



Desafio V - Sistema de Cálculo Percurso (ModalBike) – Autenticação

Neste desafio, foi implementado uma atualização do sistema ModalBike, incluindo um sistema de autenticação segura. Toda implementação e instalação das bibliotecas devem ser realizadas de acordo com os procedimentos dos desafios anteriores.

Através da utilização de tokens de segurança funções de hashing, podemos utilizar os dados de uma forma que os dados sejam guardados de uma forma mais segura. Para esta aplicação, utilizamos uma biblioteca chamada bcrypt, para que possamos gerar senhas do modelo hash. Para isso devemos instalar todas as bibliotecas/pacotes através dos seguintes comandos:

Primeiramente, para que possamos estar com o sistema devidamente configurado, devemos instalar o gerenciador de pacotes para o Node.JS chamado NPM

```
NPM init -y
```

Desta forma, o projeto criará um arquivo chamado **package.json**

Após instalação do pacote NPM, devemos instalar um framework chamado **express**. Esta ferramenta é usada para os desenvolvedores criarem um ambiente de execução para criação rotas e subir o servidor local. Para que possamos instalar esta biblioteca, devemos digitar o seguinte comando:

```
NPM install express
```

Além da instalação do Express, devemos instalar outra biblioteca chamada **body-parser**. Esta biblioteca tem a função de converter os dados que virão do corpo das requisições do tipo json.

```
NPM install body-parser
```

Outra biblioteca muito importante para o desenvolvimento chama-se nodemon, capaz de sempre atualizar e recarregar o servidor, toda vez que um arquivo for atualizado. Devemos digitar o seguinte comando:

```
NPM install -save-start nodemon
```

Para instalação do banco de dados MySQL, devemos instalar através do seguinte código:



NPM install mysql2

Para instalação do *Sequelize* devemos digitar o seguinte comando:

NPM install sequelize sequelize-cli path

Para iniciar as dependências do *Sequelize*, devemos digitar o seguinte comando:

npx sequelize-cli init

Para instalar o *bcrypt*, ferramenta necessária para geração de hashes, devemos instalar conforme código:

npm install bcrypt@ 3.0.8

Outra biblioteca otimizada do *bcrypt* chama-se *bcryptjs* que poderá ser instalado através do seguinte comando:

npm install bcrypt@js

Para que um aplicativo web seja executado em uma origem (domínio), necessitamos utilizar o CORS, sendo uma biblioteca que usa cabeçalhos adicionais HTTP. Para isso, devemos instalar de acordo com o seguinte comando:

npm install cors

No quesito de autenticação, o JWT (Jason Web Tokens) é um padrão de troca de informações definidas. É possível armazenar de forma segura e compacta objetos JSON. Para que possamos utilizar esta aplicação, devemos instalá-lo. Para isto, basta instalar pelo seguinte comando:

npm install jasonwebtoken@8.5.1

Para finalizarmos as instalações, devemos instalar o Swagger. O Swagger é uma aplicação open source, para que auxilia desenvolvedores nos processos de definir, criar, documentar e consumir APIs REST. Em suma, o *Swagger* visa padronizar este tipo de integração, descrevendo os recursos que uma API deve possuir, como *endpoints*, dados recebidos, dados retornados, códigos HTTP e métodos de autenticação, entre outros. Para a correta instalação, basta digitar o seguinte comando:



npm install swagger-ui-express

Observação: Para que a aplicação execute de forma correta, devemos executar o seguinte comando dentro do terminal.

npm run dev

Aplicação Swagger

Para toda a aplicação deste processo podemos observar o funcionamento através da plataforma Swagger. Abaixo podemos observar que o sistema necessita de uma autorização via token para o devido acesso.

Podemos observar as listas de rotas da nossa aplicação. Níveis, Colaboradores, Login, Lançamentos.

Figura 1: Rotas

Figura 2: Exemplo Rota Colaboradores

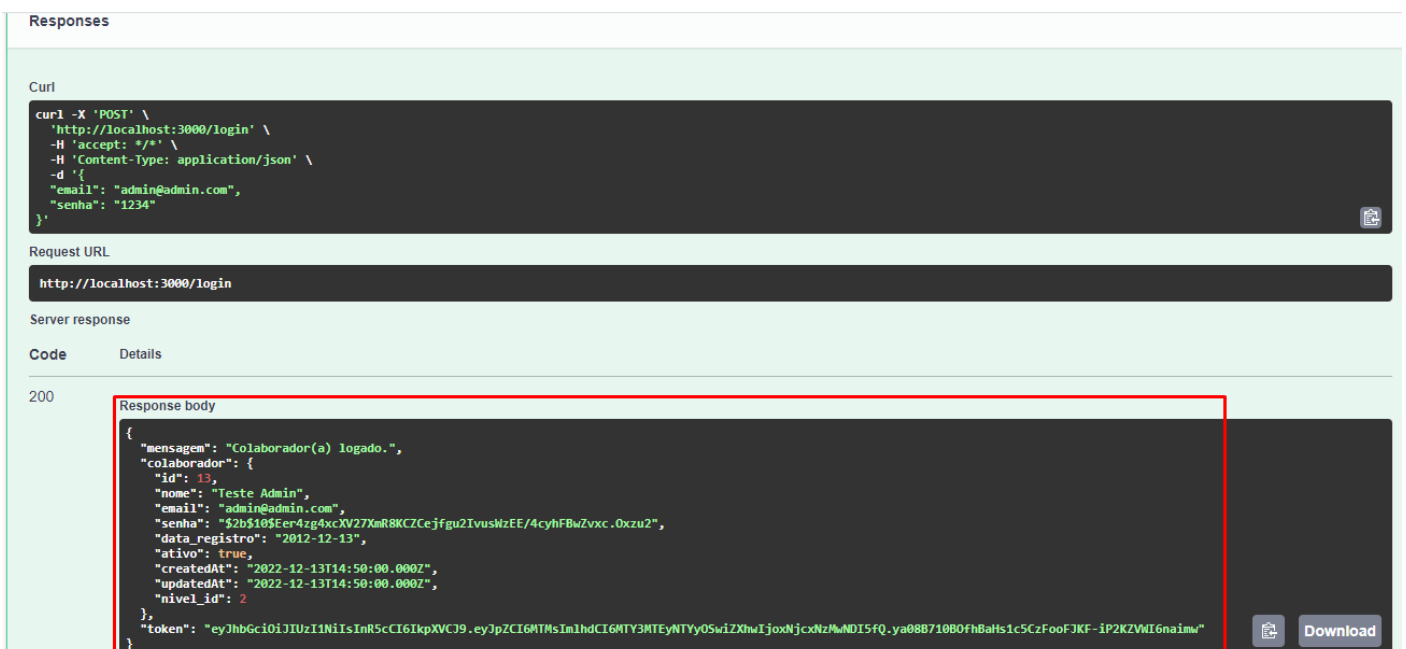
Lembrando para que o sistema consiga realizar as devidas buscas, o usuário deverá logar como (Colaborador ou Administrador) de acordo com o nível especificado. Caso não tenha permissão, a seguinte mensagem aparece ao usuário "Falta o token":



The screenshot shows a web browser's developer console with the 'Responses' tab selected. The response is a 400 status code with the message 'Error: Bad Request'. The response body is a JSON object: `{ "mensagem": "Falta o token" }`. The response headers include: `connection: keep-alive`, `content-length: 29`, `content-type: application/json; charset=utf-8`, `date: Thu, 15 Dec 2022 17:31:16 GMT`, `etag: W/"1d-kc/VU0x+51gfgKsrIz8i+dCD9cI"`, `keep-alive: timeout=5`, and `x-powered-by: Express`.

Figura 3: Mensagem Falta Token de Acesso

Ao executar como administrador (e-mail e senha), a senha é armazenada de forma correta (através de uma outra senha hash) e o acesso do token é enviado, de acordo com a imagem abaixo:



The screenshot shows a web browser's developer console with the 'Responses' tab selected. The response is a 200 status code. The response body is a JSON object: `{ "mensagem": "Colaborador(a) logado.", "colaborador": { "id": 13, "nome": "Teste Admin", "email": "admin@admin.com", "senha": "$2b$10$Eer4zg4xcXV27XmR8KCZCeJfgu2IvuszEE/4cyhFBwZvxc.0xcu2", "data_registro": "2012-12-13", "ativo": true, "createdAt": "2022-12-13T14:50:00.000Z", "updatedAt": "2022-12-13T14:50:00.000Z", "nivel_id": 2 }, "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IWRXV2VybmVudC5kaWZlIiwiaWF0IjoiMjAyMi12-13T14:50:00.000Z" }`. The response headers include: `connection: keep-alive`, `content-length: 312`, `content-type: application/json; charset=utf-8`, `date: Thu, 15 Dec 2022 17:31:16 GMT`, `etag: W/"1d-kc/VU0x+51gfgKsrIz8i+dCD9cI"`, `keep-alive: timeout=5`, and `x-powered-by: Express`.

