

UNIDAD 8: APPLICACIONES WEB CON SPRING, MVC Y THYMELEAF

Índice

UNIDAD 8: APPLICACIONES WEB CON SPRING, MVC Y THYMELEAF.....	1
1. ¿Qué es Thymeleaf?	2
2. Creación de un “New Spring Starter Project Spring Boot”	6
a) Configuración	6
b) Creando las clases necesarias.....	11
Creación del controlador.....	11
Creación de la vista	14
3. Primeros pasos con Thymeleaf	21
4. Configuración de Thymeleaf	23

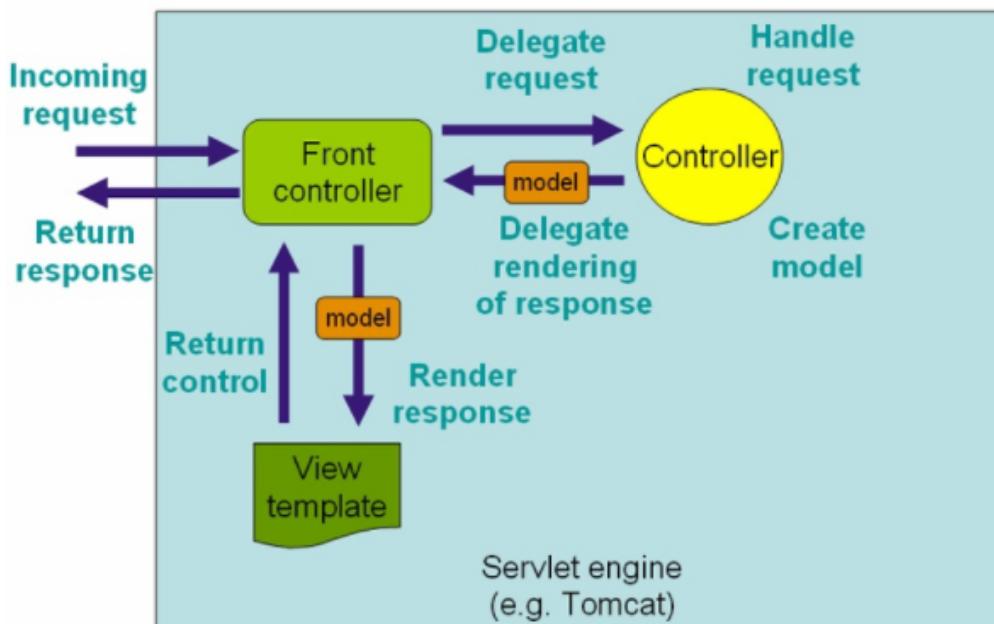


1. ¿Qué es Thymeleaf?

Veamos cómo se procesa una petición en Spring MVC:

Procesamiento de una petición en Spring MVC

A continuación se describe el flujo de procesamiento típico para una petición HTTP en Spring MVC. Esta explicación está simplificada y no tiene en cuenta ciertos elementos que comentaremos posteriormente. Spring es una implementación del patrón de diseño "front controller", que también implementan otros frameworks MVC, como por ejemplo, el clásico Struts.



- Todas las peticiones HTTP se canalizan a través del *front controller*. En casi todos los frameworks MVC que siguen este patrón, el *front controller* no es más que un servlet cuya implementación es propia del framework. En el caso de Spring, la clase `DispatcherServlet`.
- El *front controller* averigua, normalmente a partir de la URL, a qué *Controller* hay que llamar para servir la petición. Para esto se usa un `HandlerMapping`.
- Se llama al *Controller*, que ejecuta la lógica de negocio, obtiene los resultados y los devuelve al servlet, encapsulados en un objeto del tipo `Model`. Además se devolverá el nombre lógico de la vista a mostrar (normalmente devolviendo un `String`, como en JSF).
- Un `ViewResolver` se encarga de averiguar el nombre físico de la vista que se corresponde con el nombre lógico del paso anterior.
- Finalmente, el *front controller* (`DispatcherServlet`) redirige la petición hacia la vista, que muestra los resultados de la operación realizada.

En realidad, el procesamiento es más complejo. Nos hemos saltado algunos pasos en aras de una mayor claridad. Por ejemplo, en Spring se pueden usar interceptores, que son como los filtros del API de servlets, pero adaptados a Spring MVC. Estos interceptores pueden pre y postprocesar la petición alrededor de la ejecución del *Controller*. No obstante, todas estas cuestiones deben quedar por fuerza fuera de una breve introducción a Spring MVC como la de estas páginas.

Thymeleaf es un motor de plantillas para aplicaciones web.

* ¿Qué es un motor de plantillas?

La idea general consiste en poder tener un HTML plano, no muy complejo y dentro, tener funciones que nos carguen los datos enviados desde el servidor (backend), por tanto, nos permite no estar retocando constantemente el HTML. Además, diseñador y programador pueden estar trabajando simultáneamente en el mismo proyecto.

Cada vez más en el front-end, consumimos lo que se llaman "rest-api", que te ofrecen datos que vienen del back-end en formato de texto, por ejemplo, en formato JSON. Lo que ofrecen estas librerías de plantillas es pasar rápidamente de esos datos a pedazos de código HTML que puedes insertar cómodamente en la página.

Thymeleaf soporta o puede generar plantillas de estos seis tipos:

- HTML
- XML
- TEXT
- JAVASCRIPT
- CSS
- RAW

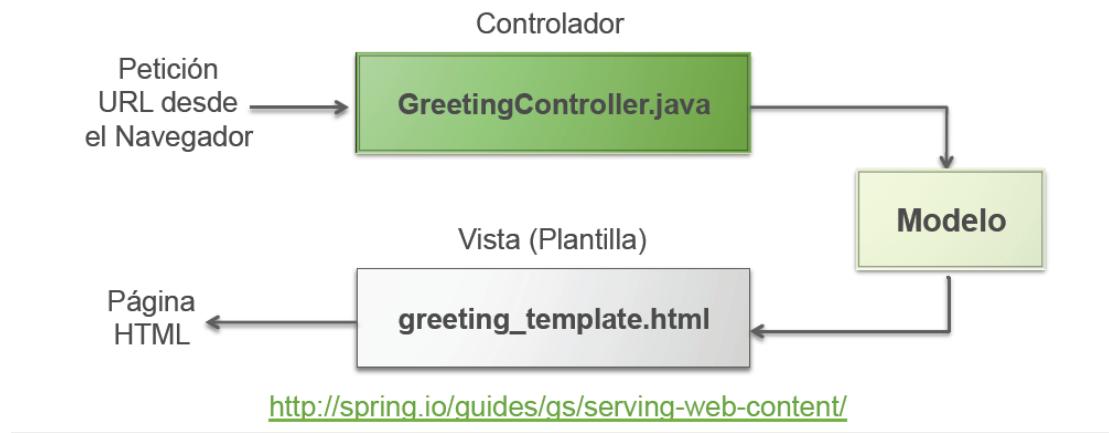
Thymeleaf permite realizar la maquetación HTML sin necesidad de que intervenga el servidor.

APLICACIONES WEB CON SPRING MVC Y THYMELEAF

1. Spring MVC

Introducción

- El modelo vista controlador (MVC) separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario.
- El esquema del patrón MVC en Spring es:



APLICACIONES WEB CON SPRING MVC Y THYMELEAF

1. Spring MVC

Controladores

- Los controladores (*Controllers*) son clases Java encargadas de atender las peticiones web.
- Procesan los datos que llegan en la petición (parámetros).
- Hacen peticiones a la base de datos, usan diversos servicios, etc...
- Definen la información que será visualizada en la página web (el modelo).
- Determinan que vista será la encargada de generar la página HTML.

