WIS Report

Group: E7.05

Repository: https://github.com/PabloAlvarezCaroUS/Acme-Toolkits

Student #1

ID Number: 29513104X

Name: Álvarez Caro, Pablo

Roles: Manager, Developer

E-mail: pabalvcar@alum.us.es

Student #2

ID Number: 77863065S

Name: García Bohórquez, José Ignacio

Roles: Developer, Tester

E-Mail: josgarboh@alum.us.es

Student #3

ID Number: 53963760X

Name: Marín Gómez, Pablo

Roles: Developer, Tester

E-Mail: pabmargom3@alum.us.es

Student #4

ID Number: 29541858Z

Name: Migueles Dominguez, Francisco Javier

Roles: Analyst, Developer

E-Mail: framigdom@alum.us.es

Student #5

ID Number: 49128512Y

Name: Ruiz Gil, Diego

Roles: Developer, Tester

E-mail: dieruigil@alum.us.es

Student #6

ID Number: 29544227Z

Name: Zamora Fernández, David

Roles: Developer, Tester

E-Mail: davzamfer@alum.us.es

Date: 28/05/2022

Índice

Historial de versiones

Versión	Fecha	Registro de cambios
1.0	23/05/2022	● Versión inicial
1.1	28/05/2022	● Actualizado

Resumen Ejecutivo

En este documento vamos a exponer nuestros conocimientos sobre las WIS que hemos adquirido durante la realización de la asignatura.

Introducción

En este informe realizaremos una descripción de los componentes e interacciones sobre la arquitectura aprendida a lo largo del curso. En específico, hablaremos sobre la arquitectura por capas de la cual está compuesta el framework y el proyecto en cuestión, así como lo que hemos aprendido sobre ello.

Contenido

Componentes de una WIS

El navegador:

Es usado por los clientes para mandar peticiones HTTP a un server y realizar las peticiones, en HTML, CSS y JS.
En nuestro caso, hemos usado el navegador de Chrome para el testing informal (comprobación in situ de funcionalidades) y Firefox para testing formal (clases de testing).

El servidor (Tomcat):

Recibe las peticiones del navegador en HTTP, las procesa y devuelve las respuestas, en HTML, CSS y JS.
Está compuesto por:

- El servidor HTTP (Coyote)
- El servidor de Servlet: es una cosa (Catalina)
- Renderizador (Jasper)

La aplicación (servlet):

Una aplicación tiene muchas funcionalidades, las cuales implican un controlador, un servicio, un repositorio, algunas entidades y algunas vistas. En nuestro caso, hemos elegido usar tecnologías como JSP, JPA y Java.

El Servidor de la Base de Datos:

Un servidor de base de datos nos permite almacenar datos en la base de datos y extraerlos de la misma. Estos datos son relacionales, mediante claves primarias y ajenas, atributos e índices.
En la asignatura hemos utilizado concretamente MariaDB, con la cual interaccionamos mediante querys JPQL.

Interacciones

Browser ←→ App server:
El navegador envía una solicitud GET/POST, luego el servidor recibe la solicitud mediante HTTP y responde a través del servidor servlet.
La respuesta del servidor de aplicaciones es un documento HTML creado por el renderizador.

App server ←→ Aplicación:
El servidor servlet envía la petición hacia la aplicación apropiada. Una vez recibida, se procesa usando un controlador, el cual depende de un servidor, que a su vez depende de un repositorio, algunas entidades y vistas.
Una vez procesado, la aplicación envía la respuesta al renderizador para poder crear un documento HTML.

Application ←→ Database Server:
La aplicación envía peticiones para seleccionar, actualizar, borrar o guardar datos en la base de datos mediante un repositorio que hace de intermediario.

Conocimientos de la WIS aplicados a la asignatura

Hemos aprendido que mucho trabajo es repetido constantemente a la hora de desarrollar en SpringBoot; por ello, es necesario crear unas clases auxiliares que encapsulan la máxima cantidad de código posible.

Un ejemplo de estos casos puede apreciarse en los Service y Controller; en estos siempre se suele seguir la misma estructura, por lo que se hace necesario extender un AbstractController o AbstractService que reduzca esa duplicidad. El Acme-Framework nos permite esto, reciclando código y trabajando siempre siguiendo una estructura común de manera sencilla en las distintas clases.

Además, también hemos notado la necesidad de tener una base de datos persistente a la hora de trabajar; a diferencia de Spring, que inicializa la base de datos siempre en memoria, el framework de la asignatura inicializa una base de datos propia en una instancia local de MariaDB. Esto permite ser más ágiles a la hora de iniciar desde cero la aplicación, ya que solo será necesario realizar un populate una única vez y después no sería necesario.

Conclusión

En conclusión, hemos aprendido a reducir el índice de duplicidad de código y a llevar a cabo una correcta estructuración de los distintos Service (Create, Update, Delete, List, Show), así como de los Controller, vistas etc.