**Semana OmniStack – 10° Edição**

*Back-end*

# NodeJS

Usado para a aplicação backend, web e mobile. O JS é a única linguagem de programação que os navegadores conseguem entender. O Node é uma plataforma consegue executar código JS fora do navegador, dessa forma, houve a possibilidade de criar o backend, acessar banco de dados usando JS.

Gerenciador de pacotes (ajuda na instalação de dependências, pacotes para o projeto em questão, e não de forma global, definição da versão): npm – node package manager

O LTS instala na versão LTS (recomendada) – long term support

Instalação: usando o chocolatey: choco install nodejs-lts

# Instalações para o VSCode

Tema: Dracula Theme

Ícones de acordo com o tipo do arquivo: Material Icon Theme

# Definição de Termos

Back-end: parte que o usuário não enxerga.



Webservices: serviços que podem prestar algum tipo de informação (serviço de consulta de CEP, cálculo de frete do correios)

Sempre que for criar um back-end é necessário criar rotas, as rotas são endereços adicionais que aparecem após a URL o endereço principal/home da aplicação. Ex: rota de usuários = **recurso**.

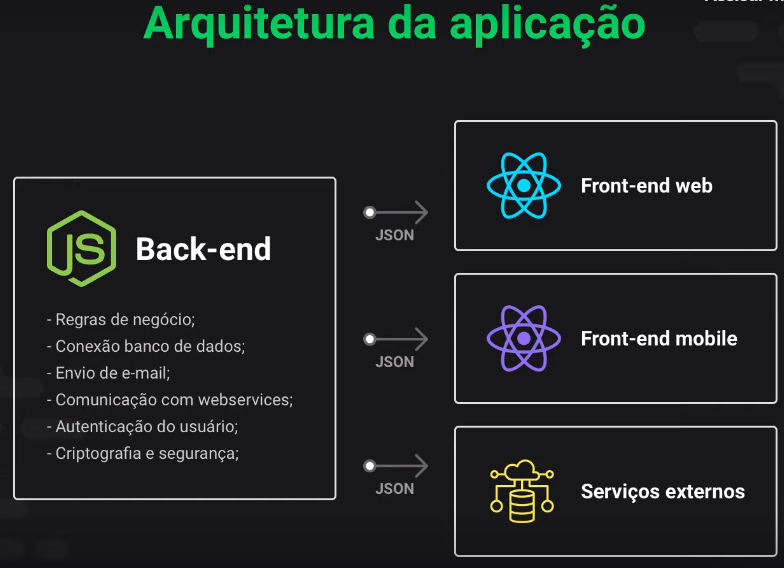
## API Restful

API – Application Programming Interface

Ser Restful significa que a API segue padrões, regras q normalmente são utilizados e conhecidos pelas demais pessoas.

## Front-end

É necessário comunicar o back-end com o front-end (visual), em que pode ser uma aplicação react (web), react native (mobile) e os serviços externos (ex: outra aplicação). Linguagem de comunicação (estrutura de dados – forma de representar dados): JSON – JavaScript Object Notation.



# Criação do Projeto

Comando: yarn init -y (significa que o projeto será JS)

Package.json: armazena informações e características sobre o projeto (nome, versão, arquivo principal, licença – se outras pessoas podem utilizar o código do projeto ou não)

Rodar o projeto: node index.js

## Instalação de Bibliotecas

### Express

Comando: yarn add express

Definição: Ajuda na criação das rotas da aplicação, criação do servidor. É um microframework pois é muito leve.

Quando está em ambiente de desenvolvimento, a aplicação sempre é acessada pelo localhost:porta.

#### Utilização

//importação do módulo express

const express = require('express')

//as variáveis são const pois elas não serão alteradas: valor fixo

//início da criação do servidor

//criação de rota - URL com resposta do servidor

const app = express()

//definição da porta da aplicação

app.listen(3333)

//o primeiro parâmetro é o caminho

//o segundo parâmetro é uma outra função arrow que é composto pela requisição e a resposta

//requisição: tudo que vem do frontend, acesso a rota, requisição ao servidor que pode conter parâmetros

//resposta: o que devolve pro frontend - json (objeto ou vetor JS)

app.get('/', (*request*, *response*) => {

    return response.json({ message: 'Hello OmniStack'})

})

### Nodemon

Fica observando as modificações no código e tira a necessidade de parar a aplicação e startar novamente: faz isso automático.

-D: informa que é uma dependência de desenvolvimento, ou seja, não será utilizada na produção (online)

Comando: yarn add nodemon -D

Execução: yarn nodemon index.js / yarn dev (script configurado)

"scripts": {

    "dev": "nodemon index.js"

  },

### Mongoose - MongoDB

Biblioteca que fornece o acesso do node a base de dados mongo para possibilitar a comunicação da aplicação com o banco mongo.

Comando: yarn add mongoose

#### Utilização

//importação do mongo

const mongoose = require('mongoose')

//conexão ao banco

//a string é a gerada no site do mongo

mongoose.connect('mongodb+srv://carla:estudo@cluster0-3aef4.mongodb.net/week10?retryWrites=true&w=majority')

Lembre-se de substituir os parâmetros: name, password e o nome do banco após /

Passagem de parâmetros para retirar os warnings:

mongoose.connect('mongodb+srv://carla:estudo@cluster0-3aef4.mongodb.net/week10?retryWrites=true&w=majority', {

    useNewUrlParser: true,

    useUnifiedTopology: true

})

