Introdução ao Desenvolvimento web usando Java e Spring boot

1. Desenvolvimento: Back-end e Front-end

1.1. Front-end

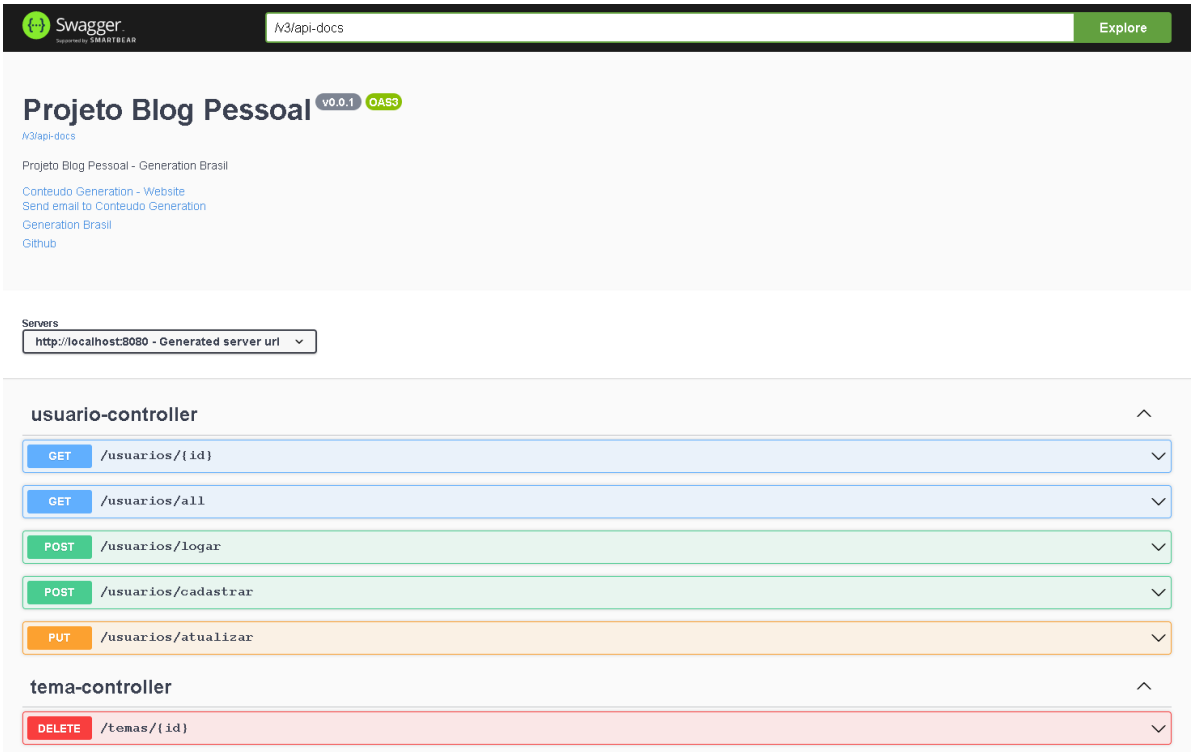
O **Front-end**, de forma sucinta, é toda parte visual de um site, a parte com a qual o usuário interage diretamente, ou seja, o código do sistema que roda no navegador.

O profissional responsável por trabalhar nessa área de um projeto desenvolve o código para a interface gráfica, normalmente por meio de linguagens. Um desenvolvedor Front-end normalmente trabalha criando toda a parte visual dos sites por meio de linguagens de marcação (HTML), estilização (CSS) e programação (JavaScript e TypeScript), além de bibliotecas (React e Bootstrap) e frameworks (Angular, VUE.js e NEXT.js). Abaixo vemos a página principal de um Front-end que irá consumir os Recursos e os Endpoints de um Back-end.

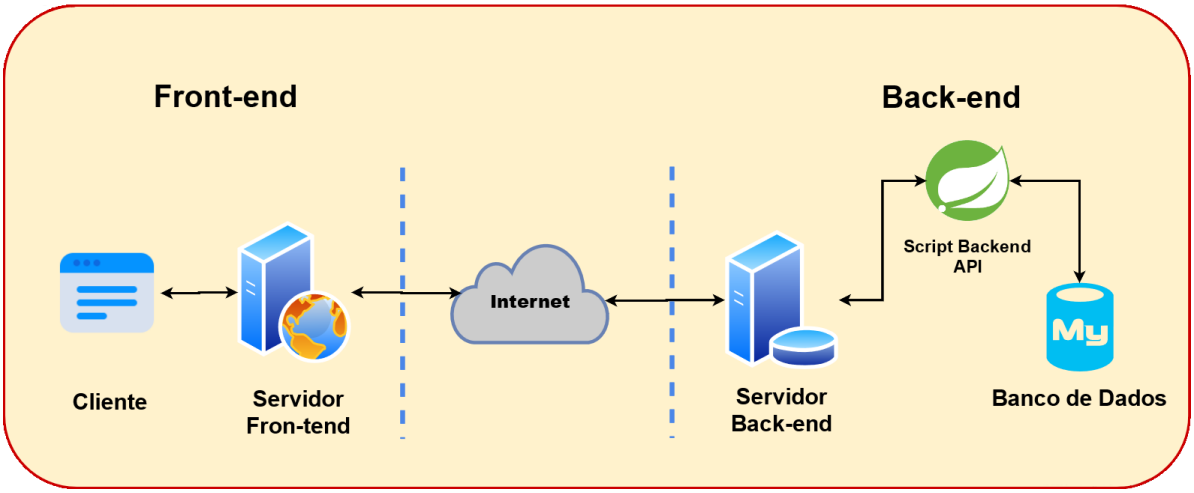
1.2. Back-end

Apesar de mais abstrato, o conceito de **Back-end** também é simples de entender: os dados publicados em uma rede social, como fotos e vídeos, por meio da interface do usuário precisam ser armazenados (persistidos) em algum local para poderem ser acessados posteriormente. Esse envio não é feito diretamente do Front-end para o Banco de dados da rede social. Ao invés disso, existe uma parte da aplicação que é responsável por receber essas informações, fazer operações específicas — postagens, filtros, exclusões, verificações de segurança e validações e somente após isso, armazenar no Banco de dados. Esse é o Back-end.
As atribuições de um desenvolvedor Back-end não incluem a criação de páginas, como acontece com profissionais Front-end. Em vez disso, eles são responsáveis por implementar arquiteturas robustas, que se comuniquem com o Banco de dados (MySQL, PostgreSQL, SQL Server, ou Oracle) e que garantam a segurança dos dados enviados pelo usuário. Para isso utiliza linguagens

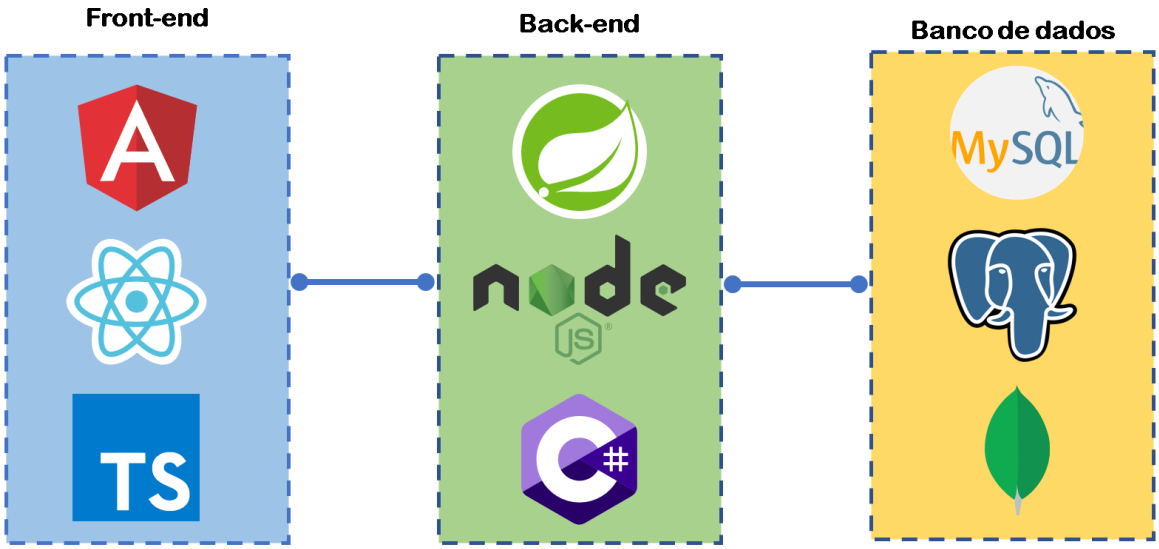
de programação como o Java e o Framework Spring. Abaixo vemos uma imagem da Documentação de um Backend contendo todos os seus Recursos e respectivos Endpoints (funcionalidades).



