



Relatório do Projeto I

Nome: Tassiáni Ritta Freitas
Disciplina: Programação Web I

Introdução

O presente relatório apresenta o uso dos métodos POST e GET, disponibilizados pelo protocolo HTTP, na construção de uma aplicação web utilizando a IDE VSCode, a tecnologia NodeJS e o framework express.

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.

Atividades desenvolvidas

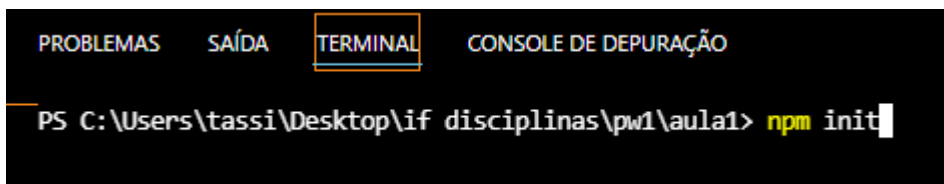
Para demonstrar de forma prática os princípios na construção de uma aplicação Web algumas etapas foram seguidas:

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 foi utilizada para permitir a execução dos programas JavaScript.

Com o NodeJS instalado criou-se um diretório chamado aula1, esse diretório serviu para colocar todos os arquivos do projeto, ou seja, é a pasta raiz do projeto.

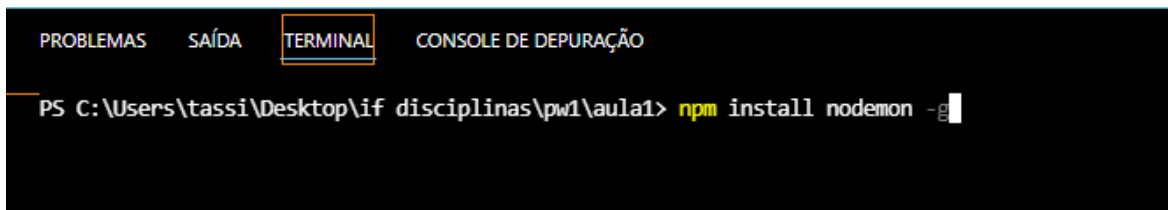
No terminal do VSCode, dentro diretório aula1 os comandos a seguir foram executados:



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

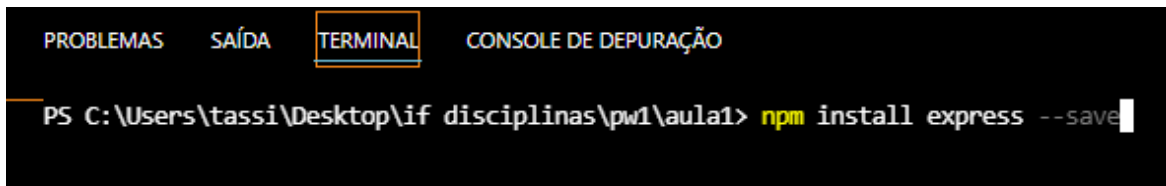
O comando *npm init* cria um arquivo *.json*, chamado *package.json*, este arquivo é onde está toda a configuração do projeto.

Logo após executar o comando anterior é preciso instalar o nodemon, como mostra a imagem abaixo, para facilitar a execução do projeto, pois este comando auxiliará a rodar o projeto inicializando o servidor uma única vez.



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

Agora, 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*.



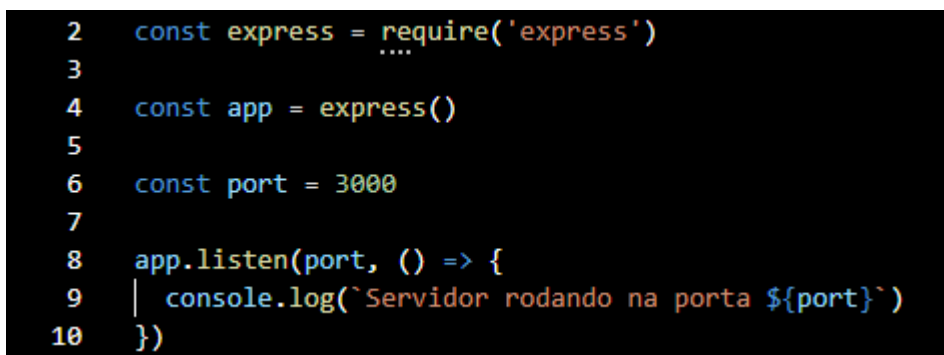
```
PROBLEMAS  SAÍDA  TERMINAL  CONSOLE DE DEPURAÇÃO  
PS C:\Users\tassi\Desktop\if disciplinas\pw1\aula1> npm install express --save
```

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

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

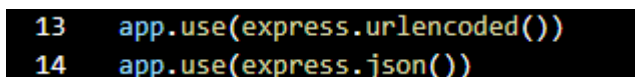
2. Criação da aplicação Web

Para fazer uso do Express é preciso usar a função *require()*, conforme a linha 2, isso faz com que um módulo do *Express* seja criado para que o servidor seja executado (linha 2 e 4) e a aplicação escute as requisições na porta 3000 conforme definido na constante *port* (linha 6). A instrução da linha 8 faz com que a aplicação escute, propriamente dito, na porta 3000 as requisições, imprimindo no console (linha 9) que o servidor ativo e a porta a qual ele está rodando.



```
2  const express = require('express')  
3  
4  const app = express()  
5  
6  const port = 3000  
7  
8  app.listen(port, () => {  
9    | console.log(`Servidor rodando na porta ${port}`)  
10  })
```

