

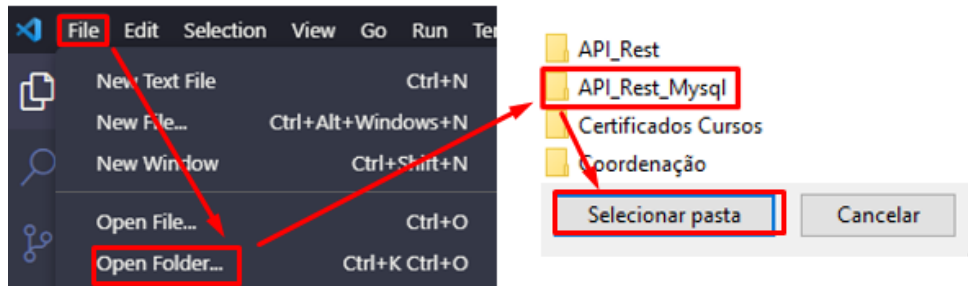
## API Rest com node



**Professor Mário de Jesus**

## Criando o Projeto

Crie uma pasta chamada **API\_Rest\_Mysql** e abra ela no Visual Studio Code

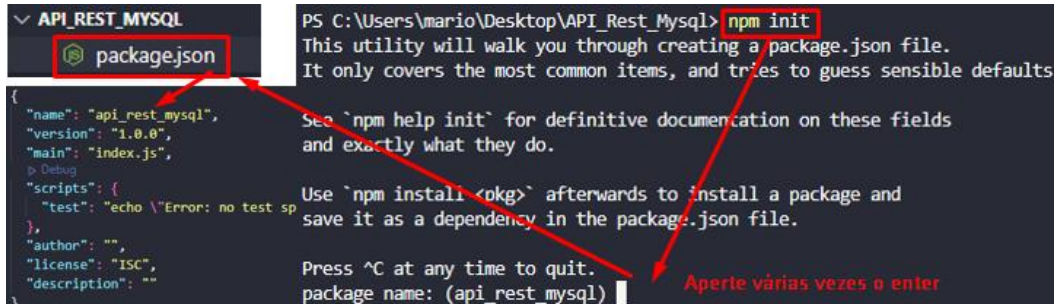


**OBS:** O comando `$ npm init` nos permite iniciar um pacote, criando o arquivo **package.json** de acordo com certas respostas que damos às perguntas feitas. Mas você pode **pular as perguntas**, fazendo com que o arquivo **package.json** seja criado imediatamente. Basta adicionar **-y** ao comando: `$ npm init -y`

## Configurando e instalando pacotes

Abra o terminal do Visual Studio Code e faça as instalações abaixo, para criar o **package.json**:

**npm init**



The screenshot shows the Visual Studio Code interface. On the left, the Explorer view shows a folder named 'API\_REST\_MYSQL' containing a file named 'package.json'. The file is highlighted with a red box. On the right, the integrated terminal shows the command 'npm init' being executed. The terminal output explains that 'npm init' will walk through creating a package.json file. It also shows the command 'npm install <pkg>' being used to install a package and save it as a dependency. A red arrow points from the 'package.json' file in the Explorer to the 'npm init' command in the terminal. Another red arrow points from the 'npm install <pkg>' command to the 'package.json' file. A red note at the bottom right says 'Aperte várias vezes o enter' (Press the enter key several times).

```
PS C:\Users\mario\Desktop\API_Rest_Mysql> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (api_rest_mysql) 
```

**npm install express --save**

```
PS C:\Users\mario\Desktop\API_Rest_Mysql> npm install express --save
```

**npm install -g nodemon**

```
PS C:\Users\mario\Desktop\API_Rest_Mysql> npm install -g nodemon
[.....] / idealTree:npm: sill idealTree buildDeps
```

**npm install --save sequelize**

```
found 0 vulnerabilities
PS C:\Users\mario\Desktop\API_Rest_Mysql> npm install sequelize
[.....] | idealTree:API_Rest_Mysql: sill idealTree buildDeps
```

**npm install --save body-parser**

```
PS C:\Users\mario\Desktop\API_Rest_Mysql> npm install --save body-parser
```

**npm install --save mysql2**

```
PS C:\Users\mario\Desktop\API_Rest_Mysql> npm install --save mysql2
[#####.....] - idealTree:API_Rest_Mysql: timing idealTree:#root Completed in 963ms
```

**npm install --save-dev sequelize-cli**

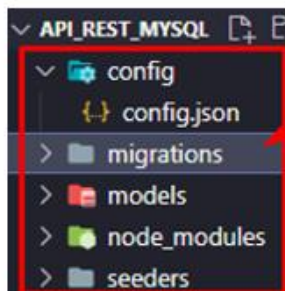
```
PS C:\Users\mario\Desktop\API_Rest_Mysql> npm install --save-dev sequelize-cli
[.....] | idealTree:API_Rest_Mysql: sill idealTree buildDeps
```

