

18-2-2019

TRABAJO 2ª EVALUACIÓN

ACCESO A DATOS

Javier Partida Molina

INDICE

1.	INTRODUCCIÓN	1
2.	SPRING BOOT	2
3.	SPRING LOADED	4
4.	FRONT-END	4
-	Invitado	4
-	Usuario registrado.....	6
-	Administrador	10
5.	LOGS	12
6.	PATRÓN DAO.....	13
-	Entidades.....	13
-	Relaciones	14
-	Repositorio	15
-	Base de datos	15
7.	PATRÓN DTO Y DOZER	16
8.	BASE DE DATOS REMOTA.....	17
9.	IMÁGENES	18
10.	JAVA 8.....	19
11.	Páginas de errores.....	19
12.	CONTROL DE ERRORES.....	20
13.	SPRING SECURITY	20
	SecurityConfiguration:	21
	Secularización de métodos:	21
	Secularización de vistas:.....	22
14.	SPRING REST.....	23
15.	WebGrafia	24

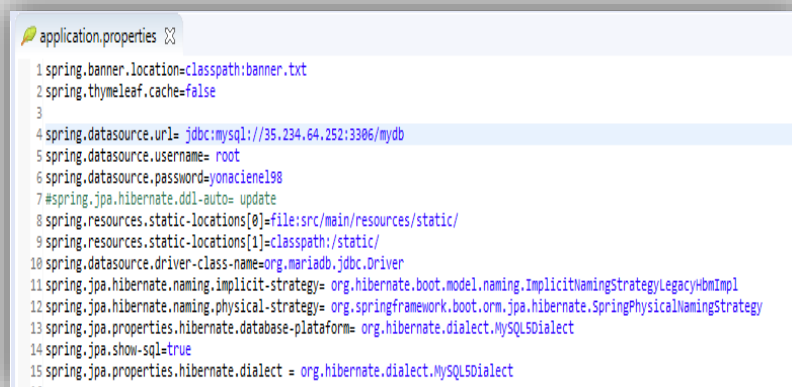
1. INTRODUCCIÓN

- Bienvenido a la memoria de mi aplicación web desarrollada con Spring, a continuación, realizaré una pequeña síntesis de que he aprendido durante el desarrollo de este proyecto y que problemas me he encontrado.
- Con este nuevo proyecto he conseguido comprender en casi toda su totalidad como funciona este Framework y a sacar jugo de todo su potencial. También he tenido mi primer contacto “real” de uso de seguridad en una aplicación (Spring Security), lo cuál me a acercado un poco a la realidad actual de como crear una aplicación con seguridad. Otra de las cosas que también me a gustado mucho es el uso de Spring REST, ya que actualmente el uso de API REST está a la orden del día, ya que su potencial es muy alto.
- La peor parte de un proyecto siempre son los odiados errores. Varios de ellos han ido apareciendo en el desarrollo de este proyecto, pero me gustaría recalcar alguno de ellos:
 - **Dependencia thymeleaf extras:** Para el funcionamiento correcto de las etiquetas de Spring Security en nuestras plantillas necesitamos instalar dicha dependencia. Pero el problema surge cuando intento utilizarlas en mi proyecto, pero no funcionan. Esto era debido, a que mi proyecto fue creado con la última versión de Spring boot y la dependencia solo era compatible con versiones más antiguas. La solución fue cambiar la versión del proyecto a una versión menos actualizada.
 - **Ddl-auto:** Como ya sabemos esta propiedad de Spring nos permite crear automáticamente nuestra base de datos, gracias a las anotaciones de hibernate. Pero por problemas que desconozco las relaciones no se creaban, con lo cual decidí crear mi base de datos a mano, lo cual solucionó mi problema.
 - **Spring REST y Spring Security:** El manejo de un Api rest en un proyecto que utiliza configuración de Spring Security, parece ser un poco complicado. Esto se debe a que Spring Security parece “capar” los métodos **post, put y delete**. Los métodos **get** funcionan perfectamente, pero los nombrados anteriormente devuelven un error 403, es decir, no tenemos permiso para realizar dichas operaciones. La url que gestiona la Api Rest está dentro de la configuración de Spring Security en el apartado **permirAll()**, pero aun así sigue retornando el error 403.

- Esta aplicación está basada en la gestión de un videoclub. La aplicación podrá tener tres tipos de usuarios:
 - Invitado:
 - Este usuario solo podrá ver los juegos que se ofrecen en la web (alquiler y venta), competiciones y noticias. Sin opción a alquilar, comprar y a apuntar a una competición.
 - Usuario registrado:
 - Este usuario podrá alquilar y comprar juegos, apuntarse a competiciones, acceder a historiales de sus alquileres y compras, consultar sus posiciones en las competiciones en la que a estado apuntado. También tendrá opción a editar su perfil de usuario.
 - Administrador:
 - Este usuario tiene control absoluto de nuestra aplicación web, además de tener los mismos permisos del usuario registrado, este usuario tendrá opción de crear cualquier CRUD (Juegos, Competiciones, Usuarios, Plataformas, Categorías, Noticias, Participaciones). Además, también puede consultar las ventas y alquileres que se han realizado, teniendo la opción de devolver juegos vendidos y devolver los juegos alquilados.

2. SPRING BOOT

- El proyecto se ha creado con Spring Boot desde 0, y se le han dado las siguientes configuraciones:



```
application.properties
1 spring.banner.location=classpath:banner.txt
2 spring.thymeleaf.cache=false
3
4 spring.datasource.url=jdbc:mysql://35.234.64.252:3306/mydb
5 spring.datasource.username= root
6 spring.datasource.password=yonacienel98
7 #spring.jpa.hibernate.ddl-auto= update
8 spring.resources.static-locations[0]=file:src/main/resources/static/
9 spring.resources.static-locations[1]=classpath:/static/
10 spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
11 spring.jpa.hibernate.naming.implicit-strategy= org.hibernate.boot.model.naming.ImplicitNamingStrategyLegacyHbmImpl
12 spring.jpa.hibernate.naming.physical-strategy= org.springframework.boot.orm.jpa.hibernate.SpringPhysicalNamingStrategy
13 spring.jpa.properties.hibernate.database-platform= org.hibernate.dialect.MySQL5Dialect
14 spring.jpa.show-sql=true
15 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
```

- Dependencias del proyecto:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>springloaded</artifactId>
  <version>1.2.8.RELEASE</version>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
</dependency>
<dependency>
  <groupId>com.querydsl</groupId>
  <artifactId>querydsl-jpa</artifactId>
</dependency>

