

Relatório

Data de criação do relatório - 16/03/2023

Desenvolvedor(a) - Elem J S Cazumbá

Objetivo:

Criar um sistema de cadastro de cliente que possua nome, telefone, email, logradouro, número, complemento, bairro, cidade, uf, cep, tipo de cliente; Onde a API seja capaz de buscar os tipos de clientes, gravar os dados enviados no banco de dados (através do método POST) e consultar os dados no banco de dados no banco de dados baseados no ID e email.

Atividades envolvidas:

- Instalação do docker para desktop
- Instalação do mysql no docker
- Criação de um banco de dados
- Escrever uma API
- Instalar no docker um servidor node.js
- Criar conexão entre container node e container mysql

Instalação do Docker para desktop

Antes da instalação propriamente dita é necessário habilitar o subsistema linux para windows, seguindo todas as etapas listadas no site:

[Manual installation steps for older versions of WSL | Microsoft Learn](#)

Instalação do WSL

1º passo: Abrir o powershell como administrador e executar o comando abaixo e após isso reinicie o sistema.

```
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
```

Dificuldades encontradas nesta etapa: *Comando executado com sucesso, sem falhas.*

2º passo: Abrir o powershell como administrador e executar o comando abaixo e após isso reinicie o sistema.

dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart

Dificuldades encontradas nesta etapa: *Comando executado com sucesso, sem falhas.*

3º passo: Fazer o download do último pacote do wsl acessando a página:

[WSL2 Linux kernel update package for x64 machines](#)

Dificuldades encontradas nesta etapa: *download realizado e executado com sucesso, sem falhas.*

4º passo: Abrir o powershell como administrador e executar o comando:

wsl --set-default-version 2

Dificuldades encontradas nesta etapa: *Download realizado e executado com sucesso, sem falhas.*

5º passo: Fazer o download utilizando a Microsoft store da aplicação linux. Foi instalado o aplicativo Ubuntu, executado e criado um login UNIX e senha.

Dificuldades encontradas nesta etapa: *download realizado e executado com sucesso, sem falhas.*

Download do Docker

Descrição: Após acessar o site [Install Docker Desktop on Windows](#) download foi concluído com êxito.

Instalação

Descrição: Após a instalação foi necessário conceder as permissões especiais para o administrador do sistema, sendo concluído com êxito.

Inicialização

Dificuldades encontradas nesta etapa: download realizado e executado com sucesso, sem falhas.

Instalação do Mysql no docker

Com o docker startado, utilizei o prompt de comando para executar as etapas abaixo:

- 1 - Criar no docker um container que contenha um imagem do mysql
docker run -p 3306:3306 --name nome-do-container -d mysql/mysql-server

```
C:\Users\Felipe>docker run -p 3306:3306 --name atvmysql -d mysql/mysql-server
Unable to find image 'mysql/mysql-server:latest' locally
latest: Pulling from mysql/mysql-server
6a4a3ef82cdc: Pull complete
5518b09b1089: Pull complete
b6b576315b62: Pull complete
349b52643cc3: Pull complete
abe8d2406c31: Pull complete
c7668948e14a: Pull complete
c7e93886e496: Pull complete
Digest: sha256:d6c8301b7834c5b9c2b733b10b7e630f441af7bc917c74dba379f24eeeb6a313
Status: Downloaded newer image for mysql/mysql-server:latest
3a466df3dd88d614c02264e3ae77a777aba4adb6baecf77bc09161b74fcdcf9
```

Dificuldades encontradas nesta etapa: Comando executado com sucesso, sem falhas.

- 2- Abrir os logs do container para que seja conhecida a senha aleatória que será utilizada em etapas a frente:

docker logs id-do-container

```
socket: '/var/lib/mysql/mysql.sock' port: 0 MySQL Community Server - GPL.
2023-03-12T00:43:06.383297Z 0 [System] [MY-011323] [Server] X Plugin ready for connec
lx.sock
Warning: Unable to load '/usr/share/zoneinfo/iso3166.tab' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/leapseconds' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/tzdata.zi' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/zone.tab' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/zone1970.tab' as time zone. Skipping it.
[Entrypoint] GENERATED ROOT PASSWORD: 7z1g_5N,e5M#61jc2:k2c#:M;=BHupGD
```

Dificuldades encontradas nesta etapa: Comando executado com sucesso, sem falhas.

