

## TRABALHO

### Relatório

**João Luís Gomes Jardim**

Nº 2087819

**Artur Pereira**

Nº 2040415

**Curso Técnico Superior Profissional em Tecnologias  
e Programação de Sistemas de Informação**

**UNIDADE CURRICULAR:**

Desenvolvimento web – Back-End

**DOCENTE:**

David Jardim

**DATA:**

16 de junho de 2020

# ESCOLA SUPERIOR DE TECNOLOGIAS E GESTÃO

Cofinanciado por:



---

# ÍNDICE

---

<b>Introdução</b>	3
<b>Ficheiros de configuração e conexão à BD</b>	4
<b>Model View Controller</b>	5
<i>Modelos do Projeto</i>	5
<i>View do Projeto</i>	5
<i>Controladores do Projeto</i>	8
<b>Acesso aos EndPoints</b>	15
<i>Acesso aos endpoints do catController</i>	16
<i>Como ter acesso aos end points do catController:</i>	16
<i>Acesso aos endpoints do userController</i>	16
<i>Como ter acesso aos end points do userController:</i>	17
<i>Acesso aos endpoints do uploaderController</i>	17
<i>Como ter acesso aos end points do uploaderController:</i>	17
<i>Acesso aos endpoints do uploadController</i>	19
<i>Como ter acesso aos end points do uploadController:</i>	19
<b>Estruturação de pastas</b>	20
<b>Processos utilizados de desenvolvimento</b>	20
<b>Conclusão</b>	21

---

# INTRODUÇÃO

---

Este projeto foi um trabalho proposto pelo professor David Jardim no âmbito da unidade curricular Desenvolvimento web – Back-End.

Neste projeto foi nos proposto que o back-end do projeto fosse desenvolvido num servidor express que respeite a arquitetura REST e que seja respeitado também o padrão de desenho MVC.

Para a utilização do padrão de desenho MVC, respetivamente aos Models nós utilizamos a ORM Sequelize uma vez que foi solicitado no enunciado do projeto a utilização da mesma, para os Views utilizamos o EJS uma vez que achamos que era o mais pratico de utilizar uma vez que já tínhamos utilizado o mesmo durante o percurso de ensino e os Controllers foi criado 4 controladores que são utilizados para gerir os nossos pedidos e para efetuar pedidos assim como verificações na nossa base de dados e por fim também foi indicado no enunciado deste projeto que seria necessário uma criação de um sistema de autenticação entre cliente e serviço nos utilizamos o JSON Web Tokens como solicitado no enunciado.

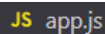
Este trabalho como um todo tinha como objetivos gerais a consolidação de conhecimentos adquiridos durante todo o semestre assim como aumentar a nossa capacidade de análise de problemas identificar e aplicar vários modelos e técnicas utilizadas no desenvolvimento de uma WEB API para o servidor.

---

# FICHEIROS DE CONFIGURAÇÃO E CONEXÃO À BD

---

No `app.js` é onde definimos tudo o que será necessário para usar no nosso projeto como por exemplo as rotas que posteriormente são chamadas para solicitar os endpoints, assim como o nosso engine de front end entre outros aspetos que são necessário para o servidor.



`app.js`

As configuração do nosso servidor estão no ficheiro `.env`.

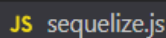


`.env`

Consta toda a informação que será utilizada para aceder a base de dados nomeadamente o `DB_SCHEMA`, `DB_USER`, `DB_PASS`, `DB_HOST` e ainda o token secret que posteriormente será utilizado.

```
DB_SCHEMA=projeto_backend
DB_USER=root
DB_PASS=''
DB_HOST=localhost
|
TOKEN_SECRET=24f8e2660e6f65f2cfd9bee2dfecb574f79784606f10efe25a793d3de34ae1155049d01161a7997c8652537ed8cac69fcf764065f8b7241fd82039c87bc3674
```

Para a ligação a nossa base de dados temos o ficheiro `sequelize.js`.



Este é o ficheiro que têm a informação sobre a base de dados em que importa os modelos que foram definidos e posteriormente exporta a informação para que a mesma seja utilizada nos controladores.

```
const Sequelize = require('sequelize');
const PersonModel = require('./models/uploader');
const ImageModel = require('./models/image');

const sequelize = new Sequelize(process.env.DB_SCHEMA, process.env.DB_USER, process.env.DB_PASS, {
  host: process.env.DB_HOST, // vai buscar a informação ao .env
  dialect: 'mysql', // engine da base de dados
  pool: {
    max: 10,
    min: 0,
    acquire: 30000,
    idle: 10000
  }
});

const User = PersonModel(sequelize, Sequelize);
const Images = ImageModel(sequelize, Sequelize);

// Adicionar o modelo à BD
sequelize.sync({ force: true })
  .then(() => {
    User.bulkCreate([ // criação dos dados para a base de dados
      { email: "admin@gmail.com", password: "admin" },
    ])
    .then(() => {
      return User.findAll();
    })
  });

module.exports = {
  User,
  Images
}
```

