Internet Essencial

A Base do Back-End Web

Jamily Batista

Introdução

Neste ebook, exploraremos os fundamentos essenciais da internet para o desenvolvimento web backend. Vamos entender como a internet funciona, os principais protocolos e tecnologias envolvidos, e como implementar funcionalidades básicas usando exemplos de código simples. O objetivo é fornecer uma base sólida para quem deseja iniciar no desenvolvimento backend, abordando temas de forma clara e direta.

CAPÍTULO 01

HTTP e HTTPS

HTTP e HTTPS

O que são HTTP e HTTPS?

HTTP (Hypertext Transfer Protocol) e HTTPS (HTTP Secure) são os protocolos que permitem a comunicação entre clientes (navegadores) e servidores. HTTP é usado para transferir dados na web, enquanto HTTPS é uma versão segura que utiliza criptografia SSL/TLS para proteger os dados.

Como funciona o HTTP?

HTTP é um protocolo baseado em requisições e respostas. O cliente (navegador) envia uma requisição HTTP ao servidor, que processa a requisição e retorna uma resposta com os dados solicitados.

Estrutura de uma Requisição HTTP

Uma requisição HTTP típica possui a seguinte estrutura:

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept: text/html
```

- **GET:** Método HTTP usado para solicitar um recurso.
- **/index.html:** Caminho do recurso solicitado no servidor.
- **HTTP/1.1:** Versão do protocolo HTTP.
- **Host:** Domínio do servidor.
- **User-Agent:** Informações sobre o cliente que está fazendo a requisição.
- **Accept:** Tipo de conteúdo que o cliente aceita.

Estrutura de uma Resposta HTTP

Uma resposta HTTP típica possui a seguinte estrutura:

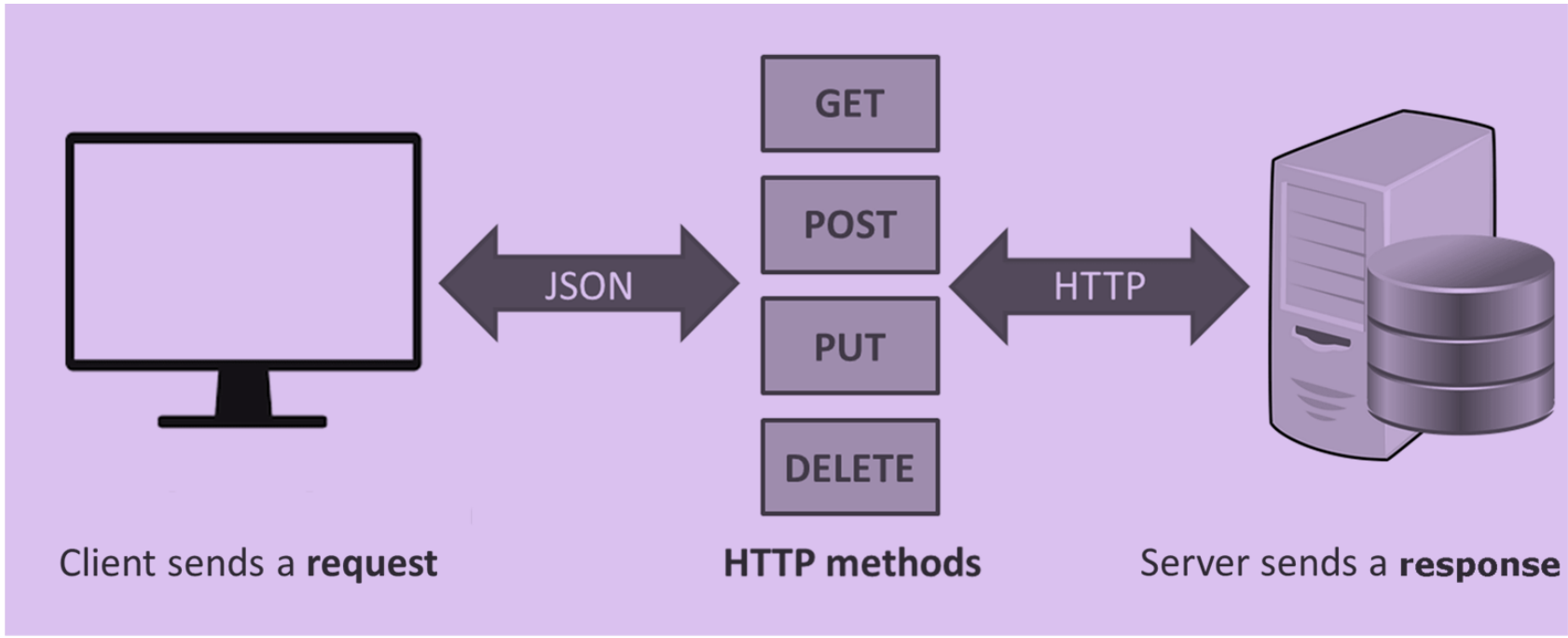
```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1234

