

0101010  
0100101  
1101010

# UD10.2- PROGRAMACIÓN CON INTERFACES GRÁFICAS

Programación – 1º DAW/DAM

# 0. CONTENIDOS

---

- Introducción
  - Interfaces gráficas
  - Evolución de la web
  - Modelo **cliente–Servidor**
- Desarrollo de Aplicaciones en Java
  - ¿Qué es un Framework?
  - Aplicaciones Web con Spring Boot
- Creación de un **Proyecto Web** en **STS**
- Anatomía de un Aplicación Web
- Paso de Parámetros en las Peticiones
  - Método GET
  - Definición de Vistas y Recursos estáticos
  - Método POST
- Anotaciones
- Bibliografía

# 1. INTRODUCCIÓN

- Hasta el momento, **hemos interactuado** con nuestros programas java haciendo uso de la **entrada/salida** estándar.

```
private String getDescripcion(){
    System.out.println("Introduzca la Descripción");
    String descripcion = teclado.nextLine();
}
```

codigo	descripcion	fecha creación	fecha entrega
1	Estudiar Java	30-04-2021	11-11-2021

```
Vamos a introducir una nueva tarea
Introduzca la Descripción
Estudiar Java
Introduzca la fecha de entrega
02-05-2021 23:50:00
Tarea añadida con éxito
```

# 1. INTRODUCCIÓN

- En esta unidad **aprenderemos** a crear una **interfaz web** que permita al usuario **interactuar** con nuestros **programas**. Para ello, vamos a aprender a crear **páginas web dinámicas** y **aplicaciones web**.

## TODO

<input type="text" value="Add more stuff to the &lt;u&gt;TODO&lt;/u&gt; List..."/>			<input type="button" value="Add"/>
<input type="checkbox"/> Clean the bathroom	09-08-2019	<input type="button" value="Delete"/>	
<input type="checkbox"/> Water the plants	07-08-2019	<input type="button" value="Delete"/>	
<input type="checkbox"/> Feed the cat	01-08-2019	<input type="button" value="Delete"/>	

## BATBATCAR

<b>Login Page</b>	
<b>Usuario</b>	<input type="text" value="Usuario"/>
<b>Password</b>	<input type="text" value="Password"/>
<input type="button" value="Log In"/>	

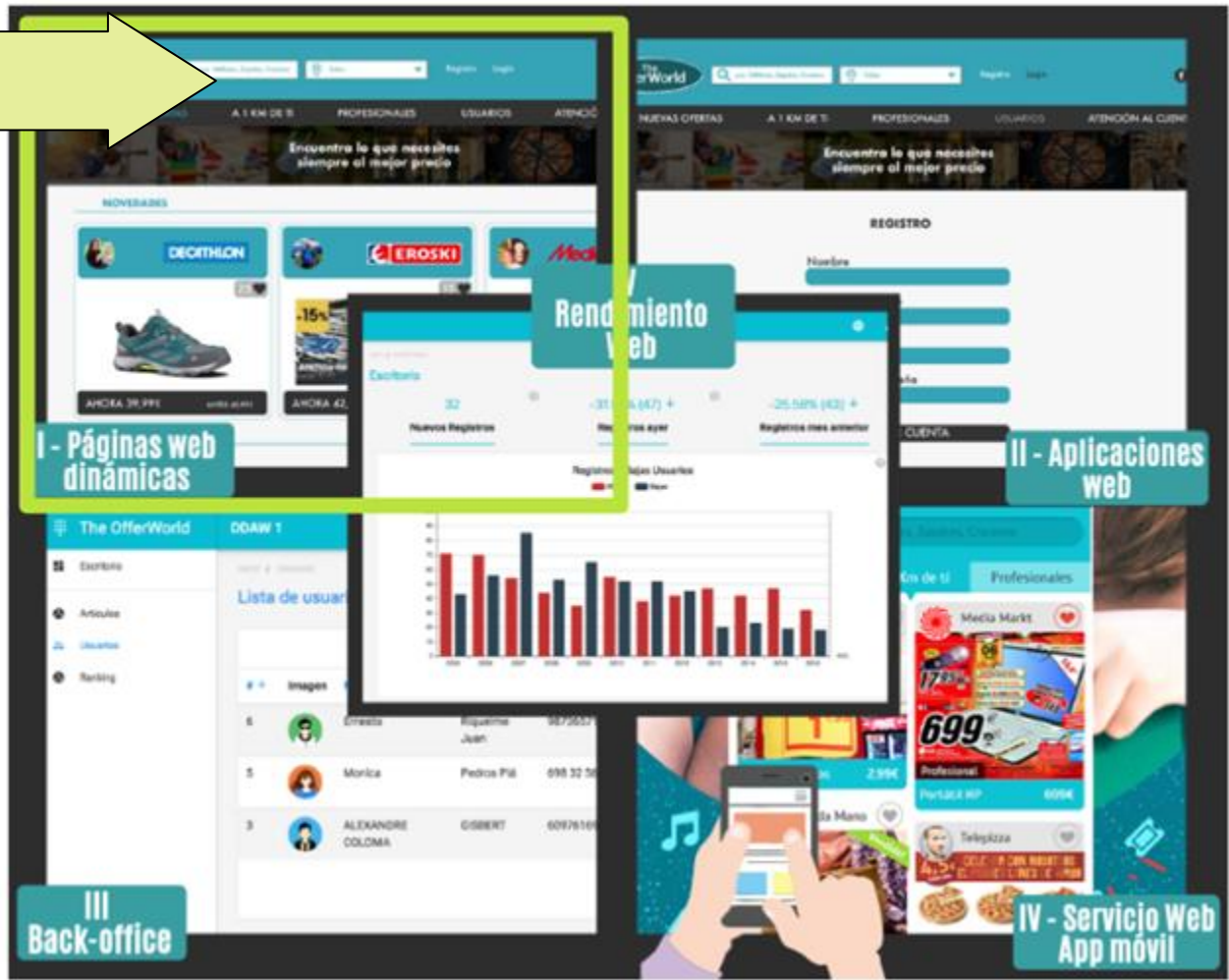
# 1.1 EVOLUCIÓN DE LAS PÁGINAS WEB

## Páginas

**Web Dinámicas:** páginas que cambian y se adaptan en función de la interacción con el usuario.

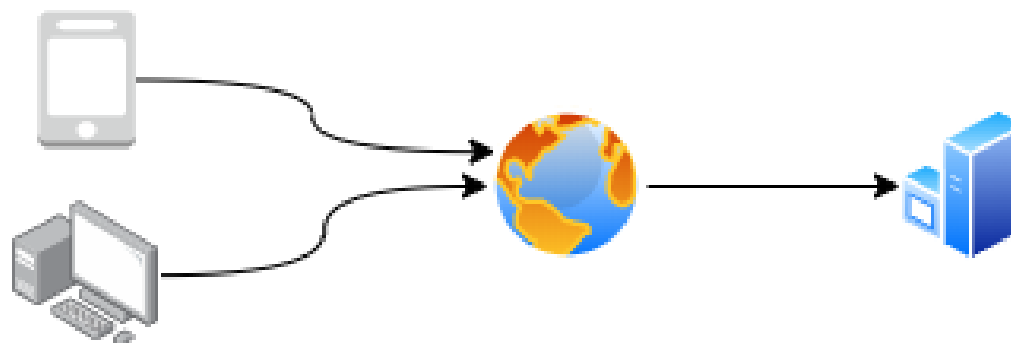
¿Cómo podríamos **mostrar** los productos cuyo nombre coincide con los **datos introducidos** en el **buscador**?

¿Y los productos de la **cesta de la compra**?



## 1.2 MODELO CLIENTE-SERVIDOR

- La arquitectura de la **World Wide Web** (WWW) provee un modelo de programación muy **poderoso y flexible**.
  - Mientras las **aplicaciones tradicionales** se ejecutan en la misma máquina en la que trabaja el usuario...
  - Las **aplicaciones web** se encuentran en **máquinas remotas** a las que se **accede** mediante una serie **de estándares** que hacen posible la comunicación y representación de la información.



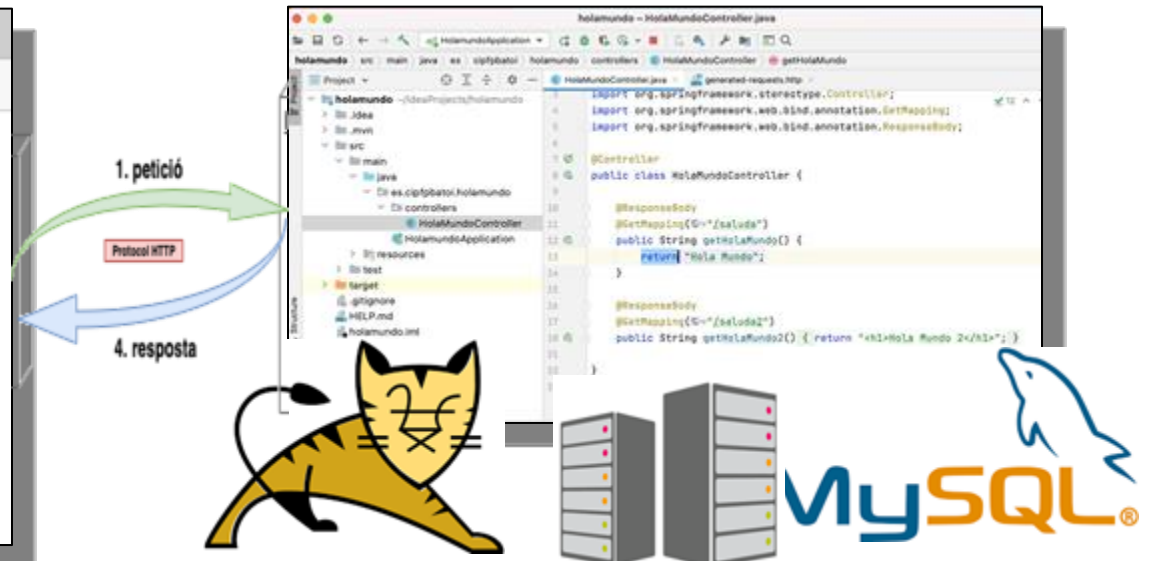
Arquitectura de la www

## 1.2 MODELO CLIENTE-SERVIDOR

- Las **arquitecturas web** se basan en el modelo **cliente / servidor**.
- Se trata de una **comunicación asimétrica** en la que uno de los extremos (**el Servidor**) ofrece uno o más servicios y el otro hace uso de ellos (**el Cliente**).



