

Revisão - Módulo 4

Labenu_



Sumário

Labenu_



O que vamos ver hoje? 🙄

- Vamos visitar os tópicos mais importantes do módulo 4.

Relembrando:

- Semana 15: TS, Node e Express;
- Semana 16: MySQL e Knex ;
- Semana 17: Ferramentas de backend;



Semana 15

Labenu_



Semana 15

- Nesta semana vimos os seguintes conteúdos:
 - Node e Package.json;
 - Introdução à Typescript;
 - Express.js
 - APIs REST
- O que é necessário absorver dessa semana? Vamos por partes!



Semana 15

- **Node e Package.json**

- Conceito de node;
- NPM;
- Adicionar dependências;
- preencher o package.json

- **Introdução a TS**

- Tipagem de variáveis;
- Tipagem de funções (parâmetros e saídas);
- **type;**
- **enum;**

- **Express.js**

- Criando a primeira API;
- Headers, Body; Path e Query params;

- **APIs REST**

- Conceito de REST;
- Entidades;
- Métodos;



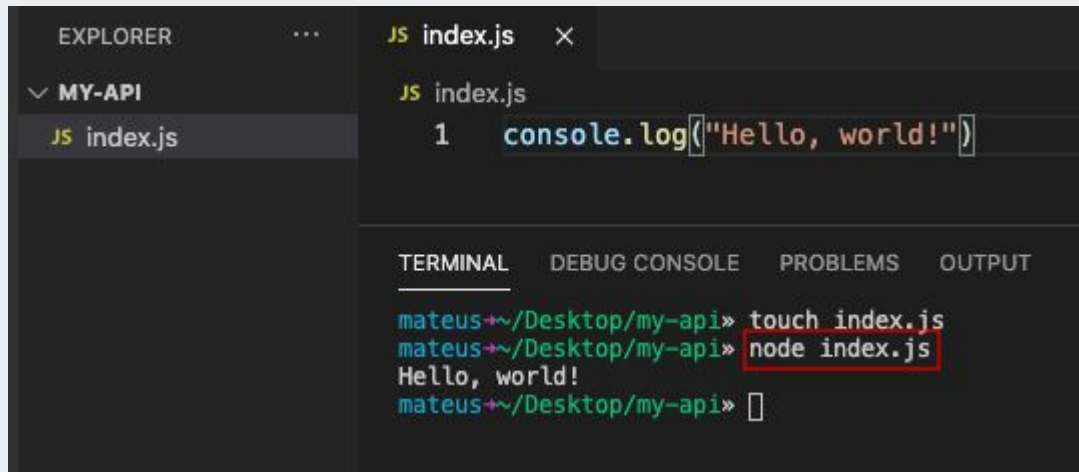
Node & Package.json

Labenu_



Conceito de node

O node é um **runtime**, isto é, um ambiente de execução de Javascript que roda no terminal do computador, permitindo que seu uso transcenda os navegadores.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar shows a folder named 'MY-API' containing a file 'index.js'. The main editor area displays the content of 'index.js', which is a single line of code: `console.log("Hello, world!");`. Below the editor, the TERMINAL panel is active, showing a series of commands and their output. The commands are: `mateus+~/Desktop/my-api» touch index.js`, `mateus+~/Desktop/my-api» node index.js` (the command is highlighted with a red box), and `mateus+~/Desktop/my-api»`. The output of the second command is `Hello, world!`.

```
EXPLORER  ...
  ▼ MY-API
    JS index.js

JS index.js
1 console.log("Hello, world!");

TERMINAL  DEBUG CONSOLE  PROBLEMS  OUTPUT
mateus+~/Desktop/my-api» touch index.js
mateus+~/Desktop/my-api» node index.js
Hello, world!
mateus+~/Desktop/my-api»
```



NPM

- **N**ode **P**ackage **M**anager, é o gerenciador de pacotes do Node
- De acordo com a documentação oficial, pacote é qualquer arquivo ou pasta contendo um programa descrito por um package.json (ex: React)
- Os pacotes são buscados e salvos com nomes únicos no NPM
- As bibliotecas que usamos estão no NPM



NPM

Os projetos que usam node tem seus atributos definidos por um arquivo de gerenciamento chamado **package.json**. Este arquivo é responsável por identificar e gerenciar o seu pacote, pois sim, você também pode subir seu projeto como pacote do NPM.

Neste arquivo estão contidas todas as informações sobre o que o projeto se trata, quem é o autor, quais são suas dependências e scripts customizados. Para adicionar dependências usamos:

- **npm install**: cria a pasta node_modules com as dependências listadas
- **npm install nomeDoPacote**: inclui o pacote especificado em "dependencies" e no node_modules
- **npm install nomeDoPacote --save-dev**: inclui o pacote especificado em "devDependencies" e no node_modules
- **npm run nomeDoComando**: procura o comando especificado em "scripts" e o executa



NPM

The screenshot shows a VS Code editor with three open files: `package.json`, `index.js`, and `.env`. The `package.json` file contains the following content:

```
{
  "name": "my-api",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node ./index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "dotenv": "^8.2.0"
  }
}
```

The `index.js` file contains the following content:

```
const dotenv = require("dotenv")
dotenv.config()
console.log(
  process.env.PASSWORD
)
```

The `.env` file contains the following content:

```
PASSWORD = ahninanab
```

The terminal window shows the following commands and output:

```
mateus~/Desktop/my-api> npm init -y
Wrote to /Users/mateus/Desktop/my-api/package.json:

{
  "name": "my-api",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

mateus~/Desktop/my-api> npm i dotenv
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN my-api@1.0.0 No description
npm WARN my-api@1.0.0 No repository field.

+ dotenv@8.2.0
added 1 package and audited 1 package in 0.645s
found 0 vulnerabilities

mateus~/Desktop/my-api> touch .env
mateus~/Desktop/my-api> npm run start

> my-api@1.0.0 start /Users/mateus/Desktop/my-api
> node ./index.js

ahninanab
mateus~/Desktop/my-api>
```



Introdução a Typescript

Labenu_



Typescript

- Linguagem orientada a objetos, construída sobre o JS (*superset*)
- Permite uso das sintaxes modernas, como **import/export**
- Na maioria dos casos, não é diretamente interpretada pelo Node, precisando ser **transpilada** em JS

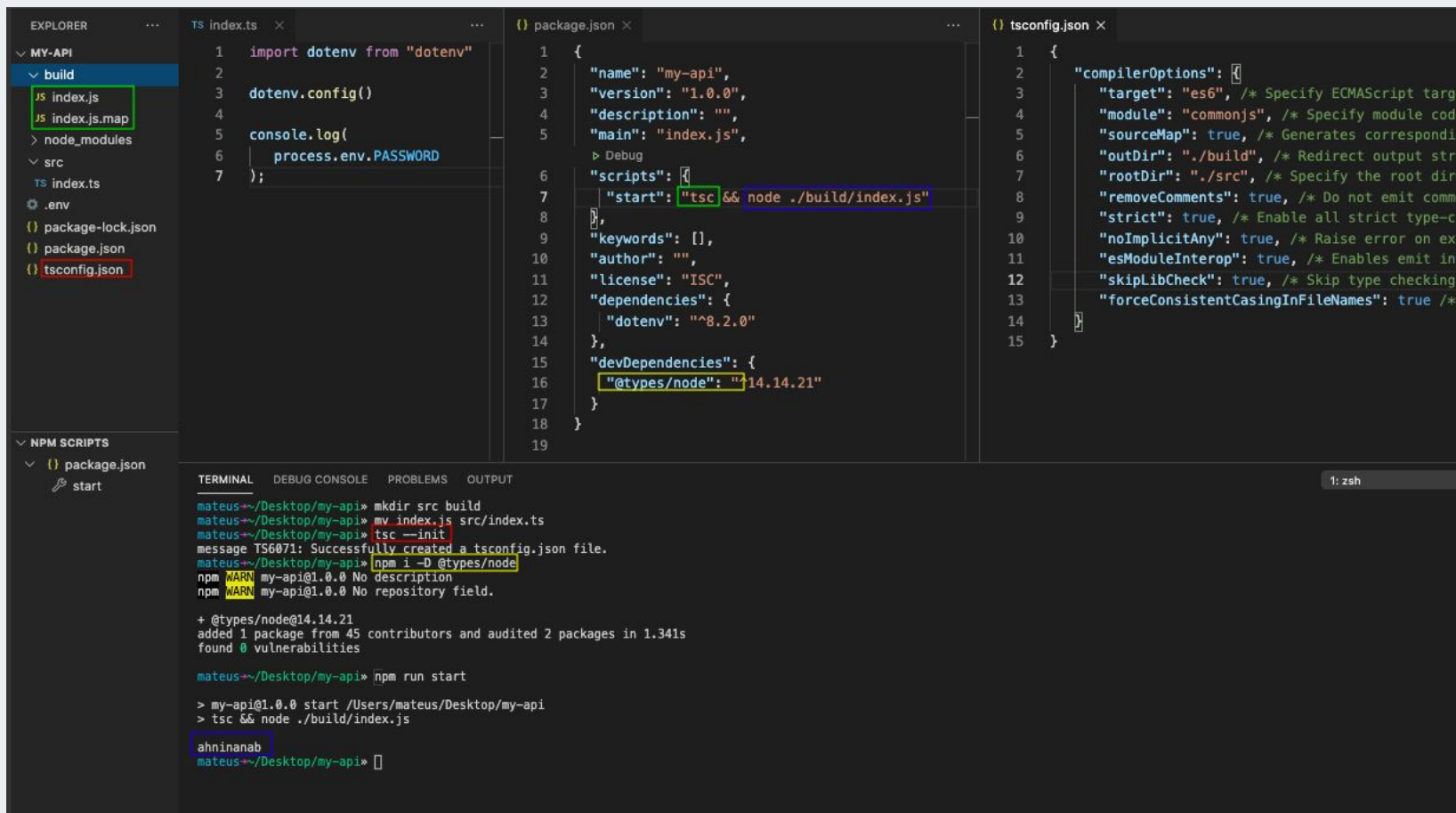


Instalação e Configuração

- Recomenda-se instalar globalmente: `npm i -g typescript`
- A maioria das lib's JS possui uma versão tipada.
- Para utilizar os módulos embutidos (*fs, process, require...*), precisamos instalar o `@types/node`
- Criamos um **tsconfig.json** para especificar todas as configurações desejadas
- Separamos o código fonte (TS) dos arquivos que serão de fato executados (JS), criando as pastas **src** e **build**



Instalação e Configuração



The image shows a VS Code editor with three files open: `index.ts`, `package.json`, and `tsconfig.json`.

index.ts:

```
1 import dotenv from "dotenv"
2
3 dotenv.config()
4
5 console.log(
6   process.env.PASSWORD
7 );
```

package.json:

```
1 {
2   "name": "my-api",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "start": "tsc && node ./build/index.js"
8   },
9   "keywords": [],
10  "author": "",
11  "license": "ISC",
12  "dependencies": {
13    "dotenv": "^8.2.0"
14  },
15  "devDependencies": {
16    "@types/node": "^14.14.21"
17  }
18 }
```

tsconfig.json:

```
1 {
2   "compilerOptions": {
3     "target": "es6", /* Specify ECMAScript target version
4     "module": "commonjs", /* Specify module code generation
5     "sourceMap": true, /* Generates corresponding
6     "outDir": "./build", /* Redirect output structure
7     "rootDir": "./src", /* Specify the root directory
8     "removeComments": true, /* Do not emit comments
9     "strict": true, /* Enable all strict type-checking
10    "noImplicitAny": true, /* Raise error on expressions
11    "esModuleInterop": true, /* Enables emit interop
12    "skipLibCheck": true, /* Skip type checking of
13    "forceConsistentCasingInFileNames": true /*
14  }
15 }
```

Terminal:

```
mateus~/Desktop/my-api» mkdir src build
mateus~/Desktop/my-api» mv index.js src/index.js
mateus~/Desktop/my-api» tsc --init
message TS6071: Successfully created a tsconfig.json file.
mateus~/Desktop/my-api» npm i -D @types/node
npm WARN my-api@1.0.0 No description
npm WARN my-api@1.0.0 No repository field.

+ @types/node@14.14.21
added 1 package from 45 contributors and audited 2 packages in 1.341s
found 0 vulnerabilities

mateus~/Desktop/my-api» npm run start

> my-api@1.0.0 start /Users/mateus/Desktop/my-api
> tsc && node ./build/index.js

ahninanab
mateus~/Desktop/my-api»
```

Tipando variáveis

- A principal diferença na hora de escrever em TS são os types

```
const title: string = "X-Men: O Filme"
```

```
let year: number = 2000
```

- Variáveis podem ter mais de um valor, e podem valer coisas como

```
let description: string | undefined
```

```
let yearOrTitle: any = 1977;
```

```
yearOrTitle = "Star Wars: A New Hope";
```



Tipando funções

Em funções, precisamos declarar os tipos dos parâmetros e também das saídas.

Quando passamos funções como parâmetros (o famoso callback) precisamos tipar também seus parâmetros e saídas.

```
async function createMovie(  
  title: string,  
  year: number,  
  validateInput: (t: string, y: number) => boolean  
) : Promise<void> {  
  
  const inputIsValid: boolean = validateInput(title, year)  
  if (inputIsValid) {  
    await insertIntoDB(title, year)  
  }  
}
```



Type

- Para não ficar repetindo código, podemos declarar uma **variável de tipo**:
- O type é apenas um "esqueleto" que definirá as propriedades que aquele tipo deve ter.

```
type movie = {  
  id:number,  
  title:string,  
  year:number  
}
```

```
const movies: movie[]=[  
  {  
    id:1,  
    title:"X-men: O Filme",  
    year:2000  
  }, {  
    id:2,  
    title:"Deadpool",  
    year:2016  
  }  
]
```



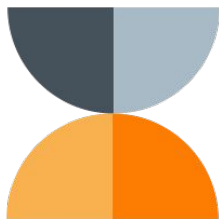
Enum

- Em Typescript, temos uma estrutura de dados que permite a declaração de tipos de variáveis quando **elas podem assumir valores restritos pré-definidos** (dados que não mudam):

```
enum GENDER {  
  MALE = "male",  
  FEMALE = "female",  
  OTHER = "other"  
}  
  
type character = {  
  id: number  
  name: string  
  gender: GENDER  
  description?: string  
}
```

```
const characters:character[]=[  
  {  
    id: 1,  
    name: "Storm",  
    gender: GENDER.FEMALE  
  }, {  
    id: 2,  
    name: "Deadpool",  
    gender: GENDER.OTHER,  
    description: "Sexy motherf***"  
  }, {  
    id: 3,  
    name: "Colossus",  
    gender: GENDER.MALE,  
  }  
]
```





