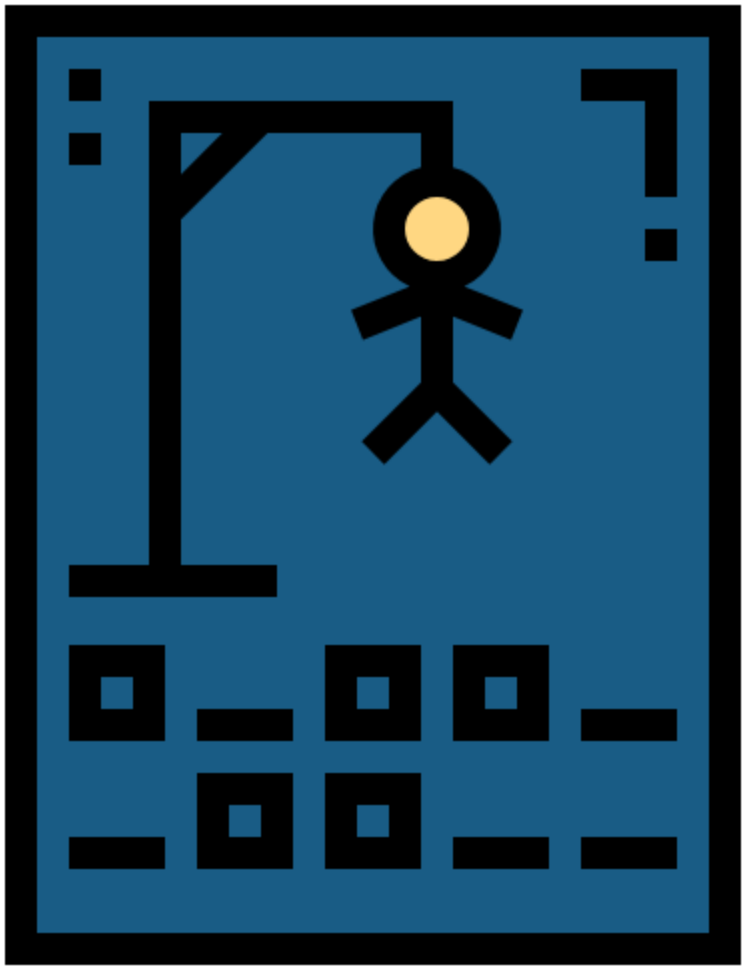
Creación de un Servicio Rest "Ahorcado" y el correspondiente Cliente Rest



IES Puerto de la Cruz Telesforo Bravo

Alanis Carolina Simoes Villalonga

Desarrollo Web en Entorno Servidor

2º DAW - Curso 2020/21

Índice

Introducción

En este informe se detalla en profundidad la realización de la práctica puntuable “Creación de un Servicio Rest "Ahorcado" y el correspondiente Cliente Rest”. Esto es con el fin de plasmar las ideas surgidas durante su desarrollo y cómo se abordaron ciertos problemas e inconvenientes a lo largo de su realización.

Cabe destacar que, debido a la falta de conocimientos en todo a lo que se refiere a un ‘Cliente Rest’, es posible que haya repeticiones de código, sentencias sobrantes y quizá alguna incongruencia.

Idea

Sobre cómo se realizaría el proyecto, opté por realizar un proyecto ‘Maven’ debido a la comodidad a la hora de gestionar las librerías. También decidí trabajar con una base de datos embebida por conveniencia en ‘sqlite’.

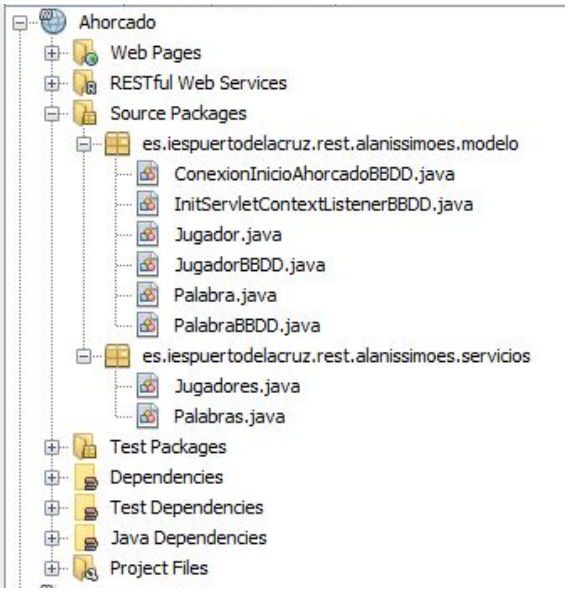
La lógica del proyecto es la siguiente:

Al jugador se le muestra un texto de guiones donde él tendrá que adivinar qué palabra se esconde detrás de estos. No tiene intentos ilimitados: existe una horca y, por cada error que cometa, aparecerá un miembro humano colgando de ella. El juego se termina si el jugador es capaz de adivinar la palabra correcta antes de formar un humano completo, ganando de esta manera. Por el contrario, si no es capaz de descubrir todas las letras que forman la palabra secreta y cuelga al humano por completo, pierde.

