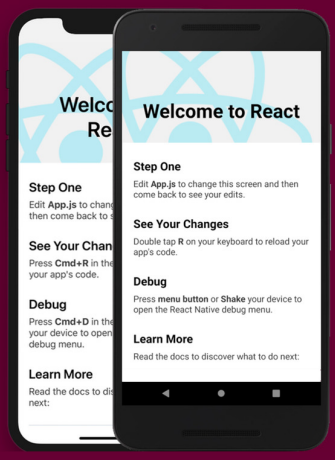
INSTITUTO POLITÉCNICO DE BRAGANÇA

Desenvolvimento Multiplataforma



Acetatos das aulas teóricas

REST

Daniel Gouveia, Henrique Alho, Paulo Matos, Pedro Oliveira

INSTITUTO POLITÉCNICO DE BRAGANÇA

REST & Web Socket's

- Tópicos
  - API's cliente (XMLHttpRequest, fetch, axios)
  - ~~Serviços REST com Express em Node.js~~
  - JWT

REST

Cliente XMLHttpRequest

REST – Cliente XMLHttpRequest

- API de baixo nível de acesso a serviços HTTP/HTTPS
- Permite acesso síncrono e assíncrono (por defeito é assíncrono)
- O React Native tem a sua própria implementação
- É praticamente semelhante à versão nativa do Javascript (mas sem as restrições relativas ao CORS)

```
//0. npm install xhr2
//5. Handler that processes the reply
let processRequest = function(){console.log(this.responseText);}

var XMLHttpRequest = require('xhr2');
var req = new XMLHttpRequest();         //1. Create the object
req.addEventListener("load", processRequest);   //2. Associate a listener
req.open("GET", "https://jsonplaceholder.typicode.com/users"); //3. Setup the request
req.send(); //4. Execute the request
```

REST – Cliente XMLHttpRequest

- São vários os eventos possíveis de capturar

```
let req = new XMLHttpRequest();
req.addEventListener("load", processRequest);
// Define the response type
var req = new XMLHttpRequest();
req.onload = function(){console.log(this.response);}
```

- Eventos (principais):
  - load – Após o término da transferência dos dados
  - error – Em caso de erro
  - abort – Após a operação ter sido abortada
  - timeout – Em caso de ocorrer timeout (após se dar o timeut)
  - loadend – Após o término, seja por load ou por error/abort
  - Loadstart – Assim que se dá o início da transferência de dados
  - progress – Sempre que mais dados são rececionados

```
var req = new XMLHttpRequest();
req.onprogress = function(oEvent){
   console.log(oEvent.loaded);
}
req.open("GET",
"https://jsonplaceholder.typicode.com/users");
req.send();
```


REST – Cliente XMLHttpRequest

- Propriedades (principais)
  - readyState (read only) – assinala se o pedido atingiu o estado "ready"
  - response (read only) – Conteúdo da resposta (dados)
  - responseText (read only) – A DOMString de response
  - responseXML (read only)– A Document de response (XML/HTML)
  - responseType – Tipo de conteúdo (texto, json, …)
  - responseURL (read only) – URL da resposta serializado
  - status (read only) – Estado da resposta (unsigned short)
  - timeout – tempo em milissegundos que o pedido tem para ser processado

INSTITUTO POLITÉCNICO DE BRAGANÇA

REST – Cliente XMLHttpRequest

- Métodos (principais)
  - open(method, URL [, async [, user [,passwd]]]) – Inicializa o pedido

    Método: GET, POST, PUT, DELETE

    Modo síncrono: open(method, URL, false);

    Pedido com autenticação: open(method, URL, true, “user”, “12345”);
  - send ([data]) – Envio o pedido – pode incluir os dados a enviar (upload/form/…)
  - setRequestHeader() – Permite definir um elemento do header do pedido

    Os header são os definidos pelo protocolo HTTP(S)

    Deve ser utilizado após o open () e antes do send()
  - getResponseHeader() – Permite apurar o valor de um header da resposta

    console.log(req.getResponseHeader(‘Content-Type’))
  - abort – Permite abortar um pedido já enviado (apenas em modo assíncrono)
  - overrideMimeType(mimetype) - Permite alterar o MimeType da resposta

INSTITUTO POLITÉCNICO DE BRAGANÇA

REST – Cliente XMLHttpRequest

- Suporte diferentes tipos de retorno
  - XMLHttpRequest.responseType
  - “” (empty string) – A resposta é tratado como texto
  - arraybuffer – Javascript ArrayBuffer (dados binários)
  - blob – Tipo Blob (também corresponde a dados binários)
  - json – Resposta é tratada como texto json
  - text – Valor por omissão, corresponde a texto
  - document – Conteúdo tipo HTML ou XML

```
let processRequest = function(){console.log(this.response);}
var XMLHttpRequest = require('xhr2');
var req = new XMLHttpRequest();
req.addEventListener("load", processRequest);
req.open("GET", "https://jsonplaceholder.typicode.com/users");
req.responseType = "json"; // Define the response type
req.send();
```

