



Sistemas Distribuidos e Internet

Introducción a JEE – Servlets y JSPs

Sesión 1

Curso 2019/2020

Contenido

1	INTRODUCCIÓN	2
1.1	CONFIGURACIÓN DEL ENTORNO DE DESARROLLO	2
1.1.1	Descargar e instalar Java	3
1.1.2	Configurar la variable de entorno del sistema JAVA_HOME	3
1.1.3	Configurar la variable PATH del sistema	4
1.1.4	Descargar e Instalar Git.....	5
1.1.5	Configurar Git en STS IDE y NOMBRE DEL REPOSITORIO para cada ALUMNO	6
1.1.6	Invitar como colaborador a la cuenta de monitorización de la asignatura SDI.....	10
1.2	IMPORTAR PROYECTO DESDE GITHUB	11
1.2.1	Hacer el primer commit desde el IDE	16
2	SERVLETS Y PARÁMETRO	18
2.1	LOS SERVLETS SON MULTITHREAD.....	23
2.2	MANEJO DE SESIÓN.....	25
3	JSP JAVA SERVER PAGES	29
3.1	MANEJO DE SESIÓN (2).....	32
3.2	CONTEXTO DE LA APLICACIÓN.....	35
3.3	LISTADO DINÁMICO DE PRODUCTOS.....	36
3.4	AGREGAR UN PRODUCTO A LA TIENDA.....	38
3.4.1	JSP Beans	40
3.4.2	JavaBeans y ámbitos	41
3.5	USO DE TAGS JSTL CORE.....	43
4	MVC, MODELO VISTA CONTROLADOR	46
4.1	RESULTADO ESPERADO EN EL REPOSITORIO DE GITHUB	48



1 Introducción

En esta práctica vamos a implementar 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 desde Servlets y JSPs (Java Server Pages) hasta JavaBeans y JSTL (JavaServer Pages Standard Tag Library). Y acabaremos por encajar 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 embebido 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.

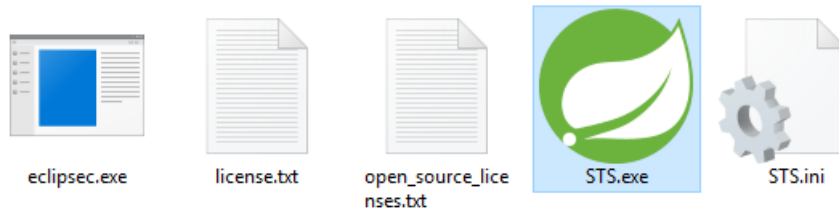
Nota: De manera progresiva el alumno deberá ir subiendo el código a GitHub en los puntos de controles especificados en el documento.

1.1 Configuración del entorno de desarrollo

Para desarrollar aplicaciones Web en Java es necesario trabajar con un IDE adecuado. Uno de los IDE utilizado es el **Spring Tool Suite (STS)** <https://spring.io/tools>. Este entorno está basado en el IDE eclipse e incluye todo lo necesario para facilitarnos el desarrollo de aplicaciones web en Java y también está optimizado para el uso del framework Spring con el que desarrollaremos más adelante.

Descargamos la versión **3.9.7** del STS correspondiente a nuestro sistema operativo y lo descomprimos (Requiere que tengamos instalado Java 1.8 o superior).

Para iniciar el entorno de desarrollo ejecutamos el fichero: **/sts-bundle/sts-3.9.7.RELEASE/STS.exe** o el correspondiente a nuestro SO. Este documento está confeccionado sobre la plataforma Windows, pero es perfectamente válido para otros SSOO. STS dispone de versiones tanto para Mac OSX como para Linux.

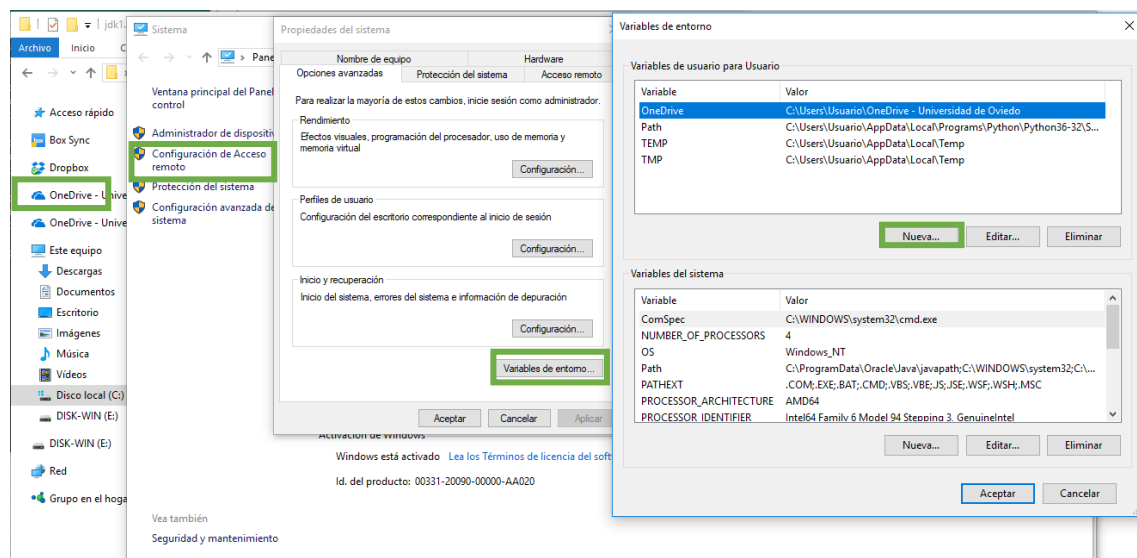


1.1.1 Descargar e instalar Java

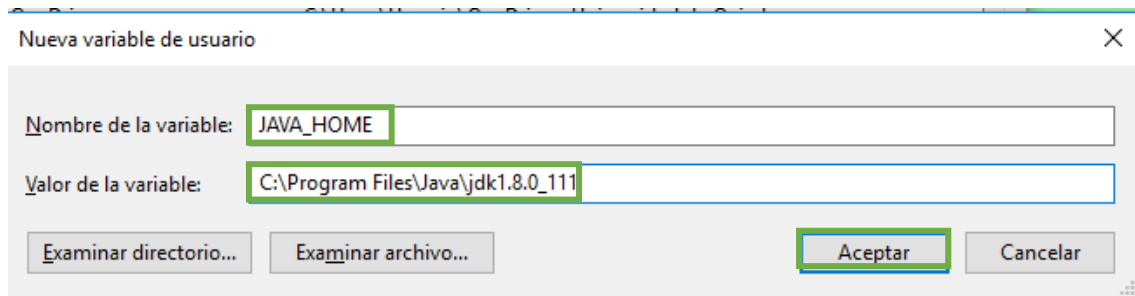
Podemos descargar la última versión de Java desde la siguiente URL:
<https://www.java.com/es/download/>

1.1.2 Configurar la variable de entorno del sistema JAVA_HOME¹

Una vez descargado e instalado Java configuramos la variable de entorno JAVA_HOME

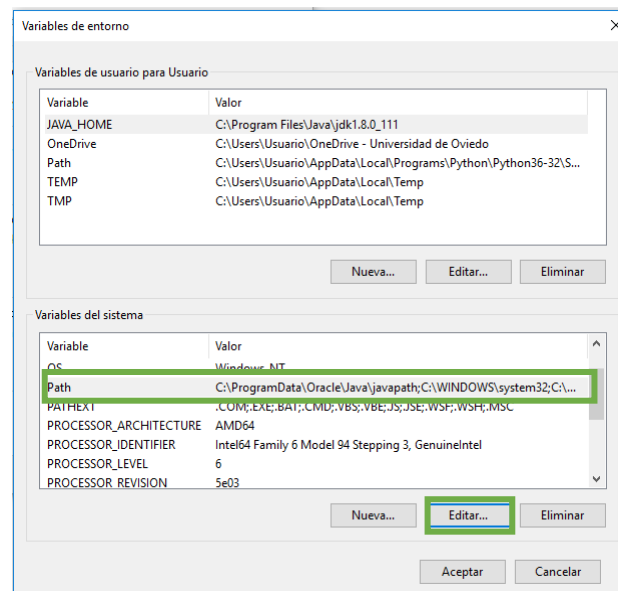


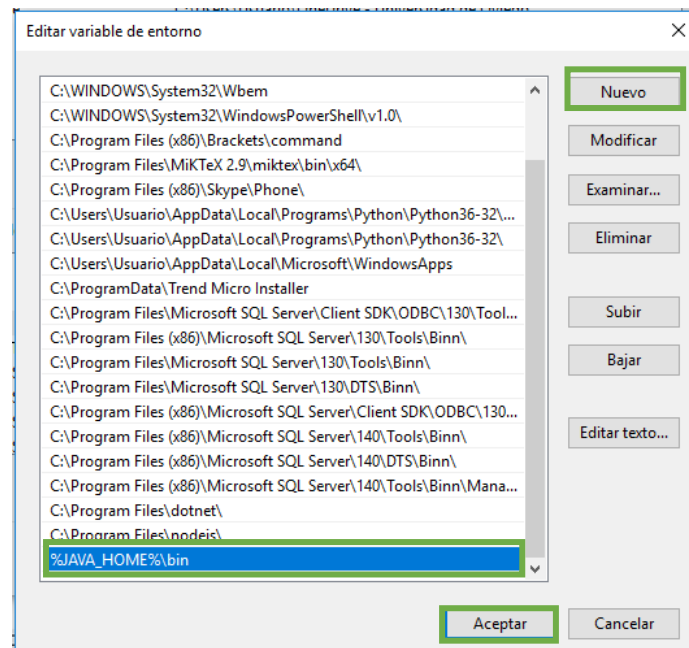
¹ Para configurar JAVA_HOME en MacOSX usa `$> export JAVA_HOME=$(/usr/libexec/java_home)`
([Enlace](#))



1.1.3 Configurar la variable PATH del sistema

En la misma ventana de configuración de variables de entorno, editar la variable PATH existente.



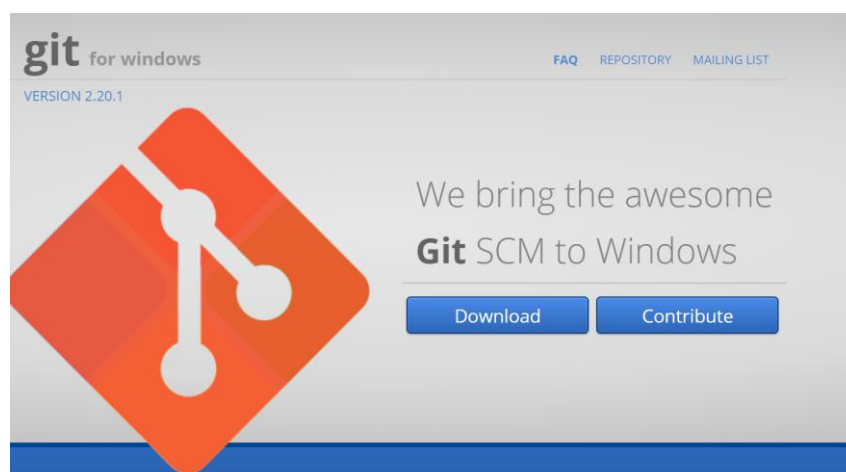


1.1.4 Descargar e Instalar Git

Git es uno de los sistemas de control de versiones más utilizados en el mercado y por ello 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 e instalarlo.

Nota: Instalar previamente un editor de texto como Notepad++ o Sublime Text para utilizarlo como editor de texto de Git.

