

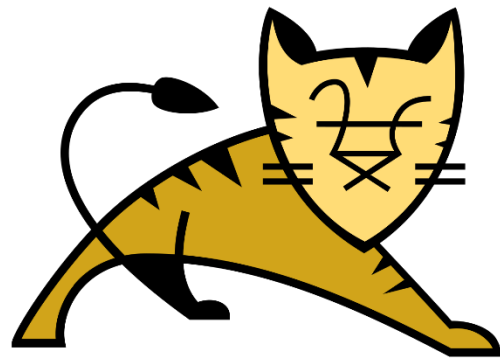
## Índice

<b>1. APLICACIÓN WEB JAVA CON MAVEN</b>	<b>1</b>
1.1. Servidor Tomcat	1
1.2. Maven	2
1.3. JSP y Servlets	3
<b>2. DESPLIEGUE CON APACHE TOMCAT</b>	<b>8</b>
2.1. Configuración	8
2.2. Despliegue .war	10
2.3. Autenticación de usuarios	11
2.4. Sesiones	13
2.5. Configuración SSL	13
2.6. Conectar Tomcat con Apache	15

## 1. APLICACIÓN WEB JAVA CON MAVEN

### 1.1. Servidor Tomcat

Vamos a realizar el despliegue en el **servidor de aplicaciones Apache Tomcat**. Como primero vamos a desarrollar una aplicación, no entraremos en detalles de configuración en el entorno de desarrollo, veremos más tarde sus posibilidades en el entorno de producción.



El servidor de aplicaciones Apache Tomcat **no dispone de instalador** por lo que la manera habitual de instalarlo es descargar la versión que queramos y descomprimirlo en la carpeta que queramos.

Web de descarga [aquí](#).

Utilizaremos el IDE [Netbeans](#) para el desarrollo de la aplicación web y para su despliegue en el entorno de desarrollo.

## 1.2. Maven

**Maven** es una herramienta **open-source**, que se creó en 2001 con el objetivo de simplificar los procesos de build (**compilar y generar ejecutables** a partir del código fuente).

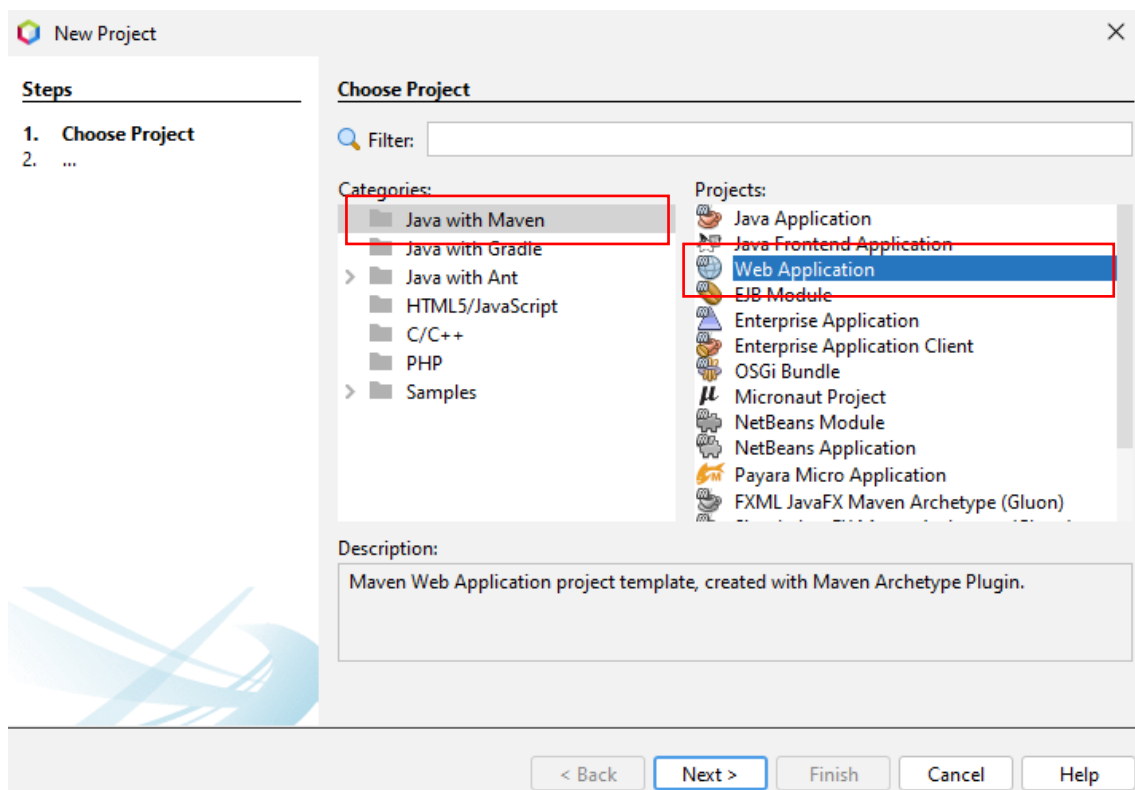
Por otra parte, con Maven la **gestión de dependencias** entre módulos y distintas versiones de librerías se hace muy sencilla. En este caso, solo tenemos que indicar los módulos que componen el proyecto, o qué librerías utiliza el software que estamos desarrollando en un **fichero de configuración** de Maven del proyecto llamado **POM** (Project Object Module).

