

UNIDAD 8: APLICACIONES WEB CON SPRING, MVC Y THYMELEAF

Índice

1. ¿Qué es Thymeleaf?	2
2. Creación de un “New Spring Starter Project Spring Boot”	6
3. Primeros pasos con Thymeleaf	24
4. Configuración de Thymeleaf	25

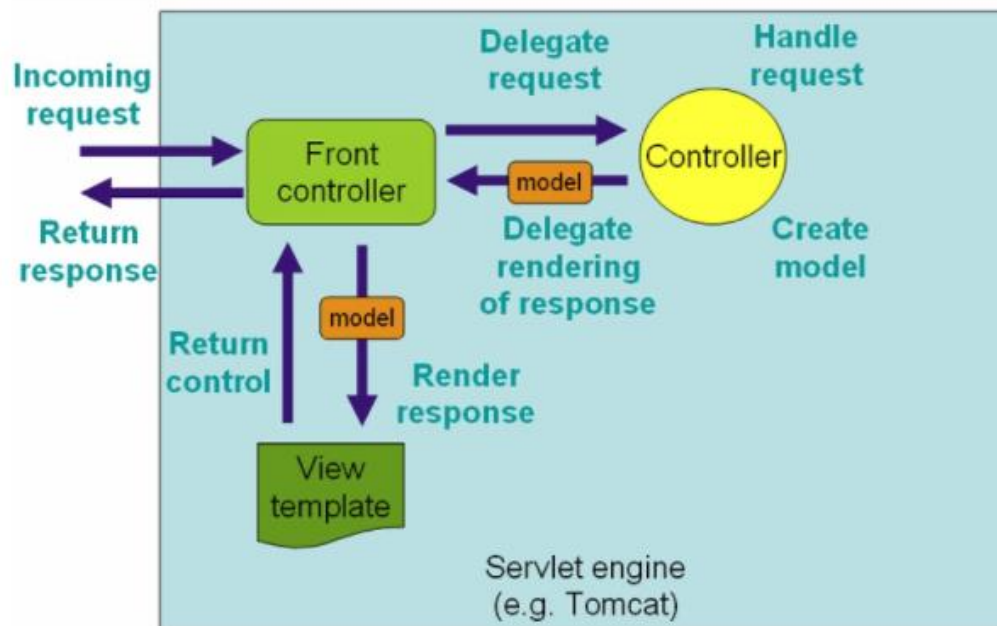


1. ¿Qué es Thymeleaf?

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* (el `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:



<http://spring.io/guides/gs/serving-web-content/>

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

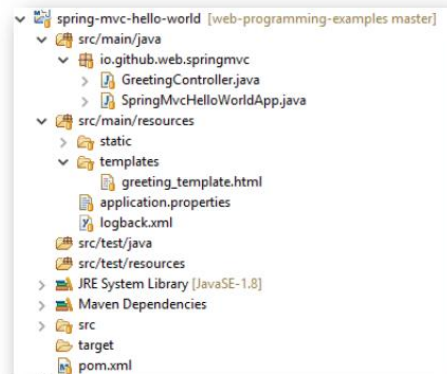
- Las vistas en Spring MVC se implementan como plantillas HTML.
- Generan HTML partiendo de una plantilla y la información que viene del controlador (modelo).
- Existen diversas tecnologías de plantillas que se pueden usar con Spring MVC: JSP, Thymeleaf, FreeMarker, etc...
- Nosotros usaremos **Thymeleaf**.

APLICACIONES WEB CON SPRING MVC Y THYMELEAF

1. Spring MVC

Ejemplo

- Estructura de la aplicación vista desde Eclipse:



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.



Renderizado

As a result, you'll have the following full-featured HTML table:

Show 10 entries

ID	Firstname	Lastname	Street	Mail
1	Salma	Maldonado	Ap 4551-1512 Eu Ave	venenata@Ouzosokpat.com
2	Vanna	Salas	947-3625 Fugiat St	blondum fermentum metus@ante.ca
3	Nadia	Roulet	1289 Aliquam St	Duis cursus diam@idmetipellentesque.ca
4	Rafael	Gonzalez	P.O. Box 374, 9474 Lacrima Street	tatac non magna@ante.edu
5	Calvin	Hill	8968 Eu Road	ipsum Etiam imperdiet@idametmetus.co.uk
6	Chana	Waller	317-8083 Ligula St	lacinia vestibulum@incursus.edu
7	Oleiv	Rivas	6489 Sed Ave	velit@sedmetusvelit.co.uk
8	Lita	Munoz	8438 Vtama Ave	Mauris ne luptus@in.ca
9	Yeni	Vargas	P.O. Box 722, 9679 Eu St	odam@consect.net
10	Lee	Graves	211-4591 Dictum Road	velit@faw.co.uk

Showing 1 to 10 of 100 entries

Previous Next

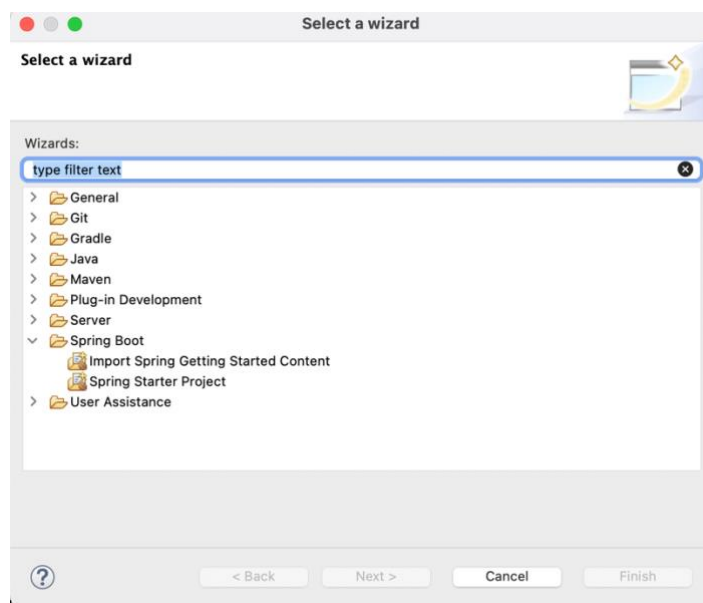
2. Creación de un “New Spring Starter Project Spring Boot”

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

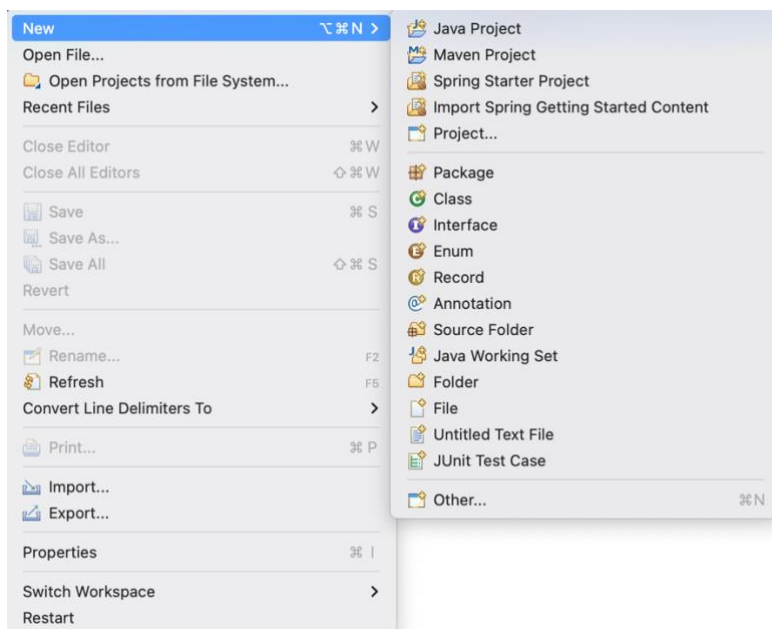
Veámoslo paso a paso.

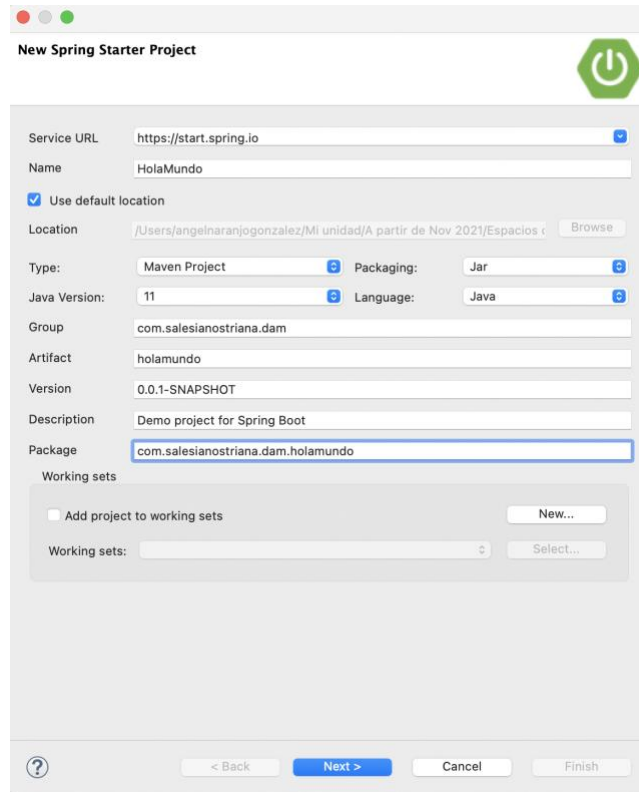
Paso 1: New/Spring Starter Project

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



Posteriormente, ya aparecerá en recientes:





New Spring Starter Project

Service URL:

Name:

☒ Use default location

Location:

Type: Packaging:

Java Version: Language:

Group:

Artifact:

Version:

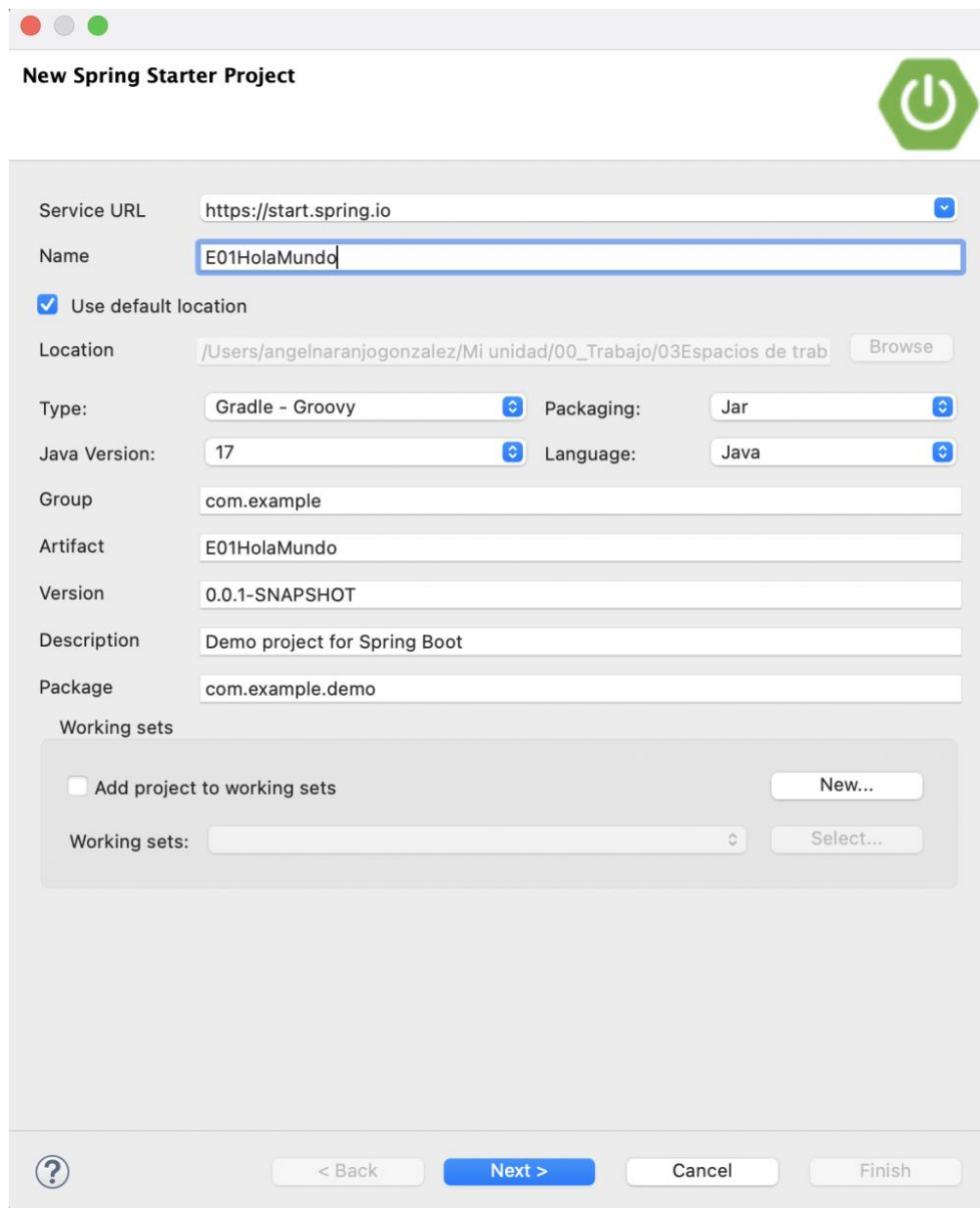
Description:

Package:

Working sets

☐ Add project to working sets

Working sets:



Debemos cambiar varias cosas:

Escribimos los distintos nombres del proyecto, Group, Artific...

- Name: Nombre del proyecto (mismo criterio de siempre sobre mayúsculas y minúsculas).
- Type: Cambiar a Maven.
- Java version: 17
- Group: Nombre del dominio del colegio (imaginario, no es real) pero siempre pondremos: com.salesianostriana.dam

