

# INFORME TFG

## CFGs DAW



ReproBeasts VIII ¡Destacado!

50.00€ ~~100.00€~~

Detalle



Salón de Ginebra 2022 ¡Destacado!

400.00€ ~~450.00€~~

Detalle



Classic Rules

40.00€ ~~60.00€~~

Detalle

**Miguel Ángel del Rincón Manrique**

# ÍNDICE

<b>1. Introducción .....</b>	<b>2</b>
<b>2. Asignaturas.....</b>	<b>3</b>
<b>3. Tecnologías.....</b>	<b>4</b>
1. Front.....	4
2. Back .....	5
<b>4. Herramientas .....</b>	<b>5</b>
<b>5. Objetivos.....</b>	<b>6</b>
<b>6. Fases del proyecto.....</b>	<b>7</b>

Fase 0: Base de Datos .....	7
Fase 1: Gestión de eventos y noticias - parte web .....	9
Fase 2: Servicio REST .....	10
Fase 3: Diseño de parte front .....	10
Fase 4: Inclusión de Spring Security + Reservas + Fotos.....	10
Resultado Final: Explicación .....	12
Parte 1: Paquetes Java .....	12
Parte 2: Vistas JSP.....	26
<b>7. Conclusiones y puntos de mejora .....</b>	<b>35</b>
 <b>8. Bibliografía.....</b>	 <b>36</b>
 <b>Anexo 1: Guía Servicio REST .....</b>	 <b>37</b>

## **1. Introducción**

Nótese que el proyecto se encuentra alojado en:

[https://github.com/miguelang611/TFG\\_MiguelRM](https://github.com/miguelang611/TFG_MiguelRM)

Así mismo, se dispone dentro de dicho repositorio de una versión en vídeo del presente informe, detallando la estructura de la aplicación (parte 1), así como el funcionamiento de la parte web (parte 2) y del servicio REST (parte 3)

El proyecto a desarrollar va a estar relacionado con el mundo del motor, y contará con varias partes, enfocadas a distintos grupos de usuarios; y con caracteres más estáticos/dinámicos según cada caso.

La web se compone de 3 partes imprescindibles:

1. Noticias → divididas en categorías → rol público / redactor
2. Eventos → divididos por tipos → rol público / organizador
3. Reservas → permitimos que se reserven eventos → rol cliente (registrado) + rol administrador

El motivo principal por el que se ha elegido esta amalgama de bloques dentro del proyecto son las amplias necesidades de los usuarios del mundo del motor. Existen plataformas que tienen dichos conceptos por separado, pero ninguna ofrece una amalgama de todas las propuestas: noticias, eventos importantes y reservas en una sola web.

Considero que la integración de los 3 bloques puede generar sinergias en la empresa, de modo que, se potencien entre ellas, dando lugar a un mayor conocimiento de la marca, y por ende, un mayor crecimiento de nuestra plataforma.

Desarrollando los anteriores bloques:

Por el lado del bloque 1, será el contenido de la web creado por nosotros mismos, es decir, será contenido que sea interesante para los aficionados al mundo del motor, y también a aquellos ajenos que, por ejemplo, están en busca de coche y necesitan consejo acerca de qué modelo puedes convenirles más.

Por otro lado, el segundo bloque, puramente orientado a los aficionados al mundo del motor, permitirá asistir a eventos patrocinados por nuestra marca y también de organizadores externos → el objetivo de estos eventos no es la ganancia de dinero directa, sino la publicidad indirecta en estos eventos.

Por último, el tercer bloque, en línea con el anterior, permitirá que los usuarios / clientes puedan reservar dichos eventos, adquiriendo entradas.

Las personas usuarias de la aplicación serán en principio:

- 🚦 No aficionados al mundo del motor → buscan opiniones de un coche, consejos para su cuidado → se centran en la sección noticias
- 🚦 Aficionados al mundo del motor → pueden buscar alguna de las anteriores, así como asistir a eventos
- 🚦 Circuitos y otros colaboradores → indispensables para realizar vídeo-pruebas de coches, así como para organizar los eventos.
- 🚦 Redactores → indispensables para poder ofrecer contenido relevante para el usuario.

## **2. Asignaturas**

1. Programación → asignatura base para todo lo demás
2. Base de Datos → necesitaremos de las mismas para los 3 bloques: noticias, comparador de coches, eventos.
3. Sistemas Informáticos → lo necesitaremos para crear una máquina virtual servidor, base para la parte de Despliegue de Aplicaciones Web.
4. Lenguaje de marcas → parte de la vista de la web
5. Entornos de desarrollo → realización de diagramas, utilización de IDEs para facilitar la tarea de programación, así como el uso de Git.

6. Inglés → parte de la web se ofertará en ambos idiomas
7. Desarrollo Web Entorno Cliente → enfocado a JavaScript, para que la web pueda ser dinámica, así como la realización de validaciones y similares.
8. Desarrollo Web Entorno Servidor → enfocado a Java Enterprise, se utilizará principalmente para realizar conexiones a la Base de Datos y a la generación de parte del contenido.
9. Diseño de Interfaces Web → enfocado a la parte de CSS, para el diseño agradable de la UI, así como un cierto toque de experiencia de usuario UX
10. Despliegue de Aplicaciones Web → necesario para la preparación e instalación de Apache, Tomcat, UFW, DNS, etc.

### **3. Tecnologías**

En línea con lo anteriormente desarrollado, en un principio, utilizaremos:

#### **1. Front**

1. **HTML5** → utilizaremos las ventajas que nos ofrece HTML5 de cara a formularios y verificaciones sin necesidad de JavaScript, y también de cara a tener organizadas las partes de la página con una optimización mayor (main, aside, etc) de cara a posicionamiento SEO
2. **JavaScript** → el uso de JavaScript ha sido más bien reducido, enfocado de manera principal a la asistencia en las verificaciones de formularios, y de manera interna, a apoyar a Bootstrap
3. **CSS3 + Bootstrap** → hemos utilizado una plantilla de Bootstrap 4 para la parte del cliente, integrándola con la parte de gestión, que se realice en un primer momento en Bootstrap 5 sin plantilla alguna.

## **2. Back**

**Spring Boot** → ha sido la tecnología vehículo del proyecto, con inclusión de varias de sus variantes:

- ✓ **JPA** → parte indispensable de la aplicación, base del proyecto, realizaremos conexión a una base de datos MySQL/MariaDB, de dónde se obtendrá la mayoría del contenido.
- ✓ **WEB** → utilizada para la generación de las vistas y los correspondientes controladores que se encargarán de tramitar las peticiones de la aplicación.
- ✓ **REST** → utilizaremos peticiones GET, POST, PUT, DELETE que nos permitirán dar de alta, consultar, modificar y eliminar eventos / tipos (por el momento sólo se ha implementado dicha parte).
- ✓ **Security** → toda la aplicación queda securizada bajo la adicción de la capa de seguridad ofrecida por Spring. Realizará autenticación contra una base de datos de usuarios, donde además cada usuario tendrá un tipo de perfil que le otorga un rol determinado con sus correspondientes permisos.

**MariaDB** → hemos optado por esta solución de base de datos debido a su carácter libre y su compatibilidad cuasi total con la tecnología MySQL.

El motivo por el que me he decantado por MySQL en vez de por otras alternativas como Oracle PL/SQL es su mayor simpleza y ligereza, al no necesitar una base de datos de una complejidad muy elevada.

## **4. Herramientas**

1. **Eclipse** → utilizado como IDE principal de la aplicación desde donde se ejecuta la aplicación Spring Boot.
2. **VS Code** → utilizado de manera auxiliar para realizar algunas pruebas de front: HTML5/CSS3/JS y BootStrap.
3. **Spring Intialitzer** → utilizado para la generación de un proyecto inicial de Spring Boot

4. **Tomcat** → utilizado como servidor de aplicaciones, sobre él correrá toda la aplicación, a excepción de la BBDD:
5. **MySQL WorkBench** → utilizaremos WorkBench de manera principal para la generación de los esquemas E-R de nuestra Base de Datos.
6. **HeidiSQL** → utilizado para la ejecución de scripts, pruebas y consultas de la Base de Datos de manera directa, sin interactuar con nuestra aplicación.
7. **Postman** → utilizado como medio para lanzar peticiones REST hacia el servicio generado.

## **5. Objetivos**

El objetivo principal del Proyecto es la generación de una aplicación completa que sea capaz de cumplir con los requisitos de lo que sería una solución a ofrecer a un posible cliente, o incluso un proyecto realizado de manera autónoma por nuestra parte.