`sequelize.js`

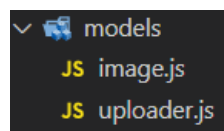
---

# MODEL VIEW CONTROLLER

---

## Modelos do Projeto

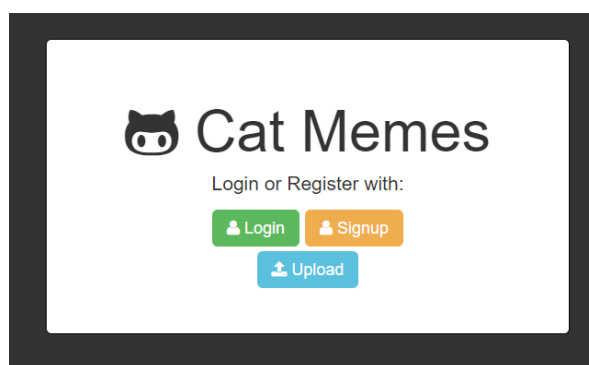
Para a criação das nossas tabelas da base de dados nos criamos dois modelos, o modelo “**uploader.js**” em que o utilizador ao criar uma nova conta fica registado o seu ID, o email e password e criamos também o modelo “**image.js**” em que após o utilizador ao introduzir uma ou mais imagens é guardado na nossa base de dados o ID, o nome original da imagem, a localização onde foi guardada a imagem, o nome da imagem mais a data atual em unix time de forma a que nunca possa existir imagens com o mesmo nome e ainda guardamos o user id do utilizador como uma CHAVE ESTRANGEIRA de forma a posteriormente quando formos chamar os endpoints conseguirmos verificar quem terá inserido as imagens.



Models

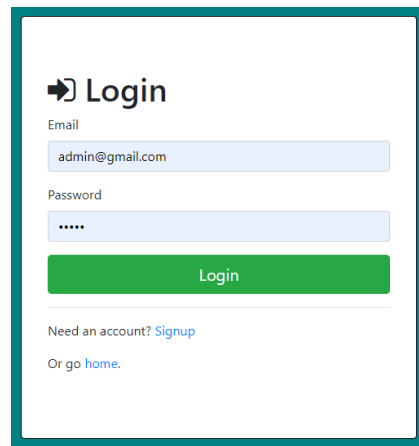
## View do Projeto

Foi criado o simples layout de forma a que possa ser testada todas as nossas funcionalidades. Posteriormente vamos também explicar todo o back-end que se encontra por tras do front-end como por exemplo validações de contas entre outras situações.

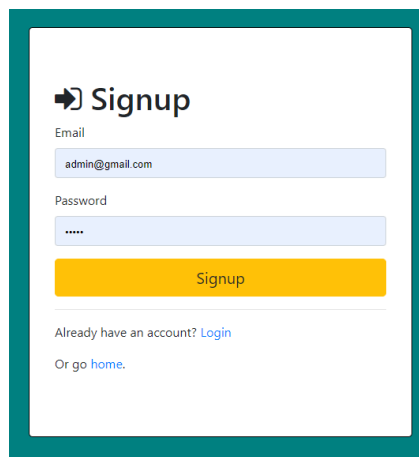


index.ejs

A login page é o local onde o utilizador pode efetuar o login se já tiver uma conta criada, caso exista a necessidade de criar a conta têm a possibilidade de efetuar a criação de uma conta na opção Signup.

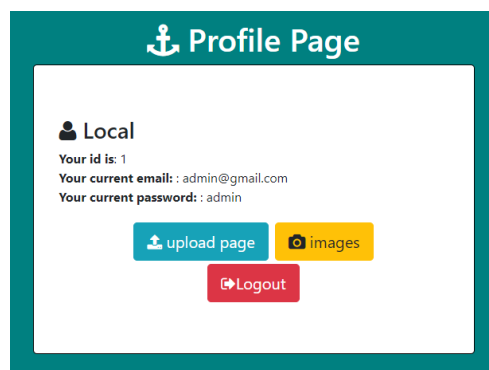
The screenshot shows a web form titled "Login" with a right-pointing arrow icon. It contains two input fields: "Email" with the value "admin@gmail.com" and "Password" with masked characters "\*\*\*\*\*". Below the fields is a green "Login" button. At the bottom, there are two links: "Need an account? Signup" and "Or go home."

login.ejs

The screenshot shows a web form titled "Signup" with a right-pointing arrow icon. It contains two input fields: "Email" with the value "admin@gmail.com" and "Password" with masked characters "\*\*\*\*\*". Below the fields is a yellow "Signup" button. At the bottom, there are two links: "Already have an account? Login" and "Or go home."

