Backend para Iniciantes

Uma Jornada pelo Desenvolvimento de Servidores e APIs

Joseli Madureira

Projeto criado utilizando Microsoft COPILOT, para conclusão do Bootcamp DIO/ CAIXA - IA Generativa com Microsoft Copilot

Joseli Madureira

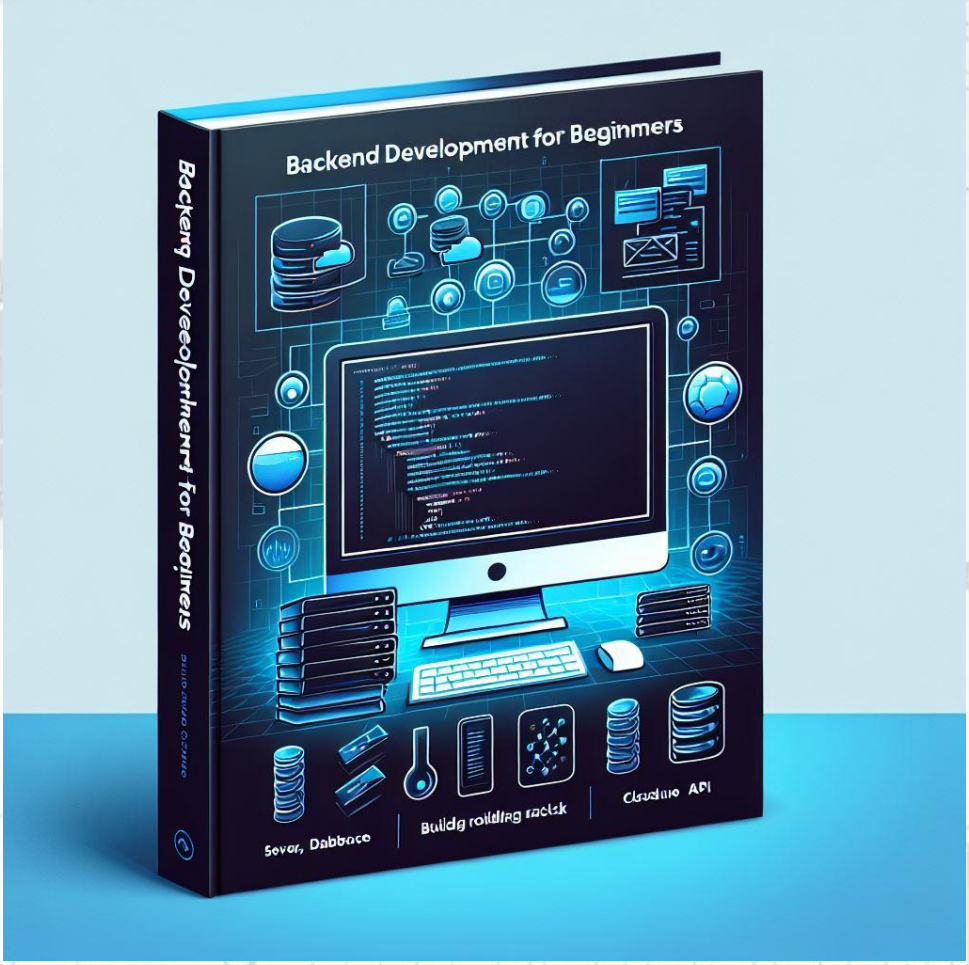
Introdução ao Backend

O backend é crucial para garantir que as aplicações web funcionem corretamente e de forma eficiente.

Ele lida com a lógica de negócios, gerenciamento de dados e segurança, tornando-se a espinha dorsal de qualquer aplicação.

•O que será abordado

- Vamos explorar desde a arquitetura de servidores até as melhores práticas de segurança, passando por linguagens de programação e ferramentas essenciais.



Introdução ao Banckend

- **O que é backend?**
  - O backend é a parte do desenvolvimento web que lida com a lógica do servidor, banco de dados e integração de APIs.
- **Importância do backend no desenvolvimento web**
  - Essencial para a funcionalidade e desempenho de aplicações web.