Además, en el caso de las librerías, no tienes ni tan siquiera que descargarlas a mano. Maven posee un repositorio remoto (Maven central) donde se encuentran la mayoría de librerías que se utilizan en los desarrollos de software, y que la propia herramienta se descarga cuando sea necesario. Incluso, establece una estructura común de directorios para todos los proyectos.

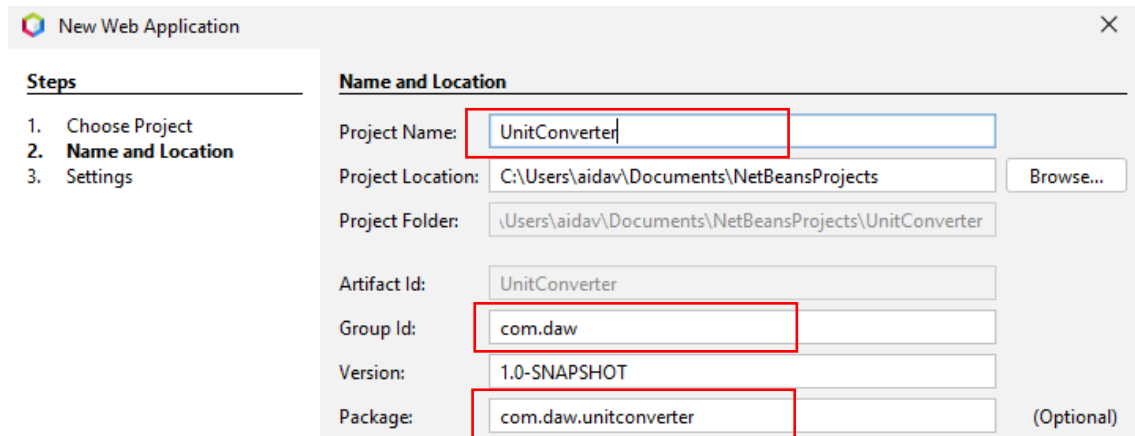


Así que, para **crear un proyecto web con Maven**, seleccionaremos:

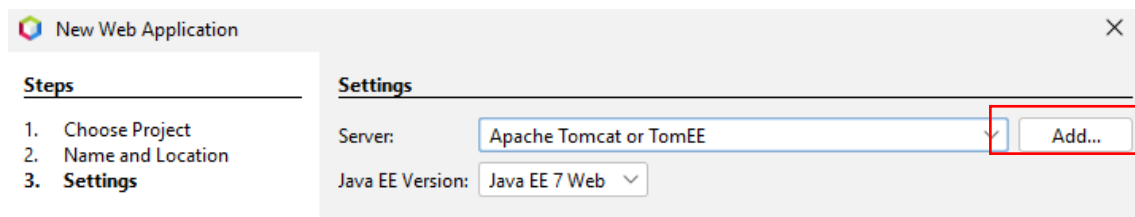
**New Project > Java with Maven > Web Application**



Vamos a crear una aplicación sencilla, llamada **UnitConverter**, que transforme kilómetros a millas.

