

## 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**.

La clasificación la he creado como atributo de tipo enum, en una clase a parte.

```
1 package com.example.demo;
2
3 public enum Clasificacion {
4     GRYFFINDOR, RAVENCLAW, SLYTHERIN, HUFFLEPUFF
5 }
6
```

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 – **1 pregunta**.

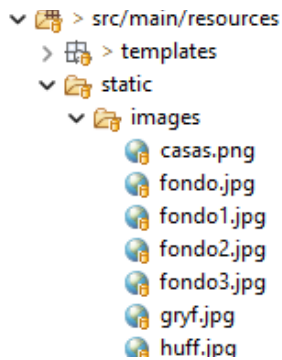
CheckBox – **1 pregunta**.

RadioButton – **2 preguntas**.

Button – **2 preguntas**

Select (desplegable.) – **1 pregunta**

**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.**

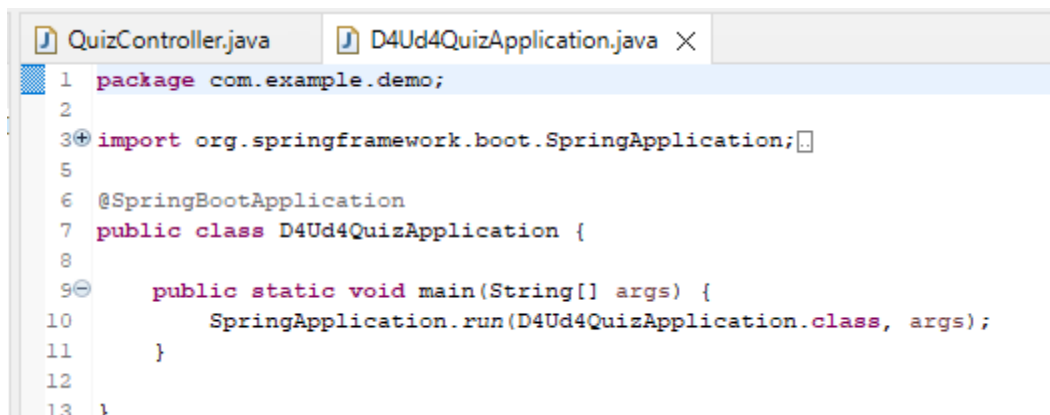
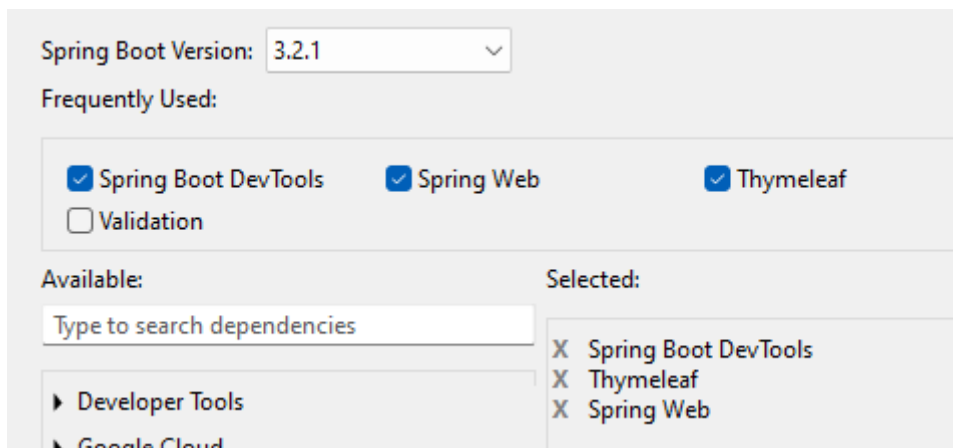


**5º Hojas de estilo:** No se entiende una página desarrollada en HTML sin su CSS, por ello deberás incorporar estilos a los dos anteriores ejercicios. **En cada plantilla html, he creado una hoja de estilos css dentro de las etiquetas <style>.**

## DESARROLLO Y ORGANIZACIÓN:

### 1. 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:



## 2. 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 clasificación (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:

```
1 package com.example.demo;
2
3 public class Resultado {
4     private Clasificacion clasificacion;
5     private int puntos;
6
7     public Clasificacion getClasificacion() {
8         return clasificacion;
9     }
10
11     public void setClasificacion(Clasificacion clasificacion) {
12         this.clasificacion = clasificacion;
13     }
14
15     public int getPuntos() {
16         return puntos;
17     }
18
19     public void setPuntos(int puntos) {
20         this.puntos = puntos;
21     }
22 }
```

## 3. 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:

```
@GetMapping("/")
public String inicio(HttpSession session, Model model) {
    // Reiniciar los puntos en el inicio creando nuevo objeto/modelo Resultado:
    Resultado resultado = new Resultado();
    session.setAttribute("resultado", resultado);
    return "inicio"; //se muestra la pagina inicio.
} //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.

```
@GetMapping("/pregunta1")
public String pregunta1() {
    return "pregunta1";
}
```

#### Método pregunta1 @PostMapping ("pregunta1"): tipo de pregunta radio button con name=respuesta:

```
@PostMapping("/pregunta1")
public String pregunta1(
    @RequestParam(name = "respuesta") String respuesta,
    HttpSession session,
    Model model) {
```

Parámetros usados:

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

```
form method="post" action="/pregunta1">
    <!-- Pregunta para Gryffindor -->
    <input type="radio" name="respuesta" value="gryffindor" id="gryffindor">
    <label for="gryffindor">Valentia</label><br>

    <!-- Pregunta para Hufflepuff -->
    <input type="radio" name="respuesta" value="hufflepuff" id="hufflepuff">
    <label for="hufflepuff">Lealtad</label><br>
```

- **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.

```
int puntos = 0;

//manejar la respuesta de la pregunta 1 (radio button)
if (respuesta.equals("gryffindor")) {
    puntos = 4;
} else if (respuesta.equals("hufflepuff")) {
    puntos = 1;
} else if (respuesta.equals("slytherin")) {
    puntos = 3;
} else if (respuesta.equals("ravenclaw")) {
    puntos = 2;
}
```

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

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.

```
private Resultado obtenerResultado(HttpSession session) {
    Resultado resultado = (Resultado) session.getAttribute("resultado");
    // Si no existe en la sesión, crear uno nuevo y se guarda en la sesión:
    if (resultado == null) {
        resultado = new Resultado();
        session.setAttribute("resultado", resultado);
    }
    return resultado;
}
//obtenerResultado
```

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:

```
// Obtener el objeto Resultado de la sesión
Resultado resultado = obtenerResultado(session);

// Actualizar los puntos acumulados en el objeto Resultado
resultado.setPuntos(resultado.getPuntos() + puntos);

// Agregar el objeto Resultado actualizado al modelo
model.addAttribute("resultado", resultado);

return "pregunta2";
```

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

```
@PostMapping("/pregunta2")
public String pregunta2(
    @RequestParam (value = "opciones", required = false)
    String[] opciones,
    HttpSession session,
    Model model) {
```

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:

```
<form method="post" action="/pregunta2">
  <!-- Opciones para asignaturas de Hogwarts -->
  <input type="checkbox" name="opciones" value="pociones"> Pociones<br>
  <input type="checkbox" name="opciones" value="defensa"> Defensa Contra las A
  <input type="checkbox" name="opciones" value="astronomia"> Astronomía<br>
```

- 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.