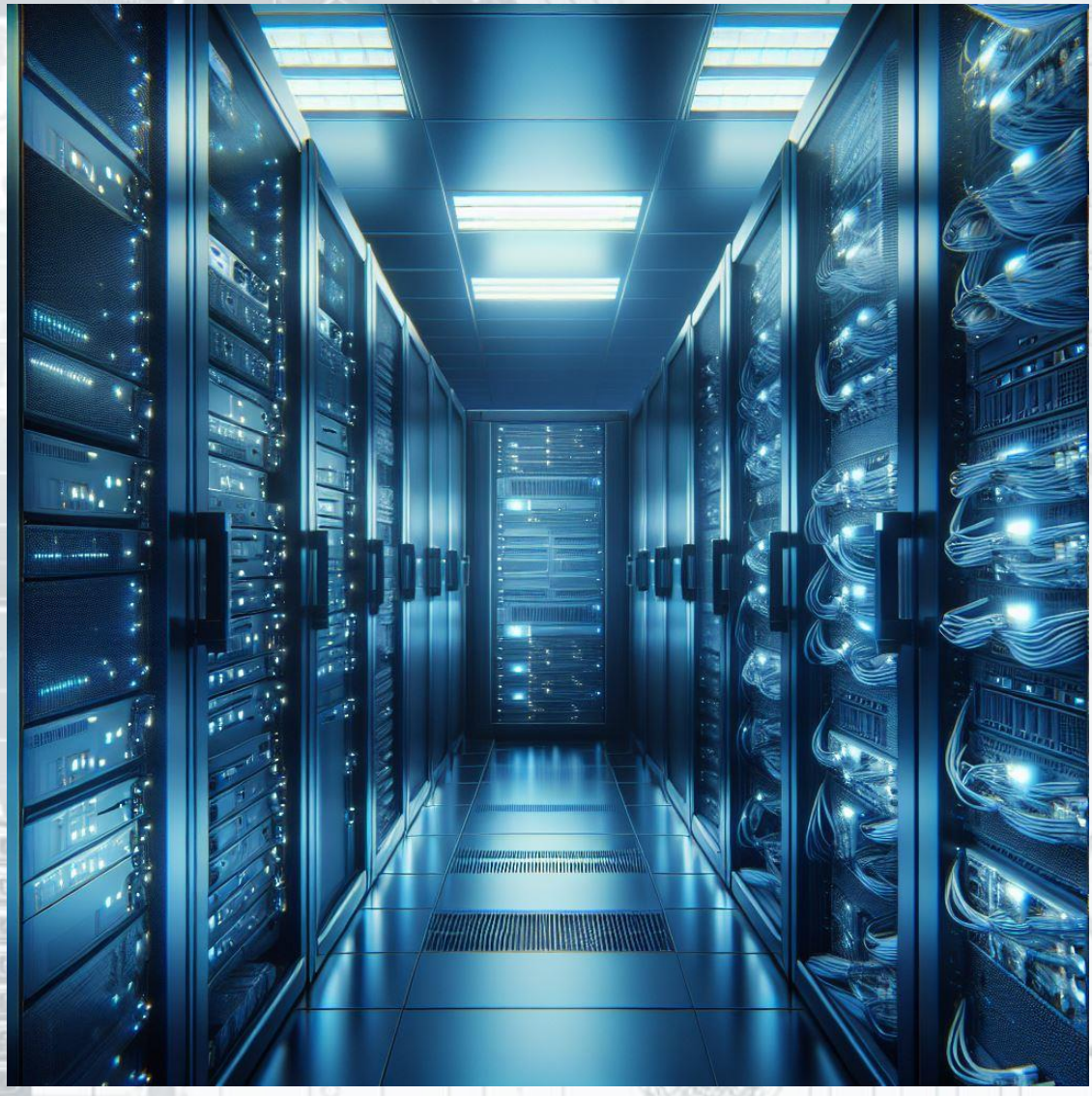
- **Arquitetura de Servidores**
  - Diferentes tipos de arquiteturas (monolítica, microserviços, serverless).
  - Como escolher a arquitetura certa para o seu projeto.
- **Linguagens de Programação**
  - Principais linguagens usadas no backend (Java, Python, Node.js, Ruby).
  - Vantagens e desvantagens de cada linguagem.
- **Bancos de Dados**
  - Bancos de dados relacionais (SQL) e não relacionais (NoSQL).
  - Como escolher e otimizar o banco de dados para o seu projeto.
- **APIs**
  - Criação e manutenção de APIs RESTful e GraphQL.
  - Boas práticas para documentação e uso de APIs.

