

MP_0490
Programación de
servicios y procesos

TEMA 3: Servicios web REST

¿Qué son los metadatos?

Las anotaciones Java proveen de un sistema para añadir metadatos al código fuente, que estarán disponibles para la aplicación en tiempo de ejecución.

La traducción literal de metadato es *sobre dato*, y hace referencia a **un dato que describe a otro dato**.

En aplicaciones empresariales, se utiliza para añadir información de configuración a un objeto cuya misión es servir como recurso a un proyecto

Un **ejemplo de anotación**, que con seguridad te has encontrado en más de una ocasión, es la anotación `@Override` delante de un método, para indicar que dicho método viene heredado de una clase base y está siendo sobrescrito.

Ejercicio

Vamos a crear y a utilizar tu propia anotación personalizada.

1. Crearemos la anotación personalizada *@Autor*, que servirá para añadir información a las clases y métodos que queramos sobre quién es el autor que desarrolló dichas clases o métodos. La anotación contará con las propiedades *nombre* y *direccion*.
2. Crearemos una clase llamada *Coche* y la anotaremos. Dentro de la clase *Coche* implementaremos un método llamado *acelerar()*, que también anotaremos.
3. En la clase *Principal* crearemos un objeto de la clase *Coche* y accederemos a los datos suministrados por la anotación *@Autor*.

```
package prueba;  
  
public @interface Autor {  
  
}
```

Las anotaciones son como interfaces especiales. Observa que la palabra *interface* va precedida por el símbolo @.

Clase (class)

- Es un **molde para crear objetos**.
- Puede contener **atributos (variables)** y **métodos (funciones)**.
- Se puede instanciar usando new.
- Tiene comportamiento y lógica propia.

Anotación (@interface)

- Es una **meta-información**, NO un objeto.
- No tiene comportamiento ni lógica propia.
- Se usa para marcar clases, métodos o variables con **información extra**.
- Se lee mediante **reflexión** (como hace Spring o JPA).
- No se puede instanciar con new.

Resumen de diferencias

Característica	Clase (<code>class</code>)	Anotación (<code>@interface</code>)
Se instancia con <code>new</code>	✓ Sí	✗ No
Tiene atributos y métodos	✓ Sí	✗ Solo tiene valores de configuración
Puede contener lógica	✓ Sí	✗ No
Sirve como metadato para el código	✗ No	✓ Sí
Se usa con reflexión para leer valores	✗ No necesariamente	✓ Sí

Las anotaciones son útiles cuando necesitas proporcionar información adicional a tu código sin modificar su comportamiento directamente. Se usan mucho en frameworks como **Spring**, **JPA**, **JUnit**, **Lombok**, etc.

Introducción a REST

Un servicio web es un sistema software definido para permitir la comunicación a través de la red entre dos máquinas.

La comunicación es posible mediante el intercambio de mensajes, utilizando, por lo general, **el protocolo HTTP**.

Importantes sitios web, como Facebook, Twitter o Instagram, son aplicaciones que hacen uso de servicios web (*web services*).

Un *web service* es, como hemos mencionado, un conjunto de estándares y protocolos utilizados para intercambiar datos entre aplicaciones.

- **SOAP (*Simple Object Access Protocol*)**: protocolo que utiliza el lenguaje XML para definir el lenguaje utilizado entre dos procesos para comunicarse. SOAP emplea XML para definir un conjunto de tipos de datos, y configurar los mensajes de solicitud y de respuesta.
- **REST (*Representational State Transfer*)**: utiliza el protocolo HTTP para la comunicación entre máquinas. Puesto que utiliza el protocolo HTTP, que se caracteriza por no tener estado, los servicios REST corresponden al tipo de servicios *stateless* (sin estado); es decir, que en cada solicitud que llega de un cliente, el servidor no tiene información de las solicitudes anteriores del mismo cliente.

Principios en los que se basa REST

Utiliza el protocolo **HTTP** para establecer comunicaciones cliente/servidor.

El cliente solicita recursos al servidor a través de una **URI** (*Uniform Resource Identifier*) que sirve para identificar inequívocamente un recurso dentro de la red.

Los datos se transmiten con un formato específico identificado mediante la nomenclatura **MIME** (image/jpeg, text/html, text/xml, video/mpeg, etc).

Más adelante hablaremos más en profundidad de estos tipos MIME.

REST utiliza las acciones estándar definidas por el protocolo HTTP: **GET, POST, PUT, DELETE**, etc.

El servidor **expone recursos** que podrán ser devueltos en distintos formatos como **texto, HTML, XML, JSON**, etc.

Para cada solicitud que realiza un cliente se obtiene un **código de respuesta**, una característica propia del protocolo HTTP. Entre estas respuestas, el código 404 (recurso no encontrado).

Cuando hablamos de REST hablamos de un tipo de arquitectura de software para el desarrollo de servicios web. RESTful, sin embargo, es algo más concreto. Denominamos así a los servicios web que siguen los principios que rigen la arquitectura REST.

VENTAJAS

Separación de los datos del recurso y su representación. El mismo recurso puede enviarse representado en formato XML, JSON o texto plano.

Visibilidad: se accede a cada recurso a través de una URI, de manera muy simple.

Seguridad: el acceso a un recurso nunca cambiará su estado.

Escalabilidad: los servicios web REST son fácilmente escalables, lo que significa que pueden crecer a lo largo del tiempo sin que suponga un coste muy alto.

Rendimiento: el rendimiento hace referencia al tiempo medio de demora en responder a una petición de un cliente. Los servicios REST, precisamente por utilizar un protocolo sin estado, son rápidos.

Cuando escribes una dirección web en tu navegador realizas una petición a un servidor web que localizará ese recurso (en este caso, una página web) y te lo devolverá (en formato HTML) para que tu navegador pueda interpretarlo y mostrarlo.