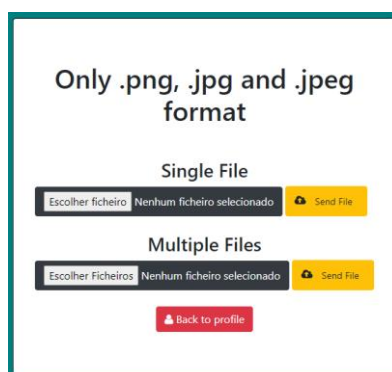
Signup.ejs

Após o utilizador efetuar o login ou criar conta o mesmo é redirecionado para a pagina do profile em que têm várias opções que pode utilizar.

The screenshot shows a web page titled "Profile Page" with an anchor icon. Under the heading "Local", it displays user information: "Your id is: 1", "Your current email: : admin@gmail.com", and "Your current password: : admin". Below this information are three buttons: "upload page" (blue), "images" (yellow), and "Logout" (red).

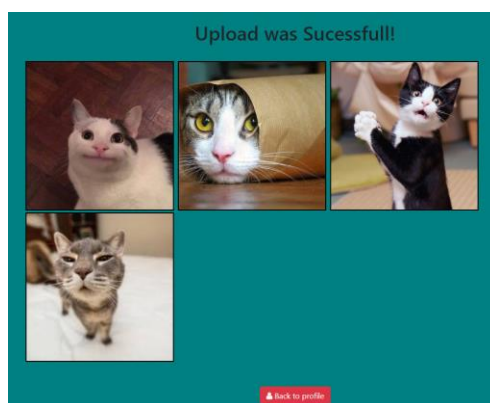
Profile.ejs

Se o utilizador pretende efetuar o upload de uma ou mais imagens têm essa possibilidade criamos também um layout basico para o intuito dessa situação.



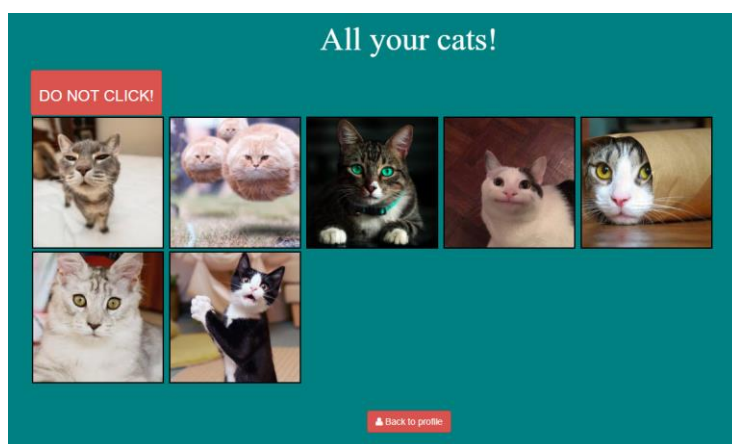
Upload.ejs

Após introdução de uma ou varias imagens de forma a que o utilizador também tenha conhecimento que as suas imagens foram introduzidas com sucesso redirecionamos novamente o utilizador para uma pagina que mostra as imagens introduzidas.



upload\_sucess.ejs

Como pode verificar anteriormente na pagina do profile existia um botão com o nome imagens, o objetivo desse botão é após o utilizador ter uma ou mais imagens com o seu user conseguir consultar todas as imagens que têm no seu profile, posteriormente também será explicado o motivo do botão “DO NOT CLICK”.



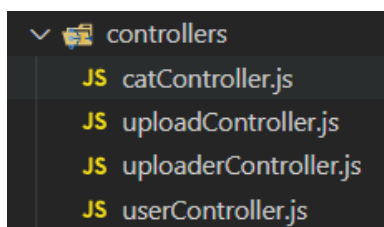
images.ejs

De forma a renderizar as imagens que estão na base de dados, quando é efetuado o upload de uma foto nos gravamos a localização da foto na BD, posteriormente no backend quando o utilizador faz um pedido para visualizar as suas imagens é criado um GET que vai buscar todas as imagens do utilizador, puxa a localização dessas imagens para um array, envia a localização para o front end e depois no front end chamamos a localização dessa foto com um ciclo for para renderizar todas as fotos que estão dentro desse array.

```
<div class="container">
  <% for (var i = 0; i < locations.length; i++) { %>
    <div class="text-center testing">
      
    </div>
  <% } %>
</div>
```

## Controladores do Projeto

Na nossa pasta de controladores temos vários controladores como pode verificar.



Controllers

O nosso primeiro controlador é o **“userController.js”**, este é o primeiro controlador que o utilizador interage ao aceder aos nossos serviços.

Uma vez que é necessário também a autenticação ao aceder aos nossos serviços foi utilizado o JWT, para conseguirmos utilizar o JWT é necessário indicar ao programa que vamos utilizar o mesmo.

```
var jwt = require('jsonwebtoken');
```

JSON WEB TOKENS

Neste controlador pode verificar a função **generateAcessToken**, em que cria um token para o utilizador efetuar o login.

```
function generateAcessToken(email, password) {
  var token = jwt.sign({ email, password }, process.env.TOKEN_SECRET, { expiresIn: '1800s' });
  return token;
}
```

Funcao 1

Esta nossa função recebe dois parâmetros nomeadamente o email e a password, e após retorna o token do utilizador.



