

# Introdução ao desenvolvimento web usando Java e Spring boot

---



## Diferença entre Back end e Front end

---

### Front-end:

O front-end é todo o código da aplicação responsável pela apresentação do software (client-side).

Em se tratando de aplicações web, é exatamente o código do sistema que roda no navegador.

Um desenvolvedor *front-end*, geralmente, trabalha com linguagens como HTML, CSS e JavaScript, além de frameworks e bibliotecas, como por exemplo Angular, React, Vue.js, etc.

criar páginas ou telas com boa usabilidade e carregamento rápido, garantir o funcionamento nos diferentes navegadores, integrar com os serviços do \_back-end, etc.

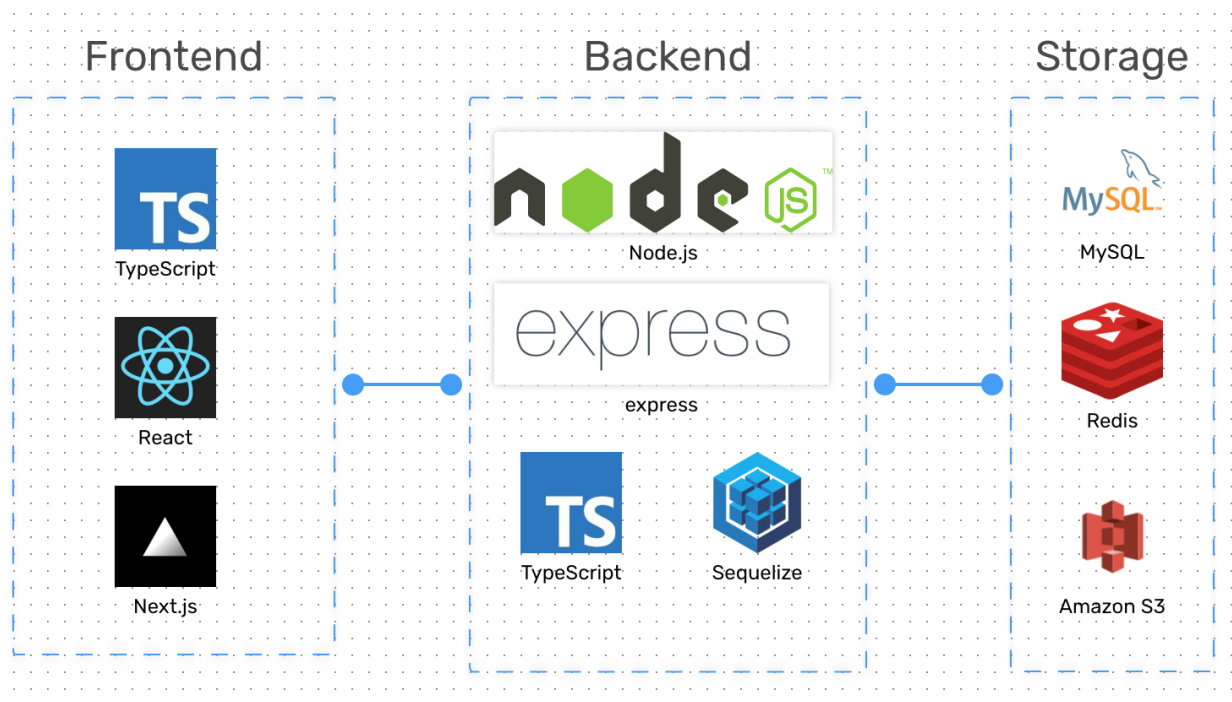
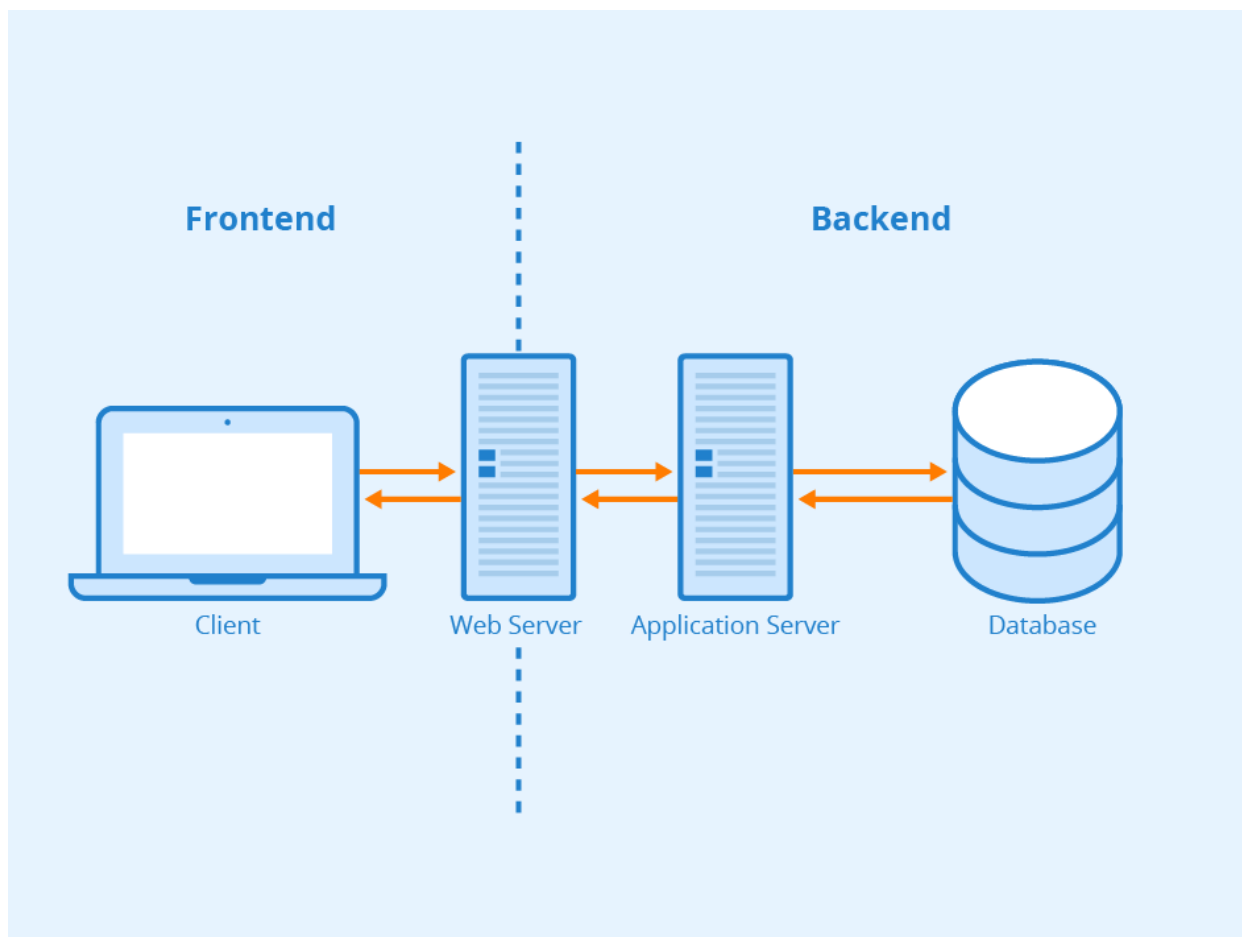
### Back-end:

**back-end** é a parte do software que roda no servidor, por isso também é conhecida como *server-side*.

É o **back-end** que fornece e garante todas as regras de negócio, acesso a banco de dados, segurança e escalabilidade.

## Ilustração Front-End/Back-End

---



Obs: mostrar um projeto pronto.

**Fullstack:**

Quem trabalha tanto com *front-end* quanto *back-end* é conhecido como “Desenvolvedor **Fullstack**” ou “**Programador Fullstack**”.

Esse é um tipo de profissional pode entregar um projeto do início ao fim, sem necessariamente precisar de ajuda de outra pessoa para criar uma parte do sistema.

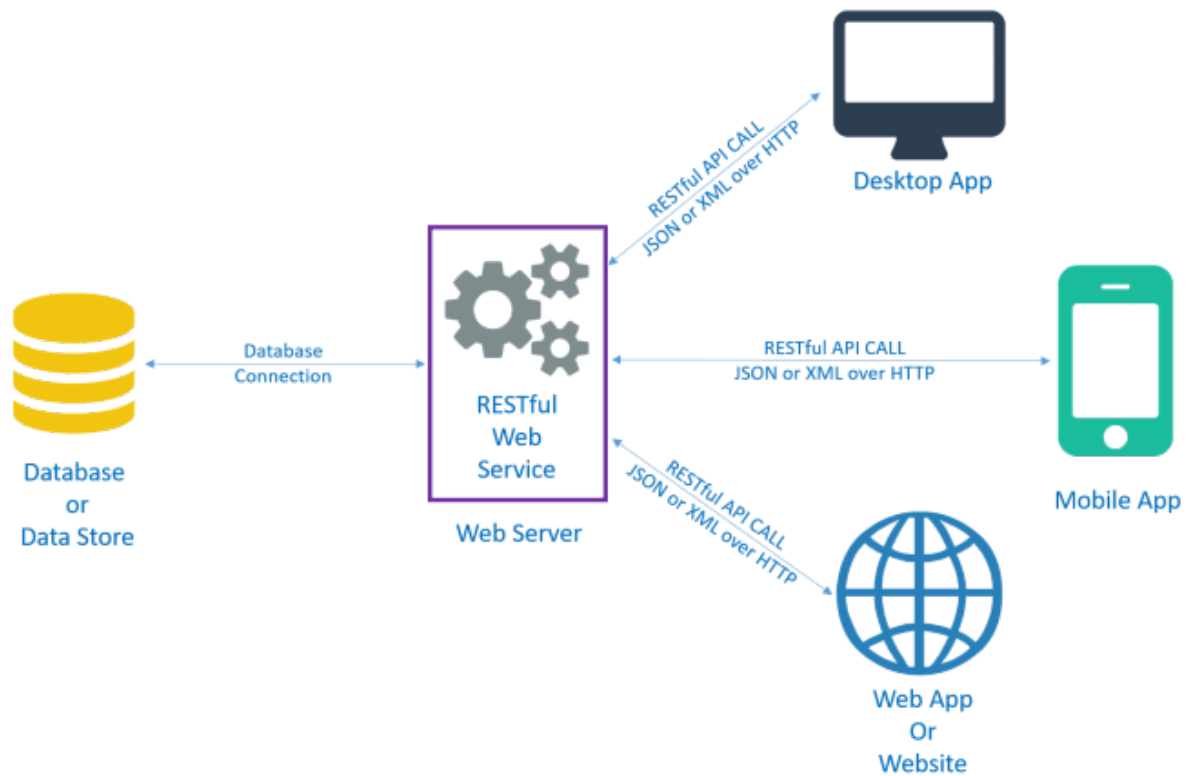


## O que é uma API REST

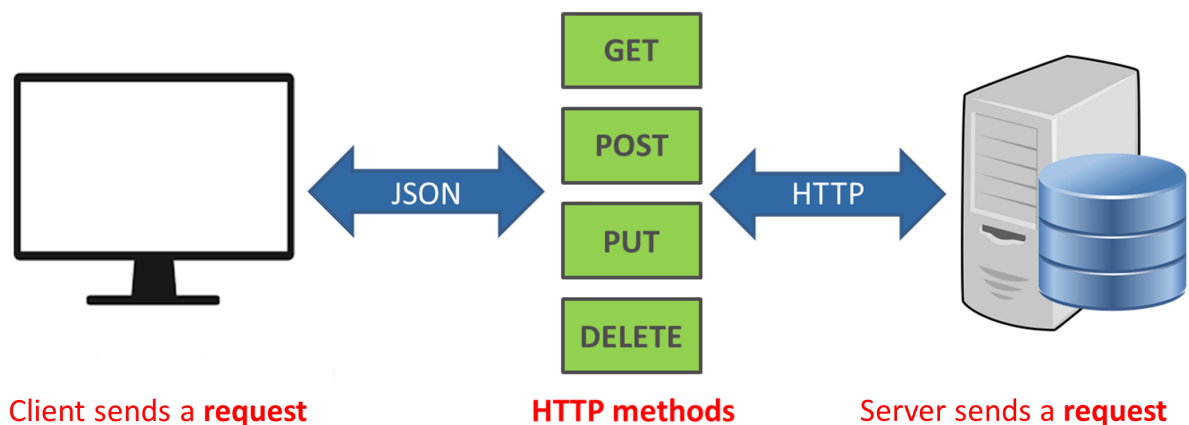
---

- ❑ **REST** Representational state transfer “Transferência de Estado Representacional” criado no ano 2000 por Roy Fielding em sua tese de doutoramento na qual ele descreve um design de arquitetura de software construído para servir aplicações em rede.
- **REST** é uma arquitetura utilizada para integrar **Back end** com o **Front end**, se comunicando por meio de protocolo HTTP.
- **Api REST** O acrônimo **API** que provém do inglês **Application Programming Interface** (Em português, significa Interface de Programação de Aplicações), trata-se de um conjunto de rotinas e padrões estabelecidos e documentados por uma aplicação A, para que outras aplicações consigam utilizar as funcionalidades desta aplicação A, sem precisar conhecer detalhes da implementação do software.