<!DOCTYPE html>
<html>
...
</html>
```

- **HTTP/1.1 200 OK:** Versão do protocolo HTTP e código de status da resposta.
- **Content-Type:** Tipo de conteúdo retornado.
- **Content-Length:** Tamanho do conteúdo em bytes.
- O corpo da resposta contém o conteúdo HTML solicitado

Métodos HTTP Comuns

GET: Solicita um recurso.
POST: Envia dados ao servidor para criar um novo recurso.
PUT: Atualiza um recurso existente.
DELETE: Remove um recurso.



HTTP Status Codes

Os códigos de status HTTP são divididos em cinco categorias:

- **1xx (Informational): I**ndica que a requisição foi recebida e o processo continua.
- **2xx (Success):** Indica que a requisição foi bem-sucedida.
- **200 OK:** Requisição bem-sucedida.
- **201 Created:** Recurso criado com sucesso.
- **3xx (Redirection):** Indica que a ação adicional é necessária para completar a requisição.
- **301 Moved Permanently:** Recurso movido permanentemente para um novo URL.
- **302 Found:** Recurso encontrado em um novo URL.
- **4xx (Client Error):** Indica que houve um erro na requisição do cliente.
- **400 Bad Request:** Requisição inválida.
- **401 Unauthorized:** Autenticação necessária.
- **404 Not Found:** Recurso não encontrado.
- **5xx (Server Error):** Indica que houve um erro no servidor.
- **500 Internal Server Error:** Erro genérico do servidor.

Como funciona o HTTPS?

HTTPS é a versão segura do HTTP. Ele utiliza SSL/TLS para criptografar os dados transmitidos entre o cliente e o servidor, garantindo que os dados não possam ser interceptados por terceiros.

Estabelecimento de Conexão HTTPS

- **Cliente se conecta ao servidor:** O cliente inicia uma conexão ao servidor na porta 443.
- **Troca de certificados:** O servidor envia seu certificado SSL/TLS ao cliente para verificação.