Na ilustração abaixo vemos o Front-end e o Back-end da aplicação funcionando de forma independente, se comunicando através da Internet, via requisições HTTP.



Na ilustração abaixo vemos algumas soluções para o Desenvolvimento do Front-end, Back-end e Banco de dados:

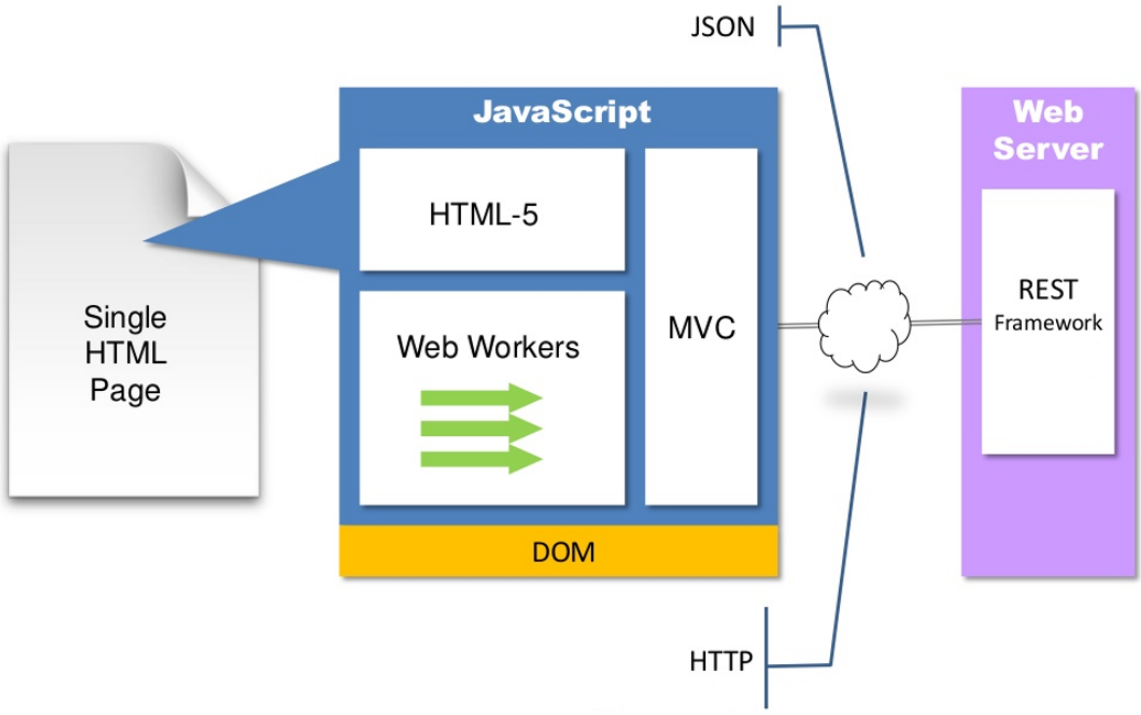


1.3. Desenvolvedor(a/e) Full Stack:

Quem trabalha tanto no Desenvolvimento do Front-end quanto no Desenvolvimento do Back-end é conhecido como **"Desenvolvedor (a/e) Fullstack"** ou **"Programador (a/e) Fullstack"**. Esse profissional é capaz de entregar um projeto do início ao fim, sem necessariamente precisar de ajuda de outras pessoas desenvolvedoras para criar uma parte do sistema, dependendo do seu tamanho, complexidade, tempo e recursos disponíveis.

2. SOFEA

SOFEA (Service Oriented Front-end Architecture - Arquitetura de Serviço Orientado ao Front-end) é uma arquitetura para Desenvolvimento WEB em Ambientes Cloud, onde o Front-end é independente do Back-end e inclusive pode estar hospedado em um Servidor diferente do Back-end. Nesta arquitetura, a lógica de apresentação (código do cliente) é desacoplada da lógica de negócios (Banco de dados e o Back-end), e a interface do usuário obterá dados dos serviços em a forma de JSON \ XML com a ajuda de solicitações HTTP Requests e Respostas HTTP Response.



A arquitetura SOFEA é desenvolvida com o auxílio do SPA (Single Page Application). Na Primeira solicitação, todo o HTML , CSS e JavaScript é baixado e, em seguida, cada página solicitada é ligada com os dados que vêm com a ajuda de Requisições HTTP.

2.1. O que é Single Page Application (SPA)?

São aplicações cuja funcionalidade está concentrada em uma única página. Ao invés de recarregar toda a página ou redirecionar o usuário para uma página nova, apenas o conteúdo principal é atualizado de forma assíncrona, mantendo toda a estrutura da página estática. Imagine um dashboard, em que os menus lateral e superior são os mesmos para todas as telas da aplicação. Ao clicar em uma opção como "Cadastro de produtos", o usuário não precisaria recarregar toda a página para ver que no fim apenas o conteúdo central mudou. Para evitar isso, mantemos os menus fixos e alteramos apenas a parte do meio, em que estarão os formulários, tabelas, etc.

2.2. Benefícios da Arquitetura SOFEA

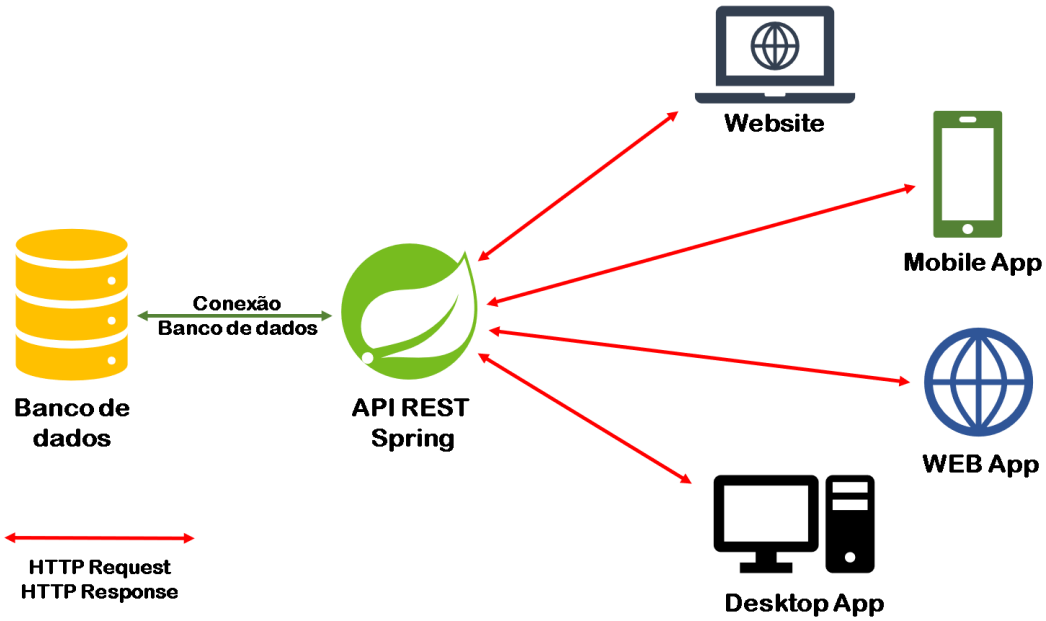
- A camada de apresentação é desacoplada da camada de negócios, o que ajuda o aplicativo a obter uma resposta mais rápida.
- Executa em uma variedade de dispositivos.
- Muito bem suportado em ambiente de nuvem.
- Escalabilidade - o servidor tem menos trabalho a fazer; chega de geração de apresentação, apenas fornece serviços
- Melhor resposta do usuário - Baixa latência = usuários finais felizes - Após o download do aplicativo, nenhuma apresentação é transportada pela rede, apenas dados de negócios
- Ajuste natural em ambiente Cloud
- Modelo de programação organizado - Os desenvolvedores do cliente se concentram na Interface com o Usuário - os desenvolvedores do lado do servidor se concentram na lógica do negócio
- Aplicativos off-line - quando a rede falha, o cliente desacoplado pode alternar dinamicamente seus objetos de modelo
- Interoperabilidade - Integração mais fácil com menor sobrecarga de várias plataformas - Os clientes não se importam se os serviços são Java, C #, Python, ou uma combinação heterogênea

3. API REST

Primeiro vamos compreender os 2 conceitos:

API é um acrônimo que provém do inglês Application Programming Interface (Interface de Programação de Aplicações), que se trata de um conjunto de rotinas e padrões estabelecidos e documentados por uma determinada aplicação, para que outras aplicações consigam utilizar rodas as suas funcionalidades sem precisar conhecer os detalhes da implementação do software.

