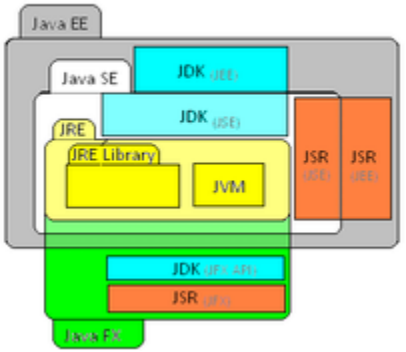
Spring Framework

INTRODUCCIÓN A SPRING FRAMEWORK

- Framework: conjunto de clases que nos permiten resolver un problema en específico.
  - Spring: permite resolver muchos de los problemas que se presentan al desarrollar aplicaciones con tecnología JEE (Java Enterprise Edition).
- Spring Framework utilizado para el desarrollo de aplicaciones empresariales con tecnologías JEE.
  - Objetivo: simplificar el desarrollo de aplicaciones empresariales Java.
- Principal ventaja de Spring: La forma modular en el que fue creado, permitiendo habilitar/deshabilitar las características a utilizar según se requiera.
- Página Oficial de Spring: www.springsource.org
  - Se pueden encontrar todos los proyectos relacionados con dicha tecnología.
- Spring es utilizado en proyectos muy diversos, como puede ser en Instituciones Bancarias, Aseguradoras, Instituciones Educativas y de Gobierno, entre muchos otros tipos de proyectos y empresas.

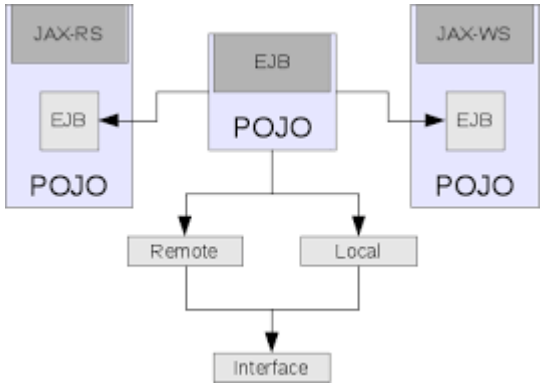
Java
Enterprise Edition

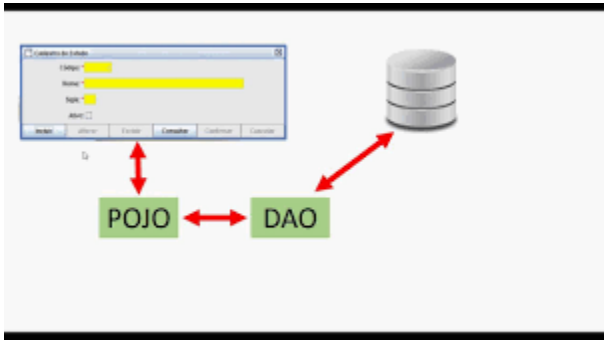


CARACTERÍSTICAS DE SPRING FRAMEWORK

- Permite desarrollar aplicaciones flexibles, altamente cohesivas y con un bajo acoplamiento.

- Permite simplificar el desarrollo JEE al utilizar clases Java Simples (POJO – Plain Old Java Object) para la configuración de servicios.





- Muchos proyectos muestran las mismas tareas a realizar una y otra vez: Localización de Servicios, Manejo de Transacciones, Manejo de Excepciones, Parametrización de la aplicación, entre muchos más.
  - Spring permite resolver muchos de estos problemas de manera muy simple. Para lograr lo anterior el framework se base en dos conceptos fundamental:
    - DI (Dependency Inyection): Este patrón de diseño permite suministrar objetos a una clase (POJO) que tiene dependencias, en lugar de ser ella misma sea quien los proporcione.
    - AOP (Aspect Oriented Programming): AOP es un paradigma de programación que permite modularizar las aplicaciones y mejorar la separación de responsabilidades entre módulos y/o clases.

DI y AOP son la base para la creación de Contenedores ligeros (lightweight containers).

Spring es uno de los contenedores ligeros más completos y populares al día de hoy.

MODULOS DE SPRING FRAMEWORK

Spring se compone de distintos módulos, permitiendo seleccionar solo algunos de ellos o todos, dependiendo de la naturaleza de la aplicación. A continuación listaremos varios de ellos:

1. Spring Core: Este módulo provee la funcionalidad básica de la fábrica de Spring. El componente principal es BeanFactory, el cual aplica el concepto de Inversion of Control (IoC) o también conocido como Dependency Injection (DI).
2. Spring Context: Aquí es donde se realiza la configuración del framework. Incluye la configuración de servicios empresariales tales como JNDI, EJB, Internacionalización, validación, entre varios más.
3. Spring AOP: Permite aplicar los conceptos de Programación Orientada a Aspectos (AOP), además incluye clases de soporte para el manejo transaccional, la seguridad, entre varias clases más, permitiendo desacoplar estas características de nuestra aplicación.

MODULOS DE SPRING FRAMEWORK (cont.)

4. Spring DAO: Permite aplicar conceptos de la capa de datos Data Access Object (DAO) a través de POJOs (Plain Old Java Object), abstrayendo la complejidad, permitiendo crear un código JDBC más limpio y simple.
5. Spring ORM: Permite integrarse con tecnologías tales como JPA, Hibernate, entre otras.
6. Spring Web: Permite el desarrollo y la integración con tecnologías como Struts, JSF, Tapestry, entre otros.
7. Spring MVC: Este módulo implementa el patrón MVC para ser utilizado en la capa de presentación.

controller

REST CONTROLLERS / ENDPOINTS

service

FUNCTIONALITIES AND LOGIC

model

DATA REPRESENTATION - ENTITIES

repository

ACCESS TO THE DATABASE

ARQUITECTURA MULTICAPAS

Una aplicación empresarial en Java se compone de distintas capas, cada capa tiene una función muy específica.

- Dividir una aplicación en capas tiene varias ventajas, como son separación de responsabilidades, un mejor mantenimiento a la aplicación, especialización de los programadores en cada capa, entre muchas más.

Spring es un framework que resuelve varios problemas de distintas capas, desde la capa de presentación, la capa de negocio y la capa de datos.

- Aunque lo más común es que se combine con otras tecnologías y Spring quede como el orquestador de la capa de Servicio.

capas de una aplicación multicapas.

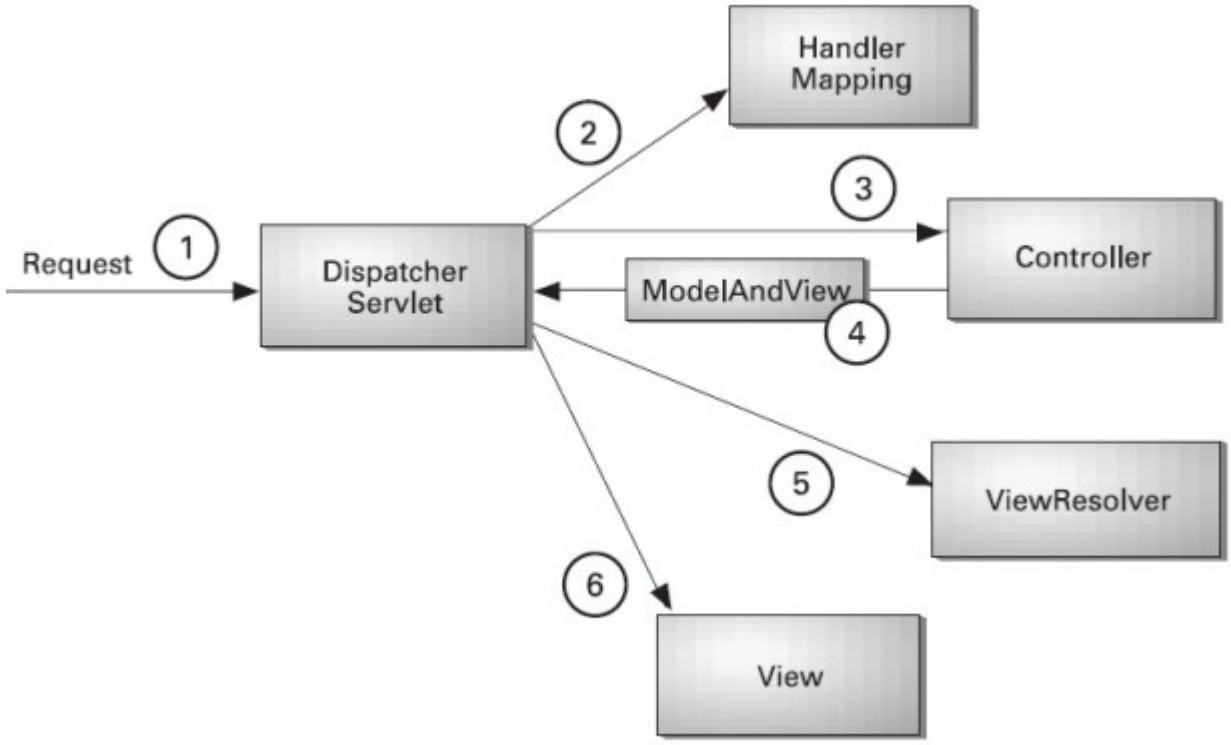
- Capa Web: La capa del Cliente es donde el cliente interactúa por medio de un navegador Web, un cliente móvil, una aplicación de escritorio, entre otros.
  - Puede residir en un servidor web, las tecnologías mas básicas que podemos encontrar en este servidor web son los JSP's y los Servlets.
- Capa de Negocio: en esta capa podemos encontrar tecnología como son los Enterprise Java Beans (EJBs) o frameworks como Spring.
- Capa de Datos: aquí vamos a encontrar tecnologías como JDBC, Hibernate, entre otras. Este código nos va a permitir comunicarnos con nuestra base de datos para leer y almacenar información en ella.

PORTAFOLIO DE SPRING

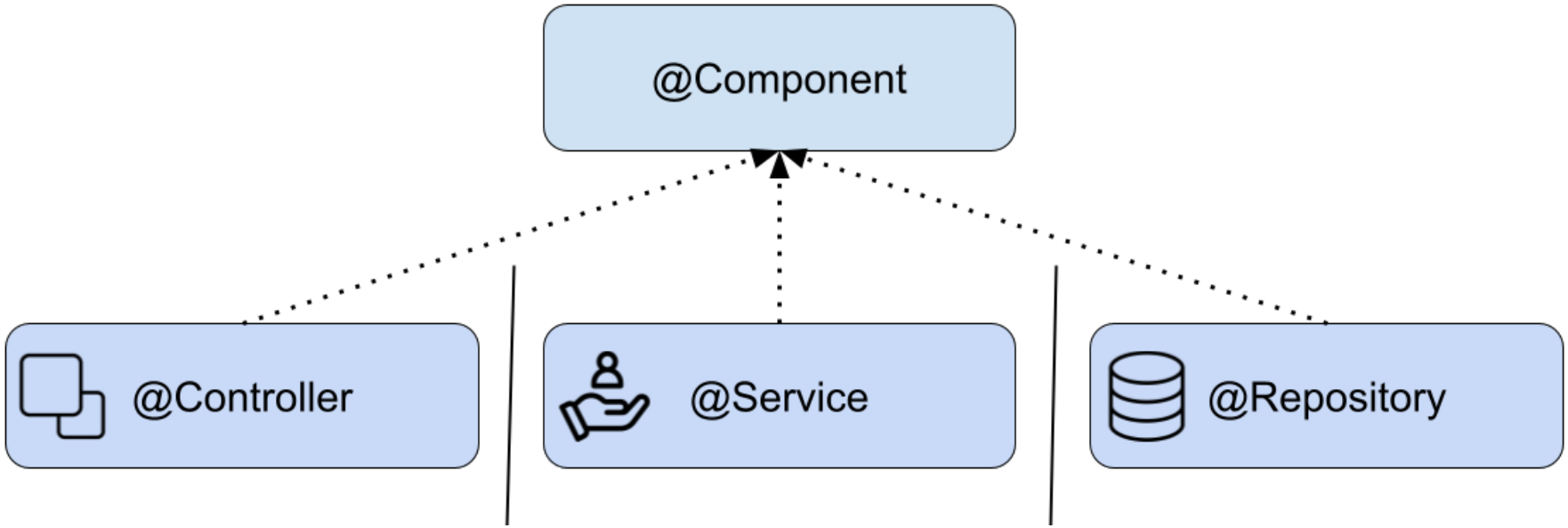
El portafolio de soluciones bastante amplia, además de Spring Core.

- Spring Web Flow está construido sobre Spring MVC, con el objetivo de definir y gestionar flujos entre páginas dentro de una aplicación Web.
- Spring Web Services (Spring-WS) permite facilitar la creación de Servicios Web basados en el intercambio de documentos (document driven o contract first).
- Spring Security es el módulo de seguridad para aplicaciones Web, inicialmente conocido como ACEGI framework.
- Spring Batch es el módulo de Spring que nos permite crear procesos batch, formado por una secuencia de pasos.
- Spring Social provee conectividad y autorización a redes sociales como Facebook, Twitter, Google+, Linkedin, etc.
- Spring Mobile es una extensión de Spring MVC, con el objetivo de simplificar el desarrollo de aplicaciones Web móviles.
- Spring Roo permite el desarrollo rápido de aplicaciones Java.

Varios más.

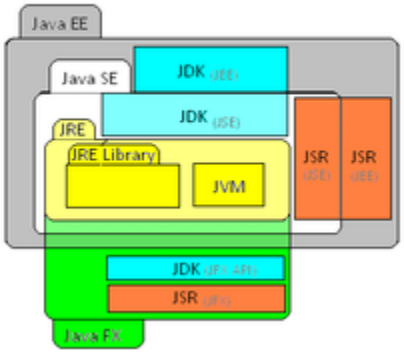

Handler
Mapping
2
3
Request
1
Dispatcher
Servlet
ModelAndView
Controller
4
5
ViewResolver
6
View

@Component - **estereotipos**



INTRODUCCIÓN A SPRING FRAMEWORK

1.@SpringBootApplication: es una anotación de conveniencia que combina las anotaciones @Configuration, @EnableAutoConfiguration y @ComponentScan, lo que significa que se puede utilizar para marcar la clase principal de una aplicación Spring Boot.
2.@RestController: es una anotación que se utiliza para marcar una clase que proporciona servicios web RESTful. Esta anotación combina la anotación **@Controller** con la anotación **@ResponseBody**.
3.@RequestMapping: es una anotación que se utiliza para mapear una solicitud HTTP a un método de controlador. Puede especificar la URL, el método HTTP, el tipo de contenido y otros parámetros.
4.@Autowired: es una anotación que se utiliza para inyectar automáticamente una dependencia en un componente de Spring. Spring Boot buscará un objeto que coincida con el tipo de la dependencia y lo inyectará en el componente.
5.@Component: es una anotación que se utiliza para marcar una clase como un componente de Spring. Spring Boot buscará clases anotadas con @Component y las registrará como beans de la aplicación.
6.@Configuration: es una anotación que se utiliza para marcar una clase como una fuente de configuración para la aplicación. Puede definir beans y configurar otros componentes de Spring.

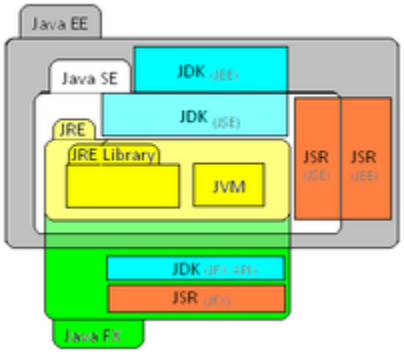


INTRODUCCIÓN A SPRING FRAMEWORK

7.@Service: se utiliza para marcar una clase como un servicio en la arquitectura de la aplicación. **Un servicio es una clase que encapsula la lógica de negocio de la aplicación.** Por lo general, un servicio se utiliza para procesar la entrada del usuario, realizar validaciones y llamar a otros componentes de la aplicación, como los repositorios. Un servicio es uno de los componentes de la capa de negocio de la arquitectura de la aplicación.

8.@Repository: se utiliza para marcar una clase como un repositorio de datos en la aplicación. Un repositorio es una clase que se utiliza para interactuar con una fuente de datos, como una base de datos. Un repositorio normalmente proporciona métodos para recuperar, actualizar y eliminar datos.

9.@Transactional se utiliza para marcar un método de servicio o repositorio como transaccional. Una transacción es una serie de operaciones que se realizan como una sola unidad. Si una de las operaciones falla, la transacción se deshace y todas las operaciones anteriores se revierten. @Transactional asegura que todas las operaciones se ejecuten dentro de una transacción y se deshagan si alguna de ellas falla.

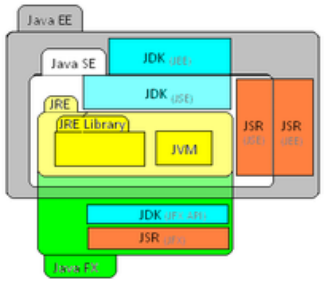


INTRODUCCIÓN A SPRING FRAMEWORK

Paquetes comunes en un proyecto de Spring BOOT

com.example.app.config: este paquete contiene las clases de configuración de Spring Boot que definen la configuración de la aplicación, como la configuración de la base de datos, la configuración de seguridad, la configuración del servidor, etc.

com.example.app.controller: este paquete contiene las clases de controlador que manejan las solicitudes HTTP entrantes y envían las respuestas HTTP correspondientes.



com.example.app.model: este paquete contiene las clases de modelo que representan los objetos de dominio de la aplicación, como los usuarios, las publicaciones, las entradas de blog, etc.

com.example.app.repository: este paquete contiene las interfaces de repositorio de Spring Data JPA que proporcionan métodos para interactuar con la base de datos.

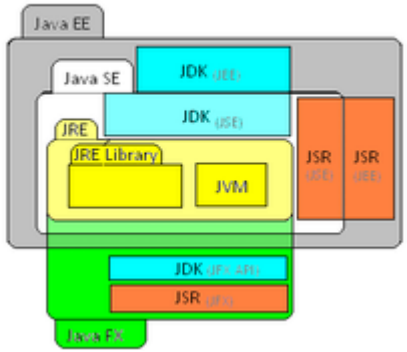
INTRODUCCIÓN A SPRING FRAMEWORK

Paquetes comunes en un proyecto de Spring BOOT

com.example.app.service: este paquete contiene las clases de servicio que contienen la lógica de negocio de la aplicación y se comunican con los repositorios para realizar operaciones de lectura y escritura en la base de datos.

com.example.app.exception: este paquete contiene las clases de excepción personalizadas que se utilizan para manejar errores específicos de la aplicación.

com.example.app.security: este paquete contiene las clases de seguridad que se utilizan para asegurar la aplicación, como el filtro de autenticación, el proveedor de autenticación, etc.



MÉTODOS HTTP

Las operaciones HTTP estándar, también conocidas como métodos HTTP, son las siguientes.

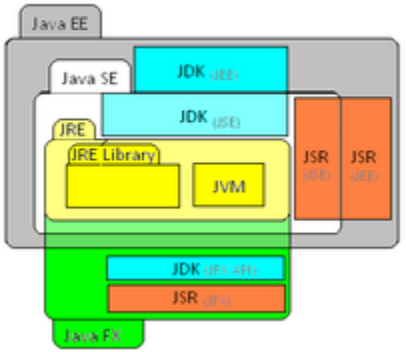
•GET: Solicita una representación de un recurso específico. La solicitud no debe tener efectos secundarios en el servidor.

•POST: Envía datos para que un recurso sea creado en el servidor.

- PUT: Envía datos para actualizar un recurso específico en el servidor.
- DELETE: Elimina un recurso específico en el servidor.
- HEAD: Solicita una respuesta que contenga solo los encabezados de respuesta, sin el cuerpo de la respuesta.
- OPTIONS: Solicita información sobre las opciones de comunicación disponibles para un recurso específico.

•PATCH: Aplica modificaciones parciales a un recurso.

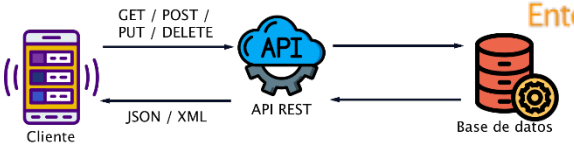
Java™
Enterprise Edition



INTRODUCCIÓN A SPRING FRAMEWORK

Java Enterprise Edition

ARQUITECTURA REST



La arquitectura REST (Representational State Transfer) es un conjunto de principios y restricciones que se aplican a la creación de servicios web que utilizan el protocolo HTTP (Hypertext Transfer Protocol).

La arquitectura REST se basa en la idea de que un servicio web debe proporcionar acceso a recursos a través de un conjunto de operaciones HTTP estándar, como **GET, POST, PUT, DELETE**, etc. Los recursos son identificados mediante URIs (Uniform Resource Identifiers), y el intercambio de información entre el cliente y el servidor se realiza en formato JSON o XML.

Las principales características de la arquitectura REST son:

•Separación entre cliente y servidor: el servidor no guarda ningún estado sobre el cliente entre peticiones, lo que hace que el cliente sea independiente del servidor y permita una mayor escalabilidad.

•Operaciones estándar: los servicios REST utilizan las operaciones HTTP estándar, lo que hace que sean fáciles de implementar y entender.

•los recursos son identificados mediante una URI, que permite al cliente Recursos identificables: acceder a ellos de manera sencilla.

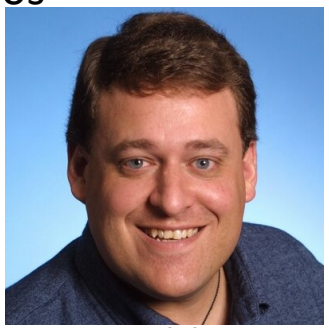
•Representación de recursos: los recursos son representados en un formato estándar, como XML o JSON, lo que permite su fácil interpretación por parte del cliente.

API REST

API Rest o API de Restfull, significa "**Application Programming Interface Representational State Transfer**" y es un conjunto de principios y convenciones para el diseño y el acceso a servicios web.

En términos simples, una API Rest es un protocolo que permite que dos sistemas se comuniquen entre sí a través de Internet para intercambiar información. La API Rest utiliza un conjunto de operaciones HTTP estándar (GET, POST, PUT, DELETE, etc.) para enviar y recibir datos en formato JSON o XML.

Una característica fundamental de las API Rest es que son sin estado, lo que significa que cada solicitud que se hace al servidor es independiente de las demás solicitudes. Además, las API Rest son escalables y permiten a los desarrolladores crear aplicaciones web y móviles altamente flexibles e interconectadas.



Roy Fielding

ARQUITECTURA REST

La arquitectura REST (Representational State Transfer) es un conjunto de principios y restricciones que se aplican a la creación de servicios web que utilizan el protocolo HTTP (Hypertext Transfer Protocol).

La arquitectura REST se basa en la idea de que un servicio web debe proporcionar acceso a recursos a través de un conjunto de operaciones HTTP estándar, como **GET, POST, PUT, DELETE**, etc. Los recursos son identificados mediante URIs (Uniform Resource Identifiers), y el intercambio de información entre el cliente y el servidor se realiza en formato JSON o XM

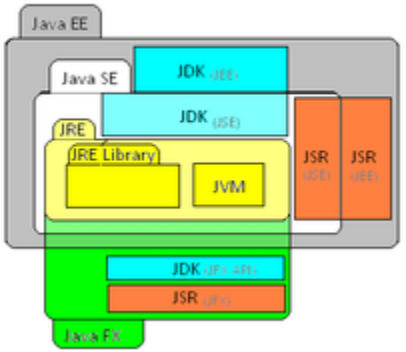
Las principales características de la arquitectura REST son:

•Separación entre cliente y servidor: el servidor no guarda ningún es sobre el cliente entre peticiones, lo que hace que el cliente independiente del servidor y permita una mayor escalabilidad.

•Operaciones estándar: los servicios REST utilizan las operaciones I estándar, lo que hace que sean fáciles de implementar y entender.

•los recursos son identificados mediante una URI, que permite al cliente Recursos identificables: acceder a ellos de manera sencilla.

•Representación de recursos: los recursos son representados en un formato estándar, como XML o JSON, lo que permite su fácil interpretación por parte del cliente.



INTRODUCCIÓN A SPRING FRAMEWORK

Request

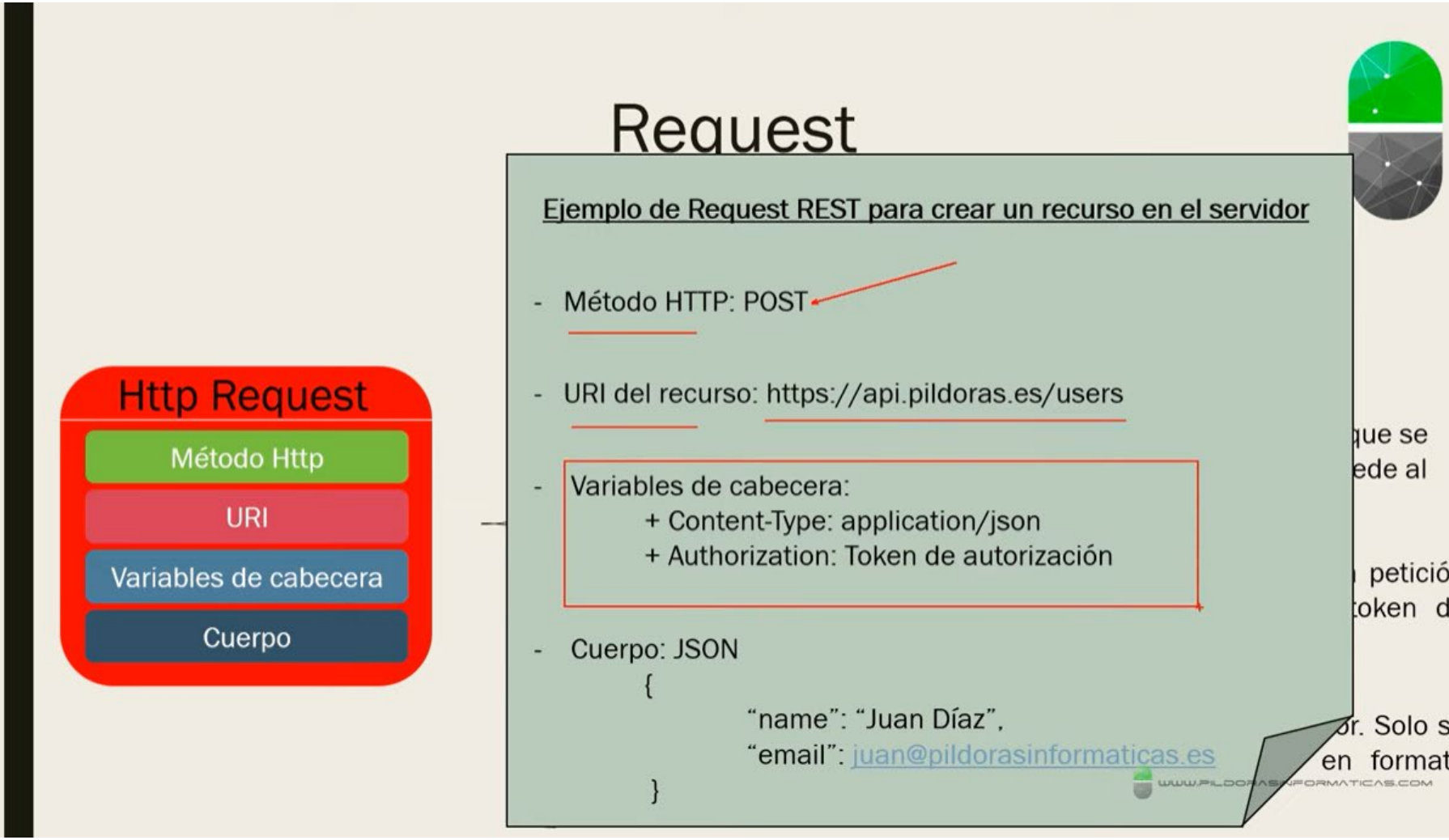
Método Http: el comando Http
- POST
- GET
- PUT
- DELETE

URI: Identifica de manera única el recurso en el Servidor con el que se quiere interactuar. Ej: https://api.pildoras.es/user/5485, se accede al recurso de usuario con ID 5485 en el servidor "api.pildoras.es"

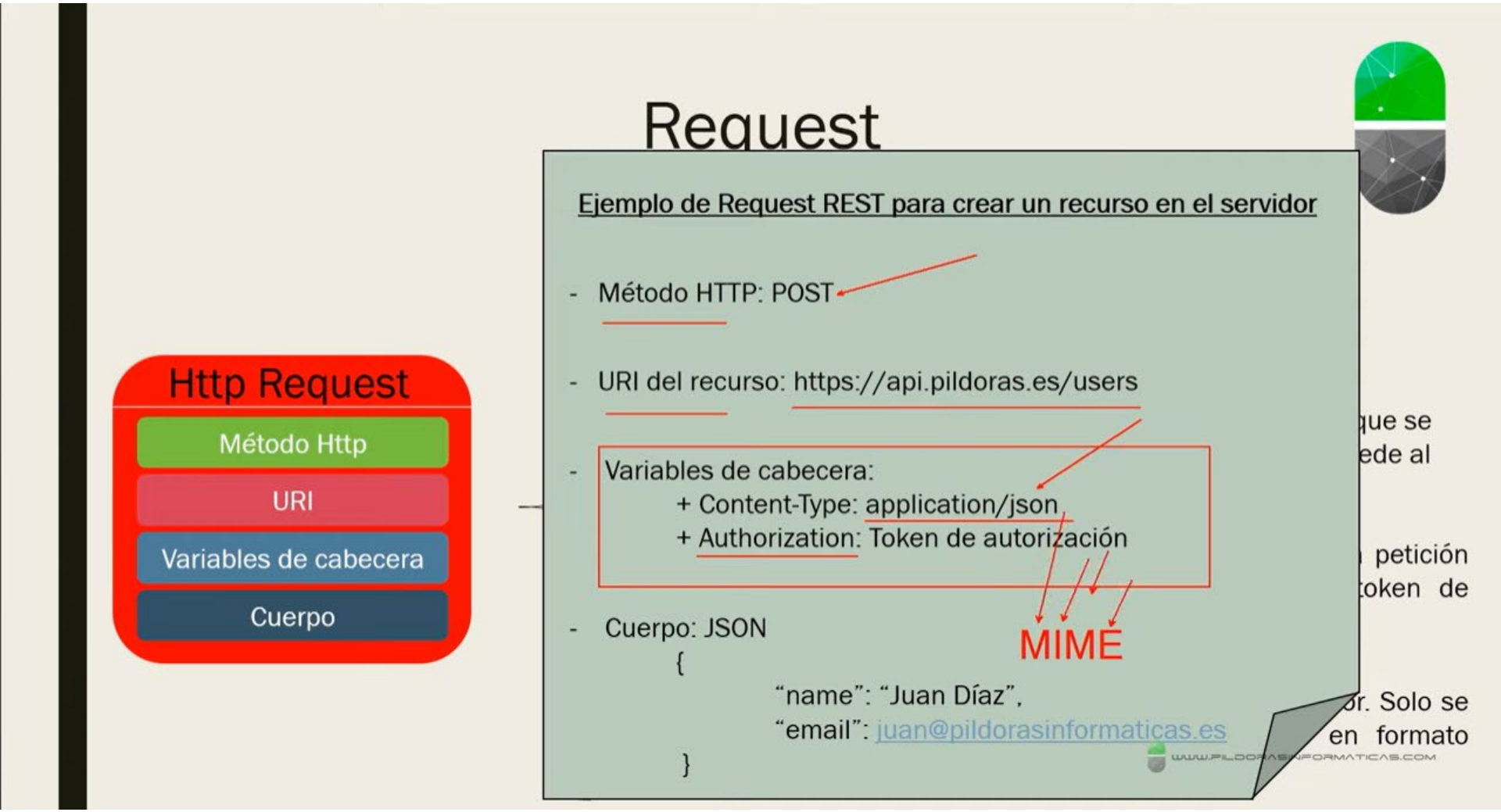
Variables de cabecera: Metadatos: información adicional de la petición como por ejemplo tipo de contenido de datos enviados, token de autorizaciones etc

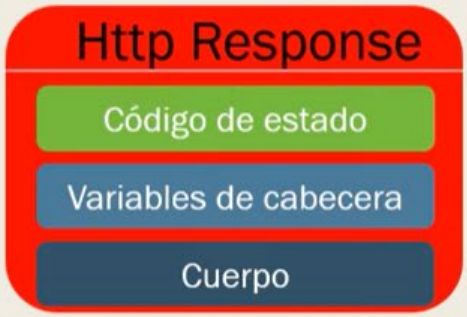
Cuerpo (Opcional): Contiene los datos que se envían al servidor. Solo se incluye con POST, PUT y PATCH. Suele contener datos en formato JSON/XML

INTRODUCCIÓN A SPRING FRAMEWORK



INTRODUCCIÓN A SPRING FRAMEWORK

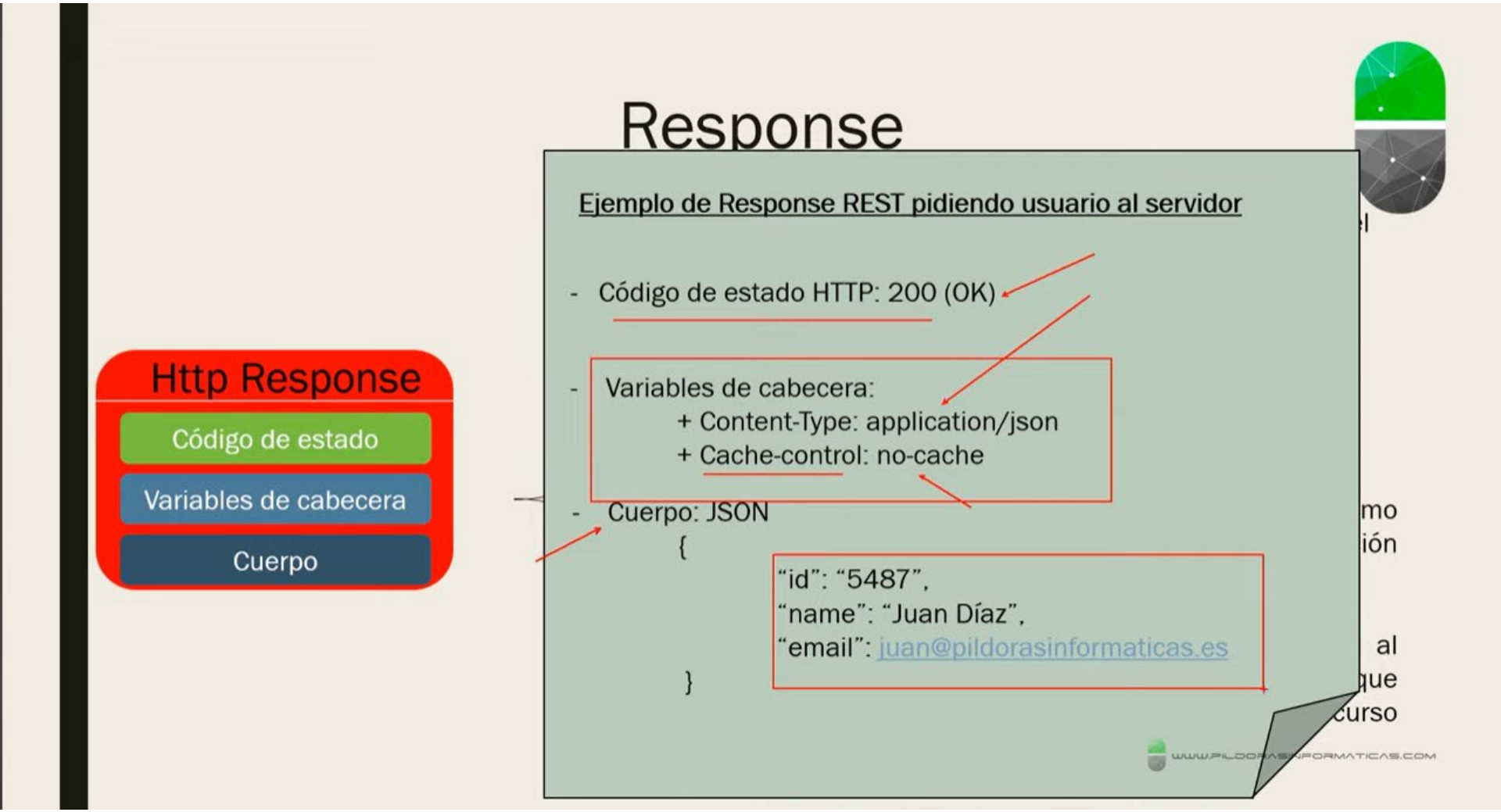




Response

- Código de estado: código numérico de 3 dígitos que indica el resultado de la solicitud. Algunos códigos:
  - 200 (OK)
  - 201 (Created)
  - 400 (Bad Request)
  - 401 (Unauthorized)
  - 404 (Not Found)
  - 500 (Internal Server Error)
- Variables de cabecera: Metadatos. Información adicional como tipo de contenido (Content-Type), información de caché, duración de la sesión etc.
- Cuerpo (Opcional): Datos que el servidor envía de vuelta al cliente. Solo se incluye cuando hay información relevante que enviar: mensaje de error detallado, representación del recurso solicitado etc. En formato JSON/XML

INTRODUCCIÓN A SPRING FRAMEWORK



REST

(Representational State Transfer)

- ¿Qué es?
  - *REST es un estilo de arquitectura de software para diseñar servicios web. Fue introducido por Roy Fielding en su tesis doctoral en 2000. La idea de REST es permitir que los sistemas interactúen de forma simple y estandarizada a través de la red, como lo hacen las aplicaciones web.*
- REST se basa en un conjunto de principios, también conocidos como los principios de RESTful, que incluyen la utilización de URLs para identificar recursos, el uso de instrucciones HTTP (como GET, POST, PUT y DELETE) para indicar la acción a realizar en un recurso, y el uso de un formato estándar para intercambiar información entre aplicaciones cliente y servidor, como JSON o XML.
- En resumen, REST es una arquitectura de software para diseñar servicios web que se basa en la simplicidad, la estandarización y la escalabilidad. Es comunmente utilizado en el diseño de APIs REST para permitir a los desarrolladores acceder a recursos en un servidor de una forma estandarizada y sencilla.

Roy Fielding

Uno de los padres de la especificación Http. Mesías de la arquitectura de redes

Conceptos sobre APIs REST

11-04-2013

En esta entrada voy a resumir los conceptos más importantes que he tratado en mis ponencias sobre REST.

¿Qué es REST?

REST, REpresentational State Transfer, es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP.

REST nos permite crear servicios y aplicaciones que pueden ser usadas por cualquier dispositivo o cliente que entienda HTTP, por lo que es increíblemente más simple y convencional que otras alternativas que se han usado en los últimos diez años como SOAP y XML-RPC.

REST se definió en el 2000 por Roy Fielding, coautor principal también de la especificación HTTP. Podríamos considerar REST como un framework para construir aplicaciones web respetando HTTP.

Por lo tanto REST es el tipo de arquitectura más natural y estándar para crear APIs para servicios orientados a Internet.

Existen tres niveles de calidad a la hora de aplicar REST en el desarrollo de una aplicación web y más concretamente una API que se recogen en un modelo llamado **Richardson Maturity Model** en honor al tipo que lo estableció, Leonard Richardson padre de la arquitectura orientada a recursos. Estos niveles son:

1. Uso correcto de URIs
2. Uso correcto de HTTP.
3. Implementar Hypermedia.

Además de estas tres reglas, **nunca se debe guardar estado en el servidor**, toda la información que se requiere para mostrar la información que se solicita debe estar en la consulta por parte del cliente.

Al no guardar estado, REST nos da mucho juego, ya que podemos escalar mejor sin tener que preocuparnos de temas como el almacenamiento de variables de sesión e incluso, podemos jugar con distintas tecnologías para servir determinadas partes o recursos de una misma API.

Nivel 1: Uso correcto de URIs

Cuando desarrollamos una web o una aplicación web, las URLs nos permiten acceder a cada uno de las páginas, secciones o documentos del sitio web.

Cada página, información en una sección, archivo, cuando hablamos de REST, los nombramos como **recursos**.

El recurso por lo tanto es la información a la que queremos acceder o que queremos modificar o borrar,** independientemente de su formato**.

Las URL, Uniform Resource Locator , son un tipo de URI, Uniform Resource Identifier, que además de permitir **identificar de forma única el recurso**, nos permite localizarlo para poder acceder a él o compartir su ubicación.

Una URL se estructura de la siguiente forma:

```
{protocolo}://{dominio o hostname}[:puerto (opcional)]/{ruta del recurso}?{consulta de fil
```

Por ejemplo, /2013/conceptos-sobre-apis-rest/, sería la URL para visualizar este artículo.

Principios REST

- Stateless (Sin estado): Las interacciones entre el cliente y el servidor en una API REST deben ser independientes y no depender del contexto almacenado en el servidor. Cada petición del cliente debe contener toda la información necesaria para que el servidor la procese y comprenda. Esto mejora la escalabilidad y la simplicidad del servidor al no tener que manejar y almacenar el estado del cliente entre peticiones.
- Client-Server (Cliente-Servidor): REST sigue el modelo cliente-servidor, donde el cliente es responsable de la interfaz de usuario y el servidor es responsable de procesar las peticiones y manejar la lógica empresarial. Esto permite la separación de responsabilidades y facilita la evolución independiente de ambas partes.
- Cacheable (Almacenable en caché): Las respuestas del servidor en una API REST pueden ser marcadas como cacheables o no cacheables. Si una respuesta es cacheable, el cliente puede almacenarla localmente y reutilizarla para peticiones futuras similares, lo que mejora la eficiencia, reduce la latencia y disminuye la carga en el servidor.
- Layered System (Sistema en capas): Una arquitectura REST puede organizarse en capas, con cada capa realizando una función específica. Esto permite la modularidad, la separación de responsabilidades y la mejora de la escalabilidad y la seguridad. Los componentes en una capa no necesitan conocer detalles sobre componentes en otras capas más allá de su interfaz de comunicación.
- Uniform Interface (Interfaz uniforme): REST enfatiza en la consistencia y simplicidad en la comunicación entre el cliente y el servidor a través de una interfaz uniforme. Esto incluye la identificación de recursos a través de URIs (Uniform Resource Identifiers), la manipulación de recursos a través de representaciones (por ejemplo, JSON o XML) y el uso de métodos estandarizados de HTTP (GET, POST, PUT, DELETE, PATCH) para realizar acciones en los recursos.
- Code on Demand (Código bajo demanda) - Opcional: Este principio establece que el servidor puede enviar código ejecutable al cliente, como scripts de JavaScript, para ampliar la funcionalidad del cliente según sea necesario. Aunque este principio es opcional en una arquitectura REST, puede ser útil en ciertas situaciones.

Arquitectura orientada a recursos

De Wikipedia, la enciclopedia libre

En ingeniería de software , una **arquitectura orientada a recursos** (**ROA**) es un estilo de arquitectura de software y paradigma de programación para el diseño y desarrollo de software de apoyo en forma de interconexión de recursos con interfaces " RESTful " . Estos recursos son componentes de software (piezas discretas de código y/o estructuras de datos) que se pueden reutilizar para diferentes propósitos. Los principios y directrices de diseño de ROA se utilizan durante las fases de desarrollo de software.e integración de sistemas .

REST, o Transferencia de estado representacional, describe una serie de restricciones arquitectónicas que ejemplifican cómo surgió el diseño web. [1] Se han creado varias implementaciones concretas de estas ideas a lo largo del tiempo, pero ha sido difícil discutir el estilo arquitectónico REST sin desdibujar las líneas entre el software real y los principios arquitectónicos detrás de él.

En el Capítulo 5 de su tesis, Roy Fielding documenta cómo la World Wide Web está diseñada para estar restringida por la serie de limitaciones REST. Estos todavía son bastante abstractos y se han interpretado de varias maneras en el diseño de nuevos marcos, sistemas y sitios web. En el pasado, se han realizado intercambios acalorados sobre si las arquitecturas REST de estilo RPC son RESTful. [1] [2]

Directrices para la aclaración [editar]

La arquitectura orientada a los recursos, tal como lo documentaron Leonard Richardson y Sam Ruby en su libro de 2007 *RESTful Web Services* , [3] brinda consejos concretos sobre detalles técnicos específicos. Nombrar estas colecciones de pautas como "Arquitectura Orientada a Recursos" puede permitir a los desarrolladores discutir los beneficios de una arquitectura en el contexto de ROA.

Algunas pautas ya son comunes dentro de las comunidades REST más grandes, como: que una aplicación debe exponer muchos URI, uno para cada recurso; y que el procesamiento de cookies que representan ID en una sesión del lado del servidor no es RESTful.

Marcos existentes [editar]

Richardson y Ruby también analizan muchos marcos de software que brindan algunas o muchas características del ROA. Estos incluyen /db, [4] Django , TurboGears , Flask , EverRest, [5] JBoss RESTEasy, [6] JBoss Seam , Spring , [7] Apache Wink, [8] Jersey , NetKernel , Recess, [9] Ruby on Rails , Symfony , Yii2, [10] Play Framework , [11] y API Platform . [12]