<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
</dependency>
<dependency>
  <groupId>net.sf.dozer</groupId>
  <artifactId>dozer</artifactId>
  <version>5.5.1</version>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>org.mariadb.jdbc</groupId>
  <artifactId>mariadb-java-client</artifactId>
</dependency>
<dependency>
  <groupId>org.thymeleaf.extras</groupId>
  <artifactId>thymeleaf-extras-springsecurity4</artifactId>
  <version>3.0.4.RELEASE</version>
</dependency>
```

3. SPRING LOADED

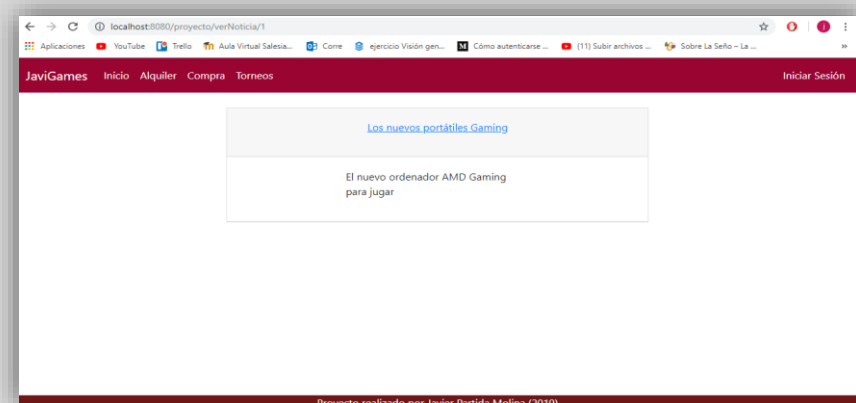
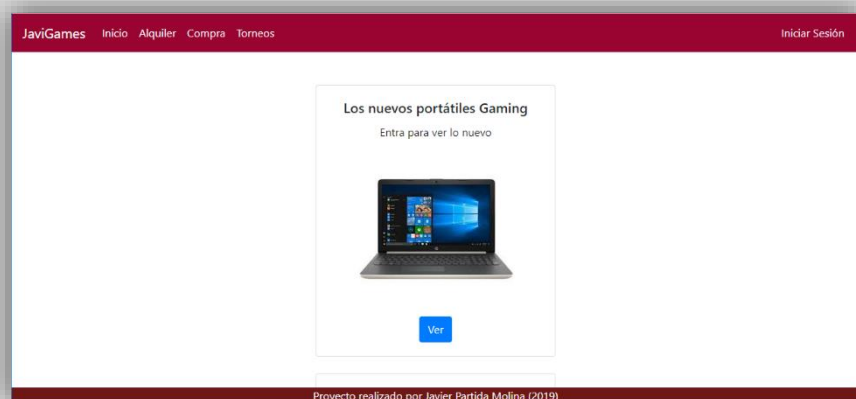
- Para ello instalo la dependencia correspondiente:

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>springloaded</artifactId>
  <version>1.2.8.RELEASE</version>
</dependency>
```

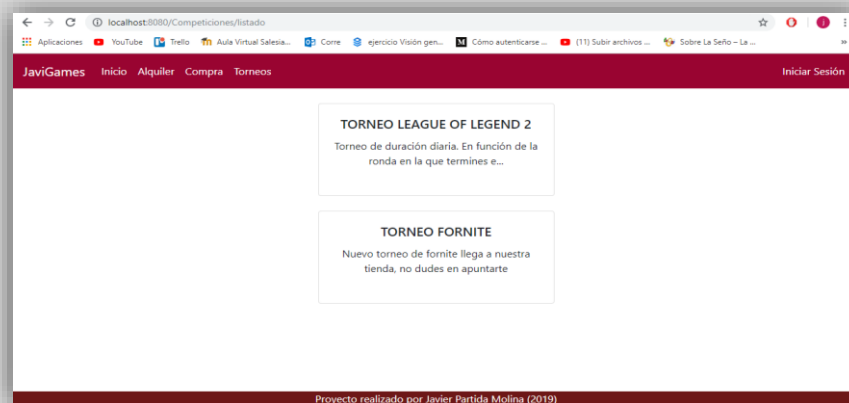
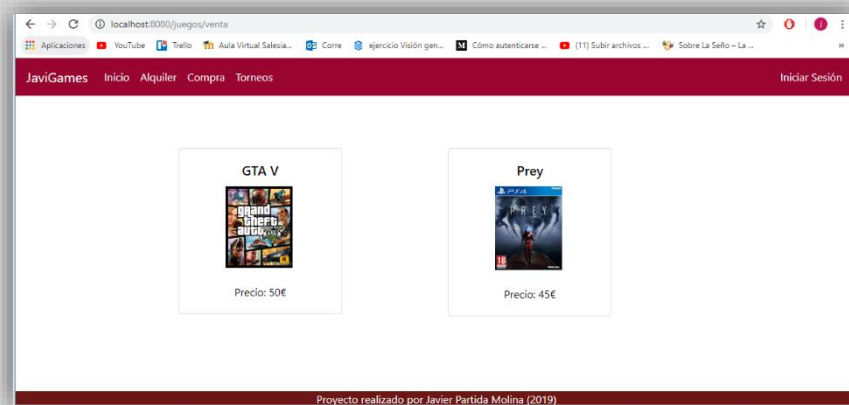
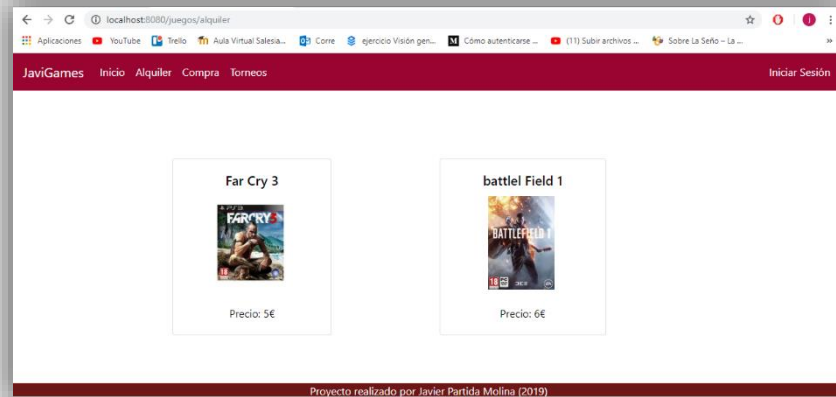
- Y configuro el application.properties para que las plantillas se actualicen sin necesidad de actualizar el servidor:
 - o spring.thymeleaf.cache=false

4. FRONT-END

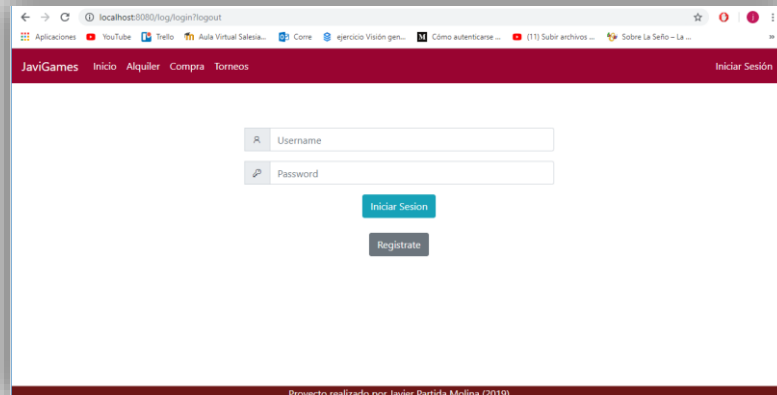
- Invitado:
 - o Un usuario que no ha iniciado sesión, se encontrará en primer lugar con la pestaña de noticias, en la cual aparecerán las noticias que creamos para nuestra página (admin). Cada una de las noticias tiene un botón “ver” para entrar en las noticias y poder leer más en profundidad.



