**Tarea 4 – UD4**

**QUIZ**

**DESCRIPCIÓN:**

Desarrollar un Quiz en el que a través de una serie de preguntas de diferente modalidad, se determine una determinada clasificación en su temática.

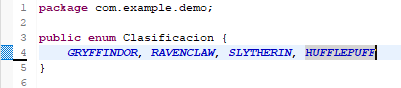
El objetivo es acumular puntos según las respuestas seleccionadas y determinar una clasificación al final del quiz.

**Temática elegida: HARRY POTTER, ¿A qué casa de Hogwarts perteneces?**

**Requisitos para el quiz:**

**1º Cuatro clasificaciones:** el resultado del juego debe dar una clasificación entre **GRYFFINDOR**, **SLYTHERIN, RAVENCLAW y HUFFLEPUFF.**

**Esta la clasificación la he creado en una clase enum:**

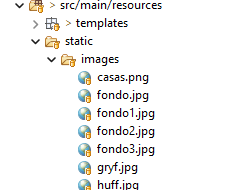


Cada pregunta supondrá una puntuación añadida para cada marcador de la clasificación. El resultado final será la clasificación con mayores puntos en el marcador.

**2º Uso de la SESSION:** Al usuario se le plantarán diversas preguntas en las que se deberá almacenar la puntuación obtenida. Para almacenar el resultado de las preguntas entre cada página se verá hacer uso de las sesiones, **Necesario para la suma de puntos, usando HttpSession.** Se recomienda simplificar la aplicación y el conteo de estas puntuaciones para que el groso de las horas invertidas no recaiga en la lógica de negocio, **para ello he hecho uso de varios métodos.**

**3º preguntas:** Se deberá hacer como mínimo 7 preguntas al usuario empleando cada una de estas entradas al menos una vez: textArea, CheckBox, RadioButton, Button, Select (desplegable.)

**4º Resultado final**: Habiendo calculado el resultado, para emitir al veredicto se le deberá establecer una categoría al usuario empleando una imagen para dicha categoría. **Para ello he creado una carpeta images dentro de static en src/main/resources, donde he almacenado las imágenes correspondientes a cada categoría y otras imágenes usadas en las plantillas.**

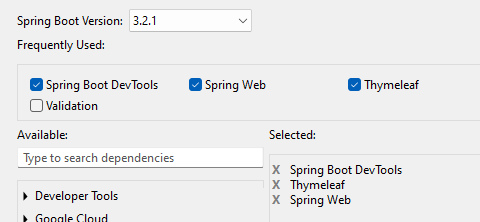


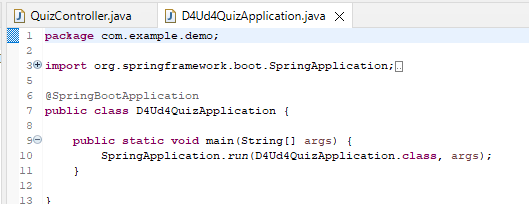
**5º Hojas de estilo:** No se entiende una página desarrollada en HTML sin su CSS, por ello deberás incorporar estilos a los dos anteriores ejercicios. **En cada plantilla html, he creado una hoja de estilos css dentro de las etiquetas <style>.**

**DESARROLLO Y ORGANIZACIÓN:**

**- Crear proyecto Spring Boot con dependencias:**

Primero se crea el proyecto Spring Boot al que llamamos D4\_UD4\_QUIZ, con las dependencias Dev Tools, Spring Web y Thymeleaf:



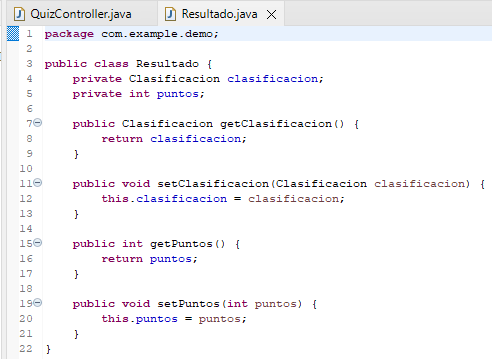


**- Crear clase Modelo:**

Ahora creo la clase del Modelo, me planteo cuál puede ser el objeto de esta aplicación y pienso en la lógica del quiz y creo que el objeto será Resultado, que es lo que vamos a mostrar al final y los datos se irán transfiriendo entre el quiz.

Creo clase Resultado con los atributos Clasificación clasificaión (del tipo enum ya creado) y int puntos, pues este objeto resultado se compone de la clasificación y de los puntos.

Genero también los getters y setters:



**- Crear clase Controlador:**

Creo clase QuizController donde manejar las solicitudes http del formulario/quiz y gestionar las sesiones.