APLICACIONES WEB CON SPRING MVC Y THYMELEAF

1. Spring MVC

Vistas

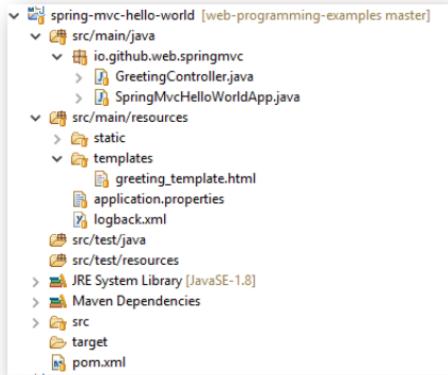
- Las vistas en Spring MVC se implementan como plantillas HTML.
- Generan HTML partiendo de una plantilla y la información que viene del controlador (modelo).
- Existen diversas tecnologías de plantillas que se pueden usar con Spring MVC: JSP, Thymeleaf, FreeMarker, etc...
- Nosotros usaremos **Thymeleaf**.

APLICACIONES WEB CON SPRING MVC Y THYMELEAF

1. Spring MVC

Ejemplo

- Estructura de la aplicación vista desde Eclipse:



Fork me on GitHub

Creación de un proyecto Spring con Thymeleaf. Mi segundo "Hola mundo".

En esta imagen vemos el aspecto de un archivo html con código y cómo queda cuando es renderizado, es decir, cuando se pasa a su representación gráfica final.

```
<table id="myTableId" dt:table="true">
<thead>
<tr>
<th>Id</th>
<th>Firstname</th>
<th>Lastname</th>
<th>Street</th>
<th>Mail</th>
</tr>
</thead>
<tbody>
<tr th:each="person : ${persons}">
<td th:text="${person.id}">1</td>
<td th:text="${person.firstName}">John</td>
<td th:text="${person.lastName}">Doe</td>
<td th:text="${person.address.street1}">Nobody knows 1</td>
<td th:text="${person.mail}">john@doe.com</td>
</tr>
</tbody>
</table>
```

Renderizado

As a result, you'll have the following full-featured HTML table!				
	FirstName	LastName	Street	Mail
1	Selma	Martinez	Ap #905-1812 Eu. Ave	selmane@Chaseandpeal.com
2	Vince	Sato	947-3605 Feugiat St.	libero.libertum.metus@arte.ca
3	Holte	Rosales	1389 Aliquam, Bl.	Duis curus.diam@dominiquefotestaque.ca
4	Nehru	Gonzalez	P.O. Box 374, 9474 Laconia Street	tellus.nam.magna@node.edu
5	Calista	Hill	8968 Eu Road	enim.Elsam.imperdiet@starmetius.ca.uk
6	Chast	Wilder	317-0653 Ligula St.	iacus.vestibulum@curcuma.edu
7	Oliver	Rivas	6489 Sed Ave	velit@dominalevada.co.uk
8	Lita	Monroe	8438 Vienns Ave	Mauris.ac.turpis@ri.ca
9	Yeti	Vargas	P.O. Box 722, 6079 Eu St.	odio@Donec.net
10	Lee	Graes	211-4991 Dictum Road	velit@utiae.co.uk

2. Creación de un “New Spring Starter Project Spring Boot”

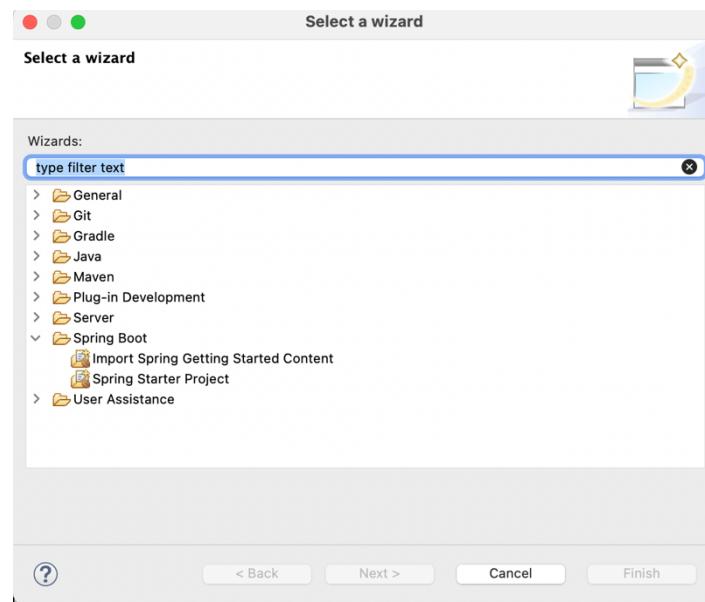
Nota: Solo hay que tener descargado antes el IDE Spring Tool Suit.

Veámoslo paso a paso.

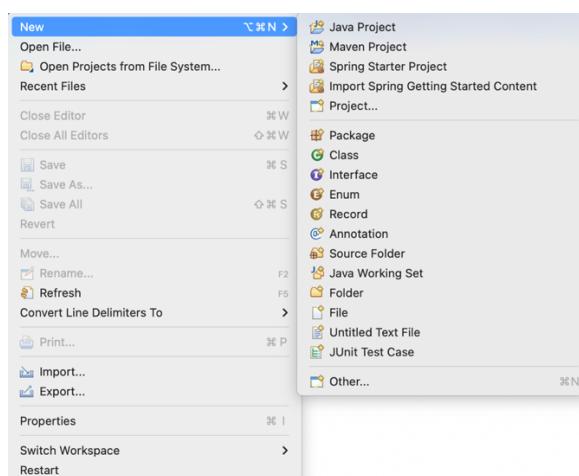
a) Configuración

Paso 1: New/Spring Starter Project

La primera vez no aparecerá en los recientes, por lo que estará en:



Posteriormente, ya aparecerá en recientes:



En la ventana que aparece hay que modificar, elegir y nombrar las siguientes cosas (si no se dice nada sobre ellas aquí se dejan tal como salen por defecto):

- Name: debemos cumplir lo de cualquier proyecto java, mayúsculas, minúsculas...

Ejemplo: UD8E01HolaMundo

- Type: Elegimos Maven
- Java Version: 17
- Group: Todo minúscula. Nombre del dominio del colegio (imaginario, no es real) pero siempre pondremos: com.salesianostriana.dam

Es la base o raíz de las rutas que usaremos en el proyecto para las distintas carpetas y archivos.

- Artifac: Mismo nombre del proyecto, pero todo en minúscula (aunque no tendría que ser en minúscula obligatoriamente, nosotros lo haremos así).