3 - Acessar o container com a senha que foi informada na etapa anterior.

docker exec -it nome-do-container mysql -uroot -p

```
C:\Users\Felipe>docker exec -it atvmysql mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 718
Server version: 8.0.32

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> █
```

Dificuldades encontradas nesta etapa: Na primeira tentativa retornou o “*ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)*”. Notei que a senha foi digitada incorretamente. Com atenção redobrada o comando foi executado com sucesso, sem falhas.

4 - Alterar a senha de usuário do localhost

ALTER USER 'root'@'localhost' IDENTIFIED BY '123456';

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY '123456';
Query OK, 0 rows affected (6.65 sec)
```

Dificuldades encontradas nesta etapa: Comando executado com sucesso, sem falhas.

5 - Criar um novo usuário

CREATE USER 'user1'@'%' IDENTIFIED WITH mysql_native_password BY '123456';

```
mysql> CREATE USER 'user1'@'%' IDENTIFIED WITH mysql_native_password BY '123456';
Query OK, 0 rows affected (1.46 sec)
```

Dificuldades encontradas nesta etapa: Comando executado com sucesso, sem falhas.

6 - Conceder privilégios a esse novo usuário em todo o banco de dados

*GRANT ALL PRIVILEGES ON *.* TO 'user1'@'%' WITH GRANT OPTION;*

```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'user1'@'%' WITH GRANT OPTION;  
Query OK, 0 rows affected (1.24 sec)
```

Dificuldades encontradas nesta etapa: Comando executado com sucesso, sem falhas.

7 - Atualizar tabelas de permissões e fazendo com que as modificações feitas entrem em vigor

```
FLUSH PRIVILEGES;
```

```
mysql> FLUSH PRIVILEGES;  
Query OK, 0 rows affected (1.15 sec)  
  
mysql>
```

Dificuldades encontradas nesta etapa: Comando executado com sucesso, sem falhas.

Criação de um banco de dados

Foi criado um banco de dados utilizando os seguintes comando:

```
CREATE DATABASE atv1403;
```

```
USE atv1403;
```

```
CREATE TABLE cliente(idcliente int NOT NULL AUTO_INCREMENT PRIMARY KEY, nome  
varchar(100) NOT NULL, telefone varchar(25) NOT NULL, email varchar(100) NOT NULL);
```

```
CREATE TABLE endereco(idendereco int NOT NULL AUTO_INCREMENT PRIMARY KEY,  
logradouro varchar(300) NOT NULL, numero varchar(5) NOT NULL, complemento  
varchar(500) NOT NULL, bairro varchar(50) NOT NULL, cidade varchar(50) NOT NULL, uf  
VARCHAR(3) NOT NULL, cep varchar(15) NOT NULL);
```

```
CREATE TABLE tipocliente(idtipocliente int NOT NULL AUTO_INCREMENT PRIMARY  
KEY, tipo varchar(50) NOT NULL);
```

Observações: Na tabela “tipocliente” foi inserido 4 tipos de clientes na coluna “tipo”, foram tipos: Prime, Gold, Silver e basic. Para isso foi utilizadoo comando *INSERT INTO tabela(campo)VALUE('valor');*

Dificuldades encontradas nesta etapa: Comando executado com sucesso, sem falhas.

Escrever uma API

Foi criada uma nova pasta no disco c: do computador na pasta documentos chamada `atv1403`; este diretório foi acessado pelo prompt de comando e criado também um `package.json` com o comando: `npm init -y`. Esta pasta tbm foi aberta utilizando o VS Code. Foi adicionado no arquivo `package.json` na parte “script” a linha “`start`”: “`nodemon index.js`”:

```
package.json > ...
1  {
2    "name": "atv1403",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "start": "nodemon index.js",
8      "test": "echo \"Error: no test specified\" && exit 1"
9    },
10   "keywords": [],
11   "author": "",
12   "license": "ISC",
13   "dependencies": {
14     "cors": "^2.8.5",
15     "express": "^4.18.2",
16     "knex": "^2.4.2",
17     "mysql2": "^3.2.0",
18     "nodemon": "^2.0.21"
19   }
20 }
21
```

Depois foi instalado as extensões `mysql2`, para que fosse possível utilizar o `mysql` juntamente ao `node`; o `express` para gerenciar pacotes e requisições de diferentes verbos HTTP em diferentes URLs, Integrar "view engines" para inserir dados nos templates e instalado tbm o `nodemon` para ajudar no desenvolvimento de sistemas com o `Node.js` reiniciando automaticamente o servidor, ele monitora a aplicação em `Node`, e assim que há qualquer mudança no código, o servidor é reiniciado automaticamente. Foi utilizado respectivamente os seguintes comandos:

```
npm init -y
npm install mysql2
npm install express
npm install nodemon
```

Dificuldades encontradas nesta etapa: Comandos executados com sucesso, sem falhas.

Criar um arquivo.js para conexão com o banco de dados

Utilizando o VS Code foi criado um arquivo chamado “db.js” para definir uma conexão com o banco e uma constante para o objeto que vai carregar a extensão mysql - e que mais tarde usaremos para conectar, executar SQL, etc. Para isso foi utilizado o seguinte código:

```
const mysql      = require('mysql2');
const connection = mysql.createConnection({
  host      : 'atvmysql', //Nome do container no docker contendo
o mysql
  port      : 3306, //Porta padrão
  user      : 'user1', //Nome de usuário do msq1
  password  : '123456', //Senha do usuário
  database  : 'atv1403' //Nome do banco de dados a ser acessado
});

connection.connect((err) => {
  if(err) return console.log(err);
  console.log('conectou!');
});

module.exports = connection; // Utilizado para que o arquivo
principal tenha acesso ao arquivo db.js
```

O comando “*node db.js*” foi utilizado para fazer o teste da conexão.

Dificuldades encontradas nesta etapa: Comandos executados com sucesso, sem falhas.

Criar um arquivo.js para API

Utilizando o VS Code foi criado um arquivo chamado "index.js" para criar uma API usando Express para conseguir conectar com Node.js + MySQL.

Foi criado um arquivo index.js na pasta do projeto onde foi criado o servidor da API para tratar as requisições, configurado a aplicação (app) express para usar o body parser do Express, criado um middleware inicial que apenas exibe uma mensagem de sucesso quando o usuário requisitar um GET na raiz da API (/) para ver se está funcionando. Por fim, adicionamos as linhas abaixo no final do arquivo que dão o start no servidor da API. Foram usadas as seguintes linhas de código:

```
const express = require('express'); //Configuração de servidor
const app = express();
const port = 3000;
const mysql = require('mysql2');
const connecion = require('./db.js');

app.use(express.json()); //configurarr para usar o body parser do
express

//Iniciar o servidor
app.listen(port);
console.log('API funcionando!');
}
```

Para iniciar os testes foi dado o comando npm start e o retorno foi uma de sucesso no terminal e tbm no navegado (192.168.1.68:3000) , como mostrado na imagem abaixo:

```
PS C:\Users\Windows 11\Documents\DevMobile\atv1403> npm start

> atv1403@1.0.0 start
> nodemon index.js

[nodemon] 2.0.21
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
API funcionando!
conectou!
```


← → ↻ 🏠 ⚠ Não seguro | 192.168.1.68:3000

```
{"message": "Funcionando!"}
```

Em seguida foi adicionado a esse arquivo as linhas seguintes para estabelecer conexão com o banco de dados e a função `execSQLQuery(sqlQry, res)` para que fosse possível termos queries SQL nas funções de resposta das rotas:

```
function execSQLQuery(sqlQry, res) {  
  const connection = mysql.createConnection({  
    host      : 'atvmysql',  
    port      : 3306,  
    user      : 'user1',  
    password  : '123456',  
    database  : 'atv1403'  
  });  
  
  connection.query(sqlQry, (error, results, fields) => {  
    if(error)  
      res.json(error);  
    else  
      res.json(results);  
    connection.end();  
    console.log('executou!');  
  });  
}
```

Mostrando a lista de tipos de cliente

Foi criada uma função que executa consultas SQL no banco usando uma conexão que será criada a cada uso, e adicionado uma rota/tipocliente que lista todos os tipos de cliente quando feita uma requisição URL

localhost:3000/tipoclientes

```
app.get('/tipocliente', (req, res) => {  
  execSQLQuery('SELECT * FROM tipocliente', res);  
})
```

← → ↻ 🏠 ⚠ Não seguro | 192.168.1.68:3000/tipocliente

```
[{"idtipocliente":1,"tipo":"Prime"}, {"idtipocliente":2,"tipo":"Gold"}, {"idtipocliente":3,"tipo":"Silver"}, {"idtipocliente":4,"tipo":"basic"}]
```

E com isso finalizamos a listagem de todos os tipos de clientes na nossa API!

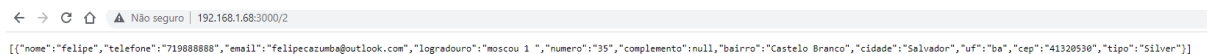
Buscas por id ou email

Caso o usuário quiser ver apenas um cliente, ele deverá passar o ID do mesmo na URL, logo após o /clientes. da seguinte forma: URL *localhost:3000/cliente/(id)*

Para fazer isso, foi criada uma rota para aceitar um parâmetro opcional ID. Além disso, dentro do processamento da rota, se vier o ID, devemos fazer uma consulta diferente da anterior, como mostra o código abaixo:

```
app.get('/cliente/:id?', (req, res) => {
  let filter = '';
  if(req.params.id) filter = ' WHERE idcliente=' +
parseInt(req.params.id);
  execSQLQuery('select nome, telefone, email, logradouro, numero,
complemento, bairro, cidade, uf, cep, tipo from cliente join
tipocliente on cliente.idtipocliente = tipocliente.idtipocliente'
+ filter, res);
})
```

Fazendo o teste no navegador, veremos que está funcionando perfeitamente. Será mostrado - referente ao cliente - o nome, telefone, email, logradouro, número, complemento, bairro, cidade, uf, cep e qual o tipo de cliente, como mostra a imagem abaixo:



The screenshot shows a web browser address bar with the URL `192.168.1.68:3000/2`. Below the address bar, a JSON array is displayed: `[{"nome": "felipe", "telefone": "719888888", "email": "felipecazumba@outlook.com", "logradouro": "moscou 1", "numero": "35", "complemento": null, "bairro": "Castelo Branco", "cidade": "Salvador", "uf": "ba", "cep": "41320530", "tipo": "Silver"}]`

Caso o usuário queira buscar os dados de um cliente filtrando por nome de e mail deverá passar a seguinte requisição URL *localhost:3000/email?email=*

```
app.get('/email', function (req, res) {
  let email = req.query['email'];

  execSQLQuery('select nome, telefone, email, logradouro, numero,
complemento, bairro, cidade, uf, cep, tipo from cliente join
tipocliente on cliente.idtipocliente = tipocliente.idtipocliente
where cliente.email = ' + "'" + email + "'" , res )
});
```

Fazendo o teste no navegador, veremos que está funcionando perfeitamente. Será mostrado - referente ao cliente - o nome, telefone, email, logradouro, número, complemento, bairro, cidade, uf, cep e qual o tipo de cliente, como mostra a imagem abaixo:

← → ↻ 🏠 Não seguro | 192.168.1.68:3000/email?email=elem@hotmail.com

```
[{"nome":"elem","telefone":"52","email":"elem@hotmail.com","logradouro":"1521","numero":"56","complemento":"65","bairro":"654","cidade":"4","uf":"56","cep":"6456","tipo":"Prime"}]
```

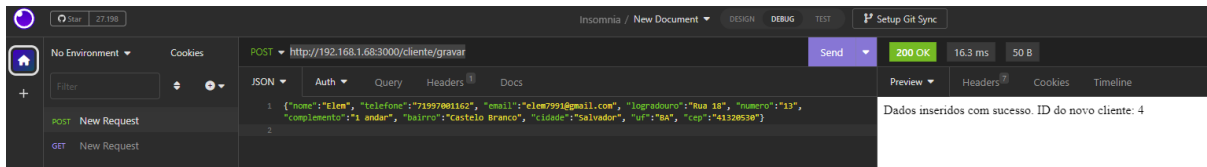
Por fim, foi criada uma rota usando o método post onde que capacita a API a receber informações enviadas pelo usuário e essas, por sua vez, são salvas no banco de dados que foi anteriormente criado e que está conectado à API (atv1403). Isso foi feito utilizando as seguintes linhas de código:

```
app.post('/cliente/gravar', (req, res) => {
  const {nome, telefone, email, logradouro, numero, complemento,
    bairro, cidade, uf, cep, idtipocliente} = req.body;

  const sql = 'INSERT INTO cliente (nome, telefone, email,
    logradouro, numero, complemento, bairro, cidade, uf, cep,
    idtipocliente) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)';
  connection.query(sql, [nome, telefone, email, logradouro,
    numero, complemento, bairro, cidade, uf, cep, idtipocliente],
    (err, result) => {
      if (err) {
        console.log(`Erro ao inserir dados no banco de dados:
        ${err.message}`);
        res.status(500).send('Erro interno do servidor');
      } else {
        console.log(`Dados inseridos com sucesso. ID do novo
        cliente: ${result.insertId}`);
        res.status(200).send(`Dados inseridos com sucesso. ID do
        novo cliente: ${result.insertId}`);
      }
    });
});
```

Esse teste não pode ser feito no navegador por se tratar do método post. Então utilizei o software chamado Insomnia para testar essa rota. Abrindo o aplicativo selecionei o método post e a forma de envio como formato JSON. Após isso, respeitando as regras de sintaxe e com os nomes dos campos idênticos aos campos do banco de dados conectado, fiz um texto teste para ser enviado utilizando a URL `http://localhost:3000/cliente/gravar`:
`{"nome":"Elem", "telefone":"71997001162", "email":"elem7991@gmail.com", "logradouro":"Rua 18", "numero":"13", "complemento":"1 andar", "bairro":"Castelo Branco", "cidade":"Salvador", "uf":"BA", "cep":"41320530"}`

O retorno foi uma mensagem de sucesso, como mostra a imagem:



Dificuldades encontradas nesta etapa: As maiores dificuldades encontradas estavam ligadas à falta de conhecimento no que diz respeito à sintaxe que deveria ser utilizada para escrever SQL usando as funções de request e response. As informações foram buscadas na internet em páginas on-line e diversos vídeos além dos colegas que foram de grande ajuda. A ferramenta Insomnia foi escolhida para fazer os testes iniciais do método post pois devido ao pouco tempo que restava para concluir a atividade em questão ela se mostrou mais simples, com relação a sua utilização.

Instalar no docker um servidor node.js / conectá-lo ao container mysql

Na pasta criada para conter os arquivos dessa atividade foi criado dois arquivos: um docker file e um .dockerignore, com o auxílio da ferramenta VisualCode.

Dockerfile - Foi um arquivo utilizado para construir uma imagem personalizada baseada em node js a ser utilizada posteriormente no container que será criado para ser utilizado como servidor web. Nesse arquivo conteve todas as informações e comandos que seriam necessários para a nossa aplicação.

```
FROM node:alpine
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

.dockerignore - Foi um arquivo utilizado para informar quais arquivos deveriam ser ignorados no momento em que a imagem fosse criada, para evitar que arquivos desnecessários ocupassem algum espaço.

```
node_modules
```

Após isso foi dados no terminal o comando para construir a imagem com as informações acima: *docker build -t nome-da-imagem* . (o ponto faz parte do comando).

```

PS C:\Users\Windows 11\Documents\DevMobile\atv1403> docker build -t imgnode .      [+] Building 1.1s (10/10) FINISHED
=> [internal] load build definition from Dockerfile                                0.0s
=> => transferring dockerfile: 32B                                                0.0s
=> [internal] load .dockerignore                                                  0.0s
=> => transferring context: 34B                                                  0.0s
=> [internal] load metadata for docker.io/library/node:alpine                    1.0s
=> [1/5] FROM docker.io/library/node:alpine@sha256:a67a33f791d1c86ced985f33      0.0s
=> [internal] load build context                                                0.0s
=> => transferring context: 1.53kB                                              0.0s
=> CACHED [2/5] WORKDIR /usr/src/app                                             0.0s
=> CACHED [3/5] COPY package*.json ./                                           0.0s
=> CACHED [4/5] RUN npm install                                                  0.0s
=> [5/5] COPY . .                                                              0.0s
=> exporting to image                                                            0.0s
=> => exporting layers                                                            0.0s
=> => writing image sha256:b2eda40edcd3ea36654eea0de206b0567c1d8f6d5a76f198    0.0s
=> => naming to docker.io/library/imgnode                                       0.0s

