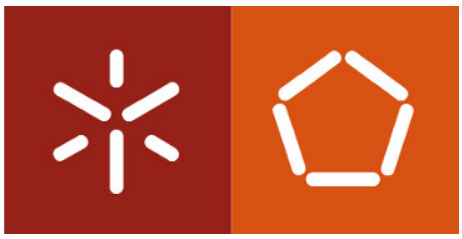
Desenvolvimento de Aplicações WEB

07 de Fevereiro de 2021

Trabalho Prático - Grupo 15

a83899	André Morais
a84802	João Abreu
a84485	Tiago Magalhães

PGR - Plataforma de Gestão e Disponibilização de Recursos Educativos



Mestrado Integrado em Engenharia Informática
Universidade do Minho

Conteúdo

1 Introdução

Neste trabalho prático, o objetivo era construir uma plataforma de gestão de recursos com as ferramentas e conhecimentos aprendidos ao longo do ano.

O PGR é uma aplicação onde se podem registar como produtores e dar upload dos mais variados recursos que pretenderem, ou se preferirem apenas ser um consumidor, onde não podem dar upload, mas têm acesso a qualquer conteúdo que esteja público, sendo possível o seu download.

Nestas próximas secções, aquilo que pretendemos é explicar cada um dos passos para a conceção deste projeto e para a sua estruturação.

2 Concepção/Desenho da resolução

Nesta secção será feita uma abordagem à arquitectura da aplicação, bem como às suas componentes e à forma como elas se relacionam.

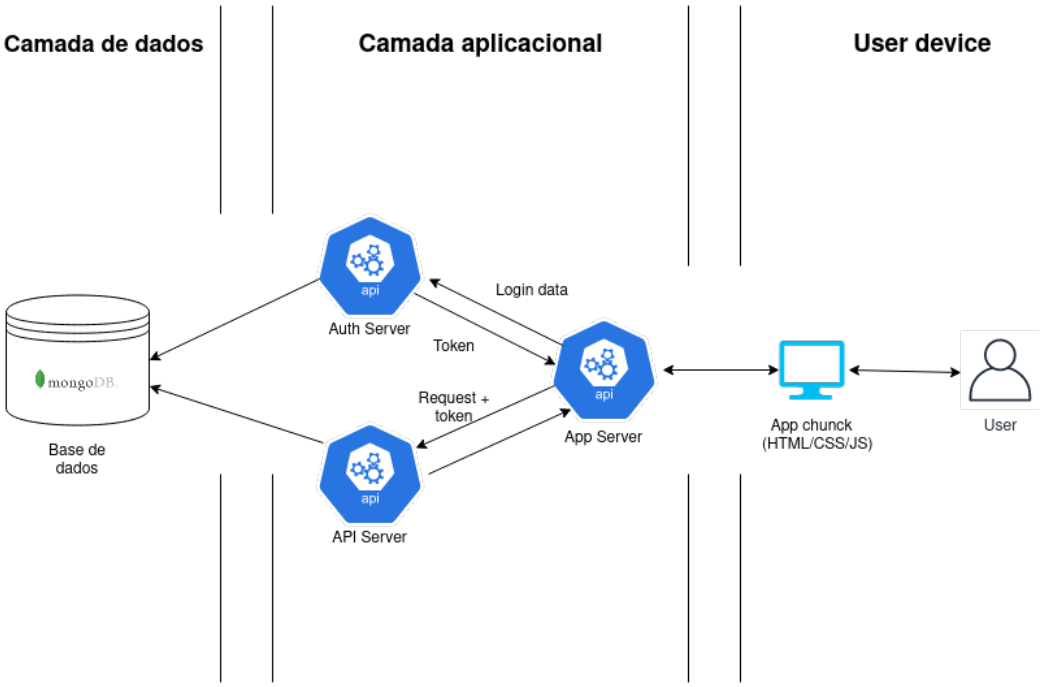


Figura 1: Modelo do Sistema

2.1 Descrição da arquitetura

Através do modelo do sistema podemos observar que existem 4 componentes principais sendo estas:

- **Auth Server**: Servidor responsável por receber os dados de *login* de um utilizador e com isto retornar-lhe um *token*, que irá permitir ao utilizador não ter a necessidade de estar sempre a autenticar-se, bem como a aplicação saber alguma informação do utilizador através do *token*.
- **API Server**: Servidor que assegura a comunicação entre o App Server e a base de dados, através de *requests* do App Server e o API Server

devolve informação acerca das principais coleções da base de dados.