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



Uma API Rest se comunica com o Front end através do envio e recebimento (request, response) de Objetos nos formatos **XML** ou **JSON** ( veremos Json neste curso), juntamente com os métodos **GET**, **POST**, **PUT** e **DELETE** conforme vimos no modulo de Ciência da Computação.



## 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 a conseguida pelo modelo XML, tornando mais rápido o parsing 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

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.

---

## EXEMPLO

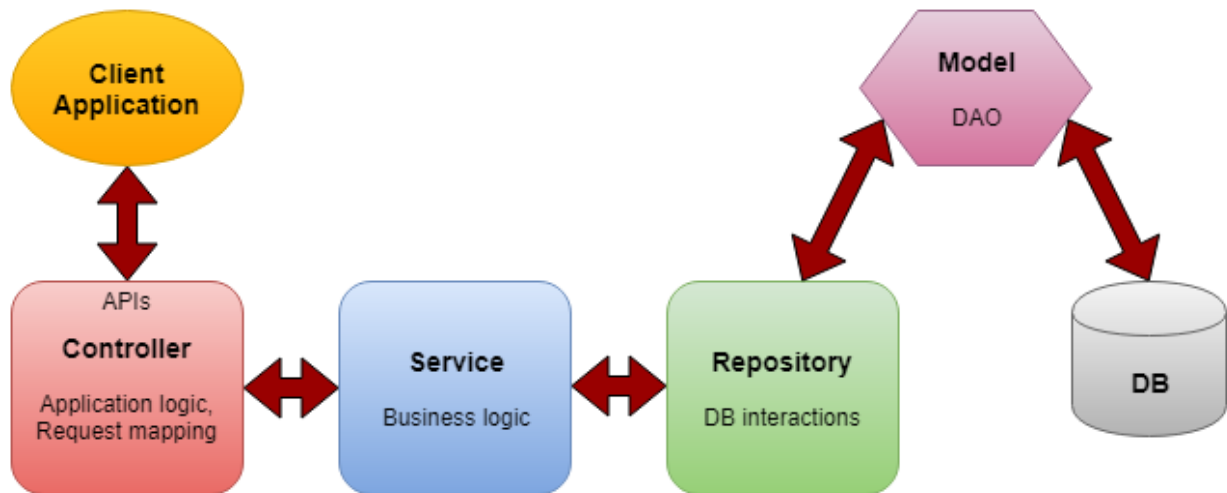
```
{
  "id": 1000501,
  "name": "Wilson Júnior",
  "city": {
    "name": "Rio de Janeiro",
    "state": "RJ"
  },
  "age": 24
}
```

## Camadas básicas de um projeto da Web API

- ☐ **Model/Entity** - classe responsável pela abstração de nossos objetos e tabelas em nossos banco de dados.
- ☐ **Repository/DAO** -Interface responsável pela comunicação direta com o banco de dados.
- ☐ **Service** - Classe responsável por toda regra de negocio e tratativa de dados, sempre seguindo o modelo de negocio da aplicação.
- ☐ **Controller** - “End point”, é a camada responsável por manipular todas as requisições feitas do lado de fora da nossa API, essas requisições são feitas através

de URL's seguindo o protocolo HTTP

## Arquitetura básica de uma WEB Api



## Primeiros passos com Spring BOOT

---

### Como funciona um projeto SPRING BOOT?

---

O que é o Maven e como usá-lo.

#### ☐ Gerenciador de dependências

É ele quem faz os downloads das bibliotecas que você vai precisar no seu projeto (claro que, obviamente, você informando em um arquivo de configuração).

#### ☐ Repositório central


Todas as bibliotecas estão em um servidor na nuvem, chamado Maven Central, facilitando e centralizando o download (você não tem que ficar caçando bibliotecas no google)

#### ☐ Automatizador de tarefas

Um projeto que tem muitas bibliotecas e muitas dependências tem alguns “problemas” no seu dia-a-dia (manter as bibliotecas atualizadas, fazer todo o build da sua aplicação, fazer alguns testes, etc). Então o

**MAVEN** possui scripts prontos para automatizar tudo isso.

The Maven logo, featuring the word "maven" in a bold, lowercase, sans-serif font. The letter 'a' is orange, while the other letters are black.

 enter image description here

Obs: mostrar um projeto pronto.

## Como funciona um projeto SPRING BOOT?

Basicamente:

- 1 Classe Principal (com método main mesmo), que vai iniciar um servidor WEB (o TOMCAT) e ele vai gerenciar todas as URLs disponíveis
- Cada URL deve ser mapeada para um determinado método de uma classe. É a execução desse método que dá o retorno da resposta quando acionamos a URL.
- A partir daí, criamos nossos objetos que implementarão nossas lógicas.

## Como planejar um projeto SPRING BOOT?

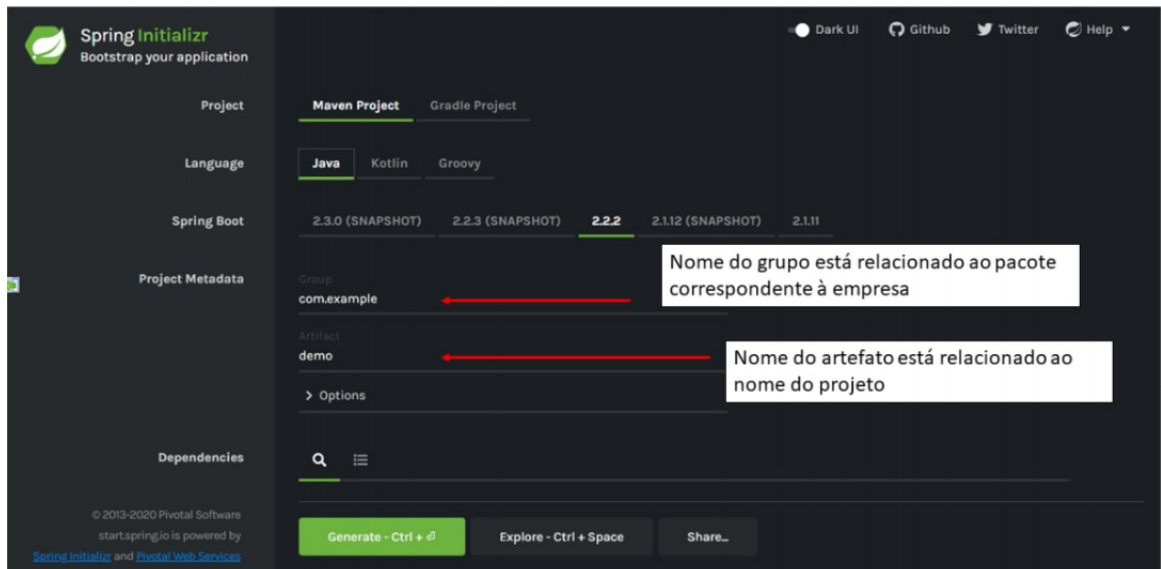
- Quais ENDPOINTS vamos oferecer? (Um Endpoint é uma URL associada a um método do protocolo HTTP - GET, POST, PUT, DELETE).
  - Em geral, temos 1 endpoint para cada objeto do nosso modelo de negócios  
Objeto de Negócios: PRODUTO
    - URL para recuperar dados de um produto (GET)
    - URL para inserir novo produto (POST)
    - URL para atualizar dados de um produto (PUT)
    - URL para remover um produto do sistema (DELETE)

***Como estes ENDPOINTS estarão estruturados no nosso sistema***

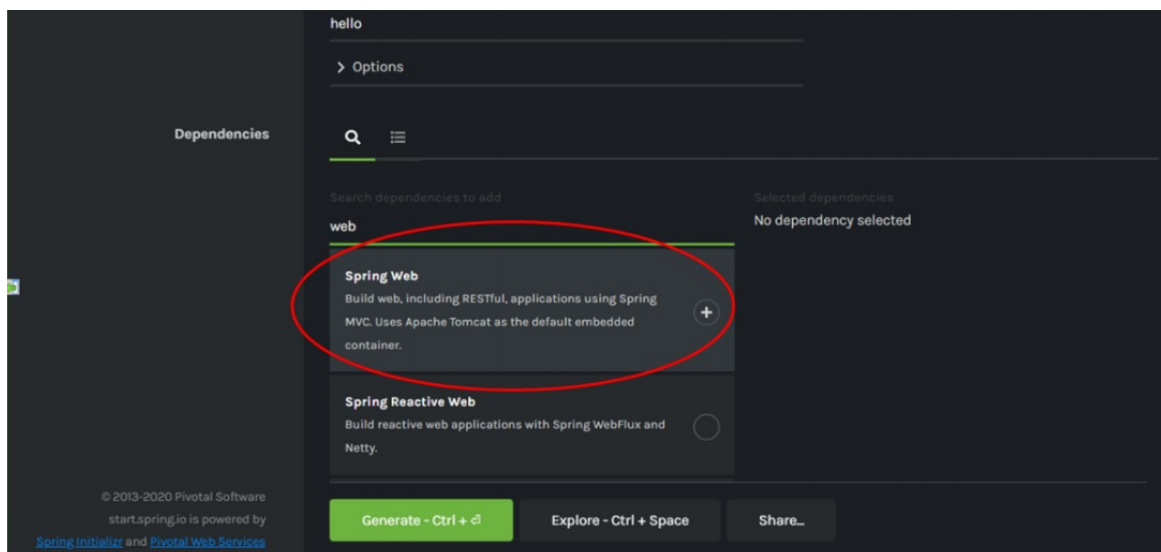
Vamos discutir isto Mais adiante.

## Mão na massa

- Acessar o gerador de templates do Spring Boot
  - <http://start.spring.io>
- Definir o grupo e o artefato do nosso projeto

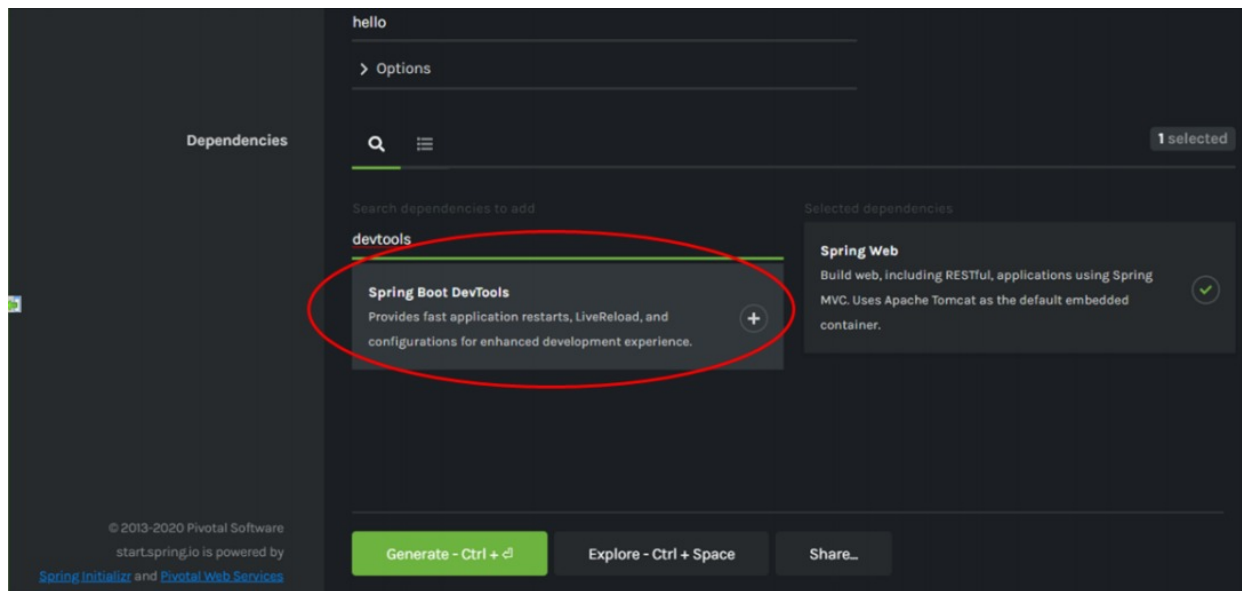


- Adicionar o plugin base para o SpringBoot (SpringWEB)

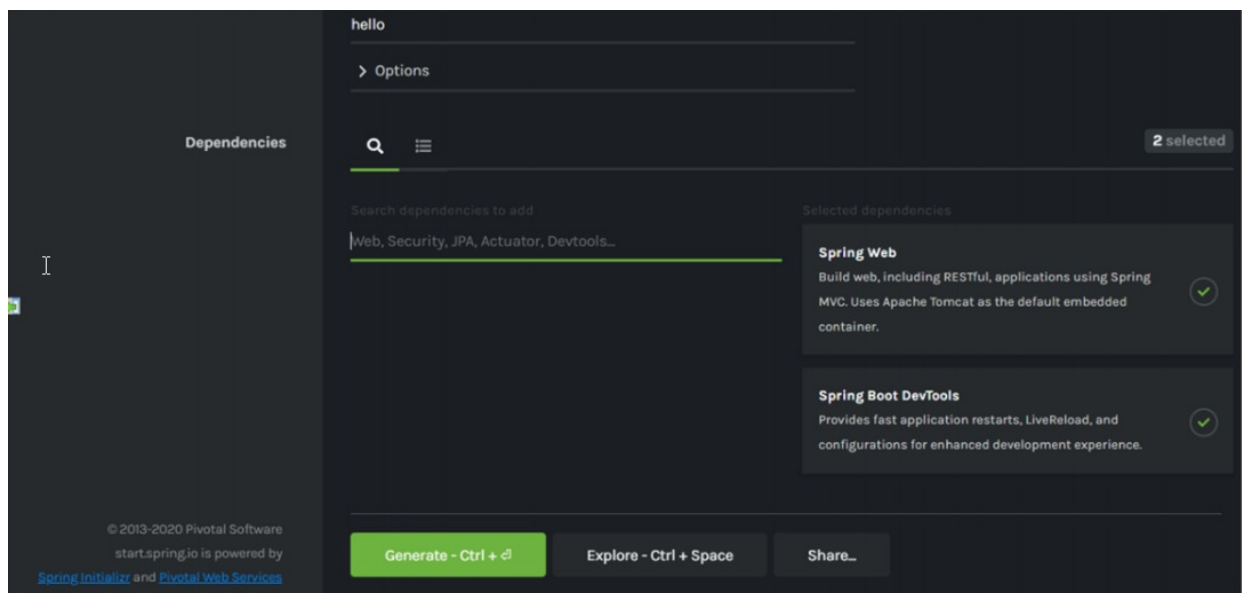


- Adicionar a dependência do DevTools (que poderá nos facilitar com o LiveReload acelerando o desenvolvimento).

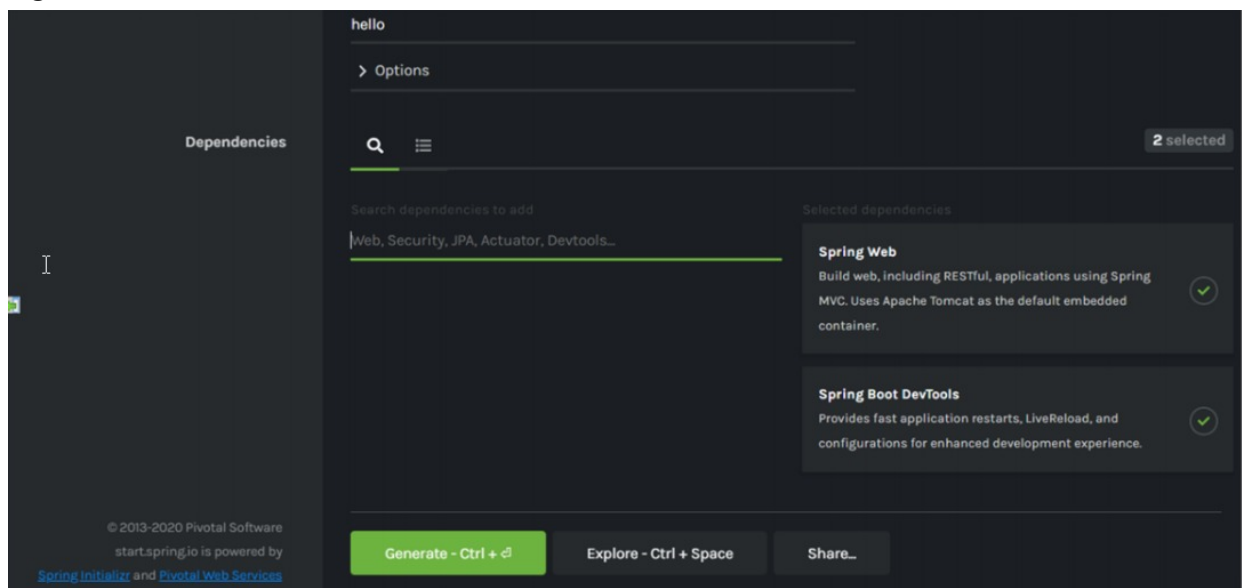




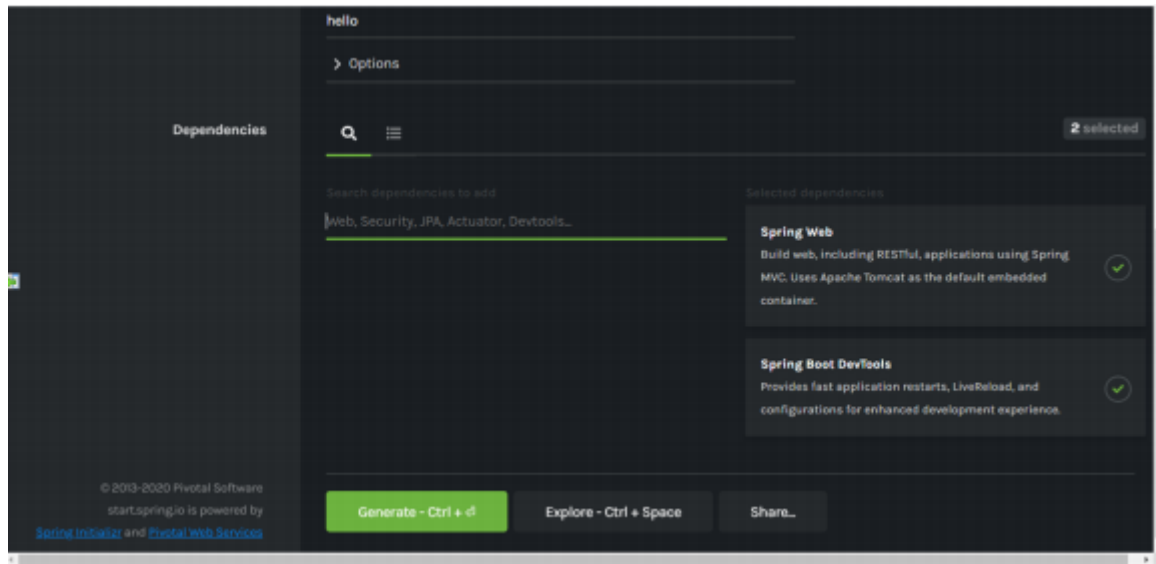
- Ao final, a especificação do projeto deverá estar deste jeito aqui:



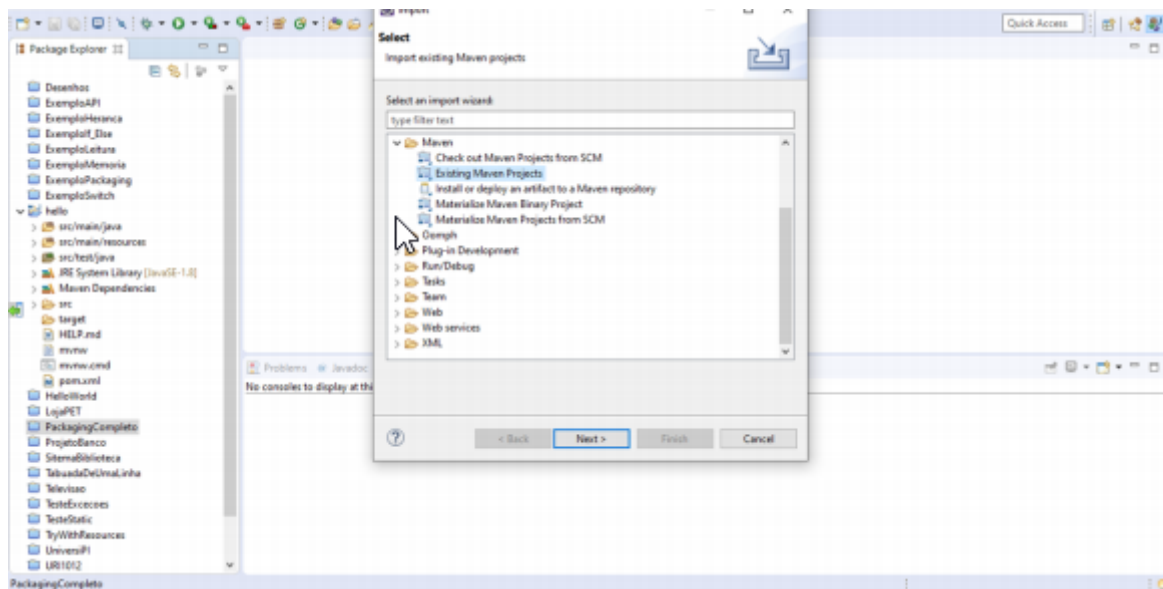
Agora basta clicar em **\*\*Generate - Ctrl + G\*\***



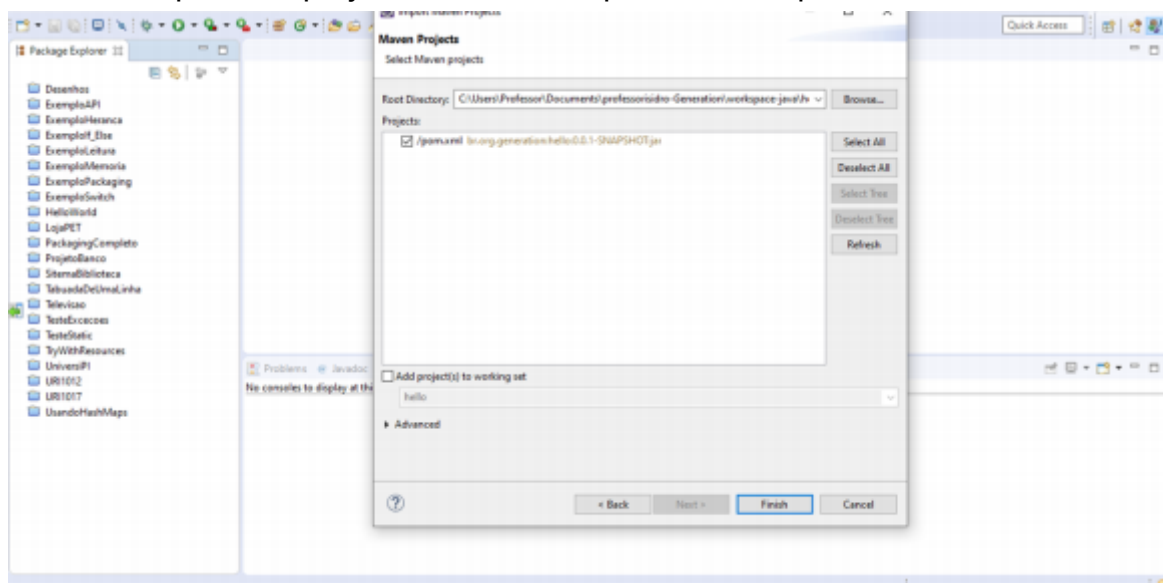
Ao final, a especificação do projeto deverá estar deste jeito aqui



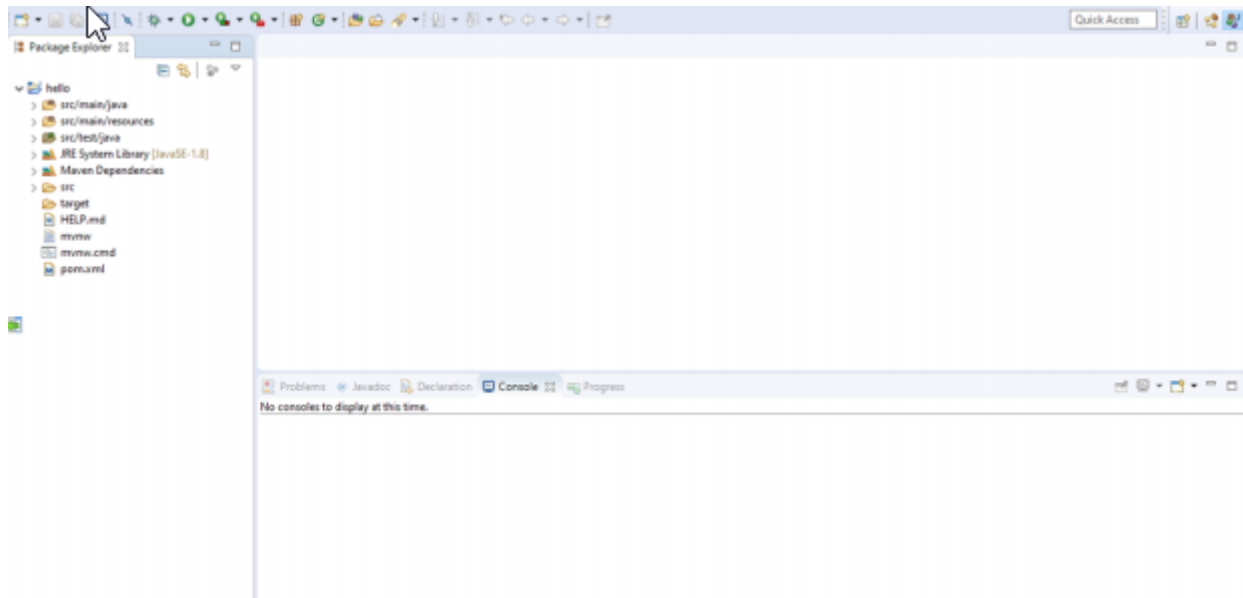
Importe o projeto no Eclipse (Importar como um Existing Maven Project)