## 1.2 MODELO CLIENTE-SERVIDOR





## 2. DESARROLLO DE APLICACIONES WEB EN JAVA

- **Java provee** de diferentes entornos de trabajo (**Frameworks**) que nos van a permitir el desarrollo de Aplicaciones Web.
  - Estos hacen uso de la edición empresarial de Java, conocida como J2EE (Java 2 Enterprise Edition)
  - JSF (Java Server Faces), Spring, Servlets, JSP, **Spring Boot**, JSP, ...



## 2.1 ¿QUÉ ES UN FRAMEWORK?

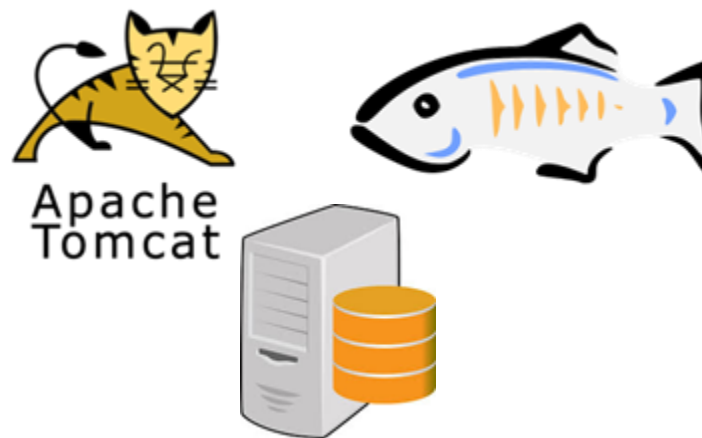
- Un **framework** es un marco de trabajo sobre el que desarrollar nuestra aplicación y que ofrece:
  - Una **estructura base** como punto de partida para elaborar tu proyecto.
  - Conjunto de **buenas prácticas** para el desarrollo de aplicaciones mediante la implementación de **patrones** y **estándares** de desarrollo.
  - Un **conjunto de herramientas** y componentes que nos **facilitan** implementar **tareas** comunes a todas las aplicaciones como son:
    - Seguridad.
    - Validación de formularios.
    - Acceso a datos
    - Autenticación
    - ...



## 2. APLICACIÓN WEB CON SPRING BOOT

---

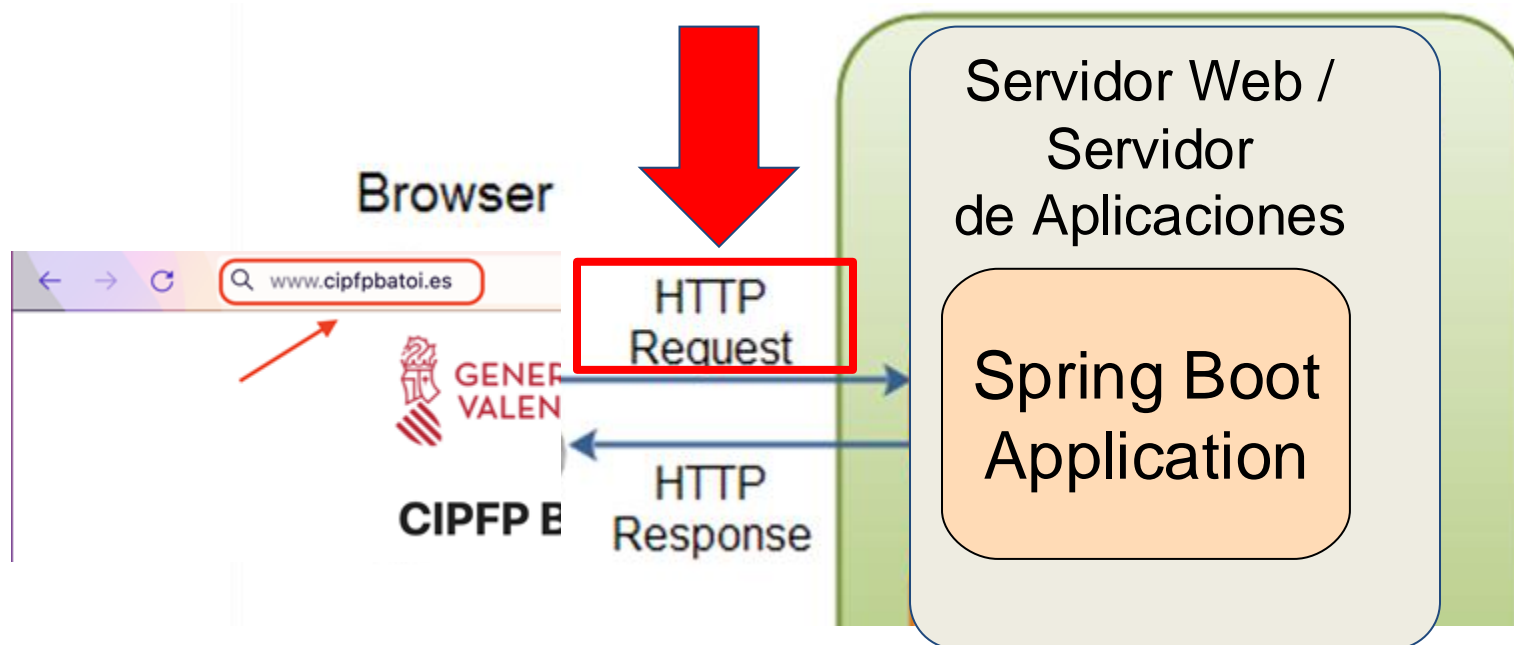
- **Es un programa Java que se ejecuta en un servidor de aplicaciones** (Apache Tomcat, GlassFish, etc.)
  - Este **servidor es proporcionado por el propio framework** e incluido en nuestra aplicación.



**Web Application  
Server**

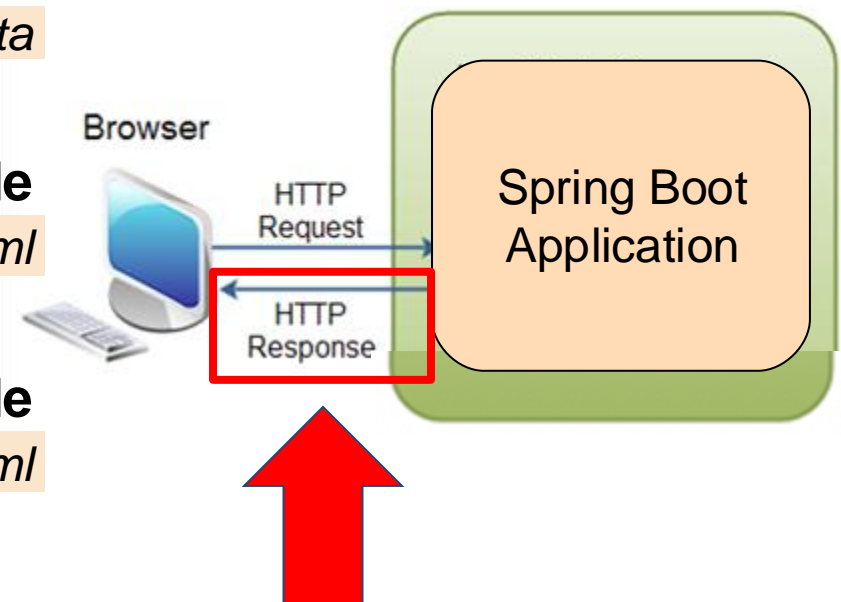
## 2.1 ¿EN QUÉ CONSISTE?

- Un **cliente** (por ejemplo **un navegador**) envía una **petición http** al **servidor** solicitando un recurso, por ejemplo, un **documento html**.
  - Esta **petición** se genera **automáticamente** al introducir la **URL** en el navegador y pulsar IR.

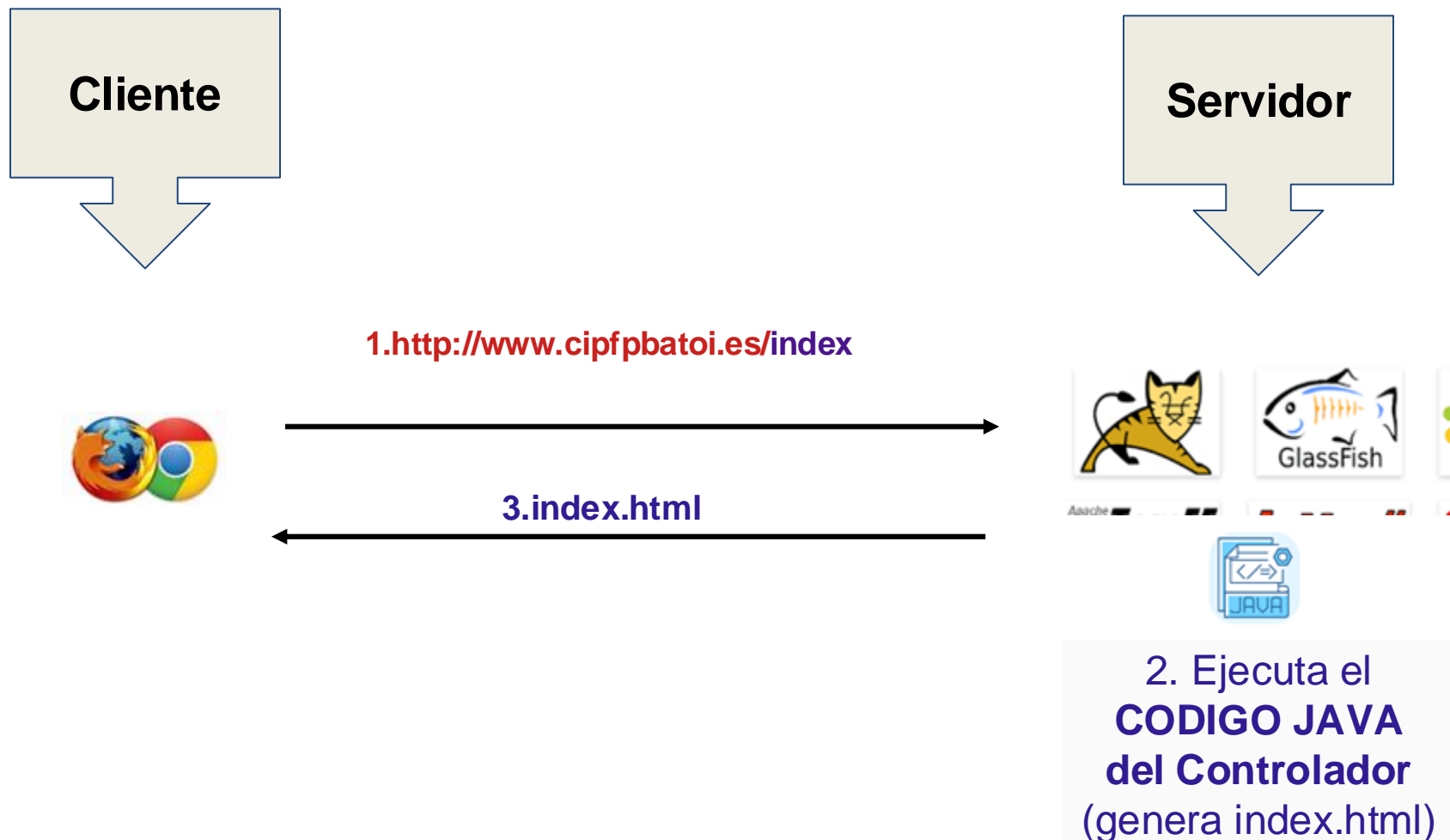


## 2.1 ¿EN QUÉ CONSISTE?

- El servidor recibe la **petición (HttpServletRequest)** del cliente, **ejecuta el código java** asociado **que realiza la tarea solicitada** y devuelve una respuesta http (**HttpResponse**) con el resultado. **Algunos ejemplos de tareas son :**
  - Generar un **documento HTML** a partir de los datos **almacenados en una BD** → *Devolverá un documento html en la respuesta http*
  - Crear un **nuevo registro** en la **Base de datos.** *Devolverá un documento html indicando si el resultado ha sido satisfactorio*
  - Modificar un **registro** en la **Base de datos.** *Devolverá un documento html indicando si el resultado ha sido satisfactorio*



## 2.1 ¿EN QUÉ CONSISTE?



### 3. CREACIÓN DE UN PROYECTO WEB EN STS

- En primer lugar, deberemos descargar el entorno de desarrollo STS (Spring Tool Suite) a través de [este enlace](#) (versión Linux para Eclipse). [Resto de versiones.](#)

#### Spring Tools 4 for Eclipse

The all-new Spring Tool Suite 4. Free.  
Open source.

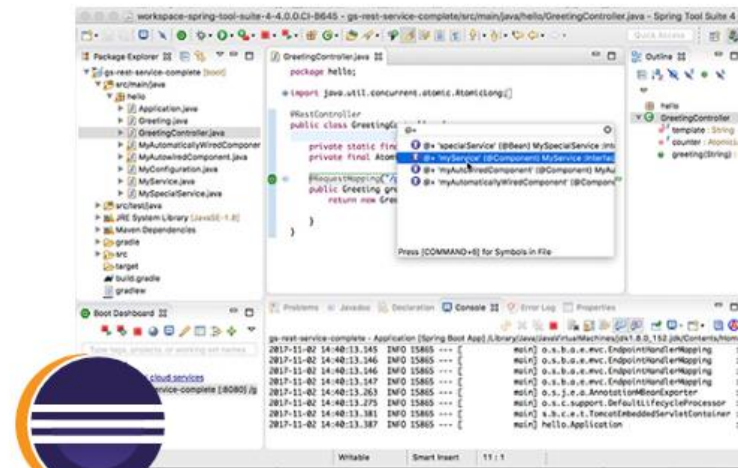
4.22.0 - LINUX X86\_64

4.22.0 - LINUX ARM\_64

4.22.0 - MACOS X86\_64

4.22.0 - MACOS ARM\_64

4.22.0 - WINDOWS X86\_64



## 3.1 CREACIÓN DEL PROYECTO

1. Seleccionamos el **menú File** → **New** → **Spring Starter Project**



2. Introducimos **los parámetros de configuración** que **identificarán a nuestra aplicación** y con los que ya estamos familiarizados.
- En la diapositiva siguiente se indican los parámetros que deberías indicar para comenzar el proyecto.



## 3.2 CREACIÓN DEL PROYECTO

**New Spring Starter Project**

Service URL:

Name:

☒ Use default location

Location:

Type:  Packaging:

Java Version:  Language:

Group:

Artifact:

Version:

Description:

Package:

Working sets

☐ Add project to working sets

Working sets:

## 3.2 CREACIÓN DEL PROYECTO

---

- 3. Como vamos a desarrollar una app web, en la posterior ventana seleccionaremos las siguientes dependencias:
  - **Developers Tools** → Spring Boot DevTools : conjunto de herramientas que facilitan el desarrollo de aplicaciones.
  - **Web** → Spring Web: Herramientas específicas para el desarrollo de aplicaciones web
  - **Thymeleaf**: Herramienta para la creación de vistas dinámicas

## 3.2 CREACIÓN DEL PROYECTO

### New Spring Starter Project Dependencies



Spring Boot Version: 3.2.5

Frequently Used:

☒ Spring Boot DevTools ☒ Spring Web ☒ Thymeleaf

Available:

Type to search dependencies

- AI
- Developer Tools
  - ☐ GraalVM Native Support
  - ☐ GraphQL DGS Code Generation
  - ☒ Spring Boot DevTools
  - ☐ Lombok
  - ☐ Spring Configuration Processor
  - ☐ Docker Compose Support
  - ☐ Spring Modulith
- Google Cloud
- I/O
- Messaging
- Microsoft Azure
- NoSQL
- Observability
- Ops
- SQL
- Security

Selected:

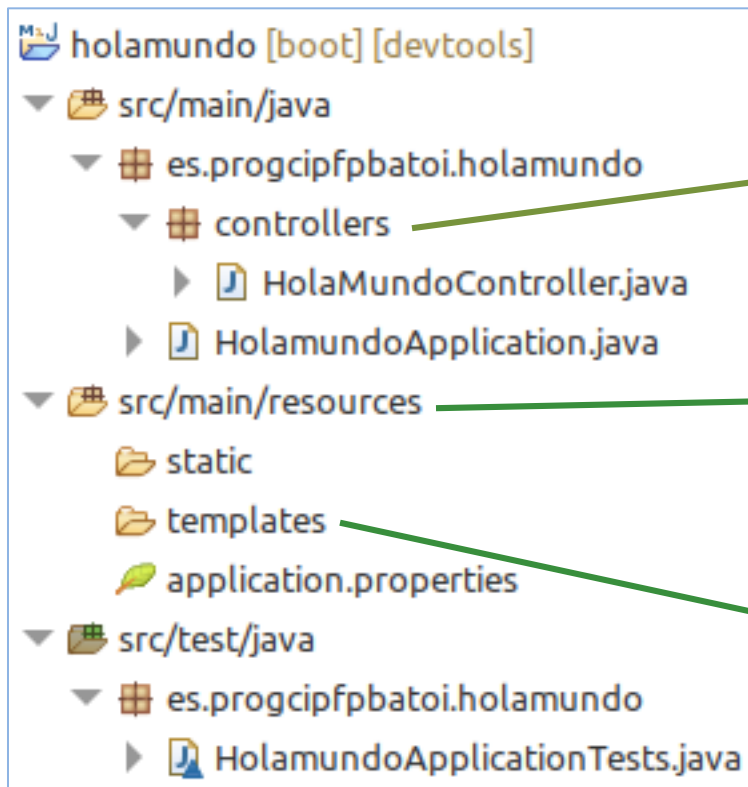
- X Spring Boot DevTools
- X Thymeleaf
- X Spring Web

Make Default Clear Selection

? < Back Next > Cancel Finish

Pulsamos *Finish* y se creará el proyecto.

## 3.3 ESTRUCTURA DE LA APLICACIÓN



- Carpeta que contendrá nuestros **controladores** (IMPORTANTE: se debe definir justo en esa ubicación)
- Carpeta que contendrá **nuestros recursos** que no necesitan ser compilados tales como; imágenes, ficheros de configuración,...
- Carpeta que contendrá **nuestras vistas**, es decir los **ficheros "html"**

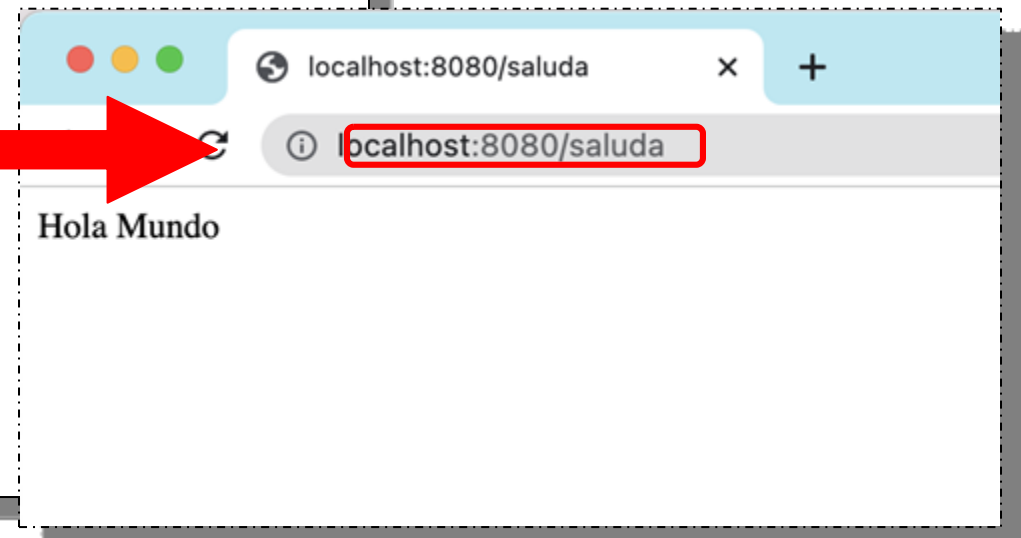
## 4. DEFINICIÓN DE UN CONTROLADOR

- Los controladores permiten definir **puntos de acceso (path)** a nuestra aplicación (**endpoints**).
  - Cada uno de los **puntos de acceso** serán **accesibles** a través de una **url**.

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class HolaMundoController {

    @ResponseBody
    @GetMapping("/saluda")
    public String getHolaMundo() {
        return "Hola Mundo";
    }
}
```



## 4.1 ANOTACIONES

- Al arrancar la aplicación, **Spring Framework** analiza el código fuente, en **busca de anotaciones** (empiezan por @ y definiremos más en profundidad en una próxima diapositiva), que van indicando cómo debe comportarse la aplicación (anotaciones como `@SpringBootApplication`, `@Controller`, `@ResponseBody`, `@GetMapping`...)

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ResponseBody;
```

```
@Controller
public class HolaMundoController {
```

Indica que esta clase es un controlador que definirá los **puntos de acceso a la aplicación**

```
@ResponseBody
@GetMapping("/saluda")
public String getHolaMundo() {
    return "Hola mundo";
}
```

Indica que lo que **devuelve** el método debe ser **enviado directamente** al navegador

La dirección de acceso al **endpoint** será <https://127.0.0.1:8080/saluda>

## 4.1.2 INICIO DE LA APLICACIÓN

Como es habitual, para iniciar la **aplicación**, deberemos **ejecutar** la clase que disponga del método `main()`.

- Necesitamos decirle a **Spring Boot** cual es la **clase principal** por lo se utiliza la anotación `@SpringBootApplication`.


```
package es.progcipfpbatoi.holamundo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class HolamundoApplication {

    public static void main(String[] args) {
        SpringApplication.run(HolamundoApplication.class, args);
    }

}
```



Indica que esta clase es la **clase principal** de la aplicación

## 4.1.3 EJECUCIÓN DE LA APLICACIÓN

- Para ejecutar la aplicación nos situaremos con el ratón encima del nombre del proyecto (árbol de proyectos de la parte izquierda) y con el botón derecho seleccionaremos la opción "**Run As --> Spring Boot App**"
- Se pondrá en marcha el servidor **Tomcat** integrado con el entorno de desarrollo. Deberías ver algo similar a esto en consola:



```

:: Spring Boot :: (v3.2.4)

2024-04-10T16:05:34.428+02:00 INFO 7798 --- [holamundo] [ restartedMain] e.p.holamundo.HolamundoApplication : Starting Holamundo
2024-04-10T16:05:34.431+02:00 INFO 7798 --- [holamundo] [ restartedMain] e.p.holamundo.HolamundoApplication : No active profile
2024-04-10T16:05:34.469+02:00 INFO 7798 --- [holamundo] [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property
2024-04-10T16:05:34.470+02:00 INFO 7798 --- [holamundo] [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional we
2024-04-10T16:05:35.218+02:00 INFO 7798 --- [holamundo] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialize
2024-04-10T16:05:35.225+02:00 INFO 7798 --- [holamundo] [ restartedMain] o.apache.catalina.core.StandardService : Starting service
2024-04-10T16:05:35.226+02:00 INFO 7798 --- [holamundo] [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet
2024-04-10T16:05:35.248+02:00 INFO 7798 --- [holamundo] [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Sprin
2024-04-10T16:05:35.248+02:00 INFO 7798 --- [holamundo] [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicati
2024-04-10T16:05:35.450+02:00 INFO 7798 --- [holamundo] [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server
2024-04-10T16:05:35.469+02:00 INFO 7798 --- [holamundo] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on
2024-04-10T16:05:35.478+02:00 INFO 7798 --- [holamundo] [ restartedMain] e.p.holamundo.HolamundoApplication : Started Holamundo

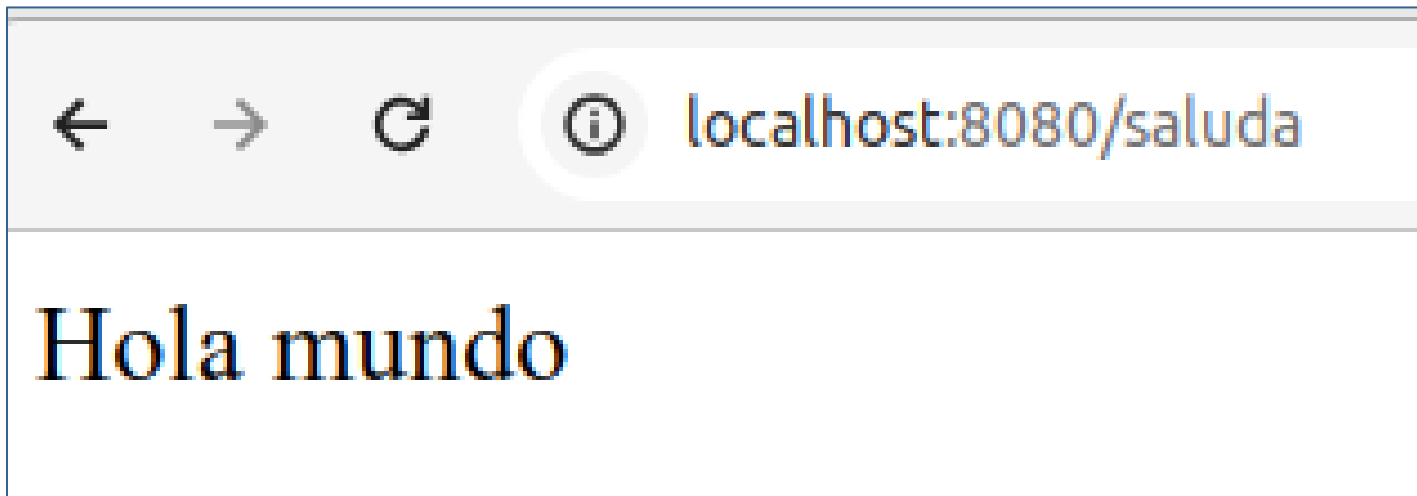
```

Aplicación arrancada esperando peticiones



## 4.1.3 EJECUCIÓN DE LA APLICACIÓN

- Ahora sólo queda acceder a través de cualquier navegador web a la siguiente URL:



# ACTIVIDADES PREVIAS

**Actividad 1.-** Observa con atención el **siguiente código** y contesta:

- ¿A qué **dirección deberíamos acceder** en el navegador suponiendo que la aplicación se está ejecutando en local?
- ¿Qué se mostraría en el navegador?

```
@Controller
public class TestController {

    @ResponseBody
    @GetMapping("/test")
    public String getTest() {
        Random random = new Random();
        int numero = random.nextInt(101);
        return "<html><body>" +
            "<h1> Bola: " + numero + "</h1>" +
            "</body></html>";
    }
}
```

# ACTIVIDADES PREVIAS

---

**Actividad 2.-** Crea un nuevo proyecto de tipo **Spring Starter Project** y define un controlador `Actividad2Controller` con los siguientes **endpoints**:

**Url:** /saluda

**Contenido:** Hola Mundo

**Url:** /bingo

**Contenido:** Número aleatorio entre 1 i 99

**Url:** /fecha-actual-es

**Contenido:** Fecha y hora actual en formato español dd-mm-yyyy hh:mm:ss

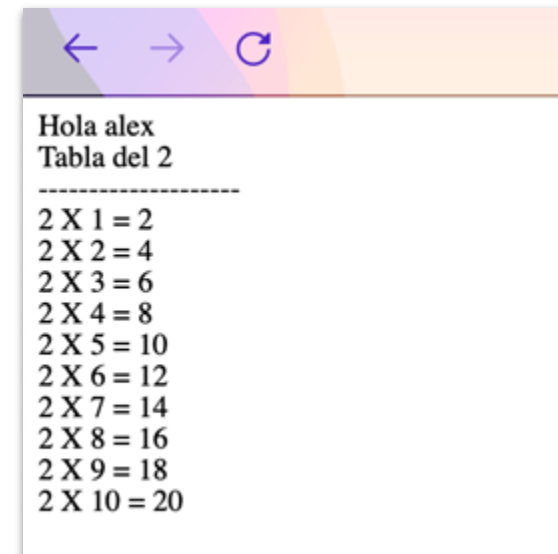
**Url:** /fecha-actual-en

**Contenido:** Fecha y hora actual en formato inglés yyyy-mm-dd hh:mm:ss

## 5. Paso de parámetros en las peticiones

- En **muchas ocasiones**, los **endpoints/recursos** van a necesitar información del cliente para **llevar a cabo la operación**. Por ejemplo:
  - Indicar el **código del pedido** del que necesitamos consultar la información: *(p1, p2, p3,...)*
  - Indicar de **qué número** queremos mostrar la **tabla de multiplicar**; *(1,2,3,4,...)*
  - Indicar **parámetros** para realizar la búsqueda de un **artículo**: *marca de ropa, nombre de un libro, editorial,...*

Para llevar a cabo **dichas acciones**,  
**los clientes hacen** uso  
de los FORMULARIOS (para obtener datos) y  
de ENLACES (con parámetros).



## 5.1. Paso de parámetros en peticiones (GET)

- Cuando solicitamos una nueva **página (GET)**, podemos incluir **parámetros**.
  - Los parámetros viajan en la URL en forma de pares **clave/valor**.

Delimitador  
de  
parámetro

Delimita dónde  
empiezan  
los **parámetros**

<http://localhost:8080/tabla-multiplicar?num=1&nombreUsuario=Sergio>

Parámetro **num**  
con valor **1**

Parámetro  
**nombreUsuario**  
con valor **Sergio**

HashMap<String,String>

num	1
nombreUsuario	Sergio

## 5.1. Paso de parámetros en peticiones (GET)

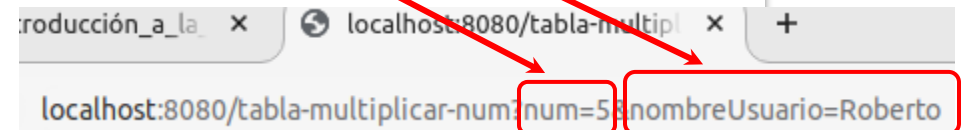
- Al definir el **nuevo endpoint** debemos indicarle al controlador que la **llamada al recurso** espera una serie de **parámetros** y que debe recuperarlos.

```
@Controller
public class ExampleController {

    @GetMapping(value = "/tabla-multiplicar-num")
    @ResponseBody
    public String getTablaMultiplicar(@RequestParam Map<String, String> params) {
        String numeroString = params.get("num");
        String nombreUsuario = params.get("nombreUsuario");
        StringBuilder stringBuilder = new StringBuilder("Hola " + nombreUsuario);

        if (numeroString != null && numeroString.matches("[1-9]")){
            int numero = Integer.parseInt(params.get("num"));
            // Mostrar Tabla de multiplicar
        }

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



## 5.1. Paso de parámetros en peticiones (GET)

- Como alternativa, si el número de parámetros que puede recibir el controlador es reducido podemos indicar el tipo del dato esperado junto al nombre de la variable que lo almacenará (*desventaja: si los parámetros no se especifican al llamar al recurso, se devolverá un error al navegador*)

```
@Controller
public class ExampleController {

    @GetMapping(value = "/tabla-multiplicar-num")
    @ResponseBody
    public String getTablaMultiplicar(@RequestParam int num,
                                     @RequestParam String nombreUsuario))

        StringBuilder htmlResponse = new StringBuilder();
        htmlResponse.append(String.format("Hola %s <br>", nombreUsuario));
        htmlResponse.append(String.format("Tabla del %s <br>", num));
        htmlResponse.append("-----<br>");
        for (int i = 1; i <= 10 ; i++) {
            htmlResponse.append(String.format("%d X %d = %d <br>", num, i, num *
i ));
        }
        return htmlResponse.toString();
    }
}
```

param num

param nombreUsuario

localhost:8080/tabla-multiplicar-num?num=5&nombreUsuario=Roberto

## 5.2. Definición de vistas

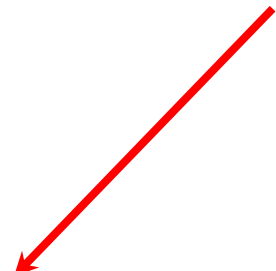
El **acceso a los controladores NO** suele llevarse a cabo **introduciendo directamente la url** y los **parámetros** ya que el usuario no conoce qué parámetros espera el controlador (esto lo hacemos en desarrollo para hacer pruebas). Podemos utilizar **distintas técnicas**.

- **Definición de enlaces**; el usuario no puede cambiar el valor de los parámetros si no que estos están **preestablecidos** en el **documento html** mediante enlaces.

### Tablas de Multiplicar

- [Ver Tabla del 1](#)
- [Ver Tabla del 2](#)
- [Ver Tabla del 4](#)
- [Ver Tabla del 6](#)
- [Ver Tabla del 8](#)
- [Ver Tabla del 10](#)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Tabla de Multiplicar</title>
</head>
<body>
<p>Tablas de Multiplicar enlaces</p>
<ul>
  <li><a href="/tabla-multiplicar-num?num=1&nombreUsuario=batoi">Ver Tabla del 1</a></li>
  <li><a href="/tabla-multiplicar-num?num=2&nombreUsuario=batoi">Ver Tabla del 2</a></li>
  <li><a href="/tabla-multiplicar-num?num=4&nombreUsuario=batoi">Ver Tabla del 4</a></li>
  <li><a href="/tabla-multiplicar-num?num=6&nombreUsuario=batoi">Ver Tabla del 6</a></li>
  <li><a href="/tabla-multiplicar-num?num=8&nombreUsuario=batoi">Ver Tabla del 8</a></li>
  <li><a href="/tabla-multiplicar-num?num=10&nombreUsuario=batoi">Ver Tabla del 10</a></li>
</ul>
</body>
</html>
```






## 5.2 Definición de vistas II

- Definición de formularios:** el valor de los parámetros es asignado por el usuario de forma dinámica al introducir los datos y pulsar sobre el botón ENVIAR (submit).

```

<html>
<body>
<p>Tabla de Multiplicar Formulario</p>
<form action="/tabla-multiplicar-num" method="GET">
  <label>Número
    <input name="num" type="number" required>
  </label>
  <label>
    Nombre Usuario
    <input name="nombreUsuario" type="text" required>
  </label>
  <input type="submit" value="Ver Tabla">
</form>
</body>
</html>

```



El atributo **action** indica el controlador al que se debe dirigir la petición

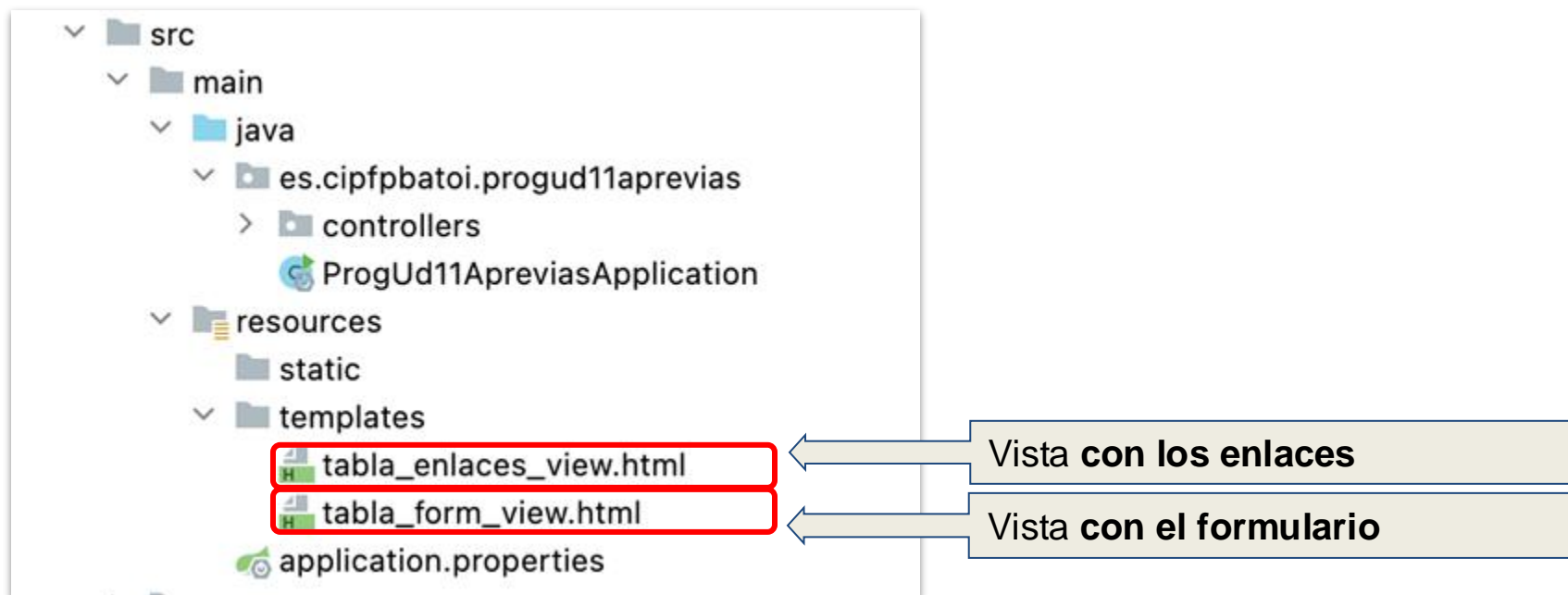
El atributo **name** de cualquier elemento **input** indica el nombre del parámetro que recibirá el controlador.

**Se reciben 2 parámetros:**

- num** → numérico
- nombreUsuari** → string

## 5.2 Definición de vistas III

- **Ubicación:** Estas vistas son simplemente **documentos html** que guardamos en la carpeta /resources/templates



## 5.2 Definición de vistas IV

- **Definición de controlador.** Definiremos un endpoint por cada una de las **vistas** a las que queremos acceder.

```
@Controller
public class ExampleController {
    @GetMapping("/enlaces-view")
    public String tablaEnlacesViewAction(){
        return "tabla_enlaces_view";
    }

    @GetMapping("/form-view")
    public String tablaMultiplicarForm() {
        return "tabla_form_view";
    }
}
```

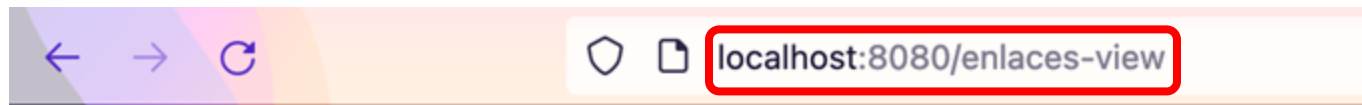
**@ResponseBody**

**Nombre de la vista**  
que queremos mostrar  
**sin extensión**

**Nombre de la vista** que  
queremos mostrar **sin**  
**extensión**

## 5.2 Definición de vistas V

Una vez **definido el controlador** podremos **acceder** a las diferentes vistas.



