

Criar uma API usando SpringBoot

Desenvolver APIs é uma atividade crucial para qualquer programador que trabalhe com sistemas distribuídos. Dentro desse contexto, o SpringBoot é um dos frameworks mais utilizados para criação de APIs em Java.

Com o SpringBoot, você pode criar APIs de maneira eficiente e rápida, aproveitando a vasta gama de recursos oferecidos pela framework. Além disso, o SpringBoot é uma opção popular entre os desenvolvedores devido à sua facilidade de uso e flexibilidade. Ao usar o SpringBoot, você pode criar APIs que atendam às suas necessidades, desde APIs simples até APIs complexas que envolvem múltiplos serviços e integrações. Com o suporte do Spring Framework, você pode se concentrar na lógica de negócios da sua aplicação e deixar que o SpringBoot cuide da infraestrutura da API.

Se você está procurando uma maneira fácil e eficiente de criar APIs em Java, o SpringBoot é uma excelente opção. Com sua ampla gama de recursos e suporte do Spring Framework, você pode criar APIs robustas e escaláveis que atendam às suas necessidades de negócios.

Todo o material para download desde projeto está disponível para download [aqui](#).

O SpringBoot é uma ferramenta que facilita o processo de configuração e inicialização de aplicações Java, permitindo que desenvolvedores criem APIs de maneira rápida e simples.

Importante: Durante esse tutorial veremos diversos termos técnicos. Caso precise, no final, teremos um glossário para ilustrar alguns desses termos.

Índice

Índice

Resumo do Processo

Desenvolvimento da API

 Setup do Projeto

 Hora de Configurar!

 Criando a estrutura de pastas.

 Hora de Codar!

Glossário

 Spring e Spring Boot

 Maven

 Persistência

Fechando o livro, mas não podemos encerrar o aprendizado.

Resumo do Processo

Para criar uma API usando SpringBoot, é necessário seguir alguns passos básicos:

1. Configurar o ambiente de desenvolvimento: Para criar uma API usando SpringBoot, é necessário ter instalado em seu ambiente de desenvolvimento o JDK (Java Development Kit) e o Maven. O JDK é necessário para compilar e executar a aplicação, enquanto o Maven é um gerenciador de dependências que permite gerenciar as bibliotecas que serão utilizadas no projeto.
2. Criar um novo projeto SpringBoot: Para criar um novo projeto SpringBoot, é possível utilizar o Spring Initializr, que é uma ferramenta online para geração de projetos SpringBoot. Nesta ferramenta, é possível escolher as dependências que serão utilizadas no projeto, como por exemplo, o Spring Web, que é utilizado para criar aplicações web.
3. Implementar a camada de serviço: A camada de serviço é responsável por implementar as regras de negócio da aplicação. Nesta camada, é possível implementar a lógica da API, definindo os endpoints que serão disponibilizados e as operações que serão realizadas.

4. Implementar a camada de persistência: A camada de persistência é responsável por realizar a comunicação com o banco de dados. Nesta camada, é possível implementar as operações de CRUD (Create, Read, Update, Delete), que são utilizadas para criar, ler, atualizar e deletar informações do banco de dados.
5. Configurar o servidor: Para disponibilizar a API, é necessário configurar o servidor onde a aplicação será executada. O SpringBoot utiliza o Tomcat como servidor padrão, mas é possível utilizar outros servidores, como o Jetty ou o Undertow.
6. Testar a API: Após implementar a API, é necessário testá-la para garantir que ela está funcionando corretamente. Neste processo, é possível utilizar ferramentas como o Postman, que permite enviar requisições HTTP para a API e validar as respostas recebidas.

Em resumo, criar uma API usando SpringBoot é uma tarefa relativamente simples, que pode ser realizada seguindo alguns passos básicos. Com essa ferramenta, é possível criar APIs robustas e escaláveis, que atendam às necessidades dos desenvolvedores e dos usuários finais.

Para esse tutorial vamos usar uma base dados do Oscar para gerar os dados da API.

Dentro do seu SGBD, crie um banco de dados chamado **oscar_database** e execute o script abaixo (que também pode ser encontrado neste repositório [aqui](#) para download).

