Pós-Graduação Lato Sensu

Curso de Especialização em Tecnologia Java

Frameworks Web

Introdução a Frameworks

Prof. Esp. Hugo Baker Goveia

1. Divisão do BackEnd e FrontEnd

Nos últimos anos, o desenvolvimento de software evoluiu significativamente, com a necessidade crescente de aplicativos mais rápidos, escaláveis e interativos.
Com isso surgiu a necessidade de separar o Back-End do Front-End devido ao crescimento da complexidade dos aplicativos e sistemas web, especialmente com o advento da internet e a evolução das tecnologias de desenvolvimento.

Anteriormente, muitos softwares eram implementados com todas suas funcionalidades dentro do mesmo projeto, resultando em um código monolítico difícil de manter e escalar a longo prazo para alguns cenários.

Essa abordagem permitiu uma divisão clara de responsabilidades, especialização de competências e melhorias significativas na eficiência e qualidade do desenvolvimento de software.

RESPONSABILIDADES DO BACK-END E FRONT-END EM UMA APLICAÇÃO
O Back-End e Front-End possuem papéis diferentes na estruturação de aplicativos. Enquanto o Front-End lida com a interface do usuário e a interação do usuário, o Back-End trabalha nos bastidores para fornecer funcionalidades e gerenciar os dados.

O **Back-End** é responsável por gerenciar os aspectos técnicos de um aplicativo que os usuários não veem diretamente. Ele lida com o processamento de dados, a lógica de negócios e a comunicação com o banco de dados, visando fornecer funcionalidades para os usuários.
Ele é como o motor de um carro => invisível aos olhos do “usuário/motorista”, mas crucial para o funcionamento do veículo como um todo.
Entre as principais responsabilidades do Back-End estão o processamento de solicitações do cliente, a manipulação de dados e a execução de operações complexas. Uma das linguagens que usamos para elaborar um projeto de Back-End é o Java, ao qual temos como foco em nossa pós-graduação aqui na UTFPR.

O **Front-End** lida com a parte visível do aplicativo, incluindo a interface do usuário, o design e a interação do usuário. Ele é responsável pela experiência do usuário, também conhecida como lado do cliente.
Os desenvolvedores de Front-End trabalham com tecnologias como HTML, CSS e JavaScript para criar interfaces de usuário atraentes e funcionais. Além disso, eles podem utilizar frameworks como Angular ou Vue.js para simplificar o processo de desenvolvimento e melhorar a qualidade do código.

ESPECIALIZAÇÃO DE COMPETÊNCIAS
Com o tempo os desenvolvedores foram se especializando de acordo com as habilidades que mais se destacavam entre as áreas de Back-End e Front-End.

Alguns se tornaram desenvolvedores Front-End, lidando com criação de interfaces de usuários e outros se tornaram desenvolvedores Back-End, com foco em lidar com a lógica de negócios, manipulação de dados e gerenciamento de servidores. Separar essas responsabilidades permitiu que cada integrante da equipe se concentrasse em suas áreas de especialização.

MELHORIA NA EFICIÊNCIA DO DESENVOLVIMENTO

Com essa separação os desenvolvedores podem trabalhar de forma mais eficiente e colaborativa. Eles podem desenvolver e testar as interfaces de usuário independentemente da lógica de negócios, o que acelera o ciclo de desenvolvimento e permite uma iteração mais rápida.

REUTILIZAÇÃO DE CÓDIGO E COMPONENTIZAÇÃO

Outra vantagem com essa separação é a liberdade maior para os desenvolvedores criarem componentes reutilizáveis e modulares para Back-End e Front-End que podem ser facilmente integrados em diferentes partes de uma aplicação. Isso promove a consistência no design e na funcionalidade, facilitando a manutenção e a expansão do código ao longo do tempo.

ESCALABILIDADE E MANUTENÇÃO

A separação do Front-End e do Back-End facilita a escalabilidade de um aplicativo, permitindo que cada parte seja dimensionada independentemente, conforme necessário. Além disso, torna a manutenção do código mais gerenciável, pois as alterações em uma parte do aplicativo têm menos probabilidade de afetar outras partes.

2. Protocolo HTTP

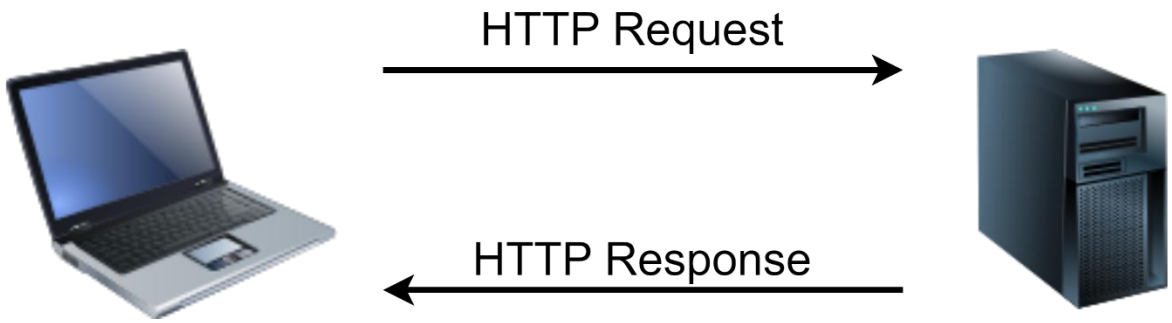
O HTTP(Hypertext Transfer Protocol) é um protocolo de comunicação cliente-servidor, onde um cliente, geralmente uma aplicação web Front-End, faz uso desse protocolo para fazer solicitações a um servidor contendo uma aplicação de Back-End para acessar seus recursos(APIs), como listagem de dados, imagens, arquivos, etc.

Essas solicitações são enviadas por meio de mensagens HTTP, que consistem em um cabeçalho e, opcionalmente, um corpo de mensagem.

Dessa maneira o protocolo HTTP é o método usado para as aplicações Front-End se comunicarem com as aplicações Back-End.

COMO É O FUNCIONAMENTO DO PROTOCOLO HTTP?

O HTTP opera no modelo de requisição(HTTP Request) e resposta(HTTP Response). Quando um cliente deseja acessar um recurso, ele envia uma solicitação HTTP para o servidor que hospeda o recurso desejado. O servidor processa a solicitação e envia uma resposta HTTP de volta ao cliente, contendo o recurso solicitado ou informações sobre o resultado da solicitação.



RECURSO

Um recurso é utilizado para identificar de forma única um objeto abstrato ou físico e é representado pela **URI** da **API** de Back-End que nosso Front-End ira consumir. Tal recurso como é um objeto deve ser representado por um substantivo e nunca por um verbo.
Exemplos de recursos: **/produtos** ou **/produtos/1**

Exemplo da diferença de Verbo e Substantivo:

Substantivo	Adjetivo	Verbo
alegria	alegre	alegrar
ambição	ambicioso	ambicionar
amor	amoroso	amar

A **representação do recurso** é o retorno que obtemos quando acessamos aquela determinada URI, como por exemplo na imagem abaixo, que temos o mesmo recurso representado de 3 formas diferentes (imagem, JSON, XML)

Então o JSON que recebemos de retorno quando batemos na API por exemplo, não é um recurso, mas sim a representação dele em formado de um JSON.

MÉTODOS HTTP

Toda requisição HTTP tem **um verbo**, ou seja, **um método** que podemos usar para realizar as operações. E nós desenvolvedores devemos usar esses métodos da melhor forma possível.

O HTTP define diversos métodos ou verbos que indicam a ação a ser realizada sobre um recurso.

