Tema 5. Servicios REST

Programación web

Boni García

Curso 2016/2017

Índice

Índice

Introducción

Servicios web

- Un **servicio distribuido** consiste en varios procesos que se ejecutan en diferentes equipos terminales y que se comunican a través de una red de datos (típicamente Internet)
- Los **servicios web** son un tipo de servicios distribuido ofrecido mediante tecnología web (protocolo HTTP)
  - Podemos ver un servicio web como una aplicación web en la que hay un cliente que hace peticiones y un servidor que las atiende
  - Se utiliza el protocolo HTTP para la interacción entre el cliente y el servidor
  - Cuando se hace una petición, no se espera obtener una página web en formato HTML, en vez de eso, se espera obtener datos estructurados (en formato **XML** o **JSON**) para que sea procesada por el cliente

Introducción

Servicios web

- Los clientes de los servicios web puede ser diferente naturaleza:
  - Páginas web con AJAX o SPA (*Single Page Application*)
  - Otro tipo de clientes: Aplicación móviles, TVs, consolas,...
  - Servidores de otras aplicaciones web
- Por ejemplo, la aplicación de Facebook para Android es un cliente de un servicio web proporcionado por Facebook
- Una de las mayores ventajas de los servicios web es la transparencia del lenguaje, tanto el cliente como el servidor pueden estar escritos en cualquier lenguaje de programación (no tienen que utilizar el mismo lenguaje)
- Hay dos tipos principales de servicios web: SOAP y **REST**

Introducción

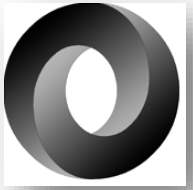
Servicios web

- **REST** (*REpresentational State Transfer*) es un tipo de servicio web que hace uso del protocolo HTTP para realizar operaciones CRUD en recursos remotos
  - Se usan los métodos (verbos) GET, POST, PUT, DELETE de HTTP 1.1 (RFC 2616) para definir las operaciones
  - Hay una extensión a HTTP 1.1 (RFC 5789) que define un nuevo método: PATCH (modificación parcial de un recurso)
  - Se usan los códigos de repuesta HTTP (200 OK, 500 *Internal Server Error*, …) como resultado de las operaciones
- El término se acuñó en el año 2000, en la tesis doctoral sobre la web escrita por Roy Fielding, uno de los autores de la especificación del protocolo HTTP
  - A los servicios web que siguen la arquitectura REST se les suele conocer como *RESTful*
  - Si no se usa la arquitectura REST de forma estricta (por ejemplo, sólo usando GET y POST para todas las operaciones) se dice que el servicio es *REST-like*

Formato JSON

JSON

- **JSON** (*JavaScript Object Notation*), es un formato ligero para almacenar o enviar información estructurada
- No es realmente un estándar como tal, pero está basado en el estándar de JavaScript (ECMAScript)
- Se utiliza para la codificación de la información en la mayoría de los servicios REST(aunque también se puede usar XML)
- JSON se está haciendo cada vez más popular (cada vez se emplea más en lugares donde antes se empleaba XML):
  - Ficheros de configuración, información estructurada, etc…



http://www.json.org/

Formato JSON

JSON

- Los datos JSON pueden ser:

1. Una colección de pares de objetos nombre/valor:
   - Un objeto comienza con el símbolo **{** y termina con **}**
   - El nombre y el valor se separan mediante el símbolo **:**
2. Una lista ordenada de valores:
   - Una lista comienza con el símbolo **[** y termina con **]**
   - Se usa el símbolo **,** para separar los elementos de la lista

- Los valores pueden ser cadenas que van entre comillas dobles (**" "**), números, valores lógicos (**true false**), o **null**

object
{}
{ *members* }
members
pair
pair , *members*
pair
string : *value*
array
[]
[*elements*]
elements
value
value , *elements*
value
string
number
object
array
true
false
null

Formato JSON

JSON

- Ejemplo de información estructurada con JSON:

```
{
    "menu": {
        "id": "file",
        "value": "File",
        "popup": {
            "menuitem": [
                {
                    "value": "New",
                    "onclick": "CreateNewDoc()"
                },
                {
                    "value": "Open",
                    "onclick": "OpenDoc()"
                },
                {
                    "value": "Close",
                    "onclick": "CloseDoc()"
                }
            ]
        }
    }
}
```

Formato JSON

JSON

- Existen multitud de librerías en cualquier lenguaje para procesar JSON
- Las principales librerías de JSON para Java son:
- **Jackson:**
  - http://jackson.codehaus.org/
  - Es la librería por defecto en Spring para JSON
- **Gson:**
  - https://code.google.com/p/google-gson/
  - Es más ligera que Jackson

Índice

Diseño de un servicio REST

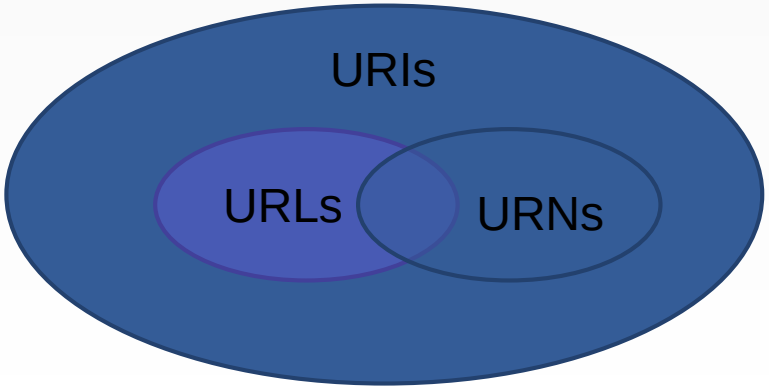
Diseño de un servicio REST

- El esquema habitual que define el funcionamiento de los servicios REST es el siguiente:
  1. La identificación de recursos se realizan mediante URLs
  2. Las operaciones se realizan mediante métodos HTTP
  3. La información se devuelve codificada en JSON
  4. Los códigos de respuesta HTTP notifican el resultado de la operación

Diseño de un servicio REST

Diseño de un servicio REST

URI vs URL vs URN
- URI = *Uniform Resource Identifier*
- URL = *Uniform Resource Locator*
- URN = *Uniform Resource Name*
- Las URIs son cadenas que sirven para **identificar** un recurso
- Las URLs son cadenas que sirven para **localizar** un recurso
- Las URNs son cadenas que sirven para **nombrar** un recurso
- Todas las URLs son URIs pero no siempre ocurre a la inversa



- Ejemplos URLs:
  - http://www.ietf.org/rfc/rfc2396.txt
  - mailto:john.doe@example.com
- Ejemplos URNs:
  - urn:ietf:rfc:2648
  - urn:issn:0167-6423

http://www.w3.org/TR/uri-clarification/

Diseño de un servicio REST

Diseño de un servicio REST

1. La identificación de recursos se realizan mediante URLs

- Parte de la URI es fija y otra parte apunta al recurso concreto
- Ejemplos:
  - http://server.tld/users/bob
  - http://server.tld/users/bob/anuncio/comparto-piso
  - http://server.tld/users/bob/anuncio/44

Diseño de un servicio REST

Diseño de un servicio REST

2. Las operaciones se realizan mediante métodos HTTP

- **GET**: Devuelve el recurso, generalmente codificado en JSON. No envían información en el cuerpo de la petición
- **DELETE**: Borra el recurso. No envían información en el cuerpo de la petición
- **POST** y **PUT**: Añade/modifica un recurso. Envía el recurso en el cuerpo de la petición
  - La diferencia entre una y otra está que PUT debería ser una operación idempotente (aunque se llame varias veces tiene el mismo efecto) mientras que POST no lo será
  - **PATCH**: Modificación parcial de un recurso

Diseño de un servicio REST

Diseño de un servicio REST

3. La información se devuelve codificada el cuerpo de la respuesta

- Petición:
  - URL: http://server/bob/bookmarks/6
  - Método: `GET`
- Respuesta:
  - `mime-type`: `application/json`
  - Cuerpo petición (*body*):

```
{
    id: 6,
    uri: "http://bookmark.com/2/bob",
    description: "A description"
}
```

Diseño de un servicio REST

Diseño de un servicio REST

4. Los códigos de respuesta HTTP notifican el resultado de la operación

- `100-199`: No están definidos
- `200-299`: La petición fue procesada correctamente
- `300-399`: El cliente debe hacer acciones adicionales para completar la petición, por ejemplo, una redirección a otra página
- `400-499`: Se usa en casos en los que el cliente ha realizado la petición incorrectamente (ejemplo típico: 404 No existe)
- `500-599`: Se usa cuando se produce un error procesando la petición

Servicios REST

Implementación de un servicio REST

- Para implementar los servicios REST con Java se puede usar:
- **JAX-RS** (*Java API for RESTful Web Services*)
  - Estándar Java EE
  - https://jersey.java.net/
- **Spring MVC**
  - Parte del Framework Spring
  - Mismo sistema usado para generar páginas web
  - Diferencias con Spring MVC para generar HTML
    - Se usa la anotación `@RestController` (en lugar de `@Controller`)
    - Los métodos devuelven el valor que tiene que enviarse al cliente, en vez de devolver el objeto **`ModelAndView`**

Servicios REST

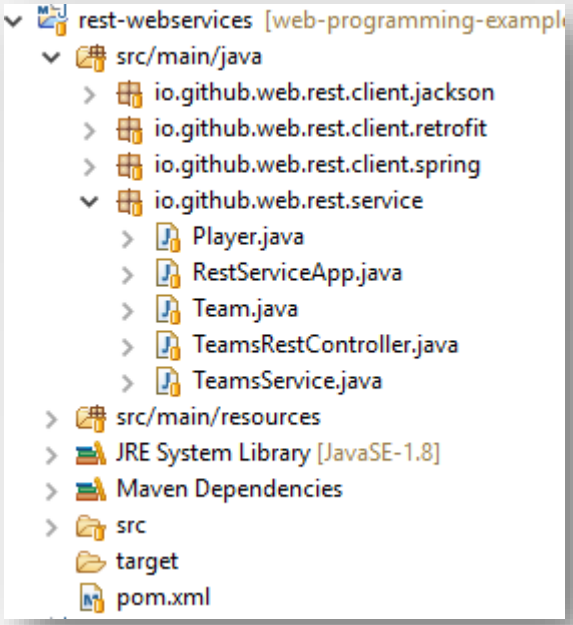
Implementación de un servicio REST

- Ejemplo de servicio REST con Spring MVC
  - Gestiona una lista de equipos (clase `Team`)
  - Cada equipo tiene un nombre y una lista de jugadores (clase `Player`)
  - Permite obtener todos los equipos (con los jugadores)
  - Permite obtener un equipo concreto por su índice

Servicios REST

Implementación de un servicio REST

- Ejemplo de servicio REST con Spring MVC




```
<parent>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-parent</artifactId>
   <version>1.5.2.RELEASE</version>
</parent>

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

No necesitamos capa de presentación (Thymeleaf), con lo que usamos la dependencia de aplicaciones web de Spring Boot

Servicios REST

Implementación de un servicio REST

- Ejemplo de servicio REST con Spring MVC

```
public class Player {

   private String name;
   private String nickname;

   public Player() {
   }

   public Player(String name, String nickname) {
      this.name = name;
      this.nickname = nickname;
   }

   // Getters, setters

}
```

```
public class Team {

   private List<Player> players;
   private String name;

   public Team() {
   }

   public Team(String name, List<Player> players) {
      this.name = name;
      this.players = players;
   }

   // Getters and setters

}
```

Modelo de datos

Servicios REST

Implementación de un servicio REST

- Ejemplo de servicio REST con Spring MVC

```
@RestController
public class TeamsRestController {

   @Autowired
   private TeamsService teamsService;

   @RequestMapping(value = "/teams", method = RequestMethod.GET)
   public List<Team> getTeams() {
      return teamsService.getTeams();
   }

   @RequestMapping(value = "/team/{index}", method = RequestMethod.GET)
   public Team getTeam(@PathVariable("index") int index) {
      return teamsService.getTeam(index);
   }

   @RequestMapping(value = "/teams", method = RequestMethod.POST)
   public ResponseEntity<Boolean> addTeam(@RequestBody Team team) {
      teamsService.addTeam(team);
      return new ResponseEntity<Boolean>(true, HttpStatus.CREATED);
   }
}
```

Controlador REST

Para acceder al cuerpo de la petición POST se usa la anotación `@RequestBody` en lugar de `@RequestParam`

Servicios REST

Implementación de un servicio REST

- Ejemplo de servicio REST con Spring MVC

```
@Service
public class TeamsService {
   private List<Team> teams;
   public TeamsService() {
      teams = new ArrayList<>();
      Player p1 = new Player("Player 1", "p1");
      Player p2 = new Player("Player 2", "p2");
      Player p3 = new Player("Player 3", "p3");
      Player p4 = new Player("Player 4", "p4");
      List<Player> l1 = new ArrayList<>();
      l1.add(p1);
      l1.add(p2);
      Team t1 = new Team("t1", l1);
      List<Player> l2 = new ArrayList<>();
      l2.add(p3);
      l2.add(p4);
      Team t2 = new Team("t2", l2);
      teams.add(t1);
      teams.add(t2);
   }
   public Team getTeam(int index) {
     return teams.get(index);
   }

   public List<Team> getTeams() {
      return teams;
   }

   public void addTeam(Team team) {
      teams.add(team);
   }

}
```

El servicio que implementamos en este ejemplo maneja una lista en memoria (objeto de tipo `ArrayList`)

Servicios REST

Implementación de un servicio REST

- Ejemplo de servicio REST con Spring MVC

Como siempre, para ejecutar el ejemplo usamos una aplicación Java Spring Boot

```
@SpringBootApplication
public class RestServiceApp {

   public static void main(String[] args) {
      SpringApplication.run(RestServiceApp.class, args);
   }
}
```

Índice

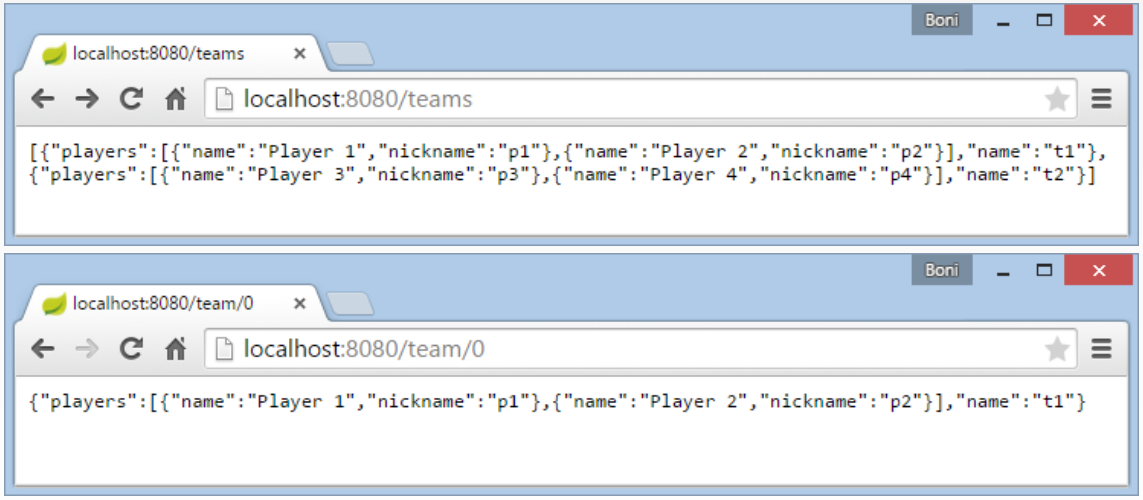
Clientes de servicios REST

- Los servicios REST están diseñados para ser utilizados por aplicaciones
- Estas aplicaciones estarán implementadas en algún lenguaje de programación
- Estudiaremos clientes implementados en **Java** y en **JavaScript**
- Como desarrolladores podemos usar **herramientas interactivas** para hacer peticiones y ver las respuestas

Clientes de servicios REST

Herramientas interactivas

- El navegador web es una herramienta básica que se puede usar para hacer peticiones **GET**



Clientes de servicios REST

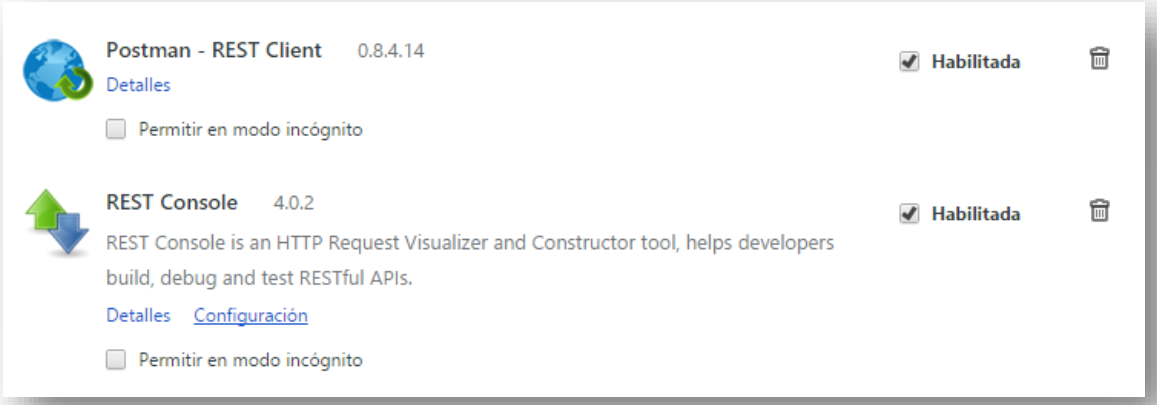
Herramientas interactivas

- El JSON resultante su puede indentar automáticamente con otras herramientas online como http://jsbeautifier.org/

Clientes de servicios REST

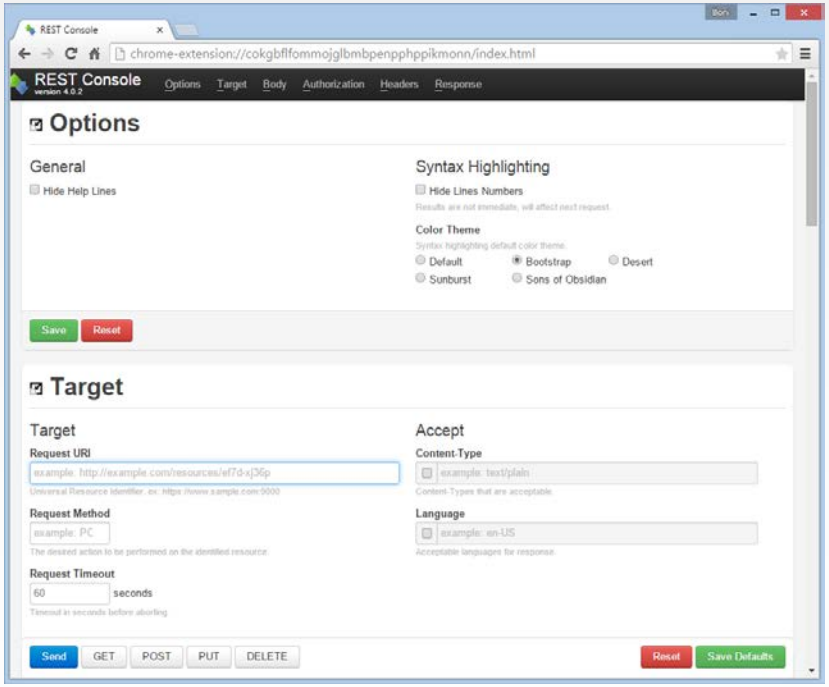
Herramientas interactivas

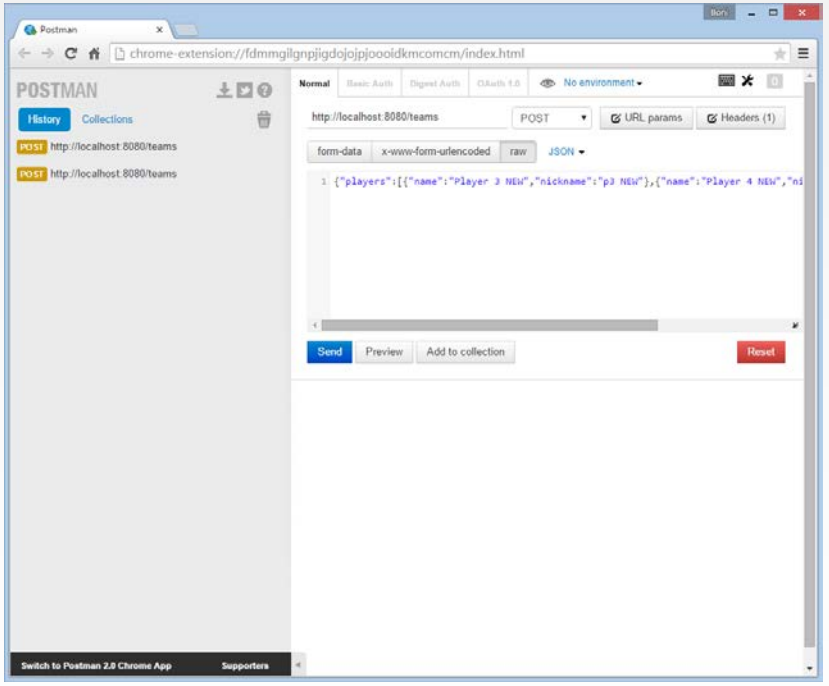
- Existen extensiones de los navegadores que nos permiten realizar cualquier tipo de petición REST
- Por ejemplo, hay extensiones de Chrome específicas para ser usadas como clientes REST: Postman o REST Console
- En Chrome las extensiones se gestiona en la página `chrome://extensions/`



Clientes de servicios REST

Herramientas interactivas





Clientes de servicios REST

Cliente Java con Jackson

- El cliente se puede implementar con las clases básicas de la librería estándar de Java que permiten hacer una petición HTTP a una URL
- Para procesar la información JSON en el cliente usaremos la librería Jackson (http://jackson.codehaus.org/)
- Dependencia:

```
<dependency>
   <groupId>com.fasterxml.jackson.core</groupId>
   <artifactId>jackson-databind</artifactId>
   <version>2.7.3</version>
</dependency>
```

Clientes de servicios REST

Cliente Java con Jackson

- Ejemplo:

```
public class JacksonClient {

   public static void main(String[] args) throws Exception {
      // Http request
      URL url = new URL("http://localhost:8080/team/0");
      HttpURLConnection conn = (HttpURLConnection) url.openConnection();
      conn.connect();

      // Configure Jackson parser
      ObjectMapper mapper = new ObjectMapper();

      // Parse response
      Team team = mapper.readValue(conn.getInputStream(), Team.class);

      // Use response
      System.out.println(team);
   }

}
```

Clientes de servicios REST

Cliente Java con Spring REST Template

- Podemos usar librerías de más alto nivel para realizar las peticiones REST
- **Spring REST Template** es la implementación de Spring
- Encapsula en una única llamada la petición y el "*parseo*" de la respuesta
- Ejemplo GET:

```
RestTemplate restTemplate = new RestTemplate();
String url = "http://localhost:8080/team/1";
Team team = restTemplate.getForObject(url, Team.class);
System.out.println(team);
```

- Más información en: https://spring.io/guides/gs/consuming-rest/

Clientes de servicios REST

Cliente Java con Retrofit

- La librería Retrofit ofrece un enfoque de un nivel más alto de abstracción para implementar un cliente REST en Java
- Se define un interfaz Java con los métodos que reflejan los servicios de la API
- Estos métodos se anotan para especificar detalles de la API REST
- La aplicación cliente sólo tiene que invocar estos métodos para consumir el servicio REST
- Dependencia:

```
<dependency>
   <groupId>com.squareup.retrofit</groupId>
   <artifactId>retrofit</artifactId>
   <version>1.9.0</version>
</dependency>
```

http://square.github.io/retrofit/

Clientes de servicios REST

Cliente Java con Retrofit

Interfaz con la descripción del servicio REST

```
public interface TeamsService {

    @GET("/teams")
    List<Team> getTeams();

    @GET("/team/{index}")
    Team getTeam(@Path("index") int index);

    @POST("/teams")
    boolean addTeam(@Body Team team);
}
```

La anotación @Body se usa para indicar que el parámetro va en el cuerpo de la petición (no en la URL)

Clientes de servicios REST

Cliente Java: Retrofit

Ejemplo de consulta GET y POST

```
public static void main(String[] args) throws Exception {
   // GET
   RestAdapter adapter = new RestAdapter.Builder().setEndpoint(
      "http://localhost:8080").build();
   TeamsService service = adapter.create(TeamsService.class);
   Team team = service.getTeam(0);
   System.out.println(team);

   // POST
   List<Player> players = new ArrayList<Player>();
   players.add(new Player("M.A.", "Barracus"));
   players.add(new Player("Murdock", "Crazy"));
   Team aTeam = new Team("A Team", players);
   boolean created = service.addTeam(aTeam);
   System.out.println("Created: " + created);
}
```

Clientes de servicios REST

Cliente JavaScript con jQuery

- Las aplicaciones web con AJAX o con arquitectura SPA, implementadas con JavaScript, usan servicios REST desde el navegador
- Al igual que en Java, existen muchas formas de usar servicios REST en JavaScript en el navegador
- Uno de los mecanismos más usados es usar la librería **jQuery**

Clientes de servicios REST

Cliente JavaScript con jQuery

Ejemplo GET

```
<!DOCTYPE html>
<html>
<head>
<script src="https://code.jquery.com/jquery-3.0.0.min.js"></script>
<script>
   $(function() {
      $.ajax({
         url : "http://localhost:8080/team/0"
      }).then(function(data) {
         $('.team-name').append(data.name);
         $('.team-players').append(JSON.stringify(data.players));
      });
   });
</script>
</head>
<body>
   <div>
      <p class="team-name">Team:</p>
      <p class="team-players">Players:</p>
   </div>
</body>
</html>
```

Esta función convierte el objeto data.players en un String

Clientes de servicios REST

Cliente JavaScript con jQuery

Ejemplo POST

```
$(function() {
   var newTeam = {
      name : "New team name",
      players : [ {
         "name" : "Player 1",
         "nickname" : "Nick 1"
      }, {
         "name" : "Player 2",
         "nickname" : "Nick 2"
      } ]
   };
   $.ajax({
      type : "POST",
      data : JSON.stringify(newTeam),
      contentType : "application/json",
      url : "http://localhost:8080/teams"
   }).then(function(data) {
      $('.result').append(data);
   });
});
```