Tema 4. Servicios REST

Programación web

Boni García

Curso 2017/2018

Índice

1. Introducción
2. Servicios REST
3. Clientes de servicios REST

Índice

1. Introducción
   - Servicios web
   - Servicios REST
   - JSON
   - Arquitectura orientada en servicios
2. Servicios REST
3. Clientes de servicios REST

Introducción

Servicios web

- Un **servicio distribuido** (o servicio telemático) consiste en varios procesos que se ejecutan en diferentes equipos terminales y que se comunican a través de una red de datos
- 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 (típicamente XML o JSON)

Introducción

Servicios web

- Los clientes de los servicios web puede ser diferente naturaleza:
  - JavaScript, AJAX, aplicaciones móviles, …
  - Servidores de otras aplicaciones web
- Los servicios web tienen como principal ventaja la disponibilidad: el puerto 80 TCP saliente suele estar abierto en Firewalls, con lo que cual los clientes REST podrán hacer peticiones de forma directa
- Hay dos tipos principales de servicios web:
  - **SOAP** (*Simple Object Access Protocol*)
  - **REST** (*REpresentational State Transfer*) ← Vamos a estudiar solamente estos

Introducción

Servicios REST

- REST es un estilo arquitectónico para servicios distribuidos
  - A los servicios que siguen la arquitectura REST se les conoce como *RESTful*
  - Si no se usa la arquitectura de forma estricta se dice que el servicio es *REST-like*
- El terminó lo acuñó Roy Fielding en su tesis doctoral en el año 2000
- Roy Fielding es uno de los autores del protocolo HTTP y también del servidor web Apache
- Los servicios REST permite realizar operaciones CRUD sobre **recursos**, llamadas **acciones**
- Cada recurso se identifica con una URL (también conocida como ***endpoint***)
- Cada recurso tienen una **representación**, típicamente en formato JSON

Introducción

Servicios REST

- Los servicios REST se implementan normalmente sobre el protocolo HTTP (aunque en teoría se podría usar con otro protocolo)
  - **`GET`**: Leer un recurso
  - **`POST`**: Enviar un recurso al servidor
  - **`PUT`**: Actualizar un recurso
  - **`DELETE`**: Eliminar un recurso
  - **`PATCH`**: Actualizar parcialmente un recurso
  - **`HEAD`**: Preguntar si un recurso existe sin devolver su representación
  - **`OPTIONS`**: Obtener la lista de acciones disponibles para un determinado recurso

Las acciones **`GET`**, **`DELETE`** y **`PUT`** deberían ser idempotentes

Introducción

Servicios REST

- Los códigos de respuesta HTTP se reutilizan en servicios REST:
  - **`200 OK`**: La petición ha ido bien (normalmente en respuestas a `GET`)
  - **`201 Created`**: Recurso creado (normalmente en respuestas a `POST` o `PUT`)
  - **`204 No content`**: Acción correcta pero no se envía contenido de respuesta (normalmente se usa como respuesta a `DELETE`)
  - **`400 Bad request`**: Petición errónea (por ejemplo, faltan parámetros)
  - **`404 Not found`**: La URL no identifica a ningún recurso
  - **`405 Method not allowed`**: El verbo HTTP no se permite para un recurso (por ejemplo un `PUT` de un recurso de sólo lectura)
  - **`500 Internal server error`**: Error genérico en el servidor

JSON

JSON

- **JSON** (*JavaScript Object Notation*), es un lenguaje de marcado ligero para almacenar o enviar información estructurada
- 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**

