CONCEPTOS DE JAVA EE NECESARIOS

“

Una aplicación web con Spring MVC no deja de ser una aplicación web Java.

Por tanto, necesitamos conocer algunos conceptos de este modelo de desarrollo de aplicaciones.

SERVLET

- Clase que nos permite gestionar peticiones/respuestas de una aplicación servidora.
- Usualmente usados en contextos web bajo el protocolo HTTP.
- Se ejecutan en un contenedor de servlets (por ejemplo, Tomcat)

EJEMPLO DE UN SERVLET

```java
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.annotation.WebServlet;

@WebServlet("/index")
public class ServletExample extends HttpServlet{

    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException{
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("<h1>Hello Servlet Get</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

https://javaee.github.io/javaee-spec/javadocs/javax/servlet/http/HttpServlet.html

WEB.XML:
DESCRIPTOR DE DESPLIEGUE

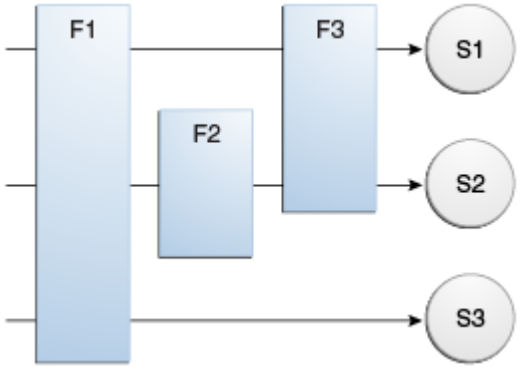
- Fichero que indica cómo desplegar en el servidor los componentes de la aplicación (servlets entre otros)
- Formato XML
- Ruta /WEB-INF/web.xml

WEB.XML:
DESCRIPTOR DE DESPLIEGUE

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_9" version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <display-name>Servlet 3.0 application</display-name>
    <servlet>
        <servlet-name>welcome</servlet-name>
        <servlet-class>WelcomeServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>welcome</servlet-name>
        <url-pattern>/hello.welcome</url-pattern>
    </servlet-mapping>
</web-app>
```

FILTER

- Los filtros permiten transformar peticiones (sobre todo) o respuestas HTTP
- También nos permiten alterar el *circuito* normal que seguiría una petición/respuesta en función del contenido de estas.
- Por ejemplo, seguridad, cambiar codificación a UTF-8, ...



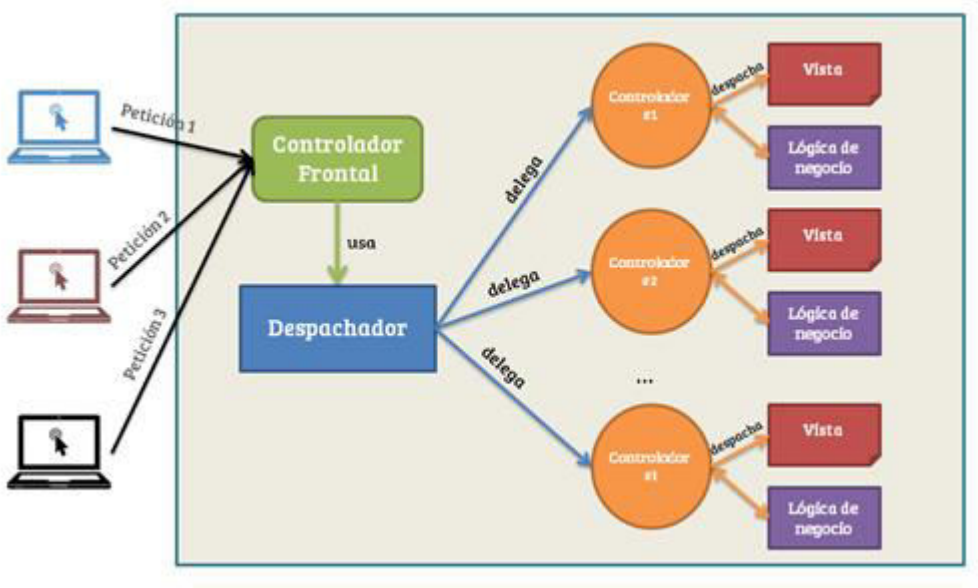
FILTER

```java
@WebFilter("/AuthenticationFilter")
public class AuthenticationFilter implements Filter {

    public void init(FilterConfig fConfig) throws ServletException {
    }

    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {

        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse res = (HttpServletResponse) response;

        String uri = req.getRequestURI();
        this.context.log("Requested Resource::"+uri);

        HttpSession session = req.getSession(false);

        if(session == null && !(uri.endsWith("html") || uri.endsWith("LoginServlet"))){
            this.context.log("Unauthorized access request");
            res.sendRedirect("login.html");
        }else{
            // pass the request along the filter chain
            chain.doFilter(request, response);
        }
    }

    public void destroy() {
        //close any resources here
    }
}
```

¿Y CÓMO USA TODO ESTO SPRING MVC?

DISPATCHER SERVLET

- Implementación del patrón **Front Controller**



DISPATCHER SERVLET

- Despacha todas las peticiones (delegando en otras clases).
- Necesita ser mapeado (descriptor de despliegue o JavaConfig).
- ***Si usamos Spring Boot, esto no es necesario, él se encarga de esta configuración.***

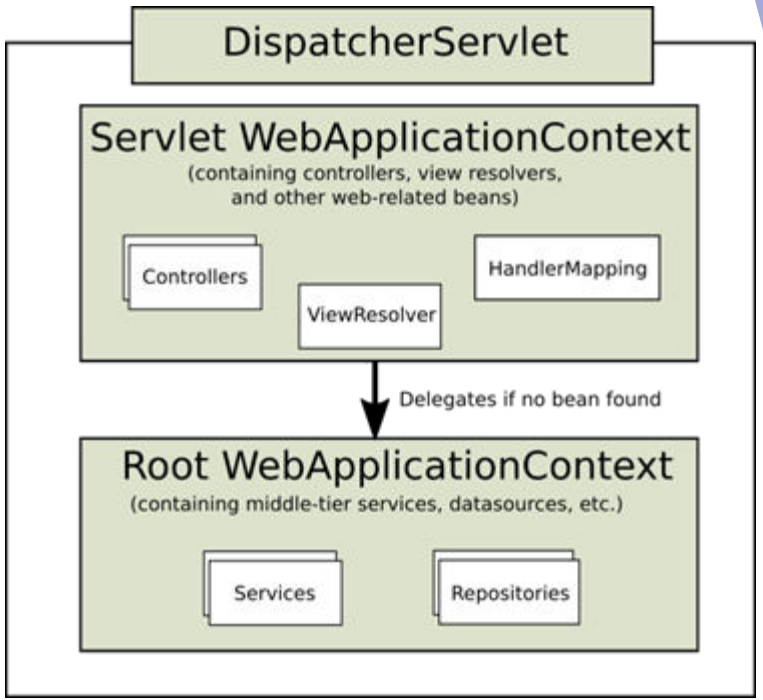
https://docs.spring.io/spring/docs/5.1.2.RELEASE/spring-framework-reference/web.html#mvc-servlet

CONTEXTOS

- Una aplicación Spring necesita un *ApplicationContext*: contenedor de inversión de control.
- Para apps web tenemos *WebApplicationContext*.

CONTEXTOS

Usualmente (y así trabajaremos durante el curso) solo necesitaremos un *WebApplicationContext*, pero podemos tener una jerarquía.



ALGUNOS BEANS ESPECIALES

Colaboran con el DispatcherServlet

- *HandlerMapping* y *HandlerAdapter*: mapeo de peticiones a métodos de un controlador.
- *ViewResolver*: transforma nombres de vistas en vistas con las que renderizar una respuesta.

¿Y QUÉ HACE SPRING BOOT POR NOSOTROS?

CONFIGURACIÓN DE DISPATCHER SERVLET

Sin Spring Boot, tendríamos que hacer esto *manualmente*.

```
import org.springframework.web.WebApplicationInitializer;

public class MyWebApplicationInitializer implements WebApplicationInitializer {

    @Override
    public void onStartup(ServletContext container) {
        XmlWebApplicationContext appContext = new XmlWebApplicationContext();
        appContext.setConfigLocation("/WEB-INF/spring/dispatcher-config.xml");

        ServletRegistration.Dynamic registration = container.addServlet("dispatcher",
new DispatcherServlet(appContext));
        registration.setLoadOnStartup(1);
        registration.addMapping("/");
    }
}
```

CONFIGURACIÓN DE DISPATCHER SERVLET

Sin Spring Boot, tendríamos que hacer esto *manualmente*.

```
public class MyWebAppInitializer extends
AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return null;
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class<?>[] { MyWebConfig.class };
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }
}
```

CONFIGURACIÓN DE DISPATCHER SERVLET

Sin Spring Boot, tendríamos que hacer esto *manualmente*.

```
public class MyWebAppInitializer extends AbstractDispatcherServletInitializer {

    // ...

    @Override
    protected Filter[] getServletFilters() {
        return new Filter[] {
            new HiddenHttpMethodFilter(), new CharacterEncodingFilter() };
    }
}
```