- **App Server**: Servidor que fornece páginas dinâmicas ao cliente e que consome dados do API Server.
- **Base de dados**: Base de dados responsável por guardar dados acerca dos utilizadores, recursos, publicações, votos para o sistema de pontuação e comentários.

2.2 Tecnologias utilizadas

As tecnologias utilizadas foram as abordados nas aulas, tendo-se por isso utilizado Mongodb para a base de dados, NodeJs para a camada aplicacional com a *framework* Express, que otimiza a construção de aplicações *web* e API's. As páginas servidas pelo App Server foram construídas com o uso de PUG, CSS e JavaScript.

3 Camada Aplicacional

3.1 Auth Server

Neste servidor, são tratados apenas os pedidos de *login* e de registo, sem necessidade de *token*. Caso os dados de um *login* sejam corretos é enviado um *token*.

Para geração do *token* é usado o padrão *JSON Web Token* , sendo assinado usando um segredo privado que se encontra numa variável ambiente, presente no ficheiro de configuração *.env* .

3.2 Api Server

Neste servidor, todos os pedidos têm de possuir um *token* válido, para que os pedidos feitos possam ser atendidos. Aqui são também tratados os dados presentes na camada de dados, através dos controladores fornecendo por exemplo listas, paginação e informações sobre as coleções presentes na base de dados.

3.3 App Server

Além de fornecer páginas, será neste servidor que irão ficar armazenados os recursos e fotos de perfil dos utilizadores recebidas através da operação de *upload*.

3.3.1 Política de armazenamento

Quanto à política de armazenamento, na diretoria *public*, existe um diretoria *profilepics* onde irão ficar armazenadas as fotos de perfil de cada utilizador, casos os utilizadores optem por não carregar uma foto, será apresentada a imagem definida como *default.png*, caso contrário, esta irá ficar guardada com o seguinte formato: <*email_utilizador*.extensão>, uma vez que o email é único, é possível encontrar a partir deste a foto de perfil do utilizador.

Também existe dentro da pasta *public*, uma diretoria chamada *fileStore* onde irão ficar armazenados os recursos de cada utilizador, em que cada diretoria vai estar identificada pelo seu email, que por sua vez contém diretorias identificadas com o título de cada recurso que este possui. Nesta diretoria serão guardados os recursos extraídos do zip para que seja possível a prévisualização na página do recurso. Sendo assim, para pouparmos espaço e evitar repetição de informação, apagamos o ficheiro zip após o upload. No entanto, quando pretendemos fazer o download, temos de zippar o recurso de novo, fazer o download e voltar a apagar o zip.

3.3.2 Política de upload

Para que a operação de upload seja bem sucedida na nossa aplicação, o cliente tem de seguir um conjunto de regras. Apenas poderá dar upload de ficheiros **.zip**, e estes têm de ter uma determinada estrutura. De seguida podemos ver uma demonstração de upload de um recurso que possui 3 imagens.

Figura 2: Exemplo Zip Upload

Na imagem, temos um .zip chamado "demo"e dentro teríamos uma pasta chamada **data** com o/os ficheiro/os que representa o recurso e um ficheiro chamado **manifesto.json**. Neste ficheiro, temos um objeto JSON que apenas contém um campo chamado "ficheiros"que possui como valor um array com o path para cada ficheiro dentro da pasta "data". Podemos ver de seguida um exemplo deste ficheiro manifesto.json.

