

Introducción

En este documento, se creará una aplicación web donde se conectará a mi api rest realizado, donde se utilizaron las herramientas de MySQL, Apache (como servidor) y mi aplicación de desarrollador con el eclipse.

En dicho documento se revisará como se creó el servicio api REST desde cero y los comentarios de lo que está pasando se encontraran en el programa y dicho api rest será consumido por una aplicación web.

Primero lo que se hizo fue con mi Api Rest se utilizó el postman para comprobar la información de que realiza las acciones de GET/POST/DELETE/PUT para obtener, agregar, borrar, y actualizar, comprobando dicha funcionalidad y revisando en nuestra base de datos.

Nuestra arquitectura se muestra la aplicación web y mi api rest consumiendo la información.

WEB APP CONECTADO A LA REST API

Realacion de los jugadores					
Agregar jugador					
Nombre	Apellido Paterno	Correo	Edad	Profesional	Acción
Victor	Cruz	victor@mexico.com	29	true	Actualizar Borrar
Martin	Lectones	martin@mexico.com	23	false	Actualizar Borrar
Luis	Chavez	luis@mexico.com	24	true	Actualizar Borrar
Ana	Karens	ana@mexico.com	23	true	Actualizar Borrar
Nayeli	Trejo	Nayeli@mexico.com	23	false	Actualizar Borrar
Ernesto	Cruz	hernandez@gmail.com	28	true	Actualizar Borrar
Sha	Ramos	ramos@gmail.com	29	false	Actualizar Borrar



REST API
REST CONTROLLER

```
[{"id":1,"nombredeportista":"Victor","apellidoPaterno":"Cruz","correo":"victor@mexico.com","edad":29,"profesional":true}, {"id":2,"nombredeportista":"Martin","apellidoPaterno":"Lectones","correo":"martin@mexico.com","edad":23,"profesional":false}, {"id":3,"nombredeportista":"Luis","apellidoPaterno":"Chavez","correo":"luis@mexico.com","edad":24,"profesional":true}, {"id":4,"nombredeportista":"Ana","apellidoPaterno":"Karens","correo":"ana@mexico.com","edad":23,"profesional":true}, {"id":5,"nombredeportista":"Nayeli","apellidoPaterno":"Trejo","correo":"Nayeli@mexico.com","edad":23,"profesional":false}, {"id":6,"nombredeportista":"Ernesto","apellidoPaterno":"Cruz","correo":"hernandez@gmail.com","edad":28,"profesional":true}, {"id":7,"nombredeportista":"Sha","apellidoPaterno":"Ramos","correo":"ramos@gmail.com","edad":29,"profesional":false}]
```



Para poder entender más claro esto tendremos dos aplicaciones uno llamado Eb app Rest Cliente donde la principal acción es consumir la información de mi otra aplicación llamada API REST que viene siendo como el backend, ya que uno tiene que ver la vista y el otro lo que está por debajo consumiendo los datos.



