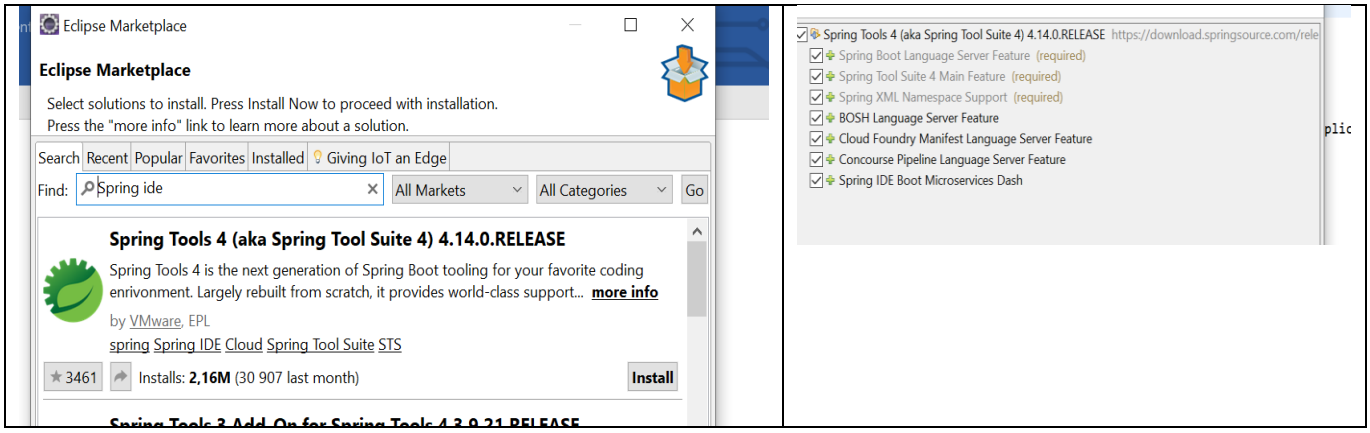
→ Framework Spring

https://spring.io/

1 – Consultando o Quickstart Guide (https://spring.io/quickstart), implementar o seu primeiro exemplo de uma aplicação web em Spring:

a) Se usar o IDE eclipse, deve começar por instalar o package "Spring Tools". Antes de instalar verifique se a sua versão do java é compatível. Os exemplos descritos nesta ficha foram testados com Java 17, mas pode trabalhar com as versões mais recentes.

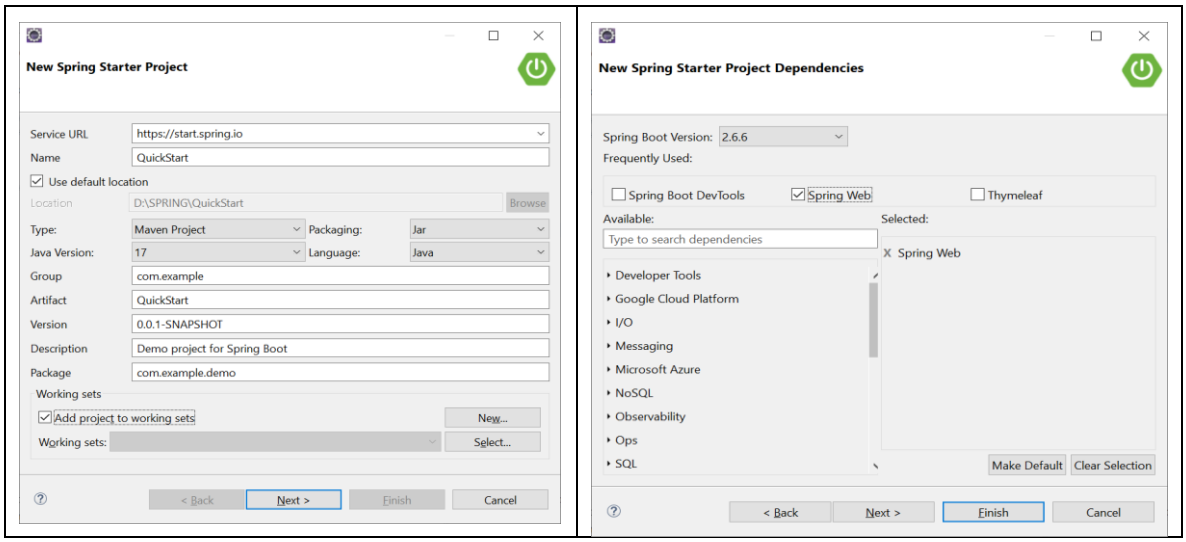
Para instalar o Spring Tools, em Help, selecionar Eclipse *marketplace*, e pesquisar por Spring IDE: Selecionar "Spring Tools 4" e instalar (ver figura). Provavelmente, encontrará uma versão mais recente que a da figura.



b) Para agora implementar o exemplo descrito em https://spring.io/quickstart deve em File, selecionar "new / Spring starter project".