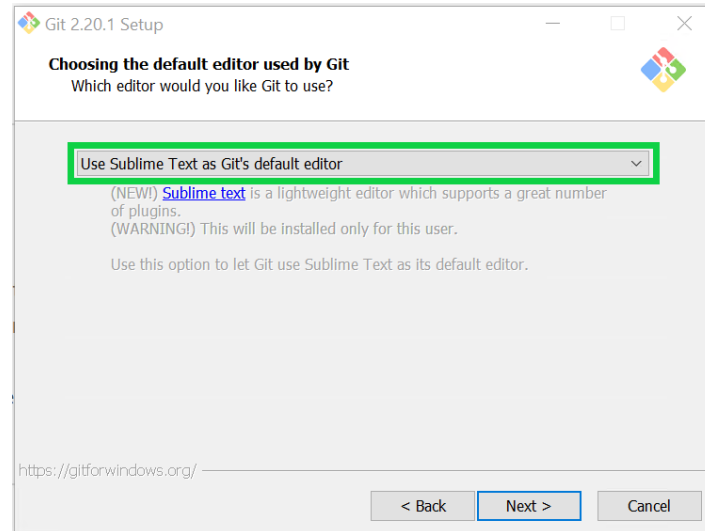
Descargar e instalar desde la siguiente URL: <https://gitforwindows.org/>



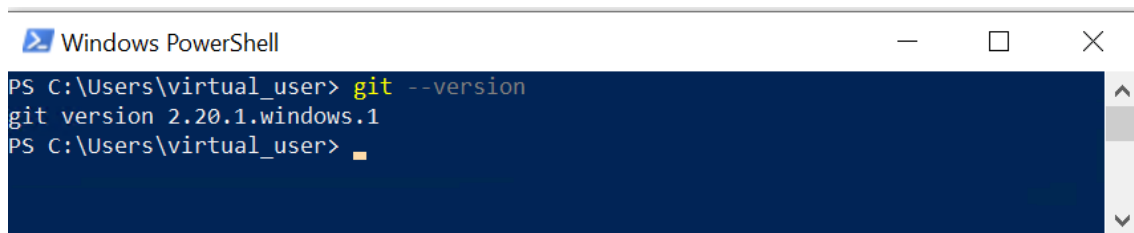
Si quieres instalar Git en MACOSX puedes descargarlo desde aquí: <https://git-scm.com/download/mac>



Durante la instalación nos pedirá que seleccionemos un editor de texto. Se recomienda utilizar un editor más moderno como Notepad++, Sublime Text, etc.²



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



1.1.5 Configurar Git en STS IDE y NOMBRE DEL REPOSITORIO para cada ALUMNO

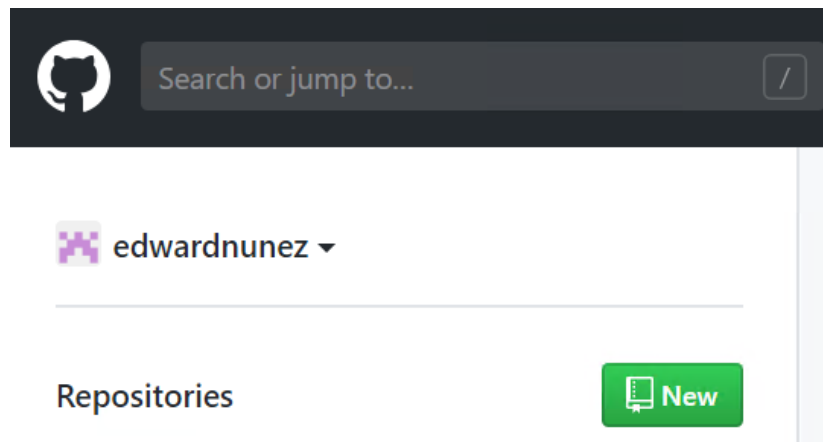
Una vez instalado Git en Windows lo siguiente es configurar nuestro IDE para poder trabajar con nuestros repositorios de código desde el entorno de desarrollo.

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 proyecto. Por ejemplo, vamos a crear un **nuevo Repositorio** en GitHub para el proyecto que denominaremos **sdi-lab-jee** y que utilizaremos para poder completar esta práctica.

Para ello, desde la página Web de **GitHub** buscamos la sección de repositorios y hacemos click en el botón **New** (<https://github.com/new>).

² En la instalación desde MACOSX no nos solicitará que suministremos editor por defecto.



!!!!!!MUY IMPORTANTE!!!!!!

Aunque en este guión se ha usado como nombre de repositorio para todo el ejercicio **ServletExample**, cada alumno deberá usar como nombre de repositorio **sdix-lab-jee**, donde **x** = columna **IDGIT** en el Documento de ListadoAlumnosSDI1920.pdf del CV.

Por ejemplo el alumno con **IDGIT=1920-101**, deberá crear un repositorio con nombre **sdi1920-101-lab-jee** y un proyecto **JEE** en **STS** con nombre también **sdi1920-101-lab-jee**.

En resumen:

Nombre repositorio GitHub: **sdix-lab-jee**

Nombre proyecto STS : **sdix-lab-jee**


Otra cuestión **IMPRESINDIBLE** 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 tal como se muestra en la siguiente imagen y hacemos click en el botón **Create Repository** (ojo que el nombre no debe ser **sdix-lab-jee** como se ha indicado en el cuadro anterior). Una vez creado el repositorio en GitHub, debemos configurar STS para el acceso a GitHub.

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner:  edwardnunez ▾ / Repository name *: ✓

Great repository names are short and memorable. Need inspiration? How about [upgraded-barnacle](#).


Description (optional):

☐ Public
Anyone can see this repository. You choose who can commit.

☒ Private
You choose who can see and commit to this repository.

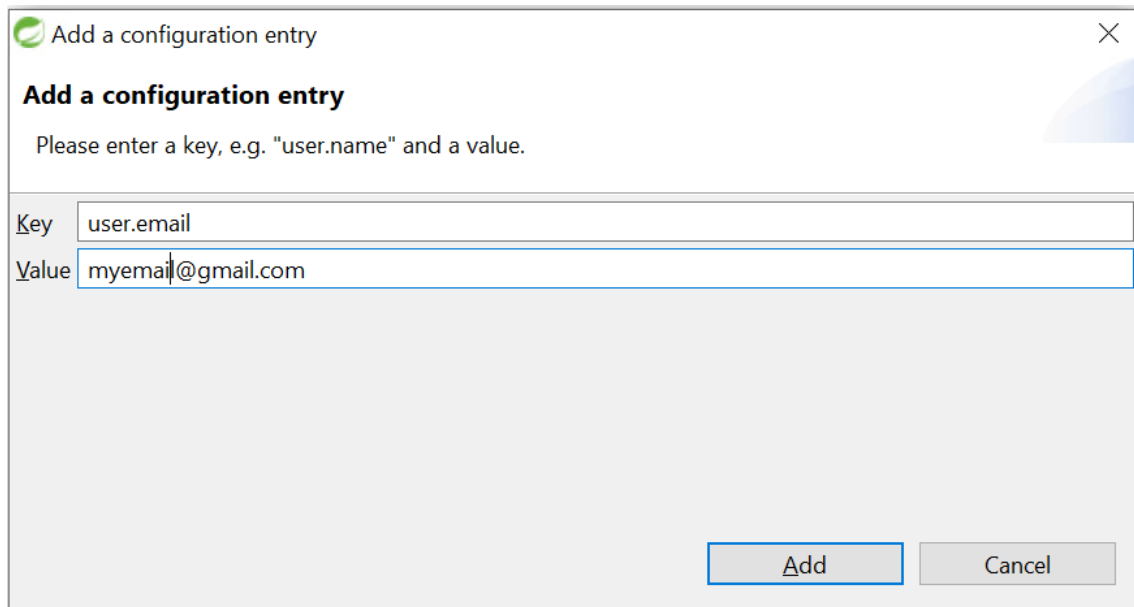
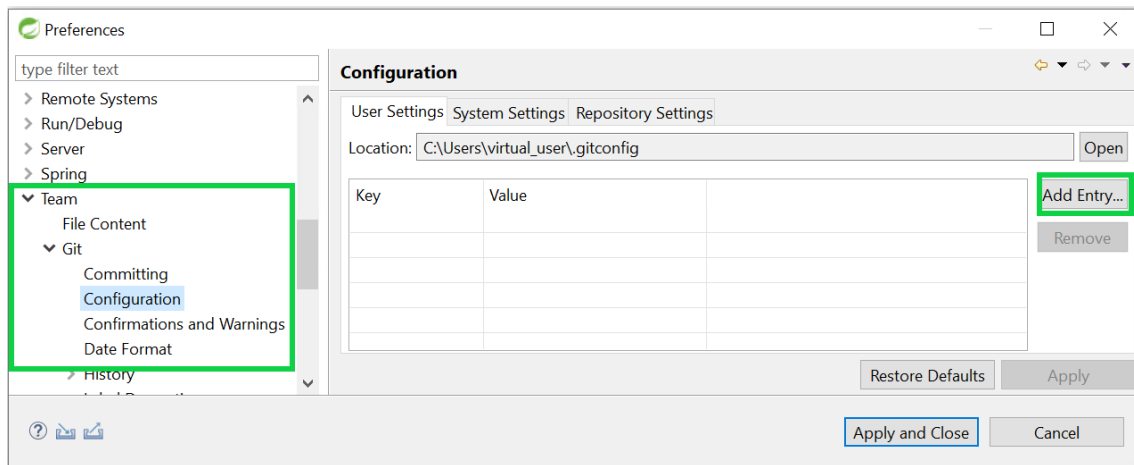
☒ Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

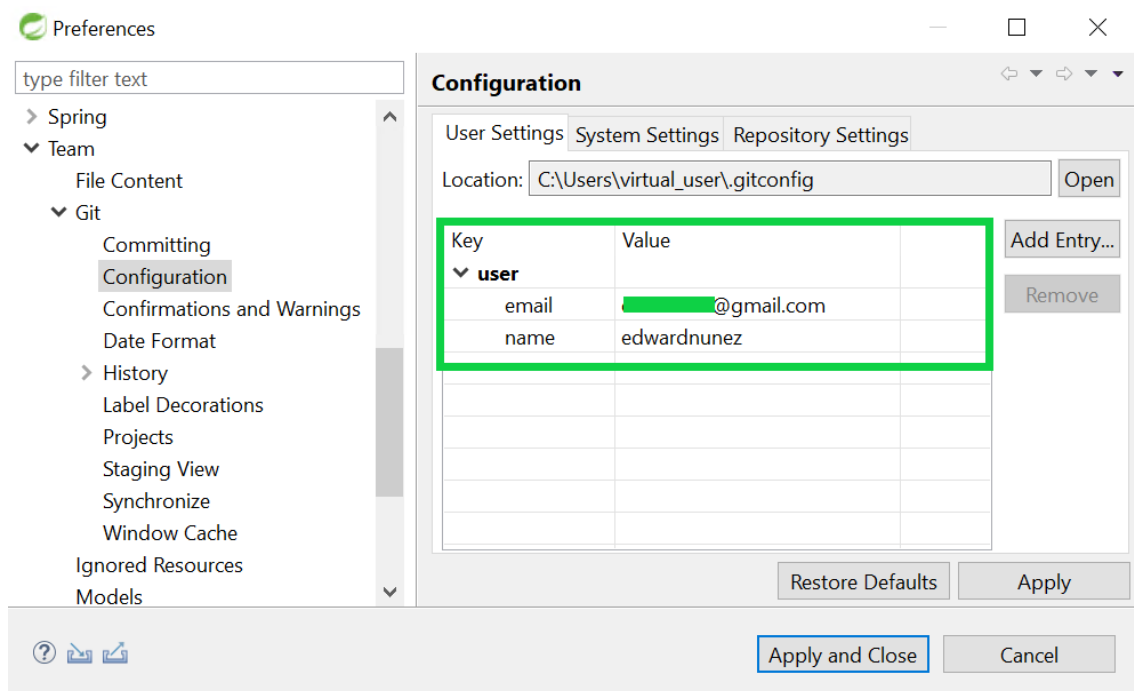
Add .gitignore: / Add a license: ⓘ



En STS vamos al Menú: **Window/Preferences**³, buscamos el submenú **Team/Git/Configuration** y añadimos nuestros datos de configuración que serán los datos por defecto para hacer los commit al repositorio. Añadimos nuestro nombre y correo electrónico, siguiendo los pasos como se muestran en la siguiente imagen, para cada uno de los campos a incluir (**user.email** y **user.name**) en la configuración:

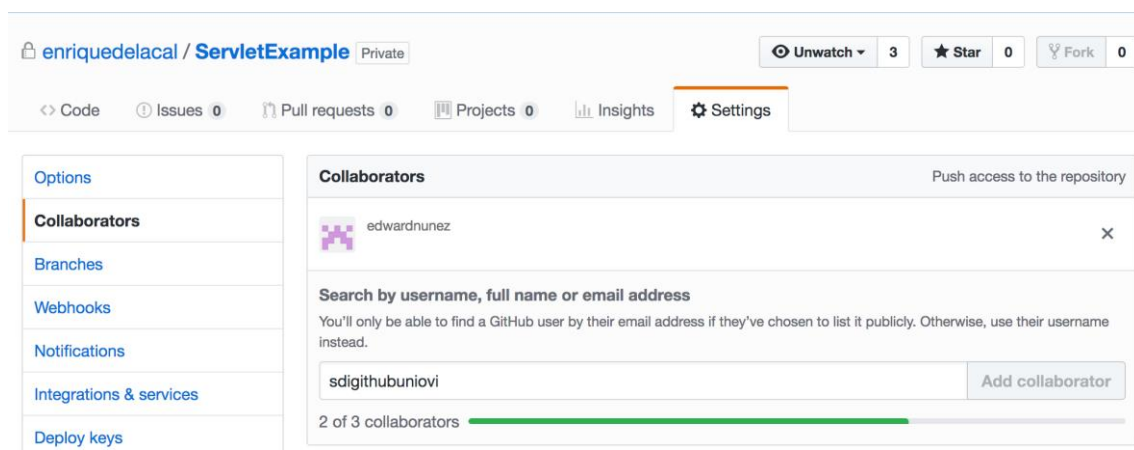
³ En MACOSX **Spring Tool Suite/Preferencias**.