Tablas de Multiplicar enlaces

- [Ver Tabla del 1](#)
- [Ver Tabla del 2](#)
- [Ver Tabla del 4](#)
- [Ver Tabla del 6](#)
- [Ver Tabla del 8](#)
- [Ver Tabla del 10](#)



Al **pulsar** sobre los **enlaces** o el **botón “Ver Tabla”** realizaremos una llamada a los controladores encargados de **mostrar la tabla de multiplicar**.

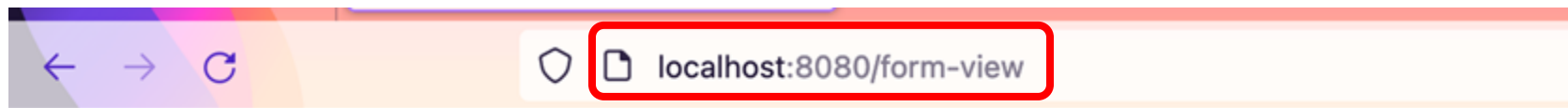


Tabla de Multiplicar Formulario

Número  Nombre de Usuario

# ACTIVIDADES PREVIAS

**Actividad 3.-** Crea un nuevo controlador `Actividad3Controller` y define los siguientes recursos (con sus parámetros) que nos permita llevar a cabo **operaciones entre 2 números**.

**URL:** /suma

**parámetro:** sumando1

**parámetro:** sumando2

**Contenido a mostrar:** Sumando1+sumando2 = resultado

Ej.  $4+3 = 7$

**URL:** /multiplica

**parámetro:** multiplicador

**parámetro:** multiplicando

**Contenido a mostrar:** multiplicador\*multiplicando = producto

Ej.  $4*2 = 8$

**URL:** /resta

**parámetro:** operando1

**parámetro:** operando2

**Contenido a mostrar:** operando2 - operando1 = resultado1

operando2-operando1 = resultado2

Ej.  $4-3 = 1$

$3-4 = -1$

# ACTIVIDADES PREVIAS

**Actividad 3.- (continuación).**- Crea una nueva llamada en el controlador llamada `/calculadora` que se encargue de cargar una vista `calculadora.html` que nos permita probar cada uno de los controladores anteriores. El resultado deberá ser como el siguiente.

## Calculadora

**Prueba de algunos enlaces fijos**

- [Ver resultado Suma 4 + 5](#)
- [Ver resultado Multiplica 4 \\* 2](#)
- [Ver resultados Resta 3 y 2](#)

**Prueba Formulario suma**

sumando 1  sumando 2

**Prueba Formulario resta**

operando 1  operando 2

**Prueba Formulario multiplica**

multiplicador  multiplicando

Lamará al controlador  
**/suma**

Lamará al controlador  
**/resta**

Lamará al controlador  
**/multiplica**

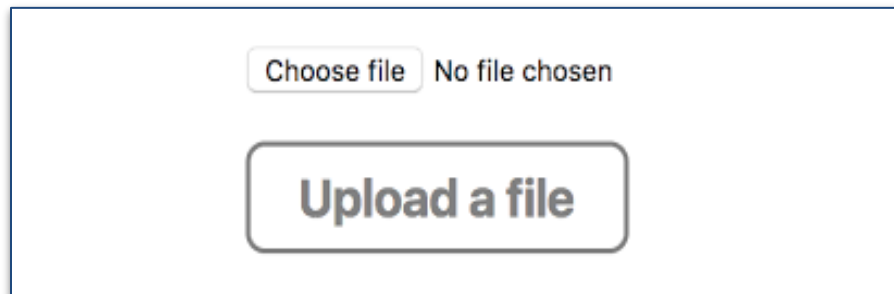


## 6. Paso de parámetros por POST

Ya **sabemos cómo solicitar datos** (páginas web) **al servidor** .

En muchas ocasiones, necesitamos **realizar acciones** que nos permitan **crear/guardar** datos de **entidades** en la aplicación (con mucha información y variada). Como ejemplo podemos citar:

- Añadir un **nuevo tipo de viaje**
- Añadir una **venta** de un **videojuego**
- Añadir **datos de registro de un usuario**
- Crear una **nuevo pedido**
- Añadir una **imagen** a nuestro **perfil de instagram**

A screenshot of a web form for file upload. It features a light gray rectangular container. At the top, there is a small button labeled 'Choose file' followed by the text 'No file chosen'. Below this, centered, is a larger, rounded rectangular button with the text 'Upload a file'.

## 6. Paso de parámetros por POST

☐ **m9** - Pulled pork y guacamole - 1.25 €

☐ **m10** - PULLED PORK y queso brie - 1.25 €

☐ **m11** - FILETE RUSO, cebolla caramelizada y salsa de qu

☐ **m12** - SALMÓN AHUMADO y crema de queso - 1.25 €

☐ **m13** - CARNE MECHADA DESHILACHADA y cebolla

☐ **m14** - JAMÓN GRAN RESERVA, tomate y aceite de oliv

☐ **m15** - CARRILLERA AL VINO TINTO y queso ibérico -

☐ **m16** - QUESO IBÉRICO, tortilla de patatas y mayonesa -

☐ **m17** - ALBÓNDIGAS y salsa BBQ - 1.25 €

☐ **m18** - Pollo, cebolla caramelizada y mayonesa trufada - 1.

☐ **m19** - CHISTORRA, bacon ahumado y salsa brava - 1.25

☐ **m20** - Tortilla de patatas - 1.25 €

Postres

☐ **p37** - Pastel de Queso - 1.25 €

☐ **p38** - Pastel Chocolate - 1.25 €

☐ **p39** - Helado Chocolate - 1.25 €

☐ **p40** - Helado Vainilla - 1.25 €

☐ **p41** - Helado Limón - 1.25 €

☐ **p42** - Helado Fresa - 1.25 €

Crear nuevo pedido

Personal Details

Salutation  
--None-- ▼

First name:

Last name:

Gender : ☐ Male ☐ Female

Email:

Date of Birth:  

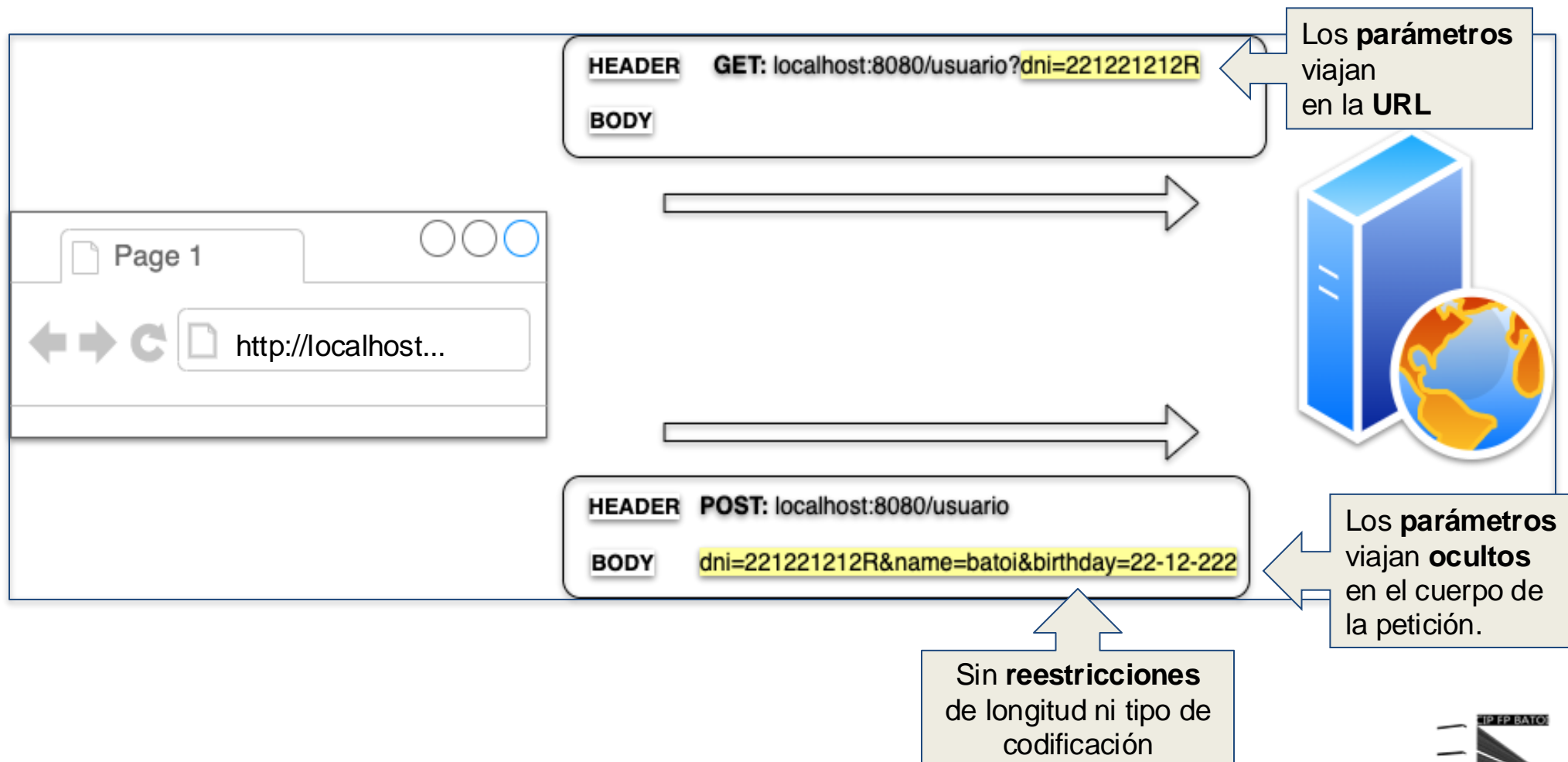
Address :

Submit



## 6. Paso de parámetros por POST

La **diferencia principal** con el método GET es **cómo viajan los parámetros** o información enviada en la petición:



## 6. Paso de parámetros por POST

- Imaginemos que necesitamos desarrollar una aplicación para gestionar las **tareas pendientes**. En la que necesitamos guardar: nombre de usuario, descripción de la tarea y fecha/hora de creación de la tarea