Introdução ao Banckend

- **Segurança**
  - Implementação de práticas de segurança para proteger dados sensíveis.
  - Autenticação e autorização robustas.

- **Ferramentas e Frameworks**
  - Principais ferramentas e frameworks usados no desenvolvimento backend.
  - Como essas ferramentas podem facilitar o desenvolvimento e a manutenção do projeto.

Arquitetura de Servidores



Arquitetura de Servidores

Tipos de Servidores

- **Dedicados**
  - **Vantagens**: Alto desempenho, controle total, segurança.
  - **Desvantagens**: Custo elevado, necessidade de gerenciamento especializado.
  - **Adequação**: Ideal para grandes empresas ou aplicações que exigem alto desempenho e segurança, como bancos e grandes e-commerces.
  - **Exemplo Prático**: Um banco que precisa de um servidor dedicado para garantir a segurança e o desempenho das transações financeiras.
- **VPS (Servidor Virtual Privado)**
  - **Vantagens**: Custo-benefício, controle quase total, escalabilidade.
  - **Desvantagens**: Recursos compartilhados, pode exigir conhecimento técnico.
  - **Adequação**: Bom para pequenas e médias empresas que precisam de mais controle e desempenho do que um servidor compartilhado pode oferecer, mas sem o custo de um servidor dedicado.

Arquitetura de Servidores

- **Exemplo Prático**: Uma startup que precisa de um ambiente de desenvolvimento e teste isolado, mas não tem orçamento para um servidor dedicado.

- **Cloud**
  - **Vantagens**: Alta escalabilidade, pagamento conforme o uso, fácil gerenciamento.
  - **Desvantagens**: Dependência de terceiros, custos variáveis.
  - **Adequação**: Ideal para empresas que precisam de flexibilidade e escalabilidade, como aquelas que têm picos de tráfego sazonais.
  - **Exemplo Prático**: Um serviço de streaming que precisa escalar rapidamente durante eventos ao vivo, como transmissões de esportes.

