

Tecnologias para o Desenvolvimento Web

ExpressJS – Aula 3

2019/2020

Aula 3

- Middleware
- Rotas / Classe de URL
- Mongoose
- Mais que uma estrutura (Rotas)

Middleware

- Função que intercepta todos os pedidos efetuados à aplicação, podendo fazer algumas operações antes de executar o código pretendido
- Imagine que o seu projeto tem 1000 endpoints e quer adicionar um `console.log` a cada um deles. O que fazer?
 - Opção A: Ir aos 1000 endpoints e adicionar o `console.log`
 - Opção B: Usar um middleware para que, antes de executar o código da ação, fazer um `console.log`?

Middleware no ExpressJS

- No expressJS, adicionar um middleware é simples:

```
app.use(function (req, res, next) {  
  console.log("New request has been made")  
  next()  
})
```



Faz com que a execução do pedido feito continue

O middleware deve ficar no index.js, antes do listen (o listen deve ser sempre a linha de código final)

Middleware no ExpressJS

- Um exemplo de um middleware que pode ser útil é para a autenticação.
- Se o utilizador estiver autenticado, o servidor deve permitir o uso dos endpoints.
- Caso contrário, mostra mensagem a indicar que não há um login feito

```
app.use(function (req, res, next) {  
  if(req.body.accessToken != null){  
    next()  
  }else{  
    res.send("Not authenticated");  
  }  
})
```

Faz com que a execução continue

Rotas

Rotas

- Determina como é que uma aplicação responde a um request de um cliente a um determinado endpoint.
 - Um endpoint é determinado pelo URI e um HTTP request method específico.
- Nas aulas passadas já foi visto como é que pode definir várias rotas para o projeto
 - `app.get("URL", (req, res) => { console.log("Get Method")})`
 - `app.post("URL", (req, res) => { console.log("Post Method")})`
 - `app.put("URL", (req, res) => { console.log("Put Method")})`
 - `app.patch("URL", (req, res) => { console.log("Patch Method")})`
 - `app.delete("URL", (req, res) => { console.log("Delete Method")})`

Rotas

- No expressJS cada rota utiliza duas variáveis:
 - Req – (request) Representa o que o utilizador envia para o servidor
 - Res – (response) Representa o que o servidor vai devolver ao utilizador

GET http://localhost:4000/users

POST http://localhost:4000/users

GET http://localhost:4000/users/5

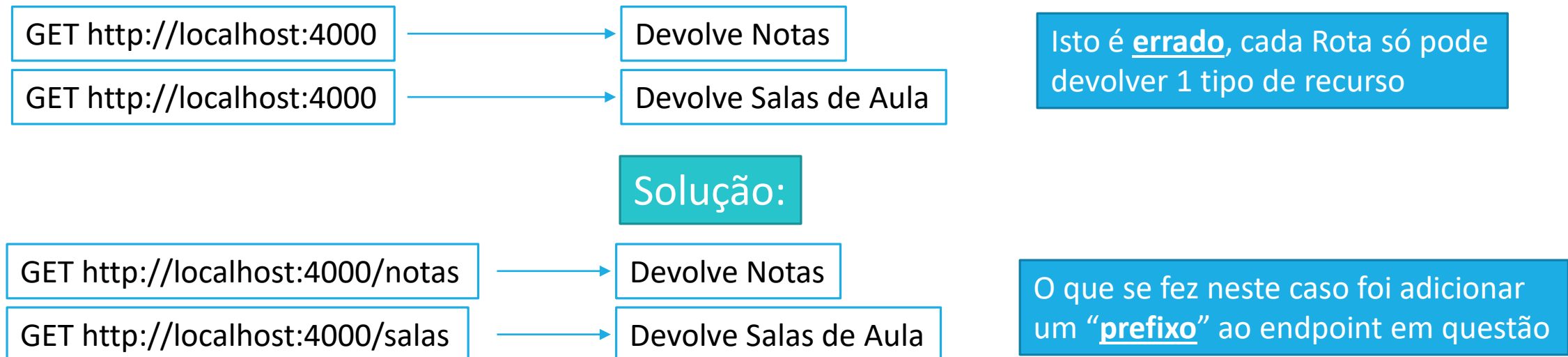
Um URL pode responder a diversos pedidos diferentes

