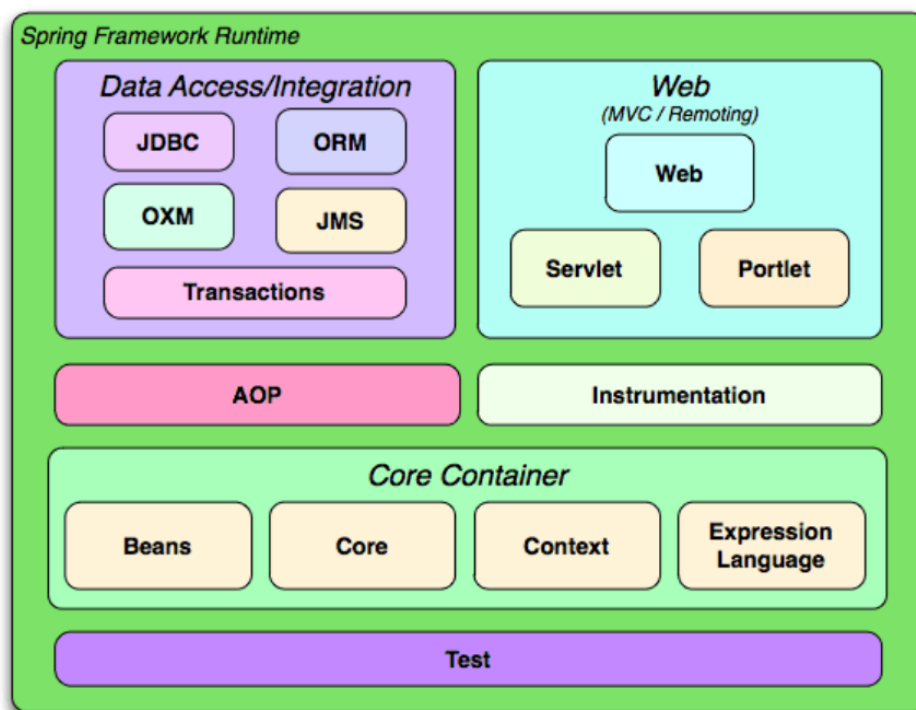




## 6. Spring Boot

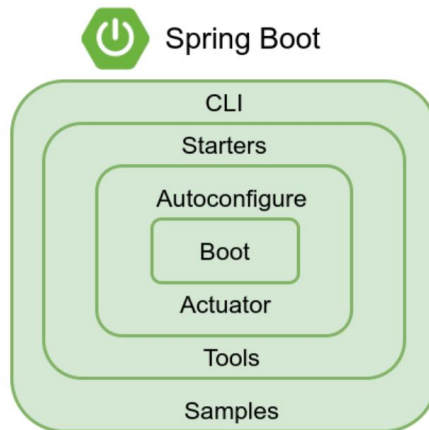
Spring Framework es un *framework* Open Source que facilita la creación de aplicaciones de todo tipo en **Java**, **Kotlin** y **Groovy**. Está formado por los siguientes módulos opcionales, utilizando los que necesitemos sin tener que llenar nuestro classpath con clases que no vamos a usar:

- ✓ **Core container:** proporciona inyección de dependencias e inversión de control.
- ✓ **Web:** nos permite crear controladores Web, tanto de vistas MVC como aplicaciones REST.
- ✓ **Acceso a datos:** abstracciones sobre JDBC, ORMs como Hibernate, sistemas OXM (*Object XML Mappers*), JSM y transacciones.
- ✓ **Programación orientada a Aspectos (AOP):** ofrece el soporte para aspectos.
- ✓ **Instrumentación:** proporciona soporte para la instrumentación de clases.
- ✓ **Pruebas de código:** contiene un *framework* de *testing*, con soporte para JUnit y TestNG y todo lo necesario para probar los mecanismos de Spring.



Spring Boot, publicado en 2012, es una solución para el framework Spring de Java que sigue el **principio de “convención sobre configuración”**, reduciendo al mínimo el número de pasos que un desarrollador debe dar en la **configuración inicial** del proyecto, por lo tanto, **reduce la complejidad del desarrollo** de nuevos proyectos basados en Spring. Para ello, Spring Boot **proporciona la estructura básica configurada** del proyecto, que incluye las pautas para usar el marco y todas las bibliotecas de terceros relevantes para la aplicación. De esta manera se simplifica

mucho la creación de aplicaciones independientes y reproducibles, por lo que **la mayoría de las nuevas aplicaciones basadas en Spring se desarrollan con Spring Boot**.



Las razones **de usar Spring Boot** son las siguientes:

- ✓ **Contenedor de aplicaciones integrado:** Spring Boot permite **compilar nuestras aplicaciones Web como un archivo .jar** que podemos ejecutar como una aplicación Java normal (como alternativa a un archivo .war, que desplegaríamos en un servidor de aplicaciones como Tomcat). **Esto lo consigue integrando el servidor de aplicaciones en el propio .jar y levantándolo cuando arrancamos la aplicación.** De esta forma, podemos distribuir nuestras aplicaciones de una forma mucho más sencilla, al poder configurar el servidor junto con la aplicación.

Esto también es **muy útil en arquitecturas de microservicios**, puesto que permite **distribuir nuestras aplicaciones como imágenes Docker** (se puede decir que son instancias de un contenedor) que podemos escalar horizontalmente. (algo muy complicado con un .war).

Spring boot permite distribuir tu aplicación como un jar, no lo impone. Si prefieres desplegar tu aplicación en un servidor de aplicaciones tradicional, **Spring Boot te deja compilar el código como un .war** que no incluya ningún servidor de aplicaciones integrado.