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

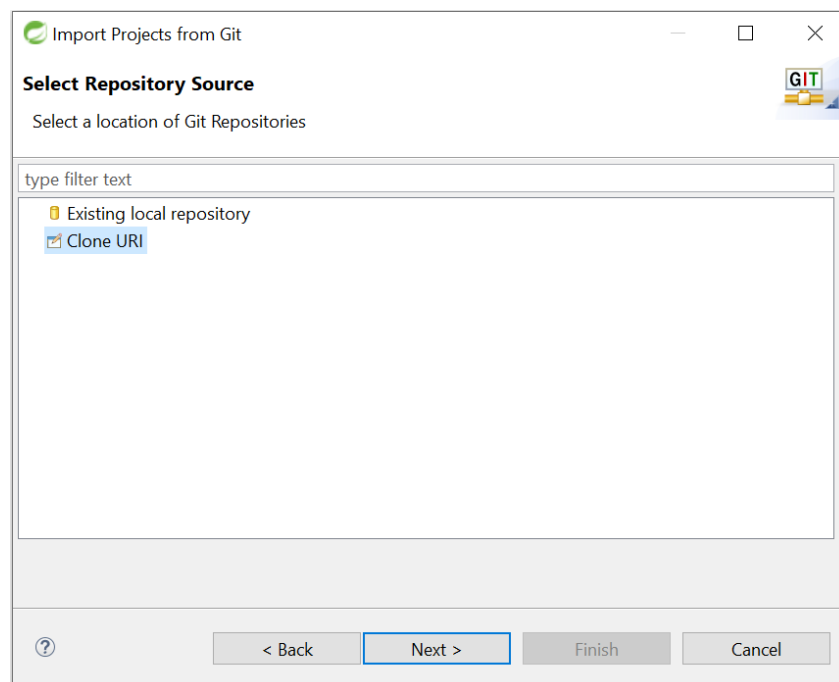
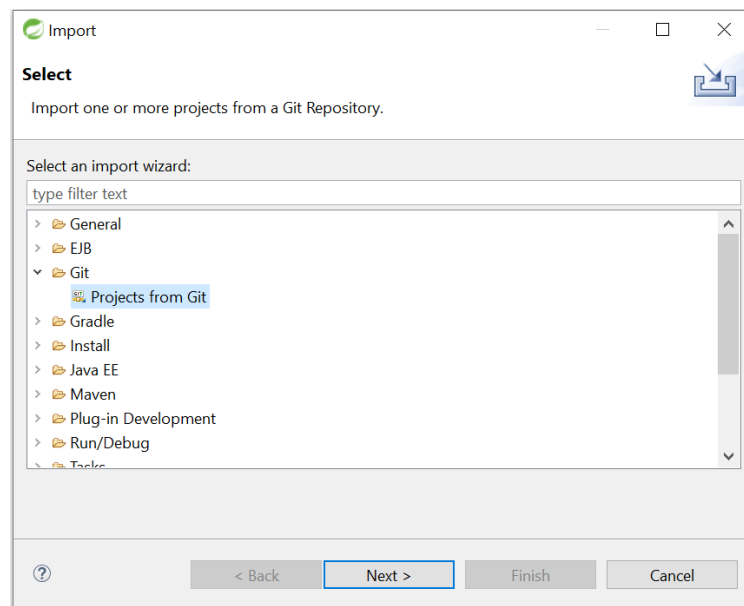
Una vez creado el repositorio deberemos obligatoriamente invitar a nuestro repositorio a la cuenta **sdigithubuniovi**. Para ellos debe ir a la opción colaboradores de nuestro repositorio **sdix-lab-je** (<https://github.com/enriquedelacal/ServletExample/settings/collaboration>) y agregar al usuario **sdigithubuniovi**.





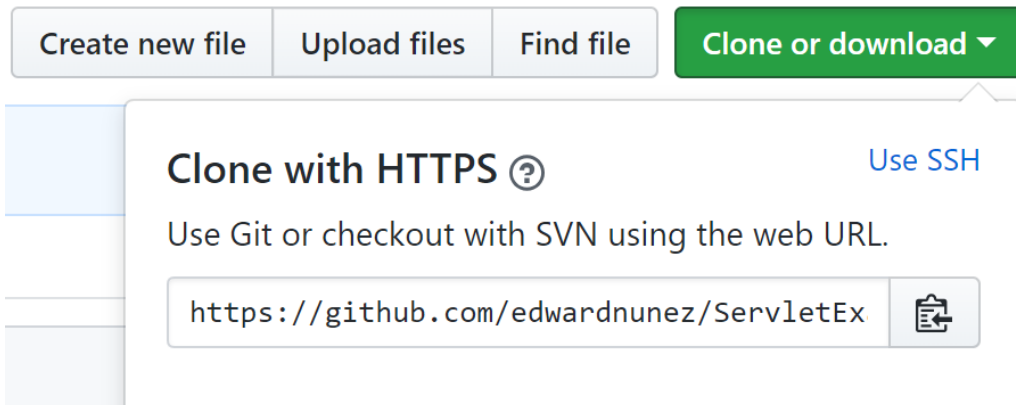
1.2 Importar proyecto desde GitHub

A continuación vamos a crear un proyecto web dinámico con nombre **sdix-lab-jee** vinculándolo al repositorio de GitHub creado anteriormente (**sdix-lab-jee**, donde x = IDGIT en el listado de IDs de GITs en CV). Vamos al menú **File -> Import** del IDE y seguimos los siguientes pasos:

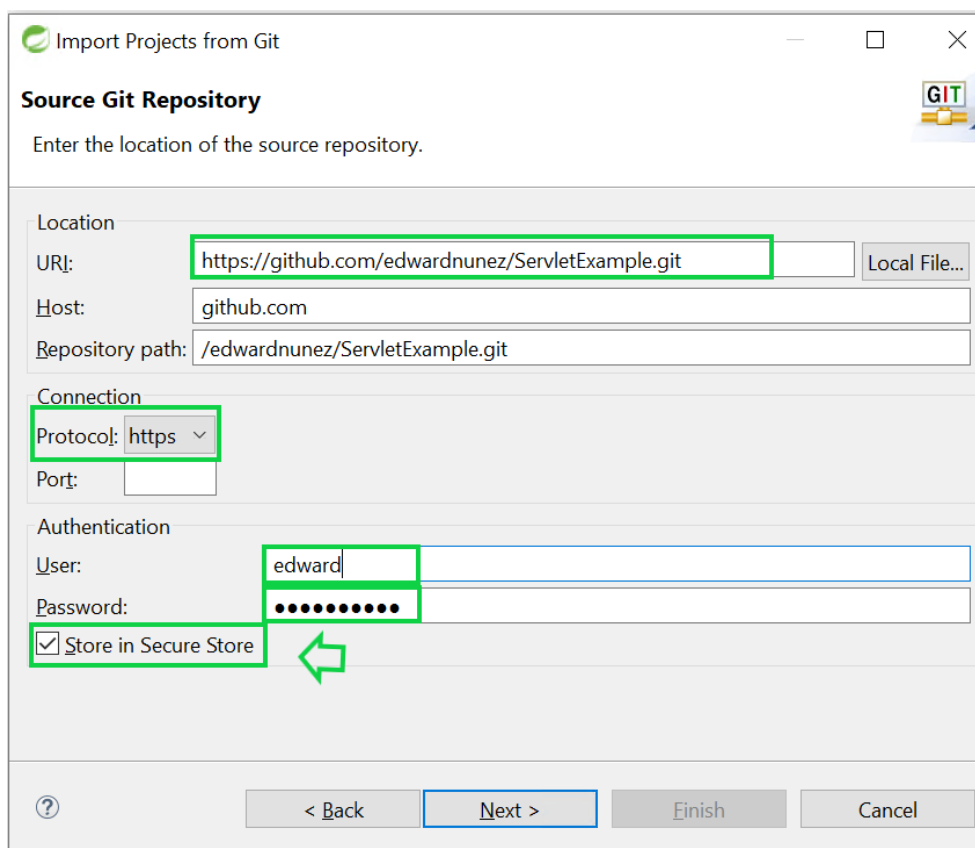




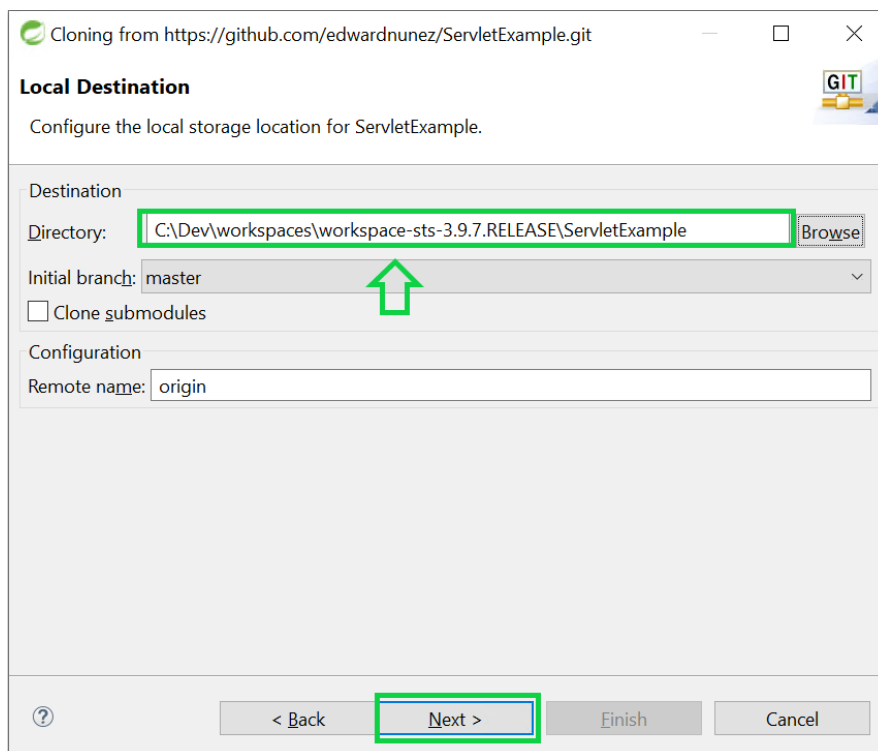
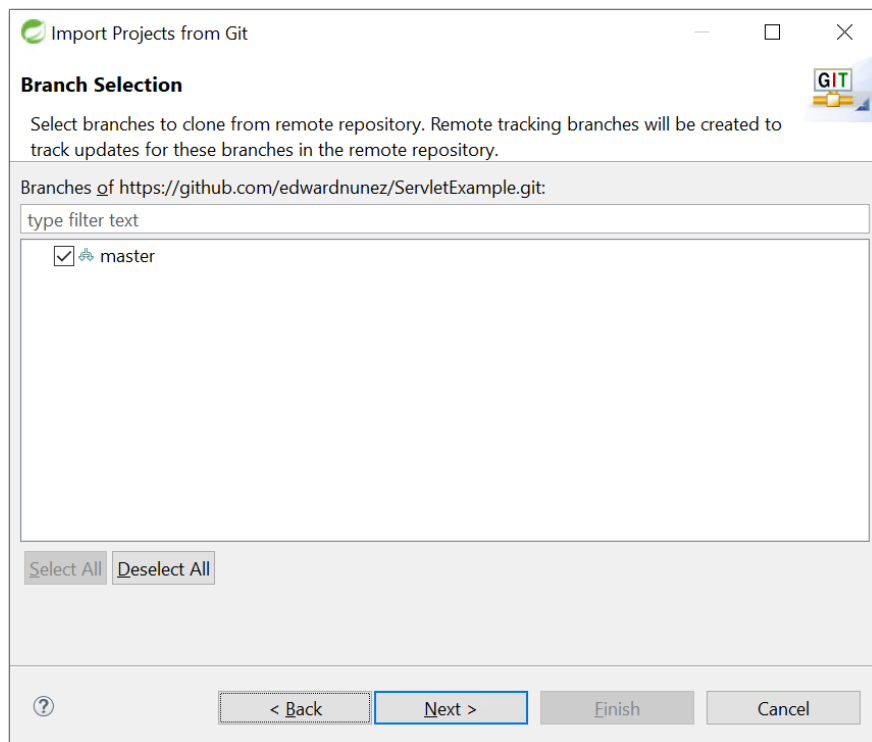
Desde la página web de GitHub copiamos la URL del repositorio para clonarlo en local. Copiamos el enlace usando el protocolo HTTPS.