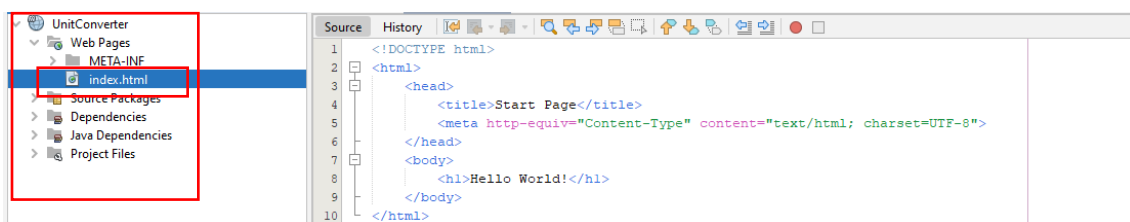


Y a la hora de indicar el **servidor**, seleccionamos Apache Tomcat y añadimos la ruta del archivo de Tomcat descomprimido.



Finalizamos y se crea toda la estructura del proyecto.

Ahora mismo, tenemos un proyecto cuya estructura y contenido de su **index.html** es:



Probemos a ejecutarlo, y accedemos con el navegador a: **http://localhost:8080**

### 1.3. JSP y Servlets

Vamos a modificar la aplicación para que realice lo que queremos: pasar de kilómetros a millas. Para ello necesitamos:

1. Un formulario que recoja los datos en kilómetros.

2. Como es una **aplicación web** de contenido dinámico, habrá que utilizar **JSP** (Java Server Pages) para ello.
3. Un **Servlet** (“controlador”) que realice el paso de kilómetros a millas.

Para ello, realizamos los siguientes pasos:

1. Eliminamos **index.html** y en su lugar (mismo sitio, en **Web pages**) creamos una clase llamada **index.jsp** con **New > JSP**:
2. Creamos un formulario donde se recojan los datos.

**index.jsp**

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"%>

    <title>JSP Page</title>

</head>

<body>

    <div align="center">

        <h1>Conversor de unidades</h1>

        <form action="convert" method="post">

            Kilómetros: <input type="number" name="km" required />

            <input type="submit" value="Convertir a millas" />

        </form>

    </div>

</body>

</html>
```

3. Creamos un **Servlet** donde poder trabajar con esos datos. Lo crearemos en **src**, para ello: **New > Servlet**

**New Servlet**

**Steps**

1. Choose File Type
2. **Name and Location**
3. Configure Servlet Deployment

**Name and Location**

Class Name:

Project:

Location:

Package:

Created File:

**Configure Servlet Deployment**

Register the Servlet with the application by giving the Servlet an internal name (Servlet Name). Then specify patterns that identify the URLs that invoke the Servlet. Separate multiple patterns with commas.

☒ Add information to deployment descriptor (web.xml)

Class Name:

Servlet Name:

URL Pattern(s):

Projects: UnitConverter

- Web Pages
  - META-INF
    - context.xml
  - WEB-INF
    - index.jsp
- Source Packages
  - com.daw.unitconverter
    - UnitConverterServlet.java
- Dependencies
  - javaee-web-api-7.0.jar
  - Java Dependencies
  - Project Files

En **URL Patterns** escribimos el *path* que queremos que nos lleve cuando se haga la conversión km a millas.

4. El **Servlet** que hemos creado debe **pasar de km a millas** (dividir por 1,609) y mostrar el resultado. Para ello, en **UnitConverterServlet.java**, modificamos el **método processRequest** como se muestra:

**UnitConverterServlet.java**

```
protected void processRequest(HttpServletRequest request,
                              HttpServletResponse response)
    throws ServletException, IOException {

    float km = Float.parseFloat(request.getParameter("km"));

    float millas = km / 1.609f;

    response.setContentType("text/html; charset=UTF-8");

    try ( PrintWriter out = response.getWriter()) {

        /* TODO output your page here. You may use
        following sample code. */
    }
}
```

```
        out.println("<!DOCTYPE html>");

        out.println("<html>");

        out.println("<head>");

        out.println("<title>Resultado</title>");

        out.println("</head>");

        out.println("<body>");

        out.println("<h1>Resultado</h1>");

        out.println("<p>" + km + " kilómetros = " + millas + " millas</p>");

        out.println("</body>");

        out.println("</html>");

    }

}
```

Verificamos que **la aplicación funciona correctamente**.

Como vemos, **utilizar un Servlet para generar código html es muy tedioso y poco práctico**, por eso sólo se recomienda cuando el código html es muy reducido. Cuando no es el caso, **es más cómodo pasar la información del Servlet a una JSP** para que se encargue ella del código html.

En el ejemplo que estamos desarrollando, lo haríamos de la siguiente forma:

1. **Modificamos el Servlet**, eliminamos todos los métodos menos el **doPost** (puesto que va a tratar la información del formulario) y en él hacemos los cálculos y enviamos los resultados a...
2. ... un archivo **result.jsp** (lo creamos en **Web pages**). Lo configuramos para que muestre un html con los datos que le pasa el Servlet.

Vemos cómo quedarían ambos archivos:

## UnitConverterServlet.java

```
protected void doPost(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {

    float km = Float.parseFloat(request.getParameter("km")
);

    float millas = km / 1.609f;

    request.setAttribute("km", km);

    request.setAttribute("millas", millas);

    RequestDispatcher
dispatcher = request.getRequestDispatcher("result.jsp");

    dispatcher.forward(request, response);

}

}
```

## result.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

    <head>

        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

        <title>Resultado</title>

    </head>

    <body>

        <div align="center">

            <h1>Resultado</h1>

            <p>
```

```
        ${km} kilómetros = ${millas} millas

    </p>

</div>

</body>

</html>

}
```

## 2. DESPLIEGUE CON APACHE TOMCAT

En este apartado vamos a ver cómo desplegar la aplicación en un **entorno de producción** con un servidor **Apache Tomcat**.

### 2.1. Configuración

1. Lo primero que tenemos que hacer es **instalar Java**:

```
apt install default-jdk -y
```

2. Podemos comprobar la instalación: `java -version`

3. Instalamos **Tomcat**:

```
sudo apt install tomcat9 -y
```

4. Vamos a crear un **usuario** que se encargue de **Tomcat**. Para ello primero creamos un grupo y luego añadimos el usuario.

```
sudo groupadd tomcat9
```

```
sudo useradd -s /bin/false -g tomcat9 -d /etc/tomcat9 tomcat9
```

5. Ahora que Tomcat está instalado, podemos **iniciar** el servicio:

```
sudo service tomcat9 start
```

6. Ahora es el momento de definir el **usuario** con **acceso a Tomcat**. Para hacerlo:

```
sudo nano /etc/tomcat9/tomcat-users.xml.
```

Modificamos:

```
<role rolename="admin"/>
```

```
<role rolename="admin-gui"/>
```

```
<role rolename="manager"/>
```

```
<role rolename="manager-gui"/>
```



