# Aula 3

| | |
|---|---|
| Created | @October 28, 2023 11:22 AM |
| Data | @January 12, 2024 |
| Status | Não iniciado |
| Tipo | Aula |

## Mergulhando no Express (views)

Adicionando rotas que processam diferentes tipos de dados

- text/plain
- text/html
- application/json

Aqui está um exemplo de como você pode criar rotas para processar diferentes tipos de dados:

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

// Rota para processar dados em text/plain
app.get('/plaintext', (req, res) => {
  res.type('text/plain');
  res.send('Isso é um exemplo de texto simples (text/plai
n).');
});
```

```
// Rota para processar dados em text/html
app.get('/html', (req, res) => {
  res.type('text/html');
  res.send('<h1>Isso é um exemplo de HTML (text/html).</h1
>');
});

// Rota para processar dados em application/json
app.get('/json', (req, res) => {
  res.type('application/json');
  res.json({ mensagem: 'Isso é um exemplo de JSON (applicatio
n/json).' });
});

// Inicia o servidor na porta especificada
app.listen(port, () => {
  console.log(`Servidor rodando em http://localhost:${port}
`);
});
```

Neste exemplo:

- A rota `/plaintext` responde com dados em formato `text/plain`.
- A rota `/html` responde com dados em formato `text/html`.
- A rota `/json` responde com dados em formato `application/json`.

Essas rotas usam o método `res.type()` para definir o tipo de conteúdo da resposta. O método `res.send()` ou `res.json()` é usado para enviar os dados de resposta ao cliente.

## Cabeçalhos HTTP no Express.js:

**Teórico:**

- **Definição:** O Express.js fornece métodos para manipular cabeçalhos HTTP em solicitações (request) e respostas (response).
- **Função:** Permite configurar e acessar informações adicionais nas requisições e respostas.

**Prático:**

- **Configurando Cabeçalhos (Request):**

```
app.use((req, res, next) => {
  req.headers.customheader = 'Valor Customizado';
  next();
});
```

- **Configurando Cabeçalhos (Response):**

```
app.get('/rota', (req, res) => {
  res.set('Custom-Header', 'Valor Customizado');
  res.send('Conteúdo da resposta');
});
```

## Classes de Resposta (HTTP Status) no Express.js:

**Teórico:**

- **Definição:** O Express.js utiliza códigos de status HTTP para indicar o resultado da requisição.
- **Função:** Indica ao cliente se a solicitação foi bem-sucedida, se há redirecionamento ou se ocorreu algum erro.

**Prático:**

- **Definindo um Código de Status:**

```
javascriptCopy code
app.get('/rota', (req, res) => {
  res.status(404).send('Recurso não encontrado');
});
```

## Tipos de Dados (MIME Types) no Express.js:

**Teórico:**

- **Definição:** O Express.js permite especificar os tipos de dados MIME nas respostas.
- **Função:** Define como o conteúdo da resposta deve ser interpretado pelo navegador ou cliente.

**Prático:**

- **Configurando o Tipo de Dados (Response):**

```
javascriptCopy code
app.get('/rota', (req, res) => {
  res.type('application/json');
  res.json({ mensagem: 'Conteúdo em JSON' });
});
```

O Express.js facilita a manipulação desses conceitos por meio de métodos convenientes fornecidos por seus objetos `req` e `res`. Incorporando esses conceitos, você pode controlar eficientemente o fluxo e o conteúdo das suas solicitações e respostas em aplicativos web construídos com Express.js.

## Conhecendo as principais classes de resposta do HTTP:

1. **1xx - Informational:**

- **Exemplo:** 100 Continue
- **Significado:** A solicitação foi recebida, o servidor está processando ou aguardando mais informações.

2. **2xx - Success:**
   - **Exemplo:** 200 OK
   - **Significado:** A solicitação foi bem-sucedida. O servidor entendeu e aceitou a solicitação.
3. **3xx - Redirection:**
   - **Exemplo:** 301 Moved Permanently
   - **Significado:** A solicitação foi redirecionada. O cliente deve seguir o novo local.
4. **4xx - Client Error:**
   - **Exemplo:** 404 Not Found
   - **Significado:** O cliente fez uma solicitação inválida ou não encontrou o recurso solicitado.
5. **5xx - Server Error:**
   - **Exemplo:** 500 Internal Server Error
   - **Significado:** O servidor encontrou uma situação inesperada e não pôde completar a solicitação.

É importante observar que essas classes fornecem uma maneira rápida de entender o tipo de resposta que um servidor está enviando em uma situação particular. Os códigos de status dentro dessas classes fornecem informações mais detalhadas sobre o resultado da solicitação. Por exemplo, um código 404 dentro da classe 4xx indica que o recurso solicitado não foi encontrado.

## Trabalhando com Server Side Rendering (SSR) no Express

O Server-Side Rendering (SSR) envolve a renderização de páginas no servidor antes de enviá-las para o cliente. No contexto do Express.js, o SSR pode ser implementado

para renderizar páginas no servidor antes de enviar as respostas ao navegador. Abaixo, um exemplo básico de como configurar o SSR com o Express.js usando um mecanismo de modelo como o EJS.

1. **Instalação de Dependências:**
   Certifique-se de ter o Express.js e um mecanismo de modelo (como EJS) instalados. Se ainda não tiver, você pode instalá-los com o seguinte comando:

```
npm install express ejs
```

2. **Configuração do Express.js:**
   Crie um arquivo `server.js` e configure o Express para usar o EJS como mecanismo de modelo. Adicione rotas para renderizar páginas no servidor.

```
const express = require('express');
const path = require('path');

const app = express();
const port = 3000;

// Configuração do mecanismo de modelo EJS
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));

// Rota para renderizar uma página no servidor
app.get('/', (req, res) => {
  res.render('index', { title: 'Página Renderizada no Serv
idor' });
});

// Inicia o servidor
app.listen(port, () => {
  console.log(`Servidor rodando em http://localhost:${por
```

```
t}`);
});
```

3. **Criação de um Template EJS:**
   Crie um diretório
   `views` no mesmo nível do seu arquivo `server.js` e adicione um arquivo `index.ejs` dentro dele.

```
<!-- views/index.ejs -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initi
al-scale=1.0">
  <title><%= title %></title>
</head>
<body>
  <h1><%= title %></h1>
</body>
</html>
```

   Este é um exemplo simples de um arquivo EJS. O `<%= title %>` é um espaço reservado que será preenchido dinamicamente pelo Express.js.

4. **Execução do Servidor:**
   Execute o servidor usando o comando:

```
node server.js
```

Visite `http://localhost:3000` no seu navegador para ver a página renderizada no servidor.

Prático:

- Adicionar uma rota que redirecione para outra, utilizando o método res.redirect();
- Adicionar uma rota que responda uma página HTML, utilizando o método res.send();
- Adicionar uma rota que responda uma página HTML, utilizando o método res.sendFile();
- Adicionar uma rota que responda uma página HTML, utilizando o método res.render();
- Instalando e configurando sua primeira view engine (PUG);