Especificamos en el IDE los **datos de nuestro repositorio** y **las credenciales de nuestro usuario** de GitHub, similar a lo que se muestra en la siguiente imagen.



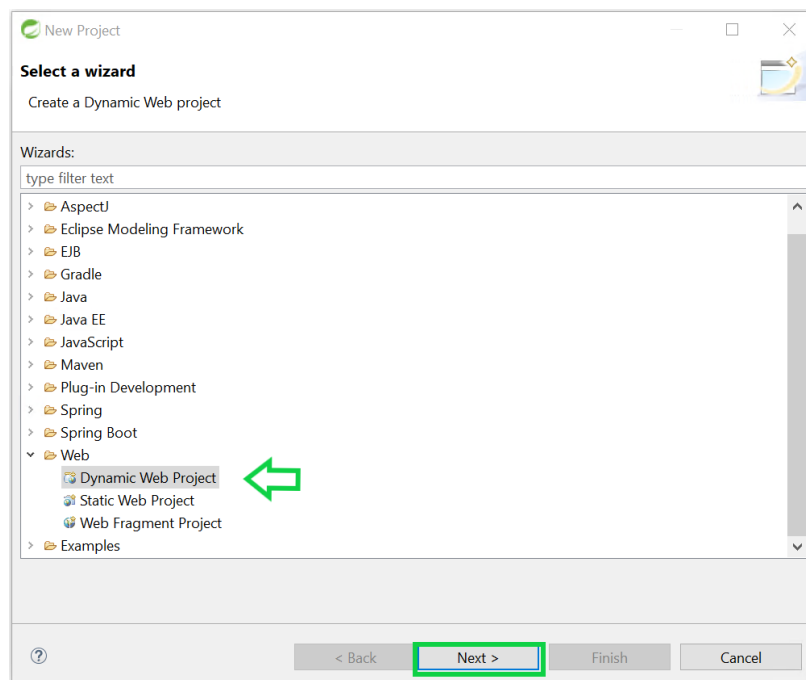
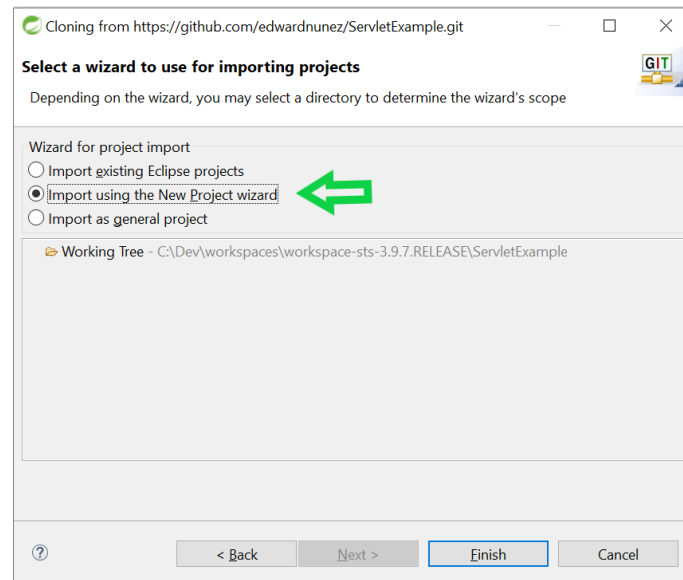
Finalmente, seleccionamos la rama master y especificamos el directorio destino de nuestro workspace en local.



Como no tenemos creado aún el proyecto, seleccionamos la opción **Import using the New Project wizard**.



Nota: Si el proyecto estuviese creado entonces deberíamos seleccionar una de las otras dos opciones.



Especificamos el nombre del proyecto (recuerde emplear **sdix-lab-je**) y hacemos click en el botón finalizar.



New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name:

Project location

☒ Use default location

Location:

Target runtime

Dynamic web module version

Configuration

A good starting point for working with Pivotal tc Server Developer Edition (Runtime) v4.0 runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership

☐ Add project to an EAR

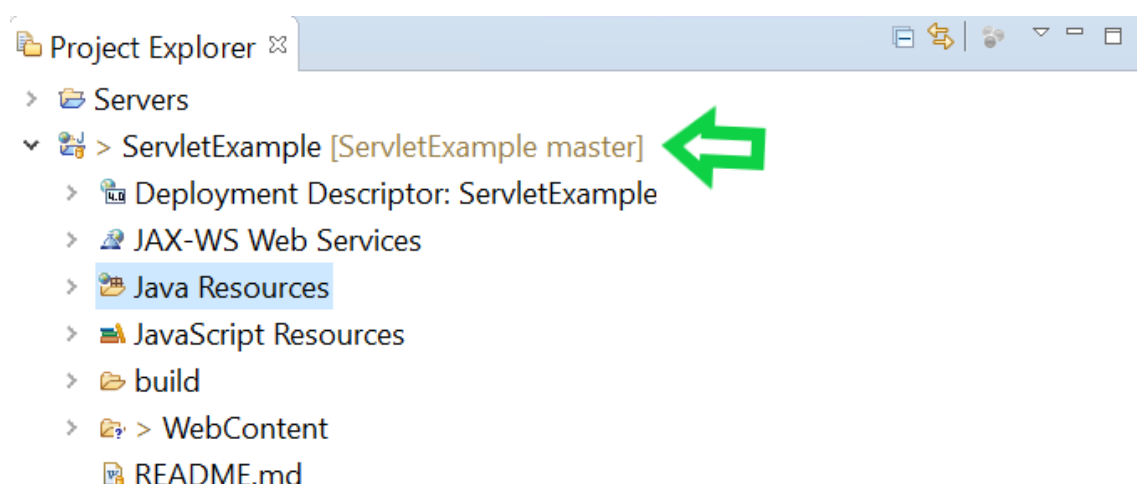
EAR project name:

Working sets

☐ Add project to working sets

Working sets:

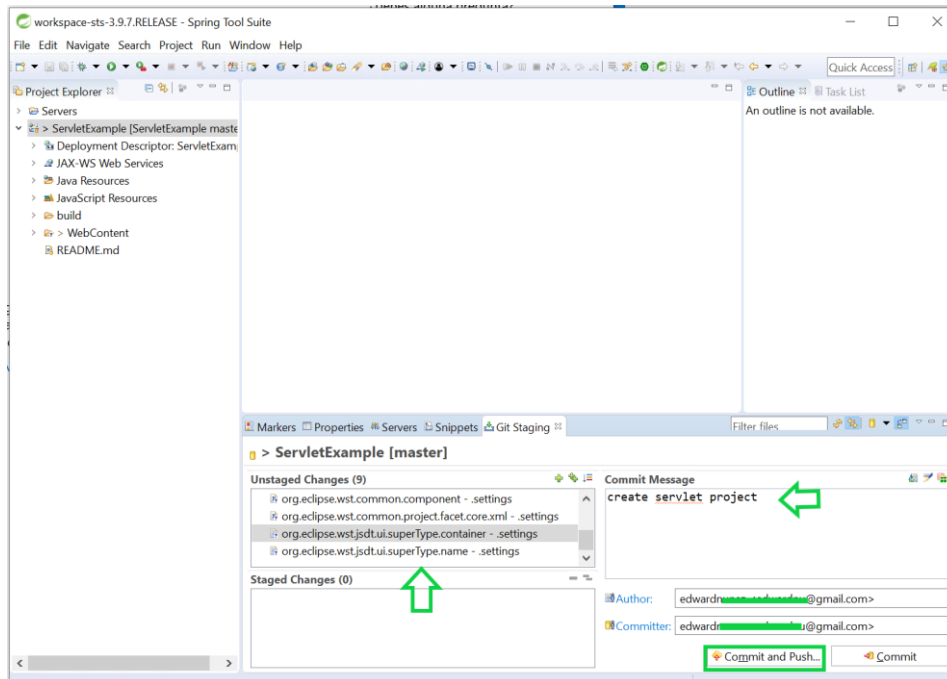
Al final, hemos creado un proyecto web dinámico que se encuentra sincronizado con la rama master del repositorio **sdix-lab-jee** en Git.



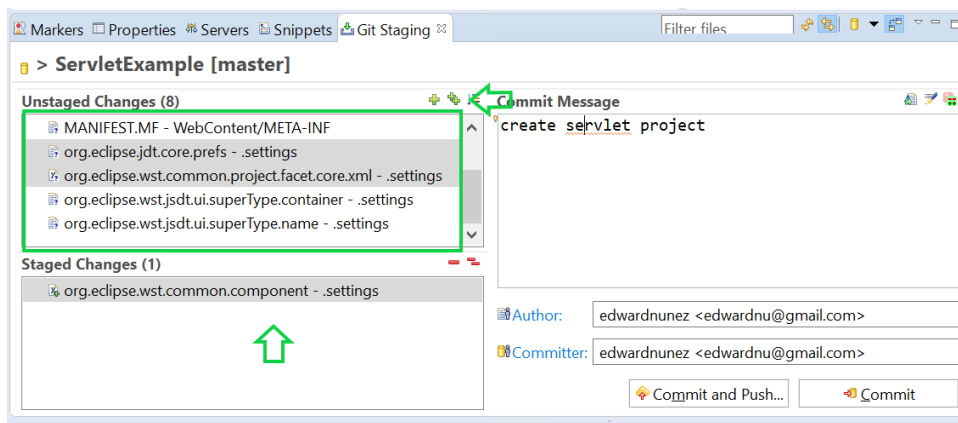


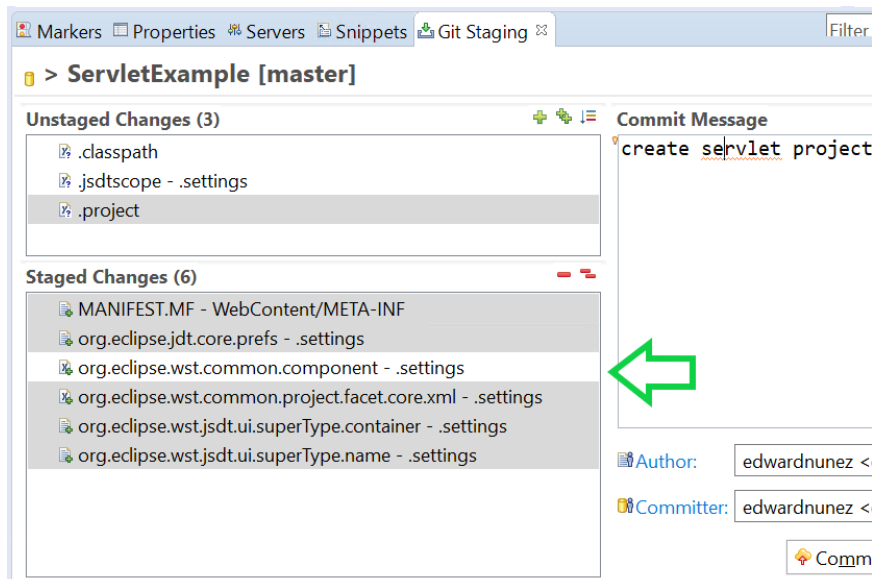
1.2.1 Hacer el primer commit desde el IDE

Vamos al proyecto y con un click derecho vamos a la opción **Team → Commit** y añadimos los cambios, escribimos el mensaje del commit y finalmente presionamos el botón **Commit and Push**.

