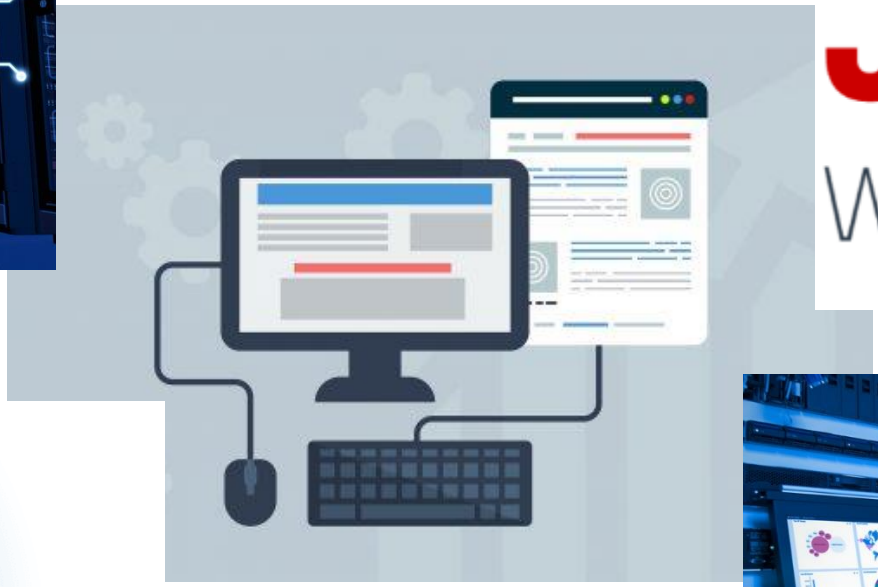
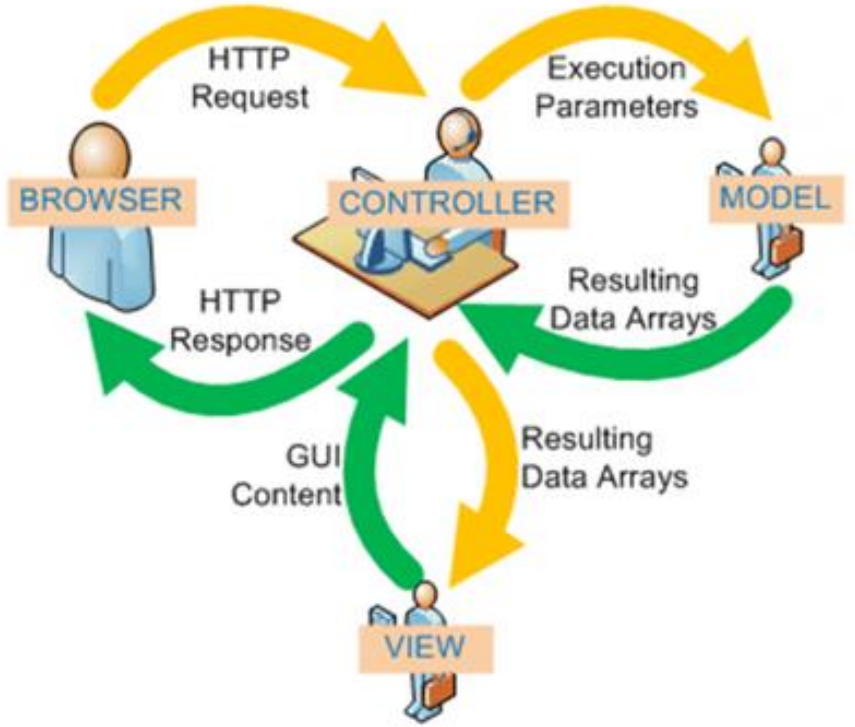
SERVIDORES DE APLICACIONES



Héctor Morán

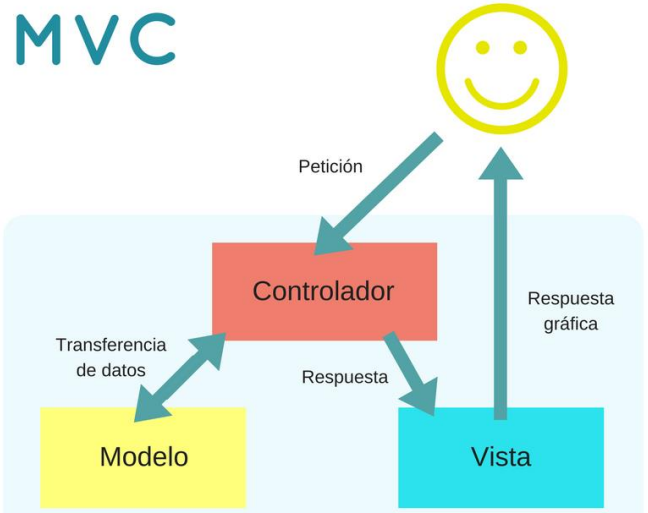
Servidor de aplicaciones

- Un servidor de aplicaciones es un programa de servidor que proporciona la lógica de negocio necesaria para un programa de aplicación. La arquitectura que se emplea en este tipo de aplicaciones es un patrón software que separa la lógica de negocio y los datos de la parte representativa que observa el usuario, los eventos y las comunicaciones entre los distintos componentes. A este modelo se le denomina Modelo-Vista-Controlador (MVC).



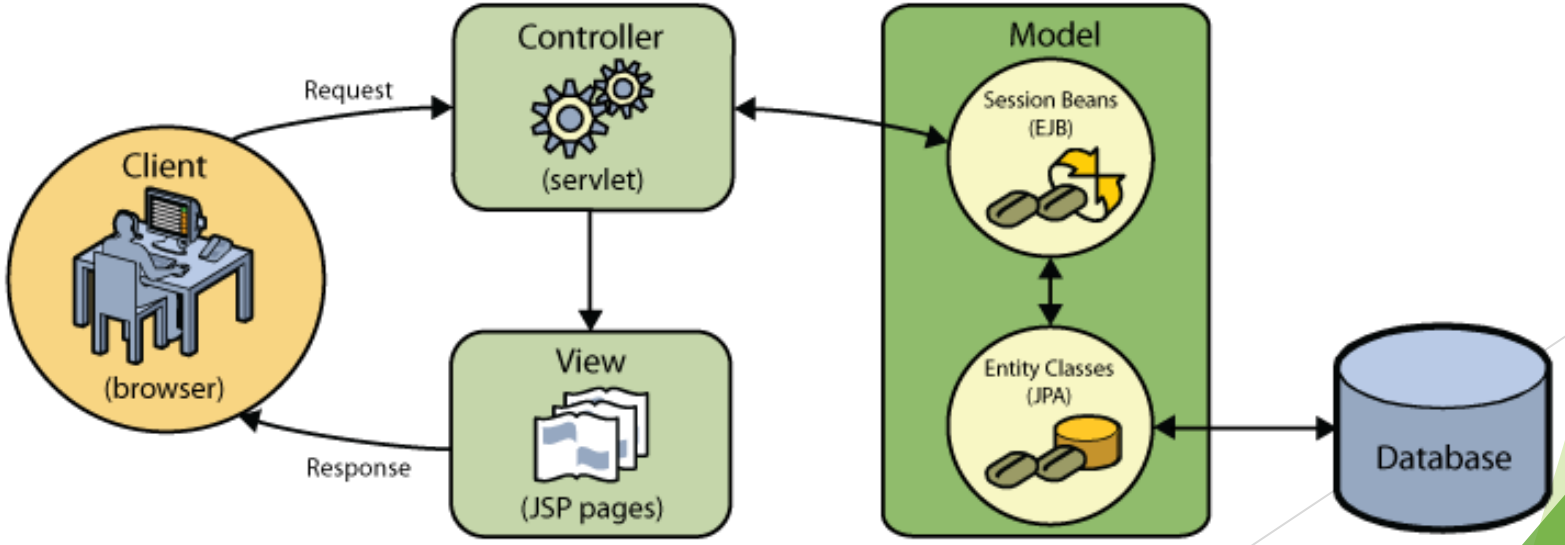
Arquitectura MVC

- Los tres grandes componentes en que se divide el modelo vista controlador son:
  - Modelo: es el componente que se encarga de representar la información con la que la aplicación trabaja. La petición llega por parte del controlador y este componente ejecuta la acción y la presenta a la Vista.
  - Vista: marca el tipo de representación para interactuar con el usuario, es decir, la intefaz de la aplicación web.
  - Controlador: es el módulo que se encarga de controlar a los otros dos componentes, responde a las peticiones realizadas por el usuario y a partir de ahí se comunica con el modelo para manipularlo haciendo cambios en la vista.



Servidor de aplicaciones Java EE

- Un servidor de aplicaciones es un software que permite dar servicio a las aplicaciones empresariales que publica en el mismo, dando soporte para la seguridad, para las transacciones, administración de sesiones, registro de logs, etc.
- Cuando hablamos de servidor de aplicaciones nos solemos referir a un servidor para aplicaciones programadas en el lenguaje Java (en la plataforma Java EE), aunque en ocasiones se utiliza también para referirse a IIS de Microsoft dando servicio a una aplicación ASP.NET.



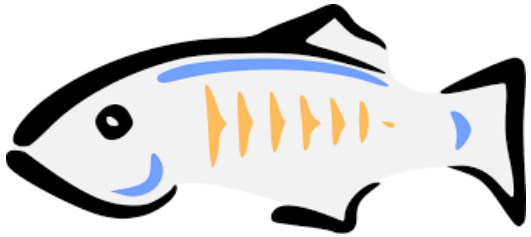
Servidor de aplicaciones Java EE

- Existe gran variedad de servidores de aplicaciones utilizados para dar soporte a aplicaciones web programadas en Java EE (o Jakarta EE):
  - Tomcat: es muy utilizado al ser mucho más sencillo que el resto, aunque no es un servidor de aplicaciones propiamente dicho, únicamente sirve como contenedor de servlets, lo que resulta suficiente para un gran número de aplicaciones web no excesivamente complejas.

  - Wildfly (anteriormente conocidio como JBoss)

