Aula 1

Created	@October 28, 2023 11:22 AM
Data	@January 8, 2024
Status	Não iniciado
Tipo	Aula

Implementando servidores web HTTP(s)

A anatomia de uma URL

1. **Protocolo:**
    - O protocolo indica como o navegador e o servidor devem se comunicar. Exemplos comuns incluem `http://` e `https://` (para comunicação segura).
2. **Domínio e DNS (Domain Name System):**
    - O domínio é o endereço principal que você digita na barra de endereços do navegador, como "www.example.com".
    - O DNS é um sistema que traduz esse domínio em um endereço IP. Cada site tem um endereço IP associado, e o DNS mapeia o domínio para esse IP.
3. **Path:**
    - O caminho (path) indica a localização específica no servidor para onde a solicitação deve ser direcionada. Por exemplo, em "https://www.example.com/pagina", "/pagina" é o path.
4. **Parâmetros de Consulta (Query Parameters):**
    - Os parâmetros de consulta são usados para enviar dados adicionais para o servidor. Eles começam com um ponto de interrogação (?) e são geralmente na forma `chave=valor`. Por exemplo, em "https://www.example.com/busca?termo=nodejs", "termo=nodejs" é um parâmetro de consulta.

Então, uma URL completa pode parecer assim:

```
https://www.example.com/pagina?chave1=valor1&chave2=valor2
```

- **Protocolo:** `https://`
- **Domínio:** `www.example.com`
- **Path:** `/pagina`
- **Parâmetros de Consulta:** `chave1=valor1` e `chave2=valor2`

Como funciona a arquitetura de comunicação cliente/servidor

A arquitetura de comunicação cliente/servidor é um modelo em que um cliente faz uma solicitação ao servidor, que por sua vez processa a solicitação e envia uma resposta de volta ao cliente. Vamos simplificar esse processo:

1. **Cliente Faz uma Solicitação (Request):**
    - O cliente é geralmente um navegador da web, aplicativo móvel ou qualquer dispositivo que solicita informações ou serviços.
    - O cliente envia uma solicitação para o servidor. Essa solicitação geralmente inclui informações sobre o que o cliente deseja (por exemplo, abrir uma página da web).
2. **Servidor Processa a Solicitação:**
    - O servidor é um computador remoto que armazena dados, recursos ou fornece serviços.
    - O servidor recebe a solicitação do cliente e processa-a. Isso pode envolver a execução de códigos, consulta de bancos de dados, ou qualquer outra operação necessária para atender à solicitação.
3. **Servidor Gera uma Resposta (Response):**

- Após processar a solicitação, o servidor cria uma resposta que contém os dados solicitados ou informações sobre o resultado da operação.
- A resposta é então enviada de volta ao cliente.

4. **Cliente Recebe a Resposta:**
    - O cliente recebe a resposta do servidor e a interpreta. Isso pode envolver renderizar uma página da web, exibir informações ou realizar outras ações com base na resposta recebida.

Em termos mais técnicos, a comunicação geralmente ocorre através de protocolos como HTTP (Hypertext Transfer Protocol) para aplicações web. O cliente envia uma solicitação HTTP para um servidor, que responde com uma mensagem contendo os dados solicitados ou informações sobre o status da operação.

Essa arquitetura é fundamental para a comunicação eficiente na internet, permitindo a distribuição de tarefas entre clientes e servidores, cada um desempenhando um papel específico para atender às necessidades dos usuários.

Diferença entre aplicações baseadas em MVC (Model View Controller) e Microservices

Aplicações Baseadas em MVC (Model-View-Controller):

1. **Modelo (Model):**
    - Representa a lógica de negócios e os dados da aplicação.
    - Gerencia o acesso e manipulação dos dados.
2. **Visão (View):**
    - É responsável por exibir as informações para o usuário.
    - Apresenta os dados do modelo ao usuário de uma maneira compreensível.
3. **Controlador (Controller):**
    - Recebe as entradas do usuário e manipula a comunicação entre o modelo e a visão.
    - Atua como um intermediário que processa as ações do usuário e atualiza o modelo ou a visão conforme necessário.

