



04. Generación de servicios en red

Programación de Servicios y procesos - 2º DAM

Luis del Moral Martínez

versión 20.10

Bajo licencia CC BY-NC-SA 4.0



Contenidos del tema

1. Desarrollo de servicios básicos en red

- 1.1 Introducción a los servicios en red
- 1.2 Protocolos estándar de comunicación en red
- 1.3 Comunicación con servidor FTP
- 1.4 Programación de servidores con Java

2. Desarrollo de servicios RESTful con Spring

- 2.1 Introducción a Spring
- 2.2 Aplicaciones y servicios web
- 2.3 Patrones de diseño útiles
- 2.4 Creando un servicio RESTful con Spring

Contenidos de la sección

1. Generación de servicios en red

- 1.1 Introducción a los servicios en red
- 1.2 Protocolos estándar de comunicación en red
- 1.3 Comunicación con servidor FTP
- 1.4 Comunicación con servidor SMTP
- 1.5 Programación de servidores con Java

1.1 Introducción a los servicios en red

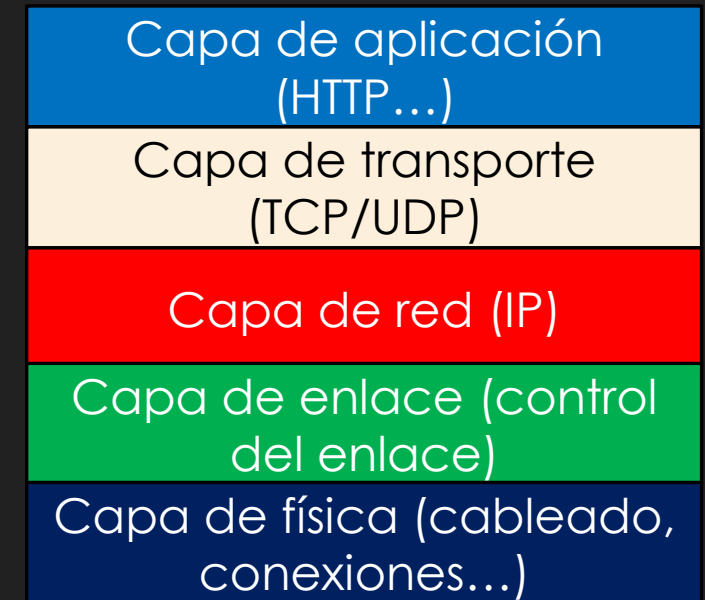
Definición de servicio

- Un servicio es un programa auxiliar que gestionar recursos
- El servicio presta funcionalidades a otras aplicaciones o a los usuarios
- El único acceso que tenemos a un servicio lo conforman las operaciones que este ofrece
 - Operaciones de lectura/escritura, transferencia, borrado, etcétera
- Los servicios de Internet implementan una relación cliente-servidor

1.2 Protocolos estándar de comunicación en red

Modelo TCP/IP

- Se compone por cinco capas o niveles
- Estos son algunos de los protocolos de la capa de aplicación:
 - **Conexión remota:** Telnet, SSH
 - **Correo electrónico:** SMTP, POP3, IMAP
 - **Acceso a ficheros:** FTP, TFTP, NFS, CIFS
 - **Resolución de nombres:** DNS, WINS
 - **World Wide Web:** HTTP, HTTPS



Capas TCP/IP

1.3 Comunicación con servidor FTP

El protocolo FTP

- Este protocolo es una herramienta útil para intercambiar ficheros entre diferentes equipos
- Uno de los extremos actúa como **servidor** y el otro como **cliente**
- Existen dos formas de acceso en FTP
 - **Acceso anónimo**: la conexión con el servidor la realiza un usuario sin autenticar
 - **Acceso autorizado**: el usuario que realiza la conexión está registrado y tiene privilegios en el servidor
- Normalmente el servidor se conecta a los puertos **21** (control port) y **20** (transfer port)