```

com isso a imagem foi criado como mostra o docker:

<div> <div>Q Search</div> <div></div> <div></div> </div>						
<input type="checkbox"/>	Name	Tag	Status	Created	Size	Actions
<input type="checkbox"/>	imgnodejs 5ef804d1f6e2	latest	In use	1 day ago	184.91 MB	▶ ⋮ 🗑

Após isso, foi dado o comando para construir o container baseado nessa imagem criada a partir do dockerfile: `docker run -p 3000:3000 -d --name=nome-container --link nome-conteiremysql:mysql nome-da-imagembuildada`

```

PS C:\Users\Windows 11\Documents\DevMobile\atv1403> docker run -p 3000:3000 -d --name contnodeservidor --link atvmysql:mysql imgnode
a96f418fa825ac278e757e516138896201cd35809b29d2763295008a7ca41cc6
PS C:\Users\Windows 11\Documents\DevMobile\atv1403>

```

com isso o container foi criado como mostra o docker:

<input type="checkbox"/>	Name	Image	Status	Port(s)	Last started	Actions
<input type="checkbox"/>	api-mysql b9cfa8b0e73c	imgnodejs	Running	3000:3000	4 seconds ago	▶ ⋮ 🗑
<input type="checkbox"/>	atvmysql 3ae8da65b865	mysql/mysql-server	Running	3306:3306	4 seconds ago	▶ ⋮ 🗑

Dificuldades encontradas nesta etapa: Comandos executados com sucesso, sem falhas.

Considerações finais

Devido a falta de atenção e conhecimento a atividade ainda não atendia as demandas e então decidi criar tudo novamente do “zero”, tendo em vista novos conhecimento que poderia ser melhor aplicados.

Criei uma pasta chamada newteste e esta pasta foi aberta no VS Code e então usado o terminal de comando utilizei, respectivamente os comando:

- `npm init -y`
- `npm install express`

- npm install mysql2
- npm install multer
- npm install cors
- npm instal nodemon

Foi adicionado no arquivo package.json na parte “script” a linha “start”: “nodemon index.js”:

Após isso criei um arquivo index com as linhas abaixo

```
const connection = require('./db');
const mysql = require('mysql2');
const express = require('express');
const cors = require('cors');
const app = express();
const multer = require('multer')
const upload = multer();

const port = 3000;

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

app.get('/', (req, res) => res.json({ message:
'Funcionando!' }));

app.get('/cliente/buscar/email/:email', function (req,
res) {

    let email = req.params.email

    execSQLQuery('select nome, telefone, email,
logradouro, numero, complemento, bairro, cidade, uf,
cep, tipo from cliente join tipocliente on
cliente.idtipocliente = tipocliente.idtipocliente where
cliente.email = ' + '"' + email + '"' , res )
```

```
});
```

```
app.get('/tipocliente', (req, res) => {  
    execSQLQuery('SELECT * FROM tipocliente', res);  
})
```

```
app.get('/cliente/buscar/id/:id', (req, res) => {  
    let filter = '';  
    if(req.params.id) filter = ' WHERE idcliente=' +  
parseInt(req.params.id);  
    execSQLQuery('select nome, telefone, email,  
logradouro, numero, complemento, bairro, cidade, uf,  
cep, tipo from cliente join tipocliente on  
cliente.idtipocliente = tipocliente.idtipocliente' +  
filter, res);  
})
```

```
app.post('/cliente/gravar', upload.any(), (req, res)  
=> {  
    const {nome, telefone, email, logradouro, numero,  
complemento, bairro, cidade, uf, cep, idtipocliente } =  
req.body;  
  
    const sql = 'INSERT INTO cliente (nome, telefone,  
email, logradouro, numero, complemento, bairro, cidade,  
uf, cep, idtipocliente) VALUES (?, ?, ?, ?, ?, ?, ?, ?,  
?, ?, ?)';
```

```
    connection.query(sql, [ nome, telefone, email,
logradouro, numero, complemento, bairro, cidade, uf,
cep, idtipocliente], (err, result) => {
    if (err) {
        console.log(`Erro ao inserir dados no banco de
dados: ${err.message}`);
        res.status(500).send('Erro interno do
servidor');
    } else {
        console.log(`Dados inseridos com sucesso. ID do
novo cliente: ${result.insertId}`);
        res.status(200).send(`Dados inseridos com
sucesso. ID do novo cliente: ${result.insertId}`);
    }
    });
});
```

```
app.listen(port);
console.log('API funcionando!');
```

```
function execSQLQuery(sqlQry, res){
    const connection = mysql.createConnection({
        host      : 'atvmysql',
        port      : 3306,
        user       : 'user1',
        password   : '123456',
        database   : 'atv1403'
    });
```



```

    connection.query(sqlQry, (error, results, fields)
=> {
        if(error)
            res.json(error);
        else
            res.json(results);
        connection.end();
        console.log('executou!');
    });
}

