M6_AE1_ABPRO-Ejercicio grupal[Actividad Evaluada]

Nombres: Laura Duhalde
Andrea Correa
Fecha: 20 de octubre de 2025.

Paso 2: Simulación del flujo de desarrollo con Maven

Comando	Fase del Ciclo de Vida	Objetivo Principal	¿Qué Genera o Hace?
mvn clean	Limpiar	Limpia	Elimina la carpeta de destino (target) generada por compilaciones anteriores. Asegura una construcción limpia. Limpia archivos compilados y artefactos previos
mvn compile	Por defecto, etapa que debe realizar.	Proceso de compilar	Compila el código fuente del proyecto (archivos .java) y genera los archivos de clase (.class) en la carpeta target/classes. Espera que se realice sin errores.
mvn test	Por defecto	surefire:test	Ejecuta las pruebas unitarias del proyecto. Genera informes de prueba con el reporte en target/surefire-reports.
mvn package	Por defecto	jar:package o war:package	Toma las clases compiladas y el código compilado y los empaqueta en un formato distribuible (generalmente un archivo JAR o WAR) en la carpeta target. Quedando tarjeta/*jar
mvn install	Por defecto	install:install	Instala el paquete (JAR/WAR) del proyecto en el repositorio local de Maven (~/.m2/repository). Esto permite que otros proyectos locales dependan de este artefacto.

Capturas:

mvn clean

```
Laura@DESKTOP-LAURA MINGW64 /d/Curso Java TD/Proyectos/Modulo6/JAVA0078_M6_FormularioSpringBoot
$ mvn clean
[INFO] Scanning for projects...
[INFO]
[INFO] --------------------< cl.web:FormularioSpringBoot >---------------------
[INFO] Building JAVA0078_M6_FormularioSpringBoot 1.0.0
[INFO]   from pom.xml
[INFO] --------------------------------[ war ]---------------------------------
[INFO]
[INFO] --- clean:3.4.1:clean (default-clean) @ FormularioSpringBoot ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.634 s
[INFO] Finished at: 2025-10-20T20:21:20-03:00
[INFO] ------------------------------------------------------------------------
```

mvn compile

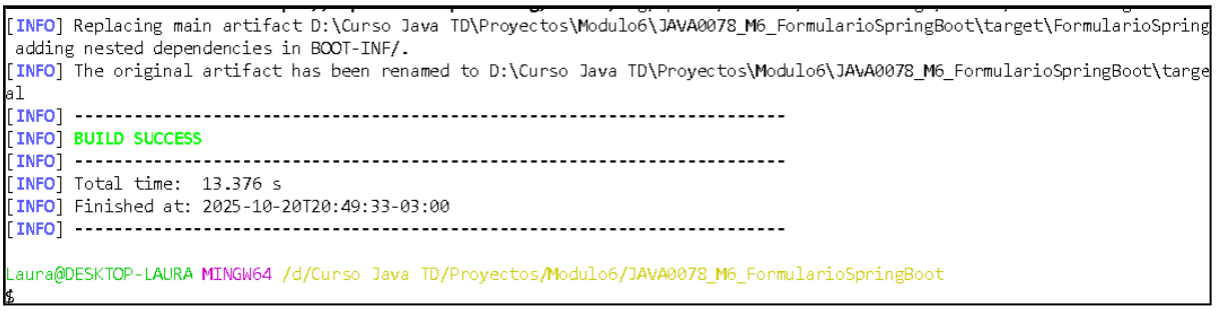
```
Laura@DESKTOP-LAURA MINGW64 /d/Curso Java TD/Proyectos/Modulo6/JAVA0078_M6_FormularioSpringBoot
$ mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] --------------------< cl.web:FormularioSpringBoot >---------------------
[INFO] Building JAVA0078_M6_FormularioSpringBoot 1.0.0
[INFO]   from pom.xml
[INFO] --------------------------------[ war ]---------------------------------
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ FormularioSpringBoot ---
[INFO] Copying 1 resource from src\main\resources to target\classes
[INFO] Copying 0 resource from src\main\resources to target\classes
[INFO]
[INFO] --- compiler:3.14.0:compile (default-compile) @ FormularioSpringBoot ---
[INFO] Recompiling the module because of changed source code.
[INFO] Compiling 3 source files with javac [debug parameters release 21] to target\classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.700 s
[INFO] Finished at: 2025-10-20T20:21:54-03:00
[INFO] ------------------------------------------------------------------------
```