Por ejemplo: si escribimos en el navegador <https://www.casadellibro.com/> estamos realizando una petición a un servidor web, que localiza el recurso (que en nuestro ejemplo es una página web) que estamos solicitando, y nos lo devuelve (en este caso, en formato HTML) para que podamos verlo en el navegador.

Este es el funcionamiento básico del protocolo HTTP y el principio por el que se rigen también los servicios web REST. El sitio web de la Casa del Libro estará localizado en un servidor web instalado en un equipo de cualquier parte del mundo, y miles de clientes pueden acceder a su contenido.

Tipos MIME

Los tipos MIME son unos códigos que utilizamos durante la implementación de los servicios web para indicarle al cliente cuál es el formato del mensaje que le hemos enviado.

- audio/mp4
- audio/mpeg
- image/gif
- image/bmp
- image/png
- text/css
- text/plain
- text/rtf
- text/xml
- text/json
- video/avi
- video/flv
- video/mpeg
- application/vnd.ms-excel

Apache HTTP Server (o simplemente Apache) es un servidor web tradicional, diseñado para servir contenido estático (HTML, imágenes, CSS, etc.) y dinámico mediante módulos como PHP o integraciones con aplicaciones externas.

Apache Tomcat es un servidor de aplicaciones especializado en la ejecución de aplicaciones Java basadas en servlets, JSP y otras tecnologías relacionadas con Java EE. No está diseñado para servir contenido estático de forma eficiente.

Los proyectos web dinámicos tienen las siguientes características:

1. Son proyectos que se crean en la **perspectiva Java EE**
2. Tienen el objetivo de ser posteriormente desplegados y ejecutados en un **servidor web, o servidor de aplicaciones**. En nuestro caso, se tratará de un proyecto de servicios web que será desplegado en el servidor Apache Tomcat.
3. Puesto que el objeto principal de un proyecto web dinámico es su despliegue en un servidor web, será necesario configurar la forma en que se montará dicho despliegue. Nos referimos a la configuración del llamado *Web Deployment Assembly*.
4. Siguen la **arquitectura cliente/servidor**.



Apache Tomcat®

Apache Tomcat

[Home](#)
[Taglibs](#)
[Maven Plugin](#)

Download

[Which version?](#)
[Tomcat 11](#)
[Tomcat 10](#)
[Tomcat 9](#)
[Tomcat Migration Tool for Jakarta EE](#)
[Tomcat Connectors](#)
[Tomcat Native](#)
[Taglibs](#)
[Archives](#)

Documentation

Apache Tomcat

The Apache Tomcat® software is an open source implementation of the [Jakarta Servlet](#), [Jakarta Pages](#), [Jakarta Expression Language](#), [Jakarta WebSocket](#), [Jakarta Annotations](#) and [Jakarta Authentication](#) specifications. These specifications are part of the [Jakarta EE platform](#).

The Jakarta EE platform is the evolution of the Java EE platform. Tomcat 10 and later implement specifications developed as part of Jakarta EE. Tomcat 9 and earlier implement specifications developed as part of Java EE.

The Apache Tomcat software is developed in an open and participatory environment and released under the [Apache License version 2](#). The Apache Tomcat project is intended to be a collaboration of the best-of-breed developers from around the world. We invite you to participate in this open development project. To learn more about getting involved, [click here](#).

Apache Tomcat software powers numerous large-scale, mission-critical web applications across a diverse range of industries and organizations. Some of these users and their stories are listed on the [PoweredBy](#) wiki page.

Apache Tomcat, Tomcat, Apache, the Apache feather, and the Apache Tomcat project logo are trademarks of the Apache Software Foundation.

Tomcat Migration Tool for Jakarta EE 1.0.9 Released

2025-01-21

The Apache Tomcat Project is proud to announce the release of 1.0.9 of the Apache Tomcat Migration Tool for Jakarta EE. This release contains a number of bug fixes and improvements compared to version 1.0.8.

The notable changes in this release are:

- Fix an issue that `matchExcludesAgainstPathName` didn't work for files. Based on a pull request by Semiao Marco.
- Added a new profile, `SERVLET` that only migrates the `javax.servlet` package and sub-packages. Provided by Ralf Wiebicke.
- Update dependencies


<https://tomcat.apache.org/>

Donde se nos solicita la ubicación de la instalación de Java: Apache Tomcat requiere que esté instalado Java. En la gran mayoría de los casos el instalador detecta la ubicación de la instalación de Java sin mayor problema, pero en algunas ocasiones puede ser necesario especificarla.

Donde aparecen los valores de configuración del servidor: presta atención al valor de "HTTP/1.1 Connector Port", que será 8080. Recuerda que una petición a un servidor se realiza a través del nombre del servidor y un puerto. Pues bien, el nombre del servidor, al tratarse de nuestro propio equipo, será *localhost* y el puerto el 8080, que es el valor que aparece por defecto.

Apache Tomcat Setup: Configuration Options

Configuration
Tomcat basic configuration.



Server Shutdown Port

HTTP/1.1 Connector Port

Windows Service Name

Create shortcuts for all users ☐

Tomcat Administrator Login (optional)

User Name	<input type="text"/>
Password	<input type="text"/>
Roles	<input type="text" value="manager-gui"/>

Nullsoft Install System v3.10

< Back Next > Cancel



Apache Tomcat Setup: Java Virtual Machine path sel...

Java Virtual Machine

Java Virtual Machine path selection.



Please select the path of a Java 17 or later JRE installed on your system.