```

Criei um arquivo chamado db.js com as linhas:

```

const mysql      = require('mysql2');
const connection = mysql.createConnection({
    host      : 'atvmysql',
    port      : 3306,
    user      : 'user1',
    password  : '123456',
    database  : 'atv1403'
});

connection.connect((err) => {
    if(err) return console.log('Erro ao tentar se
conectar com o banco de dados :(');
    console.log('conectado ao banco de dados!');
});
module.exports = connection;

```

Criei tbm um arquivo dockerfile:

```

FROM node:alpine
WORKDIR /usr/src/app
COPY package*.json ./

```

```
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

E um arquivo chamado `.dockerignore` com a linha:

```
node_modules
```

Após isso utilizei um comando para buildar a imagem e outro para contruir um container ligado ao container já existente mysql:

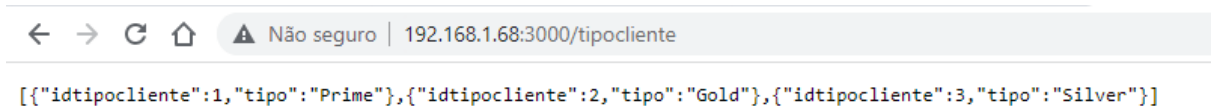
- `docker build -t nome-da-imagem .`
- `docker run -p 3000:3000 -d --name=nome-container --link nome-conteiremysql:mysql nome-da-imagembuildada`

Tudo funcionou perfeitamente no navegador e também foi testado na ferramenta postman