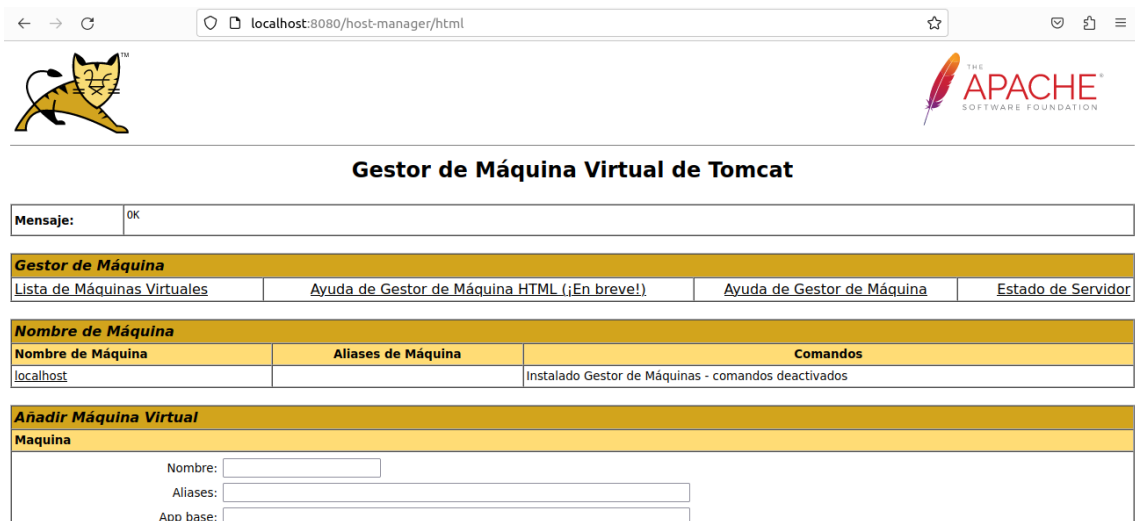
```
<user username="nombre" password="contraseña" roles="admin,admin-gui,manager,manager-gui"/>
```

7. Comprobamos que la instalación funciona: <http://localhost:8080/>

8. Ahora, instalemos el **administrador web** de Tomcat:

```
sudo apt install tomcat9-admin
```

9. Accedemos a: <http://localhost:8080/host-manager/html> con las credenciales que establecimos antes y comprobamos.



**Gestor de Máquina Virtual de Tomcat**

Mensaje: OK

**Gestor de Máquina**

[Lista de Máquinas Virtuales](#) [Ayuda de Gestor de Máquina HTML \(:En breve!\)](#) [Ayuda de Gestor de Máquina](#) [Estado de Servidor](#)

Nombre de Máquina		
Nombre de Máquina	Alias de Máquina	Comandos
localhost		Instalado Gestor de Máquinas - comandos desactivados

**Añadir Máquina Virtual**

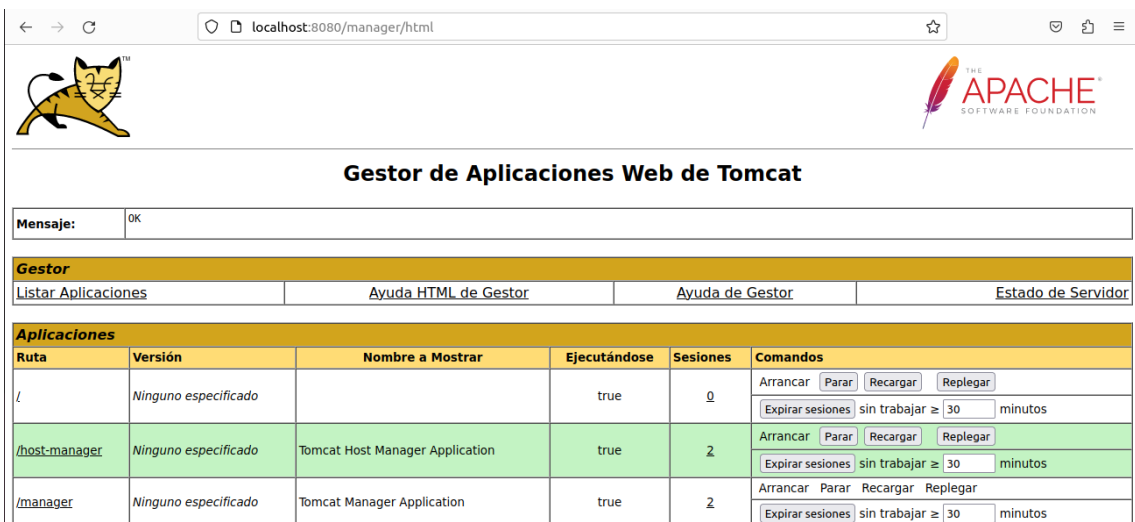
Maquina

Nombre:

Alias:

App base:

10. Y también: <http://localhost:8080/manager/html>



**Gestor de Aplicaciones Web de Tomcat**

Mensaje: OK

**Gestor**

[Listar Aplicaciones](#) [Ayuda HTML de Gestor](#) [Ayuda de Gestor](#) [Estado de Servidor](#)

Aplicaciones					
Ruta	Versión	Nombre a Mostrar	Ejecutándose	Sesiones	Comandos
/	Ninguno especificado		true	0	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos
/host-manager	Ninguno especificado	Tomcat Host Manager Application	true	2	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos
/manager	Ninguno especificado	Tomcat Manager Application	true	2	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos

## 2.2. Despliegue .war

Como hemos visto en la teoría, podemos realizar el despliegue de toda una aplicación utilizando **el archivo .war** que la empaqueta.

Para ello:

1. Localizamos el archivo en nuestra aplicación.

Este equipo > Documentos > NetBeansProjects > UnitConverter > target

Nombre	Fecha de modificación	Tipo	Tamaño
classes	14/12/2023 19:04	Carpeta de archivos	
endorsed	14/12/2023 19:04	Carpeta de archivos	
generated-sources	14/12/2023 19:04	Carpeta de archivos	
maven-archiver	14/12/2023 19:04	Carpeta de archivos	
maven-status	14/12/2023 19:04	Carpeta de archivos	
test-classes	14/12/2023 19:04	Carpeta de archivos	
UnitConverter-1.0-SNAPSHOT	14/12/2023 19:04	Carpeta de archivos	
UnitConverter-1.0-SNAPSHOT.war	14/12/2023 19:04	Archivo WAR	5 KB

2. Lo transferimos al entorno de producción (por ejemplo, por FTP).

3. Tenemos dos opciones:

- a. Desplegar con el **manager gui**:

Archivo WAR a desplegar					
Seleccione archivo WAR a cargar <input type="button" value="Examinar..."/> UnitConverter.war <input type="button" value="Desplegar"/>					

Aplicaciones					
Ruta	Versión	Nombre a Mostrar	Ejecutándose	Sesiones	Comandos
/	Ninguno especificado		true	0	<input type="button" value="Arrancar"/> <input type="button" value="Parar"/> <input type="button" value="Recargar"/> <input type="button" value="Replegar"/> <input type="button" value="Expirar sesiones"/> sin trabajar ≥ 30 minutos
/UnitConverter	Ninguno especificado		true	0	<input type="button" value="Arrancar"/> <input type="button" value="Parar"/> <input type="button" value="Recargar"/> <input type="button" value="Replegar"/> <input type="button" value="Expirar sesiones"/> sin trabajar ≥ 30 minutos
/host-manager	Ninguno especificado	Tomcat Host Manager Application	true	0	<input type="button" value="Arrancar"/> <input type="button" value="Parar"/> <input type="button" value="Recargar"/> <input type="button" value="Replegar"/> <input type="button" value="Expirar sesiones"/> sin trabajar ≥ 30 minutos
/manager	Ninguno especificado	Tomcat Manager Application	true	1	<input type="button" value="Arrancar"/> <input type="button" value="Parar"/> <input type="button" value="Recargar"/> <input type="button" value="Replegar"/> <input type="button" value="Expirar sesiones"/> sin trabajar ≥ 30 minutos

- b. Copiar el archivo .war (o copiar toda la carpeta de webapp del proyecto) a **/var/lib/tomcat9/webapps**, que es donde residen las aplicaciones desplegadas en Tomcat.

Este equipo > Documentos > NetBeansProjects > UnitConverter > src > main

Nombre	Fecha de modificación	Tipo	Tamaño
java	10/12/2023 18:44	Carpeta de archivos	
webapp	10/12/2023 20:15	Carpeta de archivos	

Para que despliegue por defecto todas las aplicaciones que residen ahí, habrá que configurar en `/etc/tomcat9/server.xml`:

```
. . .  
<Host name="localhost"  appBase="webapps"  
      unpackWARs="true" autoDeploy="true">  
. . .
```

En ambos casos, tendríamos la aplicación desplegada en:  
<http://localhost:8080/UnitConverter>.



## 2.3. Autenticación de usuarios

Si estamos utilizando el servidor de aplicaciones Tomcat, podremos hacer uso de la autenticación HTTP. Este tipo de autenticación es considerada un tipo de autenticación insegura ya que no se encriptan los datos de acceso (usuario y password), a no ser que las conexiones se apoyen sobre el protocolo seguro HTTPS.

Para poder utilizar este tipo de autenticación tendremos que configurar el fichero `web.xml`.

Simplemente tendremos que añadir un elemento `<security-constraint>` y `<login-config>` e indicar un rol y usuario correcto para las urls que queremos proteger.

Necesitaremos crear primeramente los roles y usuarios que necesitemos en el fichero `/etc/tomcat9/tomcat-users.xml`

Para crear el rol “administradores”, y los usuarios que pertenezcan a ese rol, tendremos que configurar:

```
<role name="administradores"/>
```

```
<user name="usuario" password="mi_clave_secreta"
roles="administradores"/>
```

Y añadimos en `web.xml`:

### web.xml

```
<!--
Codigo que tenemos que añadir al fichero web.xml
-->

<security-constraint>

  <web-resource-collection>

    <web-resource-name>

      Entire Application

    </web-resource-name>

    <url-pattern>/*</url-pattern>

  </web-resource-collection>

  <auth-constraint>

    <role-name>administradores</role-name>

  </auth-constraint>

</security-constraint>

<!-- Define the Login Configuration for this Application -->

<login-config>

  <auth-method>BASIC</auth-method>

  <realm-name>Area de acceso restringido</realm-name>

</login-config>
```

Verificamos el funcionamiento.

## 2.4. Sesiones

Podemos administrar las sesiones abiertas de una aplicación desplegada en:

Aplicaciones					
Ruta	Versión	Nombre a Mostrar	Ejecutándose	Sesiones	Comandos
/	Ninguno especificado		true	0	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos
/UnitConverter	Ninguno especificado		true	2	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos
/host-manager	Ninguno especificado	Tomcat Host Manager Application	true	1	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos
/manager	Ninguno especificado	Tomcat Manager Application	true	1	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos

### Sessions Administration for /UnitConverter

Tips:

- Click on a column to sort.
- To view a session details and/or remove a session attributes, click on its id.

Active HttpSessions informations

Refresh Sessions list 2 active Sessions

Session Id	Type	Guessed Locale	Guessed User name	Creation Time	Last Accessed Time	Used Time	Inactive Time	TTL
<input type="checkbox"/> <a href="#">2E879BE98DE89E692667D64839EF8A0D</a>	Primary			2023-12-14 14:14:04	2023-12-14 14:34:25	00:20:20	00:01:06	00:28:53
<input type="checkbox"/> <a href="#">F88D5043487AC305E6FE0D3F93B2B482</a>	Primary			2023-12-14 14:32:25	2023-12-14 14:32:25	00:00:00	00:03:06	00:26:53

Invalidate selected Sessions

Return to main page

Para cambiar el tiempo de timeout de sesión del tomcat necesitas especificarlo con el parámetro `session-timeout` en el archivo `web.xml` que está en bajo el directorio `WEB-INF` ej. `WEB-INF/web.xml` de nuestra aplicación.