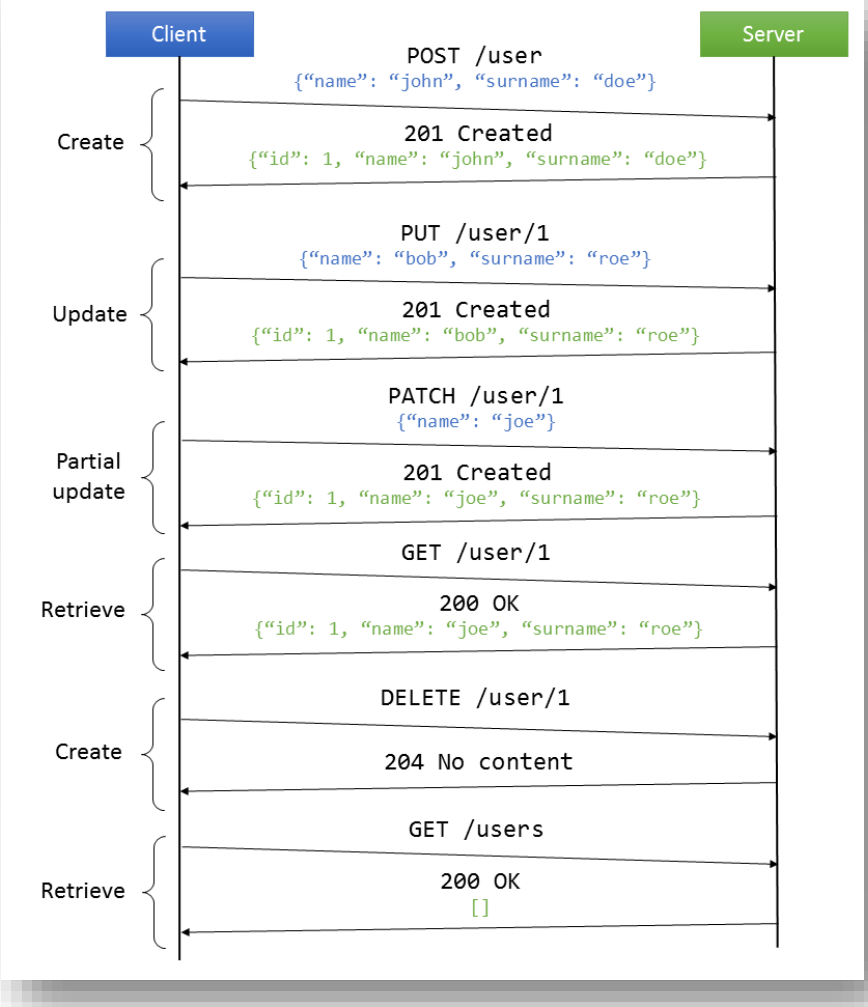
http://www.json.org/

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

Introducción

JSON

- Ejemplo de servicio REST con peticiones y respuestas en formato JSON:



Introducción

Arquitectura orientada en servicios

- La arquitectura de las aplicaciones basada en servicios distribuidos se conoce como **SOA** (*Service Oriented Architecture*)
- Un tipo de arquitectura SOA muy usado hoy día se conoce como **arquitectura de microservicios**
  - Los servicios interactúan a través de APIs (típicamente REST)
  - Se suelen desplegar en infraestructuras en la nube con técnicas de que proporcionan escalabilidad y tolerancia a fallos
  - Suelen estar basadas en tecnologías de contenedores (como **Docker** o **Kubernetes**)

Índice

1. Introducción
2. Servicio REST
   - Diseño de un servicio REST
   - Implementación de un servicio REST
3. Clientes de servicios REST

Diseño de un servicio REST

Diseño de un servicio REST

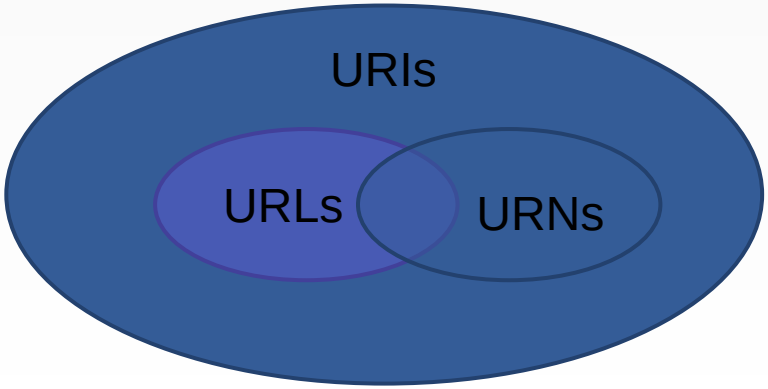
- El esquema habitual que define el funcionamiento de los servicios REST es el siguiente:
  1. La identificación de recursos mediante URLs
  2. Las operaciones se realizan mediante métodos HTTP
  3. La representación de los recursos se realiza típicamente en JSON
  4. Los códigos de respuesta HTTP notifican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

Para ampliar: **OpenAPI**
https://swagger.io/specification/

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 mediante URLs. Ejemplos:
   - http://server.tld/users/bob
   - http://server.tld/users/bob/anuncio/44
2. Las operaciones se realizan mediante métodos HTTP

- **GET**: Devuelve el recurso, generalmente codificado en JSON
- **DELETE**: Borra el recurso
- **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
- **PATCH**: Modificación parcial de un recurso