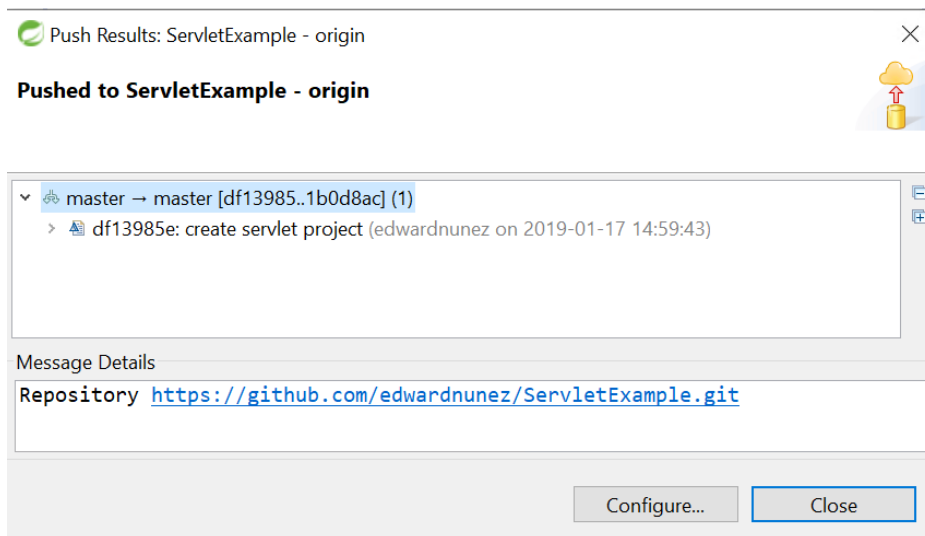


Nota: Antes de hacer el commit hay que añadir los ficheros al panel **Stages Changes**, seleccionando los ficheros y haciendo click en el botón ++. Debemos seleccionar todos los elementos de la caja “Unstaged Changes”.

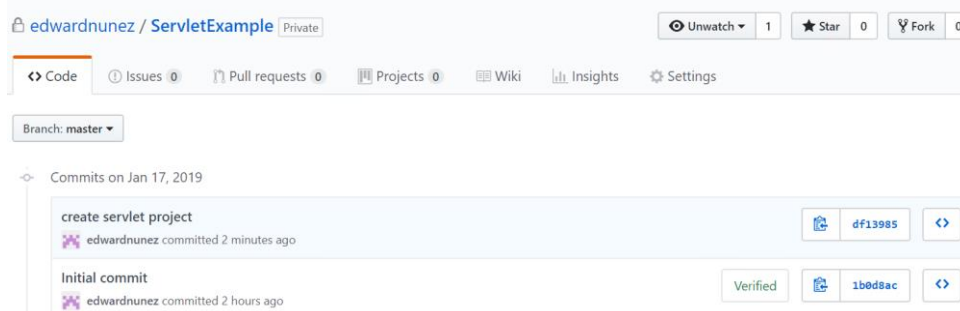




Finalmente, se abrirá una ventana como la que se muestra a continuación.



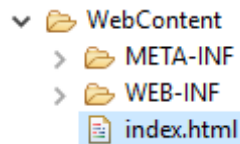
Ahora, si vamos a la web de GitHub podremos ver los cambios que hemos subido al repositorio.





2 Servlets y parámetro

Una vez creado el proyecto Java vamos a continuar desarrollando el Servlet, lo primero que haremos es añadir un fichero **index.html** en la carpeta **/WebContent/** de nuestro proyecto (botón derecho, **New** → **File**). Incluimos el siguiente contenido:



El fichero index contendrá **dos formularios uno GET y otro POST** con un parámetro con clave **nombre** que se envían a la URL **ServletSaludo**.

```
<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/3.3.7/css/bootstrap.min.css"/>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</head>
<body>

<!-- Contenido -->
<div class="container" id="contenedor-principal">

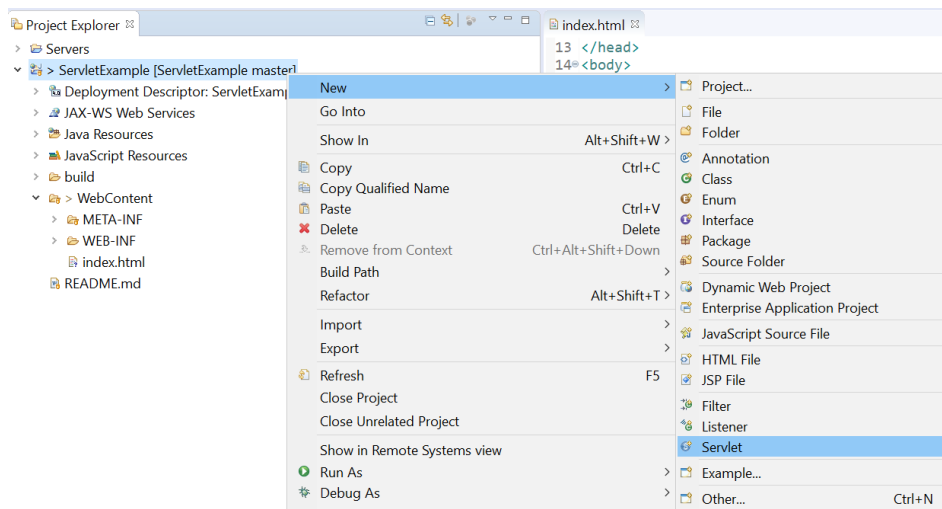
  <h2>Formulario GET Saludar</h2>
  <form action="ServletSaludo" method="get">
    <div class="form-group">
      <label for="nombre-get">Nombre</label>
      <input type="text" class="form-control" name="nombre" id="nombre-get">
    </div>
    <button type="submit" class="btn">Enviar</button>
  </form>

  <h2>Formulario POST</h2>
  <form action="ServletSaludo" method="post">
    <div class="form-group">
      <label for="nombre-post">Nombre</label>
      <input type="text" class="form-control" name="nombre" id="nombre-post">
    </div>
    <button type="submit" class="btn">Enviar</button>
  </form>

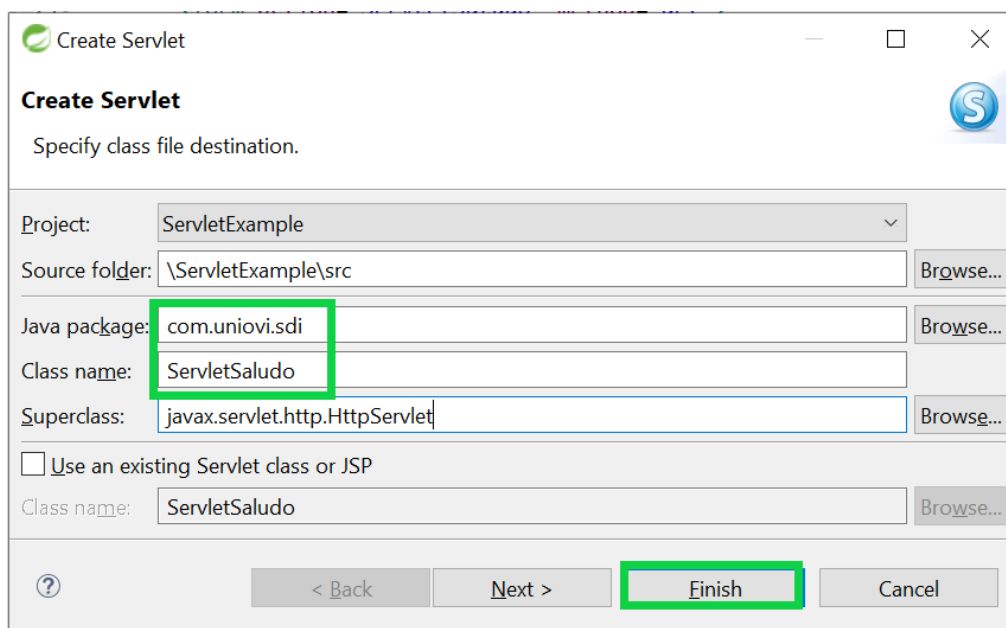
</div>

</body>
</html>
```

Para agregar un nuevo Servlet hacemos click derecho sobre el nombre del proyecto **New** → **Servlet**.



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



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

Implementamos una respuesta a **doGet()**, obteniendo el parámetro nombre. Esta función ya es capaz de responder a peticiones GET.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    // TODO Auto-generated method stub
    //response.getWriter().append("Served at: ").append(request.getContextPath());
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD><TITLE>Hola Mundo!</TITLE></HEAD>");
    out.println("<BODY>");
```



```
String nombre = (String) request.getParameter("nombre");  
if (nombre != null) {  
    out.println("Hola " + nombre + "<br>");  
}  
out.println("</BODY></HTML>");  
}
```

Nota: Hay que importar el paquete `java.io.PrintWriter`;

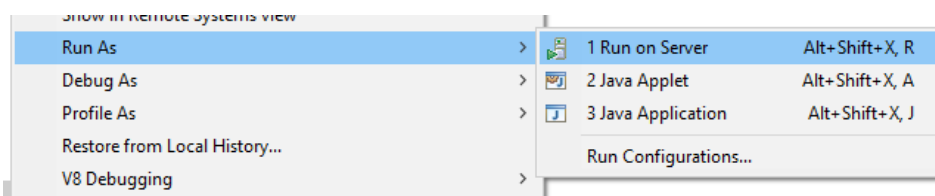
La función **doPost()** se encarga de que el servlet responda a peticiones POST. En este caso la función `doPost()` simplemente llama a función `doGet()`.

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

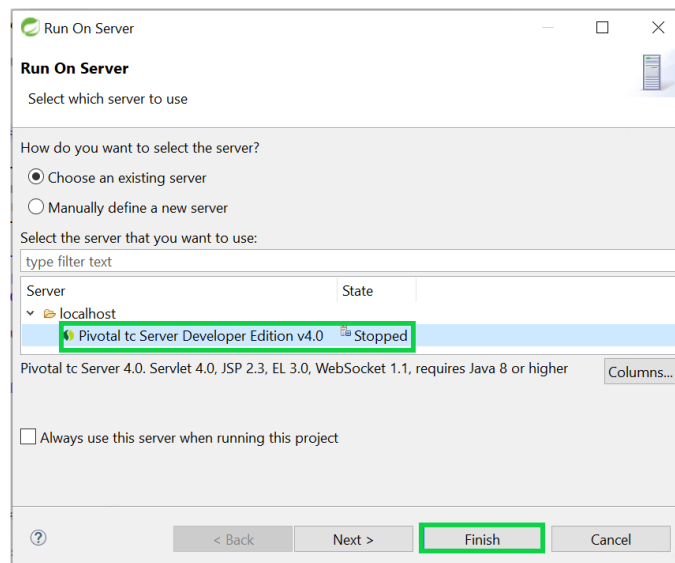
La anotación `@WebServlet` declara en el inicio de la clase nos sirve para definir la URL del servlet, en este caso `/ServletSaludo`, podremos enviarle peticiones a esta URL y el servlet responderá tanto a peticiones GET como POST.

