Universidade do Minho

Mestrado em Engenharia Informática

Representação e Processamento de Conhecimento na Web

Trabalho Prático

Angélica Cunha - PG47024

Duarte Oliveira - PG47157

Tiago Barata - PG47695

Junho 2022

Resumo

Este documento diz respeito ao trabalho prático proposto na unidade curriculare de Representação e Processamento de Conhecimento na Web da Universidade do Minho. O objetivo deste projeto consiste no desenvolvimento e implementação de uma aplicação *web* para um repositório de recursos didáticos. Este projeto surgiu pela necessidade diária que um professor tem de publicar conteúdos de forma organizada para os seus alunos.

Conteúdo

1 Introdução

A aplicação *Web* desenvolvida permite ao utilizador comum ter acesso direto a qualquer recurso didático em qualquer momento e lugar (desde que haja internet).

O presente repositório RRD (Repositório de Recursos Didáticos) foi desenvolvido segundo as orientações do modelo OASIS, desta forma conseguimmos distinguir 3 tipos de atores do sistema: administrador, produtor e utilizador. Sendo que o administrador terá acesso a todas as funcionalidades do sistema e poderá efetuar a sua manutenção, o produtor poderá depositar recursos no sistema assim como visualizar os recursos já disponíveis e o utilizador é capaz de consultar, pesquisar informação e descarregar informação selecionada.

Neste presente relatório apresenta-se a conceção e o desenho da solução. Aqui é apresentada a forma de armazenamento de toda a informação na base de dados, nomeadamente os esquemas das coleções usadas. É também especificada a forma de autenticação na aplicação, apresentando os requisitos necessários e a forma de implementação. De seguida são apresentadas as permissões e funcionalidades de cada tipo de utilizador, assim como os requisitos definidos para cada um.

2 Conceção e Desenho da Solução

2.1 Base de dados

De forma a armazenar todos os dados do repositório é necessário recorrer a uma base de dados. No caso do **RRD** vamos recorrer a uma base de dados **MongoDB**, onde foram definidas as coleções necessárias que vão suportar as operações de leitura e escrita de dados necessárias ao funcionamento da aplicação.

Para aceder à base de dados 'RRD' e a todas as coleções foi utilizada a ferramenta Mongoose.

```
1 mongoose.connect('mongodb://127.0.0.1:27017/RRD',
2       { useNewUrlParser: true,
3         useUnifiedTopology: true,
4         serverSelectionTimeoutMS: 5000});
```

Listing 1: Conexão à base de dados feita em app.js

2.1.1 Esquemas das Coleções

Os utilizadores fazem login na aplicação através do username e da palavra-passe. Estes podem assumir diferentes papéis na aplicação, o de administrador (*level* 2), produtor (*level* 1) e utilizador (*level* 0).

```
1 var userSchema = new mongoose.Schema({
2     username: { type: String, required: true, unique: true },
3     password: { type: String, required: true },
4     level: Number
5   });
```

Listing 2: Esquema dos utilizadores

Também para os recursos foi criada uma coleção para guardar toda a informação necessária. Quanto ao tipo de recurso, este pode ser slides, exame, manual, relatório ou tese, sendo distinguidos na BD por um identificador numeral.

```
1 var ficheiroSchema = new mongoose.Schema({
2     creation_date: Date,
3     submission_date: Date,
```

```
    producer: String, //nome de quem fez o ficheiro
    user: String, //nome de quem submeteu o ficheiro
    desc: String,
    type: Number,
    name: String,
    path: String,
    mimetype: String,
    size: Number,
    comments:[{
        from: String,
        date: Date,
        content: String,
        rating: Number
    }]
})
```

Listing 3: Esquema dos resursos

Para as notícias a aparecer na página principal também foi criada uma coleção. São permitidas notícias de 3 tipos, notícias escritas por o utilizador e geradas automáticamente quando alguém adiciona ou comenta um recurso.

Esta distinção é feita pela *action*, sendo que *action* = 0 é uma notícia de inserção de um recurso e tem os campos: *user*, *date*, *rec_id*, *rec_name* e *active*; *action* = 1 é uma notícia de novo comentário em um recurso e tem os campos: *user*, *date*, *rec_id*, *rec_name*, *comment*, *rating active*; *action* = 2 é uma notícia livre escrita por um utilizador e tem os campos: *user*, *date*, *free* e *active*;

```
var newSchema = new mongoose.Schema({
    user:{type:String,required:true},
    date:{type:String,required:false},
    action:{type:Number,required:true},
    rec_name:{type:String,required:false},
    rec_id:{type:String,required:false},
    active:{type:Number,required:true},
    comment:{type:String,required:false},
    rating:{type:Number,required:false},
    free:{type:String,required:false}
})
```

Listing 4: Esquema das notícias

Por fim, temos um esquema para os logs do sistema, estes são do tipo *login*, *logout*, *visualização* de um recurso e *Download* de um recurso.

```
var logSchema = new mongoose.Schema({
    user_id:{type:String,required:true},
    username:{type:String,required:true},
    action:{type:String,required:true},
    date:{type:String,required:false},
    recurso:{type:String,required:false}
})
```

Listing 5: Esquema dos Logs

2.2 Autenticação

A aplicação deverá possibilitar que os utilizadores se autentiquem e tenham acesso às funcionalidades disponibilizadas para estes. Vamos utilizar como credenciais de autenticação o *username* e a *password* de cada utilizador.

2.2.1 Requisitos

A aplicação deve ser capaz de permitir que:

- Qualquer utilizador faça login com as suas credenciais;
- Qualquer utilizador termine sessão da sua conta, caso estiver autenticado;
- Seja restringido o acesso a determinadas funcionalidades, dependendo do tipo de utilizador.

2.2.2 Implementação

```
app.use(require("express-session")({
  secret:"RPCW2022",
  resave: false,
  saveUninitialized: false
}));
```

Listing 6: App.js definição do segredo

Sempre que um utilizador faz login na aplicação um JSON Web Token é gerado com informação relativa ao utilizador em *payload* e este é guardado nos cookies.

```
router.post('/login', async function(req, res, next) {
  let username = req.body.username
  let password = req.body.password
  if (typeof username === "undefined" && !username)
            throw new Error ("Username n o definido")
  if (typeof password === "undefined" && !password)
            throw new Error ("Password n o definida")

  user = await User.validatePassword(username, password)
  if(user){
    var token = jwt.sign({ user_id: user._id, username: user.
    username,
       level:user.level, expiresIn: '1h'}, 'RPCW2022')

    Log.insert({user_id:user._id,username:user.username,
    action:'login',recurso:'---'})

    res.cookie('token', token, {
    expires: new Date(Date.now() + '1h'),
    secure: false, // set to true if your using https
    httpOnly: true
    }).status(200).redirect('/home')
  }
  else{
    res.redirect('/')
  }
});
```

Listing 7: Geração token

Após este passo, com o token gerado, protegemos todas as rotas com a função *authorization* que verifica a existência e validade do token e guarda a informação do user em request de forma a poder ser acedida para verificação do nível de utilizador. Para esta verificação criamos a função *checkLevel* que verifica que o utilizador que está a tentar aceder à rota tem permissões para tal. Desta forma, todos os pedidos tem o seguinte aspeto:

```
router.get('/umaRota',Auth.authorization,Auth.checkLevel(
   nivel_autorizado_a_aceder), function(req, res){
...
}
```

Listing 8: Geração token

Para pedidos *Axios* o token é enviado no *header* da seguinte forma:

```
1 router.get('/umaRota',Auth.authorization,Auth.checkLevel(
    nivel_autorizado_a_aceder), function(req, res){
2   axios.get('http://localhost:8000/api/rota',{
3     withCredentials: true,
4     headers: {
5         'Authorization': req.cookies.token
6     }
7     })
8     ...
9 })
```

Listing 9: Geração token

3 Tipos de Utilizadores

3.1 Administrador

Como o próprio nome indica é o administrador do sistema. Irá executar acções relacionadas com a manutenção do mesmo e terá acesso a todas as operações da aplicação desenvolvida.

3.1.1 Requisitos

O administrador, que tem o papel de gerir a informação da aplicação, deve ser capaz de:

- Consultar, editar e remover os utilizadores do sistema;
- Adicionar qualquer tipo de utilizador ao sistema, apenas uma dministrador pode adicionar outro;
- Consultar e exportar os logs do sistema;
- Consultar, descarregar, editar e remover recursos presentes no repositório;
- Adicionar recursos ao repositório;
- Visualizar, editar, remover e tornar visível/invisível notícias;
- Inserir e eliminar comentários e classificação.
- Filtrar as notícias por ativas, inativas ou todas.

3.2 Produtor

O produtor será o professor, neste caso, e poderá inserir recursos, assim como outras funcionalidades inerentes a esta.

3.2.1 Requisitos

O produtor deve ser capaz de:

- Consultar e descarregar recursos presentes no repositório;
- Adicionar recursos no repositório;
- Editar e eliminar recursos que este adicionou ao repositório;

- Adiconar notícias;
- Editar ou eliminar notícias que este adicionou;
- Adicionar comentários e classificação;
- Editar e eliminar comentários que este adicionou.
- Editar informações da sua conta, excepto o seu nível de utilizador.

3.3 Utilizador

O utilizador comum será o aluno, neste caso, e poderá apenas consultar e descarregar recursos disponíveis no repositório.

3.3.1 Requisitos

O utilizadir comum deve ser capaz de:

- Consultar e descarregar recursos presentes no repositório;
- Comentar e classificar recursos;
- Adicionar uma notícia;
- Editar informações da sua conta, excepto o seu nível de utilizador.

4 Ingestão de Sip

Aquando da reinicialização do servidor, após estabelecida a conexão com o Mongo, é verificado se existe um sip.zip caso exista, este é aberto, os ficheiros que lá se encontrarem são carregados na base de dados e a pasta uploads que contém os ficheiros carregados no filesystem é a pagada e substituída pelos ficheiros presentes no sip. Caso ainda não exista um ficheiro sip.zip, este é criado com uma cópia do RRD-SIP.json encontrado na pasta uploads e com os respetivos recursos presentes.

Sempre que um ficheiro é adicionado, este é adiconado a metainformação deste é adcionada ao RRD-SIP.json local e o ficheiro é adiconado à pasta uploads numa pasta com o username do utilizador que o adicionou. De seguida, o sip é criado novamente de forma a comtemplar este novo ficheiro. O processo é análogo para a edição e remoção de um ficheiro.

5 Conclusão e Trabalho futuro

Ao longo deste relatório está exposto todo o trabalho realizado para este projeto, assim como as decisões tomadas na implementação do código.

Consideramos que o resultado final positivo, além de haver muito por onde melhorar tanto a nível de *backend* como de *frontend*. Como trabalho futuro o grupo gostaría de melhorar em ambos os aspetos.