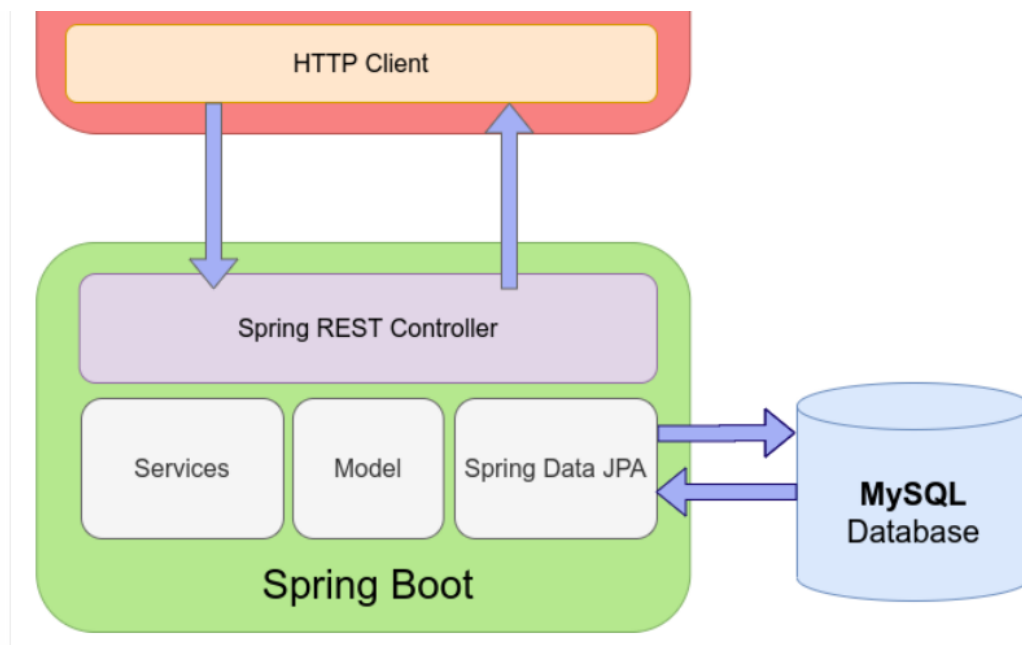
- ✓ **Starters:** Spring Boot **nos proporciona una serie de dependencias**, llamadas starters, que podemos añadir a nuestro proyecto dependiendo de lo que

necesitemos: crear un controlador REST, acceder a una base de datos usando JDBC, conectar con una cola de mensajes Apache ActiveMQ, etc.

Una vez añadimos un starter, éste **nos proporciona todas las dependencias que necesitamos, tanto de Spring como de terceros**. Además, los starters vienen configurados con valores por defecto, que pretenden minimizar la necesidad de configuración a la hora de desarrollar.

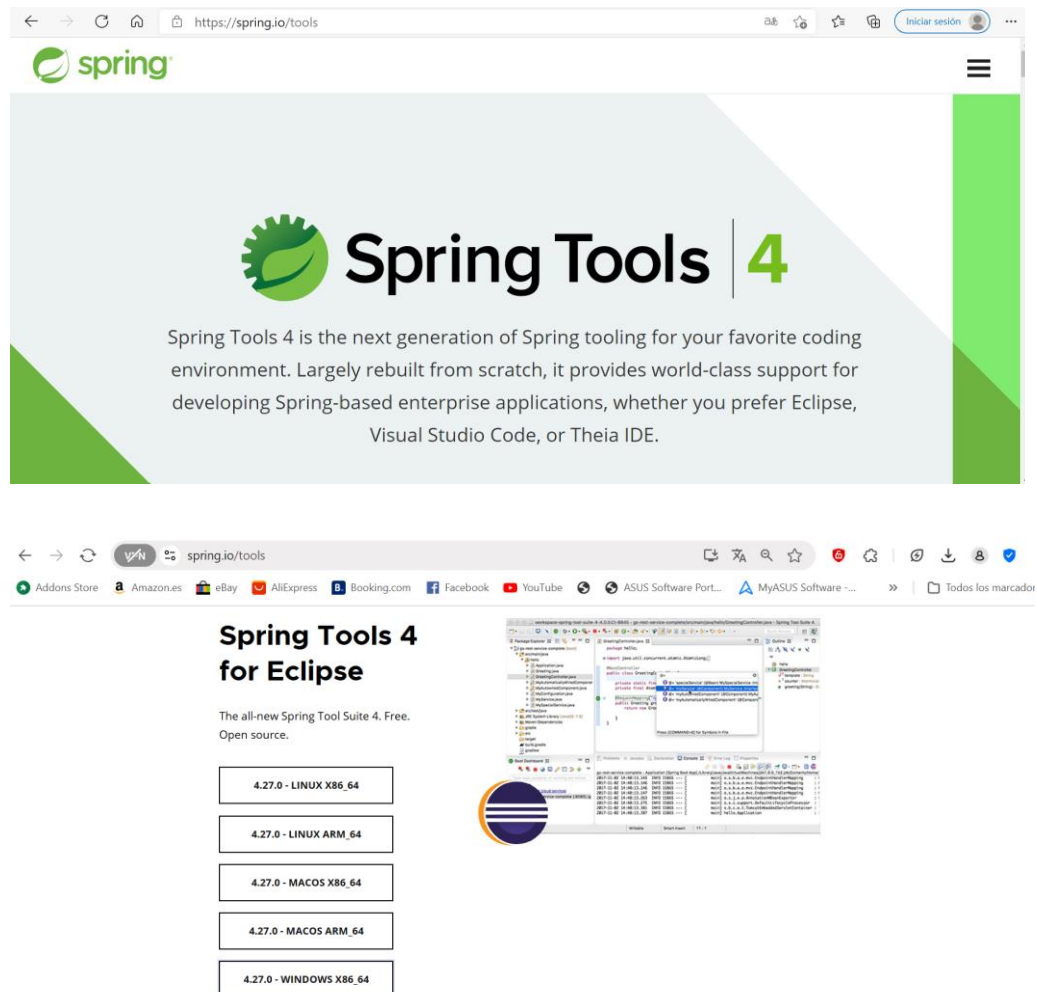
Al igual que con Spring Framework, cualquier configuración puede ser modificada de ser necesario: desde el puerto en el que la aplicación escucha peticiones, hasta el banner que sale por consola al arrancar la aplicación.

- ✓ **Spring MVC genera HTML plano** de tal forma que delega más la responsabilidad en HTML y Javascript. Spring aporta la posibilidad de utilizarlo tanto para realizar con “**vistas**” como “**Controladores**”. La diferencia con las JSP y JSF es el nivel de abstracción, tienen un nivel de abstracción alto y casi todas las responsabilidades están en el servidor que genera controles de forma automática.

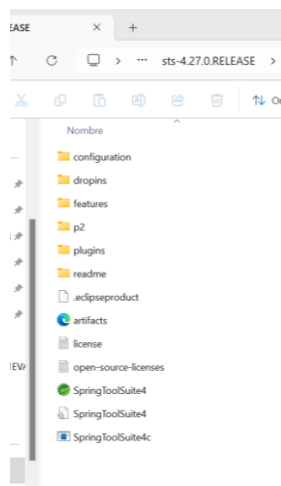


## 6.1 Instalación

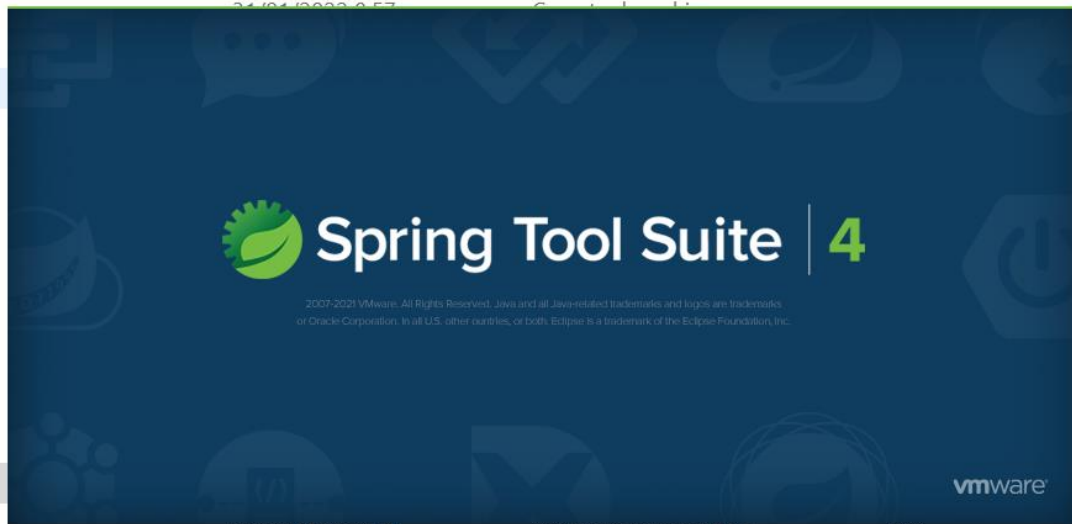
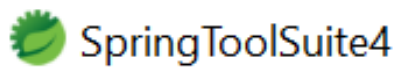
- ✓ Lo primero es descargarse el IDE Spring Tools Suite o STS desde la página: <https://spring.io/tools>



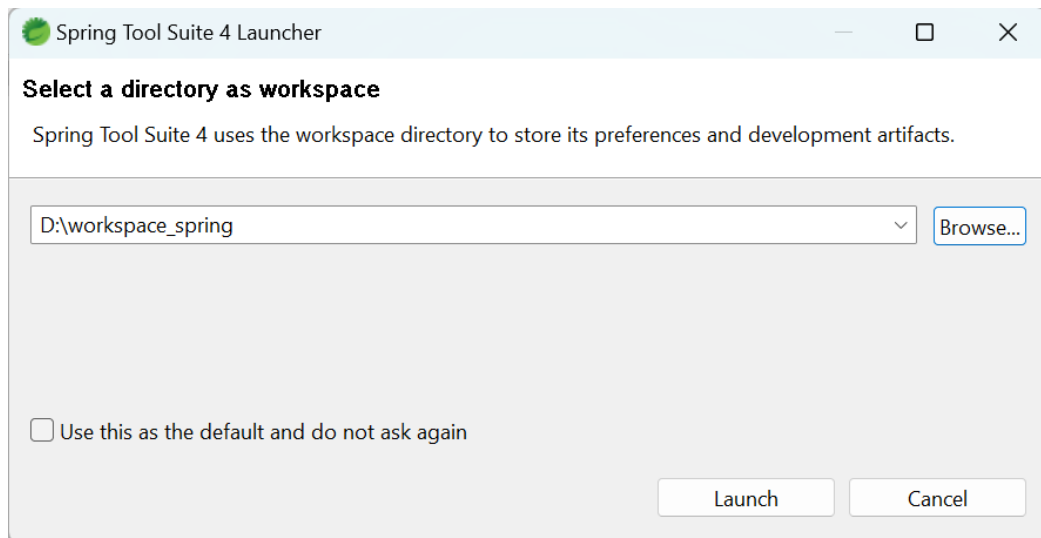
Se descomprime la carpeta .zip del portable:



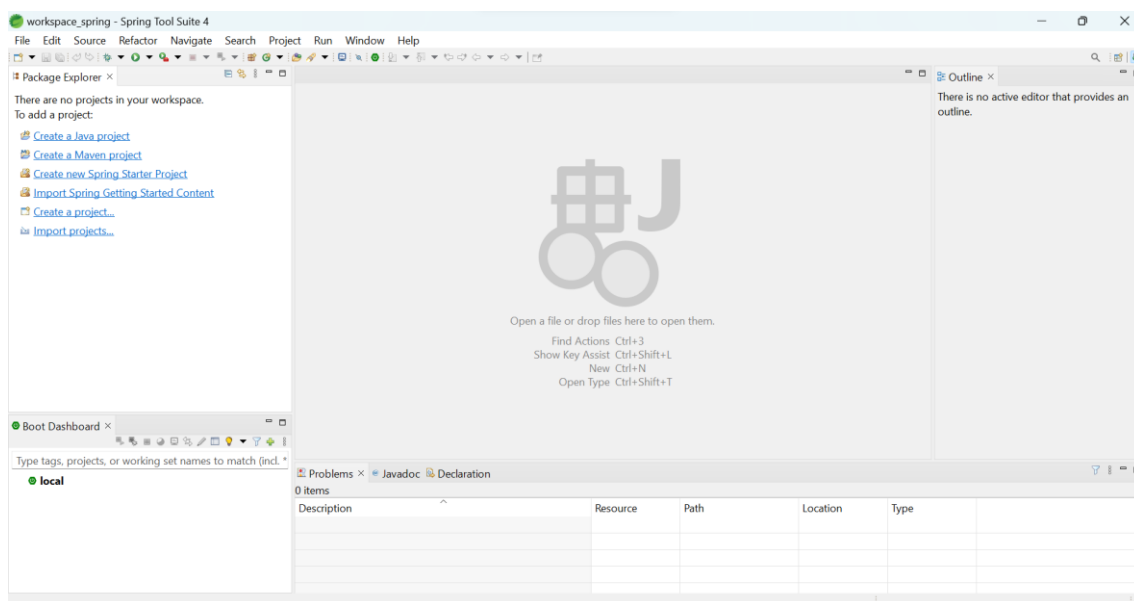
- ✓ Arrancamos el IDE desde el ejecutable:



- ✓ El editor STS está **basado en Eclipse** y preconfigurado para crear rápidamente proyectos de **Maven** y de **Spring**. También permite elegir el espacio de trabajo:



- ✓ Es importante tener instalado Java JRE 8+, Java JDK 8+ y MySQL 5+.



## 6.2 Crear un proyecto en Spring Boot

En Spring Boot todo se hace con Java, por lo que Java Runtime Environment (plataforma Java) es el principal componente de software para que el marco funcione. Este software, además del entorno de ejecución, contiene herramientas útiles para programar y testar aplicaciones Java y está disponible tanto para Linux como para Windows y macOS, por lo que nos permite elegir libremente el sistema operativo que queramos utilizar. Requiere que tengamos la versión actual del framework Spring instalada en el sistema.

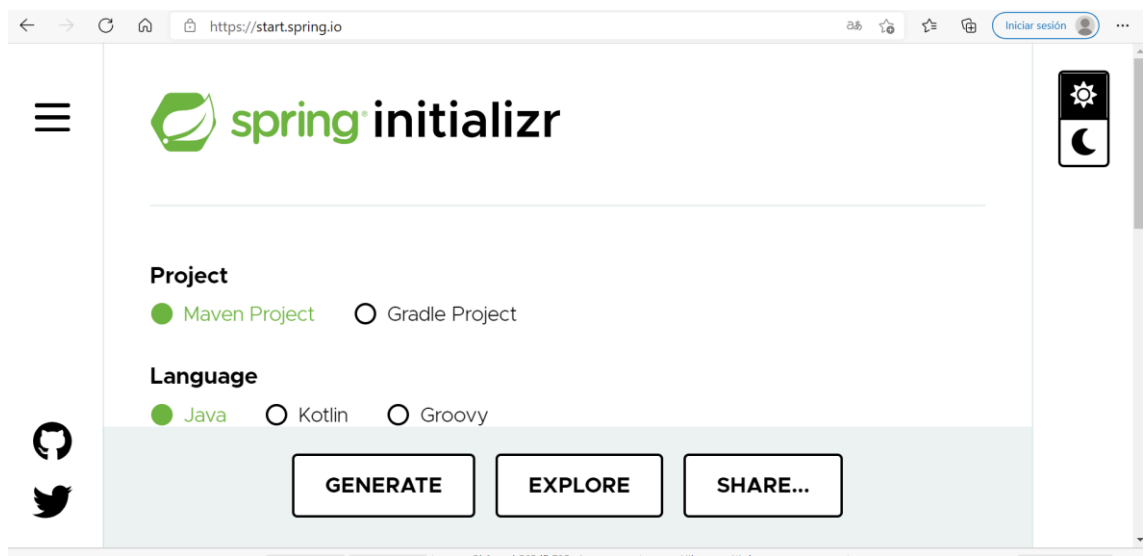
Como herramienta de compilación, podemos utilizar **Maven** (a partir de 3.3) o **Gradle** (a partir de 4.4). En nuestro caso utilizaremos Maven.

Se puede incorporar un **objeto servlet de inicialización** que ejecute la aplicación Java sobre la base de un servidor web. Para hacerlo, puedes elegir entre tres soluciones: Apache Tomcat (a partir de 9.0), Jetty (a partir de 9.4) o Undertow (a partir de 2.0).

Spring Boot se basa en la creación de **clases** que estén marcadas **con anotaciones especiales** que guíen el desarrollo de nuestro proyecto. Para empezar, debemos crear un proyecto nuevo.

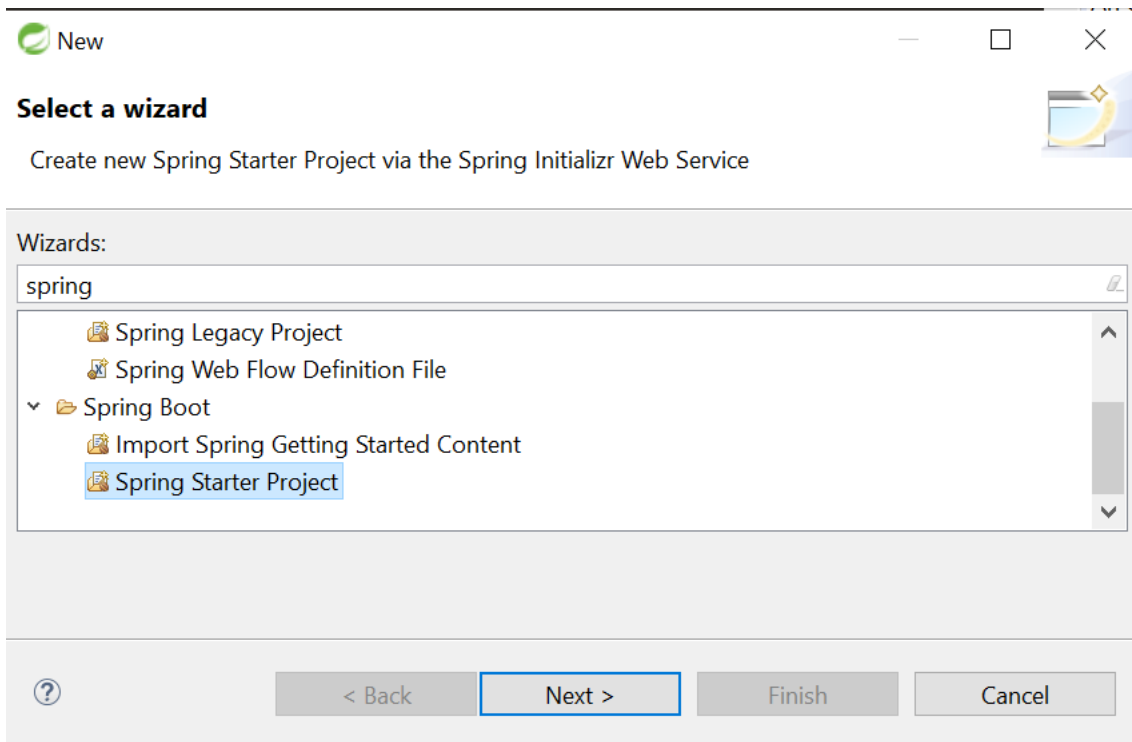
Al igual que en cualquier biblioteca Java estándar, en Spring Boot se **incluyen los correspondientes archivos JAR** (Java Archive) o **WAR** (Web Application Archive) en el **Classpath**. Java recurre a esta ruta del sistema de archivos para buscar los archivos ejecutables. Puedes crear los archivos de almacenamiento de Spring Boot de dos maneras distintas:

- Utiliza el **servicio web Spring Initializr** para establecer la configuración de Spring Boot y después, **descárgala como plantilla de proyecto final**. Una característica es que proporciona una **interfaz web muy fácil de usar** para crear los archivos JAR, lo que simplifica considerablemente el proceso. Como Initializr también emplea Maven o Gradle para generar los archivos, el resultado es el mismo que si lo haces de forma manual.



- ✓ Utiliza **Maven o Gradle** para crear por tu cuenta todo el marco del proyecto, incluidas las dependencias necesarias.
- ✓ Vamos a pulsar el ícono de nuevo proyecto en la opción de *Spring Boot* y en *Spring Starter Project* o en *File>New>Spring Starter Project*. Como se muestra en la siguiente imagen.





- ✓ A continuación, deberemos **establecer el nombre del proyecto**, el cuál debe estar relacionado con lo que estemos haciendo, este también será utilizado como nombre del **Artefacto**, el cual se refiere al nombre clave del proyecto para nuestra empresa.
- ✓ También estableceremos el **nombre del grupo de desarrollo software** que se refiere, **se usa el nombre de la empresa** de la forma **com.miempresa** u **org.miempresa**, es buena práctica repetir este mismo nombre como nombre del paquete principal.

**New Spring Starter Project**

Service URL:

Name:

☒ Use default location

Location:

Type:  Packaging:

Java Version:  Language:

Group:

Artifact:

Version:

Description:


Package:


Working sets

☐ Add project to working sets

Working sets:

- ✓ Si seguimos con la configuración en la siguiente ventana, **debemos marcar las dependencias de nuestro proyecto**. Las dependencias **cargan las librerías JAR** necesarias para utilizar diferentes tecnologías.
- ✓ De momento necesitaremos **Web para crear un proyecto Web en Spring**. **Seleccionamos dentro de Web Spring Web**:

— □ ×

**New Spring Starter Project Dependencies**

Spring Boot Version:


Available:

- ▶ AI
- ▶ Developer Tools
- ▶ Google Cloud
- ▶ I/O
- ▶ Messaging
- ▶ Microsoft Azure
- ▶ NoSQL
- ▶ Observability
- ▶ Ops
- ▶ SQL
- ▶ Security
- ▶ Spring Cloud

Selected:

Make Default

Clear Selection




< Back

Next >

Finish


Cancel



—

□

×

New Spring Starter Project Dependencies

Spring Boot Version: 3.4.1

Available:

web

▼ Messaging

☐ WebSocket

▼ Template Engines

☐ Thymeleaf

☐ Apache Freemarker

☐ Mustache

▼ Testing

☐ Testcontainers

▼ Web

☒ Spring Web

☐ Spring Reactive Web

☐ Spring Web Services

☐ Jersey

Selected:

X Spring Web

Make DefaultClear Selection

?

< Back

Next >

Finish

Cancel

**New Spring Starter Project**

Site Info

Base Url: `https://start.spring.io/starter.zip`

Full Url: `https://start.spring.io/starter.zip?name=demo&groupId=com.gf&artifactId=demo&version=0.0.1-SNAPSHOT&description=Demo+project+for+Spring+Boot&packageName=com.gf.demo&type=maven-project&packaging=jar&javaVersion=17&language=java&bootVersion=3.2.1`

< Back   Next >   **Finish**   Cancel

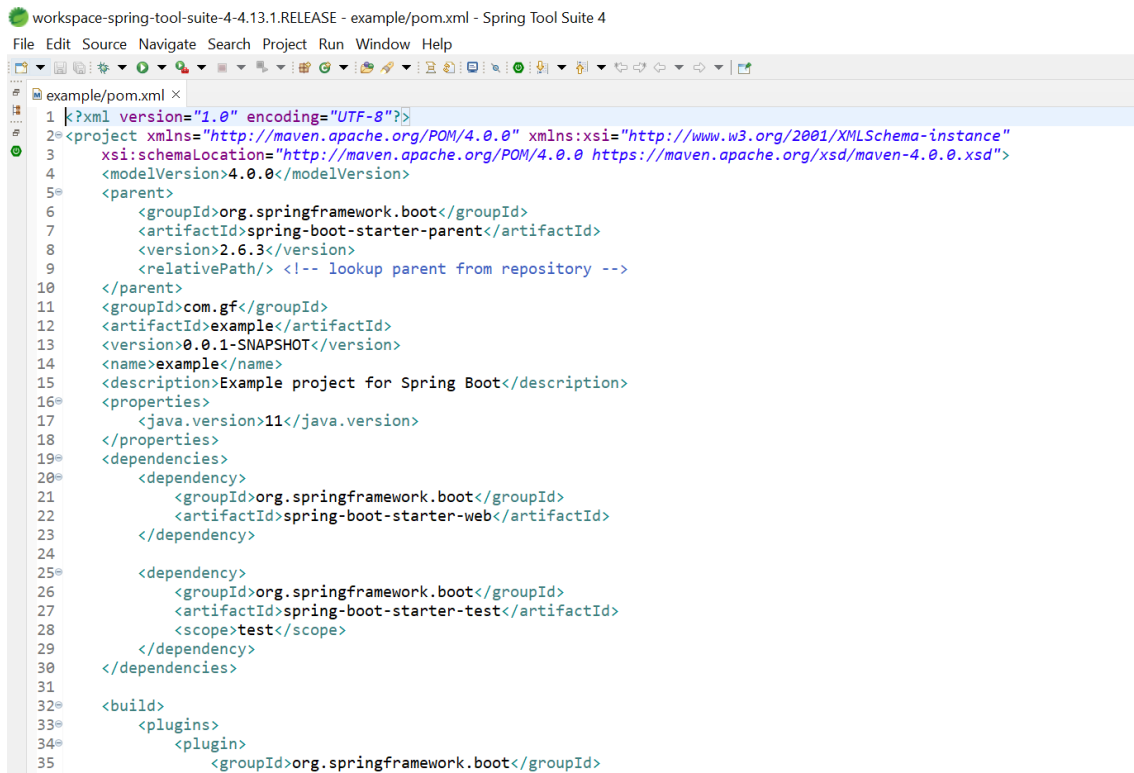
✓ La estructura del proyecto quedará de la siguiente forma:

```

v demo-1 [boot]
  v src/main/java
    v com.gf.demo
      > Demo1Application.java
      > ServletInitializer.java
    v src/main/resources
      static
      templates
      application.properties
  > src/test/java
  > JRE System Library [JavaSE-17]
  > Maven Dependencies
  target/generated-sources/annotations
  target/generated-test-sources/test-annotations
  v src
    v main
      webapp
      test
  > target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml

```

- ✓ Si por error olvidaste agregar una dependencia o quieres agregar más dependencias puedes hacerlo en el archivo **pom.xml** ubicado en la raíz del proyecto. Para añadir una dependencia basta con conocer el nombre del grupo y artefacto. En la pestaña **Dependencies** podemos acceder a un buscador de dependencias.



```
workspace-spring-tool-suite-4-4.13.1.RELEASE - example/pom.xml - Spring Tool Suite 4
File Edit Source Navigate Search Project Run Window Help
example/pom.xml x
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>2.6.3</version>
9     <relativePath/> <!-- lookup parent from repository -->
10  </parent>
11  <groupId>com.gf</groupId>
12  <artifactId>example</artifactId>
13  <version>0.0.1-SNAPSHOT</version>
14  <name>example</name>
15  <description>Example project for Spring Boot</description>
16  <properties>
17    <java.version>11</java.version>
18  </properties>
19  <dependencies>
20    <dependency>
21      <groupId>org.springframework.boot</groupId>
22      <artifactId>spring-boot-starter-web</artifactId>
23    </dependency>
24
25    <dependency>
26      <groupId>org.springframework.boot</groupId>
27      <artifactId>spring-boot-starter-test</artifactId>
28      <scope>test</scope>
29    </dependency>
30  </dependencies>
31
32  <build>
33    <plugins>
34      <plugin>
35        <groupId>org.springframework.boot</groupId>
```

- ✓ **Maven** es un **sistema de administración de dependencias** que se encarga de descargar las librerías necesarias para que el proyecto funcione, **se basa en la creación de un árbol de dependencias** el cuál marca para cada dependencia que otras dependencias (librerías) necesita y en qué versión mínimo. Si alguna dependencia falta, **Maven automáticamente la descargará de internet y actualizará el árbol.**
- ✓ Los microservicios pueden ser autocontenidos de tal forma que incluyen todo lo necesario para prestar su servicio evitando por ejemplo no depender de un servidor de aplicaciones en el que desplegar la aplicación que ha de ser instalado previamente, **para ello pueden incluir un servidor embebido de Tomcat**, de Jetty o usando Spring Boot. Esto evita malos funcionamientos por diferencias en la configuración o de versiones de los servidores en cada uno de los entornos, además hace más fácil el despliegue en una nueva máquina siendo lo único necesario el microservicio, sin necesidad de disponer previamente un servidor externo.

- ✓ Los jars dependientes para Tomcat están disponibles en tomcat jasper. Añadimos la dependencia tomcat jasper en el archivo pom.xml

```
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
  <version>11.0.1</version>
</dependency>
```

- ✓ Spring Boot DevTools es la herramienta de Spring Boot que nos permite reiniciar de forma automática nuestras aplicaciones cada vez que se produce un cambio en nuestro código.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <scope>runtime</scope>
</dependency>
```

- ✓ *Crea un carpeta para guardar las JSPs dentro de webapp->WEB-INF->view.*
- ✓ *Para poder editar JSP debes de descargar el siguiente plugging.*

Eclipse Marketplace

## Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with installation.  
Press the "more info" link to learn more about a solution.

Search
Recent
Popular
Favorites
Installed
Research at the Eclipse

Find: 
All Markets
All Categories
Go

### Eclipse Enterprise Java and Web Developer Tools 3.36

Enables Enterprise Java Bean, Java Enterprise Application, Fragments, and Connector, Java Web Application, JavaServer Faces (JSF), Java Server Pages (JSP), Java... [more info](#)

by [The Eclipse Foundation](#), EPL

[xml](#) [html](#) [CSS](#) [js](#) [jsp](#)

★ 1795
Installs: **1,30M** (10.061 last month)

Install

### IBM® watsonx™ Code Assistant for Enterprise Java Applications 1.0.0

IBM® watsonx™ Code Assistant for Enterprise Java Applications Overview IBM® watsonx™ Code Assistant is an innovative, generative AI coding companion that offers... [more info](#)

by [IBM](#), Commercial

[AI](#) [application modernization](#) [autocomplete](#) [chat](#) [class referencing](#)

★ 1
Installs: **98** (99 last month)

Install

## Marketplaces

?

< Back

Install Now >

Finish

Cancel

Package Explorer
demo-1/pom.xml
Demo1Application.java
ServletInitializer.java
HolaControlador.java
hola.jsp

- demo [boot]
- demo-1 [boot]
  - src/main/java
  - src/main/resources
  - src/test/java
  - JRE System Library [JavaSE-17]
  - Maven Dependencies
  - target/generated-sources/annotations
  - target/generated-test-sources/test-annotations
  - src
    - main
      - webapp
        - WEB-INF
          - view
            - hola.jsp
    - test
  - target
  - HELP.md
  - mvnw
  - mvnw.cmd
  - pom.xml

```

1 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2   pageEncoding="ISO-8859-1"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <meta charset="ISO-8859-1">
7 <title>Insert title here</title>
8 </head>
9 <body>
10 <p>Hola Spring</p>
11 </body>
12 </html>

```



- ✓ Ahora implementamos la clase abstracta **SpringBootServletInitializer** que implementa de la interface **WebApplicationInitializer**. Vamos a ampliar la clase **SpringBootServletInitializer** y hemos anulado el método `configure()` de la misma, lo que permite que nuestra aplicación sea configurable cuando se inicia mediante cualquier contenedor web tradicional.
- ✓ La etiqueta **@Configuration**, indica que la clase en la que se encuentra contiene la configuración principal del proyecto.
- ✓ La anotación **@EnableAutoConfiguration** indica que se aplicará la configuración automática del starter que hemos utilizado. Solo debe añadirse en un sitio, y es muy frecuente situarla en la clase `main`.
- ✓ En tercer lugar, la etiqueta **@ComponentScan**, ayuda a localizar elementos etiquetados con otras anotaciones cuando sean necesarios.
- ✓ Para no llenar nuestra clase de anotaciones, podemos sustituir las etiquetas **@Configuration**, **@EnableAutoConfiguration** y **@ComponentScan** por **@SpringBootApplication**, que engloba al resto.

```
package com.gf.demo1;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class Demo1Application extends SpringBootServletInitializer{

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(Demo1Application.class);
    }

    public static void main(String[] args) {
        SpringApplication.run(Demo1Application.class, args);
    }
}
```

- ✓ Ahora se implementa la clase **HolaControlador.java**. **@GetMapping** asigna la solicitud "/" y "/hola" al método `mostrarPagina()`, que nos redirige a la página de vista "hola"(hola.jsp)
- ✓ **@GetMapping** – abreviado de **@RequestMapping(method = RequestMethod.GET)**

```
@Controller
public class HolaControlador {

    @GetMapping({ "/", "/hola" })
    public String mostrarPagina() {
        return "hola";
    }

}
```

✓ Hay dos formas de resolver la JSP:

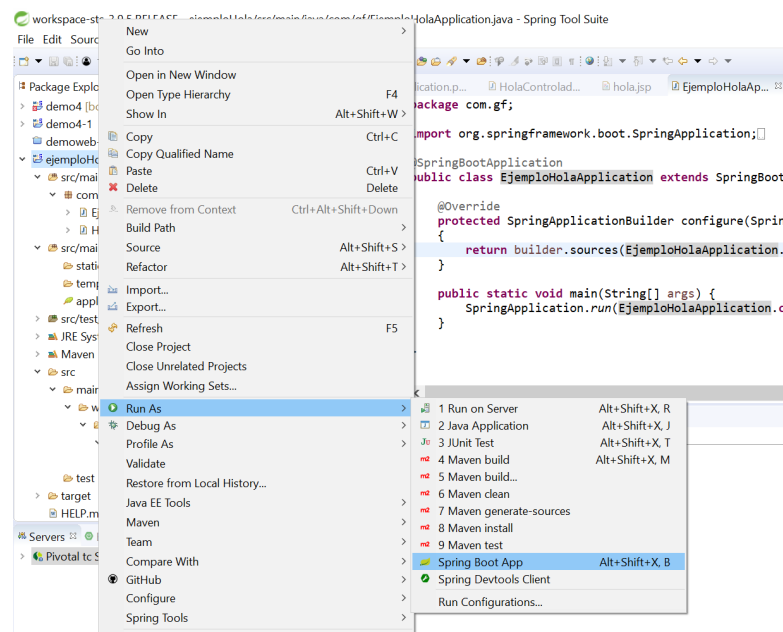
- ✓ **Uso de application.properties:** Esta es la forma más sencilla para resolver los archivos JSP, todo lo que necesita hacer es agregar las dos entradas siguientes en el archivo application.properties y Spring Boot se encargará del resto.

```
1 # Spring MVC settings
2 spring.mvc.view.prefix: /WEB-INF/view/
3 spring.mvc.view.suffix: .jsp
```

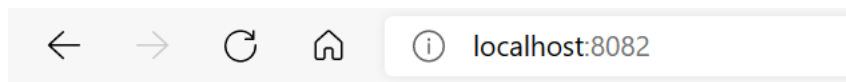
- ✓ **Modificando el fichero de configuración:** En este enfoque, tendremos que definir manualmente el View Resolver que debe utilizarse al resolver las páginas JSP

```
1 package com.gf;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.web.servlet.config.annotation.EnableWebMvc;
6 import org.springframework.web.servlet.view.InternalResourceViewResolver;
7
8 @Configuration
9 @EnableWebMvc
10 public class SpringConfig {
11     @Bean
12     public InternalResourceViewResolver viewResolver()
13     {
14         InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();
15         viewResolver.setPrefix("/WEB-INF/view/");
16         viewResolver.setSuffix(".jsp");
17
18         return viewResolver;
19     }
20 }
21 }
```

- ✓ A continuación, ejecutamos la aplicación creada desde la raíz del propio proyecto:





- ✓ Igualmente, se puede ejecutar desde Run on server ya que el empaquetado es un .war.
- ✓ Desde un navegador introducimos la URL localhost:8082, dependiendo el puerto que hayamos configurado para el servidor.



## HOLA MUNDO SPRING

- ✓ SpringBoot es un contenedor web incrustado, aunque permite el empaquetado en .war, se recomienda marcarlo como un paquete jar. Si utilizamos JSP, se puede crear WEB-INF \ web.xml y utilizar JSP como de costumbre. Usar un contenedor externo como “paquete de guerra”, equivale a perder algunas de las características de SpringBoot. Por lo que crearemos un proyecto Web empaquetando en .jar:

 **New Spring Starter Project** 

Service URL:

Name:

☒ Use default location

Location:

Type:  Packaging:

Java Version:  Language:

Group:

Artifact:

Version:



Description:

Package:

Working sets

☐ Add project to working sets

Working sets:

 **New Spring Starter Project Dependencies** 

Spring Boot Version:

Frequently Used:

☒ Spring Web

Available:

Selected:

☒ Spring Boot DevTools

☒ Spring Web

- ✓ Se necesita motor de compilación porque cada fichero JSP se transforma en un servlet que será compilado. Los jars dependientes para Tomcat están disponibles en tomcat jasper. Añadimos la dependencia tomcat jasper en el archivo pom.xml
- ✓ Crea un carpeta para guardar las JSPs dentro de webapp->WEB-INF->view.
- ✓ En el directorio WEB-INF, cree un archivo de configuración web.xml, la información de configuración es la siguiente:

```
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>

</web-app>
```

- ✓ Ahora implementamos la clase abstracta **SpringBootServletInitializer** que implementa de la interface **WebApplicationInitializer**. Vamos a ampliar la clase **SpringBootServletInitializer** y hemos anulado el método configure() de la misma, lo que permite que nuestra aplicación sea configurable cuando se inicia mediante cualquier contenedor web tradicional.
- ✓ La anotación **@SpringBootApplication** realiza el trabajo de: **@EnableAutoConfiguration @Configuration** y **@ComponentScan** juntas

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

import com.gf.demo2.Demo2Application;

@SpringBootApplication
public class Demo2Application extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(Demo2Application.class);
    }

    public static void main(String[] args) {
        SpringApplication.run(Demo2Application.class, args);
    }

}
```

- ✓ Ahora implemento la vista index.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<h1>Soy ${nombre} alumna de ${grupo}</h1>
</body>
</html>
```

- ✓ Ahora se implementa la clase **Controlador.java**. @GetMapping asigna la solicitud "/" y "/index" al método index(), que nos redirige a la página de vista "index"(index.jsp)
- ✓ @GetMapping – abreviado de @RequestMapping(method = RequestMethod.GET)

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

import jakarta.servlet.http.HttpServletRequest;

@Controller
public class Controlador {

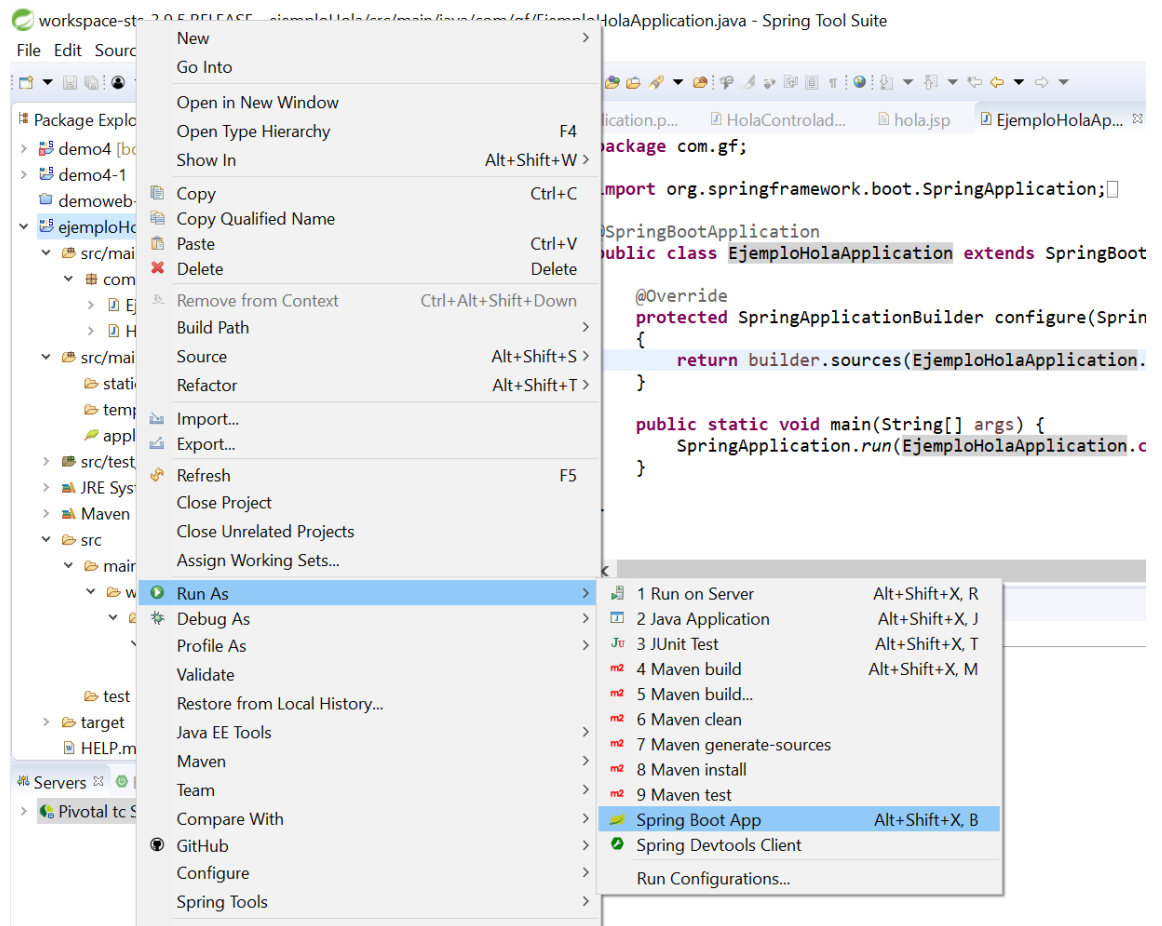
    @GetMapping("/index")
    public String viewPage(HttpServletRequest request) {
        request.setAttribute("nombre", "Ruth");
        request.setAttribute("grupo", "2°DAM");
        return "index";
    }

}
```

- ✓ **Uso de application.properties:** Esta es la forma más sencilla para resolver los archivos JSP, todo lo que necesita hacer es agregar las dos entradas siguientes en el archivo application.properties y Spring Boot se encargará del resto.

```
1 # Spring MVC settings
2 spring.mvc.view.prefix: /WEB-INF/view/
3 spring.mvc.view.suffix: .jsp
```

- ✓ A continuación, ejecutamos la aplicación creada desde la raíz del propio proyecto:



- ✓ Desde un navegador introducimos la URL localhost:8082, dependiendo el puerto que hayamos configurado para el servidor.

← → ↻ ⓘ localhost:8080/index

**Soy Ruth, alumna de 2ºDAM**