Figura 4: Resposta Acesso Administrador



```
"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MTMsIm1hdCI6MTY3MTEyNTYyOSwiZXhwIjoxNjc4NzYwNDI5fQ.ya08B710B0fhBaHs1c5CzFooFJKF-iP2KZWI6naImw"
```

Figura 5: Token Gerado Administrador

Após o acesso via token, as aplicações já estão aptas para uso.

ModalGR Bike 1.0.0 OAS3

Documentação do sistema ModalGRBike

Authorize

Níveis

GET	/niveis	✓	🔒
POST	/niveis	✓	🔒
GET	/niveis/:id	✓	🔒
PUT	/niveis/:id	✓	🔒
DELETE	/niveis/:id	✓	🔒

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:3000/niveis' \
  -H 'accept: */*' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MTMsIm1hdCI6MTY3MTEyNTYyOSwiZXhwIjoxNjc4NzYwNDI5fQ.ya08B710B0fhBaHs1c5CzFooFJKF-iP2KZWI6naImw'
```

Request URL

http://localhost:3000/niveis

Server response

Code Details

200

Response body

```
{
  "id": 1,
  "nivel": "Usuário",
  "createdAt": "2022-12-05T13:01:19.000Z",
  "updatedAt": "2022-12-05T13:01:19.000Z",
},
{
  "id": 2,
  "nivel": "Administrador",
  "createdAt": "2022-12-05T13:01:19.000Z",
  "updatedAt": "2022-12-05T13:01:19.000Z",
}
]
```

Download

Response headers

```
connection: keep-alive
content-length: 218
content-type: application/json; charset=utf-8
date: Thu, 15 Dec 2022 17:42:43 GMT
etag: W/"da-e2cCRlmpuz6HHzqt4RIgUvndyU"
keep-alive: timeout=5
x-powered-by: Express
```

Figura 6: Exemplo resposta níveis no sistema (Acesso somente administrador)



Para a aplicação, os níveis colaboradores também possuem essa funcionalidade de token, fazendo que a aplicação funcione também de maneira mais segura e confiável.

Para a documentação do Swagger, as características dos dados são referenciadas através da tabela Schemas, como podemos ver na imagem abaixo:



Figura 7: Informações do Schemas

Conclusão

Podemos observar no desafio 5 os procedimentos necessários para que todo o sistema consiga, de forma mais segura ser aplicados através de ferramentas capazes de criptografar dados importantes de uma aplicação. Para este quesito utiliza além de senhas hashes, (conjunto de caracteres, capazes de substituir a senha do usuário, para que posteriormente consiga compará-las futuramente para acesso), bem com o uso do token (outro conjunto de caracteres, que em conjunto com a senha hash dão acesso a determinada aplicação).

Toda a aplicação foi baseada no desenvolvimento do sistema de gerenciamento de percurso backend ModalBike.



VERSÕES ANTERIORES

Desafio I - Sistema de Cálculo Percurso (ModalBike) V.1.0



O desafio visa o desenvolvimento de um sistema onde o usuário possa observar a quantidade de quilômetros percorrido durante seu percurso de bicicleta, através do projeto desenvolvido pela empresa ModalGr, para seus colaboradores.

Neste projeto, o usuário consegue observar os seguintes cálculos:

- Soma das distâncias percorridas
- Soma dos tempos utilizados durante os percursos
- Velocidade média dos percursos

O Usuário deve utilizar o sistema, incluindo os dados do percurso





Controle Seu Percurso

Distância Percorrida (KM)

Tempo Percorrido

Enviar Dados

ID	Percurso (KM)	Tempo (h)
----	---------------	-----------

Após a Inclusão dos dados, o sistema automaticamente inclui todos os dados no sistema através de uma tabela e posteriormente realiza o cálculo da somatória da distância percorrida e sua respectiva somatória do tempo gasto. Através desta análise, podemos obter o valor da velocidade média percorrida pelo usuário. Abaixo podemos observar os dados inclusos no sistema.



Controle Seu Percurso

Distância Percorrida (KM)

Tempo Percorrido

Enviar Dados

ID	Percurso (KM)	Tempo (h)
1	10	0.5
2	20	1.5

Como podemos observar na imagem acima, o sistema automatiza os valores inseridos e atualiza uma tabela com os seguintes dados: ID (Identificação do lançamento), Percurso (Medido em KM), Tempo percorrido (Medido em hora – h.)



Desta forma, o usuário poderá saber a somatória do percurso e tempo utilizados.

Distância Percorrida (KM)

20

Tempo Percorrido

1.5

Enviar Dados

ID	Percurso (KM)	Tempo (h)
1	10	0.5
2	20	1.5

Soma KM

30 km

Soma Tempo

2 h.

Velocidade Média

15.00

Além dos dados informados acima, o sistema pode nos fornecer uma somatória da velocidade média empregada durante os percursos.

Para este desenvolvimento o sistema também coleta as informações dos dados no console. Os dados armazenados são: Soma das KM, Soma do tempo gasto, vem como valor da velocidade média. A imagem abaixo ilustra os dados capturados durante a simulação.



The screenshot shows the ModalBike web application interface on the left and a browser's developer console on the right. The interface has a blue header with the ModalBike logo and the title "Controle Seu Percurso". It contains two input fields: "Distância Percorrida (KM)" with the value "20" and "Tempo Percorrido" with the value "1.5". Below these is a blue button labeled "Enviar Dados". Under the button is a table with the following data:

ID	Percurso (KM)	Tempo (h)
1	10	0.5
2	20	1.5

Below the table is a label "Soma KM". The browser console on the right shows the following log entries:

```
Soma da Km: 10 script.js:30
Soma do tempo: 0.5 h(s). script.js:47
20 script.js:54
Soma da Km: 30 script.js:30
Soma do tempo: 2 h(s). script.js:47
15 script.js:54
```

Desafio II - Sistema de Cálculo Percurso (ModalBike) V.2.0

Neste desafio, foi implementado uma atualização do site com as informações básicas do desafio I, além do sistema permitir nos mostrar quanto o usuário economizou com as emissão de CO2 durante o seu percurso, bem como saberá quantas árvores serão necessárias para o consumo desta emissão de CO2 no período de 1 ano.

Para complementação do desafio II, o sistema permite coletar informações de um documento onde consta uma lista de links das fontes deste estudo de emissão de poluentes, de fundamental importância para geração deste cálculo. O sistema além de mostrar os links das fontes pesquisadas para este projeto, o próprio sistema automaticamente verifica se os links estão funcionando perfeitamente, e nos mostra os códigos de status dos links. Somente como uma observação, abaixo podemos observar uma lista de status dos códigos HTTP.

HTTP Status Codes

When a browser request a service from a web service, a response code will be given.
These are the list of HTTP Status code that might be returned

1XX Information		4XX Client (Continue)	
100	Continue	407	Proxy Authentication Required
101	Switching Protocols	408	Request Timeout
102	Processing	409	Conflict
103	Early Hints	410	Gone
2XX Success		411	Length Required
		412	Precondition Failed
200	OK	413	Payload Too Large
201	Created	414	URI Too Large
202	Accepted	415	Unsupported Media Type
203	Non-Authoritative Information	416	Range Not Satisfiable
205	Reset Content	417	Exception Failed
206	Partial Content	418	I'm a teapot
207	Multi-Status (WebDAV)	421	Misdirected Request
208	Already Reported (WebDAV)	422	Unprocessable Entity (WebDAV)
226	IM Used (HTTP Delta Encoding)	423	Locked (WebDAV)
3XX Redirection		424	Failed Dependency (WebDAV)
		425	Too Early
300	Multiple Choices	426	Upgrade Required
301	Moved Permanently	428	Precondition Required
302	Found	429	Too Many Requests
303	See Other	431	Request Header Fields Too Large
304	Not Modified	451	Unavailable for Legal Reasons
305	Use Proxy	499	Client Closed Request
306	Unused	5XX Server Error Responses	
307	Temporary Redirect		
308	Permanent Redirect	500	Internal Server Error
4XX Client Error		501	Not Implemented
		502	Bad Gateway
400	Bad Request	503	Service Unavailable
401	Unauthorized	504	Gateway Timeout
402	Payment Required	505	HTTP Version Not Supported
403	Forbidden	507	Insufficient Storage (WebDAV)
404	Not Found	508	Loop Detected (WebDAV)
405	Method Not Allowed	510	Not Extended
406	Not Acceptable	511	Network Authentication Required
Compiled by Ivan Tay.		599	Network Connect Timeout Error

OBSERVAÇÃO: Para este projeto, além do desenvolvimento web, utilizaremos o desenvolvimento backend através de códigos para via terminal. Desta forma, algumas instruções são passadas para a correta utilização desta ferramenta.

Instalação do Chalk (**VIA TERMINAL**):

O Chalk é uma ferramenta muito útil para o desenvolvimento onde permite a separação dos textos por cores, para uma correta e harmoniosa identificação das informações recebidas. Esta biblioteca está disponível no NPM (Node Package Manager), poderoso gerenciador de pacotes para aplicações. Para a instalação no terminal é bem simples, basta inserir o seguinte comando:



npm install chalk@5.0.1

```
ModalGR@DESKTOP-VHRETIC MINGW64 /c/Rodrigo/nodejs-rodri-go-basso
$ npm install chalk@5.0.1
```

Após a instalação, basta um simples comando, o sistema que ele busca dentro da pasta “arquivos”, os documentos de extensão .md para verificação dos sites. O comando para rodar o desafio II é:

Abaixo podemos ver o resultado do processo de verificação dos links:

```
ModalGR@DESKTOP-VHRETIC MINGW64 /c/Rodrigo/NODEJS-Rodrigo-Basso/estudos/desafio2 (estudos)
$ npm run cli:valida
> 2708-node-lib-md@1.0.0 cli:valida
> node ./src/cli.js ./arquivos/links.md --valida

(node:14640) ExperimentalWarning: The Fetch API is an experimental feature. This feature could change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
lista validada [
  {
    'CETESB - Companhia Ambiental do Estado de São Paulo': 'https://cetesb.sp.gov.br/veicular/',
    status: 200
  },
  {
    'CETESB - Relatório de Emissões Veiculares': 'https://cetesb.sp.gov.br/veicular/wp-content/uploads/sites/6/2020/11/Relatorio-Emissoes-Veiculares-no-Estado-de-Sao-Paulo-2019.pdf',
    status: 200
  },
  {
    'Personal CO2 - Relatório de Ranking de veículos em Emissões': 'https://www.personalco2zero.com/estudo_emissoes_CO2/EstudoEmissoesporKM_revisado14nov.pdf'
  },
  status: 200
]

ModalGR@DESKTOP-VHRETIC MINGW64 /c/Rodrigo/NODEJS-Rodrigo-Basso/estudos/desafio2 (estudos)
$
```

Conforme verificado através da imagem acima, o Status indicando **200** significa que o site está **OK** e **em funcionamento**.

Observação: Para que possamos realizar o teste perfeitamente, devemos acessar a pasta do desafio2

Através deste desafio, podemos aprender os sentes requisitos necessários para este desafio:

- Criar e exportar módulos
- Realizar chamadas HTTP
- Processamento assíncrono

Extra: Atualização do Layout da Página Web



Como informado no início do tópico, a plataforma do site foi atualizada, conforme mostrado nas imagens abaixo:

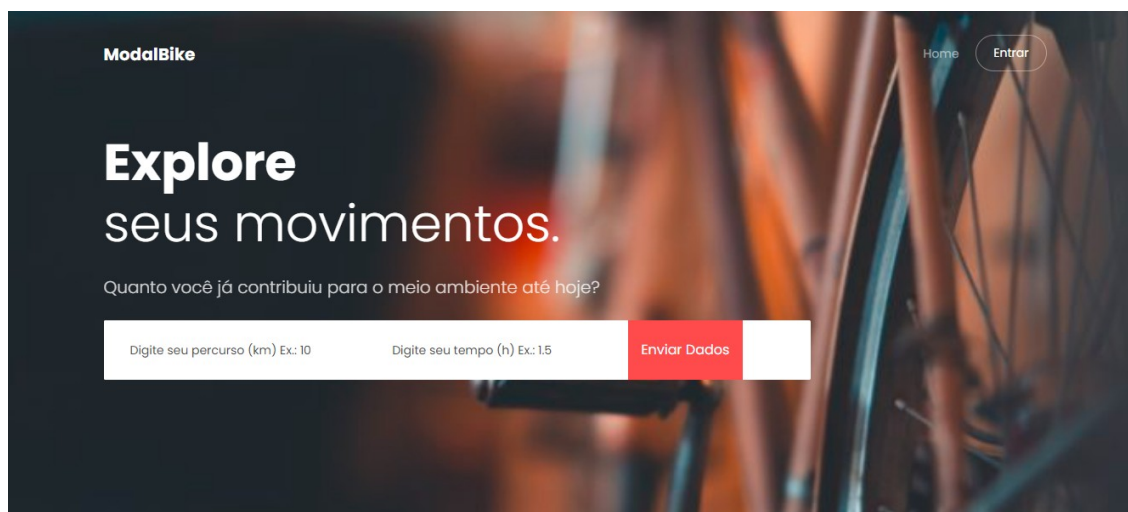


Tabela de Dados

ID	Percurso (KM)	Tempo (h)
1	30	1.3
2	20	1
3	10	0.5

Nesta Imagem abaixo podemos observar os seus resultados. Cada blocos representa um dado importante para análise. Os dados nesta versão de desenvolvimento são: Deslocamento total, Tempo Total, Economia de CO2 para natureza e Árvores compensadas para o consumo de CO2 no período de 1 ano.



Deslocamento
60 KM

Deslocamento total gasto em todos os percursos.



Tempo
2.8 h.

Tempo total gasto em todos os percursos.



Economia
49.2 grCO2

Dados baseados referente a economia, utilizando critério de consumo de um veículo pequeno de passeio, com menor consumo de CO2/KM (0,82grs/KM)/a>



Arvore(s)
0.01
arvore(s).

Árvores compensadas, se basenando no consumo de CO2 por ano

Na tela abaixo temos um texto onde informa os benefícios de se utilizar a bicicleta como meio de transporte para o trabalho. Além desta informação, temos um link que dá acesso as referências deste estudo.

Sobre

Por que Andar de Bike?

Pedalar para o trabalho é uma excelente forma de se exercitar. É um ótimo exercício cardiovascular, além de proporcionar benefícios para a circulação e oxigenação do cérebro- que inclusive é excelente para seu bem estar mental.

Bicicletas não emitem poluentes. É o meio mais limpo de transporte. Quanto mais as pessoas adotarem este estilo de vida, melhor será para nosso meio ambiente.

[Links de Referência](#)

Desafios

Explore seus movimentos





Página de referência com os Links deste estudo.

```
# Lista de Links das fontes
## Lista de Links dos relatórios

Este documento refere-se a uma lista de links para acesso as informações referente ao estudo de emissões de poluentes. Mais Especificamente do gás dióxido de carbono (CO2)

[CETESB - Companhia Ambiental do Estado de São Paulo](https://cetesb.sp.gov.br/veicular/)
[CETESB - Relatório de Emissões Veiculares](https://cetesb.sp.gov.br/veicular/wp-content/uploads/sites/6/2020/11/Relatorio-Emissoes-Veiculares-no-Estado-de-Sao-Paulo-2019.pdf)
[Personal CO2 - Relatório de Ranking de veículos em Emissões](https://www.personalco2zero.com/estudo_emissoes_CO2/EstudoEmissoesporkM_revisado14nov.pdf)
```

Desafio III - Sistema de Cálculo Percurso (ModalBike) V.3.0

Neste desafio, foi implementado uma atualização do site com as informações básicas do desafio I e II, e tem a finalidade de realizar a inclusão de todos os dados no banco de dados. Utilizamos uma API (Application Programming Interface) com modelo de arquitetura REST (Representational State Transfer) e utilizando um banco de dados MongoDB

Primeiramente, para que possamos estar com o sistema devidamente configurado, devemos instalar uma biblioteca chamada nodemon, capaz de sempre atualizar e recarregar a página, toda vez que um arquivo for salvo.

Para a instalação, devemos instalar com o seguinte código em nosso terminal:

```
npm install nodemon@2.0.15 -D
```

Outro framework de extrema importância para gerenciar requisições de diferentes verbos HTTP em diferentes URLs deve ser instalado através do seguinte comando no terminal:

```
npm install express@4.17.3
```


Observação: Para que a aplicação execute de forma correta, devemos executar o seguinte comando dentro do terminal.

npm run dev