[oscar_database_movies.sql](#)

Desenvolvimento da API

Setup do Projeto

Para criar uma API utilizando o Spring Initializr, primeiro precisamos criar o projeto dentro do site <https://start.spring.io/>. O Spring Initializr é uma ferramenta online que ajuda a configurar rapidamente projetos baseados no Spring. Aqui está o passo a passo:

1. Abra o endereço acima.
2. Em "Type", selecione "Maven". O conceito de Maven é explicado no Glossário.

3. Em "Language", escolha a linguagem de programação Java.
4. Em "Spring Boot", selecione a versão desejada do Spring Boot.
5. Em "Project Metadata", preencha as informações sobre o seu projeto, como o nome do grupo (Group), o nome do artefato (Artifact), a versão, a descrição, o nome do pacote (Package name) e assim por diante.
6. Em "Dependencies", você pode selecionar as dependências que deseja incluir no seu projeto Spring. Por exemplo, se você deseja criar uma API REST, selecione "Spring Web". Você pode adicionar outras dependências de acordo com suas necessidades, como a "Spring Data JPA" que usaremos para persistência de dados (explicada mais abaixo) ou a "Spring Security" para autenticação e autorização, entre outras.

Para o nosso projeto, usaremos a seguinte configuração:

The screenshot shows the Spring Initializr web application. On the left, there's a sidebar with 'Project' (Gradle - Maven selected), 'Language' (Java selected), and 'Spring Boot' (3.1.0 selected). The main area has 'Project Metadata' with Group set to 'com.example', Artifact to 'demo', Name to 'demo', Description to 'Demo project for Spring Boot', Package name to 'com.example.demo', Packaging to 'Jar' (selected), and Java version to '17'. On the right, under 'Dependencies', 'Spring Boot Dev Tools' is selected with a note about fast restarts and LiveReload. 'Spring Web' is also selected with a note about building web applications using Spring MVC and Apache Tomcat. 'Spring Data JPA' and 'MySQL Driver' are listed but not selected. There's a 'Dependencies...' button at the top right.

7. Clique em "Generate" para criar o projeto. Um arquivo ".zip" será gerado para download.
8. Descompacte a pasta.
8. Nesse momento, abra o IntelliJ (ou outra IDE de preferência) e abra o projeto recém gerado através da pasta criada.

Hora de Configurar!

10. Abra a estrutura do projeto. Dentro do endereço “src/main/resources” você deverá encontrar um arquivo chamado “**application_properties**”. Esse arquivo é responsável por guardar todas as informações que conectam a sua aplicação em JAVA ao banco de dados.

Um exemplo do conteúdo deste arquivo:

```
## Configura URL do banco
## É importante que o MySQL esteja preparado com o banco antes de executar esse programa
spring.datasource.url=jdbc:mysql://localhost:3306/oscar_database

## Usuário de acesso
spring.datasource.username=root

## Senha do banco
spring.datasource.password=<senha_de_acesso_ao_banco>

## Configura atualizações do banco -> Caso esteja como valor "update" a sua aplicação pode
rá fazer alterações nas tabelas no banco de acordo com o "convencional". Para não permitir
isso você pode usar o valor "none".
spring.jpa.hibernate.ddl-auto=update

##Configuração que mostra que o SQL que foi executado
spring.jpa.properties.hibernate.show_sql=true
```

Atenção: O endereço *mysql://localhost:3306/* é a parte que aponta para o **MySQL** (ou outra tecnologia para armazenamento de dados que esteja utilizando). A palavra “oscar” é referente ao banco que deseja conectar.

A aplicação tentará se conectar através da porta 8080, mas por várias vezes, precisamos configurar a porta de acesso, quando a porta 8080 já está ocupada. Nesse caso, podemos atualizar também o arquivo “**application_properties**” com a linha:

```
server.port = 8080
```

O número 8080 pode ser alterado para 8081, 8082, 8083 e assim por diante.

Criando a estrutura de pastas.

Procure o caminho **src/main/java/com/exemplo/demo**. Dentro da pasta “**demo**” (os nomes das pastas podem variar de acordo com os nomes informados na criação do projeto dentro do portal [start.spring.io](#)).

Dentro dessa pasta você deve ter um arquivo chamado **DemoApplication.java**. Essa classe contém o método principal da sua classe. Não é necessário alterá-lo.

11. Agora é o momento de criar três pastas para nossa arquitetura: models, controllers e repositories.

Dentro da estrutura de uma API, as classes são geralmente chamadas de "models", "controllers" e "repositories" devido à arquitetura do padrão MVC (Model-View-Controller) e ao uso do Spring Framework.