#### Cadastro



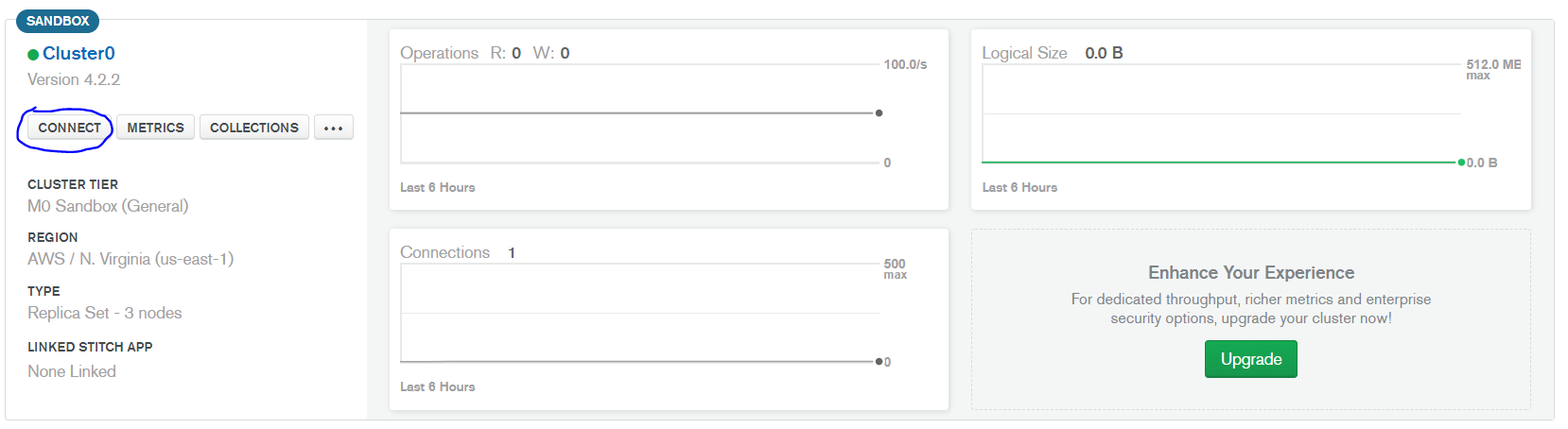
ID: identificador único gerado

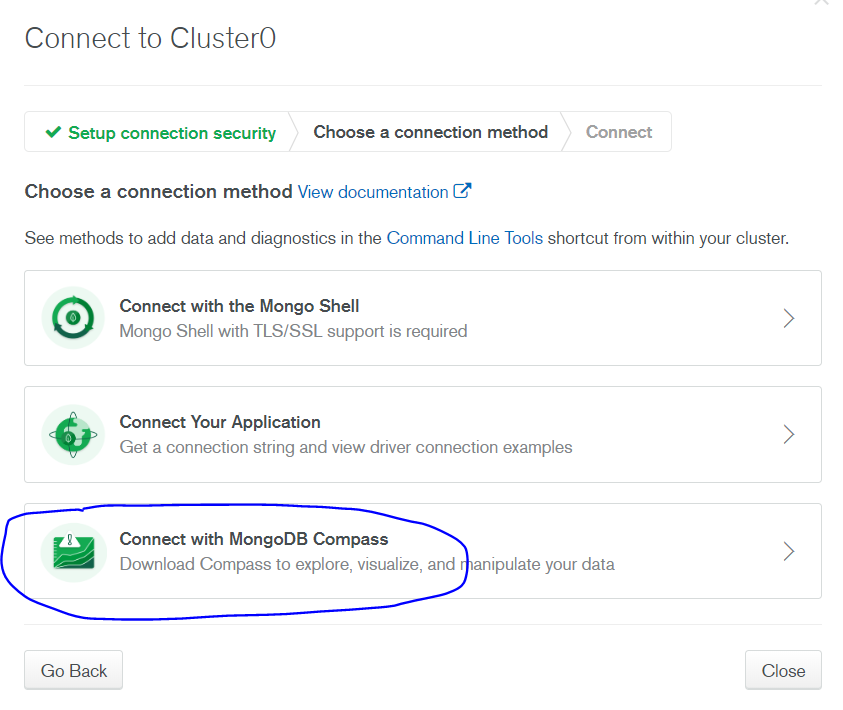
\_\_v: versionamento q o mongo faz de forma automática. Serve para visualizar a quantidade de alterações que foi feita no registro

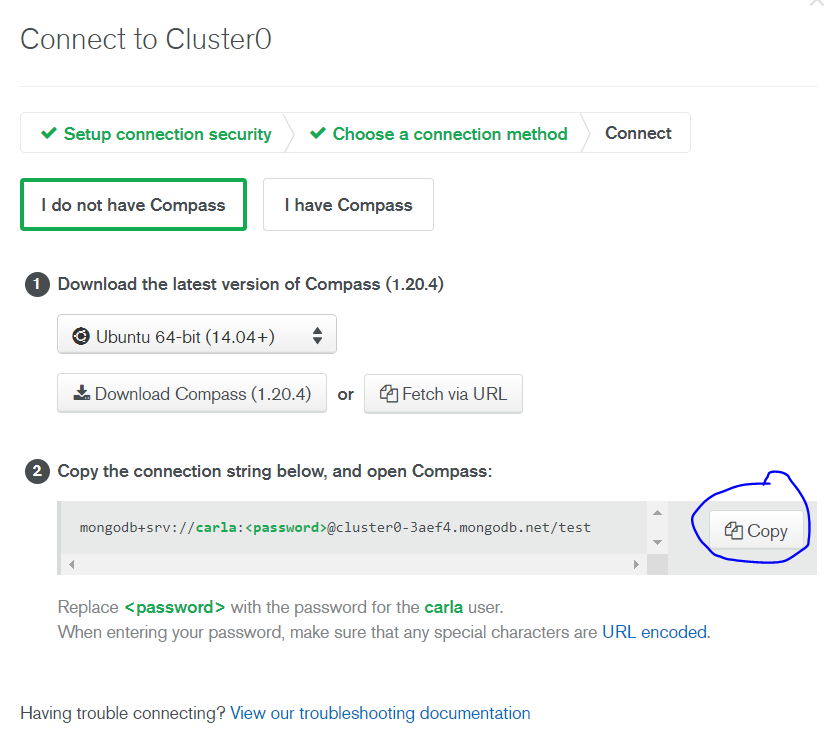
#### Visulização dos Dados

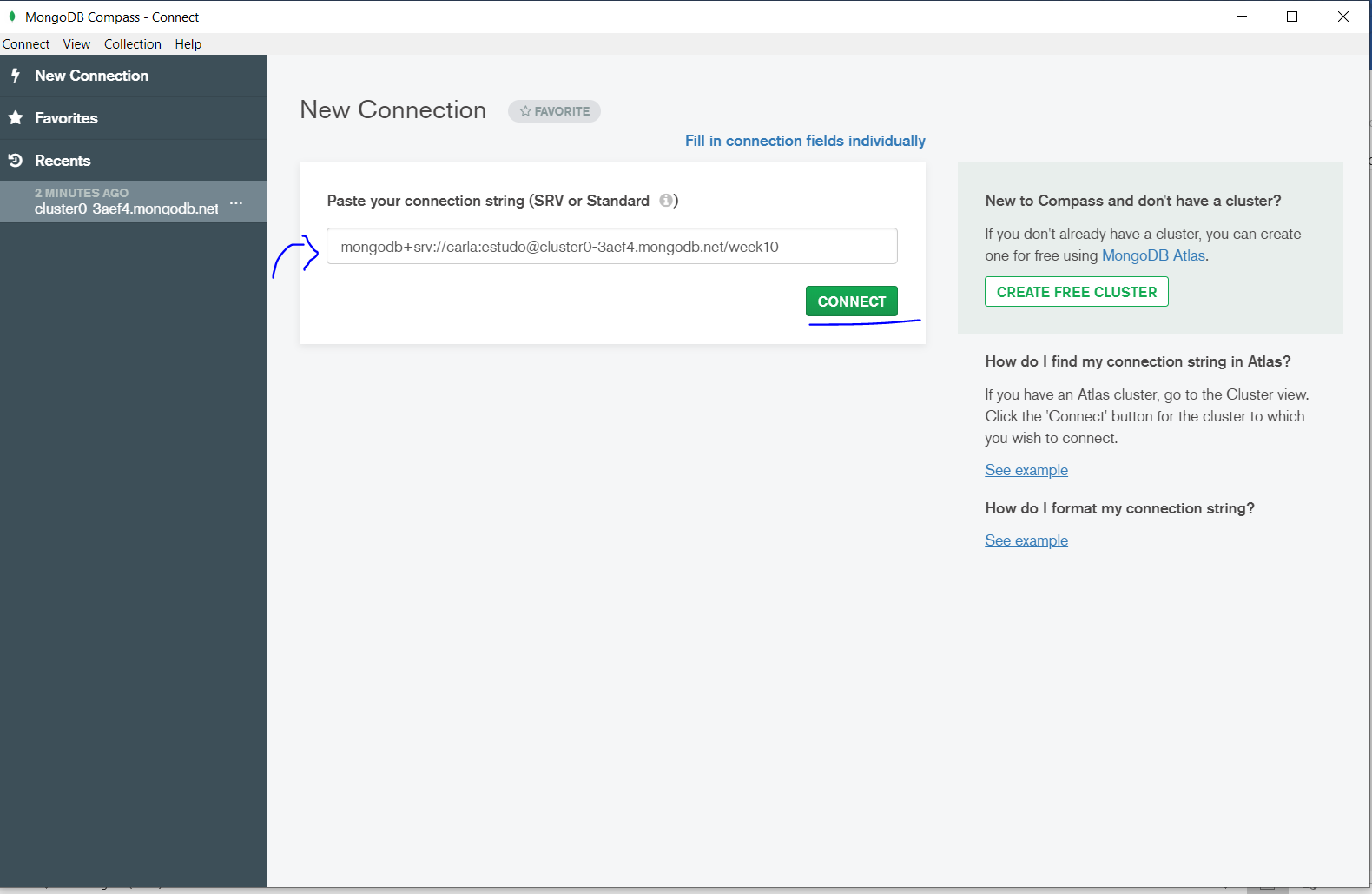
Mongo db compass comunity: cliente para acessar o mongo db <https://www.mongodb.com/download-center/compass>