Estrutura de um Servidor Backend

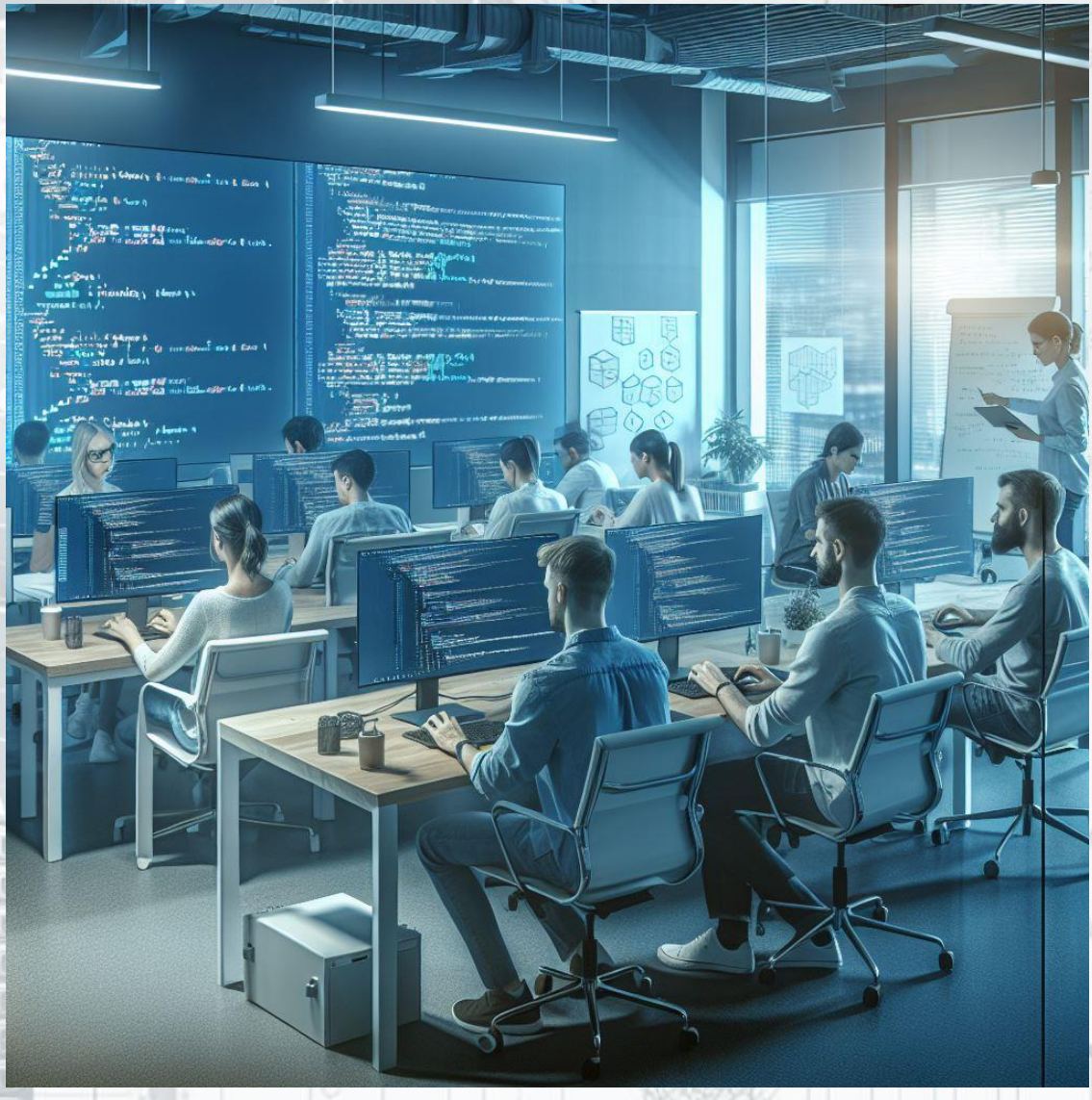
- **Componentes Principais**
  - **CPU**: Processamento das tarefas e execução de instruções.
  - **Memória (RAM)**: Armazenamento temporário para dados e processos em execução.
  - **Armazenamento**: Espaço para armazenar dados e arquivos (HDD ou SSD).

Arquitetura de Servidores

Exemplos de Arquiteturas Populares

- **Monolítica**
  - **Descrição**: Toda a aplicação é construída como uma única unidade.
  - **Vantagens**: Simplicidade, fácil de desenvolver e testar.
  - **Desvantagens**: Dificuldade de escalar, manutenção complexa.
  - **Exemplo Prático**: Uma aplicação interna de uma pequena empresa que não precisa escalar rapidamente.
- **Microservices**
  - **Descrição**: A aplicação é dividida em pequenos serviços independentes.
  - **Vantagens**: Escalabilidade, facilidade de manutenção, flexibilidade.
  - **Desvantagens**: Complexidade de gerenciamento, comunicação entre serviços.
  - **Exemplo Prático**: Uma grande plataforma de e-commerce que precisa escalar diferentes partes do sistema de forma independente, como o catálogo de produtos e o sistema de pagamento.

Linguagens de Programação



Linguagens de Programação

Python

- **Descrição**: Fácil de aprender, ótimo para prototipagem rápida.
- **Exemplo de Código**:
- `print("Hello, World!")`

Java

- **Descrição**: Robusto, amplamente usado em grandes empresas.
- **Exemplo de Código**:
- `public class HelloWorld {`
- `    public static void main(String[] args) {`
- `        System.out.println("Hello, World!");`
- `    }`
- `}`

Node.js

- **Descrição**: Baseado em JavaScript, ideal para aplicações em tempo real.
- **Exemplo de Código**:
- `console.log("Hello, World!");`

Linguagens de Programação

Ruby

- **Descrição**: Conhecido pela simplicidade e produtividade.
- **Exemplo de Código**:
- `puts "Hello, World!"`

