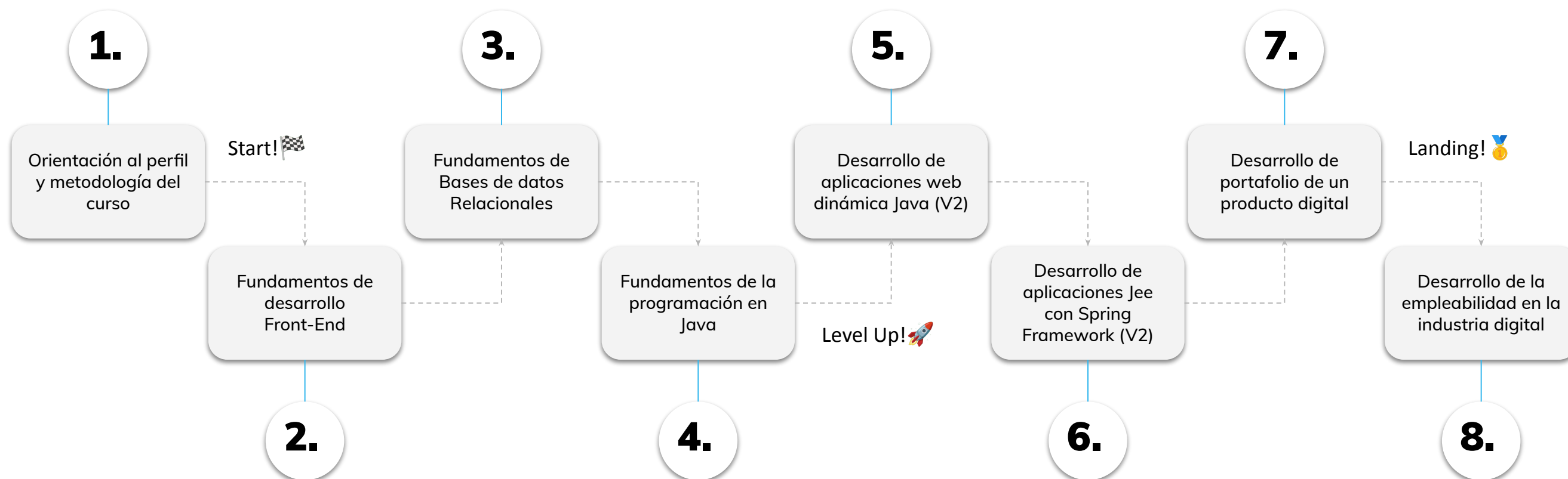


➤ El Framework Spring MVC

Plan formativo: Desarrollo de Aplicaciones Full Stack Java Trainee V2.0

HOJA DE RUTA

¿Cuáles **skill** conforman el programa?



REPASO CLASE ANTERIOR


En la clase anterior trabajamos :

- ✓ Características de *Spring Initializr*
- ✓ Configuración de un proyecto Spring

LEARNING PATHWAY

¿Sobre qué temas trabajaremos?

6.2

Start! 

El Framework Spring MVC

Vistas

“Hola Mundo” con Spring

Configuración de la tecnología de vista (JSP, Thymeleaf)
Configuración de log en Spring MVC
Configuración del datasource

OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?



Conocer las configuraciones para tecnologías de vistas



Comprender la configuración de un datasource

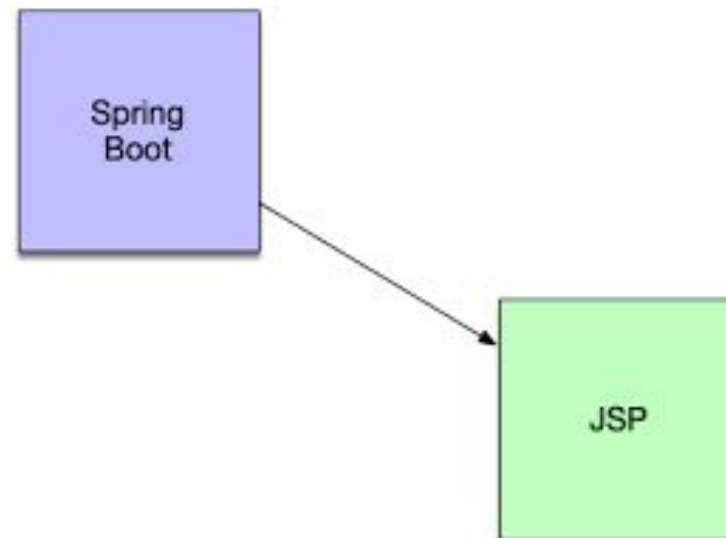


› Configuración de Tecnologías de Vista

Configuración de la tecnología de vista

Spring Boot JSP

Las aplicaciones de Spring Boot JSP pueden ser más habituales de lo que en un primer momento nos parece . Sí bien es cierto que usar Spring Boot es usar tecnologías modernas que facilitan que despleguemos arquitecturas orientadas a MicroServicios, hay que recordar que siguen existiendo aplicaciones no tan “modernas” y quizás podamos necesitar que se ejecuten dentro de Spring Boot.





