PROGRAMACIÓN AVANZADA: Entregable 4

Paula Strimber

Introducción:

Se desarrolló una aplicación para visualizar videos embebidos. En el presente informe se detallarán las principales decisiones tomadas, tanto para la parte de desarrollo, como para la parte de integración y despliegue continuo del código fuente.

Parte I:

A continuación, se detallan las decisiones tomadas con respecto a la arquitectura del proyecto.

En primer lugar, se optó por utilizar una arquitectura en capas donde tenemos tres componentes:

1. Controller: se encarga de manejar las peticiones HTTP y renderizar la vista.
2. Service: se encarga de la lógica de la aplicación (agregar likes, agregar favoritos, eliminar videos, etc).
3. Repository: maneja la persistencia.

Además, el código cuenta con la carpeta “model”, en donde se encuentra la clase video (encapsulando todos los atributos) y el archivo MyApplication.java, siendo este el punto de entrada principal a la aplicación.

Con respecto a la persistencia de la aplicación, para almacenar los vídeos se utiliza un archivo json. El motivo del json se debe a que es más simple de manejar y más fácil de leer.

Por otro lado, para crear la página web se utilizó SpringBoot, y para la interfaz gráfica se utilizó la librería Thymeleaf. Para gestionar el proyecto y las dependencias se utilizó la herramienta Maven.

Las funcionalidades implementadas en la página son:

1. Agregar videos.
2. Eliminar videos.
3. Sumar likes (en este caso la aplicación es utilizada por una persona, pero se puede simular cómo se agregan los likes en caso de soportar múltiples usuarios).
4. Marcar favoritos.
5. Filtrar por favoritos.

Se deja como futura implementación el soporte para múltiples usuarios y la visualización de videos recientemente eliminados. Actualmente, una vez eliminados no se pueden recuperar, es necesario cargarlo nuevamente.

Parte II:

En cuanto a la integración y despliegue continuo, se decidió implementar un flujo de automatización utilizando Jenkins como servidor principal de CI/CD, en conjunto con GitHub y Maven como herramientas complementarias. Esta elección se basó en la facilidad de integración entre dichas tecnologías y en la posibilidad de ejecutar el ciclo completo de compilación, prueba y despliegue de manera local y reproducible.

En primer lugar, se configuró Jenkins de forma local como servidor de integración continua. De esta manera, Jenkins se encarga de orquestar automáticamente las distintas etapas del ciclo de vida de la aplicación.

Para la gestión del código fuente se optó por utilizar GitHub, lo cual permite a Jenkins acceder a la última versión del proyecto, asegurando que cada ejecución del pipeline trabaje sobre una versión actualizada y controlada del código.

Durante la etapa de build, se decidió mantener Maven como herramienta de compilación y empaquetado, tanto en el entorno local como en el pipeline de Jenkins. De esta forma, se garantiza coherencia entre los procesos de desarrollo y los de integración continua.

En la etapa de testing, se integró la ejecución automática de JUnit, con el objetivo de validar las funcionalidades básicas del proyecto antes del despliegue.

Por último, se mantuvo un enfoque incremental de desarrollo, comenzando por una versión básica de la aplicación (“Hola Mundo”) y posteriormente incorporando las funcionalidades y etapas del pipeline de manera progresiva.

Pipelines utilizados:

Se utilizó un pipeline de Jenkins configurado de manera declarativa para automatizar la integración continua y el despliegue de la aplicación. El pipeline está compuesto por las siguientes etapas:

1. Build:

Se ejecuta el comando “mvn clean package -DskipTests”, que compila el proyecto y genera el archivo JAR. Esta etapa permite detectar errores de compilación de manera temprana.

2. Test:

Ejecuta los tests automáticos mediante mvn test. De esta forma se asegura que los cambios recientes no rompan funcionalidades existentes y que el software mantenga su calidad.

3. Deploy Prod:

Se copia el artefacto generado al directorio configurado para producción (playlist-prod) y se ejecuta la aplicación con:

java -jar playlist-0.0.1-SNAPSHOT.jar --server.port=9090
Esto automatiza el despliegue en un entorno separado del de desarrollo, corriendo en un puerto distinto (9090), simulando un ambiente productivo más realista.

4. Post Actions:

En esta sección se publican los reportes JUnit y se archivan los artefactos generados. Esto permite mantener un historial de ejecuciones y facilita el debugging cuando ocurre un error.

Se decidió que el entorno de desarrolló se probará con maven sin el uso de Jenkins.

Técnica de Refactoring:

Durante el análisis del servicio PlaylistService se identificó un code smell de tipo Duplicated Code. La lógica para buscar un video por título se repetía en varios métodos del servicio, como like(), toggleFavorito() y eliminarVideo(). En cada uno de estos métodos se recorría la lista de videos y se comparaba el título manualmente, generando código duplicado y dificultando el mantenimiento.

Esta duplicación aumentaba el riesgo de inconsistencias y hacía más difícil extender la lógica en el futuro (por ejemplo, cambiar la forma de identificar un video). Además, violaba el principio DRY (Don't Repeat Yourself).

Para resolver este problema se aplicó la técnica de refactoring Extract Method, creando un método privado encargado exclusivamente de encapsular la búsqueda de un video por título. Luego, los métodos like() y toggleFavorito() se reescribieron para reutilizar este nuevo método, eliminando la duplicación de lógica y mejorando la claridad.

Conclusión:

Por un lado, este entregable permitió reforzar el conocimiento sobre desarrollo de aplicaciones webs. Además, la incorporación de herramientas de integración y despliegue continuo brindó una visión más completa del ciclo de vida del software en entornos de producción, destacando la importancia de la automatización, las pruebas y la mejora continua del código mediante técnicas de refactoring.

Por otro lado, a nivel general del curso, creo que se logró abarcar muchas áreas de la programación. Si bien los temas que se tocaron fueron independientes unos de otros, creo que eso hizo que el curso sea dinámico y que se haya podido extraer aprendizajes en todas las áreas.