Segundo dia

- [x] ~~Node.js & Express~~
  - [x] ~~Rotas e recursos~~

Rota ou recurso é o que vem após a barra na URL, por exemplo: *http://localhost:3333/users*

Esse "*/users*" seria a rota da nossa aplicação

  - [x] ~~Métodos HTTP~~

GET: Buscar/Listar alguma informação do BackEnd

POST: Criar uma informação no BackEnd

PUT: Alterar informação no BackEnd

DELETE: Deletar uma informação no BackEnd

  - [x] ~~Tipos de parâmetros~~

Query Params: Parametros nomeados enviados na rota, após "?" (Filtros, Paginação)

URL no Insomnia: http://localhost:3333/users?name=Ikenas

```
app.post('/users', (request, response) => {
        const query = request.query
        console.log(query)
    })

    //--------------------------------------
    // vai retornar no console:

    {name: 'Ikenas'}
```

Route Params: Parametros utilizados para identificar recursos (Exemplo → *http://localhost:3333/users***/:id/**)

URL no Insomnia: http://localhost:3333/users/1

```
app.post('/users/:id', (request, response) => {
        const route = request.params

        console.log(route)
    })

    //--------------------------------------
    // vai retornar no console:

    {id: '1'}
```

Request Body: Corpo da request, utilizado para criar ou alterar recursos

URL do Insomnia: http://localhost:3333/users

Corpo da Request: (JSON)

```
{
      "name": "Matheus Ikenaga",
      "idade": "22"
    }
```

```
app.post('/users', (request, response) => {
        const body = request.body

        console.log(body)
    })

    //--------------------------------------
    // vai retornar no console:

    { name: 'Matheus Ikenaga', idade: '22' }
```

- [x] ~~Utilizando o Insomnia~~

Utilizado para fazer utilizar os métodos POST, PUT e DELETE na aplicação (A URL no navegador realiza apenas a GET)

- [x] ~~Configurando o Nodemon~~

```
npm install nodemon -D
```

O nodemon serve para sempre que houver uma alteração no código, o nodemon vai parar e inicializar o "server" sozinho, não precisando mais fazer o "*Ctrl C*" e depois o "`node index.js`"

O " -D " serve para essa "biblioteca" do nodemon ser aplicada apenas quando estivermos desenvolvendo, assim, o nodemon não vai para o deploy

```
// É necessário deixar dessa forma aqui no package.json:

  "scripts": {
      "start": "nodemon index.js"
    }

  //Assim ele executa automaticamente
```

Para inicializar o nodemon, utilizaremos o código `npm start` no terminal

☑ ~~Diferenças entre bancos de dados~~

SQL: MySQL, SQLite, PostgreSQL, Oracle, Microsoft SQL Server

NoSQL: MongoDB, CouchDB, etc

☑ ~~Configurando banco de dados~~

Iremos utilizar SQLite

Driver: SELECT * FROM users

Query Builder: `table('users').select('*').where()`

O Query Builder utilizado será o **knex.js**

No terminal:

```
npm install knex

    npm install sqlite3

    ----------------------------

    npx knex init
```

NPX executa um pacote

Irá criar um arquivo knexfile.js

```
// alterar para:

    development: {
        client: 'sqlite3',
        connection: {
          filename: './src/database/db.sqlite'
        }
      },
```

☑ ~~Pensando nas entidades e funcionalidades~~

☑ ~~Construção do Back-end~~

Primeiro colocamos as "migrations" que é um histórico de alterações das tabelas do banco

```
development: {
      client: 'sqlite3',
      connection: {
```

```
      filename: './src/database/db.sqlite'
    },
    migrations: {
      directory: './src/database/migrations'
    },
    useNullAsDefault: true,
  },
```

Criamos o arquivo de migration:

```
npx knex migrate:make create_ongs
```

Irá criar um arquivo na pasta "migrations"

Para criar tabelas:

```
exports.up = function(knex) {
    return knex.schema.createTable('ongs', function (table){
        table.string('id').primary();
        table.string('name').notNullable();
        table.string('email').notNullable();
        table.string('whatsapp').notNullable();
        table.string('city').notNullable();
        table.string('uf', 2).notNullable();
    })
};

exports.down = function(knex) {
  return knex.schema.dropTable('ongs');
};
```

Método UP, faz a criação da tabela

Método DOWN é pra caso aconteça algo errado com o UP, desfazer da seguinte forma:
... (dropTable, no caso)