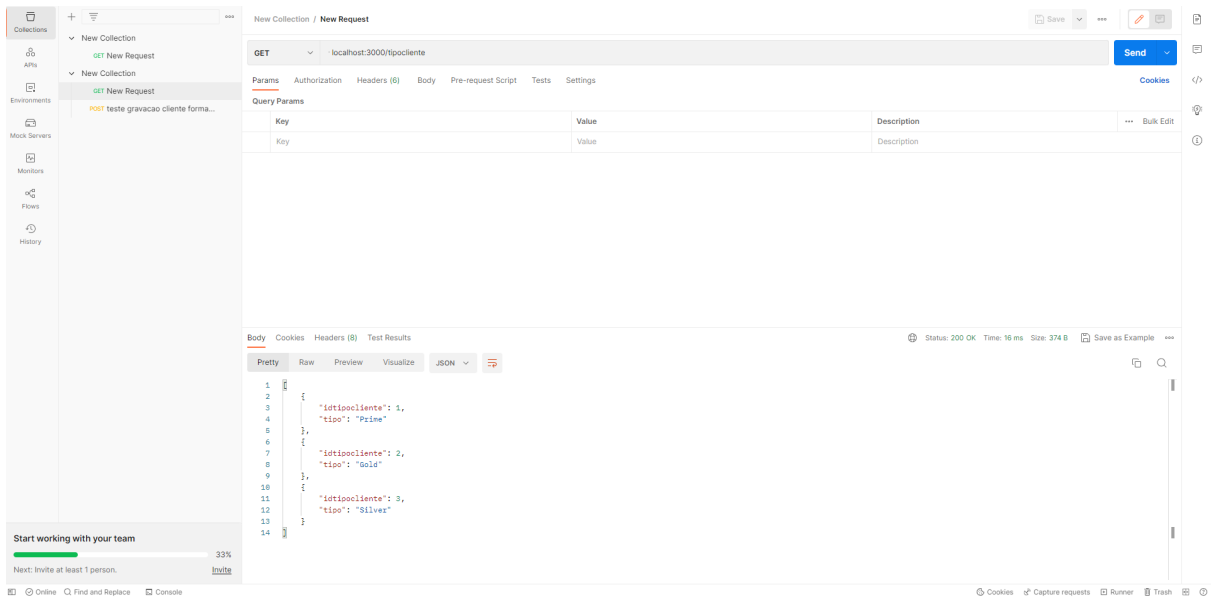
Para retornar os tipos de cliente

- **URL: localhost:3000/tipocliente**

teste no navegador

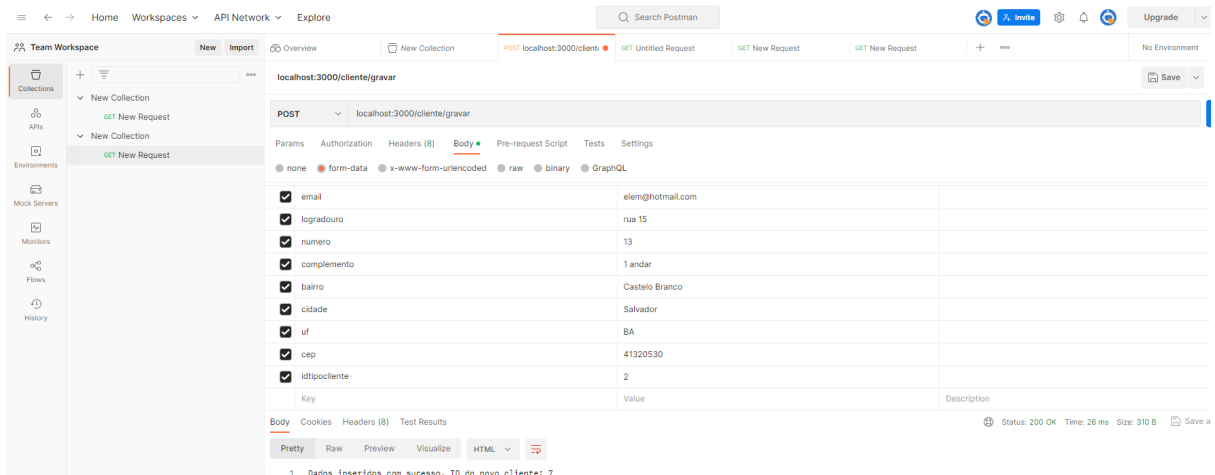


teste no postman:



Para gravar dados de cliente

- URL: localhost:3000/cliente/gravar



Para fazer busca baseado em e-mail:

URL: <http://localhost:3000/cliente/buscar/email/seila@hotmail.com>

← → ↻ ⚠ Não seguro | 192.168.1.68:3000/email?email=seila@hotmail.com

```
[{"nome":"Seilá","telefone":"65321641","email":"seila@hotmail.com","logradouro":"Rua 15","numero":"15","complemento":"1 andar","bairro":"Matadouro","cidade":"Salvador","uf":"BA","cep":"6341545","tipo":"Prime"}]
```

Para fazer busca baseado por id:

URL: <http://192.168.1.68:3000/cliente/buscar/id/6>

← → ↻ ⚠ Não seguro | 192.168.1.68:3000/cliente/6

```
[{"nome":"Seinão","telefone":"412341654","email":"seinão@yahoo.com","logradouro":"Avenida sete ","numero":"32164","complemento":"terreo","bairro":"Paripe","cidade":"Salvador","uf":"BA","cep":"3611321","tipo":"Silver"}]
```

o “6” seria o id buscado

Com isso a atividade foi finalizada e a API entre a meu colega de QA devidamente funcional.