Esto supone por un lado, desarrollar una aplicación:

1. Atractiva para el usuario → está comprobado que los usuarios que entran a una web con un aspecto descuidado y desordenado genera con alta probabilidad un rebote: una visita perdida.
2. Funcional para el usuario → buscamos permitir que el usuario pueda navegar de una manera cómoda y sencilla, en la que nunca tenga una sensación de no saber “dónde está”.
3. Afable para los colaboradores → buscamos también que nuestros colaboradores: organizadores y redactores se sientan cómodos con la web y con su sistema de gestión. Por ello, buscaremos reducir el número de páginas a ofrecerles, centrándonos en ofrecer en una sola página el compendio de acciones que pueden realizar.
4. Velocidad y fiabilidad → buscaremos una app que sea rápida, pero teniendo siempre como objeto principal la fiabilidad y confiabilidad:
  - a. No permitir que el usuario “nos la juegue” → si bien no es común, tenemos usuarios que pueden realizar acciones dañinas para la aplicación → ej: mandan la orden de cancelar una reserva que no es suya cambiando el id de la URL



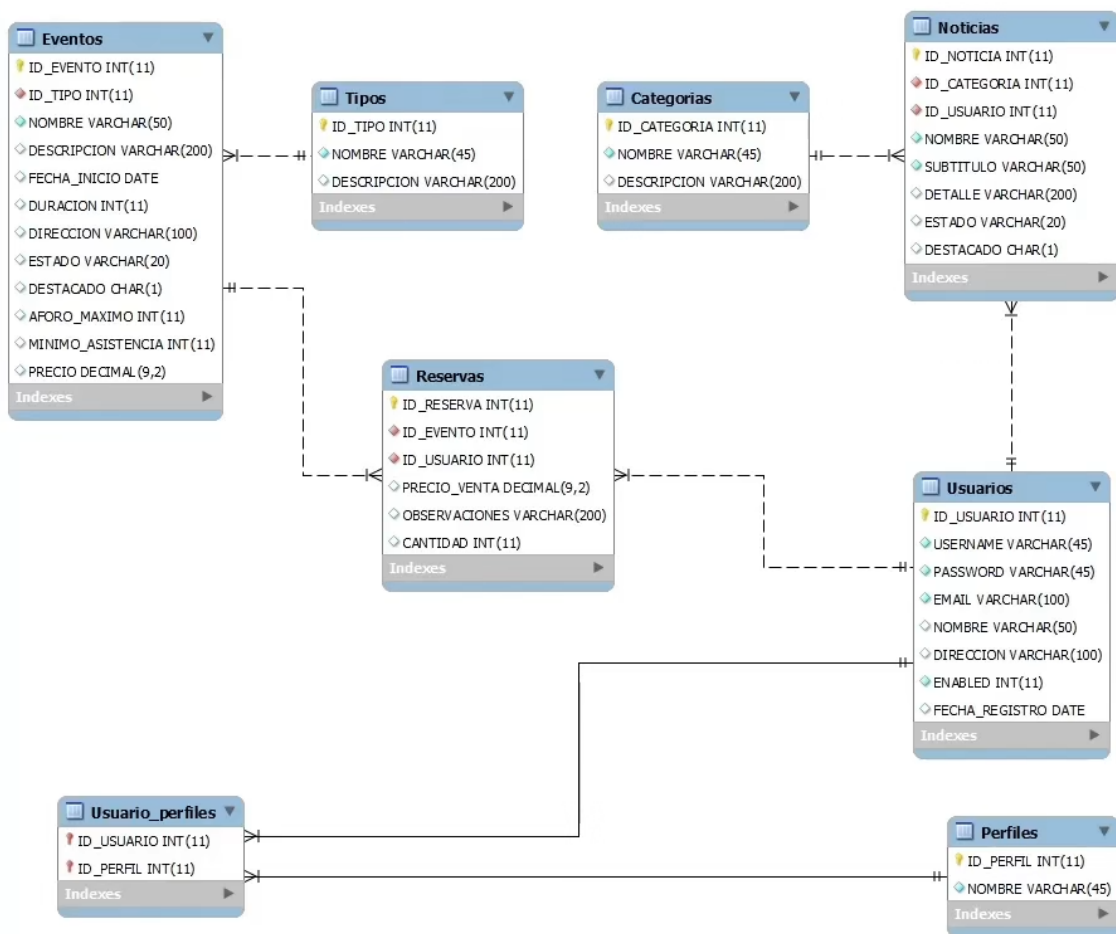
- b. Asegurar la integridad → comprobaremos en la parte de cliente las introducciones vía HTML5, así como comprobar la viabilidad del recurso solicitado. Esto es especialmente importante de cara al servicio REST, donde nos pueden lanzar peticiones incongruentes con mayor facilidad.
- 5. Informar con detalle → evitaremos proporcionar al usuario mensajes genéricos, siempre mostraremos detalle de qué ha sido exitoso o qué ha fallado.

## 6. Fases del proyecto

El proyecto se ha dividido en varias fases:

### Fase 0: Base de Datos

El planteamiento original de la Base de Datos era el siguiente:



De manera esquemática:

- 1 Evento pertenece a 1 tipo
- 1 tipo puede tener varios eventos
- 1 evento puede ser reservado por 1 o más usuarios, que tendrá 1 o más perfiles
- 1 noticia pertenece a 1 categoría
- 1 noticia pertenece a un usuario, que tendrá 1 o más perfiles
- 1 categoría tiene 1 o más noticias

Como datos principales, las claves primarias y ajenas de las tablas serían:

### **Eventos**

- ✓ ID Evento (Primary Key)
- ✓ ID Tipo (Foreign Key)

### **Tipos**

- ✓ ID Tipo (Primary Key)

### **Reservas**

- ✓ ID Reserva (Primary Key)
- ✓ ID Evento (Foreign Key)
- ✓ ID Usuario (Foreign Key)

### **Noticias**

- ✓ ID Noticia (Primary Key)
- ✓ ID Categoría (Foreign Key)
- ✓ ID Usuario (Foreign Key)

### **Categorías**

- ✓ ID Categoría (Primary Key)

## **Usuarios**

- ✓ ID Usuario (Primary Key)

**Usuario Perfiles** (tabla intermedia) → ID Usuario + ID Perfil

## **Perfiles**

- ✓ ID Perfil (Primary Key)

\*Es reseñable que esta Base de Datos se modificaría con posterioridad\*

## **Fase 1: Gestión de eventos y noticias - parte web**

El proyecto comenzó de manera inicial, tal y como se pudo ver en la entrega intermedia con la parte de gestión de eventos.

Permitíamos que el organizador/redactor (que era cualquiera en ese momento) pudiera:

1. Dar de alta eventos/noticias o tipos/categorías → proporcionando un formulario acorde
2. Modificar uno existente vía formulario → recuperábamos los datos del existente y mostrábamos un formulario
3. Cambiar estado (activo/cancelado) → sólo para eventos
4. Destacar / no destacar → restringiendo destacar eventos cancelado / sin restricción para noticias
5. Eliminar → para dar de baja dicha noticia/evento /tipo/categoría de manera total

Así como:

1. Ver todos los eventos / noticias / tipos / categorías
2. Ver eventos por tipo / noticias por categoría
3. Ver eventos/noticias por destacado/no destacado
4. Ver noticias por activo/cancelado

## **Fase 2: Servicio REST**

Una vez conseguida la parte de gestión de la aplicación decidí realizar un servicio REST que pudiera realizar las acciones que comentaba con anterioridad, pero enfocado a su uso desde Postman.

Para más información, consultar el Anexo 1: Uso de servicio REST.

## **Fase 3: Diseño de parte front**

Una vez “dominada” la parte de gestión, decidí embarcarme en la inclusión de una parte orientada al cliente final, al usuario, en la que ofrecer una versión pública de la aplicación.

Esta parte pública se componía originalmente de:

1. Ver todos los eventos / noticias / tipos / categorías
2. Ver eventos por tipo / noticias por categoría
3. Ver eventos/noticias por destacado/no destacado
4. Ver noticias por activo/cancelado

Sin embargo, carecía de ningún tipo de autenticación ni de gestión del tercer punto: las reservas.

Así mismo, se diseñó primeramente de manera única la parte de mostrar eventos / noticias, sin el detalle, cosa que se agregaría a posteriori.

## **Fase 4: Inclusión de Spring Security + Reservas + Fotos**

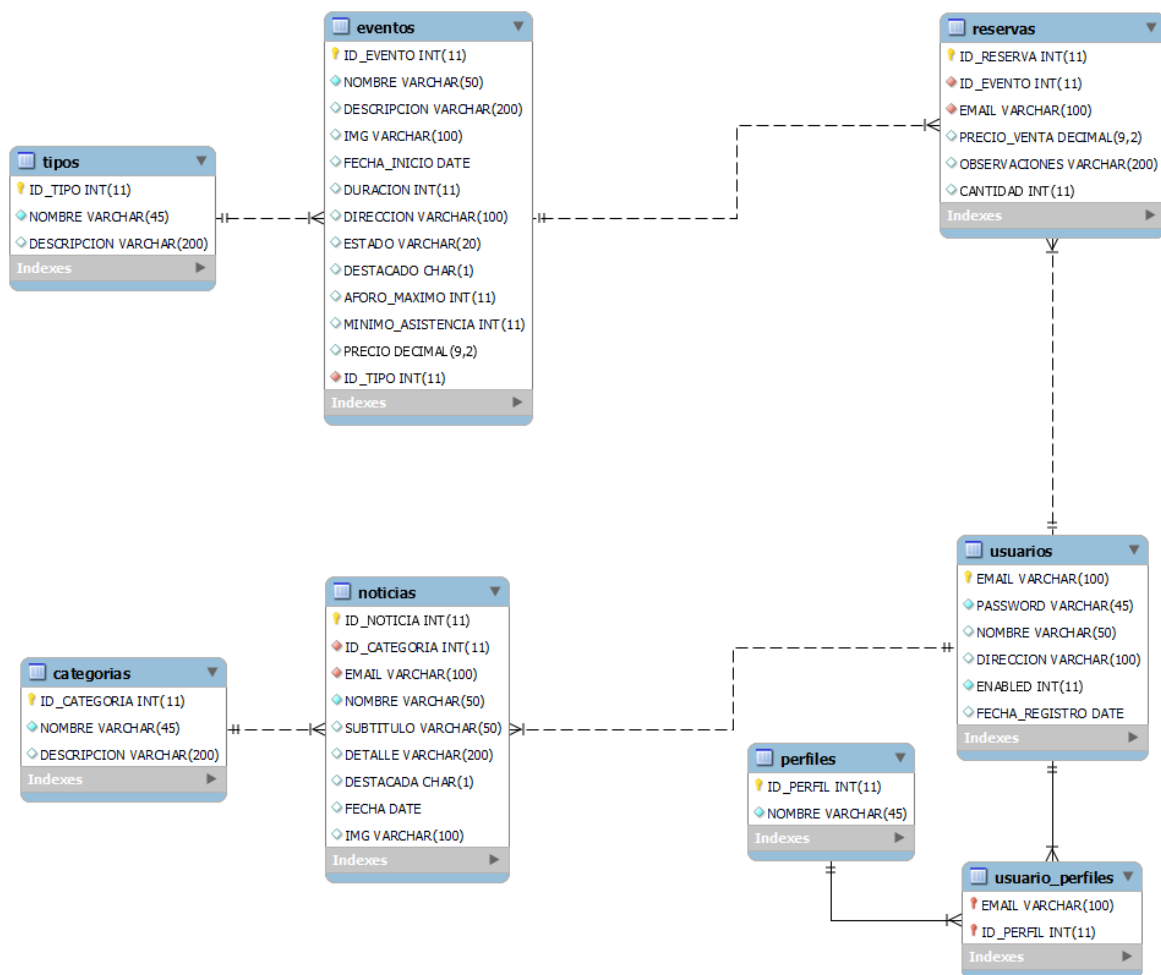
Para poder realizar reservas necesitábamos tener una parte de seguridad: saber quién estaba reservando. De esta forma, era el momento de incluir Spring Security.