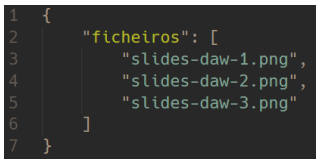


Figura 3: Exemplo Manifesto

No caso de haver subdiretorias, o manifesto terá o *path* para os ficheiros, para que seja possível a validação não só da existência do ficheiro como também do local onde está guardado. Para facilitar a criação do manifesto foi desenvolvido um *script* que gera automaticamente um ficheiro manifesto.json a partir da pasta "data"quando esta tem todos os ficheiros pretendidos para representar o recurso no upload. Este *script* pode ser encontrado no último capítulo: Anexos.

Sendo assim, para que o nosso upload seja válido, não só o zip tem de conter **apenas** um ficheiro manifesto.json e uma pasta "data", como também o conjuntos de ficheiros dentro da pasta data tem de estar corretamente representado no manifesto. Se houver ficheiros a mais ou a menos seja em que lado for o upload não será realizado.

Existem mais umas regras para fazer upload, sendo estas referentes ao tipo de recurso que o utilizador escolher. Podemos de seguida encontrar as regras que o grupo decidiu implementar para cada tipo.

```
Relatório - 1 pdf
Tese - 1 pdf
Artigo - 1 pdf
Aplicação - tudo
Slides - 1 pptx
Teste - 1 pdf ou [imagem]
Problema resolvido - 1 pdf ou [imagem]
Cartaz - [imagem]
```

Sendo assim, se por exemplo, existirem 2 pdfs na pasta "data"e o tipo for relatório, o upload não será realizado.

4 Modelo de dados

Para armazenamento de coleções na base de dados, e uma vez que estas são documentos JSON a sua representação é utilizada diretamente como *models* para a camada aplicacional. Para representação dos dados foram utilizados os seguinte modelos:

4.1 Comments

Modelo referente a cada comentário.

```
autorId: String,
text: String,
recursoId: String,
nome: String,
dataCriacao: String,
nomeId: String
```

4.2 Pub

Modelo referente a cada Publicação.

```
prodId: String,
prodName: String,
recursoTitulo: String,
recursoDescricao: String,
recursoTipo: String,
recursoId: String,
dataCriacao: String
```

4.3 Recurso

Modelo referente a cada Recurso.

```
tipo :  String,
titulo: String,
descricao: String,
subtitulo: String,
dataCriacao: String,  // Data quando recurso criado
dataRegisto : String, // Data de registo na aplicação
visibilidade: String, // Publico,Privado
hashtags: [String],
rating: Number,
numVotantes: Number,
autor: String,
nome: String,
autorID:
```

4.4 User

Modelo referente a cada Utilizador.

```
nome: String,
email: String,
filiacao: String,
nivel: String,
dataRegisto: String,
dataUltimoAcesso: String,
password: String
```

4.5 Voto

Modelo referente a cada Classificação de um recurso.

```
recursoID : String,
userID: String,
rating: Number
```

5 Conclusão

No paradigma da evolução das nossas capacidades como estudantes de engenharia informática, a realização deste projeto permitiu-nos consolidar a aprendizagem da UC de Desenvolvimento de Aplicações WEB, mais concretamente, os métodos e ferramentas que devem ser aplicados no que toca à resolução do problema de operar uma aplicação que correrá nos nossos *browsers*.

Com o desenvolvimento desta aplicação fomos capazes de criar uma tipologia Full Stack (*front-end* + *back-end*).

Apesar de considerarmos que tivemos sucesso no desenvolvimento da nossa aplicação, consideramos que existem aspetos nos quais poderíamos melhorar em iterações futuras, nomeadamente melhoramento na interface e validação dos *zips* (no manifesto apenas temos a lista de *paths* para cada ficheiro).

6 Anexos

```python
import os
import json
folder = "data/"
paths = []
for root, directories, filenames in os.walk(folder):
    for filename in filenames:
        s = os.path.join(root,filename)
        s = s.replace(folder,"")
        s = s.replace("\\","/")
        paths.append(s)

data = {}
data["ficheiros"] = paths
with open('manifesto.json', 'w+', encoding='utf-8') as
   outfile:
    json.dump(data, outfile,indent=4,ensure_ascii=False)
print("Ficheiro manifesto.json foi criado com sucesso.")
```

Listing 1: generateManifest.py