

NodeJS: autenticação

Processamento e Representação de Informação
Desenvolvimento de Aplicações Web

2018/2019

José Carlos Ramalho

Tutorial orientado por exemplos: fase 1 - preparação

- Vamos criar um servidor, numa primeira fase usaremos o Postman para testar a API e numa segunda fase o browser.

```
$ express autenticacao --pug  
$ cd autenticacao
```

Homepage inicial

Programar a resposta a um GET em '/', no router index.js

```
/* GET home page. */  
router.get('/', function(req, res) {  
  res.render('index')  
})
```

```
extends layout  
  
block content  
  h1 Autenticação básica com NodeJS  
  p Chegaste à homepage...
```

O que é um “cookie”?

- Pequeno ficheiro armazenado no computador do utilizador;
- Pensado para armazenar uma pequena quantidade de informação específica de um determinado cliente ou website;
- Pode ser acedido pelo servidor web ou pelo computador cliente;
- Permite ao servidor fornecer páginas configuradas de acordo com preferências pessoais;
- As páginas servidas também podem conter scripts que usam a informação guardada no cookie para a transportar entre visitas.

O que é uma “session”?

- Pode ser definida como o armazenamento de informação do lado do servidor que queremos que persista durante a interação do utilizador com o website ou aplicação web;
- Quando se utilizam sessões, em vez de se armazenar uma grande quantidade de informação em cookies no browser do utilizador, apenas um identificador único é armazenado do lado do cliente: o id da sessão;
- O id da sessão é então passado ao servidor sempre que o cliente faz um pedido;
- A aplicação web / servidor verificará o id enviado e atuará de acordo com a política que tiver definida.

Node.js: instalar o uuid - universally unique identifier

Inclusão do módulo:

```
var uuid = require('uuid/v4')
```

Utilização:

```
const uniqueID = uuid()
```

Várias versões:

- v1 - timestamp;
- v2 - namespace;
- v4 - random.

Vamos adicionar suporte às sessões: express-session

Vamos instalar o `express-session` que nos vai permitir gerar sessões, algo que o `express` não faz por omissão.

```
$ npm i express-session --save
```

Vamos adicionar suporte às sessões: express-session

Vamos instalar o `express-session` que nos vai permitir gerar sessões, algo que o `express` não faz por omissão.

```
$ npm i express-session --save
```


Vamos adicionar o middleware para uma sessão

```
var uuid = require('uuid/v4')

var session = require('express-session')

...

var app = express();

app.use(session({
  genid: req => {
    console.log('Dentro do middleware da sessão...')
    console.log(req.sessionID)
    return uuid() },
  secret: 'O meu segredo',
  resave: false,
  saveUninitialized: true
}))
```

E uns logs no router...

```
router.get('/', function(req, res) {
  console.log('Na cb da homepage...')
  console.log(req.sessionID)
  res.render('index')
})
```

Podemos observar os cookies no Postman

GET http://localhost:4011

Send Save

Params Authorization Headers Body Pre-request Script Tests **Cookies** Code

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies (1) Headers (7) Test Results Status: 200 OK Time: 79 ms Size: 531 B Save Download

localhost 1 cookie

connect.sid X + Add Cookie

```
connect.sid=s%3A4ef60f08-9f8a-4b64-b4e2-565e78629352.RZhfl1RoFCLUWxM7dlt7fv%2Bz14rGgi4mGuuW9vTyiWE; path=/; domain=localhost; HttpOnly; Expires=Tue, 19 Jan 2038 03:14:07 GMT;
```

Cancel Save

Exercício

1. Usa o Postman para fazer vários pedidos da homepage;
2. Observa o que acontece com os ids da sessão;
3. Pára o servidor e reinicia-o;
4. Faz uma nova sequência de pedidos e observa os ids;
5. O que podes concluir?

Precisamos de persistência para os ids de sessão!

Persistência dos dados da sessão

- Na documentação do express são referidas várias alternativas, inclusive a base de dados da aplicação que estás a desenvolver;
- Vamos usar o módulo `"session-file-store"`:

```
$ npm i session-file-store --save
```

Vamos alterar o middleware...

```
var FileStore = require('session-file-store')(session)
var app = express();
...
app.use(session({
  genid: req => {
    console.log('Dentro do middleware da sessão...')
    console.log(req.sessionID)
    return uuid()},
  store: new FileStore(),
  secret: 'O meu segredo',
  resave: false,
  saveUninitialized: true
}))
```

Persistência de sessões

A informação de sessão é guardada num ficheiro na pasta sessions.