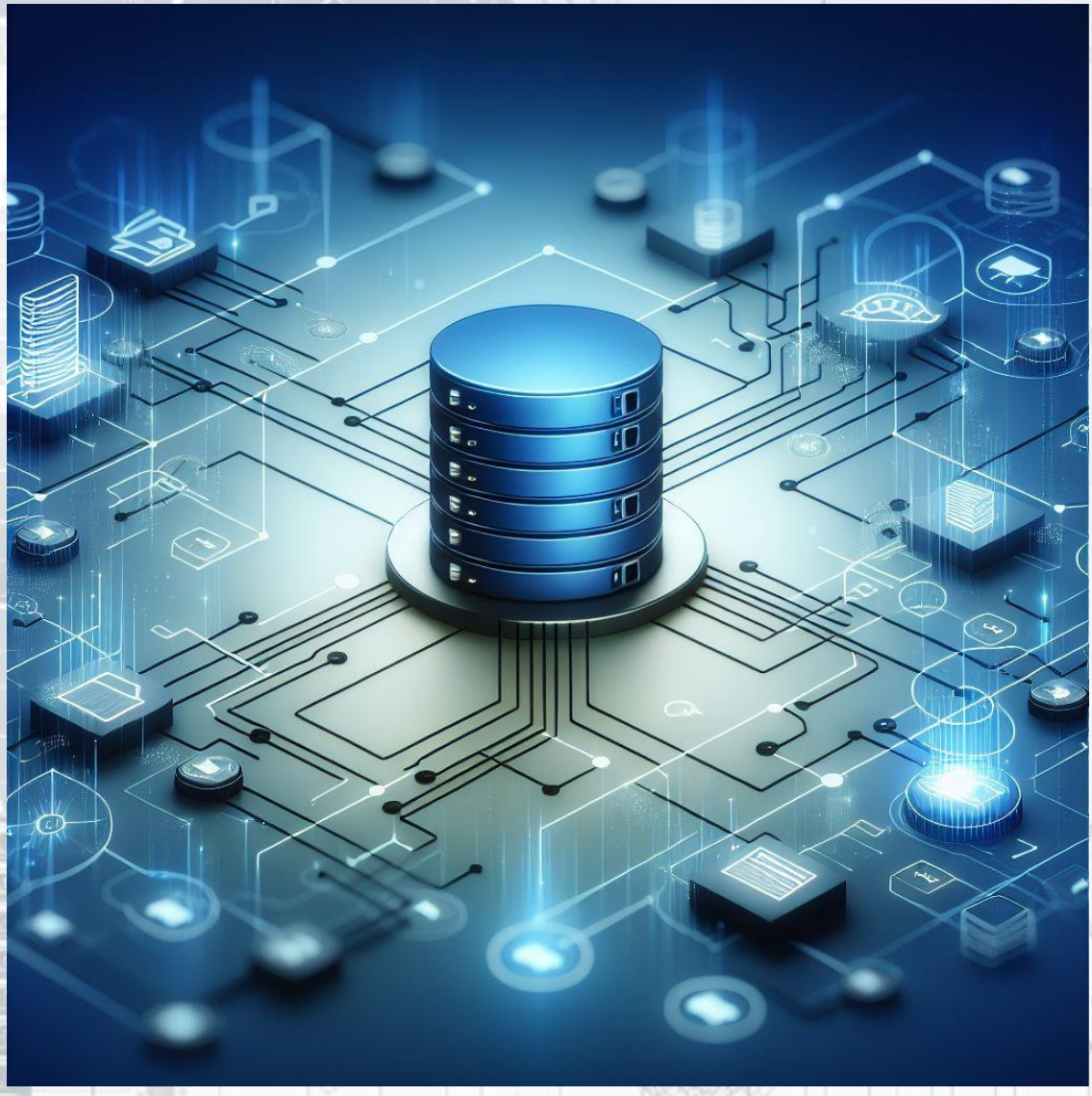
PHP

- **Descrição**: Amplamente usado para desenvolvimento web.
- **Exemplo de Código**:
- `<?php`
- `echo "Hello, World!";`
- `?>`

Comparação e Casos de Uso

- **Python**: Ideal para prototipagem rápida, ciência de dados, e automação.
- **Java**: Excelente para aplicações empresariais de grande escala, sistemas bancários.
- **Node.js**: Perfeito para aplicações em tempo real, como chats e jogos online.
- **Ruby**: Ótimo para startups e desenvolvimento rápido de aplicações web.
- **PHP**: Amplamente utilizado em desenvolvimento web, especialmente com WordPress.