Selecione a pasta do projeto e selecione o pom.xml e clique em FINISH



A estrutura do nosso projeto importado para o eclipse fica de acordo com a imagem:



## Criando nosso primeiro controller.

Lembrando: Todos os pacotes do nosso projeto deverão estar contidos dentro do pacote-base (formado pelos identificadores do grupo e do artefato)

```
ex: pacote controller  
br.org.generation.hello.controller
```

## Vamos criar nosso primeiro controller

- Criar o pacote que irá conter os controllers
- Criar uma nova classe que chamaremos aqui de ControllerTeste com o seguinte conteúdo.

```
packag
```

```
e br.org.generation.hello.controller;  
port org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;
```

```
package br.org.generation.hello.controller;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
@RestController  
public class ControllerTeste {  
    @GetMapping("/hello")
```

```
public String sayHello() {  
    return "Hello World! Nosso primeiro projeto Spring Boot";  
}  
}
```

A anotação `@RestController` define que a classe irá ser acionada a partir de URL (ela começa a atender endpoints) O método `sayHello` retorna uma mensagem de boas vindas e é anotado com `@GetMapping("/hello")` . Isso significa que o acesso será via URL pelo método GET através do caminho <http://meuservidor/hello>

## E para executar?

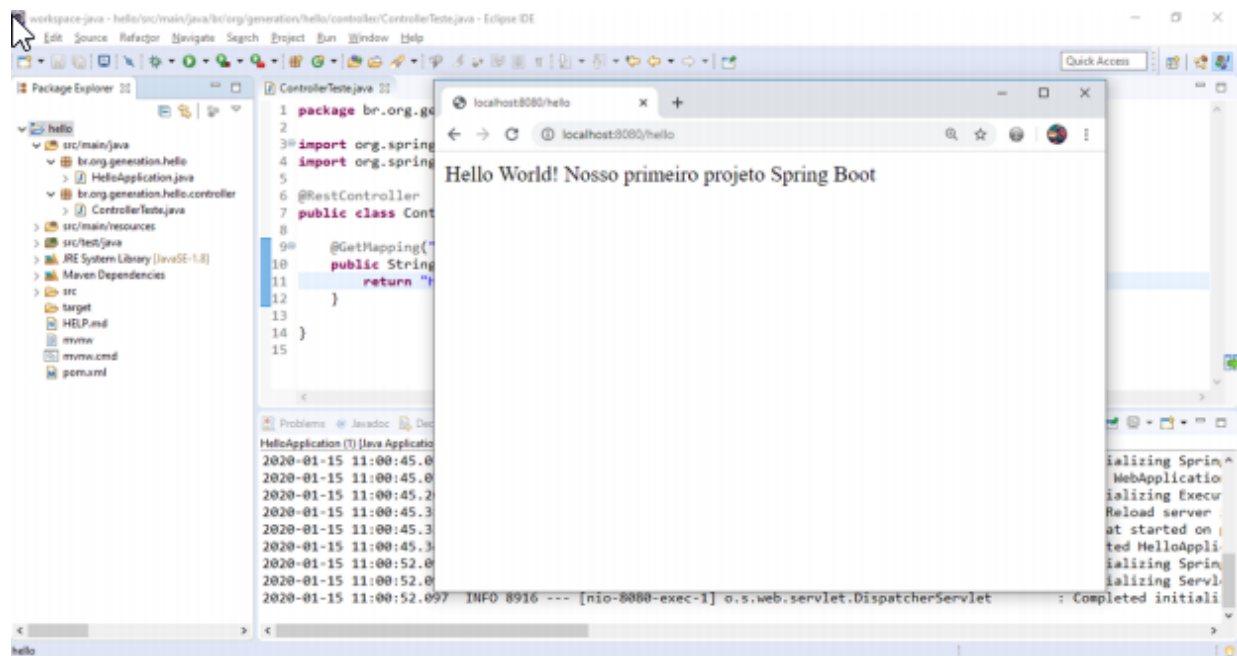
Basta selecionar o projeto, ir com o botão direito do Mouse na seguinte opção

- **Run As > Java Application** como mostra a figura abaixo

Neste ponto será mostrada uma tela uma única vez (sempre na primeira execução) que é para você selecionar a classe que corresponde à sua classe principal. O Eclipse varre todo seu projeto procurando por TODAS as classes que contenham métodos **main**, por isso é importante que você selecione a classe correta. Neste nosso caso, a classe se chama **HelloAplicação**

## Vamos testar!

Para testar, basta indicar o servidor (no nosso caso, servidor local, portanto **localhost**) a porta que o SpringBoot trabalha (por padrão porta **8080**, mas é possível mudar) e o caminho que queremos (neste caso **/hello**).



**Até a próxima...**