

# 1. Atividades Práticas - Roteamento básico no Express.js

---

Nesta atividade prática, vamos trabalhar com os conceitos roteamento básico no Express.js

## 1.1 Introdução

Em sistemas web modernos, uma API (Application Programming Interface) RESTful aguarda requisições HTTP vindas de um navegador da web (ou de outro tipo de cliente). Cabe lembrar que o REST (Representational State Transfer) trata-se de uma abstração da arquitetura da Web que utiliza dos verbos (GET, POST, DELETE, UPDATE, entre outros) do protocolo HTTP como base para as requisições. Assim, quando uma requisição HTTP é recebida, esta API descreve quais ações são necessárias com base no padrão de URL e, também, quais informações associadas estão contidas em dados obtidos via POST ou GET.

Conforme vimos anteriormente, o Express.js é um dos frameworks mais utilizados do Node.js. Resumidamente, o Express oferece uma estrutura web rápida, flexível e minimalista para aplicativos Node.js. Desta forma, o Express fornece [métodos](#) para definir manipuladores de rotas para todas as requisições HTTP, tais como: `get()`, `post()`, `put()`, `delete()`, `patch()`, entre outras.

Vejamos este funcionamento através dos exemplos abaixo:

## 1.2 Criando nosso Projeto

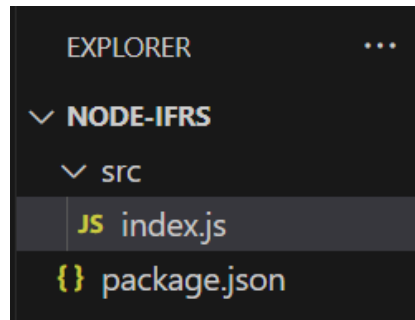
Vamos criar um diretório chamado `node-ifrs` que irá retornar e cadastrar informações sobre os cursos oferecidos no IFRS. Em seguida, vamos iniciar um projeto Node com auxílio do gerenciador de pacotes npm. Assim, digite os seguintes comandos no terminal:

```
cd node-ifrs
npm init -y
```

Agora que já temos o nosso projeto criado e configurado, vamos começar a criar nosso servidor web. Assim, vamos abrir o projeto no Visual Studio Code, digitando o seguinte comando no terminal, dentro do diretório do nosso projeto:

```
code .
```

Após, vamos criar o arquivo responsável por configurar e subir o servidor. Primeiro, crie uma pasta chamada src. Dentro desta pasta, crie um arquivo chamado index.js.



Em seguida, vamos realizar o download e instalação de nossas dependências do projeto. Neste caso, vamos utilizar o Express. Assim digite o seguinte comando no terminal, dentro do diretório do projeto:

```
npm i express
```

OBS: Você pode usar o terminal que fica integrado ao Visual Studio Code

Nós vamos utilizar em nosso projeto o [Nodemon](#). O Nodemon é um utilitário que monitora quaisquer mudanças em seu código fonte e automaticamente reinicia seu servidor. Assim, ele é perfeito para o nosso ambiente de desenvolvimento.

Vamos instalar o Nodemon via terminal como sendo uma dependência de desenvolvimento. Assim, pare o servidor e digite o seguinte comando:

```
npm install --save-dev nodemon
```

Agora, vamos configurar melhor o script de execução do Nodemon para o nosso projeto. Assim, faça as seguintes alterações no arquivo package.json:

```
{
  "name": "ifrs",
  "version": "1.0.0",
  "description": "",
  "main": "src/index.js",
  "scripts": {
    "dev": "nodemon",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
```

```
"dependencies": {
  "express": "^4.18.2"
},
"devDependencies": {
  "nodemon": "^3.0.1"
}
}
```

Inicialmente, alteramos o “main” informando o caminho do nosso arquivo principal do projeto com o seu respectivo diretório. Em seguida, criamos um “script” de desenvolvimento “dev” que irá executar o comando “nodemon”. O Nodemon irá executar automaticamente o arquivo principal definido em “main”.