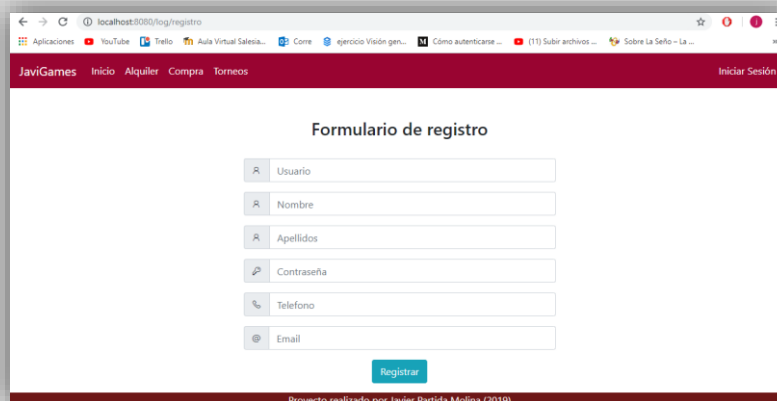
- Luego podrá interactuar con el menú superior para poder entrar juegos de alquiler, de ventas y en la zona de competiciones. Pero sin opción a comprar o alquilar, y sin poder apuntarse a una competición.



- Además, también en el menú superior encontramos la opción de inicio de sesión, en la que además de poder hacer login, también podrá registrarse como nuevo usuario.

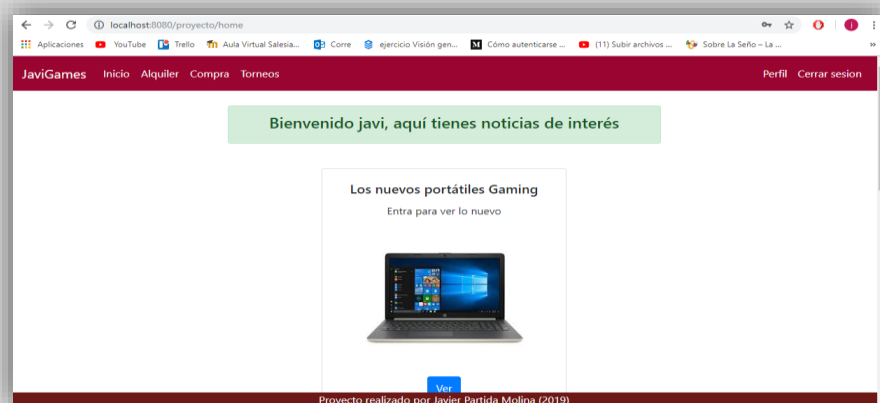


Proyecto realizado por Javier Partida Molina (2019)



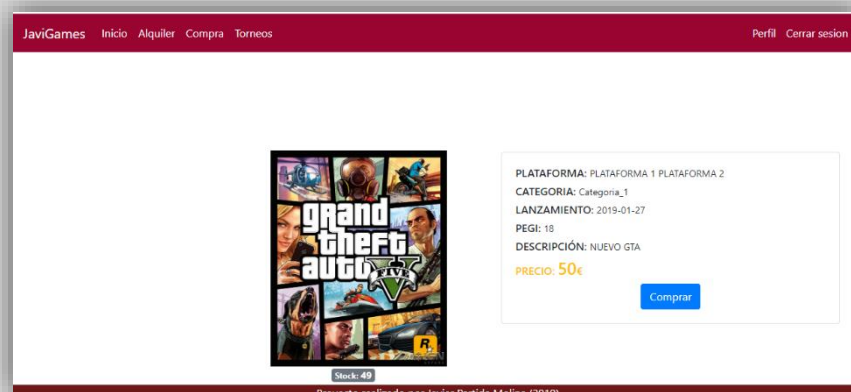
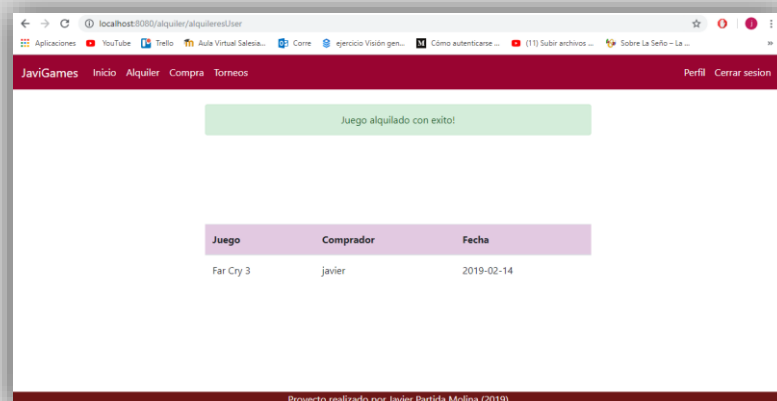
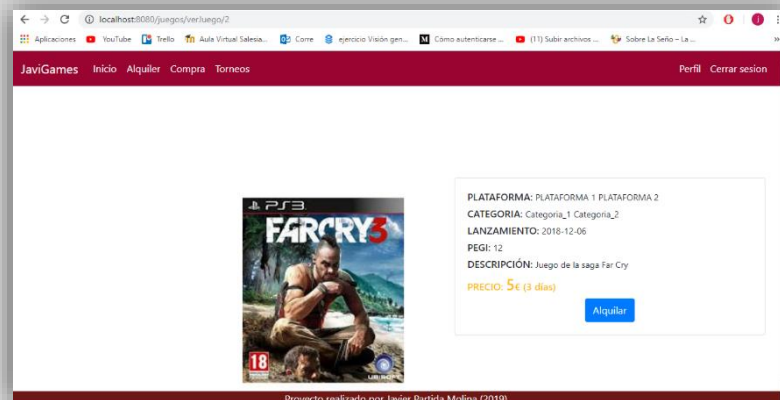
Proyecto realizado por Javier Partida Molina (2019)

- **Usuario registrado:**
 - Un usuario registrado, al igual que el invitado, estará en la pestaña noticias de forma inicial, pero con el añadido de que tiene un mensaje de bienvenida.