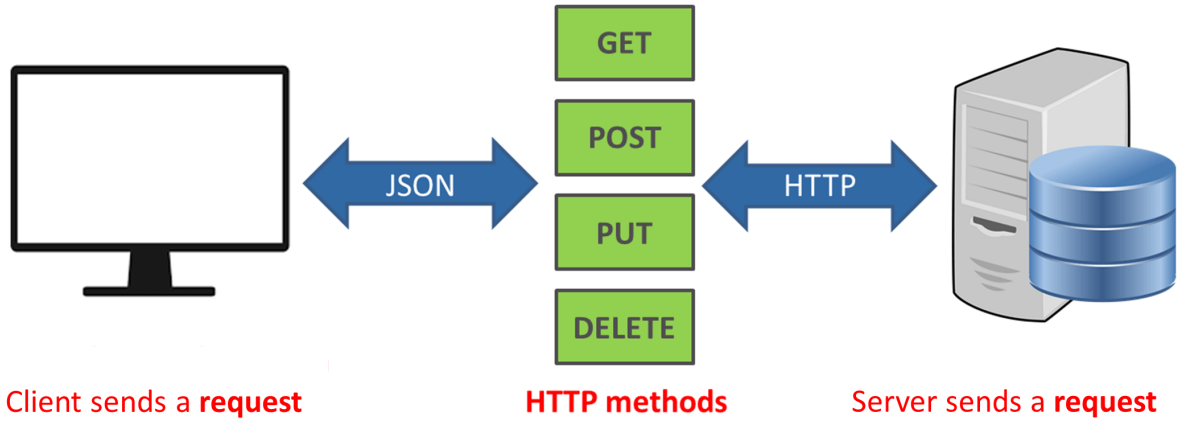
Desta forma, entendemos que as API's permitem uma interoperabilidade entre aplicações. Em outras palavras, a comunicação entre aplicações e entre os usuários de uma API pode receber requisições de diversas aplicações distintas (Web, Mobile, Desktop e até mesmo de uma outra API).



REST é um acrônimo que provém do inglês Representational State Transfer (Transferência de Estado Representacional), que criado no ano 2000 por Roy Fielding, em sua tese de doutorado, na qual ele descreve um design de arquitetura de software construído para servir aplicações em rede.

REST é uma arquitetura utilizada para integrar o Back-end com o Front-end, através do protocolo HTTP. Uma API Rest se comunica com o Front-end através do envio de Requests (Requisições) e do e recebimento de Responses (Respostas).

Estas Requests e Responses, são compostas por Objetos no formato JSON, em conjunto com os métodos GET, POST, PUT e DELETE conforme vimos no modulo de Ciência da Computação.



3.1. Formato JSON

JSON (JavaScript Object Notation) é um modelo para armazenamento e transmissão de informações no formato texto. Apesar de muito simples, tem sido bastante utilizado por aplicações Web devido a sua capacidade de estruturar informações de uma forma bem mais compacta do que o modelo XML, tornando mais rápido a interpretação dessas informações. Isto explica o fato de o JSON ter sido adotado por empresas como Google e Yahoo, cujas aplicações precisam transmitir grandes volumes de dados.

```
{
      "id": 1,
      "titulo": "Minha primeira postagem",
      "texto": "Blog Pessoal funcionando!",
      "data": "2021-06-20",
      "tema": {
            "id": 1,
            "descricao": "Spring",
      }
}
```


A ideia utilizada pelo JSON para representar informações é tremendamente simples:para cada valor representado, atribui-se um nome (ou rótulo) que descreve o seu significado. Esta sintaxe é derivada da forma utilizada pelo JavaScript para representar informações.


[Site Oficial do JSON](#)

[JSON Viewer - Chrome](#)

[JSON Viewer - Edge](#)

3.2. Rest x RestFul

Podemos dizer que a API REST e a RESTful têm objetivos distintos, mas complementares.