Se estiveres a correr o servidor com o nodemon:

```
$ nodemon --ignore sessions/
```

Autenticação: pedidos de login

```
// Login page
router.get('/login', function(req, res) {
  console.log('Na cb do GET login...')
  console.log(req.sessionID)
  res.render('login')
})

router.post('/login', function(req, res) {
  console.log('Na cb do POST login...')
  console.log(req.body)
  res.send('Login recebido e tratado...')
})
```

passport e passport-local

- O passport é um módulo com as funcionalidades básicas e comuns;
- Cada estratégia tem as suas especificidades implementadas num módulo adicional nomeado: passport-estratégia;
- Vamos usar a estratégia mais simples: local:

```
$ npm i passport passport-local --save
```


Fluxo de autenticação

1. O utilizador envia a informação de autenticação num POST /login;
2. Temos de tratar esta informação. Chamamos: `passport.authenticate('estratégia', callback(err, user, info))`; a estratégia será 'local', a callback vai dar-nos acesso ao objeto do utilizador se a autenticação tiver sucesso e ao erro em caso contrário;
3. O `passport.authenticate()` irá chamar a nossa estratégia de autenticação pelo que será preciso configurá-la: `passport.use(new strategyClass)`; vamos indicar como queremos autenticar o utilizador;
4. Na especificação da `strategyClass`, vamos retirar a informação do POST, usá-la para encontrar o utilizador na base de dados. Se tudo estiver bem, o passport irá adicionar um método `login()` ao objeto do pedido e retornará a execução para a callback do `passport.authenticate()`;
5. Na cb do `passport.authenticate()`, chamamos o método `req.login()`;
6. O `req.login()` pega no objeto do utilizador devolvido pela estratégia e chama: `passport.serializeUser(callback())`, que faz o seguinte:
 - a. Guarda o id do utilizador na **session file store**;
 - b. Guarda o id do utilizador no objeto request como **request.session.passport**;
 - c. Adiciona o objeto utilizador ao pedido/request como **request.user**

Persistência de utilizadores

- Vamos montar um serviço de persistência de informação em menos de 1 minuto!!!
- Usando o módulo json-server vamos criar um servidor de dados com uma API, configurando apenas os dados iniciais:
 - cria uma subpasta na tua aplicação: usersdb;
 - nessa pasta, cria o package.json: `npm init -y`
 - instala o módulo: `npm i json-server --save`
 - acrescenta nas scripts do package: `"json:server --watch ./db.json --port 5011`
 - cria o ficheiro db.json com os utilizadores iniciais;
 - arranca o serviço em background: `npm run json:server &`

BD de utilizadores

```
{  
  "users": [  
    {  
      "id": "A4140",  
      "email": "jcr@di.uminho.pt",  
      "password": "fixe"  
    },  
    {  
      "id": "A89192",  
      "email": "A89192@alunos.uminho.pt",  
      "password": "a1"  
    }  
  ]  
}
```

Exercício

Testa os seguintes pedidos no Postman:

1. GET /users
2. GET /users/A4140
3. GET /users?email=A11111@alunos.uminho.pt
4. POST /users (com os dados de um novo utilizador)

Módulos necessários para a autenticação

```
var uuid = require('uuid/v4')
var session = require('express-session')
var FileStore = require('session-file-store')(session)

var passport = require('passport')
var LocalStrategy = require('passport-local').Strategy
var axios = require('axios')
var flash = require('connect-flash')
```

Configuração da estratégia de autenticação

```
// Configuração da estratégia local
passport.use(new LocalStrategy(
  {usernameField: 'email'}, (email, password, done) => {
    axios.get('http://localhost:5011/users?email=' + email)
      .then(dados => {
        const user = dados.data[0]
        if(!user) { return done(null, false, {message: 'Credenciais
inválidas!\n'})}
        if(password !== user.password) { return done(null, false, {message:
'Credenciais inválidas!\n'})}
        return done(null, user)
      })
      .catch(erro => done(erro))
  })
))
```

Processamento da informação do utilizador

```
// Indica-se ao passport como serializar o utilizador
passport.serializeUser((user, done) => {
  // Serialização do utilizador. O passport grava o utilizador na sessão aqui.
  done(null, user.id)
})

// Desserialização: a partir do id obtem-se a informação do utilizador
passport.deserializeUser((uid, done) => {
  axios.get('http://localhost:5011/users/' + uid)
    .then(dados => done(null, dados.data))
    .catch(erro => done(erro, false))
})
```

POST /login

```
router.post('/login', passport.authenticate('local',  
  { successRedirect: '/protegida',  
    successFlash: 'Utilizador autenticado com sucesso!',  
    failureRedirect: '/login',  
    failureFlash: 'Utilizador ou password inválido(s)...' })  
)
```


GET /protegida

```
router.get('/protegida', (req,res) => {  
  if(req.isAuthenticated()){  
    res.send('Atingiste a área protegida!!!')  
  }  
  else{  
    res.redirect('/')  
  }  
})
```