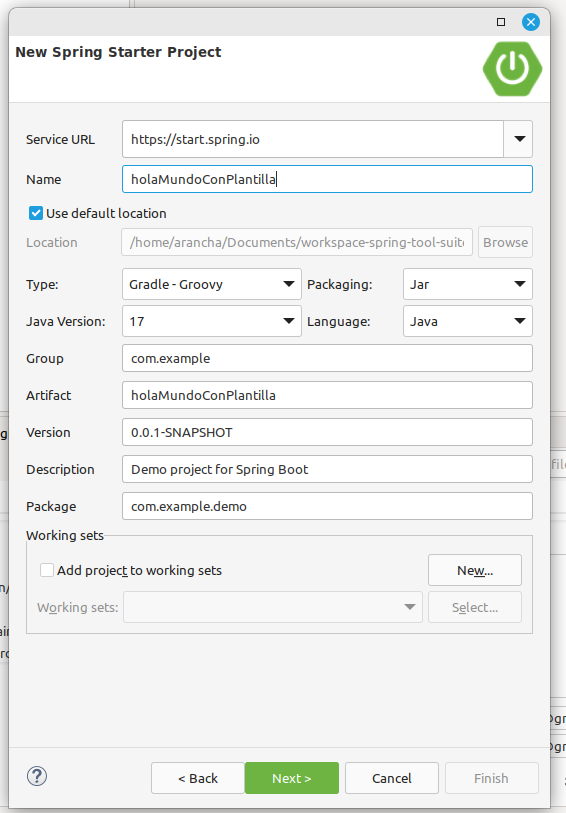
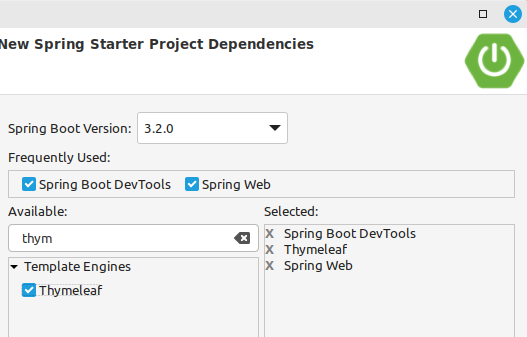
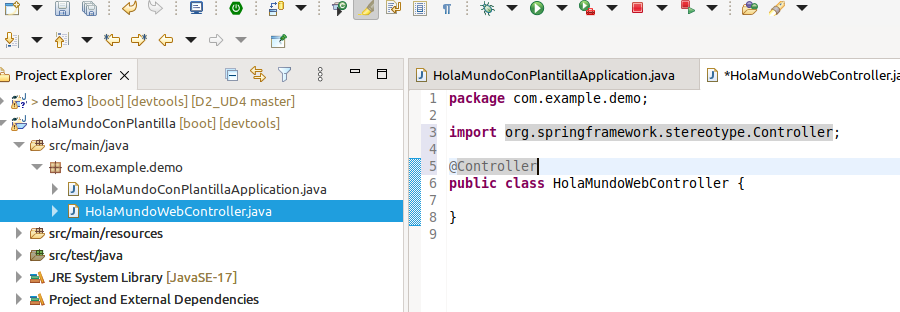
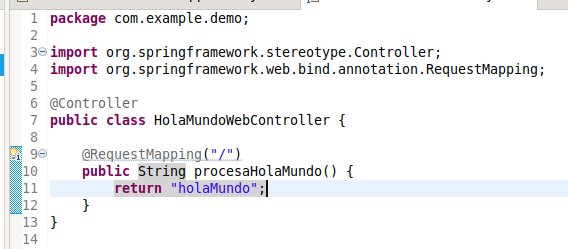
**Tarea 2 – UD4**