- Dê um **nome** ao seu projeto, selecione a sua versão do Java e após Next selecione a dependência Spring web (ver figura seguinte).

Nota: selecione a versão mais recente do Spring Boot.



- Após Finish, a estrutura base do seu projeto será criada. Repare que na pasta src/main/java/com/example/demo foi criada a classe:

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class QuickStartApplication {
	public static void main(String[] args) {
		SpringApplication.run(QuickStartApplication.class, args);
	}
}
```

- Complete a classe seguindo o passo 2 do tutorial (https://spring.io/quickstart).

- Para executar o exemplo no eclipse

  - Sobre o nome do projeto, botão direito do rato, selecionar:

  Run as Spring Boot App;

(A sua aplicação ficará em execução no servidor web Apache Tomcat)

  - Testar no browser (http://localhost:8080/hello);

  - Teste agora dando um valor para o parâmetro *name*;

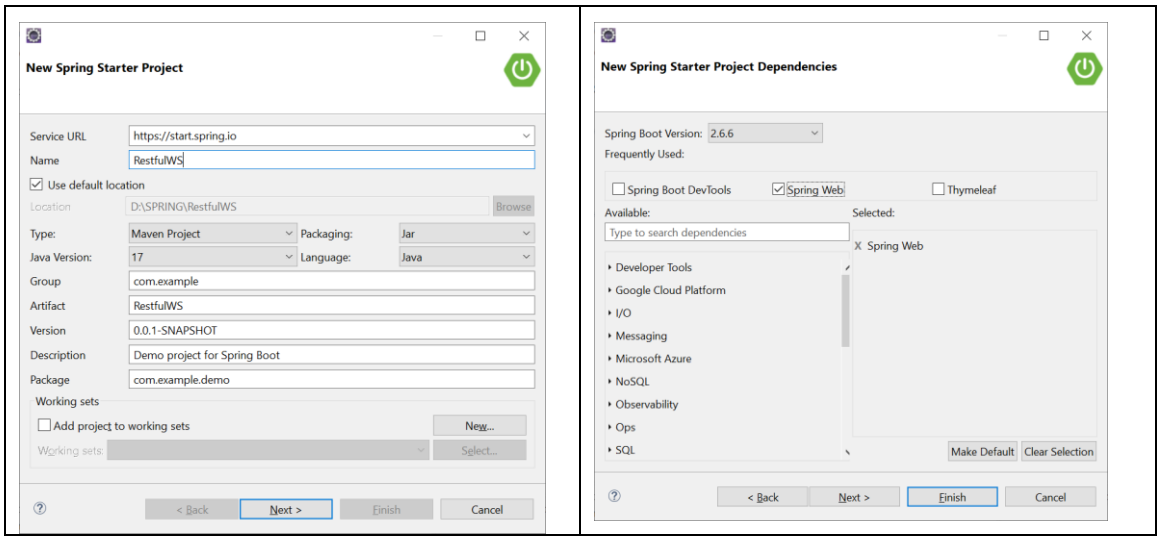
  - Para terminar a aplicação, matar o processo na consola.

  - Modifique o exemplo para que receba dois parâmetros, por exemplo, o primeiro e último nome.

2 - RESTful Web Services

Implementar, e estudar o exemplo do link: https://spring.io/guides/gs/rest-service/

a) Em File, selecionar "new / Spring starter project". Dar o nome ao projeto e adicionar a dependência Spring Web.



- Siga os passos *"Create a Resource Representation Class"* e *"Create a Resource Controller"* do tutorial.

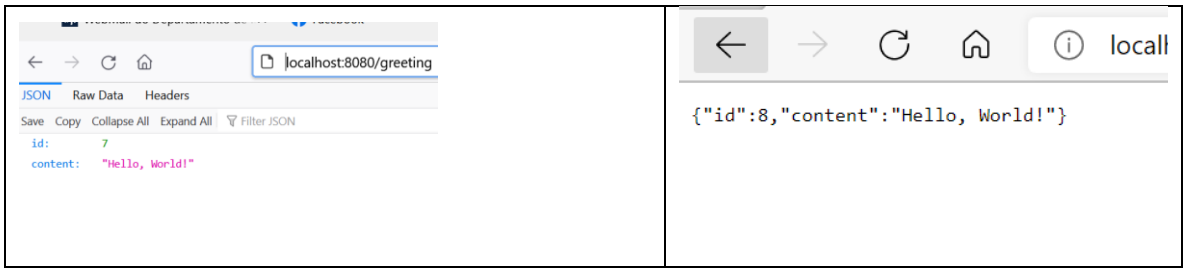
- Para executar o exemplo no eclipse
  - Sobre o nome do projecto, botão direito do rato, selecionar:

    Run as Spring Boot Application;

(A sua aplicação ficará em execução no servidor web Apache Tomcat)

  - Testar no browser (http://localhost:8080/greeting);

Se o seu browser tem instalado um JSON viewer, o output será algo como a imagem da esquerda na figura abaixo. Caso contrário obterá a imagem da direita.



- Explore o exemplo. O que acontece se recarrega a página?

- O que acontece se chama o mesmo link numa janela diferente do mesmo browser?

- O que acontece se chama o mesmo link num outro browser?

- Explore o exemplo, criando outros *web services*. Por exemplo, dado um inteiro, n, mostrar um valor aleatório entre 0 e N.

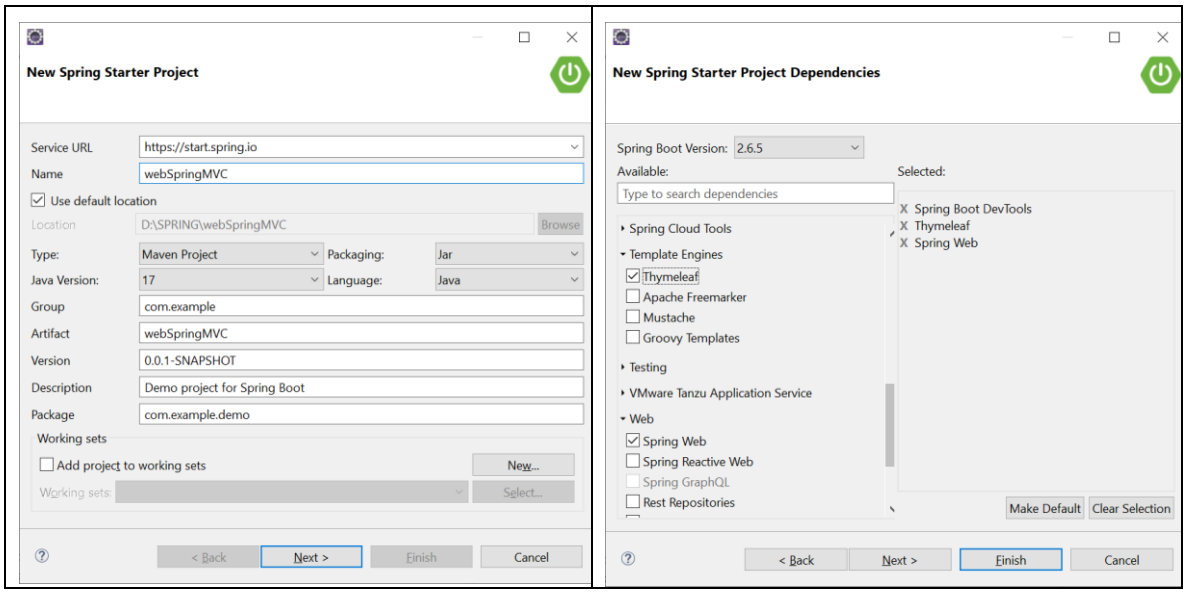
3 - Aplicação Web (Spring e html)

Vamos agora testar o exemplo descrito em:

https://spring.io/guides/gs/serving-web-content/

