Sistemas Distribuidos e Internet
Introducción a JEE – Servlets y JSPs
Sesión 1 y 2
Curso 2023/2024

Contenido

1 Introducción

En esta práctica vamos a implementar una aplicación web que simule una pequeña tienda de la compra, empleando de forma progresiva las tecnologías y conceptos base de la tecnología de Servidor JEE (Java Enterprise Edition). Emplearemos Servlets, JSPs (Java Server Pages), JavaBeans y JSTL (JavaServer Pages Standard Tag Library). Finalmente, encajaremos la aplicación desarrollada en un patrón arquitectónico MVC (Model View Controller):

- Servlet: Clase Java "desplegable" que es la base del framework JEE.
- JSP: Combinación de código Java embedido en una página HTML que es traducido a un servlet cuando es desplegado por primera vez.
- JavaBean: Clase Java que debe cumplir ciertas condiciones y que es accesible desde un JSP o código JSTL.
- JSTL: Estructuras de control que pueden ir en una página JSP.

Nota: De manera progresiva el alumno deberá ir subiendo el código a GitHub en los puntos de controles especificados en el documento.

2 Instalación y configuración del entorno de desarrollo

Para desarrollar aplicaciones web en Java es necesario trabajar con un Entorno de Desarrollo Intergrado (IDE) adecuado. Uno de los IDE más utilizado en entornos empresariales es **IntelliJ IDEA**[1]. También es un buen entorno de desarrollo integrado el IDE **Spring Tool Suite (STS)**[2], herramienta que está basada en el IDE Eclipse. Ambos entornos incluyen todo lo necesario para facilitarnos el desarrollo de aplicaciones web en Java y también están optimizado para el uso del framework Spring con el que desarrollaremos más adelante.

Prerequisitos o softwares utilizados:

1. **IntelliJ IDEA ULTIMATE** versión 2021.3.2[3]: Entorno de Desarrollo Integrado.
2. **JDK-17.0.1**: Java Development Kit.
3. **Git 2.20.1**: Sistema de control de versiones.
4. **Apache Tomcat Web Server** versión 9.0.41**:** Contendor de Servlet Open source.
5. **Bootstrap** versión 4.5.2: Framework para crear sitios web responsive.
6. **JQuery** versión 3.5.1: Biblioteca multiplataforma de JavaScript.

Nota: Para el desarrollo de aplicaciones web empresariales en JEE se debe utilizar la edición **ULTIMATE de IntelliJ IDEA**, ya que esta funcionalidad no está disponible en la versión Community Edition de Intellij IDEA.

[1] https://www.jetbrains.com/idea/download/

[2] https://spring.io/tools

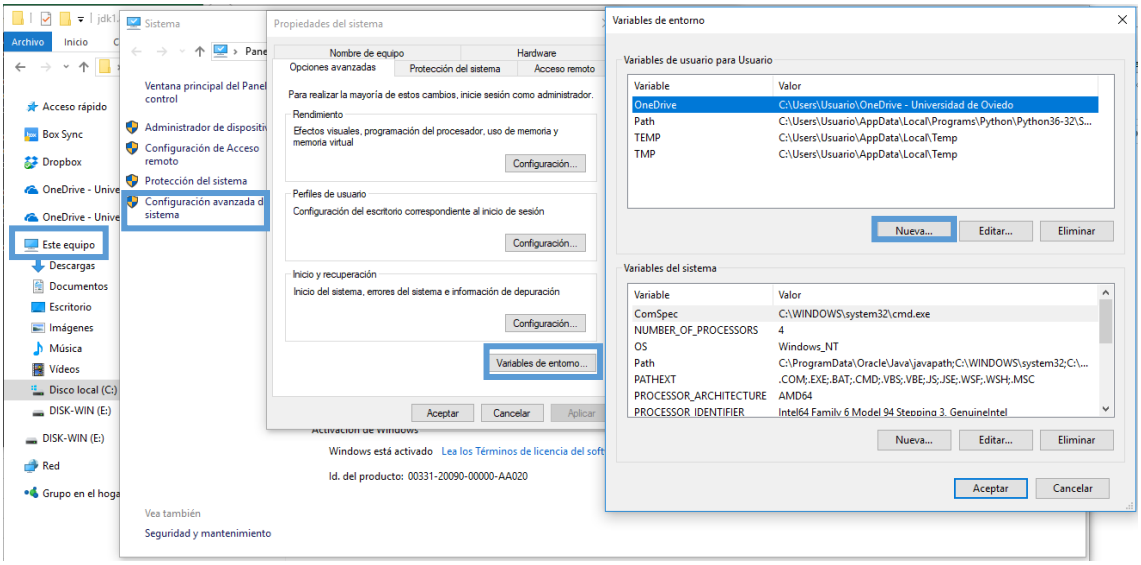
[3] En 17 de enero de 2024 la versión disponible era 2023.3.2. Es igualmente válida.

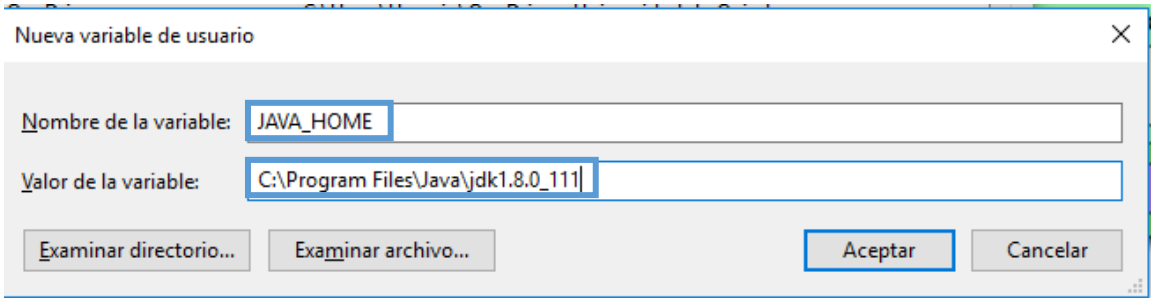
2.1 Descargar e instalar Java

Debemos descargar la versión 17 (JDK-17.0.6) de Java desde la siguiente URL: https://www.java.com/es/download/. Este documento muestra el procedimiento para la versión 8 de Java, pero el procedimiento es válido para cualquier versión.

2.2 Configurar las variables de entorno del sistema

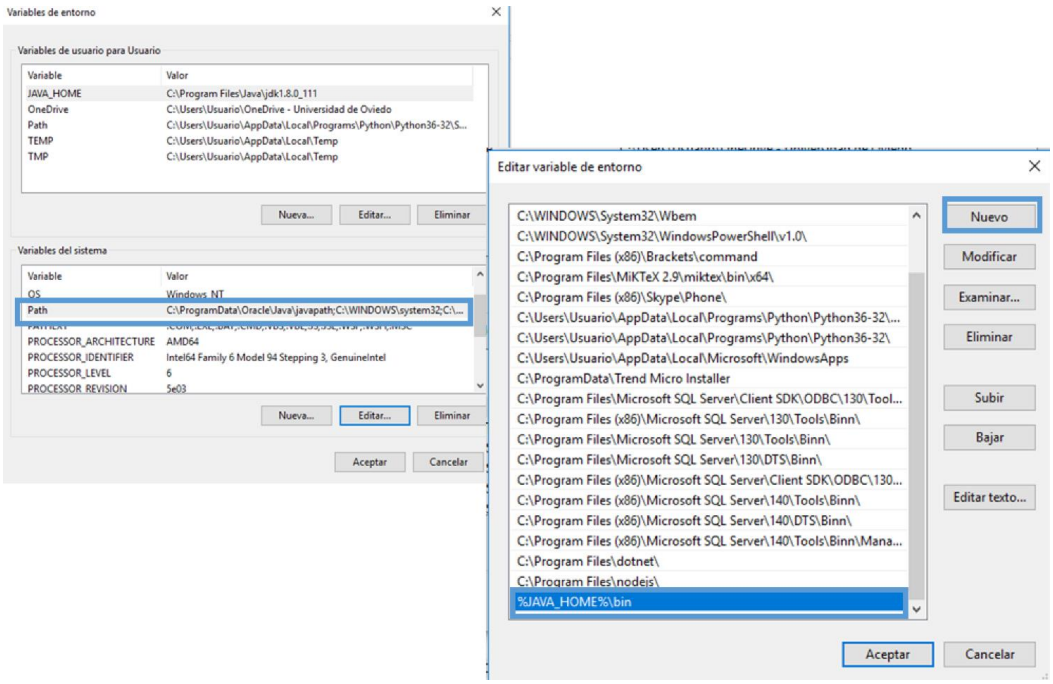
Una vez descargado e instalado Java, configuramos la variable de entorno JAVA_HOME[4]. Desde el explorador de Windows, hacemos clic derecho sobre **"Este equipo",** vamos al menú de **propiedades** y luego seguimos los pasos como se muestran en la siguiente imagen:





[4] Para configurar JAVA_HOME en MacOSX usa $> export JAVA_HOME=$(/usr/libexec/java_home) (Enlace)

En la misma ventana de configuración de variables de entorno, el siguiente paso es editar la variable **PATH** existente. Para ello, seguimos los pasos que se muestran en la siguiente imagen:



2.3 Descargar e Instalar Git

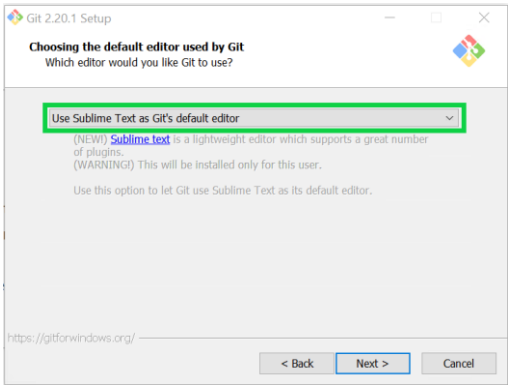
Git es uno de los sistemas de control de versiones más utilizados en el mercado y vamos a utilizarlo para llevar el control de nuestro código fuente. Instalar Git en Windows es muy fácil. Podemos descargar un instalador desde la siguiente URL y seguir los pasos indicados.

> **Nota**: Instalar previamente un editor de texto como Notepad++ o Sublime Text para utilizarlo como editor de texto de Git. Aunque en nuestro caso, usaremos el editor del IDE.

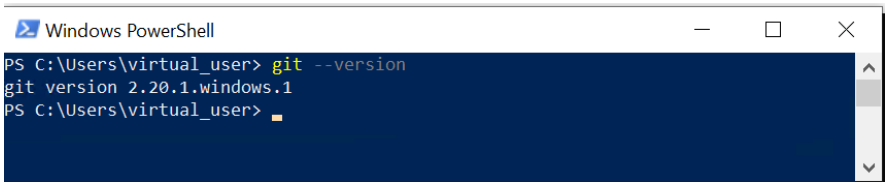
Descargar e instalar desde la siguiente URL: https://gitforwindows.org/

Si quieres instalar Git en MACOSX puedes descargarlo desde aquí: https://git-scm.com/download/mac.

Durante la instalación nos pedirá que seleccionemos un editor de texto. Se recomienda utilizar un editor más moderno como Notepad++, Sublime Text, etc.[5]



Una vez finalizada la instalación tendremos Git habilitado. Podemos comprobarlo desde la consola de Windows (CMD o PowerShell) ejecutando el siguiente comando:



2.4 Instalación de Intellij IDEA Ultimate

Descarga e instala desde la página de JetBrains la versión adecuada a tu SSOO. Desde la siguiente URL puedes descargar la última versión:

https://www.jetbrains.com/idea/download/

Nota: Para obtener una **licencia gratuita** de la versión ultimate tienes que registrarte en la página de JetBrains con la **cuenta de la universidad.**

[5] En la instalación desde MACOSX no nos solicitará que suministremos editor por defecto.

3 Creación del primer Proyecto JEE y vinculación a GitHub

Para trabajar en la asignatura vamos a proceder de la siguiente forma:

1. Primero cada alumno creará un repositorio remoto en GitHub usando el IDGIT suministrado en el Campus Virtual.
2. A continuación, crearemos un proyecto en la tecnología correspondiente (JEE, Spring o Node.js). En el primer proyecto la tecnología será JEE (Java Enterprise Project).
3. En el siguiente paso se creará un repositorio local, se vinculará nuestro proyecto a dicho repositorio local y luego se vinculará el repositorio local al repositorio remoto.
4. Finalmente realizaremos el primer COMMIT y PUSH para subir el proyecto al repositorio remoto.

Vayamos con cada uno de estos pasos.

3.1 Creación del repositorio remoto para cada alumno y Configurar Git en el IDE

Una vez instalado Git en SSOO lo siguiente es configurar el IDE para poder trabajar con nuestros repositorios de código desde el entorno de desarrollo.

Nota: También es posible hacerlo desde la consola de Windows o de MACOSX

Si no tenemos cuenta en GitHub, el primer paso es registrarse y luego crear un nuevo repositorio. Por ejemplo, vamos a crear un **nuevo repositorio** en GitHub que denominaremos **sdi-x-lab-jee** (x = IDGIT del alumno) y que utilizaremos para poder completar esta práctica.

Desde la página Web de **GitHub** buscamos la sección de repositorios y hacemos clic en el botón **New (https://github.com/new).**

¡¡¡¡¡¡MUY IMPORTANTE!!!!!!

Aunque en este guion se ha usado como nombre de repositorio para todo el ejercicio **ServletExample**, cada alumno deberá usar como nombre de repositorio **sdi-x-lab-jee**, donde **x** = columna **IDGIT** en el Documento de ListadoAlumnosSDI-2324.pdf del CV**.**

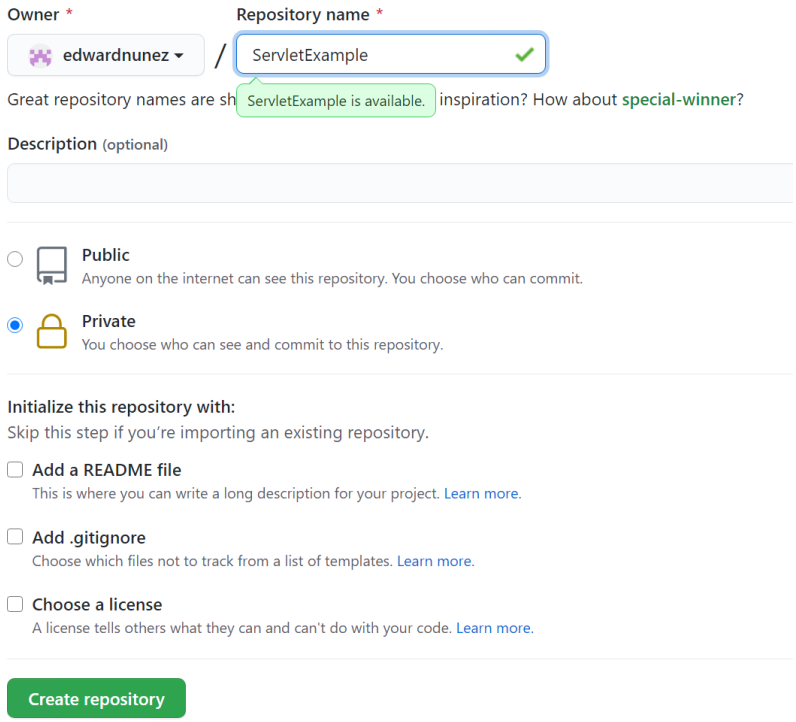
Por ejemplo el alumno con IDGIT=2324-101, deberá crear un repositorio con nombre sdi-2324-101-lab-jee y un proyecto JEE en el IDE con nombre también sdi-2324-101-lab-jee.

