

Introdução ao Back-end

Node JS

O que é **Node JS**?

- **Back-end é tudo o que o usuário não vê ao acessar uma aplicação Web**
 - Banco de Dados
 - Regra de negócio
 - Recursos do servidor
 - Tecnologias utilizadas

O que é **Node JS**?

- O Node JS permite utilizar o JavaScript para que os serviços da aplicação estejam sempre funcionando!
- Node JS não é uma linguagem mas sim uma plataforma: <https://nodejs.org/en>

Histórico Node JS

- Ryan Dahl se inspirou para criar Node.js depois de ver barra de progresso de carga de arquivos no Flickr, ele percebeu que o navegador não sabia o quanto do arquivo foi carregado e tinha que consultar o servidor web.

Características Node JS

- A principal característica e diferença de outras tecnologias (como PHP, Java, C# e C) é a execução das requisições/eventos em single-thread, onde apenas uma thread (chamada de Event Loop)
- Event Loop é responsável por executar o código Javascript, sem a necessidade de criar nova thread que utilizaria mais recursos computacionais (por exemplo memória RAM) e sem o uso da fila de espera.

Características **Node JS**

- **Arquitetura Event-Loop.**
 - Call Stack (Pilha de Eventos)
- **Single-thread**
 - Utiliza diversas bibliotecas em C++ para gerenciamento de processamento
- **Non-Blocking I/O**
 - Mantem a conexão com o front sempre ativa (diferente do PHP)
 - Exemplo: Facebook não recarrega a página sempre que temos notificações.

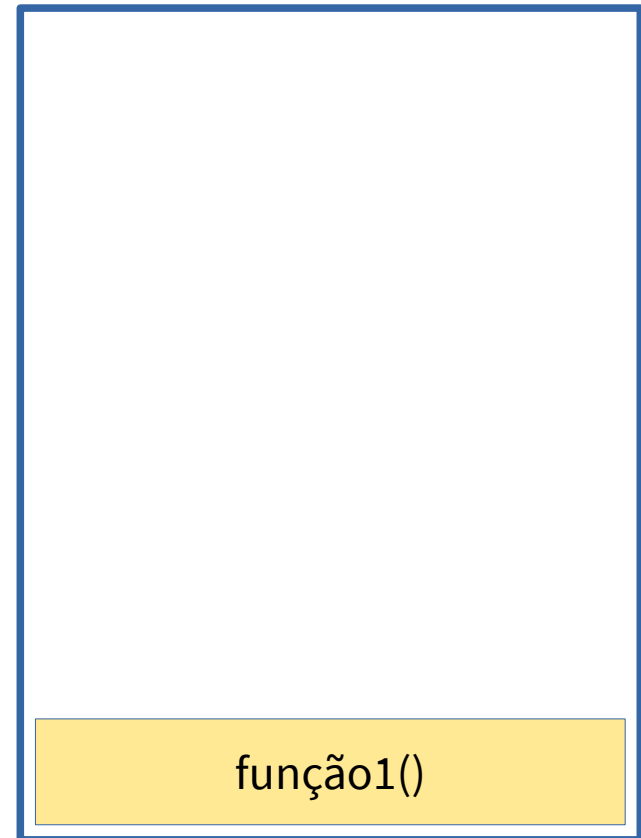
Características Node JS

- Arquitetura Event-Loop.
 - Call Stack (Pilha de Eventos)



Características Node JS

- Arquitetura Event-Loop.
 - Call Stack (Pilha de Eventos)



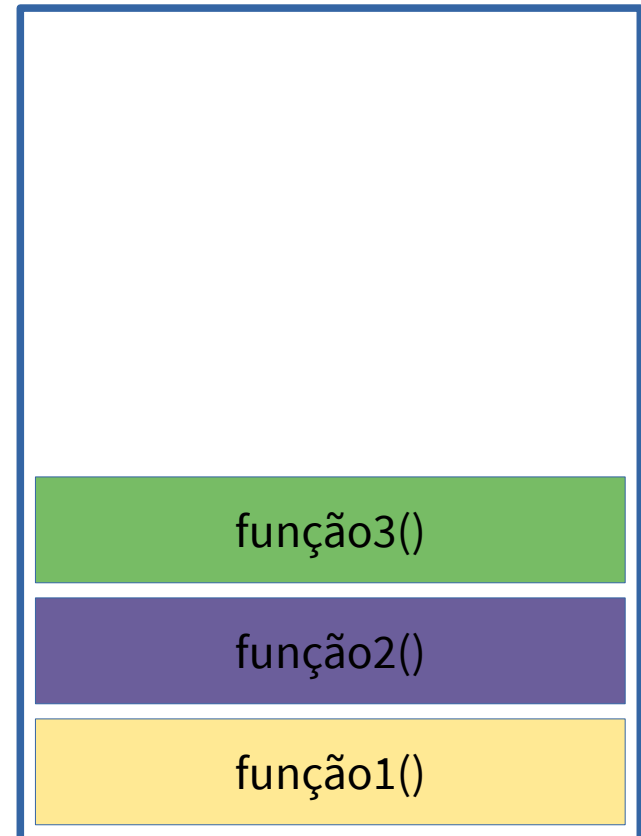
Características Node JS

- Arquitetura Event-Loop.
 - Call Stack (Pilha de Eventos)



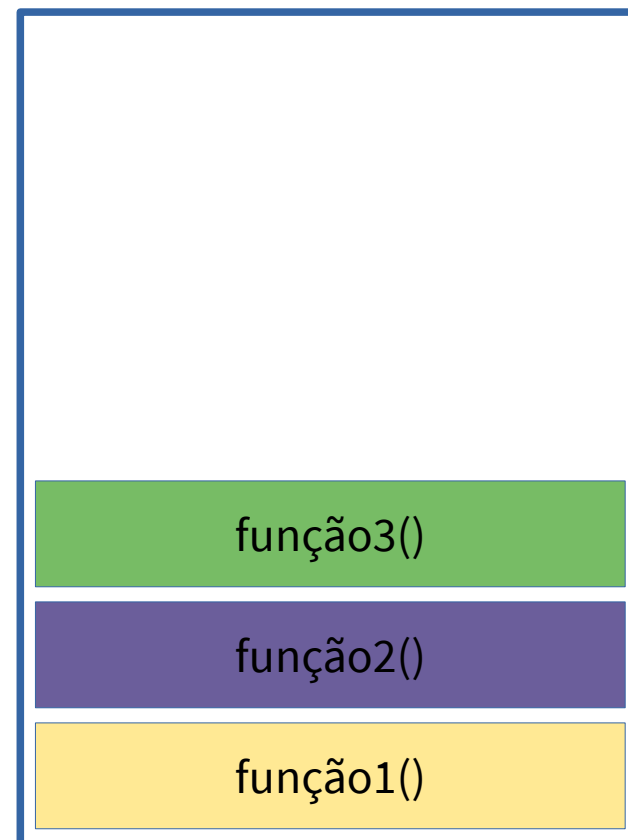
Características Node JS

- Arquitetura Event-Loop.
 - Call Stack (Pilha de Eventos)



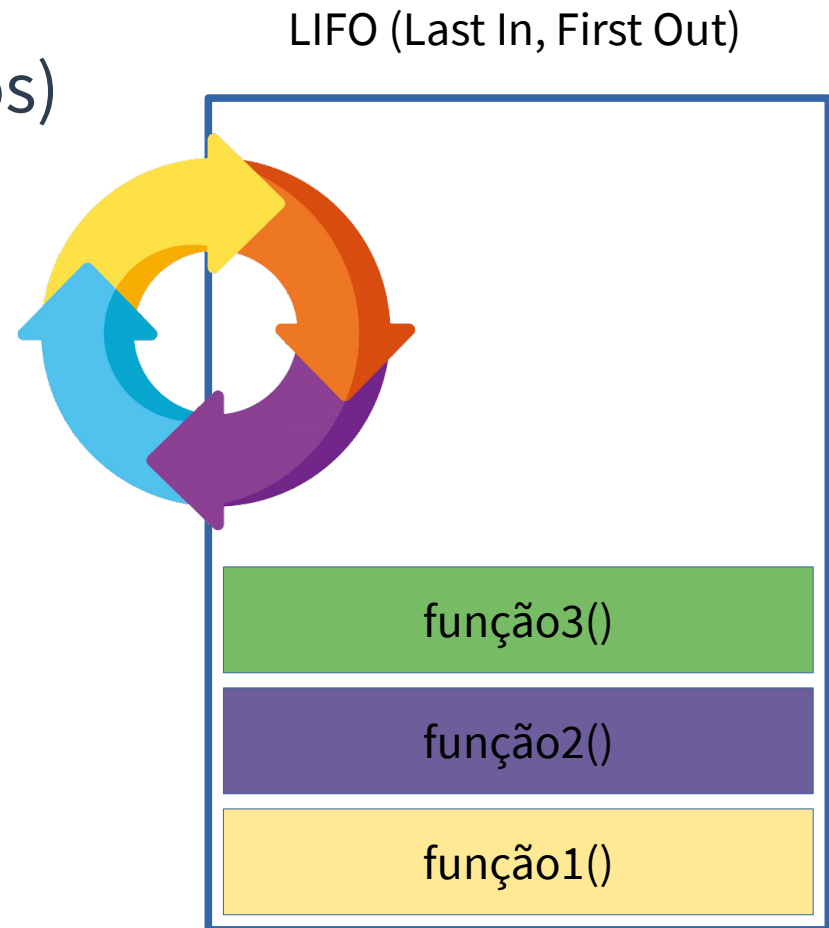
Características Node JS

- Arquitetura Event-Loop.
 - Call Stack (Pilha de Eventos)



Características Node JS

- Arquitetura Event-Loop.
 - Call Stack (Pilha de Eventos)



API RESTful



API REST

- API REST, também chamada de API RESTful, é uma interface de programação de aplicações (API ou API web) que está em conformidade com as restrições do estilo de arquitetura REST
- Permite a interação com serviços web RESTful. REST é a sigla em inglês para "Representational State Transfer", que em português significa transferência de estado representacional.
- Essa arquitetura foi criada pelo cientista da computação Roy Fielding.

API REST

- Fluxo: Requisição e resposta (cliente-servidor)
- Frontend: Recebe os dados e processa.
- Resposta: Através de uma estrutura de dados.

ROTAS - API REST

- **GET** <http://minhaapi.com/usuarios>
- **POST** <http://minhaapi.com/usuarios>
- **PUT** <http://minhaapi.com/usuarios/1>
- **DELETE** <http://minhaapi.com/usuarios/1>

Métodos HTTP

Endpoint

Rota

Parâmetros

Vantagens **API REST**

- Vários clientes (front-end)
- Comunicação padronizada
- Multiplataforma: web, mobile e desktop

Estrutura de Dados

- JSON (JavaScript Object Notation)
- É um formato compacto, de padrão aberto independente, de troca de dados simples e rápida (parsing) entre sistemas, especificado por Douglas Crockford em 2000, que utiliza texto legível a humanos, no formato atributo-valor (natureza auto-descritiva)

JSON

```
{  
  "nome": "Pedro",  
  "altura": 1.90  
}
```

Chave (key)

Valor (value)

JSON - Requisições

- **GET** <http://minhaapi.com/usuarios>
 - Quero dados de todos os usuários
- **GET** <http://minhaapi.com/usuarios/1>
 - Quero apenas dados do usuário 1

JSON - Requisições

- **POST** <http://minhaapi.com/usuarios>

```
{  
  "token": 123456  
}
```

Header

```
{  
  "nome": "joao pedro",  
  "e-mail": "jpedro@gmail.com"  
}
```

Body (Request Body)

Métodos HTTP

- **GET:** o método GET solicita a representação de um recurso específico. Requisições utilizando o método GET devem retornar apenas dados.
- **POST:** o método POST é utilizado para submeter uma entidade a um recurso específico, frequentemente causando uma mudança no estado do recurso ou efeitos colaterais no servidor.
- **PUT:** o método PUT substitui todas as atuais representações do recurso de destino pela carga de dados da requisição.
- **DELETE:** o método DELETE remove um recurso específico.
- **PATCH:** o método PATCH é utilizado para aplicar modificações parciais em um recurso.
- **HEAD:** o método HEAD solicita uma resposta de forma idêntica ao método GET, porém sem conter o corpo da resposta.
- **OPTIONS:** o método OPTIONS é usado para descrever as opções de comunicação com o recurso de destino.
- **TRACE:** o método TRACE executa um teste de chamada loop-back junto com o caminho para o recurso de destino.

Códigos HTTP

- **1xx: Informations**
- **2xx: Success**
 - 200: Success
 - 201: Created
- **3xx: Redirection**
 - 301: Moved Permanently
 - 302: Moved
- **4xx: Client Error**
 - 400: Bad Request
 - 401: Unauthorized
 - 404: Not found
- **5xx: Server Error**
 - 500: Server Error

Bora testar API?

- Qual ferramenta usar?



Karate

Airborne

Pyresttest



HttpMaster

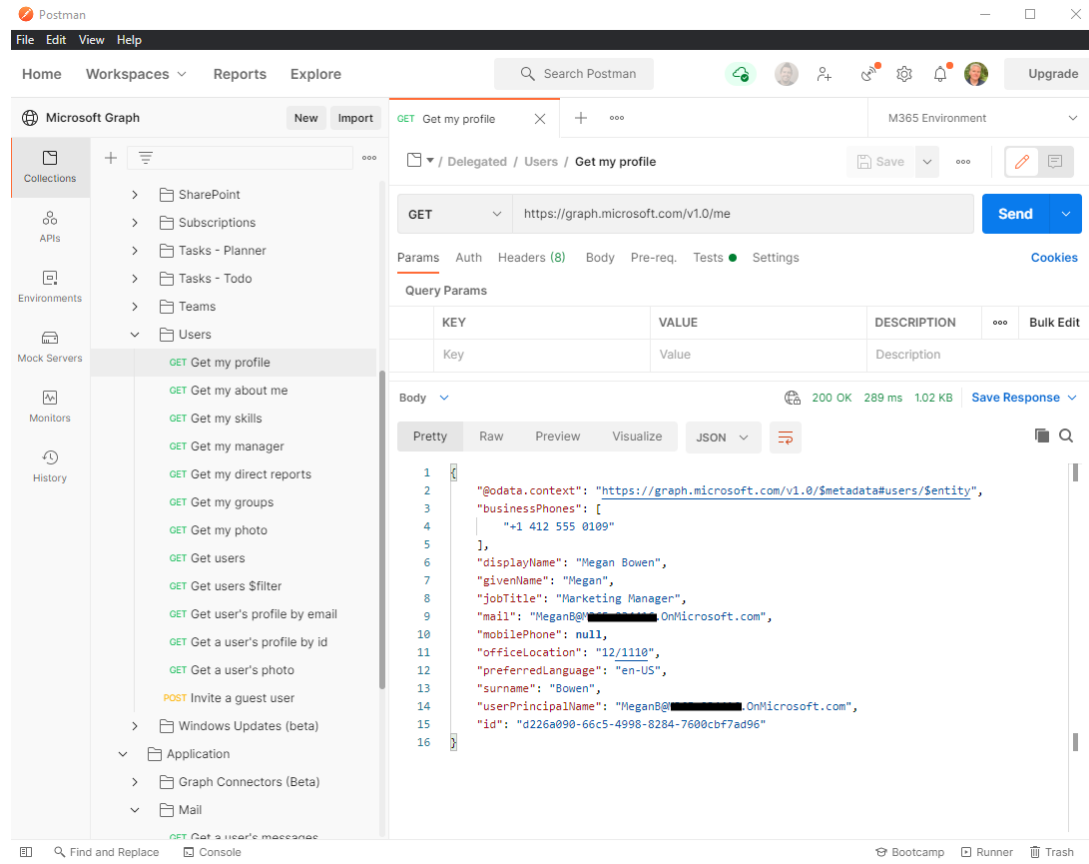


apiary



Bora testar API?

- <https://www.postman.com/>



Onde vamos conseguir os dados?

- <https://www.instantwebtools.net/>

[HOME](#)[FAKE API](#)[TIKTOK VIDEO DOWNLOADER](#)[JSON CONVERTERS](#)[UUID](#)[COUNTRY FLAGS](#)

Fake API for Testing

Are you looking for pre hosted, online and free API to test your API clients ? Here we've built and hosted 3 Fake API instances including Fake REST API, Secured REST API and Fake GraphQL API for your use.



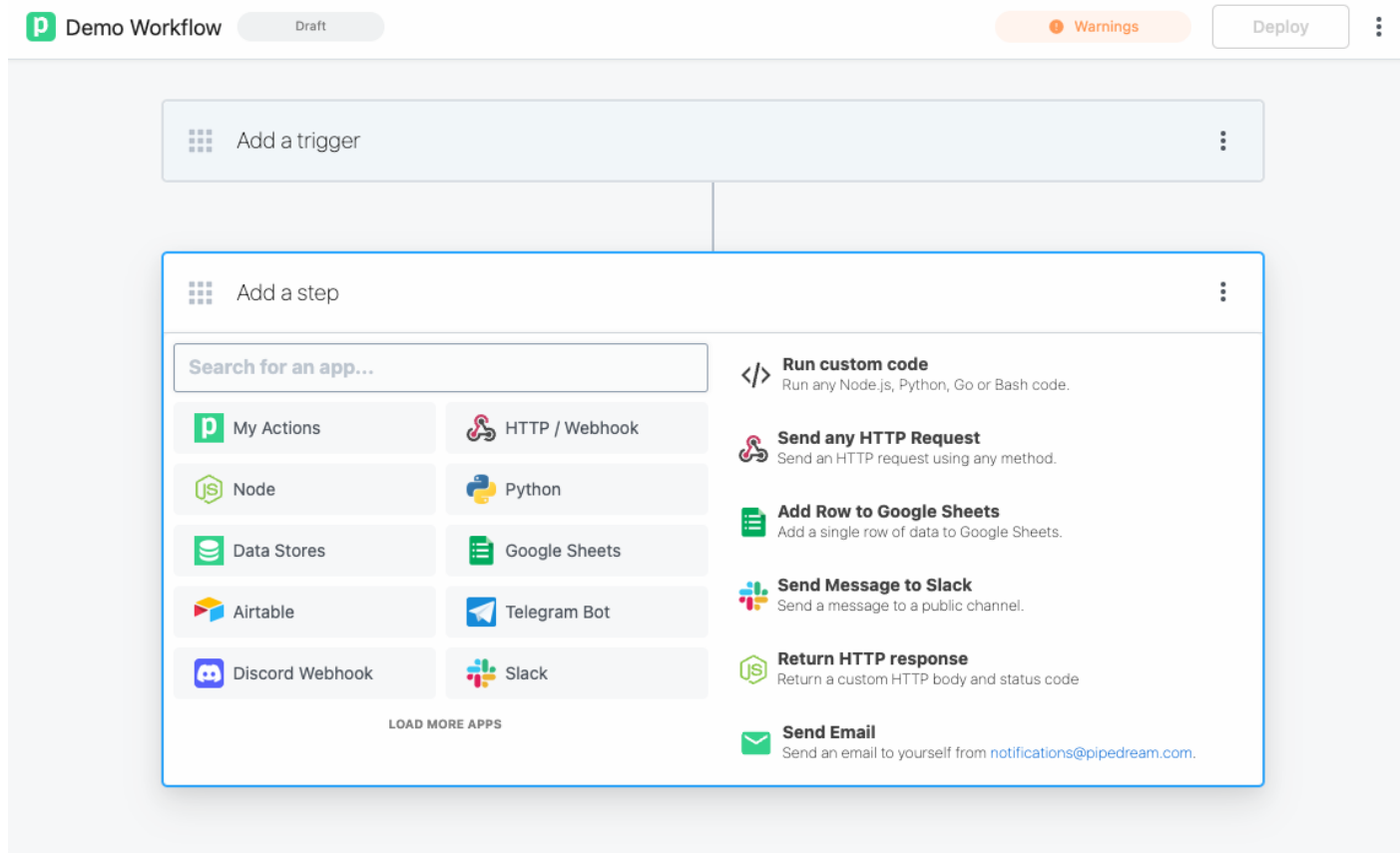
Fake Rest API



Fake Secured Rest API

Criando um Endpoint

- <https://pipedream.com/>



Iniciando - NodeJS

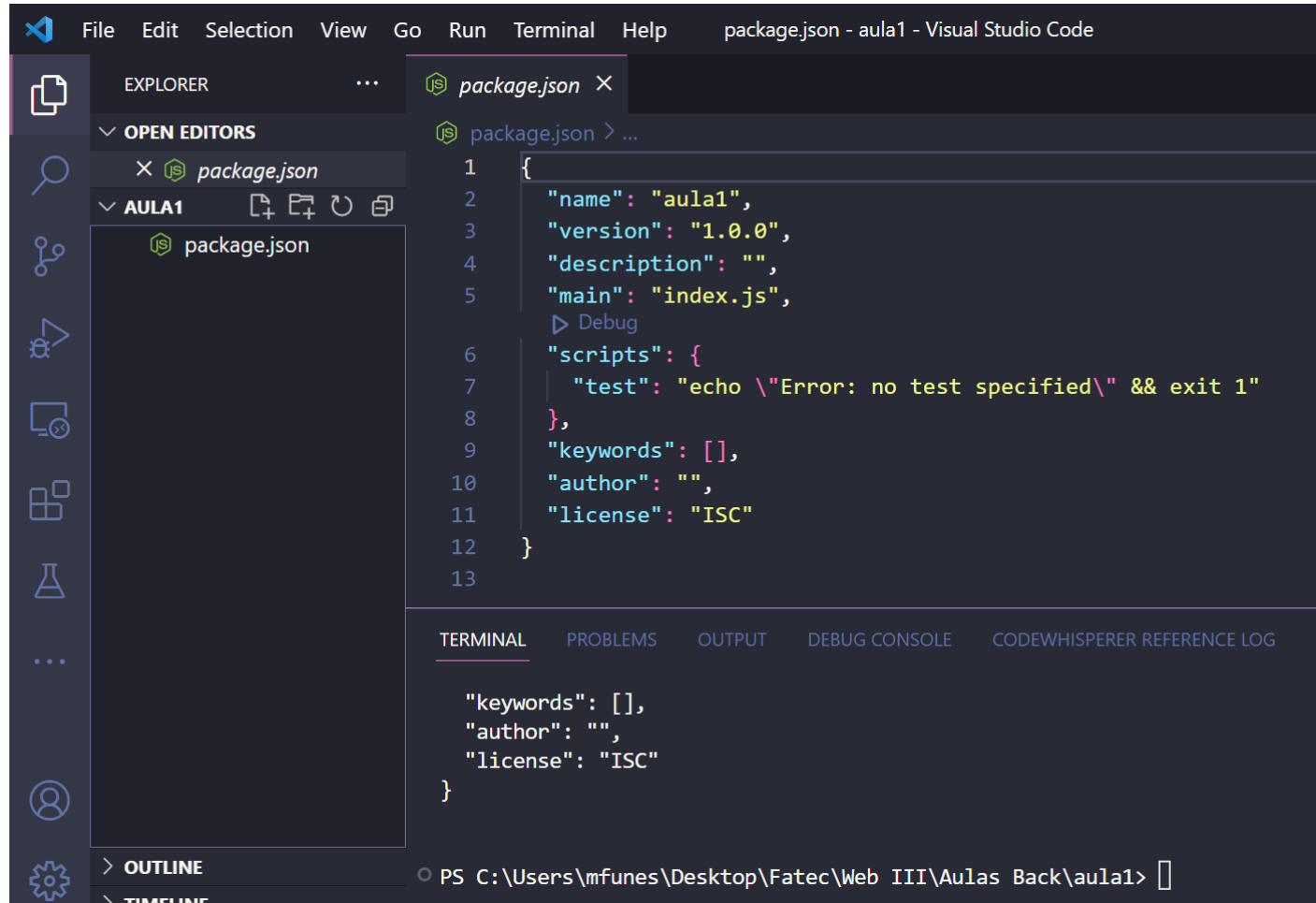
- Crie uma pasta chamada aula1
- Abra sua pasta no VSCODE
- No terminal digite “node -v”

Iniciando - NodeJS

- Criando aplicação NodeJS
- No terminal digite
 - `npm init -y`
 - ou
 - `yarn init -y`

Iniciando- NodeJS

- package.json



The screenshot shows the Visual Studio Code interface with the `package.json` file open in the editor. The Explorer sidebar on the left shows the file structure with `AULA1` containing `package.json`. The editor displays the following JSON content:

```
1 {
2   "name": "aula1",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "keywords": [],
10  "author": "",
11  "license": "ISC"
12 }
13
```

The bottom panel shows the TERMINAL tab with the following output:

```
"keywords": [],
"author": "",
"license": "ISC"
}
```

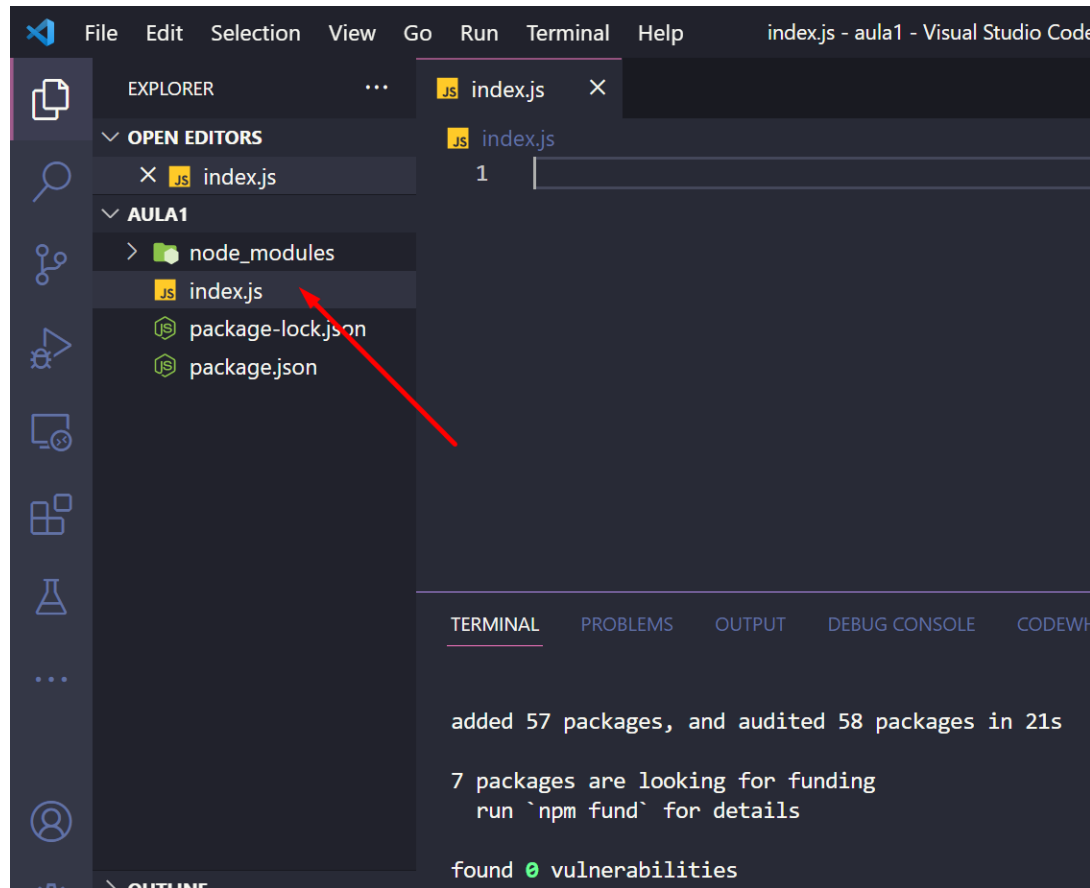
The terminal prompt is `PS C:\Users\mfunes\Desktop\Fatec\Web III\Aulas Back\aula1>`.

Express

- Framework web rápido, flexível e minimalista para Node.js
- <https://expressjs.com/pt-br/>
- No terminal digite
 - **npm** install express --save

Index.js

- Crie um arquivo chamado index.js



Index.js

JS index.js X

JS index.js > ...

```
1  const express = require('express');  
2  
3  console.log(express);
```

Index.js

- No terminal digite:
 - `node index.js`

```
[Function: createApplication] {  
  application: {  
    init: [Function: init],  
    defaultConfiguration: [Function: defaultConfiguration],  
    lazyrouter: [Function: lazyrouter],  
    handle: [Function: handle],  
    use: [Function: use],  
    route: [Function: route],  
    engine: [Function: engine],  
    param: [Function: param],  
    set: [Function: set],  
    path: [Function: path],  
    enabled: [Function: enabled],  
    disabled: [Function: disabled],  
    enable: [Function: enable],
```


Meu primeiro servidor

JS index.js X

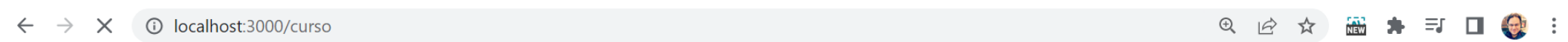
JS index.js > ...

```
1  const express = require('express');  
2  const server = express();  
3  
4  server.listen(3000);
```

Minha primeira API

```
JS index.js >  server.get('/curso') callback
1  const express = require('express');
2  const server = express();
3
4  //localhost:3000/curso
5  server.get('/curso', () => {
6      console.log('ALGUEM DEU UM GET NA ROTA CURSO');
7  });
8
9  server.listen(3000);
```

Minha primeira API






Cannot GET /curso

Testando minha API

- <https://reqbin.com/>

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 ▾

▾ ▾

Authorization

 Content Headers Raw (2)

☒ Bearer Token ☐ Basic Auth ☐ Custom ☐ No Auth

Token

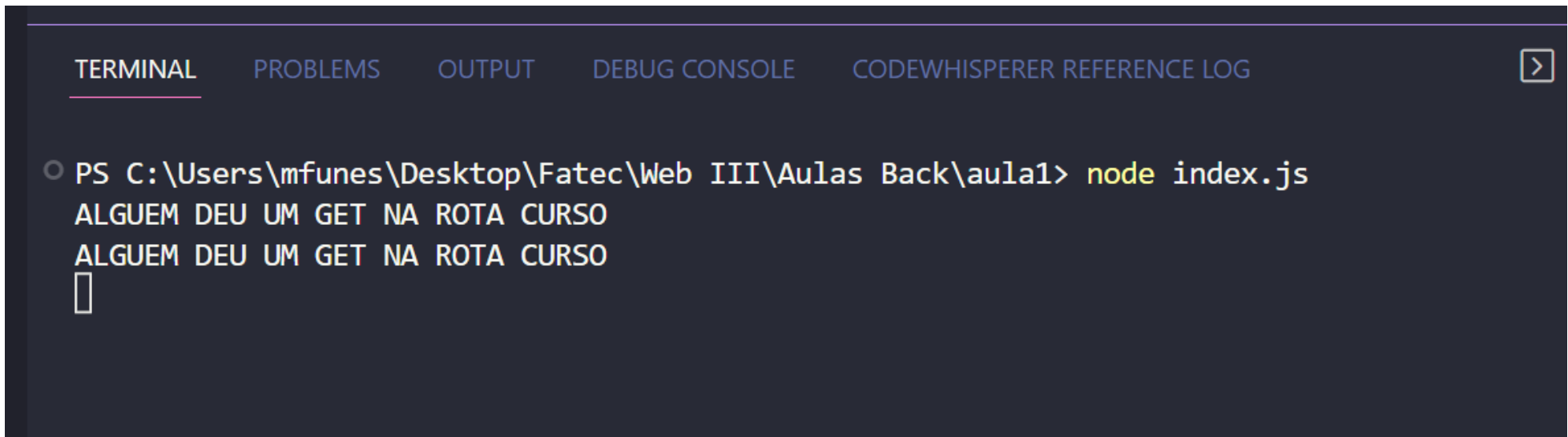
The authorization header will be automatically generated when you send the request. Read more about HTTP Authentication.

ReqBin Chrome Extension

Request timeout.

Testando minha API

- <https://reqbin.com/>




A screenshot of a VS Code terminal window. The terminal has tabs at the top: 'TERMINAL' (selected), 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'CODEWHISPERER REFERENCE LOG'. The terminal content shows a PowerShell prompt 'PS C:\Users\mfunes\Desktop\Fatec\Web III\Aulas Back\aula1>' followed by the command 'node index.js'. The output of the script is 'ALGUEM DEU UM GET NA ROTA CURSO' printed twice, with a cursor on the line following the second output.


```
PS C:\Users\mfunes\Desktop\Fatec\Web III\Aulas Back\aula1> node index.js
ALGUEM DEU UM GET NA ROTA CURSO
ALGUEM DEU UM GET NA ROTA CURSO
█
```

Tratando **Requisição** e **Resposta** da API

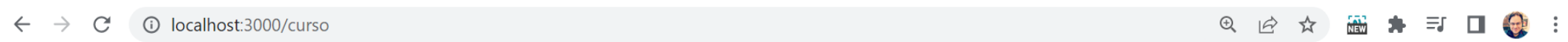
```
js index.js > ...  
1  const express = require('express');  
2  const server = express();  
3  
4  //localhost:3000/curso  
5  server.get('/curso', (req, res) => {  
6  
7  });  
8  
9  server.listen(3000);
```


Tratando **Requisição** e **Resposta** da API

```
JS index.js >  server.get('/curso') callback  
1  const express = require('express');  
2  const server = express();  
3  
4  //localhost:3000/curso  
5  server.get('/curso', (req, res) => {  
6    return res.send('Hello World');  
7  });  
8  
9  server.listen(3000);
```




Tratando **Requisição** e **Resposta** da API



Hello World



Enviando **Resposta** em JSON

```
JS index.js >  server.get('/curso') callback
1  const express = require('express');
2  const server = express();
3
4  //localhost:3000/curso
5  server.get('/curso', (req, res) => {
6    return res.json({ curso: 'Node JS' });
7  });
8
9  server.listen(3000);
```

Enviando **Resposta** em JSON

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

http://localhost:3000/curso

GET ▾

EXT ▾

Send

Status: **200 (OK)** Time: **59 ms** Size: **0.02 kb**

Authorization

Content

Headers

Raw (2)

Content (3)

Headers (8)

Raw (10)

JSON

☒ 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.

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

O que mais preciso saber para usar **NodeJS**?

- <https://nodejs.dev/pt/learn/introduction-to-nodejs/>

Certificações **NodeJS**?

- <https://openjsf.org/certification/>