A estrutura abaixo segue o princípio da separação de responsabilidades, onde cada classe tem um papel específico. Os models representam as entidades e os dados da aplicação, os controllers lidam com as requisições e respostas HTTP, e os repositories tratam da persistência dos dados. Essa abordagem ajuda a manter o código organizado, modular e facilita a manutenção e a evolução da API ao longo do tempo.

1. Models:

As classes chamadas de "models" representam as entidades de negócio do seu sistema. Essas classes modelam os dados e geralmente correspondem às tabelas em um banco de dados relacional. Os modelos encapsulam os atributos e comportamentos relacionados a uma entidade específica, como um usuário, produto, pedido etc. Eles são responsáveis por representar os dados e fornecer métodos para acessá-los e manipulá-los.

2. Controllers:

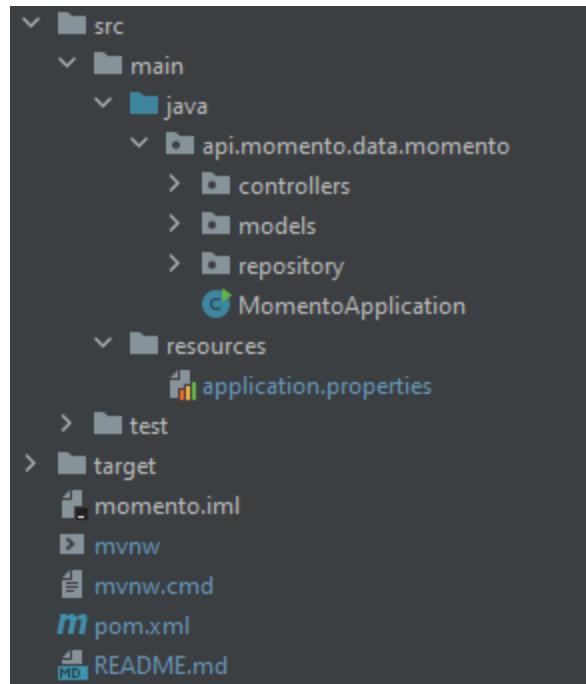
As classes chamadas de "controllers" são responsáveis por receber as solicitações HTTP dos clientes e processá-las. Os controllers lidam com a lógica da aplicação, roteando as solicitações para os métodos apropriados e retornando as respostas apropriadas. Eles atuam como intermediários entre as requisições do cliente e as operações a serem realizadas nos modelos e nos serviços. Os controllers geralmente contêm métodos que são anotados com @RequestMapping ou outras anotações do Spring para mapear os endpoints da API e definir o comportamento esperado.

3. Repositories:

As classes chamadas de "repositories" são responsáveis pela persistência dos dados. Elas são usadas para interagir com o banco de dados ou qualquer outro mecanismo de armazenamento de dados. Os repositories fornecem métodos para criar, recuperar, atualizar e excluir dados no banco de dados. Eles encapsulam a lógica de acesso aos dados e oferecem uma camada de abstração para as operações de leitura e gravação. Os repositories são tipicamente implementados

usando frameworks ORM (Object-Relational Mapping), como o Spring Data JPA, que simplificam a interação com o banco de dados.

A estrutura do seu projeto deve ser similar a essa:



Hora de Codar!

*É importante ressaltar que esses nomes de classes são apenas **convenções** utilizadas no desenvolvimento de APIs com o Spring Framework, mas você não está restrito a elas. Você pode personalizar os nomes das classes de acordo com a sua preferência ou necessidade, desde que mantenha a clareza e a consistência no código-fonte.*

14. Dentro da pasta *models*, crie a sua primeira Classe. Essa classe representará uma entidade/tabela do seu banco de dados, como “Usuários” ou “Produtos”. Nesse exemplo vamos usar a tabela “movies” do Banco de Dados “Oscar.sql”.
15. Essa Classe terá uma estrutura simples: declaração da classe, declaração dos atributos e seus métodos. Declaração da classe:
16. Declaração da classe

```
@Entity //Mostra ao SPRING que é uma tabela  
@Table(schema = "movies") // Dá nome a tabela
```

```
public class Movies{
```

Importante: @Entity e @Table são duas marcações redundantes. O ideal é trabalhar somente com @Entity fazendo correspondência exata a uma tabela do banco. Caso o nome da tabela no banco seja diferente do que é apresentado no banco, podemos usar uma configuração da @Entity ou a marcação @Table para fazer ajustes.