Para executar:

```
npx knex migrate:latest
```

Agora fazer o mesmo com as outras entidades:

```
npx knex migrate:make create_incidents
```

```
exports.up = function(knex) {
    return knex.schema.createTable('incidents', function (table){
        table.increments(); //PK
```

```
        table.string('title').notNullable();
        table.string('description').notNullable();
        table.decimal('value').notNullable();

        table.string('ong_id').notNullable();

        table.foreign('ong_id').references('id').inTable('ongs'); //Foreign Key
    })
  };

  exports.down = function(knex) {
    return knex.schema.dropTable('incidents');
  };
```

```
npx knex migrate:latest
```

Caso precise voltar a ultima alteração da migration:

```
npx knex migrate:rollback
```

caso tenha dúvidas, podemos executar apenas

```
npc knex
```

Criei dentro da pasta database o arquivo connection.js

```
const knex = require('knex')
  const configuration = require('../../knexfile')

  const connection = knex(configuration.development)

  module.exports = connection;
```

Após isso, estamos prontos para criar os métodos

No arquivo routes.js:

```
const express = require('express')
  const crypto = require('crypto')

  const connection = require('./database/connection');

  const routes = express.Router();

  //-----------------------------------------------------------------------------
  /*
   *  *********     ROTA PARA LISTAR NO BD     **************
   */
  routes.get('/ongs', async (request, response) =>{
      const ongs = await connection('ongs').select('*');
```

```
    return response.json(ongs);
})
//-----------------------------------------------------------------------------
/*
 *  *********     ROTA PARA INSERT NO BD     **************
 */

routes.post('/ongs', async (request, response) => { //o async deixa a request assincrona
    const {name, email, whatsapp, city, uf } = request.body;

    //para criar ID
    const id = crypto.randomBytes(4).toString('HEX');

    // o await espera o insert ser todo executado para prosseguir no código
    await connection('ongs').insert({
        id,
        name,
        email,
        whatsapp,
        city,
        uf,
    })

    return response.json({ id });
})
//-----------------------------------------------------------------------------

module.exports = routes;
```

Criar uma pasta "controllers" dentro da src

E dentro dessa pasta, um arquivo para cada entidade, no caso a "ongController.js"

passar os métodos para dentro do ongController.js e suas bibliotecas:

```
const crypto = require('crypto')
    const connection = require('../database/connection');

    module.exports = {


        // método para listar
        async index (request, response){
            const ongs = await connection('ongs').select('*');

            return response.json(ongs);
        },



        //método para inserir
        async create (request, response){
            const {name, email, whatsapp, city, uf } = request.body;

        //para criar ID
            const id = crypto.randomBytes(4).toString('HEX');

            // o await espera o insert ser todo executado para prosseguir no código
            await connection('ongs').insert({
                id,
                name,
                email,
```

```
            whatsapp,
            city,
            uf,
        })

        return response.json({ id });
        }
}
```

Agora no arquivo routes.js :

```
const express = require('express')
    const ongController = require('./controllers/ongController')

    const routes = express.Router();

    routes.get('/ongs', ongController.index)

    routes.post('/ongs', ongController.create)

    module.exports = routes;
```

Agora fazer o mesmo com as outras coisas, por exemplo as incidents (casos)... Criar um arquivo incidentController.js

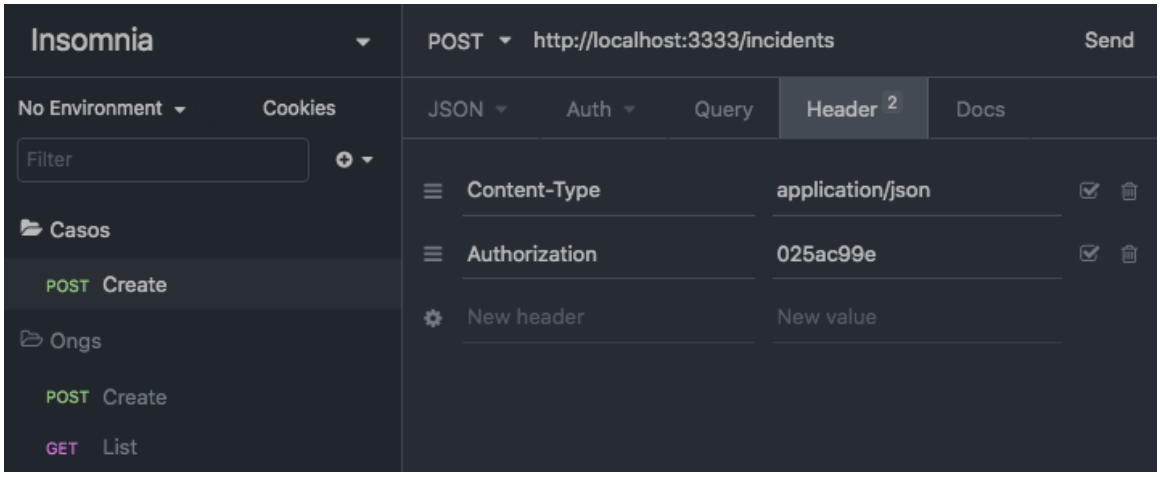
```
const connection = require('../database/connection')

    module.exports = {
        async create(request, response){
            const{title, description, value} = request.body;
            const ong_id = request.headers.authorization

            const [id] = await connection('incidents').insert({
                title,
                description,
                value,
                ong_id,
            })
            return response.json({id})
        }
    }
```