En resumen, por ejemplo, para el alumno IDGIT=2324-101:

Nombre repositorio GitHub:	**sdi-2324-101-lab-jee**
Repositorio local:	*una carpeta por cada nuevo respositorio remoto*
Nombre proyecto Java**:**	**sdi-2324-101-lab-jee**
Colaborador invitado:	**sdigithubuniovi**

Otra cuestión **IMPRESCINDIBLE** es que una vez creado el repositorio deberá invitarse como colaborador a dicho repositorio a la cuenta GitHub denominada **sdigithubuniovi.**

Luego especificamos los datos de repositorio similar a como se muestra en la siguiente imagen y hacemos clic en el botón **Create Repository** (**ojo que el nombre no debe ser sdi-x-lab-jee como se ha indicado en el cuadro anterior**). Una vez creado el repositorio en GitHub, debemos configurar el IDE para el acceso a GitHub.

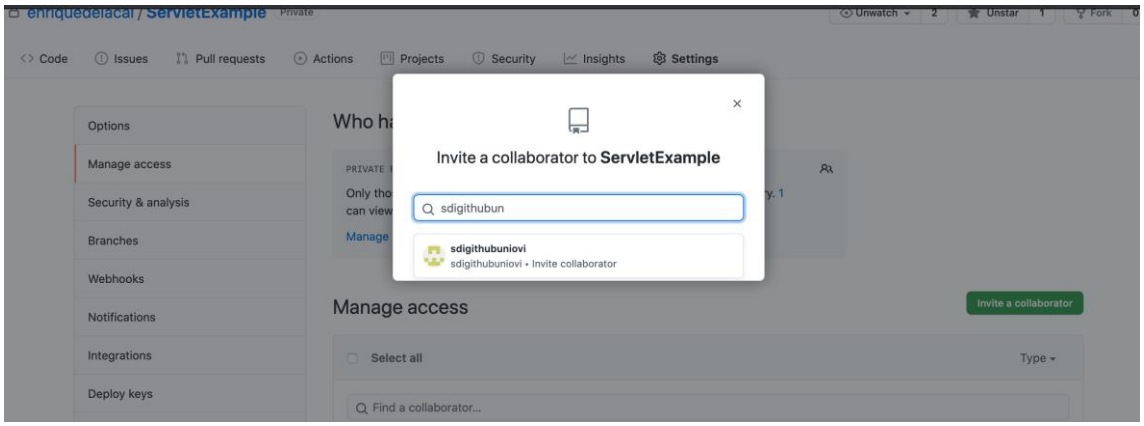


Nota: Creamos el repositorio privado y SIN inicializarlo.

3.1.1 Invitar como colaborador a la cuenta de monitorización de la asignatura SDI

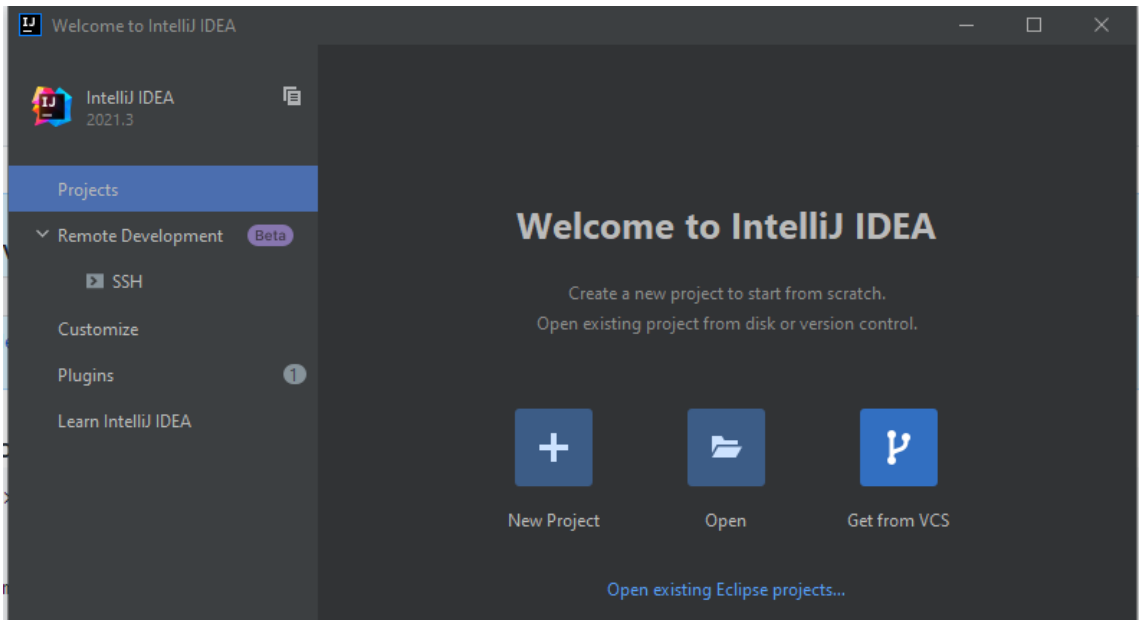
Una vez creado el repositorio, debemos obligatoriamente invitar a nuestro repositorio a la cuenta **sdigithubuniovi.** Desde la opción colaboradores de nuestro repositorio **sdi-x-lab-**

jee (recuerda que en las capturas usamos ServletExample), agregamos al usuario **sdigithubuniovi.**

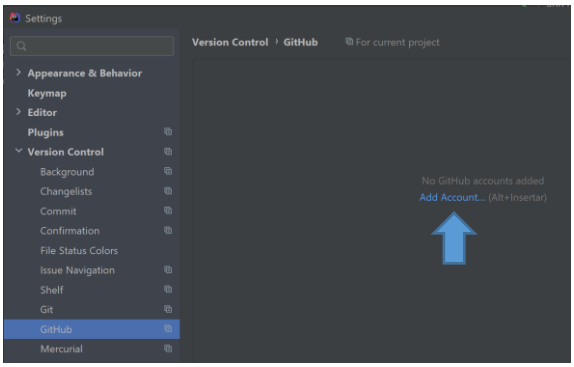


3.1.2 Configuración de las credenciales de GitHUB en Intellij IDE

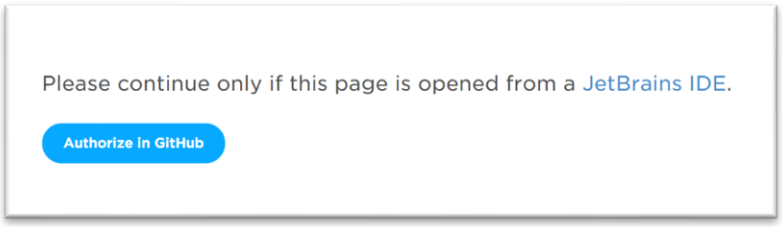
Si es la primera vez que se utiliza Intellij en el equipo, saldrá una ventana desde donde podremos crear un proyecto nuevo, abrir un proyecto existente desde un repositorio local o remoto. Para facilitar la configuración vamos a crear un proyecto nuevo para que se nos abra el entorno de desarrollo (puedes poner cualquier nombre al proyecto).

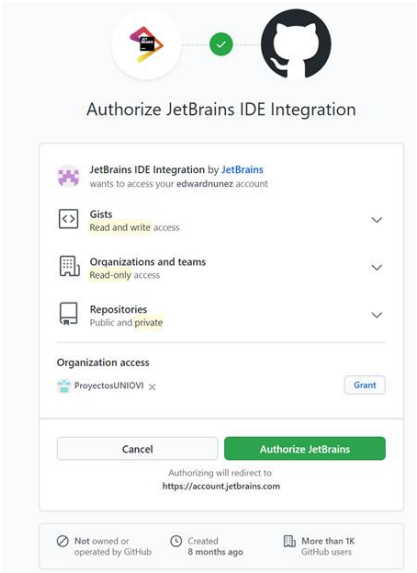


Con el IDE abierto, vamos al menú: **File | Settings | Version Control**, buscamos el submenú **Version Control | GitHub | Add Account** y enlazamos el entorno de desarrollo con nuestra cuenta de GitHub.



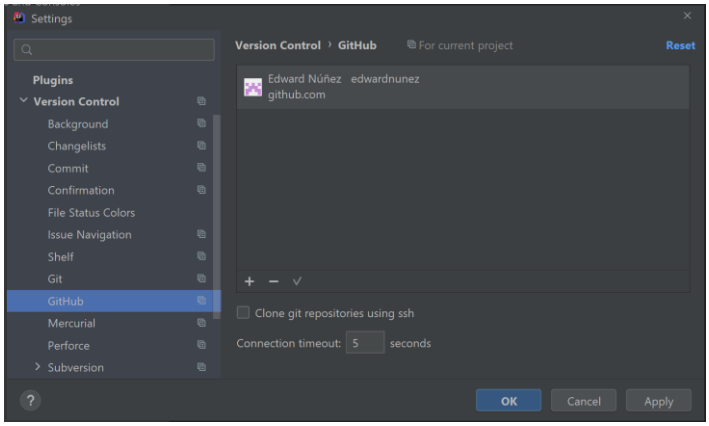
Apareceran varias opciones de autorización (**Log In via GitHub, Log In with token** o **Log In to GitHub Enterprise**). Puedes configurarlo empleando cualquiera de estas opciones. En nuestro caso, hacemos clic en la opción **Log In via GitHub**, abriéndose una página web para autorizar a JetBrains IDE a acceder a la cuenta de GitHub. Una vez autenticados en GitHub, pulsamos en el botón "**Authorize in GitHub**" y seleccionamos los recursos que queremos compartir y el nivel de acceso.





Una vez autorizado saldrá un mensaje, similar al siguiente: ***"You have been successfully authorized in GitHub. You can close the page"***

Ahora en el IDE tendremos la cuenta de GitHub asociada. Finalmente, pulsamos en el botón ***"OK"*** para finalizar el proceso.



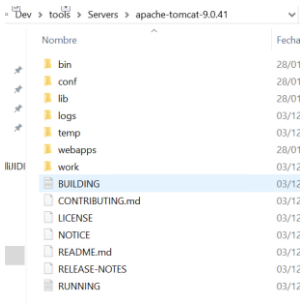
3.1.3 Instalar y configurar Apache Tomcat

IntelliJ IDEA proporciona integración con varios servidores de aplicaciones, lo que permite iniciar y detener servidores locales, conectarse a servidores remotos en ejecución e implementar artefactos en esos servidores. También permite depurar la aplicación directamente desde el IDE. En esta URL[6] de JetBrains, se muestran los servidores de aplicaciones que se pueden integrar en Intellij IDEA.

En este documento utilizaremos Tomcat para desplegar las aplicaciones web empresariales en JEE.

- **Instalación de Apache Tomcat**

Para instalar Tomcat 9 solo hay que descargarlo desde la siguiente URL[7] y descomprimir el contenido en un directorio local, similar a como se muestra en la siguiente imagen:



[6] https://www.jetbrains.com/help/idea/application-servers-support.html

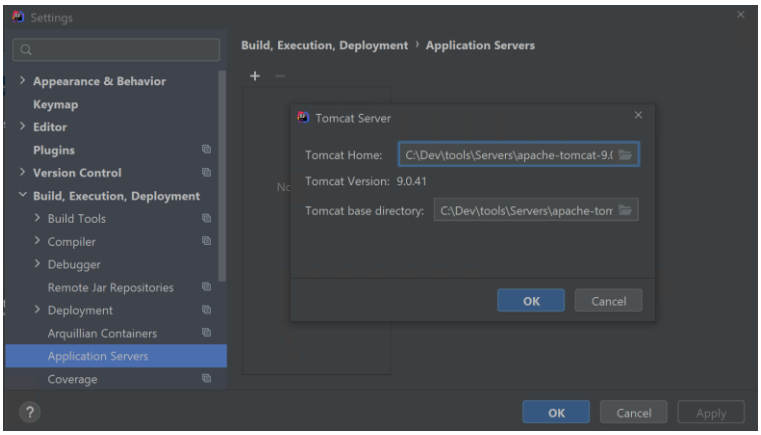
[7] http://tomcat.apache.org/, la versión para OSX descargar y descromprimes la versión Bin/zip.

Y concedes permisos de ejecución al archivo: **chmod a+x /path/apache-tomcat-9.0.71/bin/catalina.sh**

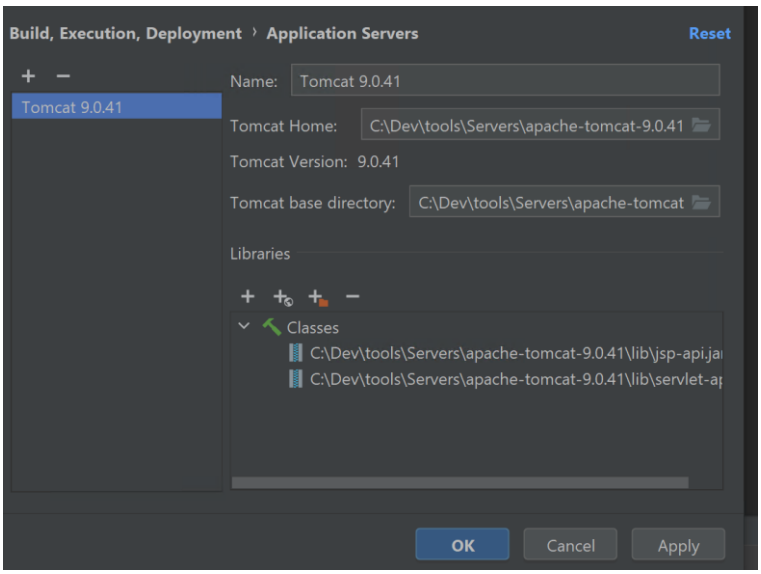
- **Configuración de Apache Tomcat en Intellij IDEA**

IntelliJ IDEA permite desplegar aplicaciones JEE en Tomcat de una forma muy sencilla, especificando la ubicación del servidor. Para configurar Apache Tomcat en Intellij IDEA seguimos los siguientes pasos:

1. Vamos al menú **File|Settings**, luego seleccionamos el submenú **Build, Execution, Deployment|Application Servers**.
2. Pulsamos el botón **+** y seleccionamos la opción ***Tomcat Server.***
3. Especificamos la ruta de la ubicación de instalación de Tomcat. En nuestro caso la ruta donde hemos descomprimido Tomcat anteriormente. Pulsamos en el botón **"OK".**



4. Una vez seleccionada la carpeta aparecerá, en el panel lateral izquierdo, la versión del servidor que hemos añadido, tal como se muestra en la siguiente imagen. Pulsamos el botón "OK".