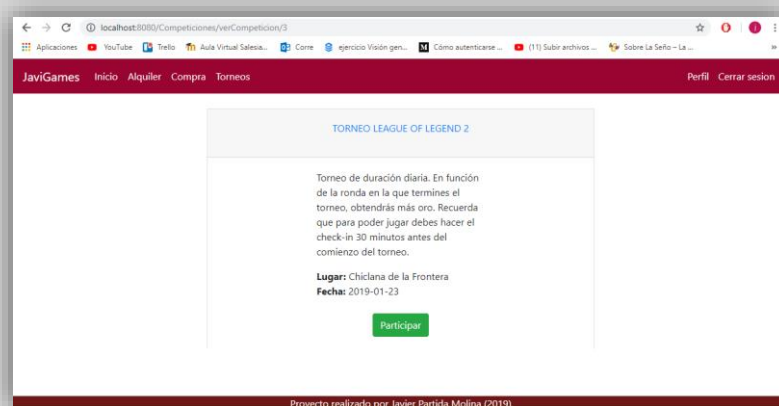
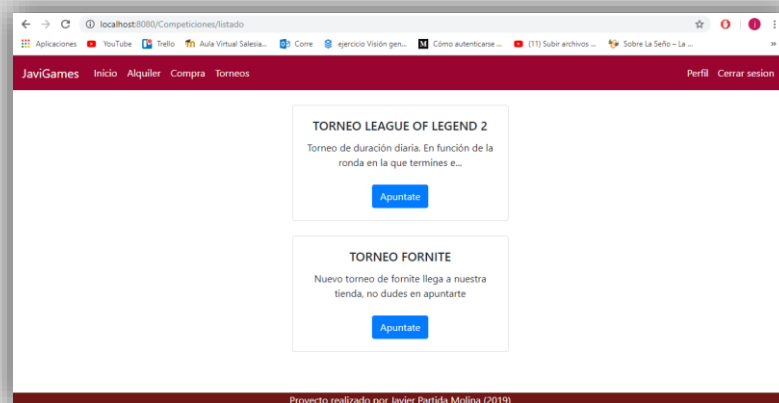
Proyecto realizado por Javier Partida Molina (2019)

- Luego podrá interactuar con el menú superior para poder entrar juegos de alquiler, de ventas y en la zona de competiciones. Y podrá entrar en cada uno de los juegos para ver información adicional y también si lo desea comprar o alquilar el juego.
- Después de realizar una compra o alquiler, el usuario es redirigido hacia una pestaña en la que se puede ver su nuevo alquiler o venta, y además del listado de todo su historial de las mismas.





- En cuanto a las competiciones, el usuario verá un botón participar, con el que accede a una nueva pestaña con información adicional de la competición y con la posibilidad de apuntarse a la misma (En caso de que esté apuntado a esa competición el botón será diferente y no clicable).

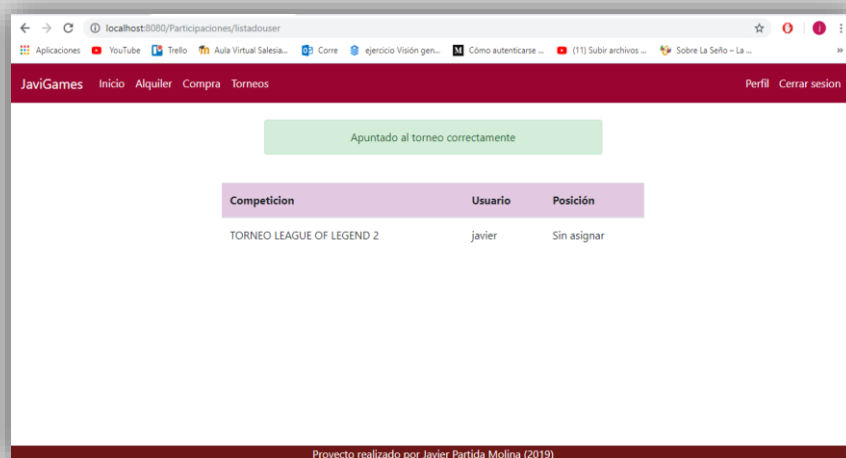


TORNEO LEAGUE OF LEGEND 2

Torneo de duración diaria. En función de la ronda en la que termines el torneo, obtendrás más oro. Recuerda que para poder jugar debes hacer el check-in 30 minutos antes del comienzo del torneo.

Lugar: Chiclana de la Frontera
Fecha: 2019-01-23

[Ya participas](#)



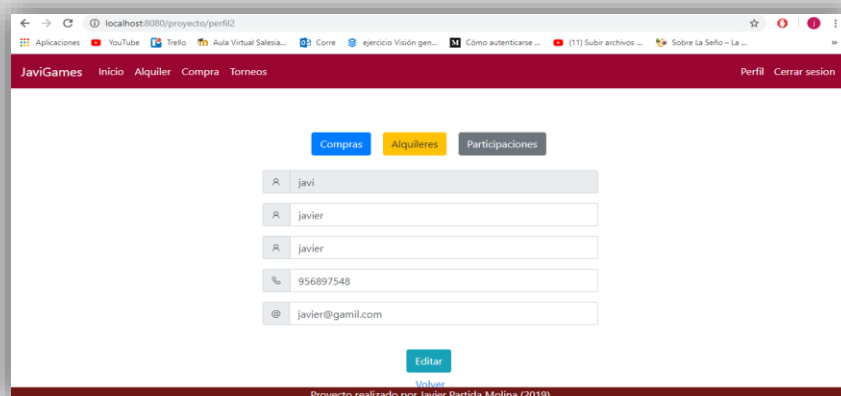
Apuntado al torneo correctamente

Competición	Usuario	Posición
TORNEO LEAGUE OF LEGEND 2	javier	Sin asignar

Proyecto realizado por Javier Partida Molina (2019)

- El usuario también podrá ver su perfil de usuario, en el cual también podrá editar sus datos y ver los listados (vistos anteriormente) de sus compras, alquileres y participaciones en campeonatos.

En caso de que uno de sus alquileres esté pasado de fecha, será aquí donde obtenga un aviso de que uno de sus alquileres ha expirado (los alquileres son de 3 días).



Compras Alquileres Participaciones

[Editar](#)

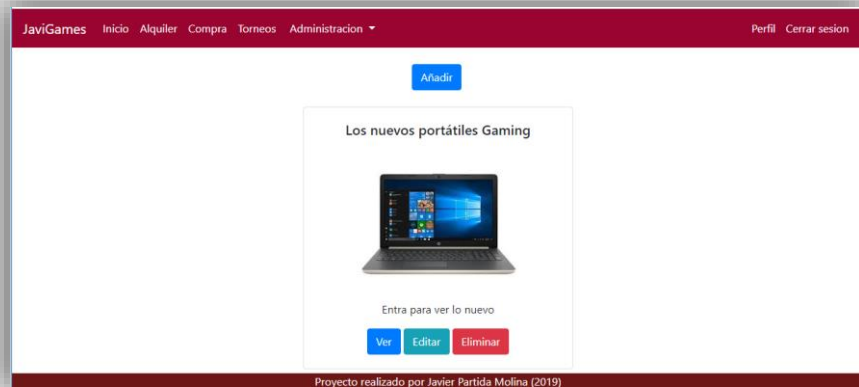
Proyecto realizado por Javier Partida Molina (2019)

- **Administrador:**

- Un administrador será como el usuario registrado visto anteriormente, pero tendrá en el menú superior acciones extras.



- En los apartados Noticias, Juegos, Categorías y Plataformas, el administrador verá cada uno de los elementos que existan y podrá realizar todo lo que envuelve un CRUD.



- En el apartado Usuarios, el administrador habilitará y deshabilitará usuarios y también los eliminará.