17. Exemplo de Declaração dos atributos:

```
@Id  
@Column(name = "id_movie")  
private int idMovie;  
  
@Column(name = "year_film")  
private int yearFilm;  
  
@Column(name = "year_ceremony")  
private int yearCeremony;  
  
private int ceremony;
```

Ao usar a anotação `@Column`, você pode personalizar várias propriedades de coluna, como nome, tipo de dados, ou o fato de ser uma coluna que representa um ID, tamanho, precisão, valores padrão, restrições de nullable, etc.

18. Declaração dos Atributos:

```
public int getYearFilm() {  
    return yearFilm;  
}  
  
public void setYearFilm(int year_film) {  
    this.yearFilm = yearFilm;  
}
```

Lembrando que o IntelliJ tem o recurso *Generate → Getters and Setters* para gerar todos os métodos automaticamente.

Dentro de um projeto, esse método precisa ser repetido para todas as entidades/tabelas do seu banco.

19. Agora, vamos montar uma nova Classe (focada na nossa classe Movies). Crie o arquivo `MoviesController.java` dentro da pasta **controllers**.

20. Como sempre, a declaração da classe é simples.

```
@RestController //Marca como uma controladora REST ou como uma API
@RequestMapping("/movies") //URL base dessa controladora
public class MoviesController {
```

21. E para cada **endpoint** que criarmos, teremos um novo método.

Aqui trabalhamos com as regras do negócio.

```
@CrossOrigin //Para evitar o erro de CORS
@GetMapping("/")
public List<Movies> findAllRecords(){
    return movies.findAll();
}

@CrossOrigin //Para evitar o erro de CORS
@GetMapping("/id/{id}")
public List<Movies> findAllRecords(){
    return movies.findMoviesByIdMovie();
}
```

Atenção: Repita os passos 20 e 21 para cada Classe dentro da pasta Model que você criou.

22. Por fim, dentro da pasta **repositories**, criamos a Classe responsável por se comunicar com o banco de dados.

```
@Repository
public interface MoviesRepository extends JpaRepository<Movies> {

    Optional<Award> findMoviesByname(String name);

    Optional<Award> findMoviesByIdMovie(int id);
}
```

O código acima define uma interface chamada `MoviesRepository` que estende a interface `JpaRepository` fornecida pelo Spring Data JPA. Vamos analisar o código em detalhes:

1. `@Repository`: É uma anotação do Spring Framework que marca essa interface como um componente que lida com a camada de persistência (repositório). Essa anotação permite que o Spring faça a injeção de dependência e gerencie a criação de instâncias dessa interface.
2. `MoviesRepository`: É o nome da interface que representa o repositório. O nome pode ser personalizado, mas é uma boa prática adicionar o sufixo "Repository" para indicar a finalidade da interface.
3. `extends JpaRepository<Movies>`: A interface `MoviesRepository` estende a interface `JpaRepository` do Spring Data JPA. `JpaRepository` é uma interface genérica fornecida pelo Spring Data JPA que oferece uma variedade de métodos **CRUD** (Create, Read, Update, Delete) pré-implementados para interagir com o banco de dados.
4. `Movies`: É a classe de modelo (ou entidade) relacionada a esse repositório. O tipo genérico `Movies` especifica o tipo de objeto que será persistido e gerenciado pelo repositório.
5. Os métodos (`findMoviesByname` e `findMovieById_movie`) são exemplos de consultas personalizadas que você pode adicionar ao repositório. Esses métodos seguem uma convenção de nomenclatura específica do Spring Data JPA, permitindo que você defina consultas simples apenas escrevendo o nome do método de acordo com um padrão predefinido. Eles são métodos de pesquisa por nome e ID do filme, respectivamente.

23. Execute o projeto.

Se tudo deu certo, o seu projeto será executado e a mensagem “Started [nome_do_seu_projeto] in 9.167 seconds” aparecerá no console.

24. Teste o seu projeto

utilizando um software como o Postman ou o Insominia. Ao usar a URL gerada pelo IntelliJ - algo como <http://localhost:8080/movies/> - você poderá receber os dados da API e consumir no front.