Neste exemplo vamos aceder a um "web service" a partir de uma página em html.

a) Em File, selecione "new / Spring starter project".
   Dê um nome ao seu projeto, selecione a sua versão do Java e após Next, selecione as dependências ilustradas na figura seguinte (**Spring web**, **Spring Boot DevTools** e **Thymeleaf**)



Após Finish, pode verificar que tal como nos exemplos anteriores foi criada uma classe com o main da aplicação.

- Siga os passos do tutorial para criar o *Web Controler* e a página em html (greeting.html).

- Execute a aplicação com Run as Spring Boot Application e teste no browser com http://localhost:8080/greeting.

- Teste dando um valor para o parâmetro.

- Experimente modificar a página greeting.html (por exemplo adicionado um texto) e verifique que o output é alterado sem ter de lançar novamente a aplicação.

b) Siga o tutorial para criar uma página index.html e teste a sua aplicação com

http://localhost:8080

- Experimente modificar a página index.html e verifique que o output é alterado sem ter de lançar novamente a aplicação.

c) Explore como adicionar um ficheiro css às suas páginas html.

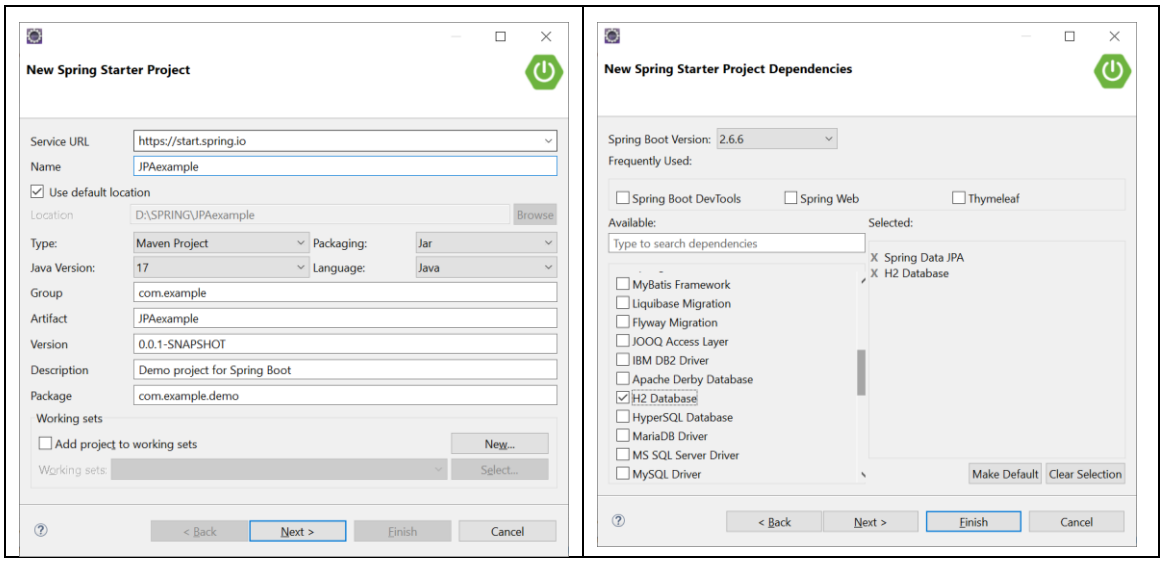
4 - Spring JPA (Persistência de dados),

Vamos implementar um exemplo que cria e acede a uma base de dados H2, usando o tutorial **https://spring.io/guides/gs/accessing-data-jpa/**.

*Nota: H2 é um sistema de gestão de base de dados relacional, escrito em java (**http://www.h2database.com/html/main.html**).*

a) Em File, selecionar "new / Spring starter project".

- Dê um nome ao seu projeto, selecione a sua versão do Java e após Next selecione as dependências **Spring Data JPA e H2 Database** (ver figura abaixo).



Após Finish, a classe de inicialização será criada.

- Seguindo o tutorial **https://spring.io/guides/gs/accessing-data-jpa/,** no passo *Define a Simple Entity,* criar a entity Customer que irá dar origem à tabela Customer da sua base de dados.

- De seguida, siga o passo *Create Simple Queries,* e crie a interface CustomerRepository. Esta interface é subinterface da interface genérica *CrudRepository<T, ID)* que define as operações base de manipulação de uma base de dados relacional. A implementação das operações para os parâmetros dados será feita de forma automática pelo Spring data JPA.

- Finalmente modificar a classe de inicialização.

Pode seguir o passo Create an Application Class do tutorial ou alterar de acordo com a listagem abaixo:

```
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class JpAexampleApplication {
	public static void main(String[] args) {
		SpringApplication.run(JpAexampleApplication.class, args);
	}
	@Bean
	public CommandLineRunner demo(CustomerRepository repository) {
	    return (args) -> {
	      // save a few customers
	      repository.save(new Customer("Jack", "Bauer"));
	      repository.save(new Customer("Chloe", "O'Brian"));
	      repository.save(new Customer("Kim", "Bauer"));
	      repository.save(new Customer("David", "Palmer"));
	      repository.save(new Customer("Michelle", "Dessler"));
	      repository.save(new Customer("Kim", "Dessler"));
	      // fetch all customers
	      System.out.println ("Customers found with findAll():");
	      System.out.println ("-------------------------------");
	      for (Customer customer : repository.findAll()) {
	         System.out.println (customer.toString());
	      }
	      System.out.println ("");
	      Customer customer = repository.findById(1L);
	      System.out.println (customer.toString());
	      System.out.println ("Customer found with findById(1L):");
	      System.out.println ("--------------------------------");
	      System.out.println( customer.toString());
	      System.out.println ("");
	      System.out.println ("Customer found with
	            findByLastName('Bauer'):");
	      System.out.println ("----------------------------------------");
	      for (Customer bauer : repository.findByLastName("Bauer")) {
	          System.out.println(bauer.toString() );
	       }
	       System.out.println("**************End");
	    };
	  }
}
```

- Execute a aplicação com Run as Spring Boot Application e observe o output na consola.

b) Experimente criar um novo query, agora para consultar os clientes pelo primeiro nome.

c) Apague um registo da tabela Customer.

d) Modifique um registo da tabela Customer.