WildFly

  - Glassfish

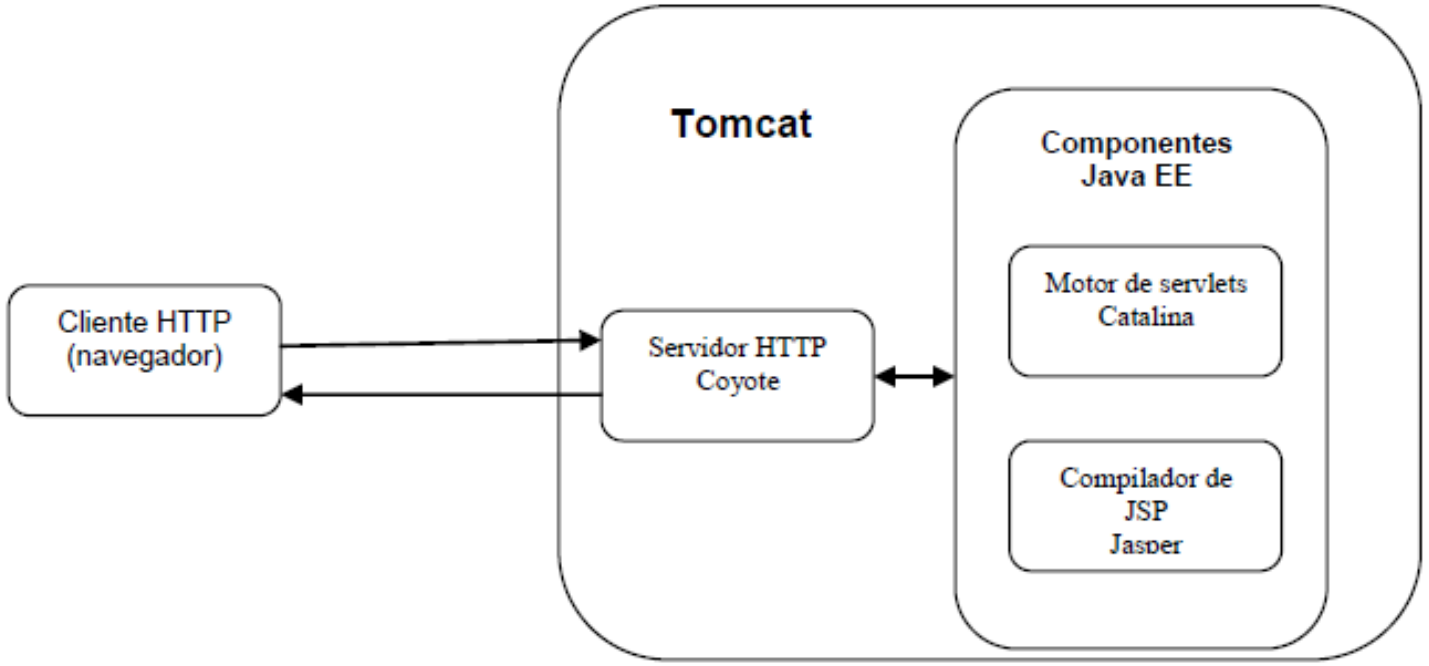


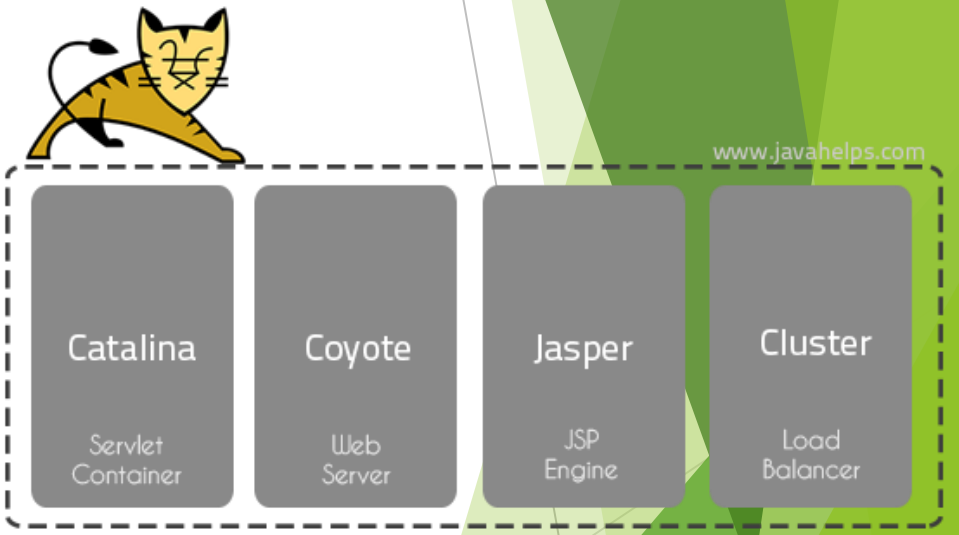
  - Oracle WebLogic Server

ORACLE
WebLogic Server

Apache Tomcat

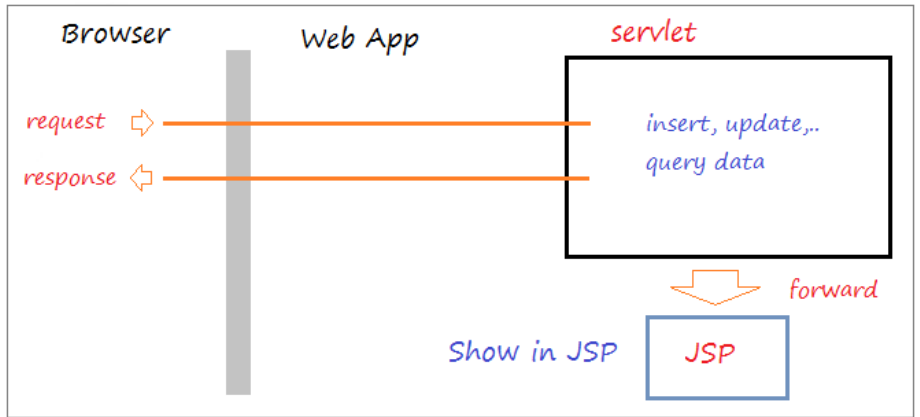
- Las tecnologías que utiliza una aplicación web basada en Java son principalmente los servlets y los JSP (Java Server Pages). El servidor Apache Tomcat tiene por un lado el motor de servlets *Catalina* y por otro el compilador de jsp *Jasper*, incorporando también un servidor http de nombre *Coyote*.



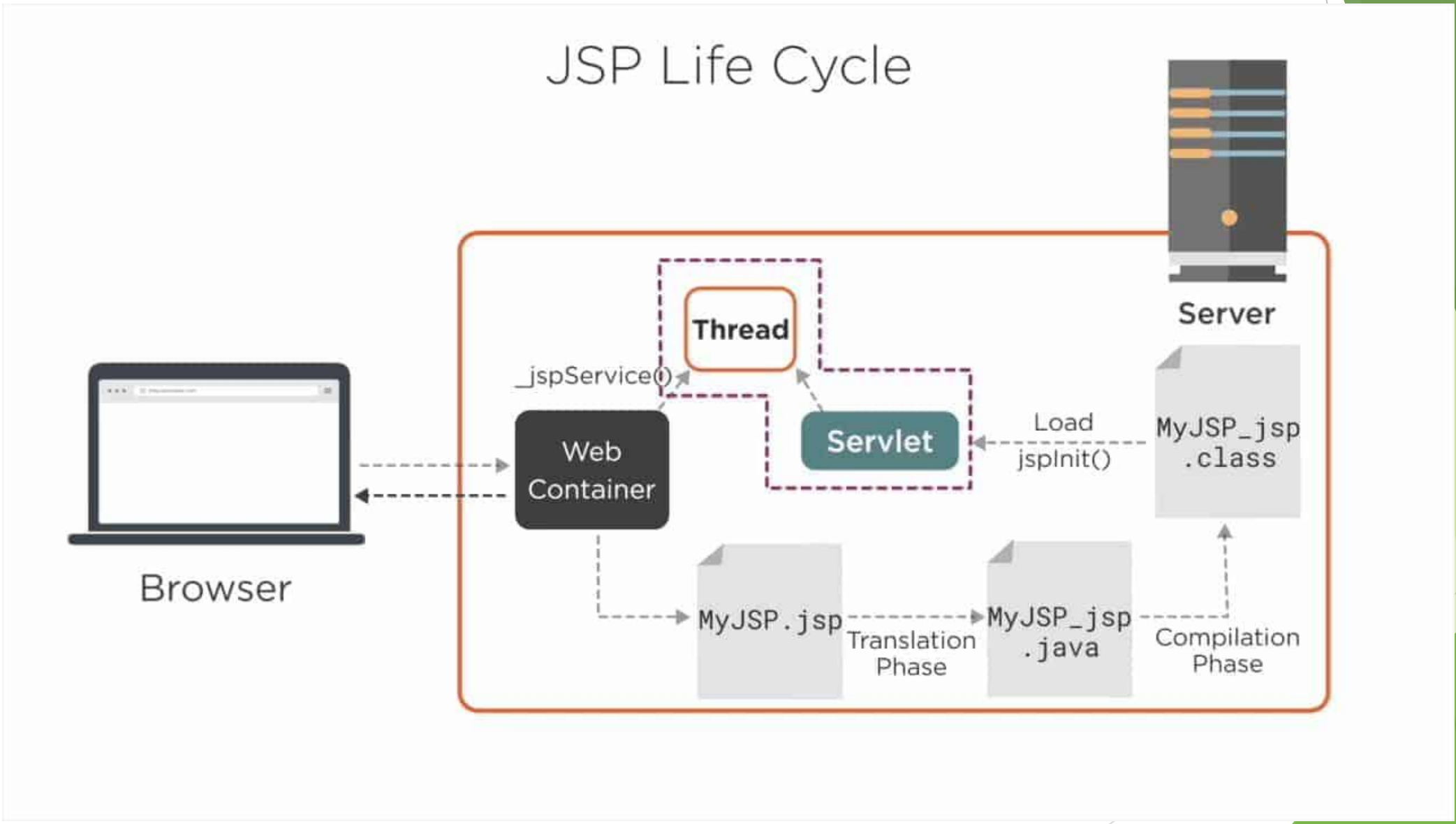


Servlets / JSP

- Los servlets son tecnología que se ejecuta del lado del servidor de aplicaciones y procesan todas la peticiones que hace un cliente. En el esquema MVC un servlet hace de controlador, esto quiere decir que recibe la petición desde el navegador; por ejemplo si la petición es la búsqueda de un cliente (en una base de datos), el servlet delega al modelo a recuperar el cliente y lo envía (response) al navegador a través de una página JSP.
- JSP (Java Server Pages) es una tecnología que permite crear contenido dinámico para aplicaciones web con Java (vista en MVC). Una página JSP (extensión *.jsp*) contiene código HTML, y permite embeber código Java utilizando etiquetas <% %>, todo lo que vaya dentro de estas etiquetas la JVM de Java lo reconoce y lo ejecuta como código Java (similar a los scripts PHP).

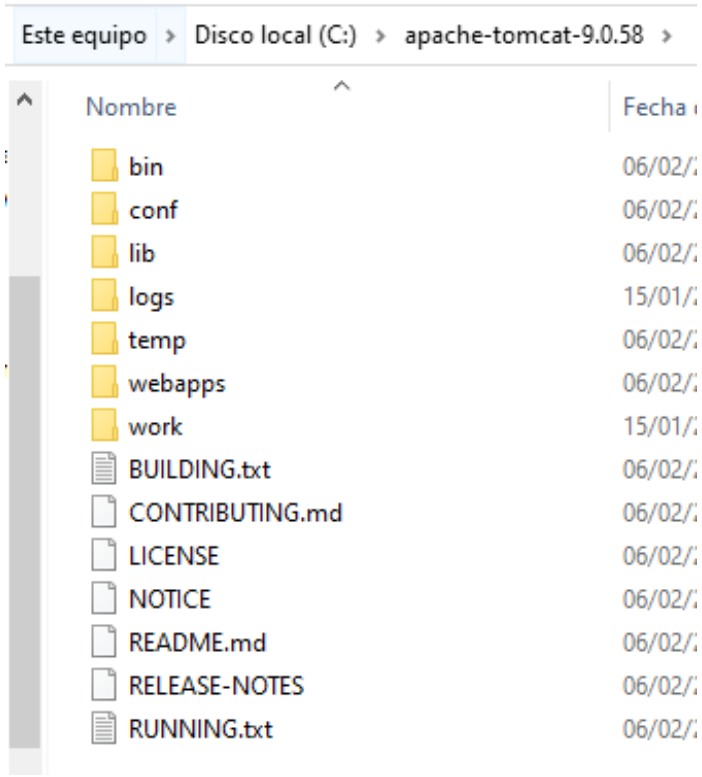


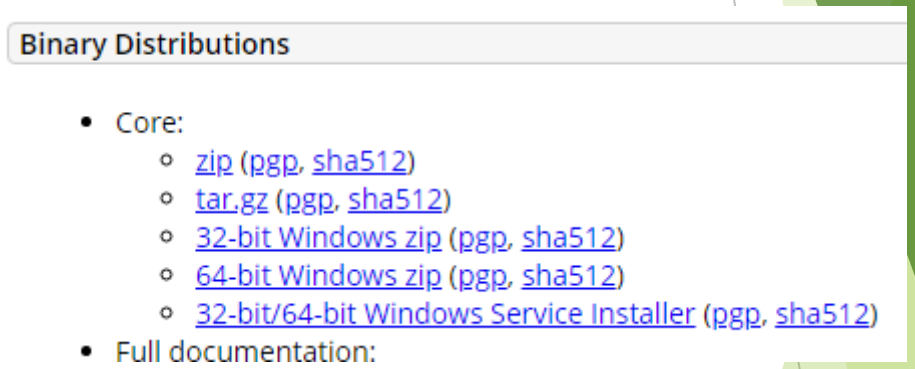
Servlets / JSP



Instalación de Tomcat

- Para instalar el “servidor de aplicaciones” Tomcat la máquina virtual de Java tiene que estar instalada previamente. Podemos descargar la última versión de Tomcat –en este momento es Tomcat 10- desde https://tomcat.apache.org. Para este ejemplo descargaremos Tomcat 9.





Instalación de Tomcat

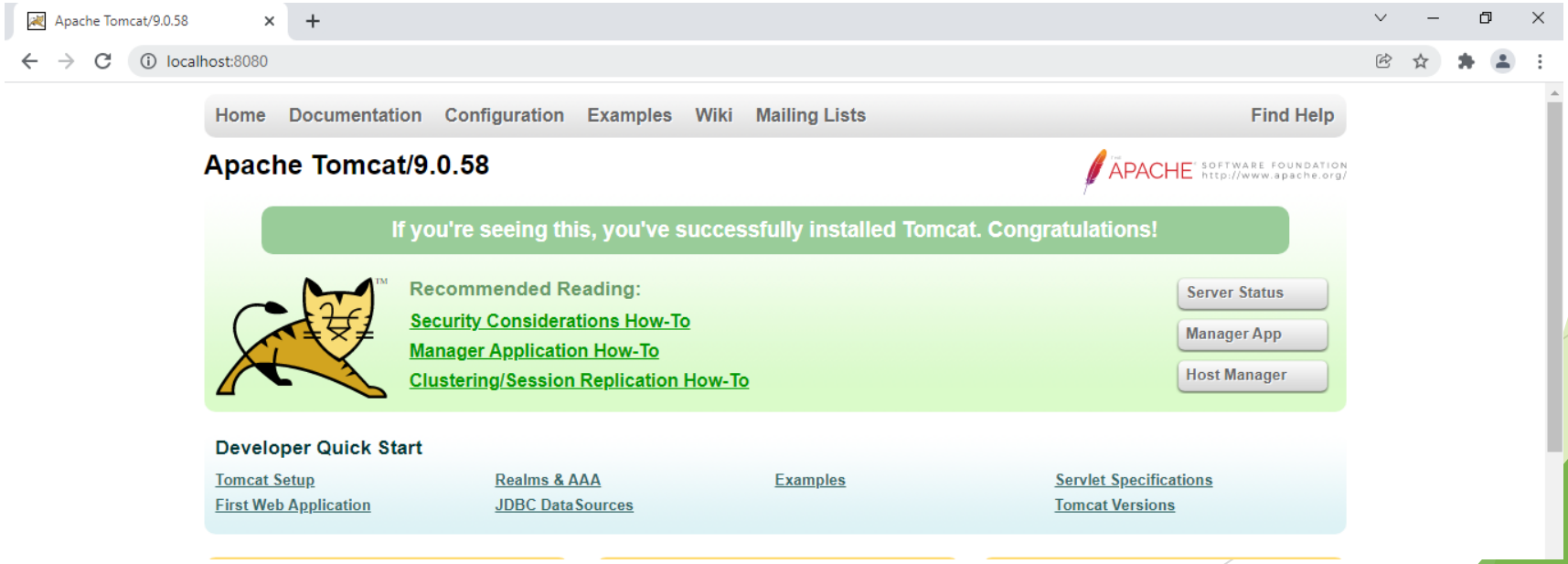
- Para iniciar el “servidor de aplicaciones” Tomcat debemos ejecutar startup.bat situado en la carpeta bin del directorio de Tomcat (también existe una versión que nos podemos descargar que nos instala Tomcat como servicio de Windows). Lo más habitual si trabajamos con un IDE para desarrollar en Java (como por ejemplo Eclipse) es integrar el servidor en él, para poder desplegar nuestra aplicación y ejecutar Tomcat desde el IDE sin necesidad de salir.

```
PS C:\apache-tomcat-9.0.58\bin> .\startup.bat
Using CATALINA_BASE:   "C:\apache-tomcat-9.0.58"
Using CATALINA_HOME:   "C:\apache-tomcat-9.0.58"
Using CATALINA_TMPDIR: "C:\apache-tomcat-9.0.58\temp"
Using JRE_HOME:        "C:\Program Files\Java\jdk-11.0.10+9\"
Using CLASSPATH:       "C:\apache-tomcat-9.0.58\bin\bootstrap.jar;C:\apache-tomcat-9.0.58\bin\tomcat-juli.jar"
Using CATALINA_OPTS:   ""
```

```
06-Feb-2022 20:06:49.100 INFORMACI├ôN [main] org.apache.catalina.startup.HostConfig.deployDirectory Desplegando el directorio [C:\apache-tomcat-9.0.58\webapps\ROOT] de
la aplicaci├│n web
06-Feb-2022 20:06:49.258 INFORMACI├ôN [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [C:\apache-tomcat-9.0.58\web
apps\ROOT] has finished in [158] ms
06-Feb-2022 20:06:49.359 INFORMACI├ôN [main] org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["http-nio-8080"]
06-Feb-2022 20:06:49.983 INFORMACI├ôN [main] org.apache.catalina.startup.Catalina.start Server startup in [11522] milliseconds
```

Instalación de Tomcat

- Tomcat escucha por defecto peticiones que llegan al puerto 8080 del servidor, por lo que para comprobar que la instalación y posterior arranque de Tomcat han funcionado correctamente deberemos entrar en http://localhost:8080. Nos saldrá la página de configuración del servidor Tomcat, donde podremos ver el estado del servicio, desplegar aplicaciones web, etc. (En el directorio *conf* de Tomcat está el archivo *tomcat-users.xml* para configurar las credenciales de los usuarios con los que poder administrar Tomcat).



Aplicaciones en Tomcat

- En el gestor de aplicaciones de Tomcat podemos ver las aplicaciones desplegadas actualmente en el servidor y su estado. Entre ellas tenemos *examples* (para acceder http://localhost:8080/examples, donde podemos ver varios sencillos ejemplos de uso de servlets y jsp).

Gestor de Aplicaciones Web de Tomcat

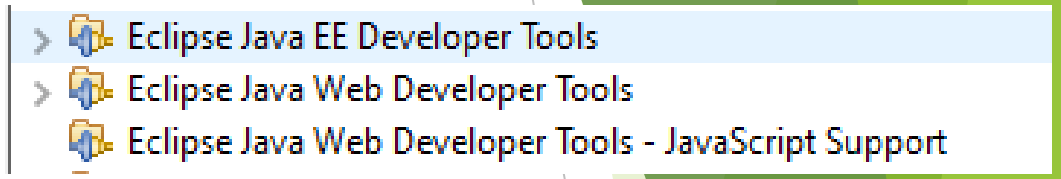
Mensaje:	OK

Gestor			
Listar Aplicaciones	Ayuda HTML de Gestor	Ayuda de Gestor	Estado de Servidor

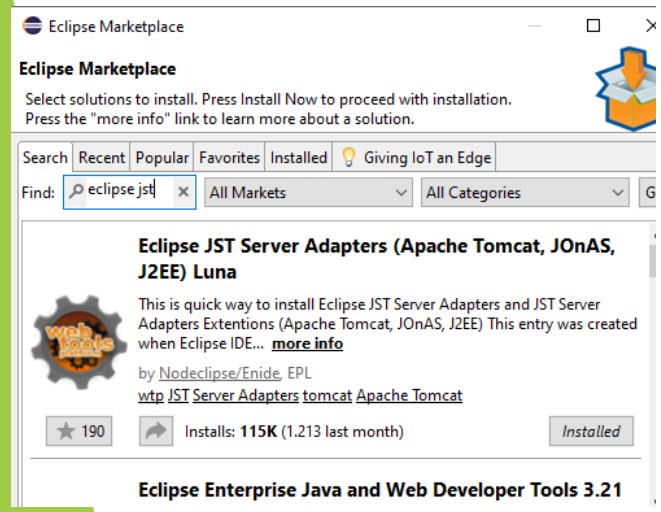
Aplicaciones					
Ruta	**Versión**	**Nombre a Mostrar**	**Ejecutándose**	**Sesiones**	**Comandos**
/	*Ninguno especificado*	Welcome to Tomcat	true	0	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos
/docs	*Ninguno especificado*	Tomcat Documentation	true	0	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos
/examples	*Ninguno especificado*	Servlet and JSP Examples	true	0	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos
/host-manager	*Ninguno especificado*	Tomcat Host Manager Application	true	0	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos

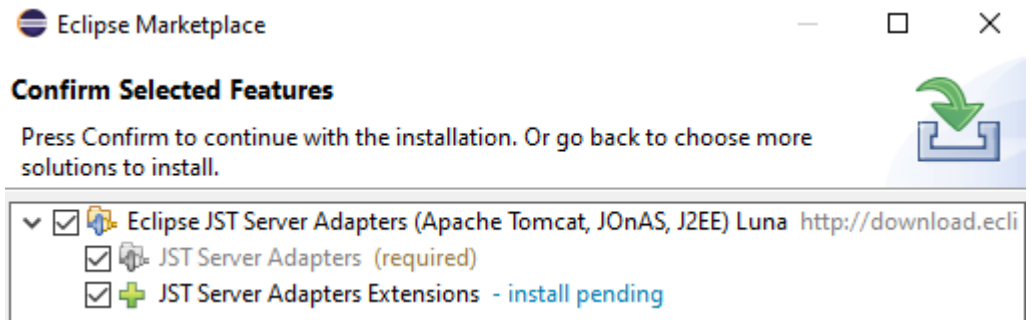
Tomcat en Eclipse

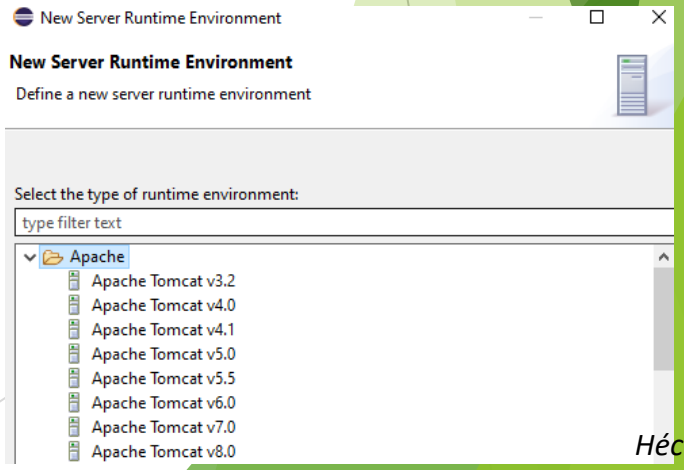
- Para trabajar con Java EE en Eclipse debemos instalar estos módulos (si tienes instalada la versión de Eclipse para desarrollo web ya vienen incluidos).



- Para integrar Tomcat en Eclipse en *Windows → Preferences → Server → Runtime Environments* añadiremos el Tomcat instalado. Para que nos salga Apache Tomcat en esta sección debemos instalar previamente los Eclipse JST Server Adapters y Extensions en Eclipse Marketplace.

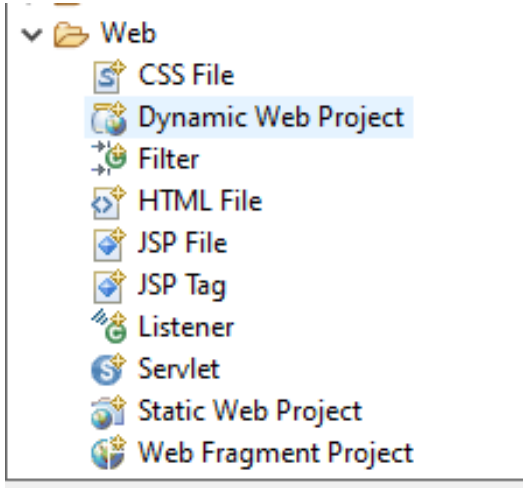


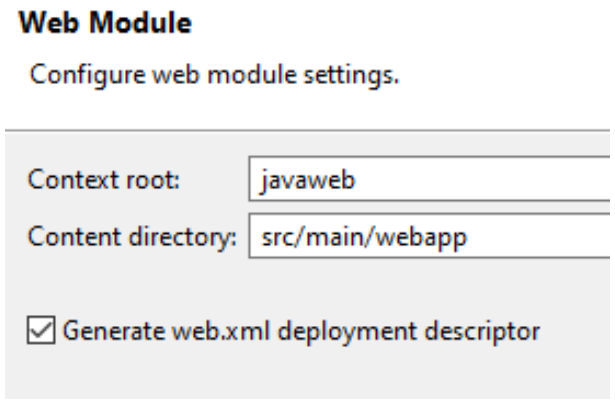


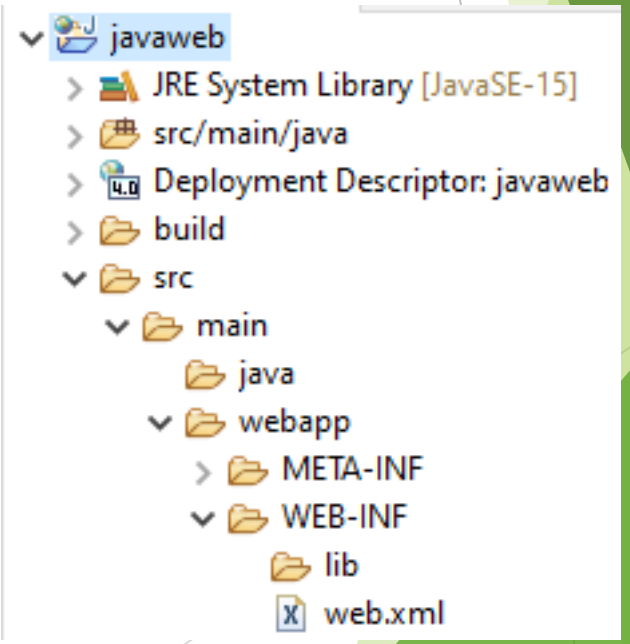


Estructura de una aplicación web Java

- Las aplicaciones web Java están estructuradas según las especificaciones que marca Java EE. En ellas se indica cómo se deben organizar los archivos y las carpetas para que el despliegue de las aplicaciones web sea el correcto.
- Podemos ver la estructura general al iniciar un nuevo *Dynamic Web Project* en Eclipse.







Estructura de una aplicación web Java

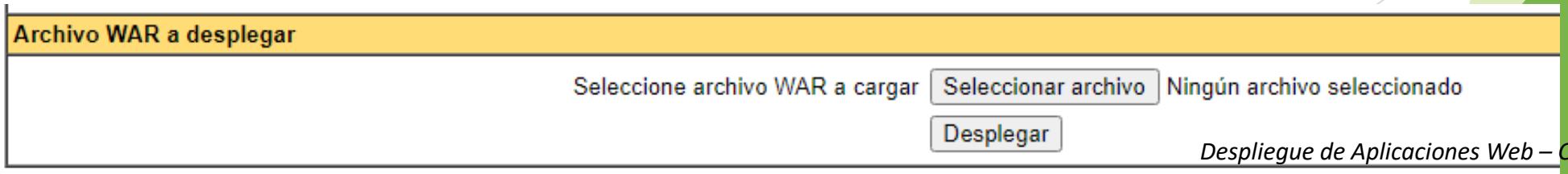
- Una aplicación web es una colección de recursos que guardan relación y que estarán disponibles a través de un servidor de aplicaciones. Cada aplicación web desplegada en un servidor se llama contexto y cada contexto se ha de iniciar, detener y desplegar de forma independiente del resto dentro del servidor. Los recursos de un contexto se despliegan en una carpeta llamada raíz del contexto (Context root) o en sus subdirectorios, de forma que todo bajo la raíz de un contexto sea parte de la misma aplicación web (en Eclipse por defecto el Context root recibe el nombre del proyecto).
- Un archivo fundamental es web.xml. Este archivo proporciona información de configuración y despliegue para los componentes web que componen una aplicación web, definiendo las asignaciones entre las rutas de URL y los servlets que controlan las solicitudes con esas rutas.

Despliegue en Tomcat

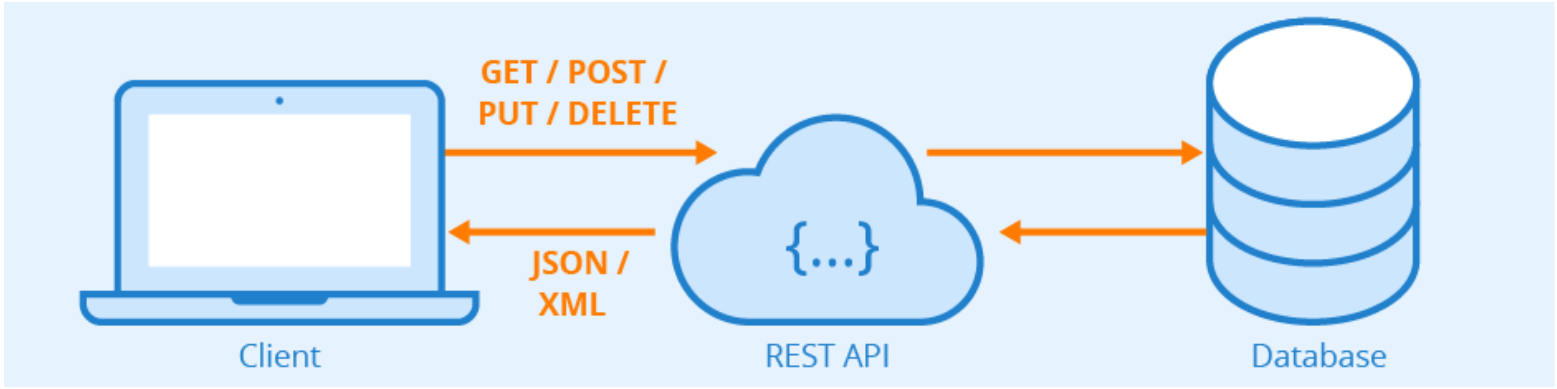
```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello World!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello World!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

```jsp
<%@ page import = "num.NumberGuessBean" %>

<jsp:useBean id="numguess" class="num.NumberGuessBean" scope="session"/>
<jsp:setProperty name="numguess" property="*"/>

<html>
<head><title>Number Guess</title></head>
<body bgcolor="white">
<font size=4>

<% if (numguess.getSuccess()) { %>

  Congratulations!  You got it.
  And after just <%= numguess.getNumGuesses() %> tries.<p>

  <% numguess.reset(); %>

  Care to <a href="numguess.jsp">try again</a>?

<% } else if (numguess.getNumGuesses() == 0) { %>

  Welcome to the Number Guess game.<p>
```

- Las aplicaciones web en Java se exportan en archivos con extensión .war. Estos archivos se pueden desplegar en el servidor Tomcat desde la aplicación web de gestión de aplicaciones.



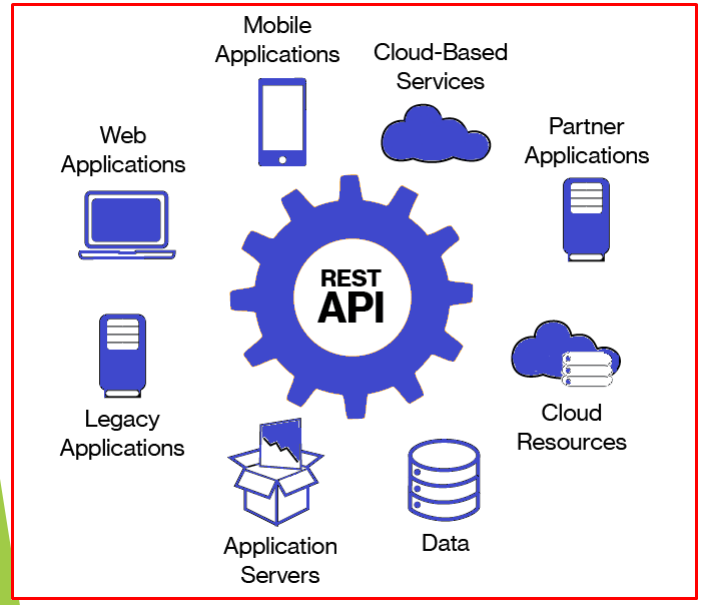
SERVICIOS WEB



JBoss

WildFly

{REST}



RESTful Web Services

XML

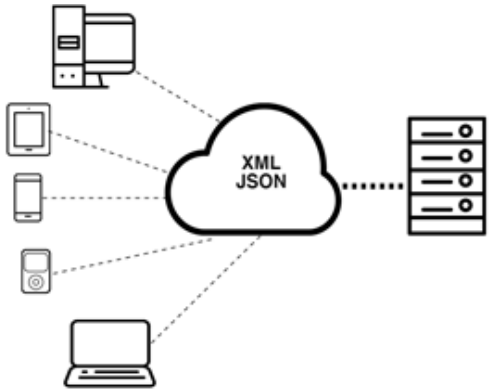
```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  "empinfo" :
    {
        "employees" :  [
        {
            "name" : "James Kirk",
            "age" : 40,
        },
        {
            "name" : "Jean-Luc Picard",
            "age" : 45,
        },
        {
            "name" : "Wesley Crusher",
            "age" : 27,
        }
                    ]
    }
}
```

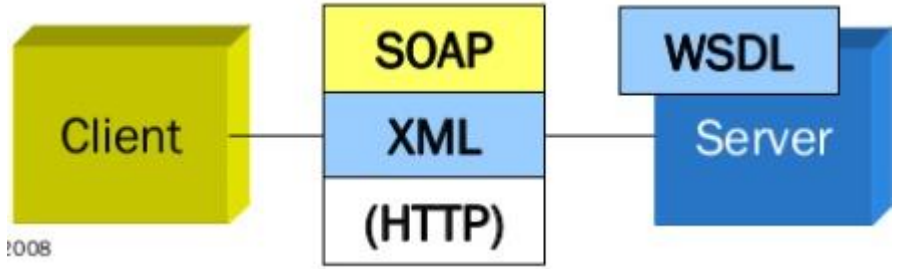
Servicios web

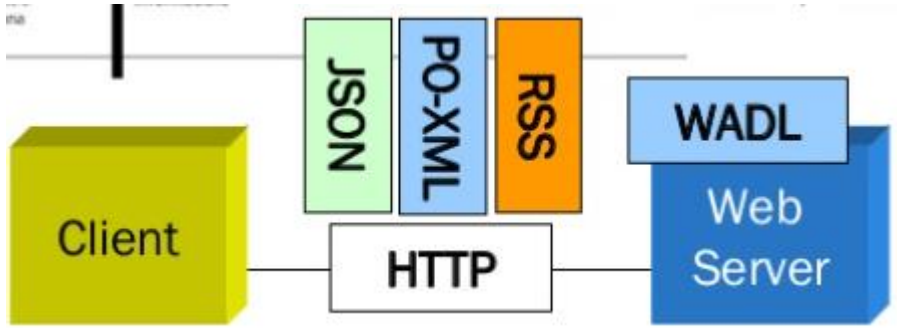
- Además de las aplicaciones web Java basadas en servlet/jsp vistas anteriormente, otra de las aplicaciones servidor típicamente desplegadas en un servidor de aplicaciones son los llamados servicios web. Un servicio web es una aplicación distribuida, basada en el modelo de comunicaciones cliente-servidor, que utiliza el protocolo de nivel de aplicación HTTP para la transferencia de mensajes. El uso de este protocolo, que es el usado en la web, facilita la interoperabilidad entre clientes y servidores.
- Al usar HTTP, los mensajes del protocolo de nivel de aplicación de un servicio web están encapsulados dentro de peticiones y respuestas HTTP. Por lo tanto, cuando un cliente de un servicio web realiza una petición a un servidor esta se envía en el interior de un mensaje HTTP, como un GET, POST o similar.



Tipos de Servicios web

- Existen fundamentalmente dos tipos de servicios web, clasificados según sus características principales y los mecanismos que utilizan para representar los mensajes del protocolo de nivel de aplicación:
  - Servicios web **SOAP**
  - Servicios web **REST**

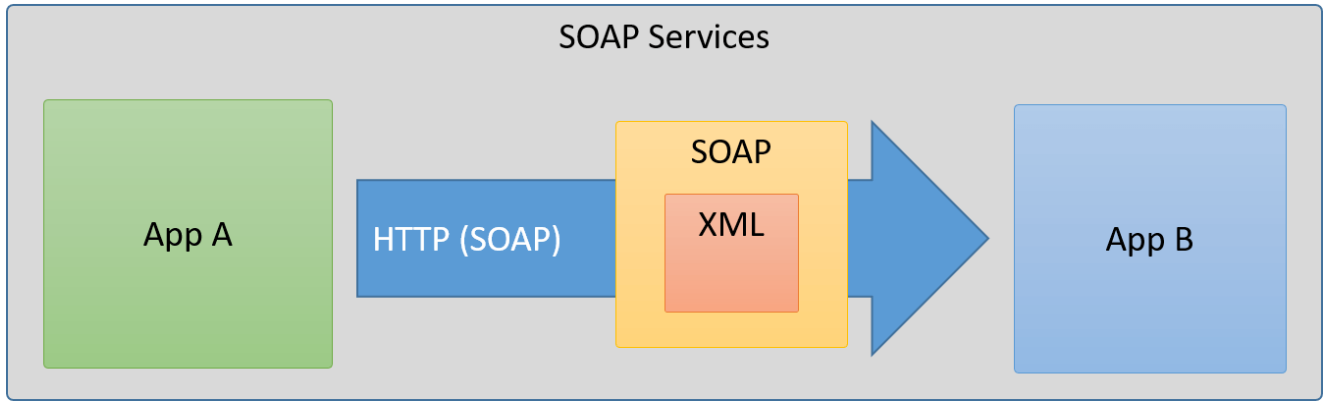




Servicios web SOAP

- Los servicios web SOAP fueron el primer tipo de servicios web que existieron. El formato en que representan los mensajes sigue el estándar SOAP (Simple Object Access Protocol). En SOAP se utiliza XML para definir tanto el protocolo de mensajes como el contenido de estos. Esto hace que los mensajes sean fácilmente procesables de forma automática, a la vez que permite una alta flexibilidad a la hora de diseñarlos.
- La descripción de interfaz de servicio de un servicio web SOAP se realiza usando un lenguaje basado en XML llamado WSDL (Web Services Description Language).
- Un servicio web basado en SOAP debe cumplir los siguientes requisitos:
  - Se debe expresar de manera formal y pública la interfaz del servicio (mediante WSDL).
  - La arquitectura del servicio debe ser capaz de soportar la realización y procesamiento de peticiones de forma asíncrona.
  - Los servicios web basados en SOAP pueden o no tener estado (stateful o stateless).

Servicios web SOAP



Servicios web SOAP en Java

- La librería utilizada para los servicios web SOAP en Java es JAX-WS (Java API for XML Web Services), podemos descargar la implementación de Eclipse en la siguiente url: https://github.com/eclipse-ee4j/metro-jax-ws

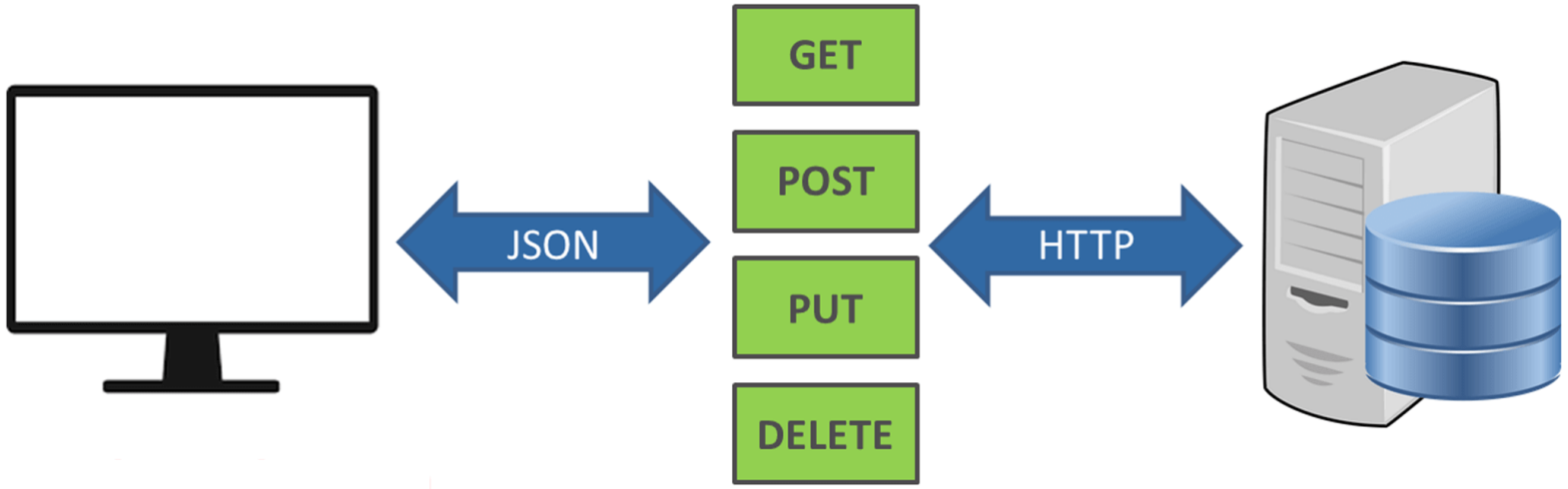
```
public static void main(String[] args) {