No Insomnia, criar a pasta Casos e colocar o método create:

Endereço: http://localhost:3333/incidents

```
{
      "title": "Caso1",
      "description": "Detalhes do caso",
      "value": 120
    }
```

E nos HEADERS, colocar o ID da ONG:



Em seguida, criar o método index dos casos:

```
const connection = require('../database/connection')

    module.exports = {

        async index (request, response) {
            const incidents = await connection('incidents').select('*')
            response.json(incidents)
        },

        async create(request, response){
            const{title, description, value} = request.body;
            const ong_id = request.headers.authorization

            const [id] = await connection('incidents').insert({
                title,
                description,
                value,
                ong_id,
            })
            return response.json({id})
        }


    }
```

E implementar na routes.js

```
const express = require('express')
    const ongController = require('./controllers/ongController')
    const incidentController = require('./controllers/incidentController')

    const routes = express.Router();

    routes.get('/ongs', ongController.index)
    routes.post('/ongs', ongController.create)

    routes.get('/incidents', incidentController.index)
    routes.post('/incidents', incidentController.create)

    module.exports = routes;
```

Criar o Delete

```
// no routes.js
    const express = require('express')
    const ongController = require('./controllers/ongController')
    const incidentController = require('./controllers/incidentController')
    const profileController = require('./controllers/profileController')
    const sessionController = require('./controllers/sessionController')

    const routes = express.Router();

    routes.post('/sessions', sessionController.create)

    routes.get('/ongs', ongController.index)
    routes.post('/ongs', ongController.create)

    routes.get('/profile', profileController.index)

    routes.get('/incidents', incidentController.index)
    routes.post('/incidents', incidentController.create)
    routes.delete('/incidents/:id', incidentController.delete)


    module.exports = routes;
```

```
// no IncidentController
    async delete (request, response){
            const {id} = request.params;
            const ong_id = request.headers.authorization

            const incident = await connection ('incidents')
            .where('id', id)
            .select('ong_id')
            .first()

            if(incident.ong_id !== ong_id){
                return response.status(401).json({error: 'Operation not permitted.'})

            }

            await connection('incidents').where('id', id).delete()

            return response.status(204).send()

        }
```

Também é necessário criar a listagem de uma ong específica:

profileController.js

```
const connection = require('../database/connection')

    module.exports = {
        async index(request,response){
            const ong_id = request.headers.authorization
```

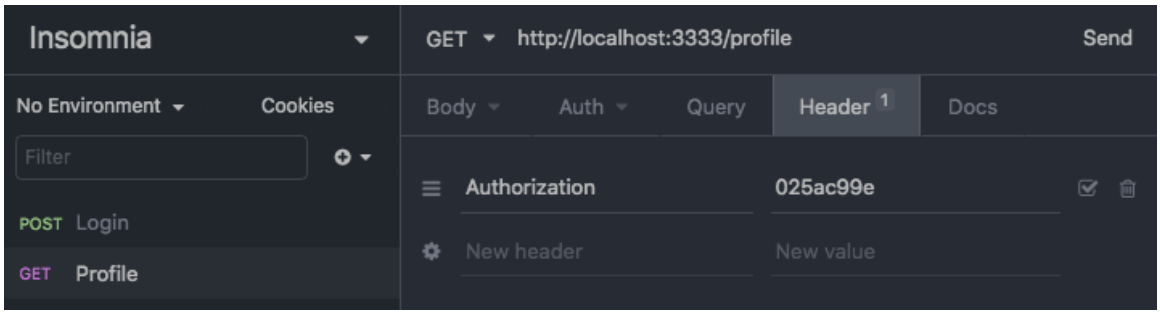
```
        const incidents = await connection('incidents').where('ong_id', ong_id).select('*')
        return response.json(incidents)
    }
}
```

A parte de Login

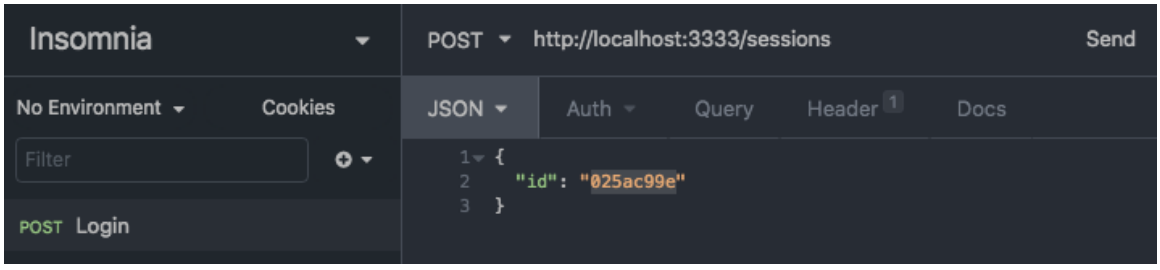
sessionController.js

```
const connection = require('../database/connection')

    module.exports = {

        async create(request,response){
            const {id} = request.body

            const ong = await connection('ongs')
            .where('id',id)
            .select('name')
            .first()


            if(!ong) {
                return response.status(400).json({error: 'NO ONG found with this ID'})
            }

        return response.json(ong)
        }

    }
```

No Insomnia, criar os métodos Profile



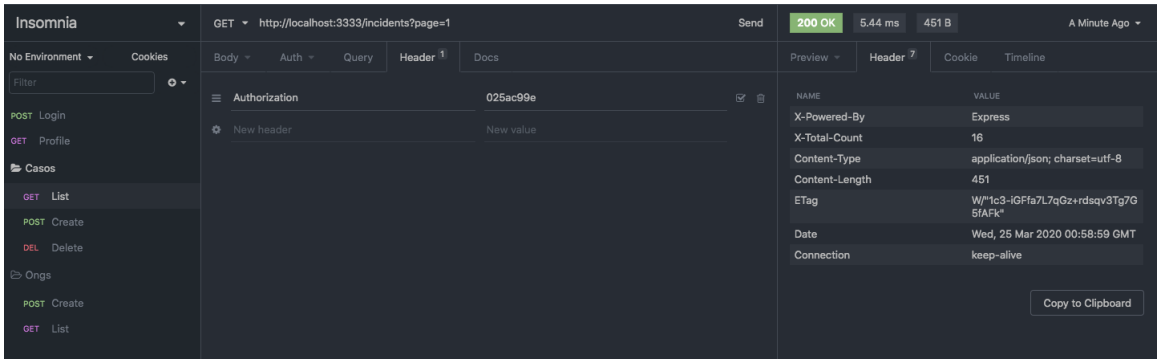
E o Login



E agora vamos implementar paginação no incidentController

```
incidentController.js

    async index (request, response) {

            const { page = 1 } = request.query

            const [count] = await connection('incidents').count()
            console.log(count)

            const incidents = await connection('incidents')
            .limit(5)
            .offset((page - 1)* 5)
            .select('*')

            response.header('X-Total-Count', count['count(*)'])
            response.json(incidents)
        }
```

No insomnia : (retorna a quantidade no X-Total-Count), mesmo na page=1



Agora colocar as informações da ong(email, etc), junto com os incidentes

```
//incidentController.js

    async index (request, response) {

            const { page = 1 } = request.query

            const [count] = await connection('incidents').count()

            const incidents = await connection('incidents')
            .join('ongs', 'ong_id', '=', 'incidents.ong_id' )
            .limit(5)
            .offset((page - 1)* 5)
            .select(['incidents.*',
                'ongs.name',
                'ongs.email',
                'ongs.whatsapp',
                'ongs.city',
                'ongs.uf'])

            response.header('X-Total-Count', count['count(*)'])
            response.json(incidents)
        }
```

FINALIZADO O BACKEND

☑ ~~Adicionando módulo CORS~~

É para a segurança da aplicação, primeiro parar ela (Ctrl C) e em seguida

```
npm install cors
```

E atualizar o index.js

```
const express = require('express');
    const cors = require('cors')
    const routes = require('./routes')


    const app = express();

    app.use(cors())
    app.use(express.json())
    app.use(routes)


    app.listen(3333);
```

☑ ~~Enviando back-end ao GitHub~~

Entidades

- ONG
- Caso (incident)

Funcionalidades

- Login de ONG
- Logout de ONG
- Cadastro de ONG
- Cadastrar novos casos

- Deletar casos
- Listar casos específicos de uma ONG
- Listar todos os casos
- Entrar em contato com a ONG