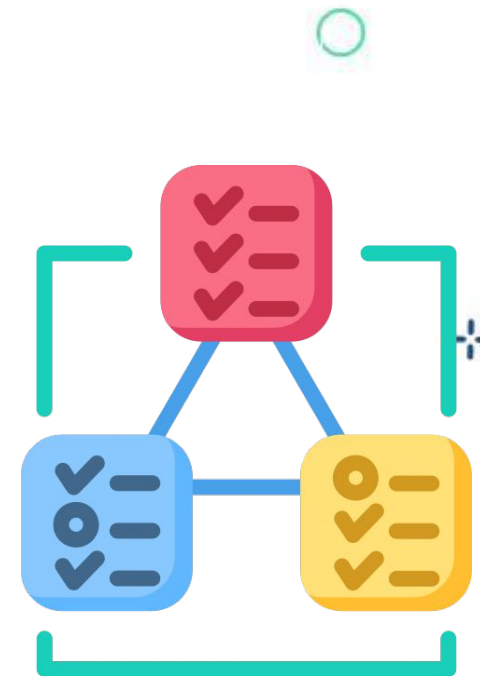
Configuración de la tecnología de vista

Spring Boot JSP

Con el proyecto generado en el punto anterior, ya tenemos la primera dependencia necesaria para un proyecto Spring Boot JSP, a continuación, deberíamos integrar también la siguiente dependencia (puede colocarse directamente en el pom.xml en el caso de que el proyecto ya esté generado).

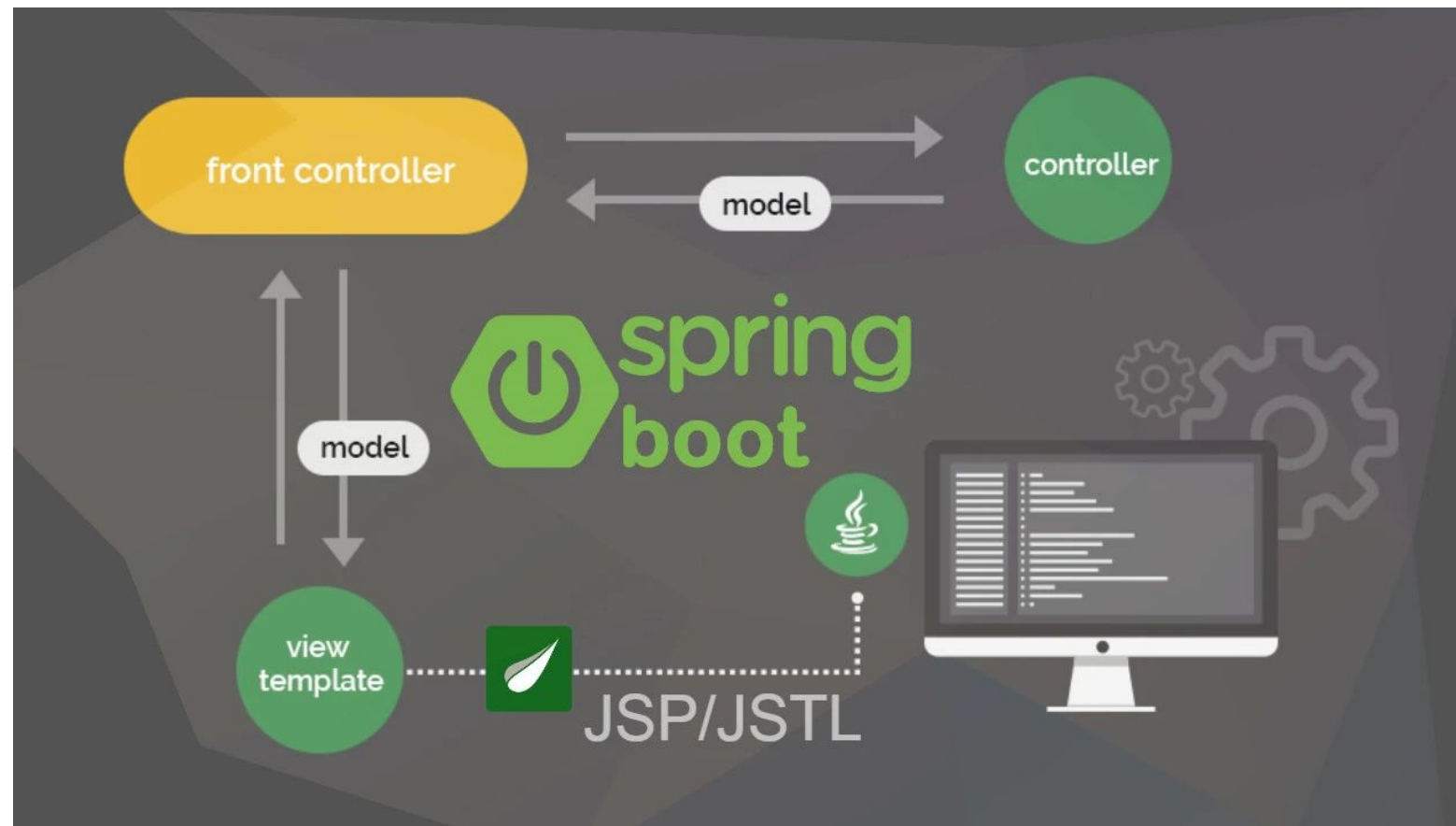
JSP requiere la dependencia del lenguaje JSTL, basado en etiquetas (Jakarta Standard Tag Library):

```
<dependency>
+   <groupId>org.glassfish.web</groupId>
    <artifactId>jakarta.servlet.jsp.jstl</artifactId>
</dependency>
```



Configuración de la tecnología de vista

Spring Boot JSP



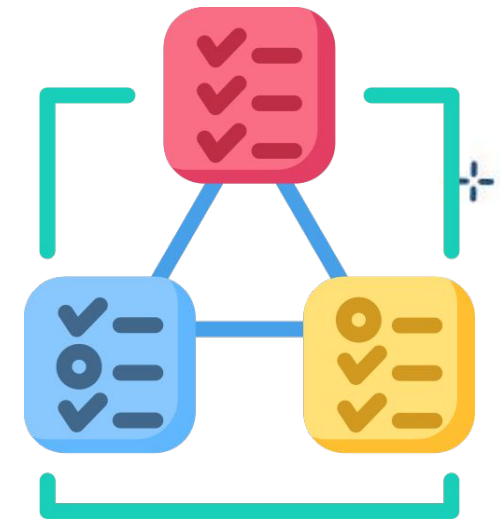


Configuración de la tecnología de vista

Spring Boot JSP

También precisa un motor de compilación porque cada fichero JSP se transforma en un servlet que será compilado. Los servlets son clases Java que procesan peticiones y respuestas HTTP.

```
<dependency>  
  <groupId>org.apache.tomcat.embed</groupId>  
  <artifactId>tomcat-embed-jasper</artifactId>  
  <scope>provided</scope>  
</dependency>
```





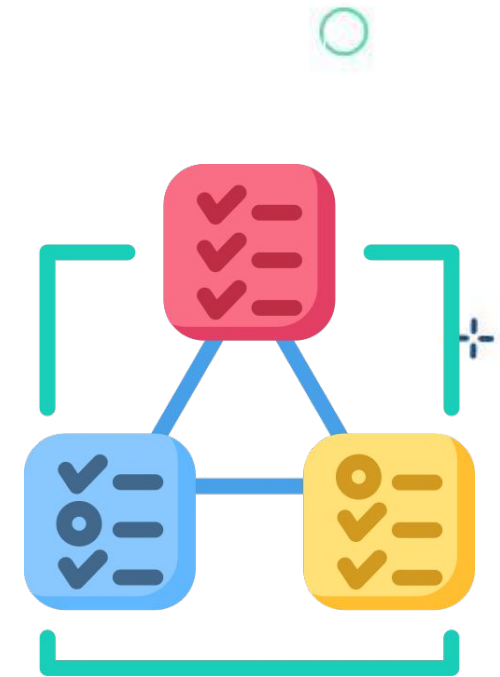
Configuración de la tecnología de vista

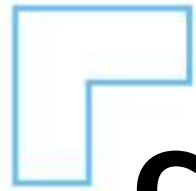
Spring Boot JSP

En el fichero de configuración `application.properties` definimos el prefijo y sufijo de los ficheros `.jsp`. El primero es la ruta en la que se ubican dentro de la carpeta `/src/main/webapp` y el segundo la extensión.

```
spring.mvc.view.prefix= /WEB-INF/jsp/  
spring.mvc.view.suffix= .jsp
```

De esta manera ya está completa la infraestructura inicial. Luego deberíamos escribir una clase especial denominada «controlador» con métodos que serán invocados cuando llamemos a ciertas direcciones vía HTTP.





Configuración de la tecnología de vista



Spring Boot Thymeleaf

Thymeleaf es una tecnología que permite definir una plantilla y, junto con un modelo de datos, obtener un nuevo documento. Es lo que se conoce como motor de plantillas.



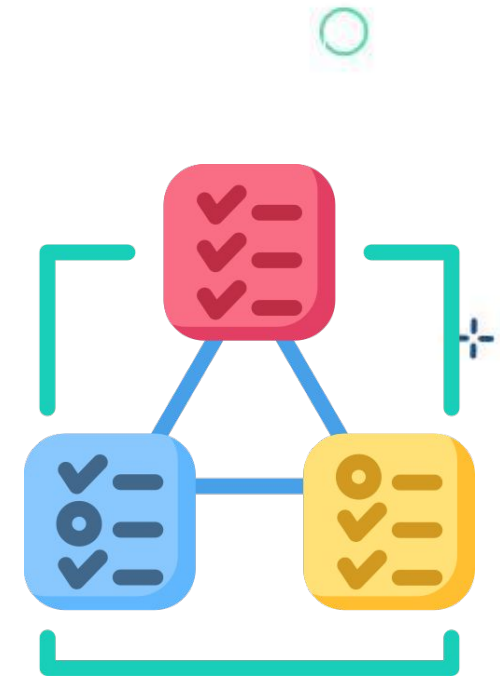


Configuración de la tecnología de vista

Spring Boot Thymeleaf

El motor de plantillas se genera para separar la interfaz de usuario (Vistas), de los datos comerciales (Modelos), pudiendo generar documentos en un formato específico, en este caso el motor de plantillas para el sitio web generará un documento HTML estándar.

Los motores de plantillas (templates engines) leen un fichero de texto que contiene la presentación ya preparada en HTML, e inserta en él la información dinámica que le ordena el Controlador, la parte que une la vista con la información.





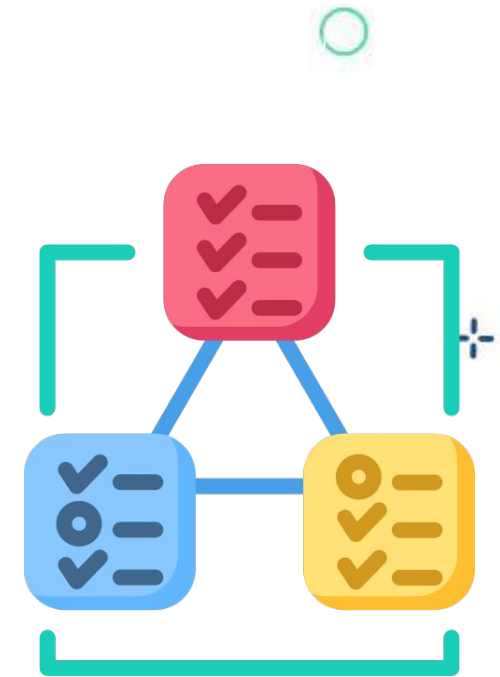
Configuración de la tecnología de vista

Spring Boot Thymeleaf

La sintaxis a utilizar depende del motor de plantillas utilizado, pero todos son muy similares. Los motores de plantillas suelen tener un pequeño lenguaje de script que permite generar código dinámico, como listas o cierto comportamiento condicional, pero esto también depende del lenguaje.

Este lenguaje de script es absolutamente mínimo, lo justo para posibilitar ese comportamiento dinámico:

```
<html>
+ <body>
    Hola ${nombre}
  </body>
</html>
```



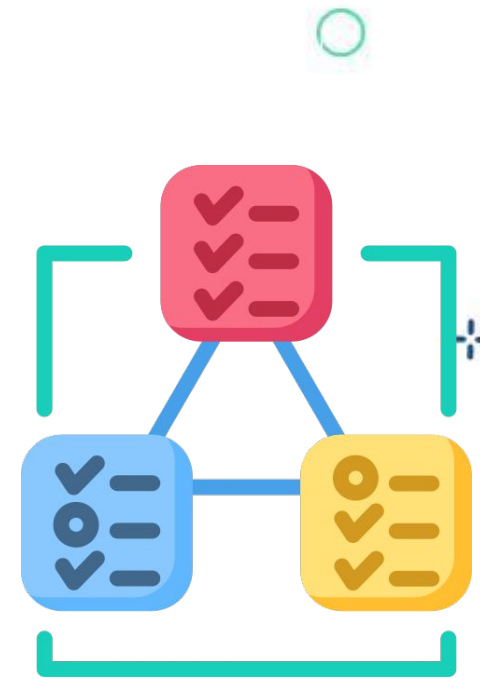


Configuración de la tecnología de vista

Spring Boot Thymeleaf

Básicamente, el motor de plantillas se encarga de recibir una variable de tipo **String** llamada **nombre**, que luego se va a enviar desde el controlador a la vista y la hará parte del HTML, **haciéndolo dinámico**. Por lo que los diferentes usuarios verán diferentes resultados.

Thymeleaf permite realizar tareas que se conocen como natural templating. Es decir, cómo está basada en añadir atributos y etiquetas, sobre todo HTML, va a permitir que nuestras plantillas se rendericen en local, para que luego sean procesadas dentro del motor de plantillas (por lo cual las tareas de diseño y programación se pueden llevar a cabo conjuntamente).



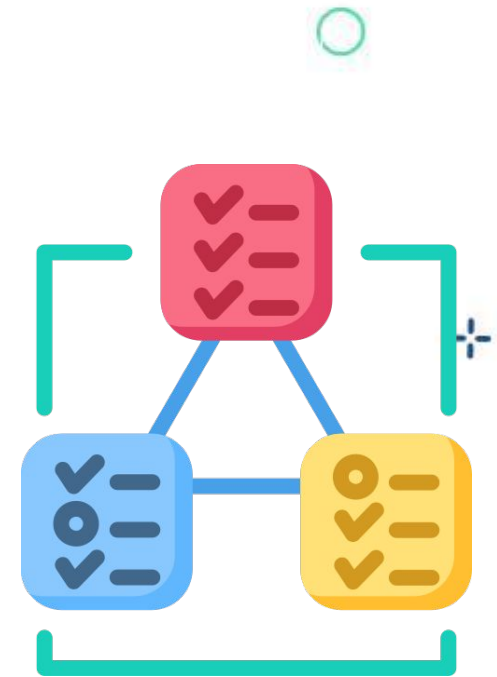


Configuración de la tecnología de vista

Spring Boot Thymeleaf

Permite trabajar con varios tipos de expresiones:

- **Expresiones variables:** Son quizás las más utilizadas y hacen referencia a una variable en particular, como por ejemplo `${variable}`
- **Expresiones de selección:** Son expresiones que nos permiten reducir la longitud de la expresión si pre fijamos un objeto mediante una expresión variable, como por ejemplo `*{objeto}`
- **Expresiones de enlace:** Nos permiten crear URL que pueden tener parámetros o variables, como por ejemplo `@{link}`




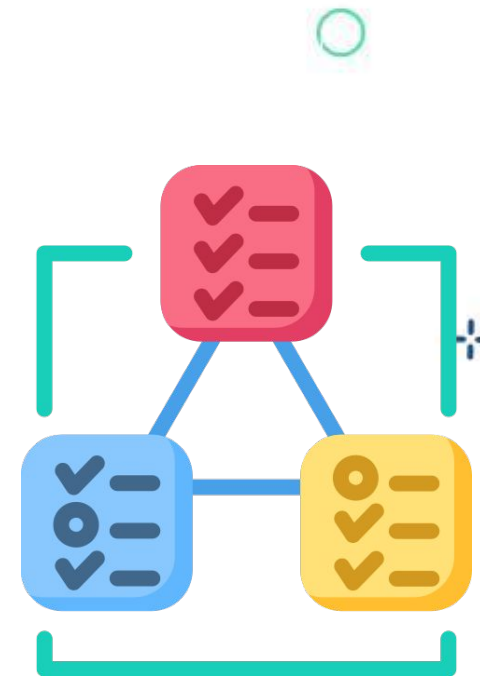


Configuración de la tecnología de vista

Spring Boot Thymeleaf

Permite trabajar con varios tipos de expresiones:

- **Expresiones variables:** Son quizás las más utilizadas y hacen referencia a una variable en particular, como por ejemplo `${variable}`
- **Expresiones de selección:** Son expresiones que nos permiten reducir la longitud de la expresión si pre fijamos un objeto mediante una expresión variable, como por ejemplo `*{objeto}`
-  **Expresiones de enlace:** Nos permiten crear URL que pueden tener parámetros o variables, como por ejemplo `@{link}`



› Configurando un log en Spring MVC

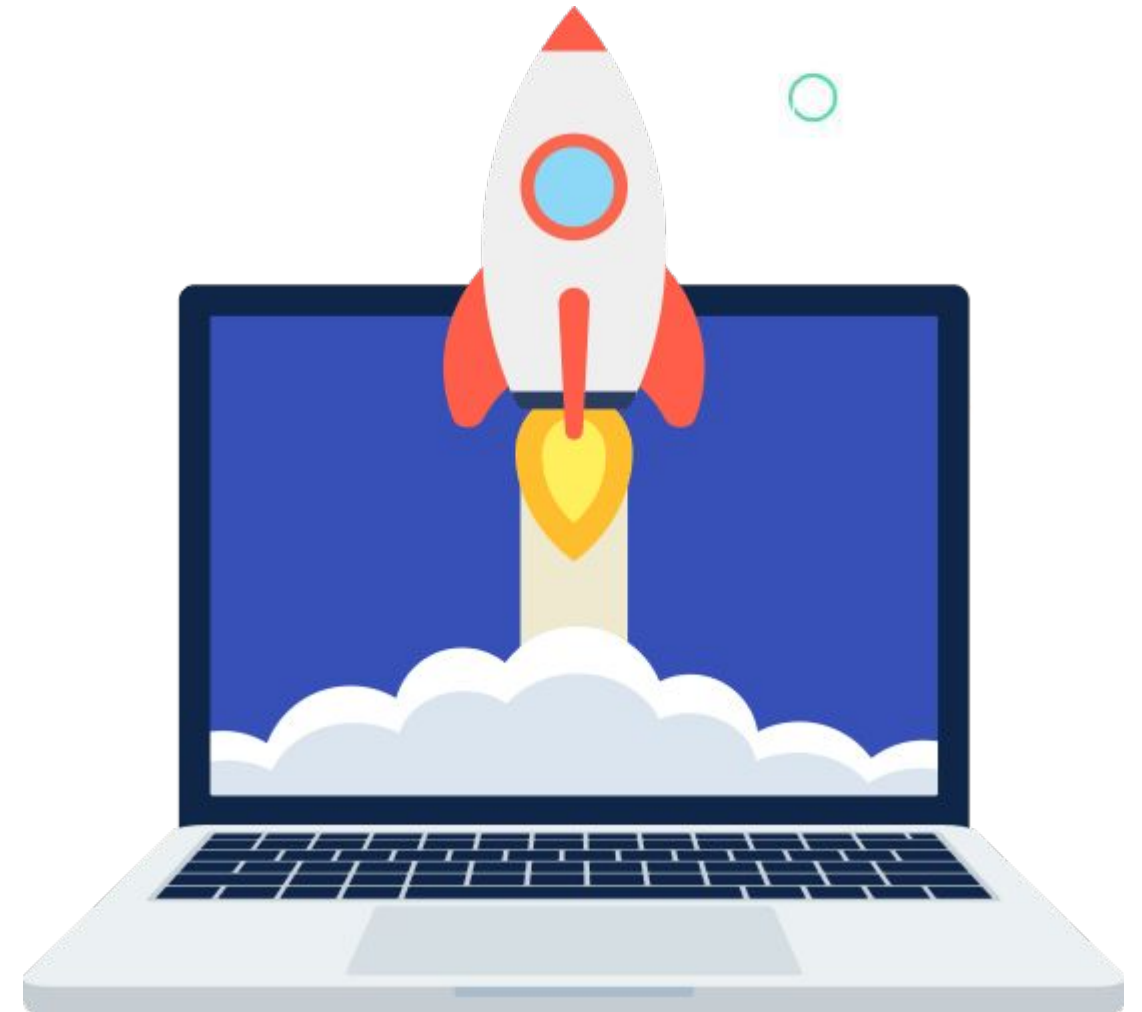


Configurando un log en Spring MVC

Una vez que tenemos nuestra fantástica aplicación realizada con Spring lo normal es que queramos ver cómo se va comportando. Para ello, la manera más fácil es escribir mensajes dentro de la aplicación, explicando por cuáles funciones entra, como se toman las decisiones y, en general, como se va comportando.

Redirigiendo la salida

Spring Boot, como cualquier aplicación, por defecto mostrará todos los mensajes y/o errores por la salida estándar, pero a menudo deseamos guardar esos mensajes en un fichero. Para ello hay varias opciones.





Configurando un log en Spring MVC

La primera y más obvia es redirigir salida estándar y errores a un fichero. Así, en Linux y/o Windows, podríamos ejecutar nuestra aplicación de esta manera:

```
java -jar APP.JAR > PATH/FICHERO.LOG 2>&1
```

Esta manera funciona, pero tendremos que mirar en el fichero generado para ver como va todo, sin posibilidad de ver inmediatamente en nuestra consola como está funcionando el proceso recién lanzado.