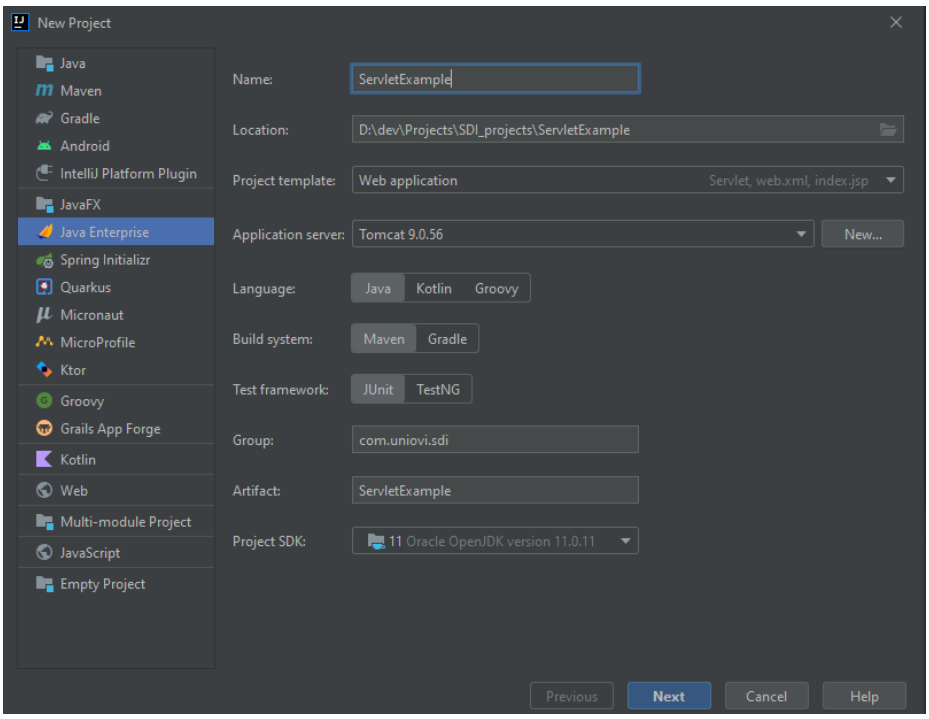
3.2 Creación del primer proyecto JEE (Java Enterprise Project)

Una vez ya tenemos todo configurado, vamos a crear un proyecto JEE en Intellij IDEA totalmente independiente de GitHub, y a continuación lo vincularemos a GitHub.

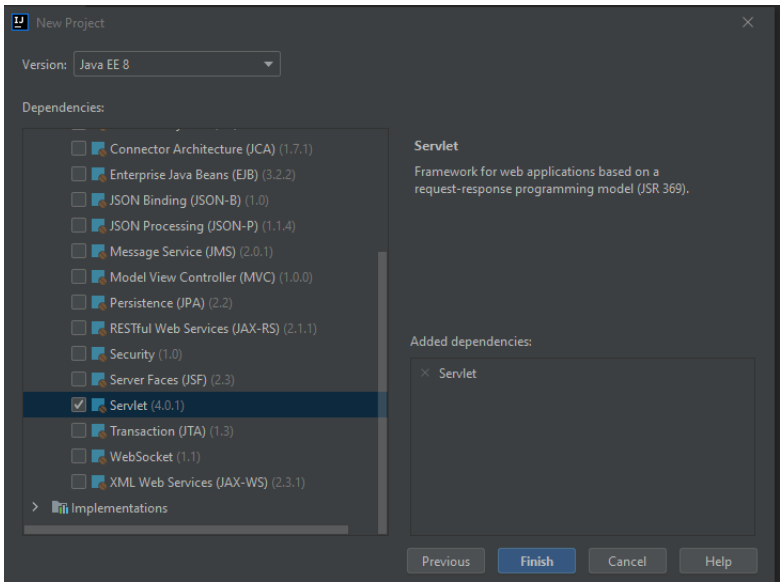
Ahora crea tu proyecto con nombre **sdi-x-lab-jee** (donde x = IDGIT en el listado de IDs de GITs en CV, por simplicidad nosotros usaremos el nombre **ServletExample**), mediante la opción **File|New|Project** y seguimos los siguientes pasos:

Primero, seleccionamos un proyecto de tipo ***Java Enterprise o Jakarta EE (apartir de la versiom 2022.1 de Intellij IDEA)*** y luego los siguientes datos y configuraciones:

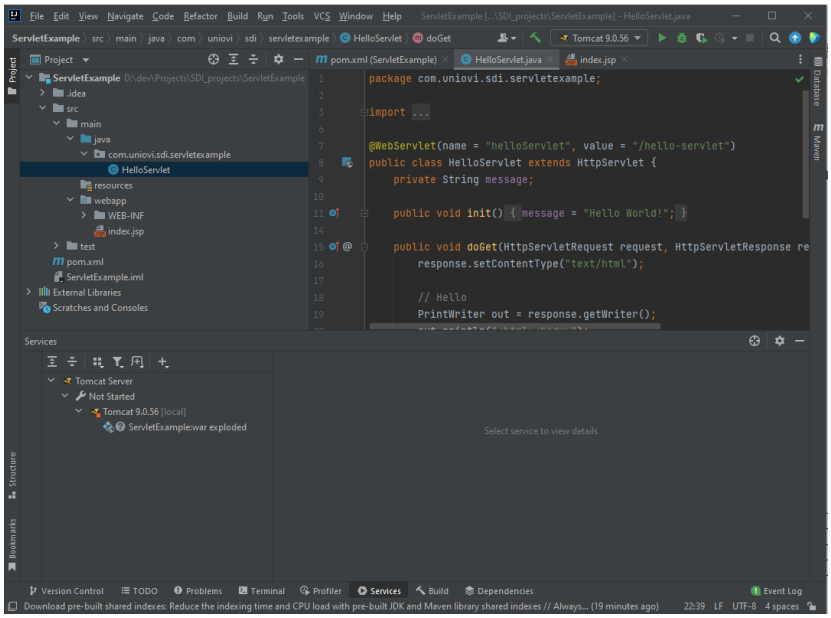
- **Nombre:** sdi-x-lab-jee (donde x = IDGIT en el listado de IDs de GITs en CV).
- **Localización**: ruta donde almacenarás el proyecto.
- **Group (paquete):** com.uniovi.sdi
- **Plantilla**: Web application.
- **Otras configuraciones a elegir**: ***Tomcat*** que es el contenedor de servlet donde vamos a desplegar la aplicación**,** ***Maven*** para gestionar las dependencias del proyecto, ***JUnit*** para el desarrollo de las pruebas unitarias, y la versión del SDK. En la siguiente imagen se muestran las opciones seleccionadas, luego pulsamos en el botón ***"Next"***



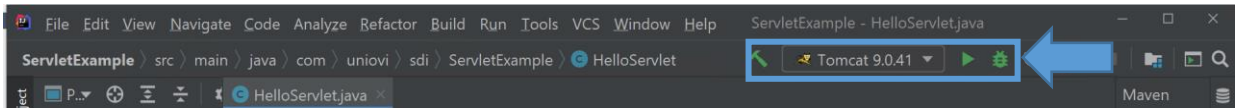
Seleccionamos la versión ***Java EE 8 y la especificación de Servlet*** (vienen seleccionada por defecto) y pulsamos en el botón ***"Finish" o "Create"*** en funcion de la versión del IDE que estemos utilizando.



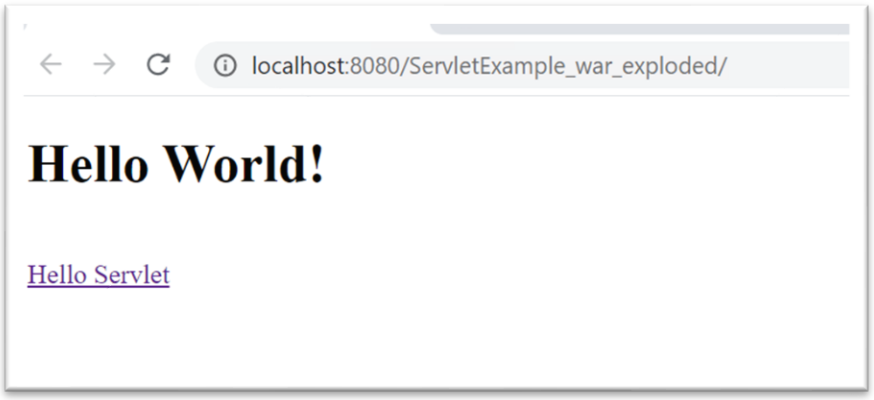
Finalmente, se generará un proyecto Java con una estructura similar a la que se muestra en la siguiente imagen:



Para ejecutar la aplicación y verificar que todo funciona correctamente, seleccionamos el servidor y pulsamos en el botón **Play**, tal como se muestra en la siguente imagen:



Automaticamente abrirá el navegador y mostrará la aplicación, tal como se muestra en la siguiente imagen:



Nota: Si en este punto no podemos desplegar el proyecto porque no lo reconoce como un proyecto Maven entonces hacemos click derecho sobre el fichero *pom.xml* y seleccionamos la opción *"add as maven project" y desplegamos.*

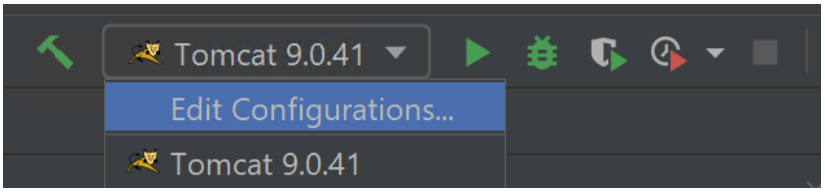
3.2.1 Editar la configuración del servidor de aplicaciones

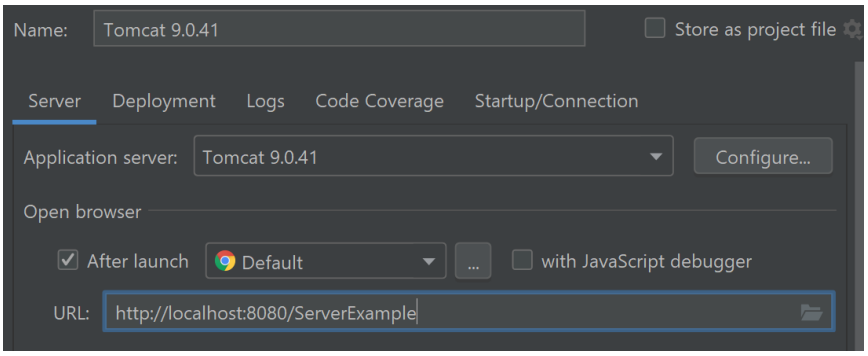
Para poder ejecutar o depurar aplicaciones en un servidor de aplicaciones, es necesaria una configuración de **ejecución/depuración** del servidor en el IDE.

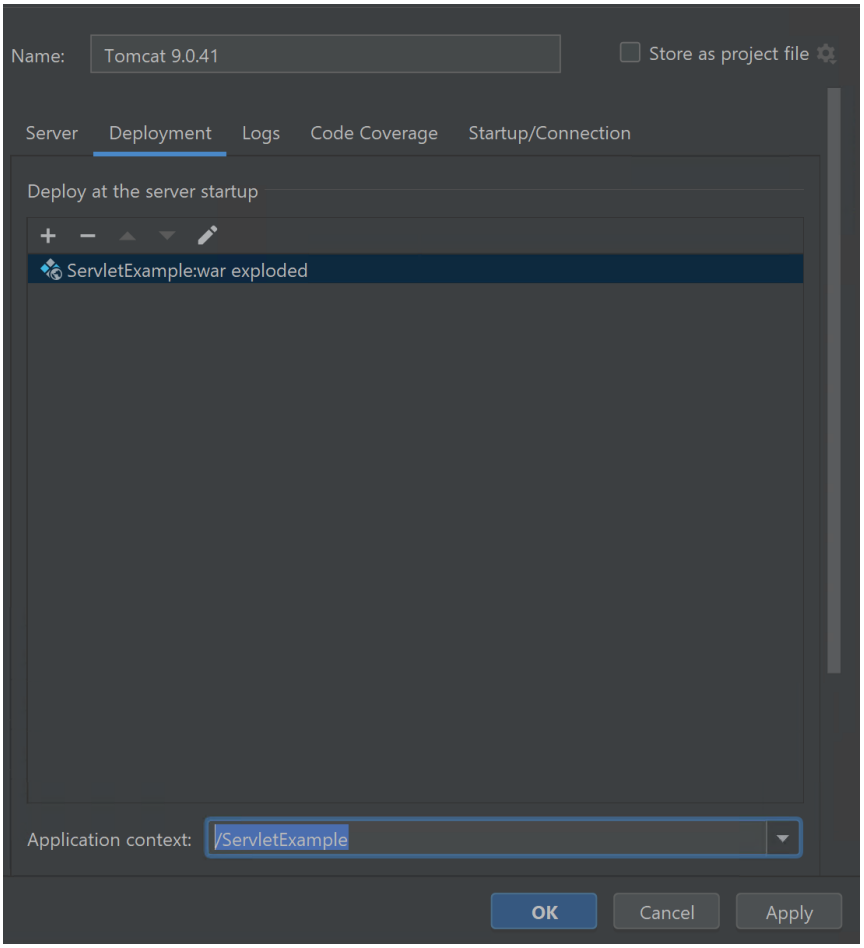
Ahora mismo la aplicación se está desplegando en un contexto diferente al nombre del proyecto: http://localhost:8080/ServletExample_war_exploded/

Para desplegar la aplicación en: http://localhost:8080/ServletExample, tenemos que seguir los siguientes pasos:

1. Desde el menú principal, seleccionamos la opcion **Edit Configurations.**
2. Cambiamos la URL y el contexto de la aplicación a http://localhost:8080/ServletExample.





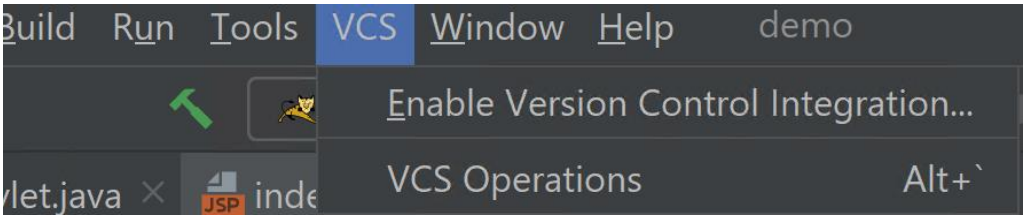


Ahora hacemos un redespligue o paramos y desplegamos de nuevo la aplicación y veremos que la aplicación se desplegará en la nueva URL.

