INFORME DE ARQUITECTURA DE SISTEMAS DE INFORMACIÓN WEB

27 DE FEBRERO DE 2022

E2.07

https://github.com/jaimemosber/Acme-Toolkits

Jaime Borrego Conde: jaiborcon@alum.us.es

Antonio Campos Gil: antcamgil@alum.us.es

Ana Conde Marrón: anaconmar@alum.us.es

Gonzalo Martínez Fernández: gonmarfer2@alum.us.es

Jaime Moscoso Bernal: jaimosber@alum.us.es

Enrique Muñoz Pérez: enrmunper@alum.us.es

Tabla de contenido

Resumen ejecutivo

El objetivo del presente documento es exponer a los profesores de la asignatura los conocimientos que el grupo de trabajo ha adquirido a través de formación previa y que actualmente se posee acerca de los sistemas de información web, pues sobre ellos será sobre los que se trabaje en la asignatura Diseño y Pruebas II.

Tabla de revisión

Número de revisión	Fecha	Descripción
1	27/02/2022	Primera versión del documento

Contenido

La arquitectura del *software* de un sistema define la organización de los módulos que lo comprenden: desde elementos *software* (vistas, puntos de variabilidad y extensión…) hasta sus relaciones e interacciones, incluyendo sendas propiedades.

Por otra parte, una aplicación *web* es una aplicación informática distribuida cuya interfaz de usuario es accesible desde clientes web, normalmente navegadores que funcionan sobre dispositivos electrónicos como ordenadores, teléfonos móviles…

Generalmente, una aplicación *web* trabaja sobre el protocolo de aplicación HTTP y el protocolo de transporte TCP/IP, a través de peticiones y respuestas. Los usuarios interactúan localmente con la aplicación, que envía los datos a un servidor, el cual los procesa. Asimismo, las aplicaciones *web* suelen ser sistemas de información, que almacenan datos en bases de datos, ora con el fin de guardarlos, ora para procesarlos y obtener nueva información útil.

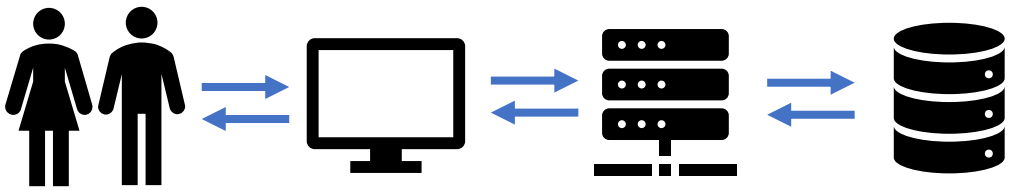

Figura 1. Esquema de funcionamiento de una aplicación web

De esta manera, la arquitectura de los sistemas de información *web* (WIS) es el conjunto de estructuras necesarias para razonar sobre este tipo de sistemas, definiendo formas y guiando su construcción. Dentro de este concepto, es menester mencionar los estilos frente a los patrones.

En primer lugar, los patrones son soluciones generales y aceptadas para un problema de diseño arquitectónico común, que expresa una relación entre un contexto, un problema y una solución. Como ejemplo, el patrón MVC (Modelo, vista y controlador) ofrece una manera de separar los datos de una aplicación de su interfaz de usuario y de la lógica de negocio en tres componentes diferenciados. Por el contrario, los estilos son restricciones sobre el sistema completo para abordar un problema arquitectónico más general. Los estilos más renombrados son el estilo en capas y el estilo por microservicios.

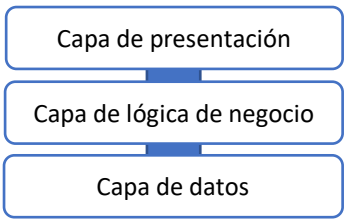

Figura 3. Diagrama del estilo en capas

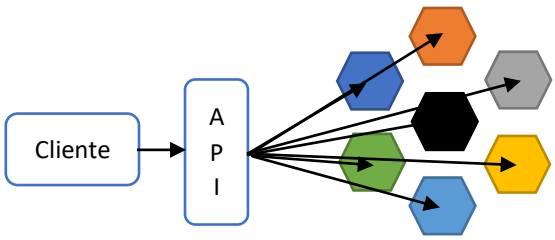

Figura 2. Diagrama del estilo de microservicios

Como se mencionó anteriormente, el patrón arquitectónico MVC define tres componentes: el modelo, la vista y el controlador. El modelo representa la información, incluyendo tanto los datos como la lógica de negocio necesaria para trabajar con ellos. Por otro lado, la vista es la presentación del modelo, es decir, de la información, de forma que el usuario pueda interactuar con ella. Para ello, se suele emplear una interfaz de usuario. Finalmente, el controlador responde

a eventos de la interfaz de usuario, como las peticiones HTTP, y lleva a cabo cambios en el modelo, y posiblemente, también sobre la vista.

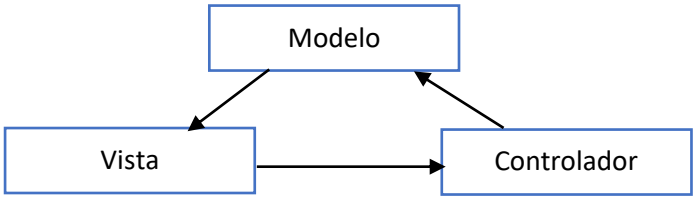

Figura 4. Diagrama básico del patrón MVC

El modelo es un componente conformado a su vez por tres elementos: las entidades, los repositorios y los servicios. Primeramente, las entidades representan objetos del dominio del problema que serán almacenados en la base de datos. Los repositorios proveen métodos para interactuar con la base de datos, habiendo uno por entidad. Por último, los servicios exponen la funcionalidad del dominio como una API, y se encargan de la validación del dominio, aunque no de las entradas, tarea de los controladores.

Figura 5. Logo de Spring MVC

Spring MVC es uno de los *framework* web más famosos en la actualidad. Este introduce ciertos *servlet*, módulos que permiten extender las capacidades de los servidores *web*. El funcionamiento general de una aplicación *web* construida en torno a *Spring MVC* es el siguiente:

1. Un cliente web envía una petición al servidor, la cual es recibida por el *Dispatcher Servlet*, encargado de gestionarla.
2. Este *servlet* acude a un mapeo de "*handlers*", que decide cuál es el controlador apropiado para procesar la petición. Esto se conoce gracias a las anotaciones halladas en los controladores.
3. Una vez, decidido, la petición se envía a su controlador destino, transformándola de formato HTML/HTTP a objetos Java.
4. Una vez procesada, el controlador devuelve una vista junto a información que mostrar en ella.
5. Para mostrar la vista, se decide primero qué tecnología utilizar. Es común JSP (Java Server Pages), aunque existen numerosos tipos, con sus ventajas e inconvenientes.
6. Finalmente, la vista se muestra en el navegador del cliente *web*.

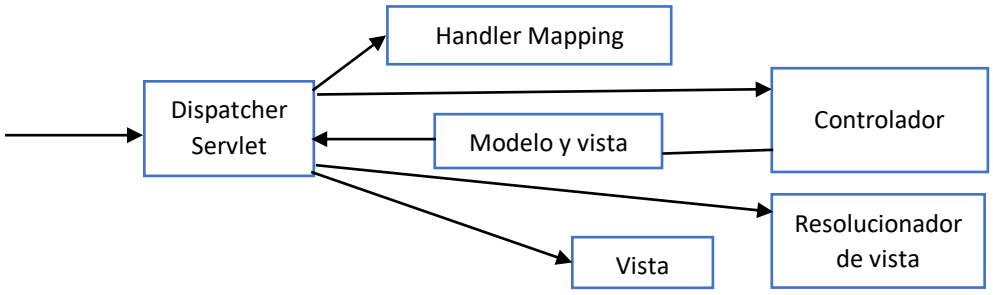

Figura 6. Esquema de funcionamiento del Dispatcher Servlet

¿Cómo reconoce el *Dispatcher Servlet* cuál es el controlador adecuado? Por medio de las anotaciones *@RequestMapping*, que se encuentran en los controladores. Estas indican, según la URI que contengan, a qué controlador deben ir las peticiones que llegan al *Dispatcher Servlet*. Por ejemplo, si la URI es *localhost:8080/forums/3/edit*:

```
@Controller
@RequestMapping("/forums")
public class ForumController {
```

Figura 7. Cabecera de un controlador

```
// Editar foro
@GetMapping(value = "/{forumId}/edit")
public String initUpdateForumForm(@PathVariable("forumId") int forumId, Model model) {
    String view = VIEW_EDIT_FORUM;
    Forum forum = this.forumService.findForumById(forumId).get();
    model.addAttribute(forum);
    return view;
}
```

Figura 8. Ejemplo de método GET de controlador

Se enviará la petición al controlador de foros, el cual ejecutará el método "initUpdateForumForm", que nos permitirá acceder al formulario de edición de un foro. En el caso de que la petición fuese un *post* accedería a este otro método que rellenaría la información del objeto foro creado.

```
@PostMapping(value = "/{forumId}/edit")
public String processUpdateForumForm(@Valid Forum forum, BindingResult result,
        @PathVariable("forumId") int forumId) {
    String view = VIEW_EDIT_FORUM;
    if (result.hasErrors()) {
        return view;
    } else {
        forum.setId(forumId);
        this.forumService.save(forum);
        return "redirect:/forums";
    }
}
```

Figura 9. Ejemplo de método POST de controlador

Un último aspecto para remarcar sobre los sistemas de información web son las pruebas (*testing*), imprescindible para asegurar la calidad del software. Probar el código implica introducir datos en la ejecución para comprobar que el comportamiento esperado coincide con el resultado real. Si se detecta un fallo, un comportamiento inapropiado o errático en la aplicación, posiblemente se deba a la existencia de errores en el código.

Probar código puede ser llevado a cabo con diferente granularidad, cubriendo diferentes caminos y siguiendo distintas estrategias, generalmente, dirigidas a automatizar el proceso, pues es costoso y tedioso.

Bibliografía

Intencionalmente en blanco.