4. **Arquitetura:**
    - As aplicações baseadas em MVC seguem um padrão arquitetônico que separa as preocupações relacionadas à lógica de negócios, à apresentação e à interação do usuário.
    - É comum em aplicações monolíticas, onde o código-fonte é geralmente organizado em camadas que representam o modelo, a visão e o controlador.

Microservices:

1. **Arquitetura de Microservices:**
    - Divide uma aplicação em pequenos serviços independentes e autônomos.
    - Cada serviço tem sua própria lógica de negócios, banco de dados e pode ser desenvolvido, implantado e escalado independentemente.
2. **Comunicação:**
    - Os microservices geralmente se comunicam entre si por meio de APIs (Application Programming Interfaces) ou mensagens.
    - Cada microserviço pode ser implementado em uma tecnologia diferente, e a comunicação entre eles é crucial para o funcionamento da aplicação como um todo.
3. **Escalabilidade e Manutenção:**
    - Os microservices oferecem maior escalabilidade e flexibilidade.
    - Facilitam a manutenção e o desenvolvimento contínuo, já que cada serviço pode ser atualizado separadamente.
4. **Desacoplamento:**
    - Cada microserviço é independente, o que permite uma maior flexibilidade e desacoplamento entre as diferentes partes da aplicação.
    - Isso facilita a evolução e a manutenção, além de facilitar a adoção de tecnologias específicas para cada serviço.

Em resumo, enquanto aplicações baseadas em MVC se concentram na organização interna do código, dividindo as responsabilidades entre modelo, visão e controlador, a arquitetura de microservices se concentra na organização da aplicação como um

conjunto de serviços independentes, permitindo escalabilidade, flexibilidade e manutenção facilitada.

Exercicio

Como implementar e executar um servidor web simples usando Node.js sem o uso de frameworks. Vamos criar um servidor básico que responde a requisições HTTP.

Primeiro, certifique-se de ter o Node.js instalado no seu sistema. Se não tiver, você pode baixá-lo em https://nodejs.org/.

Aqui está um exemplo mínimo de um servidor web usando Node.js:

1. Crie um novo arquivo chamado `server.js`:

```
// Importa o módulo HTTP do Node.js
const http = require('http');

// Configura as informações do servidor
const host = '127.0.0.1';
const port = 3000;

// Cria o servidor
const server = http.createServer((req, res) => {
  // Configura o cabeçalho de resposta com o tipo de conteúdo
  res.writeHead(200, {'Content-Type': 'text/plain'});

  // Envia a resposta para o cliente
  res.end('Olá, este é o meu servidor web!\n');
});

// Faz o servidor escutar em uma porta específica e endereço IP
server.listen(port, host, () => {
  console.log(`Servidor rodando em http://${host}:${port}/`);
```

```
});
```

1. Salve o arquivo `server.js` e abra um terminal.
2. Navegue até o diretório onde o arquivo está localizado e execute o seguinte comando:

```
node server.js
```

Isso iniciará o servidor e você verá a mensagem `Servidor rodando em http://127.0.0.1:3000/`.

Agora, seu servidor está esperando por requisições HTTP na porta 3000. Quando você abrir um navegador e acessar `http://127.0.0.1:3000/`, verá a mensagem "Olá, este é o meu servidor web!".

Este é um exemplo muito simples, mas você pode expandir e aprimorar conforme necessário. Se quiser lidar com diferentes rotas, métodos HTTP, ou adicionar outros recursos, você pode começar a explorar bibliotecas ou frameworks como Express.js.

- Pesquisar sobre headers
- Pesquisar sobre qual formato de dados é trafegado por uma API.
- Configurar servidor criado para que trabalhe com o formato de API
- Instalar express e replicar configuração de servidor criado com nodejs puro nele.