D2-UD4 Sergio Berrendero Toledano

https://github.com/SergioBerren/DWES-Spring/tree/master/Tema4/D2UD4

En esta práctica se nos pedía crear un quiz en el cual dependiendo de las respuestas pudiese dar como mínimo cuatro resultados posibles. La intención de la práctica aprender el manejo de sesiones en SpringBoot.

Para realizar esta práctica hemos utilizados las dependecias de *Validation*, *SpringBoot DevTools*, *Spring Web* y *Thymeleaf.*

Para hacer esta práctica he creado tres clases java (sin contar la que viene por defecto), dieciséis html (quince que son preguntas, un index que es un "inicio de sesión" y el html que muestra los resultados). Además he creado unas hojas de estilos css para que sea más bonito a la vista, para que me muestre distintos css dependiendo del resultado he creado cuatro hojas css y con *th:if* dependiendo de lo que dé el resultado use un css u otro.

CLASES JAVA

Clase Elemento

Esta clase define un enum *Elemento* con límites de puntuación. Cada elemento tiene un máximo (*maxPuntuacion*), y el método *obtenerElementoPorPuntuacion* devuelve el elemento correspondiente a una puntuación dada.

```java
package com.example.quiz;

public enum Elemento {
    TIERRA(30),
    AIRE(40),
    AGUA(50),
    FUEGO(Integer.MAX_VALUE); // Usamos Integer.MAX_VALUE para que Fuego cubra cualquier puntuación mayor a 50

    private final int maxPuntuacion;

    // Constructor para asignar la puntuación máxima a cada elemento
    Elemento(int maxPuntuacion) {
        this.maxPuntuacion = maxPuntuacion;
    }

    // Método para obtener la puntuación máxima
    public int getMaxPuntuacion() {
        return maxPuntuacion;
    }

    // Método estático para obtener el elemento basado en la puntuación
    public static Elemento obtenerElementoPorPuntuacion(int puntuacion) {
        for (Elemento elemento : Elemento.values()) {
            if (puntuacion <= elemento.getMaxPuntuacion()) {
                return elemento;
            }
        }
        return FUEGO;  // Si no se encuentra, se devuelve "FUEGO" por defecto
    }
}
```

Clase Usuario

En esta clase creamos los campos nombre, apellidos y descripción junto a sus etiquetas de validación, *@Size* para determinar el tamaño que debe tener cada String y *@NotBlank* para no poder dejarlo vacío. Todas estas etiquetas junto a sus respectivos mensajes de error.
En esta clase también hemos creados los getters y los setters.

```java
package com.example.quiz;

import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Size;

public class Usuario {

    @Size(min = 6, message = "El nombre debe tener al menos 6 caracteres")
    @Size(max = 12, message = "El nombre no puede tener más de 12 caracteres")
    @NotBlank(message = "El nombre es obligatorio")
    private String nombre;

    @Size(min = 6, message = "El apellido debe tener al menos 6 caracteres")
    @Size(max = 15, message = "El apellido no puede tener más de 15 caracteres")
    @NotBlank(message = "El apellido es obligatorio")
    private String apellido;

    @Size(min = 6, message = "La descripción debe tener al menos 6 caracteres")
    @Size(max = 20, message = "La descripción no puede tener más de 20 caracteres")
    @NotBlank(message = "La descripción es obligatoria")
    private String descripcion;

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getApellido() {
        return apellido;
    }

    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    public String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }
}
```

Clase CalculoQuiz

Esta clase *CalculoQuiz* usa el enum *Elemento* para determinar el elemento correspondiente a una puntuación dada, devolviendo su nombre como cadena (*name()*).

```java
package com.example.quiz;

import org.springframework.stereotype.Service;

@Service
public class CalculoQuiz {

    public String determinarElemento(int puntuacion) {
        Elemento elemento = Elemento.obtenerElementoPorPuntuacion(puntuacion);
        return elemento.name();  // Devolver el nombre del elemento como cadena
    }
}
```

Clase QuizController

Esta clase gestiona el flujo de las preguntas, la validación de respuestas y el cálculo del resultado del quiz.

Método mostrarInicio

Este método maneja las peticiones *GET* a la ruta raíz. Es el primer punto de contacto cuando el usuario ingresa a la aplicación. Crea un objeto *Usuario* vacío y lo pasa al modelo, lo que permite que el formulario en la vista *index* se rellene con este objeto. Este formulario servirá para que el usuario ingrese su nombre, apellido y descripción.

Método comenzarQuiz
Este método maneja las peticiones *POST* a la ruta */inicio*. Recibe los datos del usuario a través de un formulario y los valida. Si la validación falla (por ejemplo, si hay campos vacíos o incorrectos), el formulario se vuelve a mostrar con los mensajes de error. Si la validación es exitosa, se guarda la información del usuario en la sesión y se inicializa su puntuación en 0. Finalmente, redirige al usuario a la primera pregunta del quiz.