```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURAÇÃO  TERMINAL  SQL CONSOLE

> nodemon server.js

[nodemon] 2.0.15
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Servidor escutando em http://localhost:3000

ModalGR@DESKTOP-VHRETIC MINGW64 /c/Rodrigo/NODEJS-Rodrigo-Basso/nodejs-api-mongodb (estudos)
$ npm run dev
```

Para

acesso e interação entre banco de dados e integração devemos utilizar uma biblioteca chamada Mongoose. Para a instalação deve seguir o seguinte código:

npm install mongoose@6.2.6

Após a instalação do mongoose, devemos criar na pasta package.json uma linha de instrução de script para que possamos acionar mais facilmente esta biblioteca


```

1  {
2    "name": "desafio3",
3    "version": "1.0.0",
4    "description": "",
5    "main": "server.js",
6    "dependencies": {
7      "chalk": "^5.0.1",
8      "express": "^4.18.2"
9    },
10   "scripts": {
11     "dev": "nodemon src/index.js",
12     "test": "echo \"Error: no test specified\" && exit 1",
13     "start": "node server.js"
14   },
15   "keywords": [],
16   "author": "",
17   "license": "ISC",
18   "devDependencies": {
19     "nodemon": "^2.0.15"
20   }

```

Posteriormente somente acessar no terminal “npm run dev” que a biblioteca nodemon funcionará automaticamente.

Banco de dados:

A estrutura do banco de dados deste projeto utiliza-se da seguinte informações:

lancamentos
id
km
tempo
DataLancamento
colaborador

colaboradores
id
nome
dataNascimento

Para este Projeto, primeiramente utilizamos um banco de dados não relacional chamado MongoDB. Para o processo de envio de requisições REST, utilizamos um framework Open Source chamado Insomnia. Esta ferramenta de teste, possibilita o envio das requisições para o banco de dados através dos principais métodos GET, POST, PUT e DELETE. Podemos observar nos próximos tópicos, cada detalhe dos métodos empregados.

Servidor local (LocalHost)

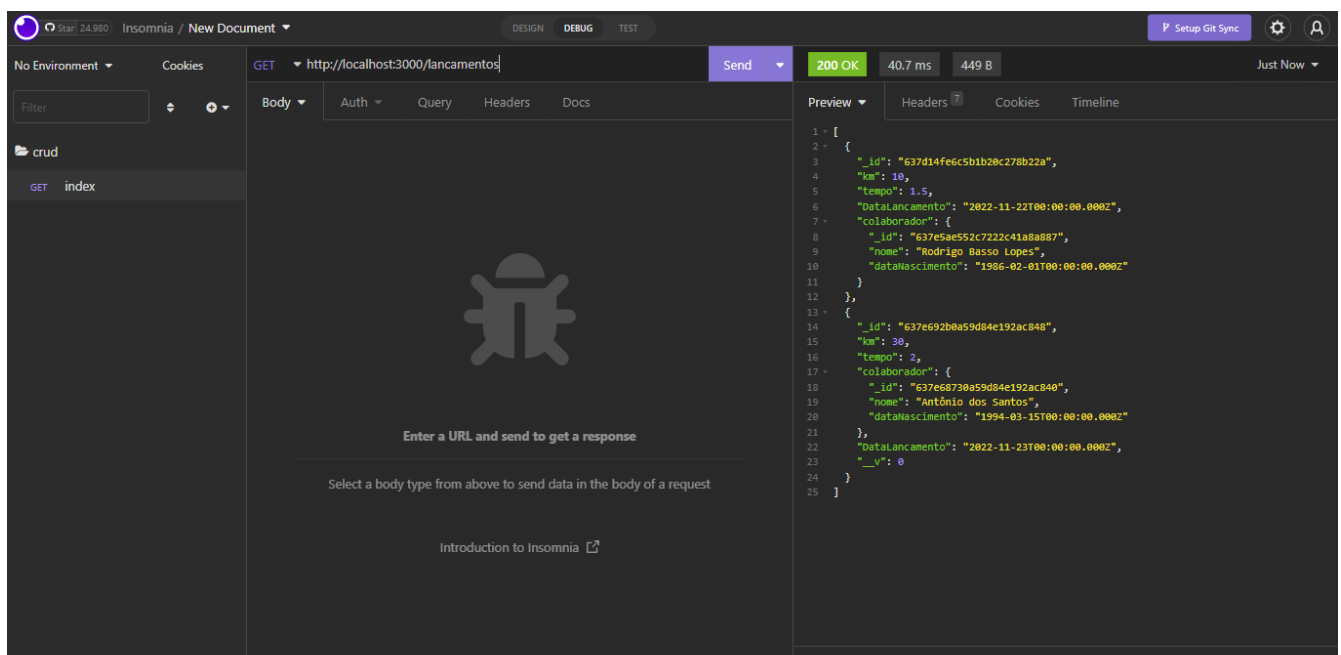
Para este desenvolvimento, foi incluso toda a plataforma do site ModalBike no servidor local (localhost), através de uma definição de porta número 3000.



Mé

Est
da
lan

5



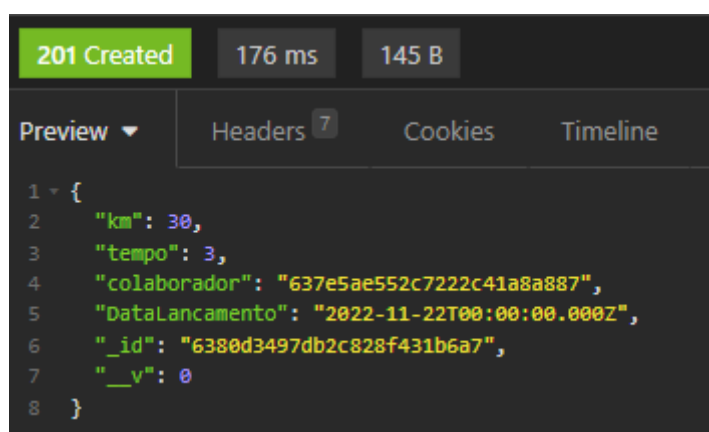
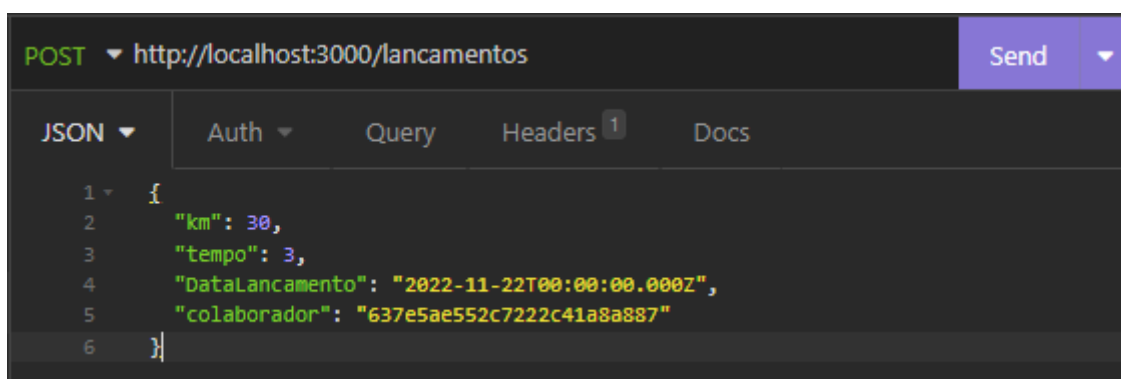
Vale notar que os lançamentos estão relacionados com os dados do colaborador.





Método POST (Registrar)

Neste método, podemos incluir dados de tanto colaborador, quanto os dados dos lançamentos.



Dados Castrados com sucesso



```
200 OK 30.3 ms 676 B
Preview Headers Cookies Timeline
5
6 "tempo": 1.5,
7 "dataLancamento": "2022-11-22T00:00:00.000Z",
8 "colaborador": {
9   "_id": "637e5ae552c7222c41a8a887",
10  "nome": "Rodrigo Basso Lopes",
11  "dataNascimento": "1986-02-01T00:00:00.000Z"
12 },
13 {
14   "_id": "637e692b0a59d84e192ac848",
15   "km": 30,
16   "tempo": 2,
17   "colaborador": {
18     "_id": "637e68730a59d84e192ac840",
19     "nome": "Antônio dos Santos",
20     "dataNascimento": "1994-03-15T00:00:00.000Z"
21   },
22   "dataLancamento": "2022-11-23T00:00:00.000Z",
23   "__v": 0
24 },
25 {
26   "_id": "6380d3497db2c828f431b6a7",
27   "km": 30,
28   "tempo": 3,
29   "colaborador": {
30     "_id": "637e5ae552c7222c41a8a887",
31     "nome": "Rodrigo Basso Lopes",
32     "dataNascimento": "1986-02-01T00:00:00.000Z"
33   },
34   "dataLancamento": "2022-11-22T00:00:00.000Z",
35   "__v": 0
36 }
37 ]
```

Método PUT (Atualizar)

Neste método, o sistema atualiza as informações de acordo com o id de lançamento. Caso queira realizar uma atualização deste lançamento, devemos somente colocar o endereço do lançamento correto e realizar a alteração necessária (Ex.: [http://localhost:3000/\(número de id\)](http://localhost:3000/(número de id))). Esta atualização serve tanto para os dados lançamentos quanto para os dados do colaborador.

```
Star 24,980 Insomnia / New Document DESIGN DEBUG TEST P Setup Git Sync
No Environment Cookies PUT http://localhost:3000/lançamentos/6380d3497db2c828f431b6a7 Send 200 OK 30.3 ms 676 B 3 Minutes Ago
Filter crud PUT index
JSON Auth Query Headers Docs Preview Headers Cookies Timeline
1 {
2   "km": 10,
3 }
4
5 "tempo": 1.5,
6 "dataLancamento": "2022-11-22T00:00:00.000Z",
7 "colaborador": {
8   "_id": "637e5ae552c7222c41a8a887",
9   "nome": "Rodrigo Basso Lopes",
10  "dataNascimento": "1986-02-01T00:00:00.000Z"
11 },
12 },
13 {
14   "_id": "637e692b0a59d84e192ac848",
15   "km": 30,
16   "tempo": 2,
17   "colaborador": {
18     "_id": "637e68730a59d84e192ac840",
19     "nome": "Antônio dos Santos",
20     "dataNascimento": "1994-03-15T00:00:00.000Z"
21   },
22   "dataLancamento": "2022-11-23T00:00:00.000Z",
23   "__v": 0
24 },
25 {
26   "_id": "6380d3497db2c828f431b6a7",
27   "km": 30,
28   "tempo": 3,
29   "colaborador": {
30     "_id": "637e5ae552c7222c41a8a887",
31     "nome": "Rodrigo Basso Lopes",
32     "dataNascimento": "1986-02-01T00:00:00.000Z"
33   },
34   "dataLancamento": "2022-11-22T00:00:00.000Z",
35   "__v": 0
36 }
37 ]
Reformat JSON
```



Método DELETE (Apagar dados)

Da mesma forma que o procedimento de atualizar (PUT), o processo de apagar o registro deve ser feito do mesmo procedimento, informar o id do lançamento correspondente e acionar a função deletar.

```
DELETE http://localhost:3000/lançamentos/6380d3497db2c828f431b6a7 Send 200 OK 32.8 ms 182 B 3 Minutes Ago
```

```
JSON Auth Query Headers 1 Docs
```

```
1 ...
```

```
Preview Headers 7 Cookies Timeline
```

```
1 {
2   "_id": "6380d3497db2c828f431b6a7",
3   "km": 10,
4   "tempo": 3,
5   "colaborador": {
6     "_id": "637e5ae552c7222c41a8a887",
7     "nome": "Rodrigo Basso Lopes"
8   },
9   "DataLancamento": "2022-11-22T00:00:00.000Z",
10  "__v": 0
11 }
```

```
DELETE http://localhost:3000/lançamentos/6380d3497db2c828f431b6a7 Send 200 OK 25.1 ms 46 B Just Now
```

```
JSON Auth Query Headers 1 Docs
```

```
1 ...
```

```
Preview Headers 7 Cookies Timeline
```

```
1 {
2   "message": "Lançamento removido com sucesso"
3 }
```

Lista de Colaboradores

Abaixo podemos verificar a listagem de colaboradores que foram cadastrados no sistema através do endereço: <http://localhost:3000/colaboradores>

```
GET http://localhost:3000/colaboradores Send 200 OK 28.6 ms 217 B Just Now
```

```
JSON Auth Query Headers 1 Docs
```

```
1 ...
```

```
Preview Headers 7 Cookies Timeline
```

```
1 [
2   {
3     "_id": "637e5ae552c7222c41a8a887",
4     "nome": "Rodrigo Basso Lopes",
5     "dataNascimento": "1986-02-01T00:00:00.000Z"
6   },
7   {
8     "_id": "637e68730a59d84e192ac840",
9     "nome": "Antônio dos Santos",
10    "dataNascimento": "1994-03-15T00:00:00.000Z"
11  }
12 ]
```



Extra: Atualizando o site ModalBike

No site ModalBike, tivemos uma atualização quanto a tabela de dados. Foi inserido botões dinâmicos em JavaScript, onde na própria tabela, o usuário conseguirá atualizar ou deletar um lançamento. Abaixo temos mais detalhes desta funcionalidade.

ModalBike

[Home](#)

[Documentação](#)

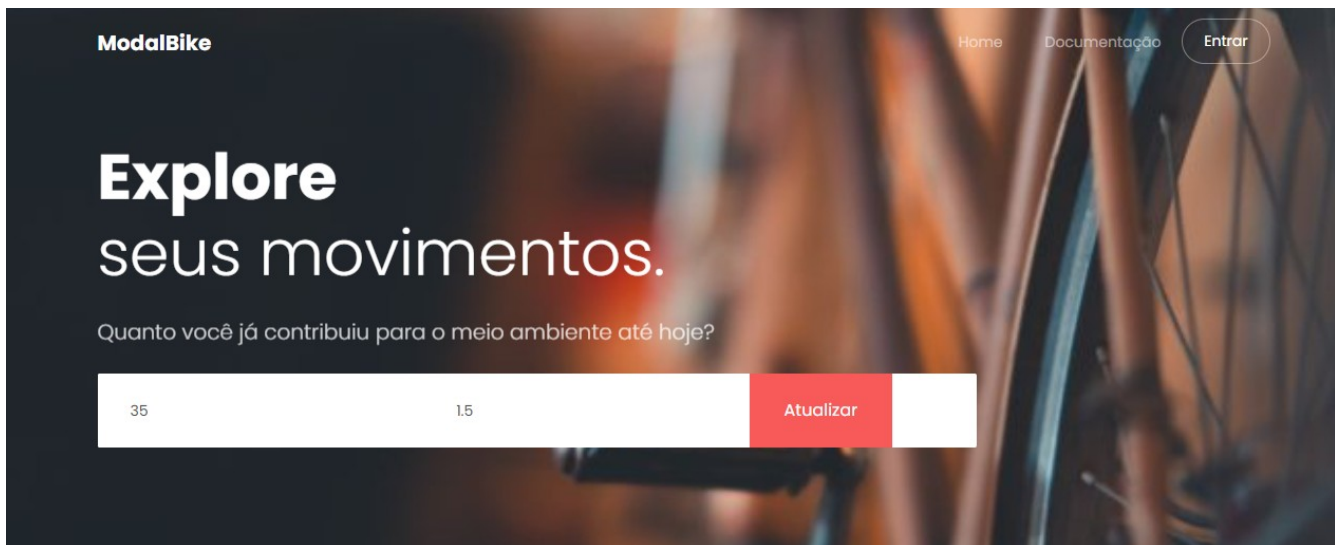
[Entrar](#)



Tabela de Dados

ID	Percurso (KM)	Tempo (h)	Ações
1	20	1	 
2	35	1.5	 

ID	Percurso (KM)	Tempo (h)	Ações
1	20	1	 
2	35	1.5	 



Desafio IV - Sistema de Cálculo Percurso (ModalBike) V.4.0

Neste desafio, foi implementado uma atualização do site com as informações básicas do desafio I e II e IV, e tem a finalidade de realizar a inclusão dos dados através da conexão com banco de dados, utilizando o processo CRUD (Copy, Read, Update e Delete). Utilizamos uma API (Aplication Programming Interface) com um ORM (Object Relational Mapper) ou (Mapeador de Objeto Relacional) chamado Sequelize e utilizando um banco de dados MySQL

Primeiramente, para que possamos estar com o sistema devidamente configurado, devemos instalar o gerenciador de pacotes para o Node.JS chamado NPM