Nas linhas 13 e 14 foram chamadas duas funções de configuração para fazer ajustes quando necessário.



```
13  app.use(express.urlencoded())  
14  app.use(express.json())
```

Da linha 17 até a linha 31 são apresentadas formas de requisições com o método GET para diferentes caminhos e com diferentes parâmetros:

```
17 app.get('/', (req, res) => {
18   | res.send('Hello World!')
19   })
20
21 app.get('/html', (req, res) => {
22   | res.sendFile(__dirname+ "/html/index.html")
23   })
24
25 app.get('/formulario', (req, res) => {
26   | res.sendFile(__dirname+ "/html/formulario/cadastro_pessoa.html")
27   })
28
29 app.get('/formulario/cadastro_produtos', (req, res) => {
30   | res.sendFile(__dirname+ "/html/formulario/cadastro_produtos/cadastro_produto.html")
31   })
32
```

A função `app.get()` responde somente às requisições HTTP feitas pelo método GET. Esse método carrega todas as suas informações na URL. Ela 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.sendFile(mensagem ou diretório)` que trará a resposta de requisição do método.

Exemplo: Na função da linha 21 a 23, o caminho de chamada usado no browser é `/html` e a função descrita no corpo da função passa o caminho de onde está a página html retornada como resposta. O mesmo acontece com as funções das linhas de 25 a 27, porém o caminho usado no browser é `/formulario` e a resposta é o caminho descrito como parâmetro da linha 26. O mesmo acontece da linha 29 a 31 só que com caminhos diferentes.

```
33 app.post("/produto_cadastrado",(req, res)=>{// não consigo acessar pela url
34   | res.send(`Nome do produto: ${req.body.nome} Quantidade de produto: ${req.body.qntProduto} Valor do produto:
35   | //res.send("FORMULARIO RECEBIDO")
36   })
37
38 app.post("/listar_dados_formulario",(req, res)=>{// não consigo acessar pela url
39   | res.send(`Nome : ${req.body.nome} ${req.body.sobreNome} \nIdade: ${req.body.Idade} \nCPF: ${req.body.cpf} \nTe
40   | //res.send("FORMULARIO RECEBIDO")
41   })
```

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 *res.sendFile(mensagem informada na tela)* que trará a resposta de requisição do método. Essa mensagem disponibilizada na tela do browser se dá por meio de do caminho descrito dentro do arquivo HTML do formulário conforme imagem abaixo:

```
9   <body>
10  <form action="/listar_dados_formulario" method="POST">
```

3. Iniciando o servidor e testando

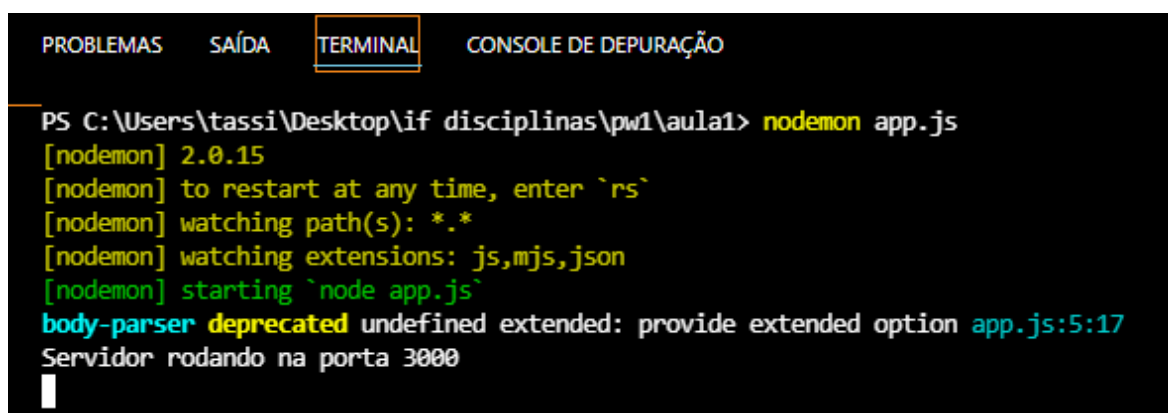
Por fim, no terminal devemos executar o *nodemon app.js* e ir no browser e executar os seguintes caminhos para ver os resultados:

-<http://localhost:3000/>

-<http://localhost:3000/html>

-<http://localhost:3000/formulario>, neste caso após submeter o formulário o método post retornará a resposta.

-http://localhost:3000/formulario/cadastro_produtos, neste caso após submeter o formulário o método post retornará a resposta.



```
PROBLEMAS  SAÍDA  TERMINAL  CONSOLE DE DEPURAÇÃO

PS C:\Users\tassi\Desktop\if disciplinas\pw1\aula1> 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:5:17
Servidor rodando na porta 3000
```

Conclusões

Diante do exposto é possível concluir que 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.

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.

Acesso em: 15fev.2022.