mvn test

```
ava HotSpot(TM) 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 3.530 s -- in cl.web.Java0078M6FormularioSpringBootApplicationTests
INFO]
INFO] Results:
INFO]
INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
INFO]
INFO] ------------------------------------------------------------------------
INFO] BUILD SUCCESS
INFO] ------------------------------------------------------------------------
INFO] Total time:  8.156 s
INFO] Finished at: 2025-10-20T20:48:47-03:00
INFO] ------------------------------------------------------------------------

aura@DESKTOP-LAURA MINGW64 /d/Curso Java TD/Proyectos/Modulo6/JAVA0078_M6_FormularioSpringBoot
```

mvn package



mvn install

Paso 4: Manejo de dependencias, repositorios y estructura

- Expliquen cómo se descargaron las dependencias.

  Maven lee `pom.xml` y descarga los artefactos necesarios desde repositorios remotos (por defecto Maven Central) la primera vez que los necesita. Descargas se colocan en tu repositorio local.

- Localicen el **repositorio local de Maven** y detallen qué tipo de archivos contiene.

  El repositorio local es un caché donde Maven almacena todos los artefactos (librerías, JARs, WARs) que descarga de repositorios remotos (como Maven Central) o que tú mismo instalas usando `mvn install`.

  Dependencias en el archivo pom.xml

```xml
<!-- Soporte para JSP -->
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
</dependency>

<!-- JSTL (necesario para JSP) -->
<dependency>
    <groupId>jakarta.servlet.jsp.jstl</groupId>
    <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
</dependency>
<dependency>
    <groupId>org.glassfish.web</groupId>
    <artifactId>jakarta.servlet.jsp.jstl</artifactId>
</dependency>
```

El repositorio local está estructurado por carpetas que reflejan las coordenadas de los artefactos (librerías) declarados en el archivo pom.xml.

Contiene dos tipos principales de archivos:

A. Artefactos Binarios (Librerías)

Estos son los archivos que contienen el código compilado de las dependencias que tu proyecto necesita o el código compilado de tu propio proyecto si ejecutas mvn install.

- **.jar** (Java ARchive): El formato más común para librerías.
- **.war** (Web Application Archive): Para aplicaciones web.
- **.pom** (Project Object Model): Una copia del archivo de configuración de la librería que detalla sus propias dependencias (transitivistas).
- **.zip** / **.tar.gz**: Raramente usados para el código principal, a veces para fuentes o *javadoc*.