```
@WebServlet("/ServletSaludo")  
public class ServletSaludo extends HttpServlet {
```

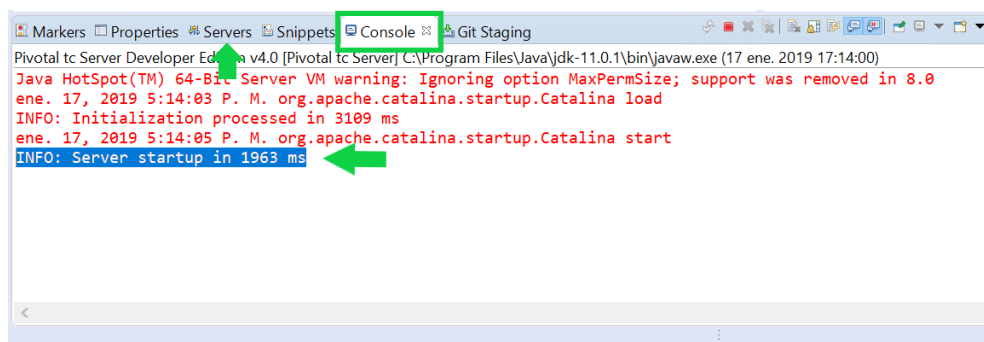
Pulsamos el botón derecho sobre el nombre del proyecto, seleccionamos **Run as → Run on Server**



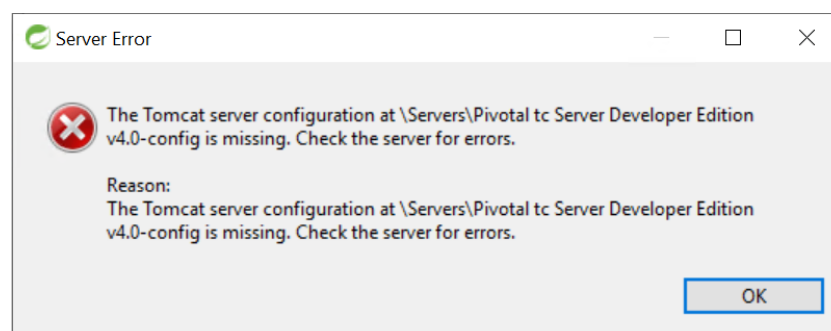
Seleccionamos el servidor interno **Pivotal tc Server Developer Edition** y pulsamos **Finish**.



Cuando el servidor esté iniciado veremos un mensaje “Server startup in xxx ms” en la pestaña **Console**, desde la pestaña **Servers** podemos gestionar el servidor.

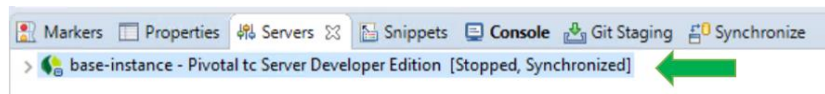


Nota: Si al ejecutar la aplicación nos ocurre el siguiente error, tendremos que eliminar el servidor actual y crear uno nuevo.

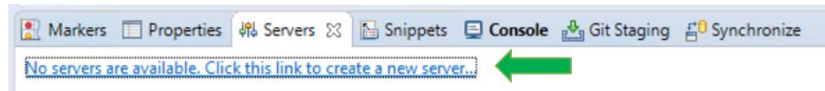


Para solucionar el error realizamos los siguientes pasos:

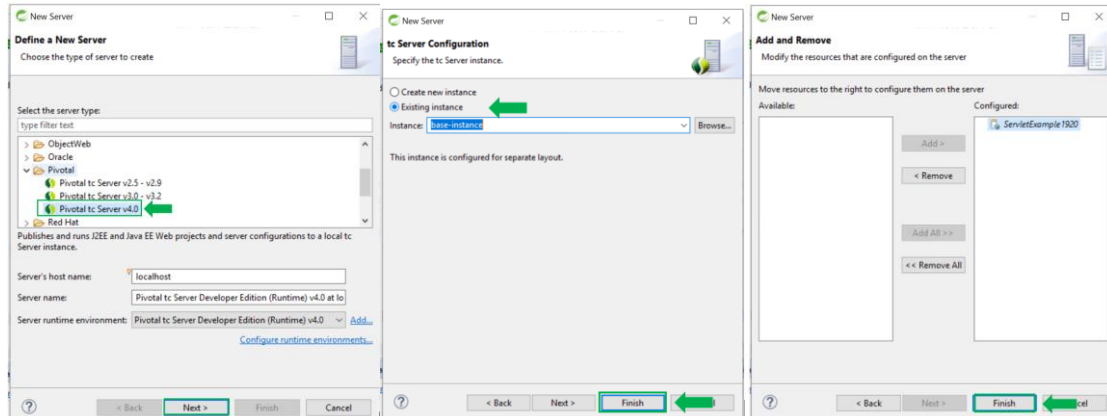
- 1- Vamos a la pestaña servers y borramos el server actual, click *derecho* -> *delete*



- 2- Hacemos click en el enlace *create a new server...*



- 3- Seleccionamos Pivotal tc Server v4.0



Una vez desplegado abrimos la aplicación desde la siguiente URL en un navegador, por ejemplo **Chrome**: <http://localhost:8080/ServletExample/>.

Formulario GET Saludar

Nombre

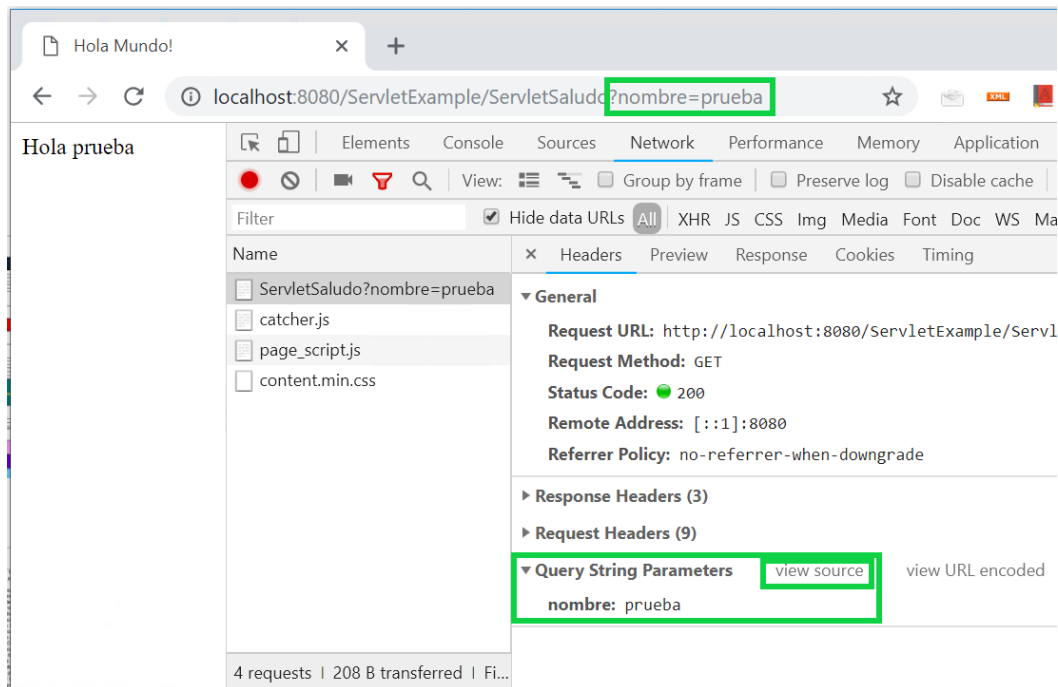
Enviar

Formulario POST

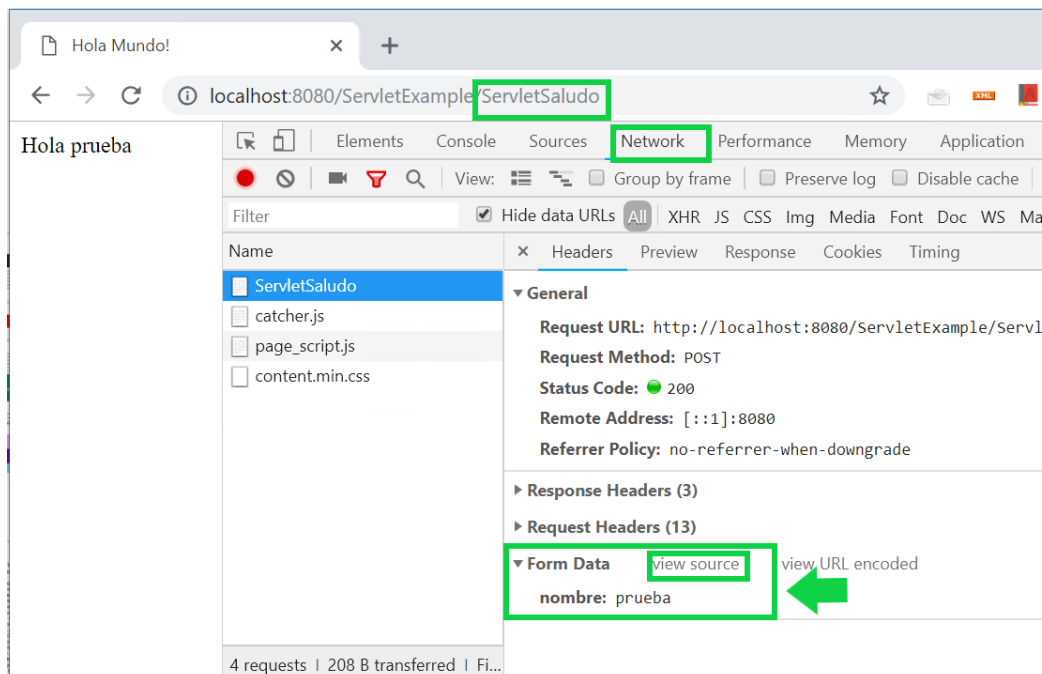
Nombre

Enviar

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.



2.1 Los Servlets son multihilo

Cada vez que el servidor accede al servlet este se ejecuta, si se reciben varias peticiones estas se ejecutaran 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 **ServletSaludo** añadimos el siguiente bloque de código.

```
@WebServlet("/ServletSaludo")
public class ServletSaludo extends HttpServlet {
    private static final long serialVersionUID = 1L;
    int contador = 0;

    public ServletSaludo() {
        super();
    }

    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>Hola Mundo!</TITLE></HEAD>");
        out.println("<BODY>");
        String nombre = (String) request.getParameter("nombre");
        if (nombre != null) {
            out.println("Hola " + nombre + "<br>");
        }
        try {
            Thread.sleep(15000);
        } catch (InterruptedException e) {}
        out.println("ID del hilo:"+Thread.currentThread().getId()+"<br>");
        contador++;
        out.println("Visitas:"+contador+"<br>");
        out.println("</BODY></HTML>");
    }
}
```

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/Servlets/ServletSaludo?nombre=prueba> y comprobamos que los servlets se ejecutan en diferentes hilos.

Hola prueba
ID del hilo:232
Visitas:2

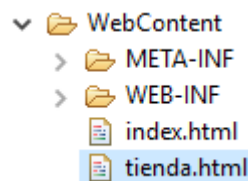
Nota: Subir el código a GitHub en este punto. Commit Message -> ***”SDI - 2.1 Los Servlets son multihilo.”***



2.2 Manejo de sesión

Los servlets proporcionan una solución simple para el **seguimiento de sesiones de usuario** 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. Creamos un nuevo fichero **tienda.html** en la carpeta **/WebContent** agregando el siguiente contenido:



```
<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/3.3.7/css/bootstrap.min.css"/>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</head>
<body>

<!-- Contenido -->
<div class="container" id="contenedor-principal">
  <h2>Productos</h2>
  <div class="row ">
    <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
      <div>
        
        <div>Manzanas</div>
        <a href="incluirEnCarrito?producto=manzanas" class="btn btn-default" >
          2.05 €
        </a>
      </div>
    </div>
    <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
      <div>
        
        <div>Fresas</div>
        <a href="incluirEnCarrito?producto=fresas" class="btn btn-default" >
          2.20 €
        </a>
      </div>
    </div>
    <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
      <div>
        
        <div>Naranjas</div>
        <a href="incluirEnCarrito?producto=naranjas" class="btn btn-default" >
          2.10 €
        </a>
      </div>
    </div>
    <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
```



```
<div>
  
  <div>Pan</div>
  <a href="incluirEnCarrito?producto=pan" class="btn btn-default" >
    0.80 €
  </a>
</div>
</div>
</div>
</body>
</html>
```

Crear una carpeta **img** para incluir las imágenes de la aplicación. Descargamos las imágenes desde **PL-SDI-Material1.zip** que hemos compartido y la copiamos en la carpeta img del proyecto.

```
▼ > WebContent
  ▼ > img
    iconfinder_apple.png
    iconfinder_bread.png
    iconfinder_orange.png
    iconfinder_strawberry.png
  > META-INF
  > WEB-INF
```

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

Para recibir esta petición URL vamos a crear un nuevo Servlet, **ServletCarrito**, al que le asociaremos la URL **"incluirEnCarrito"**.