C:\Program Files\Java\jdk-23

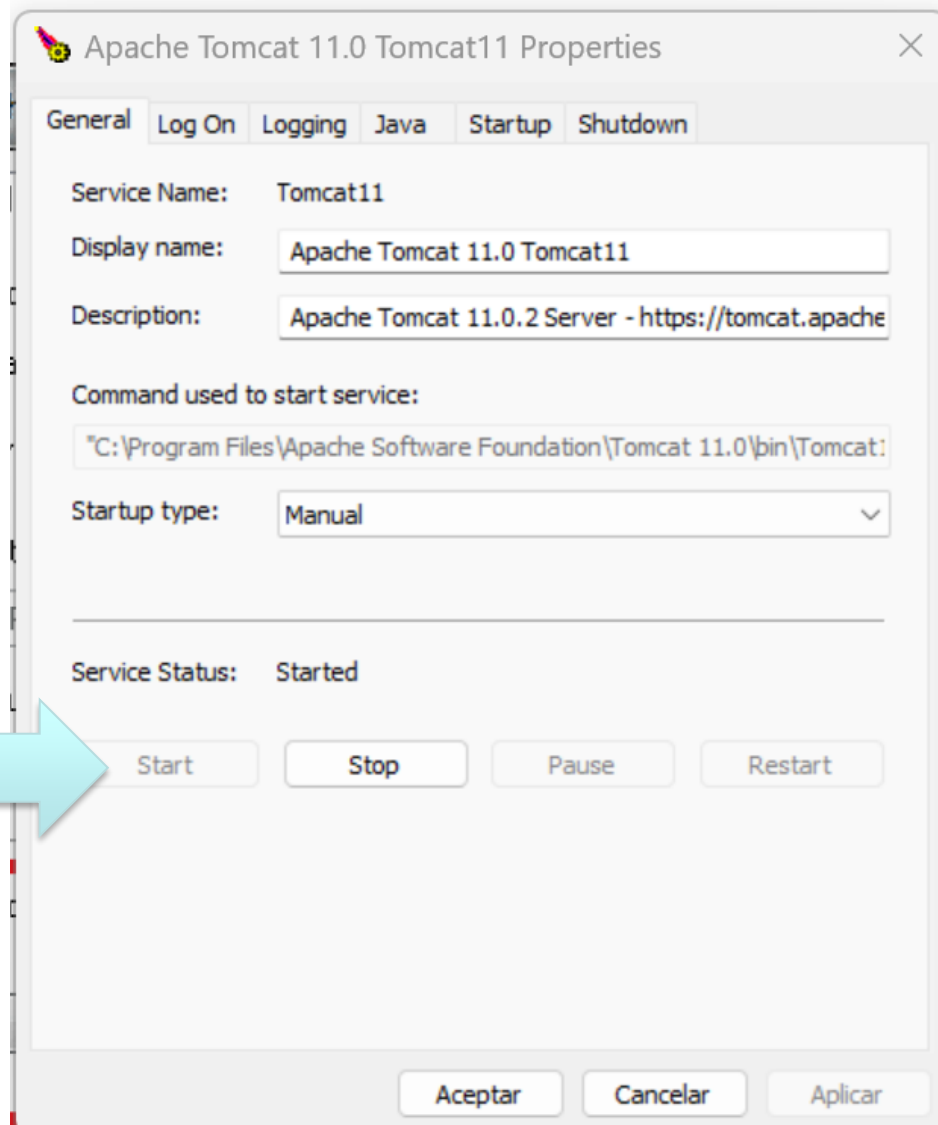
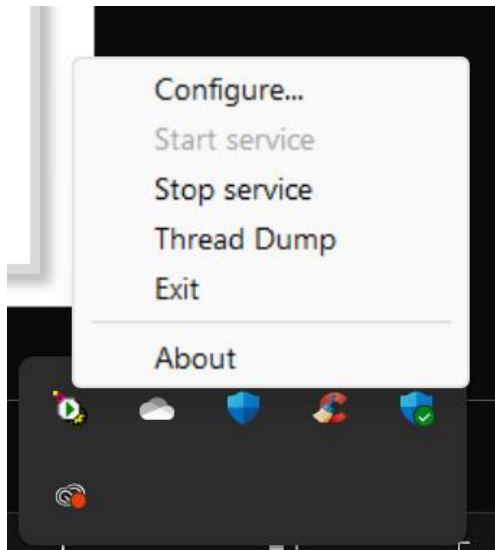


Nullsoft Install System v3.10

< Back

Next >

Cancel



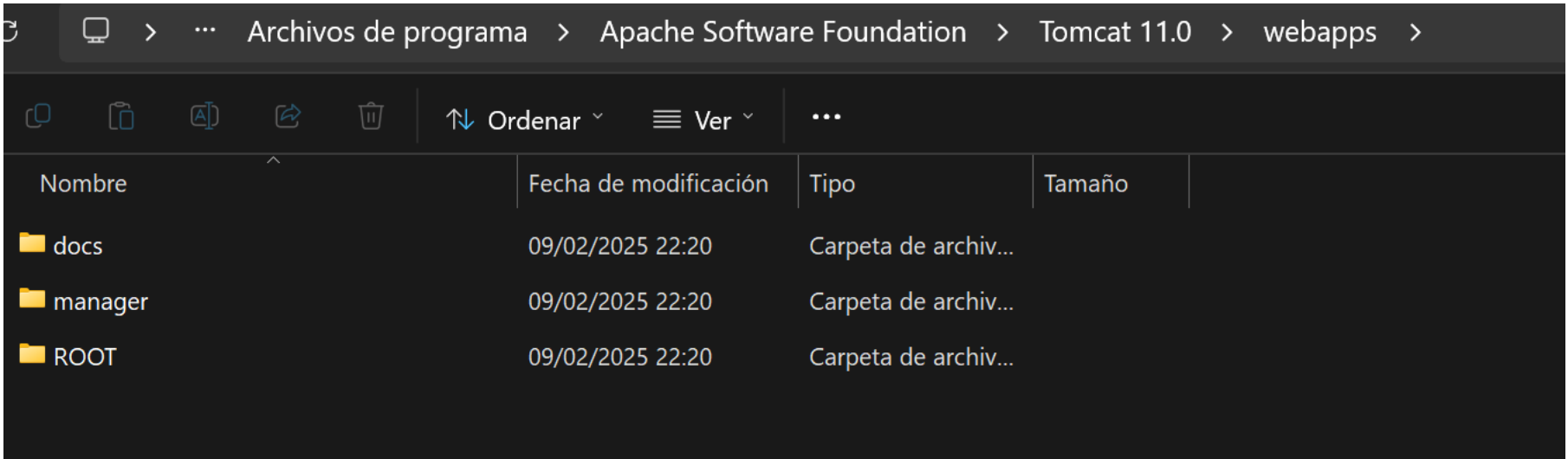
Puedes utilizar los botones *Start* y *Stop* para iniciar y detener el servicio. Cuando el servicio esté iniciado, podrás acceder a la dirección

<http://localhost:8080/>

en tu navegador para ver la página principal de administración de Apache Tomcat.

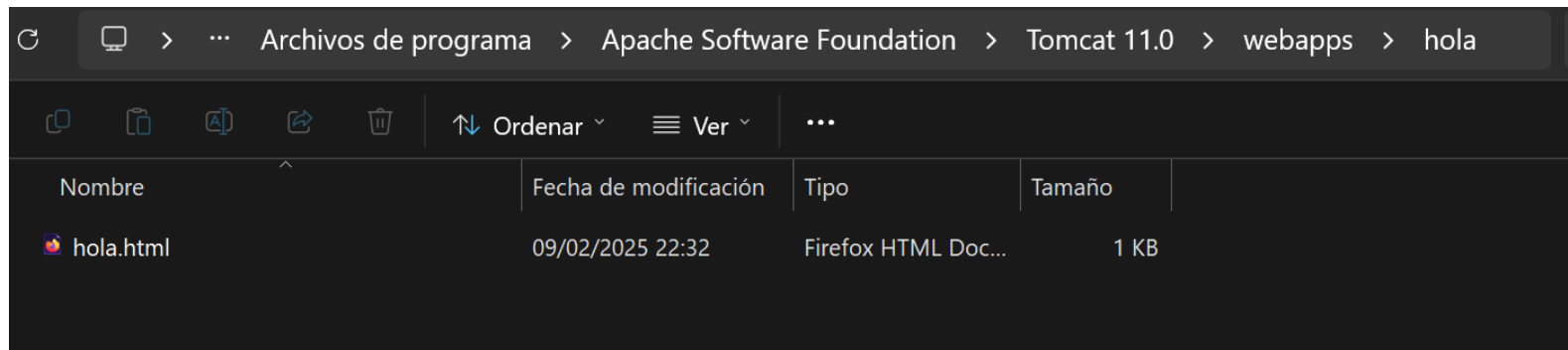
Apache Tomcat almacena los diferentes recursos (páginas web, imágenes y otros) en carpetas que llamaremos **aplicaciones web** o *webapps*. Para cada nueva aplicación, tiene que existir la correspondiente carpeta dentro de la ubicación "Apache Software Foundation/Tomcat 11.0/webapps".

Como mínimo, aparecerán tres carpetas que se crean durante la instalación. Carpeta *webapps* de Apache Tomcat que aloja las aplicaciones web.



Dentro de la carpeta *webapps*, crea una subcarpeta llamada *hola* y, dentro de esta, un archivo denominado *hola.html* con el siguiente código HTML:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Mi pagina hola mundo</title>
</head>
<body>
<h1>Hola Mundo</h1>
</body>
</html>
```



Si vamos a desarrollar en Java, como es nuestro caso, hay que hablar de JAX-RS (Java API for RESTful Web Services).

JARX-RS define algunas anotaciones que permiten especificar la forma en la que una clase Java normal se convierte en un recurso web. Algunas de estas anotaciones son: *@GET*, *@POST*, *@Path*, *@Produces*.

Pero JAX-RS sigue siendo una especificación (determinada para aplicaciones Java), y existen varias librerías que siguen dicha especialización y que pueden considerarse librerías JAX-RS.

Jersey es un API que sigue la especificación JAX-RS y que es, además, la más utilizada hoy en día en las aplicaciones Java empresariales.

[Edit my account](#) [Manage Cookies](#)[Projects](#)[Working Groups](#)[Members](#)[More ▾](#)[Download](#)

Eclipse Jersey

Eclipse Jersey is a REST framework that provides a JAX-RS (JSR-370) implementation and more

[Home](#) / [Eclipse Jersey](#)

About

Developing RESTful Web services that seamlessly support exposing your data in a variety of representation media types and abstract away the low-level details of the client-server communication is not an easy task without a good toolkit. In order to simplify development of RESTful Web services and their clients in Java, a standard and portable [JAX-RS API](#) has been designed.

Jersey RESTful Web Services 2.x framework is open source, production quality, framework for developing RESTful Web Services in Java that provides support for JAX-RS APIs and serves as a JAX-RS (JSR 311 & JSR 339 & JSR 370) Reference Implementation.

Jersey RESTful Web Services 3.x framework is open source, production quality, framework for developing RESTful Web Services in Java that provides support for Jakarta RESTful Web Services 3.0.

Jersey framework is more than the JAX-RS Reference Implementation. Jersey provides it's own [API](#) that extend the JAX-RS toolkit with additional features and utilities to further simplify RESTful service and client development. Jersey also exposes numerous extension APIs so that developers may extend Jersey to best suit their needs.

Project Resources

[► Sources](#)[► APIs](#)[► Documentation](#)[► Download](#)[► Mailing list](#)

Jakarta RESTful WebServices 3.1.0 / Jersey 3.1.10

Jersey 3.1.10, that implements [Jakarta RESTful WebServices 3.1](#) API is the future release of Jersey. Note that Jersey 2.x releases will continue along with Jersey 3.1.x releases.

For the convenience of non-maven developers the following links are provided:

- [Jersey 3.1.x bundle](#) bundle contains the Jakarta RESTful WebServices 3.1.0 API jar, all the core Jersey module jars as well as all the required 3rd-party dependencies.

All the Jersey 3.1 release binaries, including the source & apidocs jars, are available for download under the Jersey 3.1 maven root group identifier [org.glassfish.jersey](#) from the [maven central repository](#) as well as from the [Sonatype maven repository](#).

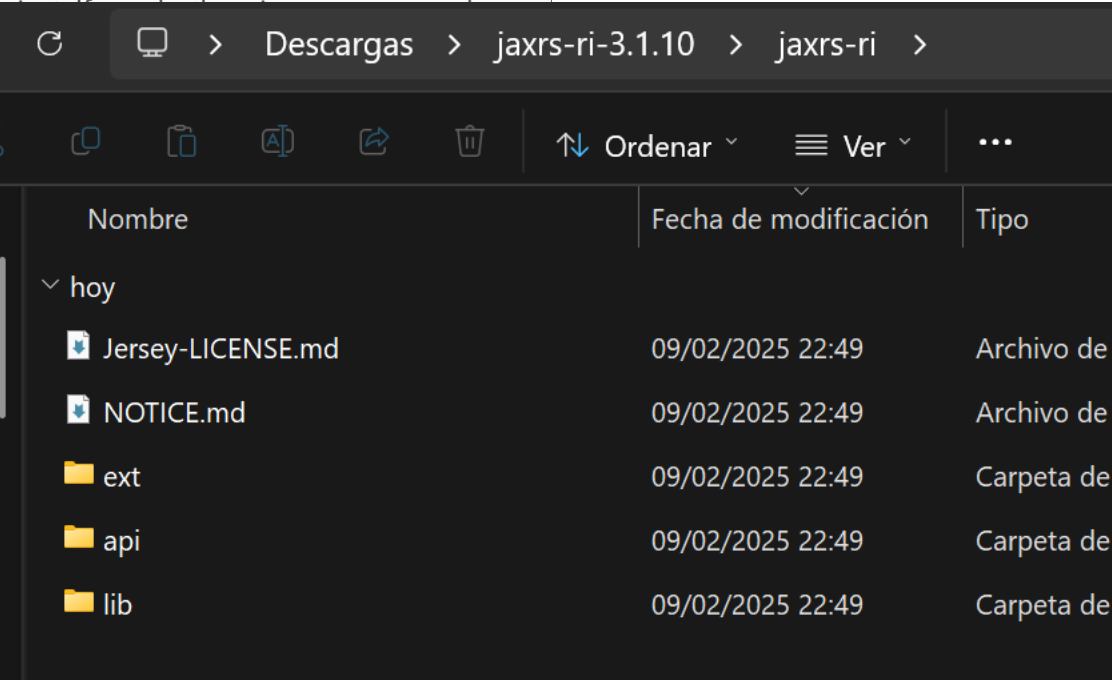
Chances are you are using Apache Maven as a build & dependency management tool for your project. A convenient way to start playing with Jersey 3.1.10 by generating the skeleton application from one of the archetypes that we provide.

For instance, to create a Jersey 3.1.10 application using the Grizzly 3.1 HTTP server container, use the following command:

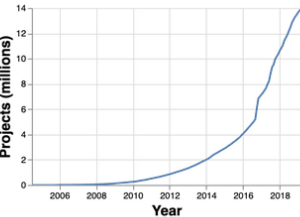
```
mvn archetype:generate -DarchetypeGroupId=org.glassfish.jersey.archetypes \
-DarchetypeArtifactId=jersey-quickstart-grizzly2 -DarchetypeVersion=3.1.10
```

If you want to create a Servlet container deployable Jersey 3.1.10 web application instead, use the following command:

```
mvn archetype:generate -DarchetypeGroupId=org.glassfish.jersey.archetypes \
-DarchetypeArtifactId=jersey-quickstart-webapp -DarchetypeVersion=3.1.10
```



Indexed Artifacts (50.7M)



Popular Categories

- Testing Frameworks & Tools
- Android Packages
- Logging Frameworks
- JVM Languages
- Java Specifications
- JSON Libraries
- Core Utilities
- Mocking
- Annotation Libraries
- Web Assets
- Language Runtime
- HTTP Clients
- Logging Bridges
- Dependency Injection

Home » [jakarta.platform](#) » jakarta.jakartaee-api



Jakarta EE Platform API

Jakarta EE Platform API

License	EPL 2.0 GPL
Categories	Java Specifications
Tags	jakarta standard api platform specs
Ranking	#1857 in MvnRepository (See Top Artifacts) #57 in Java Specifications
Used By	293 artifacts

Central (13)

	Version	Vulnerabilities	Repository	Usages	Date
11.0.x	11.0.0-M4		Central	4	Jun 30, 2024
	11.0.0-M3		Central	0	Jun 12, 2024
	11.0.0-M2		Central	0	Apr 16, 2024
	11.0.0-M1		Central	1	Dec 20, 2023
10.0.x	10.0.0		Central	163	Sep 13, 2022
	10.0.0-RC1		Central	5	Sep 13, 2022
9.1.x	9.1.0		Central	38	May 25, 2021
	9.1.0-RC1		Central	5	Feb 24, 2021

Downloading Apache Maven 3.9.9

Apache Maven 3.9.9 is the latest release: it is the recommended version for all users.

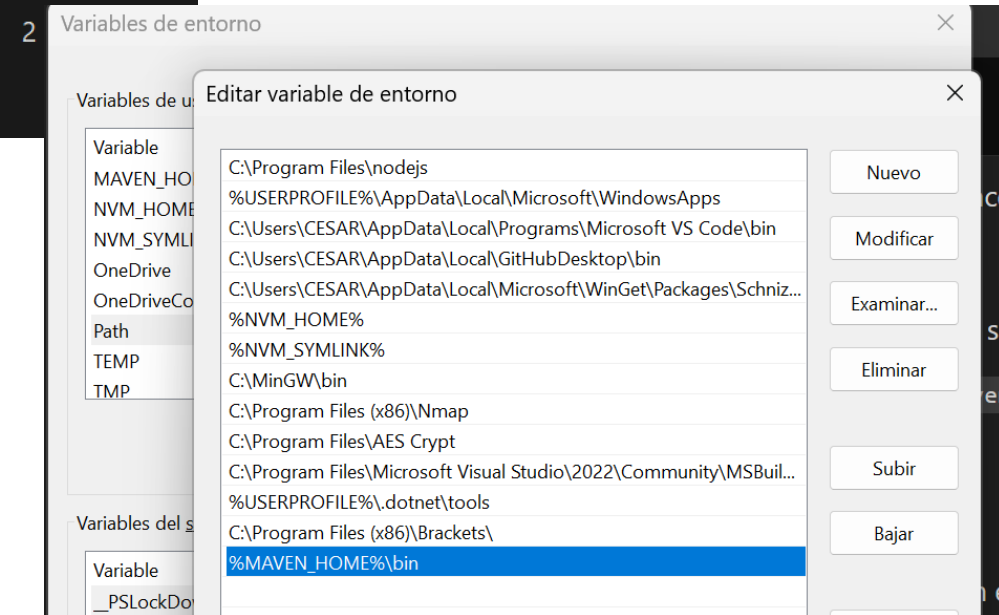
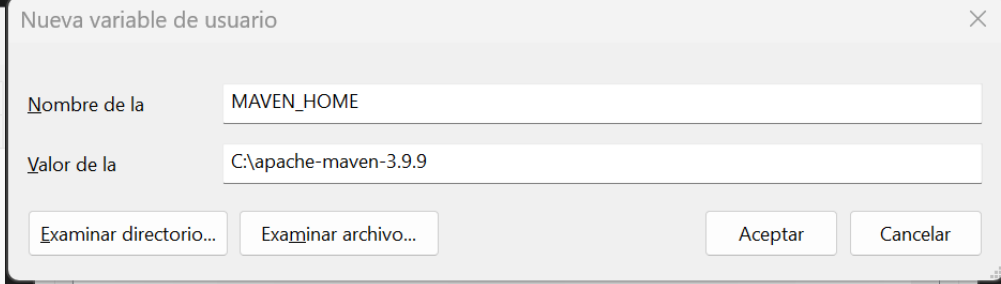
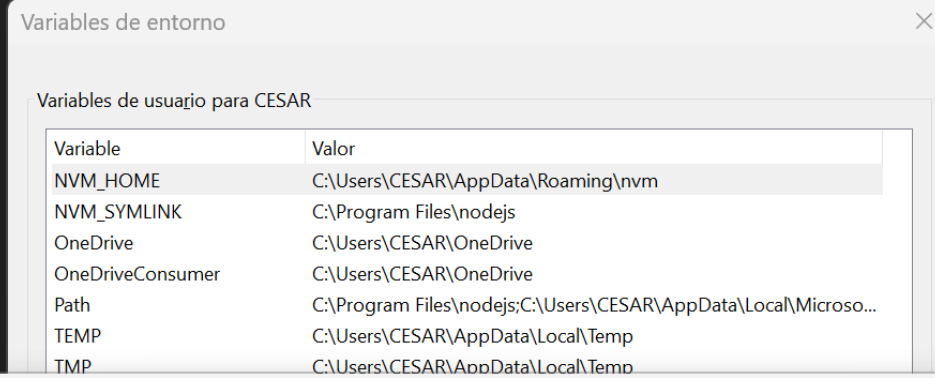
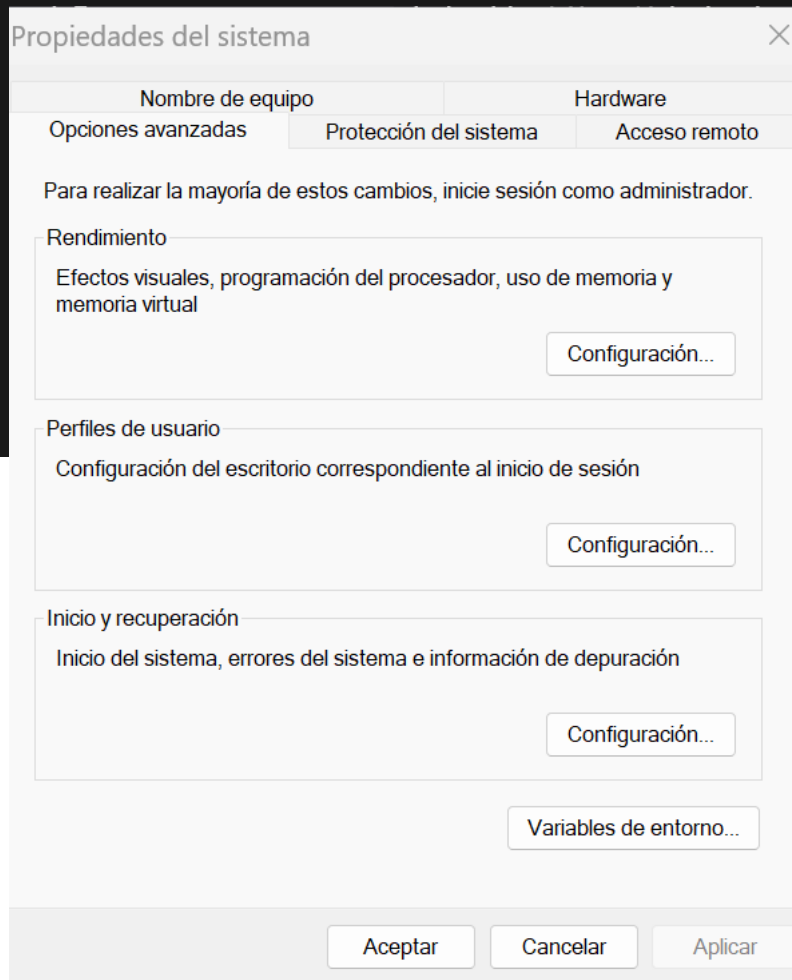
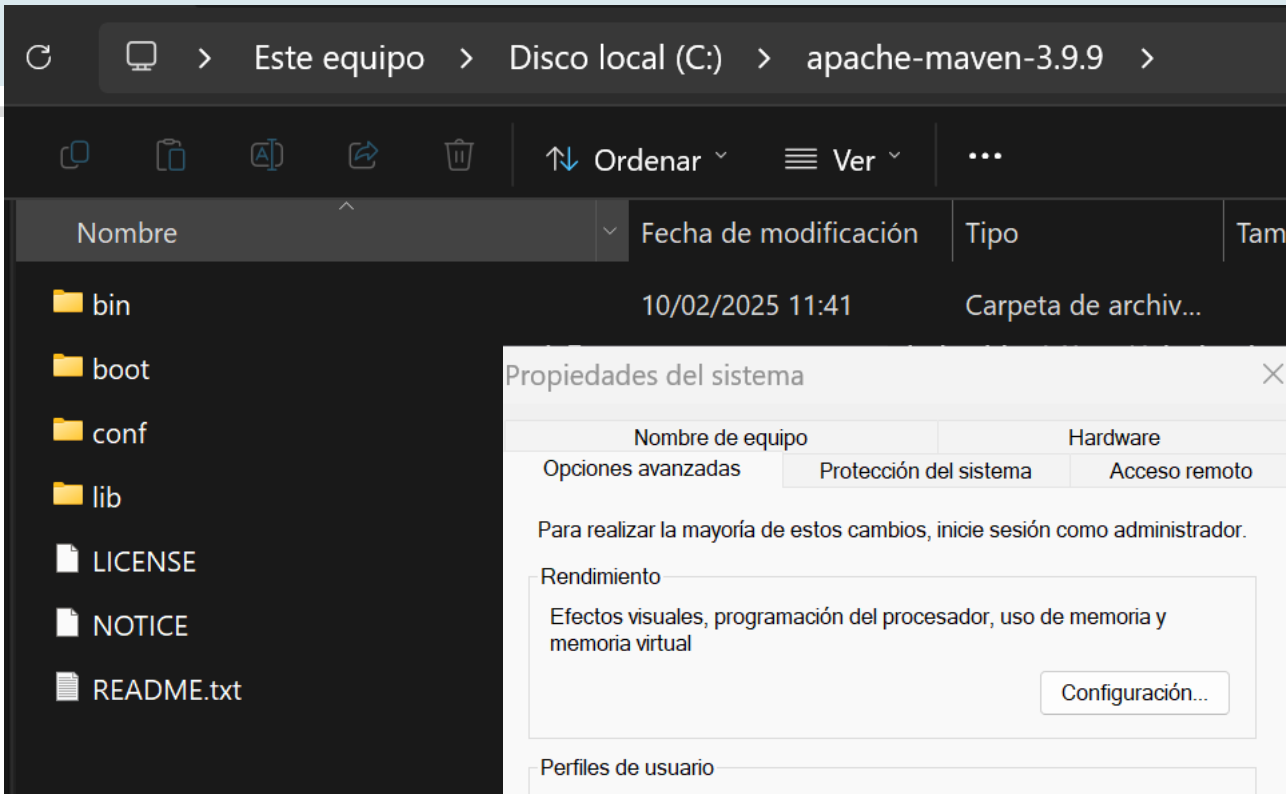
System Requirements

Java Development Kit (JDK)	Maven 3.9+ requires JDK 8 or above to execute. It still allows you to build against 1.3 and other JDK versions by using toolchains .
Memory	No minimum requirement
Disk	Approximately 10MB is required for the Maven installation itself. In addition to that, disk space will be used for your local Maven repository. The size of your local repository will vary depending on usage but expect at least 500MB.
Operating System	No minimum requirement. Start up scripts are included as shell scripts (tested on many Unix flavors) and Windows batch files.

Files

Maven is distributed in several formats for your convenience. Simply pick a ready-made binary distribution archive and follow the [installation instructions](#). Use a source archive if you intend to build Maven yourself.

In order to guard against corrupted downloads/installations it is highly recommended to [verify the signature](#) of the release bundles against the public [KEYS](#) used by the