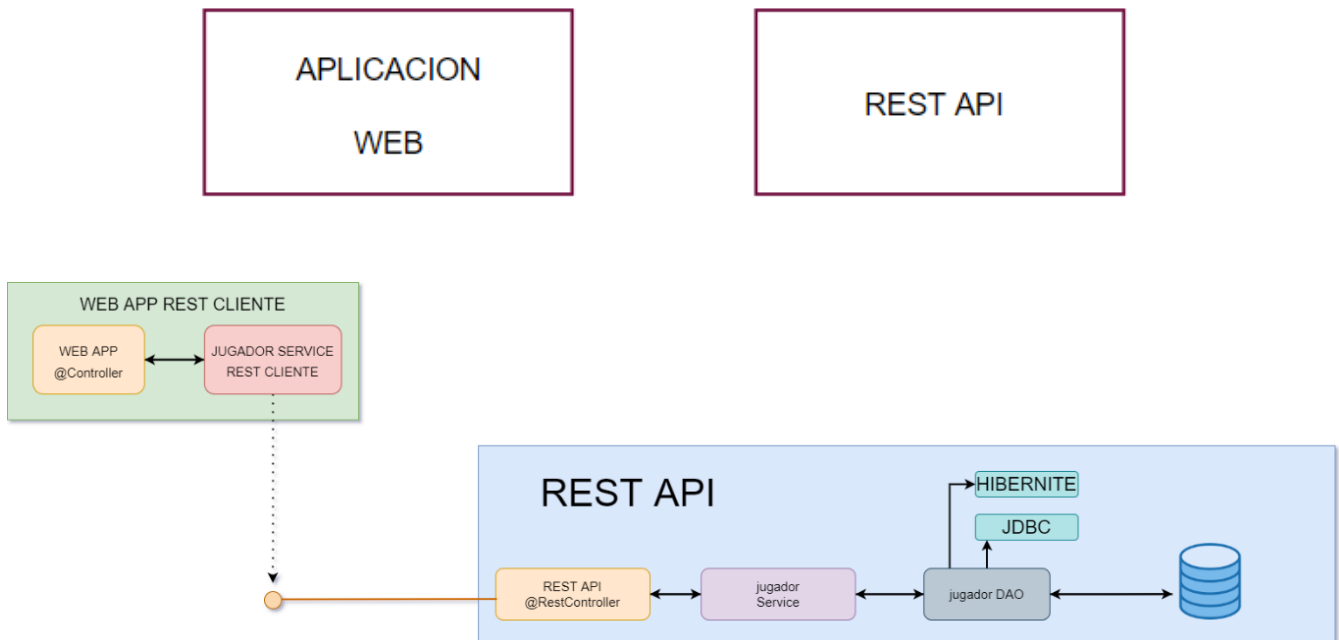
Descripción general de las aplicaciones

1. Arquitectura de aplicaciones.
2. Revisión de la estructura de los proyectos.
3. Archivo de POM de Maven.
4. Archivo de propiedades de configuración de la aplicación.
5. Configuración de Spring Bean para Rest Template.
6. GET: Obtención de una lista de jugadores.
7. GET: Obtener un solo jugador por el id.
8. POST: agrega un nuevo jugador a mi aplicación.
9. PUT: Añadir un nuevo jugador a la lista.
10. DELETE: eliminar un jugador de la lista con el id.
11. Aplicación web corriendo en servidor

1. Arquitectura de aplicaciones

En el primer nivel el bajo, la aplicación tendrá la arquitectura denominada de dos proyectos uno APLICACIÓN WEB Y otro REST API, los cuales la aplicación web se conectará a mi servicio REST API.

DEPORTISTA /JUGADOR



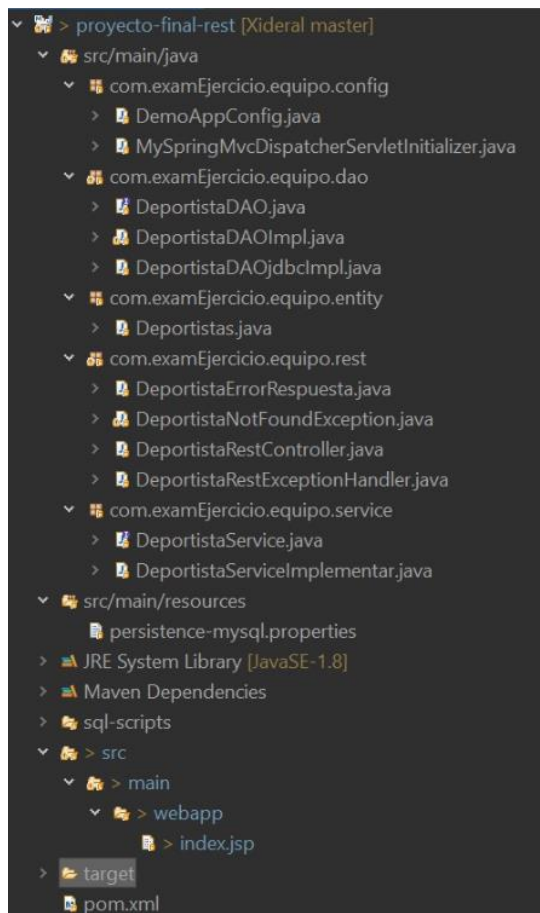
El cliente REST de mi WebApp tendrá una nueva implementación del DeportistasService. Anteriormente se realizó un proyecto donde el cliente REST y mi aplicación web usaba una implementación de DAO con hibernate, Sin embargo el código de DAO de hibernate ya no es necesario en el cliente REST de mi aplicación WEB app. La interacción de la base de datos ahora es manejada por API REST conocido la parte del back-end y la aplicación Web solo realiza la conexión teniendo una vista.

En esta ocasión se realiza la implementación de dos maneras una por hibernate y otra por JDBC, en donde lo único que haremos será utilizar mi aplicación Web recuperando datos a través de mi API REST

2. Revisión de la estructura de los proyectos

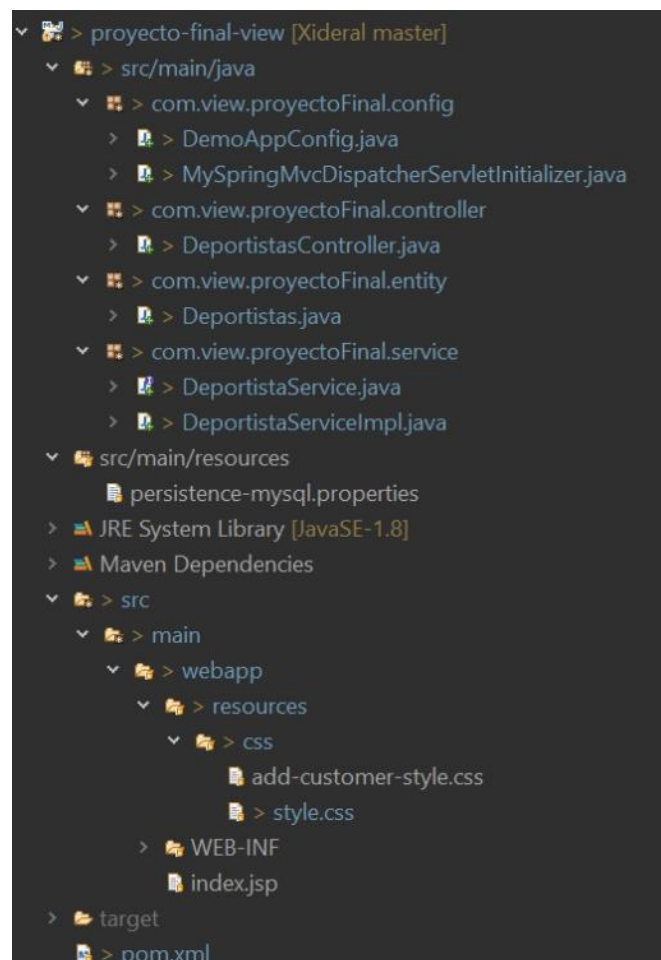
Se mostrará a continuación las imágenes de los archivos utilizados en ambos proyectos en donde tendremos la aplicación web y el proyecto de la API REST, donde tendrán sus archivos y sus propias configuraciones para que se realizara correctamente el proyecto. De un lado tenemos los archivos de API REST y del otro lado los Archivos de la APLICACIÓN WEB.

REST API



VS

APLICACION WEB



3. Archivo de POM de Maven

Para el cliente REST de mi aplicación web, el archivo POM es diferente al archivo de mi API REST en donde se utilizan diferentes dependencias y teniendo algunas similares continuación se mostrara ambos archivos POM y sus diferencias

REST API

```
<properties>
  <springframework.version>5.1.9.RELEASE</springframework.version>
  <hibernate.version>5.4.4.Final</hibernate.version>
  <mysql.connector.version>8.0.17</mysql.connector.version>
  <c3po.version>0.9.5.2</c3po.version>

  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
</properties>

<dependencies>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${springframework.version}</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>${springframework.version}</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>${springframework.version}</version>
  </dependency>

  <!-- Hibernate -->
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>${hibernate.version}</version>
  </dependency>

  <!-- MySQL -->
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>${mysql.connector.version}</version>
  </dependency>

  <!-- C3PO -->
  <dependency>
    <groupId>com.mchange</groupId>
    <artifactId>c3p0</artifactId>
    <version>${c3po.version}</version>
  </dependency>

  <!-- Servlet+JSP+JSTL -->
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
  </dependency>
```



Aplicación Web

```
<properties>
  <springframework.version>5.1.9.RELEASE</springframework.version>
  <hibernate.version>5.4.4.Final</hibernate.version>
  <mysql.connector.version>8.0.17</mysql.connector.version>
  <c3po.version>0.9.5.2</c3po.version>

  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
</properties>

<dependencies>
  <!-- Spring -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${springframework.version}</version>
  </dependency>

  <!-- Add Jackson for JSON converters -->
  <dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.9.5</version>
  </dependency>

  <!-- Servlet+JSP+JSTL -->
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
  </dependency>

  <dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>javax.servlet.jsp-api</artifactId>
    <version>2.3.1</version>
  </dependency>

  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
  </dependency>

  <!-- to compensate for java 9 not including jaxb -->
  <dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.0</version>
  </dependency>
```


Observe que en las dependencias de mi proyecto de aplicación web no hay dependencias de Hibernate o dependencias JDBC de MySQL a comparación del proyecto API REST donde si está manejando estas dependencias ya que el si se va conectar a la base de datos. La única nueva entrada necesaria es para Jackson. Se utilizará para convertir automáticamente los datos JSON a objetos Java.

4. Archivo de propiedades de configuración de la aplicación

El cliente de mi aplicación web lo único que tendrá es la conexión a mi API REST donde se indicara en donde va consumir la información para mostrarla, este archivo se encontrara en las propiedades de mi aplicación web, en el caso de mi aplicación web vendrán las propiedades de la conexión a la base de datos creada para consumir la información implementado Hibernate y JDBC

File: src/main/resources/application.properties

```
# The URL for the CRM REST API
# - update to match your local environment
#
crm.rest.url=http://localhost:8080/spring-crm-rest/equipo/jugador
```

5. Configuración de Spring Bean para-Rest Template

Spring incluye la clase auxiliar RestTemplate para realizar las llamadas del cliente REST. En donde podemos crear una instancia de esta clase como spring vean. Como resultado podemos inyectarlo fácilmente en cualquiera de nuestros componentes. Esta información se encuentra en el proyecto de API REST donde se encontrará con el nombre com.view.proyectoFinal.config.DemoAppConfig

```
@Configuration
@EnableWebMvc
@ComponentScan("com.view.proyectoFinal")
@PropertySource({ "classpath:persistence-mysql.properties" })
public class DemoAppConfig implements WebMvcConfigurer {

    @Bean
    public ViewResolver viewResolver() {

        InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();

        viewResolver.setPrefix("/WEB-INF/view/");
        viewResolver.setSuffix(".jsp");

        return viewResolver;
    }
}
```

Este código también carga el archivo de la aplicación de propiedades visto anteriormente, donde se debe declarar con la anotación `@PropertySource` dentro declarando el nombre de dicho archivo.

Después, podemos inyectar este componente a nuestro `DeportistaServiceRestImplementar` llamado `File: DeportistaServiceImpl.java`

```
@Service
public class DeportistaServiceImpl implements DeportistaService {

    private RestTemplate restTemplate;
    private String crmRestUrl;
    private Logger logger = Logger.getLogger(getClass().getName());

    @Autowired
    public DeportistaServiceImpl(RestTemplate theRestTemplate, @Value("${crm.rest.url}") String theUrl) {
        restTemplate = theRestTemplate;
        crmRestUrl = theUrl;

        logger.info("CHAVEZ CRUZ VICTOR ----- Loaded property: crm.rest.url=" + crmRestUrl);
    }
}
```

Además, observe que se inyectó la propiedad `rest.url` mediante la anotación `@Value`, Este es un ejemplo de inyección por medio de constructores recordando que también puede ser por medio de clases y variables. Como otra opción se pudo haber inyectado usando por las otras dos maneras.

6. GET: Obtención de una lista de jugadores.

La API REST expone el siguiente extremo para obtener una lista de clientes, para entender mas claro esto es necesario saber qué es lo que hace GET y mediante el mapeo para que muestre la lista.

GET /equipo/jugador



Este código es para realizar la llamada del cliente rest

File: DeportistaServiceImpl.java

```
@Override
public List<Deportistas> getDeportista() {
    logger.info("CHAVEZ CRUZ VICTOR ----- in getDeportista(): Calling REST API " + crmRestUrl);

    ResponseEntity<List<Deportistas>> responseEntity = restTemplate.exchange(crmRestUrl, HttpMethod.GET, null,
        new ParameterizedTypeReference<List<Deportistas>>() {
        });
    List<Deportistas> deportistas = responseEntity.getBody();

    logger.info("CHAVEZ CRUZ VICTOR ----- in getDeportista(): deportista" + deportistas);

    return deportistas;
}
```

En este Código realizamos una llamada para conseguir los jugadores. El código principal ocurre con la llamada del método restTemplate.exchange(...)

Dicho método tiene los siguientes parámetros

- URL
- Método: el método HTTP para GET
- RequestEntity: que es el encabezado o cuerpo de las solicitudes adicionales o null en nuestro caso
- responseType: el tipo del valor devuelto
- ParametizedTypeReference que es el que se utiliza para pasar la información de tipo genérico

La API REST devuelve un objeto JSON. En segundo plano, Spring utiliza JACKSON para convertir los datos JSON en una lista de objetos. Deportista con List<Deportistas>.

7. GET: Obtener un solo jugador por el id.

La API REST expone el siguiente extremo donde indica el único cliente identificando por ID

File: CustomerServiceRestClientImpl.java

```
@Override
public Deportistas getDeportista(int theId) {

    logger.info("CHAVEZ CRUZ VICTOR ----- in getDeportista(): Calling REST API " + crmRestUrl);

    // make REST call
    Deportistas jugador = restTemplate.getForObject(crmRestUrl + "/" + theId, Deportistas.class);

    logger.info("CHAVEZ CRUZ VICTOR ----- in saveDeportista(): theDeportista=" + jugador);

    return jugador;
}
```

El trabajo principal ocurre en el método restTemplate.getForObject(...). El método realiza una solicitud HTTP GET a la dirección URL.

El método tiene los siguientes parámetros

- URL: la dirección URL
- responseType: el tipo del valor devuelto

Como se explico anteriormente, la API REST devuelve los datos como JSON. Jackson se encarga de convertir el JSON en un objeto de tipo deportista.