REST

Cliente fetch

REST – Cliente fetch

- É considerada como uma alternativa mais flexível e poderosa do que o XMLHttpRequest
- API assíncrona de acesso a serviços HTTP/HTTPS que resulta numa promise
- Faz parte dos métodos disponibilizados pelo *window* do browser, mas é também disponibilizada pelo React Native
- O único argumento obrigatório é o URL
  - `fetch(URL [, config])`
- A promise é resolvida (com sucesso) assim que são rececionados os header – mesmo em caso de erro
- Apenas resolve com insucesso caso não seja possível concluir o pedido
- config é opcional e é uma objeto que permite configurar vários parâmetros do fetch

REST - Cliente fetch

- Exemplo:

```
fetch('https://jsonplaceholder.typicode.com/users')
.then(resp=>resp.json())
.then(data=>console.log(data));
```

- O segundo parâmetro é um objeto com os seguintes campos:
  - method – Permite configurar o método utilizado no acesso HTTP(S) – por omissão é o GET
  - headers – Permite especificar os headers a utilizar no pedido
  - body – Permite definir o body do request (dados a remeter), como um objetivo tipo: Blob, BufferSource, FormData, URLSearchParams, USVString ou ReadableStream
  - mode – Modo de utilização do pedido {cors, no-cors, same-origin}
  - credentials –Credenciais a utilizar pelo pedido

REST - Cliente fetch

- Exemplo com parâmetros de configuração

```
const fetch = require("node-fetch");
fetch('https://jsonplaceholder.typicode.com/posts', {
  method: 'POST', // Alteração do método
  body: JSON.stringify({ // Definição dos dados a enviar
    title: 'foo',
    body: 'bar',
    userId: 1,
  }),
  headers: {  // Configuração do header
    'Content-type': 'application/json; charset=UTF-8',
  },
})
.then((response) => response.json())
.then((json) => console.log(json));
```

- Resultado: {id: 101, title: 'foo', body: 'bar', userId: 1}

REST - Cliente fetch

- React Native – componentes definidos por classes

```
export default class PostsScreen extends React.Component{
  constructor(props){
    super(props);
    this.state={posts:[]};
  }
  componentDidMount(){
    fetch('https://jsonplaceholder.typicode.com/posts')
      .then((resp) => resp.json()).then(posts => {this.setState({posts})});
  }
  render(){
    return (
      <View style={styles.container}>
        <Text style={styles.header}>List of posts</Text>
        <FlatList data={this.state.posts} renderItem={item=>Post(item)}
            keyExtractor={item=>item.id}/>
       </View>
    );
  }
}
```


REST - Cliente fetch

- React Native – componentes definidos por funções

```
export default function PostsScreen(){
  const [posts, setPosts] = useState([]);
  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/posts')
      .then(resp => resp.json())
      .then((posts) => {setPosts(posts);})
  }, []);
  return (
      <View style={styles.container}>
        <Text style={styles.header}>List of posts</Text>
        <View style={styles.list}>
        <FlatList stle={styles.flat} data={posts} renderItem={item=>Post(item)}
          keyExtractor={item=>item.id}>
        </FlatList>
        </View>
      </View>
    );
}
```

INSTITUTO POLITÉCNICO DE BRAGANÇA

REST

Cliente axios

INSTITUTO POLITÉCNICO DE BRAGANÇA

REST – Cliente axios

- API assíncrona de acesso a serviços HTTP/HTTPS que resulta numa promise
- Comparativamente ao fetch, possui funcionalidades complementares, a saber:
  - Interceção do pedidos e respostas
  - Handlers para transformar os dados de envio e de receção
  - A conversão para JSON é transparente (não requer uma conversão explícita)
  - A sua sintaxe permite efetuar vários pedidos de forma simultânea
  - Tem também maior compatibilidade ao nível dos browsers
- Um pouco mais abrangente que o fetch, permitindo por exemplo múltiplos pedidos
- API assíncrona de acesso a serviços HTTP/HTTPS que resulta numa promise
  - npm install axios => import axios from 'axios';

REST – Cliente axios

- O principal método contempla um único parâmetro que é um objeto

```
axios({method:'GET', url:'https://jsonplaceholder.typicode.com/posts'}).then(…)

axios({
  method: 'post',
  url: 'https://jsonplaceholder.typicode.com/posts',
  data: {title: 'foo', body: 'bar', userId: 1}
}).then( ...)
```

- Disponibiliza vários shortcuts para os diferentes tipos de pedido do HTTP(S)
  - axios.request(config)
  - axios.get(url [,config])
  - axios.post(url [,config])
  - axios.put(url [,config])
  - axios.delete(url [,config])

```
axios.get('https://jsonplaceholder.typicode.com/posts').then(…)
axios.post('/auth', {login: 'user',passwd: '12345'});
```


REST – Cliente axios

- Exemplo

```
export default function ProfileScreen(){
  const [posts, setPosts] = useState([]);
  useEffect(() => {
    axios({method:'GET', url:'https://jsonplaceholder.typicode.com/posts'})
      .then((posts) => {setPosts(posts.data);})}, []);
    return (
      <View style={styles.container}>
        <Text style={styles.header}>List of posts</Text>
        <View style={styles.list}>
        <FlatList style={styles.flat} data={posts} renderItem={item=>Post(item)}
          keyExtractor={item=>item.id}>
        </FlatList>
        </View>
      </View>
    );
}
```