# Añadir Nueva Tarea

Codigo:

Nombre:

Descripción:

## 6.1 Paso 1. Definición del formulario

- Para poder enviar una **petición POST** al servidor debemos definir un nuevo **formulario** indicando que el **método utilizado es POST**.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Calculadora</title>
</head>
<body>
<h1>Añadir Tarea</h1>
```

Indicamos la  
**ruta del  
controlador**

Indicamos que  
el **método** a  
utilizar es  
**POST**

```
<form action="/tarea-add" method="post">
  <label>Código: <input name="code" type="number" required></label>
  <label>Usuario<input name="user" type="text" required></label>
  <label>Descripción<input name="description" type="text" required></label>
  <input type="submit" value="añadir" required></label>
</form>
</body>
</html>
```

```
@Controller
public class TareaController {

    @GetMapping("/tarea-form")
    public String tareaFormAction(){
        return "tarea_form_view";
    }
}
```

src

- main
  - java
    - es.cipfpbatoi
  - resources
    - static
    - templates
      - tarea\_form\_view.html
  - application.properties

## 6.1 Paso 2. Definición del modelo (entidades)

- Siguiendo el **patrón MVC**, definimos una nueva clase que se encargue de **representar una tarea** en nuestro sistema y la situamos en el paquete **modelo.entidades**

```
public class Tarea {  
  
    private int codigo;  
  
    private String nombre;  
  
    private String descripcion;  
  
    private LocalDateTime creadoEn;  
  
    public Tarea(int codigo, String nombre, String  
descripcion) {  
        this.codigo = codigo;  
        this.nombre = nombre;  
        this.descripcion = descripcion;  
        this.creadoEn = LocalDateTime.now();  
    }  
  
}
```



## 6.1 Paso 3. Definición del modelo (capa de acceso a datos)

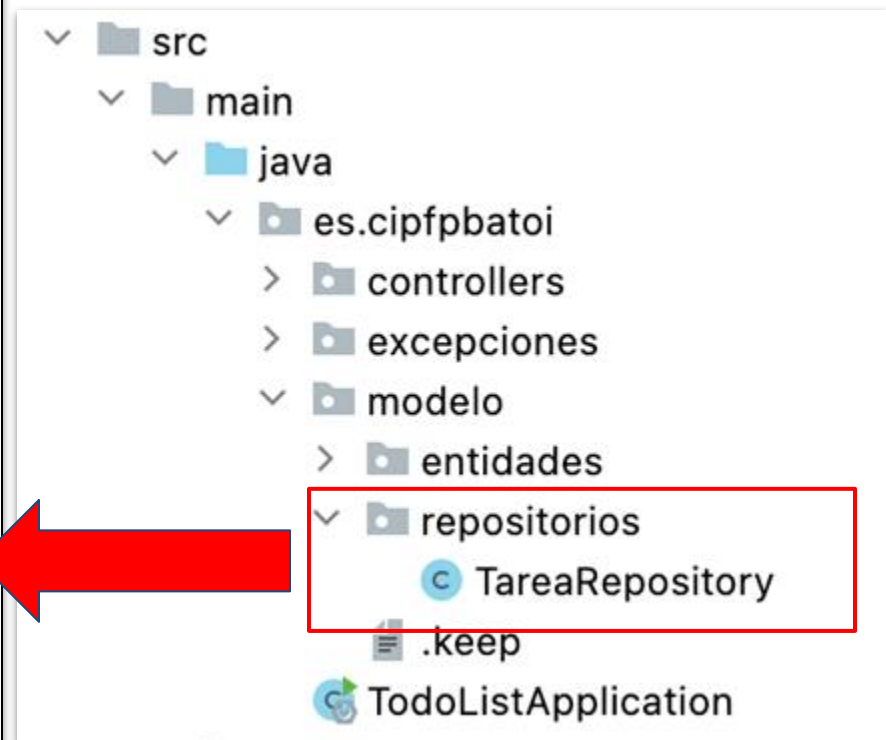
- Siguiendo el **patrón MVC**, definimos una **nueva clase** que **represente la capa de acceso a datos** de nuestro sistema y la situamos en el paquete **modelo.repositorios**

@Repository

```
public class TareaRepository {  
  
    private ArrayList<Tarea> tareas;  
  
    public TareaRepository() {  
        this.tareas = new ArrayList<>();  
    }  
  
    public void add(Tarea tarea) {  
        this.tareas.add(tarea);  
    }  
  
    public Tarea get(int codTarea) throws NotFoundException {  
        throw new NotFoundException("La tarea xxxx no existe");  
    }  
  
    public ArrayList<Tarea> findAll() {  
        return null;  
    }  
    ...  
}
```

@Repository

Equivalente a lo que, hasta ahora, hemos llamado **managers**



## 6.1 Paso 4. Definición del controlador

- Una vez tenemos el formulario, **definimos el controlador** que atenderá la petición del formulario utilizando la anotación **@PostMapping**.

```

@Controller
public class TareaController {

    @Autowired
    private TareaRepository tareaRepository;

    @PostMapping(value = "/tarea-add")
    @ResponseBody
    public String postAddAction(@RequestParam Map<String, String> params) {

        int code = Integer.parseInt(params.get("code"));
        String user = params.get("user");
        String descripcion = params.get("descripcion");

        Tarea tarea = new Tarea(code, user, descripcion);
        tareaRepository.add(tarea);

        String respuestaHtml = "<html>" +
            "<body>Tarea " + tarea.getCodigo() + " recibida con éxito</body>" +
            "</html>";

        return respuestaHtml;
    }
}

```

**@Controller** permite que el atributo (objeto de TareaRepository) se reciba ya creado e inicializado para ser utilizado (sino valdría null)

**@Autowired** permite que el atributo (objeto de TareaRepository) se reciba ya creado e inicializado para ser utilizado (sino valdría null)

**@PostMapping(value = "/tarea-add")** Indica que la petición debe realizarse haciendo uso del método **POST**

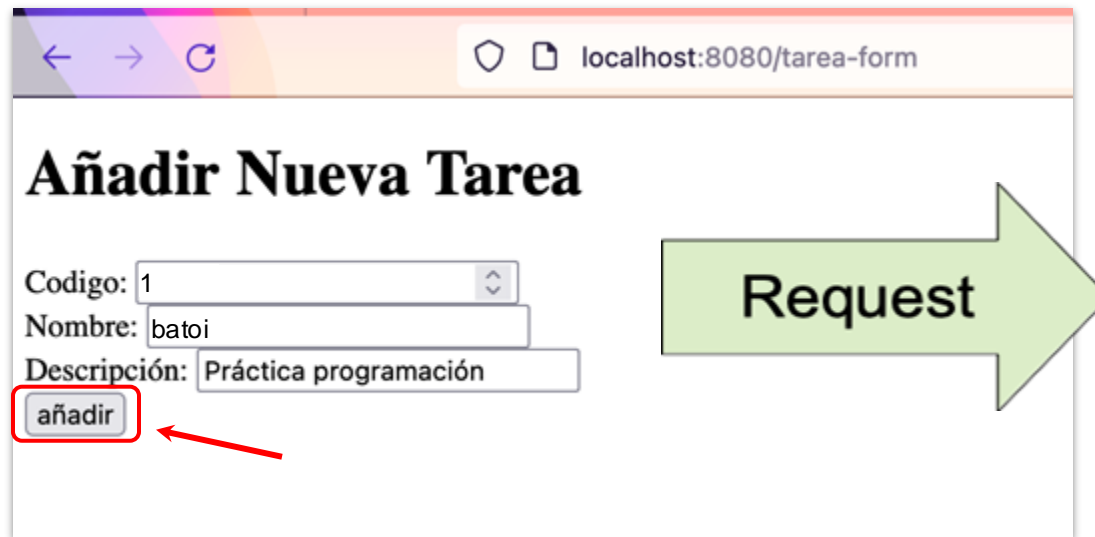
**@ResponseBody** Sólo se pueden pasar en este caso los parámetros de este modo (con Map)

**Recuperamos los datos de la tarea, instanciamos un nuevo objeto y lo guardamos**

**Devolvemos la respuesta al navegador**

## 6.1 Paso 5. Ejecutar la aplicación

Al cargar el formulario y pulsar sobre **el botón añadir**, el **navegador enviará la petición**, junto con los datos del formulario, al **controlador** haciendo uso del **método POST** y éste contestará al usuario **informando del resultado**.



Añadir Nueva Tarea

Codigo: 1

Nombre: batoi

Descripción: Práctica programación

**añadir**

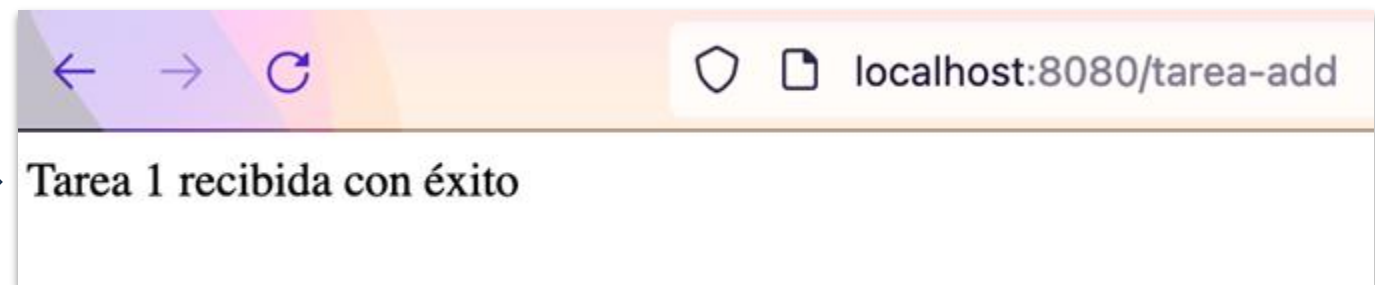
**Controlador** → /tarea-add

**code:** 1

**user:** batoi

**description:** Práctica programación

Respuesta →



Tarea 1 recibida con éxito

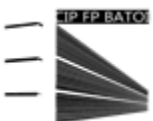
## 7. Anotaciones



Al principio de esta presentación hemos hecho mención a las **anotaciones** (@xxxx) que aparecen en el código visto hasta ahora.

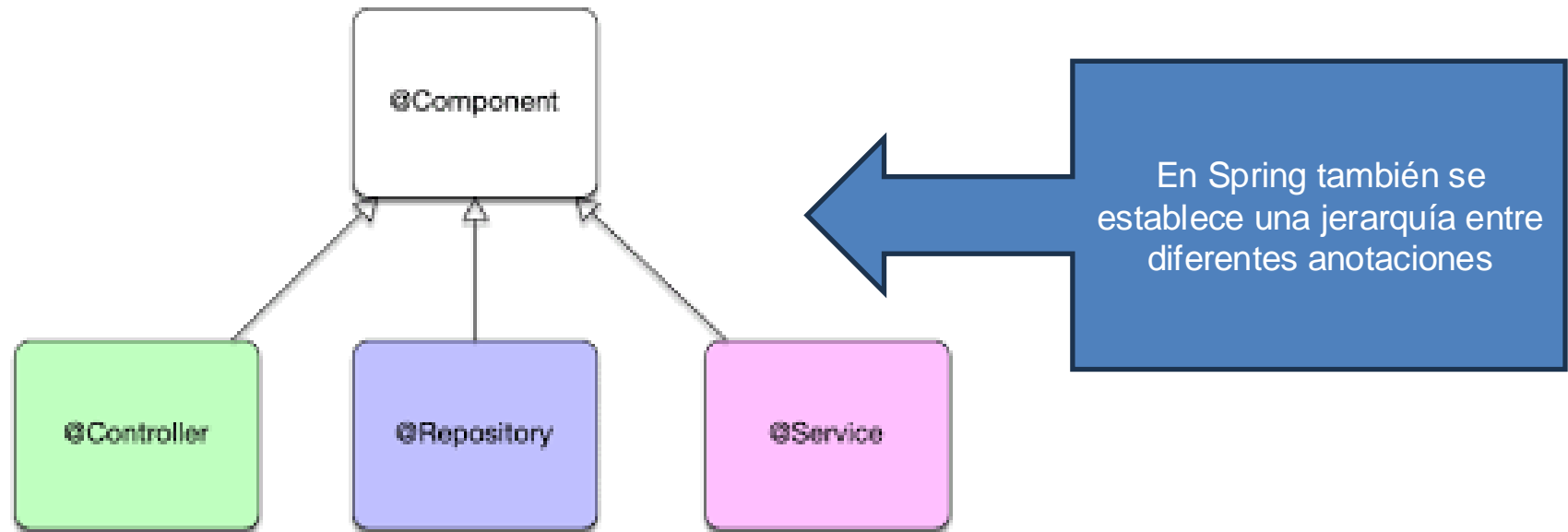
### ¿Qué son? ¿Para qué sirven?

- Una anotación en Java es una forma de **agregar** metadatos o **información adicional** a un elemento de programación, como una clase, un método o un campo.
- Las anotaciones proporcionan información **sobre cómo debe ser tratado el elemento al compilar, ejecutar o procesar el código**.
- Ejemplos en *Spring Boot*: @Controller, @Repository, @Service, @Component, @Autowired, @GetMapping, @PostMapping, @ResponseBody, etc.
- Muchas de ellas ya han sido explicadas durante la presentación.





## 7. Anotaciones



- **@Repository**: marca una clase como un **componente de acceso a datos** en la capa de persistencia de tu aplicación.
- **@Controller**: marca una clase como un **componente controlador** en la capa de presentación de tu aplicación.
  - Estos controladores manejan las solicitudes HTTP, interactúan con el usuario y coordinan la lógica de negocio.

## 7. Anotaciones

---

- **@Service**: marca una clase como un **componente de servicio** en la capa de negocio de tu aplicación.
  - Los servicios suelen contener la **lógica de negocio** y se utilizan para realizar operaciones específicas, como la manipulación de datos, el cálculo de resultados o la coordinación de varias tareas (*lo veremos más adelante*).
- **@Component**: marca una clase como un **componente** e indica a Spring que una clase debe ser instanciada y administrada por Spring.
  - Spring se encargará de crear instancias de la clase, gestionar su ciclo de vida y proporcionar **inyección de dependencias** si es necesario. **No la usaremos de forma directa.**

## 7. Anotaciones. Inyección de dependencias

- **@Autowired**: se utiliza para la [inyección de dependencias](#). Permite usar un componente directamente sin necesidad de instanciarlo.

