Desenvolvimento de Software para Web

UFC - Universidade Federal do Ceará

André Meireles
andre@crateus.ufc.br

Desenvolvimento de Software para Web

PRÁTICA em Fundamentos da Web E Servidores HTTP

UFC - Universidade Federal do Ceará

André Meireles
andre@crateus.ufc.br

Web Server e HTTP Server

- **Servidor web** (web server) pode referir ao hardware ou ao software, ou ambos trabalhando juntos.
  - Referente ao **hardware**: é um computador que armazena arquivos que compõem os sites (ex: documentos HTML, imagens, CSS, e scripts) e os entrega para o dispositivo do usuário final.
  - Referente ao **software**: inclui diversos componentes que controlam como os usuários acessam os arquivos hospedados (armazenados para disponibilização), no mínimo um servidor HTTP.
- **Servidor HTTP** é um software que compreende URLs (endereços web) e HTTP.

Servindo conteúdo estático

- **Web servers mais conhecidos**
  - **NGinx - https://www.nginx.com**
    - Nginx (pronounced "engine-x") is an open source reverse proxy server for HTTP, HTTPS, SMTP, POP3, and IMAP protocols, as well as a load balancer, HTTP cache, and a web server (origin server).
  - **Apache HTTP Server - https://httpd.apache.org**
    - The Number One HTTP Server On The Internet… provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards

Servindo conteúdo estático

Pré-requisitos para esta prática

- Computador com Windows ou Linux
- **Docker** (nesse caso, se estiver usando Windows, terá que escolher entre Hyper-V e WSL)
  - Verifique se o Docker está executando:
    - `# docker ps`
  - Se necessário, instale o Docker

Servindo conteúdo estático

- Usando Docker, execute um container utilizando uma imagem do NGinx

```
# docker run --name ufc-web-nginx -p 8080:80 nginx
```

- Abra **http://localhost:8080** no seu navegador
- Verifique o log do NGinx no terminal

- Exercício:
  - Acesse esta mesma página utilizando o programa CURL no Linux
  - Qual porta está sendo utilizada?
  - Agora, salve o conteúdo em um arquivo e depois abra o arquivo pelo sistema operacional

Servindo conteúdo estático

Por padrão, o conteúdo servido pelo NGinx é o conteúdo contido no diretório: /usr/share/nginx/html (do container)

- Você pode testar acessando o container e editando o arquivo
  - Copie o index.html do container para a sua pasta

```
# docker cp ufc-web-nginx:/usr/share/nginx/html/index.html
```

  - Edite e copie de volta para o mesmo local no container

```
# docker cp index.html ufc-web-nginx:/usr/share/nginx/html/index.htm
```

- Para acessar o bash do container: `docker exec -it ufc-web-nginx bash`

ALTERNATIVA (se você tiver acesso ROOT): Você também pode executar um container com um volume associado a um diretório do host e manipular o conteúdo diretamente no host (pode necessitar acesso root para editar a pasta no host):

```
# docker run --name ufc-web-nginx -p 80:80 -v \ full dir path on host/:/usr/share/nginx/html nginx
```

Servindo conteúdo estático

Exercício:

- Altere o index.html do seu servidor NGinx em execução no container
- Adicione um arquivo ao seu servidor que seja acessível através da URL http://localhost:8080/about/about.html
  - Dica: para acessar o terminal do container: `docker exec -it ufc-web-nginx bash`
  - Dica: crie um diretório chamado *about* no diretório de conteúdo do NGinx
- Sirva um arquivo de imagem qualquer (JPG ou PNG) no seu servidor NGinx e acesse usando o navegador
- Acesse o servidor NGinx que esteja sendo executado em outro computador do laboratório
  - Dica: pergunte o IP do computador do seu colega (comando: `ip a`)

Avaliação do Lab

Servindo conteúdo estático

- Executar servidor NGinx com Docker (2 pontos)
- Alterar conteúdo do index.html (2 pontos)
- Servir conteúdo em http://localhost:8080/about/about.html (3 pontos)
- Servir imagem (3 pontos)

Servindo conteúdo dinâmico

Pré-requisitos - Instalando Node com NVM:

Se o Node não estiver instalado, execute os comandos abaixo no seu terminal Linux para instalar o NVM (Node Version Manager):

```
# curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash
```