El planteamiento es, al ejecutar la aplicación se abrirá una plantilla a modo de inicio del quiz con un botón de siguiente que nos redirigirá a la primera pregunta, y ésta a la segunda pregunta, y así sucesivamente hasta llegar a la última pregunta que nos llevará a una página final donde mostramos el resultado.

Según esta lógica, debe haber una plantilla por pregunta además de la de inicio y final, y por lo tanto un método que devuelva cada una de estas plantillas. Este método debe ser Post porque maneja datos del servidor a través de un formulario y usando sesiones.

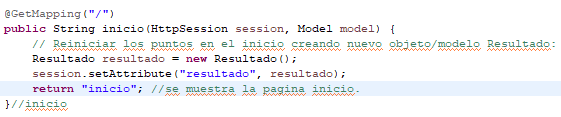
**Método inicio @GetMapping(“/”):**

Creo método inicio y como su función es mostrar la página de inicio del quiz y no envía datos al servidor, usamos getMapping y no postMapping, pues las solicitudes al cargar una página suelen ser get.

Pasamos por parámetro HttpSession y Model, pues aquí creamos un objeto Resultado y con sesión.setAttribute, lo almacenamos en la sesión. De esta manera cada vez que vayamos a la página de inicio, el objeto resultado se creará de nuevo y así se reiniciarán los puntos.

Este objeto Resultado lo usaremos más adelante para ir actualizando los puntos en cada pregunta.

Y devuelve plantilla inicio:

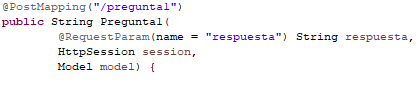


**Método pregunta1 @GetMapping (“pregunta1”):**

Este método simplemente devuelve la plantilla pregunta1. Ha sido necesario crearlo porque el método de su página anterior devuelve la página de inicio.

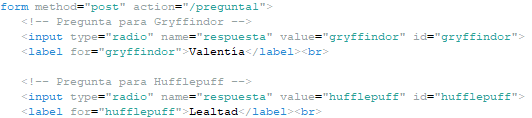


**Método pregunta1 @PostMapping (“pregunta1”): tipo de pregunta radio button con name=respuesta:**



Parámetros usados:

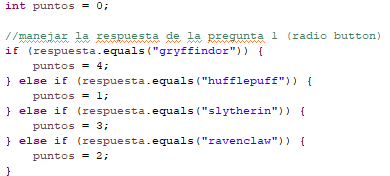
* **@RequestParam(name = "respuesta") String respuesta**: lo usamos para obtener la respuesta seleccionada. En la plantilla debe de haber un atributo name=respuesta.



* **HttpSession session**: lo usaremos en todos los métodos para gestionar los datos que almacenamos en la sesión, en este caso accedemos para ir actualizando los puntos.
* **Model model**: también lo usaremos en todos los métodos para agregar atributos al modelo, en este caso los puntos.

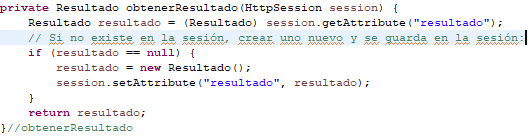
Creamos variable int para almacenar los puntos de esta pregunta.

Y la lógica de esta pregunta de tipo radio button, es que si la respuesta es igual a x, se dan unos puntos. Como respuesta es de tipo String, se usa función equals.

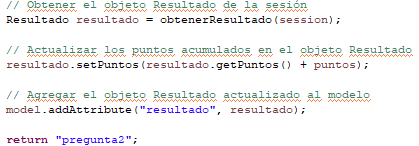


Luego obtenemos el objeto Resultado de la sesión. Para ello he creado un método llamado obtenerResultado donde paso por parámetro la sesssion.

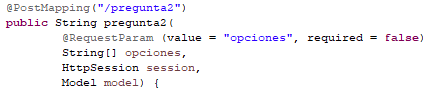
Este método accede y devuelve el objeto Resultado almacenado en la sesión, que si no existe, si es nulo, crea uno nuevo y se almacena.



Y una vez tenemos el objeto Resultado, con SetPuntos, actualizamos los puntos de esta pregunta, sumando los puntos de esta pregunta a los ya almacenados (con GetPuntos).