## 2.5. Configuración SSL

Configurar Tomcat para utilizar el protocolo HTTPS. Utilizaremos el comando `keytool` para generar un fichero de claves y un certificado.

```
sudo keytool -genkey -alias tomcat -keyalg RSA
```

Por defecto el fichero se crea en la carpeta del usuario que ha ejecutado el comando con el nombre `.keystore`. Podemos copiarlo dentro de la carpeta `/etc/tomcat9` que es donde se almacenan los ficheros de configuración de Tomcat.

```
sudo cp /home/USUARIO/.keystore /etc/tomcat9
```

A continuación, en el fichero de configuración principal de Tomcat, **server.xml**, tendremos que buscar la etiqueta donde se configura el conector para el puerto HTTPS y añadir algunos parámetros para comenzar a usarlo y asignarle el certificado que acabamos de crear.

La configuración por defecto es la que se muestra a continuación:

```
<Connector port="8443"
protocol="org.apache.coyote.http11.Http11Protocol"

    maxThreads="150" SSLEnabled="true" scheme="https" secure="true"

    clientAuth="false" sslProtocol="TLS" />
```

Y tenemos que modificar los atributos de la etiqueta **Connector** para que quede como en el siguiente fragmento:

```
<Connector SSLEnabled="true" acceptCount="100" clientAuth="false"

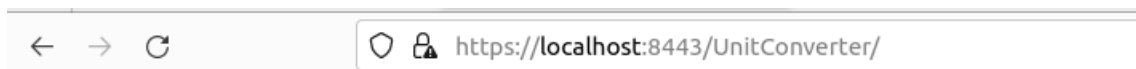
    disableUploadTimeout="true" enableLookups="false" maxThreads="25"

    port="8443" keystoreFile="/etc/tomcat9/.keystore"
keystorePass="CONTRASEÑA"

    protocol="org.apache.coyote.http11.Http11NioProtocol"
scheme="https"

    secure="true" sslProtocol="TLS" />
```

Una vez hechos estos cambios, tendremos que reiniciar el servidor de Tomcat y podremos visitar la dirección <https://localhost:8443> para comprobar que el certificado funciona. Habrá que tener en cuenta que, puesto que el certificado es autofirmado, aparecerá una advertencia en el navegador avisando de ello.



## 2.6. Conectar Tomcat con Apache

En el caso de que en el mismo servidor se encuentre ya funcionando un **servidor Apache** nos puede resultar útil poder configurar en él un **host virtual cuyo destino real sea una aplicación web desplegada en el servidor de aplicaciones**, simplificando así en gran medida la configuración de ambos servidores y permitiendo que convivan sin mayor problema.

Podremos **configurar un host virtual en Apache** como lo veníamos haciendo hasta ahora ya que, en este caso, se trata simplemente de añadir un par de líneas en dicha configuración:

```
<VirtualHost *:80>

    ServerName unitconverter.com

    ServerAlias www.unitconverter.com

    ProxyPass / ajp://localhost:8009/UnitConverter/

    ProxyPassReverse / ajp://localhost:8009/UnitConverter/

</VirtualHost>
```

El siguiente paso será **activar el módulo proxy\_ajp** si no está ya activado y, a continuación, **reiniciar el servidor** para que los cambios tengan efecto (acordarse de modificar **/etc/hosts**).

```
sudo a2enmod proxy_ajp

sudo service apache2 restart
```

En la **configuración del servidor Tomcat** habrá que activar la siguiente línea en el fichero **/etc/tomcat9/server.xml**, que encontraremos comentada.

```
<Connector protocol="AJP/1.3" address="0.0.0.0" port="8009"
redirectPort="8443" secretRequired="false"/>
```

Y a partir de ahora podemos **visitar el nuevo host virtual** creado en Apache y, de forma transparente, se **cargará la aplicación web que se encuentra desplegada en el servidor Tomcat**.



### Conversor de unidades

Kilómetros: