

➤ 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 :

- ✓ Aplicar la capa de servicios en proyectos Spring
- ✓ Comprender la inyección de servicios en los controladores

LEARNING PATHWAY

¿Sobre qué temas trabajaremos?

6.2

Start! 🏁

El Framework Spring
MVC

Test y empaquetamiento
Creando unidades de prueba con Spring
Empaquetando una aplicación Spring
MVC en un archivo WAR

Test y Empaquetamiento

Unidades de prueba

Empaquetando

OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?



Comprender el proceso de test y empaquetamiento en Spring



Aprender a crear unidades de prueba

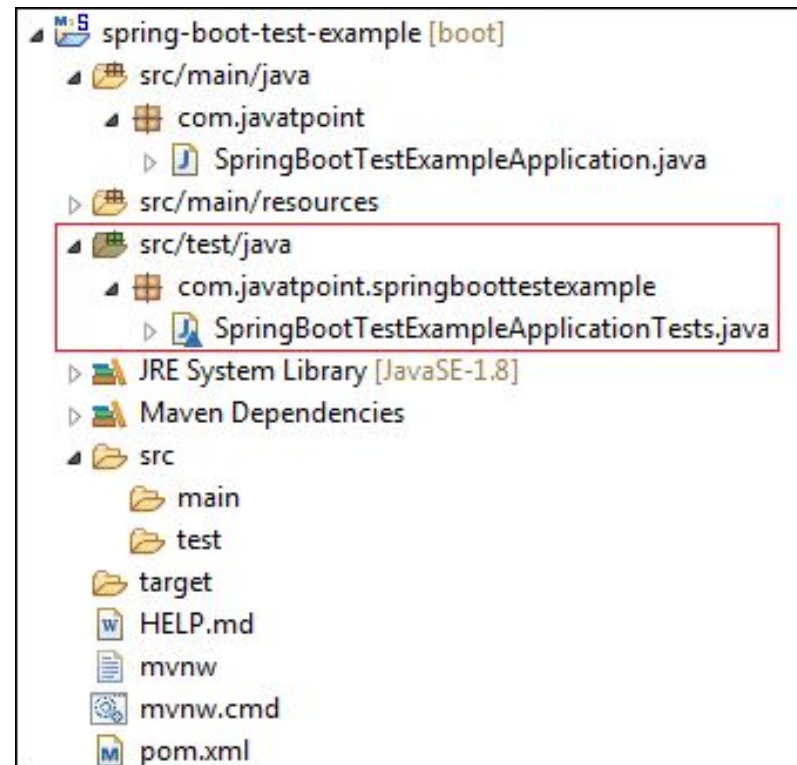


› Test y empaquetamiento



Test y empaquetamiento

Las pruebas y el empaquetamiento son dos aspectos esenciales en el desarrollo de aplicaciones Spring. Las pruebas te permiten asegurarte de que tus componentes funcionen correctamente, mientras que el empaquetamiento es crucial para la distribución y el despliegue de tu aplicación.



Test

Pruebas en Spring:

Las pruebas en Spring son fundamentales para garantizar que tus componentes, clases y métodos funcionen de acuerdo a lo previsto. Existen diferentes tipos de pruebas que puedes realizar en un entorno Spring:

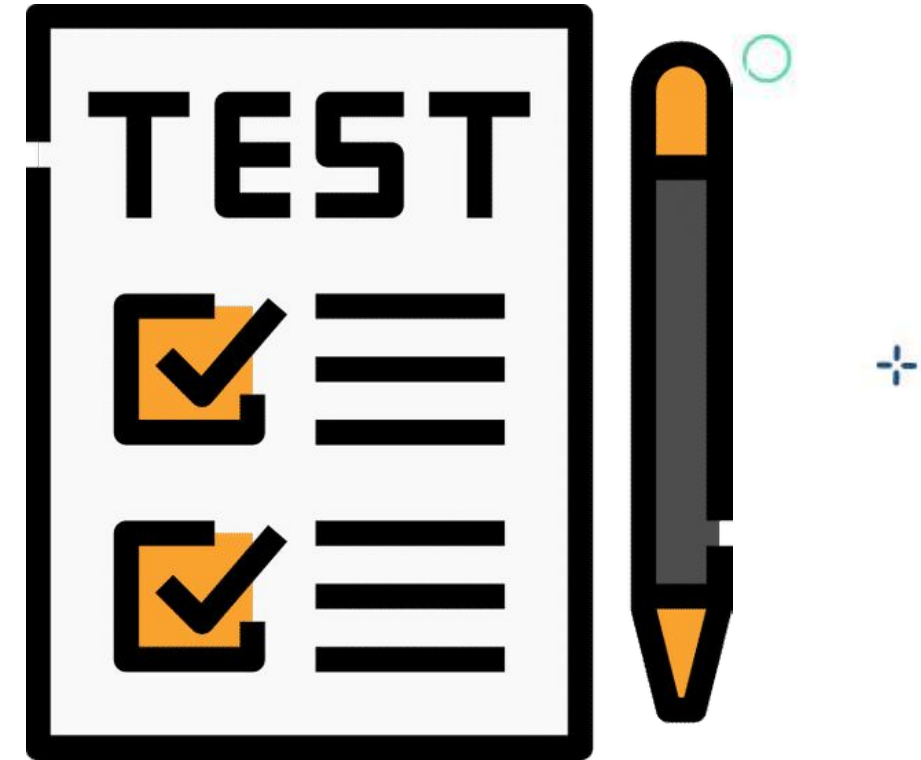


Test

1. Pruebas Unitarias:

Las pruebas unitarias son pruebas que se centran en unidades individuales de código, como clases o métodos. En Spring, puedes realizar pruebas unitarias utilizando frameworks de pruebas como **JUnit** o **TestNG**. A menudo, estas pruebas se realizan en aislamiento, lo que significa que no se interactúa con componentes externos, como bases de datos o servicios web.

Spring brinda anotaciones como **@RunWith(SpringRunner.class)** y **@SpringBootTest** para facilitar la configuración del contexto de Spring en las pruebas unitarias y la inyección de dependencias.



Test

2. Pruebas de Integración:

Las pruebas de integración se utilizan para verificar la interacción entre componentes dentro de tu aplicación. En el contexto de Spring, esto podría implicar la interacción entre componentes gestionados por el contenedor de Spring, como controladores, servicios y repositorios.

Spring Test ofrece otras anotaciones útiles, como `@MockBean` para simular componentes dependientes y `@AutoConfigureMockMvc` para probar controladores web.



Test

3- Pruebas de extremo a extremo:

También puedes realizar pruebas de extremo a extremo para verificar el comportamiento de la aplicación en un entorno similar al de producción. En Spring, puedes utilizar herramientas como [RestTemplate](#) o [WebTestClient](#) para realizar solicitudes [HTTP](#) y validar respuestas en pruebas de extremo a extremo.





Empaquetamiento

El empaquetamiento se refiere a **la forma en que se construye y distribuye una aplicación Spring en un archivo ejecutable**, como un archivo **JAR** o un archivo **WAR**. Spring ofrece diferentes opciones de empaquetamiento para adaptarse a los diferentes requisitos de implementación.





Empaquetamiento



1- Archivo JAR ejecutable:

Puedes empaquetar una aplicación Spring en un archivo **JAR** ejecutable que incluya todas las dependencias necesarias y que pueda ser ejecutado directamente mediante el comando **java -jar**. Para ello, se utiliza el plugin **Maven** correspondiente para generar el **JAR** ejecutable.





Empaquetamiento



2- Archivo WAR:

Si estás desarrollando una aplicación web basada en Spring, puedes empaquetarla en un archivo **WAR** (Web Application Archive). Este archivo contiene la aplicación web, junto con las dependencias y los recursos necesarios para ser desplegados en un servidor de aplicaciones compatible con **Java EE**.





Empaquetamiento



2- Empaquetamiento personalizado:

En algunos casos, es posible que necesites un empaquetamiento más personalizado para cumplir con requisitos específicos. Spring **ofrece flexibilidad** para personalizar el proceso de empaquetamiento según tus necesidades, utilizando herramientas como el plugin **Maven Shade** o el plugin **Gradle Shadow**.



➤ Creando unidades de prueba con Spring



Creando unidades de prueba

Crear unidades de prueba efectivas en Spring es esencial para garantizar que tu aplicación funcione correctamente y se comporte según lo previsto. Spring ofrece una serie de herramientas y enfoques que facilitan la creación de pruebas unitarias para tus componentes.





Creando unidades de prueba



1. Herramientas de Pruebas de Spring:

Spring TestContext Framework: Esta parte de Spring Testing proporciona soporte para cargar y gestionar el contexto de Spring en las pruebas. Te permite ejecutar pruebas en un contexto de Spring configurado, lo que es esencial para la inyección de dependencias y la ejecución de tus componentes en un entorno controlado.



SpringJUnit4ClassRunner: Este corredor de **JUnit** se utiliza para ejecutar pruebas con soporte de Spring. Facilita la carga del contexto de Spring y la inyección de dependencias en las pruebas.

@RunWith(SpringJUnit4ClassRunner): Anotar tus clases de prueba con esta anotación permite que **SpringJUnit4ClassRunner** ejecute las pruebas, lo que facilita la integración de Spring en tus pruebas unitarias.



Creando unidades de prueba

2. Configuración del Contexto de Prueba:



Un aspecto fundamental de la creación de unidades de prueba en Spring es la configuración del contexto de prueba. Puedes definir un contexto específico para tus pruebas, que puede incluir configuraciones y beans personalizados para simular el entorno de producción. Esto se logra con la anotación **@ContextConfiguration**.



```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = {AppConfig.class})
public class MiClaseDePrueba {
    // ...
}
```

En este ejemplo, AppConfig es la configuración de Spring que se utilizará para cargar el contexto de la prueba. Esto te permite definir un contexto específico para tus pruebas, configurando solo los componentes necesarios para cada caso de prueba.



Creando unidades de prueba



3. Inyección de Dependencias:

Spring se encarga de inyectar las dependencias necesarias en tus componentes de prueba, lo que te permite probar unidades individuales de código de manera aislada.



```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = {AppConfig.class})
public class MiClaseDePrueba {
    @Autowired
    private Servicio servicio; // Spring inyectará automáticamente
    // ...
}
```



La anotación **@Autowired** se utiliza para inyectar automáticamente la dependencia Servicio en la clase de prueba. Esto permite que tus pruebas interactúen con componentes reales de tu aplicación.



Creando unidades de prueba



4. Anotaciones de Prueba: Es común utilizar anotaciones específicas para las pruebas. Algunas de las más utilizadas incluyen:

- **@Test:** Anota un método como una prueba. JUnit utiliza esta anotación para identificar los métodos que deben ejecutarse como pruebas.
- **@Before:** Anota un método que se ejecutará antes de cada prueba. Esto es útil para inicializar el estado necesario para las pruebas.
- **@After:** Anota un método que se ejecutará después de cada prueba. Puedes utilizarlo para liberar recursos o limpiar el estado después de las pruebas.
- **@RunWith:** Como se mencionó anteriormente, esta anotación se utiliza para especificar el corredor de pruebas que debe utilizarse para ejecutar las pruebas.





Creando unidades de prueba

5. Escribir Pruebas Efectivas:

- Cada prueba debe centrarse en un aspecto específico del comportamiento de tu componente.
- Asegúrate de que tus pruebas cubran todos los caminos posibles a través de tu código, incluyendo casos de éxito y de error.
- Proporciona datos de prueba que sean relevantes para la funcionalidad que estás probando. Esto facilitará la identificación de problemas cuando las pruebas fallen.
- Las pruebas deben ejecutarse rápidamente y no deben depender de otros componentes o recursos externos.
- A medida que tu código evoluciona, asegúrate de mantener y actualizar tus pruebas para que sigan siendo efectivas.



Evaluación Integradora ✨

¿Listos para un nuevo desafío? En esta clase comenzamos a construir nuestro...

Trabajo Integrador del Módulo 💪

Iremos completándolo progresivamente clase a clase.



LIVE CODING

Ejemplo en vivo

Wallet Service:

- *Vamos a crear una clase Test en el paquete src/test/java, donde probaremos las funcionalidades de la clase 'UsuarioServicio' (método 'crearUsuario')*

  **Tiempo: 20 minutos**



Momento: ✚

Time-out!

🕒 5 min.



➤ Empaquetando una aplicacion Spring MVC en un archivo WAR

Empaquetando una aplicacion Spring MVC en un archivo WAR

Empaquetar una aplicación Spring MVC en un archivo **WAR** (Web Application Archive) es un paso fundamental en el desarrollo de aplicaciones web en Java. El formato WAR se utiliza para distribuir y desplegar aplicaciones web en servidores de aplicaciones compatibles, como **Apache Tomcat**, Jetty, o servidores de aplicaciones empresariales.






Empaquetando una aplicacion Spring MVC en un archivo WAR



Paso 1: Configuración del Proyecto Spring MVC:



Antes de empaquetar tu aplicación en un archivo WAR, asegúrate de que ya tienes una aplicación Spring MVC en funcionamiento. Esto implica la configuración de controladores, vistas, modelos y otros componentes necesarios. Si aún no has desarrollado tu aplicación Spring MVC, este es el primer paso.



Empaquetando una aplicación Spring MVC en un archivo WAR

Paso 2: Dependencias de Maven:

Para empaquetar tu aplicación como un archivo WAR, necesitas agregar las dependencias apropiadas en tu archivo pom.xml. Asegúrate de que tengas las siguientes dependencias de Spring Boot:

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
    <scope>provided</scope>
  </dependency>
</dependencies>
```

Empaquetando una aplicación Spring MVC en un archivo WAR

Paso 2: Dependencias de Maven:

La dependencia **spring-boot-starter-web** proporciona las funcionalidades necesarias para una aplicación web Spring, mientras que la dependencia **spring-boot-starter-tomcat** es necesaria, pero con el alcance provided, para indicar que Tomcat (u otro servidor) proporcionará la implementación de servidor incorporada.

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
    <scope>provided</scope>
  </dependency>
</dependencies>
```

Empaquetando una aplicación Spring MVC en un archivo WAR

Paso 3: Anotación @SpringBootApplication:

Asegúrate de que tu clase principal de la aplicación Spring MVC esté anotada con `@SpringBootApplication`. Esta anotación combina varias anotaciones importantes, como `@Configuration`, `@EnableAutoConfiguration`, y `@ComponentScan`, que son fundamentales para la configuración automática de Spring Boot.

```
@SpringBootApplication
public class MiAplicacionSpringMvcApplication {

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


Empaquetando una aplicación Spring MVC en un archivo WAR

Paso 4: Configuración del Despliegue:

Spring Boot facilita la configuración del proceso de empaquetamiento en un archivo WAR. Debes asegurarte de que tu archivo **pom.xml** esté configurado de la siguiente manera:

La configuración del plugin **spring-boot-maven-plugin** es esencial para que Spring Boot reempaquete tu aplicación como un archivo WAR cuando ejecutes el comando de empaquetamiento.

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <version>2.7.0</version> <!-- Reemplaza con la versión actual -->
      <executions>
        <execution>
          <goals>
            <goal>repackage</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```


Empaquetando una aplicación Spring MVC en un archivo WAR

Paso 5: Cambio del Tipo de Empaquetamiento:

Para empaquetar tu aplicación como un archivo WAR, debes especificar el tipo de empaquetamiento en tu archivo **pom.xml**. Asegúrate de que el tipo de empaquetamiento sea "**war**". Esto le indica a Maven que debe generar un archivo WAR en lugar del JAR predeterminado.

```
<packaging>war</packaging>
```

Empaquetando una aplicacion Spring MVC en un archivo WAR

Paso 6: Empaquetamiento:

Una vez que hayas realizado todos los pasos anteriores, estás listo para empaquetar tu aplicación Spring MVC en un archivo WAR. Utiliza el siguiente comando Maven para empaquetar tu aplicación. Este comando generará un archivo WAR en la carpeta target de tu proyecto.



```
mvn clean package
```

LIVE CODING

Ejemplo en vivo

Wallet WAR:

- *Es hora de empaquetar nuestra aplicación en un archivo WAR.*

  **Tiempo: 15 minutos**

○

¿Alguna consulta?

+



RESUMEN

¿Qué logramos en esta clase?

- ✓ **Aprender la implementación de pruebas y empaquetamiento**
- ✓ **Comprender el empaquetado de una aplicación Spring en un archivo WAR**



#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 33 - 38*
3. Traer al próximo encuentro, todas tus dudas y consultas para verlas antes de iniciar nuevo tema.

Momento: ✚

Time-out!

🕒 5 min.