INSTITUTO POLITÉCNICO DE BRAGANÇA

REST – Cliente axios

- Pedidos múltiplos

```
axios.all([
  axios.get('https://jsonplaceholder.typicode.com/posts/1'),
  axios.get('https://jsonplaceholder.typicode.com/posts/2')
])
.then(response => {
  console.log('Title: ', response[0].data.title);
  console.log('Title: ', response[1].data.title);
});
```


INSTITUTO POLITÉCNICO DE BRAGANÇA

REST

Express e Node.js

REST –Express e Node.js

- O Express é uma framework para desenvolvimento web sobre Node.js
- Permite implementar um serviço de HTTP(S) em 4 linhas de código:
  - npm i express --save // Instalar o Express

```
import express from 'express'

const app = express();

app.get('/', (req,res)=>res.send('HTTP GET request received'));

app.listen(3000, ()=>console.log('Starting HTTP server ...'));
```

- O serviço pode ser testado num browser: http://localhost:3000/
- Mas é recomendável instalar um cliente mais versátil, tipo:
  - RESTED - Chrome plugin
  - Simple REST Client + Visual Code Plugin
  - Insomnia – Aplicação standalone

REST –Express e Node.js

- A API de base é muito acessível
- Há métodos para cada tipo de pedido (GET, POST, PUT, ….)

```
import express from 'express'

const app = express();

app.get('/', (req,res)=>res.send('HTTP GET request received'));

app.post('/', (req,res)=>res.send('HTTP POST request received'));

app.put('/', (req,res)=>res.send('HTTP PUT request received'));

app.delete('/', (req,res)=>res.send('HTTP DELETE request received’));

…

app.all(‘/testall', (req,res)=>res.send(‘Capture all HTTP requests’));

app.listen(3000, ()=>console.log('Starting HTTP server ...'));
```

REST -Express e Node.js

- O primeiro parâmetro define a route (path a partir do domínio)
- E o segundo é a função que recebe o pedido e processa a resposta

```
import express from 'express'
const app = express();

app.get('/', (req,res)=>res.send('Home route'));
app.get('/test', (req,res)=>res.send('/test route'));

app.listen(3000, ()=>console.log('Starting HTTP server ...'));
```

- O envio de dados pode ser feito de diversas formas

```
// Query params
// http://localhost:3000/test1?qParam1=10&qParam2=500
app.get('/test1', (req,res)=>{
    res.send('Query param 1:' + req.query.qParam1 + ' Query param 2:'+req.query.qParam2);
});
// Path params
// http://localhost:3000/test2/100
app.get('/test2/:idParam', (req,res)=>{
    res.send('Path Param:'+req.params.idParam);
});
```


REST -Express e Node.js

- Incluindo através de POST
- Para pedidos URLEncoded (forms com pares chave x valor):

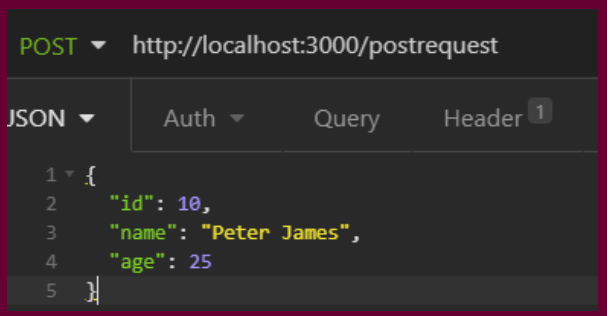
```
import express from 'express'
const app = express();

// Permite extrair os parâmetros do body - para pedidos tipo URLEncoded
app.use(express.urlencoded({ extended: true }));

app.post('/postrequest', (req,res)=>{
    console.log(req.body);
    res.send('Thanks ' + req.body.name);
});

app.listen(3000,
()=>console.log('Starting HTTP server ...'));
```

```
Starting HTTP server ...
{ id: '10', name: 'Peter James', age: '25' }
```

REST -Express e Node.js

- Para pedidos com dados JSON:

```
import express from 'express'
const app = express();

// Permite extrair parâmetros em formato JSON do body
// E codificar objetos em JSON para o envio de dados
app.use(express.json());

app.post('/postrequest', (req,res)=>{
    let myjson = req.body;
    console.log(myjson);
    res.send('Thanks '+myjson.name);
});

app.listen(3000, ()=>console.log('Starting HTTP server ...'));
```

