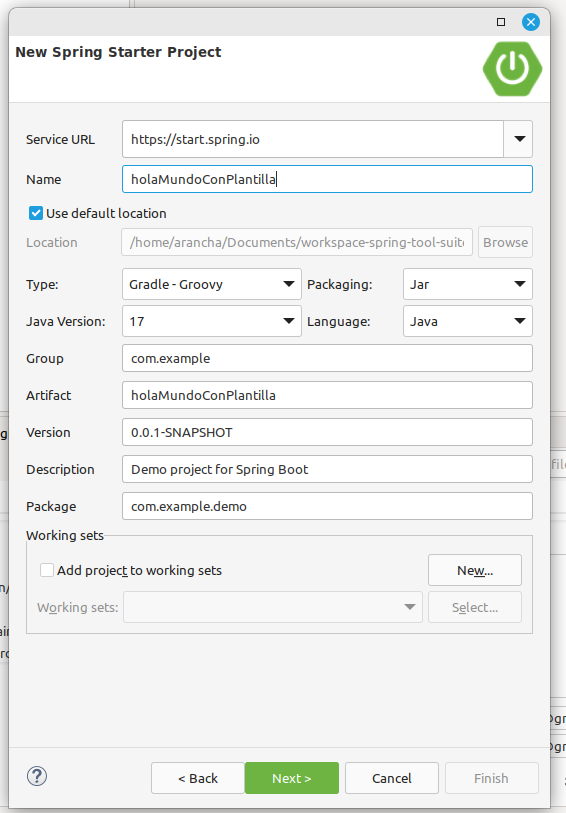
**Tarea 3 – UD4**

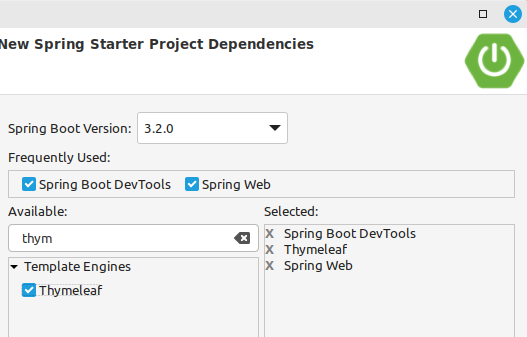
**FORMULARIOS**

**APARTADO 1:**

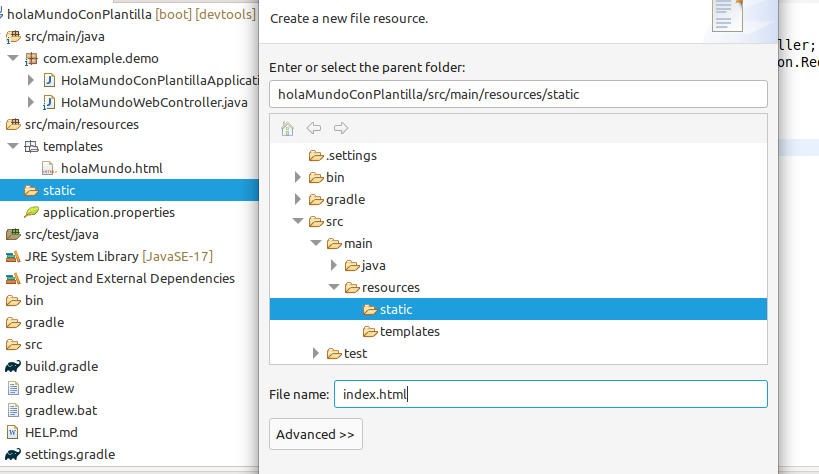
* **Tutorial 1: aplicación de Saludo con nombre por parámetro.**
  + Creo proyecto spring starter: holaMundoConPlantilla (tipo gradle-groovy, packaging en Jar y language en Java:



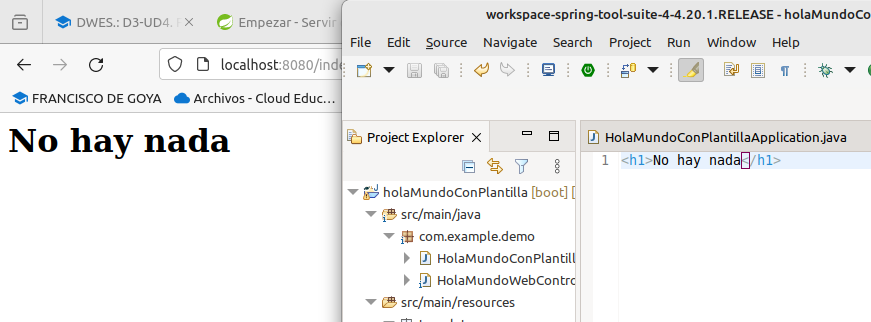
* + Selecciono las dependencias DevTools, SpringWeb y Tymeleaf.



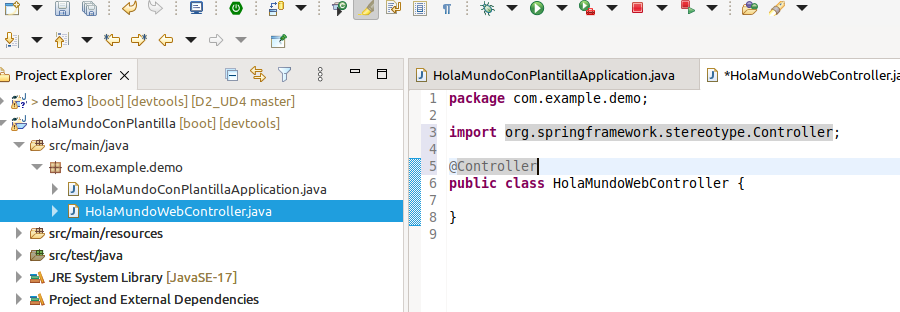
* Creo en carpeta static (dentro de resources) un index.html, la cual nos va a servir para asignar a nuestra aplicación una página predeterminada cuando acceda a la raíz, que servirá como una página estática sin procesamiento dinámico por parte del servidor.



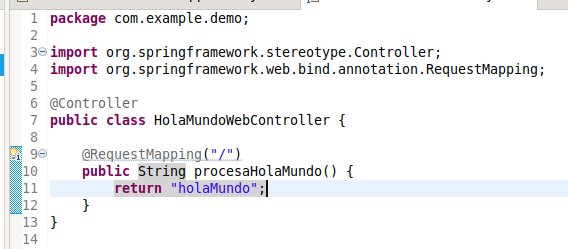
Ponemos línea de texto y cargamos en el navegador:



* Creo clase **Controlador** HolaMundoController:



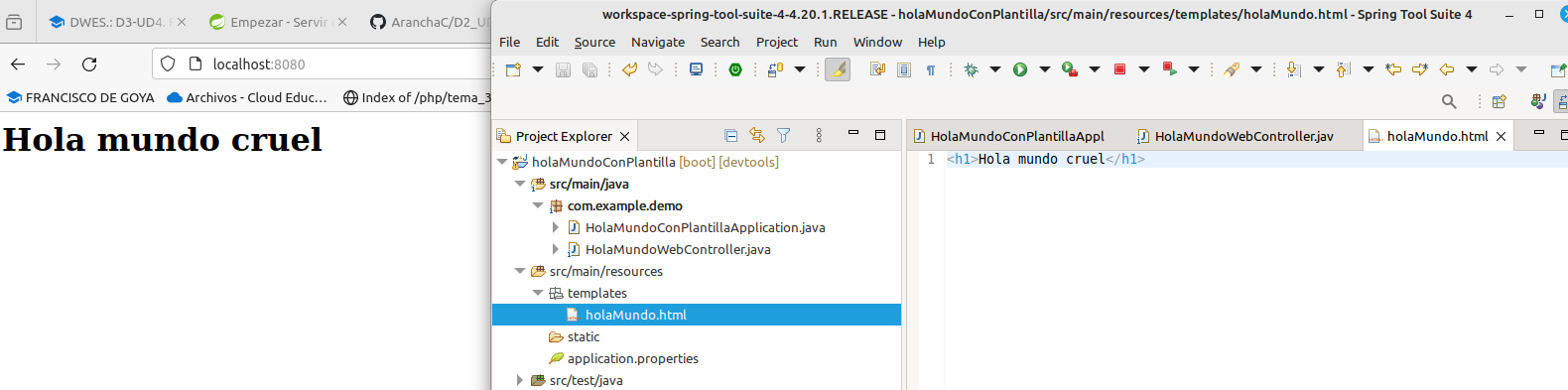
Y añado método procesaHolaMundo con anotación @RequestMapping a la raíz, que devuelve una plantilla holaMundo (la cuál correspondería a la vista):



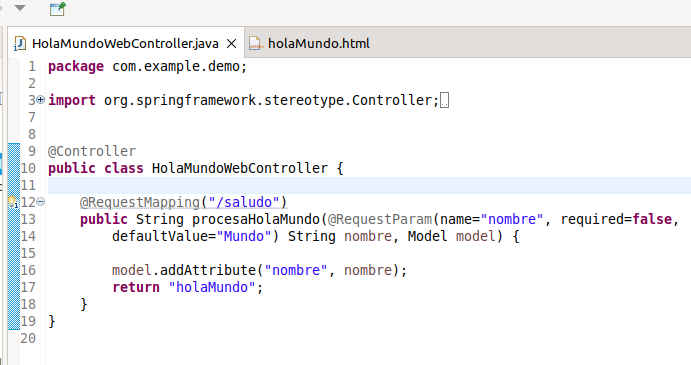
Ejecutamos y probamos en el navegador, da error porque no encuentra holaMundo (no lo hemos creado):



* **Creamos plantilla (VISTA)** html en carpeta templates, con una línea de prueba en etiqueta h1, y volvemos a ejecutar en el navegador. Ahora sí que sale correctamente el texto de nuestra plantilla:

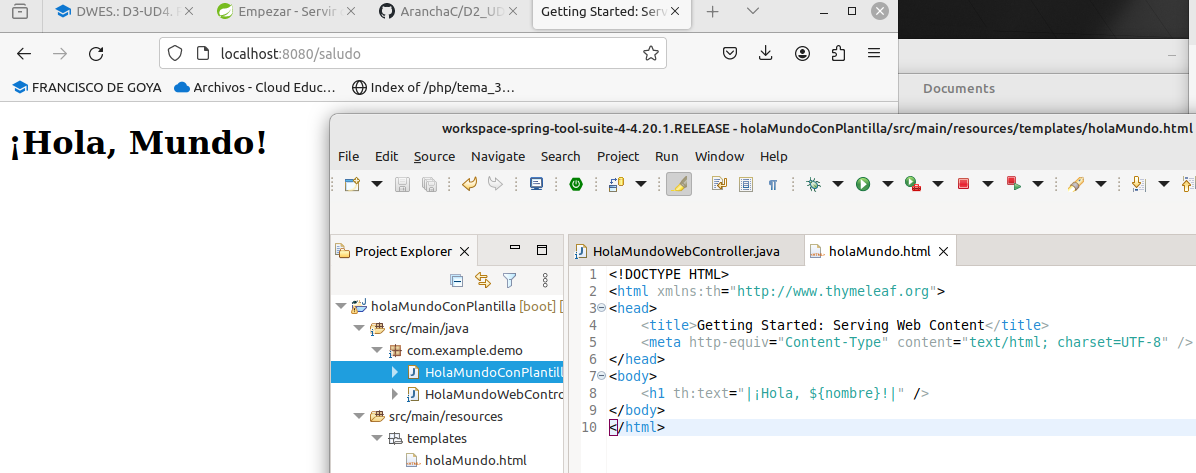


* Modificamos controlador para que reciba un parámetro nombre y usarlo en la plantilla para volcarlo en el navegador. Para que esto pase, se añade a un Model para hacerlo accesible a la plantilla medante model.addAtributte. Y si no recibe un parámetro, coge valor por defecto “Mundo”.

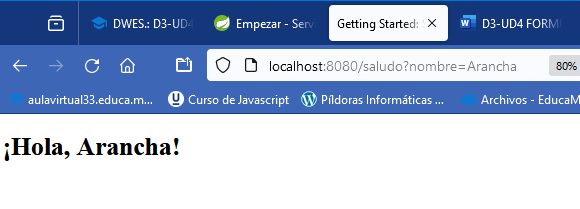


Modificamos la VISTA, plantilla holaMundo para que reciba el valor de nombre.

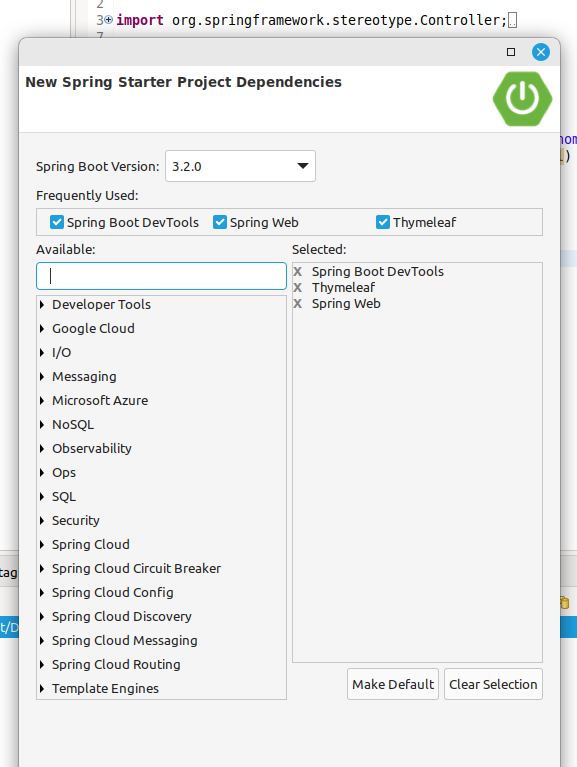
- Primero probamos en el navegador sin enviarle parámetro:

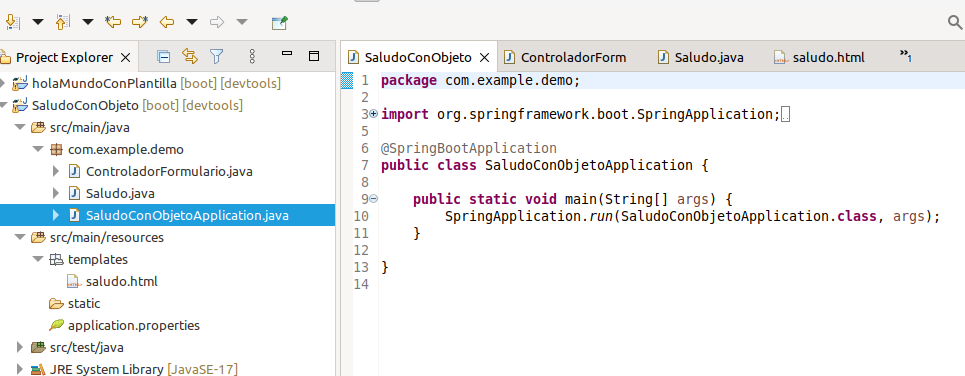


**-** Y ahora enviamos un parámetro nombre con valor Arancha:

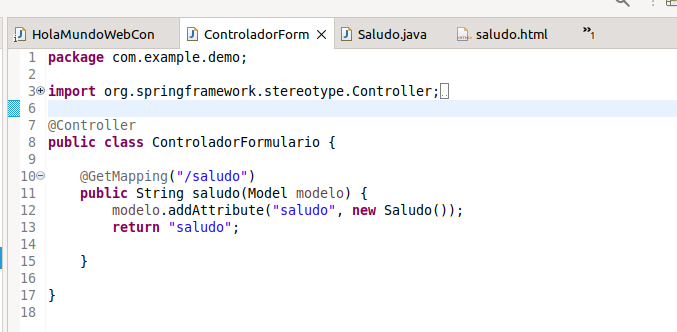


* **Tutorial 2: aplicación formulario de datos ID y mensaje sin parámetros.**
  + Creo proyecto spring starter llamado SaludoConObjeto con las 3 dependencias vistas en el ej anterior:

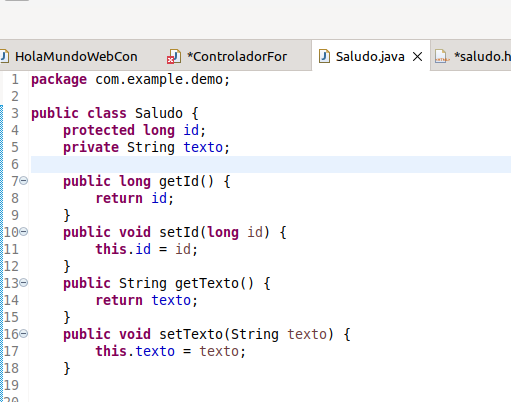
****

****

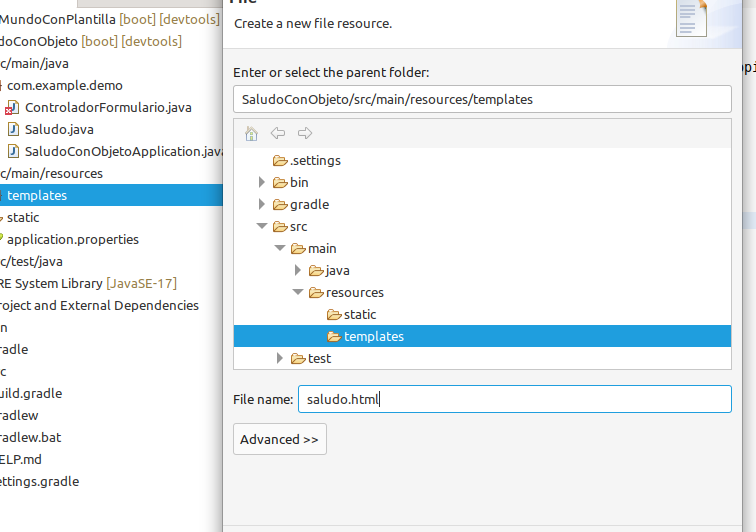
* **Creo Contolador** llamado ControladorFormulario. Añado anotación GetMapping /Saludo y método que recibe un Model y le añadimos el atributo saludo de tipo Objeto Saludo, y devuelve plantilla VISTA saludo:



* **Creo MODELO,** objeto, llamado Saludo con los atributos id y texto, y añado sus getters y setters:



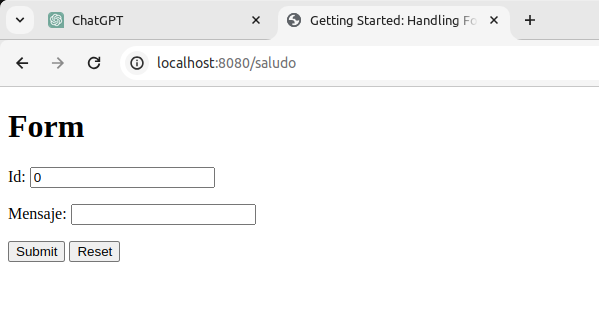
* **Creo VISTA** plantilla html llamada saludo.html (la que tiene que devolver según el getMapping del controlador:

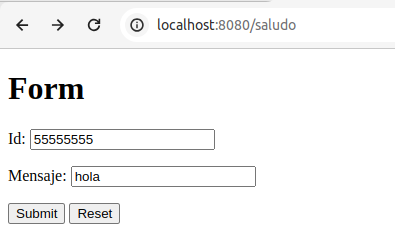


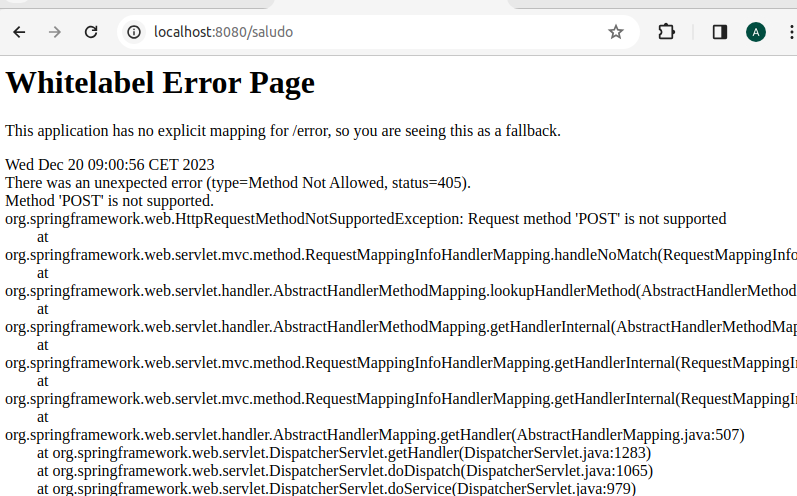
Copio plantilla del tutorial, adaptándola a mis datos



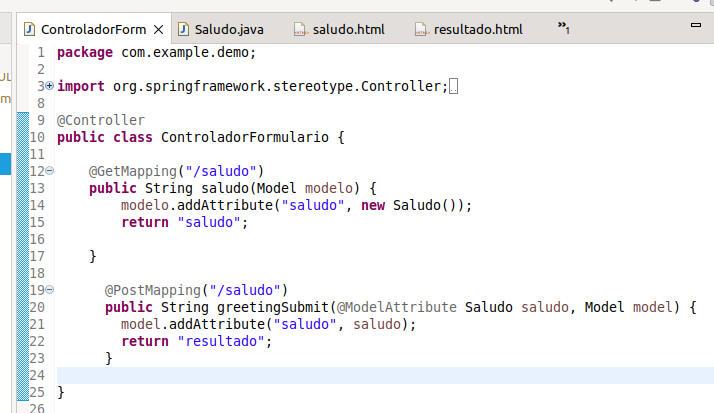
Y ejecutamos en el navegador poniendo saludo (el getMapping):



Ponemos datos y al enviar da error porque no hay método POST y según la plantilla, este formulario se envía por POST:  


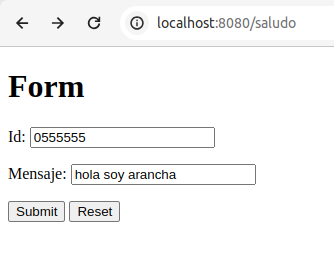
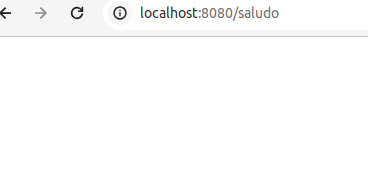


Modifico Controlador: pongo anotación con Post, que recibe objeto de tipo saludo y de tipo Model, que en model.addAttirbute, añadimos el valor de saludo al model para enviarlo a la vista/plantilla. Y devuelve plantilla/vista llamada resultado:



Ahora cargo página de nuevo en el navegador y envío datos, ya no da error, el formulario se envía a sí mismo y aparece nuevamente. Ahora funciona porque en la plantilla, el th:action="@{/saludo}" dirige el formulario a /saludo por POST.

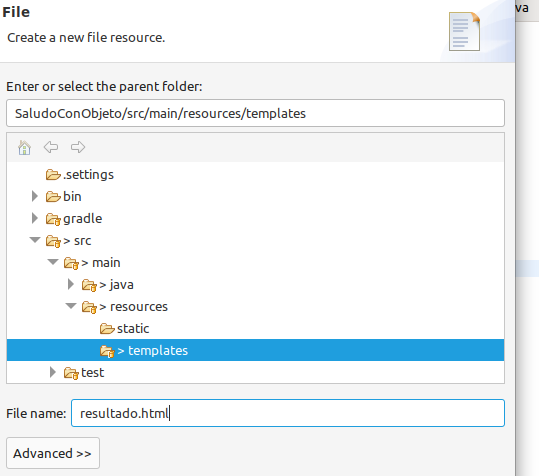
Envío los datos y no hay error, se envía el formulario aunque no aparece nada porque aún no hemos creado plantilla resultado:

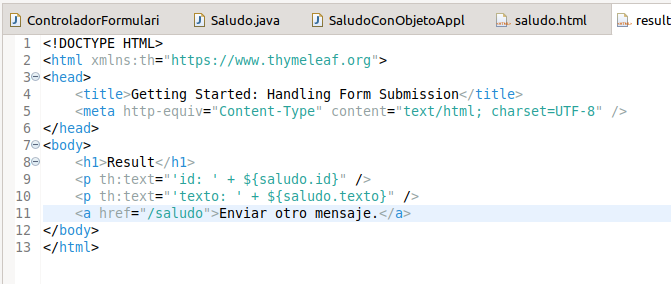
 

Esto cubre el modelo, vista y controlador para presentar el formulario. Ahora podemos revisar el proceso de presentación del formulario. El formulario se ejecuta en /saludo mediante el uso de una llamada POST.

El método del PostMapping recibe el objeto Saludo que es a @ModelAttribute, por lo tanto está unido al contenido de la plantilla, vista.

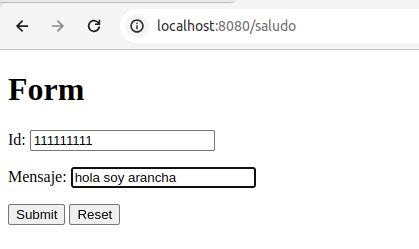
Ahora creo plantilla de resultado, adaptándola a mis datos. Pongo saludo.id y saludo.texto para captar los valores de mis atributos.

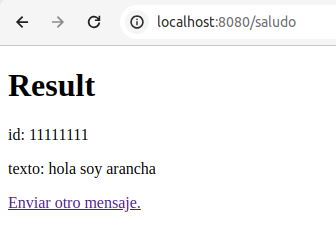




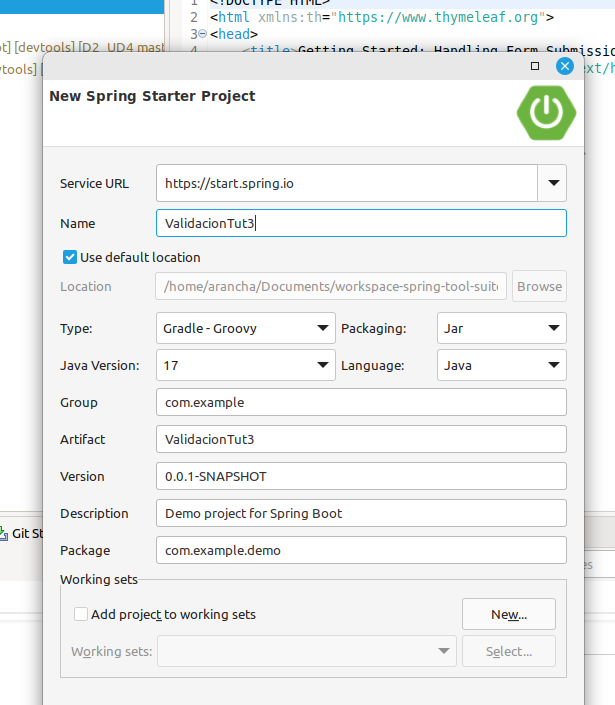
Podemos usar una sola plantilla para los dos propósitos, pero es más claro usar dos vistas separadas para representar el formulario y mostrar los daros.

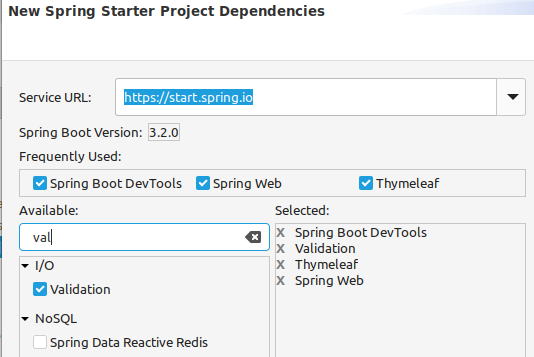
Ejecutamos nuevamente en navegador:

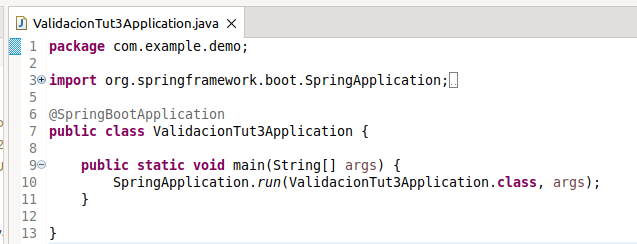


Envío datos y ahora aparece la página con los datos enviados, correspondiente a la plantilla html resultado:  


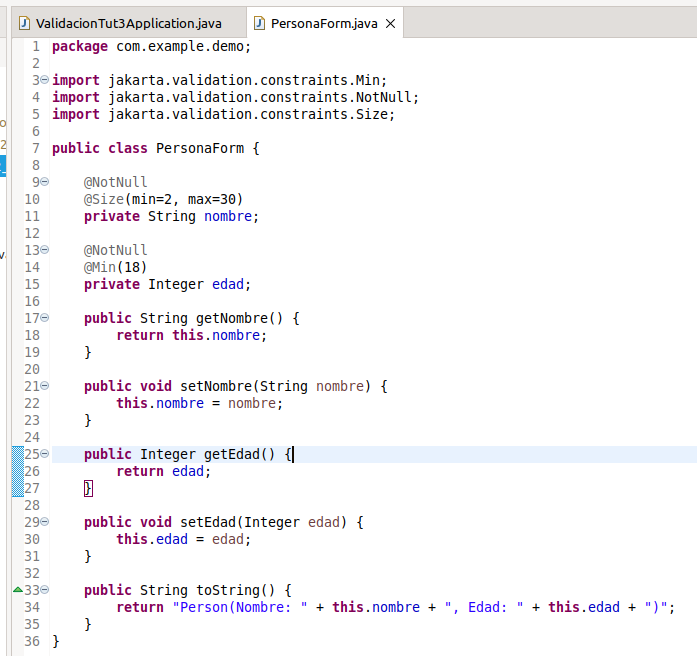
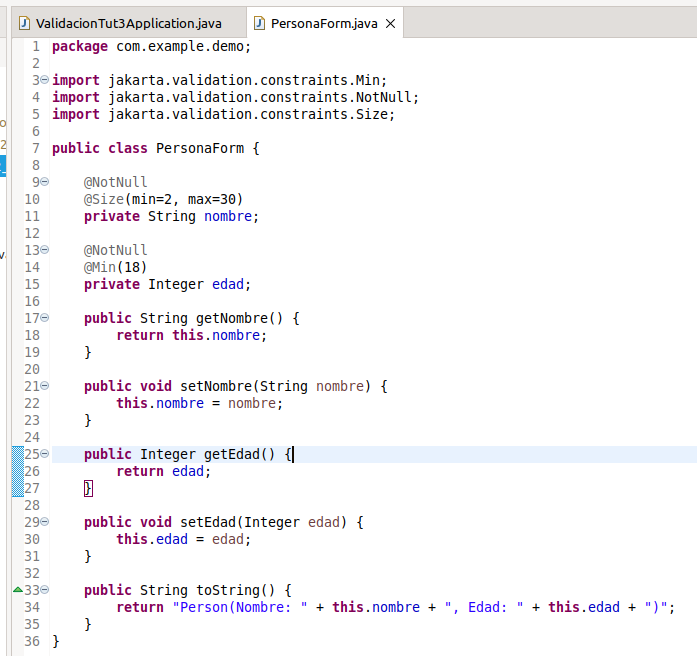
* **Tutorial 3: aplicación formulario de datos nombre y edad con validación.**
  + Creo proyecto spring starter llamado ValidacionTut3 con las 3 dependencias vistas y además añadimos la depencia validation:





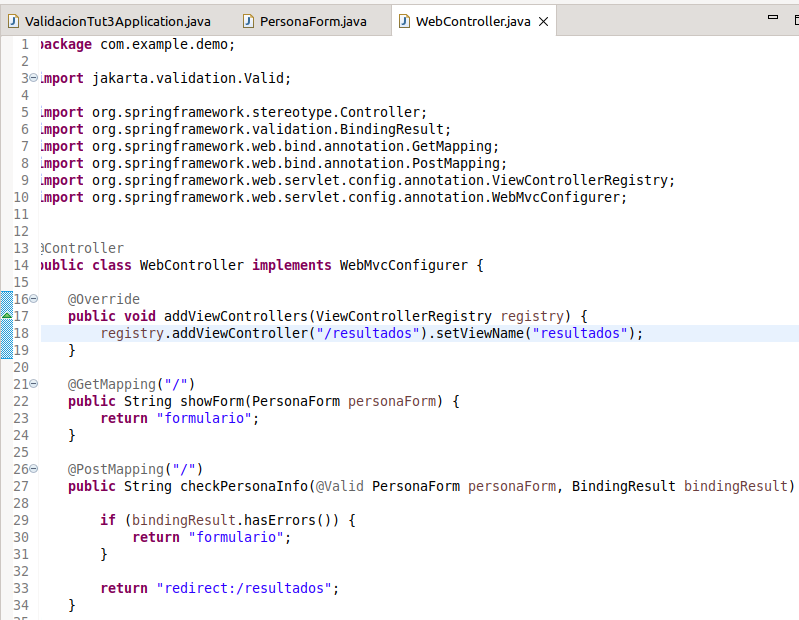


* Creamos **MODELO** PersonaForm con atributos nombre y edad y los getters y setters:

Describimos las siguientes anotaciones:

* [@NotNull](mailto:-@NotNull): hace que no permita valor nulo, que es lo que Spring genera si el campo está vacío.
* [@Size](mailto:-@Size): permite string entre el min y el max de caracteres de longitud.
* [@Min](mailto:-@Min): no permite valores inferiores a ese número.
* Creo **CONTROLADOR** llamado WebController:



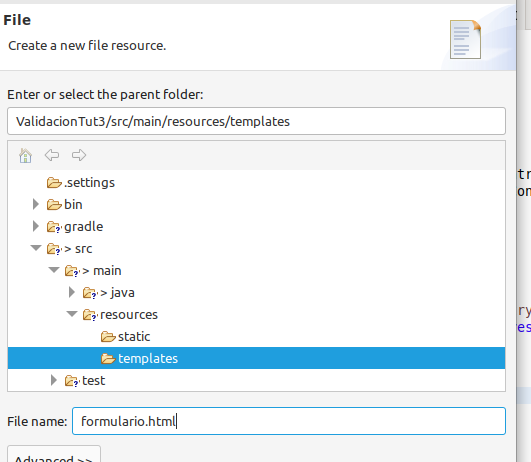
El método showForm devuelva la plantilla formulario, que recibe un objeto PersonaForm para que el html pueda asociar los atributos, y en el método checkPersonaInfo vemos cosas nuevas, acepta estos dos argumentos:

* personForm: objeto marcado con **@Valid** para recoger los atributos rellenados en el formulario.
* **bindingResult**: objeto para que pueda probar y recuperar errores de validación.

Si hay errores, devuelve nuevamente la plantilla del formulario, mostrando los atributos de error, pero si los atributos son válidos, nos redirige a la plantilla resultados con redirect:/.

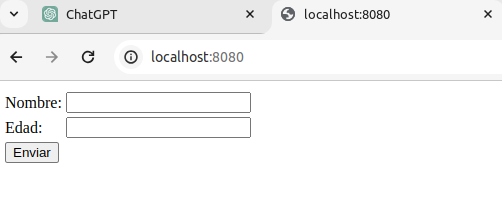
* Creo **VISTA** , plantilla hml formulario.html, copio del tutorial adaptando los nombres a mis atributos:

Observamos que los mensajes de error, estarán en fileds.hasErrors(---).





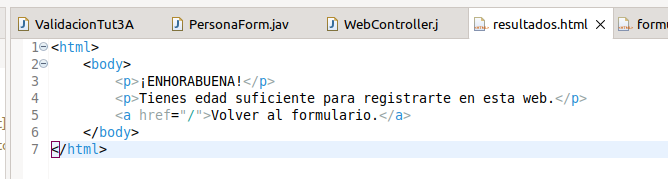
Aunque nos falte hacer la plantilla resultados, vamos a ejecutarlo en el navegador para ir viendo que funciona:



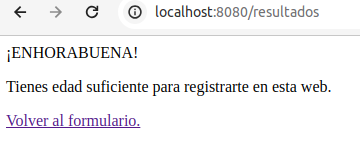
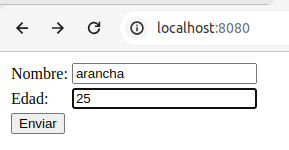
Y al enviar los datos, obviamente tenemos error al no existir resultados:



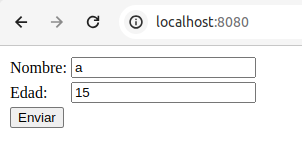
* Creamos **VISTA** html resultados (también en la carpeta templates). Creo también un enlace para volver al formulario, volver a atrás:



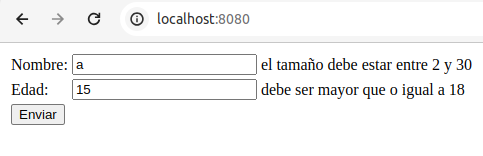
Y ejecutamos en el navegador, vemos que se redirige correctamente a la página resultados, con el contenido de esa plantilla:

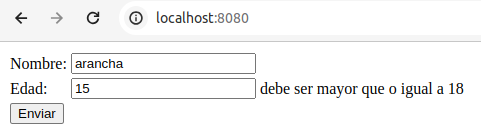


Ahora voy a provocar errores. En el campo nombre voy a poner una sola letra (el mínimo son 2 caracteres) y en edad voy a poner 15 (el mínimo es 18):



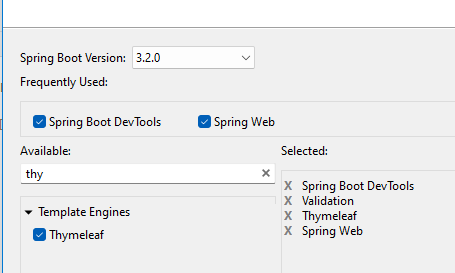
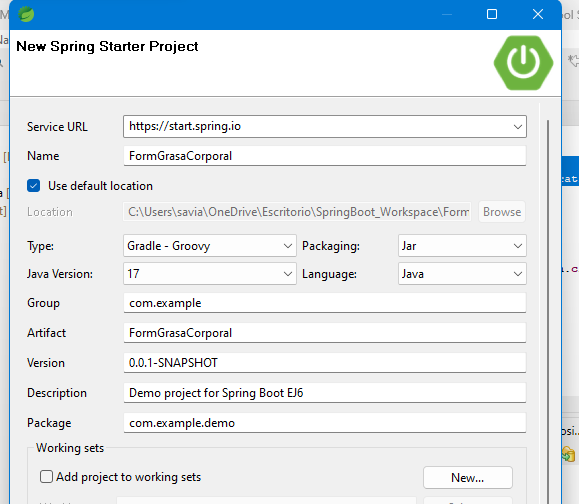
Podemos ver los mensajes de error. Estos mensajes son estándar y están ya creados, predefinidos, pues estos mensajes no los hemos creado en ninguna parte de nuestro código ni en las plantillas html. Muy curioso que además salgan en español:

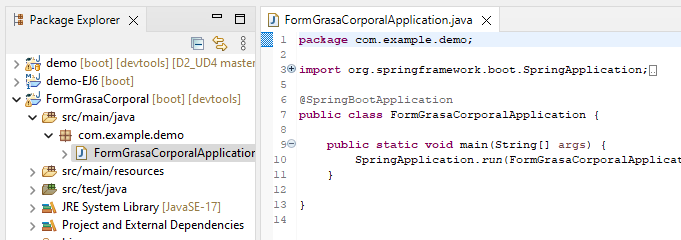


Probamos poniendo solo la edad errónea:

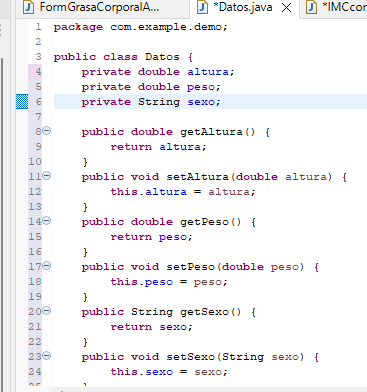
**APARTADO 2: aplicación con formulario de entrada de datos que permita calcular nuestro índice de grasa corporal, a partir del valor proporcionado nos tendrá que indicar que clasificación nos encontramos.**

* Creo pryecto spring con las dependencias vistas anteriormente:





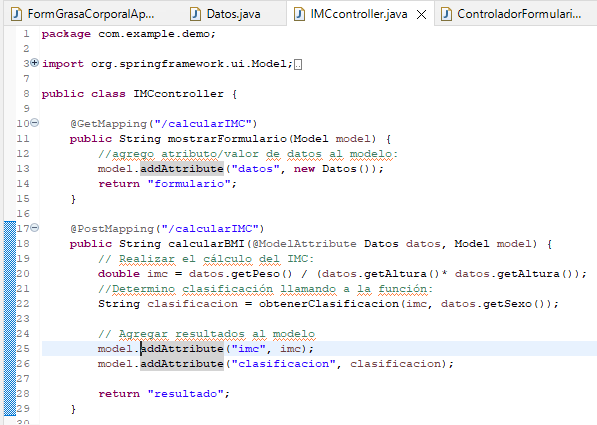
* **Ahora creo clase objeto/modelo** llamada datos que represente los datos del formulario, con los atributos de altura, peso y sexo y sus getters y setters correspondientes:



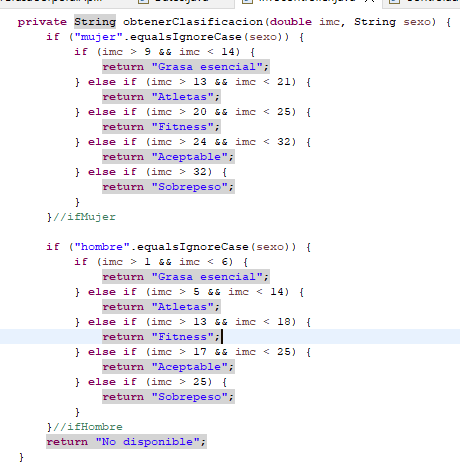
* **Creo clase controlador** para que maneje las solicitudes del formulario y los cálculos, llamada IMCcontroller, y ahí creo las anotaciones correspondientes para hacer los cálculos.

Fijándome en los tutoriales anteriores, voy a crear una anotación GetMapping que devuelva el formulario y una anotación PostMapping que devuelva el resultado, ambas siendo plantillas html.