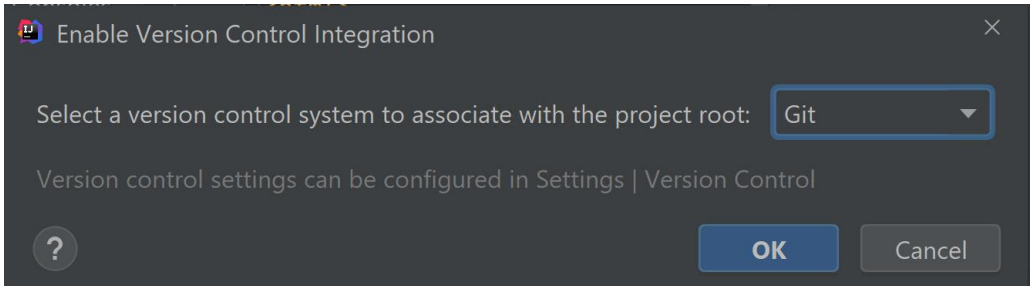
3.3 Vinculación del proyecto con GitHub

Ahora vamos a vincular el proyecto con el repositorio local y remoto para poder ir versionado el proyecto a medida que vayamos añadiendo funcionalidades. A continuación, seguimos los siguientes pasos:

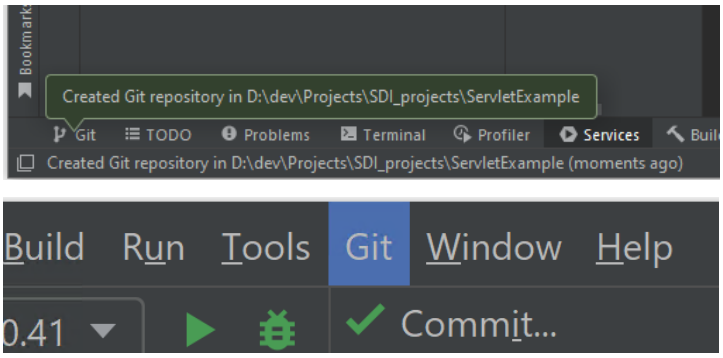
1. Desde el menú principal vamos al menú **VCS|Enable Version Control Integration**



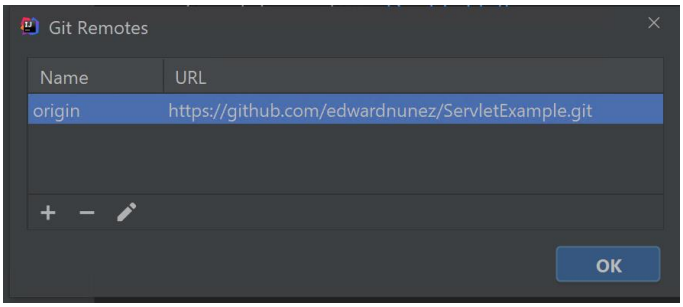
2. Elegimos **Git** como sistema de control de versión y hacemos clic en el botón **OK**



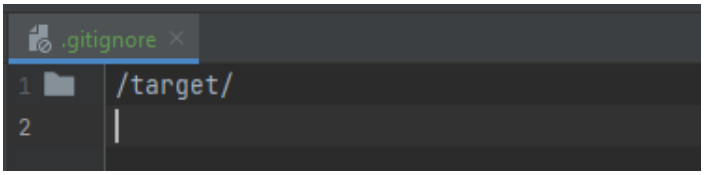
3. Ahora nos saldrá un mensaje de creación de un **repositorio local** y la opción del menú **VCS cambiará Git** en el menú principal.



4. Para vincularlo al **repositorio remoto**, vamos al menú **Git|Manage Remotes** y añadimos la URL del repositorio que hemos creado en GitHub, similar al que se muestra en la siguiente imagen:

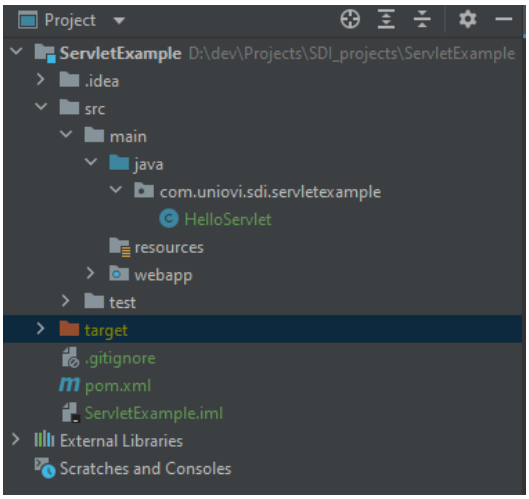


5. Finalmente, añadimos la carpeta target al fichero .gitignore para no subir a GitHub el código compilado, para esto hacemos click derecho sobre la carpeta **target** y luego vamos al menú **Git|Add to .gitignore** (si no aparece esta opción es porque el fichero.gitignore ya existe).



3.4 Hacer el primer commit desde el IDE

Primero, nos colocamos **sobre la carpeta del proyecto**, hacemos clic derecho y luego vamos a la opción **Git|Add** para añadir los nuevos cambios al **repositorio LOCAL**. Los nuevos ficheros subidos al repositorio cambian a **color verde**, como se muestra en la siguiente imagen:



Lo siguiente es hacer **commit y push** para subir los cambios a ambos repositorios (**Local y remoto**). Nos colocamos sobre la carpeta del proyecto, hacemos clic derecho y pulsamos sobre la opción **Git|Commit Directory.** Luego verificamos los cambios añadidos, escribimos el mensaje del commit y finalmente presionamos el botón **Commit and Push.**

La opción "Commit" guardará los datos en el repositorio local solamente (no nos interesa en esta ocasión).

La opción "**Commit and Push"** los enviará también al repositorio remoto, **que es lo que queremos hacer.**

El mensaje de este primer commit deberá llamarse como se indica en la caja que se incluye a continuación (sin las comillas):

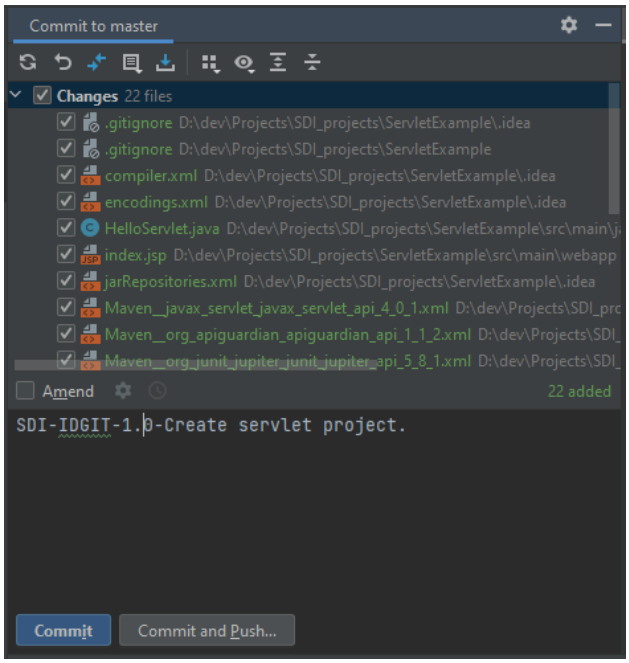
Nota: Subir el código a GitHub en este punto. Incluir el siguiente Commit Message ->

"SDI-IDGIT-1.0-Create servlet project."

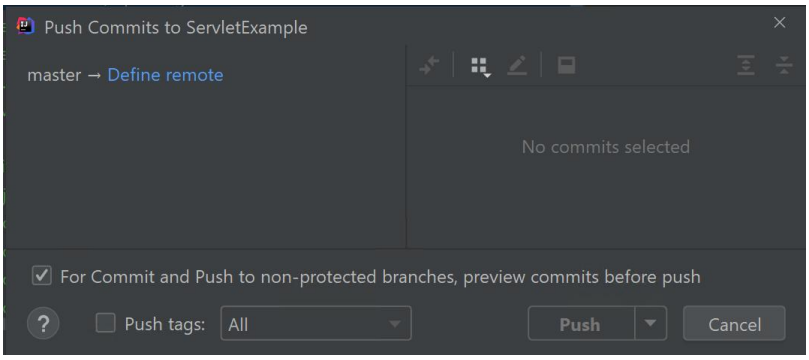
OJO: sustituir IDGIT por tu número asignado (p.e. 2324-101):

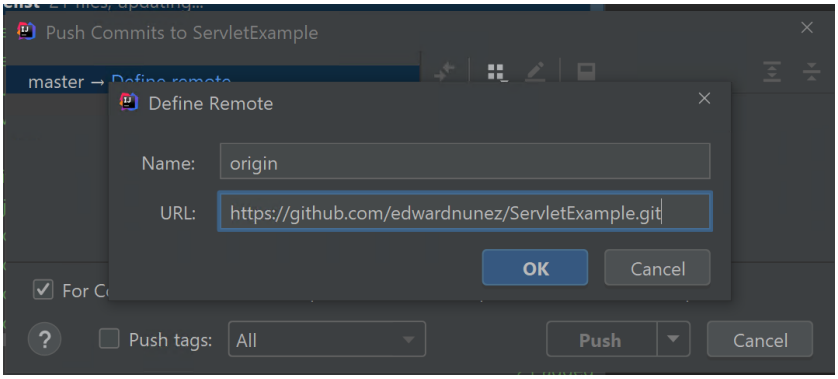
"SDI-2324-101-1.0-Create servlet project."

(*No olvides incluir los guiones y NO incluyas BLANCOS*)

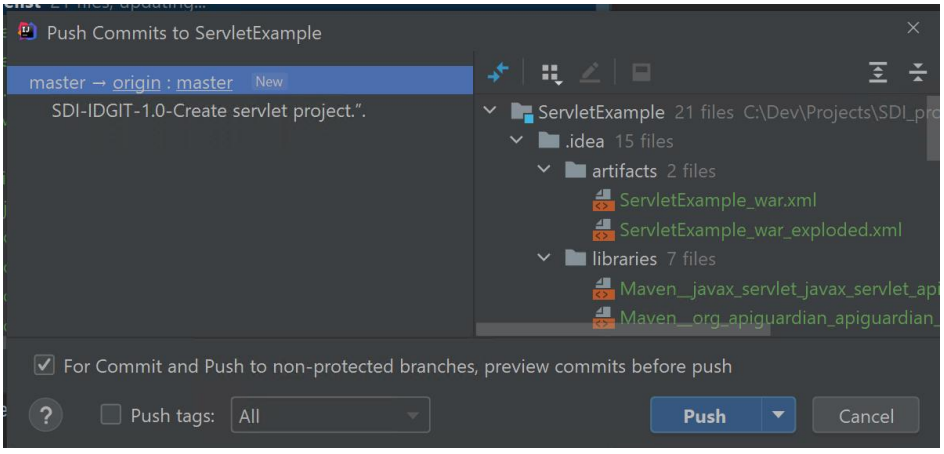


Si no sale la rama remota por defecto al hacer el push, entonces debemos definirla, tal como se muestra en las siguientes imágenes (Click sobre el enlace ***Define remote***):

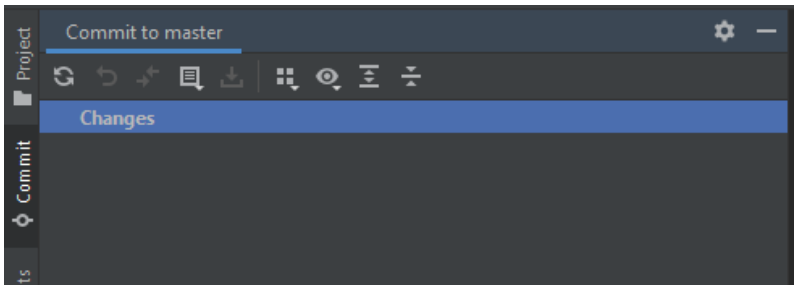




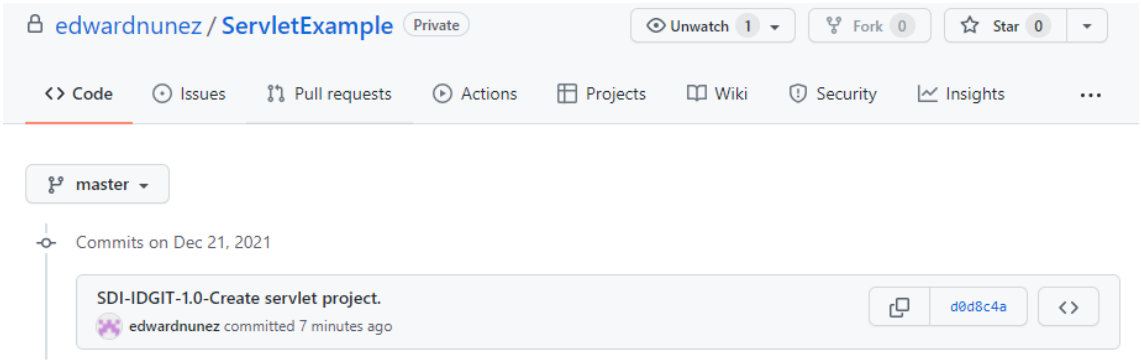
Finalmente, pulsamos el botón ***"Push"*** para subir los cambios al repositorio remoto.



A continuación se puede comprobar que el proyecto ya no muestra cambios pendientes y que se ha hecho una actualización en el **repositorio remoto.**



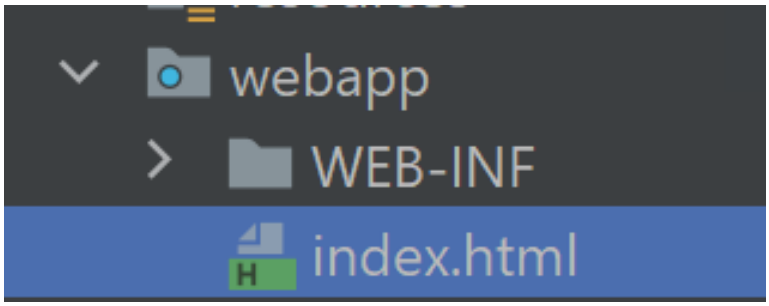
Ahora, vamos a la **web de GitHub** para ver los cambios que hemos subido al repositorio. Para esto, seleccionamos en **Github** el repositorio que hemos creado y hacemos clic en el enlace **"Commits".**



En este punto, ya tenemos el proyecto vinculado a GitHub y lo siguiente es ir añadiendo funcionalidades al proyecto y subiendo los resultados al repositorio paso a paso.

4 Servlets y peticiones HTTP GET y POST

Una vez creado el proyecto Java, vamos a continuar desarrollando el proyecto, lo primero que haremos es renombrar el fichero ***index.jsp*** a ***index.html*** en la carpeta **/webApp/** de nuestro proyecto (botón derecho, **Refactor |Rename**). Incluimos el siguiente contenido:

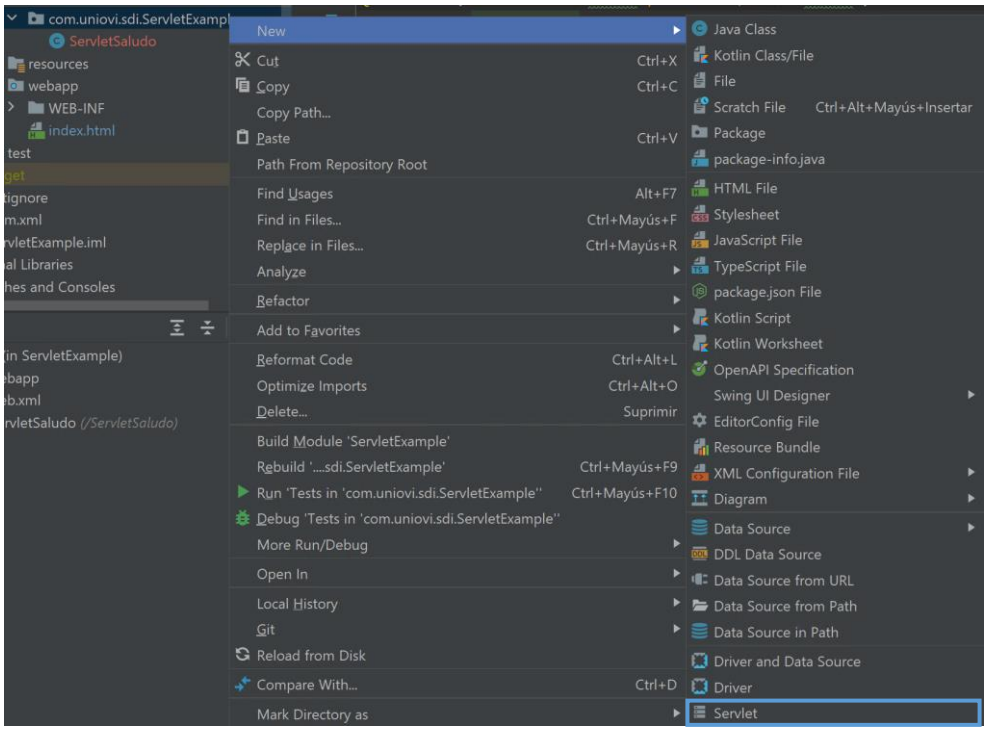


El fichero index contendrá **dos formularios: uno GET y otro POST;** con un parámetro con clave **nombre** que se envía a la URL **GreetingServlet**.