REST (representational state transfer) é como um guia de boas práticas. Uma vez que é um modelo de arquitetura de software que define uma série de requisitos para que as APIs sejam desenvolvidas, ou seja, é uma base para a construção de APIs que conecta os usuários com as aplicações em nuvem.

RESTful é a capacidade de um determinado sistema aplicar os principios REST

Os principais critérios para uma API ser **RESTful** são:

- **Hipermídias (HATEOAS):** Basicamente, **sua API deve ser um livro aberto**, você não deve precisar de acesso a documentação para saber que para adicionar um usuário no Banco de dadoas, precisará de uma requisição POST para http://locahots:8080/usuarios/cadastrar. Deve ser possível descobrir todas as manipulações de seus recursos através da própria **API**;
- **Stateless:** Significa que cada solicitação HTTP ocorre em isolamento completo. Quando o cliente faz uma Requisição HTTP, ele inclui todas as informações necessárias para o servidor atender à solicitação. O servidor nunca depende de informações de solicitações anteriores do cliente. Se alguma dessas informações for importante, o cliente a enviará como parte da solicitação atual;
- **Cacheable:** Mantém em cache as Respostas das últimas Requisições;
- **Cliente-Servidor;**
- **Layered system:** O Desenvolvimento da aplicação é distribuído em Camadas com funções pré-determinadas.

Sem seguir todas estas regras, sua API não será RESTful, será apenas mais uma **implementação** em cima do protocolo HTTP. O primeiro ítem da lista, Hipermídias, é o item menos implementado, por isso que poucas API's são do tipo Restful.

Para definir se uma API é Rest ou Restful, foi criado o Modelo de Maturidade de Richardson:

0. Uma API que não atenda nenhuma das regras
1. **Recursos**: A partir do momento em que seja possível fazer requisições de diferentes recursos em diferentes *endpoints*.
2. **Verbos HTTP**: Aqui, os diferentes métodos **HTTP** são colocados em prática, em contraposição ao uso quase exclusivo do **POST** no protocolo **SOAP**. Além disso, cada verbo possui sua utilidade específica: **PUT** para atualizar, **DELETE** para excluir, **GET** para adquirir e **POST** para criar. Em alguns casos o **PATCH** também é utilizado.
3. **Hypermedia**: Os recursos passam a possuir links para recursos relacionados, além de links para realizar ações em cima dessas coleções, a partir desse ponto, a API se auto-documenta e possibilita a funcionalidade de descoberta.

```
{
    "id": 1,
    "titulo": "Minha primeira postagem",
    "texto": "Texto da postagem",
    "data": "2021-06-23",
    "links": {
        "mostrarListaPostagens":{
            "href": "/postagens"
        }
}
```

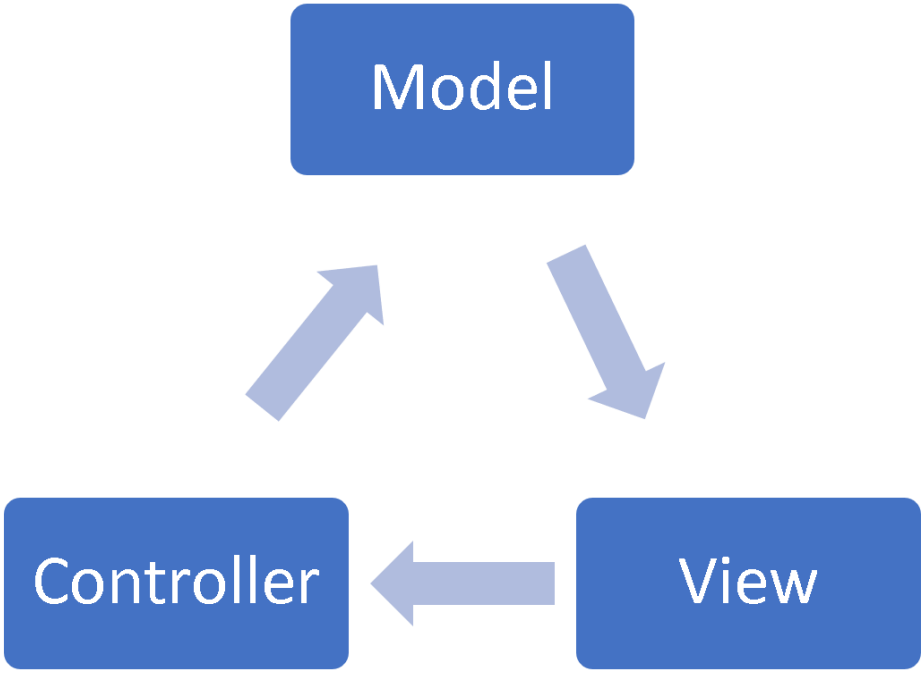
Na imagem acima, vemos um JSON de uma API RESTful com Hypermídia. Observe que além do Objeto, também foi retornado um link indicando o Endpoint que lista todos os Objetos da Classe Postagem.