### 1.2.1 Retornando todos os cursos

Vamos, agora, configurar o servidor web. Assim, abra o arquivo index.js e digite o seguinte código:

```
// Importa o modulo do Express
const express = require("express");
// Cria uma aplicação Express
const app = express();
//Define uma rota
app.get("/courses", (req, res) => {
  // Envia um retorno
  res.send("Cursos do IFRS");
});
// Inicia o servidor na porta '3000'
app.listen(3000, () => {
  // imprime um comentário de log no console
  console.log("Exemplo de aplicativo ouvindo a porta 3000");
});
```

Agora, vamos iniciar o nosso servidor web digitando no terminal o seguinte com

```
npm run dev
```

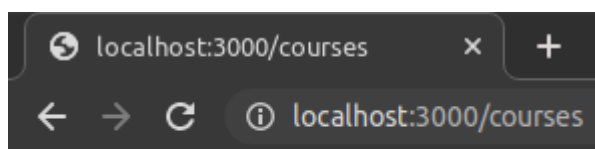
Observe que ele executou o script “dev” que acabamos de criar e executou o nodemon.

```
found 0 vulnerabilities
PS C:\Users\Mauri\Downloads\node-ifrs> npm run dev

> ifrs@1.0.0 dev
> nodemon

[nodemon] 3.0.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node src/index.js`
Exemplo de aplicativo ouvindo a porta 3000
```

Com o servidor em execução, você pode acessar o localhost:3000/courses em seu navegador para ver o exemplo de resposta retornado.



## Cursos do IFRS

Pronto. Agora podemos implementar novas funcionalidades neste projeto. Desta forma, uma vez que não estamos trabalhando com persistência de dados, vamos armazenar em memória uma listagem dos cursos que são oferecidos no IFRS. Ainda, vamos alterar a nossa rota atual para retornar estes cursos.

Veja como está o nosso arquivo index.js:

```
// Importa o módulo do Express
const express = require("express");

// Cria uma aplicação Express
const app = express();

// Define um array de cursos
let courses = [
  {
    id: "66e9aa36-eac0-4667-b0ba-324a4d39cdc9",
    name: "Análise e Desenvolvimento de Sistemas",
    type: "Superior",
  },
  {
    id: "ddd9c637-7bef-416c-b4d1-91d291fbc1c2",
```

```

    name: "Técnico em Informática",
    type: "Técnico",
  },
];

//Define a rota /courses para o verbo get
app.get("/courses", (req, res) => {
  // Retorna todo o array de courses
  res.json(courses);
});

// Inicia o servidor na porta '3000'
app.listen(3000, () => {
  // imprime um comentário de log no console
  console.log("Exemplo de aplicativo ouvindo a porta 3000");
});

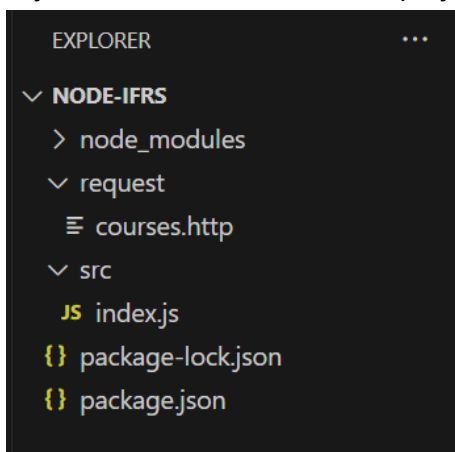
```

**OBS:** Cabe salientar que o Nodemon irá reiniciar nosso servidor Node.js a qualquer alteração que realizamos no nosso código fonte. Assim, uma vez que nossos cursos estão sendo armazenados em memória, iremos perder as alterações que fizemos no array `courses`.

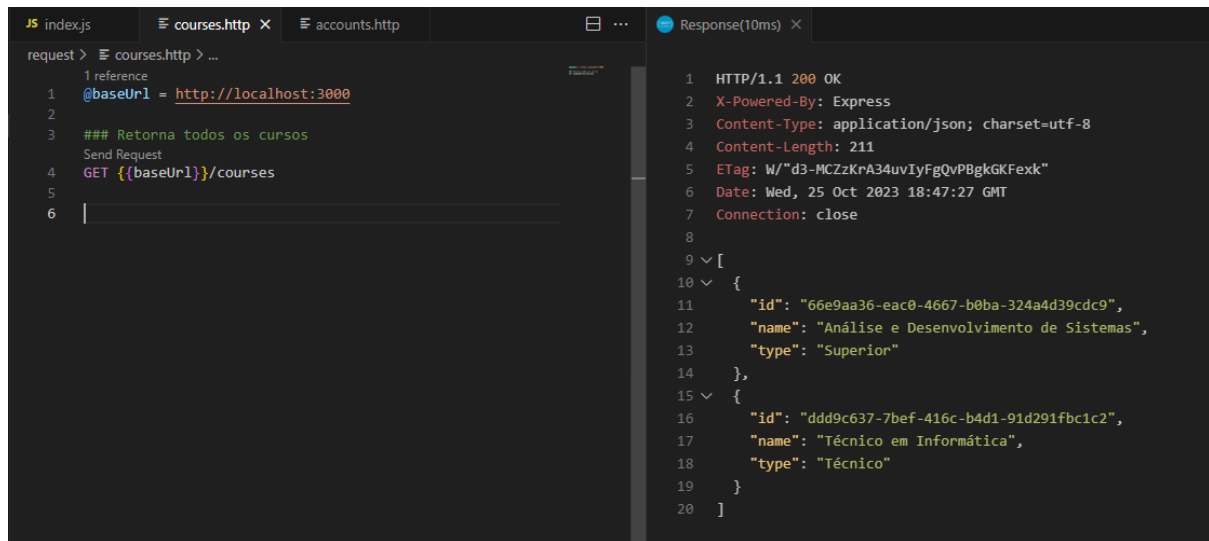
Agora, vamos testar este código com o Rest Client. Cabe lembrar que o Rest Client é uma extensão que permite executar requisições diretamente do Visual Studio Code, com o objetivo de testar APIs. Ele é bem simples de utilizar: basta abrir um arquivo e escrever o comando de requisição. O resultado aparecerá ao lado.

Neste contexto, abra o seu projeto (neste caso, o node-express) e crie uma pasta nome "request". Em seguida, dentro desta pasta, crie um arquivo chamado `courses.http`.

Veja como está a estrutura do projeto:



Assim, no arquivo recém criado, vamos escrever a requisição para retornar todos os cursos.



```
request > courses.http > ...
1 reference
1 @baseUrl = http://localhost:3000
2
3 ### Retorna todos os cursos
4 Send Request
5 GET {{baseUrl}}/courses
6

1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Content-Type: application/json; charset=utf-8
4 Content-Length: 211
5 ETag: W/"d3-MCZzKrA34uvIyFgQvPBgkGKFexk"
6 Date: Wed, 25 Oct 2023 18:47:27 GMT
7 Connection: close
8
9 [
10 {
11   "id": "66e9aa36-eac0-4667-b0ba-324a4d39cdc9",
12   "name": "Análise e Desenvolvimento de Sistemas",
13   "type": "Superior"
14 },
15 {
16   "id": "ddd9c637-7bef-416c-b4d1-91d291fbc1c2",
17   "name": "Técnico em Informática",
18   "type": "Técnico"
19 }
20 ]
```

Como resultado, ao consultarmos a rota `http://localhost:3000/courses`, teremos todas as contas em nosso array `courses`

### 1.2.2 Retornando um curso específico

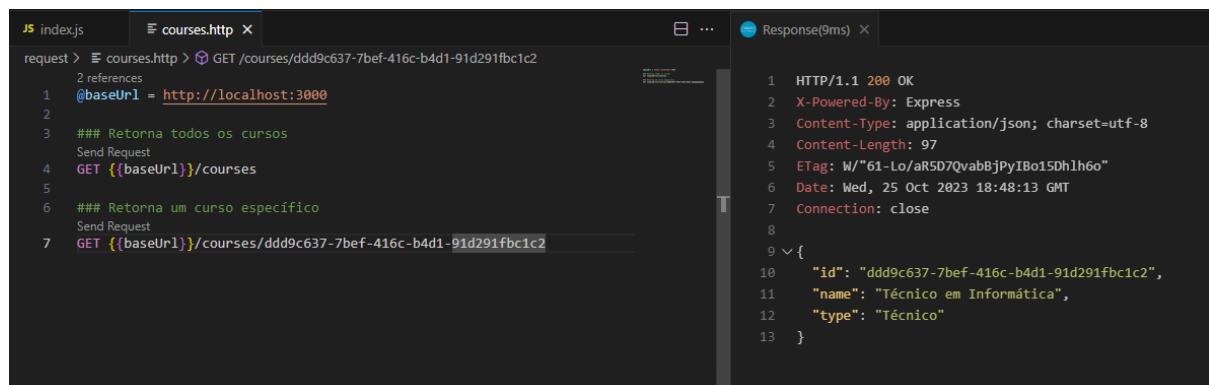
Agora, vamos criar em `index.js` uma rota que permita retornar um curso específico através de seu `id`:

```
...
// Define a rota /courses com o parâmetro id para o verbo get
app.get("/courses/:id", (req, res) => {
  // Uso Desestruturação para obter o parametro id da requisição
  const { id } = req.params;
  // Verifica se este id existe no array de courses
  const getCourse = courses.find((course) => course.id === id);
  if (!getCourse) {
    res.status(500).json({ error: "Curso não encontrado!" });
  } else {
    res.json(getCourse);
  }
});
...
```

Observe que fizemos uso da [Desestruturação de objetos](#) para extrair o `id` do objeto de requisição (`res`). Lembre-se de que Desestruturação (ou destructuring) é uma funcionalidade do JavaScript ES6 que nos permite extrair dados de arrays, objetos e até

retorno de funções, em variáveis distintas. Ainda, cabe lembrar que o método `find()` retorna o valor do primeiro elemento do array que satisfizer a função de teste provida. Caso contrário, `undefined` é retornado. Assim, ele percorre o array de `courses` até encontrar um registro (`course`) com a propriedade `id` com valor e tipo igual ao da constante `id`.

Assim, agora podemos usar nossa API para filtrar por um `id` específico que irá cair na rota definida por `/courses/:id`. Assim, se informarmos a URL `http://localhost:3000/courses/ddd9c637-7bef-416c-b4d1-91d291fbc1c2`, teremos como retorno apenas um curso que corresponde ao parâmetro `id = ddd9c637-7bef-416c-b4d1-91d291fbc1c2`:



The screenshot shows a VS Code editor with a file named `index.js` and a REST client tab named `courses.http`. The REST client shows a GET request to `http://localhost:3000/courses/ddd9c637-7bef-416c-b4d1-91d291fbc1c2`. The response is a JSON object with the following structure:

```
{
  "id": "ddd9c637-7bef-416c-b4d1-91d291fbc1c2",
  "name": "Técnico em Informática",
  "type": "Técnico"
}
```

### 1.2.3 Criando um novo curso

Agora, vamos criar em `index.js` uma rota que permita criar um novo. Porém, conforme podemos observar, o campo de `id` nos cursos é um identificador único universal (do inglês `universally unique identifier - UUID`). Assim, `UUID` é um identificador universalmente exclusivo utilizado para identificação de qualquer coisa no mundo da computação. O `UUID` é um número de 128 bits representado por 32 dígitos hexadecimais, exibidos em cinco grupos separados por hífens, na forma textual 8-4-4-4-12 sendo um total de 36 caracteres (32 caracteres alfanuméricos e 4 hífens). Por exemplo: `3d0ca315-aff9-4fc2-be61-3b76b9a2d798`

Desta forma, vamos importar a biblioteca `uuid` em nosso projeto. Assim, pare o serviço do Node se ele estiver em execução (`CTRL+C`) e digite o seguinte comando no terminal:

```
npm i uuid
```

Agora, devo adicionar a importação deste módulo no arquivo `index.js`:

```
// Importa o módulo do Express
const express = require("express");
// importa somente a função uuidv4 do módulo uuid
const { v4: uuidv4 } = require('uuid');
...
```

Observe que usamos a Desestruturação de Objetos para importar somente a função `uuidv4` que pertence ao módulo `uuid`.

Antes de adicionar uma nova rota usando `app.post()`, que irá usar o corpo (body) da requisição para adicionar novos itens ao nosso array `courses`, devemos nos lembrar que o Node.js não sabe converter os dados da requisição para o formato JSON que queremos. Para analisar o conteúdo do corpo (body) da requisição, precisaremos instalar e configurar o middleware [body-parser](#). O `body-parser` é um módulo do Node.js capaz de converter o body da requisição para vários formatos, dentre eles, o JSON.

Dessa forma em `index.js`, precisamos dizer para o Express que ele deve usar o middleware `body-parser` para converter os dados da requisição.

```
// Importa o módulo do Express
const express = require("express");

// importa somente a função uuidv4 do módulo uuid
const { v4: uuidv4 } = require('uuid');

// Cria uma aplicação Express
const app = express();

// Configura o parser para requisições com JSON
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
```

Agora sim, podemos criar a rota para criar um novo curso em nosso array `courses`. Logo, crie o seguinte código:

```
...
app.post("/courses", (req, res) => {
  // Usa Desestruturação para pegar os campos enviados pelo body
  const { name, type } = req.body;
  // Adiciona um novo UUID
  const newCourse = { id: uuidv4(), name, type };
  // Acrescenta este novo curso no final do array
  courses.push(newCourse);
  // Retorna o novo curso criado
  res.json(newCourse);
});
...
```

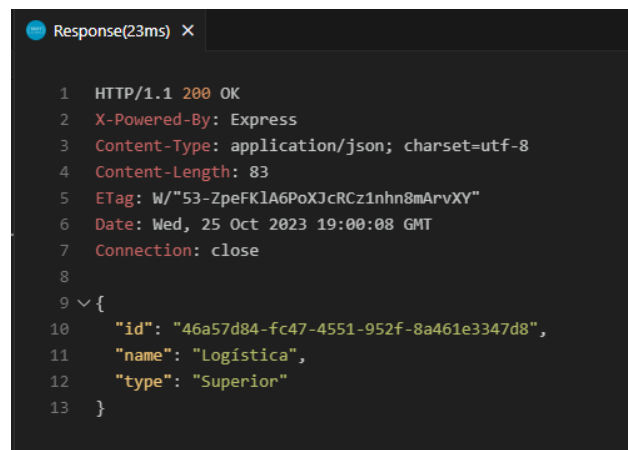


Agora sim, podemos voltar para o arquivo `courses.http` e criar uma nova requisição do tipo POST trabalhando com dados JSON. A URL será a mesma (<http://localhost:3000/courses/>) apenas teremos a opção de informar os campos que queremos enviar para a API.

```
### Cria um novo curso
POST {{baseUrl}}/courses
Content-Type: application/json

{
  "name": "Logística",
  "type": "Superior"
}
```

Como estamos retornando somente o novo curso criado com `res.json(newCourse)`, o body retornado é:



```
Response(23ms) X
1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Content-Type: application/json; charset=utf-8
4 Content-Length: 83
5 ETag: W/"53-ZpeFKIA6PoXJcRCz1nhn8mArvXY"
6 Date: Wed, 25 Oct 2023 19:00:08 GMT
7 Connection: close
8
9 {
10   "id": "46a57d84-fc47-4551-952f-8a461e3347d8",
11   "name": "Logística",
12   "type": "Superior"
13 }
```

## 1.2.4 Modificando um curso

Podemos editar/atualizar uma conta específica definindo uma rota para o verbo PUT. Isso é semelhante ao que fizemos com outras rotas até agora. Assim, altere o arquivo `index.js`:

```
...
// Define a rota /courses com o parâmetro id para o verbo put
app.put("/courses/:id", (req, res) => {
  // Busca o parametro id da requisição
  const { id } = req.params;
  // Pega os dados vindos via body da requisição
  const body = req.body;
  // Verifica se este id existe no array de courses e retorna seu índice
  const courseIndex = courses.findIndex((course) => course.id === id);
```

```

if (courseIndex < 0) {
  return res.status(500).json({ error: "Curso não encontrado!" });
}
// Cria um registro com os dados atualizados
const updatedCourse = { id, ...body };
// Sobrescreve o registro no array
courses[courseIndex] = updatedCourse;
// Retorna o registro atualizado
res.json(updatedCourse);
});
...