```
@PostMapping(value = "/tarea-add")
@ResponseBody
public String postAddAction(@RequestParam Map<String,
String> params) {
    int code = Integer.parseInt(params.get("code"));
    String user = params.get("user");
    String descripcion = params.get("description");
    Tarea tarea = new Tarea(code, user, descripcion);
    tareaRepository.add(tarea);
    ...
    return respuestaHtml;
}
```

Recordando  
el ejemplo del punto 6.

Para poder hacer uso de **tareaRepository** tan solo habría que crear un **atributo** en el Controlador con ese nombre y anteponerle **@Autowired**. Spring se encarga del resto.

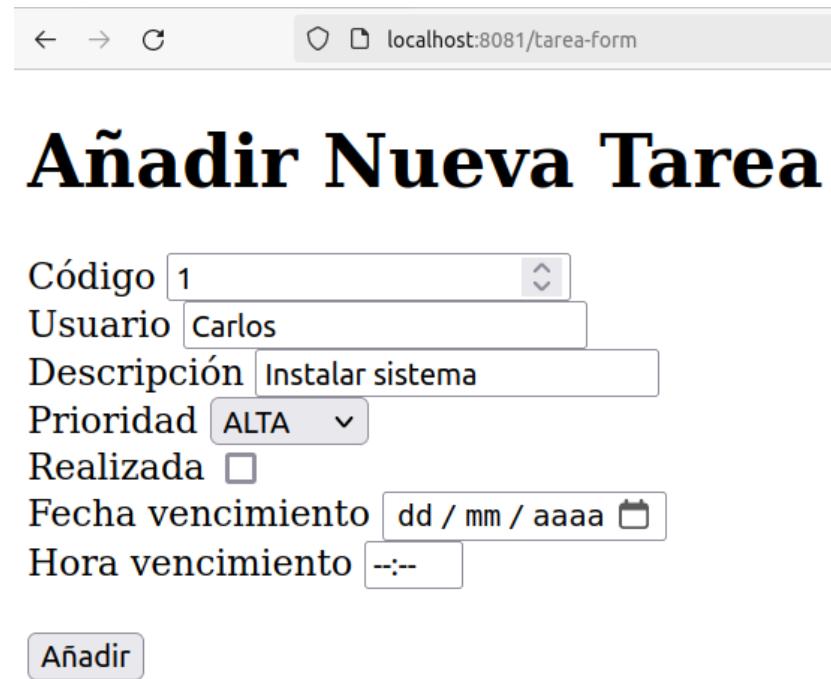
```
@Controller
public class TareaController {

    @Autowired
    private TareaRepository tareaRepository;
}
```

# ACTIVIDADES PREVIAS

**Actividad 4.-** Crea un fork del [siguiente proyecto/plantilla](#) y, a partir de él, implementa las siguientes funcionalidades:

- **Añadir el código necesario** para disponer de la fecha y hora de vencimiento, prioridad (Alta / Media / Baja) y si la tarea ha sido realizada o no. Modifica el caso de uso para que nos permita **añadir una nueva tarea con los nuevos atributos**.



The screenshot shows a web browser window with the address bar displaying 'localhost:8081/tarea-form'. The main heading of the page is 'Añadir Nueva Tarea'. Below the heading, there is a form with the following fields:

- Código**: A text input field containing the value '1'.
- Usuario**: A text input field containing the value 'Carlos'.
- Descripción**: A text input field containing the value 'Instalar sistema'.
- Prioridad**: A dropdown menu with 'ALTA' selected.
- Realizada**: A checkbox that is currently unchecked.
- Fecha vencimiento**: A date input field with the placeholder 'dd / mm / aaaa' and a calendar icon.
- Hora vencimiento**: A time input field with the placeholder '--:--'.

At the bottom of the form, there is a button labeled 'Añadir'.

# ACTIVIDADES PREVIAS

---

- **Visualizar la tarea a partir del código.** El recurso recibirá como parámetro (a través de una petición get) el código de la tarea que se quiere visualizar, la buscará haciendo uso del repositorio y mostrará toda la información de la tarea.
- **Borrar una tarea a partir del código:** El recurso, recibirá como parámetro (a través de una petición get) el código de la tarea que se quiere borrar, la buscará haciendo uso del repositorio y mostrará toda la información de la tarea.
  - Una vez borrada, debe mostrar el mensaje “Tarea xxx borrada con éxito” Si no se encuentra una tarea con dicho código deberá mostrar el mensaje “La tarea con código xxxx no existe”

# ACTIVIDADES PREVIAS

**Actividad 5.-** Debate en gran grupo.

**¿Y si necesitamos visualizar todas las tareas?**

Obtener del repositorio todas las tareas existentes y mostrarlas en formato tabla (código, descripción, días vencimiento, usuario y si ha sido realizada).

**¿Creamos toda la página html con todos los datos y enlaces en el controlador?**

## Listado de tareas

Código	Descripción	Días Vencimiento	Responsable	Realizado	Acciones
1	Ir al Cine	2	Pepe	Sí	<a href="#">Ver detalle</a>
2	Practicas programación 2		Elena	Sí	<a href="#">Ver detalle</a>

Elemento de tipo  
<a href="...">  
con referencia al  
controlador definido en  
el punto anterior  
(visualizar tarea).

## 8. WEBGRAFIA

---

- . **Introducción** al desarrollo de **aplicaciones web**.  
<https://github.com/igomis/apunts/blob/master/docs/1.Introduccio.md>.  
Ignaci Gomis
- . **Documentación** Oficial Framework.  
<https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>.  
Spring Team

- Eso es todo... de momento :-)