- **Node e Package.json**
 - Conceito de node;
 - NPM;
 - Adicionar dependências;
 - preencher o package.json
- **Introdução a TS**
 - Tipagem de variáveis;
 - Tipagem de funções (parâmetros e saídas);
 - ○ **type**;
 - ○ **enum**;



Express.js

Labenu_

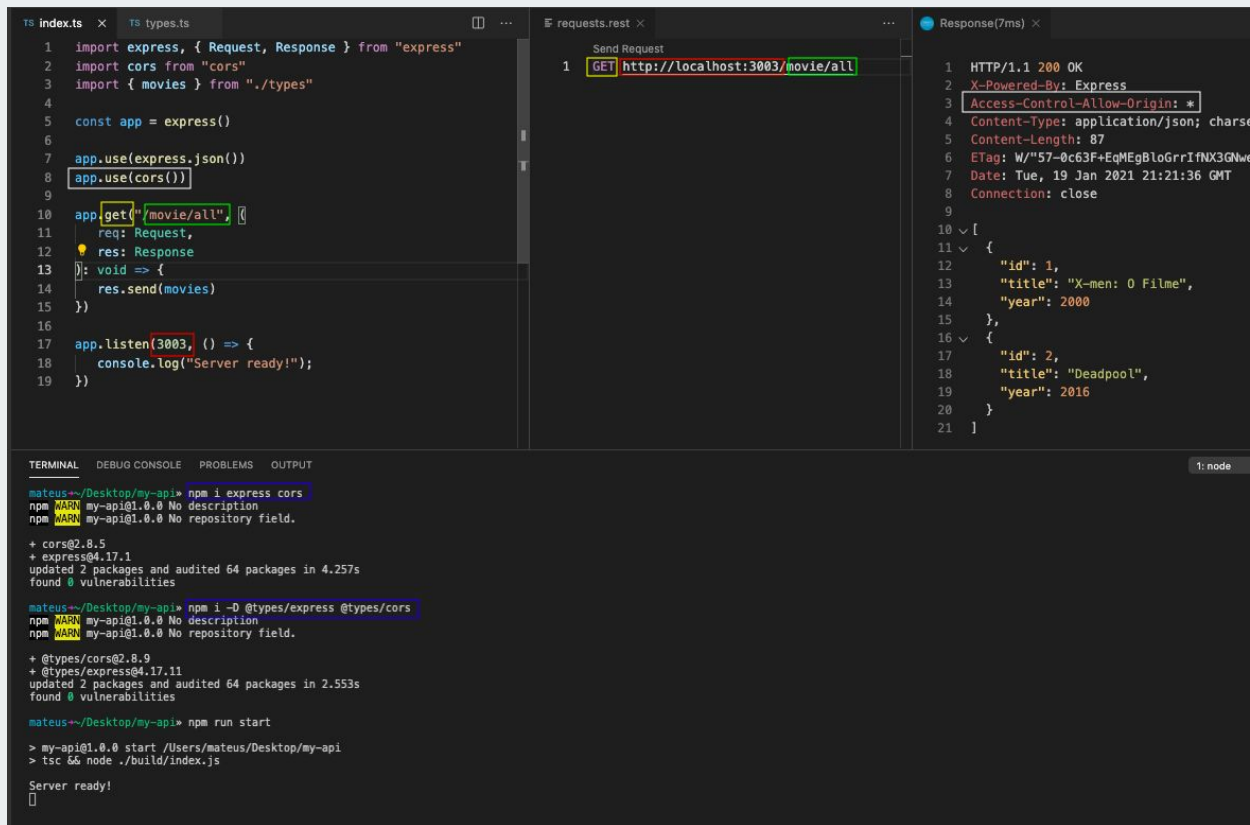


Express.js: criando uma API

O express é uma biblioteca construída sobre o módulo **HTTP** do Node para criarmos um **servidor**. Seus endpoints sempre têm o mesmo formato, permitindo o recebimento de **requests** (solicitações) e o envio de **responses** (respostas).



Instalação e Configuração



```
TS index.ts x TS types.ts x ...
1 import express, { Request, Response } from "express"
2 import cors from "cors"
3 import { movies } from "../types"
4
5 const app = express()
6
7 app.use(express.json())
8 app.use(cors())
9
10 app.get("/movie/all", (req, res) => {
11   res.send(movies)
12 })
13
14 app.listen(3003, () => {
15   console.log("Server ready!")
16 })
```

```
requests.rest x
1 GET http://localhost:3003/movie/all
```

```
Response(7ms) x
1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Access-Control-Allow-Origin: *
4 Content-Type: application/json; charset=utf-8
5 Content-Length: 87
6 ETag: W/"57-0c63F+EgMEgBloGrrIFnX3GNw"
7 Date: Tue, 19 Jan 2021 21:21:36 GMT
8 Connection: close
9
10 [
11   {
12     "id": 1,
13     "title": "X-men: O Filme",
14     "year": 2000
15   },
16   {
17     "id": 2,
18     "title": "Deadpool",
19     "year": 2016
20   }
21 ]
```

```
TERMINAL DEBUO CONSOLE PROBLEMS OUTPUT
mateus~/Desktop/my-api> npm i express cors
npm WARN my-api@1.0.0 No description
npm WARN my-api@1.0.0 No repository field.
+ cors@2.8.5
+ express@4.17.1
updated 2 packages and audited 64 packages in 4.257s
found 0 vulnerabilities

mateus~/Desktop/my-api> npm i -D @types/express @types/cors
npm WARN my-api@1.0.0 No description
npm WARN my-api@1.0.0 No repository field.
+ @types/cors@2.8.9
+ @types/express@4.17.11
updated 2 packages and audited 64 packages in 2.553s
found 0 vulnerabilities

mateus~/Desktop/my-api> npm run start
> my-api@1.0.0 start /Users/mateus/Desktop/my-api
> tsc && node ./build/index.js

Server ready!
```



Sintaxe: Request e Response

```
TS index.ts x ... requests.rest x ... Response(31ms) x

17 app.post("/character/new", (
18   req: Request,
19   res: Response
20 ): void => {
21
22   console.log(req.headers.auth)
23
24   const newCharacter: character = {
25     id: Date.now(),
26     name: req.body.name as string,
27     gender: req.body.gender as GENDER,
28     description: req.body.description as string
29   }
30
31   characters.push(newCharacter)
32
33   res.statusCode = 201 // "Created"
34   res.statusMessage = "A new character is born"
35   res.send({newCharacter})
36 })

Send Request
1 POST http://localhost:3003/character/new
2 Content-Type: application/json
3 auth: bananinha
4
5 {
6   "name": "Juggernaut",
7   "gender": "MALE",
8   "description": "O irmão do Professor X!!!"
9 }

1 HTTP/1.1 201 A new character is born
2 X-Powered-By: Express
3 Access-Control-Allow-Origin: *
4 Content-Type: application/json; charset=utf-8
5 Content-Length: 116
6 ETag: W/"74-GDUJg0GjFBX4Irrd0877H8cdd2I"
7 Date: Tue, 19 Jan 2021 22:01:29 GMT
8 Connection: close
9
10 {
11   "newCharacter": {
12     "id": 1611093689061,
13     "name": "Juggernaut",
14     "gender": "MALE",
15     "description": "O irmão do Professor X!!!"
16   }
17 }
```