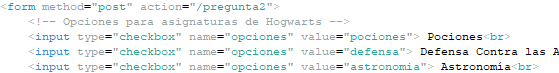
Y por último, devuelve la pregunta2, es decir, la siguiente pregunta:  


**Método pregunta2 @PostMapping (“pregunta2”): tipo de pregunta checkbox con value=opciones:**



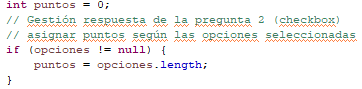
Parámetros usados:

* **@RequestParam(value = "opciones", required = false) String[] opciones**: para obtener las opciones seleccionadas por el usuario en la pregunta 2. Al ser checkbox multirespuesta, será de tipo array llamado opciones y sus valores serán los value de cada checkbox:



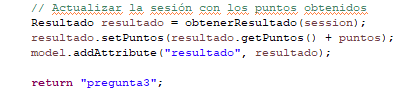
* **HttpSession sesión**.
* **Model model**.

Creamos variable puntos y la lógica de esta pregunta es que tendrás puntos acorde con el número de respuestas seleccionadas, por lo tanto al ser un array, usamos la función .length para asignar el valor a los puntos. Usamos la condición de que el array no sea nulo.

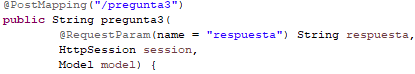


El resto de líneas es igual que la pregunta1, actualizar el Resultado de la sesión sumando los puntos obtenidos en esta pregunta.

Y por último devuelve la pregunta3:

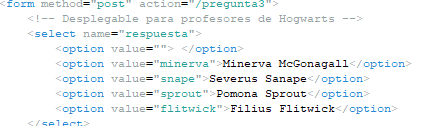


**Método pregunta3 @PostMapping (“pregunta3”): tipo de pregunta select/option con name=respuesta:**



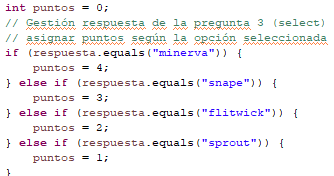
Parámetros usados, mismos al pregunta1:

* **@RequestParam(name = "respuesta") String respuesta**: para obtener la respuesta seleccionada. En la plantilla el select debe tener el atributo name=respuesta.



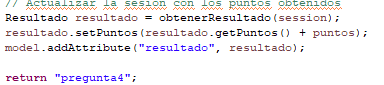
* **HttpSession sesión**.
* **Model model**.

La lógica de puntuación de esta pregunta es igual que la pregunta1, comparamos la respuesta con la función equals al value de cada option, y según sea la respuesta, damos x puntos.

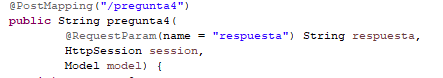


El resto de líneas es igual que los métodos anteriores, actualizar el Resultado de la sesión sumando los puntos obtenidos en esta pregunta.

Y por último devuelve la pregunta4:



**Método pregunta4 @PostMapping (“pregunta4”): tipo de pregunta button con name=respuesta:**



Parámetros usados, mismos al pregunta1 y 3:

* **@RequestParam(name = "respuesta") String respuesta**: para obtener la respuesta seleccionada. En la plantilla cada button debe tener el atributo name=respuesta.

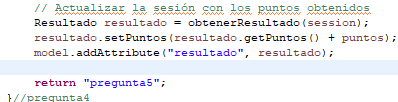


* **HttpSession sesión**.
* **Model model**.

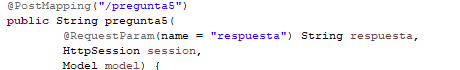
La lógica de puntuación de esta pregunta es igual que la anterior, según la respuesta, según el botón seleccionado, se dan x puntos. En este caso he usado una estructura switch que recibe respuesta, y en cada case el value de cada button.

Y las últimas líneas igual que los métodos anteriores, actualizar puntos en el Resultado de la sesión.

Y devuelve la pregunta5:



**Método pregunta5 @PostMapping (“pregunta5”): tipo de pregunta textarea con name=respuesta:**



Parámetros usados, mismos al pregunta1 , 3 y 4:

* **@RequestParam(name = "respuesta") String respuesta**: para obtener la respuesta seleccionada. En la plantilla el textarea debe tener el atributo name=respuesta.

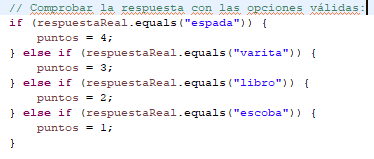


* **HttpSession sesión**.
* **Model model**.

La lógica de puntuación es que según el texto introducido en el textarea, damos x puntos. Este texto lo tenemos que comparar con los valores correspondientes.

Para ello creamos variable de tipo String donde almacenamos este texto de que cogemos de respuesta, al que pasamos a minúculas con la función .toLowerCase() y quitamos espacios en blanco con la función .trim(). Así nos aseguramos lo más posible de que el texto introducido sea válido.