GET http://localhost:4000/something/5

É importante que as rotas criadas identifique de forma clara o recurso com que esta a trabalhar

Rotas

- No seu projeto prático tem o controlador notas.js que contem os endpoints para o recurso Notas
- Imagine que pretende criar um novo recurso para guardar as salas de aula existentes no IPV.



Implementar Prefixos ExpressJS

1) No index.js, actualizar:

```
require("../Controllers/notas")(minhas_notas);
```



```
var notas = require("../Controllers/notas")(minhas_notas);
```

2) Logo após esta linha, adicionar

```
app.use('/notas', notas);
```

Isto associa o /notas a todos os endpoints dentro da variável

3) No Controllers Notas, no topo, fazer os seguintes imports:

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

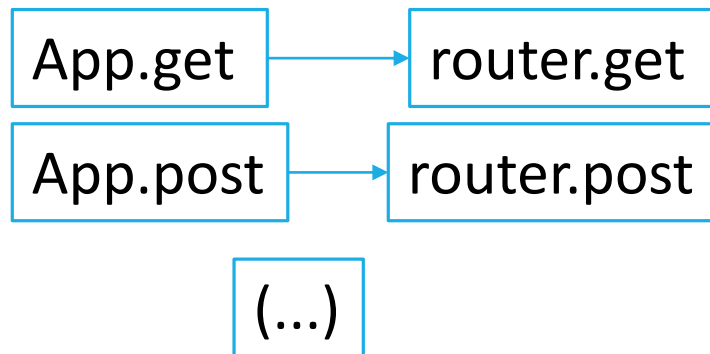
Implementar Prefixos ExpressJS

4) Remover o `async` da função

5) No final da função, adicionar

```
return router;
```

6) Mudar o **app** para **router**

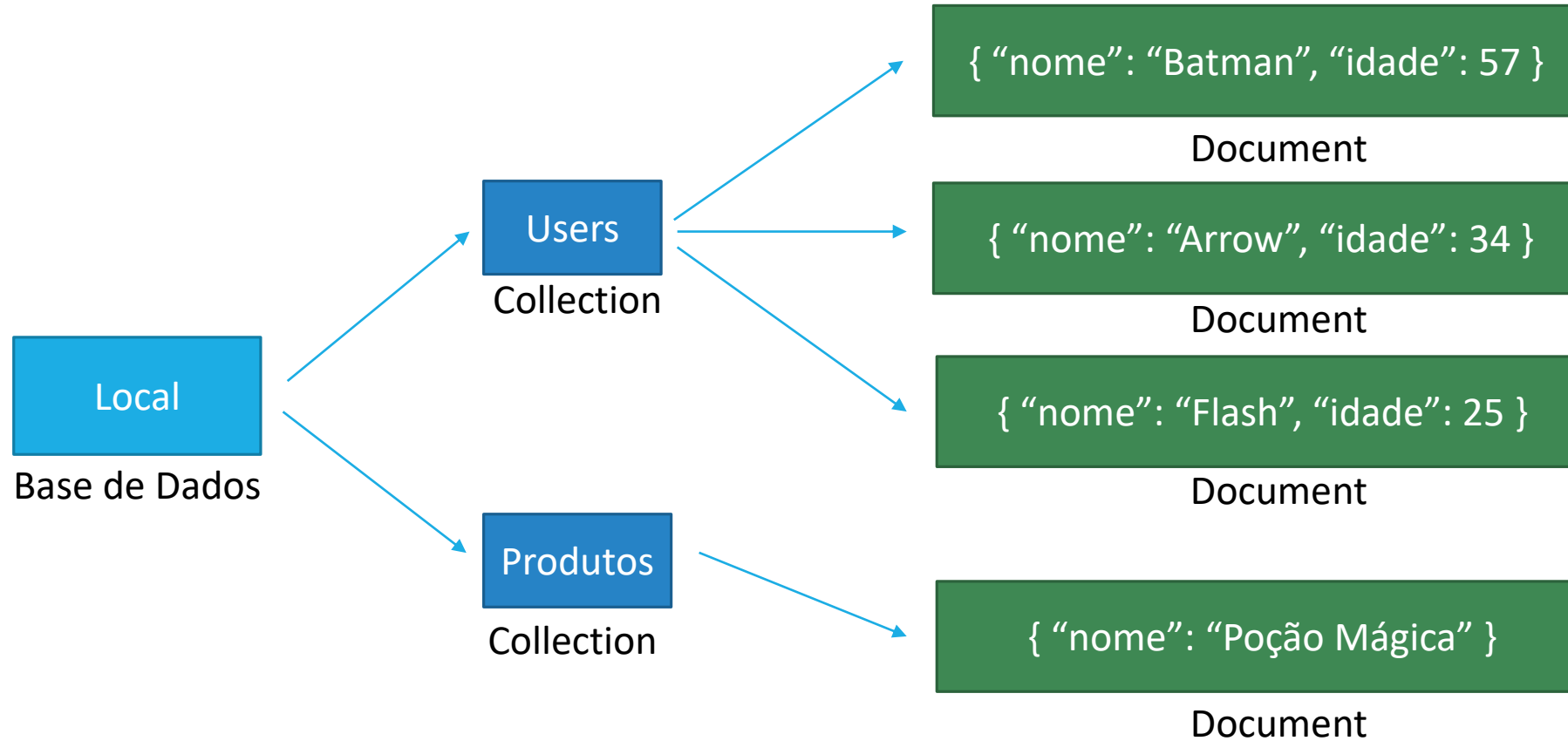


Como sempre, antes de fazer mais alterações, verifique que está tudo a funcionar.

Relembrar MongoDB

- O MongoDB guarda os dados em forma de documentos
- No **MongoDB Compass**, pode criar documentos com estruturas diferentes, algo que não é ideal.
 - A estrutura dos documentos depende da disciplina do developer
- Todos os documentos presentes numa collection devem ter a mesma estrutura (mesmos número de campos, campos do mesmo tipo, etc)

Relembrando a estrutura do MongoDB



Mongoose

Mongoose

- Mongoose ODM (Object Data Model) – Gere os dados dos objetos
- Adiciona uma **estrutura base** aos documentos do MongoDB através de **Schema**
- **Schema** – Usado para criar **um Modelo** para **cada Recurso**
 - Estrutura dos dados a serem guardados
 - Define os campos que o documento vai ter, e os seus tipos
 - Tipos permitidos: String, Number, Date, Buffer, Boolean, Mixed, ObjectId, Array
 - Os Schemas podem ser encadeados para permitir subdocumentos

Instalar Mongoose no Projeto

- Para instalar, basta correr o seguinte comando no terminal

```
npm install mongoose
```

- Os Schemas criados vão ser colocados na pasta Models.
- Os Schemas vão permitir que seja feita uma validação automática dos campos enviados pelo utilizador, poupando trabalho ao programador do Backend.