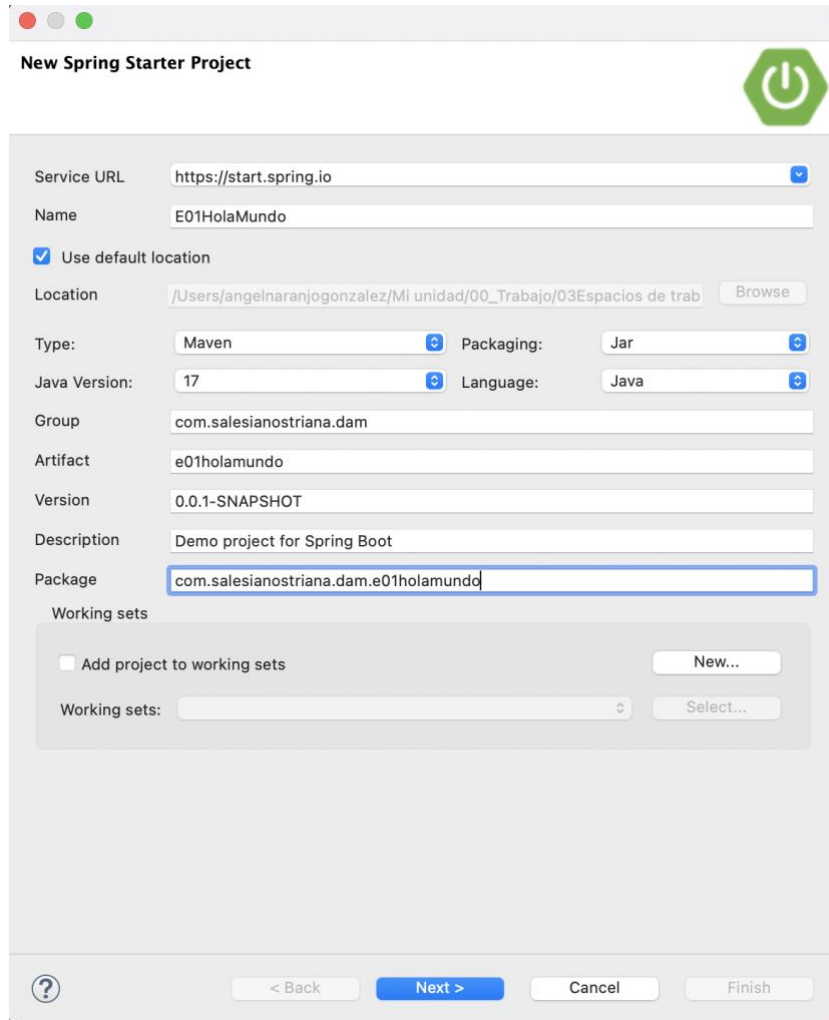
Es la base o raíz de las rutas que usaremos en el proyecto para las distintas carpetas y archivos.

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

Un artifact es como se le llama en Maven a un proyecto.

- Package: Composición del nombre del group y del artifact, todo en minúscula.
- Next

Quedaría algo así:

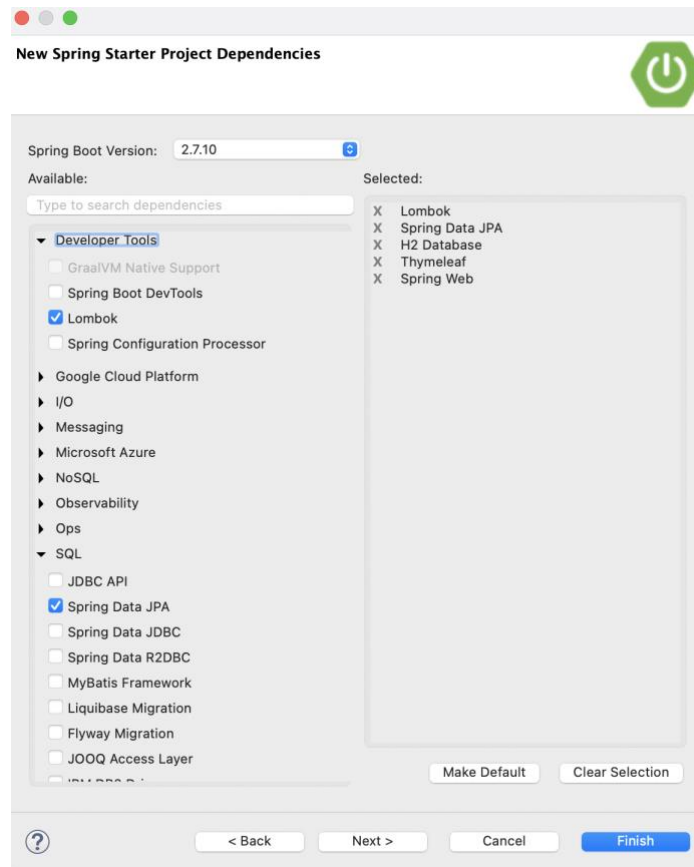


Debemos marcar los tipos de dependencias que queremos incluir. Al menos de momento debemos marcar:

- 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.

En próximos proyectos, nos aparecerán separadas las “usadas frecuentemente” y también se podrán agregar más tarde.