En este punto, fue necesario modificar de manera integral la aplicación, al ser preciso modificar la Base de Datos.

Las modificaciones consistieron:

1. Inclusión de nuevos campos en noticias y eventos → creamos un campo img para poder incluir ruta a la imagen asociada a ese evento / noticia
2. Modificación de la tabla Usuarios → nos deshacemos de la Primary Key ID Usuario, para ser reemplazada por email, que hará ahora las veces de clave primaria. Esto se debe a que Spring Security necesita trabajar con username/email para funcionar correctamente
3. Modificación de Usuario\_Perfiles → reemplazar ID\_Usuario por Email

Así pues, el esquema final de Base de Datos pasó a ser el siguiente:



Nótese que las relaciones y lógica de la misma se han mantenido invariables.

La inclusión de Spring Security nos permitió en un principio poder saber quién estaba gestionando cada cosa, de modo que las navbar de gestión se mostraran de manera automática según el tipo de usuario logeado.

De cara a permitir que los clientes pudieran realizar reservas, procedí a crear un formulario de registro, que se encargaría de permitir que un usuario se diera de alta. Por “debajo”; asignaríamos siempre a esos usuarios el perfil/rol de usuario.

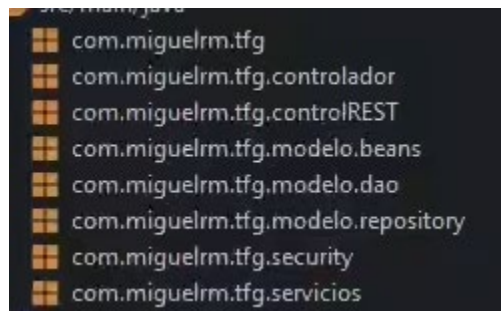
Resuelto esto, desde la parte pública de la página, pasaríamos a permitir la realización de reservas, únicamente de evento en evento.

Tras la reserva, llevaríamos al cliente a una página donde consultar sus reservas.

## **Resultado Final: Explicación**

### **Parte 1: Paquetes Java**

En cuanto a la parte de Java, tendremos varios paquetes:



1. **Security** → solamente contiene una clase, llamada ConfigSecurity. En ella:

```
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    //Le decimos que autentique contra el datasource de la BD
    auth.jdbcAuthentication().dataSource(dataSource)
    //Hacemos la select de SQL puro
    .usersByUsernameQuery("select email, password, enabled from Usuarios where email=?")
    //Qué tienes que recoger de aquí
    .authoritiesByUsernameQuery("select u.email, p.nombre from Usuario_Perfiles up " + "inner join Usuarios u on u.email = up.email " +
    "inner join Perfiles p on p.id_perfil = up.id_Perfil " + "where u.email = ?");
}
```

Indicamos como tiene que realizar la comprobación de login a la BD.

Y también,

```

@Override
public void configure(HttpSecurity http) throws Exception {
    http
        .csrf().disable()
        .authorizeRequests().antMatchers("/bootstrap/**", "/images/**", "/tinymce/**", "/logos/**").permitAll()

        // Asignar permisos a URLs por ROLES
        .antMatchers("/tipos/**").hasAnyAuthority("organizador", "admin")
        .antMatchers("/gestion/eventos/**").hasAnyAuthority("organizador", "admin")
        .antMatchers("/gestion/noticias/**").hasAnyAuthority("redactor", "admin")
        .antMatchers("/categorias/**").hasAnyAuthority("redactor", "admin")

        // Las vistas públicas no requieren autenticación
        .antMatchers("/",
            "/login",
            "/eventosREST/**",
            "/tiposREST/**",
            "/registro",
            "/search",
            "/eventos/**",
            "/noticias/**",
            "/doLogin",
            "/resources/**"
        ).permitAll()

        // Todas las demás URLs de la Aplicación requieren autenticación
        .anyRequest().authenticated()

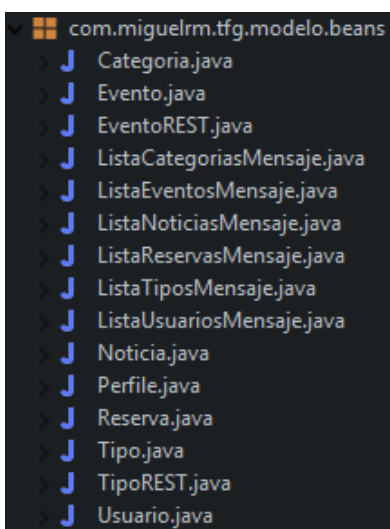
        .and().formLogin().loginPage("/login")
            .defaultSuccessUrl("/doLogin", true)
        .and().logout();
}

```

Indicamos los accesos permitidos a usuarios públicos, a usuarios registrados, a organizadores y redactores, así como el formulario de login y la tarea a ejecutar posterior.

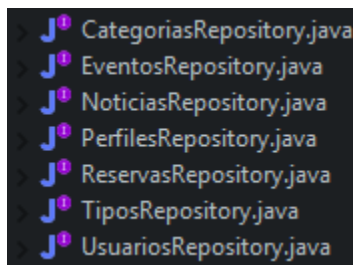
Permitiremos acceso libre a url de /eventos, /noticias, /eventosREST, /noticiasREST (el servicio REST no será autenticado), /login, /resources y /doLogin serán las únicas sin autenticación.

2. **Beans** → paquete que contiene todos los beans asociados a la aplicación. Tenemos tres grandes grupos:



- a. Categoría/Evento/Tipo/Noticia “puro” → es donde tenemos las variables con su correspondencia en BBDD directa y sus join (si es el caso), así como constructores, getter/setter, hashCode/equals y toString
- b. ListaEventosMensaje/ListaReservasMensaje → son clases que se basan en la creación de un ArrayList de evento / reservas, etc y un String de mensaje → las utilizamos para devolver respuestas desde los métodos de los daos con mensajes custom y ser universales entre web/REST
- c. EventoREST / TipoREST → son modificaciones de los beans puros donde se han eliminado parte de tipos/eventos para evitar referencias cruzadas a la hora de devolver resultados en el servicio.

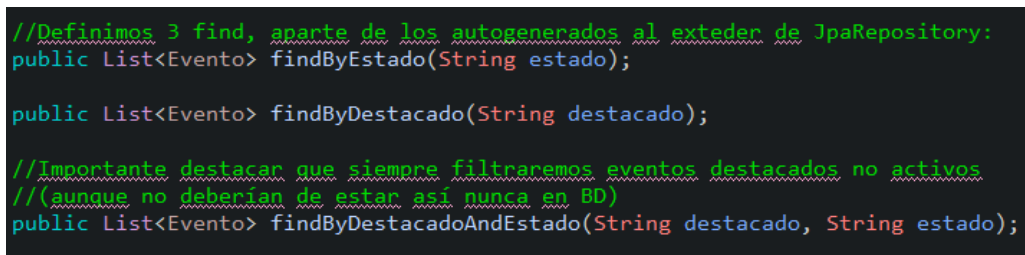
3. **Repository** → paquete para las peticiones a realizar desde los DAOS:



```
Ji CategoriasRepository.java
Ji EventosRepository.java
Ji NoticiasRepository.java
Ji PerfilesRepository.java
Ji ReservasRepository.java
Ji TiposRepository.java
Ji UsuariosRepository.java
```

Nótese que extenderán de JpaRepository, de modo que tendremos un compendio de acciones por defecto como findAll() que no será necesario desarrollar.

