Express Conceitos e Aplicação

**1. Express e suas principais funcionalidades:**

Express é o framework Node mais popular e a biblioteca subjacente para uma série de outros frameworks do Node. O Express oferece soluções para:

* Gerenciar requisições de diferentes verbos HTTP em diferentes URLs.
* Integrar "view engines" para inserir dados nos templates.
* Definir as configurações comuns da aplicação web, como a porta a ser usada para conexão e a localização dos modelos que são usados para renderizar a resposta.
* Adicionar novos processos de requisição por meio de "middleware" em qualquer ponto da "fila" de requisições.

O *Express* é bastante minimalista, no entanto, os desenvolvedores têm liberdade para criar pacotes de middleware específicos com o objetivo de resolver problemas específicos que surgem no desenvolvimento de uma aplicação. Há bibliotecas para trabalhar com cookies, sessões, login de usuários, parâmetros de URL, dados em requisições POST, cabeçalho de segurança e tantos outros.

Express é um framework não opinativo. Você pode inserir qualquer middleware que você goste no manuseio das solicitações em quase qualquer ordem que desejar. Pode estruturar o aplicativo em um arquivo ou em vários, usar qualquer estrutura de pastas dentro do diretório principal, etc.

Node foi inicialmente lançado em 2009, mas naquela época apenas para Linux. O gerenciador de pacotes NPM veio no ano seguinte, 2010, e o suporte nativo do Windows chegou em 2012. A versão atual do Long Term Support (LTS) é o Node v8.9.3, enquanto a versão mais recente é o Node 9.

O Express foi lançado em novembro de 2010 e atualmente está na versão 4.16 da API.

**2. Node.js puro e o uso de Express**

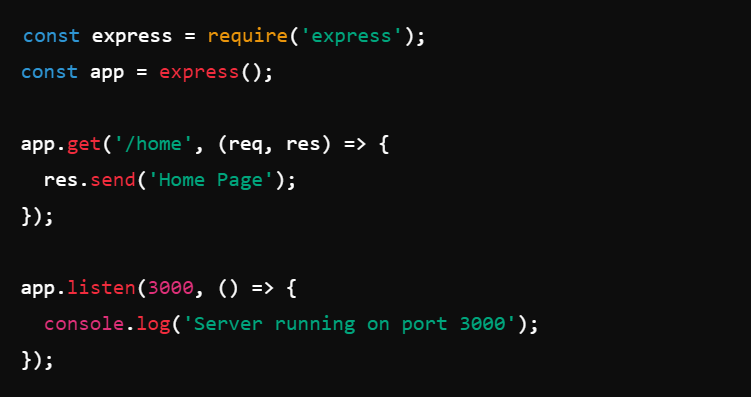
O Node.js puro requer mais configurações e código manual para tarefas como manipulação de rotas e requisições HTTP. Já o Express fornece uma interface de mais alto nível, permitindo que o desenvolvedor foque na lógica da aplicação.

**Exemplo de servidor básico com Node.js puro:**

****

Neste exemplo, estamos criando manualmente um servidor HTTP que responde à rota /home com um texto simples. No entanto, rotas, tratamento de erros e outras funcionalidades precisam ser codificadas manualmente, o que aumenta a complexidade.