Uma vez que também é necessário validar se o utilizador já existe ou não, e também é preciso ter acesso a informação da nossa base de dados temos de requerer também o sequelize de forma a depois poder realizar queries.

```
const User = require('../sequelize').User;
```

Sequelize

A função **logout()** é utilizada para destruir o token que terá sido criado anteriormente desta forma faz com que o utilizador deixe de estar autenticado.

```
exports.logout = function(req, res) {  
  req.session.destroy();  
  res.redirect('/')  
}
```

logout()

A função **signup()** é a função que vai efetuar a criação do utilizador na nossa base de dados.

Inicialmente o utilizador ao inserir um email para a criação da sua conta, o programa vai verificar se o utilizador está a respeitar o padrão de email que se encontra definido se for o caso o mesmo após procura na nossa DB se o utilizador existe se o utilizador não existir o mesmo será criado.

### Validação de erros

- ➔ Se o email já existir na BD apresenta essa informação.
- ➔ Se o utilizador tentar inserir uma password inferior a 6 caracteres ou tentar introduzir o email e a password iguais.
- ➔ Se o utilizador não estiver a introduzir um email.
- ➔ Se o utilizador estiver a tentar criar uma conta sem nenhuma informação.

```
exports.signup = function(req, res) {  
  var { email } = req.body;  
  var { password } = req.body;  
  var pattern = /^[a-zA-Z_]+?\.?[a-zA-Z]{2,3}$/;  
  if (pattern.test(email)) {  
    User.findOne({  
      where: {  
        email: email  
      }  
    }).then(result => {  
      if (result == null) {  
        if (req.body.password.length < 6) {  
          req.flash('signupMessage', 'Your password is too short you need at least 6 characters!');  
          res.redirect('/signup');  
        } else if (password == email) {  
          req.flash('signupMessage', 'Your email cant be the same has your password!');  
          res.redirect('/signup');  
        } else {  
          User.create({ 'email': email, 'password': password })  
            .then(user => {  
              req.session.user = user;  
              req.session.token = generateAccessToken(email, password);  
              res.redirect('/profile');  
            });  
        }  
      } else {  
        req.flash('signupMessage', 'That e-mail is already taken.');        res.redirect('/signup');  
      }  
    }).catch(function(err) {  
      req.flash('signupMessage', err);  
      res.redirect('/signup');  
    });  
  } else {  
    req.flash('signupMessage', 'That format is not valid');  
    res.redirect('/signup');  
  }  
}
```

signup()

E por ultimo neste controlador temos a função **login()** o utilizador ao inserir a informação é procurado na BD se a informação corresponde ao que se encontra guardado na BD e se for o caso depois cria um token para o utilizador que está a aceder aos nossos serviços.

### Validação de erros

- ➔ Verifica se o utilizador existe na BD se não existir indica essa informação.
- ➔ Verifica se a password está correta.
- ➔ Verifica se o formato de email que está a ser introduzido está correto.

```
exports.login = function(req, res) {
  var { email } = req.body;
  var { password } = req.body;
  var pattern = /^\\w+@[a-zA-Z_]+?\\. [a-zA-Z]{2,3}$/;
  if (pattern.test(email)) {
    User.findOne({
      where: {
        email: email
      }
    }).then(result => {
      if (result == null) {
        res.render('login.ejs', req.flash('loginMessage', 'Account does not exist'));
      } else if (password != result.password) {
        req.flash('loginMessage', 'Wrong password');
        res.redirect('/login');
      } else {
        req.session.user = result.dataValues;
        req.session.token = generateAccessToken(email, password);
        res.redirect('/profile');
      }
    }).catch(err => {
      req.flash('loginMessage', err);
      res.redirect('/login');
    });
  } else {
    req.flash('loginMessage', 'That email is not valid');
    res.redirect('/login');
  }
};
```

login()

O segundo controlador que tambem pode verificar é o **“uploadController.js”** este é o controlador que permite ao utilizador efetuar o upload das imagens para os nosso serviços para o upload das imagens utilizamos o multer e o utilizador ao introduzir a imagem é enviado para a BD toda a informação da imagem que foi definida nos modelo **“image.js”**.

Uma vez que é necessário aceder a base de dados das nossas imagens para depois criar o que foi definido no modelo é necessário um require como anteriormente.

```
const Imges = require('../sequelize').Imges;
```

Criamos duas funções a primeira função é para efetuar o upload apenas de 1 imagem e a segunda função serve para efetuar upload até 10 imagens.

