Eduasis - DAW2020

Catarina Cruz - A84011 Jorge Mota - A85272

7 February 2021

1 Introdução

Neste projeto foi criada uma plataforma educacional (Eduasis - Education Oasis) com o objetivo de partilhar vários tipos de ficheiros como livros, dissertações, apontamentos, entre outras coisas.

Estes ficheiros partilhados são chamados de recursos(*resources*) e podem ser criados *posts* sobre estes, contendo uma descrição. Para além disso, os utilizadores podem também classificar os recursos. Um *post* pode conter ainda comentários de outros utilizadores.

Nesta plataforma, existem vários tipos de utilizadores, todos eles têm acesso aos conteúdos disponibilizados, e existem ainda os produtores que podem adicionar novos recursos. Existem ainda os administradores, aqueles com as permissões mais elevadas, que conseguem controlar o sistema.

Neste relatório, é explicada a arquitetura implementada, especificando os servidores desenvolvidos. Para o servidor da base de dados foi utilizado *MongoDB*. Foram ainda realizados *scripts* populacionais em *bash*, *Pyton* e *JavaScript*. Para o servidor da API foi utilizado *NodeJS* e por último, para as interfaces dinâmicas foi utilizado a linguagem *Pug*.

2 Arquitetura

Para o desenvolvimento deste projeto foram criados três servidores para constituir uma arquitetura orientada ao microserviço de cada componente:

- DB (Database) server
- API server
- HTTP server

Ficamos com uma divisão clara das componentes que fornecem a lógica do sistema (DB e API Server) e a apresentação do serviço (App Server).

Estes servidores estão organizados (e a sua comunicação) segundo o esquema:

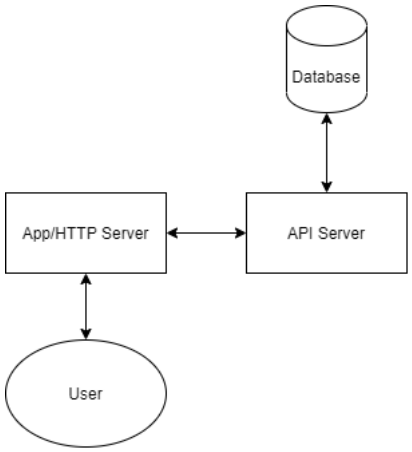

Figure 1: Arquitetura implementada

O DB server é a base de dados de todo o sistema, onde toda a informação está guardada, isto é, todos os utilizadores, recursos, posts e comentários.

O API server acede à base de dados e opera sobre a informação, controlando o comportamento de todo o sistema.

Por último, o HTTP server tem por objetivo apresentar as *views* da aplicação e interagir com o utilizador através de um *browser*, necessitando para isso, de fazer pedidos ao API server para obter a informação a apresentar.

3 API Server

3.1 Modelos de Dados

Modelamos os dados segundo esquemas defenidos no *mongoose* para obter as **Entidades** pretendidas.

Seguidamente apresenta-se os *schemas* das coleções modeladas:

User (user_schema)	
username:	String,
nickname:	String,
password_hash:	String,
email:	String,
affiliation:	String,
permissions:	Number,
registration_date:	Date,
last_login_date:	Date,
avatar_url:	String

Resource (resource_schema)	
resource_id:	String,
type_id:	Number,
author:	String,
title:	String,
description:	String,
filename:	String,
created_date:	Date,
visibility:	Number,
rate :	rate_schema

Rates(rates_schema)	
username:	String,
rated:	Number

Rate(rate_Schema)	
current_rate:	Number,
num_rates:	Number,
rates:	[rates_schema]

Resource_type (resource_type_schema)	
type_id:	Number,
name:	String

Post (post_schema)	
post_id:	String,
resource_id:	String,
content:	String,
author:	String,
created_date:	Date,
comments:	[comment_schema]

Comment (comment_schema)	
message:	String,
created_date:	Date,
author:	String

3.2 Autenticação

A autenticação neste tipo de sistemas é crucial, para assegurar esta componente, utilizamos *JSON Web Tokens* (JWT) com assinatura sobre uma chave secreta.

Utilizar este tipo de *tokens* permitou-nos garantir condições de segurança sobre a autenticação e ainda fornecer de modo imediato dados imutáveis calculados pelo *API Server*, alguns desses dados incluem o *username* do utilizador, as permissões que este utilizador tem e as UNIX *timestamps* da data de emissão do token e de expiração do mesmo.

```
{
  "username": "nome",
  "perms": 2,
  "exp": 1613327980,
  "iat": 1612723180
}
```

Para poder fazer *logout* com um token temos uma estratégia para invalidar estes, utilizando Token Blacklisting, registando todos os tokens que fizeram logout para invalida-los e impedir que estes sejam usados outra vez.

3.3 Permissões

Existem três tipos de permissões, Consumidores, Produtores e Administradores.

Os Consumidores apenas podem visualizar os Recursos dos outros Produtores, adicionar posts relativos a um recursos e comentar posts. Os produtores, para além de terem acesso a tudo o que os consumidores podem realizar, podem ainda adicionar recursos. Por último, os administradores conseguem também gerir recursos, ou seja, podem, por exemplo adicionar novos tipos de recursos, ou apagar qualquer tipo de entidade.

Para terem acesso à informação disponibilizada, todos os utilizadores necessitam de se autenticar, caso contrário apenas conseguem visualizar as páginas de registar e login.

3.4 Data id's

Tal como já mencionado anteriormente, existem *Users* que representam os utilizadores. Cada *user* apresenta um *username* único, sendo este o seu identificador, podendo assim apresentar *nicknames* comuns e repetidos.

Para o caso dos *Posts* e dos *Resources*, estes apresentam um identificador único gerado aleatoriamente pelo *mongodb*, passando estes a ser mencionados por *post_id* e *resource_id* respetivamente. Quando um post é relarivo a um *resource*, este apresenta o *resource_id* para identificar o respetivo *resource* sobre o qual está a referir-se

3.5 Endpoints

O API Server funciona a base de endpoints de interação, todos estes endpoints estão listados no seguinte link:

`https://github.com/k1llbyte/eduasis/wiki/api-documentation`

3.6 Armazenamento

Todo o armazenamento de dados relativos ao sistema é feito no API Server numa pasta tornada uma rota pública `/storage/`.

São armazenados Recursos e Avatares de utilizadores (`/storage/avatars/` e `/storage/resources/`).

4 App Server

No app server cria-se a parte visual da aplicação para o cliente poder navegar e aceder aos recursos facilmente. Inicialmente o utilizador encontra a página de *login* para se poder autenticar. Caso ainda não tenha uma conta criada, pode faze-lo se for para a página *register*.

Após autenticado, é apresentada uma página com os recursos e posts mais recentes, podendo o utilizador clicar nos mesmos para se dirigir às páginas específicas de cada um.

Caso o mesmo procure um recurso em específico, pode faze-lo na página de *resources*, aonde encontra uma lista de todos os recursos, podendo realizar procuras por título ou tipo de recursos.

Na página de um recurso, para além de encontrar a informação básica do mesmo, bem como todos os posts realizados sobre esse recurso. É possível ainda classificar, fazer o download do mesmo ou postar sobre o recurso. É possível ainda realizar o download de cada ficheiro separadamente, em vez do ficheiro completo.

Na sua página de perfil, um utilizador, por realizar o *logout* ou editar o perfil, podendo alterar a sua imagem, email, nickname e a sua afiliação.

Para além de criar um *post*, pode criar ainda um recurso, caso o utilizador tenha permissões para tal. E, se for um administrador, pode, na página de *management*, adicionar um tipo de recurso.

5 Conclusão

Com a realização deste projeto conseguimos aplicar vários dos conceitos abordados nas aulas da unidade curricular de Desenvolvimento e Aplicações Web, dando para consolidar bem estes conhecimentos fundamentais.

Ao longo do mesmo, deparamo-vos com algumas dificuldades, principalmente a nível da implementação da verificação do *bagit*. Todavia, na sua globalidade consideramos que conseguimos alcançar principais objetivos.