## Configurando nosso sequelize com as pastas e conexões

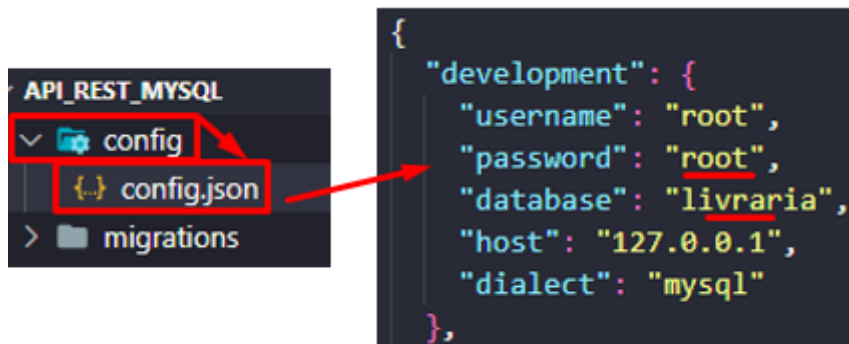
O comando abaixo cria as pastas **config**, **migrations**, **models** e **seeders**

**npx sequelize-cli init**

```
PS C:\Users\mario\Desktop\API_Rest_Mysql> npx sequelize-cli init  
Sequelize CLI [Node: 16.13.2, CLI: 6.5.1, ORM: 6.23.2]
```



Agora configure o arquivo **config.json** para criarmos o banco de dados **livraria**



Use o comando abaixo para criar a **database livraria**. Confira depois no mysql

**npx sequelize db:create**

```
PS C:\Users\mario\Desktop\API_Rest_Mysql> npx sequelize db:create  
Sequelize CLI [Node: 16.13.2, CLI: 6.5.1, ORM: 6.23.2]  
  
Loaded configuration file "config\config.json".  
Using environment "development".  
Database livraria created.  
PS C:\Users\mario\Desktop\API_Rest_Mysql>
```

## Vamos criar os models e os migrates da nossa database

`npx sequelize-cli model:generate --name Editora --attributes descricao:STRING`

```
PS C:\Users\mario\Desktop\API_Rest_Mysql> npx sequelize-cli model:generate --name Editora --attributes descricao:STRING
Sequelize CLI [Node: 16.13.2, CLI: 6.5.1, ORM: 6.23.2]
```

`npx sequelize-cli model:generate --name Categoria --attributes descricao:STRING`

```
PS C:\Users\mario\Desktop\API_Rest_Mysql> npx sequelize-cli model:generate --name Categoria --attributes descricao:STRING
Sequelize CLI [Node: 16.13.2, CLI: 6.5.1, ORM: 6.23.2]
```

`npx sequelize-cli model:generate --name Autor --attributes nome:STRING`

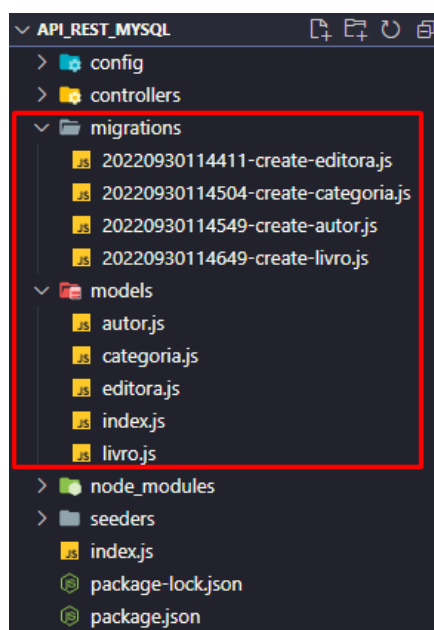
```
PS C:\Users\mario\Desktop\API_Rest_Mysql> npx sequelize-cli model:generate --name Autor --attributes nome:STRING
Sequelize CLI [Node: 16.13.2, CLI: 6.5.1, ORM: 6.23.2]
```

`npx sequelize-cli model:generate --name Livro --attributes  
fk_editora:INTEGER,fk_categoria:INTEGER,fk_autor:INTEGER,titulo:STRING`