ABRA OUTRO TERMINAL e execute:

```
# nvm -v
```

```
0.39.0
```

Se a versão do NVM for apresentada, instale a versão mais recente do node executando:

```
# nvm install 16.14.0
```

Ao fim da instalação, verifique se deu certo:

```
# node -v
```

```
v16.14.0
```

```
# npm -v
```

```
8.3.1
```

Servindo conteúdo dinâmico

Vamos executar um servidor HTTP utilizando Node com o framework Express

Pré-requisitos:

- É necessário ter Node e NPM instalados no seu PC, para verificar:

```
# node -v
# npm -v
```

1. Crie um diretório para o projeto e vá para ele

```
# mkdir projeto-express
# cd projeto-express
```

2. Inicialize um projeto Node: `npm init` e confirme todas as perguntas no terminal
3. Instale o pacote Express: `npm install express --save`

Servindo conteúdo dinâmico

4. Crie um arquivo chamado **index.js** na raiz do projeto e adicione o seguinte conteúdo:

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

5. Execute `node .` na raiz do projeto e acesse http://localhost:3000 no seu navegador

Servindo conteúdo dinâmico

Vamos testar o método POST

- Os principais formatos para envios dos dados utilizando o BODY em um POST são **JSON** e **URL Encoded**
- Vamos entender as diferenças entre eles. Adicione o seguinte código na linha 4:

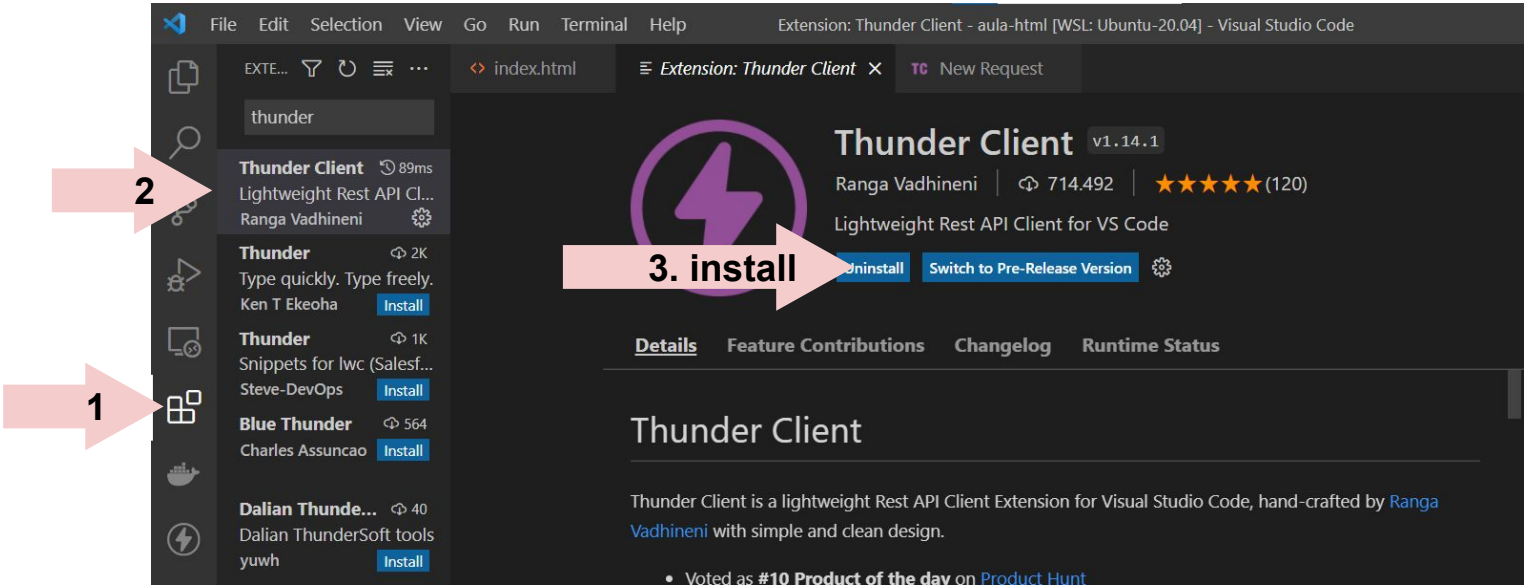
```
app.use(express.json())
app.use(express.urlencoded({ extended: true }))

app.post('/mirror', function (req, res, next) {
  console.log(req.body)
  res.send(req.body)
})
```

No terminal, pare a execução (CTRL+C) e rexecute o projeto.