Sin embargo, sí tendremos que desarrollar algunos:



```
//Definimos 3 find, aparte de los autogenerados al extender de JpaRepository:
public List<Evento> findByEstado(String estado);

public List<Evento> findByDestacado(String destacado);

//Importante destacar que siempre filtraremos eventos destacados no activos
//(aunque no deberían de estar así nunca en BD)
public List<Evento> findByDestacadoAndEstado(String destacado, String estado);
```

- a. findByEstado / findByDestacado → buscará por el valor que le pasemos en la BBDD
- b. findByDestacadoAndEstado → nos permite hacer una select con doble where

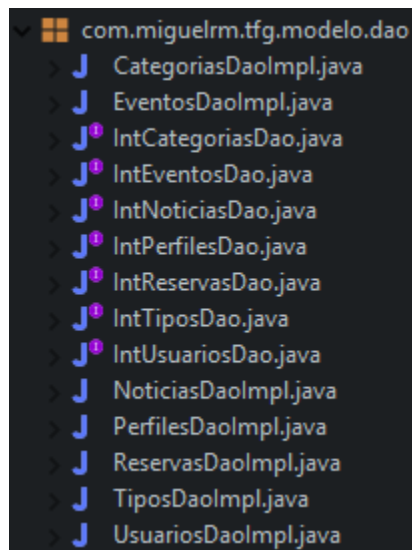


```
@Query("select e from Evento e where e.tipo.idTipo = ?1")  
public List<Evento> findByTipo(int idTipo);
```

- c. `findByTipo` → pasando un `idTipo` nos buscará en la BBDD → es necesario crear una query manual de tipo JPQL, de modo que recuperemos eventos donde `evento.tipo.idTipo` sea X

(De este estilo lo utilizaremos en reservas, para recuperar por email o por evento y en noticias para recuperar por categoría)

4. **DAOS** → paquete para las peticiones a realizar desde los DAOS, tendremos interfaces y clases que implementarán las interfaces correspondientes:



Vamos a ver la clase `DaoImpl` de eventos, ya que constituye la base de todas las demás, al ser la primera que realicé, y la que más tiempo me conllevó su diseño.

```
@Autowired  
EventosRepository miEventosRepo;  
  
@Autowired  
TiposRepository miTiposRepo;
```

- 1) Primeramente, crearé dos variables, de `EventosRepository` y de `TiposRepository`, las cuales serán `Autowired` (etiqueta de Spring Boot para autoconectarlas).

2) Destacaré el método devuelveByDestacado(String mensaje):

```
@Override
public ListaEventosMensaje devuelveByDestacado(String destacado) {
    List<Evento> miListaEventos = null;
    String mensaje = null;
    try {
        if (destacado.equals("s")) {
            // los eventos destacados tienen que estar activos obligatoriamente
            miListaEventos = miEventosRepo.findByDestacadoAndEstado(destacado, "activo");
        } else {
            // Pero los no destacados no tienen por qué:
            miListaEventos = miEventosRepo.findByDestacado(destacado);
        }

        } catch (Exception e) {
            mensaje = "Error -> fallo de conexión a la BBDD";
            e.printStackTrace();
        }
        ListaEventosMensaje miListaEventosMensaje = new ListaEventosMensaje(miListaEventos, mensaje);
        return miListaEventosMensaje;
    }
}
```

Como explico, cuando mandemos buscar destacados, forzaremos que a su vez estén activos; pero cuando sean no destacados, no lo forzaremos, ya que no destacados pueden ser activos y no activos (limpiaremos posteriormente desde el controlador si son cancelados y estamos en la parte pública).

En este, y en la mayoría de métodos de devolver, se devuelve una variable del tipo compuesto ListaEventosMensaje → esto nos permite a la vez devolver un mensaje custom de fallo y la propia lista con un solo método.

3) El resto de métodos de búsqueda son más simples y sólo ejecutan la llamada al método correspondiente del Repository:

```
@Override
public ListaEventosMensaje devuelveTodos() {
    List<Evento> miListaEventos = null;
    String mensaje = null;
    try {
        miListaEventos = miEventosRepo.findAll();
    } catch (Exception e) {
        mensaje = "Error -> fallo de conexión a la BBDD";
        e.printStackTrace();
    }
    ListaEventosMensaje miListaEventosMensaje = new ListaEventosMensaje(miListaEventos, mensaje);
    return miListaEventosMensaje;
}
```

## 4) Método que busca por palabra:

```

@Override
public ListaEventosMensaje devuelveByPalabra(String palabra) {
    List<Evento> milistaEventos = null;
    String mensaje = null;
    try {

        List<Evento> milistaEvNombre = miEventosRepo.findByNombreContains(palabra);
        if (milistaEvNombre != null) {
            List<Evento> milistaEvDescripcion = miEventosRepo.findByDescripcionContains(palabra);
            // Si no había encontrado nada por nombre, la lista lo que encuentre por
            // descripción
            if (milistaEvNombre.size() == 0) {
                milistaEventos = milistaEvDescripcion;

                // PERO, si la lista obtenida por nombre tiene algún dato,
                // leeremos la lista obtenida por descripción
                // y comprobaremos objeto a objeto contra la otra lista
                // Si no lo contiene, añadimos dicho objeto a la lista original
            } else {
                milistaEventos = milistaEvNombre;
                for (int i = 0; i < milistaEvDescripcion.size(); i++) {
                    Evento miEvento = milistaEvDescripcion.get(i);
                    if (!milistaEventos.contains(miEvento)) {
                        milistaEventos.add(miEvento);
                    }
                }
            }
        }
    } catch (Exception e) {
        mensaje = "Error de conexión a la BBDD";
        e.printStackTrace();
    }
    ListaEventosMensaje milistaEventosMensaje = new ListaEventosMensaje(milistaEventos, mensaje);
    return milistaEventosMensaje;
}

```

La parte más reseñable de este método es que realiza una doble consulta: en nombre y en descripción → sin embargo, no tendría sentido que un evento que contuviera la misma palabra en ambos se nos devolviera dos veces. De esta forma, recorreremos la lista, y sólo si no está presente el evento lo añadiremos.

## 5) Método insertarEvento:

```

@Override
public String insertarEvento(Evento miEvento) {
    String insertarRdo = "";
    try {

        String msgCheckID = devuelvePorId(miEvento.getIdEvento()).getMensaje();
        System.out.println(msgCheckID);

        // Hay un evento con ese ID
        if (msgCheckID == null) {
            insertarRdo = "Error -> No se puede insertar un evento con dicho id, ya que en la BBDD hay otro evento con dicho ID. ¡Verifique!";
        }

        // El mensaje no viene vacío pero la lista lo está --> no hay eventos con ese id
        if (msgCheckID != null && msgCheckID.contains("Error") && !msgCheckID.contains("Error -> Evento con ID 0 no encontrado")) {
            List<Evento> milistaTotal = devuelveTodos().getListasEventos();
            String mensaje = devuelveTodos().getMensaje();
            // Si no hay error
            if (mensaje == null) {
                // Y el evento no está en la lista de eventos
                // (compara por debajo, pero excluye tipo, estado, destacado e id)
                if (!milistaTotal.contains(miEvento)) {
                    // Si no está, guardamos
                    miEventosRepo.save(miEvento);
                    insertarRdo = "El evento " + miEvento.getNombre() + " se ha insertado correctamente";
                }
                // En caso contrario, no hacemos la inserción
            } else {
                insertarRdo = "Error -> No se puede insertar un evento que ya existe. Si lo desea, puede editarlo";
            }
        } else {
            insertarRdo = mensaje;
        }
    }
}

```

```

        if(msgCheckID != null && msgCheckID.contains("Error -> Evento con ID 0 no encontrado")) {

            miEventosRepo.save(miEvento);
            insertarRdo = "El evento " + miEvento.getNombre() + " se ha insertado correctamente";

        }

    } catch (Exception e) {
        insertarRdo = "Error al insertar el evento. Verifique que los datos a insertar son correctos";
        e.printStackTrace();
    }

    return insertarRdo;

```

- a. Comprueba que no exista con ese ID → de cara al servicio REST (desde las vistas no podemos mandar ID)
- b. Comprueba que no exista vía equals → al haber redefinido el método equals, ya no considera distintos dos eventos con distinto id pero idénticos nombre, descripción, precio y fecha de inicio.
- c. Si todo es correcto, ejecuta la inserción

6) Método modificarEvento: se utiliza para grabar cambios de estado / destacado

```

@Override
public String modificarEvento(int idEvento, String newEstado, String newDestacado) {
    String mensaje = "Error desconocido";

    Evento miEventoIntermedio = null;

    boolean BDconexionOK = false;
    try {
        // Hemos definido este método en el EventosRepository,
        // que interpreta a través de JPARepository
        miEventoIntermedio = miEventosRepo.findById(idEvento).orElse(null);
        BDconexionOK = true;
    } catch (Exception e) {
        mensaje = "Error -> fallo de conexión a la BBDD";
    }

    String oldEstado = "";
    boolean eventoExiste = false;
    if (BDconexionOK) {
        if (miEventoIntermedio != null) {
            eventoExiste = true;
            oldEstado = miEventoIntermedio.getEstado();
        } else {
            mensaje = "Error -> Evento con ID " + idEvento + " no encontrado";
        }
    }

    boolean changeEstado = false;
    if (eventoExiste) {
        try {
            // Si viene sin newDestacado es que sólo quiere cambiar estado
            if (newDestacado == null && newEstado != null) {
                // Si es cancelado, forzamos que pase a NO destacado
                if (newEstado.equals("cancelado")) {
                    miEventoIntermedio.setDestacado("");
                    miEventoIntermedio.setEstado(newEstado);
                } else {
                    // Si no, simplemente grabamos el nuevo estado
                    miEventoIntermedio.setEstado(newEstado);
                }
            }
            changeEstado = true;
        }
    }
}

```

Comprobamos que la conexión es correcta y que el evento exista (de nuevo, orientado al servicio REST, ya que se entiende que si viene del formulario de edición no va a haber problema (el ID no es modificable en el formulario)).

Si damos la orden de cambiar estado sin newDestacado sólo queremos cambiar el estado → si pedimos cancelado, además, cancelará por defecto el evento.

```
// Y viceversa
} else if (newDestacado != null && newEstado == null) {

    // Si el evento está cancelado no podemos destacarlo, mandamos mensaje
    if (oldEstado.equals("cancelado") && newDestacado.equals("s")) {
        mensaje = "Error -> No se puede destacar un evento cancelado";
        // Si está activo no hay problema, pasamos a si destacado y avanzamos al
        // siguiente paso
    } else {
        miEventoIntermedio.setDestacado(newDestacado);
        changeEstado = true;
    }
} else {
    mensaje = "Error -> Acción no permitida";
}

} catch (Exception e) {
    mensaje = "Error -> Se ha encontrado el evento " + idEvento + ", pero no se ha podido cambiar su estado";
}

}

if (changeEstado) {
    try {
        miEventosRepo.save(miEventoIntermedio);
        // Customizamos el mensaje para que diga que ha quedado activo/cancelado y si/no
        // destacado
        if (miEventoIntermedio.getDestacado().equals("s")) {
            mensaje = "El evento " + miEventoIntermedio.getNombre() + " ha pasado a estar "
                + miEventoIntermedio.getEstado() + " y destacado";
        } else {
            mensaje = "El evento " + miEventoIntermedio.getNombre() + " ha pasado a estar "
                + miEventoIntermedio.getEstado() + " y NO destacado";
        }
    } catch (Exception e) {
        mensaje = "Error -> No se ha podido salvar en la BBDD la modificación de del evento a "
            + miEventoIntermedio.getEstado() + " y " + miEventoIntermedio.getDestacado();
    }
}

return mensaje;
```

Si sólo hemos pedido cambio de destacado, comprobaremos que esté activo → no podemos destacar un evento cancelado. De nuevo, desde la web no habría problema ya que el botón de destacar se deshabilita con eventos cancelados.

Cuando está listo, salvamos el evento modificado, y generamos un mensaje personalizado de resultado.

**\*\*El método editarEvento funciona análogamente, sólo que le pasamos un evento como objeto, y hace estas comprobaciones, además de hacer el save del evento \*\***

## 7) Método borrarEventosByTipo

```

public String borrarEventosByTipo(int idTipo) {
    String mensaje = "Error desconocido";

    Tipo miTipoIntermedio = null;

    boolean BDconexionOK = false;
    try {
        miTipoIntermedio = miTiposRepo.findById(idTipo).orElse(null);
        BDconexionOK = true;
    } catch (Exception e) {
        mensaje = "Error -> fallo de conexión a la BBDD";
    }

    boolean tipoExiste = false;
    if (BDconexionOK) {
        if (miTipoIntermedio != null) {
            tipoExiste = true;
        } else {
            mensaje = "Error -> Tipo con ID " + idTipo + " no encontrado";
        }
    }

    List<Evento> milistaEventos = null;
    boolean readyDeleteEventos = false;
    if (tipoExiste) {

        milistaEventos = devuelveByTipo(idTipo).getListaEventos();
        String mensajeEv = devuelveByTipo(idTipo).getMensaje();

        // Si la lista está en null, ha habido un error, asignamos mensaje
        if (milistaEventos == null) {

            mensaje = mensajeEv;
            // Si no es nula y
        } else {

            // tiene tamaño de 0, sin mensaje de error
            if (milistaEventos.size() == 0 && mensajeEv == null) {
                mensaje = "Error -> El tipo " + miTipoIntermedio.getNombre()
                    + " existe, pero no había ningún evento, así que no se ha eliminado nada";
            }
            // Estamos listos para eliminar el tipo directamente
            // tiene tamaño distinto de 0, sin mensaje de error
            // quiere decir que hay eventos asociados a ese tipo
            else if (milistaEventos.size() != 0 && mensajeEv == null) {
                // Estamos listos para eliminar esos eventos, antes de eliminar
                // el tipo
                readyDeleteEventos = true;
            }
        }
    }
}

```

Este método se usa internamente para cuando forzamos una eliminación de un tipo  
 → primero eliminamos sus eventos asociados.

- Comprueba la existencia del tipo, y si no existe informa de dicho problema (bien podría usarse desde la web para eliminar todos los eventos por un tipo, pero no está implementado).
- Recupera eventos por tipo, y si no hay ninguno, informamos de que no se puede eliminar porque no había nada.
- Si todo okey, hacemos delete de esa lista de eventos

**\*\*El resto de Daos utilizan lógicas similares, donde siempre prima el estilo descrito con anterioridad → gran cantidad de información y gran cantidad de verificaciones y acciones automáticas para evitar inconsistencias\*\***

La excepción quizás más reseñable sería el método insertarReserva:

```
public String insertarReserva(Reserva miReserva) {
    String insertarRdo = "";
    try {
        String msgCheckID = devuelvePorId(miReserva.getIdReserva()).getMensaje();
        System.out.println(msgCheckID);

        List<Reserva> lista = null;
        List<Reserva> listaResEvento = null;
        String rdo2 = "";

        // Hay un reserva con ese ID
        if (msgCheckID == null) {
            insertarRdo = "Error -> No se puede insertar un reserva con dicho id, ya que en la BBDD hay otro reserva con dicho ID. ¡Verifique!"
        }

        boolean isNotRepeated = true;
        if (msgCheckID != null && msgCheckID.contains("Error -> Reserva con ID 0 no encontrado")) {
            lista = devuelveByEmail(miReserva.getUsuario().getEmail()).getListaReservas();

            if (lista != null && lista.size() > 0) {
                for (int i = 0; i < lista.size(); i++) {
                    if (lista.get(i).getEvento().equals(miReserva.getEvento())) {
                        insertarRdo = "Ya dispone de una reserva de este evento. Si lo desea puede aumentar la cantidad desde su panel de usuario";
                        isNotRepeated = false;
                    }
                }
            }
        }
    }
}
```

Primero comprueba que no tengamos una reserva existente → no nos autoriza a realizarla. Si es del mismo evento, sólo podemos modificarla, pero no crear otra nueva.

```
boolean allOK = false;
if (isNotRepeated) {
    listaResEvento = devuelveByEvento(miReserva.getEvento().getIdEvento()).getListaReservas();

    if (listaResEvento != null) {
        int cantidadRes = 0;
        if (listaResEvento.size() > 0) {
            for (int i = 0; i < listaResEvento.size(); i++) {
                cantidadRes += listaResEvento.get(i).getCantidad();
            }
        }

        int totalInicial = cantidadRes + miReserva.getCantidad();
        int aforoMaximo = miReserva.getEvento().getAforoMaximo();
        if (totalInicial > aforoMaximo) {

            int cantidadSiReservable = aforoMaximo - cantidadRes;
            if (cantidadSiReservable > 0) {
                miReserva.setCantidad(cantidadSiReservable);
                allOK = true;
                rdo2 = "\n\nNOTA: Sólo quedaban "+cantidadSiReservable+" plazas disponibles, de modo que su reserva se ha reducido a ";
            } else {
                insertarRdo = "Error -> No se puede realizar la reserva al superarse el aforo máximo";
            }
        } else {
            allOK = true;
        }
    } else {
        allOK = true;
    }
}

}
```

Después comprueba que no se supere el aforo máximo → si se supera y la cantidad disponible es 0 informa al usuario de que no se puede realizar.

Si la cantidad disponible es >0 pero <pedida, ejecuta la reserva por dicha cantidad e informa al usuario → se entiende que el usuario prefiere tener una reserva de 3 entradas que de nada.

```
if(alloK) {  
    try {  
        miReservasRepo.save(miReserva);  
        insertarRdo = "La reserva de " + miReserva.getEvento().getNombre()+ " se ha insertado correctamente";  
        insertarRdo += rdo2;  
    } catch (Exception e) {  
        insertarRdo = "Error --> no se pudo realizar reserva contra BBDD";  
    }  
}
```

Si todo okey, finalmente realiza la reserva ejecutando el save.

\*El método de edición hará también esta comprobación, ya que el usuario podrá modificar la cantidad reservada desde la aplicación\*

5. **Servicios** → paquete auxiliar para las vistas. Contiene varias un interface con su Impl correspondiente → utilizamos los daos realizados con anterioridad

a. PreparaServImpl → dentro de esta clase, lo que haremos será preparar:

- i. Lista de tipos filtrada → para la navbar pública. Al usuario sólo le ofreceremos tipos que tengan eventos activos. No tiene sentido ofrecer un tipo sin eventos activos.
- ii. Lista de tipos full → para la navbar de organizador → aquí sí tendremos todos los tipos, sin filtrado alguno
- iii. Lista de categorías → mandamos las categorías completas. No tenemos distinción de activos, ya que las noticias no tienen dicho atributo.

Llamaremos a este servicio cada vez que ejecutemos una acción que vaya a una vista → no almacenamos en sesión porque podría quedar desactualizado mientras el usuario navega (además de no actualizarse si estamos gestionando como organizador o redactor).

6. **Controlador** → paquete principal para las vistas. Contiene varias clases:

```
J CategoriasController.java  
J EventosGestionController.java  
J EventosPublicController.java  
J HomeController.java  
J NoticiasGestionController.java  
J NoticiasPublicController.java  
J ReservasGestionController.java  
J ReservasPublicController.java  
J TiposController.java  
J UsuariosController.java
```



- 1) HomeController → este controlador se encargará de gestionar de manera principal:
  - a. Errores → en vez de permitir un mensaje genérico, para mantener el estilo de la web, asignaremos errores que dirigirán a un jsp
  - b. Login → mandaremos formulario y procesaremos si da error o si viene de logout
  - c. doLogin → lo utilizaremos para redirigir a /eventos/destacados y guardar el usuario como atributo de sesión
- 2) EventosPublicController → este controlador se encargará de gestionar los accesos a la parte pública, a la que accederemos por /eventos

```
@Autowired
IntEventosDao eventosDao;

@Autowired
IntTiposDao tiposDao;

@Autowired
IntPreparaServ prepWeb;
```

Creamos variables de IntEventosDao, IntTiposDao y de IntPreparaServ (servicio auxiliar explicado arriba)

Devuelven buyEventos.jsp: vemos de ejemplo /contiene/{palabra}

```
@GetMapping("/contiene/{palabra}")
public String verPorPalabra(Model model, @PathVariable (name="palabra") String palabra) {

    List<Evento> nuevaLista = null;
    String mensaje = "";
    String tipo = "";
    if(palabra != null) {
        List<Evento> listaEventos = eventosDao.devuelveByPalabra(palabra).getListaEventos();
        mensaje = eventosDao.devuelveByPalabra(palabra).getMensaje();
        nuevaLista = prepWeb.listaLimpia(listaEventos);
        if(nuevaLista == null || nuevaLista.size() == 0 ) {
            mensaje = "No hay eventos que contengan la palabra "+palabra;
        }
        tipo = "que contienen '"+palabra+"'";
    }else {
        mensaje = "Error: Indique una palabra correcta";
    }

    model.addAttribute("mensajeError", mensaje);

    model.addAttribute("milistaEventos", nuevaLista);
    model.addAttribute("tipo", tipo);
    model.addAttribute("origen","/contiene/"+palabra);

    return "eventos/buyEventos";
}
```

- a. /contiene/{palabra} → nos permitirá buscar por palabra en el nombre o descripción del evento → nos indica si palabra no es correcta, o si no hay (mensaje custom) → agregamos al model la lista, el mensaje, el tipo (utilizado para mostrar el título en el jsp) y el origen (para el enlace inferior); además de llamar a prepWeb para que nos prepare las listas a mostrar en la navbar.
- b. / → nos devolverá eventos activos
- c. /destacados → nos devolverá eventos destacados
- d. /tipo/{id} → nos devuelve los eventos por tipo

Devuelve detalleEvento.jsp

- e. /detalle/{idEvento} → nos mandará al jsp de detalle, de modo que recupera el evento pedido. Adicionalmente, puesto que en la vista se nos muestran eventos destacados en un aside, mandamos una lista con 2 eventos destacados.

```
@GetMapping("/detalle/{id}")
public String verPorId(Model model, @PathVariable(name = "id") int idEvento) {

    List<Evento> listaEventos = eventosDao.devuelvePorId(idEvento).getListaEventos();
    String mensaje = eventosDao.devuelvePorId(idEvento).getMensaje();
    System.out.println(listaEventos);
    System.out.println(mensaje);
    model = prepWeb.envia(model);

    List<Evento> listaDestacados = eventosDao.devuelveByDestacado("s").getListaEventos();
    for(int i=listaDestacados.size() - 1; i > 1; i--) {

        listaDestacados.remove(i);

    }
    model.addAttribute("listaEventos", listaDestacados);

    Evento miEvento = null;
    if (mensaje == null) {
        listaEventos = prepWeb.listaLimpia(listaEventos);
        if(listaEventos.size() >0) {
            miEvento = listaEventos.get(0);
        }else {
            mensaje = "Error --> No se ha encontrado ningún evento activo con ID "+idEvento;
        }
        System.out.println(miEvento);
    }

    model.addAttribute("mensajeError", mensaje);

    model.addAttribute("evento", miEvento);

    return "eventos/detalleEvento";
}
```

3) EventosGestionController → accedemos por /gestion/eventos →

Tenemos una parte inicial similar a la diseñada en el Public, con la salvedad de que tenemos un /todos verdadero que devuelve todos los eventos, incluidos los cancelados; y que sí podemos consultar eventos no destacados o eventos cancelados

Así mismo, tenemos las partes correspondientes de CRUD:

- a. /estado/{id}/{url}/{estado} o /destacado/{id}/{url}/{destacado} → cambiar estado, id del evento, url de procedencia y nuevo estado/destacado → la URL se utiliza para saber a dónde volver (no volver siempre a todos, por ejemplo)

```
@GetMapping("/estado/{id}/{url}/{estado}")
public String cancelar(RedirectAttributes miRedirAttrib, @PathVariable(name = "id") int idEvento, @PathVariable(name = "url") String destino) {
    String mensaje = "";
    if(estado.equals("activo") || estado.equals("cancelado")) {
        try {
            mensaje = eventosDao.modificarEvento(idEvento, estado, null);
        } catch (Exception e) {
            mensaje = "Fallo desconocido al estado del evento";
            e.printStackTrace();
        }
    } else {
        mensaje = "Sólo puede modificar el estado de un evento a 'activo' o a 'cancelado'";
    }

    miRedirAttrib.addFlashAttribute("mensaje", mensaje);
    miRedirAttrib.addFlashAttribute("urlDestino", "/"gestion/eventos/"+destino);

    return "redirect:/gestion/eventos/"+destino;
}
```

Llama al método modificarEvento y pasa como parámetro el estado/destacado a cambiar.

- b. /procesarFormulario → se utiliza para procesar creación / edición de evento:

```
public String procesarFormulario(RedirectAttributes miRedirAttrib, @PathVariable(name = "url") String destino, Evento evento) {

    System.out.println(evento);

    evento.setEstado("activo");

    String mensaje = eventosDao.insertarEvento(evento);

    System.out.println(evento);

    System.out.println(mensaje);

    /////////////////////////////////////////////////// IMPORTANTE ///////////////////////////////////
    miRedirAttrib.addFlashAttribute("mensaje", mensaje);
    miRedirAttrib.addFlashAttribute("urlDestino", "/"gestion/eventos/"+destino);

    return "redirect:/gestion/eventos/"+destino;
}
```

También tendremos /créate y /editar que devolverá simplemente el jsp de formulario, y /eliminar que simplemente ejecutará el método de eliminación del DAO.

El resto de controladores tiene un funcionamiento análogo (incluidos los REST, que se asemejan a gestión, pero devolviendo un String / JSON). Sin embargo, en el caso de reservas tendremos también un método que nos permite modificar la cantidad desde la propia vista:

```
@GetMapping("/add/{id}")
public String editarAdd(Model model, @PathVariable(name = "id") int idreserva, HttpSession miSesion, RedirectAttributes miRedirAtrib) {
    List<Reserva> listaReservas = reservasDao.devuelvePorId(idreserva).getListaReservas();
    String mensaje = reservasDao.devuelvePorId(idreserva).getMensaje();
    boolean isAdmin = false;
    Usuario usuario = (Usuario) miSesion.getAttribute("usuario");
    Reserva reserva = null;
    if (listaReservas != null) {
        if (listaReservas.size() == 1 && mensaje == null) {
            reserva = listaReservas.get(0);
            int cantidad = reserva.getCantidad();
            System.out.println("CANTIDAD: "+cantidad);
            cantidad++;
            System.out.println("CANTIDAD NUEVA: "+cantidad);
            reserva.setCantidad(cantidad);
            mensaje = reservasDao.editarReserva(reserva, isAdmin, usuario);
            System.out.println("CANTIDAD SUPERNUEVA: "+reserva.getCantidad());
        }
    }

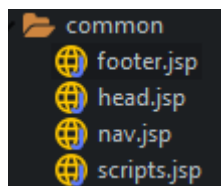
    miRedirAtrib.addFlashAttribute("mensaje", mensaje);
    System.out.println(reserva);
    return "redirect:/cliente/reservas/todas";
}
```

Recuperará el usuario, al entrar por /cliente/reservas, no será admin, y realizaremos un cambio en el objeto recuperado por id, asignando la nueva cantidad. Después llamaremos al método de edición del DAO de reservas.

La especialidad de admin y usuario se refiere a la comprobación que se hace en el DaoImpl → si la reserva no pertenece al usuario no se puede editar, y sólo si es admin puede editar la de otra persona.

## Parte 2: Vistas JSP

Todos los jsp contendrán partes comunes: nav y footer; además de importar un head y unos scripts comunes:



Es reseñable que el nav variará según el tipo de usuario:

1. Si no está logeado, mostraremos el siguiente:

MRM PERFORMANCE	Home	Eventos	Coches Nuevos	Noticias	Contacto	Conócenos	Login
-----------------	------	---------	---------------	----------	----------	-----------	-------

Daremos opción de login.

2. Si está logeado y es cliente, añadimos sección de reservas, nombre y logout:

MRM PERFORMANCE	Home	Eventos	Coches Nuevos	Noticias	Contacto	Conócenos	Reservas	Logout	¡¡Hola Arturo!!
-----------------	------	---------	---------------	----------	----------	-----------	----------	--------	-----------------

3. Si es organizador, añadimos barra de gestión de eventos:

MRM PERFORMANCE	Home	Eventos	Coches Nuevos	Noticias	Contacto	Conócenos	Reservas	Logout	¡¡Hola Marta!!
PANEL DE GESTIÓN	Eventos	Eventos / estado	Eventos / destacado	Eventos / tipo	Tipos	Reservas	ROL: organizador		