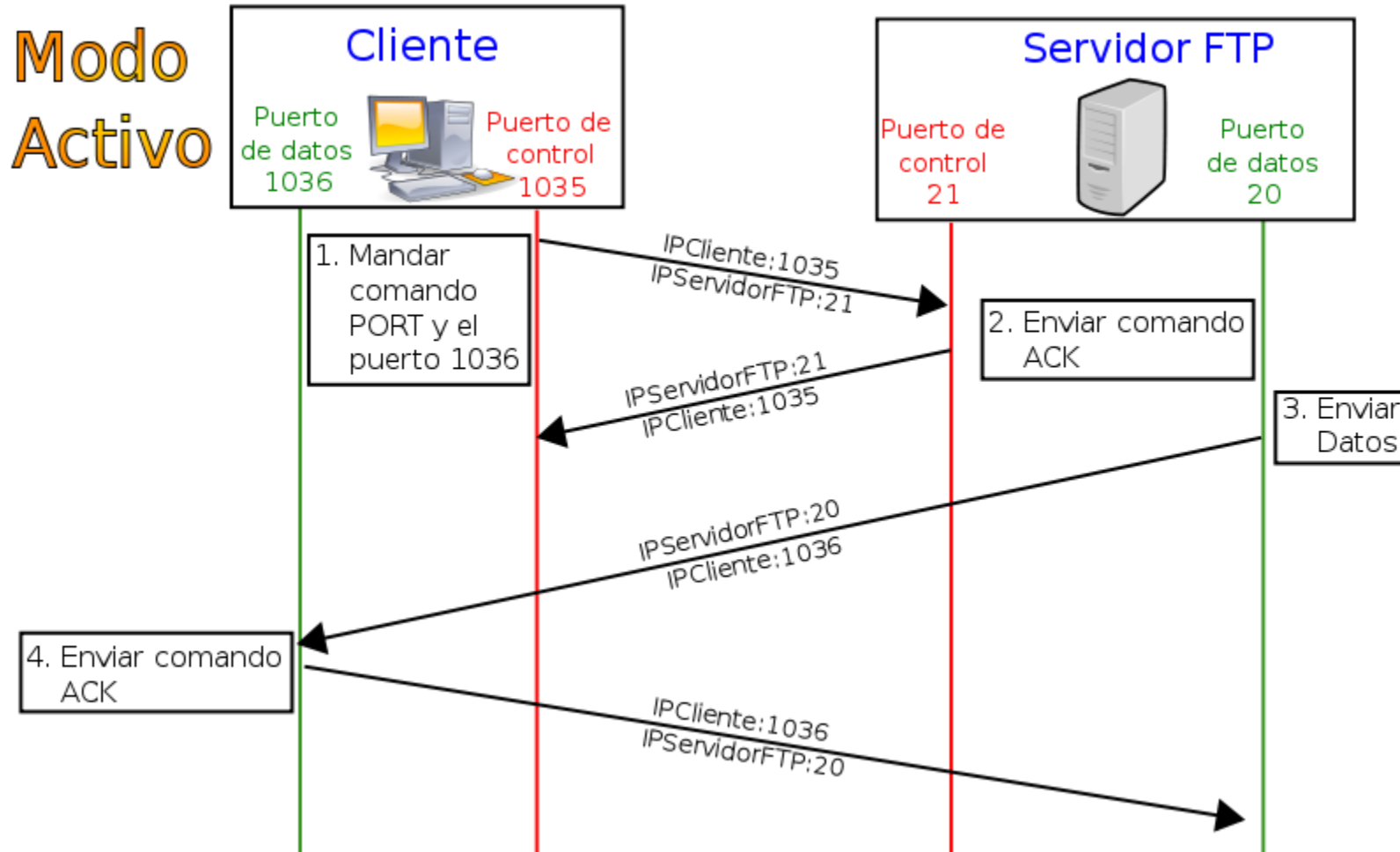
1.3 Comunicación con servidor FTP

El protocolo FTP

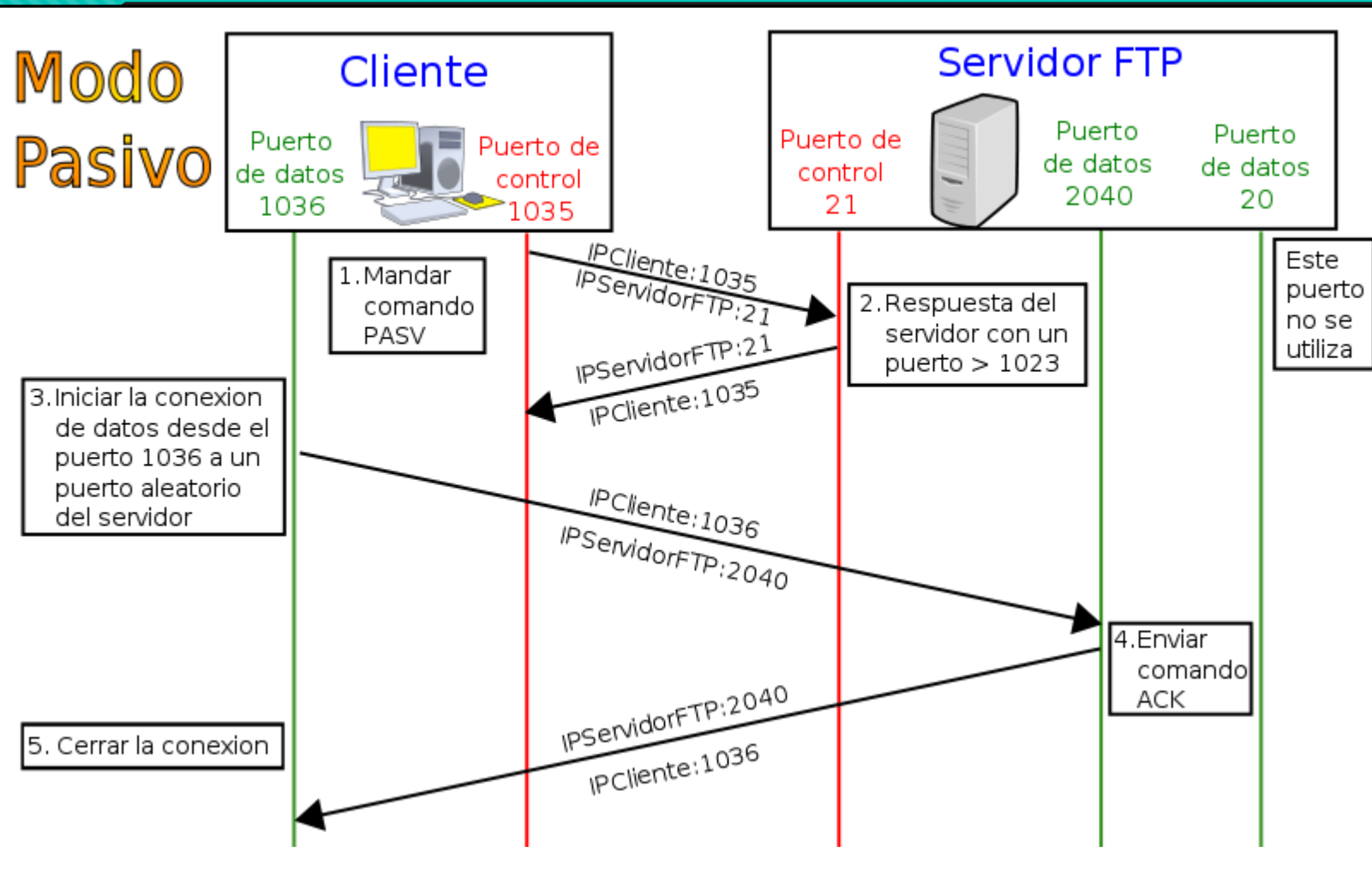
- El cliente, por su parte, puede actuar de modo activo o pasivo
 - **Modo activo (PORT)**
 - El servidor crea el canal de datos en su puerto 20
 - En el cliente se crea un puerto aleatorio
 - **Modo pasivo (PASV)**
 - El servidor indica al cliente el puerto a usar por medio del canal de control
 - El puerto cliente debe ser mayor que 1024 (los 1024 primeros están reservados)

1.3 Comunicación con servidor FTP

Modo Activo



1.3 Comunicación con servidor FTP



1.3 Comunicación con servidor FTP

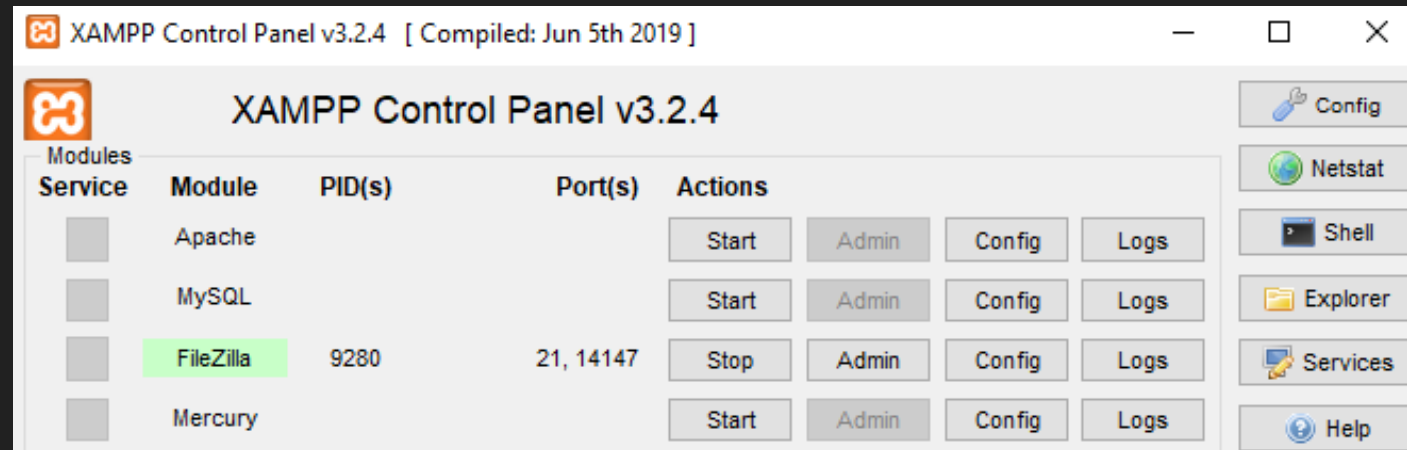
Comunicar una aplicación Java con un servidor FTP

- Usaremos **Apache Commons Net™ 3.6** y el servidor FTP **Filezilla Server**
- Esta librería permite implementar el lado cliente de muchos protocolos de Internet
- Intervienen las clases **FTPClient**, **FTPReply** y **FTPFile**: [enlace](#), [enlace](#), [enlace](#)
- Veamos algunos ejemplos de uso
 - **Nota**: el proyecto está creado en Maven (para facilitar la instalación de dependencias)
 - En **FileZilla** crearemos unas credenciales (usuario/usuario) y una carpeta para probar el FTP

1.3 Comunicación con servidor FTP

Configuración del servidor FTP

- Levantamos el servicio **FileZilla server** en el **panel de control de XAMPP**



1.3 Comunicación con servidor FTP

Configuración del servidor FTP

- En la opción **Admin**, configuramos el usuario, la carpeta del FTP y los permisos

The 'Users' dialog box is shown with the 'General' tab selected. The 'Page:' dropdown is set to 'General'. The 'Account settings' section includes checkboxes for 'Enable account' and 'Password', both of which are checked. The 'Password' field is masked with dots. Below this is a 'Group membership' dropdown menu set to '<none>'. There is a checkbox for 'Bypass userlimit of server' which is unchecked. Below it are two input fields: 'Maximum connection count' and 'Connection limit per IP', both set to '0'. Another checkbox for 'Force SSL for user login' is also unchecked. A 'Description' text area is at the bottom with the placeholder text 'You can enter some comments about the user'. On the right side, there is a list box containing the name 'usuario'. Below the list box are four buttons: 'Add', 'Remove', 'Rename', and 'Copy'. At the bottom left are 'OK' and 'Cancel' buttons.