Si lo que deseamos es que deje la salida estándar en un fichero, pero, que además, nos muestre en la consola esa misma salida, podremos lanzar el proceso de esta manera:

```
java -jar -Dlogging.file.name=PATH/FICHERO.LOG  
APP.JAR
```





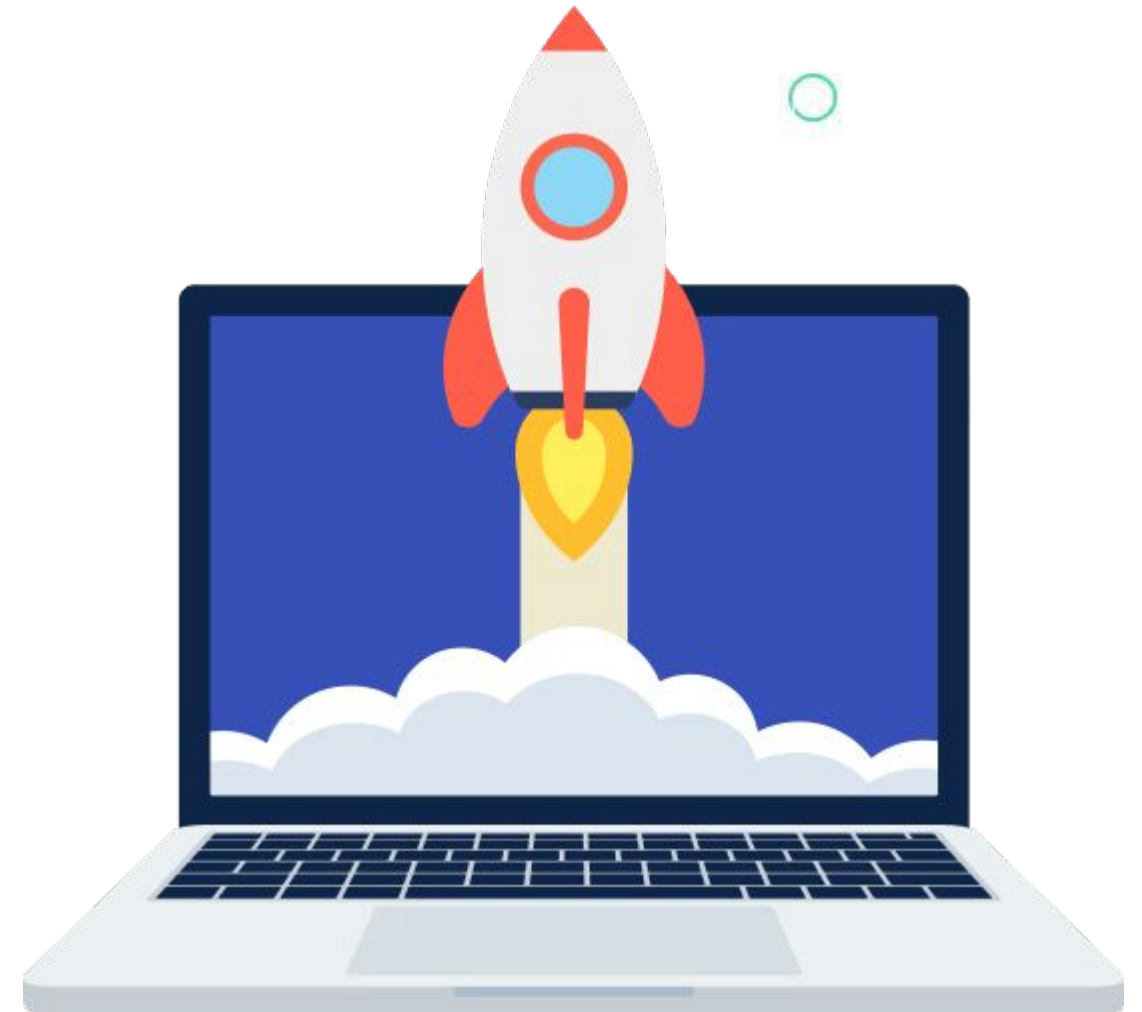
Configurando un log en Spring MVC

Por supuesto también podemos poner en nuestro fichero application.properties de configuración esa misma propiedad y podremos lanzar el proceso sin el parámetro '-D'

`logging.file.name=PATH/FICHERO.LOG APP.JAR`

Si queremos ejecutar el proceso desde maven pondremos este comando:

`mvnspring-boot:run-Dspring-boot.run.arguments=-logging.file.name=PATH/FICHERO.LOG`





Configurando un log en Spring MVC

Configurando nivel de log



Por supuesto los mensajes que deseamos mostrar los podemos escribir con un simple `system.out.print`, pero desde luego no sería muy elegante ni práctico. Siempre es recomendable usar un `logger`.



Spring utiliza Commons Logging, usando por defecto Logback, si bien incluye también configuraciones por defecto para Java Util Logging y Log4J2.

Por defecto Spring Boot muestra los mensajes de nivel `ERROR-level`, `WARN-level`, y `INFO-level`, pero esto se puede cambiar de diferentes maneras.