**OBS:** Quanto a tabela tem mais de um campo não pode ter espaços

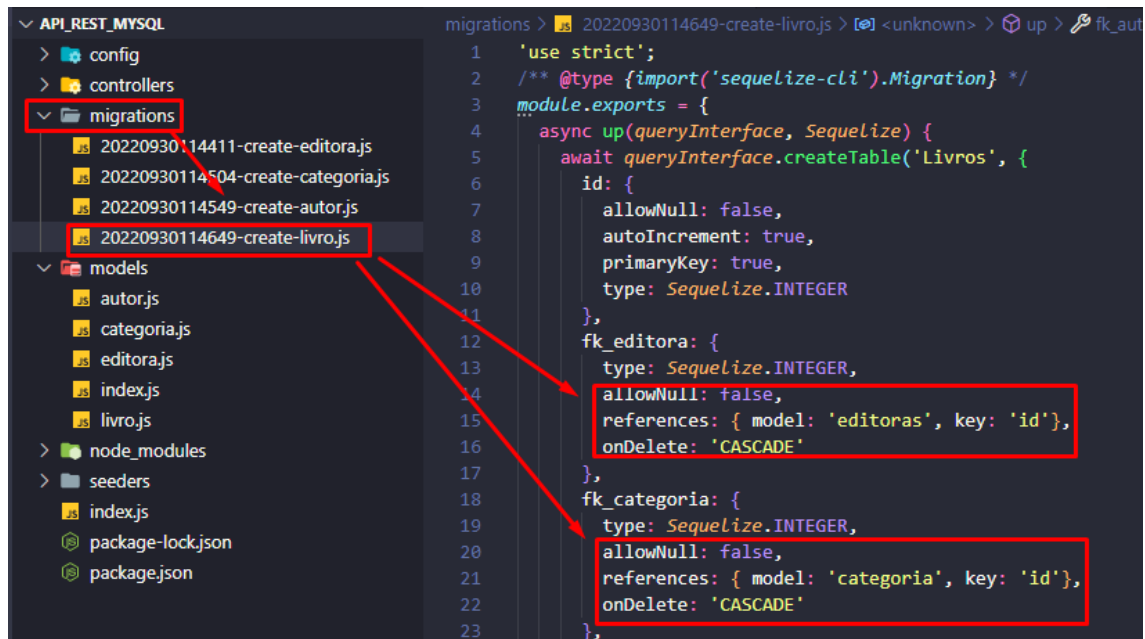
```
PS C:\Users\mario\Desktop\API_Rest_Mysql> npx sequelize-cli model:generate --name Livro --attributes fk_editora:INTEGER,fk_categoria:INTEGER,fk_autor:INTE
GER,titulo:STRING
Sequelize CLI [Node: 16.13.2, CLI: 6.5.1, ORM: 6.23.2]
```

Perceba que as pastas **models** e **migrations** ficaram assim:



## Alterando o migrate de livro

Como livro possui chaves estrangeiras devemos acrescentar os códigos de referência nos campos. Veja:



Código completo do create-livro na pasta migrations

```
'use strict';
/** @type {import('sequelize-cli').Migration} */
module.exports = {
  async up(queryInterface, Sequelize) {
    await queryInterface.createTable('Livros', {
      id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: Sequelize.INTEGER
      },
      fk_editoras: {
        type: Sequelize.INTEGER,
        allowNull: false,
        references: { model: 'editoras', key: 'id'},
        onDelete: 'CASCADE'
      },
      fk_categoria: {
        type: Sequelize.INTEGER,
        allowNull: false,
        references: { model: 'categoria', key: 'id'},
        onDelete: 'CASCADE'
      },
      fk_autor: {

```

```
    type: Sequelize.INTEGER,
    allowNull: false,
    references: { model: 'autors', key: 'id'},
    onDelete: 'CASCADE'
  },
  titulo: {
    type: Sequelize.STRING
  },
  createdAt: {
    allowNull: false,
    type: Sequelize.DATE
  },
  updatedAt: {
    allowNull: false,
    type: Sequelize.DATE
  }
});
},
async down(queryInterface, Sequelize) {
  await queryInterface.dropTable('Livros');
}
};
```

## Migrando as tabelas para o banco de dados livraria

Agora vamos usar o comando **migrate** para criar as tabelas de editora, categoria, autor e livro no banco de dados livraria

**npx sequelize db:migrate**

```
PS C:\Users\mario\Desktop\API_Rest_Mysql> npx sequelize db:migrate

Sequelize CLI [Node: 16.13.2, CLI: 6.5.1, ORM: 6.23.2]

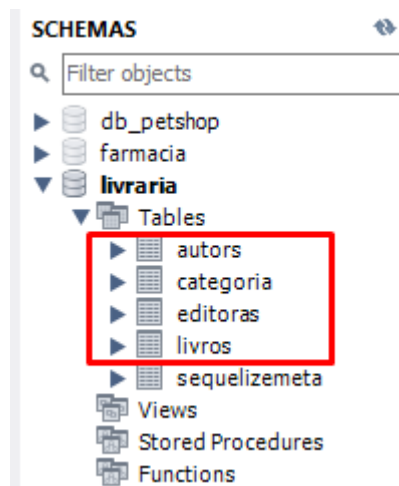
Loaded configuration file "config\config.json".
Using environment "development".
== 20220930114411-create-editora: migrating =====
== 20220930114411-create-editora: migrated (1.412s)

== 20220930114504-create-categoria: migrating =====
== 20220930114504-create-categoria: migrated (1.212s)

== 20220930114549-create-autor: migrating =====
== 20220930114549-create-autor: migrated (1.028s)

== 20220930114649-create-livro: migrating =====
== 20220930114649-create-livro: migrated (1.776s)
```

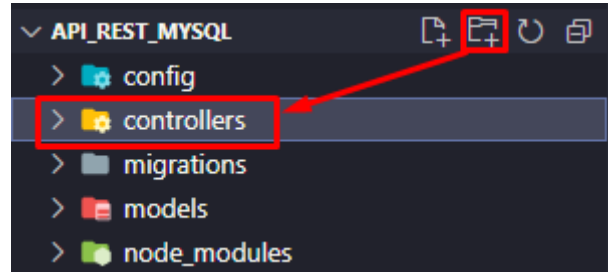
Verifique no banco dados mysql a criação das tabelas



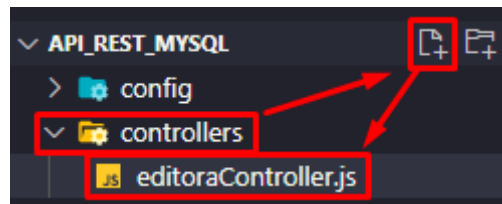


## Controllers (Editora)

Agora vamos criar nossas **rotas CRUD** para cada uma das tabelas. Crie a pasta **controllers** na raiz do seu projeto.