The 'Users' dialog box is shown with the 'Shared folders' tab selected. The 'Page:' dropdown is set to 'Shared folders'. The 'Directories' list box contains the entry 'H C:\Users\Luis\De...'. Below the list box are four buttons: 'Add', 'Remove', 'Rename', and 'Set as home dir'. On the right side, there are two sections of permissions. The 'Files' section has checkboxes for 'Read', 'Write', 'Delete', and 'Append', all of which are checked. The 'Directories' section has checkboxes for 'Create', 'Delete', 'List', and '+ Subdirs', all of which are checked. Below these sections are 'Add' and 'Remove' buttons. At the bottom left are 'OK' and 'Cancel' buttons. A note at the bottom states: 'A directory alias will also appear at the specified location. Aliases must contain the full local path. Separate multiple aliases for one directory with the pipe character (|). If using aliases, please avoid cyclic directory structures, it will only confuse FTP clients.'

1.4 Programación de servidores con Java

Servidor de ficheros de ejemplo

- A continuación analizaremos un ejemplo completo de un servidor de ficheros
- **Proceso de la aplicación**
 1. Cuando se inicia el programa servidor se elige la carpeta o directorio que se ofrecerá a los clientes
 2. Los clientes se conectarán al servidor y sólo podrán cargar o descargar ficheros
 3. El cliente solicita la descarga o la carga de un fichero y el servidor atiende la petición
 4. El servidor responde a la petición y el cliente recibe o envía el fichero

Contenidos de la sección

2. Desarrollo de servicios RESTful con Spring

- 2.1 Introducción a Spring
- 2.2 Aplicaciones y servicios web
- 2.3 Patrones de diseño útiles
- 2.4 Creando un servicio RESTful con Spring

2.1 Introducción a Spring

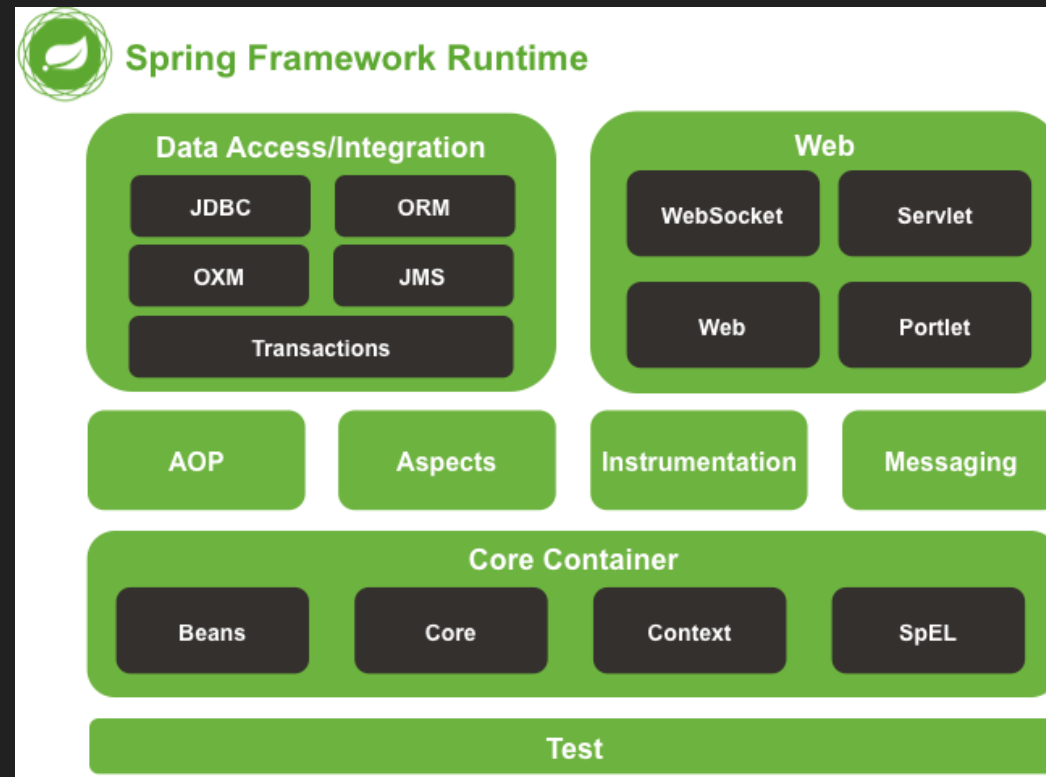
¿Qué es Spring?

- En esta sección aprenderemos a desarrollar un servicio RESTful
- Pero antes, aprenderemos conceptos básicos sobre Spring
- **Spring** es un **framework** para crear aplicaciones empresariales Java
- Spring es de código abierto
- Spring es modular y tiene una estructura flexible
- **Página web de Spring:** [enlace](#)



2.1 Introducción a Spring

¿Qué es Spring?

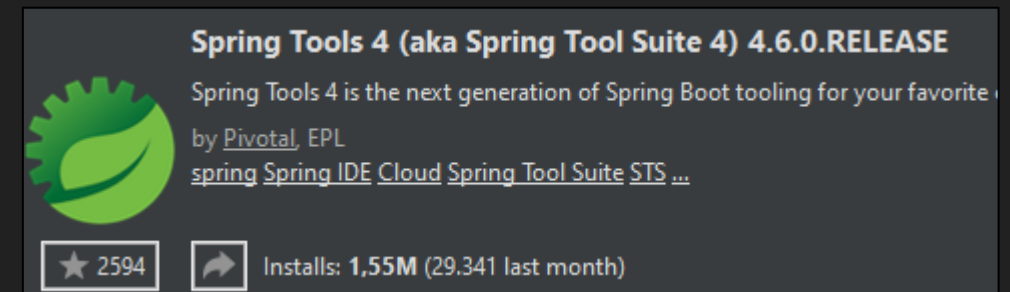


Fuente: spring.io

2.1 Introducción a Spring

¿Qué es Spring Boot?

- Permite crear aplicaciones Spring autocontenidas e independientes
- Incorpora un **servidor web** embebido (ejemplo: Apache Tomcat)
- La gestión de **dependencias** facilita el uso y la importación de componentes
- Configuración de librerías automática desde el fichero pom.xml
- Requisitos
 - Java8+
 - Instalar Spring Tools 4 en Eclipse Marketplace



2.1 Introducción a Spring

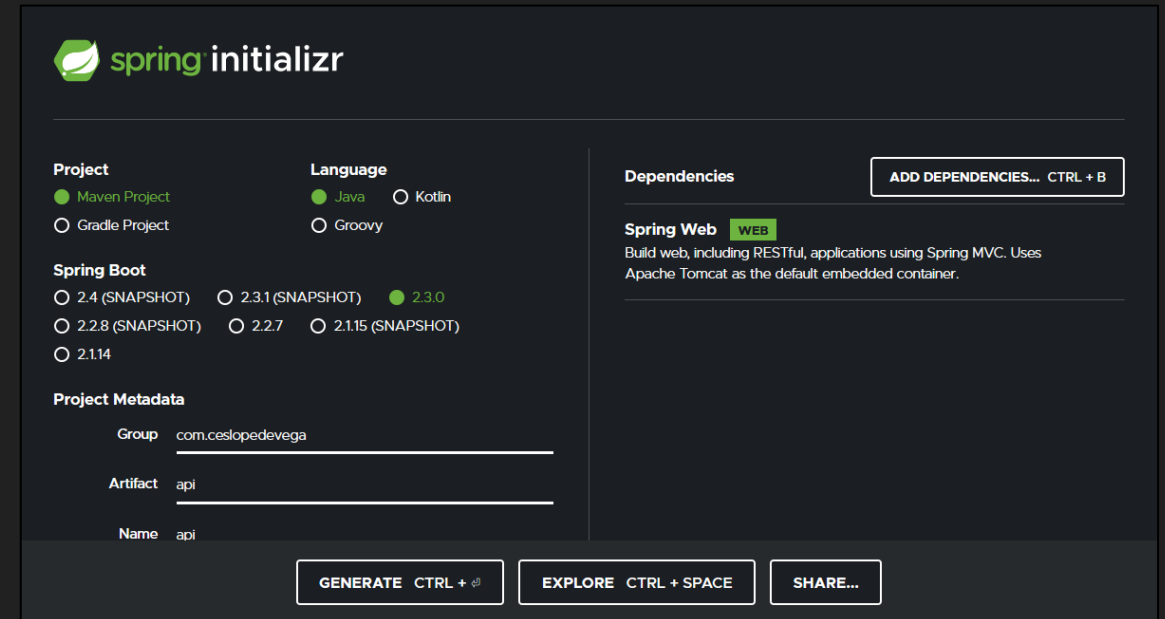
Gestión de dependencias con Maven

- Maven es un software que permite gestionar las dependencias y construir nuestro proyecto
- Se encarga de:
 - Gestionar la **Convención de la Configuración (CoC)** del proyecto
 - Gestionar las dependencias (a través del fichero **pom.xml**)
 - Gestionar el **Contenedor de Inversión de Control (IoC)** para inyectar dependencias (patrón DI)

2.1 Introducción a Spring

Spring INITIALZR

- Esta herramienta inicializa un proyecto Spring
- Permite añadir las dependencias necesarias
- **Enlace a la aplicación:** [enlace](#)



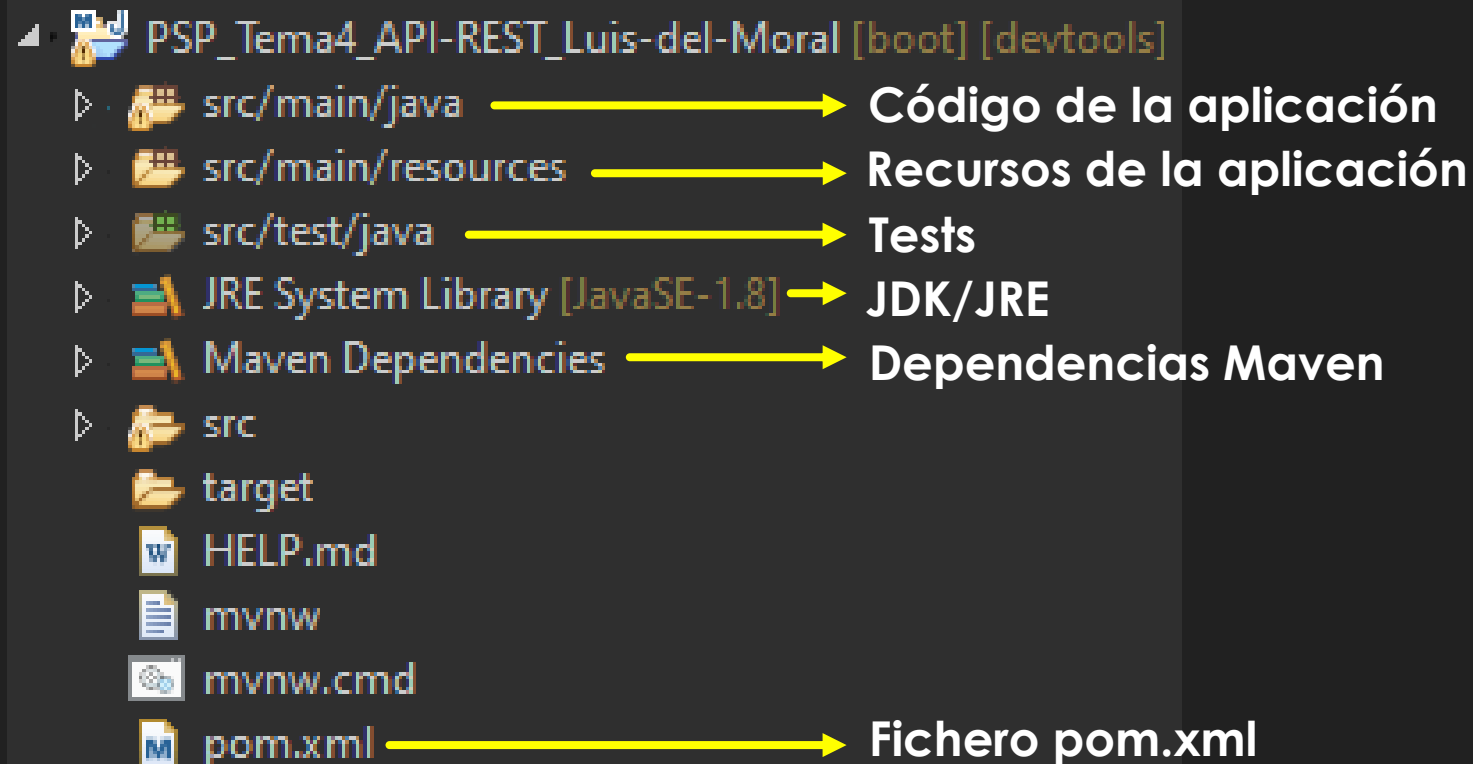
The screenshot shows the Spring Initializr web application interface. It has a dark theme with green accents. The header includes the Spring logo and the text "spring initializr". The main content area is divided into several sections:

- Project:** Radio buttons for "Maven Project" (selected) and "Gradle Project".
- Language:** Radio buttons for "Java" (selected), "Kotlin", and "Groovy".
- Spring Boot:** Radio buttons for various versions: "2.4 (SNAPSHOT)", "2.3.1 (SNAPSHOT)", "2.3.0" (selected), "2.2.8 (SNAPSHOT)", "2.2.7", "2.1.15 (SNAPSHOT)", and "2.1.14".
- Project Metadata:** Input fields for "Group" (com.ceslopedevega), "Artifact" (api), and "Name" (api).
- Dependencies:** A section with a button "ADD DEPENDENCIES... CTRL + B". Below it, "Spring Web" is selected with a "WEB" tag, and a description is provided: "Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container."

At the bottom, there are three buttons: "GENERATE CTRL + G", "EXPLORE CTRL + SPACE", and "SHARE...".

2.1 Introducción a Spring

Estructura de un proyecto Spring Boot



2.1 Introducción a Spring

El fichero pom.xml

- Contiene las dependencias de nuestro proyecto

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xs
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 h
4     <modelVersion>4.0.0</modelVersion>
5     <parent>
6         <groupId>org.springframework.boot</groupId>
7         <artifactId>spring-boot-starter-parent</artifactId>
8         <version>2.3.0.RELEASE</version>
9         <relativePath/> <!-- lookup parent from repository
10    </parent>
11    <groupId>com.ceslopedevega</groupId>
12    <artifactId>API-REST</artifactId>
13    <version>0.0.1-SNAPSHOT</version>
14    <name>API-REST</name>
15    <description>API REST</description>
```

2.1 Introducción a Spring

El fichero principal de la aplicación

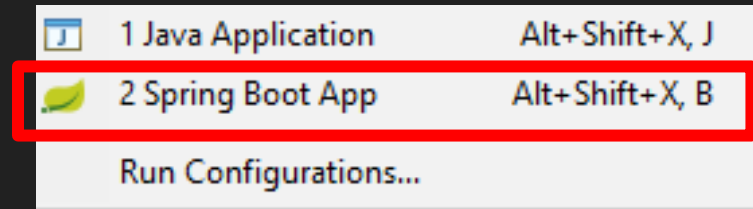
```
1 package com.ceslopedevega.api;
2
3+ import org.springframework.boot.SpringApplication;
4
5
6 @SpringBootApplication —————> Anotación que configura la aplicación
7 public class ApiApplication {
8
9-     public static void main(String[] args) {
10         SpringApplication.run(ApiApplication.class, args);
11     }
12
13 }
```

—————> Inicia la aplicación, lanza Tomcat y publica la aplicación

2.1 Introducción a Spring

Arrancando la aplicación

- La aplicación debemos ejecutarla como una aplicación Spring
- Si realizamos algún cambio y guardamos, se reinicia la app

[illegible]

2.2 Aplicaciones y servicios web

Creando aplicaciones web con Spring Boot

- En la sección anterior abordamos el desarrollo de servicios básicos
- Pero, ¿qué es una **aplicación web**?
 - Es una herramienta accesible a través de la web
 - No suele necesitar de grandes requisitos hardware
 - Se despliega en un servidor y se actualiza de forma más simple (vs. una aplicación de escritorio)
 - Es multiplataforma y multidispositivo, y sólo se precisa un navegador web
 - Su principal inconveniente es la necesidad de conectividad estable y constante con Internet

2.2 Aplicaciones y servicios web

Aplicaciones web basadas en el protocolo HTTP

- El protocolo **HTTP** es sin estados
- Funcionan bajo un esquema de **Petición-Respuesta**



2.2 Aplicaciones y servicios web

Acceso a una aplicación web

- Se realiza mediante una **URL** (Uniform Resource Locator)
- El recurso se suele denominar endpoint o punto final
- **Ejemplo**
 - <https://www.google.es>
 - <https://10.0.25.1/index.php>

2.2 Aplicaciones y servicios web

Verbos HTTP

- Indican la operación que queremos realizar con el recurso
 - **GET**: solicita al servidor el recurso especificado en la **URL**
 - **POST**: envía datos al servidor para un recurso determinado
 - **PUT**: envía un recurso determinado al servidor (similar a **PUT**)
 - **DELETE**: solicita al servidor la eliminación de un recurso
 - **HEAD**: igual que **GET**, pero solo devuelve encabezados de respuesta

2.2 Aplicaciones y servicios web

Verbos HTTP

- Indican la operación que queremos realizar con el recurso
 - **PATCH**: modifica parcialmente un recurso
 - **OPTIONS**: solicita al servidor los métodos **HTTP** para una **URL** concreta
 - **TRACE**: solicita un mensaje de respuesta
 - Se utiliza para diagnosticar problemas de conexión

2.2 Aplicaciones y servicios web

Códigos de estado HTTP

- **1xx – información**
 - Procesando la petición
- **2xx – respuesta correcta**
 - Petición procesada OK (200, 201...)
- **3xx – redirección**
 - El cliente debe realizar más acciones
 - Cuando las realice finaliza la petición
- **4xx – errores (causados por el cliente)**
 - Error procesando la petición
 - Información o formatos incorrectos...
- **5xx – errores (causados por el servidor)**
 - Se ha producido un fallo al procesar la petición
 - El error radica en el servidor

2.2 Aplicaciones y servicios web

Servicio web

- Un servicio web es un sistema software diseñado para la interacción máquina-máquina
- Dispone de una interfaz escrita en formato accesible por un equipo informático
- Permite interactuar con el servicio intercambiando mensajes (XML, JSON)
- El servicio se programa una vez y se consume en cualquier dispositivo
- El servicio es independiente de la plataforma

2.2 Aplicaciones y servicios web

Servicios SOAP (Simple Object Access Protocol)

- Es un protocolo estándar
- Define cómo se pueden comunicar dos objetos entre sí intercambiando XML
- Es más robusto que **REST** y mucho más tipado
- WSDL describe la interfaz pública a los servicios Web (SOAP)
- Útil en aplicaciones empresariales donde la seguridad de los datos transmitidos es crítica

2.2 Aplicaciones y servicios web

Servicios REST (REpresentational State Transfer)

- Es más flexible que SOAP
- Transporta datos por medio de **HTTP** usando sus verbos y los códigos de respuesta estudiados
- El tipo de datos está definido por el **Header Content-Type** (permite enviar XML o JSON)
- **JSON** es interpretado por JavaScript directamente
- Podemos apuntar un formulario HTML a un **servicio REST** directamente

2.3 Patrones de diseño útiles

Los patrones IoC (Inversión de Control) y DI (Inyección de Dependencias)

- La **inversión de control** (IoC) es el pilar fundamental de Spring
- Este patrón tiene como objetivo desacoplar el código
- El flujo lo controla el **framework**, y no el programador
- Un claro ejemplo de este hecho es una **interfaz gráfica** (GUI), que se gestiona por eventos

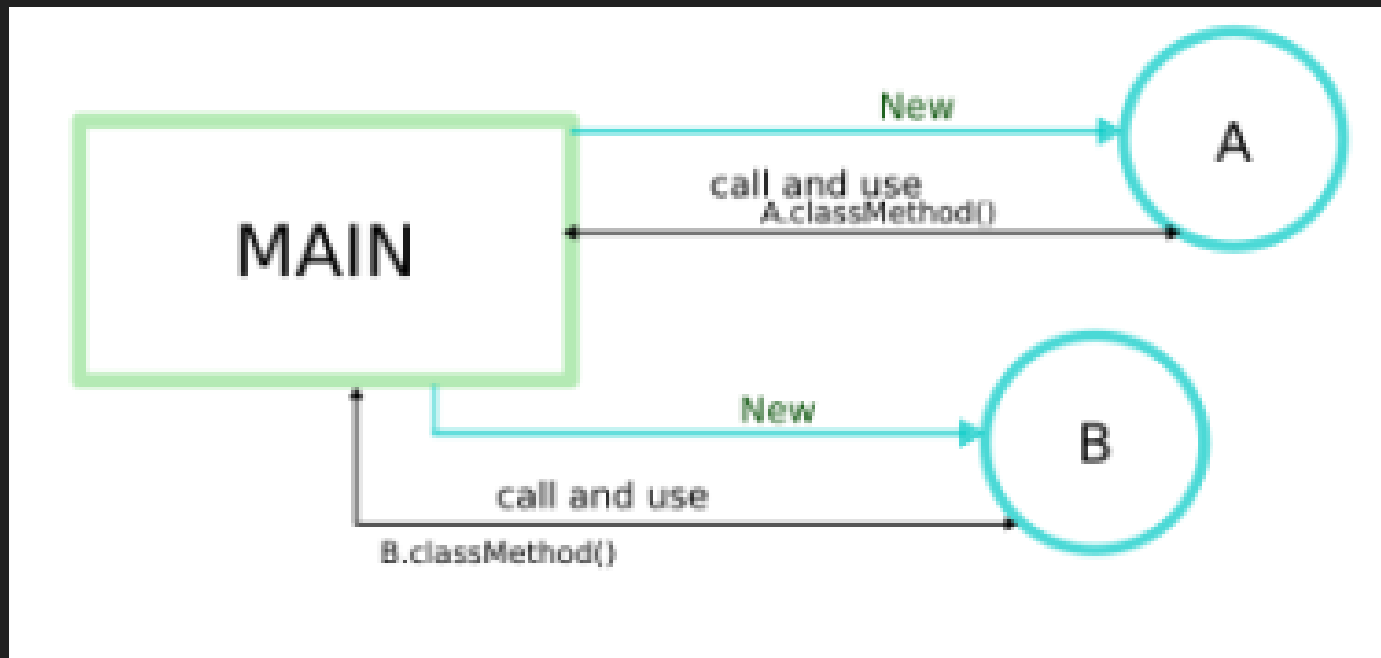
2.3 Patrones de diseño útiles

Los patrones IoC (Inversión de Control) y DI (Inyección de Dependencias)

- El patrón de **inyección de dependencias** es un mecanismo de implementación de IoC
- Este patrón **desacopla** el código y **suministra** objetos a una clase que depende de ellos
- Nuestras clases no crean objetos, sino que los reciben de una clase **contenedor**
- En Spring la clase contenedora se conoce como el **Contenedor de Inversión de Control**
- Los objetos que gestiona el contenedor se conocen como **Beans**
- El contenedor se utiliza por medio de **anotaciones** en el código (@Service, @Autowired...)

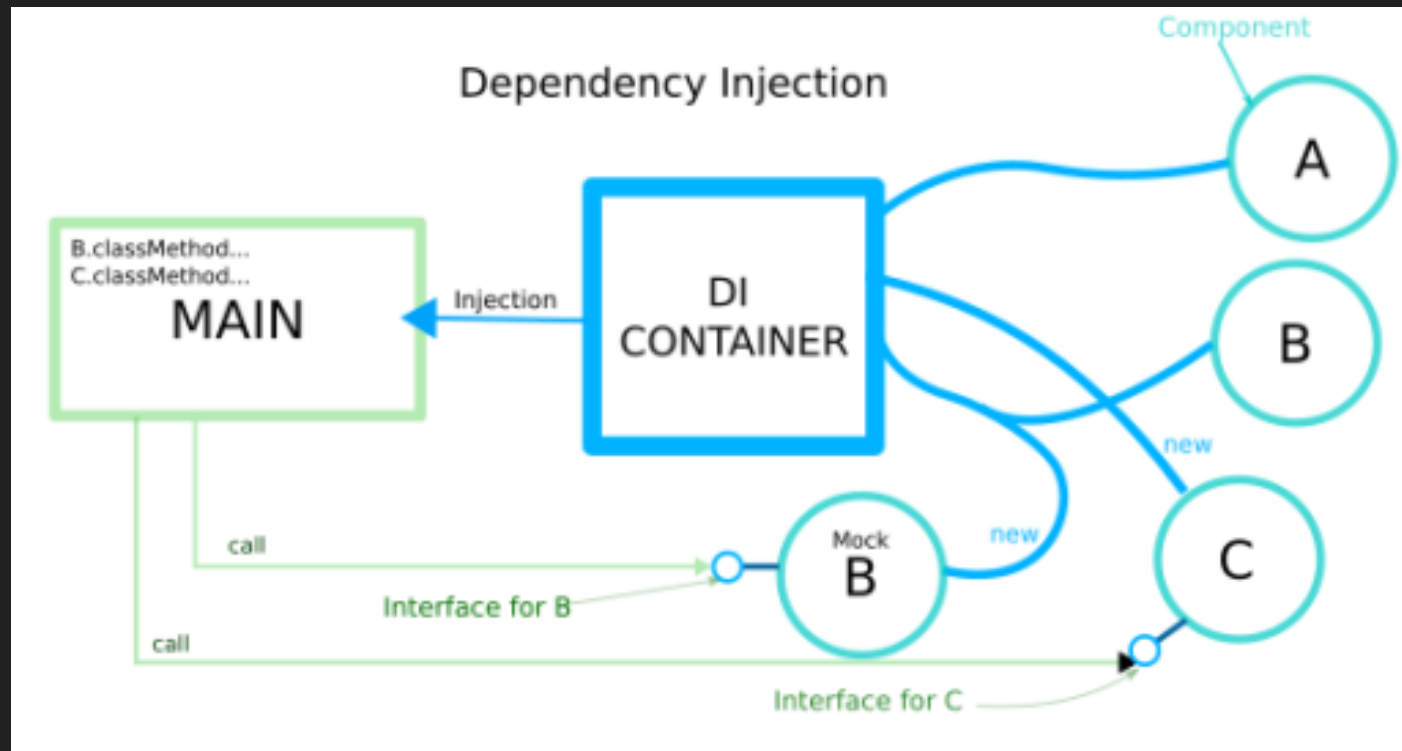
2.3 Patrones de diseño útiles

Los patrones IoC (Inversión de Control) y DI (Inyección de Dependencias)



2.3 Patrones de diseño útiles

Los patrones IoC (Inversión de Control) y DI (Inyección de Dependencias)



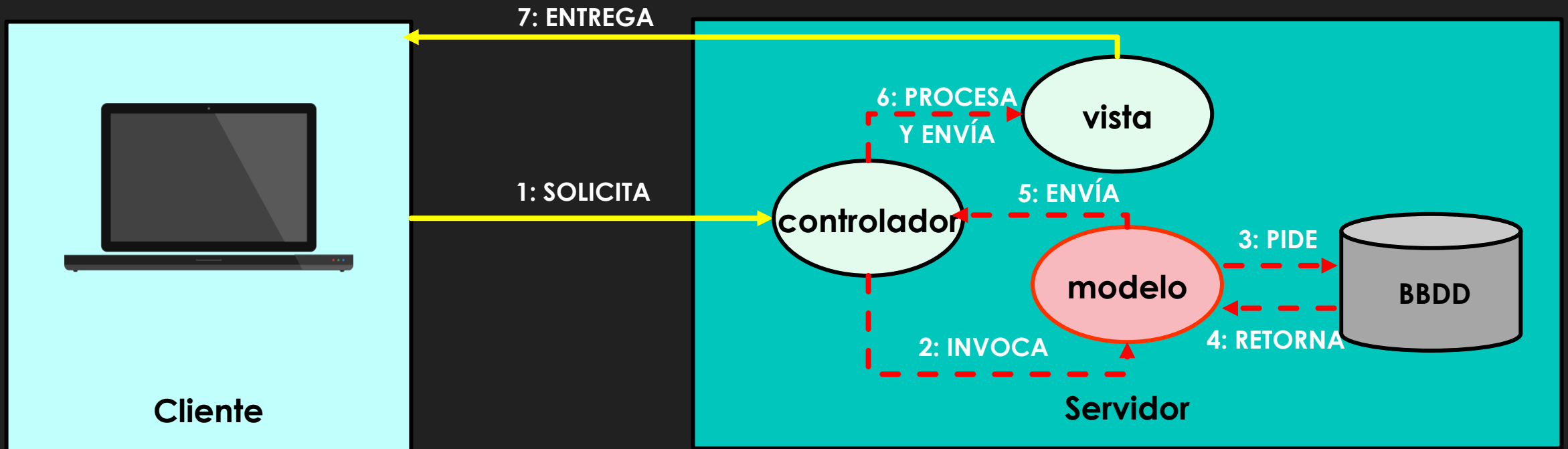
2.3 Patrones de diseño útiles

El patrón de diseño MVC (Modelo-Vista-Controlador)

- Este es un patrón de arquitectura
- Separa la estructura del software en capas:
 - **Base de datos**: datos de la aplicación
 - **Modelo**: lógica de negocio
 - **Vista**: representación gráfica de los datos solicitados por el cliente
 - **Controlador**: permite al servidor gestionar las peticiones y enviar las respuestas al cliente

2.3 Patrones de diseño útiles

El patrón de diseño MVC (Modelo-Vista-Controlador)



2.3 Patrones de diseño útiles

El patrón de diseño MVC (Modelo-Vista-Controlador)

- **Ventajas**

- Alta reutilización del código
- Gran adaptación frente a los cambios
- Soporte para múltiples tipos de vistas

- **Inconvenientes**