Os métodos mais comuns são:

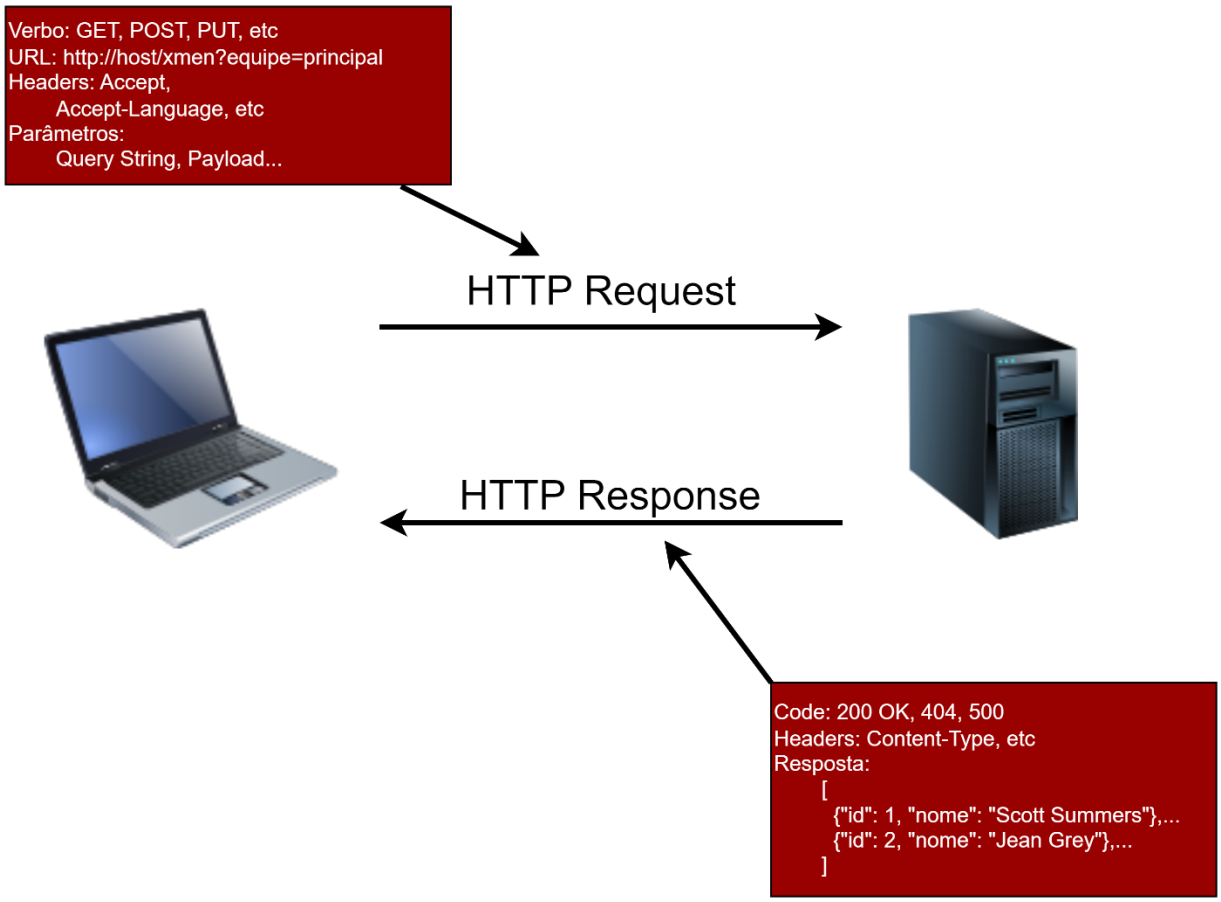
GET: Solicita a recuperação de um recurso, então usamos o método GET para obter algum retorno de nossa API.

POST: Envia dados para serem processados por um recurso, então usamos o método POST para criar algo novo.

PUT: Atualiza um recurso existente, então o método PUT usamos para atualizar algo.

DELETE: Remove um recurso, então usamos o método DELETE para deletar algo.

PATCH: Aplica modificações parciais a um recurso, então usamos o método PATCH para atualizar uma parte de algo.



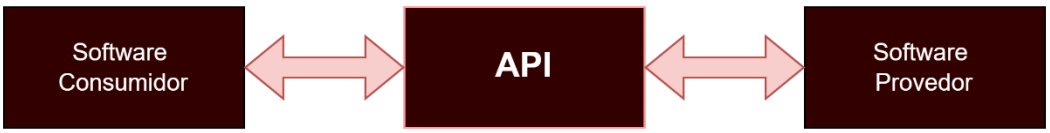
Por exemplo, para retornar uma pessoa de uma aplicação Back-End, nós poderiamos usar um **GET** para uma determinada **URL** que estaria disponível para fazer essa chamada, podendo passar parâmetros se necessários para filtrar o que desejamos buscar.
No Headers dessa requisição nós podemos configurar várias formas de retorno.

RETORNOS DO MÉTODO (Código Status HTTP)
Os retornos são os códigos e as respostas que o servidor retorna ao cliente diante de uma requisição. Os principais são:

- **1XX** - Informações
- **2XX** - Sucesso
  Exemplo=> ***200 OK: Indica que a solicitação foi bem-sucedida.***
- **3XX** - Redirecionamento
- **4XX** - Erro no cliente
  Exemplo=> ***404 Not Found: Indica que o recurso solicitado não foi encontrado no servidor.***
- **5XX** - Erro no servidor
  Exemplo=> ***500 Internal Server Error: Indica um erro interno no servidor ao processar a solicitação.***

3. API (Application Programming Interface)

Uma API (Application Programming Interface) é um conjunto de definições e protocolos que permite a comunicação e a interação entre diferentes softwares. Ela define métodos, estruturas de dados e convenções que os desenvolvedores podem utilizar para integrar seus aplicativos com serviços externos, bibliotecas ou plataformas.
A API indica a maneira correta de um desenvolvedor escrever um programa solicitando serviços de um sistema operacional ou de outra aplicação.



Dessa forma podemos concluir que uma API funciona como uma ponte entre dois sistemas de software, permitindo que eles se comuniquem e compartilhem informações de forma padronizada e controlada, definindo as operações que podem ser realizadas, os formatos de dados que podem ser enviados e recebidos, e as regras para autenticação e autorização de acesso.

CONSUMIDOR x PROVEDOR API
Nosso software pode ser classificado na situação de ser um **consumidor** ou um **provedor API**.
-**Exemplo da condição de PROVEDOR API**: Imaginemos que temos um software e queremos expor alguma funcionalidade dele, então ficamos no papel de provedor e para contemplar essa condição de ser um provedor de algo nós criamos uma API que API vai expor essa funcionalidade para quem vai consumir.

Que nesse sentido serão outros softwares que irão consumir essa funcionalidade, podendo ser softwares de terceiros ou até mesmo interno da própria empresa que temos ou trabalhamos.

-**Exemplo da condição de CONSUMIDOR:** às vezes precisamos usar alguma funcionalidade em nosso software que já foi implementada por outro desenvolvedor que disponibilizou para nós em uma API, nessa situação apenas usamos essa API. Com isso ganhamos temos de desenvolvimento.

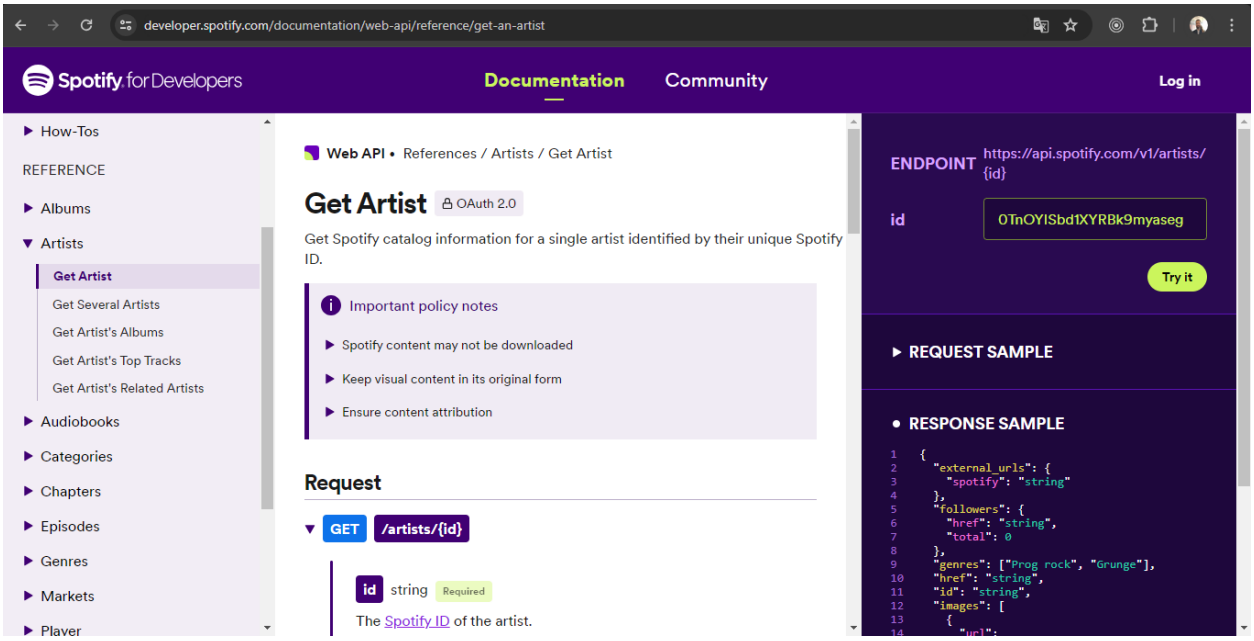
Uma API não necessariamente pode servir um público externo, podendo ser desenvolvida para uso dos desenvolvedores da própria empresa, sendo usada internamente fornecendo funcionalidades para a própria empresa.
Então podemos concluir que uma API nos fornece a interface para que acessemos interfaces de um outro sistema.

EXEMPLOS DE USO DE API

No exemplo acima imaginemos os desenvolvedores Java usando a API de Collections do Java para poder aplicar no desenvolvimento dos softwares evitando tendo de desenvolver funcionalidades que são muito usadas no dia a dia do programador Java.

O uso de API é bem amplo, não ficando restrito apenas ao uso na WEB, por exemplo, se temos uma classe que está expondo alguns métodos públicos, podemos certamente dizer que essa é a API da nossa classe.

No spotify por exemplo temos o sistema do Spotify que disponibiliza uma API do Spotify. Isso permite que nosso projeto use essa API como consumidor, para se beneficiar dos recursos que o sistema do Spotify desenvolveu, que nesse caso é o álbum de músicas do Spotify com artistas e tudo mais.



Para isso, basta acessar a documentação da API do spotify e fazer uso delas no nosso software, onde podemos desenvolver novas funcionalidades e disponibilizar para nossos usuários.

4. WebServices e API

Um WebService é uma API que fornece sua interface de comunicação pela Internet. Portanto temos que todo WebService(serviço web) é uma API, sendo assim entendemos que é uma API que realiza sua comunicação e acesso é pela WEB.
Quando falamos de WebServices, estamos falando de WEBAPI e em alguns lugares até encontramos a referência como WEBAPI.

É importante entendermos que todo WebService é uma API, mas nem toda API é um WebService, ou seja, nem toda API será disponibilizada para utilização via web.

Existem milhares de APIs disponibilizadas para o mundo todo, públicas ou privadas, e dificilmente nos dias de hoje uma aplicação é desenvolvida sem consumir pelo menos uma WebApi(WebService). Pois tudo está muito conectado, sendo que precisamos desenvolver ou consumir algum WebService para concluirmos um projeto hoje em dia.

5. REST API

REST significa Representational State Transfer, basicamente é uma forma de integrar sistemas. Por exemplo, o frontEnd é um sistema e o BackEnd é outro sistema e para integrar eles usamos o REST.

REST é um estilo arquitetural que usamos para desenvolver uma WebAPI(WebServices), mas devemos ficar atentos para não confundir com FRAMEWORK, pois REST não é um FRAMEWORK que podemos baixar e usar em nosso projeto.
Portanto são regras arquiteturais que devemos seguir. O REST foi desenhado para usar protocolos que já existem, como o HTTP. Onde temos que seguir algumas regras, que são mais conhecidas como CONSTRAINTS.

Para uma aplicação consumir uma REST API, ela precisa fazer uso do protocolo HTTP não precisando instalar nada, e essa facilidade de integração entre os sistemas fez com que o REST se tornasse a forma mais popular para desenvolver webservices.

REST é uma arquitetura para sistemas distribuídos que possui seis restrições a serem respeitadas para considerarmos nossa API Restful.
Além do REST existe o SOAP (Simple Object Access Protocol) entretanto o REST usa menos largura de banda, tornando-o mais adequado para o uso eficiente da Internet por isso é o escolhido para a maioria dos projetos hoje em dia.

O REST é uma opção lógica para criar Web APIs que permitem que os usuários se conectem, gerenciem e interajam com os serviços na nuvem de maneira flexível em um ambiente distribuído.

API RESTful
Uma API pode ser considerada RESTful quando ela utiliza em sua implementação o conceito arquitetural REST.
REST é algo abstrato, como um modelo arquitetural enquanto que RESTful é algo mais concreto, como a implementação deste modelo em alguma API. Então, para criar uma API RESTful é preciso conhecer a arquitetura REST e também a aplicar corretamente.
Uma API RESTful é uma interface que usa solicitações HTTP através de requisições GET, PUT, POST e DELETE.

Arquitetura REST
O modelo arquitetural REST nada mais é que um conjunto de boas práticas para que determinado aplicativo possa ser construído e documentado de uma maneira padronizada, fácil e que gere maior produtividade tanto para os desenvolvedores na construção, quanto para o consumo desse aplicativo por outros sistemas.

Ela possui seis constraints(regras) que devem ser obrigatoriamente seguidas para que uma aplicação seja considerada RESTful.

PRIMEIRA CONSTRAINT => A primeira constraint diz que uma aplicação, neste caso uma API, deve ser cliente/servidor a fim de separar as responsabilidades.

SEGUNDA CONSTRAINT => Diz que essa aplicação deve ser stateless, ou seja, não guardar estado no servidor, isso é importante para garantir que cada requisição que o cliente enviar ao servidor

tenha informações relevantes e únicas em sua resposta independe de qual servidor irá responder a esse cliente caso a aplicação esteja escalada em múltiplos servidores, garantindo assim a escalabilidade e disponibilidade.

TERCEIRA CONSTRAINT=> Define que a aplicação deve ter a capacidade de realizar cache para reduzir o tráfego de dados entre cliente e servidor.

QUARTA CONSTRAINT => A aplicação deve ter uma interface uniforme, onde sua modelagem deve conter recursos bem definidos, apresentar hipermídias, utilizar corretamente os métodos HTTP e códigos de retorno

QUINTA CONSTRAINT => O sistema deve ser construído em camadas, ou seja, com a possibilidade de escalar a aplicação em múltiplos servidores.

SEXTA CONSTRAINT => A aplicação deve ter a capacidade de evoluir sem a quebra dela, ou seja, código sob demanda.

TIPOS DE REPRESENTAÇÕES EM REST
Uma API REST pode utilizar 2 padrões de comunicação, XML e JSON. O padrão XML se baseia em tags, sendo um pouco mais pesado quando comparado ao JSON.

6. Frameworks

O que é um framework?
Um framework é uma estrutura abstrata que fornece funcionalidades genéricas e reutilizáveis para facilitar o desenvolvimento de software. Ele define uma estrutura ou esqueleto para a aplicação, determinando a arquitetura, o fluxo de controle e as convenções de codificação. Os frameworks geralmente incluem uma série de componentes pré-definidos, como classes, métodos e APIs, que os desenvolvedores podem utilizar para criar aplicativos mais rapidamente.

E o que é uma biblioteca?
Por outro lado, uma biblioteca é um conjunto de funções ou rotinas reutilizáveis que podem ser usadas para realizar tarefas específicas, como manipulação de strings, processamento de dados ou geração de gráficos. As bibliotecas não impõem uma estrutura ou fluxo de controle específico, mas fornecem funcionalidades específicas que os desenvolvedores podem integrar em seus aplicativos conforme necessário.

Diferença entre Framework e Biblioteca:
A principal diferença entre um framework e uma biblioteca é o nível de controle que eles fornecem ao desenvolvedor.
Enquanto um framework controla o fluxo de execução da aplicação e dita a arquitetura geral, uma biblioteca oferece funcionalidades específicas que podem ser integradas ao código do aplicativo conforme necessário.

Tratando-se da biblioteca é o desenvolvedor que chama a biblioteca, e em relação ao framework é o framework que chama o código do desenvolvedor.

Framework de Back-End e Framework de Front-End

Um **framework de Back-End** é um conjunto de ferramentas, bibliotecas e padrões de desenvolvimento projetados para facilitar a criação de aplicativos e serviços na camada do servidor. Ele fornece funcionalidades para lidar com a lógica de negócios, o processamento de dados, a comunicação com o banco de dados e outros aspectos relacionados ao servidor. Exemplo => Spring.

Já um **framework de Front-End** é um conjunto de ferramentas, bibliotecas e padrões de desenvolvimento destinados a facilitar a criação de interfaces de usuário e interações do usuário em aplicativos web ou móveis. Ele fornece funcionalidades para criar e estilizar elementos HTML, gerenciar o estado da aplicação, manipular eventos do usuário e interagir com o servidor. Exemplo => Angular e Vue.js

FRAMEWORK WEB

Um framework web é uma estrutura de software projetada para facilitar o desenvolvimento de aplicativos web, que oferece um conjunto de ferramentas, bibliotecas e padrões de desenvolvimento que permitem aos desenvolvedores criar aplicativos de forma mais rápida e eficiente.

Vantagens de usar um framework web

Produtividade: Um framework web oferece funcionalidades pré-construídas e convenções de codificação que permitem aos desenvolvedores criar aplicativos mais rapidamente, reduzindo a necessidade de escrever código repetitivo.

Padronização: Os frameworks web geralmente seguem padrões de desenvolvimento estabelecidos, o que facilita a colaboração entre desenvolvedores e a manutenção do código ao longo do tempo.

Segurança: Muitos frameworks web incluem recursos de segurança integrados, como proteção contra ataques de injeção de SQL, XSS (Cross-Site Scripting) e CSRF (Cross-Site Request Forgery), ajudando a proteger o aplicativo contra vulnerabilidades de segurança comuns.

Escalabilidade: Os frameworks web são projetados para facilitar o dimensionamento do aplicativo à medida que a demanda cresce, permitindo que os desenvolvedores adicionem novos recursos e otimizem o desempenho conforme necessário.

Comunidade e Suporte: Os frameworks web geralmente têm uma comunidade ativa de desenvolvedores que contribuem com bibliotecas, plugins e documentação, facilitando a resolução de problemas e o aprendizado contínuo.

7. Angular

Na nossa disciplina iremos adotar o Framework web Angular. Ele é um framework de Front-End desenvolvido e mantido pelo Google. Ele é amplamente utilizado para a construção de interfaces de usuário dinâmicas e interativas em aplicativos web.

Características do Angular

Arquitetura MVC: Angular segue o padrão de arquitetura MVC (Model-View-Controller), que divide a aplicação em três componentes principais: Model (Modelo), View (Visualização) e Controller (Controlador). Isso facilita a separação de preocupações e a organização do código.

Two-Way Data Binding: Angular oferece suporte a two-way data binding, o que significa que as alterações feitas no modelo de dados são refletidas automaticamente na interface do usuário e vice-versa. Isso simplifica a sincronização de dados entre o modelo e a visualização.

Injeção de Dependência: Angular possui um sistema de injeção de dependência embutido, que permite aos desenvolvedores gerenciar as dependências entre os componentes do aplicativo de forma fácil e eficiente.

Componentização: Angular incentiva a criação de aplicativos modulares e reutilizáveis através da componenteização. Os desenvolvedores podem dividir a interface do usuário em componentes independentes e reutilizáveis, facilitando a manutenção e o desenvolvimento de aplicativos complexos.

Ecossistema Rico: Angular possui um ecossistema rico de ferramentas, bibliotecas e plugins que facilitam o desenvolvimento de aplicativos web. Além disso, o Angular possui uma documentação abrangente e uma comunidade ativa de desenvolvedores que contribuem com recursos e suporte.