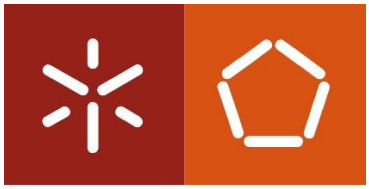


Universidade do Minho

Departamento de Informática

Gestor de Recursos Educativos

Desenvolvimente de Aplicações WEB/ Processamento e Representação de Informação

Angélica Freitas (A83761) e Rodrigo Pimentel (A83765)

7 de fevereiro de 2021

Conteúdo

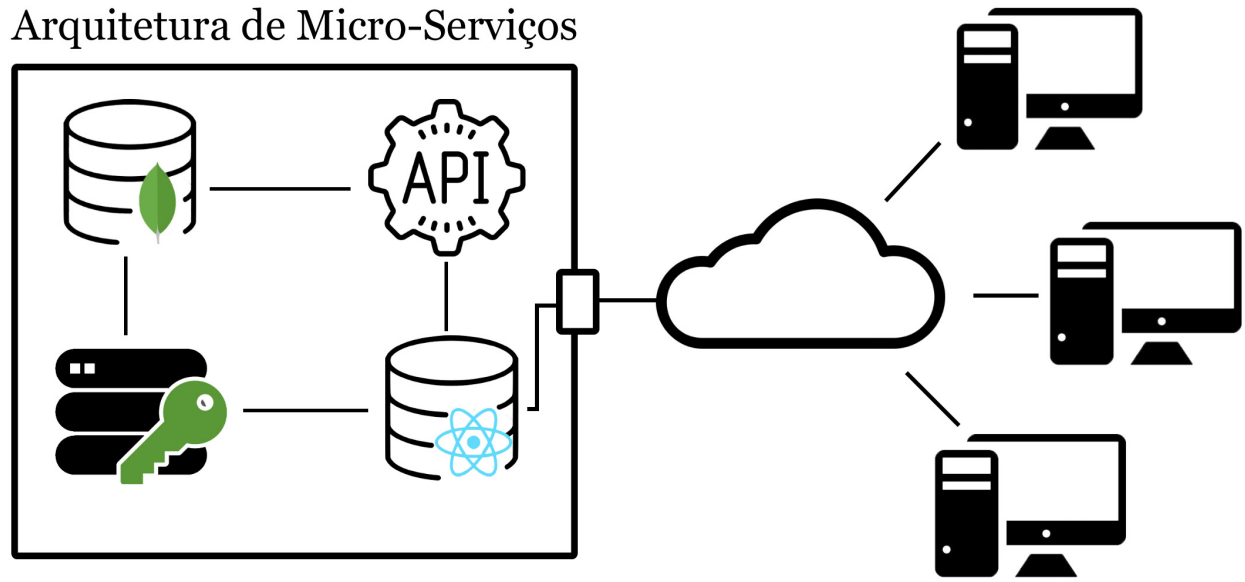
Capítulo 1

Desenvolvimento

1.1 Estrutura

Como podemos ver pela imagem de seguida, a nossa aplicação possui 4 servidores: servidor de MongoDB para a persistência de dados; servidor API que serve como ponte entre o servidor aplicacional e os dados da aplicação; servidor de autenticação que tem como trabalho gerir os utilizadores da aplicação e garantir a segurança e acesso restrito a determianda áreas da aplicação e, por fim, o servidor aplicacional de React que irá servir a literal aplicação ao cliente (Client-Side-Rendering).

Os micro-serviços são uma arquitetura e uma abordagem para escrever programas de software. Com eles, as aplicações são desmembradas em componentes mínimos e independentes. Diferentemente da abordagem tradicional monolítica em que toda a aplicação é criada como um único bloco. Sendo assim, esta foi uma abordagem essencial para a realização deste projeto.



1.2 Servidores

1.2.1 API

Para a implementação deste servidor, recorremos à estrutura básica fornecida pelo *express* e, através do *mongoose*, este vai então fazer a conexão ao servidor *mongo*. Para tal, foram criados *Controllers* e *Models* associados às coleções que serão mencionadas na secção *MongoDB*.

Modelos

Assim, apresenta-se de seguida os modelos realizados.

- Recurso

```
var mongoose = require('mongoose')

var recursoSchema = new mongoose.Schema({
    "titulo": {type: String, required: true},
    "tipo" : {type: String, required: true},
    "dataDeCriacao": {type: String, required: true},
    "pasta": {type: String, required: true},
    "ficheiros": {type: [String], required: true},
    "autor": {type: String, required: true},
    "classificadores": [{user:String, classificacao: {type: Number, default: 0}}],
    "imagem": String,
    "acesso": {type: String, required: true},
    "estado": {type: String, default: 'Em avalia  o'},
    "comentarios": [{_id: mongoose.Types.ObjectId,autor: String, dataDeCriacao:
        String, conteudo: String, comentarios: [{_id: mongoose.Types.ObjectId,autor
        : String, dataDeCriacao: String, conteudo: String}]}],
    "descricao" : String
});

module.exports = mongoose.model('recursos', recursoSchema)
```

Como cada modelo tem a mesma estrutura (atributos, nome de coleção e schema serão, obviamente, diferentes), apresentamos de seguida todas as informações para os restantes modelos em tabelas.

Dado	Tipo	Required
titulo	String	Sim
conteudo	String	Sim
imagem	String	Não
dataDeCriacao	String	Sim
autor	String	Sim
href	String	Não

Tabela 1.1: Modelo *Noticia*

Dado	Tipo	Required
tipo	String	Sim
dataDeCriacao	String	Sim
remetente	String	Sim
destinatario	String	Sim
descricao	String	Sim
href	String	Não

Tabela 1.2: Modelo *Notificacao*

Dado	Tipo	Required
recurso_id	ObjectId	Sim
dataDeCriacao	String	Sim
estado	String	Sim

Tabela 1.3: Modelo *Pedido*

Dado	Tipo	Required
tipo	String	Sim

Tabela 1.4: Modelo *TipoRecurso*

É de salientar que para cada um dos modelos existe o atributo *_id* gerado pelo mongo. Para todos estes modelos, existe um *Controller* associado a esses que tratam de fazer pedidos ao servidor *Mongo*. Estes realizam não só pedidos *CRUD* (*Create, Retrieve, Update e Delete*), mas também outros que irão melhorar o fluxo da aplicação. Algumas das principais bibliotecas usadas são:

- **Mongoose** - Comunicação com o servidor de *Mongo*.
- **jwt** - Manipulão de *JsonWebTokens*.
- **cors** - *Middleware* de *handle* de *cors*.
- **multer** - *Middleware* de manipulação de ficheiros.

Para melhor compreensão destes, explicamos de seguida cada rota associada aos pedidos nos *Controllers*.

Rotas

Como todas as rotas terão um padrão do género /api/____, utilizamos um middleware do género app.use('/api', indexRouter), para acrescentar o prefixo *api*.

Rotas relativas a *Recurso*.

- **GET /recursos/ficheiro** Executa download do(s) ficheiro(s) dependendo de uma *folder* e um ficheiro na *query string*.
- **GET /recursos** Se a query string for um id, então retorna o recurso com esse id. Caso haja query string com email, retorna todos os recursos de um autor Caso contrário, retorna todos os recursos. Estes dois últimos são ordenados pela data de criação.
- **GET /recursos/publicos** Retorna todos os recursos cujo acesso seja público e o estado seja confirmado.
- **POST /recursos** Insere um recurso. Se o acesso deste é Público, então é inserida uma notícia. Também, se o mesmo tiver em "Em avaliação"será criado um pedido.
- **POST /recursos/comment/:id_recurso** Adiciona um comentário de primeiro nível a um recurso passado como parâmetro e comentário por body.
- **POST /recursos/sub_comment** Adiciona um comentário de segundo nível através do id do recurso e comentário (primeiro nível) na query string e conteúdo no body.
- **POST /recursos/classificador** Adiciona um classificador dependendo de um recurso (query string)

- **PUT /recursos/classificador** Atualiza a classificação de um classificador
- **PUT /recursos/:id** Atualiza parâmetros de um dado recurso (parâmetro) e campos para atualizar no body. Se o acesso deste mudar para público, será publicada uma notícia relativa a este. Isto é, insere-se uma notícia.
- **DELETE /recursos/:id** Remove um recurso pelo seu identificador (parâmetro). Este vai apagar também todas as notícias e pedidos relacionados ao mesmo.

Rotas relativas a *Pedido*.

- **GET /pedidos** Retorna todos os pedidos.
- **POST /pedidos** Insere um pedido (body).
- **PUT /pedidos/:id** Atualiza um pedido através do id passado nos parâmetros e campos a atualizar no *body*. É também inserida uma notificação dependendo do seu novo estado.
- **DELETE /pedidos/:id** Remove um pedido pelo seu identificador (parâmetro).

Rotas relativas a *Noticia*.

- **GET /noticias** Retorna todas as notícias ordenadas descendentemente pela data de criação.
- **GET /noticias/:id** Retorna uma notícia pelo id (parâmetro).
- **POST /noticias** Insere uma notícia (body).
- **PUT /noticias/:id** Atualiza uma notícia pelo id (parâmetro) e dados no *body*.
- **DELETE /noticias/:id** Remove uma notícia pelo seu identificador (parâmetro).

Rotas relativas a *Notificacao*.

- **GET /notificacoes** Dá *retrieve* das notificações ordenadas descendentemente pela data de criação.
- **GET /notificacoes/:email** Retorna as notificações ordernadas descendentemente pela data de criação de um remetente (parâmetro).
- **POST /notificacoes** Insere uma notificação (*body*).
- **DELETE /notificacoes/:id** Remove uma notificação pelo seu identificador (parâmetro).

Rotas relativas a *TipoRecurso*.

- **GET /tiporecursos** Retorna todos os tipos de recursos.
- **POST /tiporecursos** Insere um TipoRecurso (*body*).
- **DELETE /tiporecursos/:id** Remove um TipoRecurso pelo seu identificador (parâmetro).

1.2.2 Autenticação

O servidor de autenticaçao é o responsável por fornecer *tokens JWT* de autenticação e criar, editar e apagar utilizadores. Para aceder ao servidor de autenticação é necessário um *token JWT* especial assinado com uma chave aplicacional. É importante notar que o servidor de autenticação utiliza chaves simétricas.

Para o cliente poder aceder ao *API* de dados, este precisará de um *token* fornecido pelo servidor de autenticação, enviado após realizar login. Cada *token* tem duração de 2 horas. Depois desse tempo, o cliente terá de submeter um novo formulário de login.

Tal como o *API*, o servidor de autenticação está protegido por um *middleware* específico criado manualmente. Este código procura a existência de um *token* no *header* ou no *url* de um pedido e valida o mesmo. O código seguinte mostra essa implementação:

```
app.use((req, res, next) => {

  if(req.headers.authorization) {
    let token = req.headers.authorization.split(' ')[1]

    jwt.verify(token, process.env.JWT_SECRET, (err, _) => {
      if(err){
        res.status(401).jsonp({erro: err})
        return
      }
      next()
    })

  } else if(req.query.token){

    jwt.verify(req.query.token, process.env.JWT_SECRET, (err, _) => {
      if(err){
        res.status(401).jsonp({erro: err})
        return
      }
      next()
    })

  } else {
    res.status(401).jsonp({erro: "Authorization header not set."})
  }

})
```

Algumas das principais bibliotecas usadas:

- **Mongoose** - Comunicação com o servidor de MongoDB (Users).
- **jwt** - Manipulão de JsonWebTokens.
- **Passport** - Torna a autenticação de utilizadores simplificada.
- **express-session** - Middleware de Sessões.
- **uuid** - Gerador de Id's aleatórios e únicos.
- **multer** - *Middleware* de manipulação de ficheiros.

As rotas disponíveis para este servidor (todas protegidas por uma chave aplicacional) todas com o prefixo de `users/` são:

Rotas relativas ao servidor Autenticacao

- **GET /** Retorna todos os *User*.
- **GET /:email** Retorna um *User* dependendo do seu email (parâmetro).
- **GET /logout** Responsável por efetuar o logout e destruir a sessão associada.

- **POST /** Insere um *User* cujos dados são passados pelo *body*.
- **POST /login** Responsável por criar um *token JWT* se as credenciais forem válidas.
- **PUT /:email** Atualiza um *User* identificado pelo email (parâmetro) cujos campos a atualizar se encontram no *body*.
- **DELETE /:email** Remove um *User* identificado pelo email (parâmetro).

1.2.3 Aplicacional

O servidor aplicacional, criado através da tecnologia React, servirá como *endpoint* principal do cliente. O cliente pedirá o conteúdo aplicacional a este servidor que, por sua vez, irá correr no lado do cliente, *Client-Side-Rendering*. Através desta tecnologia, conseguimos alcançar uma elevada fluidez e velocidade de interação com os utilizadores.

Muito resumidamente, a nossa aplicação possui 4 páginas principais: a página inicial de login; a página do administrador; a página do utilizador e a página do creador. Cada página contém as suas respetivas sub páginas de relevância, como por exemplo: Perfil, Notícias, Notificações, Recursos, entre outros.

1.2.4 MongoDB

Já no servidor de *mongo* foi criado uma base de dados denominada de GRE que, por sua vez, terá as seguintes coleções: noticias, notificacoes, pedidos, recursos, users e tipo recursos.

Como se trata de um modelo documental, tecnicamente a estrutura dos dados é livre. Porém, como mencionado na secção *API* e *Autenticação*, cada objeto a ser inserido numa das coleções acima vistas, têm uma estrutura que têm de seguir. Isto é, há dados específicos requeridos para a inserção dos mesmos.

Para a população da coleção *tiporecursos*, inserimos manualmente os seguintes dados.

```
1  db.tiporecursos.insert({tipo: "Teste"});
2  db.tiporecursos.insert({tipo: "Tese"});
3  db.tiporecursos.insert({tipo: "Trabalho de Grupo"});
4  db.tiporecursos.insert({tipo: "Grava  o de aula"});
5  db.tiporecursos.insert({tipo: "Slides Te ricos"});
6  db.tiporecursos.insert({tipo: "Fichas"});
7  db.tiporecursos.insert({tipo: "Resolu  es"});
8  db.tiporecursos.insert({tipo: "Classifica  es"});
9  db.tiporecursos.insert({tipo: "Hor rios"});
10 db.tiporecursos.insert({tipo: "Formul rios"});
11 db.tiporecursos.insert({tipo: "Apontamentos"});
12 db.tiporecursos.insert({tipo: "Livros"});
13 db.tiporecursos.insert({tipo: "Exame"});
14 db.tiporecursos.insert({tipo: "Slides Te rico-Pr ticos"});
```

Capítulo 2

Webgrafia

https://stackoverflow.com/
https://react-bootstrap.github.io/
https://react-icons.github.io/react-icons/
https://reactrouter.com/
https://mongoosejs.com/docs/api.html
https://docs.mongodb.com/