Dentro da pasta **controllers** crie o arquivo **editoraController.js**



### Código da editoraController.js

```
const express = require('express');
const router = express.Router();
//pegamos a entidade em si dessa forma usando .Editora
const Editora = require('../models').Editora;

//Busca Editoras (GET)
router.get('/', async (req, res) => {
  const editoras = await Editora.findAll();
  res.status(200).json(editoras);
});

//Cadastra Editoras (POST)
router.post('/', async (req, res) => {
  const {descricao} = req.body;
  const newEdit = await Editora.create({descricao})
  res.status(200).json({message: 'Cadastrado com sucesso'});
});

//Busca Por id a Editora (GET)
router.get('/:id', async (req, res) => {
  const id=req.params;
  const editora = await Editora.findByPk(req.params.id);
  res.status(200).json(editora);
});

//Deleta editora por id (DELETE)
```

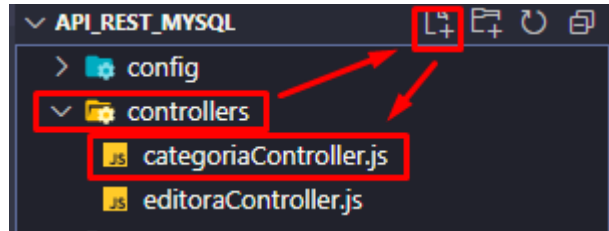
```
router.delete('/:id', async (req, res) =>{
  await Editora.destroy({
    where:{
      id: req.params.id,
    },
  });
  res.status(200).json({message: 'Excluído com sucesso'})
});

//Altera Editora por ID (PUT)
router.put('/:id', async (req, res) =>{
  const {descricao} = req.body;

  await Editora.update(
    { descricao},
    {
      where: {id: req.params.id},
    }
  );
  res.status(200).json({message: 'Atualizado com sucesso'});
});
module.exports=router;
```

## Controllers (Categoria)

Dentro da pasta **controllers** crie o arquivo **categoriaController.js**



### Código da categoriaController.js

```
const express = require('express');
const router = express.Router();
//pegamos a entidade em si dessa forma usando .Categoria
const Categoria = require('../models').Categoria;

//Busca Categorias (GET)
router.get('/', async (req, res) => {
  const categorias = await Categoria.findAll();
  res.status(200).json(categorias);
});

//Cadastra Categorias (POST)
router.post('/', async (req, res) => {
  const {descricao} = req.body;
  const newEdit = await Categoria.create({descricao})
  res.status(200).json({message: 'Cadastrado com sucesso'});
});

//Busca Por id a Categoria (GET)
router.get('/:id', async (req, res) => {
  const id=req.params;
  const categoria = await Categoria.findByPk(req.params.id);
  res.status(200).json(categoria);
});

//Deleta categoria por id (DELETE)
router.delete('/:id', async (req, res) =>{
  await Categoria.destroy({
    where:{
      id: req.params.id,
    },
  });
  res.status(200).json({message:'Excluído com sucesso'})
});

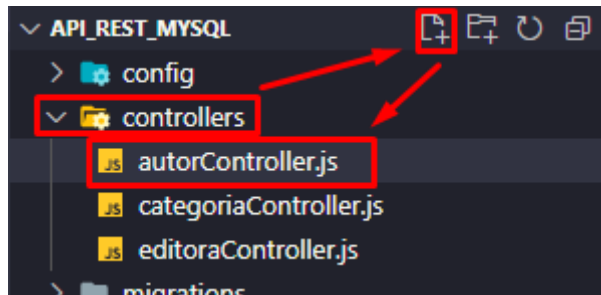
//Altera Categoria por ID (PUT)
router.put('/:id', async (req, res) =>{
```

```
const {descricao} = req.body;

await Categoria.update(
  { descricao},
  {
    where: {id:req.params.id},
  }
);
res.status(200).json({message: 'Atualizado com sucesso'});
});
module.exports=router;
```

## Controllers (Autor)

Dentro da pasta **controllers** crie o arquivo **autorController.js**



### Código da autorController.js

```
const express = require('express');
const router = express.Router();
//pegamos a entidade em si dessa forma usando .Autor
const Autor = require('../models').Autor;

//Busca Autor (GET)
router.get('/', async (req, res) => {
  const autores = await Autor.findAll();
  res.status(200).json(autores);
});

//Cadastra Autor (POST)
router.post('/', async (req, res) => {
  const {nome} = req.body;
  const newEdit = await Autor.create({nome})
  res.status(200).json({message: 'Cadastrado com sucesso'});
});

//Busca Por id o Autor (GET)
router.get('/:id', async (req, res) => {
  const id=req.params;
  const autor = await Autor.findByPk(req.params.id);
  res.status(200).json(autor);
});

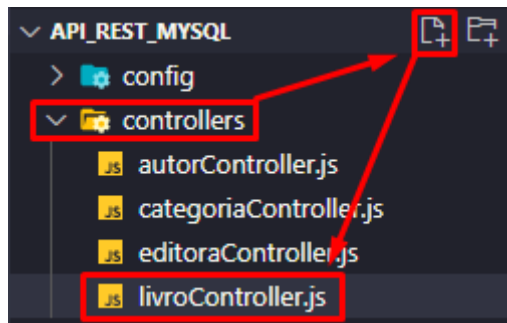
//Deleta Autor por id (DELETE)
router.delete('/:id', async (req, res) =>{
  await Autor.destroy({
    where:{
      id: req.params.id,
    },
  });
  res.status(200).json({message:'Excluído com sucesso'})
});
```

```
//Altera Autor por ID (PUT)
router.put('/:id', async (req, res) =>{
  const {nome} = req.body;

  await Autor.update(
    { nome},
    {
      where: {id:req.params.id},
    }
  );
  res.status(200).json({message: 'Atualizado com sucesso'});
});
module.exports=router;
```

## Controllers (Livro)

Dentro da pasta **controllers** crie o arquivo **livroController.js**



### Código da livroController.js

```
const express = require('express');
const router = express.Router();
//pegamos a entidade em si dessa forma usando .Livro
const Livro = require('../models').Livro;

//Busca Livro (GET)
router.get('/', async (req, res) => {
  const livros = await Livro.findAll();
  res.status(200).json(livros);
});

//Cadastra Livro (POST)
router.post('/', async (req, res) => {
  const {fk_editora, fk_categoria, fk_autor, titulo} = req.body;
  const newEdit = await Livro.create({fk_editora, fk_categoria,
fk_autor, titulo})
  res.status(200).json({message: 'Cadastrado com sucesso'});
});

//Busca Por id o Livro (GET)
router.get('/:id', async (req, res) => {
  const id=req.params;
  const livro = await Livro.findByPk(req.params.id);
  res.status(200).json(livro);
});

//Deleta Livro por id (DELETE)
router.delete('/:id', async (req, res) =>{
  await Livro.destroy({
    where:{
      id: req.params.id,
    },
  });
  res.status(200).json({message:'Excluído com sucesso'})
});
```

```
});  
  
//Altera Livro por ID (PUT)  
router.put('/:id', async (req, res) =>{  
  const {fk_editora, fk_categoria, fk_autor, titulo} = req.body;  
  
  await Autor.update(  
    { fk_editora, fk_categoria, fk_autor, titulo},  
    {  
      where: {id:req.params.id},  
    }  
  );  
  res.status(200).json({message: 'Atualizado com sucesso'});  
});  
module.exports=router;
```



## Criando nosso servidor

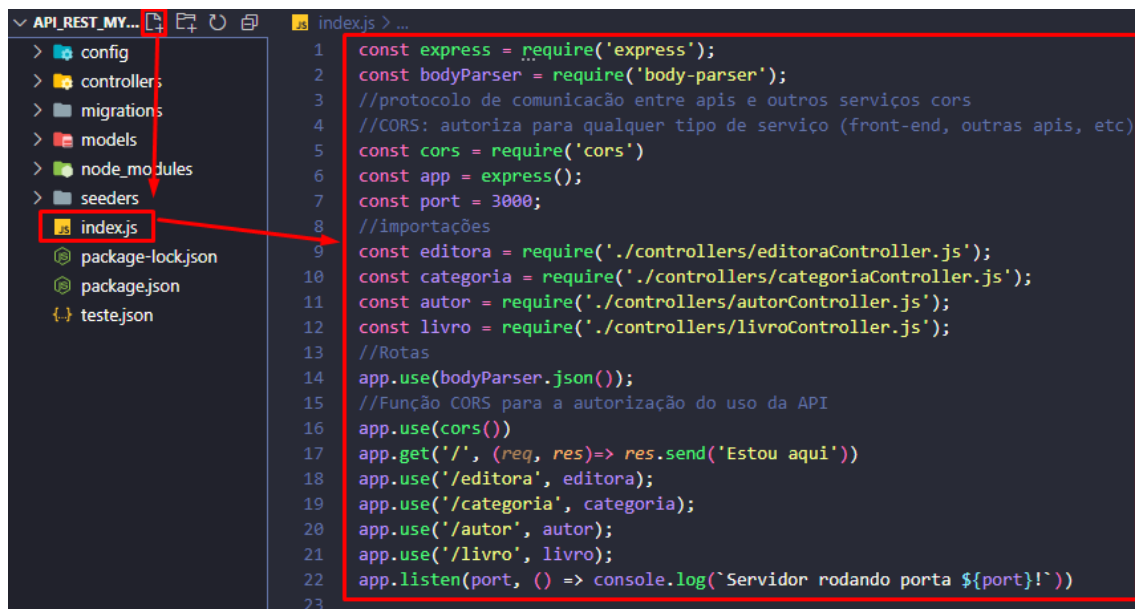
O CORS **Cross-origin Resource Sharing** (Compartilhamento de recursos com origens diferentes) é um mecanismo utilizado pelos navegadores para compartilhar recursos entre diferentes origens. O CORS é uma especificação do W3C e faz uso de headers do HTTP para informar aos navegadores se determinado recurso pode ser ou não acessado.

Vamos instalar o CORS no nosso projeto com o comando abaixo:

```
npm install cors
```

```
PS C:\Users\mario\Desktop\API_Rest_Mysql> npm i cors
added 2 packages, and audited 157 packages in 5s
14 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
PS C:\Users\mario\Desktop\API_Rest_Mysql> █
```

Vamos criar o arquivo **index.js** na pasta **raiz** do nosso **projeto**. Digite o código abaixo:



```
1 const express = require('express');
2 const bodyParser = require('body-parser');
3 //protocolo de comunicação entre apis e outros serviços cors
4 //CORS: autoriza para qualquer tipo de serviço (front-end, outras apis, etc)
5 const cors = require('cors')
6 const app = express();
7 const port = 3000;
8 //importações
9 const editora = require('./controllers/editoraController.js');
10 const categoria = require('./controllers/categoriaController.js');
11 const autor = require('./controllers/autorController.js');
12 const livro = require('./controllers/livroController.js');
13 //Rotas
14 app.use(bodyParser.json());
15 //Função CORS para a autorização do uso da API
16 app.use(cors())
17 app.get('/', (req, res) => res.send('Estou aqui'))
18 app.use('/editora', editora);
19 app.use('/categoria', categoria);
20 app.use('/autor', autor);
21 app.use('/livro', livro);
22 app.listen(port, () => console.log(`Servidor rodando porta ${port}!`))
23
```

## Código do index.js

```
const express = require('express');
const bodyParser = require('body-parser');
//protocolo de comunicação entre apis e outros serviços cors
//CORS: autoriza para qualquer tipo de serviço (front-end, outras apis,
etc)
const cors = require('cors')
const app = express();
const port = 3000;
//importações
const editora = require('./controllers/editoraController.js');
const categoria = require('./controllers/categoriaController.js');
const autor = require('./controllers/autorController.js');
const livro = require('./controllers/livroController.js');
//Rotas
app.use(bodyParser.json());
//Função CORS para a autorização do uso da API
app.use(cors())
app.get('/', (req, res)=> res.send('Estou aqui'))
app.use('/editora', editora);
app.use('/categoria', categoria);
app.use('/autor', autor);
app.use('/livro', livro);
app.listen(port, () => console.log(`Servidor rodando porta ${port}!`))
```

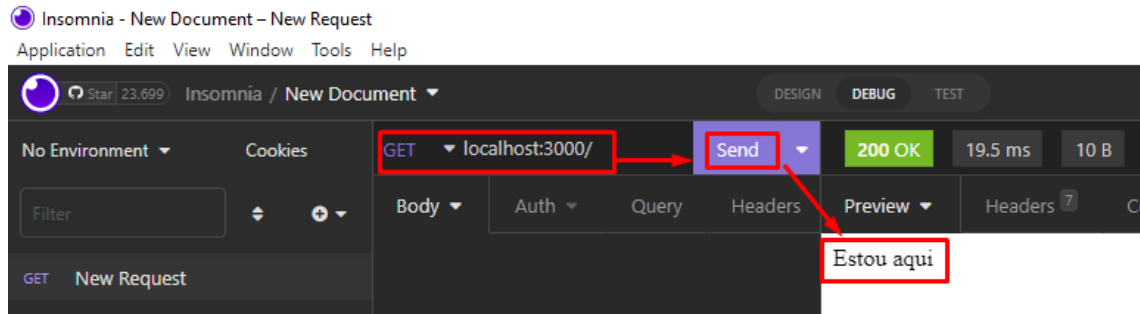
## Iniciando nossa API com nodemon

Abra o arquivo **package.json** e acrescente o código **"dev": "npx nodemon index.js"** para automatizar a inicialização o servidor.