Bancos de Dados



Bancos de Dados

SQL vs NoSQL: Diferenças principais e quando usar cada um

- **SQL (Structured Query Language)**:
  - Estrutura: Tabelas com linhas e colunas.
  - Linguagem: SQL.
  - Exemplos: MySQL, PostgreSQL.
  - Quando usar: Aplicações que requerem transações complexas e integridade de dados.
- **NoSQL (Not Only SQL)**:
  - Estrutura: Documentos, grafos, colunas, chave-valor.
  - Linguagem: Varia conforme o banco de dados.
  - Exemplos: MongoDB, Cassandra.
  - Quando usar: Aplicações que necessitam de escalabilidade horizontal e flexibilidade de dados.

Exemplos de bancos de dados populares

- **MySQL**:
  - Open-source, amplamente utilizado.
  - Bom para aplicações web e sistemas de gerenciamento de conteúdo.

Bancos de Dados

- **PostgreSQL**:
  - Open-source, altamente extensível.
  - Suporta operações complexas e é conhecido por sua conformidade com padrões.
- **MongoDB**:
  - NoSQL, orientado a documentos.
  - Ideal para aplicações que lidam com grandes volumes de dados não estruturados.

Como escolher o banco de dados certo

- **Desempenho**:
  - Avalie a velocidade de leitura/escrita necessária.
  - Considere a latência e o throughput.
- **Escalabilidade**:
  - Determine se a aplicação precisa escalar verticalmente (mais poder de processamento) ou horizontalmente (mais servidores).
- **Complexidade**:
  - Analise a complexidade das transações e a necessidade de integridade referencial.
  - Considere a curva de aprendizado e a facilidade de manutenção.

APIs e RESTful Services

APIs e RESTful Services

O que são APIs?

- **APIs (Application Programming Interfaces)**:
  - Interfaces que permitem a comunicação entre diferentes sistemas.
  - Facilitam a integração de funcionalidades e dados entre aplicações.

Princípios REST

- **Stateless**:
  - Cada requisição do cliente para o servidor deve conter todas as informações necessárias para entender e processar o pedido.
- **Cacheable**:
  - Respostas devem ser explicitamente rotuladas como cacheáveis ou não, para melhorar a eficiência.
- **Client-Server**:
  - Separação de preocupações: o cliente lida com a interface do usuário e o servidor com o armazenamento de dados.
- **Uniform Interface**:
  - Interface uniforme para facilitar a interação entre sistemas.

APIs e RESTful Services

Exemplos de endpoints e métodos HTTP

- **GET**:
  - Recupera dados do servidor.
  - Exemplo: `GET /users` - Obtém a lista de usuários.
- **POST**:
  - Envia dados ao servidor para criar um novo recurso.
  - Exemplo: `POST /users` - Cria um novo usuário.
- **PUT**:
  - Atualiza dados de um recurso existente.
  - Exemplo: `PUT /users/1` - Atualiza os dados do usuário com ID 1.
- **DELETE**:
  - Remove um recurso do servidor.
  - Exemplo: `DELETE /users/1` - Remove o usuário com ID 1.

Segurança no Backend

Segurança no Backend

Autenticação e autorização

- **Autenticação**:
  - Processo de verificar a identidade do usuário.
  - Métodos comuns: JWT (JSON Web Tokens), OAuth.
- **Autorização**:
  - Processo de verificar as permissões do usuário para acessar recursos específicos.
  - Geralmente ocorre após a autenticação.

Proteção contra ataques comuns

- **SQL Injection**:
  - Ataque onde comandos SQL maliciosos são inseridos em uma consulta.
  - Prevenção: Uso de consultas parametrizadas e ORM (Object-Relational Mapping).
- **XSS (Cross-Site Scripting)**:
  - Ataque onde scripts maliciosos são injetados em páginas web.
  - Prevenção: Validação e escape de entrada de dados.

Segurança no Backend

- **CSRF (Cross-Site Request Forgery)**:
  - Ataque onde comandos não autorizados são transmitidos de um usuário em quem a aplicação confia.
  - Prevenção: Tokens CSRF e verificação de origem.

Boas práticas de segurança

- **Validação de entrada**:
  - Sempre validar e sanitizar dados de entrada para evitar injeções e outros ataques.
- **Criptografia de dados**:
  - Criptografar dados sensíveis tanto em trânsito quanto em repouso.
- **Uso de HTTPS**:
  - Garantir que todas as comunicações entre o cliente e o servidor sejam seguras usando HTTPS.