- **Handshake SSL/TLS:** Cliente e servidor estabelecem uma chave de sessão segura.
- **Transferência de dados criptografados:** Dados são transmitidos de forma segura usando a chave de sessão.

CAPÍTULO 02

Servidores Web

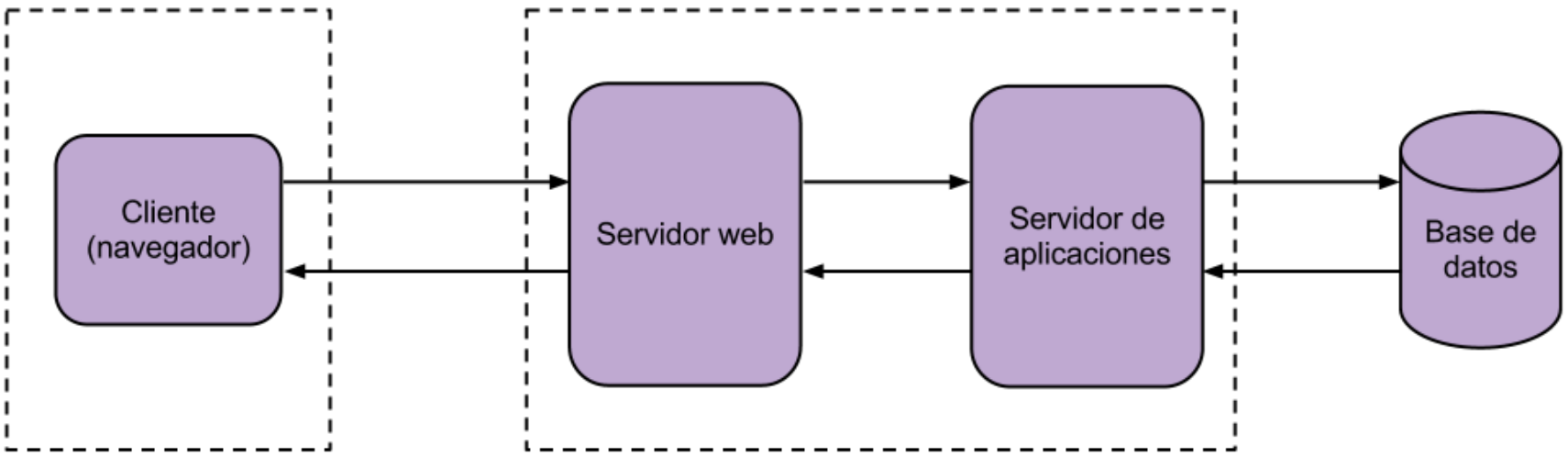
Servidores Web

O que é um Servidor Web?

Um servidor web é um software que entrega conteúdo web, como páginas HTML, arquivos, imagens e vídeos, aos clientes que solicitam esses recursos através da internet. Ele atua como intermediário, recebendo requisições de clientes (navegadores) e respondendo com os dados solicitados. Servidores web são essenciais para o funcionamento da web, pois facilitam a distribuição de informações e serviços.

Funcionamento de um Servidor Web

Quando um navegador faz uma solicitação para acessar um site, a solicitação é enviada ao servidor web. O servidor então processa a solicitação e retorna uma resposta contendo os dados requisitados.



Ciclo de vida de uma solicitação HTTP

Esse processo pode ser dividido nas seguintes etapas:

Recepção da Requisição: O servidor recebe uma requisição HTTP/HTTPS do cliente.

Processamento da Requisição: O servidor interpreta a requisição, identificando qual recurso foi solicitado.

Execução de Lógica do Servidor: O servidor pode executar scripts ou consultas a bancos de dados para gerar o conteúdo dinâmico.

Formatação da Resposta: O servidor formata a resposta HTTP, incluindo cabeçalhos e o corpo da resposta.

Envio da Resposta: O servidor envia a resposta de volta ao cliente, que a renderiza para o usuário.

Configuração de um Servidor Web

Vamos explorar como configurar um servidor web simples usando Node.js e Express.js, uma biblioteca popular para criação de servidores em Node.js.

Exemplo de Configuração de Servidor com Node.js:

```
const http = require('http');

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello, World!\n');
});

server.listen(3000, '127.0.0.1', () => {
  console.log('Server running at http://127.0.0.1:3000/');
});
```

Tipos de Servidor Web

Existem vários tipos de servidores web, cada um com suas características e casos de uso específicos:

Apache HTTP Server: Um dos servidores web mais populares e amplamente utilizados. Conhecido por sua flexibilidade e suporte a módulos.

Nginx: Famoso por seu desempenho e capacidade de lidar com um grande número de conexões simultâneas. Comumente usado como proxy reverso e balanceador de carga.

Microsoft Internet Information Services (IIS): Um servidor web desenvolvido pela Microsoft, amplamente utilizado em ambientes Windows.

CAPÍTULO 03

APIs RESTful

APIs RESTful

O que é uma API RESTful?

APIs RESTful (Representational State Transfer) são interfaces que permitem a comunicação entre diferentes sistemas através de requisições HTTP. Elas são amplamente utilizadas no desenvolvimento de aplicações web e móveis para permitir que diferentes serviços troquem dados e funcionalidades.

REST é uma arquitetura que usa métodos HTTP como GET, POST, PUT e DELETE para realizar operações CRUD (Create, Read, Update, Delete). As APIs RESTful são projetadas para serem simples, escaláveis e sem estado, o que significa que cada requisição do cliente deve conter todas as informações necessárias para o servidor processar a solicitação.

CAPÍTULO 04

Banco de Dados E SQL

Bancos de Dados e SQL

Introdução aos Bancos de Dados

Bancos de dados são essenciais no desenvolvimento back-end, armazenando dados para serem utilizados pelas aplicações. Eles permitem que você organize, manipule e recupere informações de forma eficiente. Vamos focar em bancos de dados relacionais, que são uma ótima escolha para iniciantes devido à sua estrutura bem definida e ao uso de SQL (Structured Query Language) para gerenciar os dados.

Bancos de Dados Relacionais

Os bancos de dados relacionais armazenam dados em tabelas compostas por linhas e colunas. Cada tabela representa uma entidade, como "usuários" ou "produtos", e cada linha representa uma entrada individual dentro dessa entidade.

Exemplos Populares

- MySQL
- PostgreSQL
- SQLite

Fundamentos de SQL

SQL (Structured Query Language) é a linguagem usada para interagir com bancos de dados relacionais. Vamos explorar os principais comandos SQL para operações CRUD (Create, Read, Update, Delete).

1. Criando uma Tabela

Antes de armazenar dados, você precisa criar uma tabela. Abaixo está um exemplo de como criar uma tabela para armazenar informações de usuários:

```
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL
);
```

2. Inserindo Dados

Para adicionar dados a uma tabela, usamos o comando INSERT INTO. Aqui está um exemplo de como inserir um novo usuário na tabela users:sqlC

```
INSERT INTO users (name, email) VALUES ('John Doe', 'john.doe@example.com');
```

3. Lendo Dados

Para recuperar dados de uma tabela, usamos o comando SELECT. O exemplo a seguir mostra como selecionar todos os usuários da tabela users:

```
SELECT * FROM users;
```

4. Atualizando Dados

Para modificar dados existentes em uma tabela, usamos o comando UPDATE. O exemplo a seguir mostra como atualizar o e-mail de um usuário específico:

```
UPDATE users SET email =
'john.newemail@example.com' WHERE id = 1;
```

5. Excluindo Dados

Para remover dados de uma tabela, usamos o comando DELETE. O exemplo a seguir mostra como excluir um usuário específico:

```
DELETE FROM users WHERE id = 1;
```

CONCLUSÃO

Neste ebook, cobrimos os fundamentos essenciais da internet para o desenvolvimento web backend. Desde o entendimento básico da internet e protocolos de comunicação até a criação de servidores, APIs, gerenciamento de bancos de dados e implementação de autenticação e autorização. Com esses conhecimentos, você está preparado para iniciar sua jornada no desenvolvimento backend, criando aplicações robustas e seguras.

AGRADECIMENTOS

Obrigada por ler até aqui!

Esse Ebook foi gerado por IA e diagramado por humano.

Esse conteúdo foi gerado com fins didáticos de construção, não foi realizada uma avaliação cuidadosa humana no conteúdo e pode conter erros gerados por uma IA.

Autor

Jamily Batista

https://github.com/JamilyB