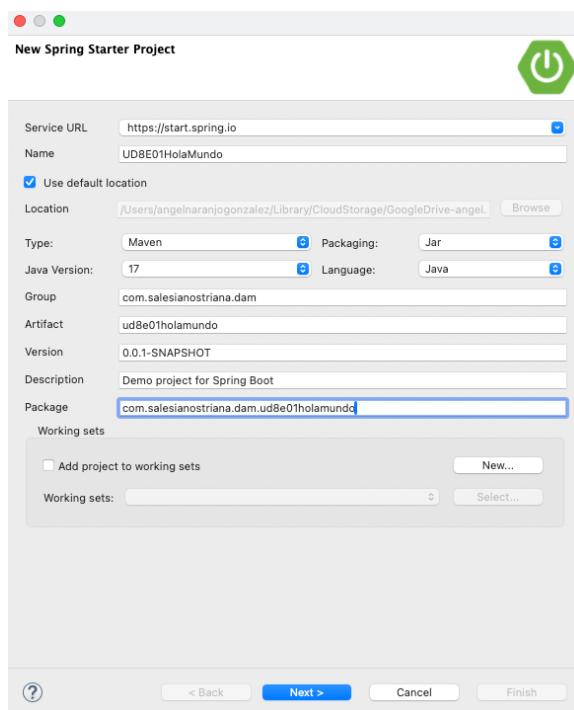
Básicamente un artifact es un bloque de código reutilizable. Si alguien quiere saber más dejó un enlace en la plataforma.

Ejemplo: ud8e01holamundo

- Package: Composición del nombre del group y del artifac, todo en minúscula. Por ejemplo:

com.salesianostriana.dam.ud8e01holamundo

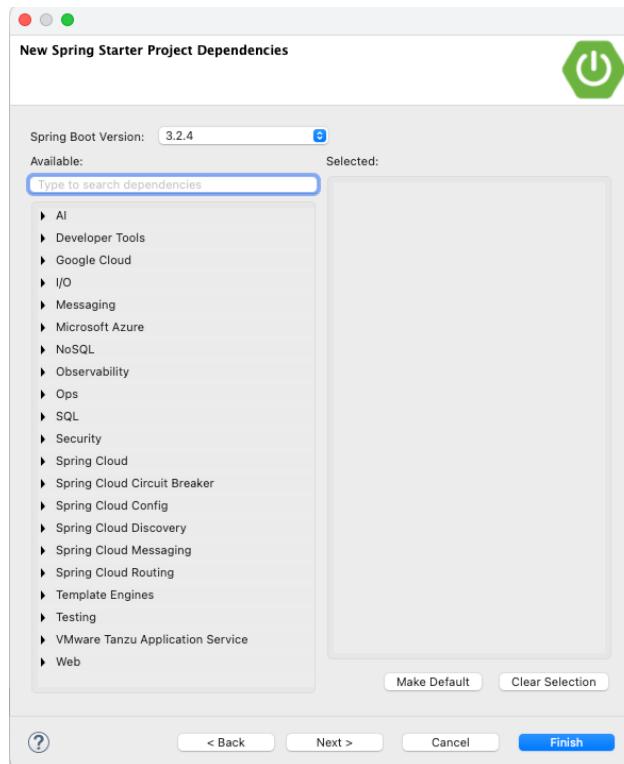
Quedaría algo así:



- **Pinchamos en Next**

Aparece la siguiente página, dedicada a las dependencias. La primera vez no, pero después aparecerán las más recientes ya buscadas. También se podrán agregar después, pero mejor hacerlo todo antes de empezar.

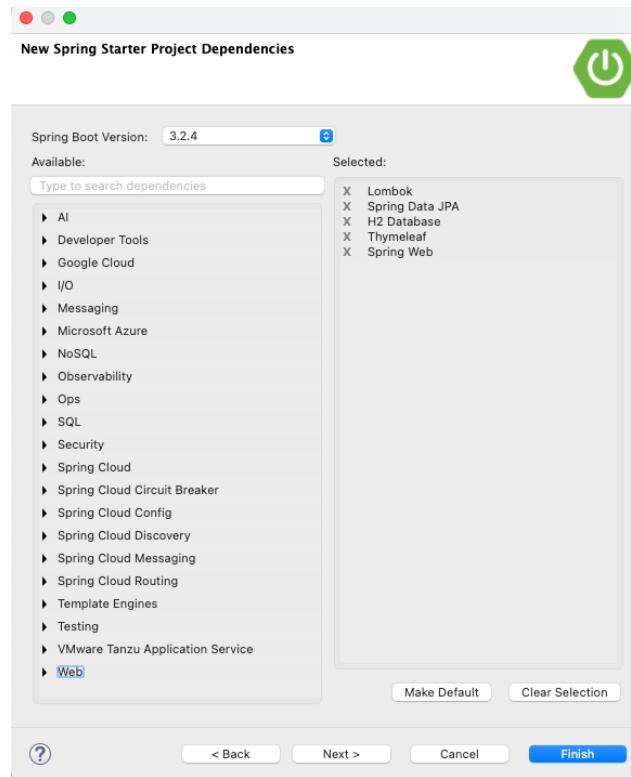
En primer lugar, elegimos la versión de Spring Boot Version. En nuestro caso, la 3.2.4



Debemos marcar los tipos de dependencias que queremos incluir. Al menos de momento debemos marcar:

- En Web (Spring Web)
- En Template Engine (Thymeleaf)
- En SQL (H2 Database y Spring Data JPA)
- En Developer Tools (Lombok)
- Y todas aquellas que vayan haciéndose necesarias conforme nuestro proyecto vaya creciendo.

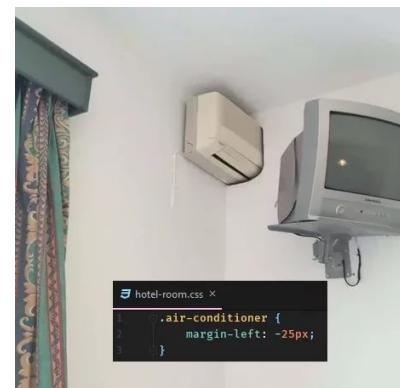
Debería quedar algo así:



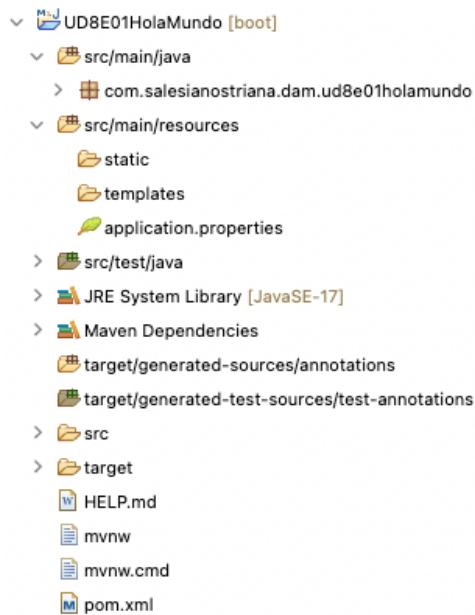
- Pinchamos en Finish

Paso 2: Nuestro proyecto comienza a crearse descargando e importando lo necesario.

Dependiendo de la conexión a internet y el ordenador, puede tardar unos minutos (no te asustes si aparece mensaje de error al principio) aparecerá nuestro proyecto. Puedes ver la barra de proceso abajo a la derecha.



