Sofía Dutra

Programación Avanzada

Informe Entregable 4

Introducción

El objetivo de este entregable es poner en práctica los conceptos aprendidos en el curso sobre Integración y deployment continuo, refactoring y code smells. Para esto, se llevó a cabo un proyecto en Java basado en una webapp que gestiona playlists de videos de YouTube con las siguientes funcionalidades:

- Agregar y quitar videos
- Reproducirlos embebidos en la web
- Darles like (Se puede más de una vez)
- Marcar como favorito
- Funcionalidades extra
    - Reordenar los videos
    - Modo oscuro/claro

A su vez, se utilizó Git y GitHub para el control de versiones y Jenkins local con un pipeline que automatiza los pasos de build, test y despliegue.

Instrucciones de Ejecución

1. En una terminal en la carpeta del proyecto ejecutar "./mvnw clean package"
2. Ejecutar
    a. En Windows: ".\scripts\deploy-windows.bat"
    b. En Mac: "./scripts/deploy-mac.sh
3. Acceder a "http://localhost:8080"

Diseño

Decisiones técnicas tomadas:

- Framework: Se eligió Spring Boot como framework por su facilidad y soporte integrado para servidores embebidos y plantillas Thymeleaf.
- Persistencia: Se implementó un repositorio basado en JSOn (playlist.json) para guardar los videos entre ejecuciones sin necesidad de que haya una base de datos completa.

- Frontend: Se usó HTML y CSS para lograr una interfaz simple pero moderna
- Control de versiones: Se utilizó Git y GitHUb como fue recomendado, permitiendo la correcta integración con Jenkins
- Uso de una sola pipeline: Se decidió usar un único pipeline de Jenkins para simplificar el flujo de IC dado que es un proyecto pequeño y con etapas secuenciales y directas, por lo que un solo pipeline es suficiente para automatizar el proceso, aunque podría haber sido interesante probar multi pipeline.

Estructura del proyecto

El proyecto está organizado según el siguiente esquema:

src/

main/

java/com/sofia/playlist/

HomeController.java

PlaylistApplication.java

controller (VideoController.java)

service (PlaylistService.java)

repository (PlaylistRepository.java)

model (Video.java)

resources/

templates (index.html)

static (style.css)

test/java/com/sofia/playlist/ (Tests Unitarios)

scripts/

deploy-windows.bat

deploy-mac.sh

pom.xml

Jenkinsfile

playlist.json

Esta estructura permite separar adecuadamente responsabilidades como el controlador para la vista, el servicio para la lógica de negocio, repositorio para persistencia y modelo para las entidades.

Pipelines utilizados

Se implementó un pipeline en Jenkins compuesto por:

- Checkout: Obtiene la versión más reciente del código desde GitHub
- Build: Ejecuta “mvnw clean package” para compilar el proyecto y generar el archivo .jar
- Tests: Corre todos los tests automáticos con “mvnw test”, asegurando que los cambios no rompan las funcionalidades existentes
- Deploy: Llama al script de deployment correspondiente, ya sea el de Windows o el de Mac para copiar y ejecutar el jar en el entorno
- pollSCM: Cada 2 minutos, Jenkins verifica si hubo cambios en el repositorio y, si los hubo, ejecuta todo el pipeline automáticamente, esto permite integración continua sin intervención manual

Code smell

Sobre el code smell, se realizó a propósito un LongMethod en “PlaylistService.java”, en el método “toEmbedUrl()”, conteniendo una lógica duplicada para extraer el ID de YouTube.

Para el refactoring, la técnica aplicada es Extract Method. En el código quedó la parte original con el code smell, un método refactorizado comentado que sustituye la lógica repetida, cambio que se realiza en la defensa del proyecto, para luego pushear el cambio y mostrar como Jenkins ejecuta el pipeline automáticamente.

Aprendizajes

Este entregable permitió, además de crear una app web funcional en Java, integrar distintas herramientas como Git y GitHub, Jenkins y pipelines de CI/CD. Uno de los aprendizajes más valiosos fue entender como automatizar el flujo de integración y despliegue, desde el push al repositorio hasta que la app se construye, testea y despliega.

Otro aprendizaje valioso fue la importancia de los tests automáticos para garantizar consistencia en el proyecto, además de la utilidad de las técnicas de refactoring y la importancia de detectar code smells para mejorar la calidad y mantenibilidad el código.

En conjunto, los principales aprendizajes fueron:

- Construir un pipeline real con Jenkins
- La utilidad del control de versiones para integrar cambios de forma segura
- La importancia de los tests automáticos
- La importancia de la identificación de code smells para mejorar la calidad del código
- Lo útil que resulta automatizar el flujo de integración y despliegue

Conclusión final del curso

Sobre el curso, me pareció muy interesante, me gustó la modalidad de los entregables porque sentí que fueron una muy buena forma de incorporar lo que dabamos en clase de forma práctica, en lugar de quedarnos solo con la base teórica que a veces sin ponerla en práctica es difícil de entender.

El tema que me resultó más interesante podría ser este, me gustó probar usar Jenkins y ver la automatización de este flujo en lugar de hacer todo manual.

No se me ocurren mejoras, me gustó la modalidad de los entregables, las defensas, también agradezco que en las clases muchas veces teníamos tiempo para dedicarle a los entregables y consultas. Las PPTs también fueron muy útiles para tener una base teórica que estaba completa y siempre podíamos consultar, muchas gracias!