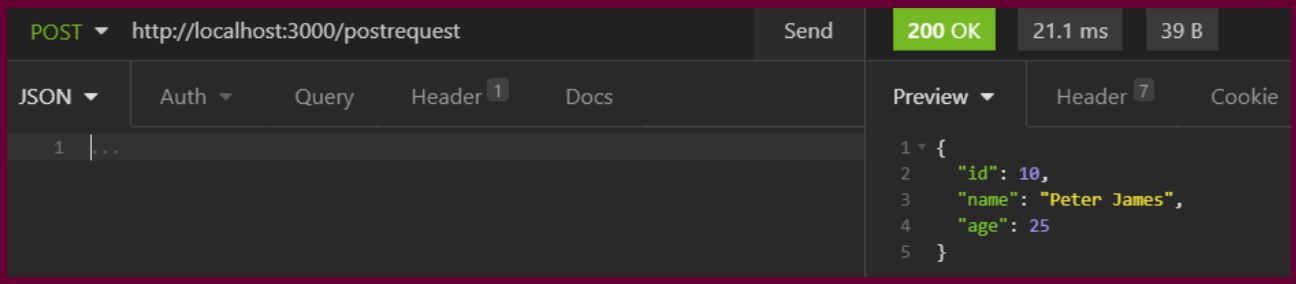



```
Starting HTTP server ...
{ id: 10, name: 'Peter James', age: 25 }
```


REST -Express e Node.js

- Para respostas com dados JSON:

```
import express from 'express'
const app = express();

// Permite extrair parâmetros em formato JSON do body
// E codificar objetos em JSON para o envio de dados
app.use(express.json());

app.post('/postrequest', (req,res)=>{
    let val = {id:10, name:'Peter James', age:25};
    res.send(val);
});

app.listen(3000, ()=>console.log('Starting HTTP server ...'));
```



INSTITUTO POLITÉCNICO DE BRAGANÇA

REST –Express e Node.js

- Route's (rotas)
- Uma forma mais versátil de gerir as rotas é com recurso ao express.Router()
  - Permite agrupar os handlers dos rotas para partes específicas do serviço, utilizando um prefixo comum
  - Bem como, utilizar middlewares específicos por rota

```
const express = require('express');
const router = express.Router();

router.get('/lusers', function(req, res) {
  res.send('lusers endpoint');
});

router.get('/userinfo', function(req, res) {
  res.send('userinfo endpoint');
});
```


INSTITUTO POLITÉCNICO DE BRAGANÇA

REST –Express e Node.js

- Middleware
- São funções executadas entre o pedido e a resposta
- Funções essas que têm acesso ao pedido (req) e à resposta (res)
- Utilizadas para validar, transformar, controlar acessos, etc
- Cada função, após realizar a tarefa para a qual foi idealizada, deve:
  - Responder de imediato, através do *res.send(…)* ou equivalente
  - Ou invocar o método *next()*, passando a execução à próxima função de middleware
  - Se não o fizer, não há resposta (timeout do lado do cliente)
- O *next* utilizado consecutivamente permite que o pedido alcance a função de resposta associada à route

REST –Express e Node.js

- Middleware
- O protótipo das funções depende do tipo de middleware, o normal é

  (req, res, next) => {..., next();}
- *Mas o middleware de tratamento de erros utiliza:*
  - *(error, req, res, next) => {..., next();}*
- O uso das funções de middleware requer que estas sejam registadas
  - Pode ser feito ao nível da aplicação: *app.use(...)*
  - Ao nível dos router's: *router.use(...)*
  - Ou através dos parâmetros das funções get, post, ... devendo ficar entre o segundo e o último (da função de router)
- A ordem pela qual se faz o registo das funções dita a ordem de invocação/aplicação

REST –Express e Node.js

- Middleware – Exemplo1

```
// Middleware function
let mfunction1 = (req,res,next)=>{
    console.log("A");
    if(req.body.uptoyou) res.send('NO CHAINING');
    req.mfunction1=true;
    next();
}
// Middleware function
let mfunction2 = (req,res,next)=>{
    console.log("B");
    req.mfunction2=true;
    next();
}
// Registo das funções ao nível do app
app.use(mfunction1);
app.use(mfunction2);

app.get('/mtest', (req,res)=>{
    console.log("C");
    res.json({mfunction1:req.mfunction1,mfunction2:req.mfunction2});
});
```

20/06/2023 Instituto Politécnico de Bragança - Escola Superior de Tecnologia e de Gestão

REST –Express e Node.js

- Middleware – Exemplo2

```
import express from 'express'
const app = express();

// Application-level middleware (app.use(…)
// Express.json() é uma função de middleware facultada pelo próprio Express
// Que intercepta os pedidos e as respostas para transformar de e para json
app.use(express.json());

app.post('/postrequest', (req,res)=>{
    let val = {id:10, name:'Peter James', age:25};
    res.send(val);
});

// Função de middleware definida pelo próprio programador
let mymiddleware = (req, res, next)=>{
    console.log("Nothing");
    next();
}
// O registo faz-se pelo segundo parâmetro do método post
app.post('/testmiddle', mymiddleware, (req, res)=>res.send("SUCCESS"));

app.listen(3000, ()=>console.log('Starting HTTP server ...'));
```


REST –Express e Node.js

- Middleware
- As funções middleware podem ser associadas a router específicos, sendo unicamente aplicadas para pedidos associados a esse router

```
const express = require('express');
const app = express();
const router = express.Router();

let validateuser = (req, res, next)=>{ ...}

// validateuser está apenas associada ao router /login
router.post('/login',validateuser, (req,res)=>{ ... });

// Router level middleware router.use(…)
// É também possível associar a função de middleware ao route através do método use()
router.use('/login',validateuser);

app.listen(3000, ()=>console.log('Starting HTTP server ...'));
```