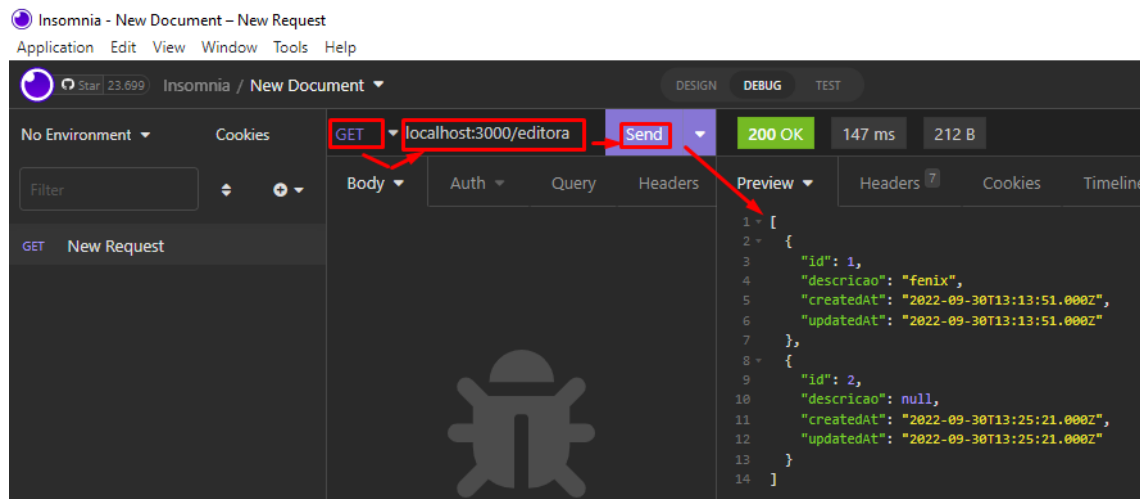
Abra o terminal e digite **npm run dev**

## Realizando os testes das rotas no Insomnia

Você pode testar sem **configurar** o insomnia digitando a **url** mais o método e o tipo. Veja:

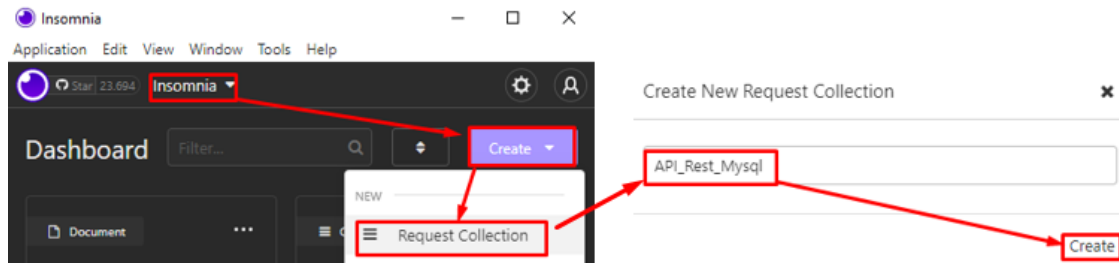


Agora você só precisa ficar **trocando** a **URL**. Veja



## Configurando o Insomnia(Editora)

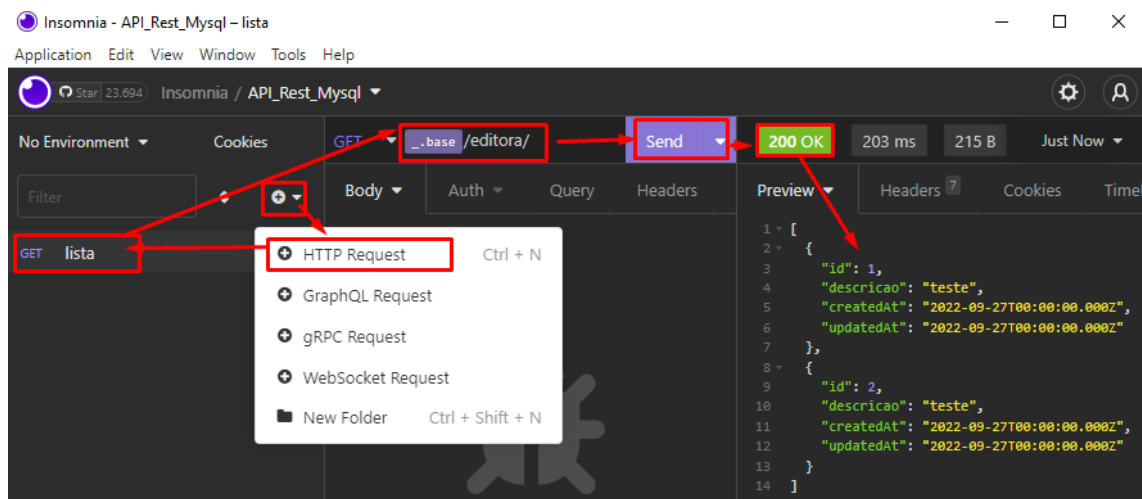
Abra o **insomnia** e faça as **configurações** abaixo para **editora**, ou teste manualmente. Crie a **API\_Rest\_Mysql**