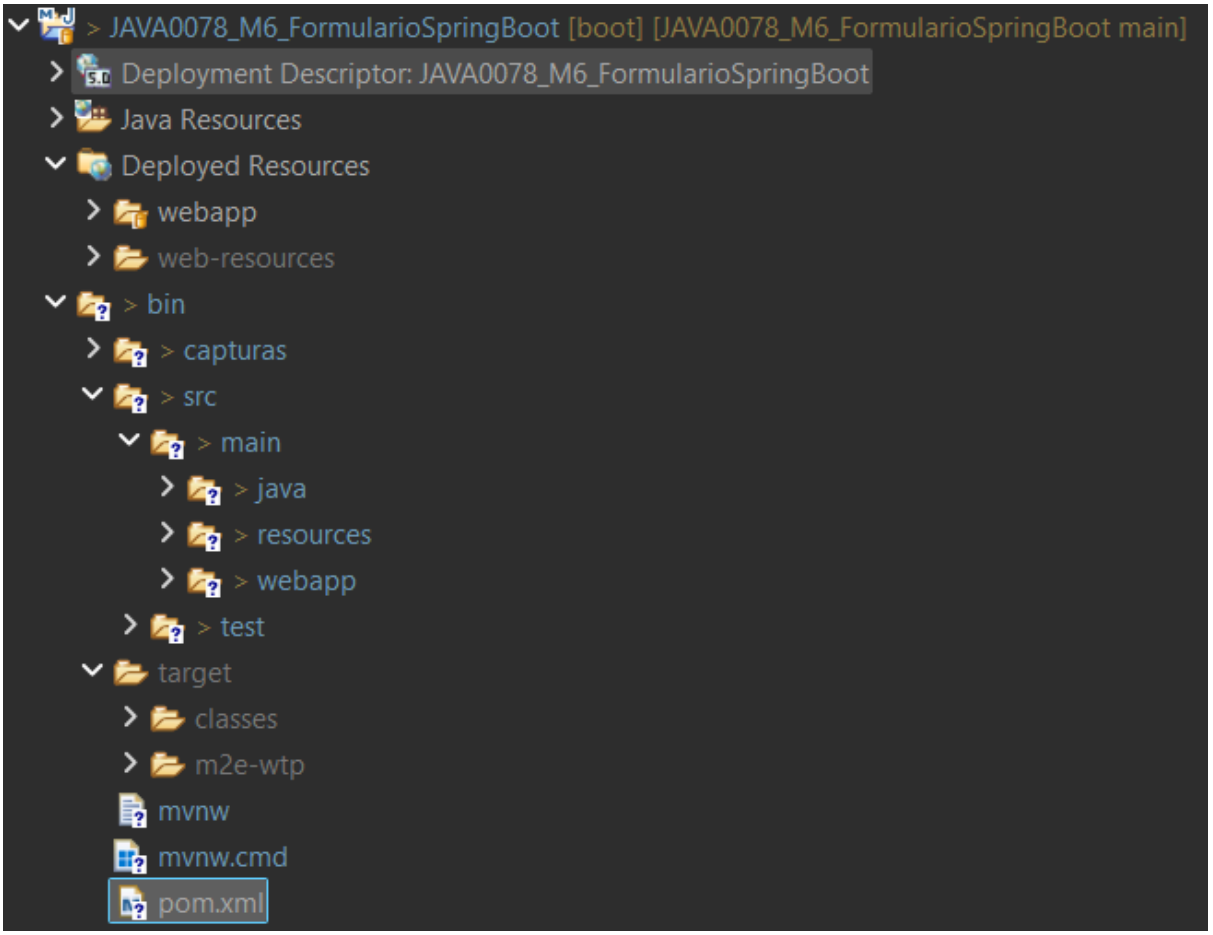
B. Archivos de Metadatos de Maven

Estos archivos son utilizados por Maven para gestionar el repositorio.

- **Archivos de *Checksum*:**
    - **.sha1** o **.md5**: Archivos pequeños que contienen sumas de verificación (checksums) para garantizar que el artefacto binario (.jar, .pom) se descargó correctamente y no fue alterado.
- **Archivos de Metadatos:**
    - **maven-metadata-local.xml**: Contiene información sobre las versiones disponibles de un artefacto en la ubicación local.

- Muestren y expliquen la estructura del proyecto generada por Maven (src/main/java, src/main/resources, etc.).

  Captura de la estructura que genera Maven



pom.xml

Es **el corazón del proyecto Maven**.

Contiene:

- Metadatos: groupId, artifactId, version
- Dependencias (bibliotecas externas)
- Plugins de compilación o empaquetado
- Configuración de compilador (versión Java, etc.)
- Información de empaquetado (jar, war, etc.)

Maven usa este archivo para:

1. Descargar dependencias desde repositorios remotos.
2. Saber cómo compilar y empaquetar el proyecto.

3. Ejecutar las fases del ciclo de vida (compile, test, package, install, etc.).

1. src/ — Código fuente

Contiene **todo el código y recursos** del proyecto, dividido por propósito.

src/main/java/

Aquí va **el código fuente principal** de tu aplicación Java.

- ProyectoBaseApplication.java: clase principal con main(), punto de entrada del programa.
- controller/: controladores web (manejan peticiones HTTP, rutas, etc.).
- dto/: objetos de transferencia de datos (reciben datos del formulario, se validan).

Todo lo que el usuario final usa en ejecución está aquí.

2. src/main/resources/

Aquí van **recursos no Java**, como configuraciones o plantillas.

En tu proyecto:

- templates/: vistas Thymeleaf (HTML dinámico renderizado por Spring Boot).
- application.properties: configuración del proyecto (puerto, base de datos, etc.).

Maven los empaqueta dentro del jar junto con las clases compiladas.

3. src/test/java/

Aquí van **los tests unitarios y de integración**.

- Contiene pruebas automáticas (JUnit, Spring Boot Test).
- Maven las ejecuta en la fase test.

No se incluyen en el jar final, solo se usan durante el desarrollo.

src/test/resources/ ***(opcional)***

Contiene recursos que se usan solo en las pruebas, por ejemplo:

- Archivos de configuración de test
- Datos de prueba (.json, .sql, etc.)

target/ — Directorio de salida (generado automáticamente)

Maven **crea esta carpeta** cada vez que compilas o empaquetas.

target/ es el **resultado final** del proceso de construcción.

Se puede borrar sin problema, ya que se regenera con mvn clean package.

Repositorio local Maven (~/.m2/repository)

Cuando Maven necesita una dependencia (por ejemplo, spring-boot-starter-web), la descarga desde Internet (Maven Central) y la guarda en:

C:\Users\<usuario>\.m2\repository\

Allí se almacenan todos los .jar y .pom que tu proyecto requiere.

Así, si otro proyecto usa las mismas librerías, no las vuelve a descargar.

Resumen

Carpeta / Archivo	Propósito
pom.xml	Configuración del proyecto y dependencias
src/main/java	Código fuente principal
src/main/resources	Archivos de configuración y recursos
src/test/java	Código de pruebas
src/test/resources	Recursos usados solo en tests
target/	Resultado de la compilación y empaquetado
~/.m2/repository	Repositorio local de dependencias descargadas

Documenten el proceso completo de configuración, decisiones tomadas, errores encontrados y soluciones.

Error encontrado:

Al enviar el formulario no se encontraba el servlet, por lo que daba error 404.

Se solucionó al poner la anotación @ServletComponentScan en Java0078M6FormularioSpringBootApplication,java

@ServletComponentScan hace que Spring Boot busque servlets, filtros y listeners anotados en el paquete base y sus subpaquetes.

Esto significa que tu aplicación Spring Boot no estaba encontrando ciertos componentes web (como Servlets, Filters o Listeners) porque estaban definidos usando anotaciones de la especificación Servlet (como @WebServlet, @WebFilter, etc.) y no anotaciones propias de Spring.

La solución fue agregar la anotación **@ServletComponentScan** a tu clase principal de la aplicación.

Explicación de @ServletComponentScan

1. ¿Qué es @ServletComponentScan?

Es una anotación de Spring Boot que le dice al contenedor de Spring que **busque y registre automáticamente** componentes de la **especificación Servlet** (JSP/Jakarta Servlet).

- **Busca:** Componentes anotados con:
    - **@WebServlet** (para Servlets)
    - **@WebFilter** (para Filtros)
    - **@WebListener** (para Listeners de eventos)
- **Alcance:** La búsqueda se realiza en el **paquete base** de la clase donde se coloca la anotación (en este caso, Java0078M6FormularioSpringBootApplication.java) y en **todos sus subpaquetes**.

2. ¿Por Qué Fue Necesario? (El Problema)

Spring Boot, por defecto, está diseñado para buscar componentes de Spring (anotados con @Component, @Controller, @Service, etc.) dentro de su "paquete base" usando la anotación implícita @SpringBootApplication (que incluye @ComponentScan).

Sin embargo, **no busca automáticamente** componentes que usan las anotaciones nativas de la especificación Servlet (@WebServlet, @WebFilter).

- **Tu Situación:** Habías definido un Servlet, Filtro o Listener usando una de esas anotaciones de especificación Servlet (probablemente en un subpaquete).

- **Resultado:** El contenedor de Servlets incrustado (como Tomcat) **no se enteró** de la existencia de ese componente, y por lo tanto, no se podía acceder a él (por ejemplo, el Servlet no respondía a la URL esperada).

3. La Solución

Al añadir **@ServletComponentScan**, le indicaste a Spring Boot que **expanda** su proceso de escaneo para incluir también esas anotaciones de Servlets, asegurando que todos los componentes web del paquete sean encontrados e inicializados correctamente por el servidor.