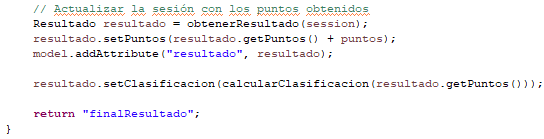
Y la comparación la hacemos con .equals:



Las siguientes líneas iguales que los anteriores métodos: actualizar el objeto resultado de la sesión sumando los puntos de esta pregunta.

Pero hay un añadido, al ser la última pregunta, también actualizamos la clasificación. Para ellos, llamamos al método calcularClasificación, que hemos creado para determinar la clasificación según los puntos obtenidos, guardados en el objeto Resultado de la sesión. Por lo que al llamar a la función, pasamos por parámetro los puntos de resultado con resultado.getPuntos.

Y por último devuelve la página finalResultado:



**Método pregunta6 @PostMapping (“pregunta6”): tipo de pregunta radio button con name=respuesta:**

En esta pregunta repito los tipos de campos del formulario de la pregunta 1, es decir, también son radio buton por lo que la gestión de la lógica de puntuación es la misma. Accedemos al value que es de tipo de string, y con equals se compara con un texto X y se dan unos determinados puntos.

El resto de código es igual que el resto

**- Crear plantillas (vista):**

**CONCLUSIONES:**

**Problemas / anotaciones:**

El planteamiento incial era crear por cada pregunta un método postMapping que devuelve la página siguiente, pero como la página anterior a la pregunta1, devolvía la página de inicio, me costó llegar a la conclusión de que tenía que crear un método getMapping para la propia pregunta1 que devolviera su plantilla correspondiente.

- Me daba cuenta que los puntos se acumulaban cuando volvía a la página de inicio en vez de reiniciarse, para ello en el método inicio, tuve que crear el objeto Resultado y almacenarlo en la sesión. Para ello, recibe por parámetro HttpSession y Model.

- También tuve el problema de que los puntos no se acumulaban entre preguntas, si no que se actualizaban, se sustituían con los puntos de la pregunta actual.

Lo que estaba haciendo era almacenar en la sesión los puntos, en lugar del objeto Resultado, y en cada método de cada pregunta creaba un objeto Resultado nuevo.

int puntosActuales = obtenerPuntos(session);

session.setAttribute("puntos", puntosActuales + puntos);

Resultado resultado = new Resultado();

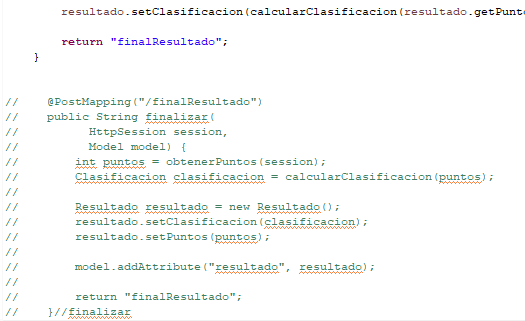
resultado.setPuntos(puntos);

model.addAttribute("resultado", resultado);

Los puntos si que los almacenaba en la sesión y los iba sumando a los puntos actuales llamando a una función obtenerPuntos, pero como los iba almacenando en el objeto nuevo en cada pregunta por este motivo nunca se iban a ir acumulando.

Para solucionarlo creé la función obtener resultado donde creo un objeto Resultado nuevo y lo guardo en la sesión, y en cada método de cada pregunta, accedo a ese objeto llamando a la función y ahí voy actualizando/sumando los puntos. Y por lo tanto, ya el método obtenerPuntos sobra y la eliminé.

Otro problema fue que no conseguía mostrar la última página, la de los resultados. Esto era porque había creado un método de sobra llamado finalResultado, que devolvía dicha página, al igual que el método de la última pregunta que también devolvía la misma página, por lo que se generaba un conflicto. Esto lo solucioné gracias al profesor, pues el error que salía no era nada claro y era difícil transcribirlo.



Otra anotación, es que en los métodos de cada pregunta la última parte del código que se repite, la de actualizar la sesión que se podría simplificar creando otra función.

Esta práctica ha sido muy formativa, desde el primer tutorial hasta el último ejercicio, he ido aprendiendo poco a poco, y avanzando en código y conocimientos. Ha sido muy interesante ir viendo esta mejoría y sobre todo el resultado en el navegador.

El resultado de todo el código junto, la unión de html/java, comprender el **modelo/vista/controlador**. Al hacer todas las partes de este reto, he avanzado mucho y aprendido sobre el código, volver al código Java y mejorar sobre él.

También la parte de validación a través de todas las anotaciones en la clase Modelo, y como relacionar los atributos del MODELO(clase objeto) con la VISTA (html) .

Después de hacer todas las pruebas y su correcto resultado, aún con problemas pero que he ido solucionando, me puedo sentir muy satisfecha y contenta con el resultado y con todo lo aprendido.