Realización

Servidor

A continuación se muestra la estructura de paquetes en el proyecto de Ahorcado servidor:



En el 'modelo' se encuentran todas las clases correspondientes a la gestión de la base de datos:

- **ConexionInicioAhorcadoBBDD:** clase que inicia la base de datos 'Ahorcado', abre y cierra la conexión con la misma, y crea sus tablas necesarias: 'palabras' y 'jugador'.
- **InitServletContextListenerBBDD:** clase que contiene un método que se ejecuta nada más iniciarse el proyecto: en él se llama a la creación de la base de datos y a la creación de sus tablas.
- **Jugador:** clase del jugador. Tiene sus 'getters' y 'setters' correspondientes, a la par que su 'toString'.
- **JugadorBBDD:** clase donde están todos los métodos que interactúan con los datos de la tabla 'jugador'. Ejemplos: 'insert', 'delete', 'update', etc.

- **Palabra:** clase de las palabras. Tiene sus 'getters' y 'setters' correspondientes, a la par que su 'toString'.
- **PalabraBBDD:** clase donde están todos los métodos que interactúan con los datos de la tabla 'palabras'. Ejemplos: 'insert', 'select', 'update', etc.

En la carpeta 'servicios' se encuentran, como su nombre indica, los servicios necesarios para poder decir que el proyecto es de tipo 'Rest':

- **Jugadores:** en esta clase se definen los diferentes tipos de acceso según la acción que queramos realizar sobre la tabla 'jugador. A continuación podemos ver un poco del código':

```
@Path("/jugador")
public class Jugadores {

    /** Comprobar el funcionamiento del Servicio ...5 lines */
    @GET
    @Path("/info")
    public Response getInfo() {
        String output = "Servicio Jugadores OK ";
        return Response.status(200).entity(output).build();
    }

    /** Obtener jugador por nombre de la BBDD jugador ...5 lines */
    @GET
    @Path("/get")
    @Produces("application/json")
    @Consumes(MediaType.APPLICATION_JSON)
    public Jugador getJugadorByNombre(@QueryParam("nombre") String nombre){
        Jugador jugador = null;
        try {
            jugador = JugadorBBDD.selectJugador(nombre);
        } catch (Exception ex) {
            System.out.println("No se ha encontrado el jugador: " + ex.getMessage());
        }finally{
            return jugador;
        }
    }

    /** Crear jugador en la BBDD jugador ...5 lines */
    @POST
    @Path("/crear")
    public Jugador crearJugador(@QueryParam("nombre") String nombre) {
        Jugador jugador = new Jugador();
```

- **Palabras**: en esta clase se definen los diferentes tipos de acceso según la acción que queramos realizar sobre la tabla palabras. A continuación podemos ver un poco del código:

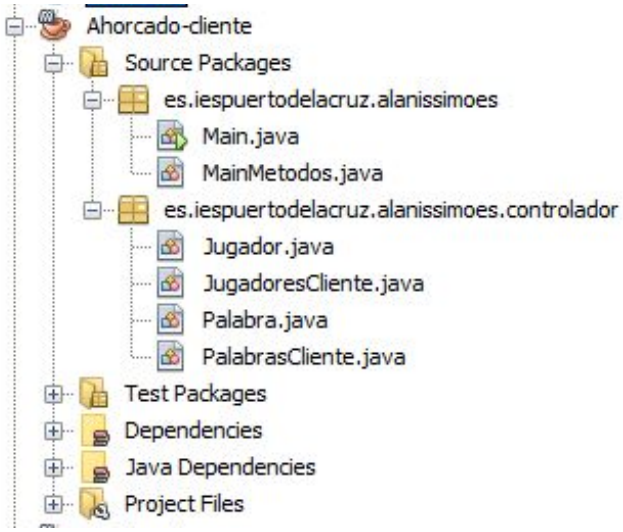
```java
@Path("/palabra")
public class Palabras {

    /**
     * Comprobar el funcionamiento del Servicio
     *
     * @return
     */
    @GET
    @Path("/info")
    public Response getInfo() {
        String output = "Servicio Palabras OK ";
        return Response.status(200).entity(output).build();
    }

    /**
     * Obtener palabra por palabra
     * @param palabra palabra
     * @return objeto de la palabra
     */
    @GET
    @Path("/get")
    @Produces("application/json")
    @Consumes(MediaType.APPLICATION_JSON)
    public Palabra getPalabraByPalabra(@QueryParam("p") String palabra){
        Palabra p = null;
        try {
            p = PalabraBBDD.selectPalabra(palabra);
        } catch (Exception ex) {
            System.out.println("No se ha encontrado la palabra: " + ex.getMessage());
        }finally{
            return p;
```

Cliente

A continuación se muestra la estructura de paquetes en el proyecto de Ahorcado cliente:



En el paquete 'alanissimoes' se encuentran todas las clases correspondientes a la ejecución del programa.¡, el 'main':

- **Main:** Clase donde se crea el método 'main'. Es aquí donde se ejecuta toda la lógica del juego (cuántos jugadores, menús, partidas, etc).
- **MainMetodos:** clase donde están todos los métodos que consideré "secundarios", para no llenar demasiado la clase 'Main' principal. Ejemplos: 'juegoIndividual', 'juegoDual', 'pintarAhorcado', etc.

En la carpeta 'controlador' están las clases donde se crean los 'clientes' necesarios para poder unir ambos proyectos:

- **Jugador:** clase del jugador. Tiene sus 'getters' y 'setters' correspondientes, a la par que su 'toString'.

- **JugadoresCliente:** clientes necesarios para acceder a los datos de la base de datos 'jugador':

```
public static Jugador getJugadorByNombre(String nombre) {
    //GET
    Client client = ClientBuilder.newBuilder()
            .register(JacksonFeature.class)
            .build();

    WebTarget target = client.target("http://localhost:8080/ahorcado/")
            .path("jugador/get").queryParam("nombre", nombre);
    Jugador jugador = target.request().get().readEntity(Jugador.class);
    return jugador;
}

public static String crearJugador(String nombre) {
    //POST
    Client client = ClientBuilder.newBuilder()
            .register(JacksonFeature.class)
            .build();

    WebTarget target = client.target("http://localhost:8080/ahorcado")
            .path("/jugador/crear");
    target = target.queryParam("nombre", nombre);
    String response = target.request().post(null, String.class);
    return response;
}


public static Response updatePuntosJugador(String nombre) {
    //PUT
    Client client = ClientBuilder.newBuilder()
            .register(JacksonFeature.class)
            .build();
    WebTarget target = client.target("http://localhost:8080/ahorcado");
    target = target.queryParam("nombre", nombre);

    Jugador jugador = target.path("/jugador/get")
            .queryParam("nombre", nombre)
            .request().get(Jugador.class);

    jugador.setPuntos(jugador.getPuntos()); //prueba

    target = client.target("http://localhost:8080/ahorcado")
            .path("/jugador/update").queryParam("nombre", nombre);
    Response response = target.request().put(Entity.json(jugador));
    return response;
}

public static void delJugadores() {
    //DELETE
    Client client = ClientBuilder.newBuilder()
            .register(JacksonFeature.class)
            .build();
    WebTarget target = client.target("http://localhost:8080/ahorcado")
            .path("/jugador/eliminar");

    target.request().delete();
}
```

- **Palabra:** clase de las palabras. Tiene sus 'getters' y 'setters' correspondientes, a la par que su 'toString'.

- **PalabrasCliente:** clientes necesarios para acceder a los datos de la base de datos 'palabras':

```
public static String getPalabraRandom() {
    //GET
    Client client = ClientBuilder.newBuilder()
            .register(JacksonFeature.class)
            .build();

    WebTarget target = client.target("http://localhost:8080/ahorcado/")
            .path("palabra/random");
    String response = target.request().get().readEntity(String.class);
    return response;
}

public static String insertarPalabra(String palabra) {
    //POST
    Client client = ClientBuilder.newBuilder()
            .register(JacksonFeature.class)
            .build();

    WebTarget target = client.target("http://localhost:8080/ahorcado")
            .path("/palabra/insert");
    target = target.queryParam("p", palabra);
    String response = target.request().post(null, String.class);
    return response;
}

 public static Response updateHaSidoUsada(String palabra) {
     //PUT
     Client client = ClientBuilder.newBuilder()
             .register(JacksonFeature.class)
             .build();
     WebTarget target = client.target("http://localhost:8080/ahorcado");

     Palabra p = target.path("/palabra/get")
             .queryParam("p", palabra)
             .request().get(Palabra.class);

     target = client.target("http://localhost:8080/ahorcado")
             .path("/palabra/update").queryParam("p", palabra);
     Response response = target.request().put(Entity.json(p));
     return response;

 }

 public static Response resetPalabras() {
     //PUT
     Client client = ClientBuilder.newBuilder()
             .register(JacksonFeature.class)
             .build();
     WebTarget target = client.target("http://localhost:8080/ahorcado");

     Palabra p = target.path("/palabra/get")
             .queryParam("p", "electroencefalograma")
             .request().get(Palabra.class);
     target = client.target("http://localhost:8080/ahorcado")
             .path("/palabra/reset");
     Response response = target.request().put(Entity.json(p));
     return response;
 }
```

Conclusión

Este trabajo considero que ha sido bastante laborioso y complejo, puesto que, debido a mi nivel de conocimientos, me ha supuesto un gran número de horas de trabajo. Sin embargo, estoy muy satisfecha de haber sido capaz de completarlo en su totalidad.

Me habría gustado que antes de que se mandara el proyecto, se hubiera practicado en clase con algunos ejemplos más sencillos, y/o que se hubiera explicado algo más del tema, sobre todo la parte del cliente.