```
int puntos = 0;
// Gestión respuesta de la pregunta 2 (checkbox)
// asignar puntos según las opciones seleccionadas
if (opciones != null) {
    puntos = opciones.length;
}
```

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:

```
// Actualizar la sesión con los puntos obtenidos
Resultado resultado = obtenerResultado(session);
resultado.setPuntos(resultado.getPuntos() + puntos);
model.addAttribute("resultado", resultado);

return "pregunta3";
```

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

```
@PostMapping("/pregunta3")
public String pregunta3(
    @RequestParam(name = "respuesta") String respuesta,
    HttpSession session,
    Model model) {
```

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.

```
<form method="post" action="/pregunta3">
  <!-- Desplegable para profesores de Hogwarts -->
  <select name="respuesta">
    <option value=""> </option>
    <option value="minerva">Minerva McGonagall</option>
    <option value="snape">Severus Sanape</option>
    <option value="sprout">Pomona Sprout</option>
    <option value="flitwick">Filius Flitwick</option>
  </select>
```

- 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.

```

int puntos = 0;
// Gestión respuesta de la pregunta 3 (select)
// asignar puntos según la opción seleccionada
if (respuesta.equals("minerva")) {
    puntos = 4;
} else if (respuesta.equals("snape")) {
    puntos = 3;
} else if (respuesta.equals("flitwick")) {
    puntos = 2;
} else if (respuesta.equals("sprout")) {
    puntos = 1;
}

```

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:

```

// Actualizar la sesion con los puntos obtenidos
Resultado resultado = obtenerResultado(session);
resultado.setPuntos(resultado.getPuntos() + puntos);
model.addAttribute("resultado", resultado);

return "pregunta4";

```

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

```

@PostMapping("/pregunta4")
public String pregunta4(
    @RequestParam(name = "respuesta") String respuesta,
    HttpSession session,
    Model model) {
    ...
}

```

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.

```

<button type="submit" name="respuesta" value="gloria">Gloria</button>
<button type="submit" name="respuesta" value="sabiduria">Sabiduría</b
<button type="submit" name="respuesta" value="amor">Amor</button>
<button type="submit" name="respuesta" value="poder">Poder</button>

```

- **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.

```
// Gestión de la respuesta de la pregunta 4 (botones)
switch (respuesta) {
    case "gloria":
        puntos = 4;
        break;
    case "poder":
        puntos = 3;
        break;
    case "sabiduria":
        puntos = 2;
        break;
    case "amor":
        puntos = 1;
        break;

    default:
        // para valores no esperados
        break;
}
```

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

Y devuelve la pregunta5:

```
// Actualizar la sesión con los puntos obtenidos
Resultado resultado = obtenerResultado(session);
resultado.setPuntos(resultado.getPuntos() + puntos);
model.addAttribute("resultado", resultado);

return "pregunta5";
} //pregunta4
```

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

```
@PostMapping("/pregunta5")
public String pregunta5(
    @RequestParam(name = "respuesta") String respuesta,
    HttpSession session,
    Model model) {
```

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.

```
<a>Elige entre varita, escoba, libro o espada</a>
<textarea name="respuesta" rows="2" cols="50" placeholder="varita, esc<br>
```

- **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 minúsculas 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()`:



```
// pasar a minúsculas y eliminar espacios en blanco
String respuestaReal = respuesta.toLowerCase().trim();

// Comprobar la respuesta con las opciones válidas:
if (respuestaReal.equals("espada")) {
    puntos = 4;
} else if (respuestaReal.equals("varita")) {
    puntos = 3;
} else if (respuestaReal.equals("libro")) {
    puntos = 2;
} else if (respuestaReal.equals("escoba")) {
    puntos = 1;
}
}
```

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

Y por último devuelve la página siguiente, la pregunta6:

```
// Actualizar la sesión con los puntos obtenidos
Resultado resultado = obtenerResultado(session);
resultado.setPuntos(resultado.getPuntos() + puntos);
model.addAttribute("resultado", resultado);

return "pregunta6";
} //preg5
```

**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 button 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 y los parámetros y resto de código es también es igual:

```
@PostMapping("/pregunta6")
public String Pregunta6(
    @RequestParam(name = "respuesta") String respuesta,
    HttpSession session,
    Model model) {

    int puntos = 0;

    //manejar la respuesta de la pregunta 6 (radio button)
    if (respuesta.equals("gryffindor")) {
        puntos = 4;
    } else if (respuesta.equals("hufflepuff")) {
        puntos = 1;
    } else if (respuesta.equals("slytherin")) {
        puntos = 3;
    } else if (respuesta.equals("ravenclaw")) {
        puntos = 2;
    }

    // Obtener el objeto Resultado de la sesión
    // y actualizo los puntos
    Resultado resultado = obtenerResultado(session);
    resultado.setPuntos(resultado.getPuntos() + puntos);
    model.addAttribute("resultado", resultado);

    return "pregunta7";
} //preg6
```

Line: 187

**Método pregunta7 @PostMapping ("pregunta7"): tipo de pregunta button con name=respuesta:**

En esta pregunta repito tipo de campos del formulario de la pregunta 4, son de tipo button. Por lo que la los parámetros y lógica de puntuación es la misma, también he usado switch.

```
@PostMapping("/pregunta7")
public String pregunta7(
    @RequestParam(name = "respuesta") String respuesta,
    HttpSession session,
    Model model) {
    int puntos = 0;

    // Gestión de la respuesta de la pregunta 7 (botones)
    switch (respuesta) {
        case "gryff":
            puntos = 4;
            break;
        case "slyth":
            puntos = 3;
            break;
        case "raven":
            puntos = 2;
            break;
        case "huff":
            puntos = 1;
            break;

        default:
            // para valores no esperados
            break;
    }
}
```

Pero hay un añadido, al ser la última pregunta, además de actualizar la sesión con los puntos, 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 finalmente devuelve la última página del quiz que se llama finalResultado. Donde a través de la plantilla hacemos que muestre la clasificación y los puntos finales.

```
// Actualizar la sesión con los puntos obtenidos
Resultado resultado = obtenerResultado(session);
resultado.setPuntos(resultado.getPuntos() + puntos);
model.addAttribute("resultado", resultado);

// y actualizamos la clasificación, accediendo a los puntos:
resultado.setClasificacion(calcularClasificacion(resultado.getPuntos()));

return "finalResultado";
} //pregunta7
```

**Método calcularClasificación:**

Como indicado en el punto anterior, este método devuelve la clasificación que para ello recibe por parámetro los puntos del objeto Resultado (que pasamos al llamar a la función con Resultado.getPuntos) y mediante condiciones "if" vamos devolviendo la clasificación del tipo enum según la cantidad de puntos conseguidos, los cuales son la suma de los puntos de todas las preguntas.