```
package com.uniovi.sdi;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.HashMap;
import java.util.Map;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

@WebServlet("/incluirEnCarrito")
public class ServletCarrito extends HttpServlet {
    private static final long serialVersionUID = 1L;
```

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.

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

    HttpSession session=request.getSession();

    HashMap<String,Integer> carrito =
        (HashMap<String,Integer>) request.getSession().getAttribute("carrito");

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

    String producto = request.getParameter("producto");
    if (producto != null){
        insertarEnCarrito(carrito, producto);
    }

    response.setCharacterEncoding("UTF-8");
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD><TITLE>Tienda SDI: carrito</TITLE></HEAD>");
    out.println("<BODY>");
    out.println(carritoEnHTML(carrito)+"<br>");
    out.println("<a href='\"tienda.html\"'>Volver</a></BODY></HTML>");
}
```

Los métodos auxiliares que completan el servlet son:

```
private void insertarEnCarrito(Map<String,Integer> carrito, String claveProducto) {
    if (carrito.get(claveProducto)==null)
        carrito.put(claveProducto, new Integer(1));
    else {
        int numeroArticulos=(Integer)carrito.get(claveProducto).intValue();
        carrito.put(claveProducto,new Integer(numeroArticulos+1));
    }
}

private String carritoEnHTML(Map<String,Integer> carrito) {
    String carritoEnHTML="";

    for (String key:carrito.keySet())
        carritoEnHTML+="<p>["+key+", "+carrito.get(key)+" unidades</p>";
    return carritoEnHTML;
}
```

Nota: Si estamos usando una versión de java igual o superior a java 9 es posible que nos dé el siguiente warning *"The constructor Integer(int) is deprecated since version 9"*. Si esto ocurre podemos modificar la siguiente línea:

```
carrito.put(claveProducto, new Integer(1));
```

por

```
carrito.put(claveProducto, Integer.valueOf(1));
```

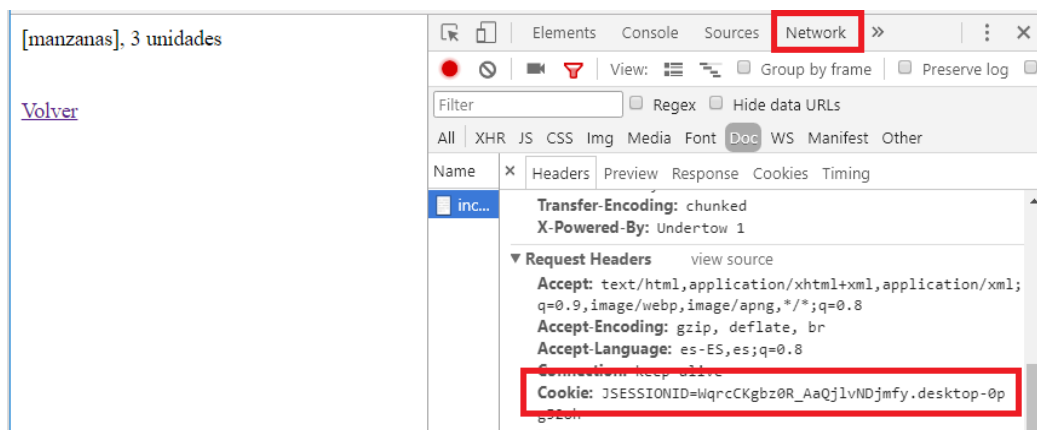


Ejecutamos la aplicación y comprobamos el correcto funcionamiento del carrito (accedemos a la web desde dos navegadores diferentes)
<http://localhost:8080/ServletExample/tienda.html>

Productos



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 `SynchronizedMap`⁴

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.

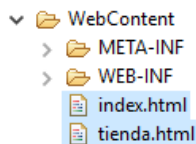
Nota: Subir el código a GitHub en este punto. Commit Message -> "SDI - 2.2 Manejo de sesiones."

⁴ <https://dzone.com/articles/java-7-hashmap-vs>

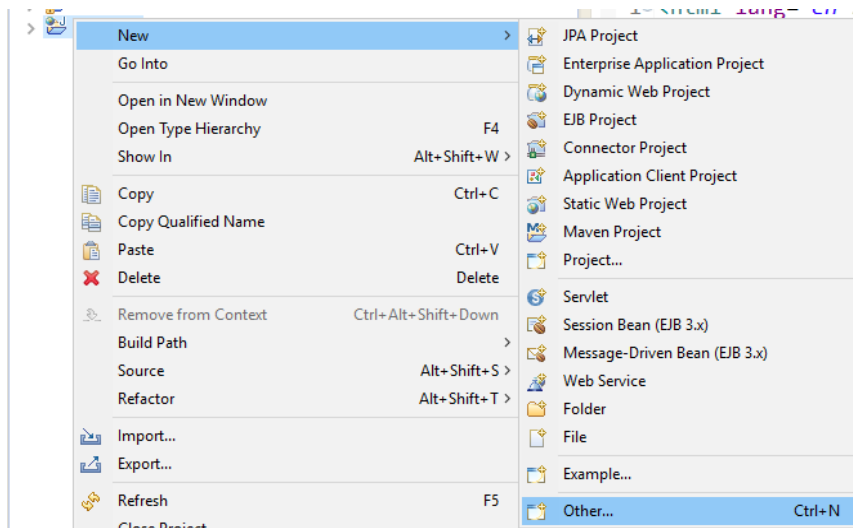


3 JSP Java Server Pages

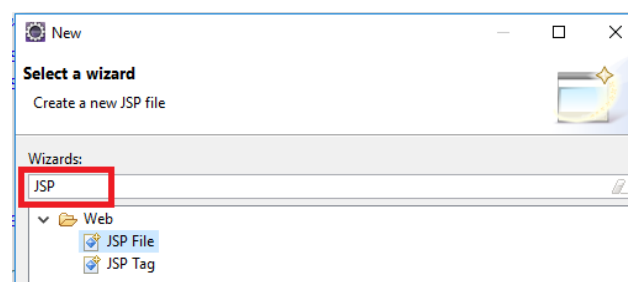
Vamos a implementar ahora la misma idea que hemos realizado con Servlets pero con JSPs. Sobre el mismo proyecto, eliminamos los dos ficheros **WebContent**, **index.html** y **tienda.html**.



A continuación, creamos un nuevo fichero JSP al que llamaremos **index.jsp**. Desde **New** → **Other...**



Buscamos el fichero de tipo **JSP File**.



Llamaremos al fichero **index.jsp** que, por defecto, se guardará dentro de la carpeta **/WebContent**

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 **ServletCarrito**.

```
<%@ 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>JSP</title>
<meta charset="utf-8"/>
<meta name="viewport" content="width=device-width, initial-scale=1"/>
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"/>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</head>
<body>

<!-- Contenido -->
<div class="container" id="contenedor-principal">
<h2>Productos</h2>
<div class="row">
<div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
<div>

<div>Manzanas</div>
<a href="incluirEnCarrito?producto=manzanas" class="btn btn-default">
2.05 €
</a>
</div>
</div>
<div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
<div>

<div>Fresas</div>
<a href="incluirEnCarrito?producto=fresas" class="btn btn-default">
2.20 €
</a>
</div>
</div>
<div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
<div>

<div>Naranjas</div>
<a href="incluirEnCarrito?producto=naranjas" class="btn btn-default">
2.10 €
</a>
</div>
</div>
<div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
<div>

<div>Pan</div>
<a href="incluirEnCarrito?producto=pan" class="btn btn-default">
0.80 €
</a>
</div>
</div>
</div>
</div>
</body>
</html>
```



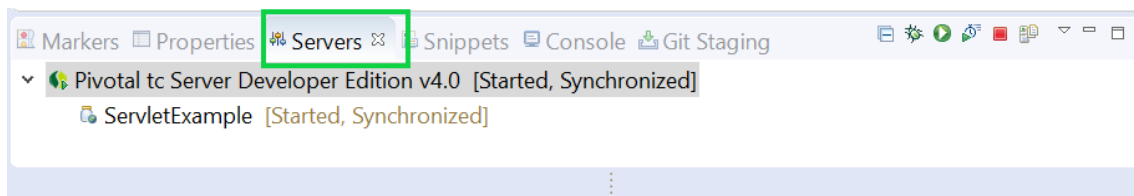
Modificamos la redirección final de la función **doGet()** en **ServletCarrito** para que nos retorne a **index.jsp**.

```
out.println(carritoEnHTML(carrito)+"<br>");  
out.println("<a href=\"index.jsp\">Volver</a></BODY></HTML>");  
}
```

Accedemos a <http://localhost:8080/ServletExample/> o <http://localhost:8080/ServletExample/index.jsp> para probar la modificación.

Sí se sigue cargando la versión anterior debido a la cache del navegador Actualiza la página (Control + F5) o abre una “Ventana de incognito”

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 Servers (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. Commit Message -> “SDI – 3 JSP Java Server Pages.”



3.1 Manejo de sesión (2)

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. 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>JSP</title>
    <meta charset="utf-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1"/>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"/>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</head>
<body>

<!-- Contenido -->
<div class="container" id="contenedor-principal">
    <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="nombre">Nombre:</label>
            <div class="col-sm-10">
                <input type="text" class="form-control" name="nombre" 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" 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**.



```
<body>

<%
    String nombre = request.getParameter("nombre");
    String password = request.getParameter("password");

    if ( nombre != null && nombre.equals("admin") &&
        password != null && password.equals("admin")){

        // Credencial valido, lo guardo en sesión
        request.getSession().setAttribute("usuario", "admin");
        response.sendRedirect("admin.jsp");

    } else {
        // Credencial invalido, lo elimino de sesion (opcional)
        request.getSession().setAttribute("usuario", null);
    }
%>

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

Cuando el usuario se identifica correctamente lo enviamos a **admin.jsp**. 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 retornamos al usuario al **login.jsp**.

- ▼ > WebContent
 - > > img
 - > > META-INF
 - > > WEB-INF
 - > > admin.jsp
 - > > index.jsp
 - > > 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>JSP</title>
    <meta charset="utf-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1"/>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"/>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/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");
    }
%>

<!-- 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 principal.

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.

```
Pivotal tc Server Developer Edition v4.0 [Pivotal tc Server] C:\Program Files\Java\jdk-11.0.1\bin\javaw.exe (17 ene. 2
Java HotSpot(TM) 64-Bit Server VM warning: Ignoring option MaxPermSize; support
ene. 17, 2019 7:25:38 P. M. org.apache.catalina.startup.Catalina load
INFO: Initialization processed in 3166 ms
ene. 17, 2019 7:25:40 P. M. org.apache.catalina.startup.Catalina start
INFO: Server startup in 2079 ms
Usuario en sesión: null
Usuario en sesión: null
Usuario en sesión: admin
```

**Nota: Subir el código a GitHub en este punto. Commit Message -
> “SDI - 2.2 Manejo de sesiones (2).”**



3.2 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 amarillo) en **index.jsp** antes de la definición del `<div class="container">`.

```
<%
    Integer contador = (Integer) application.getAttribute("contador");

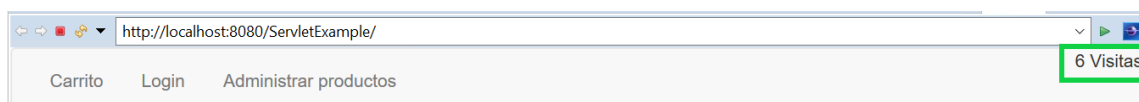
    if (contador == null) {
        contador = new Integer(0);
    }
    application.setAttribute("contador", contador.intValue() + 1);
%>

<!-- Barra de Navegación superior -->
<nav class="navbar navbar-default">
    <div class="container-fluid">
        <ul class="nav navbar-nav">
            <li><a href="incluirEnCarrito">Carrito</a></li>
            <li><a href="Login.jsp">Login</a></li>
            <li><a href="admin.jsp">Administrar productos</a></li>
        </ul>
        <div class="nav navbar-right">
            <%=contador%> Visitas
        </div>
    </div>
</nav>

<!-- Contenido -->
<div class="container" id="contenedor-principal">

    <h2>Productos</h2>
    <div class="row ">
```

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





3.3 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 **index.jsp**.

Creamos la clase **Producto** en el paquete **com.uniovi.sdi**.

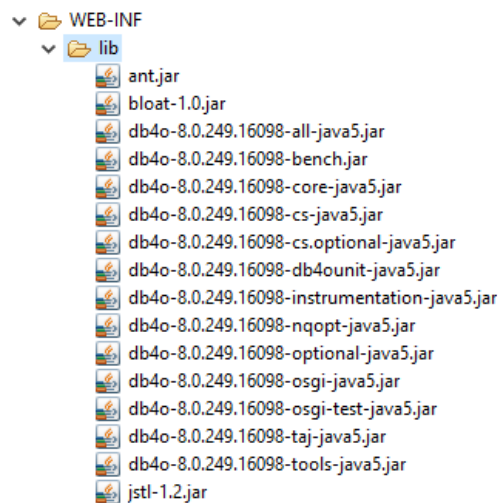
```
▼ > src
  ▼ > com.uniovi.sdi
    > Producto.java
    > ServletCarrito.java
    > ServletSaludo.java
```

```
package com.uniovi.sdi;
public class Producto {
    private String nombre;
    private String imagen;
    private float precio;

    public Producto(String nombre, String imagen, float precio) {
        this.nombre = nombre;
        this.imagen = imagen;
        this.precio = precio;
    }

    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getImagen() {
        return imagen;
    }
    public void setImagen(String imagen) {
        this.imagen = imagen;
    }
    public float getPrecio() {
        return precio;
    }
    public void setPrecio(float precio) {
        this.precio = precio;
    }
}
```

Descargamos el fichero **PL-SDI-Material1.zip** y copiamos todos los archivos jar en el directorio **/WebContent/WEB-INF/lib/**



Creamos la clase **ProductosService** y dentro de ella dos métodos: uno para retornar una lista con todos los productos en la base de datos (**getProductos**) y otro para agregar un nuevo producto (**setNuevoProducto**). 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 ProductosService {
    public List<Producto> getProductos(){
        List<Producto> productos = new LinkedList<Producto>();

        ObjectContainer db = null;
        try {
            db = Db4oEmbedded.openFile("bdProductos");
            List<Producto> respuesta = db.queryByExample(Producto.class);
            // NO RETORNAR LA MISMA LISTA DE LA RESPUESTA
            productos.addAll(respuesta);
        } finally {
            db.close();
        }

        return productos;
    }

    public void setNuevoProducto(Producto nuevoProducto){
        ObjectContainer db = null;
        try {
            db = Db4oEmbedded.openFile("bdProductos");
            db.store(nuevoProducto);
        } finally {
            db.close();
        }
    }
}
```



Volvemos a **index.jsp** y eliminamos todo el `<div class="container">` anterior en el que la lista de productos se especificaba en el propio HTML. Ahora obtendremos una instancia de **ProductosService** y recorreremos la lista que nos retorna (aunque la base de datos está actualmente vacía). Intercalamos el código Java con el código HTML.