##### Conexão ao cluster

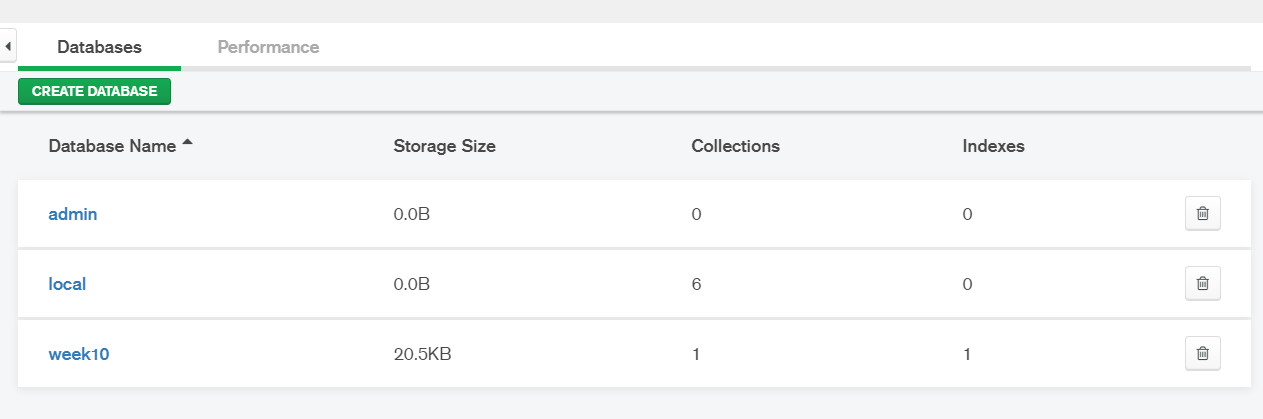




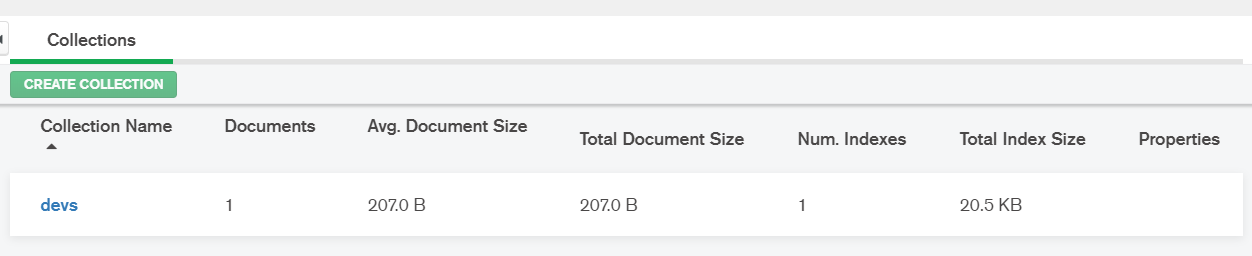




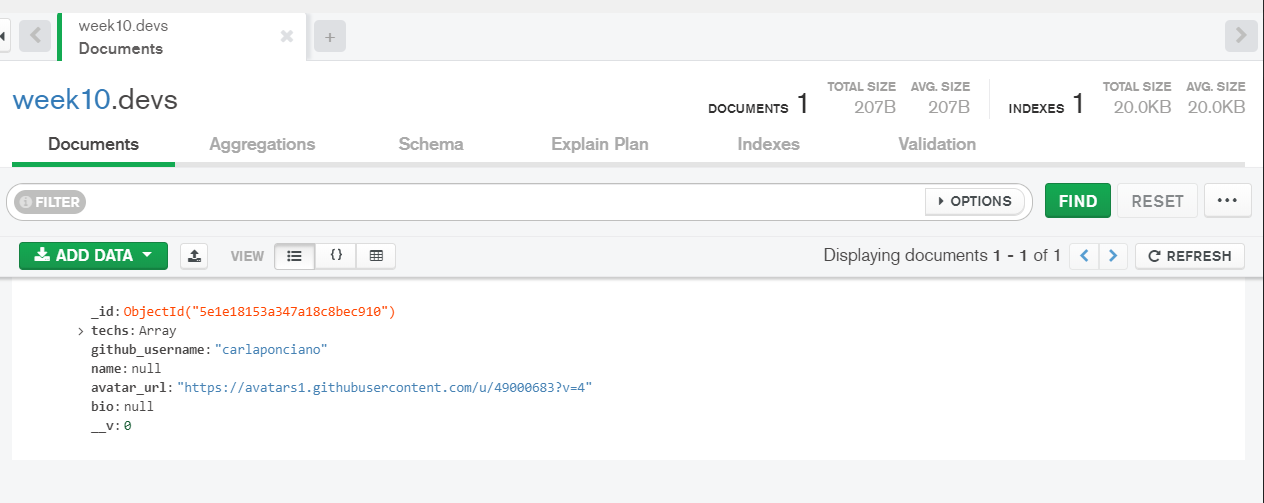
Cole a URL aqui, mudando os parâmetros



Bases de dados



Coleções: “tabelas” do banco de dados



Registros

### Axios

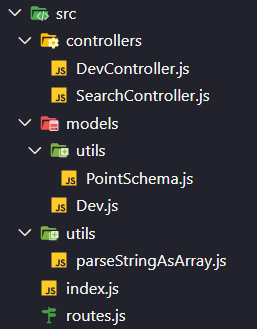
Biblioteca que faz chamadas à outras API.