```
private Clasificacion calcularClasificacion(int puntos) {
    // determinar la clasificación según los puntos
    if (puntos >= 20) {
        return Clasificacion.GRYFFINDOR;
    } else if (puntos >= 15) {
        return Clasificacion.RAVENCLAW;
    } else if (puntos >= 10) {
        return Clasificacion.SLYTHERIN;
    } else {
        return Clasificacion.HUFFLEPUFF;
    }
}
} //calcularClasif
```

**Método obtenerResultado:** explicado en el punto de la pregunta1.

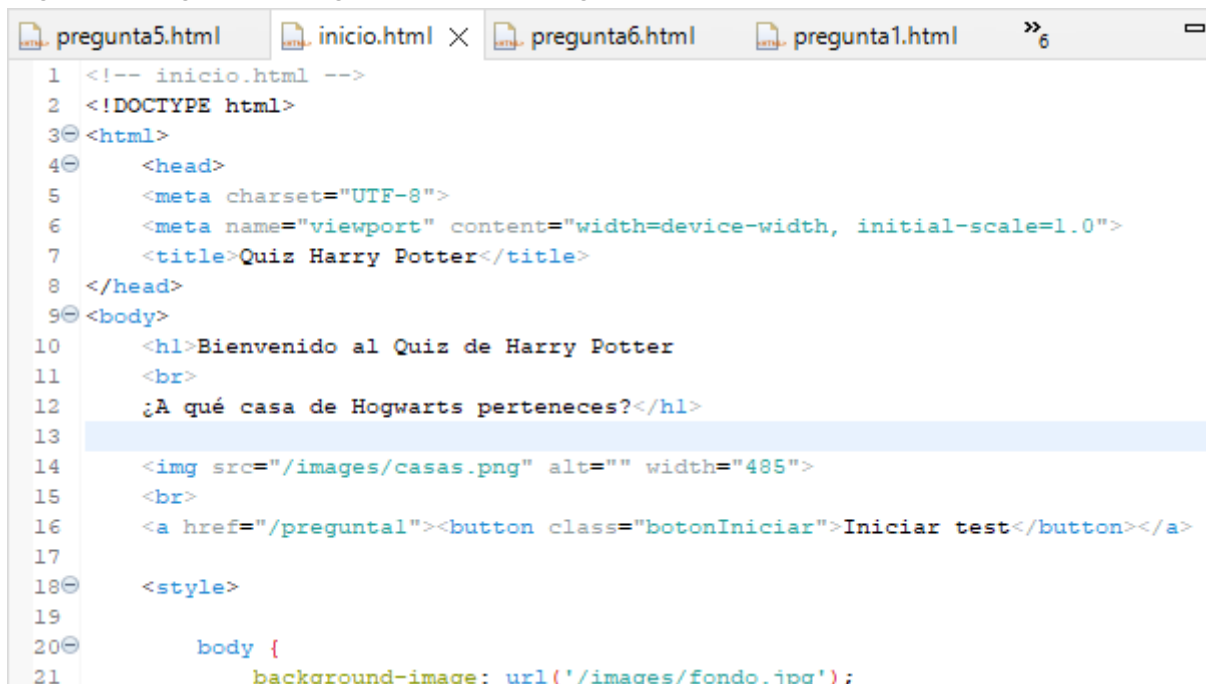
#### 4. Crear plantillas (vista):

El planteamiento es crear una plantilla por cada página del quiz, en mi caso son 9 plantillas: 1 de inicio, 1 de final y 1 por cada pregunta (7 preguntas).

Cada plantilla tiene **imágenes** que están almacenadas en una carpeta images en src/main/resources/static, y también tiene una parte de **estilos** css incluidos en etiquetas **style** (lo correcto sería crear una única página de estilos.css e importarla en cada plantilla html, lo dejo como una futura mejora de la aplicación).

**-inicio.html:** página

Página a modo de inicio que devuelve el método inicio al cargar la app en la raíz ("/"). Esta página se compone de 1 título, una imagen y un botón "Iniciar test", que gracias al método pregunta1 @GetMapping, nos redirige a la primera pregunta:



```
1 <!-- inicio.html -->
2 <!DOCTYPE html>
3 <html>
4 <head>
5 <meta charset="UTF-8">
6 <meta name="viewport" content="width=device-width, initial-scale=1.0">
7 <title>Quiz Harry Potter</title>
8 </head>
9 <body>
10 <h1>Bienvenido al Quiz de Harry Potter
11 <br>
12 ¿A qué casa de Hogwarts perteneces?</h1>
13
14 
15 <br>
16 <a href="/pregunta1"><button class="botonIniciar">Iniciar test</button></a>
17
18 <style>
19
20 body {
21     background-image: url('/images/fondo.jpg');
```

**Plantillas de las preguntas: todas se componen de:**

- título h2.
- imagen img.
- formulario form (cada plantilla tiene formularios diferentes).
- botón input submit (excepto en pregunta4.html y pregunta7.html).
- párrafo con los puntos (excepto en la pregunta1.html).

Para mostrar los puntos usamos el Thymelaf, con el atributo th:text y captamos los puntos con \${resultado.puntos}, pues los puntos son del objeto resultado.

- estilos css en style.

**Paso a explicar la parte de las preguntas/formulario de cada plantilla:****-pregunta1.html:**

En la plantilla de la pregunta 1 tenemos la pregunta de tipo radio, las opciones tienen en común el name=respuesta, que es el atributo por el que accedemos al valor en el controlador gracias al RequestParam. Hay cuatro input+label, una por cada clasificación

```
<form method="post" action="/pregunta1">
  <div class="center">
    <!-- Pregunta para Hufflepuff -->
    <input type="radio" name="respuesta" value="hufflepuff" id="hufflepuff">
    <label for="hufflepuff">Lealtad</label><br>

    <!-- Pregunta para Slytherin -->
    <input type="radio" name="respuesta" value="slytherin" id="slytherin">
    <label for="slytherin">Liderazgo</label><br>

    <!-- Pregunta para Gryffindor -->
    <input type="radio" name="respuesta" value="gryffindor" id="gryffindor">
    <label for="gryffindor">Valentía</label><br>

    <!-- Pregunta para Ravenclaw -->
    <input type="radio" name="respuesta" value="ravenclaw" id="ravenclaw">
    <label for="ravenclaw">Inteligencia</label><br>
  </div>
  <br>

  <input class="boton" type="submit" value="Siguiente">
</form>
```

**-pregunta2.html:**

En esta plantilla la pregunta es de tipo checkbox, que es multirespuesta. Hay 11 opciones y tienen en común el name=opciones, que al acceder a sus valores, opciones pasa a ser un array con sus values como valores del array.