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

    <h2>Productos</h2>
    <div class="row">

        <%
            List<Producto> listaProductos = new ProductosService().getProductos();
            for(Producto producto : listaProductos){
        %>
        <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
            <div>
                
                <div><%=producto.getNombre() %></div>
                <a href="incluirEnCarrito?producto=<%=producto.getNombre() %%" class="btn btn-
default" >
                    <%=producto.getPrecio() %> €
                </a>
            </div>
        </div>
        <%
            }
        %>
    </div>
</div>
```

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

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

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

3.4 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 contra **POST /admin.jsp**.

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

    <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="nombre">Nombre:</label>
            <div class="col-sm-10">
                <input type="text" class="form-control" name="nombre" required="true"/>
            </div>
        </div>
    </form>
</div>
```



```
</div>
</div>
<div class="form-group">
  <label class="control-label col-sm-2" for="imagen">URL imagen:</label>
  <div class="col-sm-10">
    <input type="text" class="form-control" name="imagen" required="true"/>
  </div>
</div>
<div class="form-group">
  <label class="control-label col-sm-2" for="precio">Precio (€):</label>
  <div class="col-sm-10">
    <input type="number" step="0.01" class="form-control" name="precio"
      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 **ProductosService** (en amarillo).

```
<%@ 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>JSP</title>
  <meta charset="utf-8"/>
  <meta name="viewport" content="width=device-width, initial-scale=1"/>
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"/>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/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("nombre") != null &&
        request.getParameter("imagen") != null &&
        request.getParameter("precio") != null ){

        String nombre = (String) request.getParameter("nombre");
        String imagen = (String) request.getParameter("imagen");
        float precio = Float.parseFloat(request.getParameter("precio"));

        Producto producto = new Producto(nombre, imagen, precio);
        new ProductosService().setNuevoProducto(producto);
        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/img/iconfinder_apple.png , 3
- Fresas , http://localhost:8080/ServletExample/img/iconfinder_strawberry.png , 2.5
- Naranjas , http://localhost:8080/ServletExample/img/iconfinder_orange.png , 2.10
- Pan , http://localhost:8080/ServletExample/img/iconfinder_bread.png , 0.80

Agregar producto a la tienda

Nombre:	<input type="text" value="Naranja"/>
URL imagen:	<input type="text" value="http://localhost:8080/ServletExample/img/iconfinder_orange.png"/>
Precio (€):	<input type="text" value="2.10"/>
<input type="button" value="Agregar"/>	

El fichero correspondiente a la base de datos se genera en la primera ejecución, en
`\sts-bundle\sts-3.9.7.RELEASE`

**Nota: Subir el código a GitHub en este punto. Commit Message -
> “SDI - 3.2 Contexto aplicación y 3.3 listado dinámico”**

3.4.1 JSP 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 Producto pueda ser utilizado como un Bean le tenemos que agregar un constructor sin argumentos.



```
public class Producto {  
    private String nombre;  
    private String imagen;  
    private float precio;  
  
    public Producto(){  
  
    }  
}
```

Modificamos **admin.jsp**, con la etiqueta **jsp:useBean** se crea un Bean nuevo de tipo **Producto**, con el nombre de variable “**producto**”.

Con la etiqueta **setProperty (con property=*)** analiza todos los parámetros de la 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** siempre va a contener un objeto **!= null** ya que se crea automáticamente con el **jsp:useBean**. Por lo tanto, no vale comprobar **producto != null**, debemos comprobar el valor de sus propiedades.

```
<jsp:useBean id="producto" class="com.uniovi.sdi.Producto" />  
<jsp:setProperty name="producto" property="*" />  
<%  
    if(producto.getNombre() != null){  
        new ProductosService().setNuevoProducto(producto);  
        request.getRequestDispatcher("index.jsp").forward(request, response);  
    }  
%>  
<%  
    if ( request.getParameter("nombre") != null &&  
        request.getParameter("imagen") != null &&  
        request.getParameter("precio") != null){  
  
        String nombre = (String) request.getParameter("nombre");  
        String imagen = (String) request.getParameter("imagen");  
        float precio = Float.parseFloat(request.getParameter("precio"));  
  
        Producto productoNuevo = new Producto(nombre, imagen, precio);  
        new ProductosService().agregarProducto(productoNuevo);  
        request.getRequestDispatcher("index.jsp").forward(request, response);  
    }  
%>  
<!-- Contenido -->  
<div class="container" id="contenedor-principal">
```

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

3.4.2 JavaBeans y ámbitos

Las directivas de JSP para manejar Beans nos permite 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 **Contador** en el paquete **com.uniovi.sdi**.

```
package com.uniovi.sdi;
public class Contador {
    private int total;

    public int getTotal() {
        return total;
    }

    public void setIncremento(int incremento) {
        total+=incremento;
    }
}
```

Sustituimos el anterior script Java por el nuevo Bean.

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

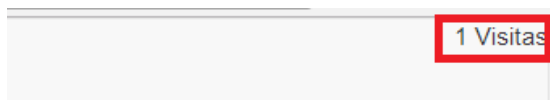
```
<%
    Integer contador = (Integer) application.getAttribute("contador");

    if (contador == null) {
        contador = new Integer(0);
    }
    application.setAttribute("contador", contador.intValue() + 1);
%>

<jsp:useBean id="contador" class="com.uniovi.sdi.Contador"/>
<jsp:setProperty name="contador" property="incremento" value="1"/>

<!-- Barra de Navegación superior -->
<nav class="navbar navbar-default">
    <div class="container-fluid">
        <ul class="nav navbar-nav">
            <li><a href="carrito.jsp">Carrito</a></li>
            <li><a href="login.jsp">Login</a></li>
            <li><a href="administrar.jsp">Administrar productos</a></li>
        </ul>
        <div class="nav navbar-right">
            <div class="center-block"> <%=contador%> <jsp:getProperty name="contador"
property="total"/> Visitas </div>
        </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 para todos los usuarios el ámbito del Bean sería “**application**”.

Añadimos el atributo **scope** a la etiqueta **useBean**.

```
<jsp:useBean id="contador" class="com.sdi.Contador" 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. Commit Message - > “SDI - 3.4 JSP Beans, JavaBeans y ámbitos”

3.5 Uso de tags JSTL Core

Las etiquetas JSTL (JavaServer Pages Standard Tag Library) encapsulan gran parte de funcionalidad común que se suele requerir en las páginas web JSP. Usando estas etiquetas se evita incluir scripts de código Java propios. <http://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/> 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).⁵

Para utilizar las etiquetas de JSTL debemos declarar el uso de JSTL, agregando la directiva al inicio de **index.jsp**.

```
<%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page language="java" 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 **ProductosService** como un Bean, obteniendo los artículos a través de productos (se invoca por detrás a **getProductos**).

⁵ Además para servidores que cumplen JEE 1.7 la librería jstl.jar ya está incluida en las librerías del servidor.



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

Tras estos cambios el nuevo código del **<div class=“container”>** pasará a ser el siguiente.

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

<h2>Productos</h2>
<div class="row ">

    <jsp:useBean id="productosService" class="com.uniovi.sdi.ProductosService"/>
    <c:forEach var="producto" begin="0" items="${productosService.productos}">
        <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
            <div>
                " /> />
                <div><c:out value="${producto.nombre}"/></div>
                <a href="incluirEnCarrito?producto=<c:out value="${producto.nombre}"/>"
                    class="btn btn-default" >
                    <c:out value="${producto.precio}"/> €
                </a>
            </div>
        </div>
    </c:forEach>
</div>
</div>
```

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.

```
<%
    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");
    }
%>

<c:if test = "${sessionScope.usuario != 'admin'}">
    <c:redirect url="/Login.jsp"/>
</c:if>

<jsp:useBean id="producto" class="com.sdi.Producto"/>
<jsp:setProperty name="producto" property="*" />

<%
    if( producto.getNombre() != null){
        new ProductosService().setNuevoProducto(producto);
        request.getRequestDispatcher("index.jsp").forward(request, response);
    }
%>
```



```
<c:if test = "${producto.nombre != null}">
  <jsp:useBean id="productosService" class="com.uniovi.sdi.ProductosService"/>
  <jsp:setProperty name="productosService" property="nuevoProducto"
value="${producto}"/>
  <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

Sí 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. Commit Message -
> "SDI - 3.5 Uso de tags JSTL Core"**



4 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. **ServletCarrito** va a continuar respondiendo a la petición **/incluirEnCarrito** 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 como atributo en la request con la clave “paresCarrito”
- Nuevo → redirige la petición a la vista **vista-carrito.jsp** (la vista puede utilizar la variable “paresCarrito” que acabamos de definir)

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

    HttpSession session=request.getSession();

    HashMap<String,Integer> carrito =
        (HashMap<String,Integer>) request.getSession().getAttribute("carrito");

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

    String producto = request.getParameter("producto");
    if ( producto != null){
        insertarEnCarrito(carrito, producto);
    }

    // Retornar la vista con parámetro "carrito"
    request.setAttribute("paresCarrito", carrito);
    getServletContext().getRequestDispatcher("/vista-carrito.jsp").forward(request,
response);
}
```

Creamos la vista **vista-carrito.jsp**, en ella hacemos uso de las etiquetas de JSTL para recorrer los elementos de la hashmap “**paresCarrito**”, como es una hashmap donde cada objeto tiene una **key** y un **value**.



```
<%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html lang="en">
<head>
<title>JSP</title>
<meta charset="utf-8"/>
<meta name="viewport" content="width=device-width, initial-scale=1"/>
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"/>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</head>
<body>

<!-- Contenido -->
<div class="container" id="contenedor-principal">
<h2>Vista-Carrito</h2>
<ul>
<li><c:forEach var="par" items="${paresCarrito}">
<tr>
<td><li>${par.key} - ${par.value} </li>
</tr>
</c:forEach>
</li>
</ul>
</div>






















</body>
</html>
```

**Nota: Subir el código a GitHub en este punto. Commit Message -
> "SDI – 4 MVC, Modelo Vista Controlador"**



4.1 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 – 4 MVC, Modelo Vista Controlador edwardnunez committed a minute ago	 833bd0a	
SDI - 3.5 Uso de tags JSTL Core. edwardnunez committed 9 minutes ago	 b753342	
SDI- 3.4 JSP Beans, JavaBeans y ámbitos. edwardnunez committed 35 minutes ago	 429dfb9	
SDI - 3.2 Contexto aplicación y 3.3 listado dinámico edwardnunez committed an hour ago	 b4681a7	
SDI-3.2 Contexto aplicación y 3.3 listado dinámico edwardnunez committed an hour ago	 2f4ae4e	
SDI - 2.2 Manejo de sesiones (2). edwardnunez committed 2 hours ago	 80ddb4b	
SDI – 3 JSP Java Server Pages. edwardnunez committed 2 hours ago	 c5a53bd	
SDI - 2.2 Manejo de sesiones. edwardnunez committed 2 hours ago	 bf07685	
SDI - 2.1 Los Servlets son multihilo. edwardnunez committed 3 hours ago	 b1c7090	
create servlet project edwardnunez committed 6 hours ago	 df13985	
Initial commit edwardnunez committed 8 hours ago	Verified  1b0d8ac	