FRAMEWORK

REST

Módulo IV

harriague+asociados

Emilio Rey | María Agustina Aliaga

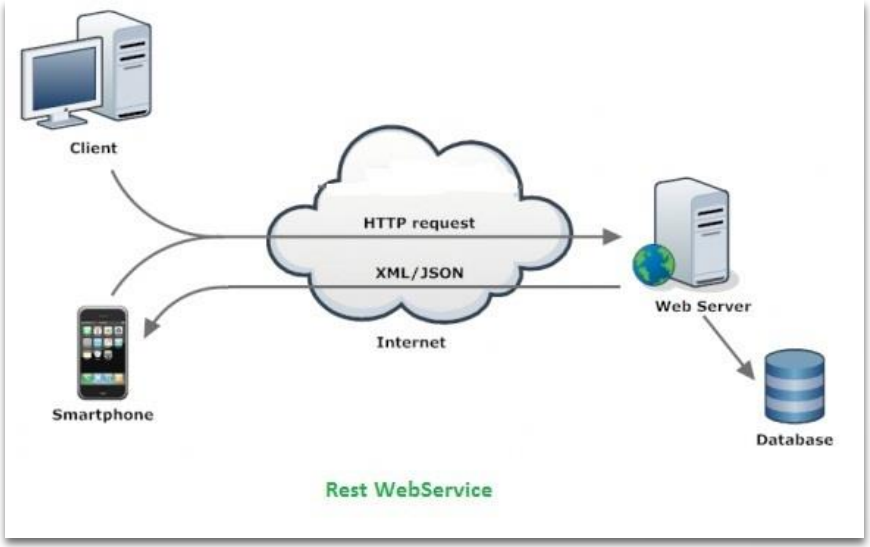
INCLUIT
empowering your business

REST

Es una arquitectura orientada a recursos. Define un conjunto de propiedades basadas en HTTP. Permite la interoperabilidad entre diferentes sistemas.

Representational **S**tate **T**ransfer

- **Representational**– Los recursos REST pueden ser representados en cualquier formato incluyendo XML, JavaScript Object Notation (JSON), o HTML.
- **State** - Al trabajar con REST el estado del recurso es más importante que las acciones contra el recurso.
- **Transfer**–REST implica transferir datos de una aplicación a otra, representados en algún formato.



JSON

JavaScript Object Notation, es un formato de texto ligero para el intercambio de datos entre aplicaciones.

Ejemplo de JSON

```
[
  {
    "id": 101,
    "firstName": "John",
    "lastName": "Doe",
    "email": "djohn@gmail.com",
    "mobile": "121-232-3435",
    "dateOfBirth": 1527183314273
  }
]
```

JSON

```
{
"siblings": [
{"firstName":"Anna","lastName":"Clayton"},
{"firstName":"Alex","lastName":"Clayton"}
]
}
```

XML

```
<siblings>
<sibling>
<firstName>Anna</firstName>
<lastName>Clayton</lastName>
</sibling>
<sibling>
<firstName>Alex</firstName>
<lastName>Clayton</lastName>
</sibling>
</siblings>
```

```
{
        "title": "Example Schema",
        "type": "object",
        "properties": {
                "firstName": {
                        "type": "string"
                },
                "lastName": {
                        "type": "string"
                },
                "age": {
                        "description": "Age in years",
                        "type": "integer",
                        "minimum": 0
                }
        },
        "required": ["firstName", "lastName"]
}
```

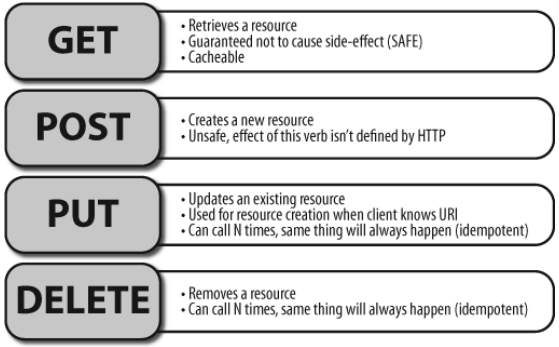
Métodos HTTP / verbos REST

Los **recursos REST** son **identificados por URL.**

No hay reglas estrictas en cuanto a la **estructura** de una URL de un recurso REST, sin embargo convencionalmente éstas deben identificar al recurso, no solamente enviarles un comando al servidor.

Si bien el **énfasis** está puesto en las **cosas no en las acciones**, **existen acciones en REST, que son definidas por los métodos HTTP** (GET , POST , PUT , DELETE , PATCH) que son los verbos REST que se corresponden con los métodos CRUD.

Operation	SQL	HTTP
Create	INSERT	PUT / POST
Read (Retrieve)	SELECT	GET
Update (Modify)	UPDATE	PUT / PATCH
Delete (Destroy)	DELETE	DELETE



Códigos de Estado HTTP

Postman

Postman es una **extensión** gratuita para el **navegador Google Chrome** que permite **probar servicios web** fácilmente.

Para esto es necesario indicar:

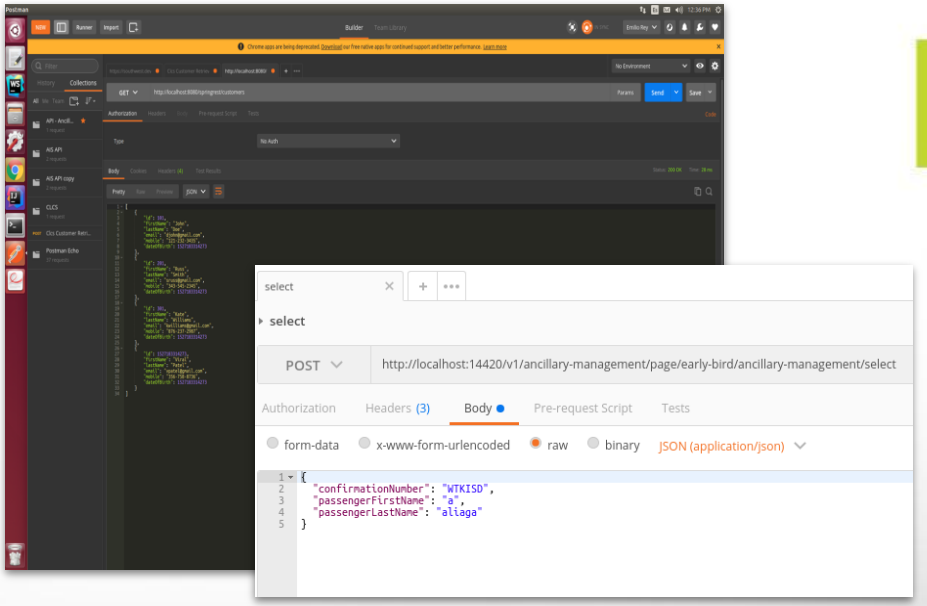
- La URL
- El método HTTP (POST, GET, etc.)
- Los parámetros de la petición.

Al utilizarlo, Postman, guarda las peticiones más recientes en el historial, que se encuentra a la izquierda de la pantalla, para tenerlas a la mano.

Permite definir colecciones y guardar una serie de métodos para reutilizarlos y poder compartirlos.

Podemos crear ambientes y definir variables específicas para ellos.

Si usamos autenticación en el servicio web, nos permite utilizar identificación http básica, con digestión, OAuth 1.0 y próximamente soportará OAuth 2.0.



HTTP Converter (org.springframwork.http.converter)

Message Converter	Description
AtomFeedHttpMessageConverter	Convierte objetos feed de ROME desde y hacia feeds Atom (media type application/atom+xml). Es registrado si la librería ROME se encuentra presente en el classpath.
BufferedImageHttpMessageConverter	Convierte BufferedImage desde y hacia datos binarios de imagen.
ByteArrayHttpMessageConverter	Lee y escribe arreglos de bytes. Lee de cualquier media type (*/*), y escribe como application/octet-stream.
FormHttpMessageConverter	Lee el contenido como application/x-www-form-urlencoded transformándolo en un MultiValueMap<String,String>. Escribe MultiValueMap<String,String> como application/x-www-form-urlencoded y MultiValueMap<String, Object> como multipart/form-data.
Jaxb2RootElementHttpMessageConverter	Lee y escribe XML (text/xml o application/xml) desde y hacia objetos anotados del tipo JAXB2. Es registrado si la librería JAXB v2 se encuentra presente en el classpath.

Message Converter	Description
MappingJacksonHttpMessageConverter	Lee y escribe JSON desde y hacia typed objects o untyped HashMaps. Es registrado si la librería Jackson JSON se encuentra presente en el classpath.
MappingJackson2HttpMessageConverter	Lee y escribe JSON desde y hacia typed objects o untyped HashMaps. Es registrado si la librería Jackson 2 se encuentra presente en el classpath.
MarshallingHttpMessageConverter	Lee y escribe XML usando un marshaler y un unmarshaler inyectados. Es compatible con los (un)marshalers Castor, JAXB2, JIBX, XMLBeans, y XStream.
ResourceHttpMessageConverter	Lee y escribe org.springframework.core.io.Resource.
RssChannelHttpMessageConverter	Lee y escribe feeds RSS desde y hacia Objetos Channel ROME. Es registrado si la librería ROME se encuentra presente en el classpath.
SourceHttpMessageConverter	Lee y escribe XML desde y hacia objetos javax.xml.transform.Source.
StringHttpMessageConverter	Lee todos los media types (*/*) y los transforma en un String. Escribe String en text/plain.
XmlAwareFormHttpMessageConverter	Una extensión de FormHttpMessageConverter que añade soporte a partes basadas en XML utilizando un SourceHttpMessageConverter.

Retornando Resource State en el Response Body

```
@RequestMapping(method=RequestMethod.GET, produces="application/json")
public @ResponseBody List<Spittle> spittles(
                        @RequestParam(value="max", defaultValue=MAX_LONG_AS_STRING) long max,
                        @RequestParam(value="count", defaultValue="20") int count) {
        return spittleRepository.findSpittles(max, count);
}
```

Result

```
[
  {
        "id": 42,
        "latitude": 28.419489,
        "longitude": -81.581184,
        "message": "Hello World!",
        "time": 1400389200000
  }
]
```

Recibiendo Resource State en el Request Body

```
@RequestMapping(method=RequestMethod.POST, consumes="application/json")
public @ResponseBody Spittle saveSpittle(@RequestBody Spittle spittle) {
        return spittleRepository.save(spittle);
}
```

Nota:
En este caso si nosotros queremos probar si funciona correctamente o mejor dicho si los datos que le estamos pasando a través de Postman matchean o bindean con la estructura **JSON** que se corresponde con el objeto **Spittle,** la estructura del JSON debería ser igual que al objeto que va a recibir el método

Ejemplo:

```
{
        "latitude": 28.419489,
        "longitude": -81.581184,
        "message": "Hello World!",
        "time": 1400389200000
}
```

Conversión por defecto para los mensajes de los controllers

Spring 4.0 introdujo la anotación **@RestController**.

Si se utiliza**@RestController** para anotar una clase en vez de **@Controller**, Spring aplica la conversión del mensaje para todos los handlers en el controller.

Ya no es necesario anotar cada método con **@ResponseBody** .

@Controller

```java
@Controller
@RequestMapping("books")
public class SimpleBookController {

    @GetMapping("/{id}", produces = "application/json")
    public @ResponseBody Book getBook(@PathVariable int id) {
        return findBookById(id);
    }

    private Book findBookById(int id) {
        // ...
    }
}
```

@RestController

```java
@RestController
@RequestMapping("books-rest")
public class SimpleBookRestController {

    @GetMapping("/{id}", produces = "application/json")
    public Book getBook(@PathVariable int id) {
        return findBookById(id);
    }

    private Book findBookById(int id) {
        // ...
    }
}
```

Manejo de Errores

Una anotación del tipo **@ControllerAdvice** permite a métodos del tipo **@ExceptionHandler , @InitBinder** y **@ModelAttributes** estar en una misma clase y ser aplicados a todos los controllers.

Esto puede ser usado para el **manejo de excepciones** a través de **múltiples controllers.**

@RestControllerAdvice es una nueva funcionalidad en Spring 4.3. Esta anotación combina **@ControllerAdvice + @ResponseBody**.

@RestControllerAdvice es útil para manejar excepciones con Apis REST a través de una solución across: **@ExceptionHandler.**

```
import javax.servlet.http.HttpServletRequest;
import java.net.ConnectException;
import java.util.Map;

/**
 *
 * @author emilio.rey
 */
@RestControllerAdvice
public class ControllerExceptionHandler {

    private static Logger LOGGER = LogManager.getLogger();

    @ExceptionHandler(ImageNotFoundException.class)
    public ResponseEntity<Map<String, Object>> handleImageNotFoundException(ImageNotFoundException imageNotFoundException, HttpServletRequest httpServletReque
        LOGGER.error(ImageNotFoundException.class.getName(),imageNotFoundException);
        final Map<String, Object> responseBody = new DefaultExceptionAttributes().getExceptionAttributes(imageNotFoundException, httpServletRequest,
                HttpStatus.NOT_FOUND);
        return new ResponseEntity<>(responseBody, HttpStatus.NOT_FOUND);
    }

    @ExceptionHandler(NoResultException.class)
    public ResponseEntity<Map<String, Object>> handleNoResultException(NoResultException noResultException,
                                                                        HttpServletRequest httpServletRequest){
```

Manejo de Errores

```
public class Error {
   private int code;
   private String message;
   public Error(int code, String message) {
      this.code = code;
      this.message = message;
   }
   public int getCode() {
     return code;
   }
   public String getMessage() {
     return message;
   }
}
```

Podríamos hacer que Status genere la respuesta de la siguiente manera:

```
@ExceptionHandler(SpittleNotFoundException.class)
@ResponseStatus(HttpStatus.NOT_FOUND)
public @ResponseBody Error spittleNotFound(SpittleNotFoundException e) {
  long spittleId = e.getSpittleId();
  return new Error(4, "Spittle [" + spittleId + "] not found");
}
```

Si el controller está anotado como **@RestController** nosotros podríamos remover la anotación **@ResponseBody**

```
@ExceptionHandler(SpittleNotFoundException.class)
@ResponseStatus(HttpStatus.NOT_FOUND)
public Error spittleNotFound(SpittleNotFoundException e) {
  long spittleId = e.getSpittleId();
  return new Error(4, "Spittle [" + spittleId + "] not found");
}
```