```

<form method="post" action="/pregunta2">
  <!-- Opciones para asignaturas de Hogwarts -->
  <input type="checkbox" name="opciones" value="pociones"> Pociones<br>
  <input type="checkbox" name="opciones" value="defensa"> Defensa Contra las Artes
  <input type="checkbox" name="opciones" value="astronomia"> Astronomía<br>
  <input type="checkbox" name="opciones" value="transformaciones"> Transformaciones
  <input type="checkbox" name="opciones" value="herbologia"> Herbología<br>
  <input type="checkbox" name="opciones" value="muggles"> Estudios muggles<br>
  <input type="checkbox" name="opciones" value="adivinación"> Adivinación<br>
  <input type="checkbox" name="opciones" value="encantamientos"> Encantamientos<br>
  <input type="checkbox" name="opciones" value="historia"> Historia de la magia<br>
  <input type="checkbox" name="opciones" value="criaturas"> Cuidado criaturas mágicas
  <input type="checkbox" name="opciones" value="vuelo"> Vuelo<br>
</br>

```

### -pregunta3.html:

El formulario de esta pregunta es con un select con opciones en un desplegable. Con cuatro opciones, una por clasificación. Las opciones se relacionan con el name=respuesta en la etiqueta select. Que es a lo que accedemos desde el controlador con requestParam.

```

<form method="post" action="/pregunta3">
  <!-- Desplegable para profesores de Hogwarts -->
  <select name="respuesta">
    <option value=""> </option>
    <option value="snape">Severus Sanape</option>
    <option value="sprout">Pomona Sprout</option>
    <option value="minerva">Minerva McGonagall</option>
    <option value="flitwick">Filius Flitwick</option>
  </select>

```

### -pregunta4.html:

Esta pregunta es de tipo button submit. Hay cuatro botones y también tienen en común el atributo name= respuesta, que accedemos a él en el controlador con requestParam.

En esta plantilla, al ser de tipo button, no hace falta botón aparte para ir a la siguiente pregunta, pues al darle a cada botón de las opciones, ya se envía el formulario.

```

<form method="post" action="/pregunta4">
  <button type="submit" name="respuesta" value="amor">Amor</button>
  <button type="submit" name="respuesta" value="gloria">Gloria</button>
  <button type="submit" name="respuesta" value="sabiduria">Sabiduría</button>
  <button type="submit" name="respuesta" value="poder">Poder</button>
<br><br>

```

### -pregunta5.html:

Aquí tenemos un párrafo y un textarea donde el usuario introduce un texto (se indica en el párrafo los textos que debe poner).

```

<form method="post" action="/pregunta5">
  <!-- Textarea para la pregunta 5 -->
  <p>Elige entre varita, escoba, libro o espada:</p>

  <textarea name="respuesta" rows="2" cols="40" placeholder="varita, escoba, libro">
<br><br>

```

**-pregunta6.html:**

En esta pregunta repetimos el fomato de radio de la pregunta 1:

```

1 <form method="post" action="/pregunta6">
2   <div class="center">
3     <!-- Pregunta para Slytherin -->
4     <input type="radio" name="respuesta" value="slytherin" id="slytherin">
5     <label for="slytherin">Basilisco</label><br>
6
7     <!-- Pregunta para Gryffindor -->
8     <input type="radio" name="respuesta" value="gryffindor" id="gryffindor">
9     <label for="gryffindor">Dragones</label><br>
10
11    <!-- Pregunta para Hufflepuff -->
12    <input type="radio" name="respuesta" value="hufflepuff" id="hufflepuff">
13    <label for="hufflepuff">Hipogrifo</label><br>
14
15    <!-- Pregunta para Ravenclaw -->
16    <input type="radio" name="respuesta" value="ravenclaw" id="ravenclaw">
17    <label for="ravenclaw">Elfo</label>
18  </div>

```

**-pregunta7.html:**

Y en esta repito el formato de formulario de la pregunta 4, son botones (button-submit), y de la misma manera, al ser botones, no es necesario poner un botón de ir a siguiente.

```

1 <form method="post" action="/pregunta7">
2   <button type="submit" name="respuesta" value="raven">
3     Presento argumentos logicos y coherentes.</button>
4   <button type="submit" name="respuesta" value="gryff">
5     Defiendo mi punto de vista con firmeza.</button><br>
6   <button type="submit" name="respuesta" value="huff">
7     Escucho a los demás e intengo llegar a un acuerdo.</button><br>
8   <button type="submit" name="respuesta" value="slyth">
9     Busco ganar la discusión a toda costa.</button><br>
10  <br>

```

**- finalResultado.html:**

Esta es la página final que devuelve el método de la pregunta 7, la última pregunta del quiz.

Aquí se trata de que según la clasificación obtenida, aparezca una imagen específica. Esto se hace con condicional if, usando el Thymelaf, dentro de la etiqueta img con th:if, poniendo la condición de si la clasificación es igual a cada palabra del enumerado, se pone dicha imagen.

A la clasificación accedemos con resultado.clasificación.toString(), el toString es necesario para recorrer dicho string. La idea del toString se lo tuve que preguntar a ChatGpt, pues no daba con la solución de por qué no se mostraba la imagen.

Estas imágenes están dentro de un div con la condición if de si resultado y clasificación no son nulos. Esta línea la puse cuando tuve el error de que no se mostraba la última página (explicado en el apartado de problemas. Y al ponerlo, cuando aún no solucioné el error, conseguí mostrar la página aunque solo saliera el título, lo que me ayudó un poco a dirigirme donde estaba el error.

```

10 <h2>;Quiz Finalizado!</h2>
11 <p th:text="'Has sido seleccionado para la casa ' + ${resultado.clasificacion}" />
12
13 <!-- Verificar si resultado y resultado.clasificacion no son nulos -->
14<div th:if="${resultado != null and resultado.clasificacion != null}">
15     <!-- Condición para mostrar imágenes según la clasificación -->
16     
17     
18     
19     
20
21     <p th:text="'Has conseguido ' + ${resultado.puntos} + ' puntos.'" /p>
22 </div>

```

## 5. ESTILOS - PUNTO ADICIONAL:

Como indicado en puntos anteriores, aunque lo ideal sería crear una página de estilos css individual, en la carpeta static, en este caso he aplicado los estilos en cada plantilla html dentro de la etiqueta style. Hay algunos estilos propios pero hay otros que son iguales, por lo que estoy repitiendo código. ***Esta mejora la dejo pendiente para hacerla en un futuro.***

Los estilos globales de la aplicación, es en cada página tiene una imagen de fondo que ocupa el 100% de la ventana (esto no estoy segura si en todas las pantallas se ve igual, pues en mi portátil de casa no se ve igual que en la pantalla del pc de clase), unos títulos a modo de nombre de pregunta con sombreado negro dado que el fondo es oscuro así se ve mejor el texto, una imagen con sombreado blanco y bordes redondeados (solo las imágenes de las preguntas), y botones para enviar el formulario a la siguiente pregunta/página.

Y todos los elementos están centrados en la ventana del navegador.

La parte de los formularios, tienen un fondo rosa claro y también sombreado blanco y los botones verdes y cuando posas el ratón encima se oscurece el color.

También he aplicado algunos divs dentro del código html a modo de separar los elementos y darles márgenes.

También aplico estilo dándole tamaño a las imágenes, a los textos y elementos del formulario. Jugando con tamaños y márgenes para que quede todo de una manera estética.

```

=====
=====

```

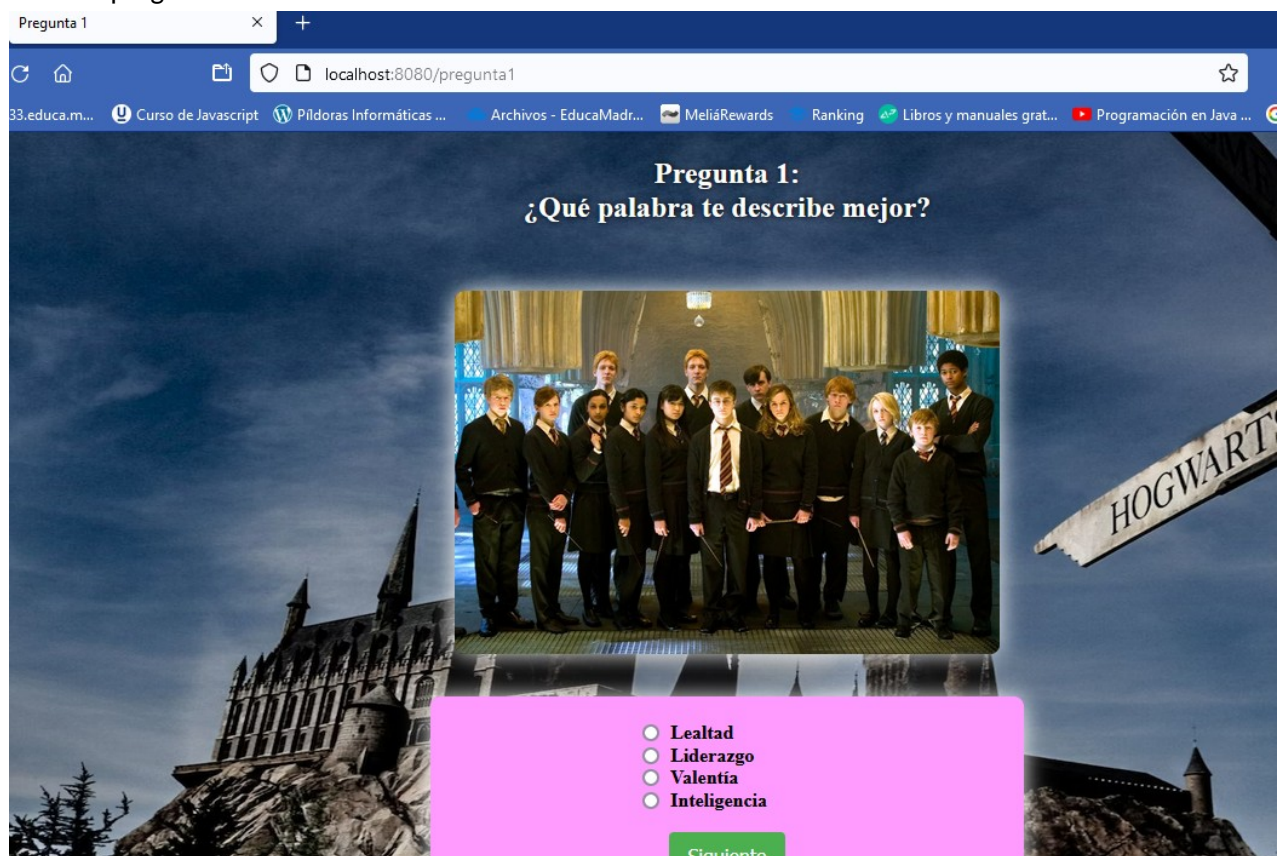


## MUESTRA:

Página inicio:



Primera pregunta:






## Segunda pregunta:

Pregunta 2

localhost:8080/pregunta1

80%

Pregunta 2:  
¿Qué asignaturas elegirías?



- ☐ Pociones
- ☐ Defensa Contra las Artes Oscuras
- ☐ Astronomía
- ☐ Transformaciones
- ☐ Herbología
- ☐ Estudios muggles
- ☐ Adivinación
- ☐ Encantamientos
- ☐ Historia de la magia
- ☐ Cuidado criaturas mágicas
- ☐ Vuelo

Siguiente


Llevas 1 puntos.

## Tercera pregunta:

Pregunta 3

localhost:8080/pregunta2

Pregunta 3:  
Elige uno de estos profesores



Siguiente

Llevas 5 puntos.

## Cuarta pregunta:

Pregunta 4

localhost:8080/pregunta3

Pregunta 4:  
Dada la opción, preferirías inventar una poción que garantizara:



Amor Gloria Sabiduría Poder


Llevas 9 puntos.

## Quinta pregunta:

Pregunta 5

localhost:8080/pregunta4

Pregunta 5:  
Escribe qué objeto elegirías:



Elige entre varita, escoba, libro o espada:

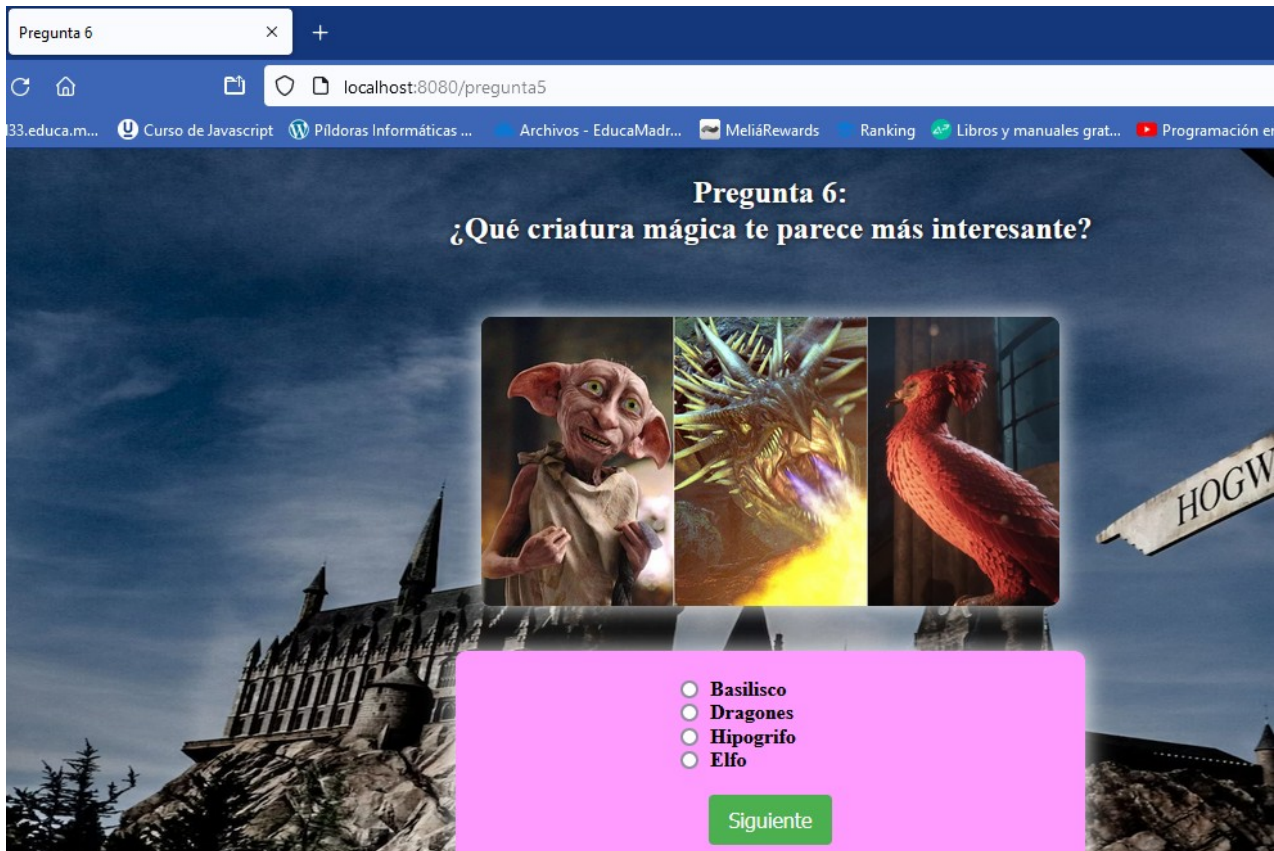
varita, escoba, libro o espada

Siguiente

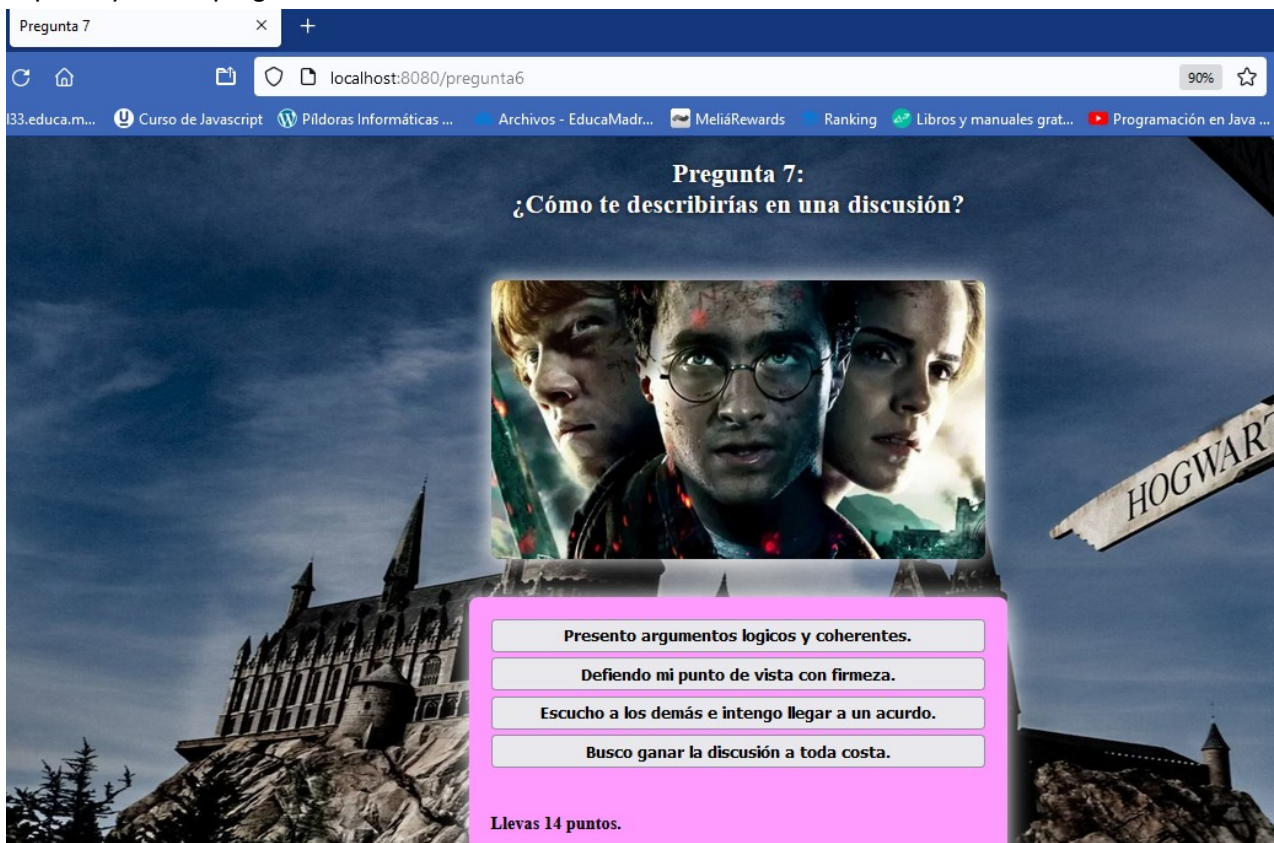
Llevas 11 puntos.



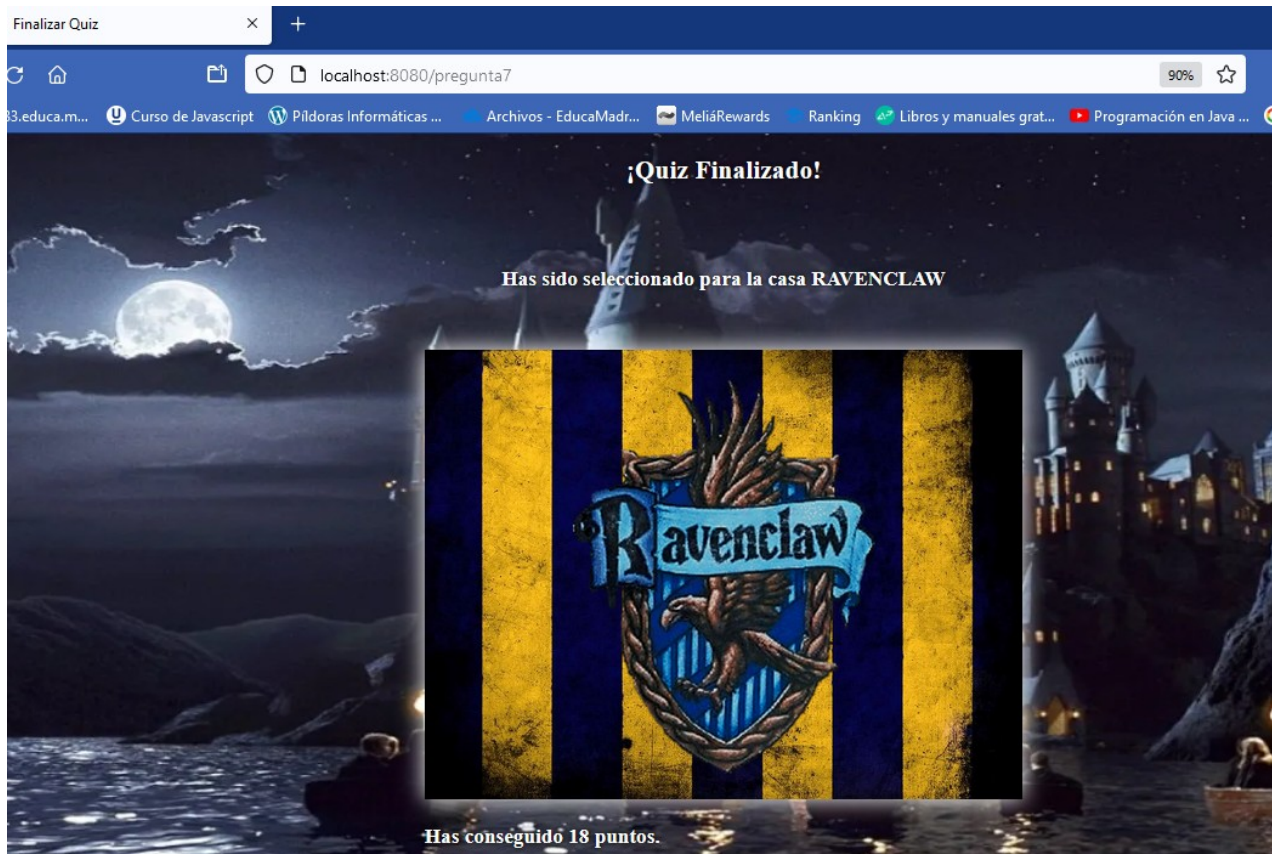
Sexta pregunta:



Séptima y última pregunta:



Página final de resultado:



### Problemas / anotaciones:

- El planteamiento inicial 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.

```
@GetMapping("/preguntal")
public String preguntal() {
    return "preguntal";
}
```

- 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.

```
@GetMapping("/")
public String inicio(HttpSession session, Model model) {
    // Reiniciar los puntos en el inicio creando nuevo objeto/modelo
    Resultado resultado = new Resultado();
    session.setAttribute("resultado", resultado);
    return "inicio"; //se muestra la pagina inicio.
} //inicio
```

- 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, en cada pregunta, almacenar en la sesión los puntos, en lugar del objeto Resultado y 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, si no que se sustituían.

Para solucionarlo creé la función obtener resultado donde creo un objeto Resultado nuevo si no existe, 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 sobraba y lo 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.

```

        resultado.setClasificacion(calcularClasificacion(resultado.getPuntos()));
    }

    return "finalResultado";
}

// @PostMapping("/finalResultado")
// public String finalizar(
//     HttpSession session,
//     Model model) {
//     int puntos = obtenerPuntos(session);
//     Clasificacion clasificacion = calcularClasificacion(puntos);
//     Resultado resultado = new Resultado();
//     resultado.setClasificacion(clasificacion);
//     resultado.setPuntos(puntos);
//     model.addAttribute("resultado", resultado);
//     return "finalResultado";
// }

```

- 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.



## CONCLUSIONES:

Esta práctica ha sido muy formativa una vez hecha. Al principio me fue muy difícil cómo plantear el código, cómo empezar, pues ha sido un gran salto de dificultad de la anterior práctica a esta. Tuve que preguntar a varios compañeros.

Después de iniciar la aplicación y probar con 3 preguntas, entendía bien el código y pude seguir más fluidamente. Pude realizar las siguientes preguntas sin mucha dificultad, pues aunque los campos de cada formulario son diferentes, la forma de acceder a los datos es igual o muy parecida.

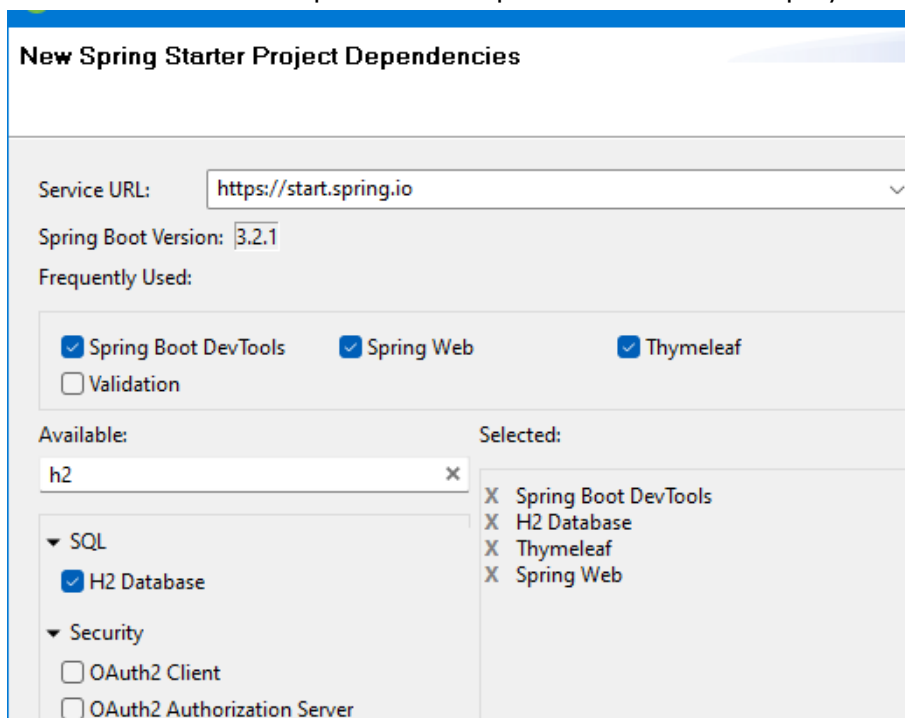
He quedado muy satisfecha y contenta con el resultado, sobre todo al aplicar los estilos y las imágenes. La parte de diseño me gusta mucho y ha sido muy placentera esta parte.

Pero le he dedicado muchísimas horas y aún así ni me he planteado intentar la parte del excelente (aprox 18h en casa).

## AMPLIACIÓN:

- **Inserción de resultados en la base de datos:** Los resultados de cada jugador debe ser almacenado en la base de datos. De esta manera cada vez que un jugador acabe el quiz se le mostrará el resultado obtenido con su clasificación.

Para usar bbdd tenemos que añadir la dependencia H2 a nuestro proyecto:



En la bbdd almacenaré el nombre de usuario, puntos y clasificación. Para recibir el nombre crearé más adelante un campo de texto en una de las plantillas.

Y por lo tanto añado el atributo nombre a mi clase Resultado, con su get y set:

```

1 package com.example.demo;
2
3 public class Resultado {
4     private Clasificacion clasificacion;
5     private int puntos;
6
7     private String nombre;
8
9     public String getNombre() {
10         return nombre;
11     }

```

Luego, tengo que crear un repositorio que maneje la lógica para guardar y recuperar los resultados. Es decir, esta clase se encargará de gestionar los resultados. A modo de base de datos:

```

1 package com.example.demo;
2
3 import java.util.ArrayList;
4
5 @Service
6 public class ResultadoRepositorio {
7
8     private List<Resultado> resultados = new ArrayList<>();
9
10    public void guardarResultado(Resultado resultado) {
11        resultados.add(resultado);
12    }
13
14    public List<Resultado> obtenerTodosLosResultados() {
15        return resultados;
16    }
17 }

```

- Creo una lista, ArrayList, para almacenar los objetos de tipo Resultado.
- El método guardarResultado recibe un resultado por parámetro y lo añade a la lista.
- Y el método obtenerTodosLosResultados simplemente devuelve la lista completa.

Ahora hay que modificar la clase Controlador para inyectar la clase ResultadoRepositorio y así poder usarlo para almacenar y recuperar la lista de los resultados.

Añadimos anotación Autowired al atributo objeto ResultadoRepositorio para que Spring lo gestione y proporcione automáticamente. La anotación Autowired es fundamental para poder acceder y manipular la lista.

```

1 @Controller
2 public class QuizController {
3
4     @Autowired
5     private ResultadoRepositorio resultadoRepositorio;
6

```

Como tengo que añadir un campo de texto en alguna plantilla para que el usuario introduzca su nombre, mi nuevo planteamiento de la aplicación es añadir una página extra que he llamado paginaNombre.html,

que va entre la pregunta 7 y la página final. Por lo que el método pregunta7 hay que modificarlo, para que su return sea paginaNombre.

```
// Actualizar la sesión con los puntos obtenidos
Resultado resultado = obtenerResultado(session);
resultado.setPuntos(resultado.getPuntos() + puntos);
model.addAttribute("resultado", resultado);

// y actualizamos la clasificación, accediendo a los puntos:
resultado.setClasificacion(calcularClasificacion(resultado.getPuntos()));

return "paginaNombre";
} //pregunta7
```

Y por lo tanto, ahora toca crear un método nuevo, con la misma dinámica que los de las preguntas pero sin gestión de puntos, para la paginaNombre.

En este método tenemos que recibir el parámetro nombre (que recibe de su plantilla a través de un campo de texto) y agregarlo al objeto resultado guardado en la sesión.

```
@PostMapping("/paginaNombre")
public String paginaNombre(
    @RequestParam(name = "nombre") String nombre,
    HttpSession session,
    Model model) {

    // Obtener el objeto Resultado de la sesión
    Resultado resultado = obtenerResultado(session);

    // Actualizar el nombre en el objeto Resultado
    resultado.setNombre(nombre);
    // Agregar el objeto Resultado actualizado al modelo
    model.addAttribute("resultado", resultado);
```

Luego este objeto Resultado lo guardamos en el repositorio, que gracias a la anotación Autowired podemos acceder).

```
// Guardar el resultado en el repositorio
resultadoRepositorio.guardarResultado(resultado);
```

Ahora ya tenemos nuestro objeto Resultado completo con los tres valores, es decir, con puntos, clasificación y con nombre.

Por lo que ya podemos acceder a él para mostrar el resultado.

Para obtener los resultados del repositorio, como es un List, creamos uno al que llamamos todosLosResultados y le damos el valor de la lista llamando al método obtenerTodosLosResultados (método creado en la clase resultadoRepositorio donde devuelve la lista completa).

```
// Obtener todos los resultados del repositorio
List<Resultado> todosLosResultados = resultadoRepositorio.obtenerTodosLosResultados();
```

Pero solo queremos mostrar los últimos 5 resultados, por lo que tenemos que crear una variable de inicio y otra de fin para asignar posiciones de la lista, y con la función .subList y estas variables conseguimos la lista con estos 5 resultados.



Y esta nueva lista, se la asignamos a una nueva variable List llamada `ultimosResultados` y la agregamos al modelo.

Y por último, en el return estará la siguiente página, que será `finalResultado`.

```
// Seleccionar los últimos 5 resultados (o menos si hay menos de 5)

//la posición será la última de la lista.
int inicio = todosLosResultados.size();
//si la lista tiene > 5 elementos, la posición es size-5, si no, es la posición 0
int fin = todosLosResultados.size() > 5 ? todosLosResultados.size() - 5 : 0;
List<Resultado> ultimosResultados = todosLosResultados.subList(inicio, fin);

// Agregar la lista de últimos resultados al modelo
model.addAttribute("ultimosResultados", ultimosResultados);

return "finalResultado";
}
```

**List<Resultado> ultimosResultados - com.example.de  
HttpSession, Model)**

Ahora que ya tenemos el código completo, vamos a las plantillas.

Creamos plantilla `paginaNombre.html`, donde tenemos que crear un campo de texto para que el usuario introduzca su nombre.

Y también ponemos un botón para que vaya a la página siguiente (en este caso a la página final):

El código es muy simple, tenemos que cumplir con poner el atributo `name="nombre"`, ya que es el parámetro que recibe el nombre para captar el nombre del usuario:

```
<form method="post" action="/paginaNombre">
  <input type="text" id="nombre" name="nombre" required placeholder="Escribe tu nombre">
  <br>
  <input class="boton" type="submit" value="Continuar">
</form>
```

Y como la tabla de resultados la voy a mostrar en la página final, hay que modificar la plantilla `finalResultado.html`.

Como tenemos que imprimir todos los elementos de la lista, y cuyos elementos son objetos, tenemos que recorrerla, para que nos devuelva un objeto por cada elemento.

Para ello usaremos el `for-each`, lo que en `thymeleaf` es `th:each`.

Al querer mostrarlos en una tabla, creamos tabla, ponemos nombre de las columnas/valores en las etiquetas `th`, y luego en una fila, etiqueta `tr`, ponemos el `th:each`, llamando a la lista `ultimosResultados`. Y en cada recorrido de la lista, se crea una fila, que con `resultado.nombre/puntos/clasificación`, llamamos a cada valor.

```

<table border="1">
  <thead>
    <tr>
      <th>Nombre</th>
      <th>Puntos</th>
      <th>Clasificación</th>
    </tr>
  </thead>
  <tbody>
    <!-- For each sobre la lista de resultados -->
    <tr th:each="resultado : ${ultimosResultados}">
      <td th:text="${resultado.nombre}" />
      <td th:text="${resultado.puntos}" />
      <td th:text="${resultado.clasificacion}" />
    </tr>
  </tbody>
</table>

```

En esta plantilla también he añadido un botón para volver la página inicio y así volver a empezar el quiz. Y como en su momento, en el método inicio del controlador puse código para inicializar los puntos, al volver a inicio desde este botón, el quiz empieza de nuevo desde 0:

```

<a th:href="@{/}"><button class="botonIniciar">Volver a empezar</button></a>

```

Muestra:

**¡Quiz Finalizado!**

Has sido seleccionado para la casa SLYTHERIN y has conseguido 12 puntos.

**Slytherin**

Volver a empezar

**Últimos 5 Resultados**

Nombre	Puntos	Clasificación
Arancha	7	HUFFLEPUFF
Eva	28	GRYFFINDOR
Pepe	23	GRYFFINDOR
Joselito	15	RAVENCLAW
Sara	12	SLYTHERIN