

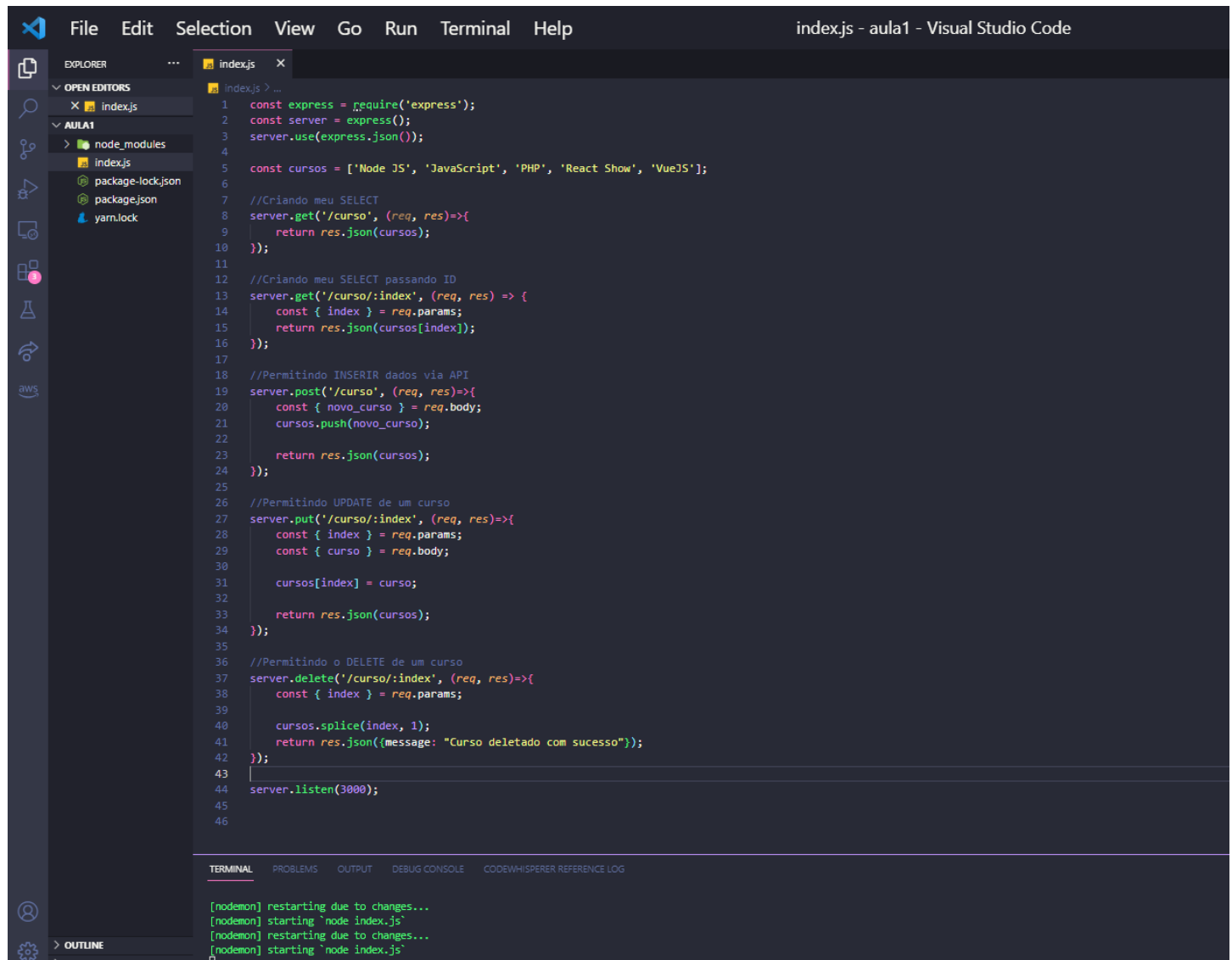
## Aula 3 – Middleware e API

Objetivo da aula:

- Middleware
- Documentação básica de API
- Exercícios sobre Middleware

# Aplicação Node atual

1 – Na última aula montamos criamos a estrutura base de um servidor Back-End para API. Nossa API permite chamadas GET, POST, PUT e DELETE de um cliente, como podemos ver abaixo.



The screenshot shows the Visual Studio Code interface with the 'index.js' file open. The Explorer sidebar on the left shows the project structure, including 'index.js', 'package-lock.json', 'package.json', and 'yarn.lock'. The main editor displays the following JavaScript code:

```
1  const express = require('express');
2  const server = express();
3  server.use(express.json());
4
5  const cursos = ['Node JS', 'JavaScript', 'PHP', 'React Show', 'VueJS'];
6
7  //Criando meu SELECT
8  server.get('/curso', (req, res)=>{
9    return res.json(cursos);
10 });
11
12 //Criando meu SELECT passando ID
13 server.get('/curso/:index', (req, res) => {
14   const { index } = req.params;
15   return res.json(cursos[index]);
16 });
17
18 //Permitindo INSERIR dados via API
19 server.post('/curso', (req, res)=>{
20   const { novo_curso } = req.body;
21   cursos.push(novo_curso);
22   return res.json(cursos);
23 });
24
25 //Permitindo UPDATE de um curso
26 server.put('/curso/:index', (req, res)=>{
27   const { index } = req.params;
28   const { curso } = req.body;
29
30   cursos[index] = curso;
31
32   return res.json(cursos);
33 });
34
35 //Permitindo o DELETE de um curso
36 server.delete('/curso/:index', (req, res)=>{
37   const { index } = req.params;
38
39   cursos.splice(index, 1);
40   return res.json({message: "Curso deletado com sucesso"});
41 });
42
43 server.listen(3000);
44
45
46
```

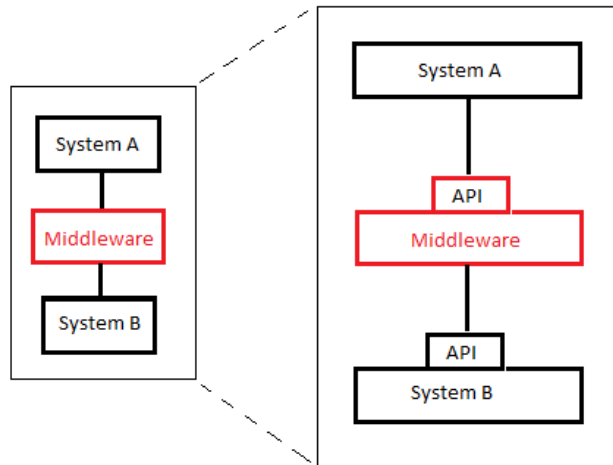
The terminal at the bottom shows the following output:

```
[nodemon] restarting due to changes...
[nodemon] starting 'node index.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node index.js'
```

## Middleware

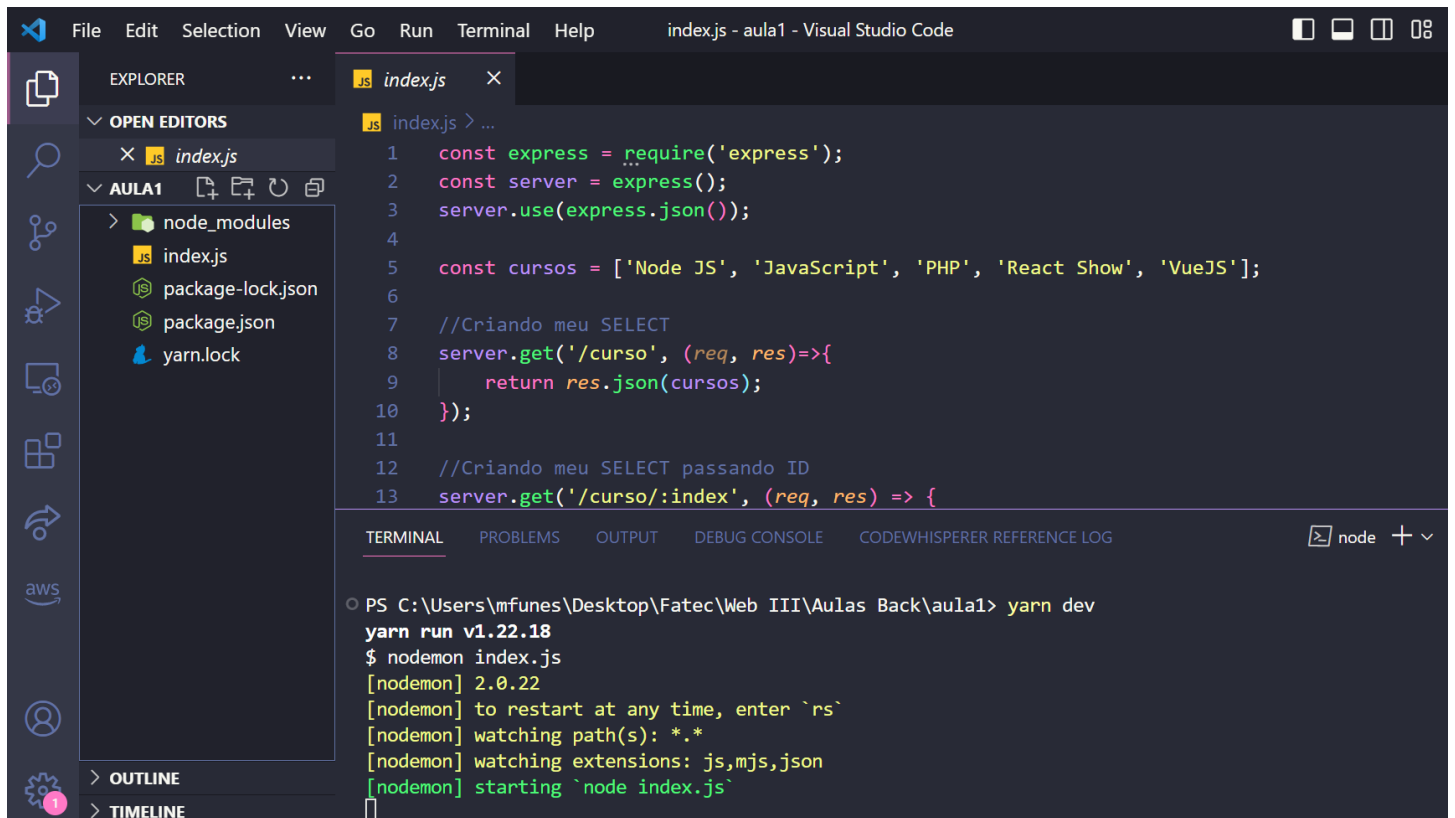
No contexto de nossa aplicação, podemos entender que estamos contruindo um Middleware. Ou sejam, dizendo em termos muito simples ***uma camada no meio*** de duas aplicações, ou seja, uma camada que ajuda duas aplicações, partes, sistemas, a se comunicarem.

Serve para facilitar ou tornar viável essa comunicação. Pensamos em interfaces, converter dados e formatos, prover protocolos, dentre outros. Serve então para prover ou facilitar o a comunicação entre duas aplicações, que podem ser de plataformas diferentes e tecnologias diferentes. Como mostra a imagem abaixo:



Mas como seria o código de Middlewares no NodeJS? Veremos nesse aula...

02 – Finalizamos a aula passada tendo o seguinte back-end rodando com **yarn dev** pelo **nodemon**.



The screenshot shows the Visual Studio Code interface with the following components:

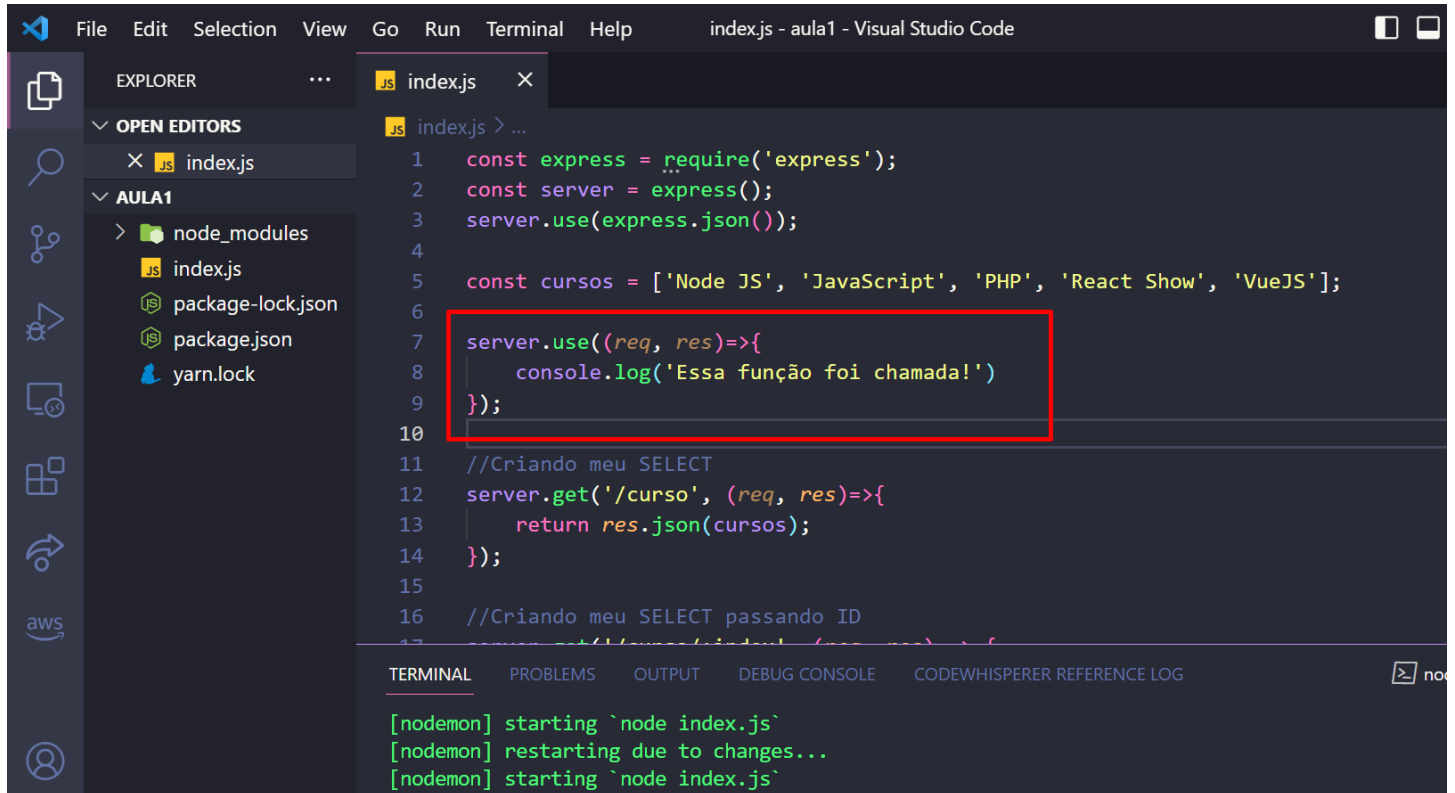
- EXPLORER:** Displays the file structure of the project 'AULA1', including 'node\_modules', 'index.js', 'package-lock.json', 'package.json', and 'yarn.lock'.
- index.js:** Contains the following code:

```
1  const express = require('express');
2  const server = express();
3  server.use(express.json());
4
5  const cursos = ['Node JS', 'JavaScript', 'PHP', 'React Show', 'VueJS'];
6
7  //Criando meu SELECT
8  server.get('/curso', (req, res)=>{
9    |   return res.json(cursos);
10  });
11
12  //Criando meu SELECT passando ID
13  server.get('/curso/:index', (req, res) => {
```
- TERMINAL:** Shows the command execution:

```
PS C:\Users\mfunes\Desktop\Fatec\Web III\Aulas Back\aula1> yarn dev
yarn run v1.22.18
$ nodemon index.js
[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
```

03 – Mas se nosso servidor já permite usar **GET, POST, PUT e DELETE** o que mais um back-end pode fazer?

Nem só de chamadas API é feito um servidor back-end, podemos criar funções que independente de uma rota a função será chamada. Vamos criar uma função que independente se for GET ou POST ela também será ativada, como no código abaixo:



```
index.js - aula1 - Visual Studio Code

EXPLORER
  OPEN EDITORS
    index.js
  AULA1
    node_modules
    index.js
    package-lock.json
    package.json
    yarn.lock

index.js
1  const express = require('express');
2  const server = express();
3  server.use(express.json());
4
5  const cursos = ['Node JS', 'JavaScript', 'PHP', 'React Show', 'VueJS'];
6
7  server.use((req, res)=>{
8    console.log('Essa função foi chamada!')
9  });
10
11 //Criando meu SELECT
12 server.get('/curso', (req, res)=>{
13   return res.json(cursos);
14 });
15
16 //Criando meu SELECT passando ID
17 server.get('/curso/:id', (req, res)=>{
18   //...
19 });

TERMINAL
[nodemon] starting `node index.js`
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
```

## 04 – Faça uma chamada no endpoint do servidor, veja que não houve resposta.

← → ↺

reqbin.com

🔑 🔍 📄 ☆ 🛠️ ⚙️ 📱 👤

REQBIN

Curl Python JavaScript Node.js PHP Java JSON XML

Contact Account

EXAMPLES SAVED

POST Request Example

Bearer Token Auth Example

HTML Form POST Example

GET Request Example

REST API POST Example

PUT Request Example

Auth Bearer Header Example

POST API Example

POST JSON Example

JSON Payload Example

Content-Length Example

REST API GET Example

Send Cookies Example

POST XML Example

JSON Response Example

POST JSON with Bearer Token

Curl POST Request Example

Curl GET Request Example

Curl POST JSON Example

Curl POST Body Example

Curl Basic Auth Example

Curl Bearer Token Example

Curl Download File Example

Curl Ignore SSL Checks

JavaScript POST Example

JavaScript GET Example

JavaScript Fetch JSON

JavaScript Array Join Example

JavaScript String Contains

JavaScript Sum Array

JavaScript Substring Example

Online REST & SOAP API Testing Tool

ReqBin is an online API testing tool for REST and SOAP APIs. Test API endpoints by making API requests directly from your browser. Test API responses with built-in JSON and XML validators. Load test your API with hundreds of simulated concurrent connections. Generate code snippets for API automation testing frameworks. Share and discuss your API requests online.

📄 File ↕ Generate Code 🛠️ Tools

localhost:3000/curso

GET EXT Send

Authorization

Content

Headers

Raw (2)

☒ Bearer Token ☐ Basic Auth ☐ Custom ☐ No Auth

Token

The authorization header will be automatically generated when you send the request. Read more about HTTP Authentication.

ReqBin Chrome Extension

Request timeout.

What is API?

API (Application Programming Interface) is a computing interface that defines how software components interact with each other. It is a way of programmatically interacting with a separate software component or resource and expose functionality for internal or external use and testing. API defines what requests can be made, how they will be made and hides complexity from developers. API extends systems to partners, organizes code, and makes components reusable.

ADVERTISEMENT

Hello??

Do you hear me?

Don't let choppy VoIP cut you off

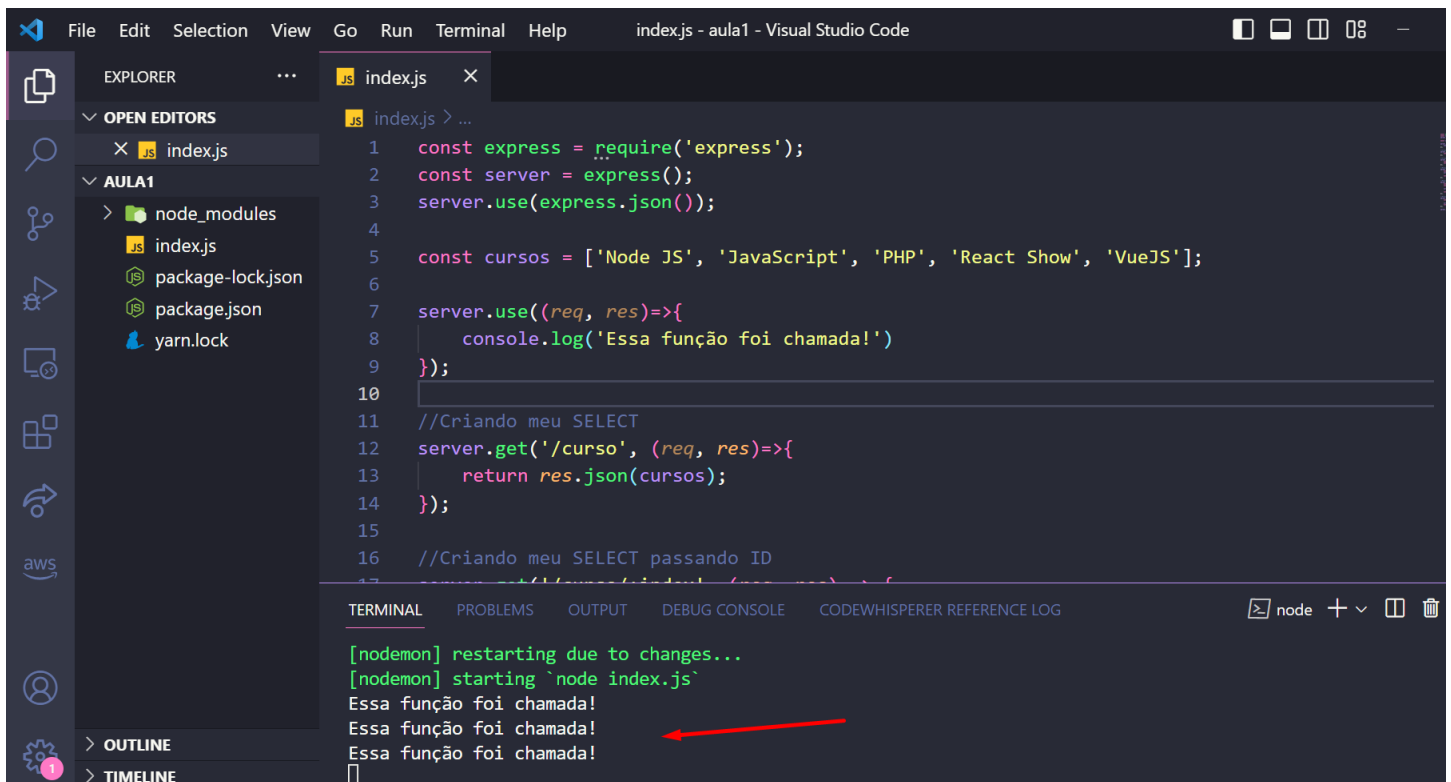
Monitor VoIP Quality with Obkio

Start Now

eZoiC

report this ad

05 – Mas no console do servidor podemos ver que essa função foi sim chamada. Aqui demonstra que alguém requisitou algo do servidor 3 vezes.



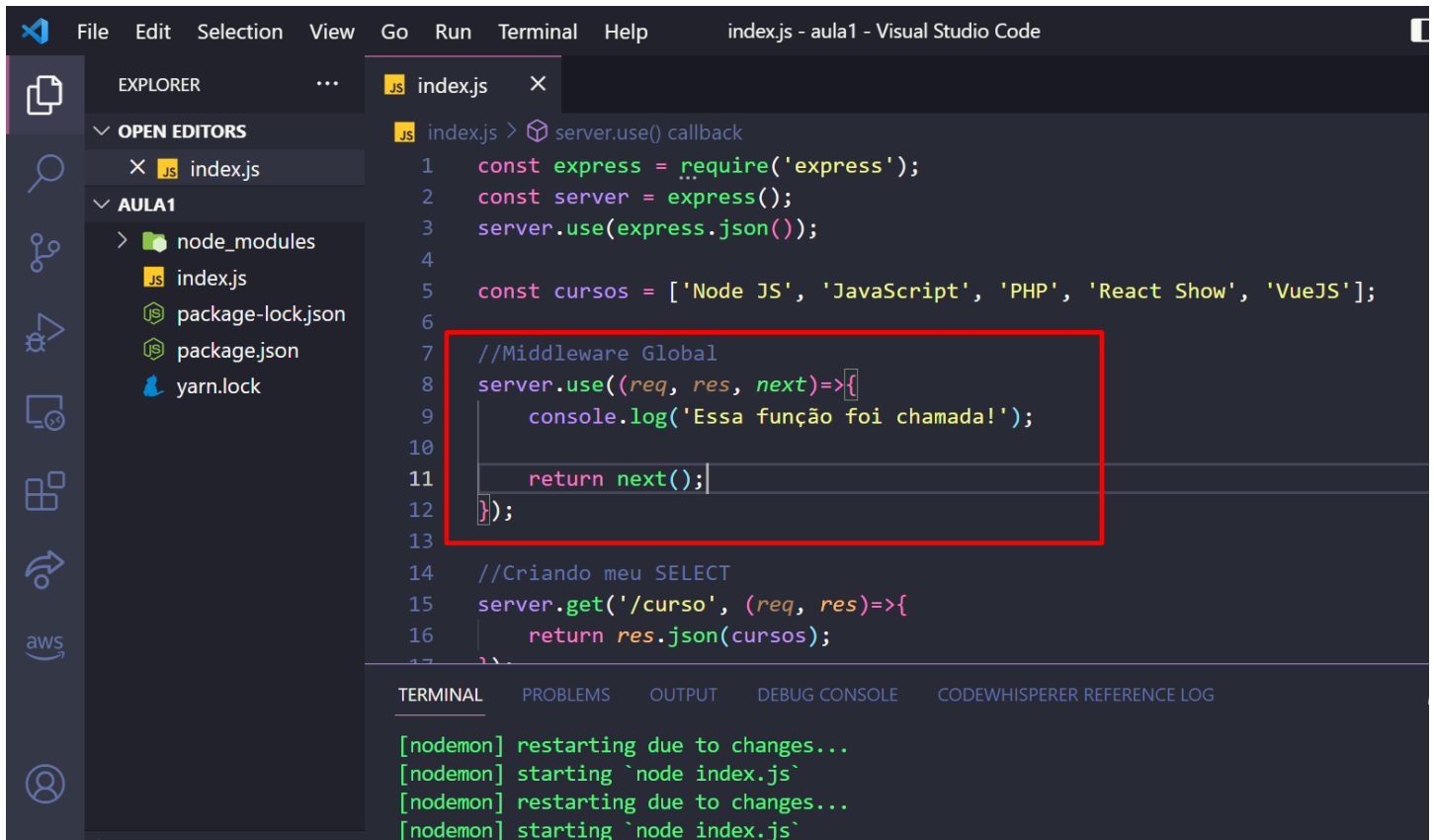
```
File Edit Selection View Go Run Terminal Help index.js - aula1 - Visual Studio Code

EXPLORER
OPEN EDITORS
  X JS index.js
AULA1
  > node_modules
    JS index.js
    package-lock.json
    package.json
    yarn.lock

1  const express = require('express');
2  const server = express();
3  server.use(express.json());
4
5  const cursos = ['Node JS', 'JavaScript', 'PHP', 'React Show', 'VueJS'];
6
7  server.use((req, res)=>{
8    console.log('Essa função foi chamada!')
9  });
10
11 //Criando meu SELECT
12 server.get('/curso', (req, res)=>{
13   return res.json(cursos);
14 });
15
16 //Criando meu SELECT passando ID
17 server.get('/:id', (req, res)=>{
18   //...
19 });

TERMINAL
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
Essa função foi chamada!
Essa função foi chamada!
Essa função foi chamada!
```

05 – O que fizemos aqui foi criar um Middleware Global, ou seja, independente da requisição ou rota sempre essa função ficará no meio do cliente e do banco de dados. Porém, ela está bloqueando as requisições, para fazer com que as requisições passem e continuem após nosso Middleware ser chamada, acrescente o parâmetro next e seu retorno, como abaixo:



```
File Edit Selection View Go Run Terminal Help index.js - aula1 - Visual Studio Code

EXPLORER
  OPEN EDITORS
    index.js
  AULA1
    node_modules
    index.js
    package-lock.json
    package.json
    yarn.lock

index.js
1  const express = require('express');
2  const server = express();
3  server.use(express.json());
4
5  const cursos = ['Node JS', 'JavaScript', 'PHP', 'React Show', 'VueJS'];
6
7  //Middleware Global
8  server.use((req, res, next)=>{
9    console.log('Essa função foi chamada!');
10
11    return next();
12  });
13
14  //Criando meu SELECT
15  server.get('/curso', (req, res)=>{
16    return res.json(cursos);
17  });

TERMINAL
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
```



## 05 – Faça uma chamada novamente, por exemplo, GET e veja que nosso back-end retorno a requisição para o cliente.

The screenshot shows the ReqBin website, an online REST & SOAP API testing tool. The interface includes a sidebar with various example requests, a main workspace for creating and testing API requests, and a right-hand panel for viewing response details and raw data. A red arrow points to the 'Send' button, indicating the action to execute the request.

**Online REST & SOAP API Testing Tool**

ReqBin is an online API testing tool for REST and SOAP APIs. Test API endpoints by making API requests directly from your browser. Test API responses with built-in JSON and XML validators. Load test your API with hundreds of simulated concurrent connections. Generate code snippets for API automation testing frameworks. Share and discuss your API requests online.

File ▾ ↗ Generate Code ▾ Tools ▾ Share ↗ ↗ Generate Code ▾ App Mode

localhost:3000/curso GET EXT **Send** Status: 200 (OK) Time: 62 ms Size: 0.05 kb

Authorization Content Headers Raw (2)

☒ Bearer Token ☐ Basic Auth ☐ Custom ☐ No Auth

Token

The authorization header will be automatically generated when you send the request. Read more about HTTP Authentication.

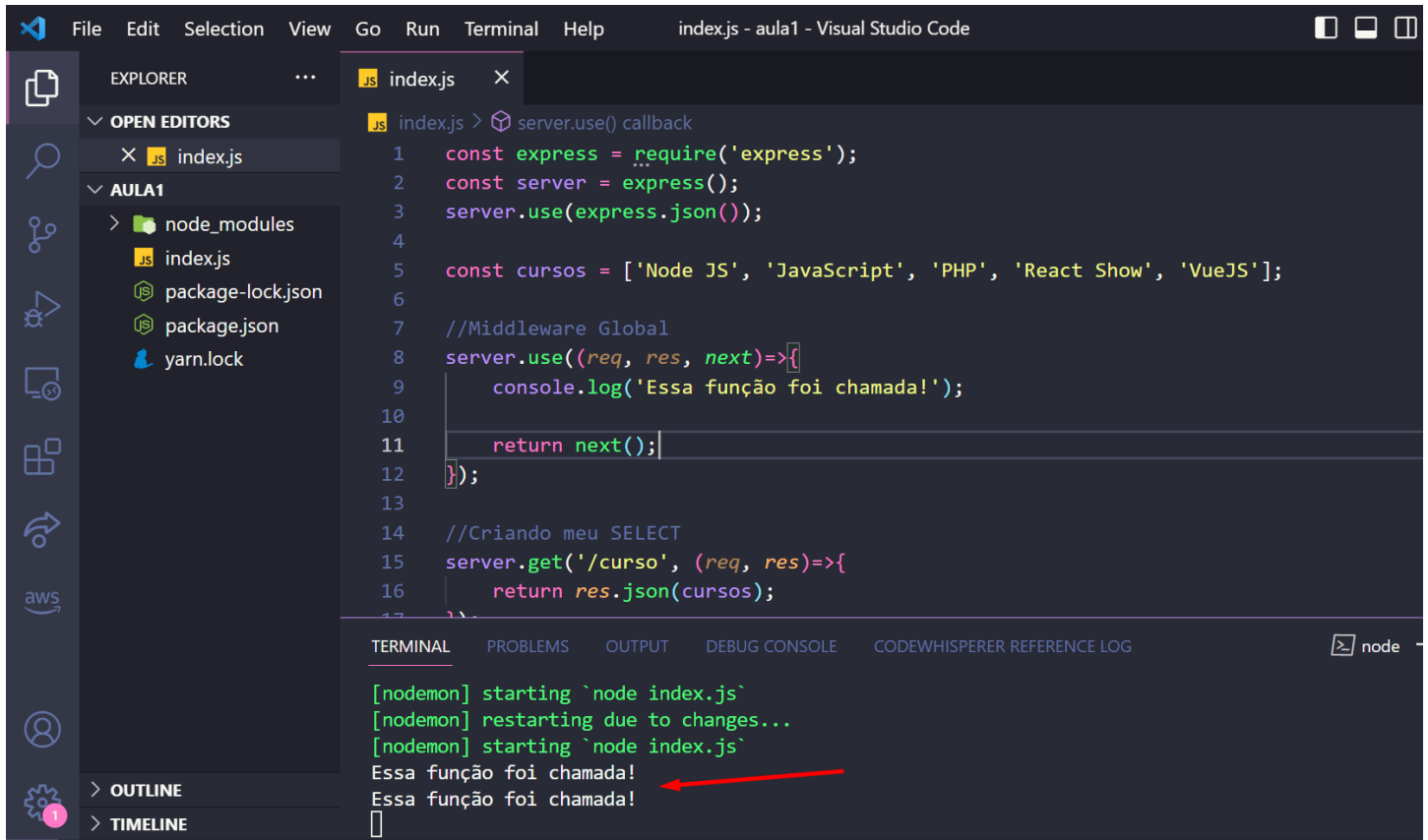
Content (1) Headers (6) Raw (8) JSON

```
[\"Node JS\", \"JavaScript\", \"PHP\", \"React Show\", \"VueJS\"]
```

**What is API?**

API (Application Programming Interface) is a computing interface that defines how software components interact with each other. It is a way of programmatically interacting with a separate software component or resource and expose functionality for internal or external use and testing. API defines what requests can be made, how they will be made and hides complexity from developers. API extends systems to enhance, organize code, and make components reusable.

06 – E também podemos ver que o fluxo de dados passa pelo nosso middleware.



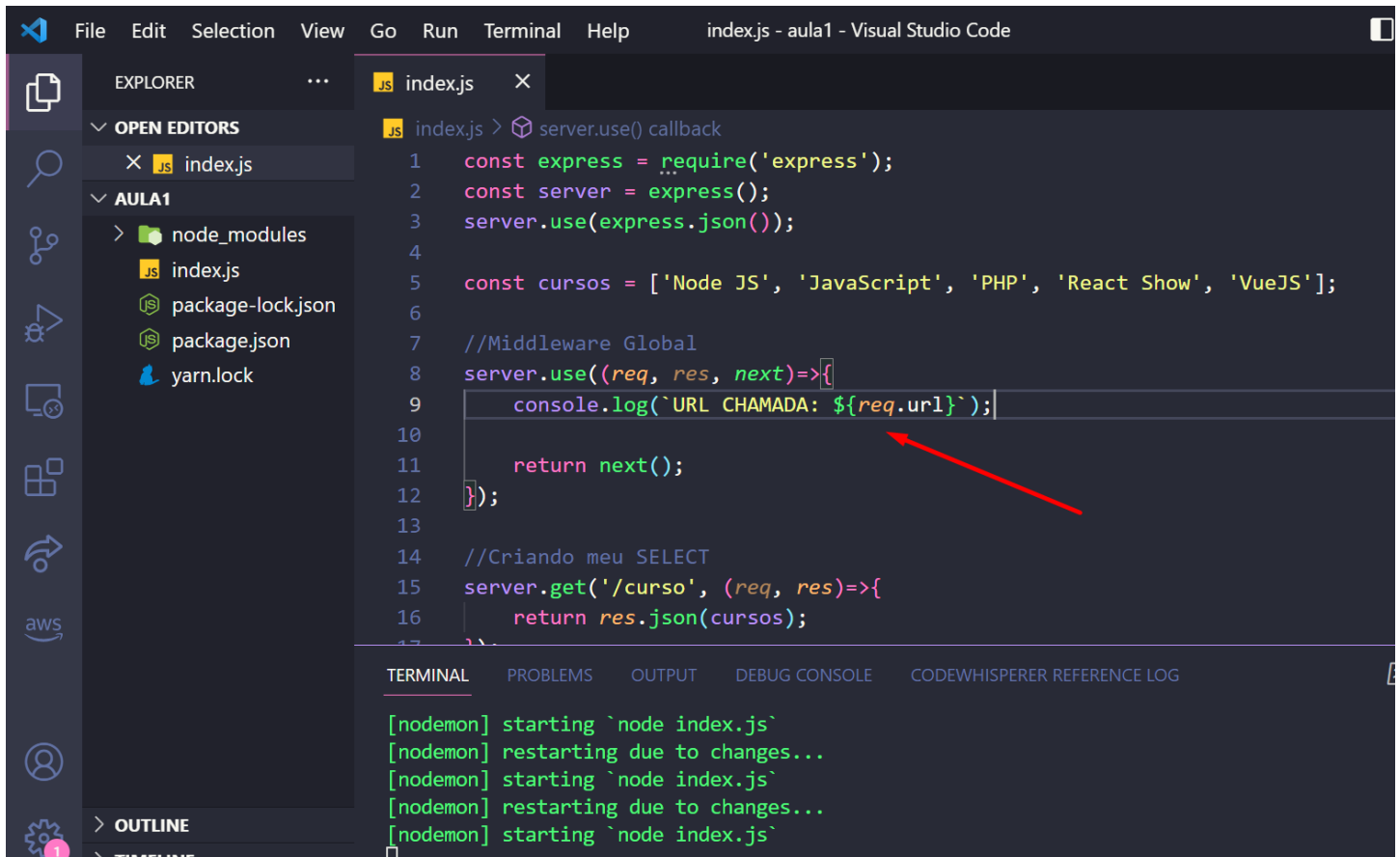
The screenshot displays the Visual Studio Code interface with a file explorer on the left, a code editor in the center, and a terminal at the bottom. The file explorer shows a project named 'AULA1' containing a 'node\_modules' directory and files like 'index.js', 'package-lock.json', 'package.json', and 'yarn.lock'. The code editor shows the contents of 'index.js', which includes a middleware function and a GET route. The terminal shows the output of running the application, including the message 'Essa função foi chamada!' which is highlighted by a red arrow.

```
index.js > server.use() callback
1  const express = require('express');
2  const server = express();
3  server.use(express.json());
4
5  const cursos = ['Node JS', 'JavaScript', 'PHP', 'React Show', 'VueJS'];
6
7  //Middleware Global
8  server.use((req, res, next)=>{
9      console.log('Essa função foi chamada!');
10
11      return next();
12  });
13
14  //Criando meu SELECT
15  server.get('/curso', (req, res)=>{
16      return res.json(cursos);
17  });
```

TERMINAL

```
[nodemon] starting `node index.js`
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
Essa função foi chamada!
Essa função foi chamada!
```

07 – Mas para que podemos usar esse Middleware? Como exemplo vamos monitorar em tempo real quais são as rotas que nossos cliente estão requisitando. No console mude as aspas para Template String (caso não lembre clique aqui <https://www.devmedia.com.br/javascript-template-literals/41193> ) e vamos agora adicionar a req.url ao nosso log.



```
File Edit Selection View Go Run Terminal Help index.js - aula1 - Visual Studio Code

EXPLORER
  OPEN EDITORS
    index.js
  AULA1
    node_modules
    index.js
    package-lock.json
    package.json
    yarn.lock

index.js > server.use() callback
1  const express = require('express');
2  const server = express();
3  server.use(express.json());
4
5  const cursos = ['Node JS', 'JavaScript', 'PHP', 'React Show', 'VueJS'];
6
7  //Middleware Global
8  server.use((req, res, next)=>{
9    console.log(`URL CHAMADA: ${req.url}`);
10
11    return next();
12  });
13
14  //Criando meu SELECT
15  server.get('/curso', (req, res)=>{
16    return res.json(cursos);
17  });

TERMINAL
[nodemon] starting `node index.js`
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
```

## 08 – Faça novamente uma chamada de API, /curso/1 por exemplo.

← → ↺

reqbin.com

🔑 🔍 📄 ☆ 🛠️ ⚙️ 🗑️ 👤 ⋮

REQBIN

EXAMPLES SAVED

POST Request Example  
Bearer Token Auth Example  
HTML Form POST Example  
GET Request Example  
REST API POST Example  
PUT Request Example  
Auth Bearer Header Example  
POST API Example  
POST JSON Example  
JSON Payload Example  
Content-Length Example  
REST API GET Example  
Send Cookies Example  
POST XML Example  
JSON Response Example  
POST JSON with Bearer Token  
  
Curl POST Request Example  
Curl GET Request Example  
Curl POST JSON Example  
Curl POST Body Example  
Curl Basic Auth Example  
Curl Bearer Token Example  
Curl Download File Example  
Curl Ignore SSL Checks  
  
JavaScript POST Example  
JavaScript GET Example  
JavaScript Fetch JSON  
JavaScript Array Join Example  
JavaScript String Contains  
JavaScript Sum Array  
JavaScript Substring Example

Curl Python JavaScript Node.js PHP Java JSON XML

Contact Account

Online REST & SOAP API Testing Tool

ReqBin is an online API testing tool for REST and SOAP APIs. Test API endpoints by making API requests directly from your browser. Test API responses with built-in JSON and XML validators. Load test your API with hundreds of simulated concurrent connections. Generate code snippets for API automation testing frameworks. Share and discuss your API requests online.

📄 File ↕ ↗ Generate Code ⚙ Tools

🔗 Share ↗ ↗ Generate Code ⚙ App Mode

localhost:3000/curso/1

GET EXT

Send

Status: 200 (OK) Time: 31 ms Size: 0.01 kb

Authorization Content Headers Raw (2)

Content (1) Headers (8) Raw (10)

☒ Bearer Token ☐ Basic Auth ☐ Custom ☐ No Auth

Token

The authorization header will be automatically generated when you send the request. Read more about HTTP Authentication.

"JavaScript"

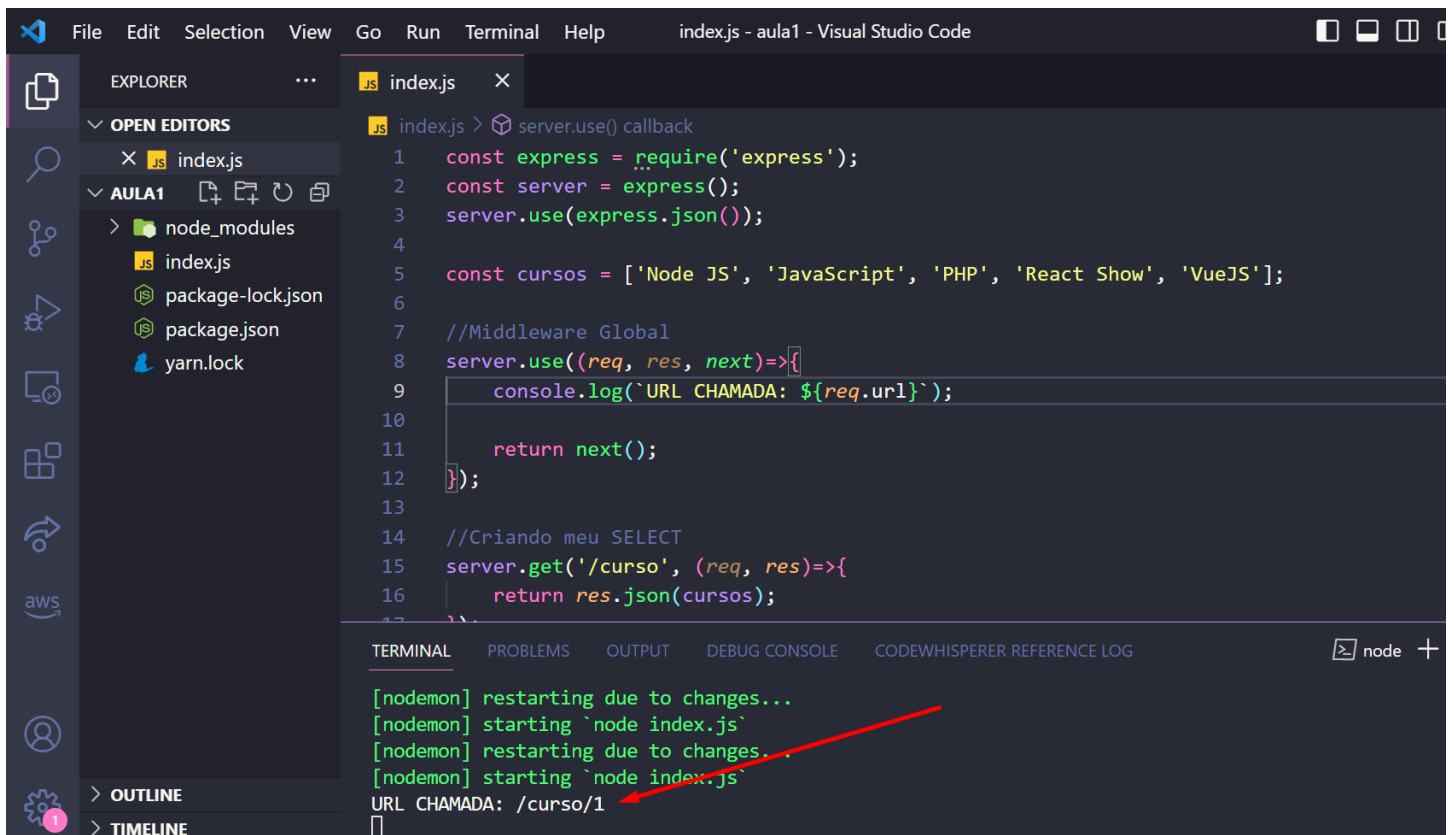
What is API?

API (Application Programming Interface) is a computing interface that defines how software components interact with each other. It is a way of programmatically interacting with a separate software component or resource and expose functionality for internal or external use and testing. API defines what requests can be made, how they will be made and hides complexity from

ADVERTISEMENT

ezoic report this ad

09 - Veja que agora nosso Middleware mantém a mensagem de todas as rotas chamadas.

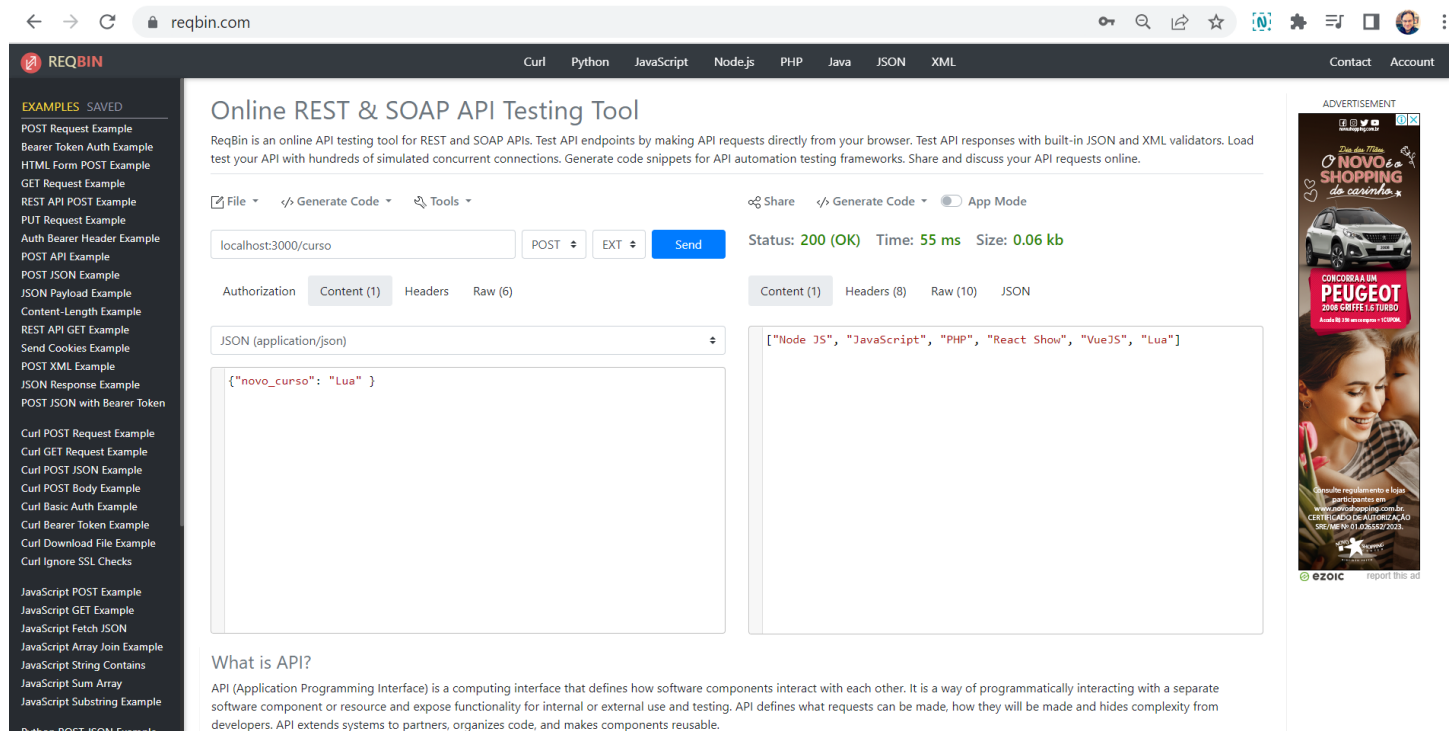


```
index.js > server.use() callback
1  const express = require('express');
2  const server = express();
3  server.use(express.json());
4
5  const cursos = ['Node JS', 'JavaScript', 'PHP', 'React Show', 'VueJS'];
6
7  //Middleware Global
8  server.use((req, res, next)=>{
9    console.log(`URL CHAMADA: ${req.url}`);
10
11    return next();
12  });
13
14  //Criando meu SELECT
15  server.get('/curso', (req, res)=>{
16    return res.json(cursos);
17  });
```

TERMINAL

```
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
URL CHAMADA: /curso/1
```

10 – Vamos fazer algo mais útil com Middleware. Veja o caso do nosso POST, para inserir um novo curso nós configuramos nossa API para esperar sempre no Body do POST um JSON, correto. Desse modo:



The screenshot shows the ReqBin website, an online REST & SOAP API testing tool. The interface includes a sidebar with various example requests, a main area for testing, and a right sidebar with an advertisement.

**ReqBin** Online REST & SOAP API Testing Tool

ReqBin is an online API testing tool for REST and SOAP APIs. Test API endpoints by making API requests directly from your browser. Test API responses with built-in JSON and XML validators. Load test your API with hundreds of simulated concurrent connections. Generate code snippets for API automation testing frameworks. Share and discuss your API requests online.

**Test Details:**

- URL: localhost:3000/curso
- Method: POST
- Status: 200 (OK)
- Time: 55 ms
- Size: 0.06 kb

**Request Headers:**

- Content-Type: application/json

**Request Body:**

```
{\"novo_curso\": \"Lua\" }
```

**Response Headers:**

- Content-Type: application/json

**Response Body:**

```
[\"Node JS\", \"JavaScript\", \"PHP\", \"React Show\", \"VueJS\", \"Lua\"]
```

**What is API?**

API (Application Programming Interface) is a computing interface that defines how software components interact with each other. It is a way of programmatically interacting with a separate software component or resource and expose functionality for internal or external use and testing. API defines what requests can be made, how they will be made and hides complexity from developers. API extends systems to partners, organizes code, and makes components reusable.

**Advertisement:**

CONCORRÊNCIA LIMPA  
NOVO SHOPPING  
de carinha

CONCORRÊNCIA LIMPA  
PEUGEOT  
2008 GRiffe 1.6 TURBO

Participantes em  
www.novoshopping.com.br  
CNPJ nº 01.073.552/2023

ezeio report this ad

11 – Mas e se o cliente não passar nada no Body, qual será o comportamento da nossa API? Veja que nesse caso ele faz o cadastro sim, porém, com um JSON vazio. Isso é péssimo pois não deveríamos aceitar um Body vazio. Nesse caso, vamos usar um Middleware local para criar essa regra.

Online REST & SOAP API Testing Tool

ReqBin is an online API testing tool for REST and SOAP APIs. Test API endpoints by making API requests directly from your browser. Test API responses with built-in JSON and XML validators. Load test your API with hundreds of simulated concurrent connections. Generate code snippets for API automation testing frameworks. Share and discuss your API requests online.

File - Generate Code - Tools - Share - Generate Code - App Mode

localhost:3000/curso POST EXT Send Status: 200 (OK) Time: 7 ms Size: 0.07 kb

Authorization Content Headers Raw (4) Content (1) Headers (6) Raw (8) JSON

JSON (application/json)

["key": "value"]

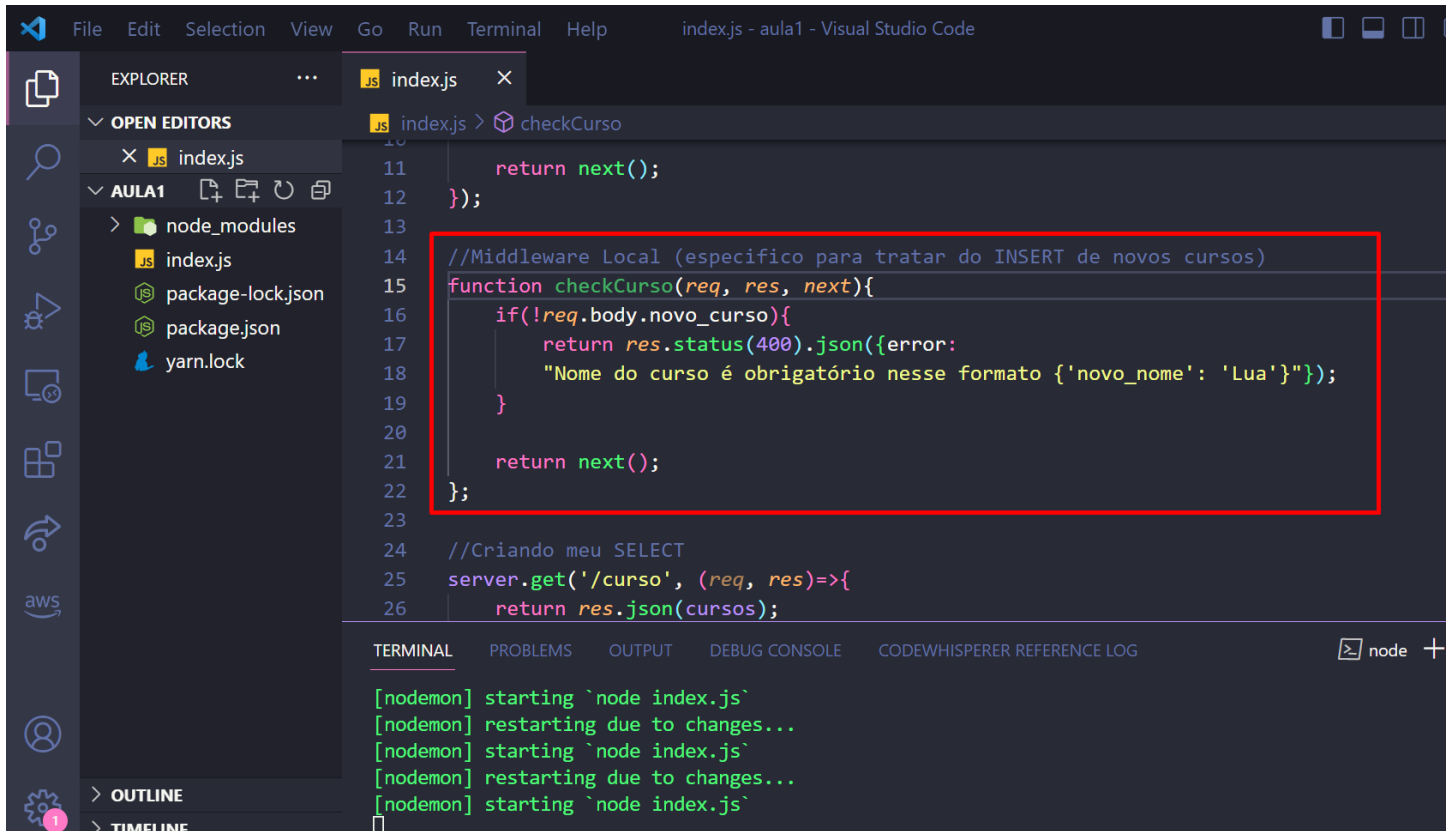
["Node JS", "JavaScript", "PHP", "React Show", "VueJS", "Lua", null, null]

What is API?

API (Application Programming Interface) is a computing interface that defines how software components interact with each other. It is a way of programmatically interacting with a separate software component or resource and expose functionality for internal or external use and testing. API defines what requests can be made, how they will be made and hides complexity from developers. API extends systems to partners, organizes code, and makes components reusable.

Aguardando nvm1-ib.adnxs.com...

12 – Crie um Middleware chamado checkCurso, vamos fazer um if que verifica se o cliente não mandou um body no POST. E se caso ele deixar o Body vazio, esse IF irá retornar um código de erro e ainda uma mensagem amigável para ajudar o cliente a entender como usar nossa API.



```
File Edit Selection View Go Run Terminal Help index.js - aula1 - Visual Studio Code

EXPLORER
  OPEN EDITORS
    index.js
  AULA1
    node_modules
    index.js
    package-lock.json
    package.json
    yarn.lock

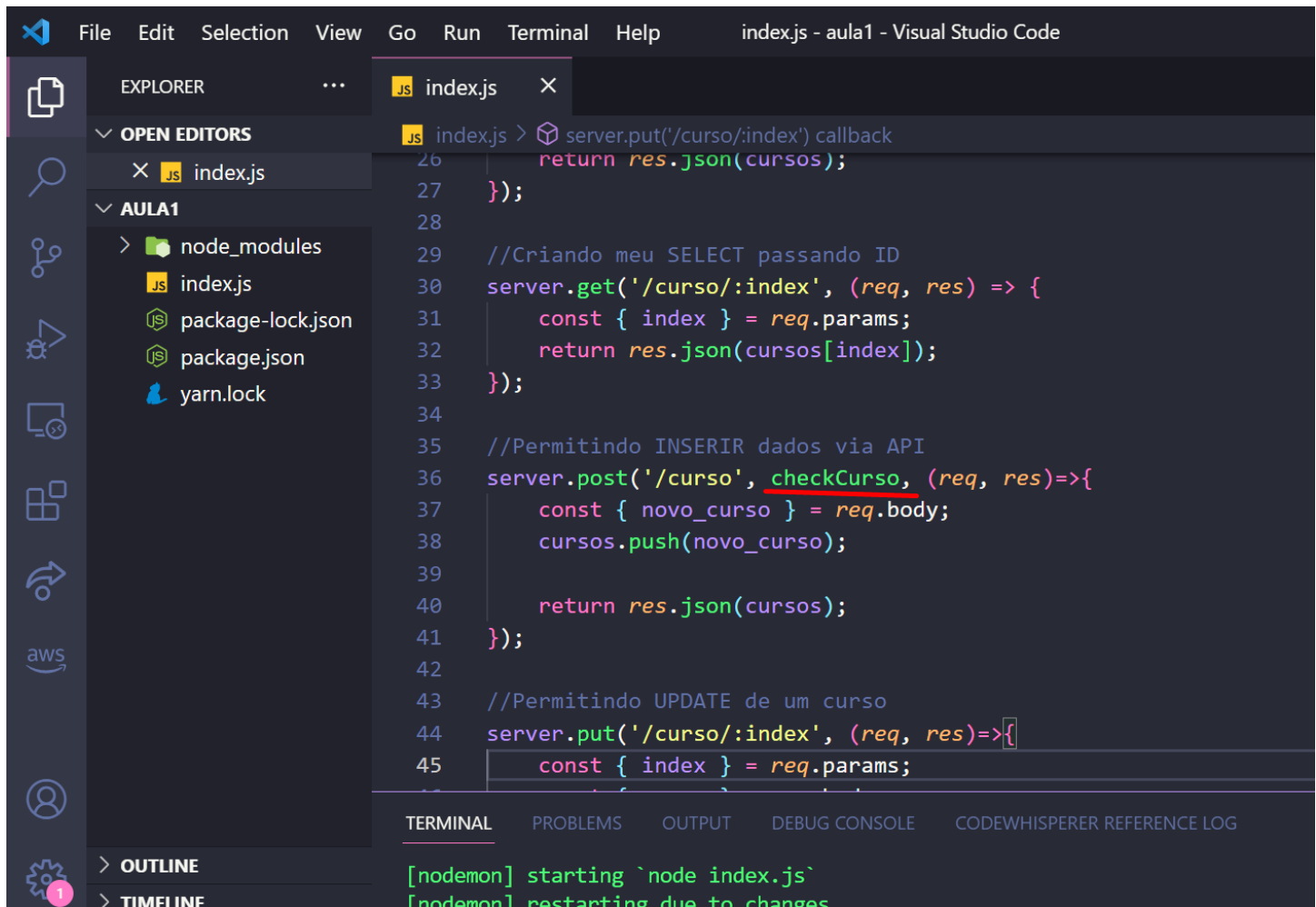
index.js
10
11   return next();
12 });
13
14 //Middleware Local (especifico para tratar do INSERT de novos cursos)
15 function checkCurso(req, res, next){
16   if(!req.body.novo_curso){
17     return res.status(400).json({error:
18       "Nome do curso é obrigatório nesse formato {'novo_nome': 'Lua'}"});
19   }
20
21   return next();
22 };
23
24 //Criando meu SELECT
25 server.get('/curso', (req, res)=>{
26   return res.json(cursos);
27 });

TERMINAL
[nodemon] starting `node index.js`
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`

> OUTLINE
> TIMELINE
```



13 – Agora como vamos configurar o servidor back-end de modo que toda vez que existir a necessidade de passar o Body {"novo\_curso": "Curso"}, dentro do POST, ele faça essa verificação. Adicione o checkCurso ao método POST após a rota.



```
File Edit Selection View Go Run Terminal Help index.js - aula1 - Visual Studio Code

EXPLORER
  OPEN EDITORS
    index.js
  AULA1
    node_modules
    index.js
    package-lock.json
    package.json
    yarn.lock

index.js
26 return res.json(cursos);
27 });
28
29 //Criando meu SELECT passando ID
30 server.get('/curso/:index', (req, res) => {
31   const { index } = req.params;
32   return res.json(cursos[index]);
33 });
34
35 //Permitindo INSERIR dados via API
36 server.post('/curso', checkCurso, (req, res)=>{
37   const { novo_curso } = req.body;
38   cursos.push(novo_curso);
39
40   return res.json(cursos);
41 });
42
43 //Permitindo UPDATE de um curso
44 server.put('/curso/:index', (req, res)=>{
45   const { index } = req.params;
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE CODEWHISPERER REFERENCE LOG

```
[nodemon] starting `node index.js`
[nodemon] restarting due to changes...
```

14 – Agora faça novamente uma requisição POST sem passar o Body, agora sim, nosso Middleware entra em ação evitando o cadastro de um curso vazio. Fizemos isso em um Middleware separado pois podemos criar regras que servem para diversas requisições diferentes e utilizar o mesmo Middleware em todas elas.

The screenshot displays the ReqBin web application interface for testing REST and SOAP APIs. The browser's address bar shows the URL `reqbin.com`. The interface includes a sidebar with a list of example requests, a main workspace for creating and testing requests, and a right-hand panel for the response details.

In the main workspace, the URL `localhost:3000/curso` is entered. The request method is set to `POST`, and the content type is `JSON (application/json)`. The request body is `{"key": "value"}`. The `Send` button is highlighted.

The response panel on the right shows the following details:

- Status: **400 (Bad Request)**
- Time: 102 ms
- Size: 0.07 kb

The response body is displayed in JSON format:

```
{  "error": "Nome do curso é obrigatório nesse formato ('novo_nome': 'Lua')"}  
```

Below the response, a section titled "What is API?" provides a brief definition of an API (Application Programming Interface).

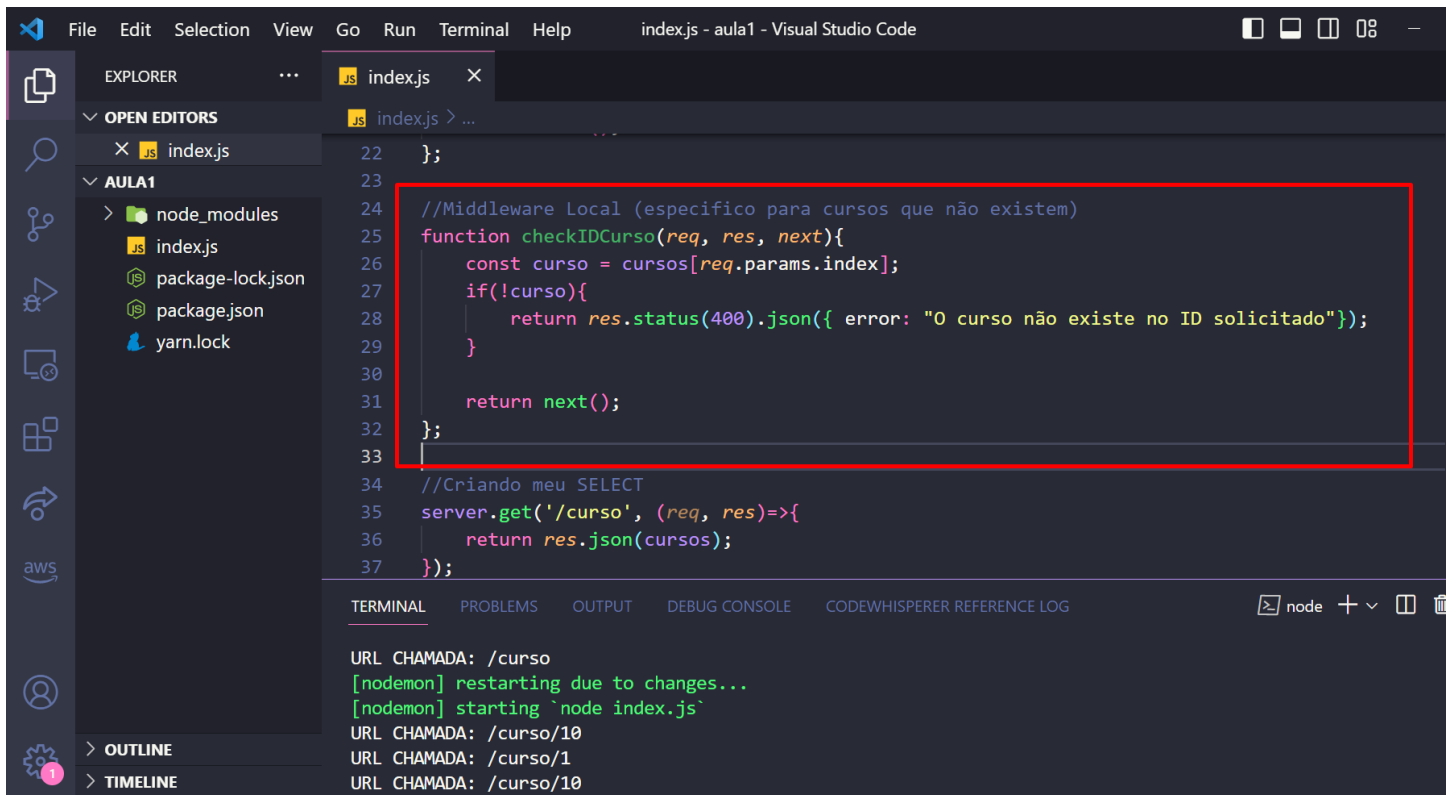
15 – Vamos aproveitar o Middleware para tratar outro erro. Quando um cliente quer buscar um curso pelo ID, caso ele passe um ID que não existe no banco de dados como por exemplo o ID:10, nossa API não informa que não existe aquele curso, isso não é profissional. Vamos tratar isso.

The screenshot shows the ReqBin website, an online REST & SOAP API Testing Tool. The interface is divided into three main sections:

- Left Sidebar (Examples):** A list of example requests categorized by method (POST, GET, PUT) and authentication type (Bearer Token, Basic Auth, Custom, No Auth). Examples include "POST Request Example", "Bearer Token Auth Example", "HTML Form POST Example", "GET Request Example", "REST API POST Example", "PUT Request Example", "Auth Bearer Header Example", "POST JSON Example", "JSON Payload Example", "Content-Length Example", "REST API GET Example", "Send Cookies Example", "POST XML Example", "JSON Response Example", "POST JSON with Bearer Token", "Curl POST Request Example", "Curl GET Request Example", "Curl POST JSON Example", "Curl POST Body Example", "Curl Basic Auth Example", "Curl Bearer Token Example", "Curl Download File Example", "Curl Ignore SSL Checks", "JavaScript POST Example", "JavaScript GET Example", "JavaScript Fetch JSON", "JavaScript Array Join Example", "JavaScript String Contains", "JavaScript Sum Array", and "JavaScript Substring Example".
- Main Testing Area:** The top section is titled "Online REST & SOAP API Testing Tool". Below the title, there's a description of ReqBin. The main interface has a red box highlighting the input field (containing "localhost:3000/curso/10"), the method dropdown (set to "GET"), and the "Send" button. Below the input field, there are tabs for "Authorization", "Content", "Headers", and "Raw (2)". The "Authorization" tab is active, showing "Bearer Token" selected. Below this, there's a "Token" input field and a note: "The authorization header will be automatically generated when you send the request. Read more about HTTP Authentication." To the right of the input field, the status is displayed as "Status: 200 (OK) Time: 8 ms Size: 0.00 kb". Below the status, there are tabs for "Content", "Headers (7)", and "Raw (7)".
- Right Sidebar (Advertisement):** An advertisement for "CHEGOU A NOVA EISENBahn UNFILTERED." featuring a beer bottle and the text "COMPAR EISENBahn".

At the bottom of the main testing area, there's a section titled "What is API?" with a brief definition: "API (Application Programming Interface) is a computing interface that defines how software components interact with each other. It is a way of programmatically interacting with a separate software component or resource and expose functionality for internal or external use and testing. API defines what requests can be made, how they will be made and hides complexity from..."

16 – Vamos criar um Middleware chamado checkIDCurso que no IF verifica primeiro se não existe o curso solicitado para então retornar a mensagem de erro ao cliente.



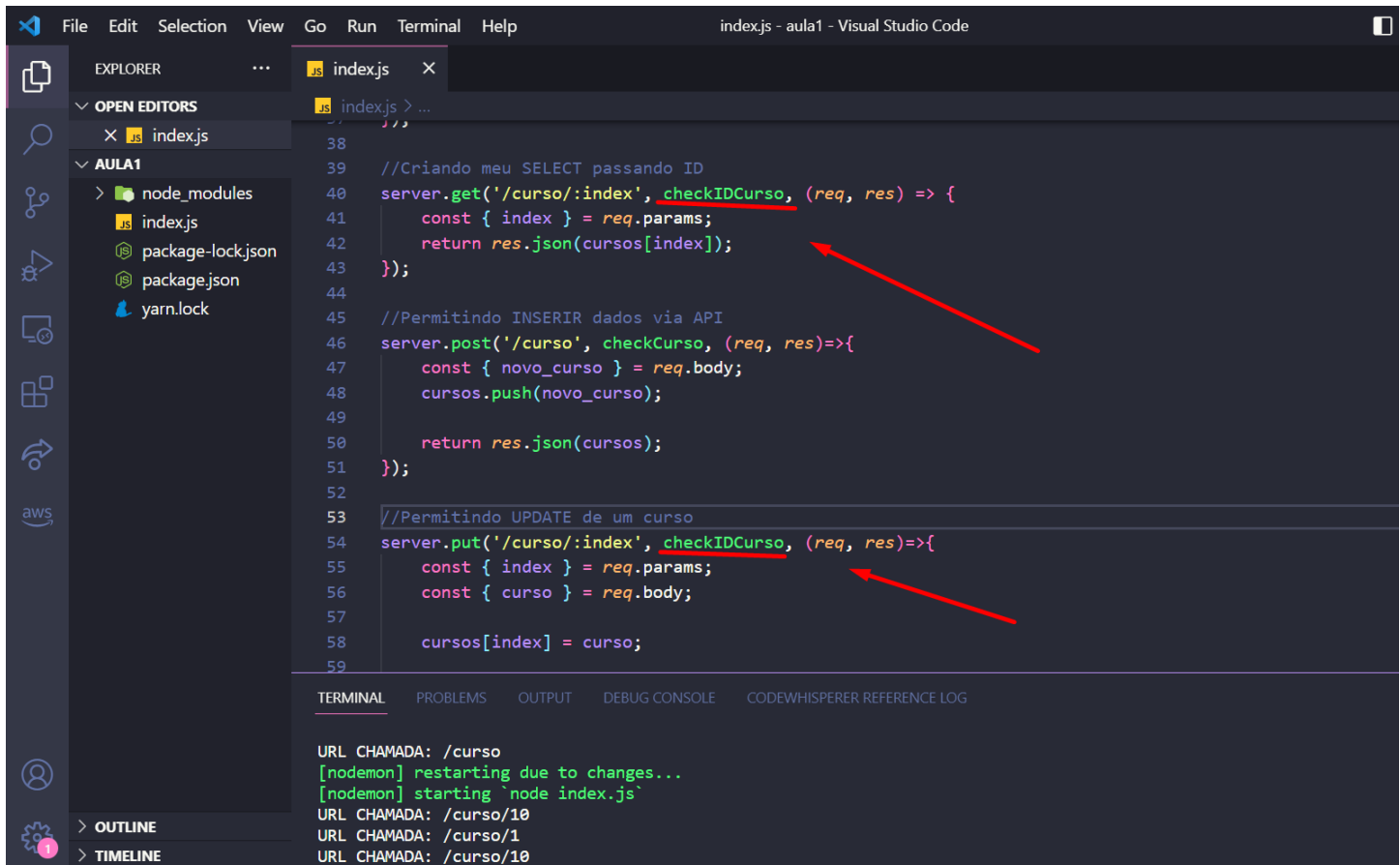
```
File Edit Selection View Go Run Terminal Help index.js - aula1 - Visual Studio Code

EXPLORER
  OPEN EDITORS
    index.js
  AULA1
    node_modules
    index.js
    package-lock.json
    package.json
    yarn.lock

22 };
23
24 //Middleware Local (especifico para cursos que não existem)
25 function checkIDCurso(req, res, next){
26   const curso = cursos[req.params.index];
27   if(!curso){
28     return res.status(400).json({ error: "O curso não existe no ID solicitado"});
29   }
30
31   return next();
32 };
33
34 //Criando meu SELECT
35 server.get('/curso', (req, res)=>{
36   return res.json(cursos);
37 });

TERMINAL
  node
  URL CHAMADA: /curso
  [nodemon] restarting due to changes...
  [nodemon] starting `node index.js`
  URL CHAMADA: /curso/10
  URL CHAMADA: /curso/1
  URL CHAMADA: /curso/10
```

17 – Veja que tanto no GET com a rota passando index (id) quanto no UPDATE, ambos precisam que seja passado um ID na rota para lidar com requisição. Portanto, nosso Middleware pode ser compartilhado tanto no GET quanto no UPDATE, você pode adicionar quantos Middlewares diferentes quiser nas requisições.



```
38
39 //Criando meu SELECT passando ID
40 server.get('/curso/:index', checkIDCurso, (req, res) => {
41   const { index } = req.params;
42   return res.json(cursos[index]);
43 });
44
45 //Permitindo INSERIR dados via API
46 server.post('/curso', checkCurso, (req, res)=>{
47   const { novo_curso } = req.body;
48   cursos.push(novo_curso);
49
50   return res.json(cursos);
51 });
52
53 //Permitindo UPDATE de um curso
54 server.put('/curso/:index', checkIDCurso, (req, res)=>{
55   const { index } = req.params;
56   const { curso } = req.body;
57
58   cursos[index] = curso;
59 }
```

TERMINAL

```
URL CHAMADA: /curso
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
URL CHAMADA: /curso/10
URL CHAMADA: /curso/1
URL CHAMADA: /curso/10
```

18 – Veja que agora seja no GET ou UPDATE caso não exista o ID solicitado dentro do banco de dados, nosso cliente é avisado do erro.

The screenshot shows the ReqBin website interface. The browser address bar displays 'reqbin.com'. The website has a dark header with navigation links for various programming languages: Curl, Python, JavaScript, Node.js, PHP, Java, JSON, and XML. On the left, there is a sidebar with 'EXAMPLES' and 'SAVED' sections, listing various API request examples like POST, GET, PUT, and DELETE. The main content area is titled 'Online REST & SOAP API Testing Tool'. It features a request input field with the URL 'localhost:3000/curso/10', a 'Send' button, and tabs for 'Authorization', 'Content', 'Headers', and 'Raw (2)'. Below the input field, there are radio buttons for 'Bearer Token', 'Basic Auth', 'Custom', and 'No Auth'. The response area shows a 'Status: 400 (Bad Request)' with a time of '24 ms' and a size of '0.05 kb'. The response body is displayed in JSON format: 

```
{  "error": "O curso não existe no ID solicitado"}
```

. At the bottom, there is a section titled 'What is API?' with a brief explanation of API (Application Programming Interface).

## **Documentando API**

19 – Agora que temos uma API funcional e online, podemos documentar seu uso. Documentar API pode ser feita de várias formas. Vamos começar com a mais básica que é criando um PDF. Crie um novo documento em branco e siga o modelo abaixo usando de base o deploy da API feita no Codebox (Aula passada).

# Exemplo de Documentação API – Cursos

**URL Base:** localhost:3000

## Lista de Cursos

**GET** /curso

**Response body**

```
[  
  
"Node JS", "JavaScript", "PHP", "React Show", "VueJS"  
  
]
```

## Lista de Cursos por ID

**GET** /curso/id

**Exemplo:** localhost:3000/curso/1

**Response body**

```
{  
  
"Node JS"  
  
}
```



## Cadastrando um novo Curso

**POST** /curso

**Exemplo:** localhost:3000/curso

**Request body**

```
{  
  "novo_curso": "lua"  
}
```

**Response body**

```
[  
  "Node JS", "JavaScript", "PHP", "React Show", "VueJS", "lua"  
]
```

## Atualizando um curso existente

**PUT** /curso/id

**Exemplo:** localhost:3000/curso/0

**Request body**

```
{  
  "curso": "Node JS Avançado"  
}
```

**Response body**

```
[  
  "Node JS Avançado", "JavaScript", "PHP", "React Show", "VueJS", "lua"  
]
```

## Deletando um curso existente

**DELETE** /curso/id

**Exemplo:** localhost:3000/curso/5

**Response body**

```
{  
  "message": "Curso deletado com sucesso"  
}
```

## Exercício

- 01 – Crie um Middleware que toda vez que um PUT seja requisitado verifique se existe um Request Body, caso não existe, informe ao cliente um código de erro e uma orientação.
- 02 – No DELETE atual, a mensagem de erro fica dentro do próprio POST. Crie um Middleware que possa lidar com essa mensagem de erro.
- 03 – Crie um Middleware que toda vez que um curso seja inserido no banco de dados, mostre no console.log a lista de cursos atualizada.
- 04 – Crie um Middleware que toda vez que um curso for deletado do banco de dados, mostre no console.log a lista de cursos atualizada.