Servindo conteúdo dinâmico

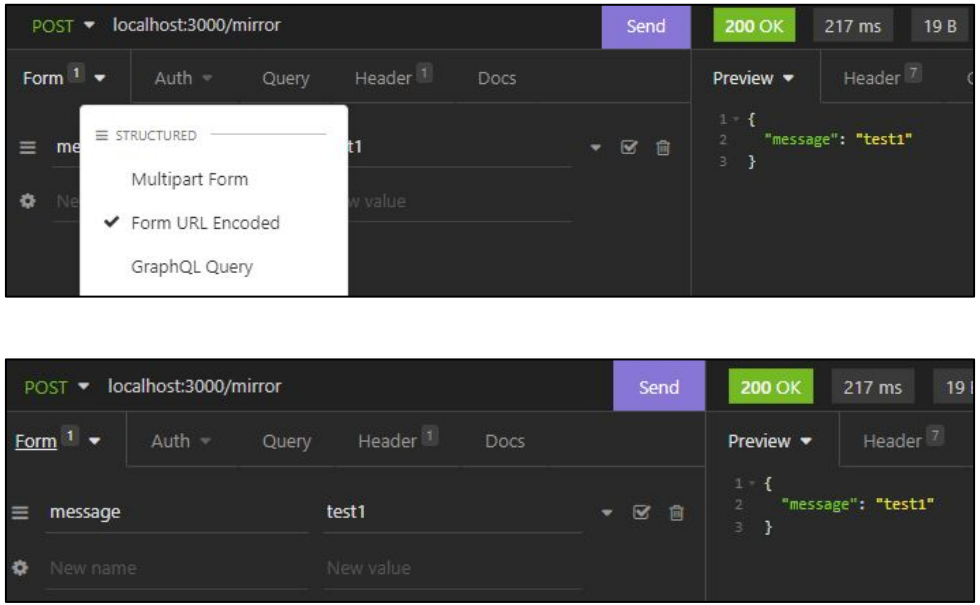
Agora utilize o software Postman ou Insomnia para realizar requisições POST para a nova rota criada

Caso você não tenha permissão para a instalação desses softwares, você pode usar a extensão Thunder Client, no VS Code, através do instalador de extensões, veja:



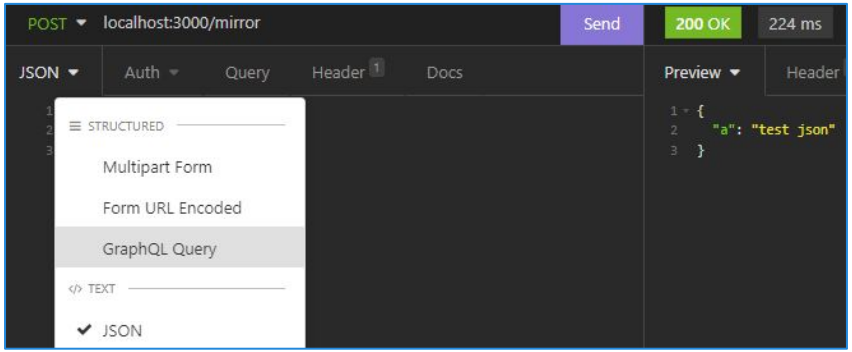
Servindo conteúdo dinâmico

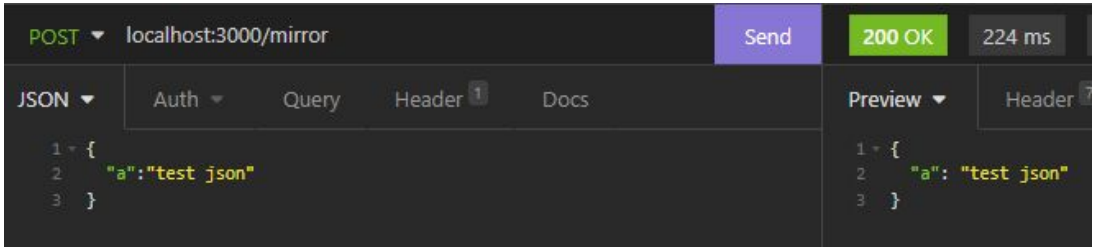
Exemplo de requisição com dados **URL Encoded** no Insomnia:



Servindo conteúdo dinâmico

Exemplo de requisição com dados **JSON** no Insomnia:





Servindo conteúdo dinâmico

Também é possível enviar dados através da URL

- Para isso, é possível utilizar **Parâmetros de Consulta** (Query Params)
- Altere a rota GET “/” com o seguinte código:

```
app.get('/querytojson', (req, res) => {
  console.log(req.query);
  res.send(req.query);
})
```

Reinicie o servidor.