Comando: yarn add axios.

#### Utilização:

No arquivo de rotas:

## Estrutura do Projeto



### Utils

Utilidades. Pode colocar funções que você usar em mais de um lugar.

//como utiliza em mais de um lugar deve separar em um arquivo diferente

//para evitar o dry: dont repeat yourself (não repita as regras de negócio da sua aplicação)

*module*.*exports* = function parseStringAsArray(*arrayAsString*) {

    return arrayAsString.split(',').map(*tech* => tech.trim())

}

### Src (source)

Onde fica o código da aplicação. Onde fica o index.

Routes.js

Arquivo onde ficam as rotas

//importação do módulo de roteamento do express

//importação específica do Router

const { Router } = require('express')

const DevController = require('./controllers/DevController')

const SearchController = require('./controllers/SearchController')

//em routes você tem acesso a todos os métodos disponíveis (get, post, use..)

const routes = Router()

//as rotas normalmente possuem os nomes no plural

//a flag async informa que a função pode demorar para responder, pois possui uma chamada à outra API (github)

routes.post('/devs', DevController.store)

routes.get('/devs', DevController.index)

//mobile

routes.get('/search', SearchController.index)

//exportação das rotas para que a aplicação tenha acesso

*module*.*exports* = routes

aplicação:

//importação das rotas

const routes = require('./routes')

//cadastro das rotas

app.use(routes)

Models

Representação de entidades (informações que deseja armazenar no banco de dados) – exemplo: dev – desenvolvedor.

//importação do mongoose para informar qual é o formato (estrutura de dados) do Dev na base de dados

const mongoose = require('mongoose')

const PointSchema = require('./utils/PointSchema')

//schema: estruturação de uma entidade no bd

const DevSchema = **new** *mongoose*.*Schema*({

    //formato que o campo será salvo no banco: String, Number, Boolean

    name: *String*,

    github\_username: *String*,

    bio: *String*,

    avatar\_url: *String*,

    //vetor de Strings

    techs: [*String*],

    //latitude e longitude serão armazenadas de forma diferente

    //passagem de configurações a mais

    location: {

        type: PointSchema,

        //em geolocalização é necessários criar um índice para facilitar a busca

        //significa esfera 2d (eixo x e y)

        index: '2dsphere'

    }

})

//param1: nome que o model vai ter, como será salvo no banco de dados

//param2: schema

*module*.*exports* = mongoose.model('Dev', DevSchema)

Utilização na rota:

//importação do módulo de roteamento do express

//importação específica do Router

const { Router } = require('express')

const DevController = require('./controllers/DevController')

//em routes você tem acesso a todos os métodos disponíveis (get, post, use..)

const routes = Router()

//as rotas normalmente possuem os nomes no plural

//a flag async informa que a função pode demorar para responder, pois possui uma chamada à outra API (github)

routes.post('/devs', DevController.store)

routes.get('/devs', DevController.index)

//exportação das rotas para que a aplicação tenha acesso

*module*.*exports* = routes

#### Utils

Utilidades

Ponto de localização:

//criado em arquivo separado para futuro reutilização

const mongoose = require('mongoose')

//código pego da documentação do mongoose

//criação do schema

const PointSchema = **new** *mongoose*.*Schema*({

    type: {

        type: *String*,

        enum: ['Point'],

        required: true,

    },

    coordinates: {

        type: [*Number*],

        required: true,

    }

})

*module*.*exports* = PointSchema

#### Controllers

Um por entidade. Responsável por receber a aquisição, faz o que deve ser feito, e devolver a resposta.

O controller normalmente possui **5 funções**:

* Index: mostrar uma lista
* Show: mostrar um único registro
* Store: criar
* Update: alterar
* Destroy: deletar

DevController:

const Dev = require('../models/Dev')

const axios = require('axios')

const parseStringAsArray = require('../utils/parseStringAsArray')

*module*.*exports* = {

    async index(*request*, *response*) {

        const devs = await Dev.find()

        //é possível utilizar filtros

        //ex: Dev.find({ name: 'Carla' })

        return response.json(devs)

    },

    async store(*request*, *response*) {

        //operador destructing para pegar username

        //a desestruturação pega apenas o desejével

        const { github\_username, techs, latitude, longitude } = request.body

        //evitar a duplicação de inserção de usuários com o msm username

        //findOne: encontrar 1 baseado em

        let dev = await Dev.findOne({ github\_username })

        //se não houver o registro do dev, cria

        if(!dev) {

            //conexão com a API do GitHub para buscar os dados do usuário (nome, avatar)

            //response: resposta que obtém a partir da chamada à API

            //passa a URL da API do GitHub

            //await: aguarda a finalização da conexão à API para dar prosseguimento

            const apiResponse = await axios.get(`https://api.github.com/users/${github\_username}`)

            //se o name não existir ele pega o login, basta colocar o =

            const { name = login, avatar\_url, bio } = apiResponse.data

            //transforma as tecnologias em um array

            //split: separa na virgula e o trim remove os espaços em branco antes e depois

            const techsArray = parseStringAsArray(techs)

            //data: dados da resposta obtida

            //console.log(apiResponse.data)

            const location = {

                type: 'Point',

                //passa primeiro a longitudo e dps a latitude (padrão mongo)

                coordinates: [longitude, latitude]

            }

            //criação do dev

            //salva o retorno da função na variável dev

            dev = await Dev.create({

                github\_username,

                name,

                avatar\_url,

                bio,

                techs: techsArray,

                location

            })

        }

        return response.json(dev)

    }

}

SearchController (mobile):

const Dev = require('../models/Dev')

const parseStringAsArray = require('../utils/parseStringAsArray')

*module*.*exports* = {

    async index(*request*, *response*) {

        //buscar todos os devs em um raio de 10km

        //filtra por tecnologias

        const { latitude, longitude, techs } = request.query

        const techsArray = parseStringAsArray(techs)

        //com filtros

        const devs = await Dev.find({

            //pode passar vários filtros

            //in: operador lógico do mongo

            techs: {

                //que estão em:

                $in: techsArray

            },

            //encontra objetos perto de uma localização

            location: {

                $near: {

                    $geometry: {

                        type: 'Point',

                        coordinates: [longitude, latitude]

                    },

                    //máximo de distância que quer retornar os objetos

                    //10000: 10km

                    $maxDistance: 10000,

                }

            }

        })

    }

}

# Rotas

## Métodos HTTP

* GET: busca de informações (pegar). O método GET não aceita corpo. Ex: listagem de usuários.
* POST: criação de informação. Ex: cadastro de usuários.
* PUT: edição de informações. Ex: editar usuário.
* DELETE: deletar informações. Ex: excluir usuário.