La más simple es añadiendo el parámetro `--trace` o `--debug` cuando ejecutemos nuestro programa.

```
java -jar APP.JAR --debug
```



Configurando un log en Spring MVC



Configurando nivel de log

Otra manera será especificar en la propiedad '**logging.level.root**'. Esto lo haremos añadiendo la línea correspondiente en el fichero application.properties, o de otra manera, a la hora de ejecutarlo, añadiendo el parámetro adecuado:



```
java -jar -Dlogging.level.root=DEBUG APP.JAR
```

Una vez más si usamos maven para ejecutar el proceso, debemos escribir el siguiente comando:

```
mvn spring-boot:run -Dspring-boot.run.arguments =-logging.level.root=TRACE
```





Configurando un log en Spring MVC

Configurando nivel de log

Por supuesto, se permite especificar niveles de logging por paquetes, así, si queremos que nuestra aplicación que se encuentra en el paquete 'com.example.demo' muestre los mensajes de nivel debug, pero el resto de aplicaciones solo muestren los mensajes de error, escribiremos esta sentencia:

```
java -jar -Dlogging.level.root=ERROR  
-D-Dlogging.level.com.example.demo=DEBUG  
APP.JAR
```





Configurando un log en Spring MVC

Formateando la salida



Spring por defecto muestra los mensajes de error con este formator:

- Día y hora, incluyendo milisegundos.
- Nivel de Log: ERROR, WARN, INFO, DEBUG, o TRACE.
- Identificador o número del proceso
- Un --- como separador
- El nombre del Thread entre corchetes.
- El nombre de la clase donde se está escribiendo el log.
- El mensaje de log.



```
2019-03-05 10:57:51.112 INFO 45469 --- [main]
org.apache.catalina.core.StandardEngine : Starting Servlet
Engine: Apache Tomcat/7.0.52
2019-03-05 10:57:51.253 INFO 45469 --- [ost-startStop-1]
o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring
embedded WebApplicationContext
```



Configurando un log en Spring MVC



También podemos especificar que use colores para cuando muestre en la consola los logs, con la siguiente variable:

`spring.output.ansi.enabled=true`.

Así, los mensajes de error los mostrara en ROJO, los de aviso (WARN) en amarillo y los demás, en verde.

Configuración avanzada

Sí Spring Boot encuentra en el `classpath` un fichero con alguno de estos nombres, lo usará para configurar los loggers.

- logback-spring.xml
- logback.xml
- logback-spring.groovy
- logback.groovy

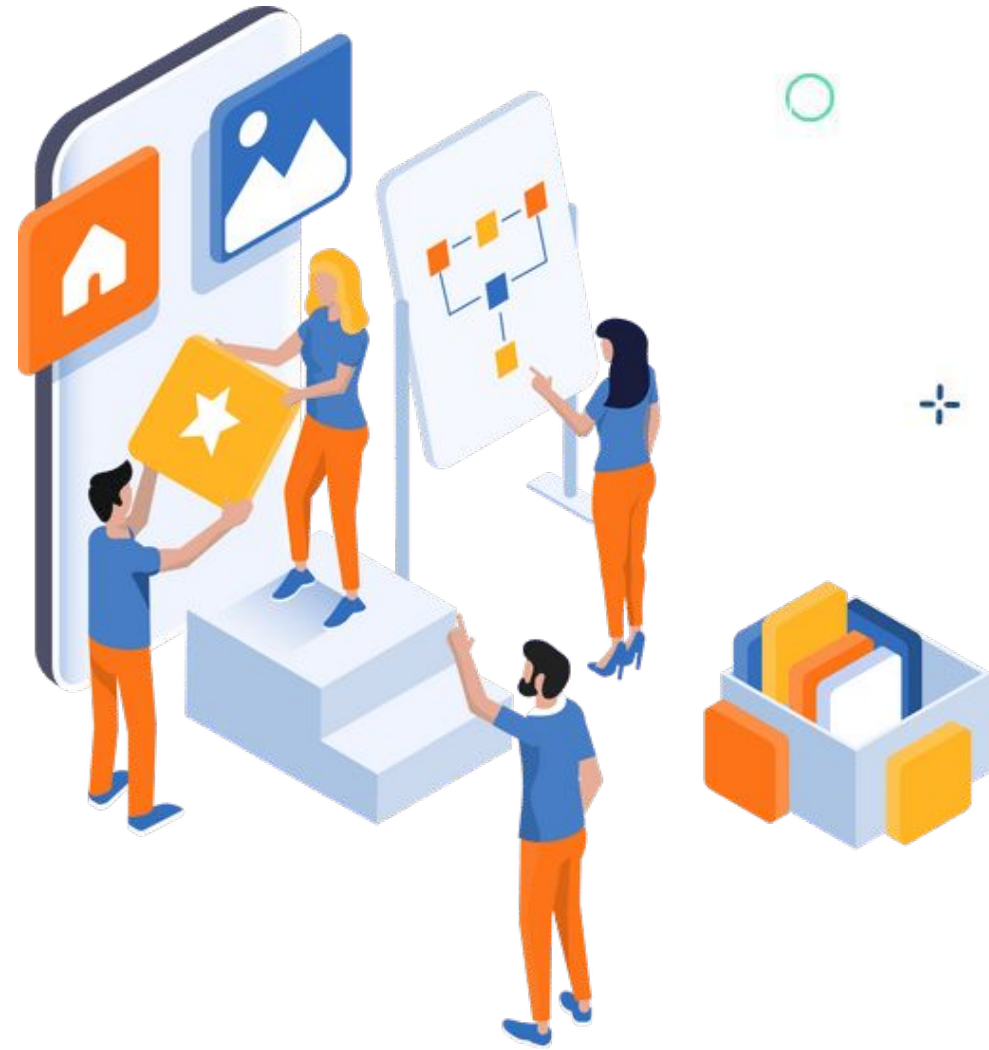
› Configurando el DataSource



Configurando el DataSource

Spring Boot utiliza un algoritmo propio para buscar y configurar un DataSource. Esto nos permite obtener fácilmente una implementación de DataSource completamente configurada de forma predeterminada. Además, Spring Boot configura automáticamente un conjunto de conexiones ultrarrápidas, ya sea HikariCP, Apache Tomcat o Commons DBCP, en ese orden, dependiendo de cuáles están en la ruta de clases.

Si bien la configuración automática de DataSource de Spring Boot funciona muy bien en la mayoría de los casos, a veces necesitaremos un mayor nivel de control.





Configurando el DataSource

Lo primero que haremos será editar el fichero de configuración del proyecto (application.properties) para personalizarlo: application.properties

Configuración para el acceso a la Base de Datos

```
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.properties.hibernate.globally_quoted_identifiers=true
```

Puerto donde escucha el servidor una vez se inicie

```
server.port=8080
```

Datos de conexión con la base de datos MySQL

```
spring.datasource.url=jdbc:mysql://localhost:3306/myapp  
spring.datasource.username=myappUser  
spring.datasource.password=myappPassword
```





Configurando el DataSource

Hay que tener en cuenta que la propiedad `spring.jpa.hibernate.ddl-auto` se utiliza para que la base de datos se genere automáticamente en cada arranque de la aplicación. Esto nos interesará cuando estemos en desarrollo pero no cuando queramos desplegarla en producción. Por lo que tendremos que tener cuidado y controlar el valor de dicha propiedad.

- **none**: Para indicar que no queremos que genere la base de datos.
- **update**: Si queremos que la genere de nuevo en cada arranque.
- **create**: Si queremos que cree la DB pero que no la genere de nuevo si ya existe.



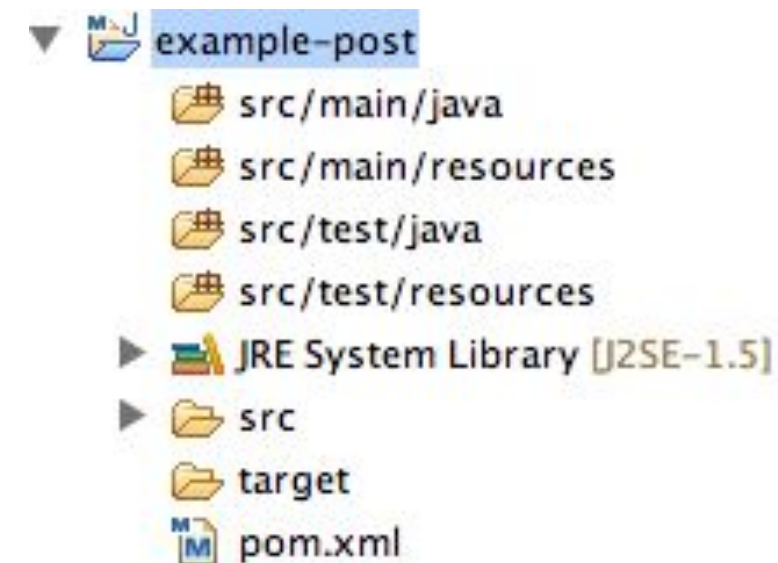


Ejecutando el proyecto

Estructura del proyecto

Un proyecto Spring Boot creado con Spring Initializr tendrá una estructura de directorios predefinida:

- **src/main/java**: Aquí se encuentra la ubicación principal para tus clases de Java.
- **src/main/resources**: Aquí se almacenan los recursos, como archivos de configuración.
- **src/main/webapp** (opcional): Si necesitas recursos web estáticos, como HTML, CSS o JavaScript, puedes ubicarlos aquí.
- **src/test**: Este directorio contiene las pruebas unitarias y de integración para tu aplicación.
- **pom.xml** (si usaste Maven): Estos archivos contienen la configuración del proyecto y las dependencias.



LIVE CODING

Ejemplo en vivo

Creando una vista Spring Boot JSP:

Vamos a poner en práctica lo aprendido! En este ejemplo vamos a desplegar una vista JSP desde nuestro proyecto Spring Boot.

1- Crea un controlador que maneje las solicitudes y proporcione el mensaje que se mostrará en la página JSP. Puedes hacerlo creando una clase, como se muestra a continuación:

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

@Controller
public class MessageController {

    @GetMapping("/mensaje")
    public String mostrarMensaje(Model model) {
        model.addAttribute("message", "¡Hola Mundo!");
        return "message";
    }
}
```


LIVE CODING

Ejemplo en vivo

*Crea un archivo JSP en el directorio :src/main/webapp/WEB-INF/views.
Crea un archivo llamado message.jsp con el siguiente contenido:*

```
<!DOCTYPE html>
<html>
<head>
  <title>Mi Aplicación Spring Boot JSP</title>
</head>
<body>
  <h1>Mensaje desde JSP</h1>
  <p>${message}</p>
</body>
</html>
```

Tiempo: 20 minutos

LIVE CODING

Ejemplo en vivo

Finalmente, ejecuta la aplicación y accede desde tu localhost:
Abre un navegador web y navega a `http://localhost:8080/mensaje` (o al puerto que hayas configurado). Deberías ver la página JSP con el mensaje "¡Hola Mundo!".

Tiempo: 20 minutos

○

¿Alguna consulta?

+



RESUMEN

¿Qué logramos en esta clase?

- ✓ Conocer la configuración de las tecnologías de vista
- ✓ Aprender a configurar un datasource



#WorkingTime

Continuemos ejercitando

¡Antes de cerrar la clase! Te invitamos a: 📌 📌 📌

1. Repasar nuevamente la grabación de esta clase
2. Revisar el material compartido en la plataforma de Moodle (lo que se vio en clase y algún ejercicio adicional)
 - a. *Lectura Modulo 6, Lección 2: páginas 1 - 10*
3. Traer al próximo encuentro, todas tus dudas y consultas para verlas antes de iniciar nuevo tema.

¡Muchas Gracias!

Nos vemos en la próxima clase 🙌