Agora acesse o seguinte link pelo navegador http://localhost:3000/querytojson?name=Mary&role=manager

Altere a requisição modificando os parâmetros e veja o que acontece

Servindo conteúdo dinâmico

- Também é possível utilizar **Parâmetros de Caminho** (Path Params ou Request Params).
- Adicione uma a rota GET com o seguinte código:

```
app.get('/paramtojson/name/:name/role/:role', (req, res) => {
  console.log(req.params);
  res.send(req.params);
})
```

Reinicie o servidor.

Agora acesse o seguinte link pelo navegador http://localhost:3000/paramtojson/**name**/Mary/**role**/Manager

Altere a requisição modificando os parâmetros e veja o que acontece

Exercício

Altere a sua aplicação Express (Node) e implemente os seguintes requisitos:

1. Crie um array chamado `animals` com os seguintes objetos:

```
{animal:”DOG”, name:”Pluto”}
{animal:”CAT”, name:”Hercules”}
{animal:”BIRD”, name:”Tweety”}
{animal:”DOG”, name:”Spiff”}
{animal:”CAT”, name:”Tom”}
{animal:”BIRD”, name:”Road Runner”}
```

2. Crie uma recurso `GET` acessível pela rota `/animal`
3. Implemente uma busca por animal e por nome utilizando parâmetros de consulta (Query Params)

Exemplo:

```
GET - http://localhost:3000/animal?name=e
Response: 200 - [{animal=”CAT”, name=”Hercules”}, {animal=”BIRD”, name=”Tweety”},
{animal=”BIRD”, name=”Road Runner”}]

GET - http://localhost:3000/animal?animal=DOG
Response: 200 - [{animal=”DOG”, name=”Pluto”}, {animal=”DOG”, name=”Spiff”}]
```

Exercício

Altere a sua aplicação Express (Node) e implemente os seguintes requisitos:

1. Crie um array chamado `animals` com os seguintes objetos:

```
{animal:”DOG”, name:”Pluto”}
{animal:”CAT”, name:”Hercules”}
{animal:”BIRD”, name:”Tweety”}
{animal:”DOG”, name:”Spiff”}
{animal:”CAT”, name:”Tom”}
{animal:”BIRD”, name:”Road Runner”}
```

2. Crie uma recurso `GET` acessível pela rota `/animal`
3. Implemente uma busca por animal e por nome utilizando parâmetros de consulta (Query Params)

Exemplo:

```
GET - http://localhost:3000/animal?name=e
Response: 200 - [{animal=”CAT”, name=”Hercules”}, {animal=”BIRD”, name=”Tweety”},
{animal=”BIRD”, name=”Road Runner”}]

GET - http://localhost:3000/animal?animal=DOG
Response: 200 - [{animal=”DOG”, name=”Pluto”}, {animal=”DOG”, name=”Spiff”}]
```

Exercício

```
animals = [
  {animal: "DOG", name:"Pluto"}
  ,{animal:"CAT", name:"Hercules"}
  ,{animal:"BIRD", name:"Tweety"}
  ,{animal:"DOG", name:"Spiff"}
  ,{animal:"CAT", name:"Tom"}
  ,{animal:"BIRD", name:"Road Runner"}
]


...
```

```
app.get('/animal', (req, res) => {
  if(req.query.name){
    busca = req.query.name
    respArray = []
    for (let i = 0; i < animals.length; i++) {
      const a = animals[i];
      if(a.name.includes(busca)){
        respArray.push(a)
      }
    }
    res.send(respArray);
  }else{
    res.send(animals);
  }
})
```

Avaliação do Lab

Servindo conteúdo dinâmico

- Executar servidor HTTP com Node + Express (1 ponto)
- Implementar POST na rota `/mirror` (1 ponto)
- Implementar GET com Parâmetros de Consulta (2 ponto)
- Implementar GET com Parâmetros de Caminho (2 ponto)
- Implementar busca por animais pelo campo `name` (2 ponto)
- Implementar busca por animais pelo campo `animal` (2 ponto)