Aula 05: Desenvolvimento Web - Requisições GET e POST com Express

Requisições HTTP com Express: GET e POST para Formulários de Cadastro

Introdução às Requisições HTTP

No contexto do desenvolvimento web, **requisições HTTP** são a forma como os clientes (normalmente navegadores) comunicam-se com os servidores. Cada requisição HTTP possui um **método**, que indica a ação desejada:

- **GET**: Utilizado para solicitar dados do servidor. Não modifica os dados no servidor e geralmente é usado para buscar informações.
- **POST**: Utilizado para enviar dados ao servidor. Frequentemente utilizado em formulários para submeter dados e, muitas vezes, para criar ou atualizar recursos no servidor.

Configuração Básica do Express

Para trabalhar com requisições HTTP usando o Express, você primeiro precisa configurar um servidor Express. Aqui está um exemplo básico de configuração:

```
const express = require('express')
const bodyParser = require('body-parser')
const app = express()

// Configurando bodyParser para manipular dados do formulár
io
app.use(bodyParser.urlencoded({ extended: true }))

const PORT = 3000
```

```
app.listen(PORT, () => {
    console.log(`Servidor rodando na porta ${PORT}`)
})
```

Nesse exemplo, utilizamos o `bodyParser` para tratar os dados do formulário enviados via método `POST`.

O que é o `body-parser` e como utilizá-lo

Introdução ao `body-parser`

No desenvolvimento de aplicações web com Node.js e Express, frequentemente precisamos processar dados enviados por meio de formulários ou APIs. Quando um cliente envia uma requisição HTTP, os dados podem vir no corpo da requisição, e esses dados precisam ser acessados pelo servidor para serem processados.

É aí que entra o `body-parser`, um middleware essencial no Express. Ele é usado para analisar o corpo da requisição e torná-lo disponível em `req.body`. O `body-parser` suporta vários formatos de dados, como JSON e dados de formulário (URL-encoded).

Principais Funcionalidades

O `body-parser` facilita o acesso aos dados da requisição de duas formas principais:

1. **URL-encoded**: Ideal para dados enviados por formulários HTML. Os dados são codificados como uma string na URL, como `name=John&age=30`.
2. **JSON**: Comumente usado em APIs RESTful, onde os dados são enviados em formato JSON.

Como Instalar o `body-parser`

Antes de utilizar o `body-parser`, você precisa instalá-lo no seu projeto. No entanto, desde a versão 4.16.0, o Express já inclui funcionalidades do `body-parser` embutidas, então você pode optar por não instalá-lo separadamente, a menos que esteja utilizando uma versão mais antiga do Express ou precise de funcionalidades específicas.

Para instalar o `body-parser`:

```
npm install body-parser
```

Como Utilizar o `body-parser`

Uma vez instalado, você pode integrar o `body-parser` ao seu aplicativo Express. Abaixo estão exemplos de como configurar o `body-parser` para processar dados no formato URL-encoded e JSON.

1. URL-encoded

O formato URL-encoded é o formato padrão de dados enviados por formulários HTML. Para que o Express entenda esses dados, você precisa configurar o `body-parser` como middleware:

```
const express = require('express')
const bodyParser = require('body-parser')
const app = express()

// Configurando o body-parser para tratar dados de formulár
io
app.use(bodyParser.urlencoded({ extended: true }))

app.post('/cadastro', (req, res) => {
    const nome = req.body.nome
    const email = req.body.email
    res.send(`Nome: ${nome}, Email: ${email}`)
})

app.listen(3000, () => {
    console.log('Servidor rodando na porta 3000')
})
```

- `extended: true`: Permite que objetos complexos sejam codificados na URL. Se `false`, apenas strings ou arrays podem ser codificados.

2. JSON

Se você está trabalhando com APIs ou precisa processar dados enviados como JSON, você deve configurar o `body-parser` para tratar esses dados:

```
app.use(bodyParser.json())

app.post('/api/data', (req, res) => {
    const data = req.body
    res.json({ receivedData: data })
});
```

Considerações Finais

- **Depreciação**: Em projetos mais novos, você pode usar as funções embutidas do Express como `express.urlencoded()` e `express.json()` ao invés de `body-parser`.
- **Erros Comuns**: Um erro comum é esquecer de configurar o `body-parser`, o que faz com que `req.body` seja `undefined` ao tentar acessar os dados enviados.