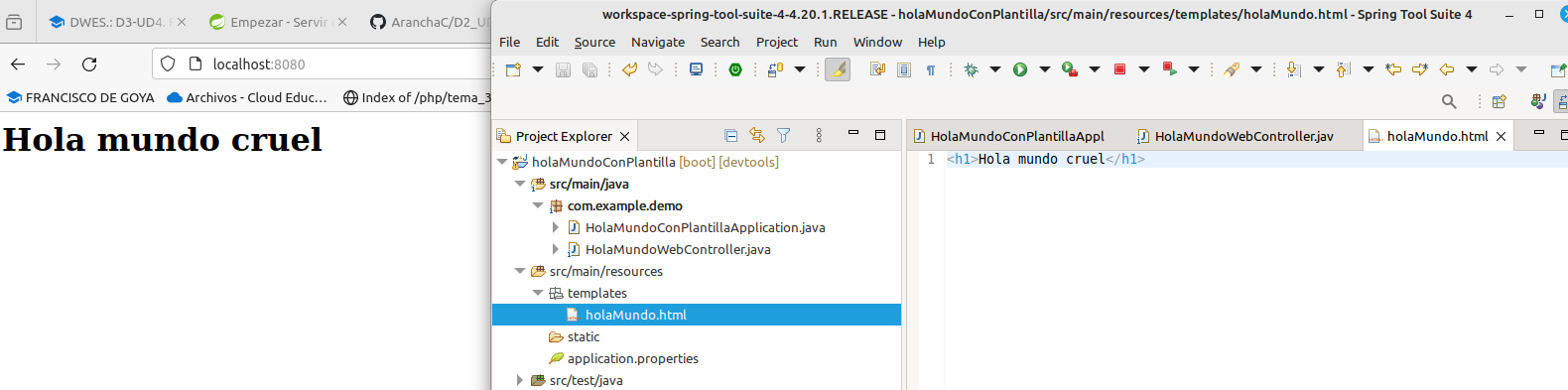
**Introducción al testing**





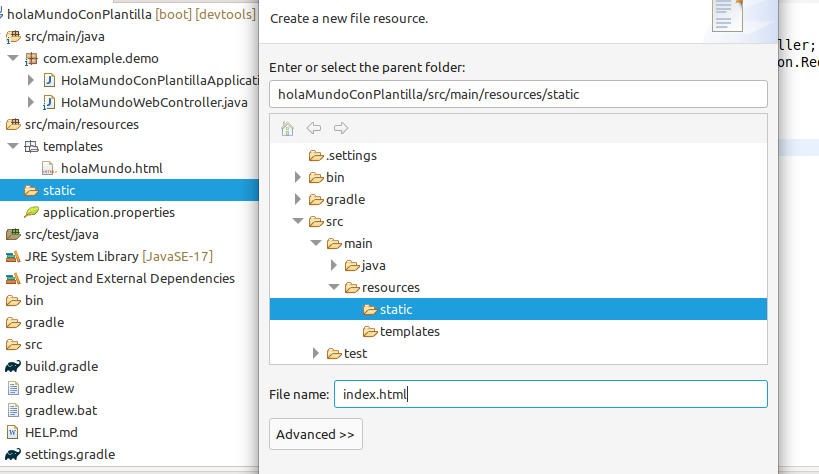


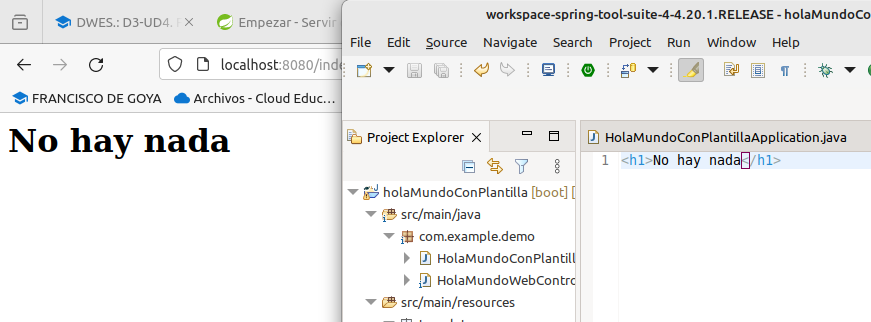


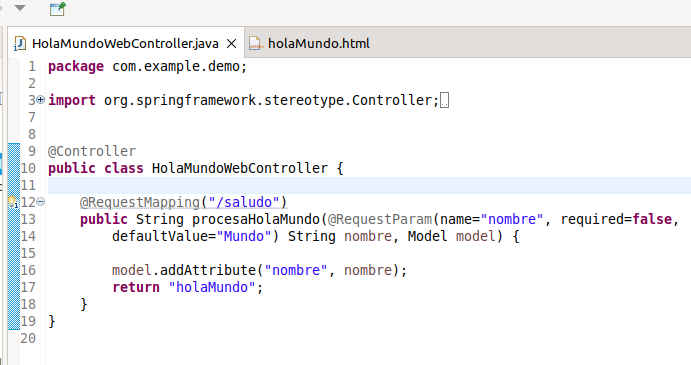


configurar hola mundo para que cuando le pasesmos un parámetro, diga hola mundo + el parámetro:

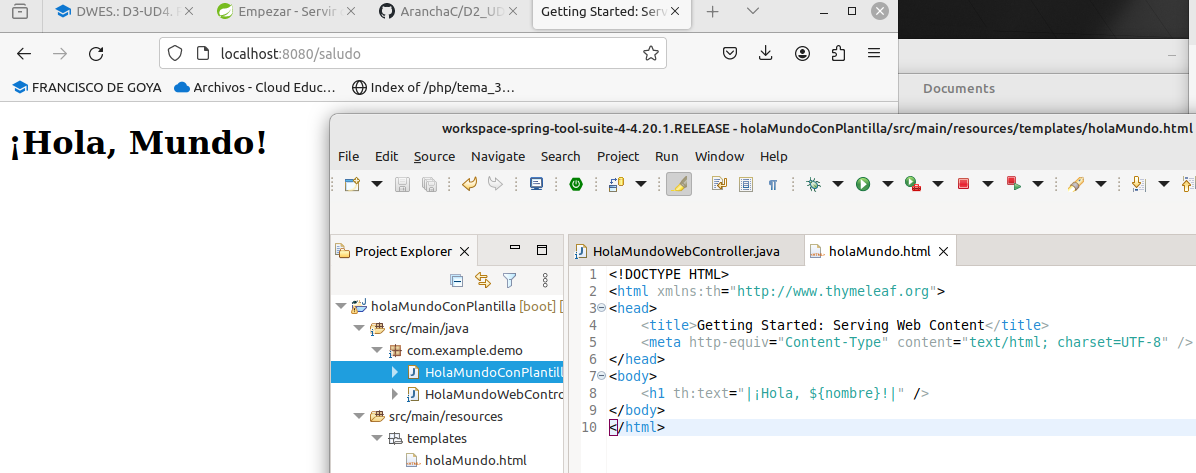
creo en static un index

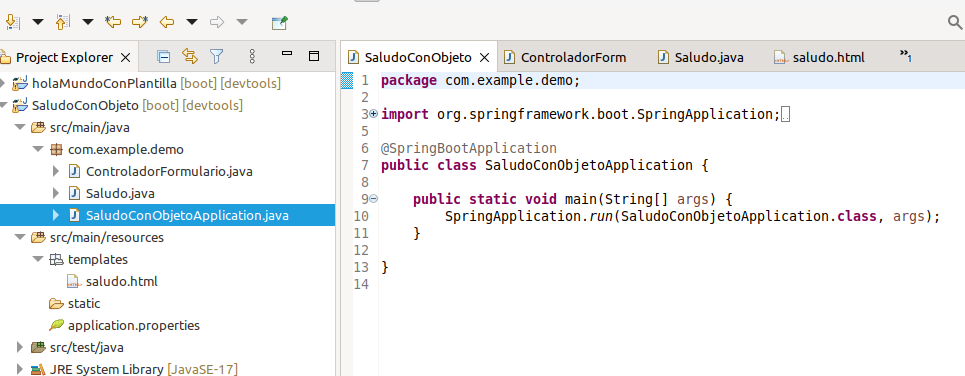


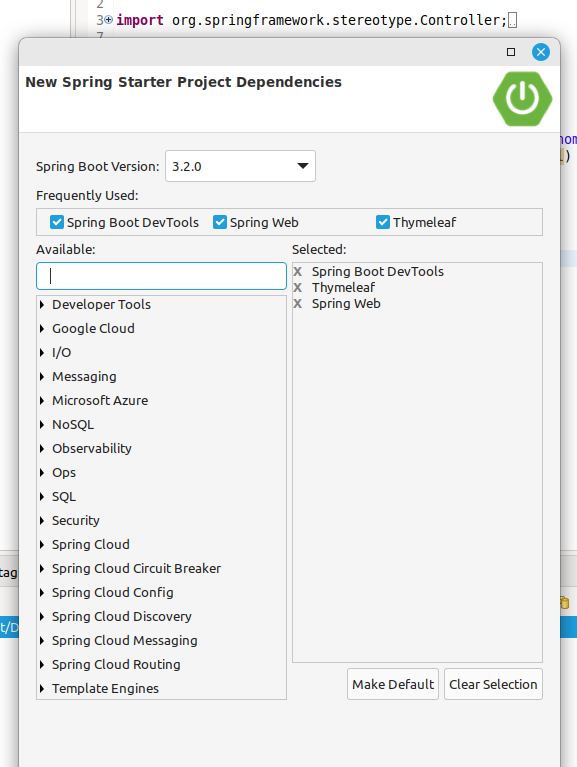


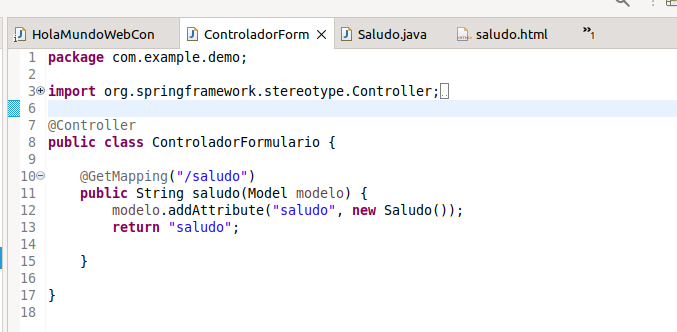


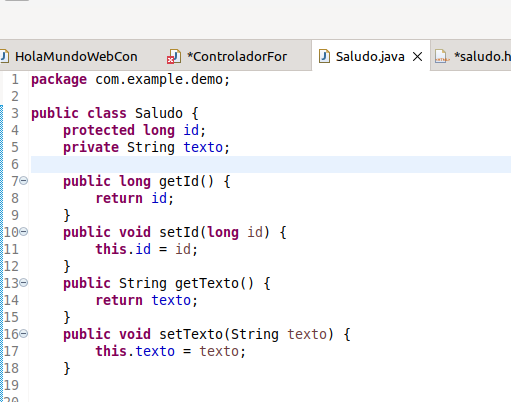
probamos sin parámetro:

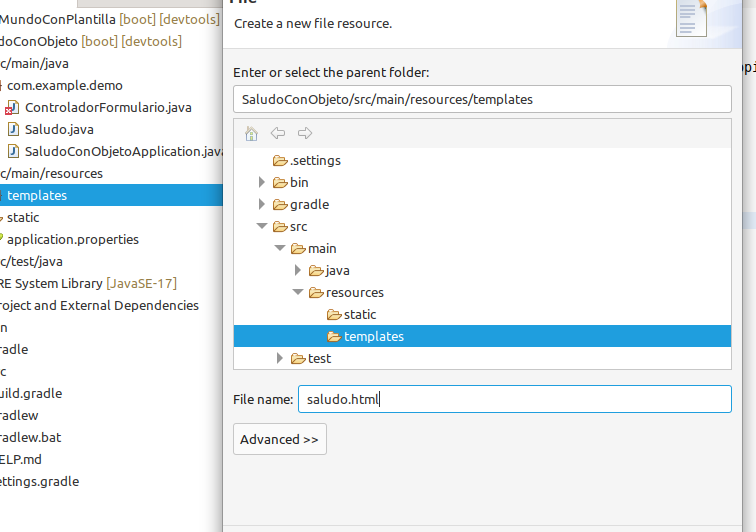
**TUTORIAL 2**

****



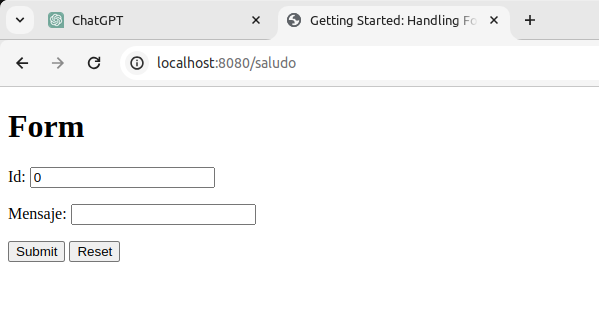


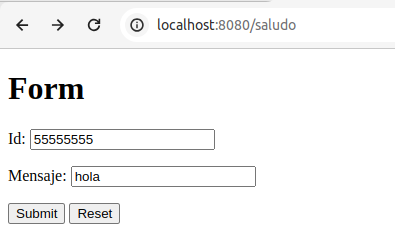


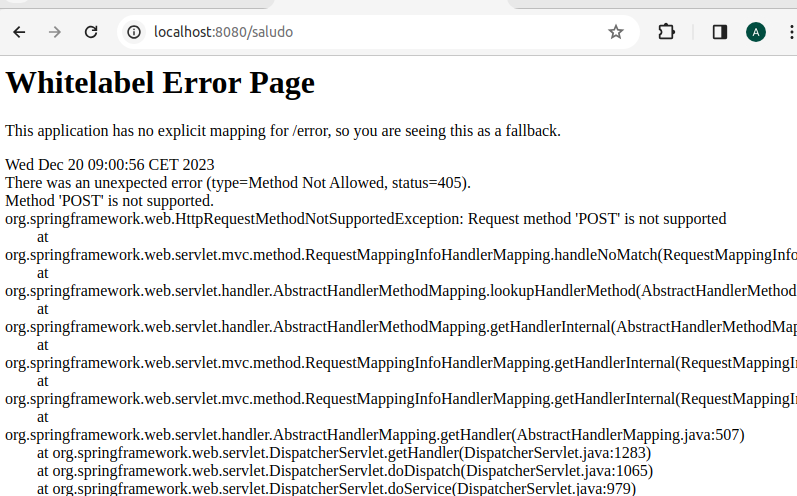
copio plantilla de html, adaptando los nombres a mis datos:



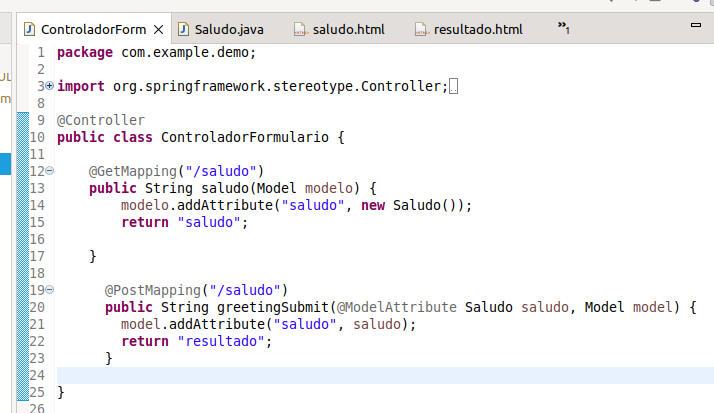
y ejecutamos en el navegador:

enviamos datos y da error:



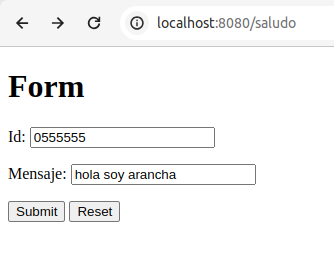


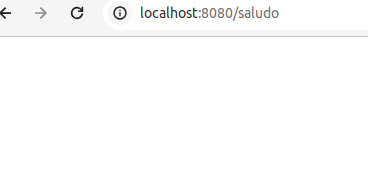
hago anotación con Post: Esta asignación permite al controller diferenciar las peticiones a la /saludo endpoint.



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 formario a /saludo por POST.

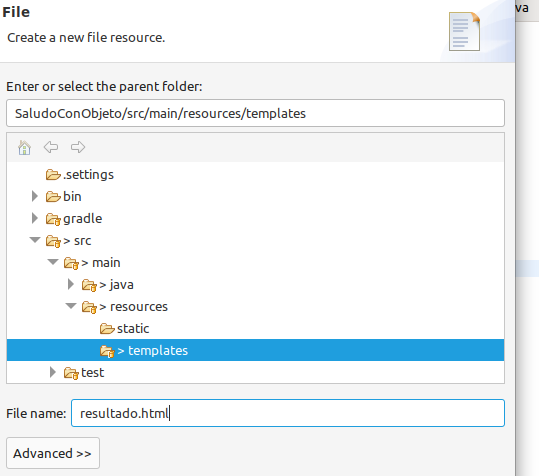
Envío los datos y no hay error, se envía el formulario aunque no aparece nada:

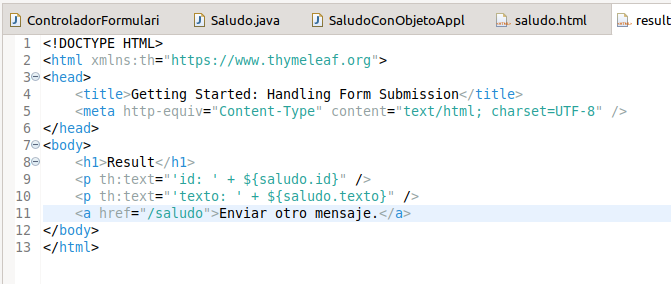


esto cubre el modelo, vista y controlador para presentar el formulario. Ahora podemos revisar el proceso de presentación del formulario. El formulario se somete a la /saludo punto final mediante el uso de una llamada POST.

el método saludo2()el método recibe el objeto Saludo. El Saludo es a @ModelAttribute, por lo tanto está unido al contenido de la forma entrante.

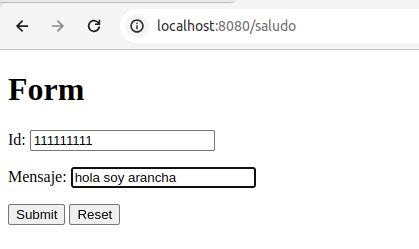
ahora copio plantilla de resultado:

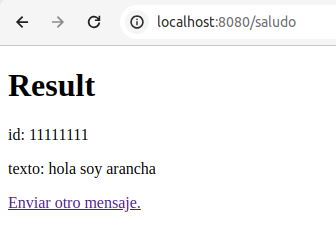




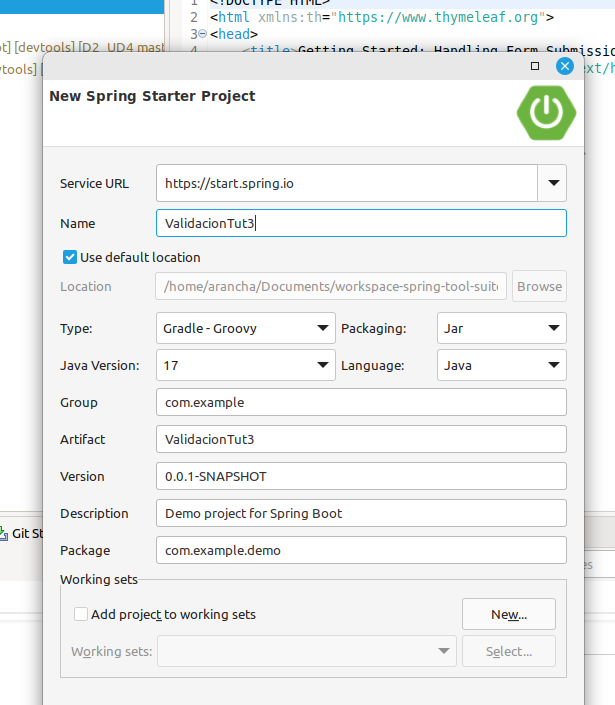
Podemos usar una sola plantilla para los dos propósitos, pero es más claro usar dos vistas separadas para representar el formulario y mostrar los daros.

Ejecutamos nuevamente en navagador:

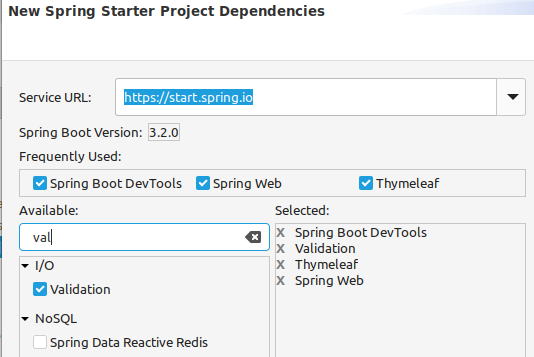


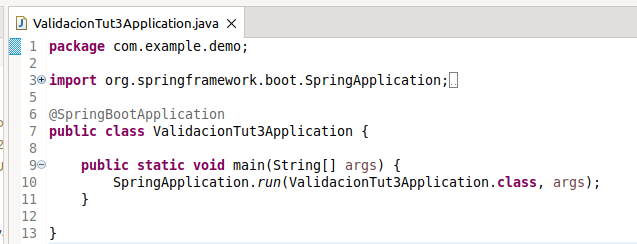
Envío datos y ahora aparece la página con los datos enviados, correspondiente a la plantilla html resultado:  


**TUTORIAL 3**

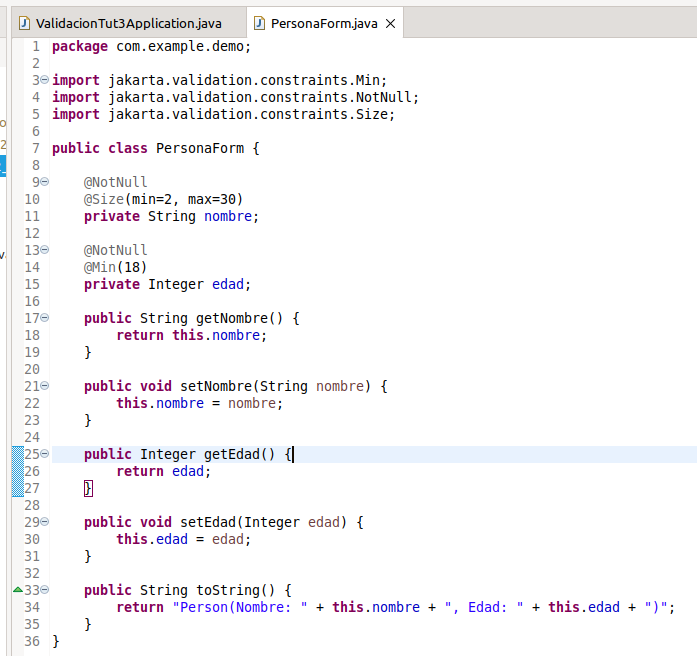


esta vez también ponemos la dependencia Validation:





Creamos objeto PersonaForm con atributos nombre y edad:



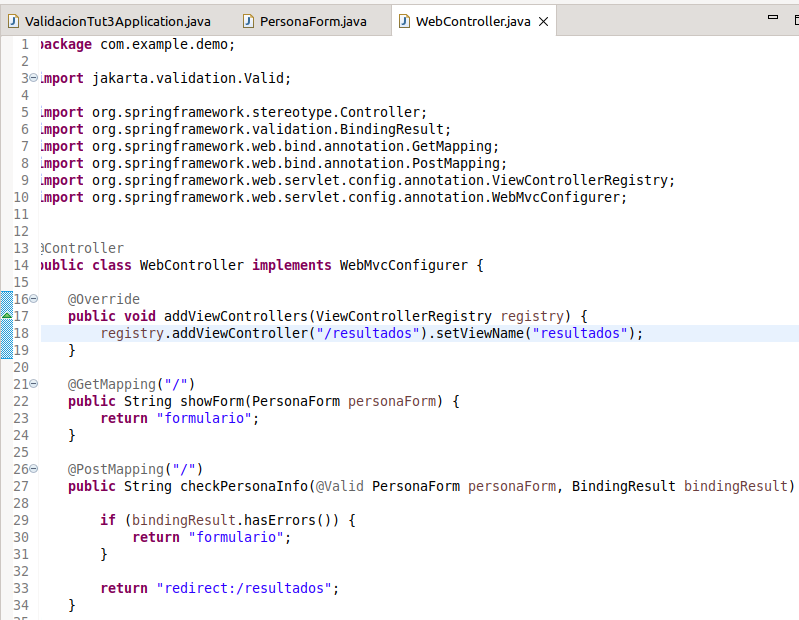
Describimos las siguientes anotaciones:

[-@NotNull](mailto:-@NotNull): hace que no permita valor nula, 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.

Creamos controlador:



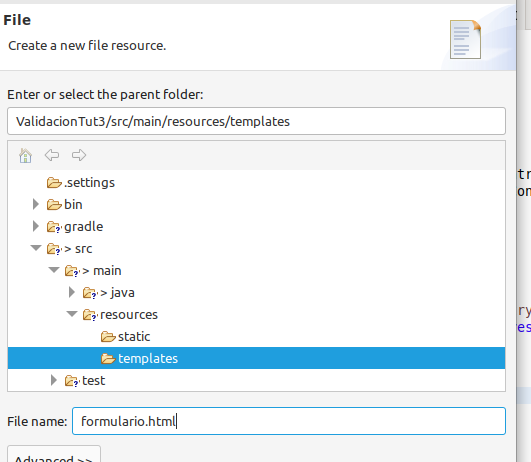
El método mostrarForm devuelva una plantilla formulario, que recibe un objeto PersonaForm para que ell html pueda asociar los atributos, y en el método checkPersonaInfo vemos cosas nuevas, acepta estos dos argumentos:

-personForm: objeto marcado con @Valid para recoger los atributos rellenados en el formulario.

-bindingResult: objeto para que pueda probar y recuperar errores de validación.

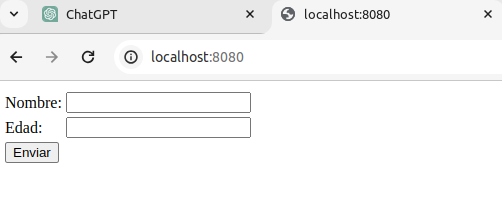
Si hay errores, devuelve nuevamente la plantilla del formulario, mostrando los atributos de error, pero si los atributos son válidos, nos redirije a la plantilla resultados.

Construimos plantila formulario:



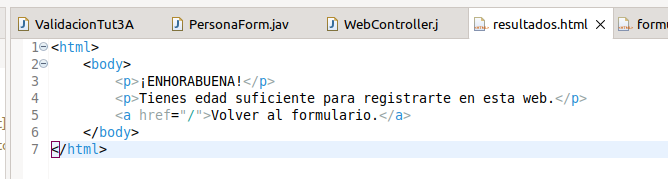
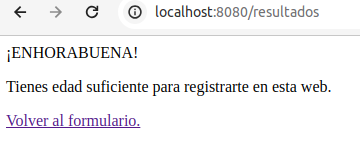


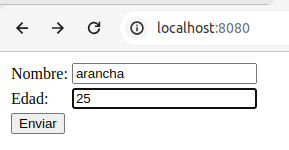
aunque nos falte hacer la plantilla resultados, vamos a ejecutarlo en el navegador para ir viendo que funciona:



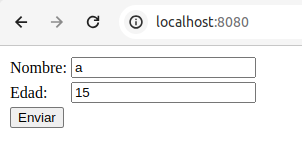
y al enviar los datos, obviamente tenemos error al no existir resultados:  