Elegimos la versión de spring boot 2.7.10

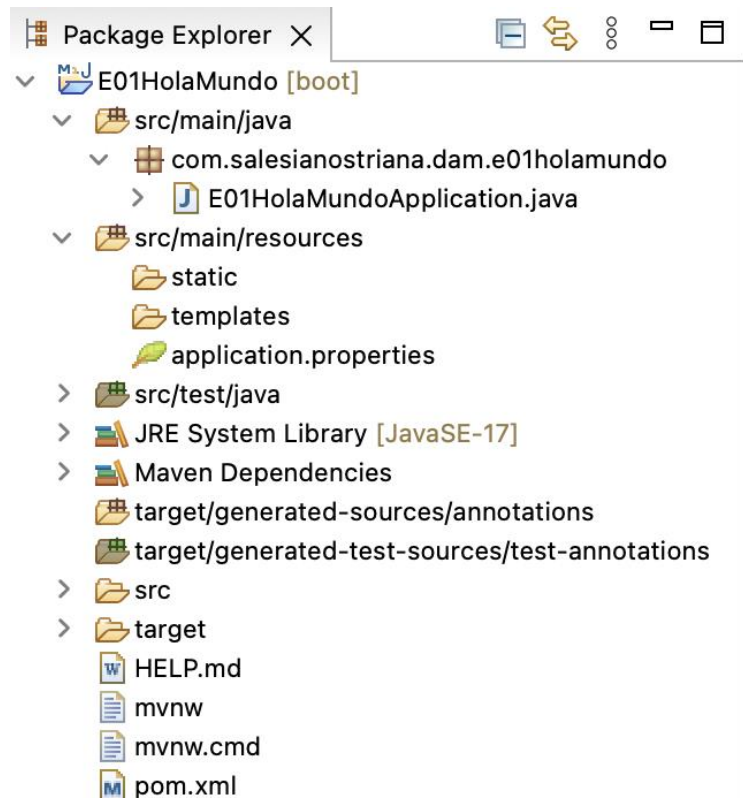


- **Finish**

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.

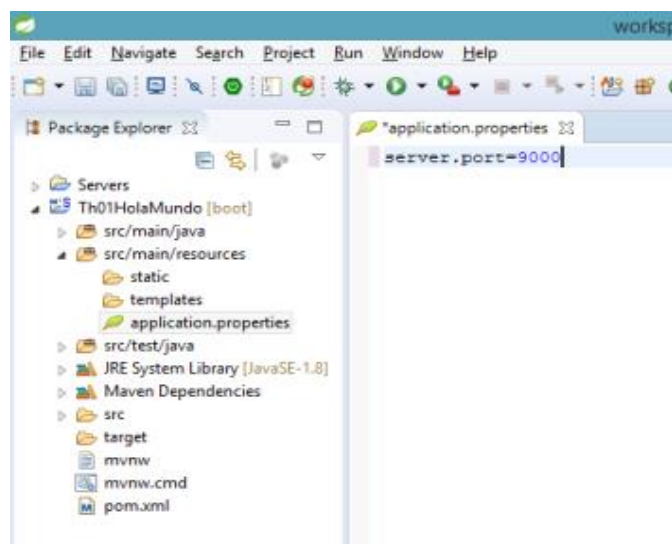
- **Estructura de carpetas creada**



- **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.



Paso 3: 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 (COMMAND+SHIFT+LETRA O en Mac)

Por otro lado, debemos observar que ya se ha creado la clase principal, llamada Th01HolaMundoApplication.java, 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.

a) **Creación del controlador:** Debemos crear una clase dentro del paquete que ya tenemos creado src/main/java/com.salesianostriana.dam.th01holamundo. A este controlador, le llamaremos ControladorSaludo.java