### Validação de erros

- ➔ Verifica se está a introduzir no minimo 1 imagem e um maximo de 10.
- ➔ Verifica se a imagem introduzida têm como formato png, jpg ou jpeg

```
exports.oneImage = (req, res, next) => {
  if (!req.file) {
    const error = new Error('There is no file to upload')
    error.httpStatusCode = 400
    return next(error)
  } else {
    const userAtual = req.session.user.id
    Imges.create({
      img_original_name: req.file.originalname,
      img_filename: req.file.filename,
      img_location: req.file.path,
      user_img_id: userAtual
    })
    .then(img => {
      res.render('upload_sucess.ejs', {
        locations: [img.img_filename]
      });
    })
  }
}
```

oneImage()

```
exports.multipleImage = (req, res, next) => {
  const files = req.files
  var img_locations = []
  if (files == 0) {
    const error = new Error('There are no files to upload')
    error.httpStatusCode = 400
    return next(error)
  } else if (files > 10) {
    const errorF = new Error('Too many files to upload')
    errorF.httpStatusCode = 400
    return next(errorF)
  } else {
    const userAtual = req.session.user.id
    req.files.forEach(element => {
      Imges.create({
        img_original_name: element.originalname,
        img_filename: element.filename,
        img_location: element.path,
        user_img_id: userAtual
      })
      img_locations.push(element.filename)
    });
    res.render('upload_sucess.ejs', {
      locations: img_locations
    });
  }
}
```

multipleImage()

O nosso terceiro controlador é “**catController.js**”, este é o controlador que vai procurar na nossa base de dados as imagens que o utilizador atual têm e depois envia para o front end a localização das imagens que após faz com que as imagens possam ser demonstradas ao utilizador, uma vez que estamos a utilizar a tabela das imagens o require a utilizar é o mesmo que foi indicado no controlador “**uploadController.js**”.

### Validação de erros

➔ Valida se o user atual têm alguma imagem se não tiver indica essa informação.

```
exports.allImages = (req, res) => {
  const userAtual = req.session.user.id
  var img_locations = [];
  Imges.findAll({
    where: {
      user_img_id: userAtual,
    }
  })
  .then(img => {
    if (img.length == 0) {
      res.json("User currently has no images");
    } else {
      img.forEach(element => {
        img_locations.push(element.img_filename)
      });
      res.render('images.ejs', {
        locations: img_locations
      });
    }
  })
}
```

Também decidimos criar um botão no front end que caso o utilizador pretende possa apagar todas as suas imagens, de forma a efetuar esse pedido é necessário e para apagar a informação da base de dados mas também os ficheiros locais, é necessário indicar que vamos utilizar o file system.

```
const fs = require('fs');
```

Utilizando o destroy e com um ciclo forEach de forma a aceder a todas as imagens do utilizador, o destroy efetuar a retirada da imagem da nossa BD e o fs.unlinkSync apaga as imagens do servidor.

```
exports.catDeleteAll = (req, res) => {
  const userAtual = req.session.user.id;
  var rowDeleted = 0;
  Imges.findAll({
    where: { user_img_id: req.session.user.id }
  })
  .then(images => {
    rowDeleted = images.length;
    if (images == null || images.length == 0) {
      res.status(400).send("Não têm gatos ou já foram apagados")
    } else {
      images.forEach(element => {
        fs.unlinkSync(element.img_location),
        (err) => {
          if (err) throw err;
        }
        Imges.destroy({ where: { id: element.id } })
      });
    }
    res.json("Rows deleted: " + rowDeleted);
  });
}
```

catDeleteAll()

## Validação de erros

➔ Se não existir imagens para o utilizador atual indica que não têm.

Também criamos um endpoint em que pode apagar um único gato introduzindo nos parametros o id do gato.

O raciocinio é o mesmo que no endpoint `catDeleteAll` a única diferença é que procura apenas por uma imagem e apaga essa imagem, a validação de erros é a mesma.

```
exports.catDelete = (req, res) => {
  const id = req.params.id;
  Imges.findOne({
    where: { id: id }
  })
  .then(image => {
    if (image == null) {
      res.status(400).send("Gato não existe ou já foi apagado.")
    } else {
      var dellocation = image.img_location;
      Imges.destroy({ where: { id: id } })
      .then(image => {
        res.json("Rows Deleted: " + image);
      });
      fs.unlinkSync(dellocation),
      (err) => {
        if (err) throw err;
        console.log('successfully deleted');
      }
    }
  });
};
```

`catDelete()`

O nosso ultimo controlador é **“uploaderController.js”**, esté o controlador que vai procurar na nossa base de dados toda a informação sobre os uploaders com a implementação CRUD.