Exemplo de dados recebidos:

```
{
    "id_movie": 42,
    "year_film": 1928,
    "year_ceremony": 1929,
    "ceremony": 2,
    "category": "ACTRESS",
    "name": "Betty Compson",
    "film": "The Barker",
    "winner": "False"
},
```

Exemplo de dados.

Glossário

Spring e Spring Boot

Spring é um framework de desenvolvimento de aplicações Java voltado para a construção de sistemas corporativos robustos e escaláveis. Ele fornece um conjunto abrangente de recursos e bibliotecas que facilitam o desenvolvimento, a configuração e a integração de aplicativos.

Spring Boot, por sua vez, é uma extensão do Spring Framework que simplifica ainda mais o processo de criação de aplicativos Java. Ele oferece convenções de configuração inteligentes e um conjunto de bibliotecas pré-configuradas para facilitar o desenvolvimento de aplicativos independentes e prontos para produção.

A relação entre Spring e Spring Boot é que o Spring Boot é construído em cima do Spring Framework, aproveitando muitos de seus recursos e aprimorando a produtividade do desenvolvedor. O Spring Boot simplifica a configuração e a inicialização de aplicativos Spring, fornecendo padrões de configuração inteligentes e um modelo de programação "convenção sobre configuração". Com o Spring Boot, os desenvolvedores podem criar aplicativos Java de forma mais rápida e eficiente, aproveitando os recursos poderosos do Spring Framework.

Maven

O Apache Maven é uma ferramenta de gerenciamento de projetos amplamente utilizada no desenvolvimento de software em Java e outras linguagens. Ele oferece uma maneira fácil e eficiente de construir, configurar e gerenciar projetos Java,

automatizando muitas tarefas comuns relacionadas à compilação, empacotamento, testes e dependências.

O Maven segue o conceito de "convenção sobre configuração", o que significa que, ao aderir às convenções e estrutura de diretórios predefinidas, muitas configurações podem ser inferidas automaticamente, reduzindo a quantidade de configuração manual necessária.

Aqui estão alguns dos principais recursos e conceitos do Maven:

1. **Gerenciamento de dependências:** O Maven facilita a gestão das dependências do projeto. Você pode especificar as dependências necessárias para sua aplicação no arquivo de configuração chamado "pom.xml" (Project Object Model). O Maven se encarrega de baixar automaticamente as dependências corretas do repositório central Maven ou de outros repositórios remotos.
2. **Ciclo de vida de compilação:** O Maven define um ciclo de vida de compilação padrão que consiste em fases, como "compilar", "testar", "empacotar" e assim por diante. Cada fase executa um conjunto de tarefas pré-definidas. Por exemplo, na fase de "compilar", o código-fonte é compilado, enquanto na fase de "testar", os testes unitários são executados.
3. **Plugins:** O Maven utiliza plugins para executar várias tarefas. Os plugins fornecem funcionalidades adicionais ao ciclo de vida de compilação. Por exemplo, o plugin "maven-compiler-plugin" é responsável pela compilação do código-fonte, e o plugin "maven-surefire-plugin" executa os testes unitários.
4. **Repositório central:** O Maven possui um repositório central que contém uma vasta quantidade de bibliotecas, frameworks e plugins comumente utilizados. Quando você especifica uma dependência em seu arquivo "pom.xml", o Maven procura automaticamente no repositório central para baixar as versões correspondentes.
5. **Personalização e extensibilidade:** O Maven permite que você personalize e configure seu projeto de acordo com suas necessidades específicas. Você pode modificar o ciclo de vida de compilação, adicionar plugins adicionais, configurar variáveis de ambiente e muito mais.

O Maven simplifica bastante o processo de construção e gerenciamento de projetos Java, oferecendo uma estrutura organizada e automatizando muitas tarefas comuns.

Ele promove boas práticas de desenvolvimento e facilita a colaboração em equipes, tornando o processo de construção e entrega de software mais eficiente.

Há uma alternativa ao Maven?

Se você não usa o Maven ou não gosta dele, uma alternativa popular é o Gradle. Ele é uma ferramenta parecida com o Maven, mas mais flexível e fácil de usar. Com o Gradle, você pode escrever as configurações do seu projeto de forma mais simples e expressiva, usando uma linguagem especial. Ele também é mais rápido, pois só recompila as partes do seu código que foram modificadas. O Gradle tem muitos plugins disponíveis e facilita a migração de projetos do Maven. No geral, é uma boa opção se você está procurando uma alternativa ao Maven.

Persistência

Persistência, no contexto de desenvolvimento de software, refere-se à capacidade de armazenar e recuperar dados de forma duradoura, ou seja, **preservar os dados além do tempo de execução da aplicação**. A persistência é fundamental em muitos sistemas, especialmente aqueles que precisam armazenar informações de forma permanente, como dados de usuários, configurações, registros de transações etc.

Quando falamos em persistência de dados, geralmente estamos nos referindo à interação entre a aplicação e algum tipo de mecanismo de armazenamento, como um banco de dados relacional (como MySQL, PostgreSQL, Oracle), um banco de dados NoSQL (como MongoDB, Cassandra, Redis), um sistema de arquivos, ou até mesmo serviços de armazenamento em nuvem.

A persistência permite que os dados sejam salvos em algum formato ou estrutura de armazenamento, geralmente organizados em tabelas (no caso de bancos de dados relacionais), documentos (em bancos de dados NoSQL baseados em documentos) ou outros modelos adequados ao mecanismo de armazenamento utilizado.

A camada de persistência em um aplicativo é responsável por realizar as operações de leitura, gravação, atualização e exclusão de dados, bem como fornecer métodos para consultas e manipulação dos dados armazenados. No contexto do Spring Framework, a persistência de dados é frequentemente tratada pelo Spring Data, que oferece uma abstração para o acesso e manipulação de dados, facilitando a interação com diferentes mecanismos de armazenamento.

A persistência de dados é essencial para muitas aplicações, permitindo que informações importantes sejam armazenadas de forma segura e recuperadas quando necessário. Ela desempenha um papel crucial em sistemas que precisam lidar com grandes volumes de dados e oferecer funcionalidades como busca, filtragem, ordenação e atualizações consistentes.

Fechando o livro, mas não podemos encerrar o aprendizado.

Vamos chegando ao fim deste treinamento de construção de APIs em Spring Boot.

Lembrando: todo o material para download desde projeto está disponível para download [aqui](#).

Esperamos que este material tenha sido enriquecedor e que você tenha adquirido valiosas habilidades no desenvolvimento de APIs usando Spring Boot. Sabemos que aprender novas tecnologias e frameworks pode ser desafiador, mas você demonstrou perseverança e comprometimento, o que é fundamental para se destacar no mundo da programação.

No entanto, lembre-se de que a jornada de aprendizado é contínua e nunca termina. A área de desenvolvimento de software está em constante evolução, e para se manter atualizado(a) e competitivo(a), é crucial continuar estudando e praticando. Aqui estão algumas áreas que você pode explorar para aprimorar ainda mais suas habilidades:

- 1. Spring Data JPA:** Aprofunde-se na camada de persistência e no mapeamento objeto-relacional com o Spring Data JPA. Aprenda a realizar consultas avançadas e otimizar o acesso ao banco de dados.
- 2. Segurança na API:** Estude as melhores práticas para proteger suas APIs, como autenticação, autorização, tokens de acesso e proteção contra ataques comuns.
- 3. Testes automatizados:** Explore diferentes tipos de testes (unitários, integração, end-to-end) e como implementá-los em suas APIs para garantir a qualidade do código.
- 4. Documentação da API:** Aprenda a utilizar ferramentas como o Swagger para gerar documentação interativa para sua API, tornando mais fácil para outros desenvolvedores utilizarem seus serviços.

5. **Desempenho e escalabilidade:** Compreenda como otimizar o desempenho da sua API e torná-la escalável para suportar um grande número de requisições.
6. **Microsserviços:** Explore o conceito de arquitetura de microsserviços, que é amplamente utilizado em aplicações modernas, e como o Spring Boot pode ser utilizado nesse contexto.
7. **Integração com outras tecnologias:** Aprofunde-se na integração da sua API com outras tecnologias, como bancos de dados NoSQL, serviços de mensageria e serviços de armazenamento em nuvem.

Lembre-se de que a prática constante é essencial para fortalecer suas habilidades de desenvolvimento. Construa projetos pessoais, participe de comunidades de desenvolvedores e esteja sempre aberto(a) a aprender com os outros.

Continue estudando e aprimorando suas habilidades, e com certeza você alcançará novos patamares em sua carreira.