Tenemos todos los filtrados de eventos, los tipos, las reservas y el rol

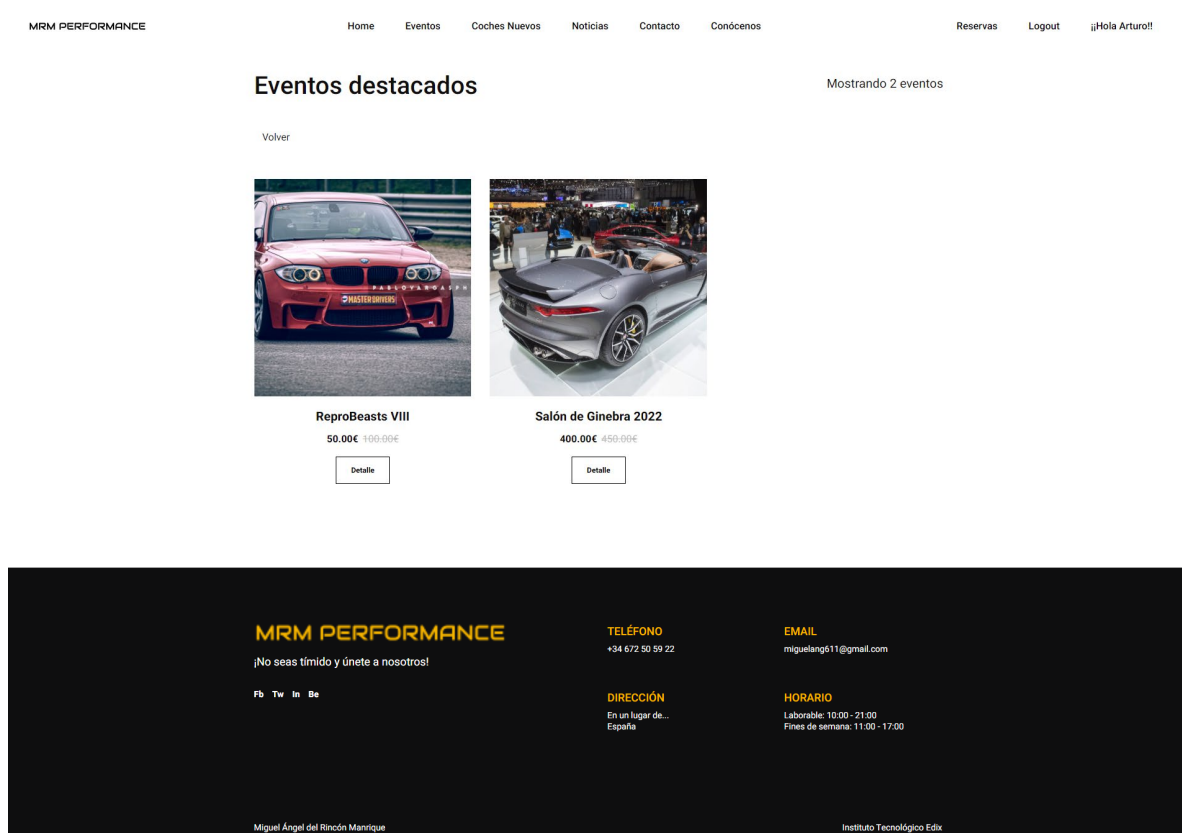
4. Si es redactor, añadimos barra de gestión de noticias:

MRM PERFORMANCE	Home	Eventos	Coches Nuevos	Noticias	Contacto	Conócenos	Reservas	Logout	¡¡Hola Pepe!!
PANEL DE GESTIÓN	Noticias	Noticias / destacada	Noticias / categoria	Categorias	ROL: redactor				

5. Si es admin, tiene todas las posibilidades:

MRM PERFORMANCE	Home	Eventos	Coches Nuevos	Noticias	Contacto	Conócenos		Reservas	Logout	¡¡Hola Administrador!!
PANEL DE GESTIÓN		Eventos	Eventos / estado	Eventos / destacado	Eventos / tipo	Tipos	Reservas	ROL: admin		
PANEL DE GESTIÓN			Noticias	Noticias / destacada	Noticias / categoria	Categorias		ROL: admin		

Un ejemplo de vista sería:



Funcionamiento:

```
<c:if test="${milistaEventos.size() != 0}">
  <main class="shop-page">
    <div class="container">
      <div class="page-header wow fadeInUp">
        <h1 class="page-title">Eventos ${tipo}</h1>
        <p class="result-count">Mostrando ${milistaEventos.size()}
          eventos</p>
      </div>
      <c:if test="${tipo == 'activos' }">
        <div class="blog-post-single">
          <a class="btn" href="/eventos/destacados">Destacados</a>
          <c:forEach items="${listaTipos}" var="miTipo"
            varStatus="varEstado">
            <a class="btn" href="/eventos/tipo/${miTipo.idTipo}">${miTipo.nombre}</a>
          </c:forEach>
        </div>
      </c:if>
      <c:if test="${tipo != 'activos' }">
        <div class="row">
          <div class="col-md-8 blog-post-wrapper">
            <div class="blog-post-single wow fadeInUp">
              <a class="btn prev-post" href="/eventos/">Volver</a>
            </div>
          </div>
        </div>
      </c:if>
    </div>
  </main>
</c:if>
```

Si la lista de eventos no está vacía, según el tipo de página pedida, ofrecemos distintas opciones:

Si son activos, mostramos cinta de opciones por destacados y categoría. Si no, no la mostramos.

```
<c:forEach var="evento" items="{milistaEventos}">
  <c:if test="{evento.destacado.equals('s')}">
    <article class="col-md-4 product-card wow fadeInUp">
      <a href="/eventos/detalle/{evento.idEvento}">
        <div class="product-thumbnail-wrapper">
          
        </div>
      </a>
      <h5 class="product-title">{evento.nombre}
      <c:if test="{evento.destacado.equals('s')} && tipo != 'destacados'">
        ¡Destacado!
      </c:if>
      </h5>
      <c:if test="{evento.destacado.equals('s')}">
        <p class="product-price">{evento.precio}€<del>{evento.precio +50}€</del>
      </p>
      </c:if>
      <c:if test="{evento.destacado.equals('')}">
        <p class="product-price">{evento.precio}€<del>{evento.precio +20}€</del>
      </p>
      </c:if>
      <form class="btn-wrapper">
        <button class="btn btn-add-to-cart" type="submit"
          formaction="/eventos/detalle/{evento.idEvento}">
          Detalle</button>
      </form>
    </article>
  </c:if>
</c:forEach>
<c:forEach var="evento" items="{milistaEventos}">
  <c:if test="{evento.destacado.equals('')}">
    <article class="col-md-4 product-card wow fadeInUp">
      <a href="/eventos/detalle/{evento.idEvento}">
        <div class="product-thumbnail-wrapper">
          
        </div>
      </a>
      <h5 class="product-title">{evento.nombre}
      <c:if test="{evento.destacado.equals('s')} && tipo != 'destacados'">
        ¡Destacado!
      </c:if>
      </h5>
      <c:if test="{evento.destacado.equals('s')}">
        <p class="product-price">{evento.precio}€<del>{evento.precio +50}€</del>
```

Después, leemos la lista de eventos, primero mostramos destacados, y después no destacados (con un doble forEach y un if).

Por cada evento, colocamos su imagen, su nombre, si es destacado, su precio y un botón de añadir al carrito.

A continuación incluiré una serie de muestras de las páginas., aunque para más detalles sobre el funcionamiento de las vistas, aconsejo comprobar el vídeo 2, disponible en el repositorio:

[https://github.com/miguelang611/TFG\\_MiguelRM/blob/main/Informe-TFG-Parte-2-Demostracion-Principal.mp4](https://github.com/miguelang611/TFG_MiguelRM/blob/main/Informe-TFG-Parte-2-Demostracion-Principal.mp4)

## Detalle de un evento

### Evento ReproBeasts VIII



¡EVENTO DESTACADO!

¡Alto número de reservas en las pasadas 24 horas!

#### 8º Edición de la competición de tandas de los 50 coches más potenciados de España

Localización: Circuito del Jarama - Madrid

Fecha: 2021-06-20

Duración: 12 horas

Aforo máximo: 100 personas

Precio: ~~70.00€~~ 50.00€

Cantidad

Comprar

[← Volver](#)

#### Etiquetas

Circuito

Destacado

2021

#### Otros Eventos



#### Categorías

Salones

Clásicos

Circuito

Monomarca



## Página de Noticias

MRM PERFORMANCE

[Home](#) [Eventos](#) [Coches Nuevos](#) [Noticias](#) [Contacto](#) [Conócenos](#)

[Login](#)

### Noticias

[Buscar](#)

#### Bienvenida

¡MRM Performance!

¡Cuéntame más! [————](#)



#### BMW M5 2022

La nueva bestia bávara

¡Cuéntame más! [————](#)



## Página de detalle de noticia

MRM PERFORMANCE

[Home](#) [Eventos](#) [Coches Nuevos](#) [Noticias](#) [Contacto](#) [Conócenos](#)

[Login](#)

### ¿Cómo lavar tu coche...



Fecha: 2021-06-03 | by Pipe

A la hora de lavar un coche son muchas las cosas que tenemos que fijarnos: esponja, champú, tipo de agua, balleta de secado y productos específicos para las llantas

[— Volver](#)

#### Etiquetas

[Consejos](#) [2021](#)

#### Otros Noticias



#### Categorías

[General](#)  
[Próximos Eventos](#)  
[Vehículos Nuevos](#)  
[Pruebas](#)  
[Consejos](#)

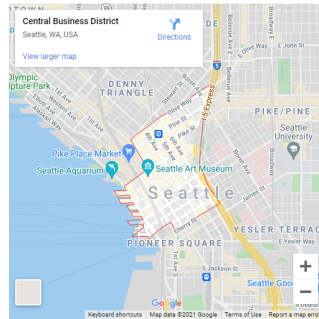
## Página de login

MRM PERFORMANCE

[Home](#) [Eventos](#) [Coches Nuevos](#) [Noticias](#) [Contacto](#) [Conócenos](#)

[Login](#)

### Login



Usuario

Contraseña

Entrar

¿Aún no registrado? ¡Sin problema!

Regístrate

## Formulario de registro

MRM PERFORMANCE

[Home](#) [Eventos](#) [Coches Nuevos](#) [Noticias](#) [Contacto](#) [Conócenos](#)

[Login](#)

### Formulario de Registro

Nombre

---

Email

---

Contraseña

---

Dirección

---

¡Vamos!

## Gestión de eventos

MRM PERFORMANCE	Home	Eventos	Coches Nuevos	Noticias	Contacto	Conócenos	Reservas	Logout	¡¡Hola Marta!!
PANEL DE GESTIÓN	Eventos	Eventos / estado	Eventos / destacado	Eventos / tipo	Tipos	Reservas	ROL: organizador		

### Eventos (todos)

Mostrando 5 eventos

[Nuevo Evento](#)

Id	Nombre	Descripción	Precio	Estado	Destacado					
1	ReproBeasts VIII	8ª Edición de la competición de tandas de los 50 coches más potenciados de España	50.00	activo	Si	<a href="#">Editar</a>	<a href="#">Cancelar</a>	<a href="#">Normal</a>	<a href="#">Reservas</a>	<a href="#">Eliminar</a>
2	Salón de Ginebra 2022	Primera edición post-covid del salón mundial del automóvil de más importancia a nivel mundial	400.00	activo	Si	<a href="#">Editar</a>	<a href="#">Cancelar</a>	<a href="#">Normal</a>	<a href="#">Reservas</a>	<a href="#">Eliminar</a>
3	Autobello Toledo	Concentración de superdeportivos: Ferrari, Lamborghini, Bugatti... Y mucho más!	70.00	cancelado	No	<a href="#">Editar</a>	<a href="#">Activar</a>	<a href="#">Destacar</a>	<a href="#">Reservas</a>	<a href="#">Eliminar</a>
4	Classic Rules	Concentración de vehículos clásicos (+25 años) en el circuito de Barcelona. Se incluye la posibilidad de 3 vueltas al circuito bajo pago de importe extra in situ	40.00	activo	No	<a href="#">Editar</a>	<a href="#">Cancelar</a>	<a href="#">Destacar</a>	<a href="#">Reservas</a>	<a href="#">Eliminar</a>
5	AudiSportiberica 2021	Concentración monomarca de vehículos Audi en Albacete, organizada en colaboración con el club AudiSportiberica	70.00	activo	No	<a href="#">Editar</a>	<a href="#">Cancelar</a>	<a href="#">Destacar</a>	<a href="#">Reservas</a>	<a href="#">Eliminar</a>

[← Volver](#)

## Gestión de tipos

MRM PERFORMANCE	Home	Eventos	Coches Nuevos	Noticias	Contacto	Conócenos	Reservas	Logout	¡¡Hola Marta!!
PANEL DE GESTIÓN	Eventos	Eventos / estado	Eventos / destacado	Eventos / tipo	Tipos	Reservas	ROL: organizador		

### Tipos

Mostrando 5 tipos

[Nuevo Tipo](#)

Id	Nombre	Descripción		
1	Deportivos	Eventos relacionados con vehículos deportivos	<a href="#">Editar</a>	<a href="#">Eliminar</a>
2	Salones	Eventos relacionados con los salones del automóvil	<a href="#">Editar</a>	<a href="#">Eliminar</a>
3	Clásicos	Eventos relacionados con los clásicos	<a href="#">Editar</a>	<a href="#">Eliminar</a>
4	Circuito	Eventos relacionados con tandas en circuitos	<a href="#">Editar</a>	<a href="#">Eliminar</a>
5	Monomarca	Eventos relacionados con una marca en concreto	<a href="#">Editar</a>	<a href="#">Eliminar</a>

[← Volver](#)

## Reservas de cliente

MRM PERFORMANCE	Home	Eventos	Coches Nuevos	Noticias	Contacto	Conócenos	Reservas	Logout	¡¡Hola Arturo!!
-----------------	------	---------	---------------	----------	----------	-----------	----------	--------	-----------------

### Mis Reservas

Mostrando 1 reservas

[Nueva Reserva](#)

La reserva de **ReproBeasts VIII** se ha insertado correctamente

Id Reserva	Nombre	Fecha	Cantidad		
1	ReproBeasts VIII	2021-06-20	3 + / -	<a href="#">Detalle Evento</a>	<a href="#">Cancelar</a>

[← Volver](#)

## Gestión de reservas (como admin)

MRM PERFORMANCE	Home	Eventos	Coches Nuevos	Noticias	Contacto	Conócenos		Reservas	Logout	¡Hola Administrador!!
PANEL DE GESTIÓN		Eventos	Eventos / estado	Eventos / destacado	Eventos / tipo	Tipos	Reservas	ROL: admin		
PANEL DE GESTIÓN			Noticias	Noticias / destacada	Noticias / categoria	Categorias	ROL: admin			

### Reservas del evento 'Autobello Toledo'

Mostrando 2 reservas

[Nueva Reserva](#)

Id Reserva	Cliente	Cantidad		
1	abc@abc.es	1 + / -	<a href="#">Detalle Evento</a>	<a href="#">Cancelar</a>
3	admin@admin.com	1 + / -	<a href="#">Detalle Evento</a>	<a href="#">Cancelar</a>

[← Volver](#)

## Página de error

MRM PERFORMANCE	Home	Eventos	Coches Nuevos	Noticias	Contacto	Conócenos	Login
-----------------	------	---------	---------------	----------	----------	-----------	-------

# 500

¡Vaya! Algo gordo ha fallado. ¡Sentimos no poder mostrarte esta web!

**MRM PERFORMANCE**

¡No seas tímido y únete a nosotros!

Fb Tw In Be

**TELÉFONO**

+34 672 50 59 22

**DIRECCIÓN**

En un lugar de...  
España

**EMAIL**

miguelang611@gmail.com

**HORARIO**

Laborable: 10:00 - 21:00  
Fines de semana: 11:00 - 17:00

## **7. Conclusiones y puntos de mejora**

Como conclusión del proyecto me gustaría destacar el aprendizaje realizado a lo largo del grado, que ahora se aúna en una sólo aplicación, en un solo proyecto, que encarna todos los aprendizajes realizados a lo largo de estos 2 años, siendo por supuesto los más destacables programación y bases de datos, así como sus “encarnaciones” en desarrollo de cliente, de servidor y de diseño de interfaces web.

En cuanto a los puntos de mejora, propondría primeramente la realización de una página de inicio donde se obtuvieran algunos de los eventos y noticias, que no he llegado a realizar.

Por otro lado, destacaría también la limpieza del código de los “jsp”, así como rehacer una plantilla de CSS más a medida y que reduzca la cantidad de código que le llega al cliente final, con objeto de mejorar la velocidad de carga.

Así mismo, considero que también habría sido interesante añadir HTTPS al proyecto, así como encriptado de los datos almacenados en la base de datos (contraseñas principalmente).

Sin embargo,

Sin embargo, pese a que no sea una aplicación lista para el salto a la producción, creo que se acerca lo suficiente, destacando la gran cantidad de verificaciones y mensajes enfocados al usuario, y la posibilidad de llevarse a cabo en la vida real con poco esfuerzo, al tratarse de un proyecto, bajo mi punto de vista, más que realizable, debido a la importancia cada vez más alta del ocio en la sociedad, y la carencia de webs que aúnen ocio e información para los aficionados del mundo del motor.

## **8. Bibliografía**

BootStrapDash (2019), *Plantillas Boorstrap, web de BootStrapDash* → <https://www.bootstrapdash.com/> (Fecha de consulta: 20 de mayo de 2021)

Spring Initialitzer (2021), *Generador de proyectos, web de Spring IO* → <https://start.spring.io/> (Fecha de consulta: 15 de abril de 2021)

TutorialRepublic (2019), *Ejemplos de Boorstrap, web de TutorialRepublic* → <https://www.tutorialrepublic.com/> (Fecha de consulta: 5 de mayo de 2021)

## **Anexo 1: Guía Servicio REST**

### **EVENTOS (/eventos)**

SERVICIO	URL	TIPO SERV
Consultar todos los eventos	/	Envía: GET Devuelve: texto plano formateado
Consultar eventos destacados, o no	/destacados/si /destacados/no	Envía: GET (parámetro si/no en URL) Devuelve: texto plano formateado
Consultar por un estado	/estado/activos /estado/cancelados	Envía: GET (par. activos/cancelados) Devuelve: texto plano formateado
Consultar eventos por un tipo	/tipo/{idTipo}	Envía: GET (parámetro idTipo) Devuelve: texto plano formateado
Consultar eventos por palabra dada (todos los que contengan esta palabra, y que estén activos) - opcional	/contiene/{palabra}	Envía: GET (parámetro palabra) Devuelve: texto plano formateado
Cambiar de estado	/estado/{idEvento}/activo /estado/{idEvento }/cancelado	Envía: GET (parámetro idEvento + activo/cancelado) Devuelve: texto plano formateado
Destacar/no destacar	/destacar/{idEvento}/si /destacar/{idEvento}/no	Envía: GET (parámetro idEvento + si/no) Devuelve: texto plano formateado
Consultar un evento	/ {idEvento} (versión JSON) / {idEvento}/texto	Envía: GET (parámetro idEvento) Devuelve: JSON del evento / texto
Alta de un evento	/	Envía: POST (JSON con los datos del evento) Devuelve: texto plano formateado
Modificación de un evento	/	Envía: PUT (JSON con los datos del evento) Devuelve: texto plano formateado
Eliminar un evento	/ {idEvento}	Envía: DELETE (parámetro idEvento) Devuelve: texto plano formateado
Eliminar eventos de un tipo	/tipo/{idTipo}	Envía: DELETE (parámetro idTipo) Devuelve: texto plano formateado

## TIPOS (/tipos)

SERVICIO	URL	TIPO SERV
Consultar todos los tipos	/	Envía: GET Devuelve: texto plano formateado
Alta de tipo	/	Envía: POST (JSON con datos del tipo) Devuelve: texto plano formateado
Modificar un tipo	/	Envía: PUT (JSON con datos del tipo) Devuelve: texto plano formateado
Consultar un tipo	/ {id} (versión JSON) / {id} /texto	Envía: GET (parámetro id en URL) Devuelve: JSON / texto plano formateado
Consultar tipos por palabra dada (todos los que contengan esta palabra, y que estén activos) - opcional	/contiene/{palabra}	Envía: GET (parámetro palabra) Devuelve: texto plano formateado
Eliminar un tipo	/ {id} (no fuerza eliminación eventos, pero informa si hay) / {id} /forceDelete (sí fuerza eliminación de ev. asociados)	Envía: DELETE (parámetro id en URL) Devuelve: texto plano formateado