Para podermos verificar todos os uploader que estam registados no nosso servidor criamos o endpoint.

```
exports.alluploaders = (req, res) => {
  User.findAll()
  .then(users => {
    res.json(users)
  });
}
```

`alluploaders()`

Introduz um novo utilizador pelo endpoint através do Postman

## Validação de erros

- ➔ Verifica se o body está vazio
- ➔ Verificar se falta o email ou password
- ➔ Verifica se o utilizador já existe

```

exports.postUploader = (req, res) => {
  if (JSON.stringify(req.body) == "{}") {
    res.status(400).send("Nao tem informação no body")
  } else {
    newEmail = req.body.email;
    newPw = req.body.password;
    User.findOne({
      where: { email: newEmail }
    })
    .then(user => {
      if (!newEmail || !newPw) {
        res.status(400).send("Falta dados");
      } else if (user == null) {
        User.create({
          email: newEmail,
          password: newPw
        })
        .then(newUser => {
          res.json(newUser)
        });
      } else {
        res.status(401).send("Utilizador já existe");
      }
    });
  }
}

```

postUploader

O endpoint recebe o ID por parametro e apaga esse uploader.

### Validação de erros

➔ Verifica se o uploader existe

```

exports.uploaderId = (req, res) => {
  const id = req.params.id;
  User.findAll({
    where: { id: id }
  })
  .then(user => {
    if (user.length == 0) {
      res.status(400).send("User não existe ou já foi apagado.")
    } else {
      res.json(user);
    }
  });
}

```

uploaderId

E por ultimo temos a atualização de password de um utilizador em que recebe o ID por parametro e a password pelo body e após atualiza o user e indica a nova password.

```

exports.uploaderPut = (req, res) => {
  const id = req.params.id;
  const newPw = req.body.password;
  User.findOne({
    where: { id: id }
  })
  .then(atualiza => {
    if (atualiza == null) {
      res.status(400).send("User não existe ou já foi apagado.")
    } else {
      atualiza.update({
        password: newPw
      })
      .then(pw => {
        res.json("Password is: " + pw.password);
      });
    }
  });
}

```

uploaderPut

---

# ACESSO AOS ENDPOINTS

---

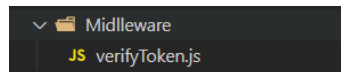
De forma a podermos ter acesso aos endpoints nós inicialmente no ficheiro app.js já indicamos quais os ficheiros que seram necessário para aceder aos endpoints e quais seram a suas rotas.

```
var userRouter = require('./routes/user');
var usersRouter = require('./routes/uploader');
var uploadRouter = require('./routes/upload');
var catRouter = require('./routes/cat');
```

```
app.use('/', userRouter);
app.use('/uploader', usersRouter);
app.use('/upload', uploadRouter);
app.use('/cat', catRouter);
```

Utilização Rotas

Também de forma a proteger os nossos endpoints para que apenas os utilizadores que estão logados possam ter acesso aos endpoints, foi criada a pasta midlleware, em que verifica primeiro se o token é valido e se for valido deixa ter acesso ao endpoint, se não for valido redireciona o user para a pagina de login.

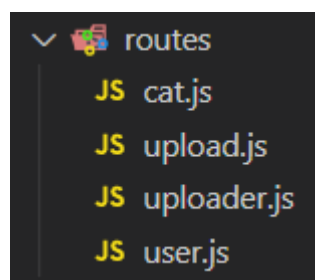


```
var jwt = require('jsonwebtoken');

module.exports = function(req, res, next) {
  const token = req.session.token;
  if (token == null) {
    req.flash('loginMessage', 'You need to be logged in to access that information!');
    res.redirect('/login');
  } else {
    jwt.verify(token, process.env.TOKEN_SECRET, function(err, user) {
      if (err) {
        return res.sendStatus(403);
      }
      next();
    });
  }
}
```

Midlleware

Para conseguir aceder aos nossos endpoints toda a informação dos endpoints está na pasta routes é necessário efetuar o require do ficheiro que contem as funções dos endpoints de forma a conseguir ter acesso a informação. Nestes ficheiros foi também efetuado um require do midlleware de autenticação de forma a proteger os endpoints como foi indicado anteriormente.



## Acesso aos endpoints do catController.

```
var express = require('express');
var router = express.Router();
var authenticateTokenFromSession = require('../Middleware/verifyToken');

var catController = require('../controllers/catController.js');

router.get('/images', authenticateTokenFromSession, catController.allImages);

router.delete('/catDelete/:id', authenticateTokenFromSession, catController.catDelete);

router.delete('/catDeleteAll', authenticateTokenFromSession, catController.catDeleteAll);

module.exports = router;
```

catController.js

## Como ter acesso aos end points do catController:

GET ▼ http://localhost:3000/cat/images Send ▼

➔ Permite verificar todas as imagens que o utilizar atual têm

DELETE ▼ http://localhost:3000/cat/catDelete/1 Send ▼

➔ Permite apagar um gato pelo seu ID enviado no parametro

DELETE ▼ http://localhost:3000/cat/catDeleteAll Send ▼

➔ Apaga todos os gatos que esse utilizar fez upload

## Acesso aos endpoints do userController.

```
var express = require('express');
var router = express.Router();
var authenticateTokenFromSession = require('../Middleware/verifyToken');

var userController = require('../controllers/userController.js');

router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});

router.post('/login', userController.login);
router.get('/login', function(req, res, next) {
  res.render('login.ejs', { message: req.flash('loginMessage') });
});

router.get('/logout', userController.logout);

router.post('/signup', userController.signup);
router.get('/signup', function(req, res, next) {
  res.render('signup.ejs', { message: req.flash('signupMessage') });
});

router.get('/profile', authenticateTokenFromSession, function(req, res) {
  res.render('profile.ejs', {
    user: req.session.user
  });
});
```



## Como ter acesso aos end points do userController:

GET ▼ http://localhost:3000/ Send ▼

→ Dá nos acesso a pagina inicial

GET ▼ http://localhost:3000/login Send ▼

→ Permite nos aceder à pagina do login, o POST que podemos verificar na imagem em cima faz com que os dados inseridos sejam enviados e confirmados pelo endpoint `userController.login`

GET ▼ http://localhost:3000/signup Send ▼

→ Permite aceder à pagina para criar um user, o POST que podemos verificar na imagem em cima faz com que os dados inseridos sejam enviados para a BD e criados pelo endpoint `userController.signup`

GET ▼ http://localhost:3000/profile Send ▼

→ Permite ter acesso a pagina do perfil

## Acesso aos endpoints do uploaderController.

```
var express = require('express');
var authenticateTokenFromSession = require('../Middleware/verifyToken');
var router = express.Router();

var uploaderController = require('../controllers/uploaderController.js');

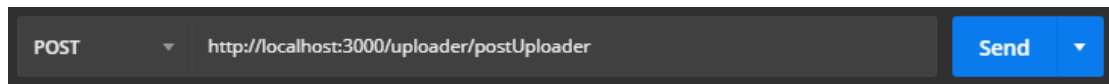
router.get('/alluploaders', authenticateTokenFromSession, uploaderController.alluploaders);
router.post('/postUploader', authenticateTokenFromSession, uploaderController.postUploader);
router.delete('/uploaderDelete/:id', authenticateTokenFromSession, uploaderController.uploaderDelete);
router.get('/uploaderId/:id', authenticateTokenFromSession, uploaderController.uploaderId);
router.put('/uploaderPut/:id', authenticateTokenFromSession, uploaderController.uploaderPut);

module.exports = router;
```

## Como ter acesso aos end points do uploaderController:

GET ▼ http://localhost:3000/uploader/alluploaders Send ▼

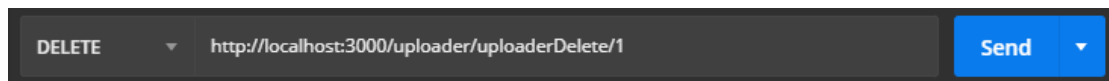
→ Vai demonstrar todos os uploaders existentes



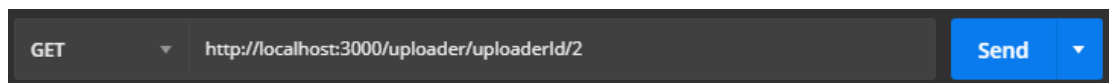
→ Cria um novo user

```
1 {
2   "email" : "artur@backend.com",
3   "password" : "projetoNota20"
4 }
```

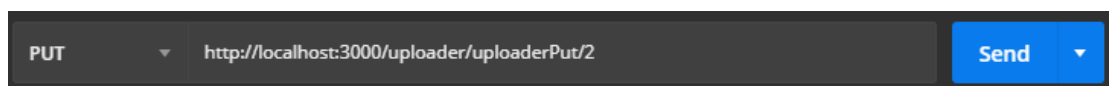
Exemplo



→ Apaga um user que recebe o ID nos parametros



→ Accede à informação de um user recebendo o ID nos parametros



→ Atualiza a palavra pass do user que foi introduzido nos parametros

```
{
  "password" : "mereco20noMinimo18"
}
```

Exemplo

O ultimo ficheiro o upload.js é onde está a informação que indica ao multer onde vai guardar os ficheiros e faz a validação do ficheiro que está a ser inserido de forma a aceitar apenas .png, .jpg e .jpeg como pode verificar na função abaixo.

```
const multer = require('multer');
var storage = multer.diskStorage({
  destination: function(req, file, cb) {
    cb(null, './public/images')
  },
  filename: function(req, file, cb) {
    if (file.mimetype == "image/png" || file.mimetype == "image/jpg" || file.mimetype == "image/jpeg") {
      cb(null, Date.now() + '-' + file.originalname)
    } else {
      return cb(new Error('Only .png, .jpg and .jpeg format allowed!'));
    }
  },
});
var upload = multer({ storage: storage })
```

multer()