Exemplo Schema

```
const mongoose = require("mongoose");  
const Schema = mongoose.Schema;
```

Não esquecer dos imports

```
const commentSchema = new Schema({  
  rating: {  
    type: Number,  
    min: 1,  
    max: 5,  
    required: true  
  },  
  comment: {  
    type: String,  
    required: true  
  }  
}, {  
  timestamps: true  
})
```

No final de declarar o Schema, tem de se exportar para que se possa utilizar em outros ficheiros

```
var Dishes = mongoose.model("Dish", dishSchema);  
  
module.export = Dishes;
```

Adiciona os campos “createdAt” e “updatedAt” à base de dados de forma automática

Exemplo Schema encadado

```
const mongoose = require("mongoose");  
const Schema = mongoose.Schema;
```

Não esquecer dos imports

```
const dishSchema = new Schema({  
  name: {  
    type: String,  
    required: true,  
    unique: true  
  },  
  description: {  
    type: String,  
    required: true,  
  },  
  comments: [commentSchema]  
}, {  
  timestamp: true  
})
```

Vai buscar o Schema do Slide Anterior

Se estiver noutro ficheiro, é preciso fazer o **require**

No final têm de se exportar sempre, como no slide anterior

Schema e Collections

- Cada Schema que vá ter no seu projeto estará associado a uma collection.
- Ou seja, se tiver 5 schemas, terá 5 collections.

Integração do Moongoose

- Após criar os Schemas necessários, procede-se à sua integração

1) Importar o mongoose

```
const mongoose = require("mongoose");
```

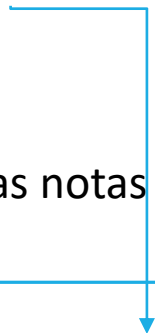
2) Substituir a ligação do MongoDB pela seguinte

```
const url = "mongodb://localhost:27017";  
const dbName = "local"; //Colocar o nome da Base de dados em Questão  
const connect = mongoose.connect(url, { dbName: dbName });
```

Integração do Moongoose

3) Substituir o MongoClient.connect por:

```
connect.then((db) => {  
  console.log("Connected correctly to server");  
  
  var notas = require("./Controllers/notas");  
  app.use('/notas', notas);  
  
  app.listen(port, () => console.log(`As minhas notas - Nuno Carapito ${port}!`))  
})
```



Ao fazer esta alteração, deixa de ser necessário enviar a variável **collection** para o Controlador Notas. Como foi feito previamente na alteração dos routes, também deixa de ser necessário enviar o **app**

Integração do Mongoose nos Controllers

Integração do Mongoose nos Controllers

Documentação Mongoose: <https://mongoosejs.com/docs/guide.html>

1) No Controlador das Notas, procede-se à importação do Modelo

```
const Notas = require("../Models/notas");
```

2) Em todos os endpoints, substitui-se o **collection** pelo nome da variável (neste caso, **Notas**)

collection.find → **Notas**.find

Integração do Mongoose no GET

No caso do GET, deixa de ser necessário o “toArray”. Em comentário pode ver o código antigo

```
router.get('/', (req, res) => {  
  // collection.find().toArray().then(result => {  
  //   return res.send(result);  
  // });  
  Notas.find({}).then(result => {  
    if (result != null) {  
      return res.status(200).send(result);  
    } else {  
      return res.status(400).send("Not Found")  
    }  
  })  
})
```


Integração do Mongoose no POST

Se faltar algum campo no body, o mongoose vai devolver um erro. Caso vá algum campo a mais, o mesmo é ignorado

```
router.post('/', (req, res) => {  
  if (req.body.nota != null) {  
    Notas.create(req.body).then((nota) => {  
      return res.status(200).send(nota);  
    }).catch((error) => {  
      return res.status(400).send(error.message);  
    })  
  }  
  return res.status(400);  
})
```

Integração do Mongoose no PATCH

O mongoose vai ignorar os campos que não estejam definidos no schema

```
router.patch('/:id', (req, res) => {  
  if (req.body != null && req.params != null) {  
    Notas.updateOne({ codigo: req.params.id }, req.body).then((nota) => {  
      return res.status(200).send(editNota);  
    }).catch((error) => {  
      return res.status(400).send(error.message );  
    })  
  }  
})
```

Veja como fazer para o endpoint do DELETE

Não se esqueça de testar sempre todas as alterações que faz

Ficha 3 - Moodle

Para submissão no final da aula