```
PS C:\Users\CESAR\IdeaProjects\cesar-webapp> mvn -version
Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfcdc97d260186937)
Maven home: C:\apache-maven-3.9.9
Java version: 23.0.1, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-23
Default locale: es_ES, platform encoding: UTF-8
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"
PS C:\Users\CESAR\IdeaProjects\cesar-webapp>
```

```
PS C:\Users\CESAR\IdeaProjects\cesar-webapp> mvn clean package
[INFO] Scanning for projects...
[INFO]
[INFO] -----< org.example:cesar-webapp >-----
[INFO] Building cesar-webapp 1.0-SNAPSHOT
[INFO]    from pom.xml
[INFO] -----[ war ]-----
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-compiler-plugin/3.8.1/maven-compiler-plugin-3.8.1.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-compiler-plugin/3.8.1/maven-compiler-plugin-3.8.1.jar
```

Esto generará el archivo cesar-webapp.war en target/

Resumen

Descargar y extraer Maven en C:\apache-maven

Configurar MAVEN_HOME y agregar %MAVEN_HOME%\bin al Path

Verificar con mvn -version en la terminal

Reimportar Maven en IntelliJ y compilar el proyecto

Conectar IntelliJ con Apache Tomcat – versión Ultimate)

- Ir a **"Build, Execution, Deployment"**
- En la barra lateral izquierda, expande la opción **Build, Execution, Deployment**.
- **2. Buscar "Application Servers"**
- Dentro de **Build, Execution, Deployment**, debería aparecer la opción **Application Servers**.
- **3. Si no aparece**
- Ve a la barra de búsqueda en la parte superior y escribe **Application Servers**.
- Si no aparece, es posible que necesites instalar el **plugin "Application Servers"** en IntelliJ:
 - Ve a **File > Settings > Plugins**.
 - Busca **"Application Servers Integration"**.
 - Instálalo y reinicia IntelliJ.

1.**Ve a:** File > New > Project.

2.**Selecciona:** Jakarta EE.

3.**Configura el SDK** (si no tienes, selecciona Download JDK y elige una versión compatible).

4.En "**Application Server**", elige:

- Tomcat** (si necesitas JSP/Servlets).
- GlassFish, WildFly, Payara** (si usas Java EE completo).
- Si no tienes un servidor, selecciona Download.

5.Marca las tecnologías que necesitas, por ejemplo:

- Web Application** → Para Servlets y JSP.
- CDI** → Para inyección de dependencias.
- JPA** → Para bases de datos con Hibernate.

6.**Haz clic en Next**, asigna un nombre y selecciona la carpeta de destino.

7.**Finaliza con Finish.**

Conectar IntelliJ con Apache Tomcat – versión Community)

Compila el .war otra vez (mvn clean package).

Copia el nuevo .war en webapps/ (Tomcat lo desplegará automáticamente).

HACER UN EJEMPLO SENCILLO

Configurar el Proyecto en IntelliJ IDEA

1. **Abrir IntelliJ IDEA** y seleccionar **New Project**.
2. En la ventana de selección, elige **Maven** como tipo de proyecto.
3. Click en **Advance Settings**, elige un **Group ID** y un **Artifact ID**, y finaliza la creación del proyecto.

Proyecto/

├─ `src/`

│ ├─ `main/java/` → Código Java

│ ├─ `main/webapp/` → Archivos web (JSP, HTML, CSS)

│ ├─ `main/resources/` → Configuración (`persistence.xml`, `log4j`, etc.)

│ └─ `test/` → Pruebas

├─ `WEB-INF/`

│ ├─ `web.xml` → Configuración del despliegue

│ └─ `lib/` → Librerías adicionales

├─ `pom.xml` (si usas Maven)

├─ `build.gradle` (si usas Gradle)

Crear un Nuevo Proyecto en IntelliJ

1. Abre IntelliJ IDEA y selecciona New Project.

2. Selecciona "Maven" como tipo de proyecto.

3. Configura las coordenadas del proyecto:

- **Group ID:** org.example
- **Artifact ID:** cesar-webapp
- **Packaging:** war

4. Selecciona la versión del JDK (usa Java 21 o 23 si es compatible con tu configuración de Jakarta EE y Tomcat).

5. Haz clic en "Create".

Este pom.xml está listo para compilar un .war compatible con Jakarta EE 10 y Tomcat.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.example</groupId>
  <artifactId>cesar-webapp</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <properties>
    <maven.compiler.source>23</maven.compiler.source>
    <maven.compiler.target>23</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <!-- Jakarta EE API -->
    <dependency>
      <groupId>jakarta.platform</groupId>
      <artifactId>jakarta.jakartaee-api</artifactId>
      <version>10.0.0</version>
      <scope>provided</scope>
    </dependency>


    <!-- Servlets API -->
    <dependency>
      <groupId>jakarta.servlet</groupId>
      <artifactId>jakarta.servlet-api</artifactId>
      <version>6.0.0</version>
      <scope>provided</scope>
    </dependency>
```

```
    <!-- JSP API -->
    <dependency>
      <groupId>jakarta.servlet.jsp</groupId>
      <artifactId>jakarta.servlet.jsp-api</artifactId>
      <version>3.1.0</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <!-- Plugin para compilar con JDK 23 -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
        <configuration>
          <source>23</source>
          <target>23</target>
        </configuration>
      </plugin>

      <!-- Plugin para empaquetar como WAR -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <version>3.3.2</version>
      </plugin>
    </plugins>
  </build>
</project>
```

Forzar IntelliJ a recargar las dependencias Maven

- Ve a la pestaña **Maven** (a la derecha de IntelliJ).
- Haz clic en "**Reimport All Maven Projects**" (el icono de .
- O ejecuta en terminal:

```
mvn clean install -U
```

Crear la Página de Prueba index.jsp

 **Ubicación:** src/main/webapp/index.jsp

```
<html>
<head>
  <title>Inicio</title>
</head>
<body>
  <h1>Bienvenido a mi aplicación Jakarta EE</h1>
  <p><a href="hello">Ir al Servlet</a></p>
</body>
</html>
```

Crear el Servlet de Prueba

 **Ubicación:** src/main/java/HelloServlet.java

```
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<h1>¡Hola desde Jakarta EE!</h1>");
        out.println("</body></html>");
    }
}
```

este servlet responderá a las peticiones en <http://localhost:8080/cesar-webapp/hello>.

Compilar el Proyecto y Generar el .war

Abre una terminal en IntelliJ y ejecuta:

```
mvn clean package
```

Esto generará el archivo target/cesar-webapp.war

Copiar el .war en el Servidor Tomcat

1. Ubica la instalación de Tomcat

2. Copia target/cesar-webapp.war en TOMCAT_HOME/webapps/

```
apache-tomcat-10.1.16/  
├── bin/  
├── conf/  
├── lib/  
├── logs/  
├── temp/  
├── webapps/  
│   ├── cesar-webapp.war <-- Tu aplicación aquí  
│   ├── manager/  
│   └── host-manager/  
└── work/
```

Probar la Aplicación en el Navegador

Abre tu navegador y accede a:

- ◆ **Página principal (index.jsp)**

`http://localhost:8080/cesar-webapp/`

- ◆ **Ejecutar el servlet (HelloServlet)**

`http://localhost:8080/cesar-webapp/hello`