



[G4-43]

TECHNICAL REPORT A.2.3.2

Periodo de Reporte: – [11/03] - [04/04], [2021]

Fecha de Emisión: 04/04, 2021

Fecha de inicio y fin del Sprint: [11/03, 2021] a [04/04, 2021]

Autores:

- Fernando Andrés Galindo
- Matthew Bwye Lera
- Fernando José González García
- Manuel Guerra Cordon
- Francisco Murillo Prior
- Jesús Aparicio Ortiz



- **SUMARIO**

En este informe cubriremos qué gestión de la configuración hemos empleado para nuestro sprint, mencionando la estructura del repositorio, la estrategia de ramificación que hemos empleado y la política de versionado que seguimos.

- **PRÓLOGO**

Este informe ha sido redactado para el primer sprint de un proyecto para la asignatura de PSG2, en la Universidad de Sevilla. El objetivo de la asignatura, y en general del trabajo que estamos realizando, es aprender a gestionarnos como equipo, utilizando las herramientas y técnicas adecuadas.

Con el objetivo anterior en mente, trabajaremos sobre un proyecto llamado Petclinic, utilizando el framework Spring y el lenguaje de programación Java. Petclinic es una página web para una clínica veterinaria, cuyo objetivo es mejorar la gestión de la clínica en general, almacenando información de mascotas, veterinarios y dueños, entre otros.

• TABLA DE CONTENIDO

TECHNICAL REPORT A.2.3.2	1
• SUMARIO	2
• PRÓLOGO.....	2
• TABLA DE CONTENIDO.....	3
1. INTRODUCCIÓN.....	3
2. CUERPO.....	4
2.1 Estructura del repositorio y ramas predeterminadas	4
2.2 Estrategia de branching, basada en Git Flow y revisión por pares.....	4
2.2.1 Feature branch.....	5
2.2.2 Release branch	5
2.2.3 Resolviendo bugs durante la producción	6
2.3 Política de versionado	6
3 CONCLUSIÓN	6
4 ANEXOS	7
4.1 ANEXO I: Referencias.....	7

1. INTRODUCCIÓN

Para la gestión de la configuración, hemos seguido las pautas dictadas por los profesores y el contenido de la asignatura.

La estrategia de ramificación que hemos empleado es GitFlow, cubriremos cómo hemos empleado esta estrategia, y también se especificará sin entrar en detalles cómo funciona cada uno de los apartados de GitFlow, para dar soporte a las explicaciones de cómo hemos aplicado cada una de las características.

Hemos aplicado la política de versionado que hemos visto en las transparencias de la asignatura. Explicaremos por qué utilizamos esa política de versionado, y sin entrar en detalles comentaremos sus características.

2. CUERPO

2.1 Estructura del repositorio y ramas predeterminadas

Explica el estado de las ramas desde la cual empezamos a aplicar la estrategia.

En nuestro repositorio tenemos una versión del proyecto Petclinic, al que le debemos aplicar cambios y mejoras. Además, se ha organizado por carpetas de actas la distinta documentación elaborada en este sprint, así como los distintos daily scrums, sprint reviews, sprint plannings y sprint retrospectives realizados. Por último, partiendo únicamente de una rama master, a continuación, explicaremos qué estrategia de branching hemos seguido para mejorar la eficacia del trabajo.

2.2 Estrategia de branching, basada en Git Flow y revisión por pares

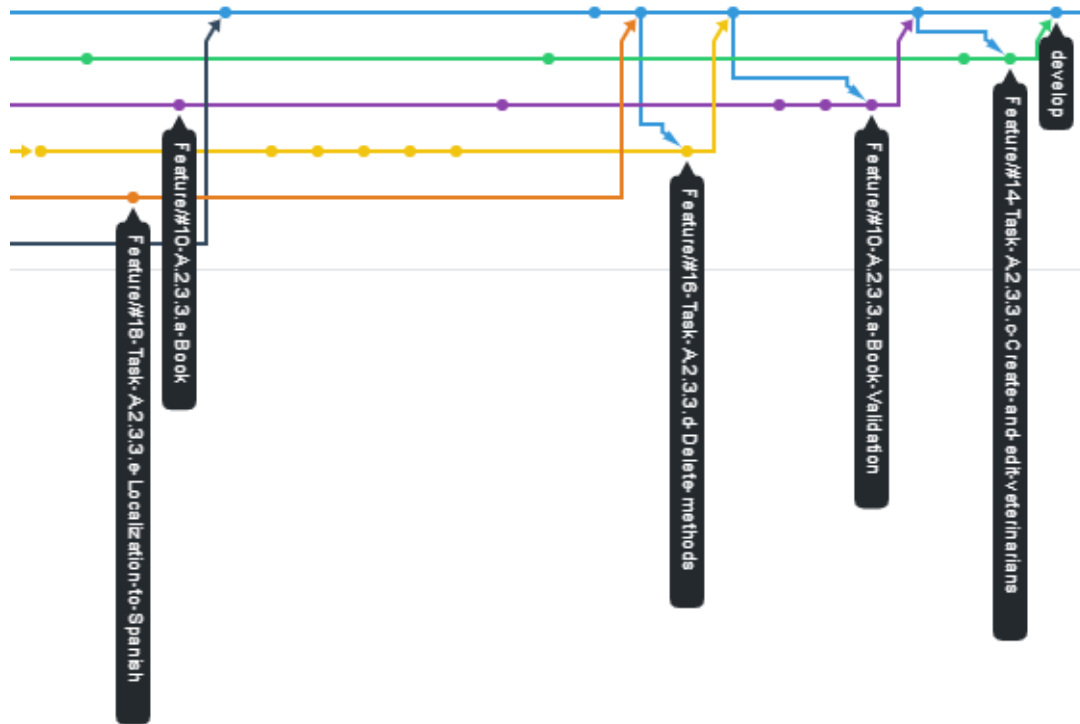
Explica en qué consiste, a grandes rasgos, la estrategia implementada.

Para aplicar nuestra estrategia de branching, partimos de una rama master, y de ahí creamos distintas ramas para distintas funcionalidades y necesidades, donde mantenemos una estructura que reduce el número de errores y ayuda a que el trabajo sea más independiente. Las ramas creadas son las ramas de develop, feature y release:

- La rama de develop es desde la que se generarán las ramas a partir de las que se irán implementando las funcionalidades.
- Las ramas de feature son ramas designadas para muchas tareas, donde algunas tareas, como tareas de revisión por pares, no tienen rama propia. Antes de juntar estas ramas, se realiza una revisión por pares de su funcionalidad, para asegurarnos de no juntar ramas que no funcionen, que podrían potencialmente dañar la funcionalidad de la aplicación.
- La rama de release es la rama en la que juntaremos todo el progreso unificado.
- A continuación, explicamos en más detalle el significado de las distintas ramas.

2.2.1 Feature branch

Aquí explicamos el funcionamiento de las feature branch y cómo las hemos llevado a cabo.



Para cada tarea de funcionalidad hemos realizado una rama nueva de feature, para que cada miembro pudiera hacer su trabajo de forma independiente de las tareas de los demás miembros, evitando así conflictos durante el trabajo. Las tareas que no tenían rama propia eran las que no estaban relacionadas con realizar código (tareas de documentación o reuniones, por ejemplo) o tareas de comprobación de código.

Cada una de estas ramas se crearon desde la rama develop (que a su vez fue creada de la rama master). Una vez se haya terminado la tarea de cada feature y ha recibido su correspondiente comprobación y ha sido considerado terminado, se hace merge de esa rama con la de develop. Si hay tareas que son dependientes de otras, se crean las ramas para esas tareas después de haberse terminado las tareas de las que depende.

2.2.2 Release branch

Aquí explicamos qué es la release branch.

La rama de release es una rama que se crea a partir de la rama de develop una vez se han terminado todas las tareas (es decir, una vez se ha hecho merge de todas las ramas de feature sobre la de develop). Esta rama es la que determina la versión del producto.

2.2.3 Resolviendo bugs durante la producción

En esta sección explicamos cómo hemos resuelto los bugs surgidos, y qué hacer en casos extremos.



Sobre la misma rama de release, si se detectan bugs o hay mejoras que aplicar, se realizan los cambios y se commitean directamente sobre ella, sin hacer ramas adicionales.

Si los cambios a realizar son muy grandes, se haría una release nueva, con su versión nueva. Para ello, se crearía una rama de develop a partir de esta rama de release, sobre la que de nuevo se crearían ramas de features, y una vez se terminasen las tareas y se hiciera merge de ellas sobre develop, se daría lugar a una nueva rama de release.

2.3 Política de versionado

Descripción de la política de versionado que se ha seguido.

Hemos empleado la política de versionado recomendada por la asignatura, el versionado semántico, que nos permite realizar distintas versiones según los cambios que se han realizado, sean cambios mayores (no compatible con la API), cambios menores (cuando se añaden nuevas funcionalidades a la aplicación siendo compatible con la API) y también, parches (para solucionar bugs).

3 CONCLUSIÓN

En este documento hemos cubierto el antes y el después de la estructura del repositorio, tras aplicar la estrategia de branching Git Flow.

- En un principio, todas las funcionalidades estaban sobre la rama master.
- Al aplicar gitflow, creamos una serie de ramas: rama de develop, ramas de feature y rama de release.
- La rama de develop se creó a partir de la rama de master.
- A partir de la rama de develop, se crearon las ramas de feature. Sobre estas ramas se creaban las funcionalidades. Cuando una funcionalidad está terminada y comprobada, se unifica en la rama de develop junto a las demás y, si fuese necesario, se crean más ramas de feature.
- La rama de release se crea una vez están implementadas todas las funcionalidades, y

representa la versión del producto una vez acabadas las tareas.

- Se pueden solucionar bugs sobre la misma rama de release, pero para cambios mayores hay que volver a trabajar sobre la rama de develop y crear más ramas de feature; esto daría lugar a una nueva versión de release.

4 ANEXOS

4.1 ANEXO I: Referencias

[Technical Report: What is it & How to Write it? \(Steps & Structure Included\) \(bit.ai\)](#)

https://www.itu.dk/people/sestoft/itu/writing_reports.pdf

https://templatelab.com/technical-reports/#google_vignette

<https://www.sample.net/business/analytics/technical-reports/>