Ferramentas e Frameworks

Ferramentas e Frameworks

Frameworks populares

- **Django (Python)**:
  - Framework web de alto nível que incentiva o desenvolvimento rápido e o design limpo e pragmático.
  - Ideal para aplicações que precisam de uma solução completa e pronta para uso.
- **Spring (Java)**:
  - Framework abrangente para desenvolvimento de aplicações empresariais.
  - Oferece suporte robusto para injeção de dependência e programação orientada a aspectos.
- **Express (Node.js)**:
  - Framework minimalista e flexível para aplicações web em Node.js.
  - Facilita a criação de APIs robustas e escaláveis.

Ferramentas e Frameworks

Ferramentas de desenvolvimento e deploy

- **Docker**:
  - Plataforma para desenvolvimento, envio e execução de aplicações em contêineres.
  - Garante que o ambiente de desenvolvimento seja consistente em todas as etapas do ciclo de vida da aplicação.
- **Kubernetes**:
  - Sistema de orquestração de contêineres para automação de deploy, escalonamento e operações de contêineres.
  - Ideal para gerenciar aplicações em contêineres em larga escala.
- **CI/CD (Continuous Integration/Continuous Deployment)**:
  - Práticas de desenvolvimento que permitem a integração contínua de código e a entrega contínua de software.
  - Ferramentas populares: Jenkins, GitLab CI, CircleCI.

Ferramentas e Frameworks

Exemplos de uso

- **Django**:
  - Configuração de um projeto básico: `django-admin startproject myproject`.
  - Criação de uma aplicação: `python manage.py startapp myapp`.
- **Spring**:
  - Configuração de um projeto básico com Spring Boot: `spring init --dependencies=web myproject`.
  - Execução da aplicação: `./mvnw spring-boot:run`.
- **Express**:
  - Configuração de um projeto básico: `npx express-generator myapp`.
  - Execução da aplicação: `npm start`.

