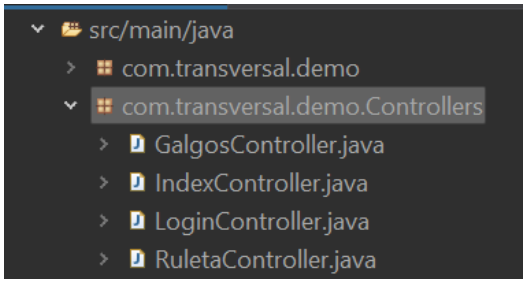
Alberto Pinilla y Manuel Palacios

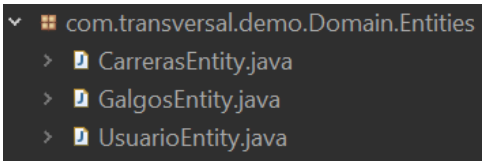
MANUAL DE USUARIO SERVIDOR, CLIENTE Y DESPLIEGUE

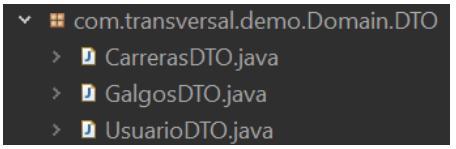
ÍNDICE

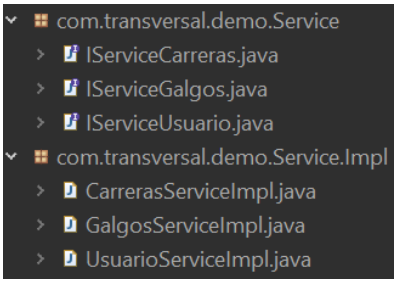
1- ELEMENTOS DEL PROYECTO

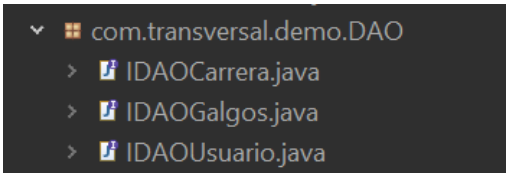
En el proyecto he creado un controlador para los Galgos, el Index, Login y Ruleta, cada uno se encarga de su respectivo HTML.
El controlador de los juegos es bastante similar ya que utilizamos prácticamente las mismas cosas como el control del Saldo.

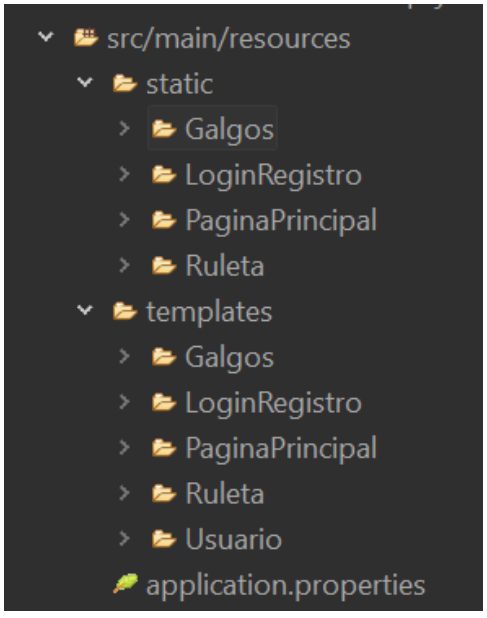












Aquí está la estructura de los HTMLs en el caso de el login tiene la funcionalidad de Login y Registro ya que está hecho de tal forma que se puede cambiar entre un formulario y otro dinámicamente por lo que en el controlador están ambas funcionalidad tanto la de añadir como la de loguearse.
Para que los Js y Css funciones tenemos que poner los archivos en la carpeta Static ya que si no Spring no los cargará y por supuesto debemos poner la ruta (relativa o absoluta) en el HTML.
Cabe recalcar que en cada HTML tenemos los th:text e IDs necesarios para recoger datos o colocarlos desde Spring.

2- Llamadas Ajax

Para las llamadas Ajax vamos a usar de ejemplo los Galgos, para recibir las listas de los galgos lo que he hecho ha sido además de llamar al método, he hecho un forEach que crea variables galgo y las rellena con los datos de cada galgo que vamos recibiendo de BBDD. Lo mismo sería para las carreras.
Cabe recalcar que estamos haciendo un Promise para que los datos lleguen y estén listos para ser utilizados, para despues llamar a un método que llame a estas funciones

```
function recibirGalgos() {
    return new Promise((resolve, reject) => {
        $.ajax({
            url: '/galgos/recuperarGalgos',
            type: 'GET',
            dataType: 'json',
            success: function(galgos) {
                console.log('GALGOS recogidos de BBDD correctamente');
                listaGalgos = []; // Limpiar la lista antes de agregar nuevos galgos
                galgos.forEach((item, i) => {
                    var galgo = new Galgo(
                        item.idGalgo,
                        item.nombre,
                        item.dorsal,
                        item.color,
                        item.ganancia,
                        item.aceleracion,
                        item.velocidad,
                        item.experiencia,
                        item.avance
                    );
                    listaGalgos.push(galgo);
                });
                console.log(listaGalgos);
                resolve(); // Resuelve la promesa cuando los galgos están listos
            },
            error: function(error) {
                console.error('Error al recoger GALGOS de BBDD: ', error);
                reject(error); // Rechaza la promesa si hay un error
            }
        });
    });
}
```