8. POST: agrega un nuevo cliente

La API REST expone el siguiente extremo para agregar un nuevo cliente

POST /equipo/jugador



El cuerpo de la solicitud contendrá los datos del cliente. Este es el Código para realizar la llamada del cliente REST.

File: CustomerServiceRestClientImpl.java

```
@Override
public Deportistas getDeportista(int theId) {

    logger.info("CHAVEZ CRUZ VICTOR ----- in getDeportista(): Calling REST API " + crmRestUrl);

    // make REST call
    Deportistas jugador = restTemplate.getForObject(crmRestUrl + "/" + theId, Deportistas.class);

    logger.info("CHAVEZ CRUZ VICTOR ----- in saveDeportista(): theDeportista=" + jugador);

    return jugador;
}
```

Cuando agregamos un cliente, el ID del cliente es 0 por lo que ejecutaremos el siguiente fragmento de Código:

```
int jugadorId = jugador.getId();
// realiza la llamada al REST
if (jugadorId == 0) {
    // AGREGA UN DEPORTISTA
    restTemplate.postForEntity(crmRestUrl, jugador, String.class);
    logger.info("CHAVEZ CRUZ VICTOR ----- en saveDeportista(): Calling REST API " + crmRestUrl);
}
```

El Código principal ocurre en el método: restTemplate.postForEntity(...) . El método envía un HTTP POST a la dirección URL

El método tiene los siguientes parámetros

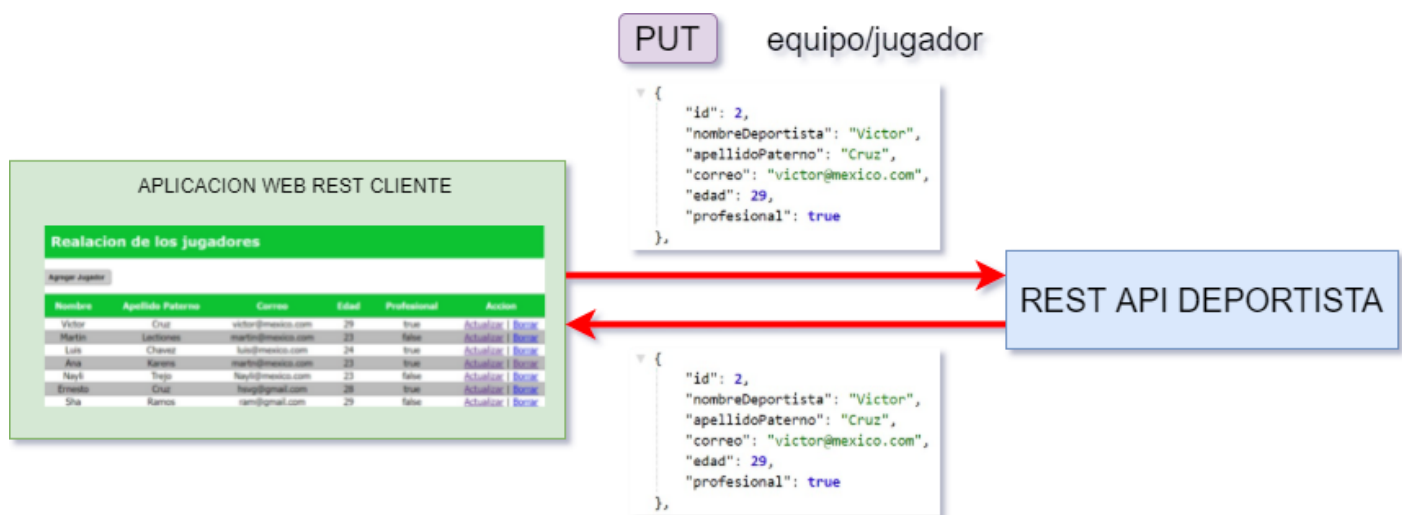
- URL - la URL
- REQUEST - el objeto que se va a POST
- RESPONSETYPE - el tipo de respuesta

Como de costumbre, Jackson se encarga de convertir los objetos Java a JSON y viceversa.

9. PUT: Añadir un nuevo jugador a la lista.

La API de REST expone el siguiente extremo para actualizar un DEPORTISTA existente.

PUT /equipo/jugador



Este es el código para realizar la llamada del cliente REST.

File: DeportistaServiceImpl.java

```
@Override
public void saveDeportista(Deportistas jugador) {
    int employeeId = jugador.getId();
    // realiza la llamada al REST
    if (employeeId == 0) {
        // AGREGA UN DEPORTISTA
        restTemplate.postForEntity(crmRestUrl, jugador, String.class);
        logger.info("CHAVEZ CRUZ VICTOR ----- en saveDeportista(): Calling REST API " + crmRestUrl);
    } else {
        // ACTUALIZA LA INFORMACION DEL DEPORTISTA
        logger.info("CHAVEZ CRUZ VICTOR ----- en UPDATEDeportista(): ha sido de manera exitosa");
        restTemplate.put(crmRestUrl, jugador);
    }
}
```

Cuando actualizamos un cliente, el ID de cliente no es igual a 0, por lo que ejecutaremos el siguiente código.

```
} else {
    // ACTUALIZA LA INFORMACION DEL DEPORTISTA
    logger.info("CHAVEZ CRUZ VICTOR ----- en UPDATEDeportista(): ha sido de manera exitosa");
    restTemplate.put(crmRestUrl, jugador);
}
```

El código principal ocurre en el método: `restTemplate.put(...)`. El método envía un HTTP PUT a la dirección URL.

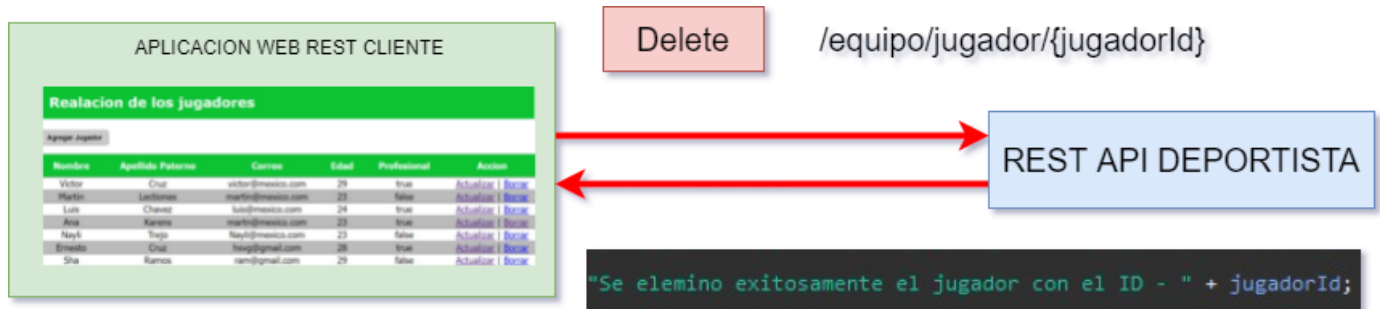
El método tiene los siguientes parámetros

- url - la URL
- request - el objeto a PUT

10. DELETE : ELIMINAR UN JUGADOR

La API de REST expone el siguiente extremo para eliminar un jugador.

DELETE /equipo/jugador/{jugadorId}



Este es el código para realizar la llamada del cliente

REST.File: DeportistaServiceImpl.java

```
@Override
public void deleteDeportista(int theId) {
    logger.info("CHAVEZ CRUZ VICTOR ----- en deleteDeportista(): Calling REST API " + crmRestUrl);
    // llama a la aplicacion REST
    restTemplate.delete(crmRestUrl + "/" + theId);
    logger.info("CHAVEZ CRUZ VICTOR ----- in deleteDeportistas(): delete Deportista theId=" + theId);
}
```

El trabajo principal ocurre en el método `restTemplate.delete(...)`. Este método envía un HTTP DELETE a la dirección URL.

El método tiene el siguiente parámetro

1. URL - la URL

Ahora, cuando ejecuta el cliente REST de la aplicación web, realiza todas las solicitudes a la API REST back-end.

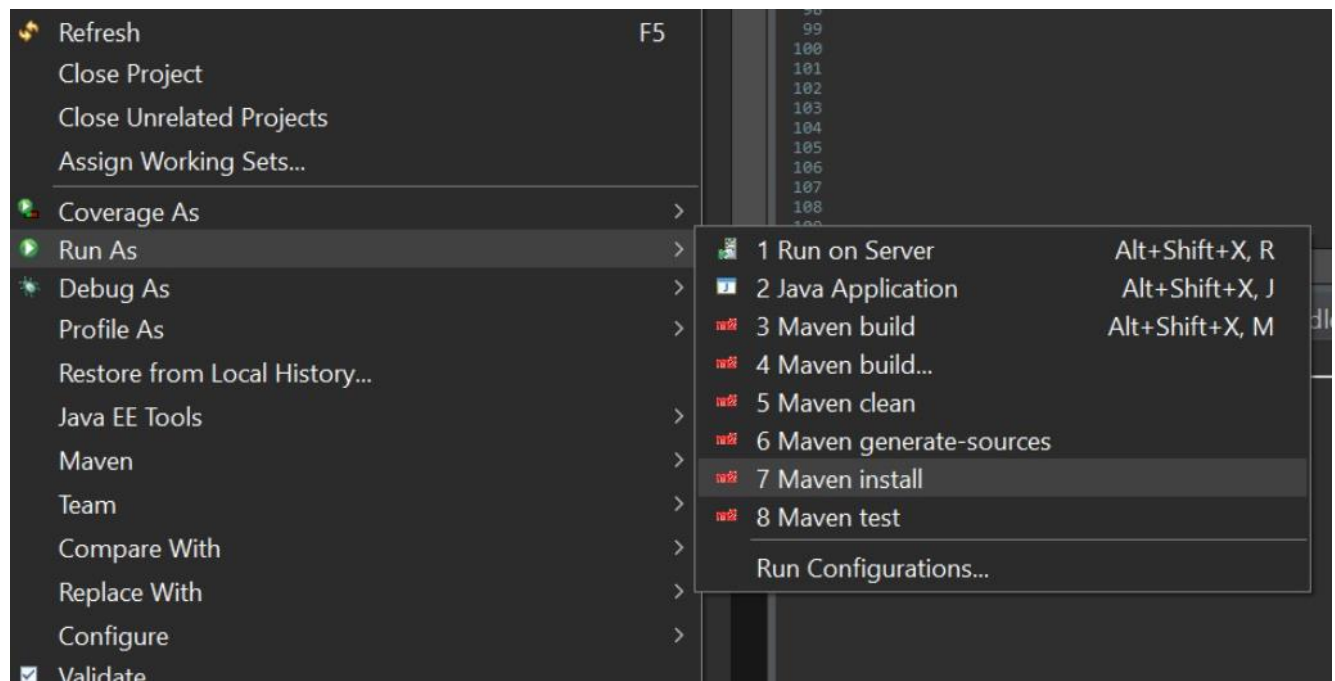
11. Aplicación web corriendo en servidor

Para correr la aplicación sin la necesidad del eclipse se requiero realizar los siguientes pasos:

1. Eliminar los archivos de mi aplicación que están dentro de mi carpeta target

Usuarios > User > Descargas > Xideral > proyecto-final-view > target					Buscar en ta
<input type="checkbox"/>	Nombre	Fecha de modificación	Tipo	Tamaño	
	classes	27/05/2022 06:45 a. m.	Carpeta de archivos		
	generated-sources	27/05/2022 06:45 a. m.	Carpeta de archivos		
	m2e-wtp	27/05/2022 06:45 a. m.	Carpeta de archivos		
	maven-archiver	27/05/2022 06:45 a. m.	Carpeta de archivos		
	maven-status	27/05/2022 06:45 a. m.	Carpeta de archivos		
	spring-web-customer-tracker-all-java-config	27/05/2022 06:45 a. m.	Carpeta de archivos		
	spring-web-customer-tracker-all-java-config....	27/05/2022 06:45 a. m.	Archivo WAR	7,336 KB	

2. Ir a ambas aplicaciones en la opción 7 Maven Install donde generaremos de nuevo los archivos y obteniendo un archivo war

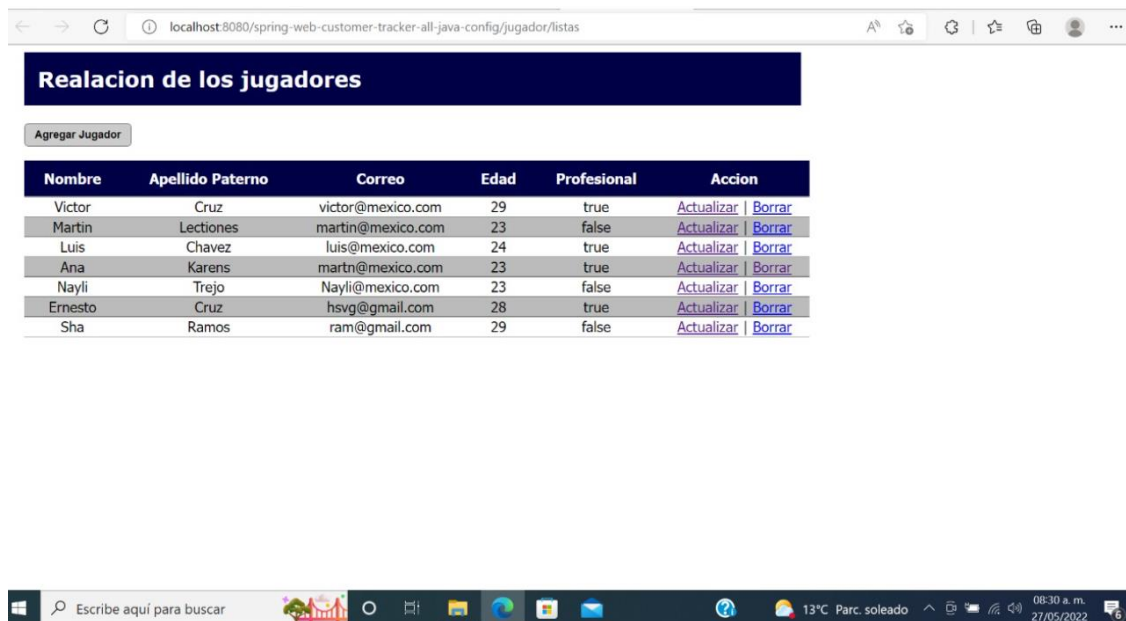


3. Después no vamos a los archivos de configuración de mi programa Tomcat donde viene toda la información de mi Tomcat, copiamos los archivos anteriores

generados de mi aplicación web y rest, para que al momento correr el tomcat se generen dos carpetas con el mismo nombre de mis archivos war.

classes	27/05/2022 06:45 a. m.	Carpeta de archivos	
generated-sources	27/05/2022 06:45 a. m.	Carpeta de archivos	
m2e-wtp	27/05/2022 06:45 a. m.	Carpeta de archivos	
maven-archiver	27/05/2022 06:45 a. m.	Carpeta de archivos	
maven-status	27/05/2022 06:45 a. m.	Carpeta de archivos	
spring-web-customer-tracker-all-java-config	27/05/2022 06:45 a. m.	Carpeta de archivos	
spring-web-customer-tracker-all-java-config....	27/05/2022 06:45 a. m.	Archivo WAR	7,336 KB

- Por ultimo se corre el servidor y se revisa ambas aplicaciones que están corriendo sin la necesidad de mi eclipse.



The screenshot shows a web browser at the URL `localhost:8080/spring-web-customer-tracker-all-java-config/jugador/listas`. The page has a dark blue header with the title "Realacion de los jugadores". Below the header is a button labeled "Agregar Jugador". The main content is a table with the following columns: Nombre, Apellido Paterno, Correo, Edad, Profesional, and Accion. The table contains seven rows of player data. Each row has two links in the "Accion" column: "Actualizar" and "Borrar".

Nombre	Apellido Paterno	Correo	Edad	Profesional	Accion
Victor	Cruz	victor@mexico.com	29	true	Actualizar Borrar
Martin	Lecciones	martin@mexico.com	23	false	Actualizar Borrar
Luis	Chavez	luis@mexico.com	24	true	Actualizar Borrar
Ana	Karens	marth@mexico.com	23	true	Actualizar Borrar
Nayli	Trejo	Nayli@mexico.com	23	false	Actualizar Borrar
Ernesto	Cruz	hsvg@gmail.com	28	true	Actualizar Borrar
Sha	Ramos	ram@gmail.com	29	false	Actualizar Borrar

```
localhost:8080/spring-crm-rest/equipo/jugador
[
  {
    "id": 2,
    "nombreDeportista": "Victor",
    "apellidoPaterno": "Cruz",
    "correo": "victor@mexico.com",
    "edad": 29,
    "profesional": true
  },
  {
    "id": 5,
    "nombreDeportista": "Martin",
    "apellidoPaterno": "Leciones",
    "correo": "martin@mexico.com",
    "edad": 23,
    "profesional": false
  },
  {
    "id": 6,
    "nombreDeportista": "Luis",
    "apellidoPaterno": "Chavez",
    "correo": "luis@mexico.com",
    "edad": 24,
    "profesional": true
  },
  {
    "id": 8,
    "nombreDeportista": "Ana",
    "apellidoPaterno": "Karens",
    "correo": "ana@mexico.com",
    "edad": 23,
    "profesional": true
  }
]
```

- Par finalizar se revisara las aplicaciones que se ejecutan en el momento, revisando desde de mi petición HTTP que este lo da por default mi Tomcat
http://localhost:8080/manager/html

localhost:8080/manager/html

Gestor de Aplicaciones Web de Tomcat

Mensaje: OK

Gestor

[Listar Aplicaciones](#) [Ayuda HTML de Gestor](#) [Ayuda de Gestor](#) [Estado de Servidor](#)

Aplicaciones					
Ruta	Versión	Nombre a Mostrar	Ejecutándose	Sesiones	Comandos
/	Ninguno especificado	Welcome to Tomcat	true	0	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos
/docs	Ninguno especificado	Tomcat Documentation	true	0	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos
/manager	Ninguno especificado	Tomcat Manager Application	true	1	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos
/spring-crm-rest	Ninguno especificado		true	0	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos
/spring-web-customer-tracker-all-java-config	Ninguno especificado		true	0	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos

Desplegar

Desplegar directorio o archivo WAR localizado en servidor