```

Agora podemos alterar os itens dentro do array `courses`. É uma implementação simples que apenas sobrescreve o objeto inicial com esta nova adição. Veja que usamos o Spread Operator (...), que costuma ser utilizado com frequência principalmente quando queremos criar um objeto novo a partir de um objeto existente.

Assim, no arquivo `courses.http` vamos criar uma requisição para o método HTTP PUT. Vamos definir o URL da requisição como `http://localhost:3000/courses/66e9aa36-eac0-4667-b0ba-324a4d39cdc9` e o corpo da solicitação JSON como:

```

### Altera um curso específico
PUT {{baseUrl}}/courses/66e9aa36-eac0-4667-b0ba-324a4d39cdc9
Content-Type: application/json

{
  "name": "Curso de ADS",
  "type": "Superior"
}

```

Após sua execução, podemos ver pelo corpo da resposta que `name` foi atualizado para "Curso de ADS" para o curso com `id = 66e9aa36-eac0-4667-b0ba-324a4d39cdc9`:

```

Response(6ms) X
1  HTTP/1.1 200 OK
2  X-Powered-By: Express
3  Content-Type: application/json; charset=utf-8
4  Content-Length: 85
5  ETag: W/"55-/8VBcceBwS0a90ai60L8JnyiklI"
6  Date: Wed, 25 Oct 2023 19:05:54 GMT
7  Connection: close
8
9  {
10   "id": "66e9aa36-eac0-4667-b0ba-324a4d39cdc9",
11   "name": "Curso de ADS",
12   "type": "Superior"
13 }

```

### 1.2.5 Excluir um curso

Os cursos podem ser excluídos do array usando `app.delete()`. Vamos dar uma olhada em como podemos implementar isso dentro de nosso aplicativo:

```
...
// Define a rota /courses com o parâmetro id para o verbo delete
app.delete("/courses/:id", (req, res) => {
  // Busca o parametro id da requisição
  const { id } = req.params;

  // Verifica se este id existe no array de courses e retorna seu índice
  const courseIndex = courses.findIndex((course) => course.id === id);
  if (courseIndex < 0) {
    return res.status(500).json({ error: "Curso não encontrado!" });
  }
  // Remove o curso do array
  courses.splice(courseIndex, 1);
  res.status(200).json({ msg: "Curso excluído com sucesso!" });
});
...
```

O método `splice()` altera o conteúdo de uma lista, adicionando novos elementos enquanto remove elementos antigos. O primeiro parâmetro desta função informa o Índice o qual deve iniciar a alterar a lista. O segundo argumento indica o número de elementos que devem ser removidos (se você não informar algum número, então todos os elementos até o fim da lista serão excluídos).

Assim, no arquivo `courses.http`, vamos criar uma requisição para o método HTTP DELETE. Assim, se enviarmos uma requisição DELETE para `http://localhost:3000/courses/66e9aa36-eac0-4667-b0ba-324a4d39cdc9`, isso removerá a conta com o `id = 66e9aa36-eac0-4667-b0ba-324a4d39cdc9` do array `courses`.

```
### Exclui um curso específico
DELETE {{baseUrl}}/courses/66e9aa36-eac0-4667-b0ba-324a4d39cdc9
```

Veja a mensagem de retorno:

```
Response(9ms) X
1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Content-Type: application/json; charset=utf-8
4 Content-Length: 38
5 ETag: W/"26-L8FufZ4YcKxfzjg9PqAADW9Zsdw"
6 Date: Wed, 25 Oct 2023 19:09:15 GMT
7 Connection: close
8
9 {
10   "msg": "Curso excluído com sucesso!"
11 }
```

Pronto! Finalizamos os exemplos desta atividade prática.