JaviGames Inicio Alquiler Compra Torneos Administracion ▾					Perfil Cerrar sesion
Usuario	Nombre	Email	Telefono	Opciones	
1234	1234	1234@1234.com	1234	Deshabilitar	✖
admin	admin	admin@admin.es	123456789	Deshabilitar	✖
byrench	byrench	byrench@byrench.com	byrench	Deshabilitar	✖
javi	javier	javier@gamil.com	956897548	Deshabilitar	✖
javier	Anama Maria	javier.partida.molina@gmail.com	650482054	Deshabilitar	✖
javier3	Anama Maria	javier.partida.molina@gmail.com	650482054	Deshabilitar	✖

Proyecto realizado por Javier Partida Molina (2019)

- En los apartados Alquileres y Ventas, el administrador verá todas las ventas y alquileres y también podrá realizar devoluciones.

JaviGames Inicio Alquiler Compra Torneos Administracion ▾					Perfil Cerrar sesion
Juego	Comprador	Fecha			
GTA V	admin	2019-02-13	Devolver		
GTA V	javier	2019-02-14	Devolver		

Proyecto realizado por Javier Partida Molina (2019)

- En los apartados Competiciones el administrador verá cada uno de los elementos que existan y podrá realizar todo lo que envuelve un CRUD, además de poder ver las participaciones existentes en una competición en concreto. Dentro de ellas podrá asignar las posiciones de los participantes en caso de que no se haya echo con anterioridad.

JaviGames Inicio Alquiler Compra Torneos Administracion ▾					Perfil Cerrar sesion
Añadir					
<div> <div>TORNEO LEAGUE OF LEGEND 2</div> <div>Torneo de duración diaria. En función de la ronda en la que termines e...</div> <div> Editar Participaciones Eliminar </div> </div>					
<div> <div>TORNEO FORNITE</div> <div>Nuevo torneo de fornite llega a nuestra tienda, no dudes en apuntarte</div> <div> Editar Participaciones Eliminar </div> </div>					

Proyecto realizado por Javier Partida Molina (2019)

Competición	Usuario	Posición
TORNEO LEAGUE OF LEGEND 2	Anama Maria	6º
TORNEO LEAGUE OF LEGEND 2	byrench	1º
TORNEO LEAGUE OF LEGEND 2	admin	2º
TORNEO LEAGUE OF LEGEND 2	javier	0 <input type="button" value="Asignar"/>

- Para la gestión en el front-end se utiliza thymeleaf. Dichas gestiones son:
 - Mostrar datos: **th:text**="*dato que queremos mostrar*"
 - Iterar: **th:each**="*nombre con el que accederemos a los datos : elemento que queremos iterar*".
 - Imágenes: **th:src**="*@{ruta de la imagen}*"
 - Rutas: **th:href**="*@{ruta a la que queremos acceder}*"
 - Formularios:
 - **th:action**="*@{ruta que indica la acción del formulario}*"
 - **th:object**="*nombre del objeto que se le pasa a la vista*"
 - **th:field**="**{nombre del campo al que queremos dar el valor}*"
 - **th:value**="*valor que queremos mostrar*"

5. LOGS

- La integración de logs es un aspecto bastante importante, ya que podemos mandar información a la consola de lo que necesitemos. Para ayudarlos en el front-end, utilizo los logs para mandar información de que acción se está ejecutando, que datos se están pasando y a que vista.
 - Creación de un log:

```
private static final Log LOG = LoggerFactory.getLog(HomeController.class);
```

- Utilizando un log:

```
LOG.info("Ver noticia " + noticia_id + " VER_NOTICIA_VIEW");
```

6. PATRÓN DAO

- Como bien sabemos, el patrón DAO, es el encargado de la conexión entre nuestra aplicación web y una base de datos. Todo esto se consigue muy fácilmente en Spring gracias a Hibernate y Jpa.
- Entidades:
 - En las entidades mediante anotaciones de hibernate configuramos la base de datos:
 - Anotaciones:
 - **@Entity**: para indicar que es una entidad
 - **@Table(name="nombre de la tabla")**
 - **@Id**: indicamos que el siguiente campo será el id.
 - **@GeneratedValue(strategy = GenerationType.IDENTITY)**: Se encarga de auto incrementar el valor del id.
 - **@Column(name="nombre de la columna")**: Indicamos las columnas de nuestra tabla.
 - Ejemplo en la entidad cars:

```
@Entity
@Table(name="noticias")
public class Noticias {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id", unique=true, nullable=false)
    public int id;

    @Column(name="titulo", nullable=false, length=60)
    private String titulo;

    @Column(name="resumen", nullable=false, length=120)
    private String resumen;

    @Column(name="cuerpo", nullable=false, length=900)
    private String cuerpo;

    @Column(name="foto", nullable=false, length=900)
    private String foto;
```

- Cada una de las variables tienen sus correspondientes getters y setters.

- La clase también tendrán un constructor vacío y otro con todas las variables, además del método toString.

- Relaciones:

- Nuestra base de datos (que explicaremos en el siguiente punto), tiene varias relaciones, las cuales son de tipo ManyToMany, pero como aprendimos en el anterior trabajo, en caso de tener campos extras se realizaba de forma diferente:

- Con campos extras:

```
@OneToMany(mappedBy = "juego", cascade=CascadeType.ALL)
private List<Ventas> Ventas = new ArrayList<Ventas>();
```

```
@ManyToOne
@JoinColumn(name = "idjuego")
private Juegos juego;
```

- Sin campos extras:

```
@ManyToMany(fetch = FetchType.LAZY, cascade = {CascadeType.MERGE, CascadeType.PERSIST})
@JoinTable(name = "juegoCategorias",
    joinColumns = @JoinColumn(name = "idJuegos"),
    inverseJoinColumns = @JoinColumn(name = "idCategorias")
)
private List<Categorias> categorias = new ArrayList<>();
```



```
@ManyToMany(mappedBy="plataformas")
private List<Juegos> juegos = new ArrayList<>();
```

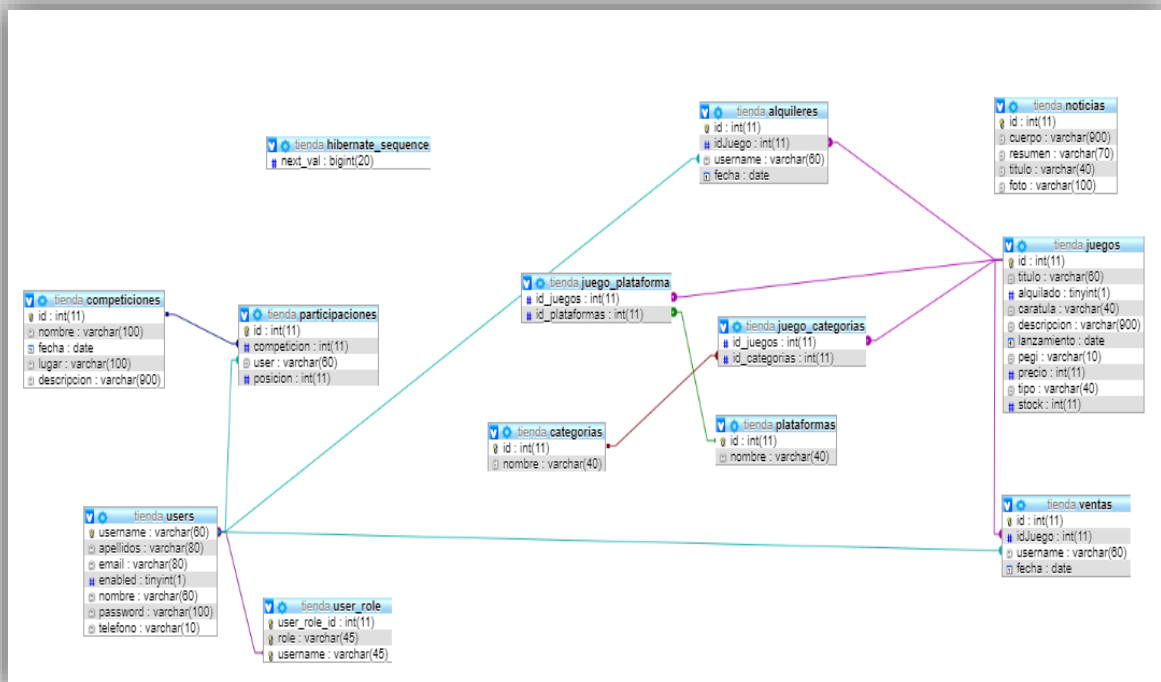
- Repositorio:

- o Gracias a Jpa, en Spring podemos realizar consultas a los campos, guardar, actualizar y eliminar muy fácilmente de la tabla que queramos.

```
@Repository("UsersJpaRepository")
public interface UsersJpaRepository extends JpaRepository<User, Serializable> {

    public abstract User findByUsername(String username);
}
```

- Base de datos:



7. PATRÓN DTO Y DOZER

- Como bien sabemos el patrón DTO, consiste en la conversión de entidad a modelo y de modelo a entidad, ya que por temas de seguridad es mejor no trabajar directamente con la base de datos.
- Los modelos son muy similares a la entidad, pero solo que no llevan ningún tipo de anotación, es decir solo se declaran variables, métodos getters y setters, método constructor vacío y otro con todas las variables, y el método toString. Ya que el modelo solo se encarga de interactuar con las vistas y no con la base de datos.
- Gracias al componente Dozer podemos realizar esta conversión muy fácilmente.

```
@Component("CategoriasConverter")
public class CategoriasConverter {

    DozerBeanMapper mapper = new DozerBeanMapper();

    public CategoriasModel entity2model(Categorias categoria) {
        return mapper.map(categoria, CategoriasModel.class);
    }
    public Categorias model2entity(CategoriasModel categoriaModel) {
        return mapper.map(categoriaModel, Categorias.class);
    }
}
```

- Como vemos en pocas líneas realizamos las dos conversiones. En nuestro caso, la conversión de la entidad y modelo de juegos, no se ha podido realizar con este componente, ya que tanto las categorías y las plataformas inicialmente en la entidad son de tipo List<Plataformas/Categorias> y en el modelo son un array de strings. Esto es debido a que a la hora de añadir un nuevo juego, las categorías y las plataformas son un select multiple, el cual te devuelve un array de strings. Entonces a la hora de la conversión tenemos que ir obteniendo cada uno de los String y pasarlo a un tipo categoría y guardarlos en la lista.

```
public JuegosModel entity2model(Juegos juegos) {
    JuegosModel juegomodel = new JuegosModel();
    juegomodel.setId(juegos.getId());
    juegomodel.setTitulo(juegos.getTitulo());
    juegomodel.setTipo(juegos.getTipo());
    juegomodel.setPrecio(juegos.getPrecio());
    juegomodel.setPegi(juegos.getPegi());
    juegomodel.setLanzamiento(juegos.getLanzamiento());
    juegomodel.setDescripcion(juegos.getDescripcion());
    juegomodel.setCaratula(juegos.getCaratula());
    juegomodel.setAlquilado(juegos.isAlquilado());
    juegomodel.setStock(juegos.getStock());

    List<Categorias> categorias = juegos.getCategorias();
    String[] arrayCategorias = new String[categorias.size()];

    for (int i = 0; i < categorias.size(); i++) {
        Categorias categoria = categorias.get(i);
        arrayCategorias[i] = categoria.getNombre();
    }

    juegomodel.setCategorias(arrayCategorias);

    List<Plataformas> plataformas = juegos.getPlataformas();
    String[] arrayPlataformas = new String[plataformas.size()];

    for (int i = 0; i < plataformas.size(); i++) {
        Plataformas plataforma = plataformas.get(i);
        arrayPlataformas[i] = plataforma.getNombre();
    }

    juegomodel.setPlataformas(arrayPlataformas);

    return juegomodel;
}
```

```
public Juegos model2entity(JuegosModel juegosmodel) {
    Juegos juego = new Juegos();
    juego.setId(juegosmodel.getId());
    juego.setTitulo(juegosmodel.getTitulo());
    juego.setTipo(juegosmodel.getTipo());
    juego.setPrecio(juegosmodel.getPrecio());
    juego.setPegi(juegosmodel.getPegi());
    juego.setLanzamiento(juegosmodel.getLanzamiento());
    juego.setDescripcion(juegosmodel.getDescripcion());
    juego.setCaratula(juegosmodel.getCaratula());
    juego.setAlquilado(juegosmodel.isAlquilado());
    juego.setStock(juegosmodel.getStock());

    List<Categorias> categorias = new ArrayList<Categorias>();
    String[] arrayCategorias = juegosmodel.getCategorias();
    for (int i = 0; i < arrayCategorias.length; i++) {
        String name = arrayCategorias[i];
        categorias.add(CategoriasJpaRepository.findByNombre(name));
    }

    juego.setCategorias(categorias);

    List<Plataformas> plataformas = new ArrayList<Plataformas>();
    String[] arrayPlataformas = juegosmodel.getPlataformas();
    for (int i = 0; i < arrayPlataformas.length; i++) {
        String name = arrayPlataformas[i];
        plataformas.add(PlataformasJpaRepository.findByNombre(name));
    }

    juego.setPlataformas(plataformas);

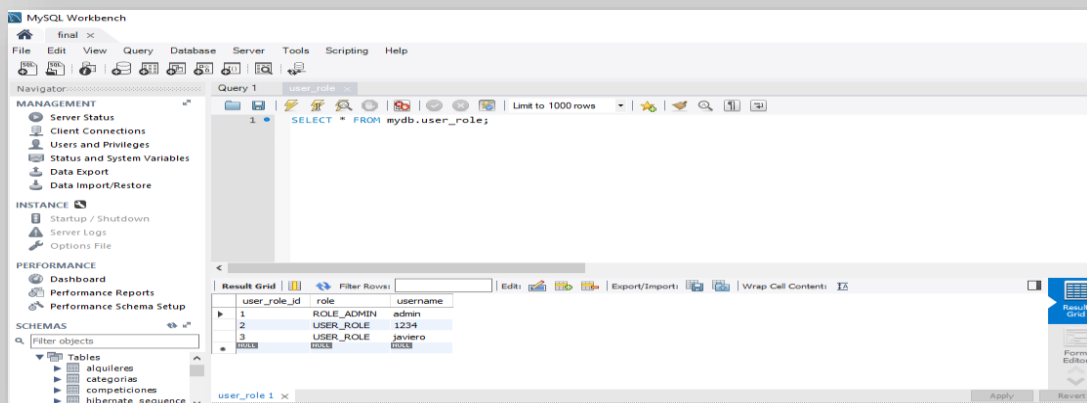
    return juego;
}
```

8. BASE DE DATOS REMOTA

- Para conectar nuestro proyecto a una base de datos externa necesitaremos cambiar las propiedades del application.properties.
 - o Cambiaremos donde poníamos localhost pondremos la ip de nuestro servidor y sin olvidarnos de poner correctamente el nombre y contraseña de acceso.

```
spring.datasource.url= jdbc:mysql://35.234.64.252:3306/mydb
spring.datasource.username= root
spring.datasource.password=yonacienel98
```

- o Para la gestión de nuestra base de datos de manera directa, utilizo MysqlWorkBench, con el cual podremos acceder a nuestra base de datos remota y realizar cualquier tipo de cambio.



9. IMÁGENES

- Para trabajar con imágenes, es necesario crear en nuestro servicio un método que guarde nuestra foto en nuestro proyecto.

```
@Override
public void savePhoto(MultipartFile file) throws IOException {
    if(!file.isEmpty()) {
        byte[] bytes = file.getBytes();
        String ruta = "../src/main/resources/static/imgs/";
        Path path = Paths.get(ruta + file.getOriginalFilename());
        Files.write(path, bytes);
        LOG.info(ruta + file.getOriginalFilename());
    }
}
```

- Además de añadir unas líneas a nuestro application.properties, para que se actualicen automáticamente las imágenes subidas a nuestro proyecto.

```
8 spring.resources.static-locations[0]=file:src/main/resources/static/
9 spring.resources.static-locations[1]=classpath:/static/
```

- Luego en el controlador se comprueba si el usuario ha subido alguna foto en el formulario, en caso de que no se pondrá una foto por defecto.

```
@PostMapping("/addNoticiasForm")
public String addNoticiasForm(Model model, @ModelAttribute(name="noticia") NoticiasModel noticiasModel, @RequestParam("file") MultipartFile file) {
    LOG.info("AÑADIENDO NOTICIA " + noticiasModel);
    NoticiasService.savePhoto(file);
    LOG.info("NOMBRE: " + file.getOriginalFilename());
    if(noticiasModel.getTitulo().isEmpty()) {
        model.addAttribute("noticia", new NoticiasModel());
        return "redirect:/noticias/addNoticias";
    } else {
        if(file.getOriginalFilename().isEmpty()) {
            if(noticiasModel.getFoto().isEmpty()) {
                noticiasModel.setFoto("nophoto.PNG");
            }
        } else {
            noticiasModel.setFoto(file.getOriginalFilename());
        }
        NoticiasService.addNoticias(noticiasModel);
        return "redirect:/noticias/opciones";
    }
}
```

10. JAVA 8

- En este proyecto se aprovecha la potencia de java 8 para realizar los listados, es decir, obtener todos los datos de una tabla. Para ello utilizo las expresiones lambda.

```
@Override
public List<JuegosModel> listAllJuegos() {
    List<JuegosModel> JuegosList = new ArrayList<>();
    LOG.info("FIND ALL" + JuegosJpaRepository.findAll());
    JuegosJpaRepository.findAll().forEach(Juegos ->{JuegosList.add(JuegosConverter.entity2model(Juegos));});
    LOG.info("ARRAY ALL" + JuegosList);
    return JuegosList;
}
```

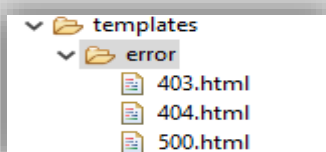
```
@Override
public List<JuegosModel> listAlquilerJuegos() {
    List<JuegosModel> JuegosList = new ArrayList<>();
    LOG.info("FIND ALL" + JuegosJpaRepository.findAll());
    JuegosJpaRepository.findAll().forEach(Juegos ->{

        if(Juegos.getTipo().equals("1") && !Juegos.isAlquilado() ) {
            JuegosList.add(JuegosConverter.entity2model(Juegos));
        }
    });
    LOG.info("ARRAY ALL" + JuegosList);
    return JuegosList;
}
```

- A demás podemos añadirle alguna condición. Solo tendríamos que añadir la condición dentro de las llaves, donde se le asigna a la variable (en este caso Juegos) con ayuda de la expresión lambda ->. Por ejemplo, en este caso solo obtenemos los juegos de tipo alquiler.

11. Páginas de errores

- Las páginas de errores que han sido configuradas, han sido las siguientes:
 - **404:** Este error consiste, en que la ruta a la que hemos intentado acceder no existe.
 - **500:** Este error consiste, en que el servidor a encontrado un error y por lo cual no puede cargar nada (Suele ocurrir por errores en el código o por datos de formularios incorrectos, como por ejemplo escribir letras en un campo que realmente es de tipo Integer).
 - **403:** Este error consiste, en que el usuario no tiene permiso para acceder a esa parte de la aplicación.



- En Spring es muy fácil controlar estos errores. Solo es necesario crear una carpeta error dentro de nuestra carpeta templates y añadir cada uno de los html dentro de ella.

12. CONTROL DE ERRORES

- Los posibles errores que se podrían haber dado y se han controlado son los siguientes:
 - **Apuntarse a competiciones:** Se controla que un usuario que ya esté inscrito en una competición, no pueda volver a apuntarse en la misma. Para ello realizamos una consulta a la base de datos buscando si existe alguna participación de un usuario en concreto en una competición en concreto. En caso de que exista, se le enviará un mensaje al usuario de que ya estaba inscrito en esa competición.
 - **Alquilar juegos:** Los usuarios, no podrán alquilar ningún juego que este alquilado en ese momento, ya que tanto en el html como en el servicio se realizan comprobaciones. En el html simplemente no aparecerá el juego y en caso de intentarlo por url, el servicio se encargará de cortar el intento de alquiler.
 - **Comprar juegos:** Los usuarios, no podrán comprar ningún juego que se stock sea menor que uno, ya que además de no aparecer en el html, en el servicio se realiza la comprobación necesaria para evitar intentos desde url.
 - **Registro:** En el registro se controla que el nombre de usuario sea único, para ello al enviar el formulario, en el controlador, se comprueba si existe o no algún usuario con ese nombre en nuestra base de datos. En caso de que exista se le notificará al usuario en la vista.

13. SPRING SECURITY

- Dependencias necesarias para spring security:
 - **Spring security:**

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```
 - **Spring security (thymeleaf):**

```
<dependency>
  <groupId>org.thymeleaf.extras</groupId>
  <artifactId>thymeleaf-extras-springsecurity4</artifactId>
  <version>3.0.4.RELEASE</version>
</dependency>
```

- Spring security requiere una serie de configuraciones, la cuales en este proyecto son realizadas en el archivo **SecurityConfiguration.java** que se encuentra en la carpeta **configuration**.

SecurityConfiguration:

- En este archivo configuramos varias opciones:

- **Autorización de acceso:** Aquí configuraremos las urls a las que un usuario anónimo puede acceder. Es muy importante añadir tanto el css como las imágenes ya que son recursos básicos de la aplicación.

También se añaden las siguientes:

```
http.authorizeRequests().antMatchers("/css/**", "/images/**", "/log/**", "/juegos/**", "/proyecto/home", "/proyecto/**", "/Competiciones/**", "/RestVocitias/**", "/RestVocitias/update/**").permitAll()  
.anyRequest().authenticated()  
.and()
```

- **Inicio de sesión:** Para el inicio de sesión, especificamos nuestra url que nos manda al html que contiene el login, la url que procesará el inicio de sesión para comprobar las credenciales, los parámetros necesarios para la autenticación (Nombre de usuario y contraseña) y la url a la que queremos que se nos redirija cuando el inicio de sesión sea satisfactorio.

```
.formLogin().loginPage("/log/login").loginProcessingUrl("/logincheck")  
.usernameParameter("username").passwordParameter("password")  
.defaultSuccessUrl("/proyecto/home").permitAll()  
.and()
```

- **Cerrar sesión:** Para el cierre de sesión, especificamos la url que procesa el cierre de sesión, y por último la url para el cierre de sesión satisfactorio.

```
.logout().logoutUrl("/log/logout").logoutSuccessUrl("/log/login?logout")  
.permitAll();
```

Secularización de métodos:

- Con Spring security podemos “capar” nuestros métodos, para que ningún usuario pueda acceder a una funcionalidad a la que no queramos que acceda, por ejemplo, un usuario que no tenga un rol determinado.

En caso de intentar acceder a esta url, sin que el usuario tenga el rol admin, obtendremos un error 403.

```
@PreAuthorize("hasRole('ROLE_ADMIN')")
@GetMapping("/listado")
public ModelAndView show() {
    LOG.info("LISTA DE CATEGORIAS VIEW: LIST_CATEGORIAS");
    ModelAndView mav = new ModelAndView(views.LIST_CATEGORIAS);
    mav.addObject("categorias", CategoriasService.listAllCategorias());
    return mav;
}
```

Secularización de vistas:

- Gracias a la dependencia thymeleaf extras especificadas anteriormente, podemos secularizar nuestras vistas. Con las siguientes etiquetas podremos ocultar partes del html o mostrar. Las comprobaciones que se pueden realizar son:

- o Si el usuario tiene un rol en específico.

```
<span sec:authorize="hasRole('ROLE_ADMIN')">
```

- o Si está logueado.

```
<a sec:authorize="isAuthenticated()">
```

- o Si es un usuario anónimo.

```
<button sec:authorize="isAnonymous()">
```

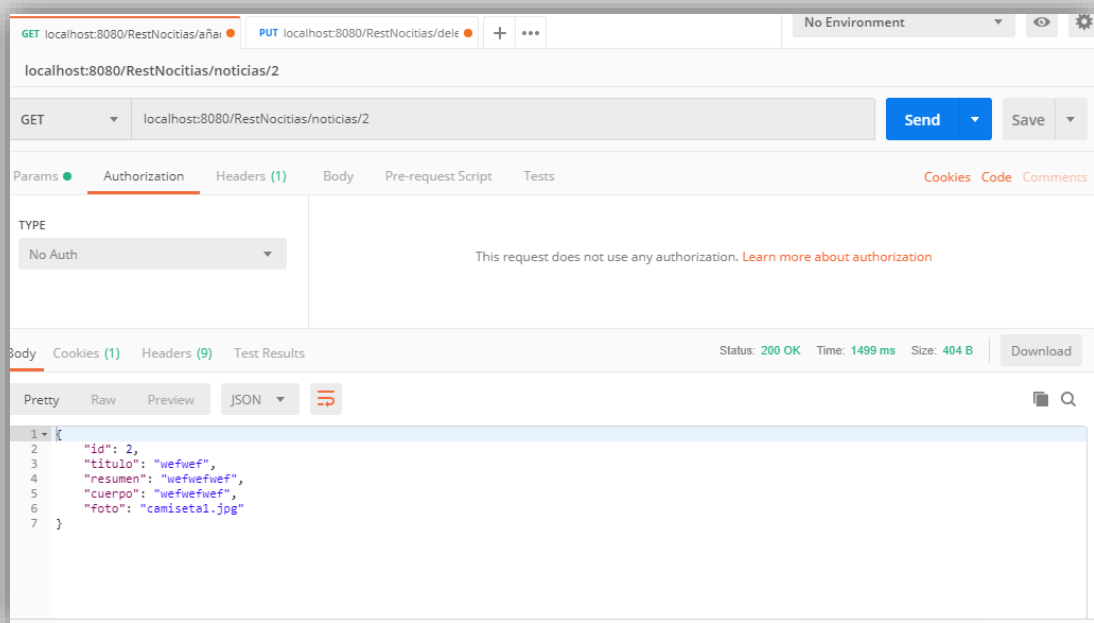
- o A demás también podemos obtener datos del usuario logueado actualmente, por ejemplo, en la siguiente imagen mostraría el nombre del usuario.

```
<span sec:authentication="principal.username"><
```


14. SPRING REST

- La integración de una Api Rest en el proyecto Spring es muy sencillo, ya que solo basta con crear un nuevo controlador y utilizar las etiquetas correspondientes para ello.
- En mi caso decido realizar dicha Api Rest en la entidad Noticias, ya que es la única que no posee ningún tipo de relación y su complejidad es mucho menor.
- Básicamente se crean los métodos para realizar un CRUD, con las anotaciones de Rest y el método llamará a el método correspondiente del servicio.
- La aplicación elegida para consumir la Api Rest es Postman.
- Etiquetas necesarias son:
 - o **@RestController**: Anotación que le “dirá” a Spring que este controlador es una Api Rest.
 - o **@GetMapping("/noticias")**: Esta anotación se encarga de obtener datos.
 - o **@PostMapping("/add")**: Esta anotación se encarga de añadir, es decir, un POST.
 - o **@PutMapping("/update/{id}")**: Esta anotación se encarga de editar.
 - o **@DeleteMapping("/delete/{id}")**: Esta anotación se encarga de eliminar.
- Ejemplo de método y funcionamiento:

```
@GetMapping("/noticias/{id}")  
public Noticias getNoticias2(@PathVariable int id){  
    return NoticiasService.findNoticiasByNoticiaId(id);  
}
```



15. WebGrafia

- <https://spring.io/>
- <https://es.stackoverflow.com/>
- <https://www.w3schools.com/>
- <https://www.youtube.com/?hl=es&gl=ES>
- <https://www.getpostman.com/>
- <http://www.programandoconcafe.com>
- <https://cleventy.com>
- <https://cloud.google.com/?hl=es>

Proyecto realizado por: Javier Partida Molina