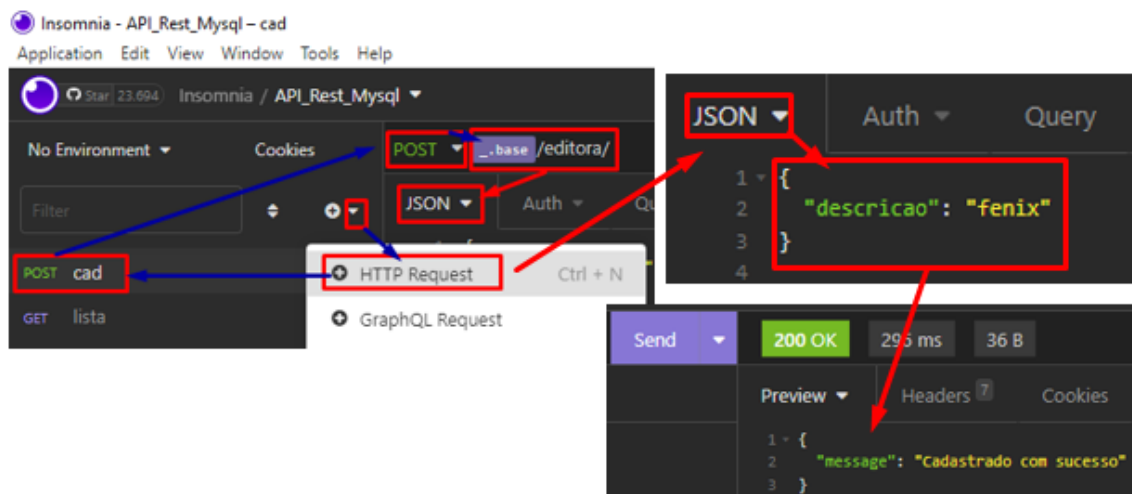
Determine a variável **base** para trazer o **localhost:300**



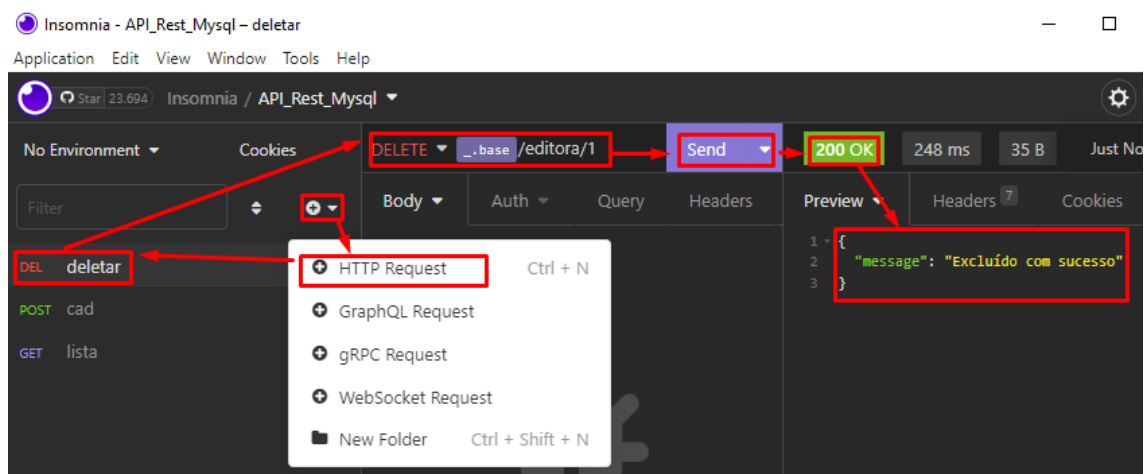
Crie a **URL** para **listar** usando o método **get**. **\_.base/editora**



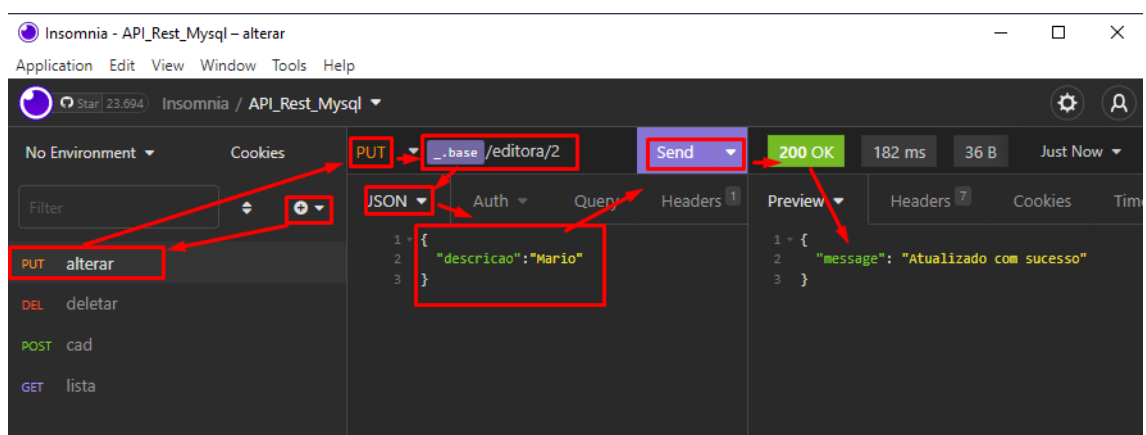
Crie a URL para **cadastro** usando o método **post** e o corpo **json**. `_.base/editora/`



Crie a URL para **deletar** usando o método **delete**. `_.base/editora/id`



Crie a URL para **alterar** usando o método **put**. `_.base/editora/id`



## Referencias



Aula #05 - Criando um CRUD completo usando Sequelize + Node.js + MySQL

<https://youtu.be/-FGCjfR9HFk>



API com NodeJs e Sequelize ORM

<https://youtu.be/MWG8HLwQdZs>