* @GetMapping(/calcularIMC): recibe un objeto tipo Model, en este caso será datos. Y con model.addAttribute, agrega el valor del atributo datos de tipo objeto Datos al modelo.
* @PostMapping(/calcularIMC): recibe dos parámetros:
  + @ModelAttribute Datos datos: indica que se debe asociar los datos del formulario a un objeto de tipo Datos.
  + Model model: objeto Model para pasar datos entre controlador y vista.
  + Double imc: creo elemento donde hago el cálculo, obteniendo la altura y peso con su método get.
  + String clasificación: creo elemento donde llamo a la función obtenerClasificación, que recibe los parámetros imc y sexo, necesarios en dicha función y se obtendrá el valor return correspondiente.
  + Model.AddAttribute: esto sirve para agreagar el dato imc o clasificación al modelo con el mismo nombre, para que la vista (formulario) acceda a este valor.

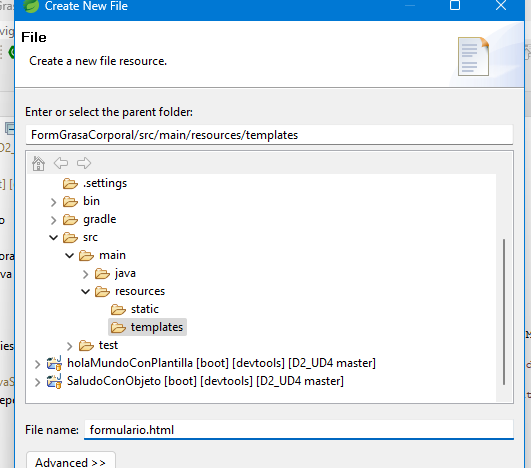


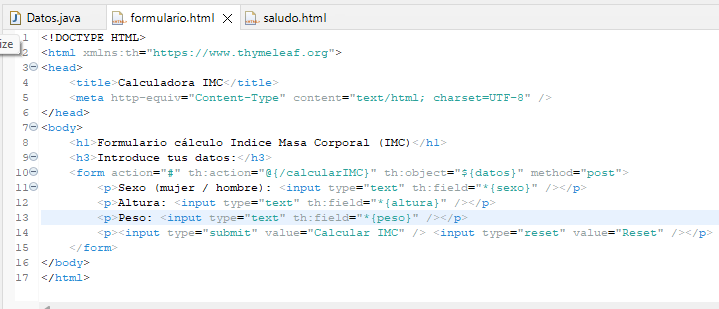
* Función obtenerClasificacion: recibe parámetros imc y sexo, primero compruebo con if si el sexo es mujer u hombre, con equalsIgnoreCase para que no diferencia entre mayúsculas y minúsculas, ya que cada sexo tiene una clasificación diferente, que dentro de su if voy comparando con más ifs, y según los datos devuelve un mensaje/clasificación:



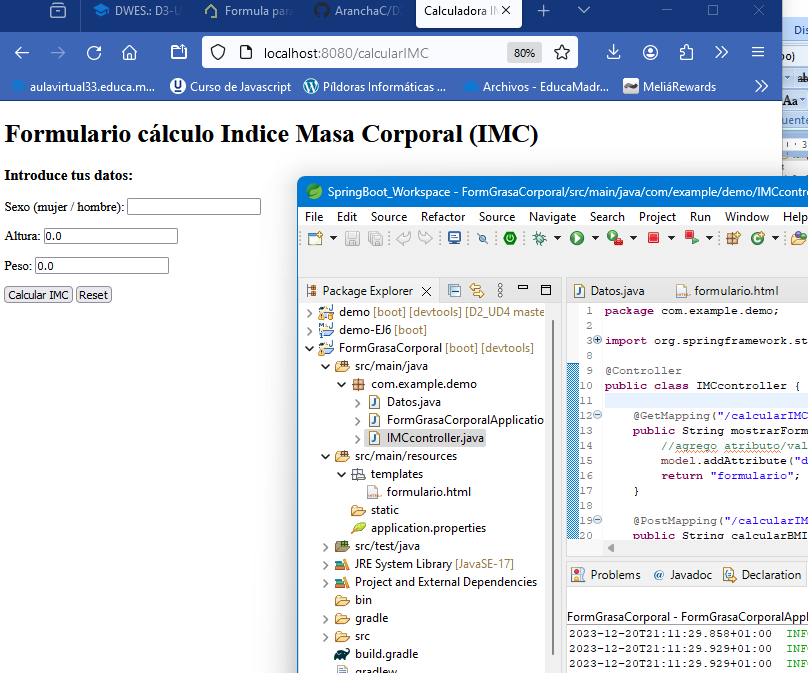
* **Ahora creo las plantillas HTML**, correspondiente a **VISTA,** en carpeta templates.

Primero formulario que devuelve el GetMapping de la clase controlado. Para que luego en controlador lea los datos, en los campos correspondientes, pongo el nombre del atributo de esta manera \*{nombre}:

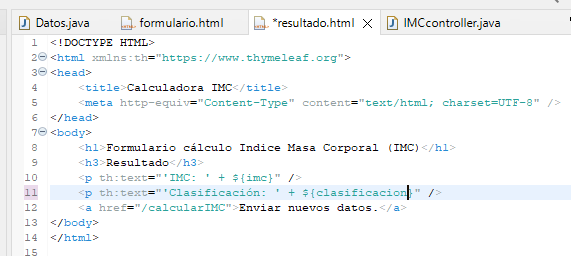




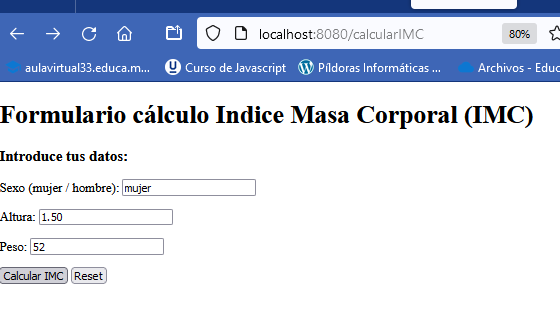
Pruebo a ejecutar la aplicación, no hay errores, y hago la prueba en el navegador y vemos que aparece el formulario que acabo de crear:



* Ahora **creo la plantila** **(VISTA)** resultado, que devuelve el PostMapping, recibiendo todos los datos introducidos en formulario:



Ejecuto y pruebo en el navegador introduciendo datos:



Y damos a calcular IMC, botón de enviar y todo funciona correctamente:



Damos al enlace para volver a enviar nuevos datos y probamos con otros:

