API/private/videos/ – Cria um novo vídeo.
- GET /API/private/videos/<int:id>/ – Retorna os dados referentes ao vídeo especificado.
- GET /API/private/videos/<int:video_id>/questions/ – Retorna as questões referentes ao vídeo especificado.
- POST /API/private/videos/<int:video_id>/questions/ – Cria uma nova questão no vídeo especificado.
- GET /API/private/videos/<int:video_id>/questions/<int:question_id>/ – Retorna a questão especificada relativa ao vídeo indicado (*endpoint* não utilizado).
- GET /API/private/videos/<int:video_id>/questions/<int:question_id>/answers – Retorna as respostas relativamente a uma questão de um vídeo.

- POST /API/private/videos/<int:video_id>/questions/<int:question_id>/answers/ – Cria uma nova resposta a uma questão de um vídeo.
- GET /API/private/videos/<int:video_id>/questions/<int:question_id>/answers/<int:answer_id>/ – Retorna a resposta especificada referente a uma questão e respetivo vídeo (*endpoint* não utilizado).
- PUT /API/private/videos/<int:video_id>/views/ – Incrementa o número de visualizações do respetivo vídeo.

# 3 Tecnologias/Bibliotecas utilizadas

## 3.1 Python

Linguagem de programação utilizada para programar o *proxy* e os restantes módulos pertencentes ao servidor (User_manager, Videos_DB, QA, Logs).

## 3.2 HTML

HTML é utilizado para construír o "esqueleto" das páginas web. A biblioteca **fomantic** é utilizada para contruir tabelas e outros elementos de UI.

## 3.3 Javascript

Javascript é utilizado para atualizar dinamicamente as páginas web, no cliente, bem como para fazer request do cliente (browser) para o servidor (*proxy*), com recurso ao **ajax**.

## 3.4 REST

A comunicação entre o browser (javascript) e o *proxy* é feita usando a API REST. No *proxy* são definidos os *endpoints* e as respetivas funções e no javascript são feitos *requests* a esses *endpoints*.

## 3.5 Flask

A biblioteca Flask é utilizada para correr as aplicações web (uma para cada módulo do servidor) e definir os *endpoints* REST no *proxy*.

## 3.6 XML RPC

A comunicação entre o proxy e os restantes módulos (User_manager, Videos_DB, QA, Logs) é feita através de Remote Procedure Calls (RPC) e é utilizada a biblioteca flaskXMLRPC. Este biblioteca permite criar um "handler" que irá registar as funções destes módulos de forma a que estas possam ser chamadas remotamente pelo proxy.

## 3.7 SQLAlchemy

A biblioteca SQLAlchemy é utilizada para criar e gerir as bases de dados. Existem 4 bases de dados, sendo que cada uma é gerida independentemente das outras, por um módulo diferente.

# 4 Interface do utilizador

A página inicial tem 3 estados (Figura 2). Se o utilizador não tiver sessão iniciada no Fenix, tem que carregar para realizar login (A). Com sessão iniciada, se o utilizador não for administrador, pode aceder aos vídeos e fazer *logout* (B). Os administradores, para além das funcionalidades de utilizadores têm acesso aos *logs* e às estatísticas (C).

Ao carregar na lista de vídeos (Video List), o utilizador é redireccionado para uma página com a lista de vídeos. Nesta página é possível adicionar novos vídeos e aceder aos vídeos disponíveis.

Ao selecionar um vídeo, o utilizador é redireccionado para uma nova página (Figura 4) em que o id do vídeo é enviado como query string. O UI é praticamente igual ao definido no enunciado. Os elementos que não são necessários estão escondidos e são revelados ao carregar nos botões correspondentes. Para fazer uma questão é necessário carregar no botão. O vídeo é parado, e o tempo do vídeo é revelado bem como o local para colocar a questão e o respetivo botão de submissão. Para ver as respostas a uma pergunta ou responder, é necessário

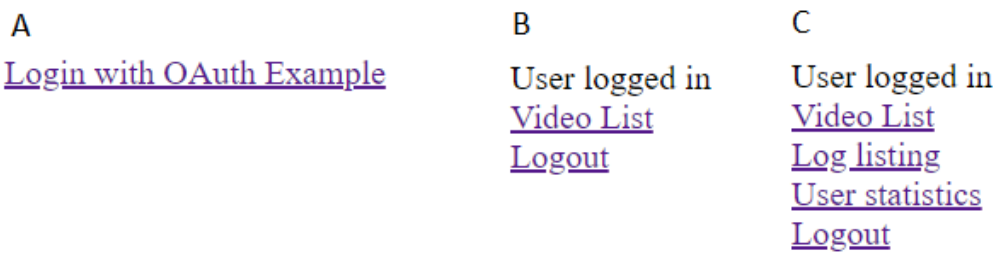


Figure 2: Diferentes estados da página inicial (http://127.0.0.1:5000/).

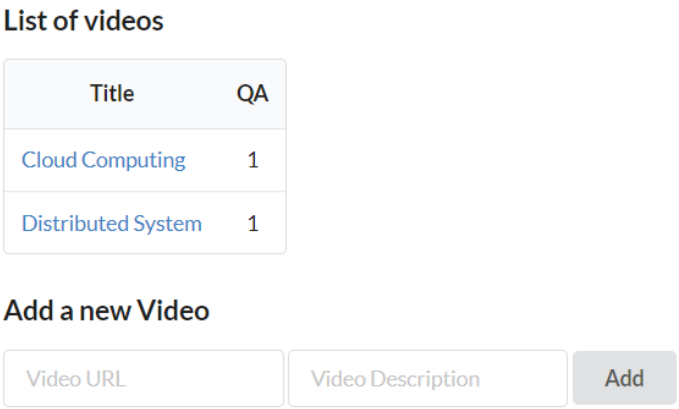


Figure 3: Lista de vídeos (http://127.0.0.1:5000/private/videos/).

selecionar a questão através do botão. É indicado o utilizador que fez a questão, as respostas e a opção de realizar uma questão. Na Figura 5 estão representados os elementos descritos.

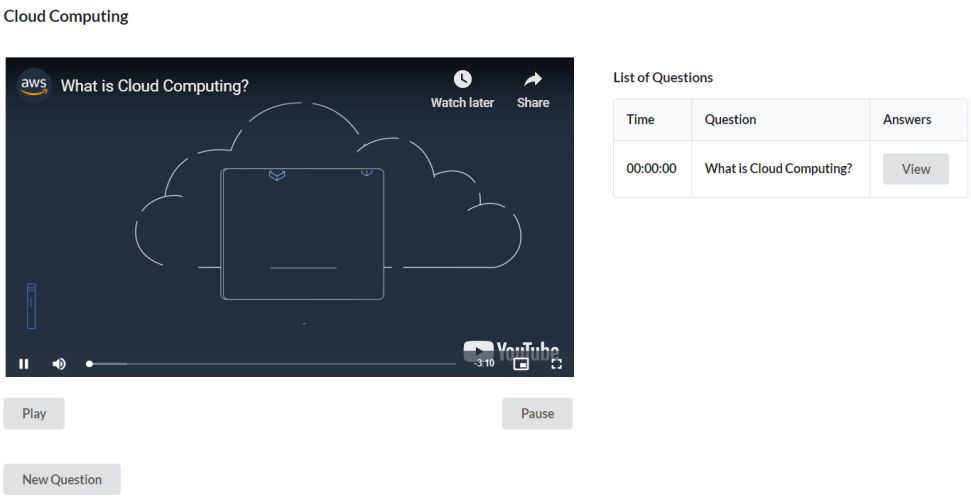


Figure 4: Vídeo especifico (http://127.0.0.1:5000/private/videos/video?id=1).

Relativamente às funcionalidades de administrador, ao selecionar o Log listing, o administrador é redireccionado para uma página com duas tabelas (Figura 6). Uma referente aos *requests* e outra referente à criação de dados.

Por fim e ainda relativamente a funcionalidades de administrador, ao selecionar o User statistics, o administrador é direccionado para uma página onde selecciona o utilizador e observa as suas estatísticas.

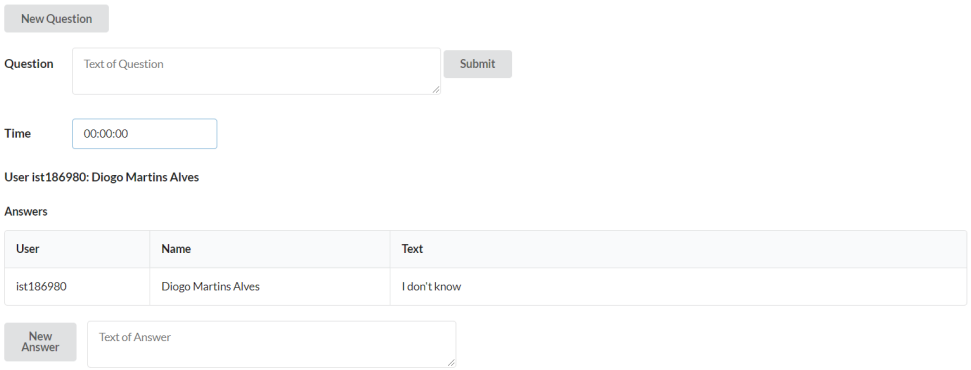


Figure 5: Fazer questões e ver/realizar respostas num vídeo (http://127.0.0.1:5000/private/videos/video?id=1).

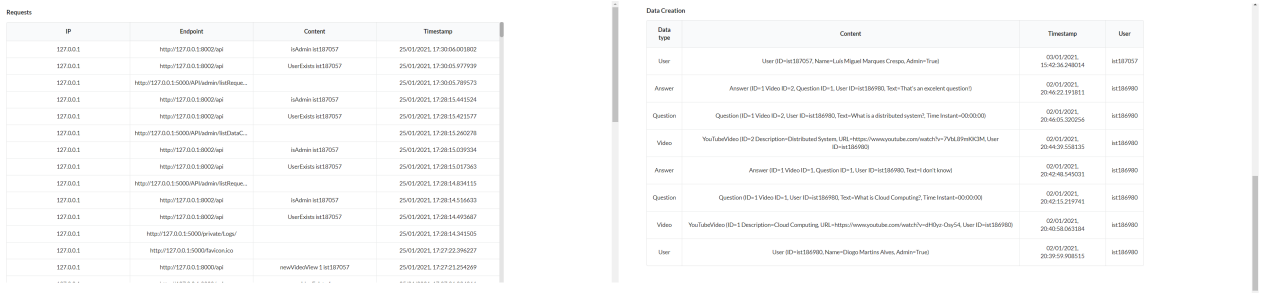

Figure 6: Logs (http://127.0.0.1:5000/private/Logs/).

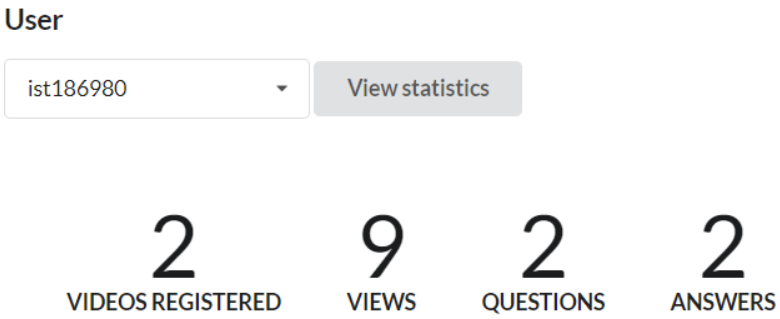


Figure 7: Estatísticas de utilizadores (http://127.0.0.1:5000/private/userstats/).

# 5 Integração do FÉNIX

A comunicação e integração do FÉNIX é realizada no *proxy*. A biblioteca flask_dance disponibiliza o módulo OAuth2ConsumenrBlueprint, com o qual é possível configurar uma *blueprint* da aplicação que criámos na nossa página pessoal do FÉNIX. Para tal, apenas necessitamos de inserir nesta configuração o "id"e o "secret" da nossa aplicação, para além dos URLs de acesso. Com esta *blueprint* conseguimos saber se um utilizador está registado no FÉNIX e obter algumas informações como por exemplo o nome e identificador ist. Foi criada a função verify_login, que utiliza esta *blueprint* para determinar se um utilizador está registado. Esta função é executada quando é feito um *request* ao *proxy*, de forma a garantir que o utilizador está autorizado a aceder ao *endpoint* pretendido.

# 6 Extensibilidade

Uma das vantagens de construir a aplicação por módulos é que esta se torna muito mais extensível. Esta melhoria na extensibilidade reflete-se não só na modificação de funcionalidades mas também na alteração e

adição de módulos. Como as componentes estão separadas, torna-se mais fácil adicionar funcionalidades a cada módulo, bastando para isso apenas adicionar ao módulo correspondente a função pretendida e o respetivo *endpoint*. Caso queiramos, por exemplo, modificar a aplicação de forma a que esta funcione agora também para livros, basta-nos adicionar um módulo que gere uma nova base de dados de livros, e no qual se encontram os *endpoints* RPC para que o *proxy* consiga aceder às funções que adicionam e retornam informação da base de dados. Os restantes módulos, como por exemplo o User_manager e as respetivas bases de dados, não precisam de ser modificados uma vez que é o *proxy* que irá realizar todas as verificações necessárias à legalidade das operações que se pretendem executar.

# 7 Tolerância a falhas e escalabilidade

Na nossa implementação desta aplicação, a tolerância a falhas é garantida também graças à separação das componentes em módulos. Isto é, se uma das componentes deixar de funcionar, dependendo do caso, o utilizador poderá receber uma mensagem de erro ou poderá nem ser notificado, mas em qualquer caso, a aplicação não deixará de funcionar.

Aqui, o uso de interfaces REST (no proxy) e RPC (nos módulos User_manager, Videos_DB, QA, Logs) bem definidas, faz também com que a aplicação fique mais extensível.

Uma das técnicas que pode ser usada para melhorar tanto a escalabilidade como a tolerância a falhas é o uso de vários servidores a correr o mesmo processo correspondente aos módulos User_manager, Videos_DB, QA e Logs. A tolerância a falhas seria garantida no sentido em que se um dos processos falhasse, os seus "clones" iriam substituí-lo e a escalabilidade seria também garantida porque, à medida que o número de utilizadores aumentasse, poderiam ser adicionados mais processos de forma a que todos os *requests* fossem processados eficazmente. Para esta implementação, teria apenas que ser garantida a concorrência no acesso à base de dados, visto que iriam existir múltiplos processos a tentar aceder ao mesmo recurso.