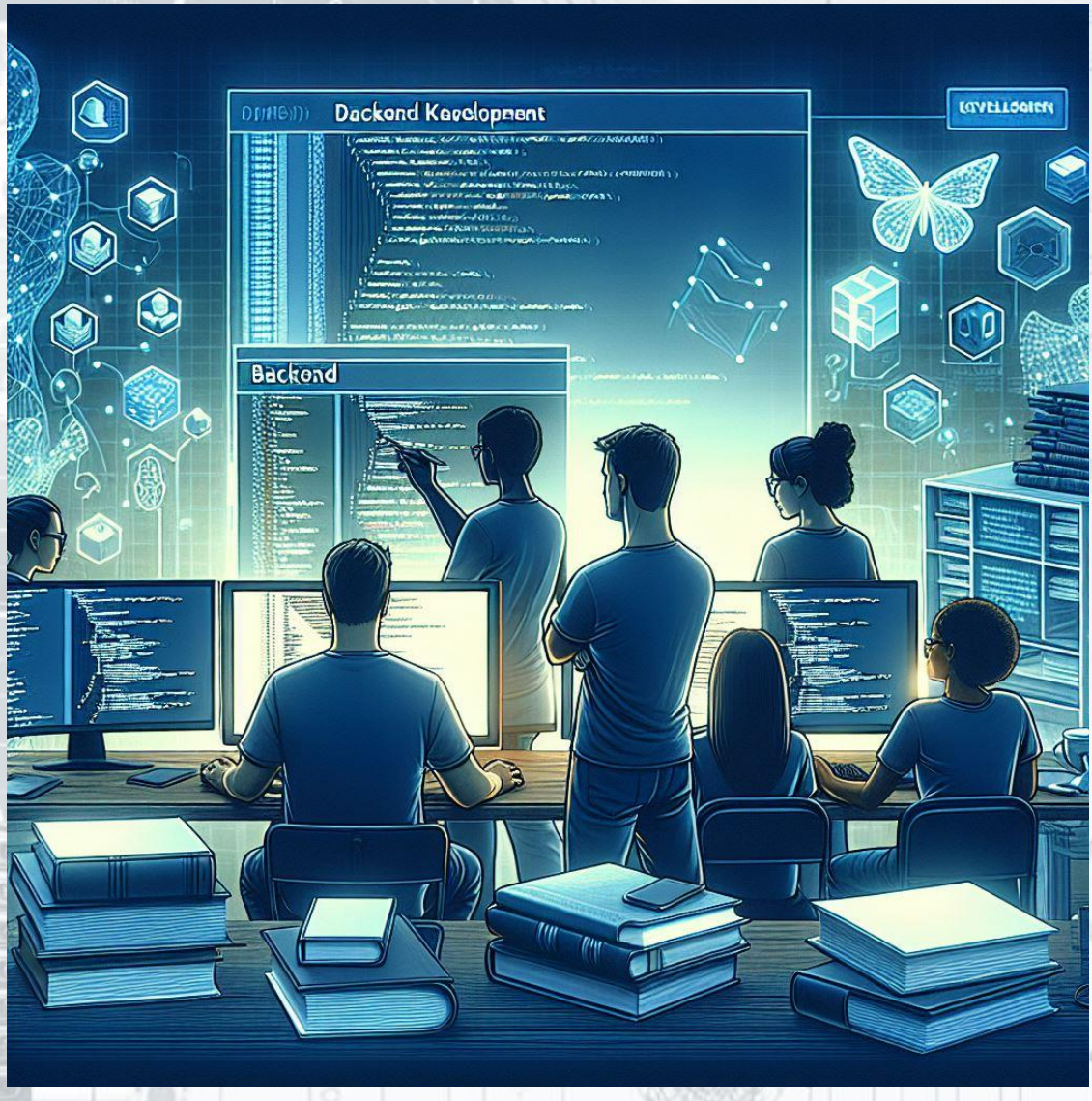
Conclusão

Conclusão

Cobrimos uma ampla gama de tópicos essenciais para o desenvolvimento e a segurança de aplicações modernas. Desde a escolha do banco de dados certo até a implementação de APIs e a proteção do backend, cada tema abordado é crucial para garantir que as aplicações sejam eficientes, escaláveis e seguras.

- **Bancos de Dados**: Entender as diferenças entre SQL e NoSQL e saber quando usar cada um é fundamental para o desempenho e a escalabilidade das aplicações.
- **APIs e Serviços RESTful**: Conhecer os princípios REST e como criar e consumir APIs é essencial para a integração de sistemas.
- **Segurança no Backend**: Implementar práticas de segurança robustas protege os dados e a integridade das aplicações.
- **Ferramentas e Frameworks**: Utilizar frameworks e ferramentas adequadas pode acelerar o desenvolvimento e facilitar o deploy e a manutenção das aplicações.

o Próximos passos para aprofundar o conhecimento



o Próximos passos para aprofundar o conhecimento

Sugestões de Livros

- **"Designing Data-Intensive Applications" por Martin Kleppmann**:
  - Excelente para entender os fundamentos de bancos de dados e sistemas distribuídos.
- **"RESTful Web APIs" por Leonard Richardson e Mike Amundsen**:
  - Um guia completo sobre como projetar e implementar APIs RESTful.
- **"Web Security for Developers" por Malcolm McDonald**:
  - Focado em práticas de segurança para desenvolvedores web.

Cursos e Tutoriais

- **Coursera**:
  - "*Databases and SQL for Data Science*" pela Universidade de Michigan.
  - "*API Design and Fundamentals of Google Cloud's Apigee API Platform*" pela Google Cloud.

o Próximos passos para aprofundar o conhecimento

- **Udemy**:
  - "*The Complete Node.js Developer Course*" por Andrew Mead.
  - "*Docker and Kubernetes: The Complete Guide*" por Stephen Grider.
- **Pluralsight**:
  - "*Spring Framework: Spring Fundamentals*" por Bryan Hansen.
  - "*Web Application Security*" por Troy Hunt.

Recursos Adicionais

Comunidades Online

- **Stack Overflow**: Excelente para tirar dúvidas e encontrar soluções para problemas específicos.
- **Reddit**:
  - r/database: Discussões sobre bancos de dados.
  - r/webdev: Discussões sobre desenvolvimento web.
- **GitHub**: Explore repositórios de projetos open-source e contribua para a comunidade.