Método mostrarPregunta
Este método maneja las peticiones *GET* a la ruta */pregunta/{numero}*, donde *{numero}* es el número de la pregunta actual. Recibe este número de pregunta como un parámetro de la URL, lo añade al modelo y devuelve la vista correspondiente a esa pregunta.

Método enviarRespuesta
Este método maneja las peticiones *POST* a la ruta */enviar*. Recibe la respuesta seleccionada por el usuario y la pregunta a la que corresponde. Si no se selecciona una respuesta, muestra un mensaje de error. Si se selecciona una respuesta, actualiza la puntuación del usuario en la sesión y redirige al usuario a la siguiente pregunta. Si el usuario ha respondido todas las preguntas, redirige a la página de resultados.

Método mostrarResultado
Este método maneja las peticiones *GET* a la ruta */resultado*. Recupera los datos del usuario y la puntuación de la sesión, y calcula el elemento asociado a esa puntuación usando el servicio *CalculoQuiz*. Luego, pasa esos datos a la vista *resultado* para mostrar el resultado del quiz.

```java
package com.example.quiz;

import jakarta.validation.Valid;
import org.springframework.validation.BindingResult;
import jakarta.servlet.http.HttpSession;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

@Controller
public class QuizController {

    @Autowired
    private CalculoQuiz calculoQuiz;

    @GetMapping("/")
    public String mostrarInicio(Model modelo) {
        modelo.addAttribute("usuario", new Usuario());
        return "index";
    }

    @PostMapping("/inicio")
    public String comenzarQuiz(@ModelAttribute @Valid Usuario usuario, BindingResult result, HttpSession sesion, Model modelo) {
        // Si hay errores de validación, vuelve al formulario de inicio
        if (result.hasErrors()) {
            modelo.addAttribute("usuario", usuario);
            return "index";  // Regresa al formulario de inicio
        }

        // Si no hay errores, continua con la lógica
        sesion.setAttribute("usuario", usuario);
        sesion.setAttribute("puntuacion", 0);
        return "redirect:/pregunta/1";
    }

    @GetMapping("/pregunta/{numero}")
    public String mostrarPregunta(@PathVariable("numero") int numero, Model modelo) {
        modelo.addAttribute("numeroPregunta", numero);
        return "pregunta" + numero;
    }

    @PostMapping("/enviar")
    public String enviarRespuesta(
            @RequestParam(value = "respuesta", required = false) Integer respuesta,
            @RequestParam("pregunta") int numeroPregunta,
            HttpSession sesion,
            Model modelo) {
        modelo.addAttribute("usuario", usuario);
        modelo.addAttribute("puntuacion", puntuacion);
        modelo.addAttribute("elemento", elemento.toLowerCase());
        modelo.addAttribute("descripcion", usuario.getDescripcion());

        return "resultado";
    }
}
```

```java
    @PostMapping("/enviar")
    public String enviarRespuesta(
            @RequestParam(value = "respuesta", required = false) Integer respuesta,
            @RequestParam("pregunta") int numeroPregunta,
            HttpSession sesion,
            Model modelo) {

        // Validar que la respuesta no sea nula
        if (respuesta == null) {
            modelo.addAttribute("error", "Debes seleccionar una respuesta antes de continuar.");
            modelo.addAttribute("numeroPregunta", numeroPregunta);
            return "pregunta" + numeroPregunta; // Vuelve a la misma página de la pregunta
        }

        // Obtener y actualizar la puntuación
        int puntuacion = (int) sesion.getAttribute("puntuacion");
        sesion.setAttribute("puntuacion", puntuacion + respuesta);

        // Redirigir a la siguiente pregunta o al resultado final
        if (numeroPregunta < 15) {
            return "redirect:/pregunta/" + (numeroPregunta + 1);
        } else {
            return "redirect:/resultado";
        }
    }

    @GetMapping("/resultado")
    public String mostrarResultado(HttpSession sesion, Model modelo) {
        Usuario usuario = (Usuario) sesion.getAttribute("usuario");
        if (usuario == null) {
            return "redirect:/"; // Redirige al inicio si no hay un usuario
        }

        int puntuacion = (int) sesion.getAttribute("puntuacion");
        String elemento = calculoQuiz.determinarElemento(puntuacion);

        modelo.addAttribute("usuario", usuario);
        modelo.addAttribute("puntuacion", puntuacion);
        modelo.addAttribute("elemento", elemento.toLowerCase());
        modelo.addAttribute("descripcion", usuario.getDescripcion());

        return "resultado";
    }
}
```

RESOURCES

Templates

Todas las preguntas contienen un *th:action* que llevan a */enviar* a excepción del *index* que nos lleva a */inicio*, también tienen un div en el que guardan los errores con un *th:if=”${error}”* en el div y un th:text=”${error}” en una etiqueta p, lo que decimos aquí es que si se cumple la condición muestre los errores los cuales fueron guardados anteriormente en el controlador.
Todas las preguntas también tienen un *<input type="hidden" name="pregunta" value="[número pregunta]">*, este guarda el número de la pregunta que necesitaremos en el controlador.

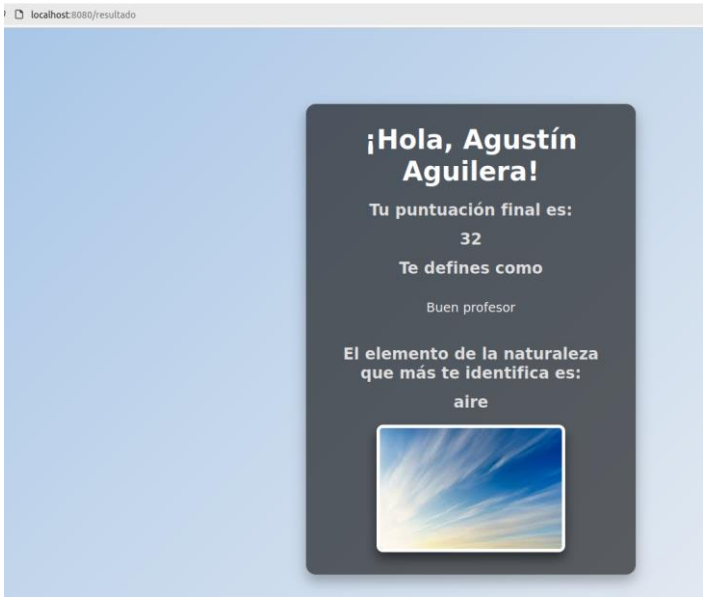
Por último cabe destacar las condiciones mencionadas al principio, con las cuales he decidido el css que debemos usar

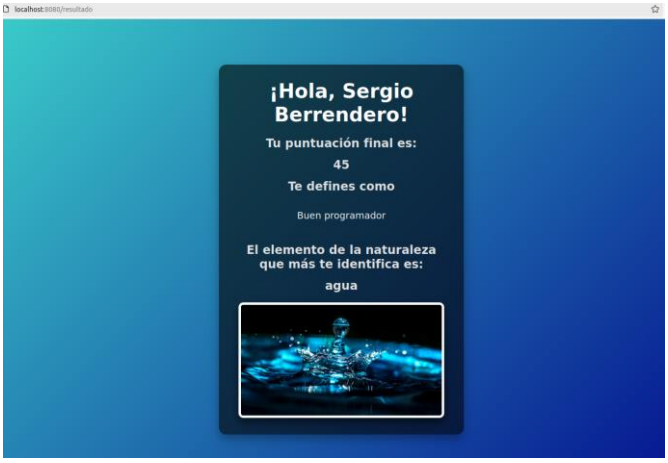
```html
<div th:if="${elemento == 'tierra'}">
    <link rel="stylesheet" href="/css/resultadoTierra.css">
</div>
<div th:if="${elemento == 'aire'}">
    <link rel="stylesheet" href="/css/resultadoAire.css">
</div>
<div th:if="${elemento == 'agua'}">
    <link rel="stylesheet" href="/css/resultadoAgua.css">
</div>
<div th:if="${elemento == 'fuego'}">
    <link rel="stylesheet" href="/css/resultadoFuego.css">
</div>
```

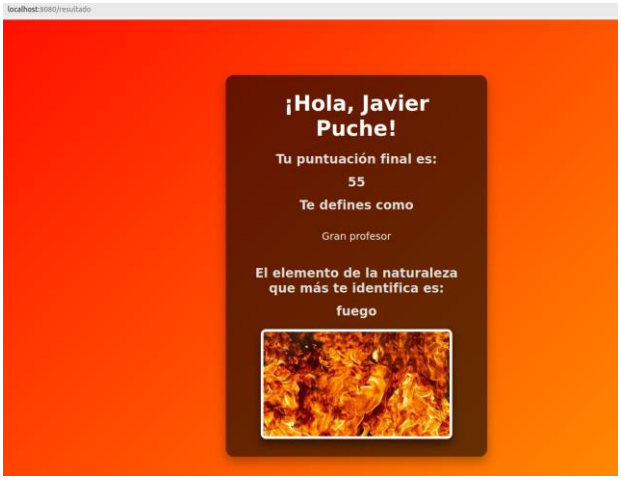
Static

En esta carpeta crearemos dos carpetas, una para los css y otra para las imágenes.

Funcionamiento







Conclusión

Este controlador forma parte de la lógica de negocio de una aplicación de quiz basada en Spring MVC.

Gestiona el flujo de preguntas, la validación de respuestas y la obtención de resultados.

La aplicación almacena el progreso del usuario en la sesión, lo que permite que el usuario continúe con el quiz sin perder su progreso.

Creo que esta práctica nos ha servido como toma de contacto con las sesiones y nos ayuda a empezar a soltarnos, además, los conocimientos básicos los tenemos de php (salvando las diferencias), lo que nos ha facilitado el trabajo a la hora de comprender su funcionamiento.