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

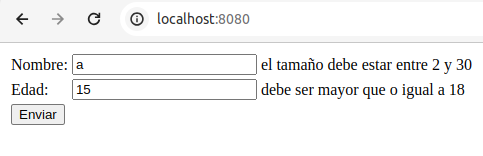
  
y ejecutamos en el navegador y vemos que se redirige correctamente a la página resultados, con el contenido de esa plantilla:

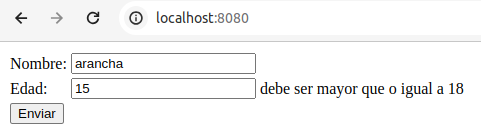


ahora voy a provocar errores. En el campo nombre voy a poner una sola letra (el mínimo son 2 caracteres) y en edad voy a poner 15 (el mínimo es 18):



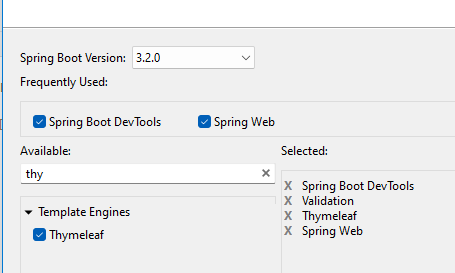
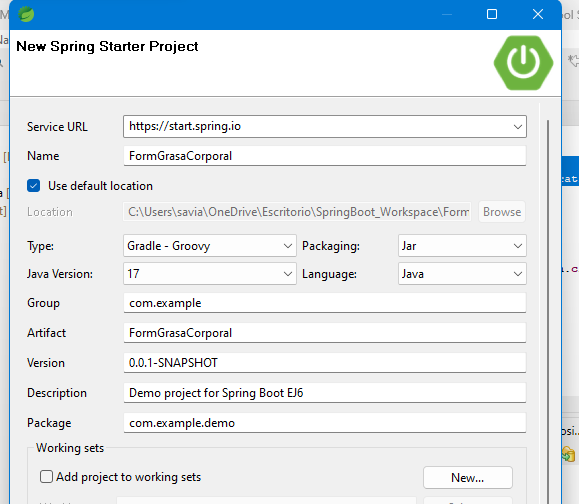
podemos ver los mensajes de error. Estos mensajes son estándars y están ya creados, predefinidos, pues estos mensajes no los hemos creado en ninguna parte de nuestro código ni en las plantillas html. Muy curioso que además salgan en español

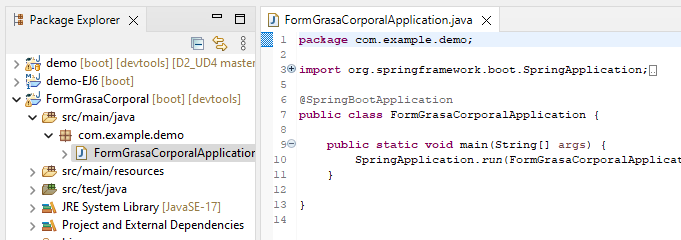


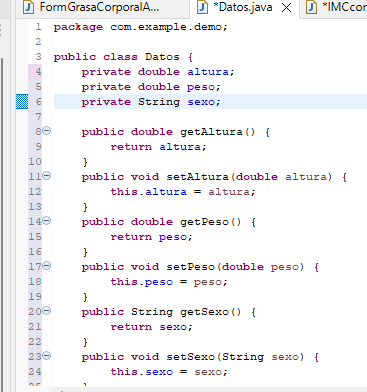
probamos poniendo solo la edad errónea:

**APARTADO 3:**

Creo pryecto spring con las dependencias vistas anteriormente:



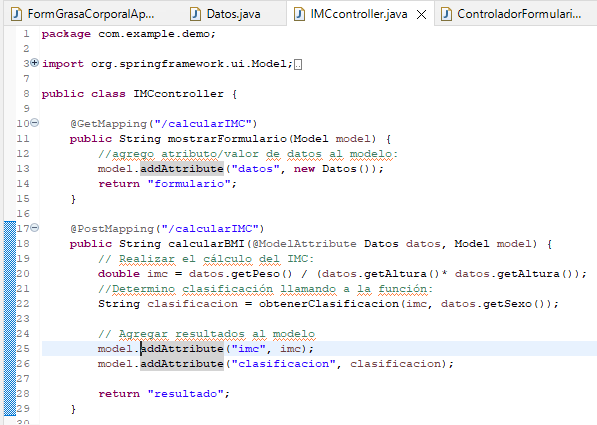


**Ahora creo clase objeto/modelo** llamada dato que represente los datos del formulario, con los atributos de altura, peso y sexo y sus getters y setters correspondientes:  


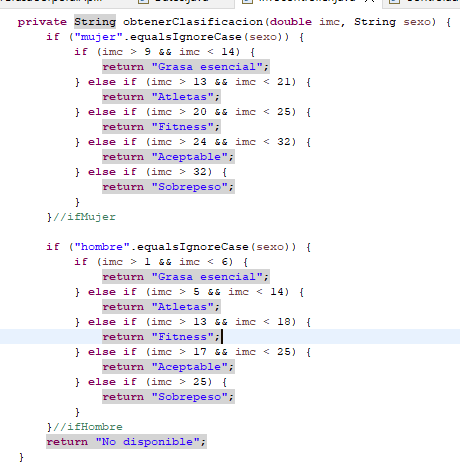
**Creo clase controlador** para que maneje las solicitudes del formulario y los cálculos, llamada IMCcontroller, y ahí creo las anotaciones correspondientes para hacer los cálculos.