## Tipo de parâmetros

### Query Params

Utilizado no método GET. É enviado através da URL, fica visível na URL.

Utilizado em filtros, ordenação, paginação, por exemplo.

Acesso: request.query

app.get('/users', (*request*, *response*) => {

*console*.log(request.query)

})



### Route Params

Utilizado em PUT e DELETE. Utilizado para identificar um único registro, que são os casos de atualizar e deletar. Não possui um identificador nominal, apenas é passado o valor na rota.

Acesso: request.params

//pode colocar o nome que quiser no identificador

app.put('/users/:id', (*request*, *response*) => {

*console*.log(request.params)

})

### Body

Corpo da requisição. Utilizado em POST e PUT, quando possui uma quantidade maior de informações, por exemplo, no cadastro/atualização de usuários (nome, email, telefone..)

Acesso: request.body

//.use - informa que será utilizado para todas as rotas da aplicação

//informa para o express entender o corpo de requisições no formato JSON

app.use(express.json())

app.post('/users', (*request*, *response*) => {

*console*.log(request.body)

    return response.json({ message: 'Semana OmniStack' })

})

# Conexão com o Banco de Dados - MongoDB

MongoDB: Banco não relacional.

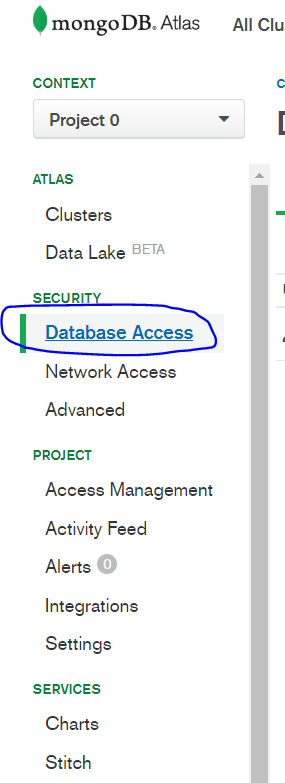
Utilizado em Aplicações que não possuem muitos relacionamentos. Exemplo: em um e-commerce você precisa de relacionamentos, o mongodb não seria aconselhável nesse caso. É possível criar relacionamentos em um banco não relacional, porém essa tarefa seria mais complicada para escalar, por isso existem casos, e cada caso se adequa com a tecnologia a ser utilizada.

É possível de ter o banco de dados hospedado na nuvem, ou seja, não é necessário que seja instalado na máquina.

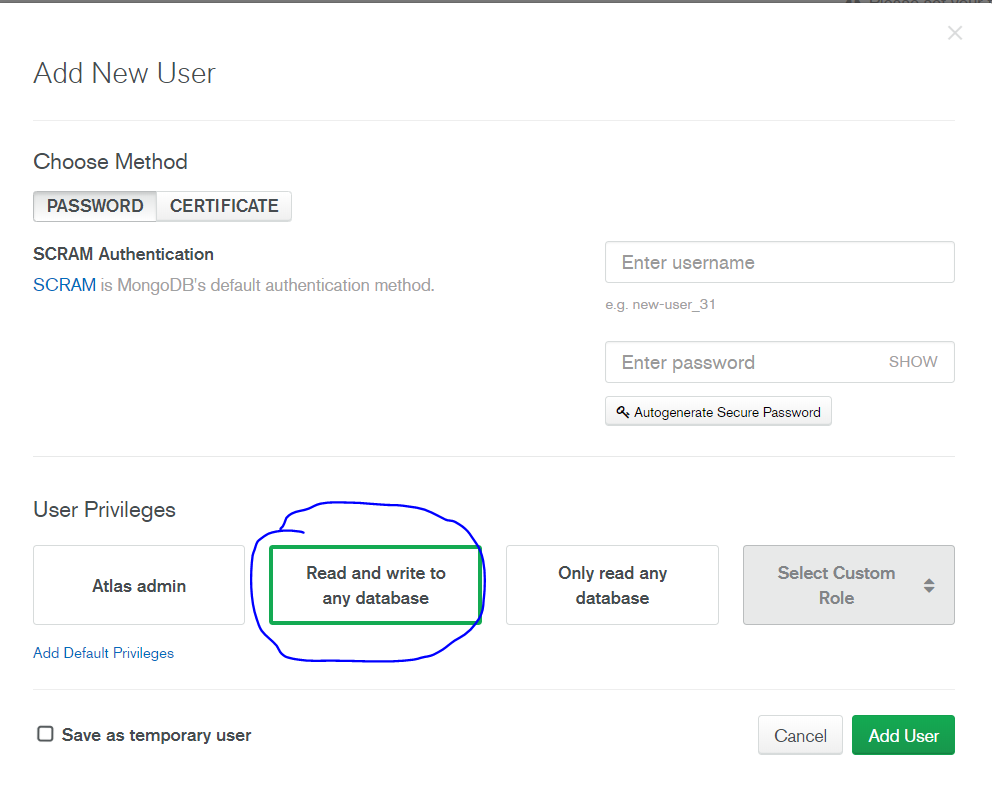
Acesso: mongodb atlas - <https://www.mongodb.com/cloud/atlas>

Cluster: servidor. É possível ter várias bases de dados mongo em um mesmo servidor.

Criação de usuários:





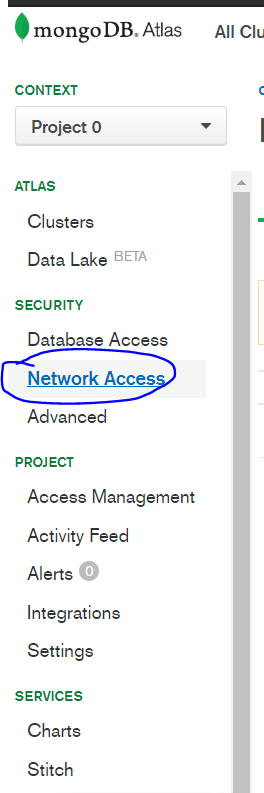


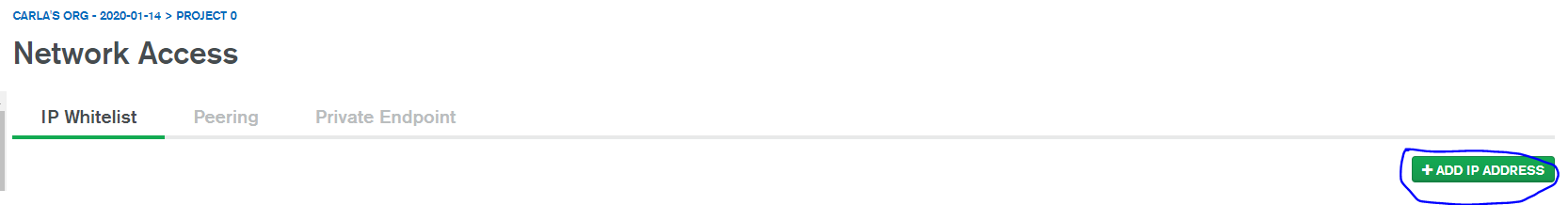
Username: carla

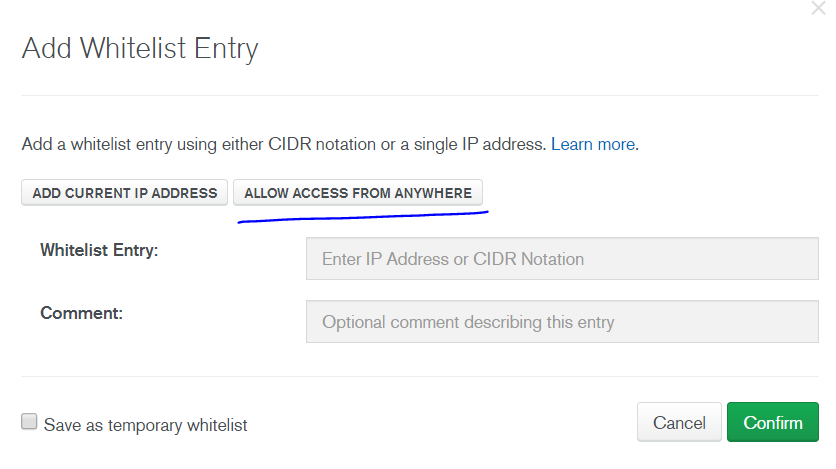
Password: estudo

## NetworkAccess

Liberação dos IPs para acessarem a base de dados.



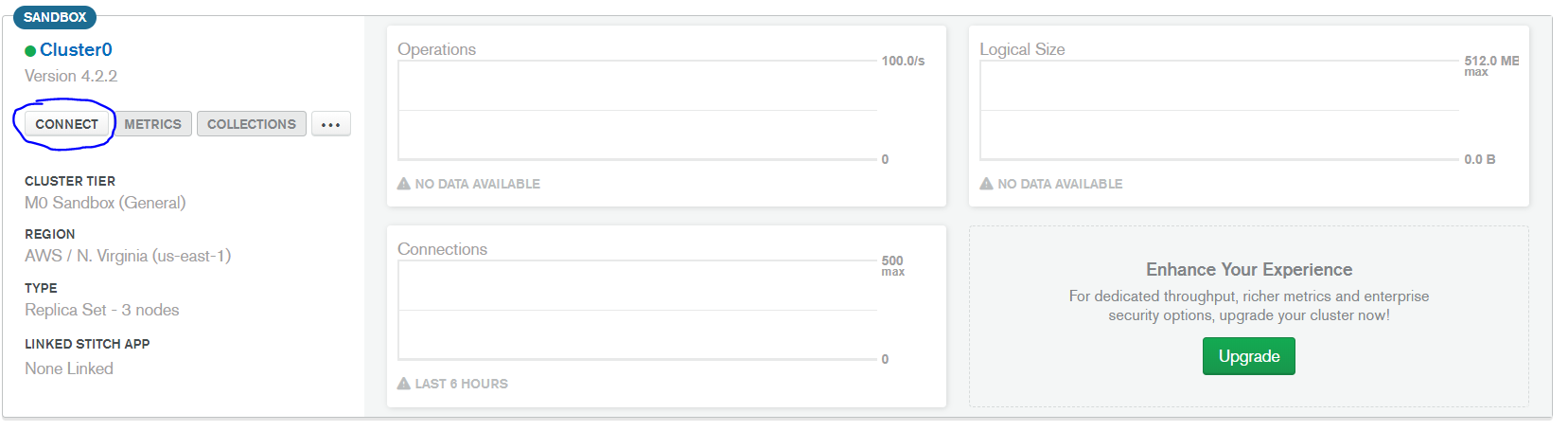


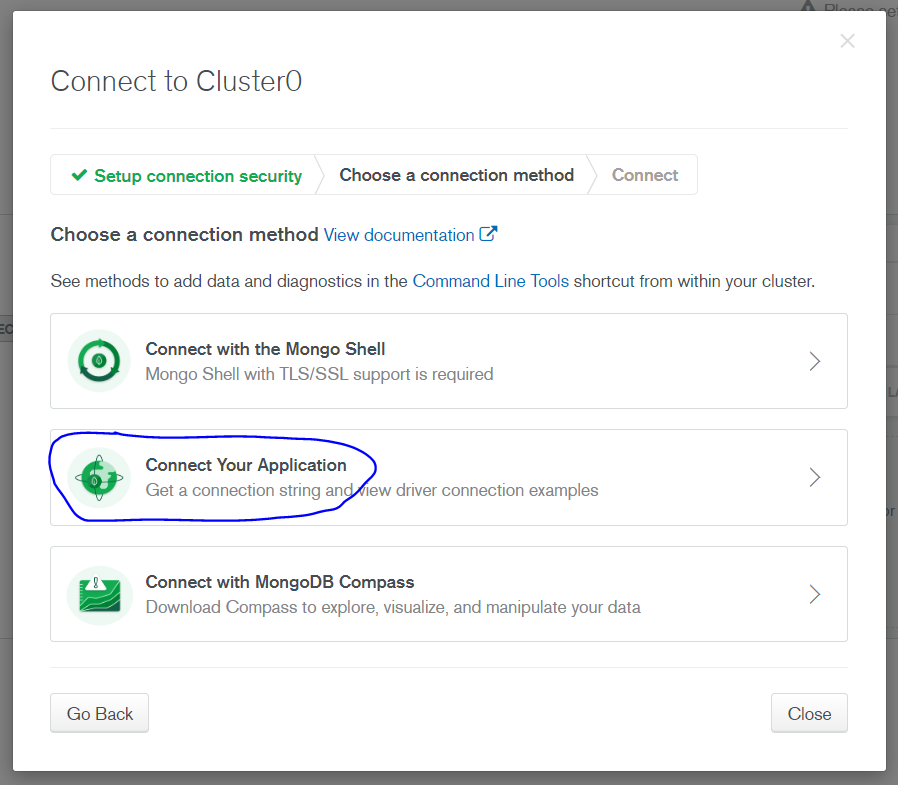


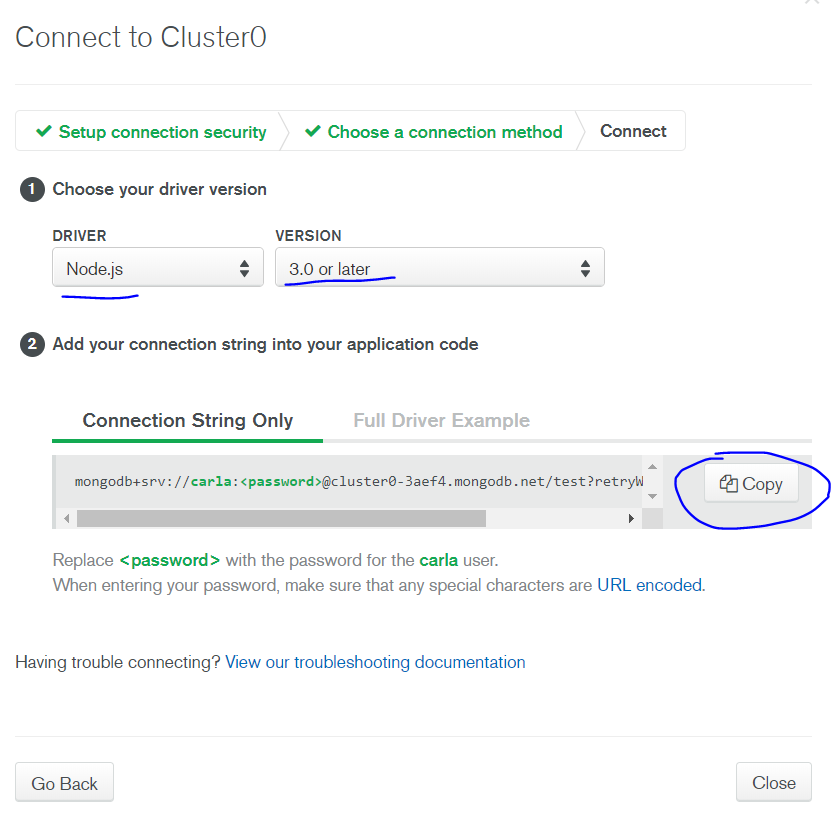
Quando a aplicação está em produção, coloque o IP da aplicação/servidor em *whitelist entry.*

Quando está em desenvolvimento coloque a opção: *allow access from anywhere.*

## Conexão







Copie a string de conexão.

Selecione o driver: Node.js.

Versão: 3.0 or later.

mongoose.connect('mongodb+srv://carla:estudo@cluster0-3aef4.mongodb.net/week10?retryWrites=true&w=majority')

E cole em: mongoose.connect(‘’)

# Curiosidades

O yarn.lock é um cash das dependências, esse arquivo não é manipulado.

## Teste de porta

<http://portquiz.net:27017/>

Verifica se não possui nenhum proxy impedindo a conexão.

27017 é a porta do mongo db

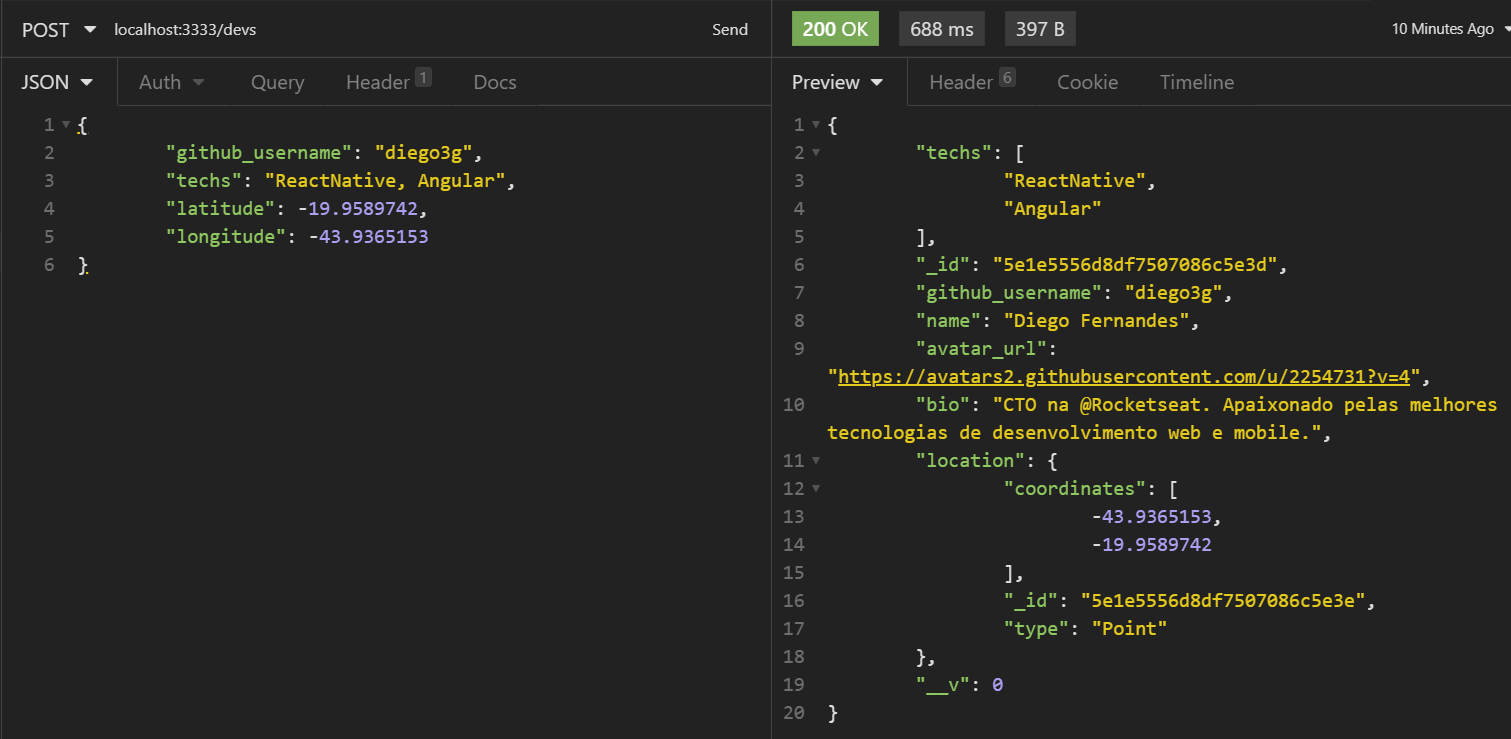
## Operador Lógico no mongo

<https://docs.mongodb.com/manual/reference/operator/query/>

# Chamadas à API criada

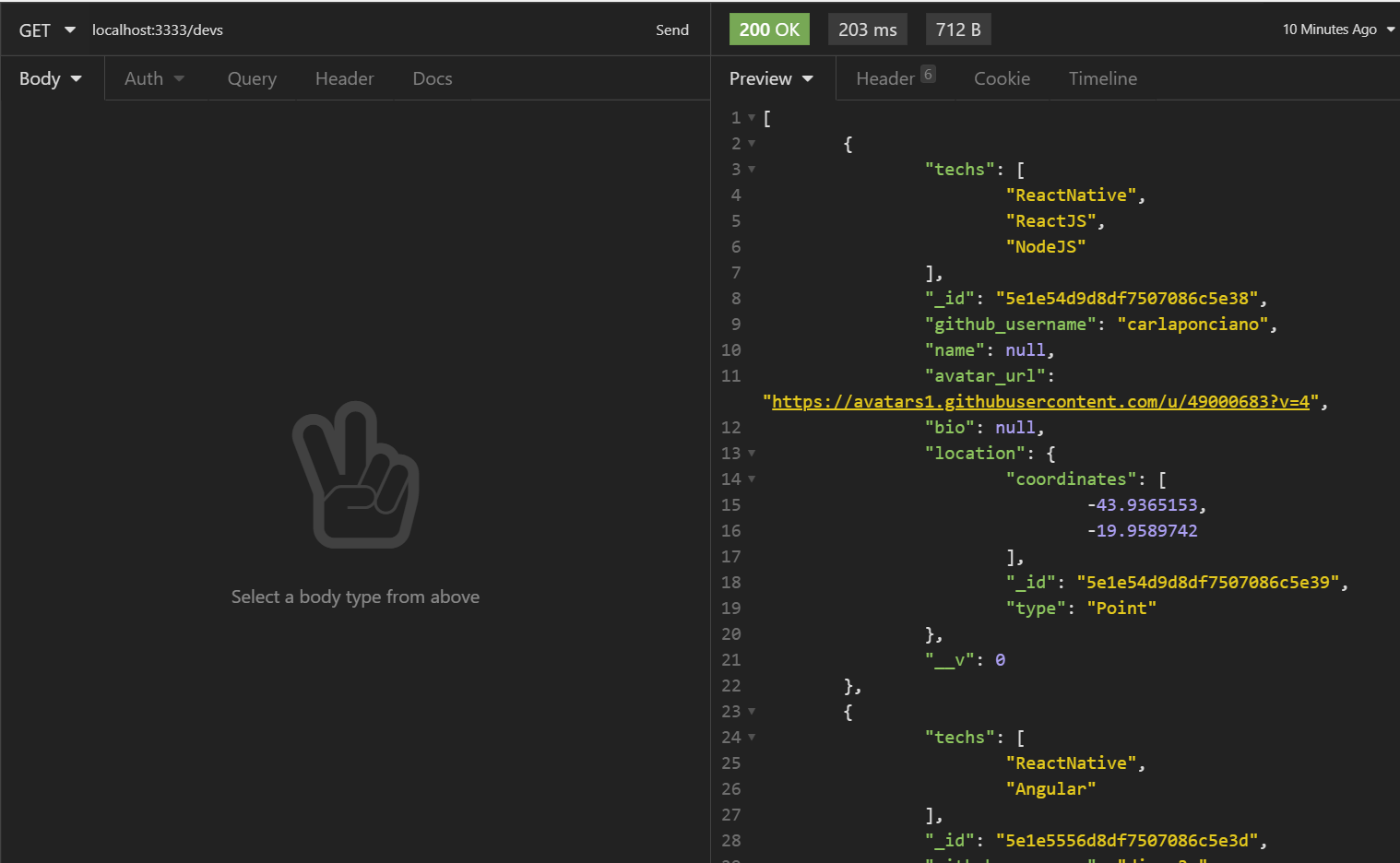
Ferramenta para testar as rotas: [insomnia.rest/](https://insomnia.rest/) (open-source)

Cadastrar Dev:



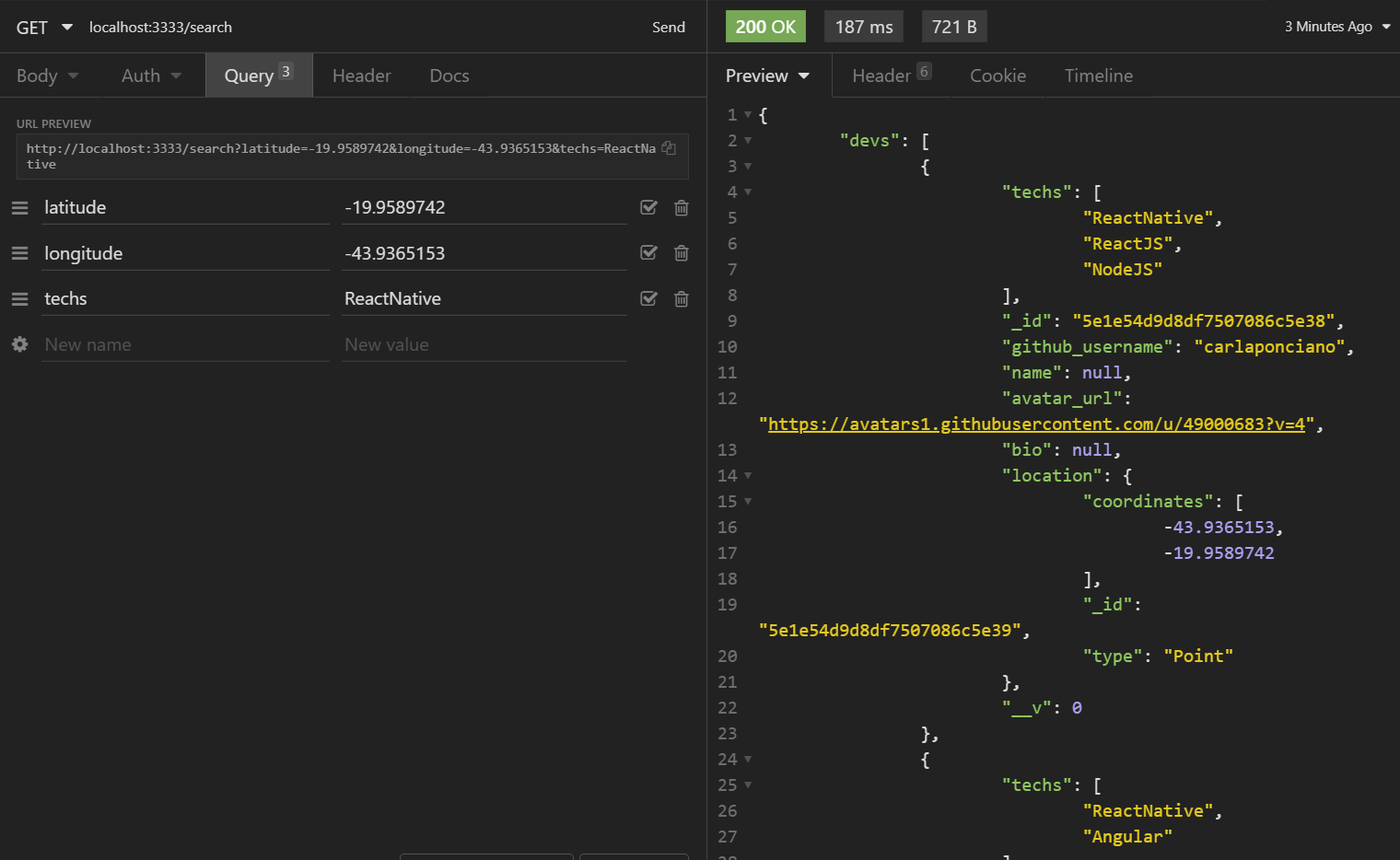
Envia os dados através do body em JSON.

Listar Devs



Sem o envio de dados. Lista todos cadastrados.

Buscar devs:



Envia os parâmetros na URL: Query