**Exemplo de servidor básico com Express:**

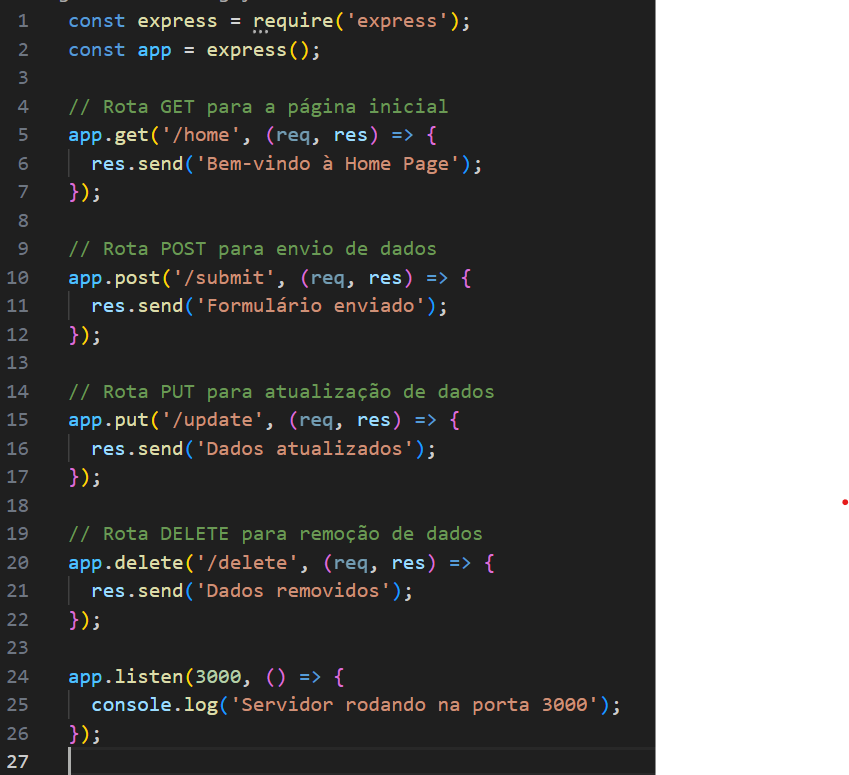
****

Com o Express, o mesmo servidor é implementado de forma mais concisa. O método app.get define diretamente a rota /home, e a função res.send() envia a resposta.

**3. Como Criar rotas básicas com Express:**

Rotas são usadas para definir como o servidor Express responde a diferentes requisições HTTP, como GET, POST, PUT, entre outros.

**Exemplo de criação de rotas:**

****

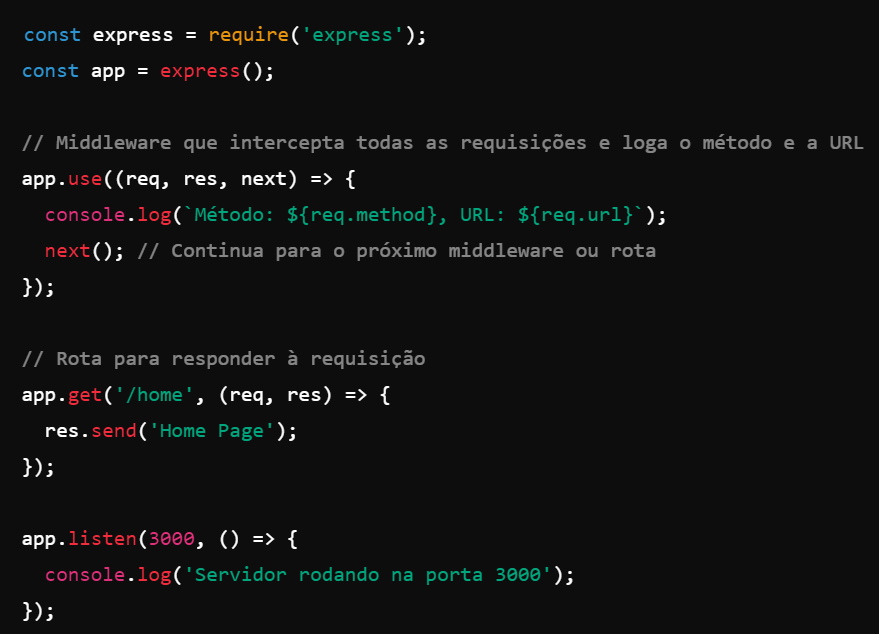
* A função app.get() define uma rota para requisições GET.
* app.post(), app.put() e app.delete() são usados para as respectivas requisições POST, PUT e DELETE.

**4. Middlewares no Express**

O Middleware é usado extensivamente em aplicativos Express para que as tarefas ofereçam arquivos estáticos ao tratamento de erros, a comprensão de respostas HTTP. Enquanto as funções de rota terminam o ciclo de solicitação-resposta HTTP, retornando alguma resposta ao cliente HTTP, as funções de middleware normalmente executam alguma operação na solicitação ou resposta e, em seguida, ligue para a próxima função na "pilha", que pode ser mais um middleware ou uma rota manipuladora. A ordem em que o middleware é chamado depende do desenvolvedor do aplicativo.

Middlewares são funções que têm acesso ao objeto de requisição (req), resposta (res), e ao próximo middleware na fila. Eles permitem interceptar, modificar e até interromper a resposta de uma requisição.

Exemplo:



* O middleware definido com app.use() será executado para todas as requisições. Ele loga o método HTTP e a URL requisitada.
* O método next() garante que a requisição continue para o próximo middleware ou rota. Se não fosse chamado, a execução seria interrompida.

**5. Proxy:**

Um proxy é um intermediário entre o cliente e o servidor de destino. Ele recebe as requisições e as redireciona para outro servidor, possivelmente modificando-as no processo. Isso é útil para esconder a complexidade do servidor backend ou para integras APIs externas.

**5.1 Configurando o Projeto Express**

Primeiramente, para iniciar um projeto Node.js e instalar o **Express**. Isso pode ser feito através do terminal, utilizando o comando npm init para configurar o projeto e em seguida instalando o Express com o comando npm install express. Express será o framework que permitirá a criação de um servidor de aplicação.

Dentro do projeto, crie um arquivo index.js ou app.js que será o ponto de entrada do seu servidor Express. Neste arquivo, você configura o servidor básico que será responsável por servir a aplicação React e atuar como um proxy para as requisições feitas pelo cliente.

**5.2 Configurando o Proxy no Express**

Para que o Express funcione como um **proxy**, você precisará instalar um middleware chamado http-proxy-middleware. Esse middleware é utilizado para interceptar as requisições que o cliente (aplicação React) faz e redirecioná-las para outro servidor, que geralmente é o backend ou uma API externa.

No arquivo de entrada do Express, você configurará esse proxy para que todas as requisições que tenham um determinado caminho (por exemplo, /api) sejam redirecionadas para o endereço do servidor de destino (que pode ser uma API hospedada em outro local). Isso é útil em cenários onde a aplicação front-end não se comunica diretamente com o servidor backend, mas sim através do proxy que foi configurado no Express.

**5.3 Servindo a Aplicação React com Express**

Para que o servidor Express sirva a aplicação React, é necessário garantir que os arquivos construídos do React (gerados pelo comando npm run build) sejam entregues corretamente. Esses arquivos estáticos geralmente ficam armazenados na pasta build da aplicação React.

Dentro do arquivo de configuração do Express, deve-se indicar o caminho da pasta de build, permitindo que, quando os usuários acessarem o site, a aplicação React seja carregada. Isso é realizado através da configuração de uma rota estática no Express que serve esses arquivos como conteúdo web.

**5.4 Configurando o Proxy no React**

Além de configurar o proxy no lado do servidor (Express), também precisa configurar o **React** para redirecionar automaticamente as requisições feitas para o backend através do servidor Express. Isso é feito adicionando uma propriedade proxy dentro do arquivo package.json do React. Esse proxy interno do React redireciona qualquer requisição que seria feita para o próprio servidor de desenvolvimento do React para o servidor Express configurado.

Por exemplo, se no React você faz uma requisição HTTP para /api/data, essa requisição será automaticamente redirecionada para o servidor Express, e o Express, por sua vez, redirecionará para o servidor backend real (ou API externa) de acordo com as regras do proxy que você configurou.

**5.5 Testando a Aplicação**

Uma vez que o servidor Express está configurado para servir o React e atuar como proxy, e a aplicação React está configurada para usar esse proxy, basta rodar o servidor Express e acessar a aplicação através do navegador.

Quando a aplicação fizer requisições que correspondem ao caminho configurado (como /api), o proxy redirecionará corretamente para o backend, permitindo que a aplicação React se comunique com o servidor através do proxy do Express.