



## Relatório do Projeto II

**Nome: Tassiáni Ritta Freitas**

**Disciplina: Programação Web I**

### Introdução

O presente relatório apresenta o uso dos métodos POST e GET, disponibilizados pelo protocolo HTTP e a realização da manipulação de um Sistema Gerenciador de Banco de Dados (SGBD), na construção de uma aplicação web utilizando a IDE VSCode, a tecnologia NodeJS, o framework express e o *Object-Relational Mapper* (ORM) Sequelize.

O protocolo HTTP (*Hypertext Transfer Protocol*) é um protocolo de comunicação pertencente à camada de aplicação do modelo OSI. Ele é a base para a troca de dados entre cliente e servidor, ou seja, ele é quem permite que o cliente e o servidor se comuniquem entre si.

HTTP é um protocolo que permite a obtenção de recursos, como documentos HTML. É a base de qualquer troca de dados na Web e um protocolo cliente-servidor, o que significa que as requisições são iniciadas pelo destinatário, geralmente pelo navegador da Web. Um documento completo é reconstruído a partir dos diferentes sub-documentos obtidos, como por exemplo texto, descrição do layout, imagens, vídeos, scripts e muito mais. (MOZILLA, 2022)

Esse protocolo faz sua comunicação por meio da troca de mensagens entre clientes e servidores. Os clientes solicitam por meio de requisições (*requests*) os recursos que eles querem ou necessitam do servidor e o servidor envia uma resposta (*responses*) para o cliente, isso acontece por meio do uso dos métodos que esse protocolo possui.

O HTTP possui alguns métodos que servem para iniciar uma comunicação com o servidor são eles: GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS, TRACE e PATCH.

O método GET solicita a representação de um recurso específico. Esse tipo de requisição devem retornar apenas dados. O método POST é utilizado para submeter uma entidade a um recurso específico, frequentemente causando uma

mudança no estado no estado do recurso ou efeitos colaterais no servidor.(MOZILLA, 2022)

## Objetivo

O presente relatório tem por objetivo demonstrar de forma prática os princípios na construção de uma aplicação Web usando o protocolo HTTP, mais específico os métodos GET e POST e realizar a conexão e manipulação com um SGBD utilizando os métodos do HTTP.

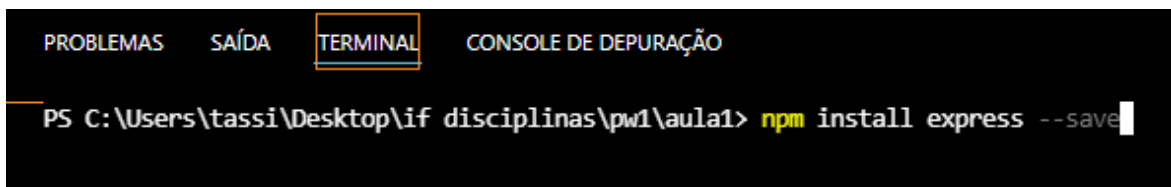
## Atividades desenvolvidas

Para demonstrar de forma prática os princípios na construção de uma aplicação Web algumas etapas foram seguidas:

### 1. Instalação e preparação do ambiente de execução

Nessa etapa foi instalado o VSCode, IDE utilizada para a confecção dos códigos, logo após foi realizada a instalação da tecnologia NodeJS, a qual cria um ambiente que permite a execução dos programas JavaScript como uma aplicação única.

Precisamos instalar o *express*, que é um framework que cria um ambiente para uso do HTTP, usando seus métodos, para isso damos o seguinte comando *npm install express - -save*.



```
PROBLEMAS  SAÍDA  TERMINAL  CONSOLE DE DEPURAÇÃO
PS C:\Users\tassi\Desktop\if disciplinas\pw1\aula1> npm install express --save
```

O *--save* usado como opção no comando serve para incluir o *express* na lista de dependências do arquivo *.json*.

Por meio da instalação do *express* foi possível o uso do motor de *template handlebars*, isto porque o *express* além de oferecer soluções para gerenciar

requisições de diferentes verbos do HTTP em diferentes URLs ele também faz a integração de “*views engines*” para inserir dados nos *templates*.

Além disso, foi realizada a instalação do ORM Sequelize, que tem como função fazer a integração com o banco de dados. Esse ORM serviu para realizar a comunicação com o banco de dados e fazer o mapeamento de dados relacionais (tabelas, colunas e linhas) para os objetos *JavaScript*.

Agora com o ambiente preparado podemos passar para a parte de programação propriamente dita, ou seja, podemos criar a aplicação Web.

## 2. Criação da aplicação Web

Para fazer uso do Express é preciso usar a função *require()*, conforme a linha 1, isso faz com que um módulo do *Express* seja criado para que o servidor seja executado (linha 1 e 2) e a aplicação escute as requisições na porta 8081 conforme definido na constante *port* (linha 4). A instrução da linha 145 faz com que a aplicação escute, propriamente dito, na porta 8081 as requisições, imprimindo no console: que o servidor está ativo ou não e a porta a qual ele está rodando. A instrução na linha é declarada uma variável para ser usada na formatação de datas.

```
1  const express = require('express');
2  const app = express();
3  const moment = require('moment');
4  const port = 8081;

145 app.listen(port, () => { console.log(`servidor rodando na url http://localhost ${port}`) });
146
```

Na linha 5, uma constante *path* foi declarada para criar um caminho para uso de imagens, na linha 6 foi instanciada uma constante para uso do handlebars e na linha 7 instanciou-se um modelo para fazer uso do banco de dados e manipulação nos dados referentes a esse modelo.

```
5  const path = require('path');
6  const handlebars = require('express-handlebars');
7  const Cavalo = require('./models/Cavalo');
```



Nesse trecho de código é configurado o motor de *template* . Com essa engine é possível *linkar* os dados com o template. Ele também nos permitiu particionar a visualização do site, possibilitando reaproveitar código. Essa engine facilitou a criação das páginas HTML e tornou a visualização e envio para essas páginas mais simples e organizado.

```
21 app.engine('handlebars', handlebars.engine({
22   |   defaultLayout: 'main',
23   |   helpers: {
24   |       //formata a data para o formato brasileiro
25   |       formatDate: (date) => {
26   |           |   return moment.date.format('DD/MM/YYYY');
27   |       }
28   |   }
29   |
30   |   }));
31 |
32 app.set('view engine', 'handlebars');
```

Nas linhas 37, 38 e 39 foram chamadas três funções de configuração para fazer ajustes quando necessário: configuração de formato de url, formato *json* e o caminho usado para tratar das imagens no projeto.

```
37 app.use(express.urlencoded({ extend: true }));
38 app.use(express.json());
39 app.use(express.static(path.join(__dirname, '/assets')))
```

Da linha 48 até a linha 62 são apresentadas formas de requisições com o método GET para diferentes caminhos e com diferentes parâmetros:

```
48 | app.get('/', (req, res) => {
49 |   |   Cavalo.findAll().then((cavalos)=>{
50 |   |       |   res.render('home');
51 |   |   });
52 |   |   });
53 |
54 | app.get('/lista', (req, res) => {
55 |   |   Cavalo.findAll().then((cavalos)=>{
56 |   |       |   res.render('lista', {cavalos: cavalos}); //leva para home os cadastros de cavalo
57 |   |   });
58 |   |   });
59 |
60 | app.get('/cadastro', (req, res) => {
61 |   |   res.render('cadastroCavalo');
62 |   |   });
```

A função `app.get()` responde somente às requisições HTTP feitas pelo método GET, o `app` é o próprio HTTP enquanto o `get` é um de seus verbos. Esse método carrega todas as suas informações na URL. Ele passa um caminho e os parâmetros de `req` e `res`, como o método é o GET ele só usa o parâmetro `res` e dentro da função ele chama a seguinte função `res.render('cadastro cavalo')` que trará a resposta de requisição do método, requisição essa feita ao servidor.

Exemplo: Na função da linha 60 a 62, o caminho de chamada usado no browser é `/cadastro` e a função descrita no corpo da função passa o caminho de onde está a página html a ser renderizada, ou seja, a página retornada como resposta. O mesmo acontece com as funções das linhas de 48 a 51 e 54 a 58, porém com algumas diferenças na função das linhas 54 a 58, o caminho de chamada usado no browser é `/lista` e a função descrita no corpo da função faz uma busca e encontra cada cadastro buscando do banco de dados e listando na página html que está a ser renderizada por meio da função `render`, ou seja, a página retornada como resposta.

```
71 | app.post('/cadastroPost', (req, res) => {
72 |     if(!req.body.nome){
73 |         console.log('nome obrigatorio')
74 |         res.redirect('/')
75 |     }
76 | }
77 | Cavallo.create({ //traz infos que estão vindo pelo corpo da requisição (vem do front-end)
78 |     baia: req.body.baia,
79 |     nome: req.body.nome,
80 |     nroRegistro: req.body.nroRegistro,
81 |     pelagen: req.body.pelagem,
82 |     raca: req.body.raca,
83 |     sexo: req.body.sexo,
84 |     dataNasc: req.body.dataNasc,
85 |     nomePai: req.body.nomePai,
86 |     nomeMae: req.body.nomeMae,
87 |     proprietario: req.body.proprietario,
88 |     cpf: req.body.cpf,
89 |     contato: req.body.contato,
90 |     email: req.body.email,
91 | }).then(()=>{
92 |     //caso seja sucesso
93 |     console.log('Informações do cavalo salva com sucesso');
94 |     res.redirect('/lista');
95 | }).catch((erro)=>{
96 |     //caso contrário lança o erro
97 |     console.log('erro ao salvar as informações do cavalo: ${erro}');
98 |     res.redirect('/');
99 | })
100 |
101 | })
```



A função `app.post()` responde somente às requisições HTTP feitas pelo método POST. Esse método carrega todas as suas informações no corpo da requisição. Ela passa um caminho e os parâmetros de `req` e `res`. Esse método usa os dois parâmetros tanto o de requisição quanto o de resposta e dentro da função ele chama a seguinte função `create` que trará as informações do objeto vinda pelo corpo da requisição e dará uma resposta a requisição do método. Essa resposta será a renderização para uma página dependendo se for sucesso ou erro, se for sucesso será renderizado para a página que mostra a lista de cadastros e se for erro renderiza a página da `home`.

