

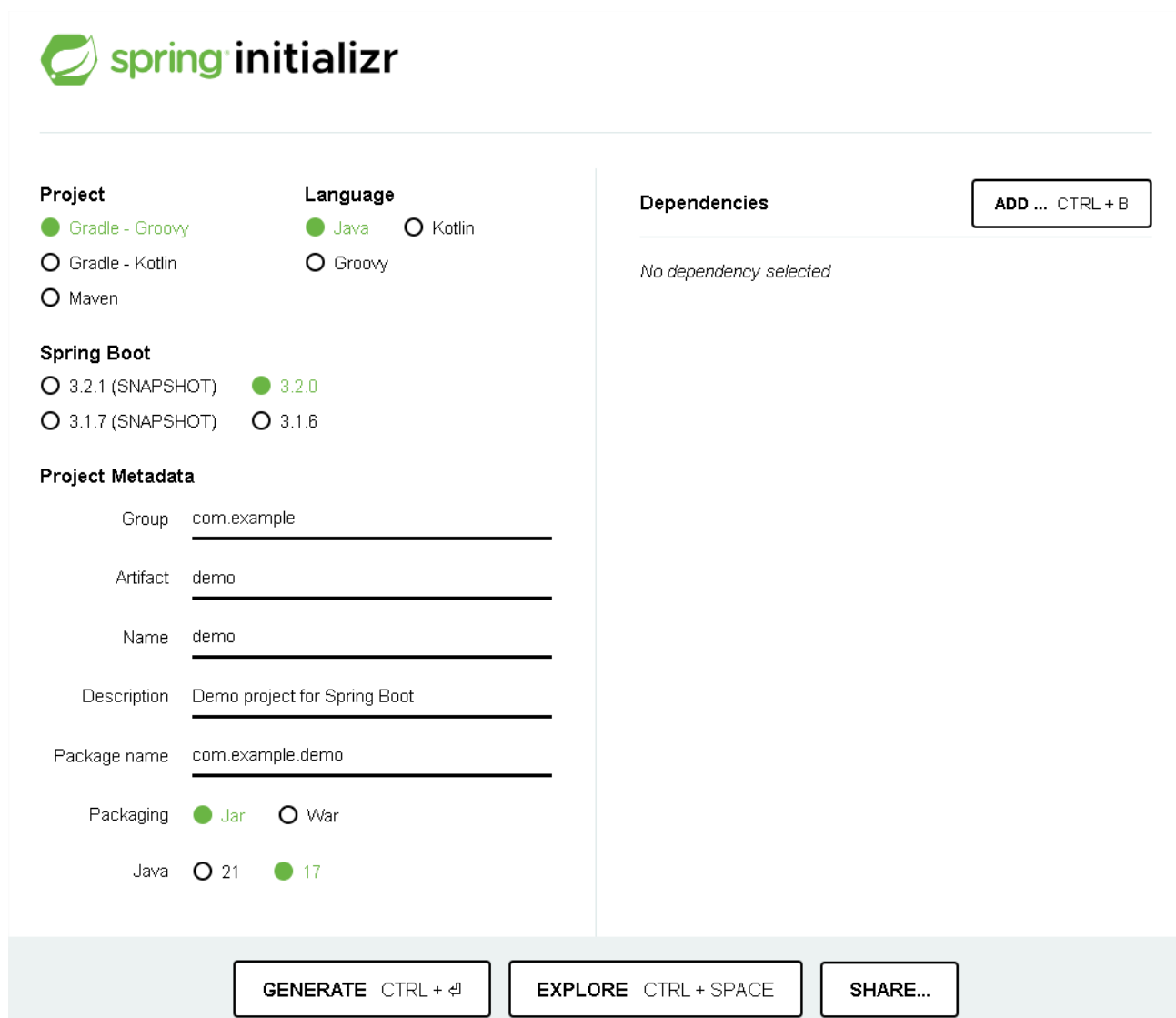
Ejercicio 10 – Primer proyecto con Spring Boot

En esta práctica vamos a realizar un primer proyecto utilizando Spring Boot, donde utilizaremos un controlador y una plantilla para mostrar información a través de la web.

Parte 1: Crear la aplicación

Para crear una aplicación Spring vamos a utilizar Spring Boot, que es un módulo del framework que facilita la creación de aplicaciones.