Mientras que para el saldo, lo que hacemos es rellenar la variable saldo del modelo para que contenga el saldo del usuario de la BBDD.

```
function recibirSaldo() {
    return new Promise((resolve, reject) => {
        $.ajax({
            url: '/galgos/recuperarSaldo',
            type: 'GET',
            dataType: 'json',
            success: function(saldo) {
                console.log('SALDO recogido de BBDD correctamente');
                resolve(saldo);
            },
            error: function(error) {
                console.error('Error al recoger el SALDO de BBDD: ', error);
                reject(error);
            }
        });
    });
}
```

Para enviar el saldo lo que hemos hecho ha sido, recoger tanto la variable saldo como la del id del Usuario que hemos pasado por parámetros al HTML de Galgos, para que dentro del método en Spring nos busque a qué usuario hay que actualizarle el saldo.

```
function enviarSaldo(idUsuario, saldo) {
    var idUsuario = document.getElementById("idUsuario").textContent;
    var saldo = document.getElementById("lblsaldo").textContent;

    console.log("VOY A ENVIAR EL ID: " + idUsuario);
    console.log("VOY A ENVIAR EL SALDO: " + saldo);

    $.ajax({
        url: '/galgos/actualizarSaldo',
        type: 'POST',
        data: JSON.stringify({ idUsuario: idUsuario, saldo: saldo }),
        contentType: 'application/json; charset=utf-8',
        dataType: 'json',
        success: function(response) {
            console.log('Saldo y usuario actualizados correctamente en el servidor:', response);
        },
        error: function(error) {
            console.error('Error al actualizar el saldo y usuario en el servidor:', error);
        }
    });
}
```

Finalmente para que el JS funcione tenemos que llamar a las llamadas Ajax que hemos hecho anteriormente para ello lo que hacemos es recibir los galgos y carreras, para después llamar a controller.init y selección de galgos. Esto lo que hace es que el programa reciba los datos y los utilice.
Para enviar el saldo a BBDD lo que hacemos en llamar al método en puntos específicos del código tanto al ingresar como terminar la carrera para que en caso de apostar o salirse el saldo no de problemas después.

```
// Modificar recibirDatos para que espere a ambas promesas
function recibirDatos() {
    return Promise.all([recibirGalgos(), recibirCarreras()]);
}

// Modificar la inicialización para que espere los datos de Ajax
$(document).ready(function() {
    recibirDatos().then(() => {
        console.log('Todas las listas han sido cargadas correctamente');
        controller.init();
        controller.seleccionGalgos();
        console.log(model.carreraSeleccionada);
        var saldoInicial = document.getElementById("lblsaldo").textContent;
        model.saldo = Number(saldoInicial) || 0;
        console.log("Saldo al entrar " + model.saldo);
    }).catch(error => {
        console.error('Hubo un error al cargar los datos', error);
    });
});
```

3- Funcionamiento en Spring

Ahora pasaré a hablar sobre el funcionamiento de los controladores de manera general ya que en el caso de la ruleta y los galgos es prácticamente la misma cambiando nombre y demás.
Primero tenemos un @GetMapping para mostrar el HTML del juego, así como un requestParam para recibir el id de Usuario de nuestro Jugador para poder cargar su información de la BBDD, y se la pasamos al HTML para poder trabajar después con esto.

A continuación tenemos 2 métodos que recogen los datos necesarios para el juego en este caso son los galgos y las carreras, los cuales son llamados mediante Ajax desde nuestro JS y estos nos devuelven una lista de Carreras y otra de Galgos, por supuesto tendremos que realizar esta funcionalidad en el servicio, pero básicamente es un findAll de la tabla en cuestión en cada metodo.

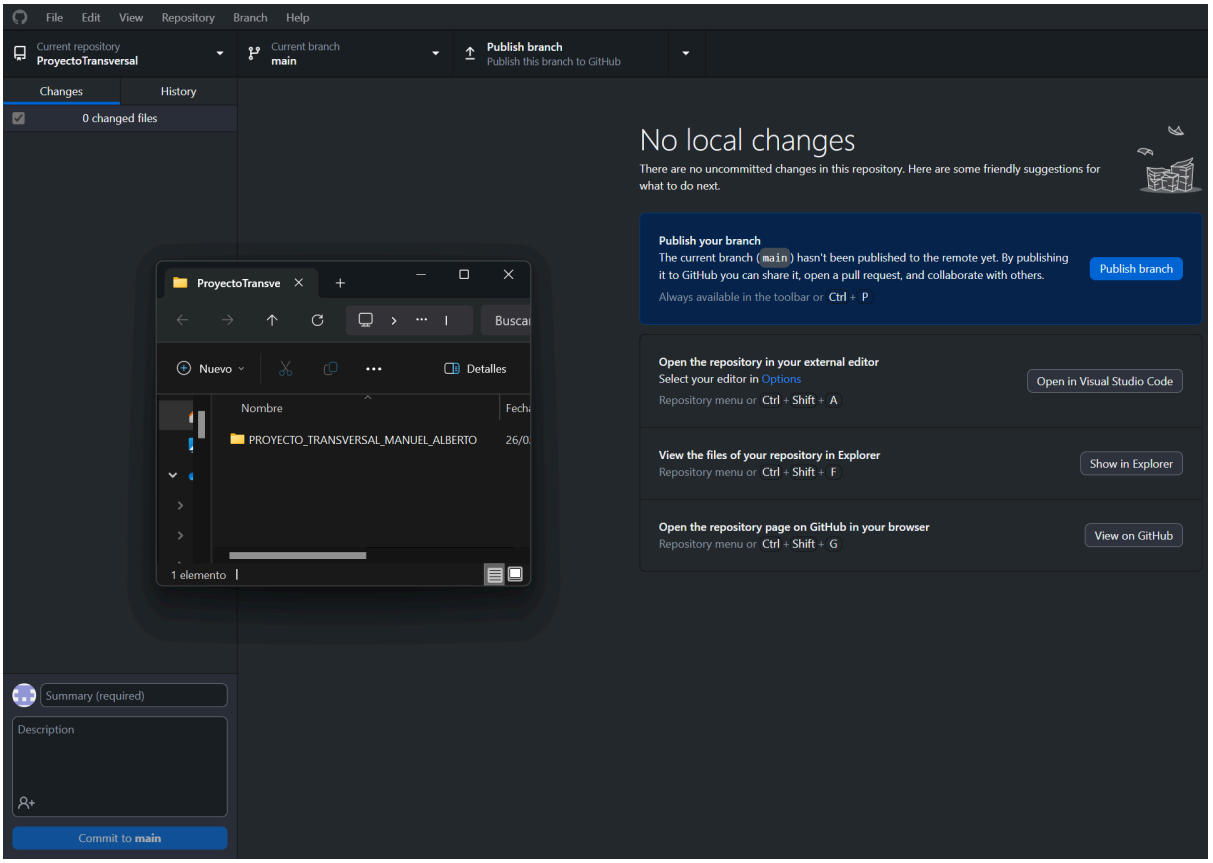
```
@GetMapping(value="verGalgos")
public String verGalgos(Model modelo, @RequestParam(name = "idUsuario") String idUsuario) {
    modelo.addAttribute("titulo","Carrera de Galgos");
    Integer idUsuarioFix = Integer.parseInt(idUsuario);
    Integer saldo = usuarioService.buscarSaldoUsuario(idUsuarioFix);
    System.out.println("TENGO EL SALDO " + saldo + "€");
    modelo.addAttribute("idUsuario" ,idUsuarioFix);
    modelo.addAttribute("lblSaldo" ,saldo);
    return "Galgos/Galgos";
}

@GetMapping(value = "/recuperarGalgos")
public ResponseEntity<List<GalgosDTO>> recuperarTodosGalgos() {
    try {
        List<GalgosDTO> galgos = galgosService.buscarGalgos();
        return new ResponseEntity<>(galgos, HttpStatus.OK);
    } catch (Exception e) {
        System.out.println("Error en buscarTodosGalgos en GalgosController" + e.getMessage(
        return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
    }
}

@GetMapping(value = "/recuperarCarreras")
public ResponseEntity<List<CarrerasDTO>> recuperarTodosCarreras() {
    try {
        List<CarrerasDTO> carreras = carrerasService.buscarCarreras();
        return new ResponseEntity<>(carreras, HttpStatus.OK);
    } catch (Exception e) {
        System.out.println("Error en buscarTodosCarreras en GalgosController"+ e.getMessage
        return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

4- DESPLIEGUE DE LA APLICACIÓN

Hemos tenido problemas para desplegar la aplicación ya que Tomcat no soportaba la subida del proyecto por lo hemos optado por subirlo a GitHub en su lugar como hemos hecho en las prácticas de Despliegue aunque no podamos ver la vista previa de la web mediante GitHub.
Aquí tenemos la subida del Proyecto de Spring:



Aquí tenemos el proyecto dentro del repositorio en la nube: