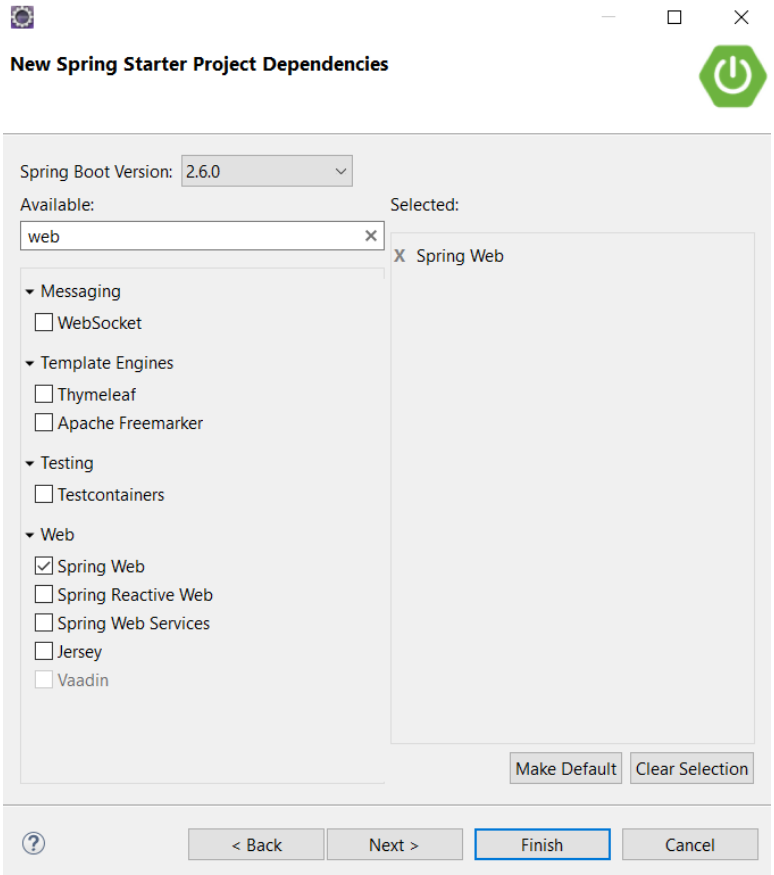
ACTIVIDAD 3 - SERVICIOS REST

REQUERIMIENTO 1:

En este apartado tenemos que crear un servicio REST que gestione una serie de videojuegos que se encontrarán alojados en el REST y tendrá 5 videojuegos preestablecidos con los datos rellenos.

Primero de todo tenemos que crear un Spring Starter Proyect, donde será necesario añadir el Spring Starter Spring Web a la hora de inicializarlo.



Para ello crearemos una clase videojuego alojada en el paquete serviciorest.entidad con las propiedades necesarias, con sus métodos getter y setter, el constructor y el método toString.

```
//CREAMOS LA CLASE CON LA ENTIDAD QUE SERÁ UN VIDEOJUEGO
//CREAMOS EL CONSTRUCTOR
```

```
//CREEAMOS GETTER Y SETTER
//CREAMOS MÉTODO TO STRING

public class Videojuego{

  @Override
  public int hashCode() {
    return Objects.hash(compañia, id, nombre, nota);
  }

  @Override
  public boolean equals(Object obj) {
    if (this == obj)
      return true;
    if (obj == null)
      return false;
    if (getClass() != obj.getClass())
      return false;
    Videojuego other = (Videojuego) obj;
    return Objects.equals(compañia, other.compañia) && id == other.id && Objects.equals(nombre, other.nombre)
        && nota == other.nota;
  }

  private int id;
  private String nombre;
  private String compañia;
  private int nota;

  public Videojuego() {
    super();
  }

  public Videojuego(int id, String nombre, String compañia, int nota) {
    this.id = id;
    this.nombre = nombre;
    this.compañia = compañia;
    this.nota = nota;
  }

  public int getId() {
    return id;
  }
  public void setId(int id) {
    this.id = id;
  }

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

  public String getCompañia() {
    return compañia;
  }
  public void setCompañia(String compañia) {
    this.compañia = compañia;
  }

  public int getNota() {
    return nota;
  }
  public void setNota(int nota) {
    this.nota = nota;
  }

  @Override
  public String toString() {
    return "Videojuego [id=" + id + ", nombre=" + nombre + ", compañia=" + compañia + ", nota=" + nota + "]";
  }

}
```

La clase application será la encargada de arrancar la aplicación con el método run.

```
//AQUÍ ARRANCARÁ NUESTRA APLICACIÓN
//CON LA ANOTACIÓN @SpringBootApplication HACEMOS QUE SPRINGBOOT:
```

```
//1. BUSQUE ANOTACIONES .
//2. SE AUTOCONFIGURE
//3. BUSCA METODOS CON @BEAN

@SpringBootApplication
public class Application {

  public static void main(String[] args) {
    System.out.println("Servicio Rest -> El contexto de Spring se está cargando, espere unos instantes....");
    //CON EL MÉTODO RUN ARRANCAREMOS EL CONTEXTO SPRING
    ConfigurableApplicationContext ctx = SpringApplication.run(Application.class, args);
    System.out.println("Servicio Rest -> Contexto de Spring cargado!");
  }

}
```

A continuación, crearemos la clase de persistencia que llamaremos daoVideojuego. DAO es el objeto que se encargará de hacer las consultas con la anotación *@Component.* Crearemos una lista con los videojuegos preestablecidos.

Crearemos sus métodos como puede ser buscar y borrar en función de su ID, añadir nuevos videojuegos o actualizar los datos de uno ya creado.

```
/**
 * DAO ES EL OBJETO QUE SE ENCARGA DE HACER LAS CONSULTAS
 * CON LA ANOTACIÓN @Component, DAMOS DE ALTA EL OBJETO ÚNICO
 *
 */
@Component
public class DaoVideojuego {

  public List<Videojuego> listaVideojuegos;
  public int contador;

  /**
   * LISTA DONDE ESTARÁN LOS VIDEOJUEGOS QUE PODRÁN CONSUMIR NUESTROS CLIENTES
   */
  public DaoVideojuego() {
    System.out.println("DaoVideojuego -> Creando la lista de videojuegos!");
    listaVideojuegos = new ArrayList<Videojuego>();
    Videojuego p1 = new Videojuego(contador++,"FORTNITE", "EPIC GAMES", 7);//ID: 0
    Videojuego p2 = new Videojuego(contador++,"FIFA 21", "EA", 8);//ID: 1
    Videojuego p3 = new Videojuego(contador++,"CALL OF DUTTY", "ACTIVISION", 9);//ID: 2
    Videojuego p4 = new Videojuego(contador++,"AMONG US", "INNERSLOTH", 6);//ID:3
    Videojuego p5 = new Videojuego(contador++,"NBA 2K21", "2K SPORTS", 7);//ID:4

    listaVideojuegos.add(p1);
    listaVideojuegos.add(p2);
    listaVideojuegos.add(p3);
    listaVideojuegos.add(p4);
    listaVideojuegos.add(p5);

  }

  /**
   *DEVUELVE EL VIDEOJUEGO EN FUNCIÓN DE SU POSICIÓN
   *
   */
  public Videojuego get(int id) {

      for(Videojuego p : listaVideojuegos) {
        if(p.getId() == id) {
          return p;
        }
      }
    return null;
  }

  /**
   * DEVUELVE TODOS LOS VIDEOJUEGOS
   */
  public List<Videojuego> list() {
    return listaVideojuegos;
  }

  /**
   * MÉTODO AÑADIR VIDEOJUEGOS
   */
```

```
  public String add(Videojuego p) {

    for(Videojuego v : listaVideojuegos) {
      if (v.getNombre().equalsIgnoreCase(p.getNombre())) {
        return null;
      }
    }
    p.setId(contador++);
    listaVideojuegos.add(p);
    return "añadido";
  }

  /**
   * BORRAR UN VIDEOJUEGO
   */
    public Videojuego delete(int id) {
    try {

      Videojuego vAux = get(id);

      int posicion = listaVideojuegos.indexOf(vAux);

      return listaVideojuegos.remove(posicion);

    } catch (Exception e) {
      System.out.println("delete -> Videojuego fuera de rango");
      return null;
    }
  }

  /**
   * MÉTODO QUE MODIFICA UN VIDEOJUEGO
   */
  public Videojuego update(Videojuego p) {
    try {
        Videojuego pAux = get(p.getId());
        pAux.setNombre(p.getNombre());
        pAux.setCompañia(p.getCompañia());
        pAux.setNota(p.getNota());

      return pAux;
    } catch (IndexOutOfBoundsException iobe) {
      System.out.println("update -> VIDEOJUEGO fuera de rango");
      return null;
    }
  }
}
```

Crearemos una clase controlador donde creando un objeto dao con la anotación *@Autowired* crearemos inyecciones de dependencias.

Se creará un CRUD completo para ser consumido por un cliente y por POSTMAN y el intercambio se hará a través de JSON.

```
//CRUD COMPLETO
@RestController
public class ControladorVideojuego {

  //USAMOS @Autowired PARA HACER INYECCIONES DE DEPENDENCIAS

  @Autowired
  private DaoVideojuego daoVideojuego;

  //GET VIDEOJUEGO POR ID
  //SE DEVOLVERÁ UNA PERSONA POR ID
  //EL RESULTADO SERÁ JSON
  //DARÁ UN Nº DE RESPUESTA EN FUNCIÓN DE SI ES CORRECTO O NO

  @GetMapping(path="videojuegos/{id}",produces = MediaType.APPLICATION_JSON_VALUE)
  public ResponseEntity<Videojuego> getVideojuego(@PathVariable("id") int id) {
    System.out.println("Buscando videojuego con id: " + id);
    Videojuego p =  daoVideojuego.get(id);
    if(p != null) {
      return new ResponseEntity<Videojuego>(p,HttpStatus.OK);//200 OK
    }else {
      return new ResponseEntity<Videojuego>(HttpStatus.NOT_FOUND);//404 NOT FOUND
    }
  }
```

```
  //POST
  //DAR DE ALTA UN VIDEOJUEGO
  //EN ESTE MÉTODO TANTO SI INTRODUCIMOS UN ID O UN NOMBRE YA EXISTENTE NO LO AÑADIRÁ
  // Y SACARÁ UN BAD REQUEST EN POSTMAN ASÍ COMO UN MENSAJE POR CONSOLA ADVIRTIENDO DE QUE EL JUEGO YA EXISTE

  @PostMapping(path="videojuegos",consumes=MediaType.APPLICATION_JSON_VALUE,
      produces=MediaType.APPLICATION_JSON_VALUE)
  public ResponseEntity<Videojuego> altaVideojuego(@RequestBody Videojuego p) {
    System.out.println("altaVideojuego: objeto videojuego: " + p);

    String añadir = daoVideojuego.add(p);

    if (añadir != null ) {

      System.out.println("ESTÁS INTENTANDO DAR DE ALTA UN JUEGO YA EXISTENTE");

      return new ResponseEntity<Videojuego>(p,HttpStatus.CREATED);//201 CREATED
    }else {
      return new ResponseEntity<Videojuego>(HttpStatus.BAD_REQUEST);
    }

  }

  //GET LISTA VIDEOJUEGOS
  //PEDIREMOS TODOS LOS VIDEOJUEGOS
  @GetMapping(path="videojuegos",produces = MediaType.APPLICATION_JSON_VALUE)
  public ResponseEntity<List<Videojuego>> listarVideojuegos() {
    System.out.println(daoVideojuego.list());
    return new ResponseEntity<List<Videojuego>>(daoVideojuego.list(),HttpStatus.OK);
  }

  //PUT
  //ESTE MÉTODO SE UTILIZA PARA MODIFICAR UN VIDEOJUEGO
  @PutMapping(path="videojuegos/{id}",consumes = MediaType.APPLICATION_JSON_VALUE)
  public ResponseEntity<Videojuego> modificarVideojuego(
      @PathVariable("id") int id,
      @RequestBody Videojuego p) {
    System.out.println("ID a modificar: " + id);
    System.out.println("Datos a modificar: " + p);
    p.setId(id);
    Videojuego pUpdate = daoVideojuego.update(p);
    if(pUpdate != null) {
      return new ResponseEntity<Videojuego>(HttpStatus.OK);//200 OK
    }else {
      return new ResponseEntity<Videojuego>(HttpStatus.NOT_FOUND);//404 NOT FOUND
    }
  }

  //DELETE
  //MÉTODO PARA ELIMINAR UN VIDEOJUEGO
  @DeleteMapping(path="videojuegos/{id}")
  public ResponseEntity<Videojuego> borrarVideojuego(@PathVariable int id) {
    System.out.println("ID a borrar: " + id);
    Videojuego p = daoVideojuego.delete(id);
    if(p != null) {
      return new ResponseEntity<Videojuego>(p,HttpStatus.OK);//200 OK
    }else {
      return new ResponseEntity<Videojuego>(HttpStatus.NOT_FOUND);//404 NOT FOUND
    }
  }
}
```

En la carpeta application.properties asignaremos el puerto 8086 que será donde haremos las peticiones en POSTMAN.

```
server.port=8086
```

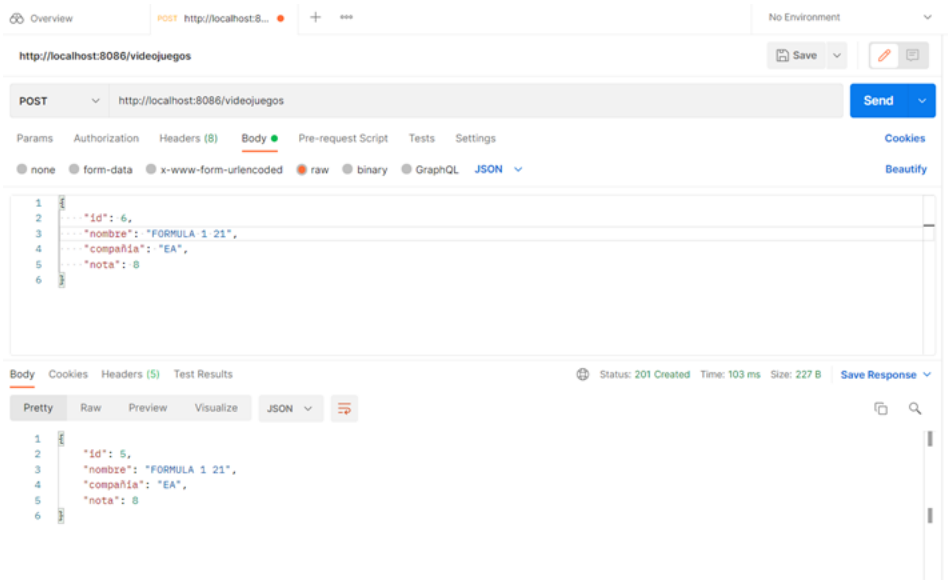
Tenemos que iniciar la aplicación ahora para arrancar el contexto de Spring:

Como podemos ver ha cargado sin errores por lo que ya podemos realizar la conexión con POSTMAN.

Nos hemos descargado la aplicación POSTMAN y vamos a realizar las peticiones requeridas:

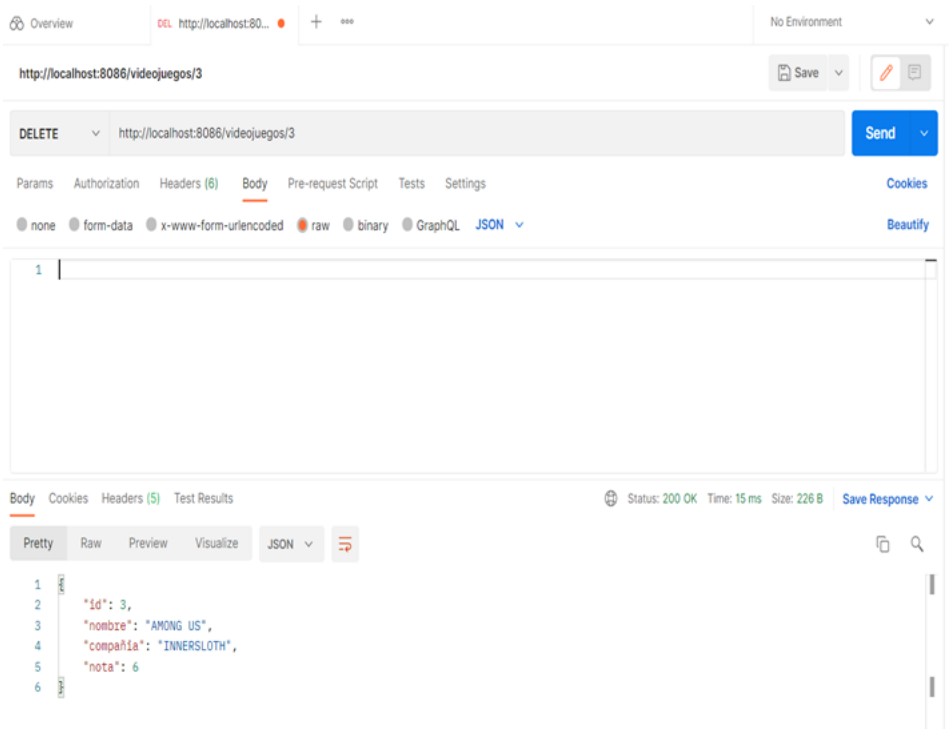
- Dar de alta un videojuego:

Mediante POST en POSTMAN damos de alta un videojuego. En la parte /8086 le estamos diciendo el puerto que queremos utilizar y al poner /videojuegos le decimos el Path que queremos utilizar, que lo hemos descrito en el método. La respuesta como podemos observar el 201 created.



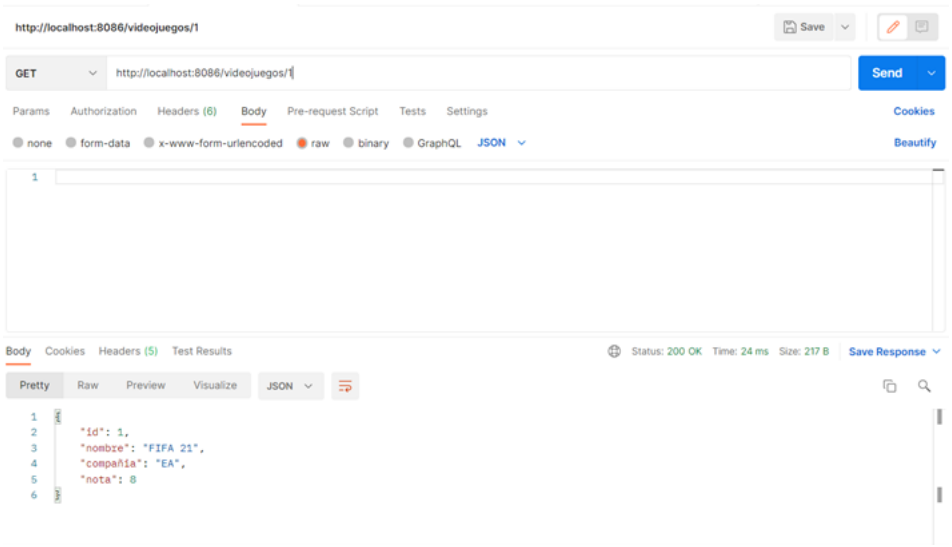
- Dar de baja un videojuego:

Para dar de baja un videojuego usaremos DELETE, tendremos que poner en el path después de /videojuegos el id que deseamos borrar. El estado que nos devuelve es 200 OK por lo tanto se ha realizado la operación.



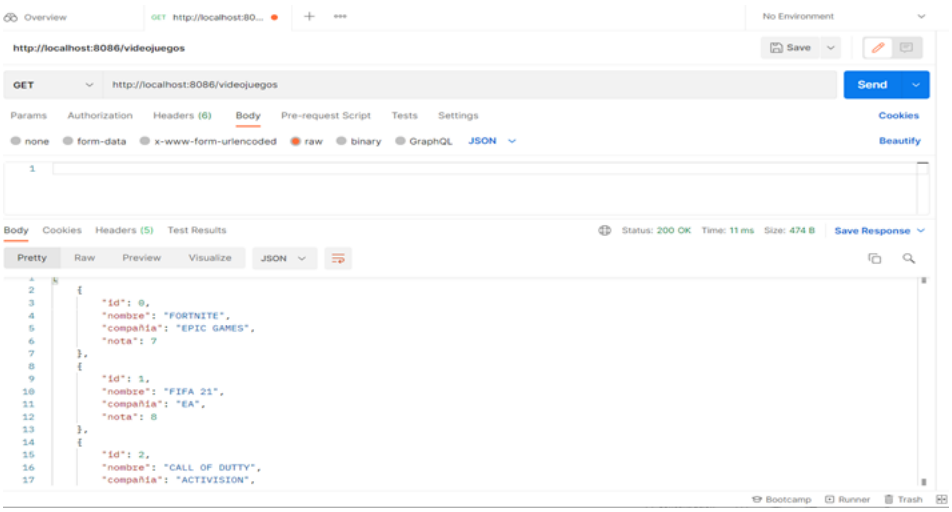
- Modificar un videojuego por ID:

Queremos modificar el videojuego NBA 2K21 que aparece abajo y cambiarle el nombre por el del año siguiente NBA 2K22. Para ello utilizaremos PUT y en el path ponderemos tras /videojuego el id del que queremos modificar. Como podemos ver el estado es 200 OK por lo tanto se ha hecho el cambio.



- Listar todos los videojuegos:

Para ello simplemente con GET lo que haríamos sería no poner el id tras /videojuegos para que nos saque todos los de la lista.



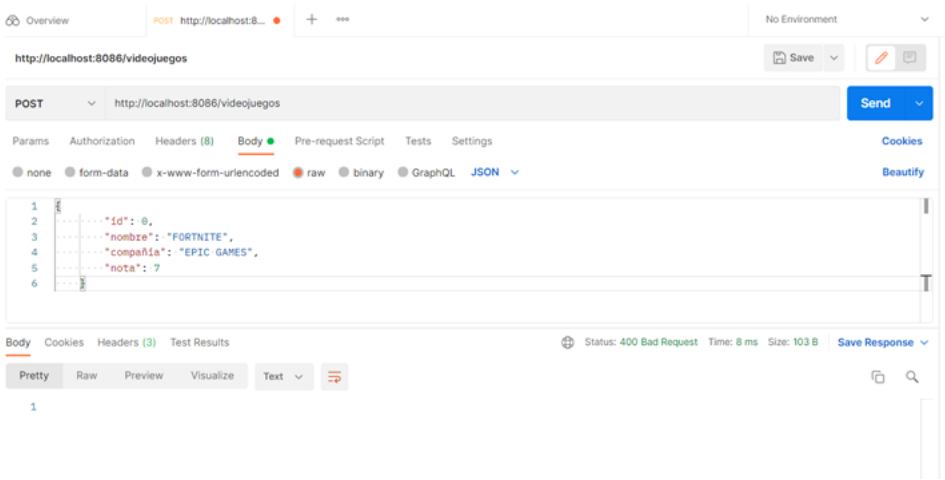
REQUERIMIENTO 2:

Se nos pide que no pueda haber dos videojuegos con id o nombre repetido.

Con este método tanto si introducimos un juego con un id o con un nombre ya existente nos saldrá un BAD REQUEST en POSTMAN y un mensaje por consola advirtiendo del error.

```
//POST
  //DAR DE ALTA UN VIDEOJUEGO
  //EN ESTE MÉTODO TANTO SI INTRODUCIMOS UN ID O UN NOMBRE YA EXISTENTE NO LO AÑADIRÁ
  // Y SACARÁ UN BAD REQUEST EN POSTMAN ASÍ COMO UN MENSAJE POR CONSOLA ADVIRTIENDO DE QUE EL JUEGO YA EXISTE

  @PostMapping(path="videojuegos",consumes=MediaType.APPLICATION_JSON_VALUE,
      produces=MediaType.APPLICATION_JSON_VALUE)
  public ResponseEntity<Videojuego> altaVideojuego(@RequestBody Videojuego p) {
    System.out.println("altaVideojuego: objeto videojuego: " + p);

    String añadir = daoVideojuego.add(p);

    if (añadir != null ) {

      System.out.println("ESTÁS INTENTANDO DAR DE ALTA UN JUEGO YA EXISTENTE");

      return new ResponseEntity<Videojuego>(p,HttpStatus.CREATED);//201 CREATED

    }else {
      return new ResponseEntity<Videojuego>(HttpStatus.BAD_REQUEST);
    }

  }
```

En este caso podemos apreciar como nos aparece un BAD REQUEST al intentar introducir un juego con un id y nombres ya existentes



Aquí aparece el mensaje por pantalla del error:

```
[Videojuego [id=0, nombre=FORTNITE, compañia=EPIC GAMES, nota=7], Videojuego [id=1, nombre=FIFA 21, compañia=EA, nota=8], Videojuego [id=2, nombre=CALL OF DU
altaVideojuego: objeto videojuego: Videojuego [id=0, nombre=FORTNITE, compañia=EPIC GAMES, nota=7]
ESTÁS INTENTANDO DAR DE ALTA UN JUEGO YA EXISTENTE
```

REQUERIMIENTO 3:

Se pide crear una aplicación java que sea capaz de trabajar con el servidor REST de videojuegos. La aplicación mostrara un menú que sea capaz de realizar todas las operaciones del servidor, como puede ser:

- Dar de alta un videojuego
- Dar de baja un videojuego por ID
- Modificar un videojuego por ID
- Obtener un videojuego por ID
- Listar todos los videojuegos
- Salir

Para cada opción, se tendrá que pedir los datos necesarios al usuario y/o mostrar los resultados pertinentes. La aplicación se ejecutará hasta que se pulse la opción de “salir”.

```
// Creamos la clase Videojuego. Será exactamente igual que la clase Videojuego
// de la parte de servidor

// Lo declaramos como @Component, de modo se da de alta dentro del contexto Spring
//y deja de ser anónimo. Su identificador será "videojuego"
@Component
public class Videojuego{

  private int id;
  private String nombre;
  private String compañia;
  private int nota;

  //CREAMOS MÉTODOS CONSTRUCTORES
  //CREAMOS GETTER Y SETTER
  //CREAMOS MÉTODO TO STRING

  public Videojuego() {
    super();
  }

  public Videojuego(int id, String nombre, String compañia, int nota) {
    this.id = id;
    this.nombre = nombre;
    this.compañia = compañia;
    this.nota = nota;
  }
```

```
  public int getId() {
    return id;
  }
  public void setId(int id) {
    this.id = id;
  }

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

  public String getCompañia() {
    return compañia;
  }
  public void setCompañia(String compañia) {
    this.compañia = compañia;
  }

  public int getNota() {
    return nota;
  }
  public void setNota(int nota) {
    this.nota = nota;
  }

  @Override
  public String toString() {
    return "Videojuego [id=" + id + ", nombre=" + nombre + ", compañia="
    + compañia + ", nota=" + nota + "]";
  }
}
```

```
// Usamos la anotación @Service, ya que sirve para acceder al servicio rest.
// Con esta anotación se da alta como objeto dentro del contexto Spring
@Service
public class ServicioProxyVideojuego {

  // COMPROBAR SI ESTÁ BIEN LA URL!!
  // Declaramos la URL base del servicio REST de Videojuegos
  public static final String URL = "http://localhost:8086/videojuegos/";

  // Inyectamos el objeto que nos permite salir fuera de la JVM y hacer peticiones http al sercicio rest
  @Autowired
  RestTemplate restTemplate;

  // Creamos el método que nos permitirá obtener/buscar un videojuego por su ID del servicio REST
  // Este método trabaja también con objetos ResponseEntity, gracias a Spring
  // @param se le pasa un número entero como ID buscado
  // Genera un objeto ResponseEntity<Videojuego> que almacena el contenido
  // al llamar al método getEntity, que recupera el ResponseEntity del controlador del servicio rest

  public Videojuego obtener (int id) {
    try {
      ResponseEntity<Videojuego> re = restTemplate.getForEntity(URL + id, Videojuego.class);
      // Asignamos a la variable "hs"el contenido del StatusCode que se manda desde el servidor (en el ResponseEntity guardado en "re")
      HttpStatus hs = re.getStatusCode();
      if (hs == HttpStatus.OK) {  // Si el status del HTTP recibido desde el servicio es un OK, se procede a mostrar el Body con el conteni
        return re.getBody(); // (devolverá el videojuego buscado)
      } else {
        System.out.println("Respuesta no contemplada por el servicio");
        return null;
      }
      // Con la excepción HttpClientErrorException podremos capturar el código del Status enviado arrojado por el servidor
      // con el método .getStatusCode()
    } catch (HttpClientErrorException e){ // Capturamos las excepeciones en caso de que no encuentre el videojuego buscado y saca el status
      System.out.println("La persona con el id " +id + " no se ha encontrdo o no existe");
      System.out.println("Código de respuesta " + e.getStatusCode());
      return null;
    }
  }

  public Videojuego alta (Videojuego vdjg) {
    try {
      // Creará un objeto responseEntity con el resultado de "Postear" en el servicio el videojuego
```

```
      // se usa el método .postForEntity() para dar de alta el videojuego. A este método se le pasan
      // como parámetros: la URL, el objeto videojuego (en el Body) y el objeto que nos devuelve el servicio
      ResponseEntity<Videojuego> re = restTemplate.postForEntity(URL, vdjg, Videojuego.class);
      System.out.println("Dar de alta. El código de respuesta es --> "+ re.getStatusCode());
      return re.getBody();
    }
    catch (HttpClientErrorException e){
      System.out.println("El videojuego no se ha podido dar de alta");
      System.out.println("Código de respuesta: " + e.getStatusCode());
      return null;
    }
  }

  public boolean modificar (Videojuego vdjg) {
    try {
      // Para modificar un videojuego usaremos el método PUT del objeto restTemplate
      // Le pasamos como parámetros la URL, el ID del videojuego y el cast a la clase Videojuego
      // El método PUT no devuelve nada, pero si no ha dado error asumiremos que se ha dado de alta
      restTemplate.put(URL + vdjg.getId(),vdjg, Videojuego.class);
      return true;
      // capturamos la excepción por si da error
    } catch (HttpClientErrorException e) {
      // Avisamos de que no se ha modificado el videojuego
      System.out.println("No se ha podido modificar el videojuego con id " + vdjg.getId());
      // Sacamos el código de status arrojado por el servidor
      System.out.println("Código de respuesta: " +e.getStatusCode());
      return false;
    }

  }


  // El método borrar para eliminar uno de los videojuegos por su id llamará
  // al método delete(), al que se le pasa el id. No devuelve nada
  // Sólo devolverá un mensaje al saltar un error o excepción, porque la capturamos

  public boolean borrar (int id) {
    try {
      restTemplate.delete(URL + id);
      return true;
    }
    catch (HttpClientErrorException e){
      System.out.println("No se ha podido borrar el videojuego deseado, con id: " + id);
      System.out.println("Código de respuesta enviado por el server: " + e.getStatusCode());
      return false;
    }
  }

  public List<Videojuego> listar () {
    try {
      ResponseEntity<Videojuego[]> response = restTemplate.getForEntity(URL, Videojuego[].class);
      //Asignamos (casteamos)a la variable arrayVideojuegos el conjunto de elementos que contiene el Body
      // de la respuesta enviada desde el server
      Videojuego [] arrayVideojuegos = response.getBody();
      // Convertimos el array en un ArrayList:
      return Arrays.asList(arrayVideojuegos);
    }
    catch (HttpClientErrorException e) {
      System.out.println("No se ha podido listar los videojuegos");
      System.out.println("Código de respuesta enviado por el server: " + e.getStatusCode());
      return null;
    }
  }
}
```

Creamos la clase Application que va a contener el menú.

```
// Es una aplicación Spring, por lo que con la anotación @SpringBootApplication
// se configura como tal, y busca anotaciones de Spring por toda la aplicación
// Hacemos que la clase implemente la interfaz CommandLineRunner para que el main pueda acceder
// a los objetos dinámicos de esta clase
@SpringBootApplication
public class Application implements CommandLineRunner {

  // Inyectamos el objeto necesario para conectar con el servidor
  // Lo creamos con la anotación @Autowired propia de Spring
  @Autowired
  private ServicioProxyVideojuego spv;
```

```
// Inyectamos también el contexto que creamos, que es un objeto
@Autowired
private ApplicationContext context;

// Damos de alta un objeto RestTemplate, que será el que haga las conexiones HTTP fuera de la JVM
// Creamos el objeto con la anotación @Bean, por lo que se da de alta en el contexto Spring.
// El método lo llamamos también restTemplate y recibe un objeto de tipo RestTemplateBuilder
// Este es el objeto que nos permitirá salir fuera de la JVM y hacer peticiones al servicio rest
@Bean
public RestTemplate restTemplate(RestTemplateBuilder builder) {
  return builder.build();
}

// Arrancamos el contexto en el método MAIN.
// Dado que es un método estático, para que pueda acceder a los objetos spv y spm (que nos permiten hacer conexiones(
// hemos hecho que la clase Application implemente "CommandLineRunner" previamente
// Para que pueda llamar al método run en cuanto cargueel contexto
public static void main(String[] args) {
  System.out.println("CLIENTE  --> Cargando el contexto de Spring...");
  SpringApplication.run(Application.class, args);

}

// Aquí ejecutaremos todo como si se tratase del main
@Override
public void run(String...args) throws Exception {
  System.out.println("----------ARRANCANDO EL CLIENTE REST---------");

  // Creamos un objeto Videojuego al que dentro del switch le cambiaremos los atributos para poder darlo de alta sin ID
  Videojuego vid1 = new Videojuego();

  // Creamos un objeto Scanner para pedir por pantalla al cliente la opcion del menú deseada
  Scanner sc = new Scanner(System.in);

  // Creamos un único Scanner para usar a lo largo del menú
  Scanner scanner = new Scanner(System.in);

  Boolean seguir = true;

  String juego;

  do {
    System.out.println("ELIJA UNA OPCIÓN:\n\n"
        + "  1. Dar de alta un videojuego \n"
        + "  2. Dar de baja un videojuego por ID \n"
        + "  3. Modificar un videojuego por ID\n"
        + "  4. Obtener un videojuego por ID\n"
        + "  5. Listar todos los videojuegos \n"
        + "  6. Salir");
  int opcion = Integer.parseInt(scanner.nextLine());
  switch (opcion) {

  case 1:
    System.out.println("----DAR DE ALTA DE UN VIDEOJUEGO----");

    System.out.println("Escriba el nombre del videojuego:");
    String nombreVideojuego = scanner.nextLine();
    vid1.setNombre(nombreVideojuego);

    System.out.println("Escriba la compañía del videojuego: ");
    String nombreCompañia = scanner.nextLine();
    vid1.setCompañia(nombreCompañia);

    System.out.println("Introduzca la nota :");
    juego=scanner.nextLine();

    vid1.setNota(Integer.parseInt(juego));

    // Damos de alta el objeto videojuego llamando al método alta
    Videojuego vdjg1 = spv.alta(vid1);
    System.out.println("Se ha dado de alta el siguiente videojuego " + vdjg1);
    break;

  case 2:
    System.out.println("----DAR DE BAJA UN VIDEOJUEGO POR ID----");
    //Pedimos por pantalla el ID del videojuego que se quiere borrar:
    System.out.println("Escriba el ID del videojuego que desea dar de baja: ");

    juego=scanner.nextLine();
    boolean borrado = spv.borrar(Integer.parseInt(juego));
```

```
      // Confirmamos por pantalla que se ha borrado: deberá salir true
      System.out.println("El videojuego con id " + juego + "ha sido borrado? " + borrado);
      break;

    case 3:
      System.out.println("----MODIFICAR UN VIDEOJUEGO POR ID----");

      System.out.println("Introduzca los datos nuevos que desea modificar:");
      //Pedimos por pantalla los datos nuevos del videojuego que se quiere modificar:
      System.out.println("Escriba el id del videojuego:");

      juego=scanner.nextLine();
      vid1.setId(Integer.parseInt(juego));

      System.out.println("Escriba el nombre del videojuego:");
      vid1.setNombre(scanner.nextLine());

      System.out.println("Escriba la compañía del videojuego: ");
      vid1.setCompañia(scanner.nextLine());


      System.out.println("Introduzca la nota :");
      juego=scanner.nextLine();
      vid1.setNota(Integer.parseInt(juego));

      // Modificamos el objeto videojuego llamando al método .modificar() pasándole como argumento
      // el objeto videojuego con los datos que nos ha introducido el cliente por pantalla:
      boolean modificada = spv.modificar(vid1);
      System.out.println("Se ha modificado la persona deseada?" + modificada);
      break;

    case 4:
      System.out.println("----OBTENER UN VIDEOJUEGO POR ID----");
      //Pedimos por pantalla el ID del videojuego que se quiere obtener o consultar:
      System.out.println("Escriba el ID del videojuego que desea obtener: ");

      juego=scanner.nextLine();
      vid1 = spv.obtener(Integer.parseInt(juego));

      System.out.println("El videojuego con el id " + juego + "es la siguiente: " + vid1);
      break;

    case 5:
      System.out.println("----LISTAR VIDEOJUEGOS----");
      // Guardamos en una lista la lista de bibliotecas
      List<Videojuego> listaVideojuegos = spv.listar();
      // Imprimimos la lista de videojuegos recorrieándola con un bucle for/each:
      listaVideojuegos.forEach((v) -> System.out.println(v));

      break;


    case 6:
      System.out.println("Cerrando la aplicación....");
      seguir= false;

      break;

    default :

      System.out.println("Introduzca un número del 1 al 6:");
    }


    }while (seguir);


    sc.close();
    scanner.close();
    pararAplicacion();
  }

  public void pararAplicacion() {
    // Cerramos el servidor web con el el metodo exit de la clase SpringApplication
    // Le pasamos el contexto de Spring y un objeto que implemente la interfaz ExitCodeGenerator.
    // Usaremos la funcion lambda "() -> 0" para simplificar
    SpringApplication.exit(context, () -> 0);

  }
}
```