Por parte de los desarrolladores de Spring, tenemos a nuestra disposición una herramienta web online denominada [Spring Initializr](https://start.spring.io) donde por medio de unos parámetros de configuración genera automáticamente un proyecto Maven o Gradle, según elijamos, en un archivo comprimido Zip conteniendo la carpeta con la estructura de la aplicación para ser importada directamente desde el IDE.



The screenshot shows the Spring Initializr web application interface. It features a logo at the top left and a form for configuring a new Spring Boot project. The form is divided into several sections: Project, Language, Spring Boot, Project Metadata, and Dependencies. The Project section has radio buttons for Gradle - Groovy (selected), Gradle - Kotlin, and Maven. The Language section has radio buttons for Java (selected) and Kotlin. The Spring Boot section has radio buttons for 3.2.1 (SNAPSHOT), 3.2.0 (selected), 3.1.7 (SNAPSHOT), and 3.1.6. The Project Metadata section includes input fields for Group (com.example), Artifact (demo), Name (demo), Description (Demo project for Spring Boot), and Package name (com.example.demo). The Packaging section has radio buttons for Jar (selected) and War. The Java section has radio buttons for 21 and 17 (selected). The Dependencies section has a button to add dependencies and a message indicating no dependency is selected. At the bottom, there are three buttons: GENERATE (CTRL + G), EXPLORE (CTRL + SPACE), and SHARE...

El asistente web nos solicita una serie de datos necesarios para poder ejecutar la plantilla que construye los primeros archivos del programa. Para todos ellos aporta una configuración por defecto que conviene cambiar, como el nombre de la aplicación, o el package que usaremos en las clases generadas.

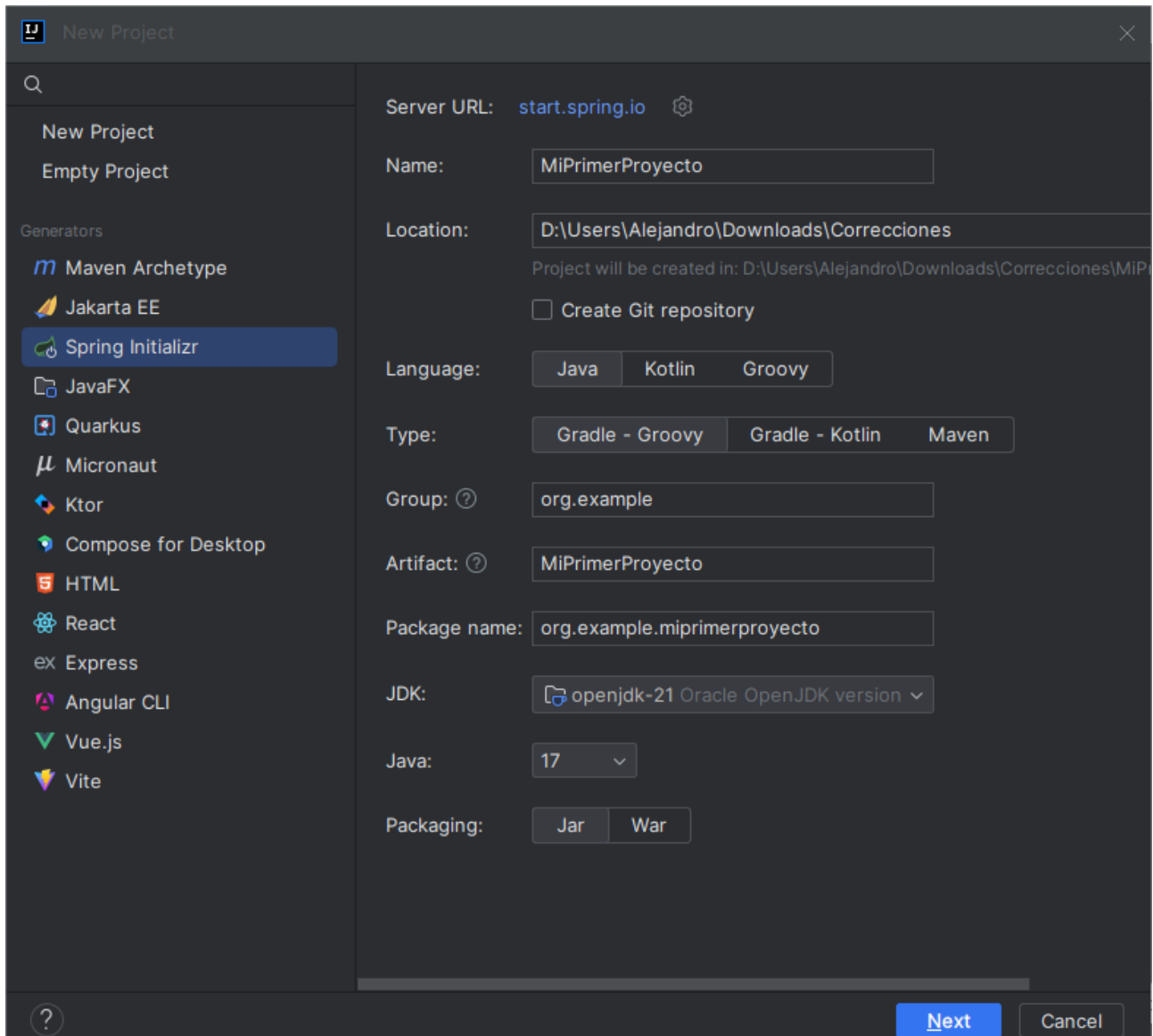
A continuación, explicamos qué parámetros hay y para qué sirven:

- **Project:** Permite elegir la herramienta de construcción de la aplicación. En Java las dos herramientas más usadas son Maven y Gradle.
- **Language:** Lenguaje de programación que se va a utilizar en la aplicación. Además de Java, tanto Kotlin como Groovy están soportados por la máquina virtual JVM.
- **Spring Boot:** Versión del Spring Boot a utilizar. Siempre que se pueda se optará por la última estable, compuesta únicamente por números, en este caso la 3.2.0
- **Project Metadata**
 - **Group:** Se refiere al descriptor groupId, utilizado para clasificar el proyecto en los repositorios de binarios. Normalmente se suele usar una referencia similar a la de los packages de las clases. Por ejemplo, com.dam.web para disponer todas las aplicaciones web en el mismo directorio.
 - **Artifact:** Se refiere a otro descriptor artifactId, e indica el nombre del proyecto y del binario resultante. La combinación de groupId y artifactId (más la versión) identifican inequívocamente a un binario dentro de cualquier organización.
 - **Packaging:** Indica qué tipo de binario se debe construir. Si la aplicación se ejecutará por sí sola se seleccionará JAR, éste contiene todas las dependencias dentro de él y se podrá ejecutar con `java -jar binario.jar`. Si, por el contrario, la aplicación se ejecutará en un servidor J2EE existente o en un Tomcat ya desplegado se deberá escoger WAR.
 - **Java:** Se selecciona la versión de Java a usar. En este caso, hay que valorar si escoger la versión más nueva o la más antigua, y así garantizar la compatibilidad con otras librerías o proyectos que se quieran incluir.
- **Dependencies:** Buscador de dependencias con los starters de Spring boot disponibles. Las dependencias más habituales son:
 - **Spring Web**, para aplicaciones web y microservicios, siempre que se requiera una comunicación http y, por tanto, el uso de Spring MVC.
 - **Thymeleaf** Incorpora el motor de plantillas para HTML dinámico, sucesor de los anteriores JSP (Java Server Page).
 - **Spring Data JPA** para utilizar la capa estándar de acceso a base de datos SQL.
 - **Spring Security**, que permite incorporar controles de acceso en base a usuarios y roles sobre URLs de la aplicación. También habilita el control de ejecución de métodos de servicio en base a roles según los estándares J2EE.
 - **Lombok** aporta utilidades que facilitan la programación y la creación de @Getters y @Setters automáticamente para las clases que forman parte del conjunto de mensajes.
 - **Mysql/Postgresql** incluye el JAR que contiene el driver JDBC necesario para configurar la capa de JPA según la base de datos a usar.