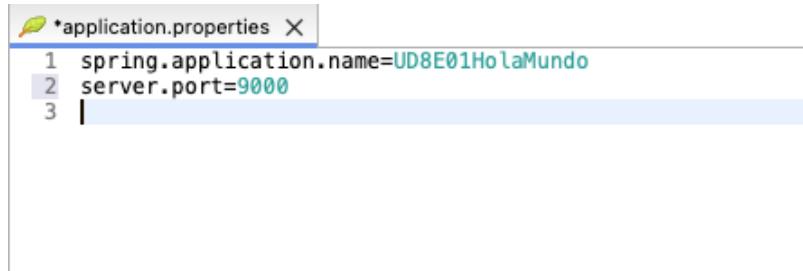
- Estructura de carpetas creada



- Cambio de puerto

Pinchamos en el archivo application.properties que está dentro de la carpeta src/main/resources y escribimos: server.port=9000 (puede ser también el 8090 o cualquier otro pero debemos recordarlo para luego escribir dicho puerto en la ruta del navegador para ver la página) como se puede ver en la imagen y guardamos. Esto se hace por si el 8080 que es el puerto por defecto está ocupado.

Poned siempre 9000 para que todos usemos lo mismo.



```
*application.properties X
1 spring.application.name=UD8E01HolaMundo
2 server.port=9000
3 |
```

b) Creando las clases necesarias

Cuando se vayan escribiendo clases, STS nos pedirá importar los paquetes o clases necesarias para poder usar las anotaciones propias de Spring.

NOTA: Un atajo para importar todas las librerías necesarias directamente es CTRL+SHIFT+"LETRA O" en Windows (COMMAND+SHIFT+LETRA O en Mac)

Por otro lado, debemos observar que ya se ha creado la clase principal, llamada **Ud8E01HolaMundoApplication**, donde se encuentra el main y desde donde arrancará nuestra aplicación. Se verá la posibilidad de escribir en el main algún código más para poder ejecutar ciertas cosas de momento para probar nuestros ejemplos.

Creación del controlador

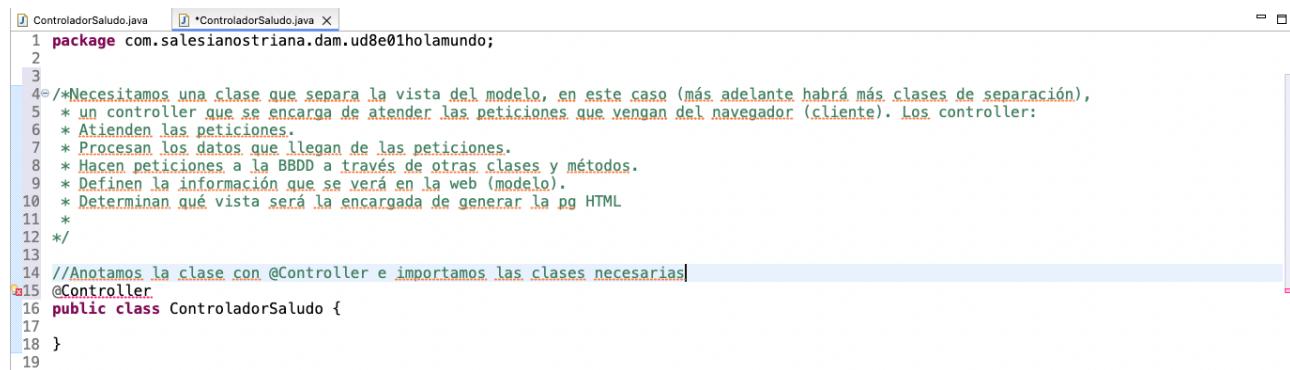
Debemos crear una clase dentro del paquete que ya tenemos creado en src/main/java/com.salesianostriana.dam.ud8e01holamundo.

De momento a esta clase, la llamaremos ControladorSaludo.java

Un controlador es un método que atiende a peticiones, por lo que debe ir anotado (palabras con @ encima de su firma) con @Controller. Al poner esta, STS nos da un error porque la clase no está importada.

Ya hemos dicho que podemos ir importando paquetes mediante control+shift+letra O. También como siempre pinchando en el error y eligiendo las clases que nos propone el IDE o escribiendo nosotros mismos los import.

El controlador en esta ocasión es una clase muy simple. Spring, facilita mucho la creación de controladores, que atenderán las peticiones de nuestra aplicación antes de enviar los datos a la vista y, en principio, que no debe heredar de ninguna clase especial.



```
ControladorSaludo.java *ControladorSaludo.java X
1 package com.salesianostriana.dam.ud8e01holamundo;
2
3
4 /*Necesitamos una clase que separa la vista del modelo, en este caso (más adelante habrá más clases de separación),
5 * un controller que se encarga de atender las peticiones que vengan del navegador (cliente). Los controller:
6 * Atienden las peticiones.
7 * Procesan los datos que llegan de las peticiones.
8 * Hacen peticiones a la BBDD a través de otras clases y métodos.
9 * Definen la información que se verá en la web (modelo).
10 * Determinan qué vista será la encargada de generar la pg HTML
11 *
12 */
13
14 //Anotamos la clase con @Controller e importamos las clases necesarias|
15 @Controller
16 public class ControladorSaludo {
17
18 }
19
```

La clase completa sería:

```

package com.salesianostriana.dam.ud8e01holamundo;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;

/* Necesitamos una clase que separa la vista del modelo, en este caso (más adelante
habrá más clases de separación),
 * un controller que se encarga de atender las peticiones que vengan del navegador
(cliente). Los controller:
 * Atienden las peticiones.
 * Procesan los datos que llegan de las peticiones.
 * Hacen peticiones a la BBDD a través de otras clases y métodos.
 * Definen la información que se verá en la web (modelo).
 * Determinan qué vista será la encargada de generar la pg HTML
 *
*/
//Anotamos la clase con @Controller e importamos las clases necesarias
@Controller
public class ControladorSaludo {

    /* Ejemplo 1: primer controlador, llamado welcome
     *
     * "/ o /welcome" es el nombre del recurso que queremos mostrar al inicio, por
     * tanto la palabra que se debe escribir en la ruta que tendremos que escribir
     * en el navegador después de localhost:9000 (o nada o welcome y posteriormente
     * se puede probar con un parámetro nombre)
     *
     * @GetMapping indica que el método justo de debajo será el que atenderá la
     * petición tipo Get (la petición es tipo Get solo hay un método y dos cuando es
     * tipo Post (formulario). Puede que en códigos antiguos os encontréis el uso
     * de RequestMapping en lugar de GetMapping.
     *
     * El @RequestParam se utiliza para pasar un parámetro a la petición tipo get, es
     * decir, información o datos que
     * vamos a pasar en una petición tipo get (recuerda que no están pensadas para esto
     * pero hay muchas veces
     * en las que llevan algún parámetro.
     * Si en la barra del navegador pasamos este parámetro con un valor aparecerá en el
     * saludo, esto es,
     * si escribes en el navegador http://localhost:9000/?nombre=Luis aparecerá el saludo
     * personalizado para Luis,
     * si no hay parámetro, el valor por defecto es Mundo
     */

    @GetMapping({ "/", "welcome" })
    public String welcome(@RequestParam(name = "nombre", required = false,
defaultValue = "Mundo") String nombre,
                           Model model) {

        /*
         * El parámetro "nombre" es la palabra que usamos para darle un identificador a
         * la variable y debemos usar en la plantilla dentro de <p>Hello <span
         * th:text="${nombre}">Friend</span>!</p> Ojo porque no tener un criterio a la
         * hora de dar estos nombres a las variables hace que tengamos muchos errores
         * (usar mayúsculas o minúsculas, que el nombre no indique qué guarda, mezclas
         * varias palabras...
         */
    }
}

```

```

model.addAttribute("nombre", nombre); // Incluimos la información en el
modelo

return "index"; // Nombre de la plantilla que generará la página HTML (sin
extensión),
// en nuestro caso estos html deben estar dentro de la carpeta
// src/main/resources/templates
// y deben llamarse igual que el String que devuelve el método,
// index.html Ojo con las mayúsculas y minúsculas del nombre
// de la plantilla porque son tenidas en cuenta
}

/*
 * Ejemplo 2: Segundo controlador llamado welcome2
 * Atiende a la petición /saludo2 escrita en el @GetMapping
 *
 * Se puede ver cómo se "pasa o carga información" al
 * model dentro del método mediante el método addAttribute
 * como un objeto de la clase Persona, con valores de sus dos
 * atributos. Más adelante iremos viendo de dónde sacamos esa "información" que
 * cambiar, como aquí la new Persona creada directamente en el método
addAttribute.

*/
@GetMapping("/saludo2")
public String welcome2(Model model) {

    model.addAttribute("persona", new Persona("Ángel", "Naranjo González"));
    return "SaludoPersonalizado";
}

/*
 * @GetMapping es una variante de requestMapping, más "nueva" que se utiliza
 * como atajo, ya que basta con el nombre del recurso, mientras que
 * con @RequestMapping, en general, tenemos que ir diciendo el tipo de petición
 * que se está atendiendo, es decir, necesita que le indiquemos el value
 * (nombre el recurso, por ejemplo, /saludo3) y el método que se usa para la
 * petición, en nuestro caso, tendríamos que escribir:
 * @RequestMapping (value="/saludo3", method=RequestMethod.GET)
 * (también existe RequestMethod.POST)
 * En general, se usa por simplificar el código
 * Se puede ver un ejemplo en:
 * https://www.arquitecturajava.com/spring-getmapping-postmapping-etc/
 *
 * Nosotros usaremos siempre GetMapping
 */
/* Ejemplo 3: Tercer controlador llamado welcome3
 * Atiende a la petición get "/saludo3"
 *
 */
/* */

@GetMapping ("/saludo3")
public String welcome3(Model model) {

    model.addAttribute("saludo", "Hola Mundo");
    model.addAttribute("mensaje", "¡Se me está haciendo largo el proyecto
final!");
    model.addAttribute("url", "https://triana.salesianos.edu");

    return "SaludoYEnlace";
}

```

En primer lugar, llama la atención la simplicidad de la clase, es Java en estado puro, anotada como hemos dicho con `@Controller`

La clase tiene un varios métodos, cuya *firma* es `public String nombreMetodo(Model model)`:

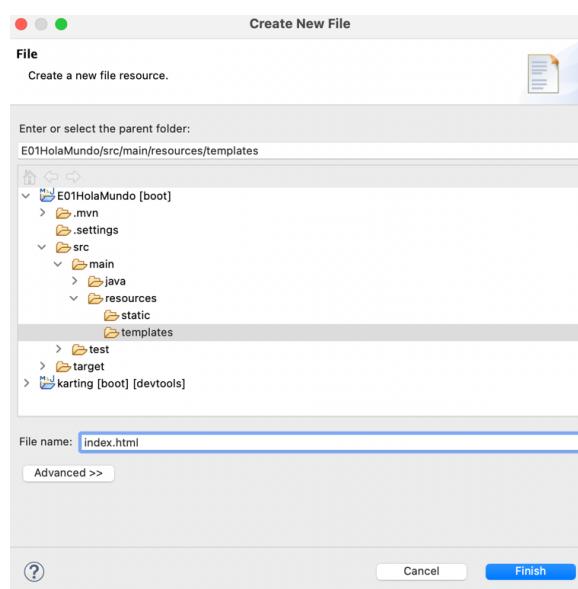
- Como decíamos más arriba, el tipo de retorno es String, ya que devolverá **el nombre** de una vista en formato cadena y sin la extensión html.
- Como argumento, recibe un elemento de tipo Model. Será el "contenedor" que nos permita añadir datos que serán "transportados" a la vista. Iremos viendo más formas de añadir cosas al model, de momento, solo con addAttribute.
- Además, están anotados con `@GetMapping("/saludo3")`. De esta forma indicamos que cualquier petición hacia la url `http://máquina:puerto>HelloWorldSpringMVC/saludo3` será procesada por este método.

Creación de la vista

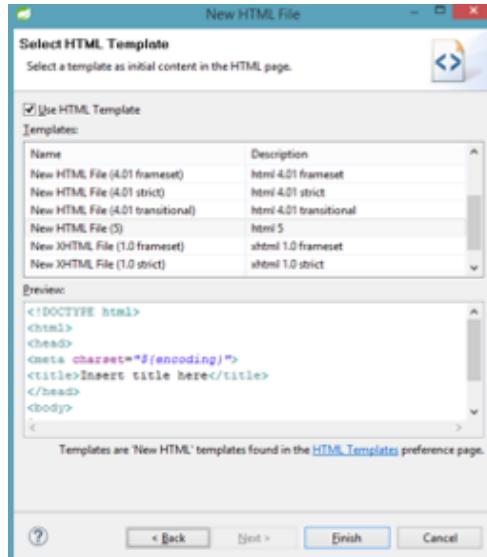
Para crear la vista, escribiremos nuestros archivos con código HTML en la carpeta "*templates*" dentro de la ruta de carpetas del proyecto `src/main/resources`.

Para ello basta crear un archivo tipo HTML File pinchando con el botón derecho en la carpeta templates, new y crear un archivo tipo File y añadir en el nombre la extensión .html. El nombre de este archivo debe ser el mismo que el String que devuelve el método controlador que atiende a la petición de welcome.

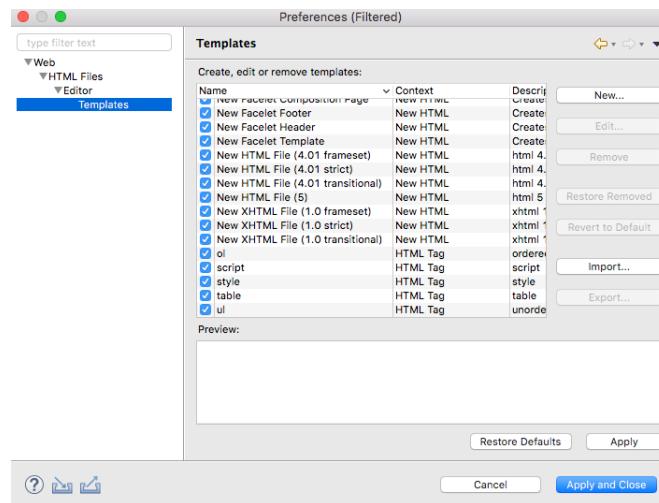
En nuestro caso, el primero que haremos será el index.html que es la página que se pinta al atender la petición del primer ejemplo.



NOTA: Si se instala el plugin adecuado, el propio STS ofrece la posibilidad de crear directamente un archivo tipo HTML (no File genérico) y con él podemos configurar algunos aspectos de nuestros archivos html. Ahora mismo no lo trae STS así que no lo haremos aquí, pero tiene más o menos este aspecto:



Se pueden realizar algunos cambios en la plantilla que por defecto ofrece STS. Para ello, pinchamos en el vínculo HTML Templates. Aparecerá la siguiente ventana:



Nosotros no instalaremos ese plugin.

Escribimos el siguiente código en el archivo html.