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

```
package com.salesianostriana.dam.ethholamundo;
```

```
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, un controller que se encarga  
 * de atender las peticiones. Los controller:  
 * Atienden las peticiones  
 * Procesan los datos que llegan de las peticiones  
 * Hacen peticiones a la BBDD  
 * Definen la información que se verá en la web (modelo)  
 * Determinan qué vista será la encargada de generar la pg HTML  
 *  
 */
```

```
@Controller
```

```
public class MainController {
```

```
    /*" / ó /welcome" es la página que queremos mostrar al inicio, el recurso a mostrar y que tendremos que escribir en el navegador después de
```

```
    localhost:9000 o nada o welcome @RequestMapping indica que el método justo debajo será el que atenderá la petición tipo Get (cuando solo hay una se supone tipo Get, cuando sea Post (formulario) tendremos dos métodos. En el próximo método, veremos que en lugar de usar RequestMapping usaremos GetMapping, pero por si os lo encontráis por el mundo de internet*/
```

```
    @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 nombre 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 templates y 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
```

```
    }
```

```
    //Segundo ejemplo, donde se pueden ver distintos parámetros que se le pasan al model, 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/*/

    @GetMapping("/saludo3")

    public String welcome3 (Model model){

        model.addAttribute("saludo", "Hola Mundo!!!");
        model.addAttribute("mensaje", "Me llena de orgullo y satisfacción...");
        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 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

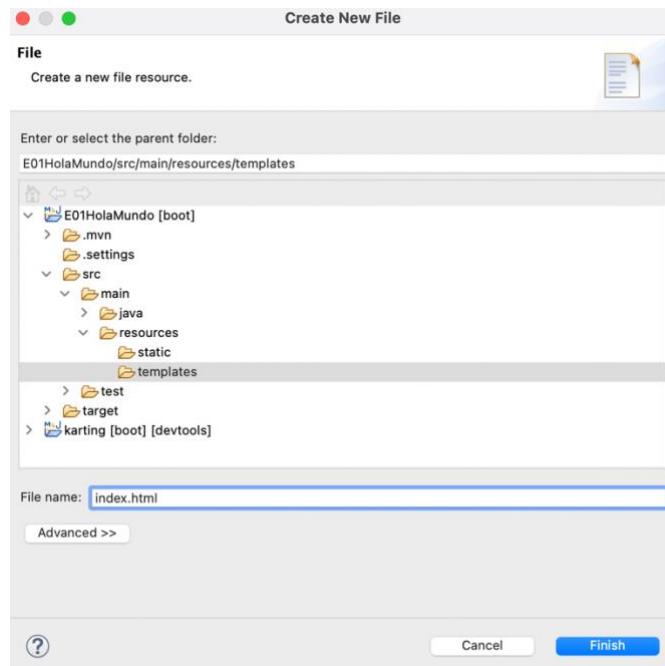
al model, de momento, solo con addAttribute.

- 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.

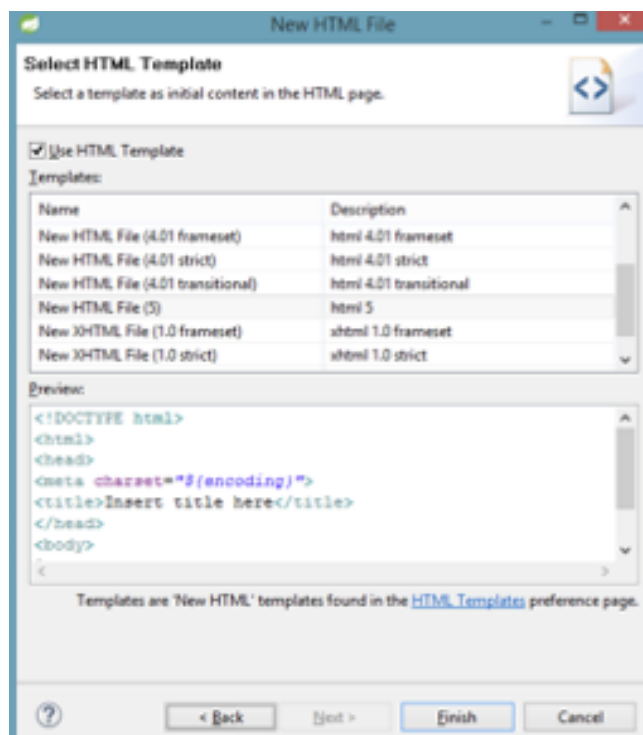
b) Creación de la vista

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*.

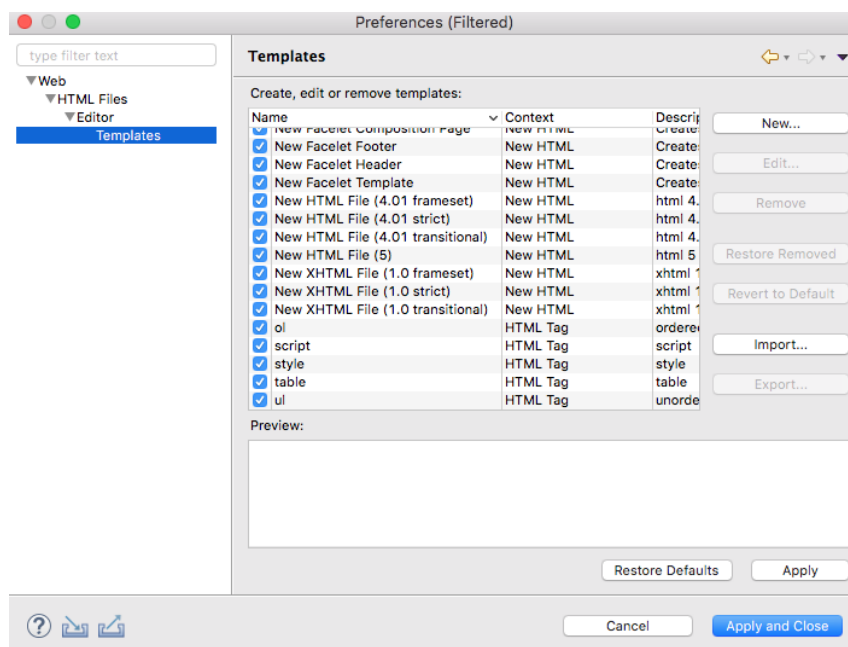
Para ello basta crear un archivo tipo HTML File pinchando con el botón derecho en la carpeta *templates*, new y crear un archivo tipo File y añadir en el nombre la extensión *.html*. El nombre de este archivo debe ser el mismo que el String que devuelve el método controlador que atiende a la petición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



En algunos casos, si detecta el tipo html escribiendo HT... en wizards, habrá que seleccionar y next, escribimos el nombre de la plantilla, next y nos aseguramos de que está marcada la opción de plantilla HTML 5:



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:



Copiamos y pegamos el siguiente código en el archivo html.

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

En la segunda línea se crea lo que se llama "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>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="/webjars/bootstrap/css/bootstrap.min.css" integrity="sha384-
Vko08x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">

    <title>Hello, world!</title>
  </head>
  <body>

    <div class="container">
      <div class="jumbotron">
        <h1>
          Hola <span th:text="{nombre}">mondo</span>
        </h1>
        <h2>Bienvenido al... Infierno</h2>
      </div>
    </div>

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="/webjars/jquery/jquery.min.js" integrity="sha384-
J6qa4849bIE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n" crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-
Q6E9RHvblyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtm13UksdQRVvoxMfooAo" crossorigin="anonymous"></script>
    <script src="/webjars/bootstrap/js/bootstrap.min.js" integrity="sha384-
wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6" crossorigin="anonymous"></script>
  </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.

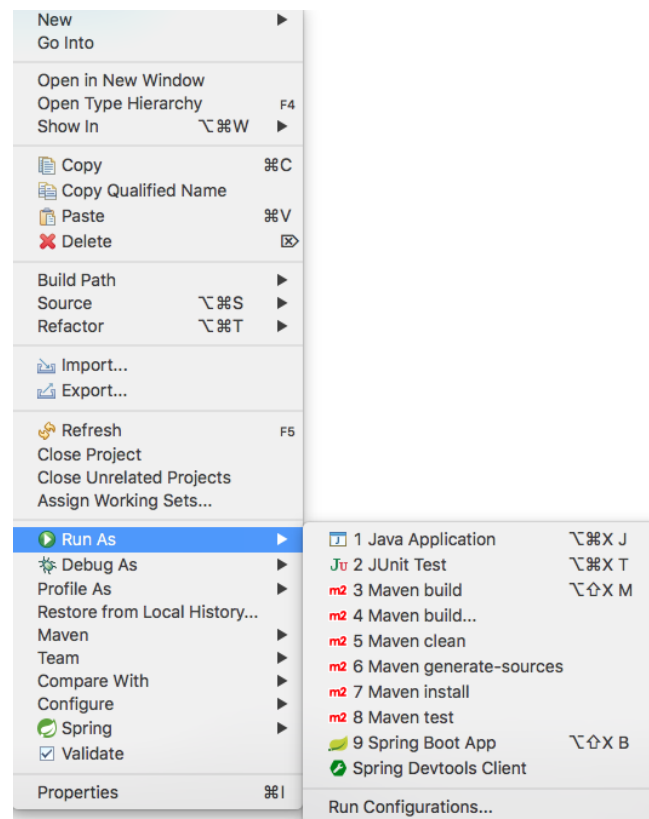
Además, encontramos que hemos usado *Expression Language*, una característica soportada por Thymeleaf que nos permite utilizar un lenguaje sencillo para usar un *JavaBean* dentro de una página HTML. $\${nombre}$ buscará dentro del contexto (ámbito de vida de la petición) una variable llamada *nombre*, y mostrará su valor.

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

Para empezar a entender la parte de Thymeleaf, vamos a utilizar los tutoriales del mismo, 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:



```

2023-04-13 10:44:42.164 INFO 81127 --- [main] c.s.d.e.E01HolaMundoApplication : Starting E01HolaMundoApplication using Java 17.0.4.1 on MacBook-Pro-de-
2023-04-13 10:44:42.167 INFO 81127 --- [main] c.s.d.e.E01HolaMundoApplication : No active profile set, falling back to 1 default profile: "default"
2023-04-13 10:44:42.755 INFO 81127 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2023-04-13 10:44:42.770 INFO 81127 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 5 ms. Found 0 JPA repositor
2023-04-13 10:44:43.267 INFO 81127 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 9000 (http)
2023-04-13 10:44:43.279 INFO 81127 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-04-13 10:44:43.279 INFO 81127 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.73]
2023-04-13 10:44:43.379 INFO 81127 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2023-04-13 10:44:43.379 INFO 81127 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1130 ms
2023-04-13 10:44:43.562 INFO 81127 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2023-04-13 10:44:43.738 INFO 81127 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2023-04-13 10:44:43.788 INFO 81127 --- [main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
2023-04-13 10:44:43.830 INFO 81127 --- [main] org.hibernate.Version : HHH000412: Hibernate ORM core version 5.6.15.Final
2023-04-13 10:44:43.971 INFO 81127 --- [main] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2023-04-13 10:44:44.085 INFO 81127 --- [main] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
2023-04-13 10:44:44.244 INFO 81127 --- [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.tran
2023-04-13 10:44:44.253 INFO 81127 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2023-04-13 10:44:44.293 WARN 81127 --- [main] spring.jpa.open-in-view is enabled by default. Therefore, database quer
2023-04-13 10:44:44.444 INFO 81127 --- [main] o.s.b.a.w.s.WelcomePageHandlerMapping : Adding welcome page template: index
2023-04-13 10:44:44.632 INFO 81127 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 9000 (http) with context path ''
2023-04-13 10:44:44.643 INFO 81127 --- [main] c.s.d.e.E01HolaMundoApplication : Started E01HolaMundoApplication in 2.779 seconds (JVM running for 3.496
2023-04-13 10:44:52.356 INFO 81127 --- [nio-9000-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2023-04-13 10:44:52.357 INFO 81127 --- [nio-9000-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2023-04-13 10:44:52.359 INFO 81127 --- [nio-9000-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
  
```

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.



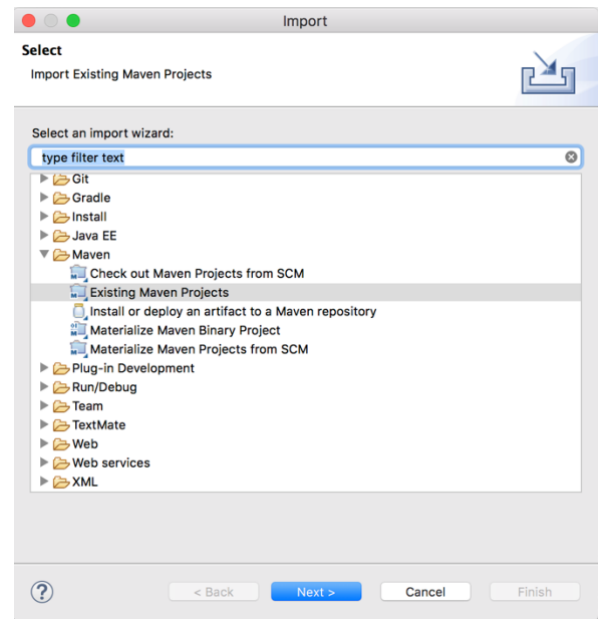
Hola Mundo

Bienvenido al... Infierno

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:

```
/*
 * Segundo ejemplo, donde se pueden ver distintos parámetros que se le pasan al
 * model, 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", "Naranja
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
- */

```
@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 serían:

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">

<!-- Bootstrap CSS -->
<link rel="stylesheet" href="/webjars/bootstrap/css/bootstrap.min.css"
      integrity="sha384-
Vko08x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh"
      crossorigin="anonymous">

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

<body>

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

<!-- Optional JavaScript -->
<!-- jQuery first, then Popper.js, then Bootstrap JS -->
<script src="/webjars/jquery/jquery.min.js"
        integrity="sha384-
J6qa4849bIE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n"
        crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
        integrity="sha384-
Q6E9RHvblyZFJoft+2mJbHaEwidlVl9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo"
        crossorigin="anonymous"></script>
<script src="/webjars/bootstrap/js/bootstrap.min.js"
        integrity="sha384-
wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6"
        crossorigin="anonymous"></script>
```

```
</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">
```

```
<!-- Bootstrap CSS -->
```

```
<link rel="stylesheet" href="/webjars/bootstrap/css/bootstrap.min.css"
```

```
integrity="sha384-
```

```
Vkoo8x4CGsO3+HhXv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh"
```

```
crossorigin="anonymous">
```

```
<title>Saludo y enlace</title>
```

```
</head>
```

```
<body>
```

```
<div class="jumbotron">
```

```
<div class="container">
```

```
<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>
```

```
</div>
```

```
</div>
```

```
<!-- Optional JavaScript -->
```

```
<!-- jQuery first, then Popper.js, then Bootstrap JS -->
```

```
<script src="/webjars/jquery/jquery.min.js"
```

```
    integrity="sha384-
```

```
J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n"
```

```
    crossorigin="anonymous"></script>
```

```
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
```

```
    integrity="sha384-
```

```
Q6E9RHvblyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo"
```

```
    crossorigin="anonymous"></script>
```

```
<script src="/webjars/bootstrap/js/bootstrap.min.js"
```

```
    integrity="sha384-
```

```
wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6"
```

```
    crossorigin="anonymous"></script>
```

```
</body>
```

```
</html>
```

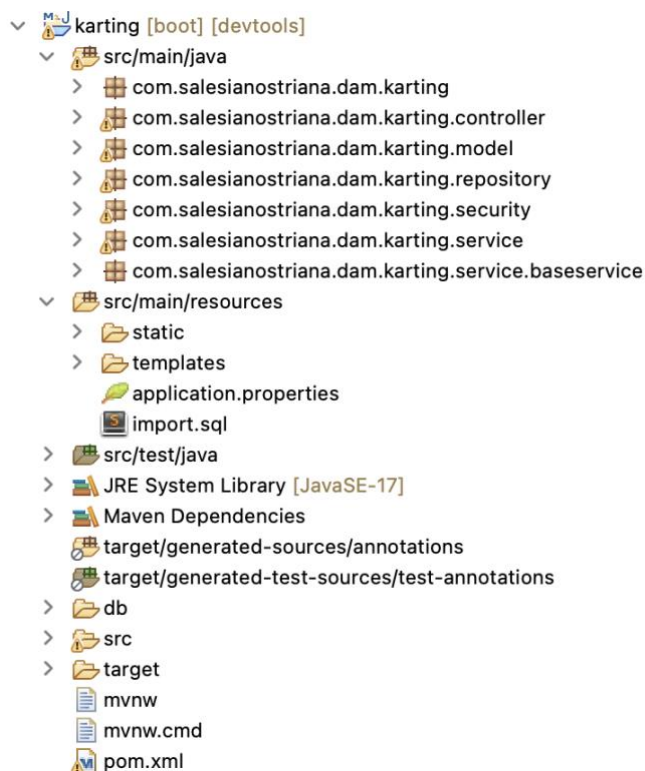
3. Primeros pasos con Thymeleaf

A partir de ahora, empezaremos a estudiar las diferentes formas de usar Thymeleaf.

Utilizaremos algunos tutoriales sueltos, como el interactivo que está colgado en la plataforma virtual (mostrar texto, condicionales, bucles), los propios tutoriales de Thymeleaf (tablas, etc.) y videotutoriales de internet.

Para elementos más complejos con Thymeleaf y su integración en el proyecto iremos viendo 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 las partes:



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 properties en lugar de hacerlo usando javaConfig. Se irá viendo a lo largo de los distintos ejemplos en otros módulos.