INSTITUTO POLITÉCNICO DE BRAGANÇA

REST –Express e Node.js

- Middleware
- São vários os packages de middleware disponíveis:
  - Morgan – Logger configurável para pedidos HTTP
  - Helmet – Configurações de segurança através dos headers do HTTP que evitam algumas vulnerabilidades bem conhecidas do protocolo HTTP
  - CORS – Cross-Origin Resource Sharing
  - Multer – Middleware para tratar pedidos multipart/form-data (upload de ficheiros)
  - Passport – Middleware de autenticação e autorização com múltiplas opções técnicas

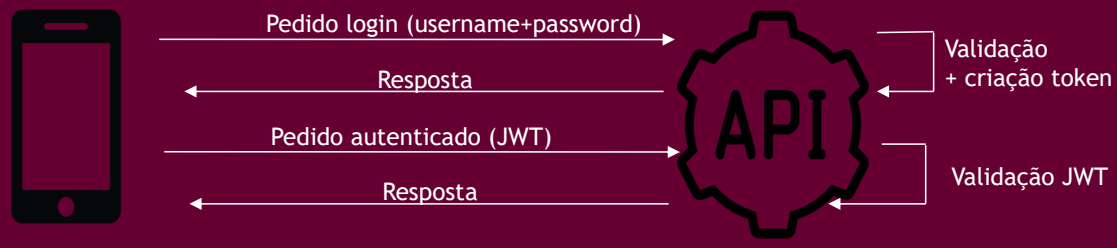
INSTITUTO POLITÉCNICO DE BRAGANÇA

REST

JWT

JSON Web Tokens (norma RFC7519)

- Objeto JSON codificado que funciona como credencial, prestando-se à gestão de sessões e autorizações
- Os dados são codificados como objetos JSON e integrados como dados no JSON Web Signature (JWS) ou na estrutura do JSON Web Encryption (JWE)
- O JWT podem ser assinados com uma chave privada ou com um par chaves RSA (pública + privada) – permitindo diferentes algoritmos de codificação
- A assinatura permite verificar a integridade dos dados nele contido
- Com utilização de chaves público-privadas permite também assegurar que quem possui a chave privada é quem efetivamente assinou



JSON Web Tokens

- O JWT é composto por três partes:
  - Separadas por '.', tipo hhhh.pppp.ssss
  - Header – Define o tipo de token (JWT) e identifica o algoritmo utilizado na assinatura: *{"alg":"HS256", "typ":"JWT"}*
  - Payload – Contém as claims (reivindicações) sob a forma de JSON. Podem ser de três tipos:

    Registadas: iss (Issuer), exp (Expiration time), sub (subject), aud (Audience)

    Públicas: Definidas pelo standard https://www.iana.org/assignments/jwt/jwt.xhtml

    Privadas – Definidas pelo utilizador
  - Signature – Assinatura gerada tendo em conta o Header, o Payload codificado, a chave secreta e o algoritmo especificado no Header

INSTITUTO POLITÉCNICO DE BRAGANÇA

JSON Web Tokens - Utilização

- Na essência consiste em enviar um pedido de registo com as credenciais do utilizador
- Uma vez aceite o registo é gerado o JWT encriptando os dados tidos como relevantes para a identificação da sessão (utilizador) e, eventualmente, com o tempo de validade do JWT
- O JWT é remetido na resposta
- Sempre que se dá um novo acesso, a end-points que requerem autenticação, o JWT deve ser remetido
- Se no servidor o JWT for descodificado com sucesso, é assegurado o acesso ao serviço
- É comum que nos dados de encriptação conste também os privilégios de acesso
- O tempo de vida do JWT deve ser restrito ao essencial

INSTITUTO POLITÉCNICO DE BRAGANÇA

JSON Web Tokens

- Para exemplificar a utilização, será implementado o seguinte conjunto de endpoints:
  - signup –POST endpoint que permite efetuar o registo do utilizador, com base no *username* e *password*
  - login – POST endpoint que permite autenticar o utilizador, com base no *username* e *password*, e remete um JWT
  - test – GET endpoint cujo acesso só é possível com JWT válido
- O acesso tanto pode ser feito pelo *signup*, como pelo *login*
- O JWT deve ser remetido no header *x-access-token* do request

INSTITUTO POLITÉCNICO DE BRAGANÇA

JSON Web Tokens

- Exemplo – index.js

```
//npm init
//npm i express -save
const  express= require('express');
const app = express();
const authRoute = require('./auth');
//npm i helmet --save
const helmet = require("helmet");
app.use(helmet());
//npm i cors --save
const cors = require("cors");
app.use(cors());
// npm i dotenv
//By default reads .env file
require('dotenv').config();
//Process json requests
app.use(express.json());
app.use('/auth', authRoute);

app.listen(3001, ()=>console.log('Server running'));
```


INSTITUTO POLITÉCNICO DE BRAGANÇA

JSON Web Tokens

- Exemplo – auth.js

```
const express = require('express');
const router = express.Router();
//npm i bcrypt --save
const bcrypt = require('bcrypt');
const rounds = 10;
//npm i jsonwebtoken --save
const jwt = require('jsonwebtoken');
//If lgogout is necessary (redis supply destroy JWT method) ...
//const jwt = require('jwt-redis');
const {addUser, findUser} = require('./user');
const middleware = require('./middleware');
//Might be useful to print objects
const util = require('util');
//Funtion that generates the access token
function generateToken(user){
    return jwt.sign({data:user}, process.env.TOKENSECRET, {expiresIn:'24h'} );
}
```

INSTITUTO POLITÉCNICO DE BRAGANÇA

JSON Web Tokens

- Exemplo – auth.js

```
//Signup endpoint
//Receives the user information {username, password, email, address}
router.post('/signup',(req,res)=>{
    // hash is que encrypted password
    bcrypt.hash(req.body.password, rounds, (error,hash)=>{
        if(error) res.status(500).json(error);
        else{
            const user =  {username:req.body.username, email: req.body.email,
                           address: req.body.address, password: hash};
            addUser(user);
            res.status(200).json({token:generateToken(user)});
        }
    });
});

module.exports = router;
```


INSTITUTO POLITÉCNICO DE BRAGANÇA

JSON Web Tokens

- Exemplo – auth.js

```
//Login endpoint, requires {username, password}, returns access token
router.post('/login',(req,res)=>{
    let user = findUser(req.body.username);
    if(user != undefined){
        bcrypt.compare(req.body.password, user.password, (error, match)=>{
            if(error)res.status(500).json(error);
            else if(match) res.status(200).json({token:generateToken(user)})
;
            else res.status(403).json({error:3, msg:'Passwords not match'});
        });
    }else res.status(404).json({error:4, msg:'Username not found'})
});
```

JSON Web Tokens

- Exemplo – auth.js

```
//Endpoint that requires authentication
//The authentication is validated by middleware.verify
router.use('/test', middleware.verify);

router.get('/test', (req, res)=>{
    //console.log(util.inspect(user, {showHidden: false, depth: null}))
    res.status(200).json(req.user);
});
```


JSON Web Tokens

- Exemplo – middleware.js

```
const jwt = require('jsonwebtoken');
const util = require('util')
const {findUser} = require('./user');
exports.verify = (req, res, next) => {
    const header = req.headers['authorization'];
    if(header != undefined){
        const token = header.split(' ')[1]; // Beared token
        if (!token) res.status(403).json({error:1, msg: "No token provided"})
        else {
            jwt.verify(token, process.env.TOKENSECRET, (err, value) => {
                if (err) res.status(500).json({error:2, msg: 'Failed to authenticate token'});
                else{
                    let user = findUser(value.data.username);
                    const {password, ...ret } = user; //value.data;
                    req.user = ret;
                    next();
                }

            })
        }
    }else res.status(403).json({error:1, msg: "No token provided"})
}
```

JSON Web Tokens

- Exemplo – user.js

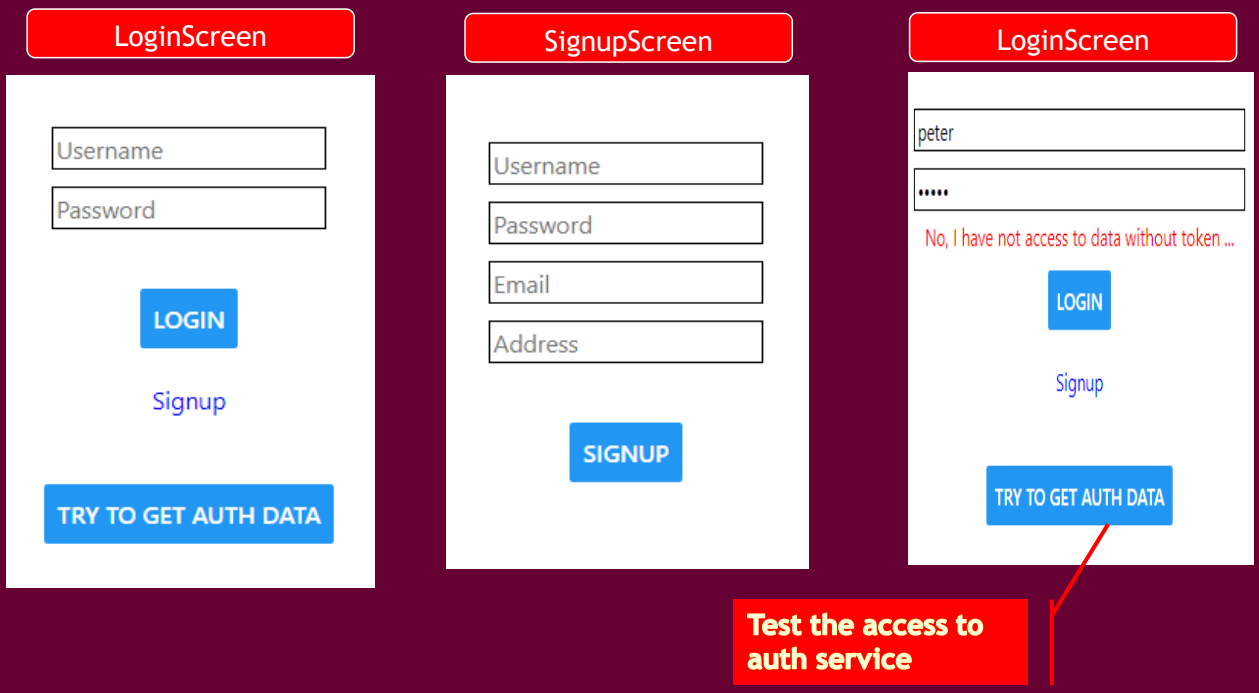
```
// Simulated Data Store for users
//Should be replaced by a persistente solution: mysql, mongoose, …
let luser = [];

let findUser = (username)=>{ return luser.find((v)=>v.username===username)};

let addUser=(user)=>{
    let res = findUser(user.username);
    if(res==undefined) luser.push(user);
}

module.exports={findUser, addUser};
```

- Exemplo – .env

```
TOKENSECRET=MYVERYSECRETKEY
```

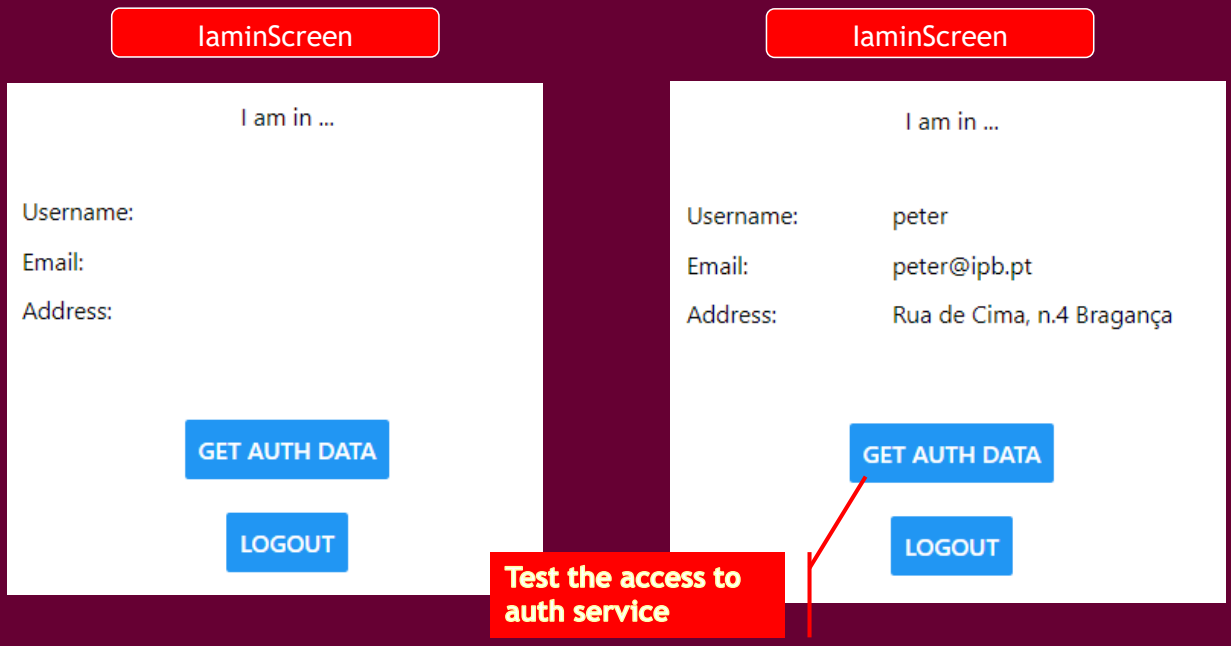

JSON Web Tokens

- Exemplo – React Native Client



JSON Web Tokens

- Exemplo – React Native Client



JSON Web Tokens

- Exemplo – React Native Client

```
import 'react-native-gesture-handler';
import React, {useState} from 'react';
import { StyleSheet, Text, TextInput, View, Button } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator} from '@react-navigation/stack';
//To access REST services
import axios from 'axios';
const util = require('util');

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
  inputs:{
    borderWidth:1,
    borderColor:'black',
    padding:2, margin:5,
    width:300
  }
});
```

20/06/2023 Instituto Politécnico de Bragança - Escola Superior de Tecnologia e de Gestão

INSTITUTO POLITÉCNICO DE BRAGANÇA

JSON Web Tokens

- Exemplo – React Native Client

```
const Stack = createStackNavigator();

export default function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName='Login'
                  screenOptions={{headerShown: false}}>
        <Stack.Screen name='Login' component={LoginScreen}
                  initialParams={{ msg: '' }}/>
        <Stack.Screen name='Iamin' component={IAmInScreen}/>
        <Stack.Screen name='Signup' component={SignupScreen}/>
      </Stack.Navigator>
    </NavigationContainer>

  );
}
```


INSTITUTO POLITÉCNICO DE BRAGANÇA

JSON Web Tokens

- Exemplo – React Native Client - LoginScreen

```
function LoginScreen({navigation, route}){
    const [username, setUsername] = useState('');
    const [password, setPassword] = useState('');
    const {msg} = route.params;
    const [message, setMessage] = useState(msg);
    let login = ()=>{
      axios({method: 'post', url: 'http://localhost:3001/auth/login',
          data: {username: username, password: password}
        }).then((res)=>navigation.navigate('Iamin', {token:res.data.token}),
                (err)=>setMessage(err.response.data.msg));
    }
    let test = ()=>{
      axios.get('http://localhost:3001/auth/test')
      .then((res)=>setMessage('I was able to get data without token ...'),
            (err)=>setMessage('No, I have not access to data without token ...'));
    }
```

JSON Web Tokens

- Exemplo – React Native Client - LoginScreen

```
  return (
    <View style={styles.container}>
      <TextInput style={styles.inputs} placeholder='Username'
                 defaultValue={username} onChangeText={x=>setUsername(x)}/>
      <TextInput style={styles.inputs} placeholder='Password'
                 defaultValue={password}
                  onChangeText={x=>setPassword(x)} secureTextEntry/>
      <Text style={{height:30,color:'red'}}>{message}</Text>
      <Button title='Login'onPress={()=>login()}/>
      <Text style={{margin:20,color:'blue'}}
            onPress={()=>navigation.navigate('Signup')}>Signup</Text>
      <View style={{padding:10, margin:10}} >
        <Button title='Try to get auth data' onPress={()=>test()}/>
      </View>
    </View>
  );
}
```


JSON Web Tokens

- Exemplo – React Native Client - SignupScreen

```
function SignupScreen({navigation}){
  const [username, setUsername] = useState('');
  const [password1, setPassword1] = useState('');
  const [password2, setPassword2] = useState('');
  const [email, setEmail] = useState('');
  const [address, setAddress] = useState('');
  const [message, setMessage] = useState('');
  let checkData=()=>{
    if(username.length<2)setMessage('Username too short');
    else if(password1.length<2) setMessage('Password is too short');
    else if(password2.length<2) setMessage('Password is too short');
    else if(password1!=password2) setMessage('Passwords do not match');
    else{setMessage(''); return true;}
    return false;
  }
```

JSON Web Tokens

- Exemplo – React Native Client - SignupScreen

```
let signup = ()=>{
   if(checkData()==true){
     axios({
         method: 'post',
         url: 'http://localhost:3001/auth/signup',
         data: {username:username, password: password1,
                  email: email, address: address}
       }).then((res)=>navigation.navigate('Login'),
              (err)=>setMessage(err.response.data.msg));
     } else setMessage('Data is not valid. Correct it please.')
 }
```


JSON Web Tokens

- Exemplo – React Native Client - SignupScreen

```
 return (
   <View style={styles.container}>
     <TextInput style={styles.inputs} placeholder='Username' value={username}
                onChangeText={x=>setUsername(x)}/>
     <TextInput style={styles.inputs} placeholder='Password' value={password1}
                onChangeText={x=>setPassword1(x)} secureTextEntry/>
     <TextInput style={styles.inputs} placeholder='Password' value={password2}
                onChangeText={x=>setPassword2(x)} secureTextEntry/>
     <TextInput style={styles.inputs} placeholder='Email' value={email}
                onChangeText={x=>setEmail(x)}/>
     <TextInput style={styles.inputs} placeholder='Address' value={address}
                onChangeText={x=>setAddress(x)}/>
     <Text style={{height:30,color:'red'}}>{message}</Text>
     <Button title='Signup' onPress={()=>signup()}/>
   </View>
 );
}
```

JSON Web Tokens

- Exemplo – React Native Client - IaminScreen

```
function IAmInScreen({navigation, route}){
  const [username, setUsername] = useState('');
  const [email, setEmail] = useState('');
  const [address, setAddress] = useState('');
  const [message, setMessage] = useState('');
  let test = ()=>{
    let token = route.params.token;
    axios.get('http://localhost:3001/auth/test',
         //IMPORTANT: Add the Authorization header
         {headers:{'Authorization': 'Bearer '+token}})
    .then((res)=>{
        setUsername(res.data.username);
        setEmail(res.data.email);
        setAddress(res.data.address);
      }, (err)=>setMessage('Token was not accepted'));
  }
```


JSON Web Tokens

- Exemplo – React Native Client - IaminScreen

```
  return (<View style={styles.container}>
    <Text style={{height:30, margin:20}}>I am in ...</Text>
    <View style={{flexDirection:'row'}}>
      <View style={{width:120}}>
        <Text style={{margin:5}}>Username:</Text>
        <Text style={{margin:5}}>Email:</Text>
        <Text style={{margin:5}}>Address:</Text>
      </View>
      <View style={{width:200}}>
        <Text style={{margin:5}}>{username}</Text>
        <Text style={{margin:5}}>{email}</Text>
        <Text style={{margin:5}}>{address}</Text>
      </View>
    </View>
    <Text style={{color:'red', height:30}}>{message}</Text>
    <View style={{padding:10, margin:10}} >
      <Button title='Get Auth Data' onPress={()=>test()}/>
    </View>
    <Button title='Logout' onPress={()=>navigation.navigate('Login')}/>
  </View>);
}
```

Ensino Superior Público
de qualidade