

Programação & Serviços Web

React – Aula 2

2019/2020

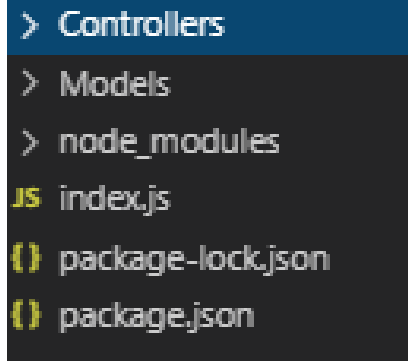
INTRODUÇÃO AO MONGODB

Aula 2

- Estrutura da Pasta
- Gravar / Ler Ficheiro
- Introdução o mongoDB

Estrutura da Pasta

- Como dito na aula anterior, nesta UC iremos usar o **expressJS** numa arquitetura de **Controlador** <=> **Modelo**, servindo apenas para gerir esta informação.
- Posto isto, um projeto **expressJS** pode ser composto por:



```
> Controllers
> Models
> node_modules
JS index.js
({} package-lock.json
({} package.json
```

Cada ficheiro na pasta **Models** corresponde a um tipo de estrutura do projeto (**Users**, **Produtos**, etc)

Cada ficheiro na pasta **Controllers** está relacionado com 1 ou mais tipos de estruturas

Incluir ficheiro da pasta dos controladores

- No ficheiro que criar, deverá colocar os endpoints dentro de um `module.exports`, tal como no exemplo seguinte:

```
module.exports = function (app, minhas_notas) {  
  app.get('/:id', (req, res) => {  
    if (minhas_notas[(req.params.id)] != null) {  
      res.send("" + minhas_notas[req.params.id])  
    } else {  
      return res.send("Erro")  
    }  
  })  
  //Adicionar outros endpoints  
}
```

Parâmetros que recebe
(neste caso são 2)

Pode colocar dentro do
`module.exports` todos os
endpoints que quiser.

Incluir ficheiro da pasta dos controladores

- Para chamar este ficheiro no index.js, basta chamar da seguinte forma

Importação do
ficheiro na pasta
Controllers,
enviando para lá
2 variáveis



```
app.use(express.json())

var minhas_notas = [15,17,19];
require("../Controllers/notas")(app, minhas_notas);

app.listen(port, () => console.log(`As minhas notas -  
Nuno Carapito ${port}!`))
```

Gravar/Ler de ficheiro

- A Leitura / Gravação de conteúdo é feita de uma forma muito simples. Começando pela leitura:

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

Declarar uma variável que vai carregar o “fs”, biblioteca que trata da leitura/gravação de conteúdo. Esta variável pode ser reutilizada na leitura e na escrita.

É importante não esquecer que o ficheiro deve ser gravado em JSON

Ler de ficheiro

- Após declarar a variável, procede-se à leitura do conteúdo do ficheiro.
 - O primeiro campo é referente ao nome do ficheiro.
 - O segundo é o formato (por norma é UTF8).
 - No terceiro campo, se o err for != de null, é porque foi lido com sucesso. Caso contrário, vai conter o erro. O contents é o conteúdo lido, que depois se pode associar a uma variável

```
fs.readFile('NOME_DO_FICHEIRO.TXT', 'utf8', function(err, contents) {  
    console.log(contents); // por exemplo: minhas_notas = contents  
});
```

Atenção: Ler de ficheiro

- Como foi dito na aula passada, o expressJS é assíncrono. Como tal, ele não vai ficar à espera da leitura do ficheiro. Deve fazer as seguintes alterações:

Adicionar o **async** para indicar que possamos utilizar a keyword “await”

```
module.exports = async function (app, minhas_notas) {
```

Coloca o código de Ler/Gravar ficheiro em funções separadas

```
function gravarNotas(data) {  
  fs.writeFile("temp.txt", data, (err) => {  
    if (err !== null) console.log(err);  
    console.log("Successfully Written to File.");  
  });  
}
```


Atenção : Ler de ficheiro

```
var minhas_notas = [];  
await lerNotas();  
  
function lerNotas() {  
  fs.readFile('temp.txt', 'utf8', function (err, contents) {  
    if (!err) {  
      minhas_notas = [contents.split(",")];  
    }  
    console.log(err);  
  });  
}
```

Na função Ler (**não é necessário no Gravar**)

Chama a função utilizando a palavra **await** de forma a execução esperar pela conclusão da função

Gravar para ficheiro

- Para gravar para ficheiro, o processo é semelhante,
 - O primeiro campo é referente ao nome do ficheiro.
 - O segundo campo é a variável que se pretende gravar em ficheiro.
 - O terceiro, err, se for != de null, é porque houve um erro na gravação e vai conter o erro. Caso contrário, é porque foi gravado com sucesso.

```
fs.writeFile("temp.txt", data, (err) => {  
  if (err != null) console.log(err);  
  console.log("Successfully Written to File.");  
});
```

Introdução ao MongoDB

Bases de Dados

- As bases de dados servem para guardar informação estruturada.
- Elas suportam vários tipos de operações:
 - Query
 - Insert
 - Update
 - Delete

Bases de Dados - NoSQL

- SQL (Structured Query Language) é usado nas bases de dados tradicionais
- O uso de bases de dados NoSQL estão a ficar mais populares visto resolverem problemas encontrados em bases de dados SQL

Categorias BD NoSQL

- Existem quatro categorias, baseados em:
 - Documentos (MongoDB);
 - Key-Value (Redis);
 - Column-Family (Cassandra);
 - Graphs (Neo4J).

Bases de Dados - Documentos

- **Document**: Um pedaço de informação
 - Por exemplo, um documento JSON: { “nome”: “Batman”, “idade”: 57 }
- **Collection**: Coleção de Documentos
- **Database**: Conjunto de Coleções

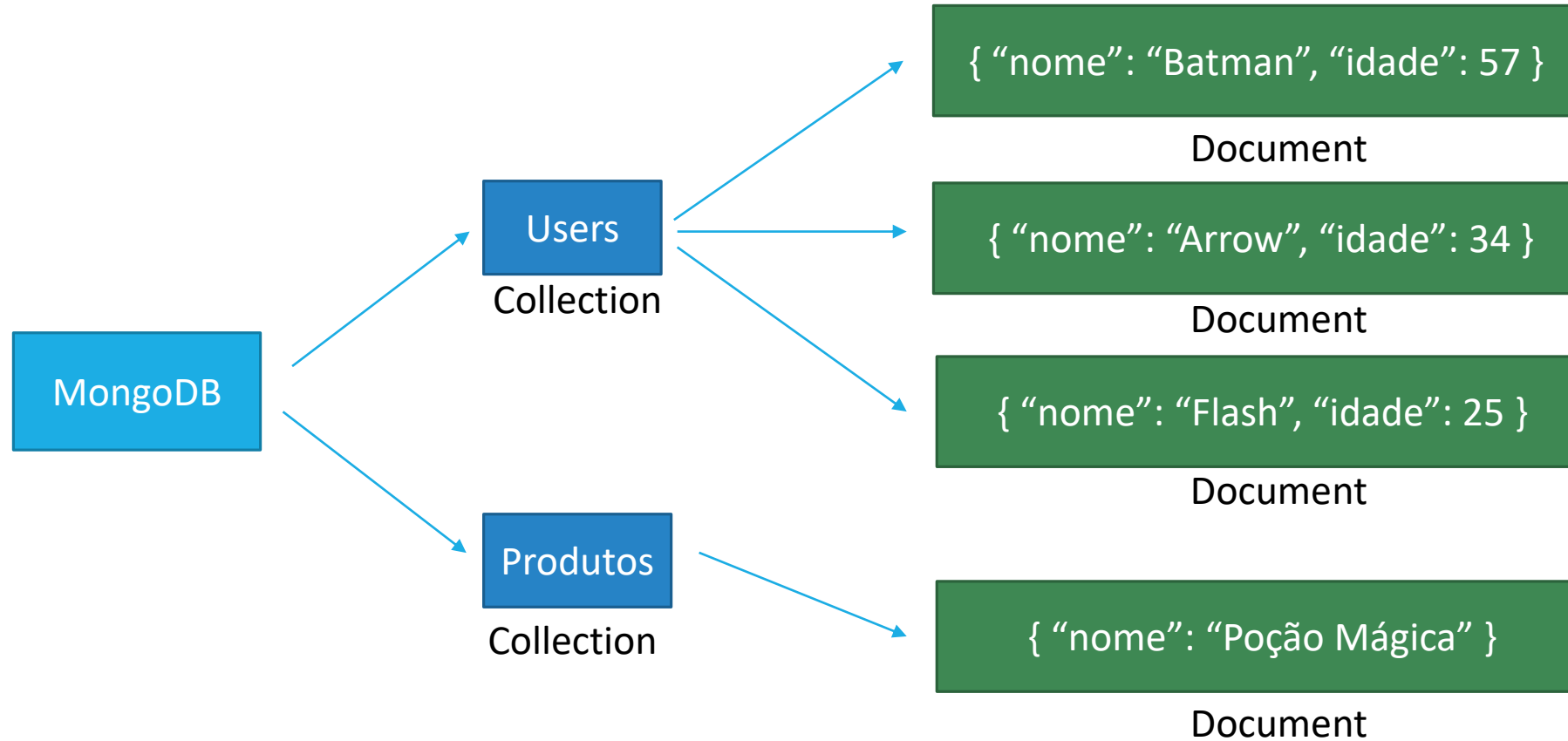
Porquê NoSQL?