- Es más complejo
- Es costoso si hay actualizaciones muy frecuentes

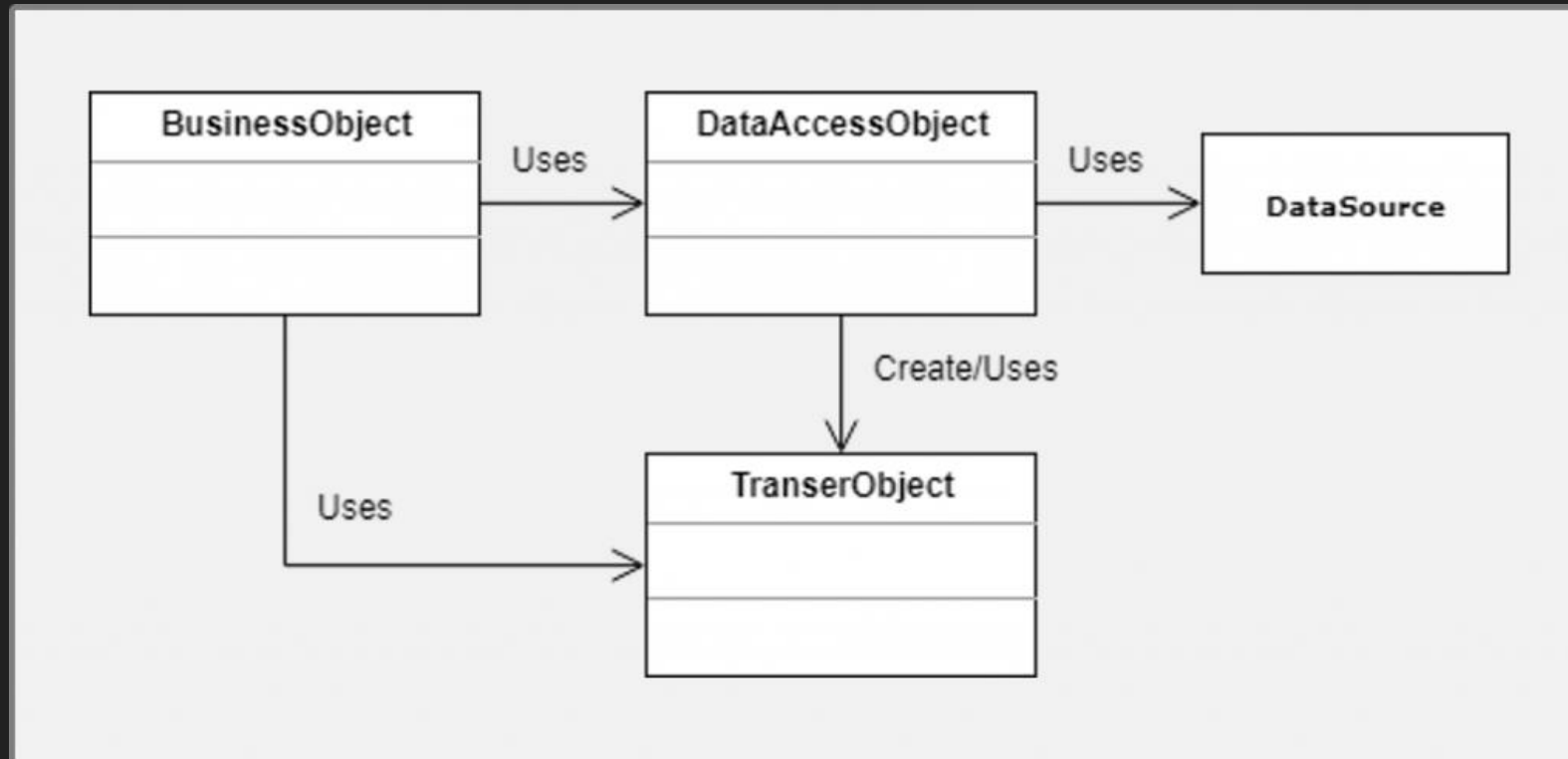
2.3 Patrones de diseño útiles

El patrón de diseño DAO (Acceso directo a objetos)

- Un objeto de acceso a datos permite abstraer el almacén de datos del servicio
- Un objeto de negocio (servicio) no tiene que saber dónde o cómo se almacenan los datos
- Esto permite desacoplar el servicio del almacenamiento de datos

2.3 Patrones de diseño útiles

El patrón de diseño DAO (Acceso directo a objetos)



2.4 Creando un servicio RESTful con Spring

Ejemplo paso a paso

- En este ejemplo desarrollaremos un **API REST** utilizando **Spring**
- La **API REST** permitirá modificar una base de datos relacional según el esquema **CRUD**:
 - **C – Create**: crear un registro en la base de datos
 - **R – Read**: leer un registro de la base de datos
 - **U – Update**: actualizar un registro de la base de datos
 - **D – Delete**: borrar un registro de la base de datos

2.4 Creando un servicio RESTful con Spring

Paso 1. Inicializar el proyecto

- Usaremos la herramienta **Spring initializr** para crear el proyecto
- Abre la URL <https://start.spring.io/>
- Dejamos el resto de opciones por defecto:
 - Proyecto Maven
 - Lenguaje Java
 - Versión 2.3
- Añadimos la dependencia **Spring Web**
- Generamos el proyecto y lo integramos en Eclipse

2.4 Creando un servicio RESTful con Spring

Paso 2. Añadir dependencias al proyecto

- Ahora añadiremos dependencias al proyecto Spring (en el fichero **pom.xml**)
 - Dependencia **JPA** (Java Persistence API)
 - Dependencia **Hibernate** (Mapeo Objeto-Relacional, se basa en JPA)
 - Dependencia para conectarnos a **MySQL**
 - Dependencia para refrescar el servidor
- Actualizar el proyecto **Maven > Update project**

2.4 Creando un servicio RESTful con Spring

Paso 3. Creación de la base de datos

- Ahora cargamos la base de datos
- Usaremos el servidor **XAMPP** para desplegar un servicio MySQL
- Crearemos la base de datos y un usuario con permisos de acceso
- Editaremos el fichero **application.properties** para indicar los parámetros de MySQL

2.4 Creando un servicio RESTful con Spring

Paso 4. Configuración de paquetes del proyecto

- En el proyecto se han configurado los siguientes paquetes
 - **controller**: controlador de la aplicación, gestiona las peticiones de la API
 - **dao**: contiene la clase para el **Data Access Object** (DAO)
 - **entity**: contiene las entidades de la aplicación (utiliza las notaciones JPA)
 - **service**: actúa de intermediario entre el **DAO** y el controlador
- **ApiApplication.java** arranca la aplicación **Spring** sobre el servidor **Tomcat** (para Java)
- A continuación analizaremos todo el código de ejemplo

2.4 Creando un servicio RESTful con Spring

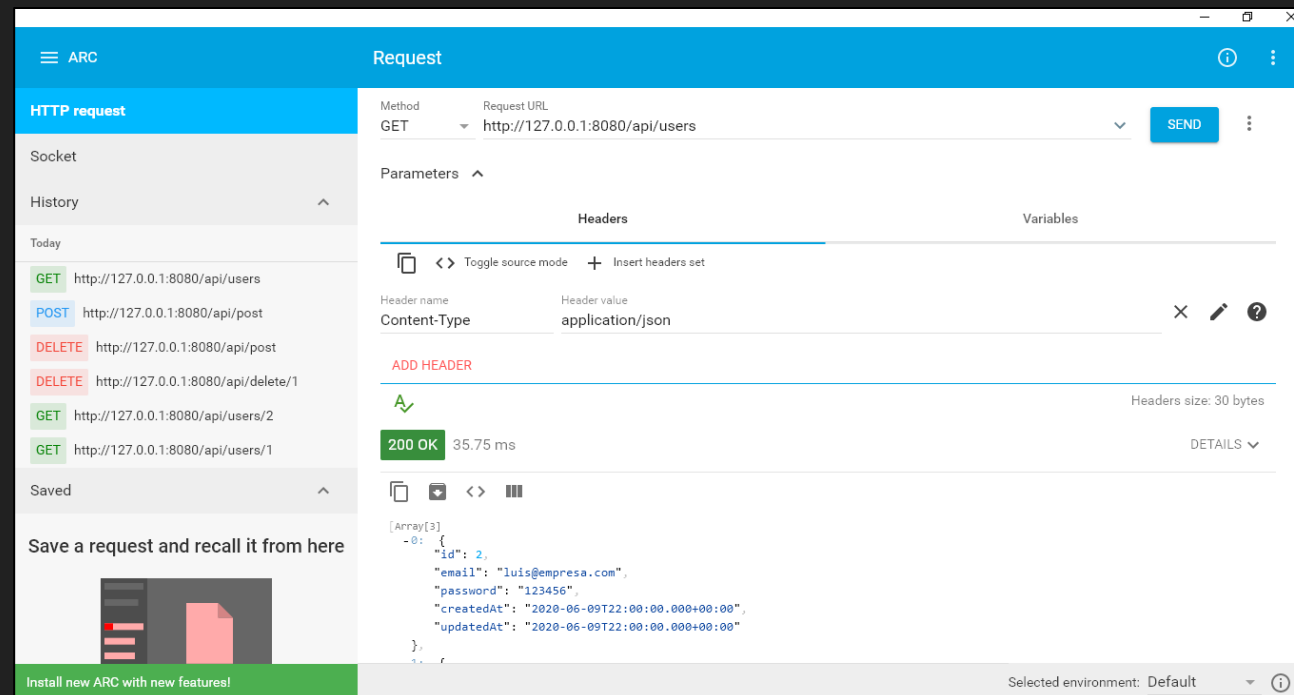
Paso 5. Arrancar la aplicación

- Al lanzar la aplicación esta se ejecutará en un servidor **Apache Tomcat**
- **Tomcat** es el servidor web equivalente a **Java** (igual que Apache para PHP)
- Podremos usar nuestra **API REST** haciendo uso de los métodos configurados en el controller
- Para esto usaremos el complemento **Advanced REST Client**, para el navegador Chrome
 - Usaremos los métodos **PUT, POST, GET...**

2.4 Creando un servicio RESTful con Spring

Paso 6. Probar el servicio

- Enlace de descarga de **Advanced REST Client**: [enlace](#)



2.4 Creando un servicio RESTful con Spring

Paso 6. Probar el servicio

- Consultar todos los usuarios (método **GET**) en **/api/users**

Method	Request URL
GET	http://127.0.0.1:8080/api/users

```
[Array[3]
  - 0: {
    "id": 2,
    "email": "luis@empresa.com",
    "password": "123456",
    "createdAt": "2020-06-09T22:00:00.000+00:00",
    "updatedAt": "2020-06-09T22:00:00.000+00:00"
  },
```

2.4 Creando un servicio RESTful con Spring

Paso 6. Probar el servicio

- Consultar un usuario conociendo su id (método **GET**) en **/api/users/{id}**

Method	Request URL
GET	http://127.0.0.1:8080/api/users/2

```
{
  "id": 2,
  "email": "luis@empresa.com",
  "password": "123456",
  "createdAt": "2020-06-09T22:00:00.000+00:00",
  "updatedAt": "2020-06-09T22:00:00.000+00:00"
}
```





2.4 Creando un servicio RESTful con Spring

Paso 6. Probar el servicio

- Borrar un usuario conociendo su id (método **DELETE**) en **/api/delete/{id}**

Method	Request URL
DELETE ▼	http://127.0.0.1:8080/api/delete/2

200 OK	43.06 ms
--------	----------



Deleted user id - 2

2.4 Creando un servicio RESTful con Spring

Paso 6. Probar el servicio

- Añadir un usuario(método **POST**) en **/api/post**

Method	Request URL
POST	http://127.0.0.1:8080/api/post

FORMAT JSON	MINIFY JSON
<pre>{ "id": 5, "email": "maria@empresa.com", "password": "123456", "createdAt": "2020-06-09T22:00:00.000+00:00", "updatedAt": "2020-06-09T22:00:00.000+00:00" }</pre>	

200 OK	33.97 ms
<pre>{ "id": 5, "email": "maria@empresa.com", "password": "123456", "createdAt": "2020-06-10T10:37:06.140+00:00", "updatedAt": "2020-06-10T10:37:06.140+00:00" }</pre>	
5	maria@empresa.com 123456 2020-06-10 2020-06-10

2.4 Creando un servicio RESTful con Spring

Paso 6. Probar el servicio

- Actualizar un usuario(método **PUT**) en **/api/put**

Method	Request URL
PUT	http://127.0.0.1:8080/api/put

FORMAT JSON MINIFY JSON

```
{
  "id": 5,
  "email": "maria@empresa.com",
  "password": "lalalala",
  "createdAt": "2020-06-09T22:00:00.000+00:00",
  "updatedAt": "2020-06-09T12:48:00.000+00:00"
}
```

200 OK 79.56 ms



```
{
  "id": 5,
  "email": "maria@empresa.com",
  "password": "lalalala",
  "createdAt": "2020-06-09T22:00:00.000+00:00",
  "updatedAt": "2020-06-10T10:51:51.525+00:00"
}
```

5	maria@empresa.com	lalalala	2020-06-10	2020-06-10
---	-------------------	----------	------------	------------

2.4 Creando un servicio RESTful con Spring

Paso 7. Documentación del servicio

- El servicio se ha documentado con **Swagger** (vía la clase **SwaggerConfig.java**)
- **Para ver la documentación acceder a la URL** <http://localhost:8080/swagger-ui.html>

user-rest-controller User Rest Controller	
DELETE	/api/delete/{userId} deteteUser
POST	/api/post addUser
PUT	/api/put updateUser
GET	/api/users findAll
GET	/api/users/{userId} getUser

Créditos de las imágenes y figuras

Cliparts e iconos

- **Obtenidos mediante la herramienta web [IconFinder](#)** (según sus disposiciones):
 - Diapositivas 1, 25 y 38
 - Según la plataforma IconFinder, dicho material puede usarse libremente (free comercial use)
 - A fecha de edición de este material, todos los cliparts son free for comercial use (sin restricciones)

Diagramas, gráficas e imágenes

- Se han desarrollado en PowerPoint y se han incrustado en esta presentación
- Todos estos materiales se han desarrollado por el autor
- Para el resto de recursos se han especificado sus fabricantes, propietarios o enlaces
- Si no se especifica copyright con la imagen, entonces es de desarrollo propio o CC0
- El logo de Spring es propiedad de *Pivotal Software, Inc.*