Diseño de un servicio REST

Diseño de un servicio REST

3. La representación se realiza típicamente en JSON

- 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"
}
```

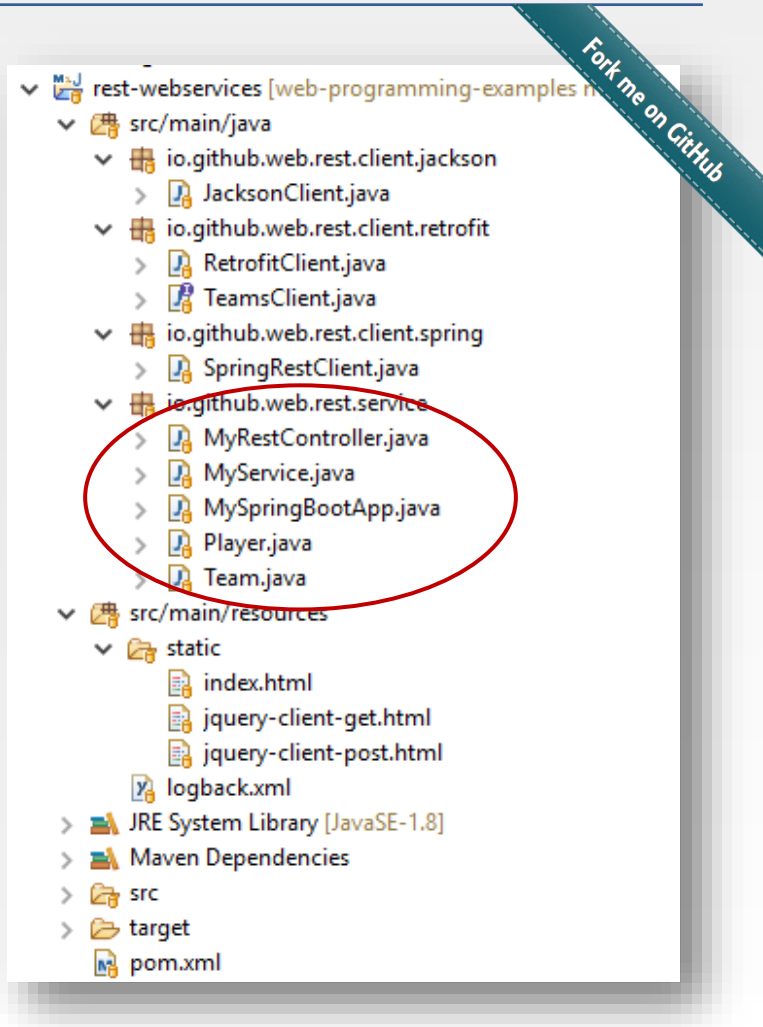
4. Los códigos de respuesta HTTP notifican el resultado de la operación

- `2xx`: La petición fue procesada correctamente
- `4xx`: Cliente ha realizado la petición incorrectamente
- `5xx`: Error procesando la petición

Servicios REST

Implementación de un servicio REST

- **Spring MVC** se usará también para implementar servicios REST
  - Se usa la anotación `@RestController` en lugar de `@Controller`
  - Los métodos devuelven objetos de tipo `ResponseEntity` en lugar de `ModelAndView`
- Proyecto de ejemplo: `rest-webservices`
  - Gestiona una lista de equipos (clase `Team`)
  - Cada equipo tiene un nombre y una lista de jugadores (clase `Player`)



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>2.0.1.RELEASE</version>
</parent>

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

Necesitamos Spring MVC pero no capa de presentación (Thymeleaf)

Servicios REST

Implementación de un servicio REST

- Ejemplo de servicio REST con Spring MVC

```java
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

}
```

```java
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

Habilitamos orígenes cruzados (CORS)

Usamos `@RequestMapping` para identificar los recursos

Devolvemos un código de respuesta HTTP junto con el cuerpo (*body*) usando el objeto `ResponseEntity`

```
@CrossOrigin
@RestController
public class MyRestController {

    @Autowired
    private MyService teamsService;

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

    @RequestMapping(value = "/teams", method = RequestMethod.POST)
    public ResponseEntity<Integer> addTeam(@RequestBody Team team) {
        teamsService.addTeam(team);
        int teamSize = teamsService.size();
        HttpStatus status = HttpStatus.CREATED;
        ResponseEntity<Integer> response = new ResponseEntity<>(teamSize, status);
        return response;
    }
                    :
```

Servicios REST

Implementación de un servicio REST

- Ejemplo de servicio REST con Spring MVC

```
    @RequestMapping(value = "/team", method = RequestMethod.GET)
    public ResponseEntity<Team> getTeamByQuery(
            @RequestParam("index") int index) {
        return getTeam(index);
    }

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

    private ResponseEntity<Team> getTeam(int index) {
        Team team = teamsService.getTeam(index);

        HttpStatus status = HttpStatus.OK;
        ResponseEntity<Team> response = new ResponseEntity<>(team, status);
        return response;
    }

}
```

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 MySpringBootApp {

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

Índice

1. Introducción
2. Servicios REST
3. Clientes de servicios REST
   - Herramientas interactivas
   - Cliente Java con Jackson
   - Cliente Java con Spring REST Template
   - Cliente Java con Retrofit
   - Cliente JavaScript con jQuery

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

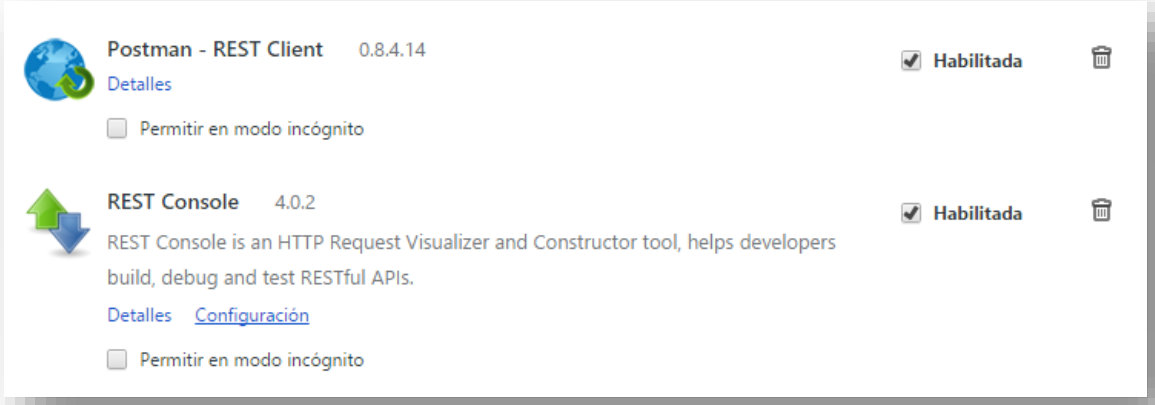
Herramientas interactivas

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

Clientes de servicios REST

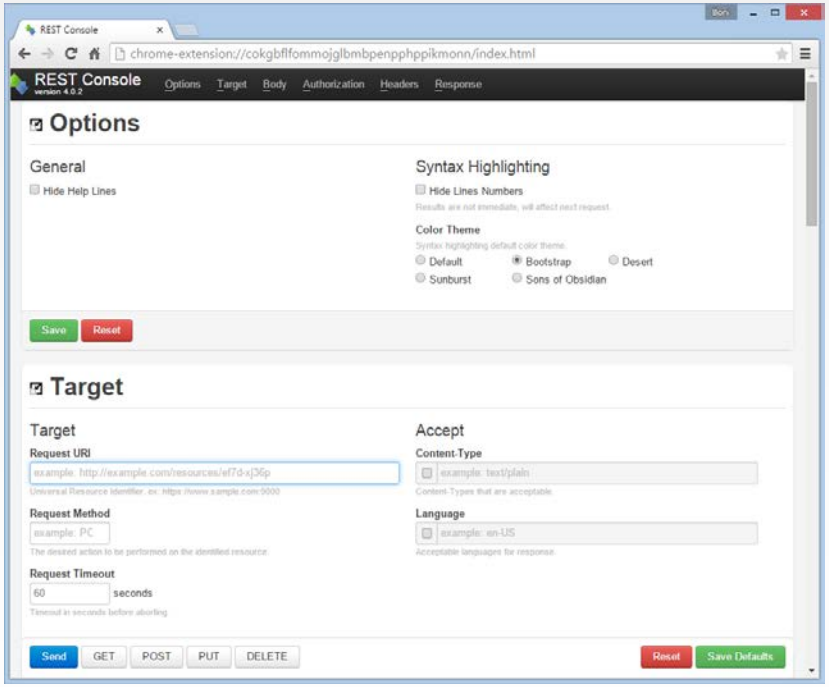
Herramientas interactivas

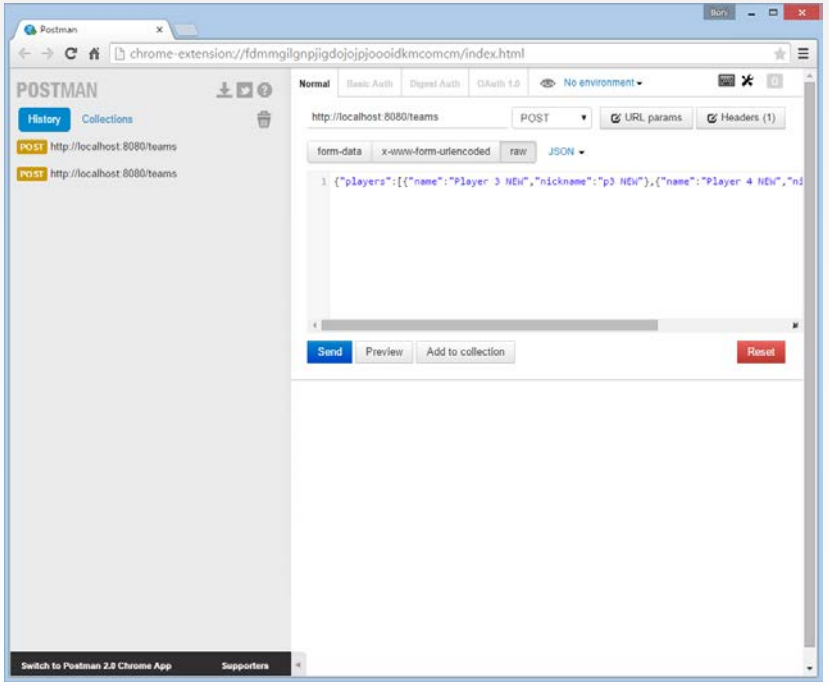
- Existen extensiones de los navegadores que nos permiten realizar cualquier tipo de petición REST
- Por ejemplo, hay extensiones de Chrome específicas para ser usadas como clientes REST: Postman o REST Console
- En Chrome las extensiones se gestiona en la página `chrome://extensions/`



Clientes de servicios REST

Herramientas interactivas





Clientes de servicios REST

Cliente Java con Jackson

- **Jackson** es un librería que permite parear JSON desde Java
- Una alternativa a Jackson es **Gson** (librería de Google para usar JSON en Java)

```
public class JacksonClient {

    private static final Logger log = LoggerFactory
            .getLogger(JacksonClient.class);

    public static void main(String[] args) throws Exception {
        // Create Jackson parser
        ObjectMapper mapper = new ObjectMapper();

        // Make request
        URL url = new URL("http://localhost:8080/team/0");
        Team team = mapper.readValue(url, Team.class);
        log.info("Response {}", team);
    }

}
```

http://jackson.codehaus.org/

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:

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

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
- Se define un interfaz 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

```
<dependency>
    <groupId>com.squareup.retrofit2</groupId>
    <artifactId>retrofit</artifactId>
    <version>${retrofit.version}</version>
</dependency>
<dependency>
    <groupId>com.squareup.retrofit2</groupId>
    <artifactId>converter-jackson</artifactId>
    <version>${retrofit.version}</version>
</dependency>
```

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

Clientes de servicios REST

Cliente Java con Retrofit

Interfaz con la descripción del servicio REST

```
public interface TeamsClient {

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

    @GET("/team/{index}")
    Call<Team> getTeamByPath(@Path("index") int index);

    @GET("/team")
    Call<Team> getTeamByQuery(@Query("index") int index);

    @POST("/teams")
    Call<Integer> 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 {
     // REST client
     String serviceUrl = "http://localhost:8080";
     Retrofit retrofit = new Retrofit.Builder()
             .addConverterFactory(JacksonConverterFactory.create())
             .baseUrl(serviceUrl).build();
     TeamsClient teamsClient = retrofit.create(TeamsClient.class);

     // GET
     Response<Team> getResponse = teamsClient.getTeamByPath(0).execute();
     int getResponseCode = getResponse.code();
     Team getResponseBody = getResponse.body();
     log.info("GET response: {} -- {}", getResponseCode, getResponseBody);

     // POST
     List<Player> players = new ArrayList<>();
     players.add(new Player("M.A.", "Barracus"));
     players.add(new Player("Murdock", "Crazy"));
     Team aTeam = new Team("A Team", players);
     Response<Integer> postResponse = teamsClient.addTeam(aTeam).execute();
     int postResponseCode = postResponse.code();
     int postResponseBody = postResponse.body();
     log.info("POST response: {} -- {}", postResponseCode, postResponseBody);
}
```

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-3.3.1.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(response) {
      $('.result').append(response);
   });
});
```