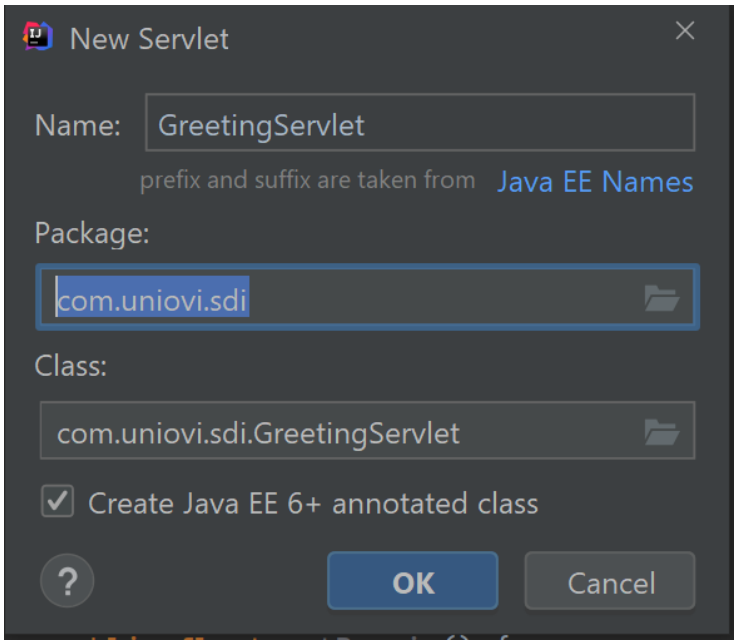
```
<html lang="en">
<head>
  <title>Servlets</title>
  <meta charset="utf-8"/>
  <meta name="viewport" content="width=device-width, initial-scale=1"/>
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</head>

<body>

<!-- Contenido -->
<div class="container" id="contenedor-principal">
  <h2>Formulario GET Saludar</h2>

  <form action="GreetingServlet" method="get">
    <div class="form-group">
      <label for="name-get">Nombre</label>
      <input type="text" class="form-control" name="name" id="name-get">
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
  </form>

  <h2>Formulario POST</h2>

  <form action="GreetingServlet" method="post">
    <div class="form-group">
      <label for="name-post">Nombre</label>
      <input type="text" class="form-control" name="name" id="name-post">
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
  </form>
</div>

</body>
</html>
```

Renombramos el paquete **com.uniovi.sdi.servletexample** a **com.uniovi.sdi** usando la opción del menú **Refactor|Rename**.

Borramos el Servlet HelloServlet y añadimos un nuevo Servlet, para esto hacemos clic derecho sobre el paquete **com.uniovi.sdi** del proyecto y luego en **New|Servlet.** Si no apareciese esta opción se puede crear el servlet como una clase Java normal e incluir el código que veremos a continuación.



Lo almacenaremos en el paquete **com.uniovi.sdi** y con el nombre ***GreetingServlet.***



Abrimos la clase **GreetingServlet**, observamos que implementa los métodos **doGet()** y **doPost()** para responder a peticiones GET y POST respectivamente.

Ahora implementamos una respuesta a **doGet(),** obteniendo el **parámetro name**. Esta función ya es capaz de responder a peticiones GET. Copia y analiza el código que se muestra a continuación:

```java
package com.uniovi.sdi;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;
import java.io.IOException;

@WebServlet(name = "GreetingServlet", value = "/GreetingServlet")
public class GreetingServlet extends HttpServlet {

@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
      ServletException, IOException {

   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
   out.println("<HTML>");
   out.println("<HEAD><TITLE>Hello World!</TITLE></HEAD>");
   out.println("<BODY>");
   String name = request.getParameter("name");
   if (name != null) {
      out.println("Hello " + name + "<br>");
   }
   out.println("</BODY></HTML>");

}
```

La función **doPost()** se encarga de que el servlet responda a peticiones POST. En este caso la función doPost() simplemente llama a función doGet().

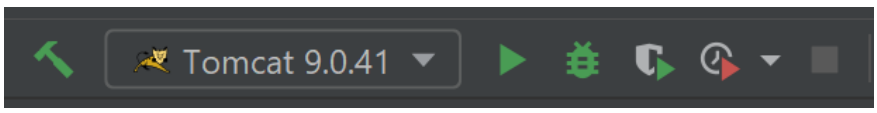
```java
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
      ServletException, IOException {

   doGet(request, response);
}
}
```

La anotación **WebServlet** declarada en el inicio de la clase nos sirve para definir la URL del servlet, en este caso **/GreetingServlet**, podremos enviarle peticiones a esta URL y el servlet responderá tanto a peticiones GET como POST.

```java
@WebServlet(name = "GreetingServlet", value = "/GreetingServlet")
public class GreetingServlet extends HttpServlet {..}
```

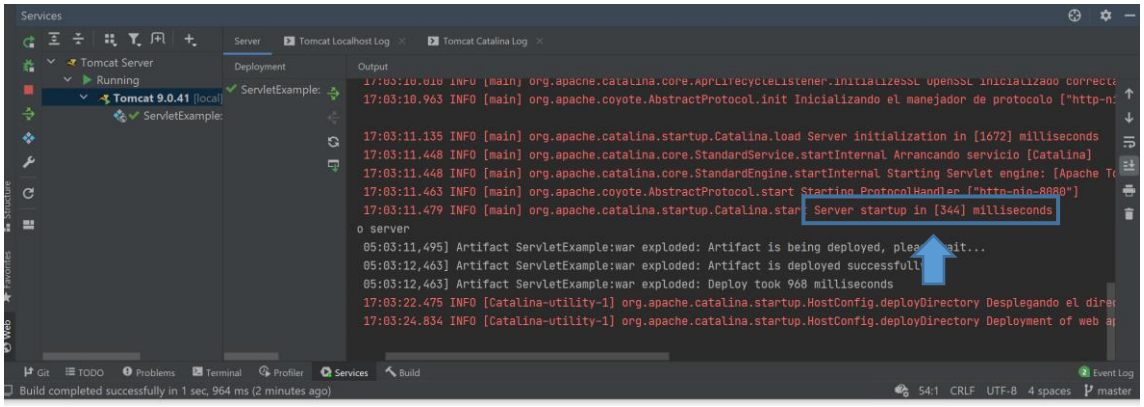
Nota: Hay que importar el paquete **java.io.PrintWriter**.

Nos colocamos sobre el nombre del proyecto, seleccionamos **y desplegamos la aplicación en el servidor Tomcat**

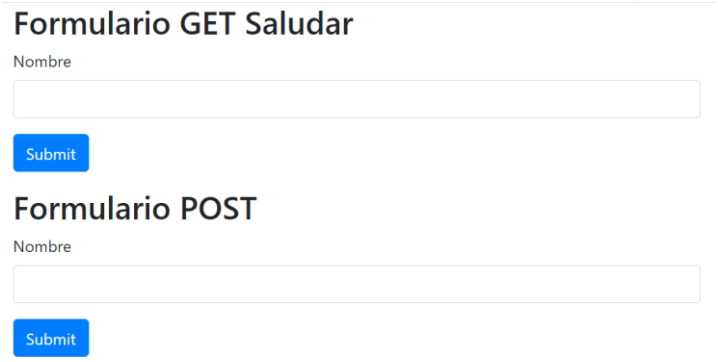


Pulsamos en el botón play para ejecutar el proyecto o en el botón debug para ejecutarlo en modo depuración.

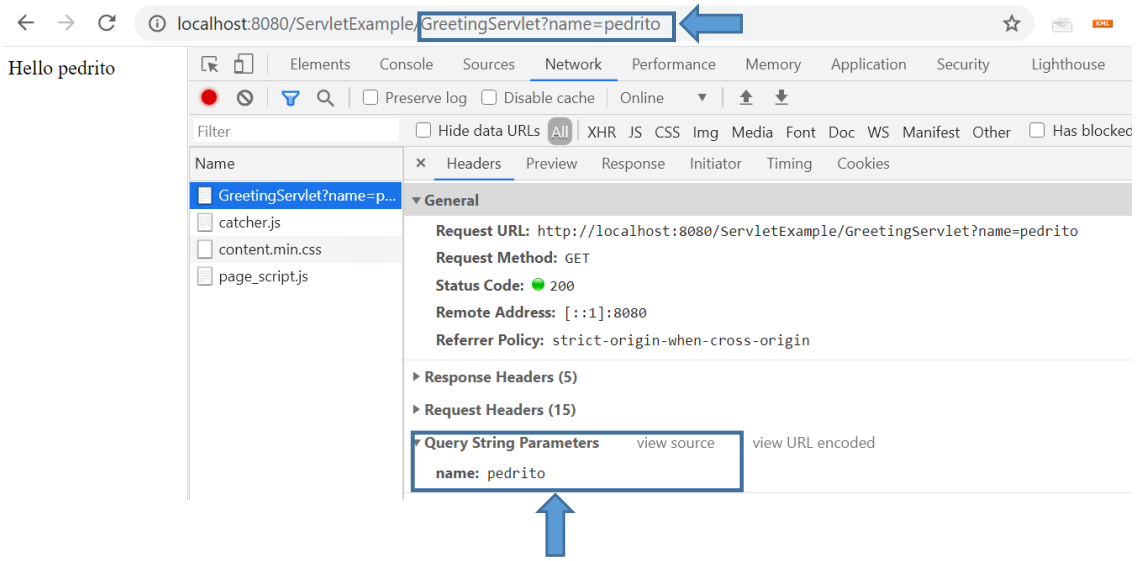
Cuando el servidor esté iniciado veremos un mensaje *"Server startup in [xxx milliseconds]"* en la pestaña **Services**, desde la pestaña **Server** podemos gestionar el servidor.



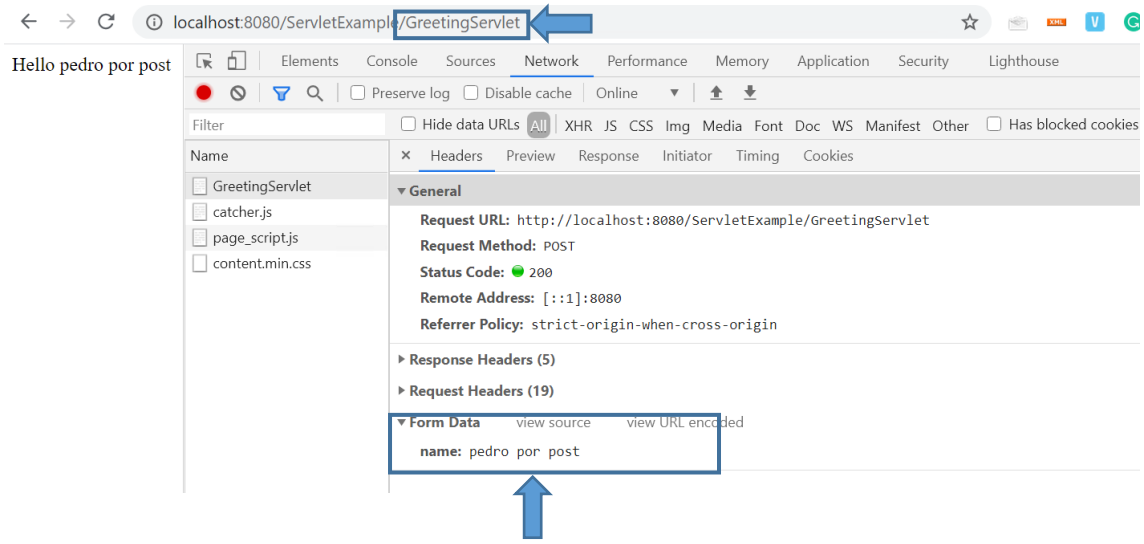
Una vez desplegado se abrirá automáticamente la aplicación desde la siguiente URL en el navegador que tengamos configurado por defecto, por ejemplo, **Chrome:** http://localhost:8080/ServletExample/.



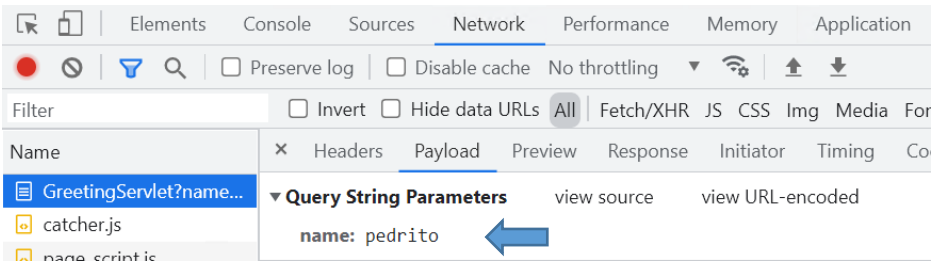
Analizamos las peticiones de ambos formularios con la herramienta **Network del Chrome** (F12 para entrar). Los parámetros GET se envían en la propia URL



En cambio, los de la petición POST se envían en el cuerpo.



Nota: En función de la versión de Chrome la opción de visualizar los parámetros puede aparecer en la pestaña **Payload** en lugar de la pestaña Headers, tal como se muestra en la siguiente imagen



4.1 Los Servlets son multihilo

Cada vez que el servidor accede al servlet, éste se ejecuta. Si se reciben varias peticiones, se ejecutarán de forma simultánea en hilos diferentes. No obstante, ambos hilos se ejecutan sobre la misma instancia del Servlet. Para verificar este funcionamiento podemos incluir un "Sleep" e imprimir la ID del hilo; si realizamos varias peticiones veremos que se trata de hilos diferentes. En el método ***doGet()*** del ***GreetingServlet*** añadimos el siguiente bloque de código.

```
@WebServlet(name = "GreetingServlet", value = "/GreetingServlet")
public class GreetingServlet extends HttpServlet {
   int contador = 0;
   private Object mutex = new Object();

   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
         ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<HTML>");
      out.println("<HEAD><TITLE>Hello World!</TITLE></HEAD>");
      out.println("<BODY>");
      String name = request.getParameter("name");
      if (name != null) {
         out.println("Hello " + name + "<br>");
      }
      out.println("</BODY></HTML>");
      try {
         Thread.sleep(15000);
      } catch (InterruptedException e) {}
      out.println("Thread ID: "+Thread.currentThread().getId()+"<br>");
      //contador es un recurso compartido entre todos los hilos.
      //Bloqueamos el acceso a un único hilo a la vez.
      synchronized (mutex)
      {
         contador++;
         out.println("Visits:"+contador+"<br>");
      }

   }
```

Al estar trabajando en un ambiente multihilo, es probable que diferentes hilos intenten modificar la variable "contador" al mismo tiempo. Mediante synchronized bloqueamos un objeto y aseguramos que ese fragmento de código sea ejecutado por un único hilo a la vez.

