Relatório: Uso do Express e Proxy com React

1. O que é o Express?

Express é um framework minimalista para Node.js, projetado para simplificar o desenvolvimento de aplicações web e APIs. Ele oferece uma abordagem eficiente para gerenciar rotas, middlewares e respostas HTTP.

Principais Funcionalidades:

- **Roteamento Simples**: Permite configurar rotas de forma intuitiva para diferentes métodos HTTP (GET, POST, PUT, DELETE).
- **Middlewares**: Funções que processam requisições e respostas, essenciais para autenticação, tratamento de dados e controle de erros.
- **APIs RESTful**: Facilita a criação de APIs estruturadas que retornam respostas em formato JSON.
- **Modularidade**: Integração com várias bibliotecas para adicionar funcionalidades, como autenticação.
- **Renderização de Páginas**: Suporte a motores de template, como EJS e Pug, para geração dinâmica de páginas HTML.

2. Diferenças entre Node.js Puro e o Uso do Express

Node.js Puro: Permite criar servidores e gerenciar requisições HTTP, mas exige que o desenvolvedor implemente a lógica de roteamento manualmente.

Express: Abstrai e simplifica o processo de roteamento e gestão de requisições, oferecendo uma solução mais estruturada.

Diferenças Principais:

- **Roteamento**: Com Express, as rotas podem ser definidas diretamente, sem a necessidade de verificações manuais.
- **Manipulação de Requisições**: Simplifica o tratamento de cabeçalhos e respostas HTTP.
- **Flexibilidade**: Modularidade para adicionar middlewares de forma eficiente.

Exemplo de Rotas com Express:

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

app.get('/rota-exemplo', (req, res) => {
  res.send('Rota definida com Express');
});

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

3. Middlewares no Express

Middlewares são funções que atuam durante o ciclo de vida de uma requisição, permitindo modificar dados ou encerrar o ciclo. Eles são essenciais para autenticação, registro de logs e tratamento de erros.

Exemplo de Middleware:

```
app.use((req, res, next) => {
  console.log('Middleware executado');
  next(); // Passa para o próximo middleware ou rota
});
```

4. O que é um Proxy e Como Implementá-lo com Express?

Um proxy atua como intermediário entre o cliente e o servidor, encaminhando requisições. Ele é frequentemente usado para redirecionar chamadas entre servidores ou ocultar o servidor real.

Implementação de Proxy no Express:

1. **Instale a dependência:**

npm install http-proxy-middleware

2. **Configure o proxy no servidor Express:**

```
const { createProxyMiddleware } = require('http-proxy-middleware');

app.use('/api', createProxyMiddleware({
  target: 'http://backend-server:5000',
  changeOrigin: true,
}));
```

5. Exemplo Prático de Proxy em React

Em projetos React, proxies são utilizados para redirecionar chamadas da interface para uma API backend, evitando problemas relacionados ao CORS.

Passos para Configurar um Proxy no React:

1. **Instale o middleware de proxy:**

npm install http-proxy-middleware

2. **Crie o arquivo setupProxy.js na pasta src:**

```
const { createProxyMiddleware } = require('http-proxy-middleware');

module.exports = function(app) {
  app.use(
    '/api',
    createProxyMiddleware({
      target: 'http://localhost:5000',
      changeOrigin: true,
    })
  );
};
```

3. **Configure a rota no servidor Express:**

```
app.get('/api/dados', (req, res) => {
  res.json({ mensagem: 'Dados do servidor' });
});
```

4. **No React, faça uma requisição para o endpoint:**

```
fetch('/api/dados')
  .then(response => response.json())
  .then(data => console.log(data));
```

6. Boas Práticas e Estrutura de Código

- **Divisão de Tarefas**: Separe rotas, middlewares e lógica de negócios em módulos distintos.
- **Documentação Clara**: Comente o código explicando as funcionalidades de cada parte.
- **Organização Modular**: Use pastas para rotas, middlewares e configurações de proxy.

Exemplo de Estrutura de Diretórios:

```
projeto/
├── src/
│   ├── middlewares/
│   ├── routes/
│   ├── setupProxy.js
│   ├── server.js
```

Conclusão

Express é uma ferramenta poderosa e flexível para criar servidores e APIs no Node.js. O uso de proxies facilita a comunicação entre o frontend e o backend, tornando o desenvolvimento mais eficiente e organizado. Com uma estrutura modular e boas práticas, é possível criar aplicações escaláveis e de fácil manutenção.