Metodología de Gestión de la Configuración

G6-63

Participantes:

Luis Cerrato Sánchez (Scrum máster y desarrollador)

Carmen Galván López (Desarrollador)

Ezequiel González Macho (Desarrollador)

Antonio Quijano Herrera (Desarrollador)

Juan Salado Jurado (Desarrollador)

Álvaro Úbeda Ruiz (Desarrollador)

1. Introducción

Nos encontramos ante el proyecto Petclinic, que es un sistema que gestiona una clínica de mascotas. Es un proyecto que incluye las funcionalidades básicas que recoge cualquier sistema de información y es fácilmente ampliable, siendo así una gran herramienta en el ámbito didáctico.
En el presente documento, buscamos como grupo al completo, definir una metodología que nos sirva para tener una correcta gestión de la configuración, lo que nos ayudará a obtener un correcto desarrollo del proyecto y a conseguir un buen trabajo en equipo, siempre y cuando se sigan los puntos que se expondrán a continuación.

2. Estándares de programación

2.1. Nomenclatura

El idioma elegido para la escritura del código será el inglés. Será utilizado para el nombrado de todos los elementos, tanto clases, paquetes y variables, como comentarios y demás sentencias.

2.2. Paquetes

Todos los paquetes se escribirán en minúsculas y sin utilizar caracteres especiales ni numéricos.

El paquete principal de la aplicación será: “java.org.springframework.samples.petclinic”.

2.3. Clases

Las clases se nombrarán usando PascalCase.

2.3.1 Entidades

Las entidades se nombrarán con sustantivos y a ser posible con una sola palabra para hacerlos más legibles.

2.3.2 Repositorios

Los repositorios se nombrarán como ”*nombreDeLaEntidad*Repository”.

2.3.3 Servicios

Los repositorios se nombrarán como ”*nombreDeLaEntidad*Service”.

2.3.4 Controladores

Los repositorios se nombrarán como ”*nombreDeLaEntidad*Controller”.

2.3.4 Validator, Formatter y otros

Los demás tipos de clases relacionadas con la entidad se nombrarán como ”*nombreDeLaEntidad*Formatter”, ”*nombreDeLaEntidad*Validator” etc.

2.4. Métodos

Los métodos se nombrarán usando camelCase.

2.5. Variables

Los métodos se nombrarán usando camelCase.

2.6. Comentarios

Se evitará en la medida de lo posible el uso de comentarios, ya que tienden a complicar el código y confundir al desarrollador. Se intentará usar en cambio un código que sea

autoexplicativo. En caso de ser imprescindible ,el comentario será utilizado para dar información adicional.

2.6. Archivos de estilos

Se usarán archivos Less para la definición de los estilos.

En la medida de lo posible se intentará usar los archivos ya creados como petclinic (El principal), typography y responsive, para evitar crear archivos innecesarios. En cambio se intentarán modificar las variables proporcionadas por bootstrap en el archivo principal, además de usar la paleta de colores y las fuentes escogidas.

En el caso de necesitar nuevos estilos se crearán archivos nuevos, y se intentarán usar variables definidas en el archivo principal para tener centralizados todos los estilos y hacerlos más sencillos de modificar.

2.6. Archivos JSP

Se intentarán usar las etiquetas proporcionadas por spring.

Se intentarán usar archivos .tag para crear componentes reutilizables para las distintas vistas de la aplicación.

3. Metodología de mensajes commit

El mensaje de cada uno de los commit debe seguir la siguiente estructura:

<tipo>[cobertura opcional]: <descripcion>
[cuerpo opcional]
[pie de página opcional]

- **tipo**: fix (arreglando un bug), feat (añadiendo una nueva funcionalidad), docs (editando o añadiendo documentación), refactor, test, build o chore.
- **cobertura opcional**: tarea asociada, o parte del código a la que afecta.
- **descripción**: pequeña descripción sobre el cambio realizado, y debe explicar el qué y el por qué frente al cómo. Se debe escribir en modo imperativo.
- **cuerpo opcional**: descripción detallada. No es necesario en la mayoría de los casos
- **pie de página opcional**: puede contener sentencias de automatización como “Closes #issue” o “Fixes #issue”

4. Estructura del repositorio y ramas

Para el desarrollo del proyecto usaremos la política de ramas Git Flow. Esta define dos ramas principales, la rama master y la rama develop. Ambas tienen un tiempo de vida infinito.

La rama **main** es la rama principal, apunta a la última versión en producción. Esta rama será desplegada automáticamente.

La rama **develop** apunta a la última versión de desarrollo. Servirá para producir la próxima release.

También se crearán ramas de tipo **feature**. Se creará una por cada funcionalidad que será implementada en la próxima release. Su nombrado debe seguir el siguiente formato: “feature/#XXX-TaskYYY”, donde XXX indica el número del issue en github asociado a la tarea

número YYY. Por ejemplo, para la tarea 31 asociada al issue #092 en el tablero del proyecto, la rama deberá nombrarse como “feature/#092-Task031”.

Así mismo, se crearán ramas de tipo **release**. Se creará una por cada versión mayor de la aplicación, en nuestro caso una por cada sprint. En esta rama se realizan pequeños cambios, como, por ejemplo, soluciones a errores, cambios de última hora, etc. Su nombrado debe seguir el siguiente formato: “release/X.0.0”, donde X.0.0 indica la versión de la release. Por ejemplo, para la versión 1.0.0, la rama deberá nombrarse “release/1.0.0”.

En último lugar, también utilizaremos una rama **hotfix.** Esta rama se utilizará para arreglar bugs críticos “en caliente” como su propio nombre indica; en la propia versión de producción.

5. Política de versionado

Como grupo, hemos acordado seguir la siguiente política de versionado, para las diferentes versiones del proyecto durante todo su desarrollo:

- En primer lugar, para cada entregable del proyecto se incrementará la versión en una unidad.
- El encargado de realizar esta tarea será el Scrum Master.
- Durante un mismo entregable se podrá incrementar la versión en menos de una unidad, en caso de que la aplicación una vez desplegada en producción sea modificada para resolver errores. Ejemplo, pasar de la versión 0.1 a la versión 0.2.

6. Definición de hecho

Nuestro grupo considerará una tarea como hecha cuando se cumplan las siguientes condiciones:

- Se haya finalizado la implementación de la tarea (o realización del documento, si se trata de una tarea de documentación).
- Se haya probado el correcto funcionamiento de la aplicación tras la implementación añadida.
- Se haya comprobado que todos los tests implementados se pasan satisfactoriamente.
- Se hayan llevado a cabo las reviews por pareja de otros dos miembros ajenos a la implementación de la tarea.
- Se hayan realizado los merges que correspondan con la tarea.
- Se haya movido la tarea en el project board a la sección de “Done”

7. Conclusión

Para concluir, pretendemos que con la metodología expuesta en el presente documento podamos tener un correcto desarrollo del proyecto y realizando un buen trabajo en equipo, lo cual consideramos que es un punto esencial en el desarrollo software, estando todos los integrantes del grupo de acuerdo con todos los puntos anteriores y comprometiéndose a cumplirlos.