Si volvemos a hacer un **Run** sobre la aplicación se desplegará una nueva versión con los cambios realizados. Accedemos desde dos pestañas a la URL http://localhost:8080/ServletExample/GreetingServlet?name=pedro y comprobamos que los servlets se ejecutan en diferentes hilos.

Hello pedro
Thread ID: 29
Visits:3

Nota: Subir el código a GitHub en este punto. Incluir el siguiente Commit Message ->

"SDI-IDGIT-1.1-Los Servlets son multihilo.".

OJO: sustituir IDGIT por tu número asignado (p.e. 2324-101):

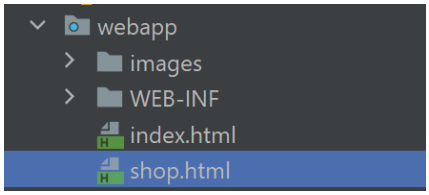
"SDI-2324-101-1.1-Los Servlets son multihilo."

(*No olvides incluir los guiones y NO incluyas BLANCOS*)

4.2 Manejo de sesión

Los servlets proporcionan una solución simple para **el seguimiento de sesiones de los usuarios** basada en la clase **HttpSession** de la API JEE. Empleando esta clase podremos identificar de manera diferenciada las sesiones de usuarios y tener la posibilidad de guardar información (objetos) asociada a cada usuario de forma independiente.

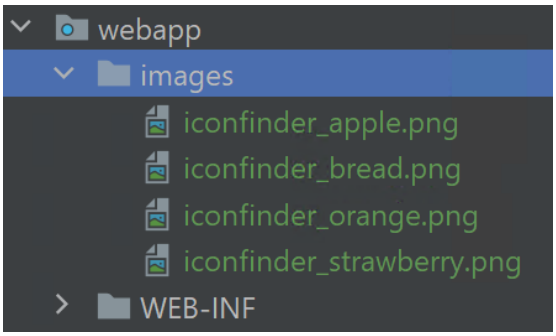
Para ilustrar el uso de la clase HttpSession vamos a ver un pequeño ejemplo consistente en la típica tienda de la compra. A continuación, creamos un nuevo fichero **shop.html** en la carpeta **/WebApp** agregando el siguiente contenido:




```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Servlets</title>
   <meta charset="utf-8"/>
   <meta name="viewport" content="width=device-width, initial-scale=1"/>
   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
   <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
   <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</head>

<body>

<!-- Contenido -->
<div class="container" id="main-container">
   <h2>Productos</h2>
   <div class="row ">
      <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
         <div>
            <img src="images/iconfinder_apple.png"/>
            <div>Manzanas</div>
            <a href="AddToShoppingCart?product=apple" class="btn btn-default">
               2.05 €
            </a>
```

```
      </div>
    </div>
    <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
      <div>
        <img src="images/iconfinder_strawberry.png"/>
        <div>Fresas</div>
        <a href="AddToShoppingCart?product=strawberry" class="btn btn-default">
          2.20 €
        </a>
      </div>
    </div>
    <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
      <div>
        <img src="images/iconfinder_orange.png"/>
        <div>Naranjas</div>
        <a href="AddToShoppingCart?product=orange" class="btn btn-default">
          2.10 €
        </a>
      </div>
    </div>
    <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
      <div>
        <img src="./images/iconfinder_bread.png"/>
        <div>Pan</div>
        <a href="AddToShoppingCart?product=breadd" class="btn btn-default">
          0.80 €
        </a>
      </div>
    </div>
  </div>
</div>
</body>
</html>
```

El siguiente paso es crear una carpeta **images** para incluir las imágenes de la aplicación. Copiamos las imágenes, disponibles en el fichero **PL-SDI-Material1.zip** que hemos compartido, en la carpeta **webapp/images** del proyecto.



Cada producto lanza una petición **/addToShoppingCart** (de tipo GET ya que se trata de un enlace) enviándole un parámetro de clave producto con el identificador único del

producto. En la siguiente imagen se muestra un elemento HTML que será uno de los enlaces (URL) definidos en el ficherlo shop.html.

```
<a href="AddToShoppingCart?product=orange" class="btn btn-default">
    2.10 €
</a>
```

Para recibir esta petición desde la URL indicada, vamos a crear un nuevo Servlet, **ServletShoppingCart**, al que le asociaremos la URL "***AddToShoppingCart".***

```
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.HashMap;
import java.util.Map;

@WebServlet(name = "ServletShoppingCart", value = "/AddToShoppingCart")
public class ServletShoppingCart extends HttpServlet {
```

Implementamos la función **doGet() en donde seguiremos los siguientes pasos:**

- Obtenemos el carrito guardado en sesión.
- Si no hay carrito - se trata de un nuevo usuario, instanciamos un nuevo carrito y lo insertamos en sesión. El carrito será un HashMap<String, Integer> donde guardaremos como clave (String) el nombre del producto y como valor (Integer) el número de unidades compradas.

Insertamos el producto dentro del carrito y mostramos el contenido del carrito.

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
      ServletException, IOException {
   HttpSession session = request.getSession();

   HashMap<String, Integer> cart =
         (HashMap<String, Integer>) session.getAttribute("cart");

   // No hay carrito, creamos uno y lo insertamos en sesión
   if (cart == null) {
      cart = new HashMap<String, Integer>();
      session.setAttribute("cart", cart);
   }

   String product = request.getParameter("product");
   if (product != null) {
      addToShoppingCart(cart, product);
   }
```

```
  response.setCharacterEncoding("UTF-8");
  response.setContentType("text/html");
  PrintWriter out = response.getWriter();
  out.println("<HTML>");
  out.println("<HEAD><TITLE>Tienda SDI: Cesta de la compra</TITLE></HEAD>");
  out.println("<BODY>");
  out.println(shoppingCartToHtml(cart) + "<br>");
  out.println("<a href=\"shop.html\">Volver</a></BODY></HTML>");
}
```

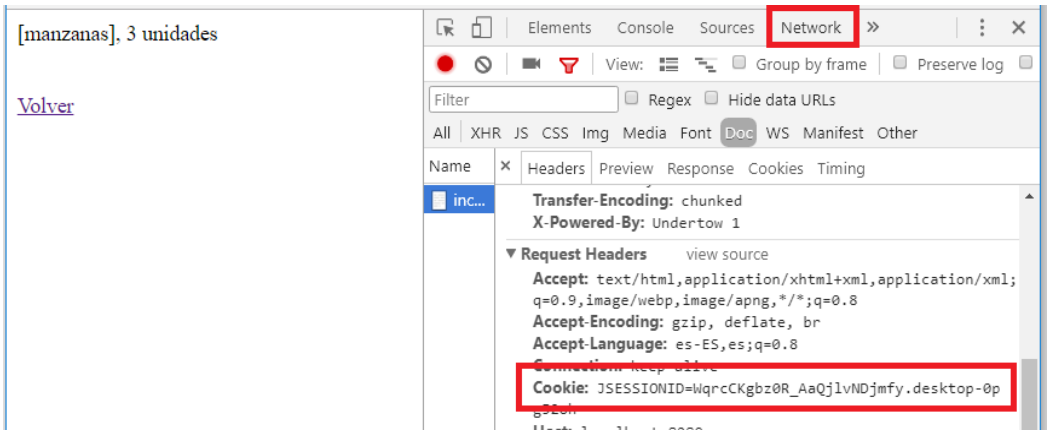
Escribe y analiza los siguientes métodos auxiliares que completan el servlet:

```
  private void addToShoppingCart(Map<String, Integer> cart, String productKey) {
    if (cart.get(productKey) == null) cart.put(productKey, 1);
    else {
      int productCount = cart.get(productKey);
      cart.put(productKey, productCount + 1);
    }
  }

  private String shoppingCartToHtml(Map<String, Integer> cart) {
    StringBuilder shoppingCartToHtml = new StringBuilder();

    for (String key : cart.keySet())
      shoppingCartToHtml.append("<p>[").append(key).append("], ").append(cart.get(key)).append("
unidades</p>");

    return shoppingCartToHtml.toString();
  }
}
```

Ejecutamos la aplicación y comprobamos el correcto funcionamiento del carrito (accedemos a la web desde **dos navegadores diferentes**)

http://localhost:8080/ServletExample/shop.html

Desde el analizador de peticiones de Chrome comprobamos que todas las peticiones contienen el ID de sesión que identifica al usuario.



En el código anterior se pueden producir **problemas de sincronización** entre diferentes hilos correspondientes a peticiones del mismo usuario (realizadas desde varias instancias del mismo navegador) en la manipulación de los objetos que se extraen, se actualizan y se insertan en el objeto sesión. Es necesario determinar qué problemas son estos y solucionarlos sincronizando las porciones de código problemáticas utilizando o bien:

```
synchronized(session) { … }
```

o bien una estructura sincronizada tal como:

```
ConcurrentHashMap o SyncronizedMap[8]
```

Teniendo en cuenta que las porciones de código sincronizadas deben tener el tamaño mínimo (sincronización de granularidad lo más fina posible) imprescindible para maximizar el rendimiento de la aplicación.

Nota: no es necesario sincronizar las sesiones en esta práctica.

Nota: Subir el código a GitHub en este punto. Incluir el siguiente Commit Message ->

"SDI-IDGIT-1.2 Manejo de sesiones."

OJO: sustituir IDGIT por tu número asignado (p.e. 2324-101):

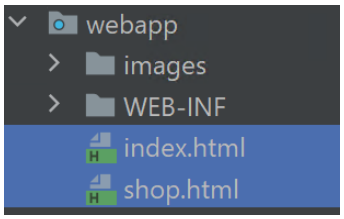
"SDI-2324-101-1.2-Manejo de sesiones."

(*No olvides incluir los guiones y NO incluyas BLANCOS*)

[8] https://dzone.com/articles/java-7-hashmap-vs

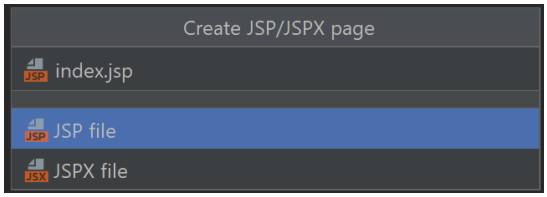
5 JSP Java Server Pages

Ahora vamos a implementar la misma idea que hemos realizado con Servlets pero con JSPs. Sobre el mismo proyecto, eliminamos o renombramos los ficheros **index.html** y **shop.html** del directorio de **WebApp**



A continuación, creamos un nuevo fichero JSP al que llamaremos **index.jsp**. Hacemos clic derecho sobre la carpeta **webapp** y luego sobre el submenú **New|JSP/JSPX**

Llamaremos al fichero **index.jsp** y se guardará dentro de la carpeta **/webapp**



Abrimos el fichero **index.jsp** y copiamos el siguiente contenido HTML. Es igual al que utilizamos en la Web anterior, pero cuenta con la directiva JSP en la cual hemos especificado utf-8 como formato (por defecto al crear ficheros JSP utiliza otro formato), y las peticiones se realizan contra el servlet **ServletShoppingCart**.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" pageEncoding="utf-8"%>

<html lang="en">
<head>
  <title>Servlets</title>
  <meta charset="utf-8"/>
  <meta name="viewport" content="width=device-width, initial-scale=1"/>
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</head>
<body>

<!-- Contenido -->
<div class="container" id="main-container">
  <h2>Productos</h2>
  <div class="row ">
    <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
      <div>
        <img src="images/iconfinder_apple.png" alt=""/>
        <div>Manzanas</div>
```

```
          <a href="AddToShoppingCart?product=apple" class="btn btn-default">
            2.05 €
          </a>
        </div>
      </div>
      <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
        <div>
          <img src="images/iconfinder_strawberry.png" alt=""/>
          <div>Fresas</div>
          <a href="AddToShoppingCart?product=strawberry" class="btn btn-default">
            2.20 €
          </a>
        </div>
      </div>
      <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
        <div>
          <img src="images/iconfinder_orange.png" alt=""/>
          <div>Naranjas</div>
          <a href="AddToShoppingCart?product=orange" class="btn btn-default">
            2.10 €
          </a>
        </div>
      </div>
      <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
        <div>
          <img src="./images/iconfinder_bread.png" alt=""/>
          <div>Pan</div>
          <a href="AddToShoppingCart?product=breadd" class="btn btn-default">
            0.80 €
          </a>
        </div>
      </div>
    </div>
  </div>
</body>
</html>
```

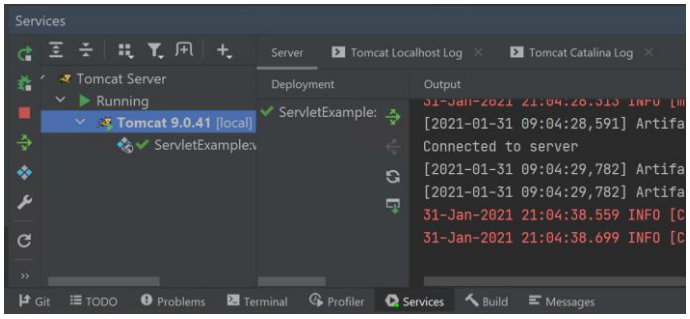
Modificamos la redirección final de la función **doGet()** en ***ServletShoppingCart*** para que nos devuelva a **index.jsp** .

```
out.println(shoppingCartToHtml(cart) + "<br>");
//out.println("<a href=\"shop.html\">Volver</a></BODY></HTML>");
out.println("<a href=\"index.jsp\">Volver</a></BODY></HTML>");
```

Desplegamos de nuevo la aplicación y accedemos a http://localhost:8080/ServletExample/ o http://localhost:8080/ServletExample/index.jsp para probar la modificación.

Nota: Si se sigue cargando la versión anterior debido a la caché del navegador, actualiza la página (Control + F5) o abre una "Ventana de incógnito"

Desplegamos la aplicación y comprobamos que el funcionamiento es correcto. Se recomienda utilizar más de un navegador para probar diferentes sesiones. Podemos restablecer todas las sesiones parando el servidor y volviendo a arrancarlo desde la pestaña Services. La sesión también se puede hacer expirar desde código o eliminando la cookie correspondiente a la sesión en la opción "Privacidad" del navegador)



Nota: Subir el código a GitHub en este punto. Incluir el siguiente Commit Message ->

"SDI-IDGIT-1.3-JSP Java Server Pages."

OJO: sustituir IDGIT por tu número asignado (p.e. 2324-101):

"SDI-2324-101-1.3-JSP Java Server Pages."

(*No olvides incluir los guiones y NO incluyas BLANCOS*)

5.1 Autenticación y manejo de sesión

Un uso muy común de la sesión es la portabilidad de datos de un usuario de una página a otra. Un ejemplo típico es la identificación de usuarios en diferentes páginas.

Para crear un sistema de autenticación, creamos un nuevo fichero **login.jsp**, donde vamos a implementar un formulario que solicite un nombre y password al usuario.

```
<%@ page language="java" contentType="text/html; charset=utf-8" pageEncoding="utf-8" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html lang="en">
<head>
   <title>Servlets</title>
   <meta charset="utf-8" name="viewport" content="width=device-width, initial-scale=1"/>
   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
   <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"
type="application/javascript"></script>
   <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"
type="application/javascript"></script>
```

```
</head>
<body>
<!-- Contenido -->
<div class="container" id="main-container">
  <h2>Identificación de usuario</h2>

  <form class="form-horizontal" method="post" action="login.jsp">
    <div class="form-group">
      <label class="control-label col-sm-2" for="name">Nombre:</label>
      <div class="col-sm-10">
        <input type="text" class="form-control" id="name" name="name" required="true"/>
      </div>
    </div>
    <div class="form-group">
      <label class="control-label col-sm-2" for="password">Password:</label>
      <div class="col-sm-10">
        <input type="password" class="form-control" id="password" name="password"
               required="true"/>
      </div>
    </div>
    <div class="form-group">
      <div class="col-sm-offset-2 col-sm-10">
        <button type="submit" class="btn btn-primary">Agregar</button>
      </div>
    </div>
  </form>
</div>
</body>
</html>
```

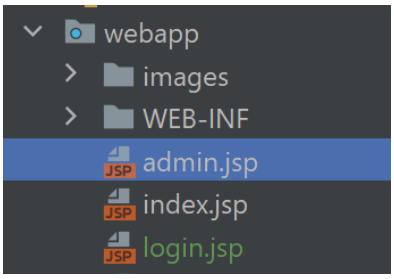
Utilizando las etiquetas **<% %>** podemos introducir código Java en la JSP. Como el formulario se envía contra la propia página **login.jsp** comprobamos si los parámetros **nombre** y **password** coinciden con ***"admin"*** y, en ese caso, introducimos un atributo en sesión con la clave **usuario**.

Analiza y escribe al principio del bloque HTML <body>, el siguiente código java:

```
<%
    String name = request.getParameter("name");
    String password = request.getParameter("password");
    //Nota de seguridad: Esto no se puede hacer NUNCA en una aplicación real
    if (name != null && name.equals("admin") &&
            password != null && password.equals("admin")) {

        // Credencial valido, lo guardo en sesión
        request.getSession().setAttribute("user", "admin");
        response.sendRedirect("admin.jsp");
    } else {
        // Credencial invalido, lo elimino de sesion (opcional)
        request.getSession().setAttribute("user", null);
    }
%>
<!-- Contenido -->
```

Cuando el usuario se identifica correctamente lo enviamos a **admin.jsp** (response.sendRedirect("admin.jsp")).

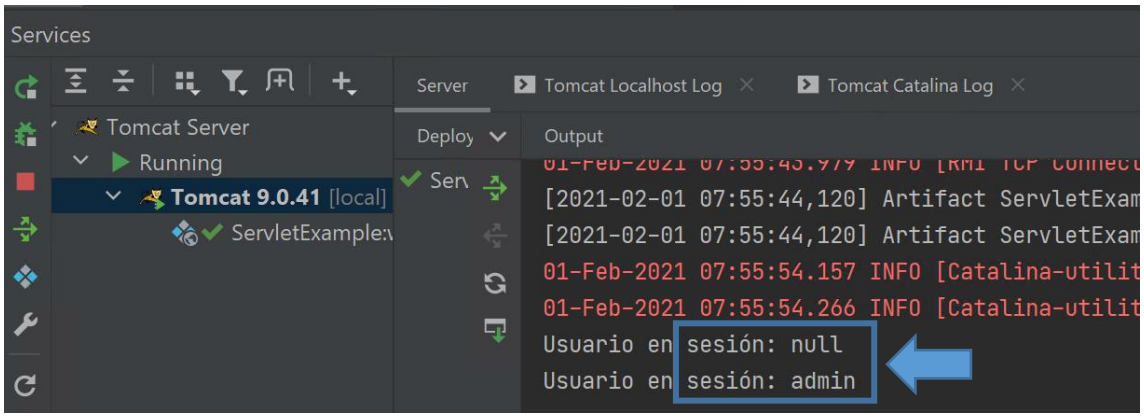
A continuación, vamos a crear un fichero **admin.jsp**, donde vamos a comprobar que la sesión tiene un atributo usuario con valor admin. Si no es así, devolvemos al usuario al **login.jsp**.




```
<%@ page language="java" contentType="text/html; charset=utf-8" pageEncoding="utf-8" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html lang="en">
<head>
  <title>Página de administración</title>
  <meta charset="utf-8" name="viewport" content="width=device-width, initial-scale=1"/>
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"
type="application/javascript"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"
type="application/javascript"></script>
</head>

<body>
<%
  String user = (String) request.getSession().getAttribute("user");
  System.out.println("Usuario en sesión: " + user);
  if (user == null || !user.equals("admin")) {
    // No hay usuario o no es admin
    response.sendRedirect("login.jsp");
  }
%>
<!-- Contenido -->
<div class="container" id="contenedor-principal">
  <h2>Administrar</h2>
</div>
</body>
</html>
```

Ejecutamos la aplicación y probamos a acceder directamente a http://localhost:8080/ServletExample/admin.jsp que debería redireccionarnos a la página de administración. Al no estar autenticados, nos redireccionará al login.

En cambio, si entramos en http://localhost:8080/ServletExample/login.jsp y nos identificamos correctamente se guardará un usuario en sesión y nos dejará acceder a **admin.jsp** sin problemas. La salida por consola nos muestra el usuario identificado.



Nota: NUNCA debemos hardcodear las credenciales de un usuario en una aplicación real, ya que esto sería un ***problema GRAVE de seguridad***.

Nota: Subir el código a GitHub en este punto. Incluir el siguiente Commit Message ->

"SDI-IDGIT-1.4-Autenticación y manejo de sesión."

OJO: sustituir IDGIT por tu número asignado (p.e. 2324-101):

"SDI-2324-101-1.4-Autenticación y manejo de sesiones."

(*No olvides incluir los guiones y NO incluyas BLANCOS*)

5.2 Contexto de la aplicación

Vamos a incluir un contador que muestre el número de visitas totales. Para ello, utilizaremos la variable **application** (nos permite compartir datos en la aplicación con todos los usuarios).

Funciona de forma muy similar a la sesión, ya que para gestionar los atributos utiliza los métodos: **application.getAttribute (clave)** y **application.setAttribute(clave,valor).**

Incluimos el siguiente fragmento en **index.jsp** antes de la definición del . Este fragmento incluye una barra de navegación superior y el codigo java que va a gestionar el contador.

```
<%
   Integer counter = (Integer) application.getAttribute("counter");
   if (counter == null) {
      counter = 0;
   }
   application.setAttribute("counter", counter + 1);
%>

<!-- Barra de Navegación superior -->
<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
   <div class="collapse navbar-collapse" id="my-navbarColor02">
      <ul class="navbar-nav mr-auto">
         <li class="nav-item ">
            <a class="nav-link" href="AddToShoppingCart">Carrito<span class="sr-only">(current)</span></a>
         </li>
         <li class="nav-item ">
            <a class="nav-link" href="login.jsp">Login<span class="sr-only">(current)</span></a>
         </li>
         <li class="nav-item ">
            <a class="nav-link" href="admin.jsp">Administrar productos<span class="sr-only">(current)</span></a>
         </li>
      </ul>
      <div class="nav navbar-right">
         <%=counter%> Visitas
      </div>
    </div>
</nav>
<!-- Contenido -->
```

Ejecutamos la aplicación y comprobamos (desde varios navegadores) que el contador funciona correctamente.

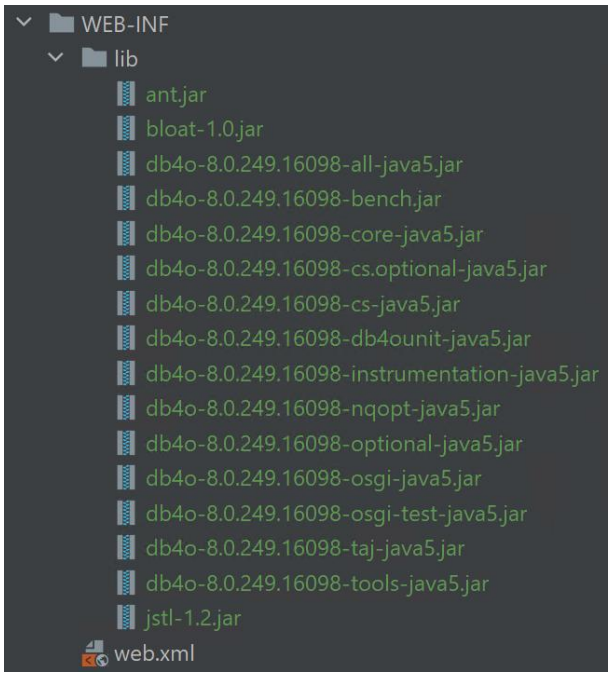
5.3 Listado dinámico de productos

En lugar de listar los productos con código HTML estático, vamos a obtenerlos de una base de datos e insertarlos dinámicamente en el código HTML de la página **index.jsp**.

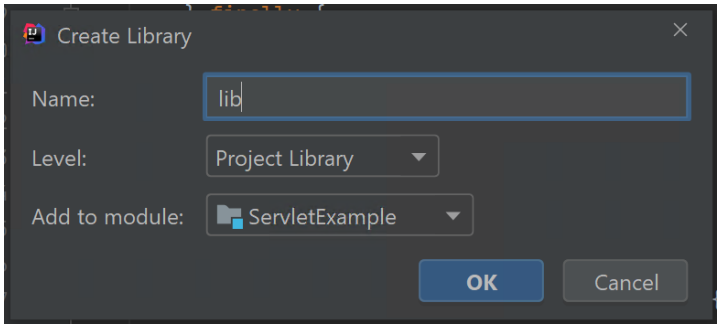
Para poder trabajar con objetos de tipo producto, creamos la clase **Product** en el paquete **com.uniovi.sdi.**


```java
package com.uniovi.sdi;

public class Product {
   private String name;
   private String image;
   private float price;

   public Product(String name, String image, float price) {
      this.name = name;
      this.image = image;
      this.price = price;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getImage() {
      return image;
   }

   public void setImage(String image) {
      this.image = image;
   }

   public float getPrice() {
      return price;
   }

   public void setPrice(float price) {
      this.price = price;
   }
}
```

Para poder trabajar con base de datos desde la aplicación web necesitamos un conjunto de librerías. Extraemos el fichero **PL-SDI-Material1.zip y copiamos todos los archivos jar** en el directorio **/webapp/WEB-INF/lib/**



Añadimos la carpeta como librería al proyecto, para esto hacemos clic derecho sobre la carpeta ***lib*** y luego vamos al menu ***Add as Library…*** y pulsamos el botón OK.



Creamos la clase **ProductsService** y dentro de ella dos métodos: uno para devolver una lista con todos los productos en la base de datos (**getProducts**) y otro para agregar un nuevo producto (**setNewProduct**). Usamos funciones con nombres **get/set** para que en el futuro puedan ser utilizadas desde un JSP Bean.

```
package com.uniovi.sdi;

import java.util.LinkedList;
import java.util.List;
import com.db4o.Db4oEmbedded;
import com.db4o.ObjectContainer;

public class ProductsService {
   public List<Product> getProducts() {
      List<Product> products = new LinkedList<Product>();
      ObjectContainer db = null;
```

```
    try {
      db = Db4oEmbedded.openFile("bdProducts");
      List<Product> response = db.queryByExample(Product.class);
      // NO RETORNAR LA MISMA LISTA DE LA RESPUESTA
      products.addAll(response);
    } finally {
      db.close();
    }

    return products;
  }

  public void setNewProduct(Product newProduct) {
    ObjectContainer db = null;
    try {
      db = Db4oEmbedded.openFile("bdProducts");
      db.store(newProduct);
    } finally {
      db.close();
    }
  }
}
```

Volvemos a **index.jsp** y ***eliminamos todo el anterior en el que la lista de productos se especificaba en el propio HTML.*** Ahora obtendremos una instancia de **ProductsService** y recorreremos la lista que nos devuelve (aunque la base de datos está actualmente vacía). Intercalamos el código Java con el código HTML.

```
<div class="container" id="main-container">
  <h2>Productos</h2>
  <div class="row ">
    <%
      List<Product> listProducts = new ProductsService().getProducts();
      for(Product product : listProducts){
    %>
    <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
      <div>
        <img src="<%=product.getImage() %>"  alt="" />
        <div><%=product.getName() %></div>
        <a href="AddToShoppingCart?product=<%=product.getName() %>" class="btn btn-default" >
          <%=product.getPrice() %> €
        </a>
      </div>
    </div>
    <%
      }
    %>
  </div>
</div>
```

Incluimos los imports necesarios en **index.jsp**

```
<%@ page contentType="text/html;charset=UTF-8" language="java" pageEncoding="utf-8" %>
<%@ page language="java" import="com.uniovi.sdi.* , java.util.List"%>
```

Nota: Antes de probar esta funcionalidad incluiremos un mecanismo para poder agregar productos a la tienda de forma dinámica.

5.4 Agregar un producto a la tienda

Modificaremos el contenido de **admin.jsp** incluyendo un formulario que solicite el ***nombre, la imagen (URL) y el precio de un producto***. Estos datos se enviarán via **POST /admin.jsp**.

```
<!-- Contenido -->
<div class="container" id="main-container">

  <h2>Agregar producto a la tienda</h2>
  <form class="form-horizontal" method="post" action="admin.jsp">
    <div class="form-group">
      <label class="control-label col-sm-2" for="name">Nombre:</label>
      <div class="col-sm-10">
        <input type="text" class="form-control" id="name" name="name" required="true"/>
      </div>
    </div>
    <div class="form-group">
      <label class="control-label col-sm-2" for="image">URL imagen:</label>
      <div class="col-sm-10">
        <input type="text" class="form-control" id="image" name="image" required="true"/>
      </div>
    </div>
    <div class="form-group">
      <label class="control-label col-sm-2" for="price">Precio (€):</label>
      <div class="col-sm-10">
        <input type="number" step="0.01" class="form-control" id="price" name="price"
            required="true"/>
      </div>
    </div>
    <div class="form-group">
      <div class="col-sm-offset-2 col-sm-10">
        <button type="submit" class="btn btn-primary">Agregar</button>
      </div>
    </div>
  </form>
</div>
```

Vamos a incluir dentro de **admin.jsp** un segundo script de código Java que obtenga los parámetros de la petición, construya un objeto Producto y lo agregue a través de **ProductsService (en azul)**.

```
<%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8" %>
<%@ page language="java" import="com.uniovi.sdi.*" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd"><html lang="en">
<head>
  <title>Servlets</title>
  <meta charset="utf-8"/>
  <meta name="viewport" content="width=device-width, initial-scale=1"/>
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</head>
<body>
<%
  String usuario = (String) request.getSession().getAttribute("usuario");
  System.out.println("Usuario en sesión: " + usuario);
  if (usuario == null || usuario.equals("admin") == false) {
    // No hay usuario o no es admin
    response.sendRedirect("login.jsp");
  }
%>
<%
  if (request.getParameter("name") != null &&
      request.getParameter("image") != null &&
      request.getParameter("price") != null) {

    String name = request.getParameter("name");
    String image = request.getParameter("image");
    float price = Float.parseFloat(request.getParameter("price"));

    Product product = new Product(name, image, price);
    new ProductsService().setNewProduct(product);
    request.getRequestDispatcher("index.jsp").forward(request, response);
  }
%>
```

Ejecutamos la aplicación, accedemos a http://localhost:8080/ServletExample/admin.jsp y agregamos algunos de los siguientes productos:

- Manzanas, http://localhost:8080/ServletExample/images/iconfinder_apple.png, 3
- Fresas, http://localhost:8080/ServletExample/images/iconfinder_strawberry.png , 2.5
- Naranjas, http://localhost:8080/ServletExample/images/iconfinder_orange.png , 2.10
- Pan, http://localhost:8080/ServletExample/images/iconfinder_bread.png , 0.80

El fichero correspondiente a la base de datos se genera en la primera ejecución, en la carpeta **bin** del servidor Tomcat en la que se está ejecutando la aplicación:

…\apache-tomcat-9.0.41\bin

En el IDE STS de Eclipse se guardaría en:

\sts-bundle\sts-3.9.7.RELEASE

Nota: Subir el código a GitHub en este punto. Incluir el siguiente Commit Message ->

"SDI-IDGIT-1.5-Contexto aplicación y listado dinámico."

OJO: sustituir IDGIT por tu número asignado (p.e. 22324-101):

"SDI-2324-101-1.5-Contexto aplicación y listado dinámico."

(*No olvides incluir los guiones y NO incluyas BLANCOS*)

5.4.1 JSP Beans

Los Beans son clases Java construidas en base a unas especificaciones:

- Constructor por defecto sin argumentos.
- Tienen "propiedades" (atributos) que pueden ser: leídas y/o escritas.
- Se manejan a través de los métodos **get** y **set** de sus propiedades.

Un uso muy común de los **Beans** en JSP es la recuperación de datos de formularios, es decir, formar un objeto a partir de los parámetros recibidos.

Para que la clase Product pueda ser utilizada como un Bean le tenemos que agregar un constructor sin argumentos.

```
public class Product {
  private String name;
  private String image;
  private float price;
```

```
  public Product() {
  }
  ...
```

Modificamos **admin.jsp,** añadiendo la etiqueta **jsp: useBean** que crea un Bean nuevo de tipo **Product**, con el nombre de variable "**product**".

Con la etiqueta **setProperty (con property=*)** se analizan todos los parámetros de la petición(request) y los guarda en las propiedades del Bean (las que tengan el mismo nombre, se hace de forma automática). Sustituimos el código anterior por el Bean.

El **producto(product)** siempre va a contener un objeto != null ya que se crea automáticamente con el jsp:useBean. Por lo tanto, no vale con comprobar *product != null*, debemos comprobar el valor de sus propiedades. Ahora modificamos el código añadiendo el código correspondiente (en azul) y eliminando el resto de código (tachado).

```
<jsp:useBean id="product" class="com.uniovi.sdi.Product"/>
<jsp:setProperty name="product" property="*"/>
<%
  if (product.getName() != null) {
     new ProductsService().setNewProduct(product);
     request.getRequestDispatcher("index.jsp").forward(request, response);
  }
%>
<%
  if (request.getParameter("name") != null &&
        request.getParameter("image") != null &&
        request.getParameter("price") != null) {

     String name = (String) request.getParameter("name");
     String image = (String) request.getParameter("image");
     float price = Float.parseFloat(request.getParameter("price"));

     Product product = new Product(name, image, price);
     new ProductsService().setNewProduct(product);
     request.getRequestDispatcher("index.jsp").forward(request, response);
  }
%>
```

Nota: Con estos cambios la aplicación debería funcionar de la misma forma.

5.4.2 JavaBeans y ámbitos

Las directivas de JSP para manejar Beans nos permiten crear/obtener y modificar de forma sencilla las propiedades de un objeto desde una JSP. Vamos a crear un nuevo contador y a utilizarlo como un Bean.

En primer lugar, creamos la clase **Counter** en el paquete **com.uniovi.sdi.**

```
package com.uniovi.sdi;

public class Counter {
   private int total;

   public int getTotal() {
      return total;
   }

   public void setIncrease(int increaseValue) {
      total += increaseValue;
   }
}
```

A continuación, sustituimos el anterior script Java por el nuevo Bean, realizando las siguientes acciones:

- Incluimos el Bean en la página **index.jsp**
- Mostramos el valor del **total** - **getProperty(property)**
- Establecemos el **incremento** de 1. – **setProperty(property , value)**

```
<%-- Cambiado por el Bean
<%
   Integer counter = (Integer) application.getAttribute("counter");
   if (counter == null) {
      counter = 0;
   }
   application.setAttribute("counter", counter + 1);
%>
--%>

<jsp:useBean id="counter" class="com.uniovi.sdi.Counter"/>
<jsp:setProperty name="counter" property="increase" value="1"/>

<!-- Barra de Navegación superior -->
<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
   <div class="collapse navbar-collapse" id="my-navbarColor02">
      <ul class="navbar-nav mr-auto">
         <li class="nav-item ">
            <a class="nav-link" href="AddToShoppingCart">Carrito<span class="sr-
only">(current)</span></a>
```

```
        </li>
        <li class="nav-item ">
          <a class="nav-link" href="login.jsp">Login<span class="sr-only">(current)</span></a>
        </li>
        <li class="nav-item ">
          <a class="nav-link" href="admin.jsp">Administrar productos<span class="sr-
only">(current)</span></a>
        </li>
      </ul>
      <div class="nav navbar-right">
        <!-- cambiado por el bean -->
        <%-- <%=counter%>Visitas --%>
        <jsp:getProperty name="counter" property="total"/> Visitas
      </div>
    </div>
</nav>
```

Ejecutamos el proyecto y comprobamos que el contador siempre marca 1 **¿Qué está sucediendo?**

Por defecto el Bean tiene un ámbito (scope) de página, cada vez que se abre la página se crea el Bean (y el objeto Contador). Podemos modificar el ámbito (scope) de los Bean. Los posibles valores del atributo scope son: **page**|**request**|**session**|**application**

Si queremos hacer un contador común para todos los usuarios el ámbito del Bean sería "**application**". Añadimos el atributo **scope** a la etiqueta **useBean**:

```
<jsp:useBean id="counter" class="com.uniovi.sdi.Counter" scope="application"/>
```

Volvemos a desplegar la aplicación y la probamos de nuevo. Ahora por cada instancia se incrementa el contador.

Nota: Subir el código a GitHub en este punto. Incluir el siguiente Commit Message ->

"SDI-IDGIT-1.6-JSP Beans, JavaBeans y ámbitos."

OJO: sustituir IDGIT por tu número asignado (p.e. 2324-101):

"SDI-2324-101-1.6-JSP Beans, JavaBeans y ámbitos."

(*No olvides incluir los guiones y NO incluyas BLANCOS*)

5.5 Uso de tags JSTL Core

Las etiquetas JSTL[9] (JavaServer Pages Standard Tag Library) encapsulan gran parte de la funcionalidad común que se suele necesitar en las páginas web JSP. Usando estas etiquetas se evita incluir scripts de código Java propios. Para usar JSTL necesitamos incluir la librería **jstl.jar** en el directorio **WebContent/WEB-INF/lib.** En nuestro caso ya la habíamos movido a ese directorio al copiar las librerías de la base de datos (la versión 1.2 de jstl).[10]

Para utilizar las etiquetas de JSTL debemos declarar el uso de JSTL, agregando la directiva al inicio de la página que se va a utilizar. En nuestro caso, vamos a incluirla en el fichero **index.jsp.**

```
<%@ page contentType="text/html;charset=UTF-8" language="java" pageEncoding="utf-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page import="com.uniovi.sdi.*,java.util.List" %>
```

Vamos a sustituir el código Java que recorría la lista de productos por etiquetas JSTL, <c:forEach> permite recorrer los elementos de una lista. Para obtener la lista utilizamos **ProductsService** como un Bean, obteniendo los artículos a través de products (se invoca por detrás a getProducts).

El elemento que se está recorriendo actualmente se guarda en la variable declarada en el atributo "var", en este caso **product;** y podemos acceder a sus atributos **#{product.<nombre_propiedad>}**. Para imprimir por pantalla el valor de la variable podemos utilizar <c:out>

Tras estos cambios el nuevo código del pasará a ser el siguiente.

```
<div class="container" id="main-container">
  <h2>Productos</h2>
  <div class="row ">
    <jsp:useBean id="productsService" class="com.uniovi.sdi.ProductsService"/>
    <c:forEach var="product" begin="0" items="${productsService.products}">
      <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
        <div>
          <img src="<c:out value="${product.image}"/>"/>
          <div><c:out value="${product.name}"/></div>
          <a href="AddToShoppingCart?product=<c:out value="${product.name}"/>"
            class="btn btn-default">
            <c:out value="${product.price}"/> €
          </a>
        </div>
      </div>
    </c:forEach>
  </div>
</div>
```

[9] http://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/

[10] Además, para servidores que cumplen JEE 1.7 la librería jstl.jar ya está incluida en las librerías del servidor.

Utilizando Beans y JSTL podríamos llegar a prescindir de los scripts Java tradicionales. Por ejemplo, en **admin.jsp** agregamos la directiva (taglib prefix="c") para poder usar JSTL mediante el prefijo "c:".

```
<%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

Cambiamos el antiguo script utilizado para agregar el producto por uno que utilice JSTL y JavaBean, la **directiva <c:if test="exp">** permite ejecución condicional.

```
<%-- Cambiado por las etiquetas JSTL
<%
    String user = (String) request.getSession().getAttribute("user");
    System.out.println("Usuario en sesión: " + user);
    if (user == null || user.equals("admin") == false) {
        // No hay usuario o no es admin
        response.sendRedirect("login.jsp");
    }
%>
--%>
<c:if test="${sessionScope.user != 'admin'}">
    <c:redirect url="/login.jsp"/>
</c:if>

<jsp:useBean id="product" class="com.uniovi.sdi.Product"/>
<jsp:setProperty name="product" property="*"/>

<%-- Cambiado por las etiquetas JSTL
<%
    if (product.getName() != null) {
        new ProductsService().setNewProduct(product);
        request.getRequestDispatcher("index.jsp").forward(request, response);
    }
%>
--%>

<c:if test="${product.name != null}">
    <jsp:useBean id="productsService" class="com.uniovi.sdi.ProductsService"/>
    <jsp:setProperty name="productsService" property="newProduct" value="${product}"/>
    <c:redirect url="/index.jsp"/>
</c:if>
```

También podemos crear nuestras propias librerías de Tags, con funcionalidades personalizadas: http://docs.oracle.com/cd/E11035_01/wls100/taglib/quickstart.html

Nota: Si quisiéramos completar la arquitectura de la aplicación, el Carrito podría ser otro Bean y la lista de productos incluidos en el carrito se podría recorrer con JSTL.

Nota: Subir el código a GitHub en este punto. Incluir el siguiente Commit Message ->

"SDI-IDGIT-1.7-Uso de tags JSTL Core."

OJO: sustituir IDGIT por tu número asignado (p.e. 2324-101):

"SDI-2324-101-1.7-Uso de tags JSTL Core."

(*No olvides incluir los guiones y NO incluyas BLANCOS*)

6 MVC, Modelo Vista Controlador

Vamos a incluir una implementación muy simple de una arquitectura MVC (Modelo-Vista - Controlador) para obtener los productos que hay en el carrito. **ServletShoppingCart** va a continuar respondiendo a la petición **/AddToShoppingCart** desde la función **doGet().**

Este Servlet hará el papel de **controlador** una vez recibida la petición:

- Igual que antes → Comprueba si hay carrito en sesión, comprueba si la petición contiene un producto.
- Nuevo → Introduce el carrito(**cart**) como attributo en la request con la clave "***selectedItems***".
- Nuevo → redirige la petición a la vista **cart.jsp** (la vista puede utilizar la variable "***selectedItems***" que acabamos de definir).

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
  HttpSession session = request.getSession();

  HashMap<String, Integer> cart =
      (HashMap<String, Integer>) session.getAttribute("cart");

  // No hay carrito, creamos uno y lo insertamos en sesión
  if (cart == null) {
    cart = new HashMap<String, Integer>();
    session.setAttribute("cart", cart);
  }

  String product = request.getParameter("product");
  if (product != null) {
    addToShoppingCart(cart, product);
  }
```

```
    // Retornar la vista con parámetro "selectedItems"
    request.setAttribute("selectedItems", cart);
    getServletContext().getRequestDispatcher("/cart.jsp").forward(request, response);

    /* Eliminado al aplicar el patrón MVC
    response.setCharacterEncoding("UTF-8");
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD><TITLE>Tienda SDI: Cesta de la compra</TITLE></HEAD>");
    out.println("<BODY>");
    out.println(shoppingCartToHtml(cart) + "<br>");
    out.println("<a href=\"index.jsp\">Volver</a></BODY></HTML>");*/
}
```

Creamos la vista **cart.jsp**, en ella hacemos uso de las etiquetas de JSTL para recorrer los elementos del hashmap ***"selectedItems"***. Como es un hashmap, cada objeto tiene una **key** y un **value**.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" pageEncoding="utf-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html lang="en">
<head>
    <title>Vista carrito</title>
    <meta charset="utf-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1"/>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</head>
<body>
<jsp:useBean id="counter" class="com.uniovi.sdi.Counter" scope="application"/>
<jsp:setProperty name="counter" property="increase" value="1"/>

<!-- Barra de Navegación superior -->
<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
    <div class="container-fluid">
        <ul class="nav navbar-nav">
            <li class="nav-item"><a class="nav-link" href="AddToShoppingCart">Carrito</a></li>
            <li class="nav-item"><a class="nav-link" href="login.jsp">Login</a></li>
            <li class="nav-item"><a class="nav-link" href="admin.jsp">Administrar productos</a></li>
        </ul>
        <div class="nav navbar-right">
            <div class="center-block">
                <jsp:getProperty name="counter" property="total"/>
                Visitas
            </div>
```

```
    </div>
  </div>
</nav>

<!-- Contenido -->
<div class="container" id="main-container">
  <h2>Vista Carrito</h2>
  <ul>
    <c:forEach var="item" items="${selectedItems}">
      <tr>
        <li>${item.key} - ${item.value} </li>
      </tr>
    </c:forEach>
  </ul>
  <a href="index.jsp">Volver</a>
</div>
</body>
</html>
```

Nota: Subir el código a GitHub en este punto. Incluir el siguiente Commit Message ->

"SDI-IDGIT-1.8-MVC, Modelo Vista Controlador."

OJO: sustituir IDGIT por tu número asignado (p.e. 2021-101):

"SDI-2223-101-1.8-MVC, Modelo Vista Controlador."

(*No olvides incluir los guiones y NO incluyas BLANCOS*)

7 Resultado esperado en el repositorio de GitHub

Al revisar el repositorio de código en GitHub, los resultados de los commits realizados deberían ser parecidos a estos:

- ⇒ **SDI-2324-101-1.0-Create servlet project.**
- ⇒ **SDI-2324-101-1.1-Los Servlets son multihilo.**
- ⇒ **SDI-2324-101-1.2-Manejo de sesiones.**
- ⇒ **SDI-2324-101-1.3-JSP Java Server Pages.**
- ⇒ **SDI-2324-101-1.4-Manejo de sesiones (2).**
- ⇒ **SDI-2324-101-1.5-Contexto aplicación y listado dinámico.**
- ⇒ **SDI-2324-101-1.6-JSP Beans, JavaBeans y ámbitos.**
- ⇒ **SDI-2324-101-1.7-Uso de tags JSTL Core.**
- ⇒ **SDI-2324-101-1.8-MVC, Modelo Vista Controlador.**