2021

METODOLOGÍA DE GESTIÓN DE LA CONFIGURACIÓN

Tutor del proyecto:

JOSÉ ANTONIO PAREJO MAESTRE

Realizado por el grupo G2-25:

JESÚS BARBA SIGÜENZA

DANIEL CASTROVIEJO NARANJO

MARCELINO GONZÁLEZ VALLE

FERNANDO RABASCO LEDESMA

VICENTE SORIA VÁZQUEZ

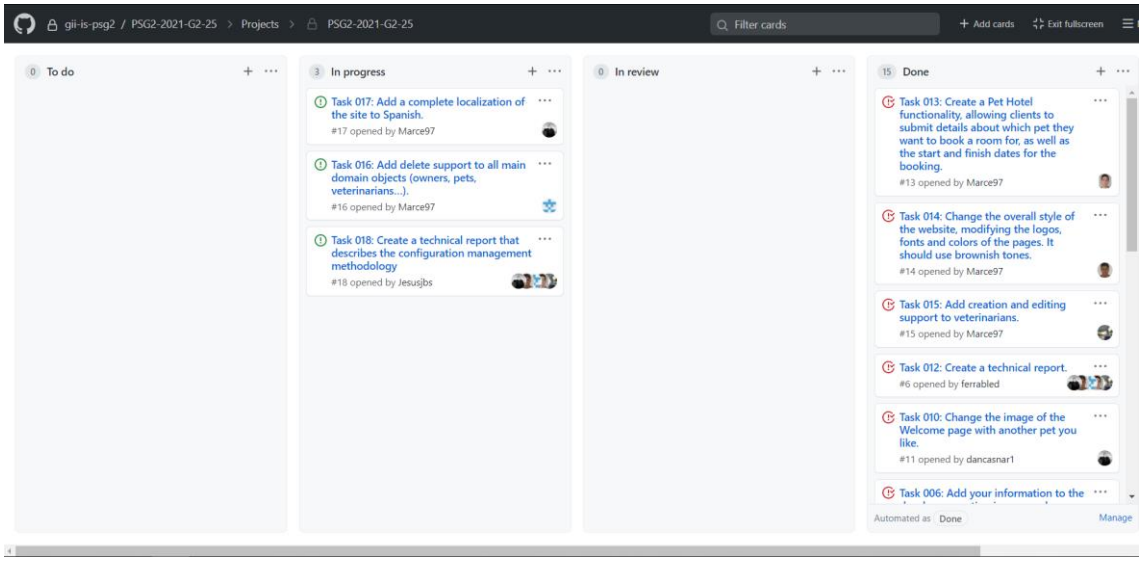
ÍNDICE

1. Estructura del repositorio y ramas principales:

La estructura del repositorio es la estructura habitual en un proyecto de una aplicación web, con la diferencia de que se incluye en el proyecto una carpeta documentation dentro de resources, donde se puede ver las definiciones de “hecho”, las actas de cada Daily Scrum y los informes técnicos.

El repositorio posee un tablero Kanban con las columnas “To do”, “In progress”, “In review” y “Done”. Estas columnas nos serán útiles para controlar las tareas que se realizan en el proyecto.

La columna “To do” es en la que se encuentran las tareas que se han creado y asignado a un miembro, y que aún no se han empezado. “In progress” es la que contiene las tareas que se están desarrollando, mientras que “In review” sólo contiene las que tras “terminarlas” están pendiente de ser revisadas por otro miembro del equipo. Finalmente, “Done” únicamente posee las tareas que se están terminadas, con una aprobación tras su revisión.



1.- Tablero Kanban realizado con GitHub

Por otra parte, usaremos dos ramas principales, master y develop.

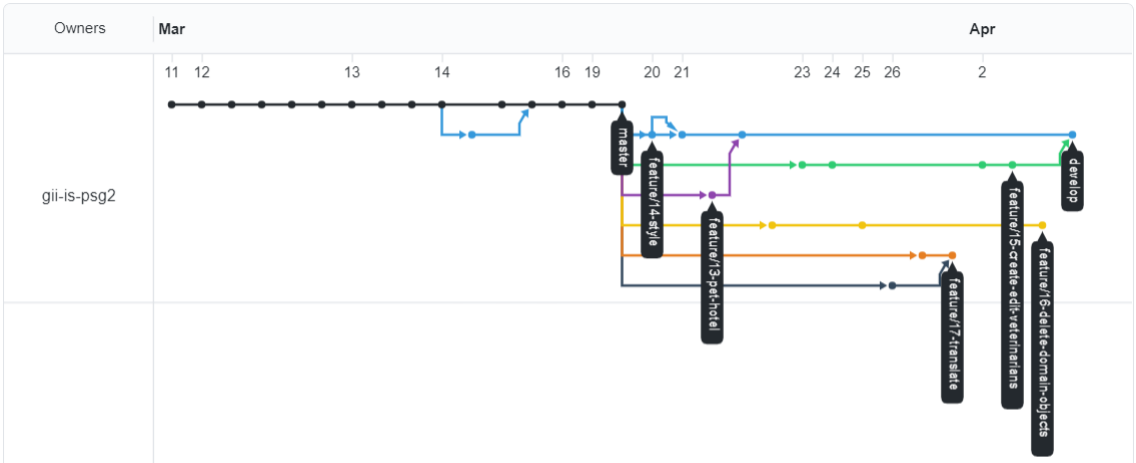
Master será la rama principal, donde se subirán los releases finalizados y listos para pasar a producción. Mientras que develop será la rama más usada, pues es donde se unirán todas las funcionalidades que se han ido desarrollando en las distintas ramas, además de servir para la creación de releases.

2. Estrategia de ramificación:

Seguiremos la estrategia de ramificación de Git Flow donde encontraremos dos ramas principales nombradas como master y develop, además tendremos una rama por cada tarea que seguirá la siguiente sintaxis “feature/ XX-tag”. Ya hemos comentado anteriormente las dos ramas principales, por lo tanto, nos centraremos en explicar que esas ramas features estarán asociadas a cada una de las tareas asignadas para cada Sprint. Antes de lanzar nuestro proyecto, nos encontraremos con la rama Release donde arreglaremos errores tales como la adición de metadatos. También tendremos la rama “hotfix” donde, si detectamos un error de última hora en nuestra rama master, lo arreglaremos en esta rama para luego crear una nueva release con los cambios de corrección en master.

2.1 Desarrollo de ramas de funcionalidades:

La creación de una rama “feature” vendrá dada por una tarea, es decir, se creará tantas ramas de funcionalidad como tareas de desarrollo haya que realizar en el sprint backlog. Estas ramas, como ya se ha indicado anteriormente, siguen las sintaxis “feature/XX-tag” donde XX indica el número de issue correspondiente a la tarea en GitHub, y tag nombre corto que indique la funcionalidad de la rama; en caso de que el tag esté formado por varias palabras, estas se separarán por guiones. Estas ramas provienen de la rama develop como se puede ver a continuación:



2.- Gráfica de las ramas en un estado en desarrollo del proyecto

Además, el cierre de estas ramas, tras acabar de desarrollar la funcionalidad correspondiente, es de nuevo a la rama develop como se muestra en la imagen anterior con las ramas “feature/14-style”, “feature/15-create-edit.veterinarians” y “feature/13-pet-hotel”, de color azul, verde y morado respectivamente.

En caso de que haya ramas que necesiten de otras como ocurre con la rama “feature/16-delete-domain-objects” y la rama “feature/17-translate” que dependen de que otras estén terminadas para poder terminar ellas, lo que se hará será lo siguiente:

cuando las ramas de las que dependen se encuentren subidas a develop, las dependientes deberán de hacer un merge de la rama develop a la rama de la correspondiente feature. Ahora, la rama dependiente posee las actualizaciones de las otras ramas y podrá terminar su funcionalidad. Tras la terminación y siguiendo con la definición de “Done”, ésta procederá a realizar un pull request para su unión con la rama develop, esperando la aprobación de la revisión por parte de otro miembro, dando lugar así a la posible unión, por parte del solicitante, de la rama en cuestión con la rama develop.

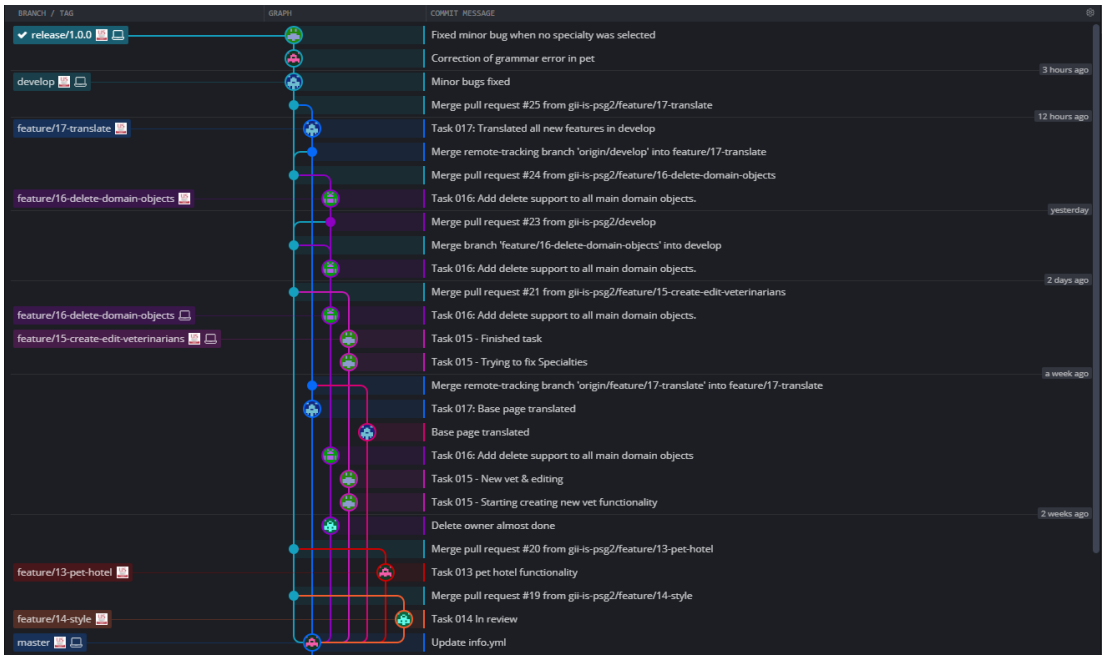
2.2 Preparar lanzamientos:

A la hora de realizar un lanzamiento, se debe de crear una rama “release”. Esta rama release se crea tras estar todas las ramas de funcionalidades finalizadas y unidas a la rama develop y seguirá la sintaxis “release/X.Y.Z”, siguiendo la política de versionado que se encuentra en el punto 3 del informe. La función principal de la rama release es poder lanzar una versión preparada para pasar a producción, es decir, a la rama master.

Aunque esta es la función principal de la rama release, no es la única. Sobre esta rama se puede realizar algunas correcciones de bugs que se encuentren a última hora, y que se deben de corregir antes de pasar la versión a producción en master.

Cuando la rama esté en una versión final lista para pasar a producción, se hará un merge de la rama release a la rama develop, y otro a master, estos merge en ningún caso harán uso de un rebase, sino que serán a través de un commit merge.

Finalmente, cuando la rama master posea la versión final de release, se creará un tag que seguirá la política de versionado.



3.- Estado final de la rama release v 1.0.0

2.3 Corrección de errores en producción:

La corrección de errores en producción se llevará a cabo en una rama llamada “hotfix” y que será creada para esta función en caso de ser necesaria. Al encontrarse un error en producción, indica que el error está en la rama master. Por tanto, para solucionarlo, se creará la rama hotfix y en ella, se realizarán toda la corrección de errores que se encuentren. Tras acabar de solucionar los errores correspondientes se procederá, siguiendo la definición de “Done”, a realizar un pull request a la rama master y otro a la rama develop. Y finalmente, tras ser aprobados por algún miembro del equipo, se llevarán a cabo los correspondientes merges de las ramas solicitadas.

Como se ha mencionado anteriormente, esta rama sólo se creará si los errores ya se encuentran en la rama master, en caso contrario se solucionarán en las propias ramas de “feature” o en la “release” si el error se encuentra a una altura más avanzada del proyecto, como es el caso del último commit realizado, un error corregido en la rama “release”.

3. Política de versionado:

Durante el proyecto se llevará a cabo la utilización de una política de versionado basado en tags y que se aplicará a las ramas release. Esta política de versionado tiene la siguiente sintaxis: X.Y.Z, donde X indicará en nuestro caso que se ha producido una release final, en la que el sprint backlog está terminado; Y indicará que se ha lanzado una versión basada en la release anterior y que incluye una nueva funcionalidad o mejora y que hace que la versión anterior perteneciente al mismo sprint backlog quede anticuada, aunque se pueda seguir usando; y Z indicará que se ha lanzado una nueva versión que corrige errores menores sobre la versión anterior del mismo sprint backlog.

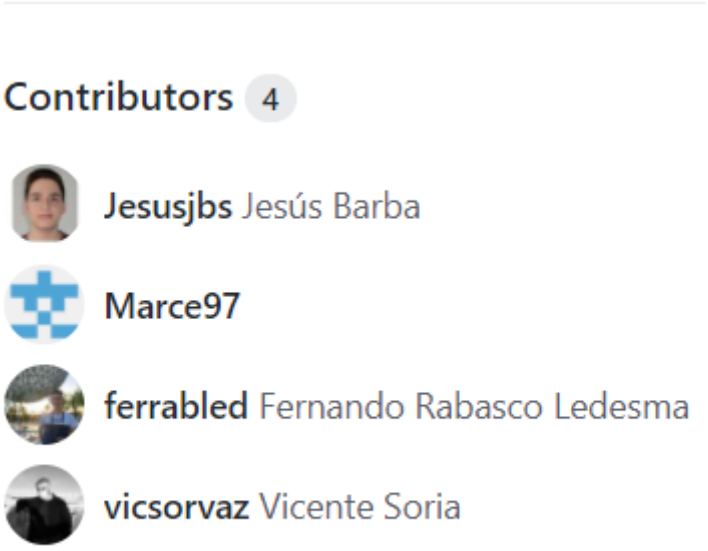
Para que se produzca un incremento en X, debe de haber un nuevo release correspondiente a un sprint backlog distinto al que ya existía. Para que se produzca un incremento en Y, debe de haberse producido un incremento en X anteriormente (además, la release que ha hecho aumentar X debe corresponder al mismo sprint que Y) y se debe de haber añadido alguna característica no implementada o algún cambio sustancial respecto al release anterior. Finalmente, para que se produzca un incremento en Z, debe de haberse producido un incremento en X anteriormente (además, la release que ha hecho aumentar X debe corresponder al mismo sprint que Z) y se debe de haber corregido algún error menor o bug de poca importancia sobre la versión anterior del mismo sprint backlog.

Cualquier incremento en X, hará que, Y y Z se pongan a 0, mientras que un incremento en Y, sólo hará que Z se ponga a 0. En cualquier caso, un incremento en Y no influirá en X y un incremento en Z no influirá ni X ni en Y.

4. Otros comentarios:

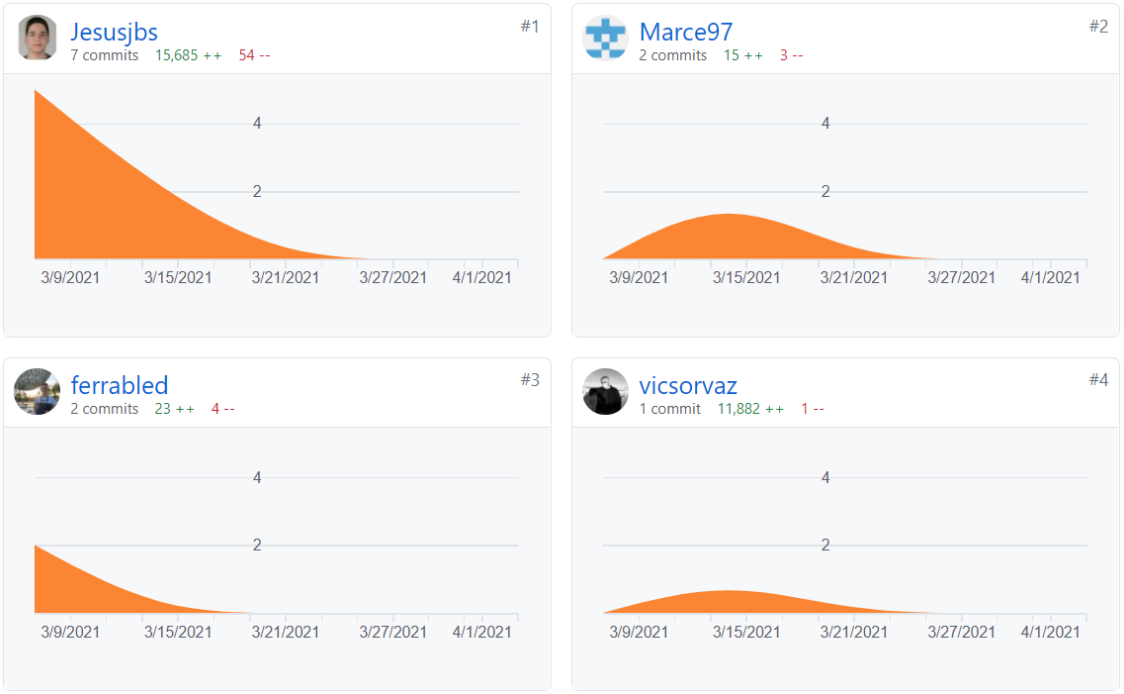
Uno de nuestros integrantes (Daniel Castroviejo Naranjo) tiene problemas con el Github, los cuales son los siguientes:

1- No aparece como contribuidor en el proyecto.



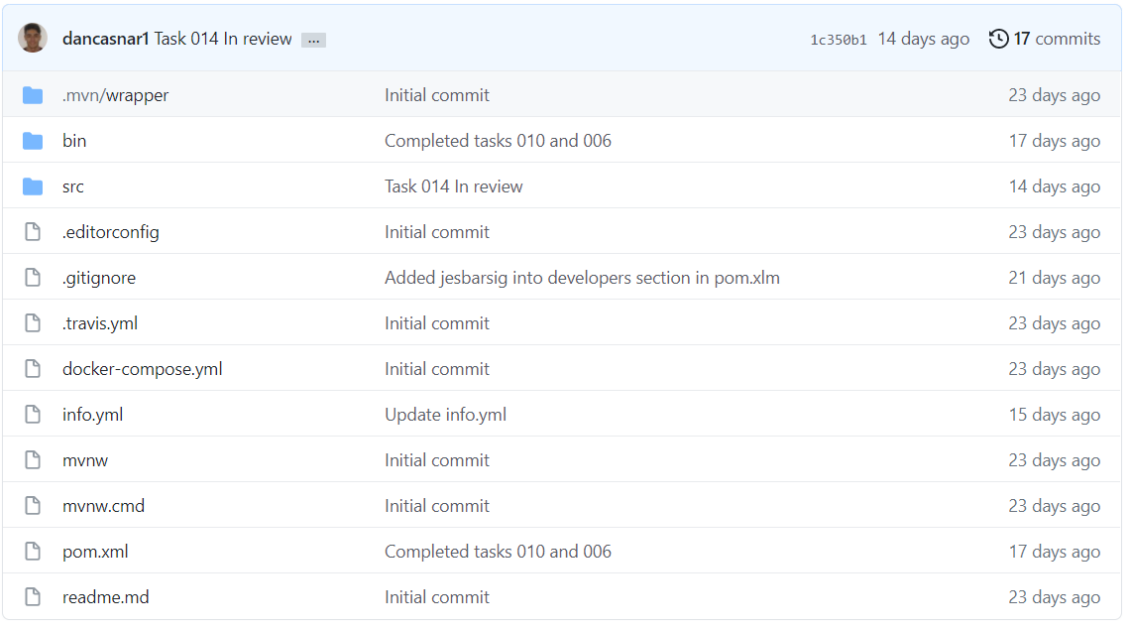
4.- Contribuidores que aparecen en el proyecto

2- Al no aparecer como contribuidor, en la parte de Insights no aparecen sus commits reflejados.

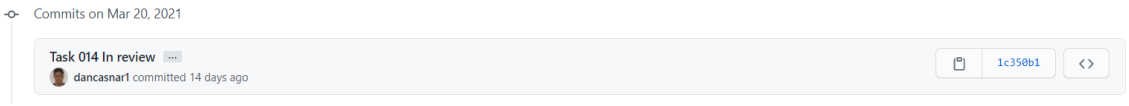


5.- Commits realizados por los miembros del equipo

Siguiendo con el punto 2 de este problema, vemos como sus commits se han realizado correctamente con su usuario asociado.



dancasnar1 Task 014 In review ...	1c350b1 14 days ago	17 commits
.mvn/wrapper	Initial commit	23 days ago
bin	Completed tasks 010 and 006	17 days ago
src	Task 014 In review	14 days ago
.editorconfig	Initial commit	23 days ago
.gitignore	Added jesbarsig into developers section in pom.xlm	21 days ago
.travis.yml	Initial commit	23 days ago
docker-compose.yml	Initial commit	23 days ago
info.yml	Update info.yml	15 days ago
mvnw	Initial commit	23 days ago
mvnw.cmd	Initial commit	23 days ago
pom.xml	Completed tasks 010 and 006	17 days ago
readme.md	Initial commit	23 days ago



Para dejar constancia de estos problemas, decidimos añadirlos a este apartado del documento.