Nota: El siguiente código pertenece al archivo html llamado "index.html".

En la segunda línea, se define lo que se llama el "espacio de nombres" que avisa a html de que dentro se utilizarán etiquetas de Thymeleaf, mediante "th:"

```
<!doctype html>
<html lang="es" xmlns:th="http://www.thymeleaf.org">

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
shrink-to-fit=no">
    <title>Hello, world!</title>
</head>

<body>
    <h1>
        Hola <span th:text="${nombre}">mundo</span>
    </h1>
    <h2>Bienvenido al... Infierno</h2>
</body>
</html>
```

Si te fijas, la parte que irá cambiando según el parámetro nombre que le demos, lleva delante th: ya que es la que se refiere a Thymeleaf. Este parámetro debe llamarse igual que la variable usada en el método controller.

```
@GetMapping({ "/", "welcome" })
public String welcome(@RequestParam(name = "nombre", required = false,
defaultValue = "Mundo") String nombre, Model model) {

    model.addAttribute("nombre", nombre);
    return "index";
}
```

Además, encontramos que hemos usado *Expression Language*, una característica soportada por Thymeleaf que nos permite utilizar un lenguaje sencillo para usar un JavaBean dentro de una página HTML.

`${nombre}` buscará dentro del contexto (ámbito de vida de la petición) una variable llamada *nombre* (ahora, el nombre de la variable es lo que va entre comillas dobles dentro de los parámetros del método *addAttribute*, y mostrará su valor (que es el parámetro que va en

segundo lugar, en marrón).

Si agregas un nombre como parámetro en la petición que se escribe en la web, aparecerá Hola y el nombre pasado.

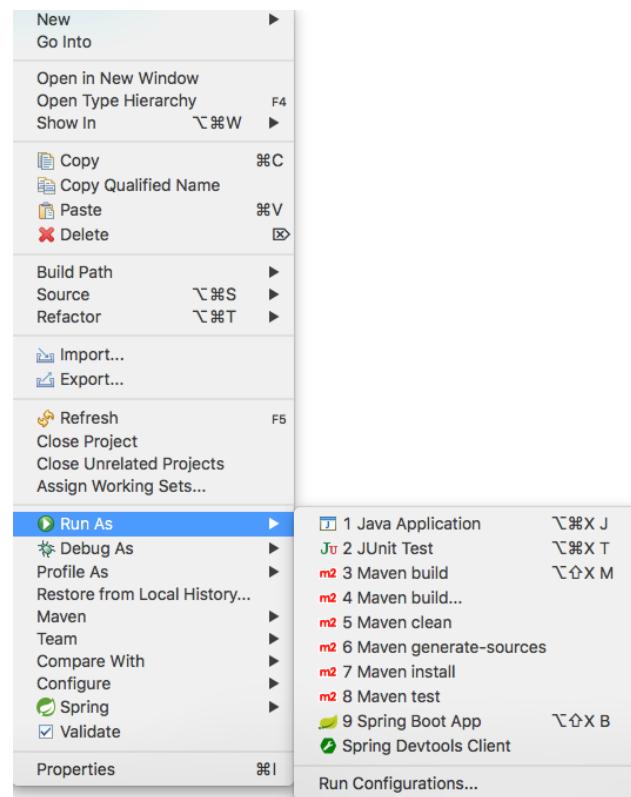
Por ejemplo: <http://localhost:9000/?nombre=Luis>

Para empezar a entender la parte de Thymeleaf, vamos a utilizar sus propios tutoriales, pero puedes ver que está relacionado con las etiquetas th: que aparecen en el html de este ejemplo.

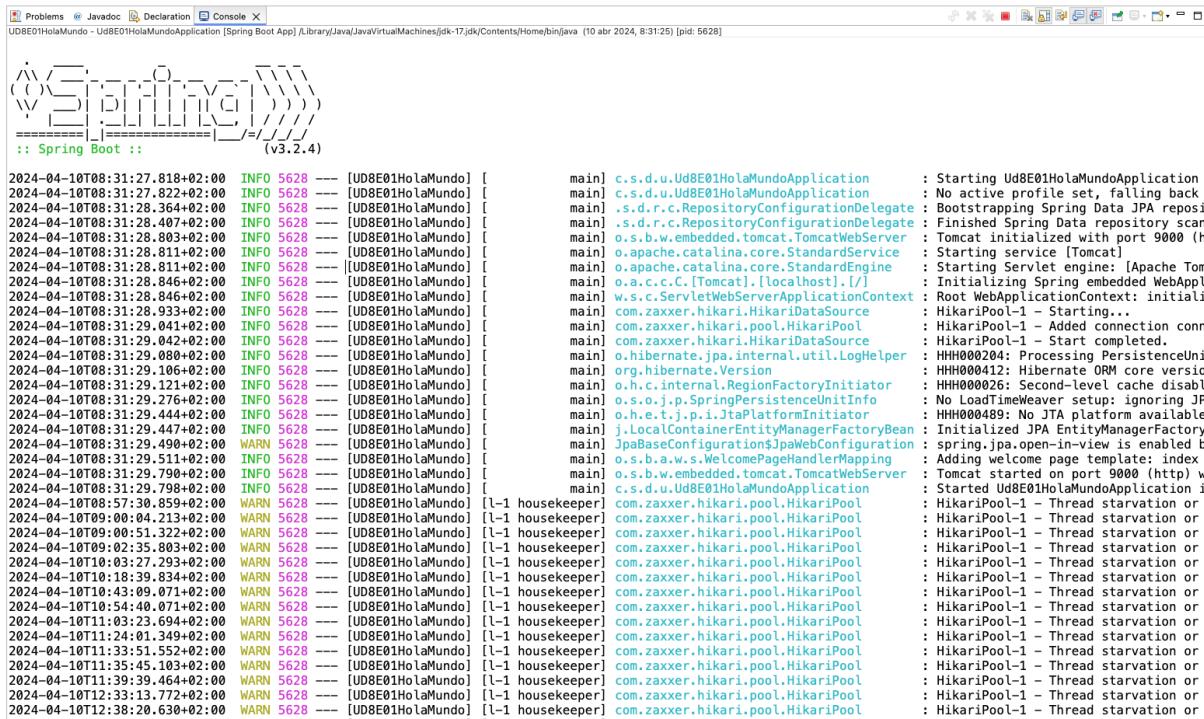
Paso 4: Ejecución

Una vez que tenemos definido nuestro proyecto, lo hemos configurado, y hemos añadido una vista y un controlador, vamos a pasar a ejecutarlo.

Para ello, tan solo tenemos que pulsar sobre el proyecto con el botón derecho > Run As >Spring Boot App.