Utilizar o `body-parser` ou suas alternativas nativas no Express é essencial para lidar corretamente com os dados enviados em requisições HTTP, garantindo que sua aplicação seja capaz de processar e responder às informações submetidas pelos usuários.

Lidando com Requisições GET

O método **GET** é utilizado para solicitar dados do servidor. Vamos criar uma rota que responde a uma solicitação GET mostrando um formulário de cadastro.

Exemplo de Rota GET

```
app.get('/cadastro', (req, res) => {
    res.send(`
        <form action="/cadastro" method="POST">
            <label for="nome">Nome:</label>
            <input type="text" id="nome" name="nome">
            <br>
```

```
            <label for="email">Email:</label>
            <input type="email" id="email" name="email">
            <br>
            <button type="submit">Cadastrar</button>
        </form>
    `)
})
```

Explicação

- **Rota** `/cadastro`: Quando o usuário acessa essa rota via GET, o servidor responde com um formulário HTML simples.
- **Formulário HTML**: Este formulário será enviado via método `POST` para a mesma rota (`/cadastro`), onde lidaremos com a submissão.

Lidando com Requisições POST

O método **POST** é utilizado para enviar dados ao servidor. Vamos criar uma rota que recebe os dados do formulário enviado via POST e os exibe ao usuário.

Exemplo de Rota POST

```
app.post('/cadastro', (req, res) => {
    const nome = req.body.nome
    const email = req.body.email
    res.send(`Obrigado por se cadastrar, ${nome}! Confirmar
emos o cadastro através do email: ${email}.`)
})
```

Explicação

- **Rota** `/cadastro` **(POST)**: Quando o usuário submete o formulário, os dados são enviados ao servidor via método POST.
- `req.body`: Utilizamos `req.body` para acessar os dados enviados no formulário. Aqui, `nome` e `email` são os campos que foram definidos no formulário HTML.

- **Resposta**: O servidor responde com uma mensagem personalizada, confirmando o cadastro.

Exemplo Completo

Aqui está o código completo, integrando as requisições GET e POST para um formulário de cadastro:

```
const express = require('express')
const bodyParser = require('body-parser')
const app = express()

app.use(bodyParser.urlencoded({ extended: true }))

// Rota GET para exibir o formulário
app.get('/cadastro', (req, res) => {
    res.send(`
        <form action="/cadastro" method="POST">
            <label for="nome">Nome:</label>
            <input type="text" id="nome" name="nome">
            <br>
            <label for="email">Email:</label>
            <input type="email" id="email" name="email">
            <br>
            <button type="submit">Cadastrar</button>
        </form>
    `)
})

// Rota POST para processar o formulário
app.post('/cadastro', (req, res) => {
    const nome = req.body.nome
    const email = req.body.email
    res.send(`Obrigado por se cadastrar, ${nome}! Confirmar
emos o cadastro através do email: ${email}.`)
})

const PORT = 3000;
app.listen(PORT, () => {
```

```
    console.log(`Servidor rodando na porta ${PORT}`)
})
```

Testando o Código

1. **Inicie o servidor**: Execute o código com `node index.js`.
2. **Acesse o formulário**: Abra um navegador e vá até `http://localhost:3000/cadastro`.
3. **Preencha e envie o formulário**: Após submeter o formulário, você verá a confirmação do cadastro.

Considerações Finais

- **Segurança**: Em aplicações reais, é importante validar e sanitizar os dados recebidos para evitar injeções de código malicioso (por exemplo, SQL Injection). Veremos mais sobre isso nas próximas aulas.
- **Validação**: É recomendado implementar validações nos campos do formulário para garantir que os dados enviados são válidos e completos. Veremos mais sobre isso nas próximas aulas.

app.set('view engine', 'ejs')

app.set('views', './views')

res.render('')