TERMINAL DEBUG CONSOLE PROBLEMS OUTPUT

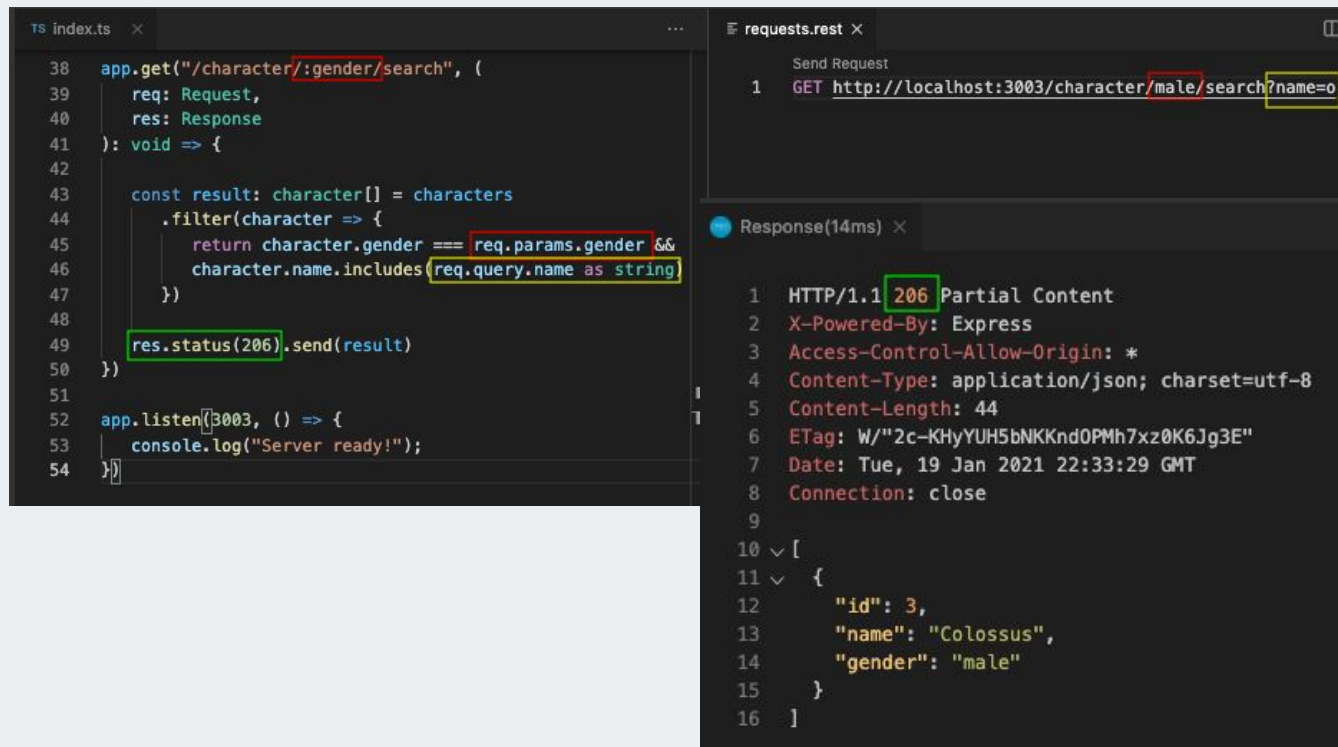
> Executing task: npm run start <

> my-api@1.0.0 start /Users/mateus/Desktop/my-api
> tsc && node ./build/index.js

Server ready!
bananinha



Sintaxe: Request e Response



The image shows a VS Code editor with two panels. The left panel displays the `index.ts` file with Express.js route and listener code. The right panel shows the REST client interface with a GET request and its corresponding HTTP response and JSON body.

```
38 app.get("/character/:gender/search", (  
39   req: Request,  
40   res: Response  
41 ): void => {  
42  
43   const result: character[] = characters  
44     .filter(character => {  
45       return character.gender === req.params.gender &&  
46         character.name.includes(req.query.name as string)  
47     })  
48  
49   res.status(206).send(result)  
50 })  
51  
52 app.listen(3003, () => {  
53   console.log("Server ready!");  
54 })
```

requests.rest

Send Request

```
1 GET http://localhost:3003/character/male/search?name=0
```

Response(14ms)

```
1 HTTP/1.1 206 Partial Content  
2 X-Powered-By: Express  
3 Access-Control-Allow-Origin: *  
4 Content-Type: application/json; charset=utf-8  
5 Content-Length: 44  
6 ETag: W/"2c-KHyYUH5bNKKndOPMh7xz0K6Jg3E"  
7 Date: Tue, 19 Jan 2021 22:33:29 GMT  
8 Connection: close  
9  
10 [  
11   {  
12     "id": 3,  
13     "name": "Colossus",  
14     "gender": "male"  
15   }  
16 ]
```



APIs REST

Labenu_



APIs REST: Conceito de REST

APIs são interfaces dadas para outros sistemas para que possam interagir com nossos dados. O tipo mais comum de API é a API REST, que tem sua estrutura dividida em **entidades** e **métodos**.



APIs REST: Entidades

- parâmetros de **path**
 - São passados logo após a entidade
 - Feitos para receber dados restritos
- parâmetros de **query**
 - Passados no final da URL num esquema de chave-valor
 - Pode receber múltiplos valores
- parâmetros de **body**
 - Enviados como JSON.
 - Chave-valor.



APIs REST: Métodos

- **GET**

- Utilizado para buscar recursos
- Não recebe body

- **POST**

- Insere ou atualiza recursos
- Mais versátil dos métodos

- **PUT**

- Atualiza ou insere recursos
- Idempotente

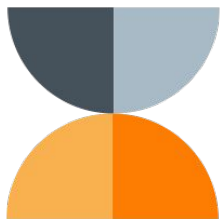
- **PATCH**

- Atualiza recursos
- Menos comum dos métodos

- **DELETE**

- Remove recursos





- **Express.js**
 - Criando a primeira API;
 - Path, Query e Body params;
- **APIs REST**
 - Conceito de REST;
 - Entidades;
 - Métodos;



Semana 16

Labenu_



Semana 16

- Nesta semana vimos os seguintes conteúdos:
 - Bancos de dados e Introdução a SQL;
 - Aprofundamento SQL;
 - Knex e Express;
 - Relações em SQL
- O que é necessário absorver dessa semana? Vamos por partes!



Semana 16

- **Introdução a SQL**

- Conceito de BD e SQL;
- Comandos de **CREATE**, **INSERT** e **SELECT**

- **Aprofundamento SQL**

- Comandos **ALTER**, **UPDATE**, **DELETE**, **DROP** e **TRUNCATE**;
- Funções no SQL;
- **SELECT** com **as**, **GROUP BY**, **ORDER BY** e **LIMIT**

- **Knex**

- Knex, Raw e QueryBuilder;
- Assincronicidade

- **Relações em SQL**

- Tipos de relação;
- Foreign Keys;
- Joins;



Introdução a SQL

Labenu_



Bancos de Dados e SQL

Bancos de dados são programas feitos para gerenciar grandes quantidade de registros.

São subdivididos em **SQL** e **NoSQL**

Bancos **SQL** são bancos relacionais, isto é, estruturados pensando em **relacionamentos** entre tabelas.



CREATE, INSERT e SELECT

```
CREATE TABLE movies(           -- CRIANDO A TABELA DE FILMES
    id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    year INT NOT NULL
);
```

```
INSERT INTO movies (title, year) -- INSERINDO FILMES
VALUES
    ("X-men: O Filme", 2000),
    ("Deadpool", 2018);
```

```
SELECT *           -- BUSCANDO FILMES
FROM movies
WHERE title LIKE "%oo%";
```



Aprofundamento em SQL

Labenu_



+Comandos SQL

Alterando a estrutura da tabela

```
ALTER TABLE movies  
ADD director VARCHAR(255);
```

```
ALTER TABLE movies  
DROP COLUMN director;
```

Atualizando colunas

```
UPDATE movies  
SET year = 2016  
WHERE title = "Deadpool";
```

Deletando registros na tabela

```
DELETE FROM movies  
WHERE year < 2015;
```

Deletando Tabelas

```
DROP TABLE movies;
```



Funções em SQL

Funções são trechos reutilizáveis de tratamento de dados no SQL. As funções alteram apenas a exibição dos dados, não alterando sua estrutura

```
SELECT CONCAT("The Movie ", title, " was  
released in ", year)  
FROM movies;
```



GROUP BY, ORDER BY E LIMIT

GROUP BY agrupa os dados a partir de uma coluna ou grupo de colunas:

```
SELECT COUNT(*), gender
FROM characters
GROUP BY gender;
```

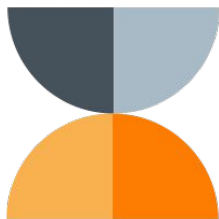
ORDER BY ordena os dados a partir de uma coluna ou grupo de colunas:

```
SELECT *
FROM movies
ORDER BY year DESC;
```

LIMIT define quantidade de registros a se exibir:

```
SELECT *
FROM movies
ORDER BY year DESC
LIMIT 10;
```





- **Introdução a SQL**
 - Conceito de BD e SQL;
 - Comandos de **CREATE**, **INSERT** e **SELECT**
- **Aprofundamento SQL**
 - Comandos **ALTER**, **UPDATE**, **DELETE**, **DROP** e **TRUNCATE**;
 - Funções no SQL;
 - **SELECT** com **as**, **GROUP BY**, **ORDER BY** e **LIMIT**



Knex

Labenu_



Knex

- O **knex** é uma biblioteca construída sobre diversos *drivers* de bancos de dados, entre eles, o **mysql**, permitindo automatizar o acesso ao banco através do JS/TS
- É utilizado em conjunto com o **dotenv**, que serve para evitar que dados sensíveis (credenciais do banco) sejam exportados com o restante do código



Knex

```
TS index.ts x ... TS index.ts x ... .env x ... .gitignore x
4 import knex from "knex"
5 import dotenv from "dotenv"
6
7 dotenv.config()
8
9 const connection = knex({
10   client: "mysql",
11   connection: {
12     host: process.env.DB_HOST,
13     port: 3306,
14     user: process.env.DB_USER,
15     password: process.env.DB_PASSWORD,
16     database: process.env.DB_NAME
17   }
18 })

25 app.get("/movie/all", async (
26   req: Request,
27   res: Response
28 ): Promise<void> => {
29   try {
30     const result: any = await connection.raw(`
31       SELECT * FROM movie
32     `)
33     res.send(result[0])
34   } catch (error) {
35     res.status(400).send(
36       error.sqlMessage || error.message
37     )
38   }
39 })

1 DB_HOST = ec2-18-229-236-15.sa-east-
2 DB_USER = mateus-gesualdo
3 DB_PASSWORD = nao_eh_bananinha
4 DB_NAME = teachers-mateus-gesualdo

1 node_modules
2 package-lock.json
3 build
4 .env

TERMINAL DEBUG CONSOLE PROBLEMS OUTPUT
mateus@~/Desktop/my-api$ npm i mysql knex dotenv [21:28:07]
npm WARN deprecated resolve-url@0.2.1: https://github.com/lydell/resolve-url#deprecated
npm WARN deprecated urix@0.1.0: Please see https://github.com/lydell/urix#deprecated
npm WARN my-api@1.0.0 No description
npm WARN my-api@1.0.0 No repository field.

+ mysql@2.18.1
+ dotenv@8.2.0
+ knex@0.21.16
added 163 packages from 149 contributors, updated 1 package and audited 227 packages in 11.286s

2 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

mateus@~/Desktop/my-api$ npm i -D @types/knex [21:23:37]
npm WARN deprecated @types/knex@0.16.1: This is a stub types definition. knex provides its own type
so you do not need this installed.
npm WARN my-api@1.0.0 No description
npm WARN my-api@1.0.0 No repository field.

+ @types/knex@0.16.1
updated 1 package and audited 228 packages in 2.344s

2 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

mateus@~/Desktop/my-api$ touch .gitignore [21:56:27]
mateus@~/Desktop/my-api$
```



Knex: Raw e QueryBuilder

O Knex Raw permite que escrevamos texto em SQL diretamente dentro do TS. O principal problema com isso é na hora de fazer buscas, onde os dados vem dentro de arrays com metadados

O queryBuilder permite que usemos uma sintaxe mais próxima ao JS/TS que estamos acostumados, deixando o código com uma cara mais procedural.



async e await

- Promises são valores que o JS/TS usa para lidar com questões assíncronas, isto é, que não tem resultado imediato, podendo ou não dar certo no caminho.
- *async*
 - **Toda** função **async** retorna uma **Promise** (padrão dessa linguagem)
 - Somente dentro de uma função **async**, podemos usar o **await**
- *Promise*
 - É o que representa a **saída** de uma função **async**
 - Ele permite que peguemos o resultado de sucesso (**then**) e o de erro (**catch**)
- *await*
 - É usado para "esperar" uma Promise ser concluída
 - Permite pegar diretamente o **resultado de sucesso**



Tipando Promises

Apesar de ser um tipo, as **Promises** também podem ser tipadas, para que tenhamos controle do que esperar quando a promise for resolvida. Para isso, colocamos o tipo de valor esperado após a palavra promise, **<T>**

Função que não espera retorno:

Promise<void>

Função que espera um número:

Promise<number>

Função que espera um Array de strings:

Promise<string[]>



try/catch e then/catch

Quando usamos `async` e `await`, precisamos estar em uma **função**.

```
const getAllMovies = async (req, res): Promise<void> => {  
  try {  
    const result: any = await connection.raw(`  
      SELECT * FROM movie  
    `)  
    res.send(result[0])  
  } catch (error) {  
    res.status(400).send(  
      error.sqlMessage || error.message  
    )  
  }  
}
```

Quando não usamos `async` e `await`, precisamos lidar com o que fazer quando o resultado da promise retornar:

```
const query = connection.raw(`  
  SELECT * FROM movie  
`)  
  
query  
  .then(result => {  
    console.table(result[0])  
  })  
  .catch(error => {  
    console.log(error.sqlMessage)  
  })
```



Relações em SQL

Labenu_



Foreign Keys

Foreign Keys são chaves primárias que vem de outras tabelas, para garantir a integridade da relação.

```
CREATE TABLE movie_character_relations(  
    movie_id INT,  
    character_id INT,  
    FOREIGN KEY(movie_id) REFERENCES movies(id),  
    FOREIGN KEY(character_id) REFERENCES characters(id)  
);
```

```
CREATE TABLE movies(  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    title VARCHAR(255) NOT NULL,  
    year INT NOT NULL  
);
```

N:M

```
CREATE TABLE characters(  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    name VARCHAR(255) NOT NULL,  
    gender ENUM('female', 'male', 'other'),  
    description VARCHAR(255)  
);
```



Joins

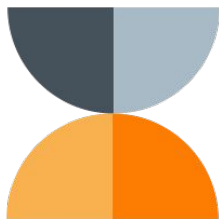
Quando queremos buscar informações de mais de uma tabela, precisamos fazer **JOINS**, isto é, encontrar referências de uma tabela em outra. É aqui que as foreign keys vêm a calhar.

```
SELECT title, name
FROM movie_character_relations
JOIN movies
ON movie_id = movies.id
JOIN characters
ON character_id =
characters.id;
```



title	name
X-Men: O Filme	Storm
X-Men: O Filme	Colossus
Deadpool	Deadpool
Deadpool	Colossus





- **Knex**
 - Knex, Raw e QueryBuilder;
 - Assincronicidade
- **Relações em SQL**
 - Tipos de relação;
 - Foreign Keys;
 - Joins;



Semana 17

Labenu_



Semana 17

- Nesta semana vimos os seguintes conteúdos:
 - Filtros, Ordenação e paginação;
 - Debugging no backend
- O que é necessário absorver dessa semana? Vamos por partes!



Semana 17

- **Filtros, ordenação e paginação**
 - Filtros;
 - Ordenação;
 - Paginação
- **Debugging no Backend**



Filtros, Ordenação e Paginação

Labenu_



Filtros

Filtros são ferramentas de limitação de busca em um banco de dados. Utilizar filtros nos faz poupar tempo e recursos de quem acessará o sistema pelo front-end. Seu uso no banco de dados depende da cláusula **WHERE**.



Ordenação

A ordenação nos permite colocar itens mais relevantes em destaque para quem usa o sistema. Utilizar ordenação nos dá maior flexibilidade de critérios. Seu uso no banco de dados depende da cláusula **ORDER BY**



Paginação

A paginação é nosso principal recurso de otimização de feed. Utilizar paginação nos dá menores tempos de carregamento. Seu uso no banco de dados depende das cláusulas **LIMIT** e **OFFSET**.



Debugging no Backend

Labenu_



Debugging

- **Console**
 - **log**
 - **count**
 - **assert**
- **VSCode**
 - **launch.json**
- **Navegador (DevTools)**
 - **node --inspect index.js**





Dúvidas?





Obrigado!