## Acesso aos endpoints do uploadController.

```
var express = require('express');
var router = express.Router();
var uploadController = require('../controllers/uploadController.js');
var authenticateTokenFromSession = require('../Middleware/verifyToken');
const multer = require('multer');

var storage = multer.diskStorage({
  destination: function(req, file, cb) {
    cb(null, './public/images')
  },
  filename: function(req, file, cb) {
    if (file.mimetype == "image/png" || file.mimetype == "image/jpg" || file.mimetype == "image/jpeg") {
      cb(null, Date.now() + '-' + file.originalname)
    } else {
      return cb(new Error('Only .png, .jpg and .jpeg format allowed!'));
    }
  },
});

var upload = multer({ storage: storage });

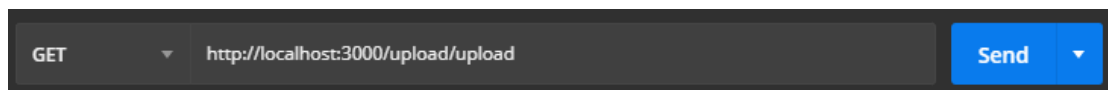
router.post('/uploadfile', authenticateTokenFromSession, upload.single('image'), uploadController.oneImage);

router.post('/uploadmultiple', authenticateTokenFromSession, upload.array('images', 10), uploadController.multipleImage);

router.get('/upload', authenticateTokenFromSession, function(req, res) {
  res.render('upload.ejs', {
    user: req.session.user
  });
});

module.exports = router;
```

## Como ter acesso aos end points do uploadController:



➔ Acede à pagina onde pode efetuar os uploads.

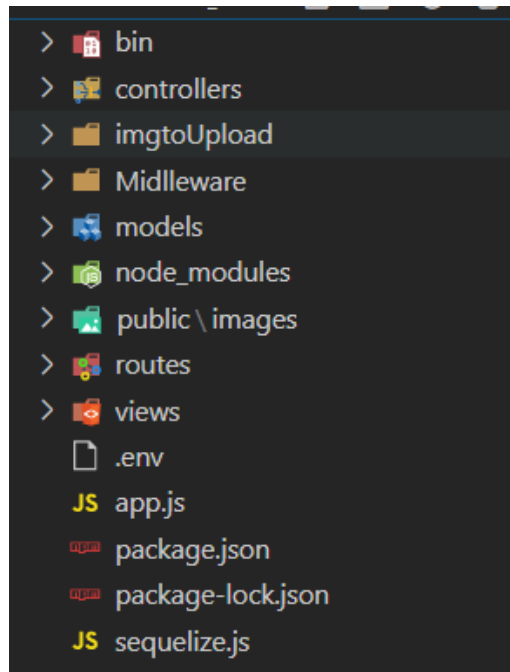
Relativamente aos POSTS estamos a utilizar o front end para ir buscar os endpoints e efetuar o upload.

---

# ESTRUTURAÇÃO DE PASTAS

---

De forma a facilitar a procura de informação e na eventualidade de ser necessário futuramente efetuar alterações no projeto como pode verificar utilizamos o modelo MVC para estruturação e ainda introduzimos uma pasta chamada Midlleware de forma a armazenar todos os midllewares que forem precisos e também de forma a evitar com que seja necessário a repetição de código.



---

# PROCESSOS UTILIZADOS DE DESENVOLVIMENTO

---

Para efetuarmos o desenvolvimento de este projeto nos utilizamos o Postman para testar todos os requests que tínhamos criado, relativamente ao código utilizamos o Visual Studio Code uma vez que já estamos bastante familiarizados com o mesmo, de forma a criar o nosso servidor de SQL utilizamos o XAMPP e para termos acesso a informação da nossa base de dados utilizamos o phpMyAdmin de forma a visualizar toda a informação e também para testar algumas queries que queríamos realizar e depois adaptar ao Sequelize, para o front-end o navegador utilizado foi o Google Chrome uma vez que facilmente temos acesso a toda a informação através da sua consola e ainda podemos editar os elementos em real-time.

---

# CONCLUSÃO

---

Após concluirmos este projeto, temos acerteza ter melhorado e aprofundado os nossos conhecimentos como programadores.

Uma das maiores dificuldades que encontramos ao desenvolver este projeto foi ao efetuar a ligação do back-end com o front-end uma vez que não tínhamos bem a certeza como podíamos criar por exemplo um botão que ao carregar apagasse todas as imagens de um utilizador mas após algum tempo de pesquisa podemos dizer que conseguimos desenvolver este projeto com sucesso.

Também tivemos algumas dificuldades com o endpoint do logout uma vez que a maior parte das pessoas quando utilizam JWT para efetuar o logout tentam expirar o token que foi criado ou apagar o token no lado do cliente quando é efetuado o logout no entanto uma vez que nenhum desses métodos achamos que seria seguro decidimos usar o express session, uma vez que estamos a usar o express e assim facilmente através da documentação do express conseguimos verificar como era realizada a destruição do o token.

Após concluirmos este projeto podemos indicar que aprendemos e adquirimos varios conhecimentos acerca de autentificação, ficamos a perceber melhor como funcionava o sequelize e expandimos os nossos conhecimentos sobre a utilização de frameworks.