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.

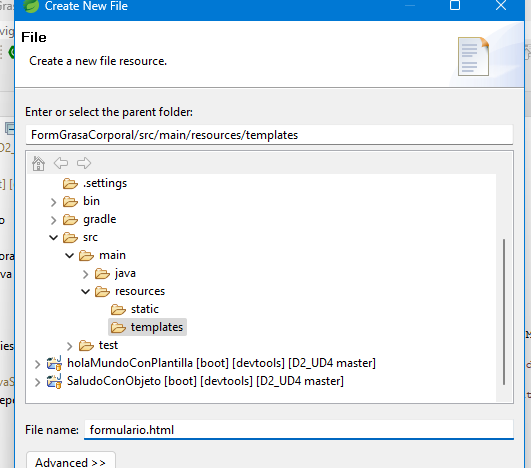


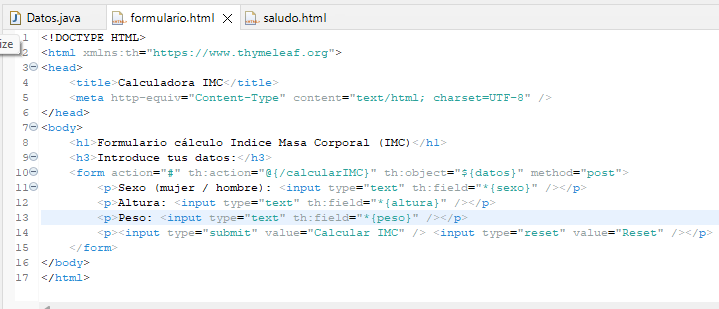
* Función obtenerClasificacion: recibe parámetros imc y sexo, primero compruebo con if si el sexo es mujer u hombre, ya que cada sexo tiene una clasificación diferente, que dentro de su if voy comparando con más ifs, y según los datos devuelve un mensaje/clasificación:

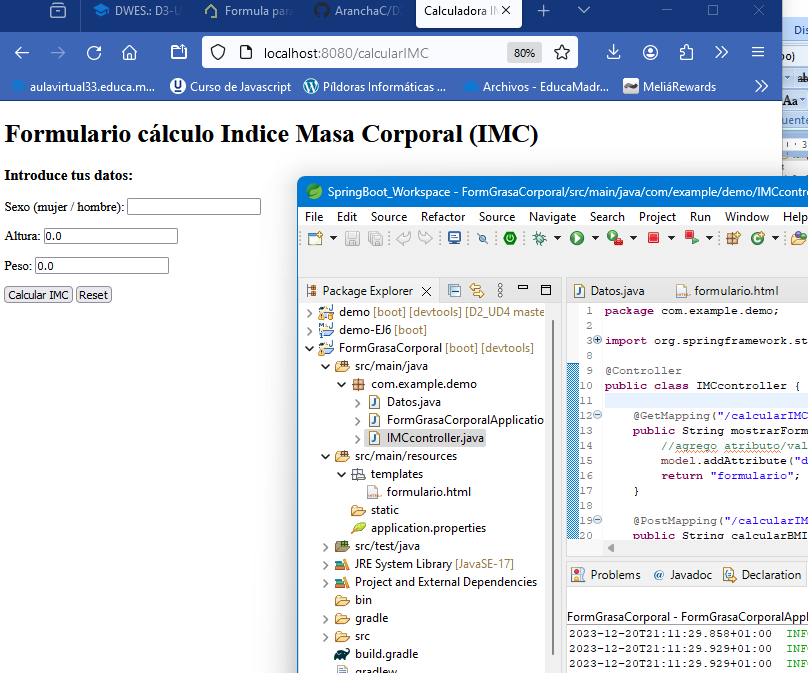


**Ahora creo las plantillas HTML**, correspondiente a **VISTA,** en carpeta templates.

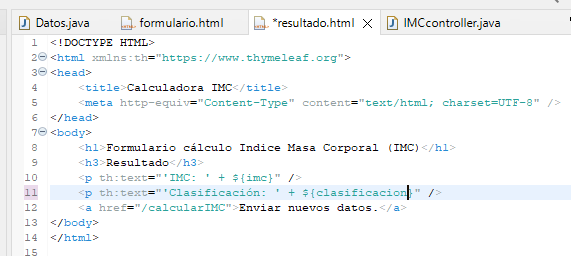
Primero formulario que devuelve el GetMapping de la clase controlado. Para que luego en controlador lea los datos, en los campos correspondientes, pongo el nombre del atributo de esta manera \*{nombre}:



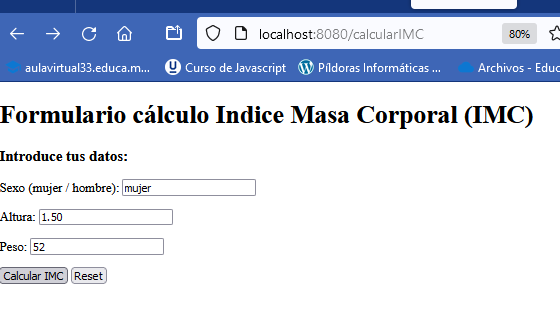
  
Pruebo a ejecutar la aplicación, no hay errores, y hago la prueba en el navegador y vemos que aparece el formulario que acabo de crear:



Ahora creo la plantila 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:  