```
NPM init -y
```

Desta forma, o projeto criará um arquivo chamado **package.json**

Após instalação do pacote NPM, devemos instalar um framework chamado **express**. Esta ferramenta é usada para os desenvolvedores criarem um ambiente de execução para criação rotas e subir o servidor local. Para que possamos instalar esta biblioteca, devemos digitar o seguinte comando:

```
NPM install express
```



Além da instalação do Express, devemos instalar outra biblioteca chamada **body-parser**. Esta biblioteca tem a função de converter os dados que virão do corpo das requisições do tipo json.

NPM install body-parser

Outra biblioteca muito importante para o desenvolvimento chama-se nodemon, capaz de sempre atualizar e recarregar o servidor, toda vez que um arquivo for atualizado. Devemos digitar o seguinte comando:

NPM install -save-dev nodemon

Para instalação do banco de dados MySQL, devemos instalar através do seguinte código:

NPM install mysql2

Para instalação do *Sequelize* devemos digitar o seguinte comando:

NPM install sequelize sequelize-cli path

Para iniciar as dependências do *Sequelize*, devemos digitar o seguinte comando:

npx sequelize-cli init

Caso o desenvolvedor possa utilizar o sistema com maior facilidade, o desenvolvedor pode popular o banco de dados através das seguintes dados pré-definidos na pasta *seeders*

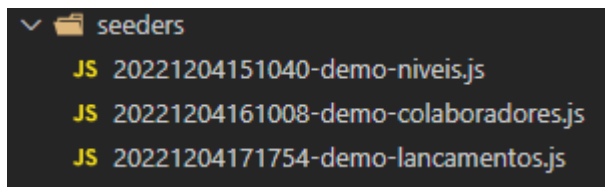


Figura 12: Arquivos Seeders


```
/** @type {import('sequelize-cli').Migration} */
module.exports = {
  async up(queryInterface, Sequelize) {
    await queryInterface.bulkInsert(
      "Niveis",
      [
        {
          nivel: "Usuário",
          createdAt: new Date(),
          updatedAt: new Date(),
        },
        {
          nivel: "Administrador",
          createdAt: new Date(),
          updatedAt: new Date(),
        },
      ],
      {}
    );
  },
};
```

Figura 13: Exemplo Dados: Tabela Níveis

```
/** @type {import('sequelize-cli').Migration} */
module.exports = {
  async up(queryInterface, Sequelize) {
    await queryInterface.bulkInsert(
      "colaboradores",
      [
        {
          nome: "Rodrigo Basso Lopes",
          email: "rodrigo.lopes@modalgr.com.br",
          senha: "1234",
          data_registro: "2012-12-04",
          ativo: true,
          nivel_id: 2,
          createdAt: new Date(),
          updatedAt: new Date(),
        },
      ],
      {}
    );
  },
};
```

Figura 14: Exemplo Dados: Tabela Colaboradores

```
/** @type {import('sequelize-cli').Migration} */
module.exports = {
  async up(queryInterface, Sequelize) {
    await queryInterface.bulkInsert(
      "lancamentos",
      [
        {
          km: 10.5,
          tempo: 0.5,
          colaborador_id: 1,
          createdAt: new Date(),
          updatedAt: new Date(),
        },
      ],
    );
  },
};
```

Figura 15: Exemplo Dados: Tabela Lançamentos

Para que possamos inserir as tabelas no banco de dados e fazer toda a migração das informações dos dados (relacionamentos, chaves estrangeiras, etc), devemos realizar a migração através do seguinte comando:

```
npx sequelize-cli db:migrate
```

Para popularmos o banco de dados de forma automática os arquivos presentes na pasta seed, basta inserir o seguinte comando:

```
npx sequelize-cli db:seed:all
```

Para este projeto, nosso banco de dados utiliza da seguinte forma:

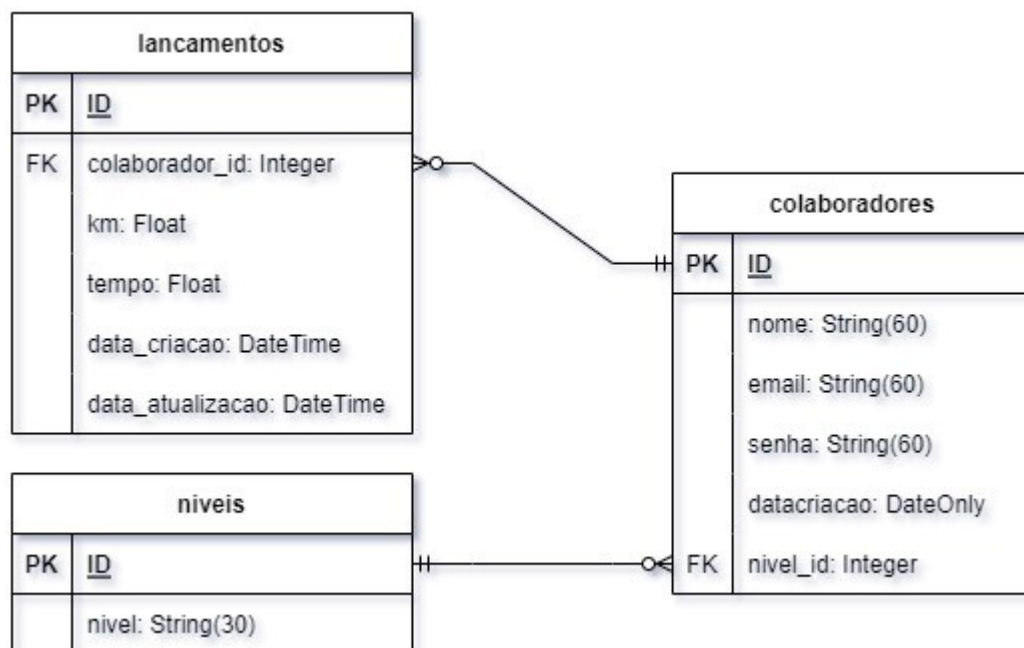


Figura 16: Estrutura Banco de Dados: modalgr_bike

Como podemos observar, todas as tabelas utiliza-se o método CRUD, presente no quesito deste desafio. Toda o processo de CRUD (Copy, Read, Update e Delete) se dá através do ORM *Sequelize*

Para que possamos observar o seu comportamento, utilizamos o framework Open Source chamado *Insomnia*.

Método POST (Criar dados)

Este método consiste em enviar novos dados ao banco de dados. Para este procedimento, devemos somente incluir as informações necessárias de acordo com os dados da tabela correspondente:

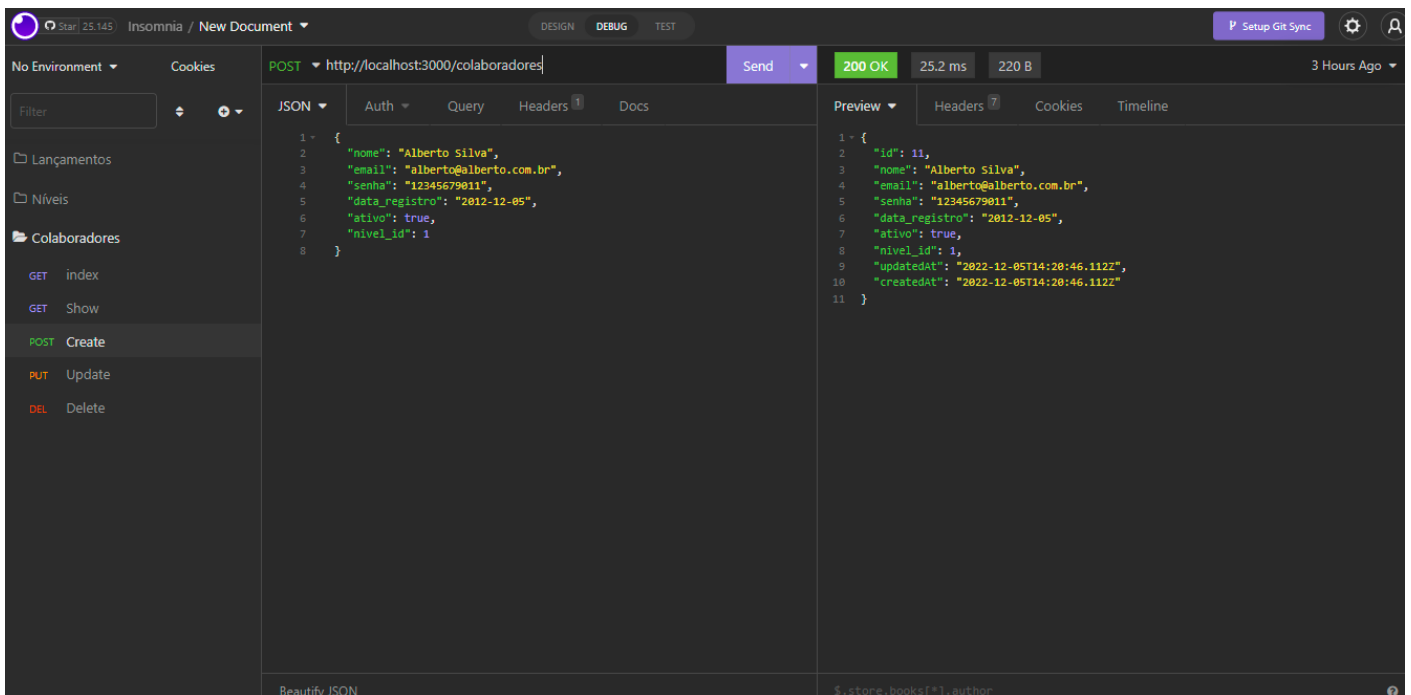


Figura 17: Método Criar

Este método é válido para todas as tabelas (Colaboradores, Níveis e Lançamentos)

Caminhos:

- <http://localhost:3000/lançamentos>
- <http://localhost:3000/níveis>
- <http://localhost:3000/colaboradores>

Método GET (Ler)

Este método visa obter os dados cadastrados no banco de dados. Abaixo podemos ver a tela de verificação, onde os dados cadastrados no banco de dados, estão presente nesta implementação deste método através das páginas lançamentos, níveis e colaboradores.

Caminhos:

- <http://localhost:3000/lançamentos>
- <http://localhost:3000/níveis>
- <http://localhost:3000/colaboradores>

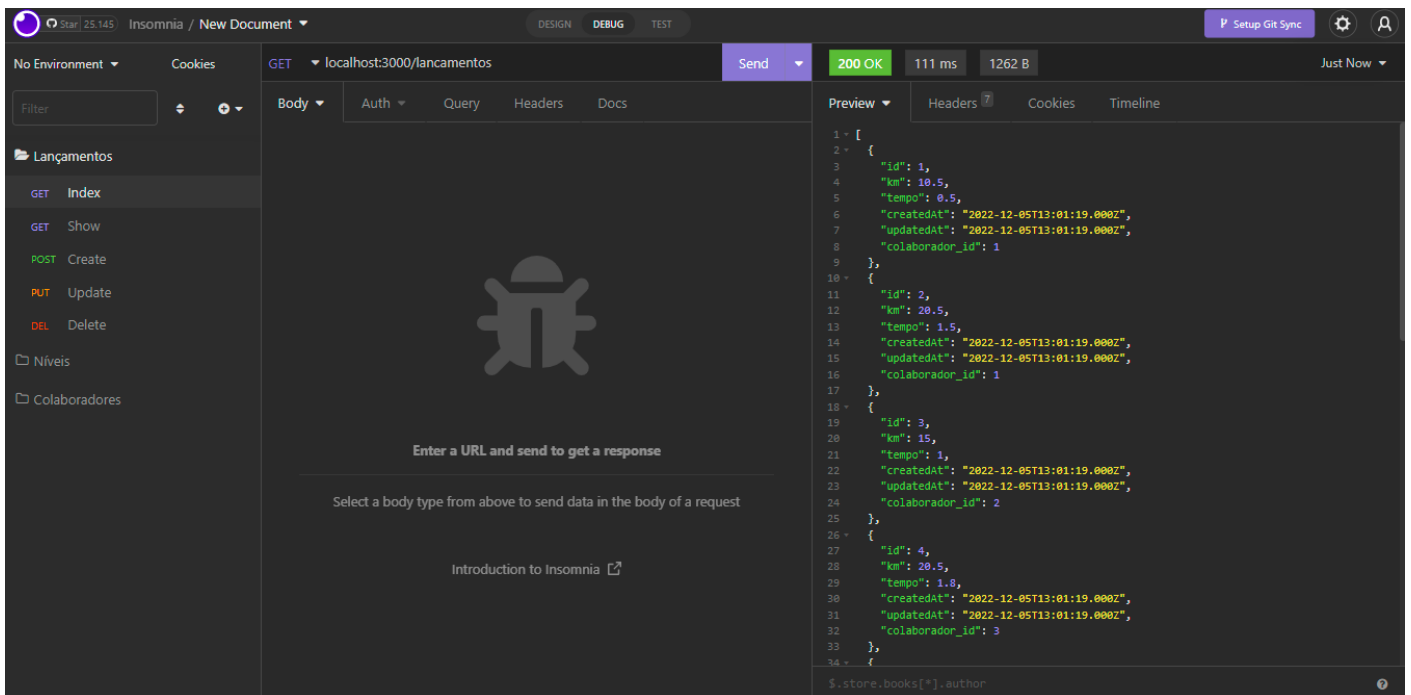


Figura 18: Método Get: Lançamentos

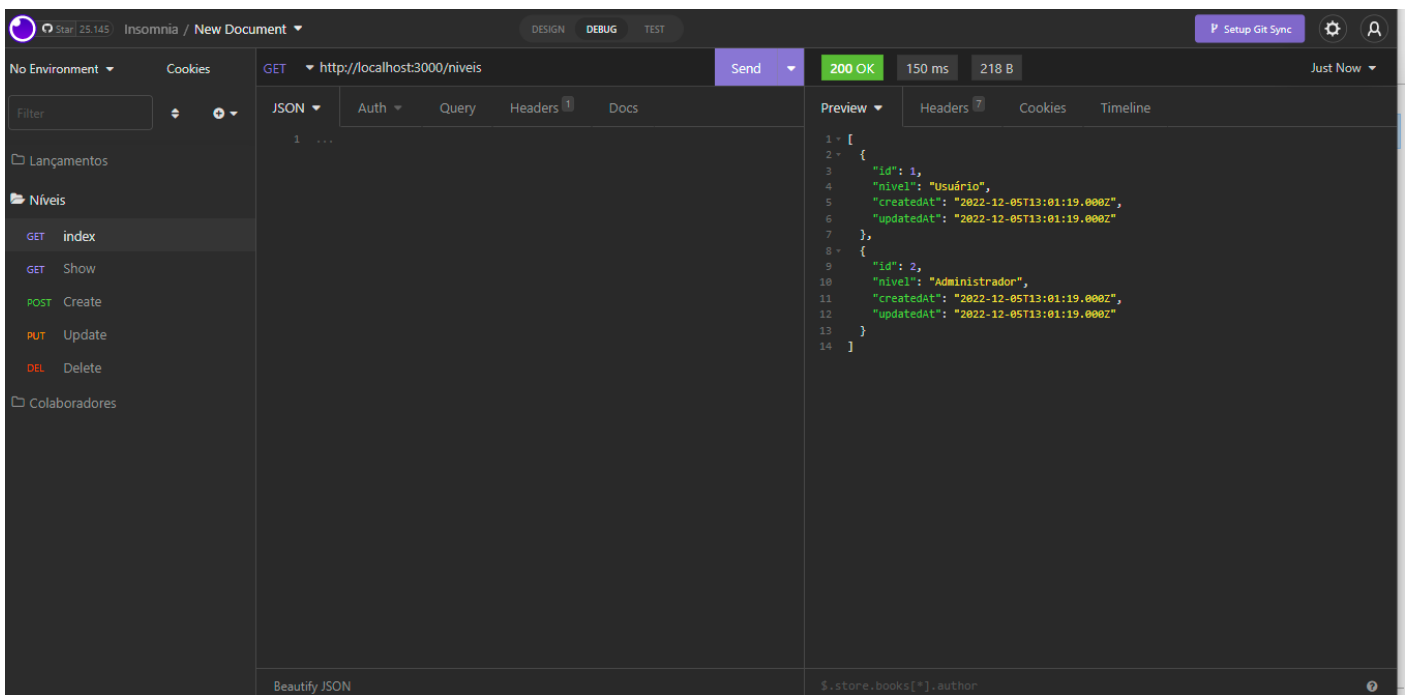


Figura 19: Método Get: Níveis

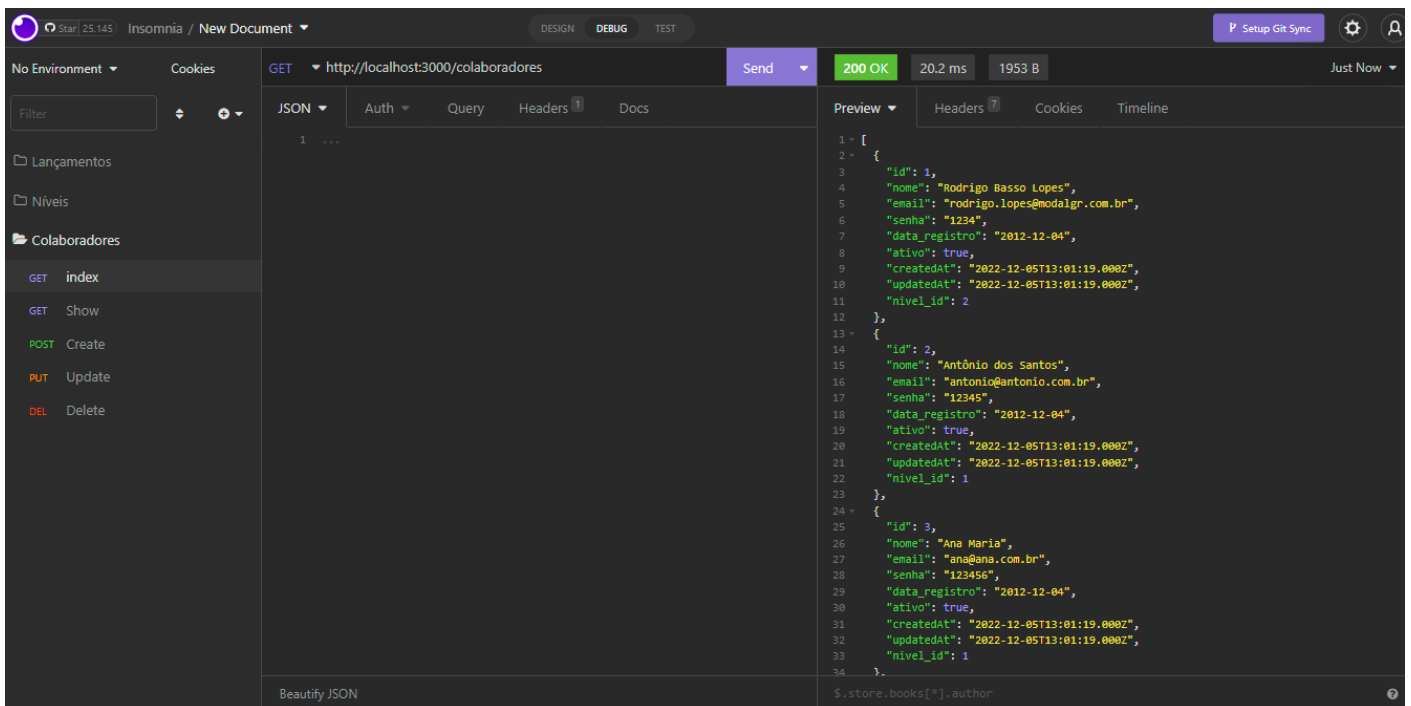


Figura 20: Método Get: Colaboradores

Além da verificação de todos os dados do banco de dados, podemos também verificar a busca dos dados pelo id correspondente (tanto para colaboradores, níveis e lançamentos)

Caminhos:

- [http://localhost:3000/lançamentos/\(número id\)](http://localhost:3000/lançamentos/(número id))
- [http://localhost:3000/níveis/\(número id\)](http://localhost:3000/níveis/(número id))
- [http://localhost:3000/colaboradores/\(número id\)](http://localhost:3000/colaboradores/(número id))

Abaixo um exemplo de como podemos realizar uma busca por um id (exemplo colaborador)

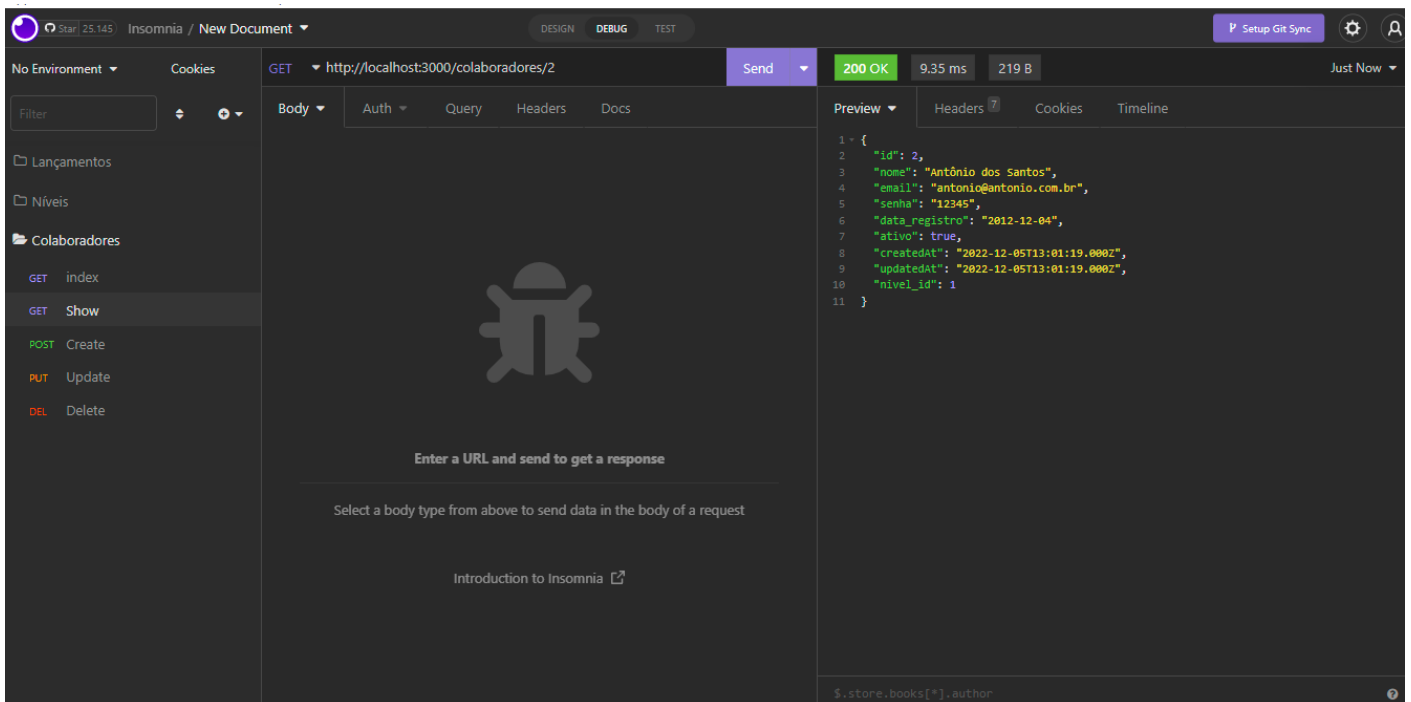


Figura 21: Busca por ID

Neste projeto, além da tabela Colaboradores, também é possível realizar a busca por nível ou também lançamentos.

Método PUT (Atualizar)

Neste método, o sistema atualiza as informações de acordo com o id de lançamento. Caso queira realizar uma atualização deste lançamento, devemos somente colocar o endereço do lançamento correto e realizar a alteração necessária (Ex.: [http://localhost:3000/\(número de id\)](http://localhost:3000/(número de id))). Esta atualização serve para os dados lançamentos, para os dados do colaboradores e níveis.

Caminhos:

- [http://localhost:3000/lançamentos/\(número id\)](http://localhost:3000/lançamentos/(número id))
- [http://localhost:3000/níveis/\(número id\)](http://localhost:3000/níveis/(número id))
- [http://localhost:3000/colaboradores/\(número id\)](http://localhost:3000/colaboradores/(número id))

Método DELETE (Apagar dados)

Da mesma forma que o procedimento de atualizar (PUT), o processo de apagar o registro deve ser feito do mesmo procedimento, informar o id do lançamento correspondente e acionar a função deletar.

Caminhos:

- [http://localhost:3000/lançamentos/\(número id\)](http://localhost:3000/lançamentos/(número id))
- [http://localhost:3000/níveis/\(número id\)](http://localhost:3000/níveis/(número id))



- [http://localhost:3000/colaboradores/\(número id\)](http://localhost:3000/colaboradores/(número id))

Conclusão

Neste projeto, aprendemos a utilização das ferramentas necessárias para o gerenciamento do banco de dados, utilizando os métodos CRUD (Copiar, Ler, Atualizar e Apagar) os dados presentes no bando de dados MySQL.

Desta forma e conforme exemplificado acima, podemos verificar a funcionalidade do sistema de banco de dados e portanto estando de acordo com os quesitos do desafio IV proposto.