Si todo va bien, debe aparecer en la consola algo como la siguiente imagen:



```

Problems Javadoc Declaration Console X
UdBE01HolaMundo - UdBE01HolaMundoApplication [Spring Boot App] /Library/Java/VirtualMachines/jdk-17/jdk/Contents/Home/bin/java [10 abr 2024, 8:31:25] [pid: 5628]

:: Spring Boot ::

2024-04-10T08:31:27.818+02:00 INFO 5628 --- [UdBE01HolaMundo] [main] c.s.d.u.UdBE01HolaMundoApplication : Starting UdBE01HolaMundoApplication
2024-04-10T08:31:27.822+02:00 INFO 5628 --- [UdBE01HolaMundo] [main] c.s.d.u.UdBE01HolaMundoApplication : No active profile set, falling back to default profiles: []
2024-04-10T08:31:28.364+02:00 INFO 5628 --- [UdBE01HolaMundo] [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repository scan
2024-04-10T08:31:28.407+02:00 INFO 5628 --- [UdBE01HolaMundo] [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scan
2024-04-10T08:31:28.803+02:00 INFO 5628 --- [UdBE01HolaMundo] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 9000 (http://localhost:9000)
2024-04-10T08:31:28.811+02:00 INFO 5628 --- [UdBE01HolaMundo] [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-04-10T08:31:28.811+02:00 INFO 5628 --- [UdBE01HolaMundo] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.14]
2024-04-10T08:31:28.846+02:00 INFO 5628 --- [UdBE01HolaMundo] [main] w.s.c.ServletWebServerApplicationContext : Initializing Spring embedded WebApp: Root WebApplicationContext: initializers=[org.springframework.boot.web.servlet.context.AnnotationConfigServletWebInitializer@53d1f3c1]
2024-04-10T08:31:28.933+02:00 INFO 5628 --- [UdBE01HolaMundo] [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2024-04-10T08:31:29.041+02:00 INFO 5628 --- [UdBE01HolaMundo] [main] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Added connection conn...
2024-04-10T08:31:29.042+02:00 INFO 5628 --- [UdBE01HolaMundo] [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2024-04-10T08:31:29.080+02:00 INFO 5628 --- [UdBE01HolaMundo] [main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo
2024-04-10T08:31:29.106+02:00 INFO 5628 --- [UdBE01HolaMundo] [main] org.hibernate.Version : HHH000412: Hibernate ORM core version
2024-04-10T08:31:29.121+02:00 INFO 5628 --- [UdBE01HolaMundo] [main] o.h.c.internal.RegionFactoryInitiator : HHH000026: Second-level cache disabled
2024-04-10T08:31:29.276+02:00 INFO 5628 --- [UdBE01HolaMundo] [main] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup; ignoring JP
2024-04-10T08:31:29.444+02:00 INFO 5628 --- [UdBE01HolaMundo] [main] o.h.e.t.j.p.l.JtaPlatformInitiator : HHH000489: No JTA platform available
2024-04-10T08:31:29.447+02:00 INFO 5628 --- [UdBE01HolaMundo] [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'spring_jpa'
2024-04-10T08:31:29.496+02:00 INFO 5628 --- [UdBE01HolaMundo] [main] JpaBaseConfiguration$JpaWebConfiguration : spring_jpa.open-in-view is enabled b
2024-04-10T08:31:29.511+02:00 INFO 5628 --- [UdBE01HolaMundo] [main] o.s.b.a.w.s.WelcomePageHandlerMapping : Adding welcome page template: index.html
2024-04-10T08:31:29.798+02:00 INFO 5628 --- [UdBE01HolaMundo] [main] o.s.d.u.UdBE01HolaMundoApplication : Tomcat started on port 9000 (http://localhost:9000)
2024-04-10T08:31:29.798+02:00 INFO 5628 --- [UdBE01HolaMundo] [main] c.s.d.u.UdBE01HolaMundoApplication : Started UdBE01HolaMundoApplication in 1.119 seconds (JVM running for 1.121)
2024-04-10T08:31:29.859+02:00 WARN 5628 --- [UdBE01HolaMundo] [l-1 housekeeper] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Thread starvation or
2024-04-10T09:00:04.213+02:00 WARN 5628 --- [UdBE01HolaMundo] [l-1 housekeeper] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Thread starvation or
2024-04-10T09:00:51.322+02:00 WARN 5628 --- [UdBE01HolaMundo] [l-1 housekeeper] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Thread starvation or
2024-04-10T09:02:35.803+02:00 WARN 5628 --- [UdBE01HolaMundo] [l-1 housekeeper] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Thread starvation or
2024-04-10T10:03:27.293+02:00 WARN 5628 --- [UdBE01HolaMundo] [l-1 housekeeper] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Thread starvation or
2024-04-10T10:18:39.834+02:00 WARN 5628 --- [UdBE01HolaMundo] [l-1 housekeeper] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Thread starvation or
2024-04-10T10:43:09.071+02:00 WARN 5628 --- [UdBE01HolaMundo] [l-1 housekeeper] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Thread starvation or
2024-04-10T11:54:40.071+02:00 WARN 5628 --- [UdBE01HolaMundo] [l-1 housekeeper] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Thread starvation or
2024-04-10T11:03:23.694+02:00 WARN 5628 --- [UdBE01HolaMundo] [l-1 housekeeper] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Thread starvation or
2024-04-10T11:24:01.349+02:00 WARN 5628 --- [UdBE01HolaMundo] [l-1 housekeeper] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Thread starvation or
2024-04-10T11:33:51.552+02:00 WARN 5628 --- [UdBE01HolaMundo] [l-1 housekeeper] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Thread starvation or
2024-04-10T11:35:45.103+02:00 WARN 5628 --- [UdBE01HolaMundo] [l-1 housekeeper] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Thread starvation or
2024-04-10T11:39:39.464+02:00 WARN 5628 --- [UdBE01HolaMundo] [l-1 housekeeper] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Thread starvation or
2024-04-10T12:33:13.772+02:00 WARN 5628 --- [UdBE01HolaMundo] [l-1 housekeeper] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Thread starvation or
2024-04-10T12:38:20.630+02:00 WARN 5628 --- [UdBE01HolaMundo] [l-1 housekeeper] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Thread starvation or

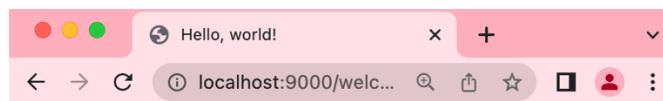
```

Para ver nuestra aplicación, basta con ir a un navegador y escribir en la barra cualquiera de estas dos rutas:

localhost:9000/

localhost:9000/welcome

Si la escribimos en el navegador, veremos que aparece nuestra esperada vista.



Hola Mundo

Bienvenido al... Infierno

Cuando queramos parar la ejecución, tenemos el botón stop o si queremos parar y ejecutar de nuevo el icono Relaunch

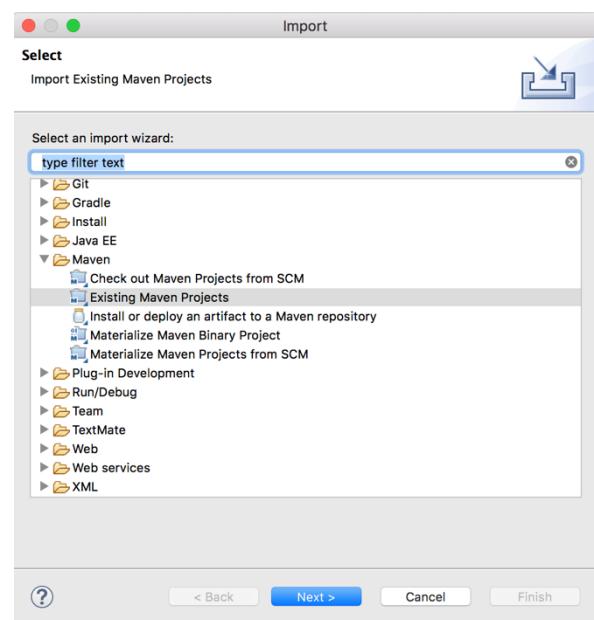


Antes de volver a compilar y ejecutar un proyecto, debemos pararlo para que no se acumulen las ejecuciones de todos los anteriores.