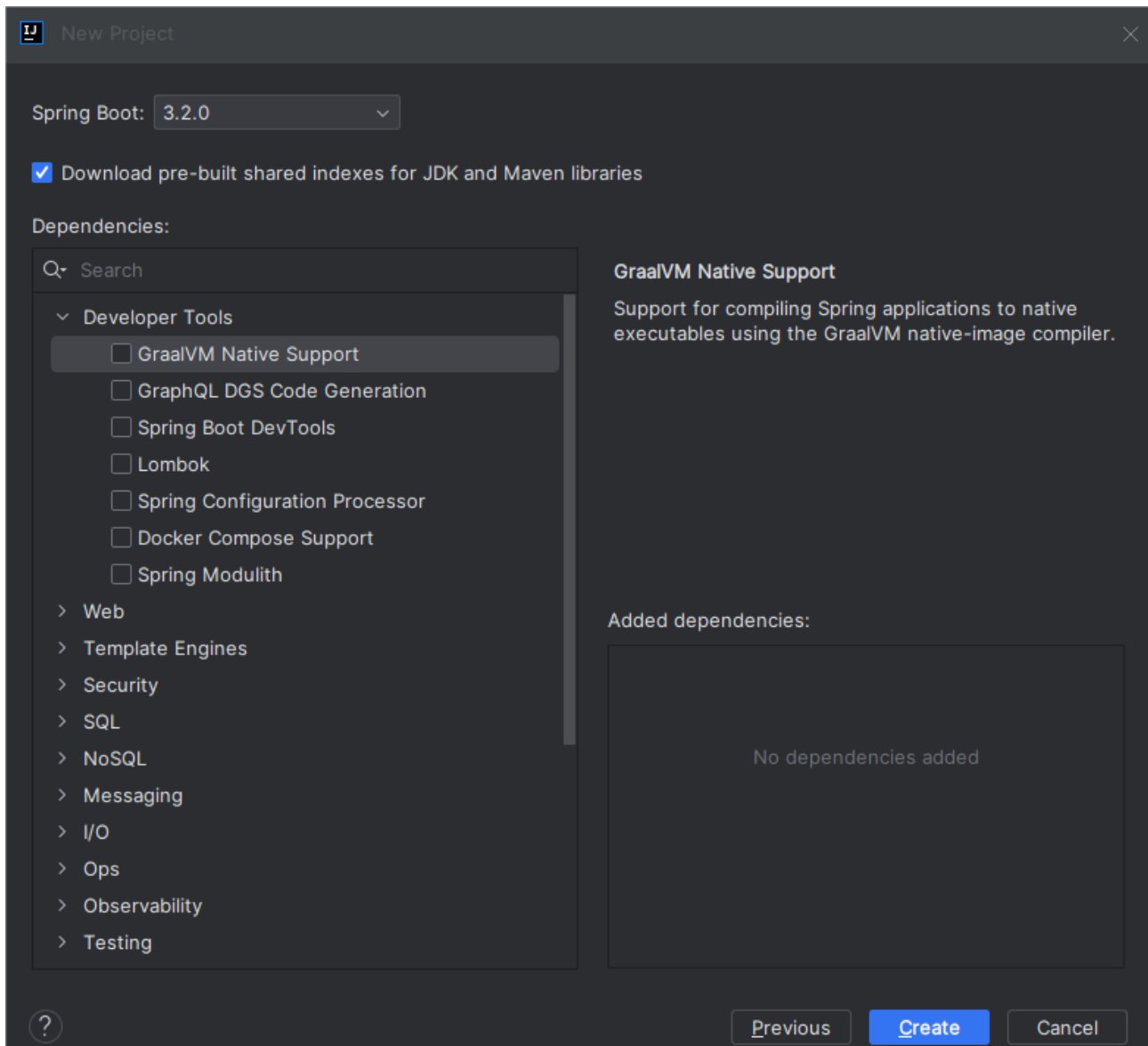
- **Otras...** El asistente permite seleccionar entre alrededor de 100 dependencias e integraciones de herramientas open source dentro de los proyectos realizados con Spring.

Una vez seleccionados los parámetros que se quieren, haciendo click en el botón **Generate** se descargará un archivo zip con el nombre del Artifact que contendrá la carpeta con la estructura de la aplicación lista para importar desde el IDE.

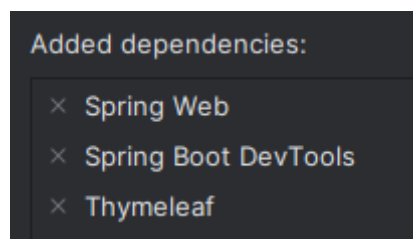
La creación de un proyecto también se puede realizar desde el IDE, teniendo instalados los plugins necesarios, como Spring Tools. En IntelliJ, podemos crear un proyecto Spring como sigue:



Como podemos comprobar, tenemos las mismas opciones que ofrece la versión web. Una vez hacemos clic en **Next**, podemos elegir los starters y dependencias a instalar.



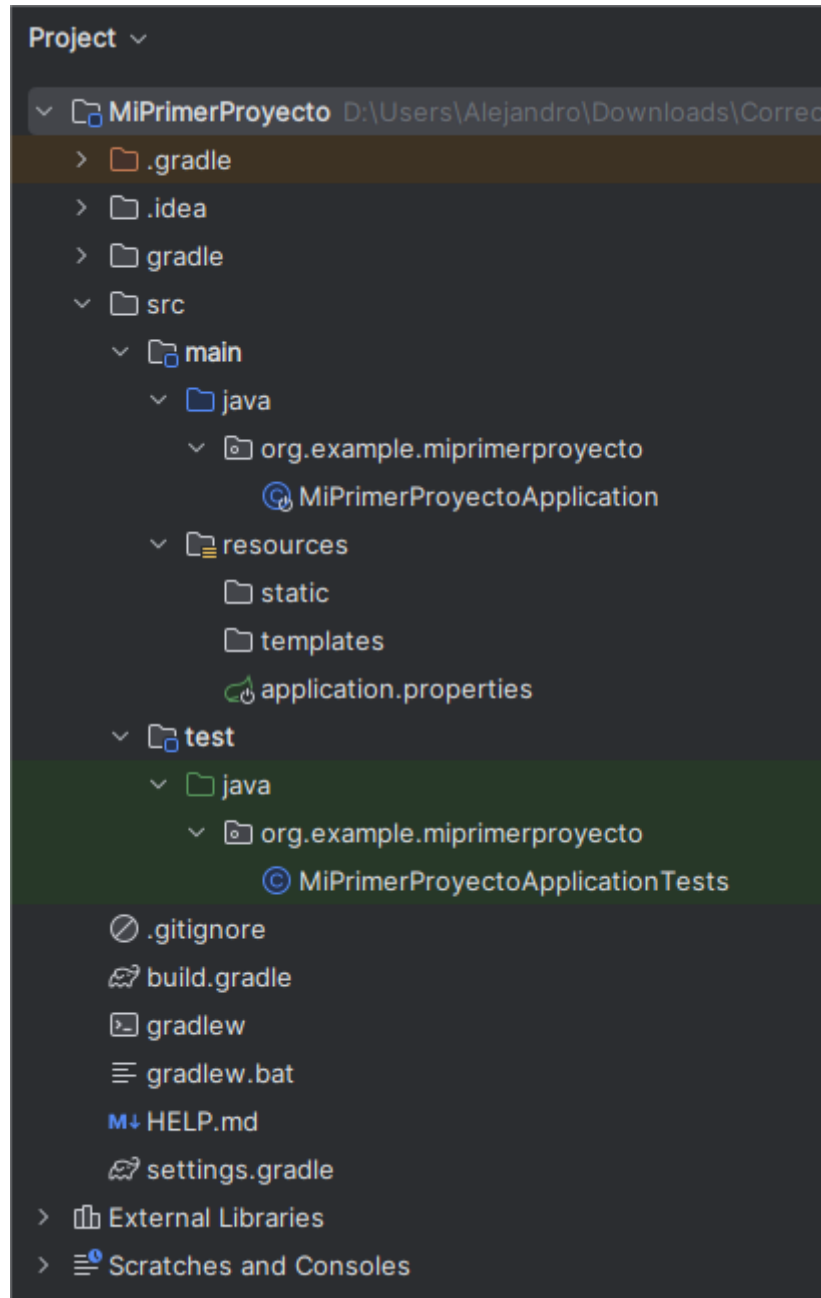
Como ejemplo, hemos realizado la creación del proyecto a través del IDE IntelliJ con las siguientes dependencias:



Haz clic en **Create** y espera que se generen los archivos del proyecto.

Parte 2: Recursos generados por Spring Boot Initializr

Una vez creado el proyecto, su contenido será el siguiente:



De los ficheros incluidos, el más importante es **MiPrimerProyectoApplication**, que corresponde con el punto de entrada a la ejecución del programa. Aquí es donde se aloja el método `main` que inicializa toda la aplicación de Spring Boot.

Como hemos visto en la teoría, dentro de este archivo tenemos una clase con la anotación **@SpringBootApplication**.

Durante el arranque del framework, Spring revisa el resto de directorios o paquetes que cuelgan del método main en búsqueda de clases marcadas con alguna anotación que permite el registro de componentes como: **@Component**, **@Service**, **@Repository**, **@Controller**, **@Entity**, etcétera.

El siguiente fichero más importante es **application.properties** que es donde se aloja toda la configuración de los componentes de Spring Boot, cómo que codificación usar, si debe usar caché en las plantillas, qué nombre y clave de acceso utilizar en las distintas bases de datos, etc... En [este enlace](#) de configuraciones y propiedades comunes de Spring Boot hay una referencia de todas las existentes y sus valores iniciales.

El siguiente fichero es **MiPrimerProyectoApplicationTests**, donde se aloja el primer test de ejemplo que genera Initializr por nosotros. Ahí deberíamos añadir los test que consideremos oportunos para aplicar en la medida de lo posible desarrollo orientado a pruebas, o TDD.

Los otros directorios son **static** que es donde se alojan los recursos estáticos que la aplicación debe servir sin procesar. Por ejemplo, en esta carpeta se incluyen los ficheros CSS, Javascript, imágenes o fuentes que se referencian desde el HTML.

Si alojamos un fichero en un subdirectorio de static, como puede ser static/css/main.css, Spring Boot lo publicará automáticamente en la url <http://localhost:8080/css/main.css>. Así que no debe haber información sensible que cuelgue del directorio static porque entonces será accesible abiertamente.

Y por último la carpeta **templates** almacena los ficheros que permiten generar HTML dinámicamente con algún motor de plantillas soportado por Spring como Thymeleaf o Freemarker.

Parte 3: Ejemplo básico de funcionalidad

El proyecto creado es muy simple. Al arrancar el método main de la clase **MiPrimerProyectoApplications**, veremos cómo Spring inicializa toda la aplicación y se pone a la escucha del puerto 8080, pero si dirigimos el navegador a esa URL no aparecerá ningún contenido. Así que veamos cómo podemos hacer para imprimir algunos datos mediante html dinámico.

Vamos a crear las rutas mediante un controlador de tipo **@Controller**. Este controlador se encargará de recibir las peticiones de los clientes y devolver las respuestas.

Las peticiones que vamos a recibir por el momento serán de tipo HTTP GET, solicitando información que mostraremos a través de una plantilla.

La información que mostraremos vendrá almacenada en una variable que traspasaremos a través del modelo a la vista.

Crea un directorio **controllers** bajo el directorio /src/main/java/org/example/miprimerproyecto. En dicho directorio crea una clase **MiPrimerControlador** con el siguiente contenido:

```
@Controller
public class MiPrimerControlador {
    @GetMapping("/")
    public String index(Model model) {
        model.addAttribute("hora", LocalDateTime.now());
        return "index";
    }
}
```

El controlador registra un método en "/", la raíz del sitio, de tal manera que cuando la aplicación esté arrancada y lista para escuchar peticiones http, normalmente por el puerto 8080, el método index será ejecutado cuando se solicite la URL de inicio `http://localhost:8080/`.

En ese momento, el método será ejecutado añadiendo la hora actual a una variable con nombre hora que se guarda en model, objeto que se pasa a la vista, para que pueda construir el html dinámicamente. En este caso, mostrando la hora en la que se invocó ese método.

Por otro lado, nada más ejecutarse el método del controlador, Spring pasará el control a la vista con nombre index correspondiente al fichero **index.html** de la carpeta de **templates**, tal y como indica el return "index", en el que no hace falta indicar la extensión .html.

Para completar el ejemplo y ver los resultados necesitamos crear la vista index.html con el siguiente contenido:

```
<!DOCTYPE html>
<html lang="es" xmlns:th="http://www.thymeleaf.org">
<head>
  <title>Mi primer ejemplo con Spring</title>
<style>
  html {
    font-family: "DejaVu Sans", Arial, Helvetica, sans-serif;
    color: #404040;
    background-color: #d3d3d3;
  }
  body {
    padding: 5em;
  }
</style>
</head>
<body>
<h1>¡Hola! ¡Este es mi primer ejemplo con Spring!</h1>
<p>Esta página ha sido servida a las <span th:text="${#temporals.format(hora, 'HH:mm')}"></span></p>
</body>
</html>
```

Arranca de nuevo la aplicación y vuelve a hacer la petición a `http://localhost:8080/`. Debería dar como resultado la impresión de la hora en la que se ejecutó el controlador.