- Escalabilidade
 - Disponibilidade
 - Consistência
 - Tolerante à partição (permite dividir em vários)
- Facilidade em deploy
 - Não é necessário fazer o mapeamento entre os vários documentos

MongoDB

- Base de dados NoSQL baseado em Documentos
- Consiste num conjunto de coleções
- Uma coleção tem vários documentos
- Cada documento é um ficheiro JSON

Resumindo a Estrutura



MongoDB - JSON

- Todos os documentos no MongoDB têm de ter o campo “_id” e que deve ser um valor único
- Quando se cria um documento novo, o campo _id é criado por defeito
- Por exemplo:

```
{  
  "_id": ObjectId("56ce74c0b02806eff4558f1f"),  
  "name": "Super-Herói",  
  "idade": 54  
}
```

Instalar o MongoDB

- Ir a www.mongodb.com
- No canto Direito, clicar em **Try Free**
- Escolher a opção **Server**
- Fazer o download da versão 4.2.1
- Na instalação, escolha a opção **Completa** e não mude nenhuma das outras opções.
- A instalação irá também instalar o **MongoDB Compass Community**, uma ferramenta para facilitar a gestão da BD

Utilização do MongoDB Compass

- Ao abrir a aplicação, não precisa de alterar nenhuma das configurações. Basta fazer o **Connect**
- Do lado esquerdo deve ter três Bases de dados (admin, config, local)
- A BD Local já deve ter uma collection criada. Para criar uma nova, é só clicar no botão do topo.

Utilização do MongoDB Compass

- Abrindo uma coleção, poderá filtrar os dados ou inserir um novo documento

Os documentos criados dentro de uma coleção devem ter todos a mesma estrutura (mesmos campos)

- Para filtrar os resultados, o conteúdo a inserir deve ser do estilo JSON

Integração MongoDB + ExpressJS

Integração MongoDB + ExpressJS

- A integração é muito simples. Comece por instalar o mongodb no projeto

```
npm install mongodb
```

- Este módulo permite com que a aplicação **expressJS** consiga comunicar com a base de dados do **MongoDB** e, conseqüentemente, inserir, apagar, atualizar ou pesquisar por documentos

Integração MongoDB + ExpressJS

1) Fazer a importação do MongoDB

```
Const MongoClient = require("mongodb").MongoClient;
```

2) Indicar o URL em que o servidor do MongoDB está inicializado e o nome da BD

```
const url = "mongodb://localhost:27017";  
const dbName = "local";
```

3) Fazer a ligação à base de dados

```
MongoClient.connect(url, (err, client) => {  
  if (err == null) { //Adicionar coisa}  
});
```

Integração MongoDB + ExpressJS

4) Ligar à **Base de Dados** definida

```
const db = client.db(dbName);
```

5) Ligar à **Collection** definida

```
const collection = db.collection("NomeCollectionCriada");
```

6) Utilizar a collection nos endpoints criados previamente

No Final da integração, deve ter algo assim

```
MongoClient.connect(url, (err, client) => {  
  if (err !== null) {  
    console.log("Connected correctly to server");  
    const db = client.db(dbName); //Ligar à Base de Dados Escolhida  
    const collection = db.collection("Teste"); //Ligar à collection  
    app.use(express.json())  
    require("./Controllers/notas")(app, collection); //Enviar a collection para o /users  
    app.listen(port, () => console.log(`As minhas notas - Nuno Carapito ${port}!`))  
  }  
});
```

Exemplos de Queries

- Para ir buscar todos os valores no get

```
app.get('/', (req, res) => {  
  collection.find().toArray().then(result => {  
    res.send(result)  
  })  
})
```

É necessário converter os dados para Array, utilizando o “ToArray()”

Exemplo de Queries

- Quando pretende adicionar um novo elemento:

```
app.post('/', (req, res) => {  
  collection.insertOne({ "name": req.body.name, "description": req.body  
.description }));  
  return res.status(200);  
})
```

Pesquise como fazer no caso de ser um PATCH / DELETE

Nota sobre MongoDB

- Todas as operações feitas no MongoDB são **Promises**. O que significa que o resultado não é imediato.
- Deve usar o **.then** para que, quando houver uma resposta, acontecer algo. Por exemplo:

```
app.get('/', (req, res) => {  
  console.log(collection.find().toArray().then(result => {  
    return res.send(result);  
  }));  
})
```

Ficha 2 - Moodle

Para submissão no final da aula