    /*
     *  Se publican los servicios a través de un servidor virtual.
        El puerto puede ser cualquiera.
        Un vez ejecutada la aplicación, se publica el contrato WSDL
      */

    Endpoint.publish("http://localhost:1516/WS/Users", new SOAPImpl());

}
```

```
@WebService(endpointInterface = "es.rosamarfil.soap.SOAPI")
public class SOAPImpl implements SOAPI{

    @Override
    public List<User> getUsers() {
        return User.getUsers();
    }

    @Override
    public void addUser(User user) {
        User.getUsers().add(user);
    }
```

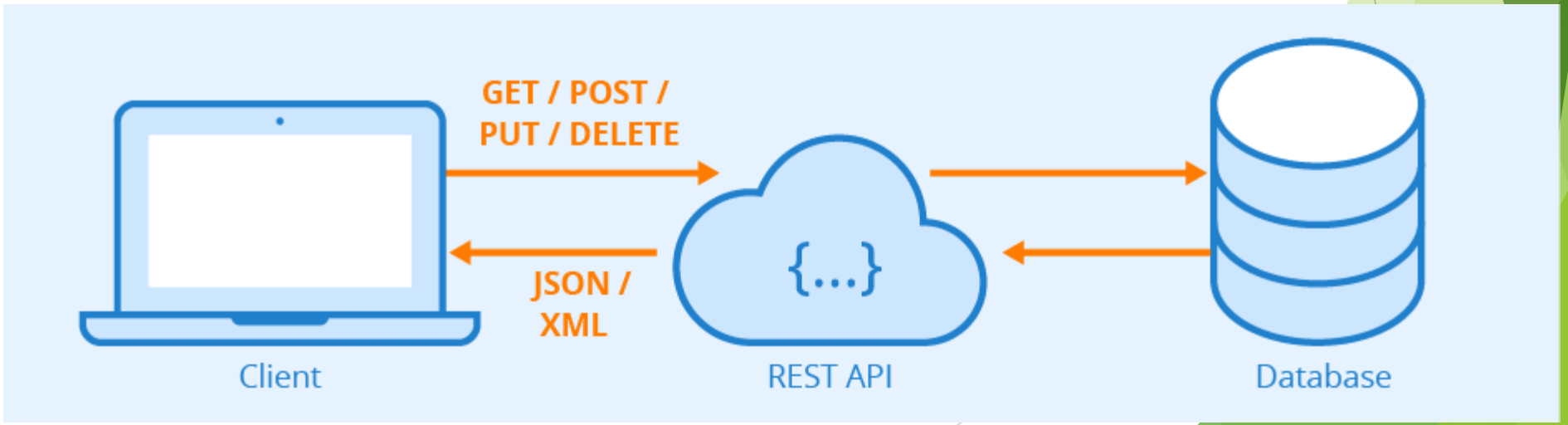
Servicios web REST

- Los servicios web REST utilizan un formato mucho más ligero y flexible que SOAP, ya que no están forzados a utilizar XML para representar sus mensajes y su interfaz.
- Los principios básicos de los servicios REST son:
  - Utiliza métodos HTTP de forma explícita (para crear un recurso utiliza POST, para recuperar un recurso GET, para actualizar un recurso PUT, para borrar un recurso DELETE)
  - Es sin estado (stateless). Los clientes de servicios web de REST tienen que enviar solicitudes completas e independientes. Una solicitud completa e independiente no requiere que el servidor recupere ningún tipo de contexto o estado de la aplicación, mientras procesa la solicitud. Las aplicaciones (o clientes) de los servicios web de REST incluyen, dentro de las cabeceras y cuerpos HTTP de una solicitud, todos los parámetros, contexto y datos que el componente por el lado del servidor necesita para generar una respuesta
  - Expone las URIs como estructuras de directorio.
  - Transfiere XML, JSON, o ambos.

Servicios web REST



Client sends a **request** **HTTP methods** Server sends a **response**



Servicios web REST en Java

- La librería utilizada para los servicios web REST en Java es JAX-RS (Java API for RestFul Web Services), podemos descargar la implementación de Eclipse (Jersey) en la siguiente url: https://eclipse-ee4j.github.io/jersey/

```java
/**
 * Root resource (exposed at "myresource" path)
 */
@Path("myresource")
public class MyResource {

    /**
     * Method handling HTTP GET requests. The returned object will be sent
     * to the client as "text/plain" media type.
     *
     * @return String that will be returned as a text/plain response.
     */
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String getIt() {
        return "Got it!";
    }
}
```

```java
public class MyResourceTest {

    private HttpServer server;
    private WebTarget target;

    @Before
    public void setUp() throws Exception {
        server = Main.startServer();

        Client c = ClientBuilder.newClient();
        target = c.target(Main.BASE_URI);
    }

    @After
    public void tearDown() throws Exception {
        server.stop();
    }

    /**
     * Test to see that the message "Got it!" is sent in the response.
     */
    @Test
    public void testGetIt() {
        String responseMsg = target.path("myresource").request().get(String.class);
        assertEquals("Got it!", responseMsg);
    }
}
```