Já no trecho de código abaixo é apresentado a função de delete, onde faz uma requisição get e destruirá no banco de dados um objeto de acordo com o id pego através da informação que vem pelo corpo da requisição e caso aconteça um erro uma mensagem é enviada.

```
103 app.get('/deletar/:id',(req,res)=>{
104     Cavalo.destroy({
105         where: {'id': req.params.id}
106     }).then(()=>{
107         res.redirect('/lista');
108     }).catch(err=>{
109         res.send('erro ao excluir informações do cavalo: ${err}');
110     })
111 })
```

No arquivo `db.js` é realizada a conexão com o banco de dados para poder fazer a troca de informações com as páginas e com o HTTP.

```
3 const Sequelize = require('sequelize');
4 //passa as informações para o banco: nome do banco, usuário do banco, password
5 const sequelize = new Sequelize('nome_do_bd', 'usuario', 'senha', {
6     host: 'localhost',
7     dialect: 'postgres',
8 });
9
10 //conexão com o banco
11 sequelize.authenticate().then(() => {
12     console.log("conectado ao banco de dados com sucesso.");
13 }).catch((error) => {
14     console.log('erro ao conectar ao banco: ${error}');
15 });
16
17 //exporta as duas variáveis
18 module.exports = {
19     Sequelize: Sequelize, //orm
20     sequelize: sequelize, //conexão com o banco
21 }
```



Já no arquivo `Cavalo.js` é criado o modelo e os campos da tabela que será criada no banco e os mesmos campos que serão usados para acesso.

```
5 | const db = require('../db/db');
6 | const Cavalo = db.sequelize.define('cavalos', {
7 |   baia: {
8 |     type: db.Sequelize.STRING, //orm faz a criação do campo nome
9 |   },
10 >   nome: { ...
12 |   },
13 >   nroRegistro: { ...
15 |   },
16 >   pelagem: { ...
18 |   },
19 >   raca: { ...
21 |   },
22 >   sexo: { ...
24 |   },
25 >   dataNasc: { ...
27 |   },
28 >   raca: { ...
30 |   },
31 >   nomePai: { ...
33 |   },
34 >   nomeMae: { ...
36 |   },
37 >   proprietario: { ...
39 |   },
40 >   cpf: { ...
42 |   },
43 >   contato: { ...
45 |   },
46 >   email: { ...
48 |   },
49 | });
```

### 3. Iniciando o servidor e testando

Por fim, no terminal devemos executar o `nodemon app.js` e ir no browser e executar os seguintes caminhos para ver os resultados:

-`http://localhost:8081/`

```
PROBLEMAS  SAÍDA  TERMINAL  GITLENS  CONSOLE DE DEPURAÇÃO

"updatedAt" FROM "cavalos" AS "cavalos";
PS C:\Users\tassi\Desktop\if disciplinas\pw1\expressHandlebar> nodemon app.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 app.js`
body-parser deprecated undefined extended: provide extended option app.js:37:17
servidor rodando na url http://localhost 8081
erro ao conectar ao banco: SequelizeConnectionError: password authentication failed for user "usuario"
```

## Conclusões

Diante do exposto é possível concluir que o *Express* junto com o *NodeJS* e o *npm* auxiliam para criação de uma aplicação Web, que usa o protocolo HTTP e seus métodos, embora o HTTP seja cheio de detalhes neste trabalho ele foi usado de maneira simples para compreensão dos seus princípios e funcionalidade. Esse uso se deu por meio da utilização dos métodos GET e POST e por rotas (caminhos). Também foi possível concluir que o método GET utiliza as informações passadas na URL, enquanto que o método POST pega as informações do corpo da requisição. Além disso, também foi possível concluir que o uso de um motor de template facilita a utilização do HTTP e a criação de maneira simples de páginas integradas e integradas com o SGBD por meio do uso de um ORM para a manipulação dos dados referentes às informações e conexões com o banco.

## Referências

\_. Exemplo Hello World:

<https://expressjs.com/pt-br/starter/hello-world.html>. Acesso em: 13fev.2022.

\_. Npm Docs. Disponível em: <https://docs.npmjs.com/>. Acesso em: 13fev.2022.

\_. HTTP: Tutoriais. Disponível em:

<https://developer.mozilla.org/pt-BR/docs/Web/HTTP>. Acesso em: 14fev.2022.

\_. Express Web Framework (NodeJS/Javascript):

[https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express\\_Nodejs](https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express_Nodejs).

Acesso em: 15fev.2022.

\_. NodeJS: <https://nodejs.org/en/>. Acesso em: 08abr.2022.

\_. Sequelize: <https://sequelize.org/>. Acesso em: 09abr.2022.



\_.Bootstrap: <https://getbootstrap.com/>. Acesso em 08abr.2022.