No Bootcamp implementaremos API's REST (Nível 2).

4. Arquitetura MVC

MVC é acrônimo de **Model, View e Controller**, é um padrão de arquitetura de software responsável por contribuir na otimização da velocidade entre as requisições feitas pelo comando dos usuários.

A arquitetura MVC é dividida em três componentes essenciais:



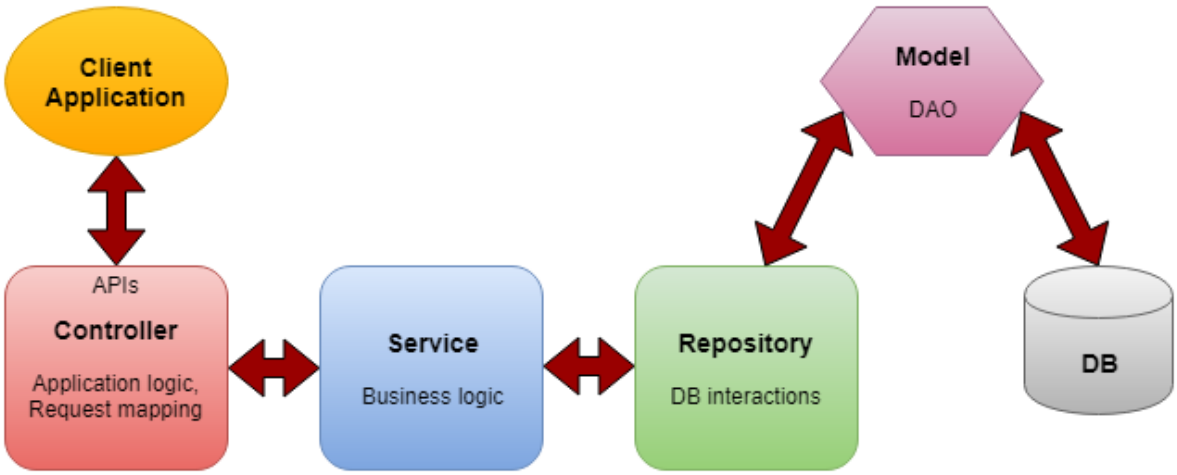
Model (Modelo): Sua responsabilidade é gerenciar e controlar a forma como os dados se comportam por meio das funções, lógica e regras de negócios estabelecidas.

Controller (Controladora): A camada de controle é responsável por intermediar as requisições enviadas pelo View com as respostas fornecidas pelo Model, processando os dados que o usuário informou e repassando para outras camadas.

View (Visão): Essa camada é responsável por apresentar as informações de forma visual ao usuário. Em seu desenvolvimento devem ser aplicados apenas recursos ligados a aparência como mensagens, botões ou telas.

Um dúvida muito recorrente na programação é se no processo de desenvolvimento pode ter apenas esses 3 componentes ou se é possível acrescentar mais alguns. A resposta é sim para a possibilidade de inserir uma camada extra. Essa sequência de códigos pode ser alterada conforme a necessidade do projeto.

4.1 Arquitetura básica de uma WEB API Spring



Camada	Descrição
Model - Entity	Camada responsável pela abstração de nossos objetos e tabelas em nossos Bancos de dados, ou seja, cria as Tabelas no Banco de dados.
Repository - DAO	Interface responsável pela comunicação direta com o banco de dados.
Controller	Camada responsável por manipular todas as requisições feitas do lado de fora da nossa API. Essas requisições são realizadas através de URL's utilizando o protocolo HTTP.
Service	Camada responsável por toda regra de negócio e tratativa de dados, sempre seguindo o modelo de negócio da aplicação.
Security	Camada responsável por implementar a segurança da aplicação
Configuration	Camada responsável por implementar configurações específicas da aplicação, como a Documentação da aplicação (Swagger), por exemplo.