Para importar un proyecto ya creado

Basta con pinchar en File/import y elegir dentro de la carpeta Maven "Existing Maven Projects".

Nos dirá si queremos copiar el proyecto a importar en el espacio de trabajo y en este sentido, todo es igual a como se hace en Eclipse.



Otros dos ejemplos de controladores

Puedes agregar estos métodos al ejemplo anterior:

```
/*
 * Ejemplo 2: Segundo controlador llamado welcome2
 * Atiende a la petición /saludo2 escrita en el @GetMapping
 *
 * Se puede ver cómo se "pasa o carga información" al
 * model dentro del método mediante el método addAttribute
 * como un objeto de la clase Persona, con valores de sus dos
 * atributos. Más adelante iremos viendo de dónde sacamos esa
 * "información" que
 * cambiar, como aquí la new Persona creada directamente en el método
 * addAttribute.
 */

@GetMapping("/saludo2")
public String welcome2(Model model) {

    model.addAttribute("persona", new Persona("Ángel", "Naranjo
    González"));
    return "SaludoPersonalizado";
}

/*
```

```

    * @GetMapping es una variante de requestMapping, más "nueva" que se
utiliza
    * como atajo, ya que basta con el nombre del recurso, mientras que
    * con @RequestMapping, en general, tenemos que ir diciendo el tipo de
petición
    * que se está atendiendo, es decir, necesita que le indiquemos el value
    * (nombre el recurso, por ejemplo, /saludo3) y el método que se usa para
la
    * petición, en nuestro caso, tendríamos que escribir:
    * @RequestMapping (value="/saludo3", method=RequestMethod.GET)
    * (también existe RequestMethod.POST)
    * En general, se usa por simplificar el código
    * Se puede ver un ejemplo en:
    * https://www.arquitecturajava.com/spring-getmapping-postmapping-etc/
    *
    * Nosotros usaremos siempre GetMapping
    */
}

/*
 * Ejemplo 3: Tercer controlador llamado welcome3
 * Atiende a la petición get "/saludo3"
 *
 */

@GetMapping ("/saludo3")
public String welcome3(Model model) {

    model.addAttribute("saludo", "Hola Mundo");
    model.addAttribute("mensaje", "¡Se me está haciendo largo el
    proyecto final!");
    model.addAttribute("url", "https://triana.salesianos.edu");

    return "SaludoYEnlace";
}

```

Las páginas html de estos nuevos ejemplos se hacen igual a la primera llamada index, new File...:

Para welcome2:

SaludoPersonalizado.html

```

<!doctype html>
<html lang="es" xmlns:th="http://www.thymeleaf.org">

<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
    shrink-to-fit=no">

    <title>Saludo personalizado</title>
</head>

```

```
<body>

    <h1>
        Hola <span th:text="${persona.nombre + ' ' + persona.apellidos}">
        amigo!</span>
    </h1>

</body>
</html>
```

Para welcome3:

SaludoYEnlace.html

```
<!doctype html>
<html lang="es" xmlns:th="http://www.thymeleaf.org">

<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
    shrink-to-fit=no">

        <title>Saludo y enlace</title>
</head>

<body>

    <h1 th:text="${saludo}"> </h1>
    <p th:text="${mensaje}"></p>
    <p>
        <a class="btn btn-primary btn-lg" target="_blank" th:href="${url}"
        role="button">Learn more &raquo;</a>
    </p>

</body>
</html>
```

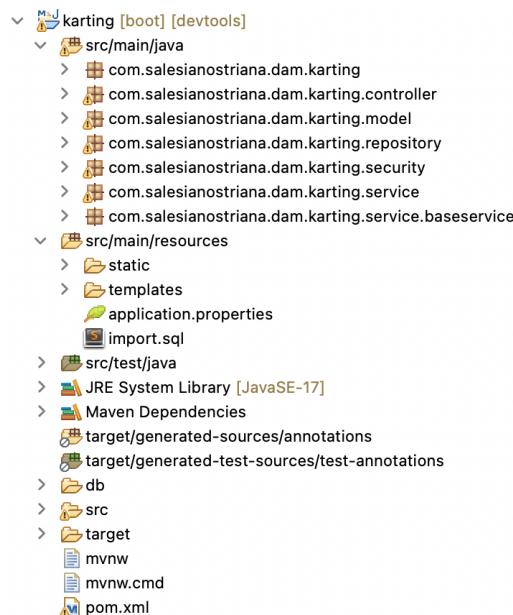
3. Primeros pasos con Thymeleaf

A partir de ahora, empezaremos a estudiar las diferentes formas de usar Thymeleaf, en cuanto a qué componentes pintar y cómo hacerlo.

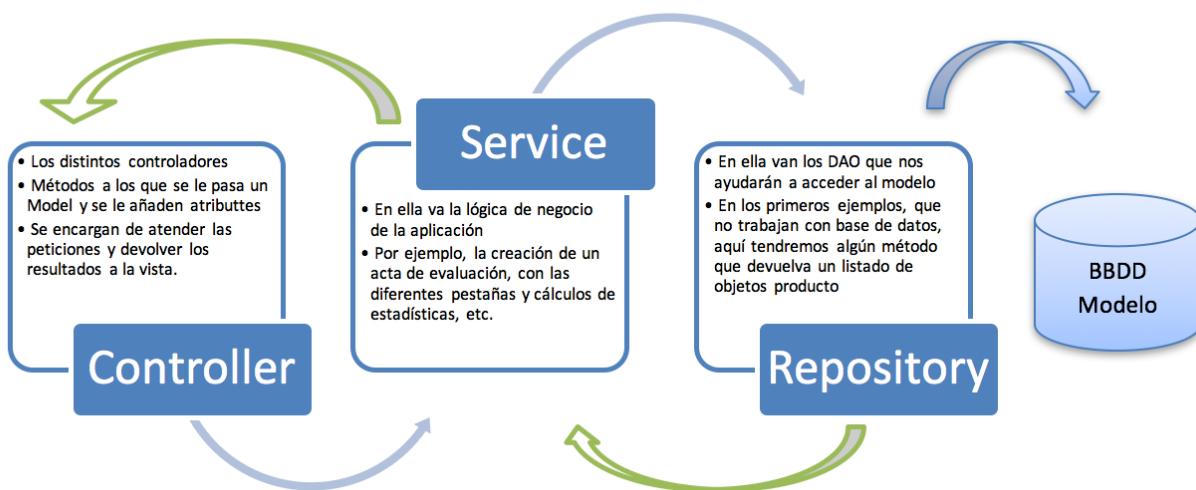
Utilizaremos algunos videotutoriales sueltos de internet y los propios tutoriales de Thymeleaf.

Para elementos más complejos con Thymeleaf y su integración en el proyecto iremos estudiando ejemplos concretos.

La estructura de carpetas de un proyecto más completo, será más o menos como en el siguiente:



Para tener una visión general de qué va en cada parte:



4. Configuración de Thymeleaf

Se puede hacer de varias formas, por ejemplo, mediante el uso de JavaConfig.

Nosotros configuraremos nuestras distintas opciones de Thymeleaf escribiendo lo necesario en el archivo application.properties en lugar de hacerlo usando javaConfig. Se irá viendo a lo largo de los distintos ejemplos en otros módulos.

