Documentación Git

Para el desarrollo del código de nuestra aplicación, una vez aclarados conceptos básicos como la estructura en microservicios y el stack de desarrollo, decidimos utilizar git en conjunto con GitHub para el control de versiones y la compartición de código.

Inicialmente buscamos un planteamiento muy sencillo, un repositorio común con todo el código al que todos teníamos acceso. Dentro de este repositorio, que estaba ya creado con anterioridad para la documentación de las primeras fases del proyecto, se crearon una carpeta para backend y otra para frontend.

De cara a contribuir al código, el equipo de front-end decidió organizarse con forks individuales de cada miembro del equipo que irían haciendo pull requests al repositorio principal para mantener y gestionar los cambios de forma centralizada.

Sin embargo rápidamente se vio que la organización era insuficiente, el hecho de tener dos equipos teóricamente independientes modificando el mismo repositorio lo hacía muy difícil ya que para tener el repositorio local actualizado no era suficiente con estar pendiente a los cambios del subgrupo correspondiente sino también a los del otro. Siendo un grupo de más de diez personas, un repositorio único era insuficiente.

Entre las soluciones propuestas se contó con poseer ramas separadas para cada subgrupo (rama backend y rama frontend) o crear forks separados para cada parte.

Entonces se adoptó la opción de hacer un fork para el frontend con React y un fork para backend con Java. Así cada subgrupo trabajaría en un repositorio diferenciado y se solventaron los problemas previos. De esta forma una vez terminado el desarrollo se podrían hacer pull requests al repositorio inicial, centralizado, con las distintas releases de cada parte.

Pese a ello y debido a que no eran prácticas del todo correctas a nivel de permisos, organización, demasiados forks entre otras cosas, el desarrollador Abel decidió dedicar tiempo del sexto sprint a investigar cuáles eran las prácticas más profesionales de cara a un proyecto del estilo y aplicarlas, creando toda la nueva infraestructura de git y github y encargándose de migrar todo el progreso hasta entonces, buscando abstraer este proceso y que fuera transparente para el resto de miembros del grupo. De esta forma se llegó a la organización que hemos mantenido hasta el final del desarrollo y que explicamos a continuación:

En primer lugar, definitivamente la mejor opción era tener dos repositorios por separado. Sin embargo a algunos compañeros no les pareció una buena opción por dar la sensación de ser completamente independientes cuando ambos repositorios estaban a manos del mismo equipo (equipo Clarke en general). Por ello la solución fue crear una GitHub Organisation, perfiles de github no personales sino institucionales que permiten la gestión de miembros, equipos y repositorios en conjunto. Así es que se creó la organización CudeClarke.

Una vez dentro de la organización se crearon tres repositorios independientes:

- Por un lado el repositorio ClarkeAPI destinado al desarrollo de la api de backend con java
- Por otro lado el repositorio ClarkeWeb con el frontend con React
- Finalmente el repositorio ClarkeDocs con toda la documentación del principio del proyecto, separada de los repositorios de código propiamente dichos.

Entonces se migraron los avances hasta entonces a los nuevos repositorios, dejando los antiguos aún operativos para los desarrolladores mientras se terminaba de configurar la organización.

Una vez añadidos los repositorios era necesario añadir a los miembros del equipo y organizarlos en sub-equipos, en lo que se conoce como GitHub Teams. Se crearon entonces tres equipos: frontend, backend y admins. En cada equipo estarían los miembros del subgrupo correspondiente y en admins habría un miembro representante de cada subgrupo y el SM de ese momento. Este último equipo sería el responsable de la organización del código y estaría formado por aquellos desarrolladores más experimentados de cada parte del proyecto.

Tras añadir los equipos y proporcionar el acceso era necesario crear un workflow adecuado y lo más cercano a un entorno profesional, dejando atrás la organización en forks unipersonales para aportar al proyecto central.

El workflow ideado era el siguiente: por un lado sólo aquellos miembros de un subequipo podían aportar código en ese subrepositorio (es decir sólo los miembros de front pueden aportar en front). Por otra parte para hacer aportaciones controladas se crearían pequeñas ramas con pequeños cambios que deberían ser subidas al repositorio principal y luego hacer una pull request a la rama correspondiente.

Además en cada repositorio se decidió tener dos ramas: por un lado la rama main con la última versión estable de la aplicación y la rama develop sobre la cual se trabajaría en el día a día, por lo cual las pull requests comentadas con anterioridad se harían siempre a develop.

Debido a que gran parte de los miembros del equipo no estaba acostumbrada a trabajar con git ni con github se tomaron dos decisiones fundamentales que fueron de gran ayuda más adelante durante el desarrollo:

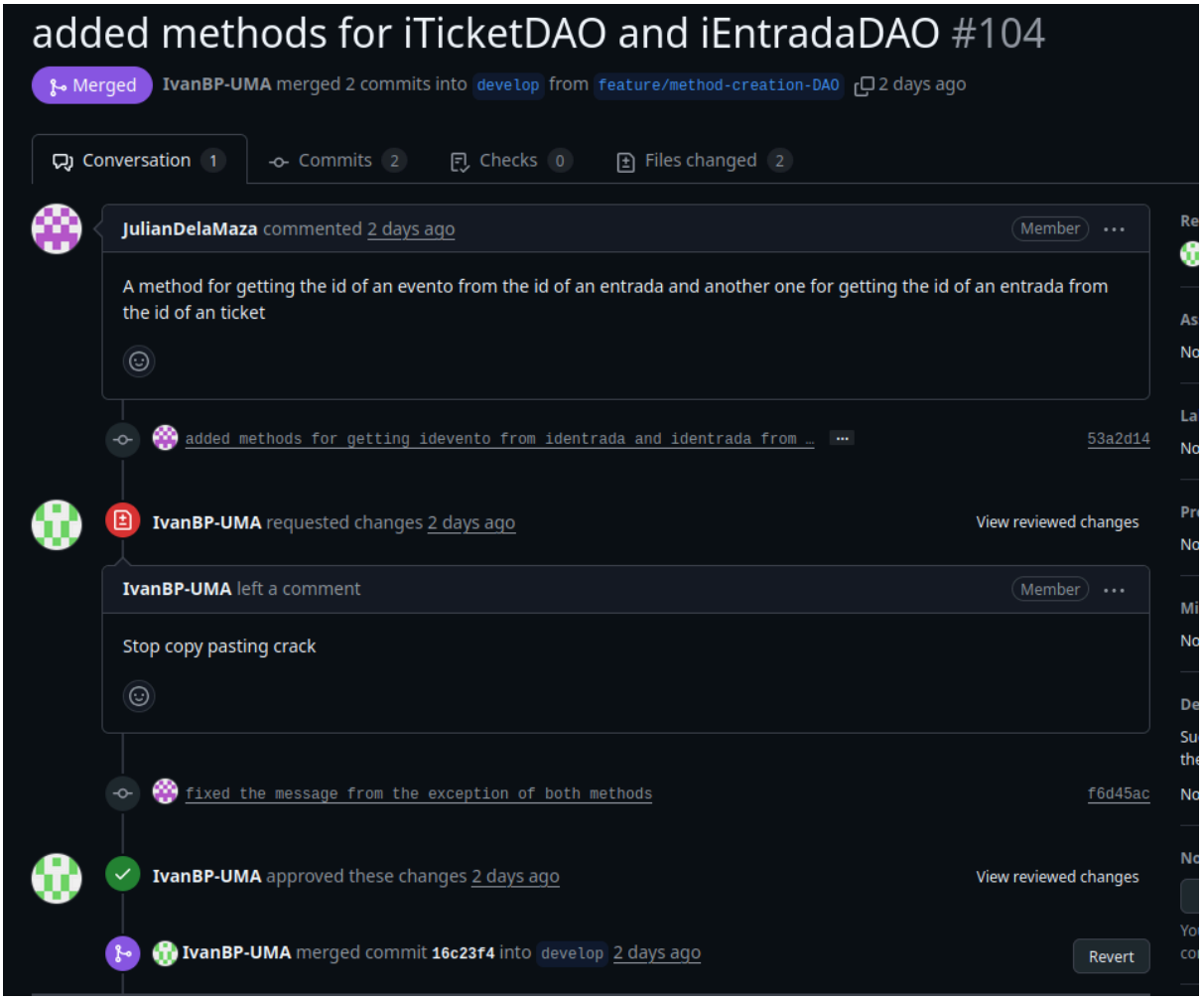
1. En primer lugar se creó un documento CONTRIBUTING.md con una lista de contribución paso a paso, explicando cómo bajar el repo, clonarlo, crear ramas, nombrar ramas, el workflow básico, cómo subir las ramas y cómo crear una pull request. Además en el documento se explica toda la organización del proyecto y la justificación de ésta. Gracias a este documento que fue enviado por los medios de comunicación del equipo, cada miembro pese a olvidar algún paso del proceso de aportación de código en cualquier momento podía acudir al propio documento y ver el paso a paso correspondiente.
2. Por otro lado para evitar errores (subir algo sin querer a main o a develop de forma directa, subir código erróneo sin hacer pull request, borrar archivos, cambiar configuraciones) se decidió crear reglas muy estrictas alrededor del repositorio web

en GitHub. Para ello se utilizaron las GitHub Rulesets (antes branch protection rules). Entre las reglas se encontraba la completa prohibición de subir nada a main a menos que fuese revisado por dos administradores a través de una pull request, lo mismo pero para develop con una sola revisión, la imposibilidad de autorevisar código o la capacidad de cambiar la configuración de estas mismas reglas restringida a dos miembros del equipo completo.

Como se ha mencionado estas medidas tuvieron un éxito rotundo. Varias veces miembros del equipo tuvieron despistes como intentar subir directamente a las ramas protegidas, intentar hacer pull request de la rama de trabajo (por ejemplo fix/button-accesible) a main cuando debería ser a develop o cientas de veces que se simplificaba el trabajo pudiendo volver en cualquier momento al archivo de contribución para poder colaborar todos de una forma común y organizada.

Además de esto, la obligatoriedad de contribuir a través de pull requests permitía la detección de fallos tempranos o evidentes así como una forma sencilla para el scrum master de seguir día a día los avances en el proyecto. Además en caso de fallos en la pr se podía usar la conversación para requerir cambios al desarrollador o recalcar aspectos a refactorizar o mejorar más adelante (deuda técnica). En general permitía un tracking impoluto del proyecto.

Vemos aquí un ejemplo de una pr a main en lugar de a develop que pudo ser rechazada a tiempo.



O aquí otro ejemplo donde se pudieron usar las conversaciones sobre las pull requests para sugerir cambios al desarrollador antes de subirlos a la rama.

En añadido a todo lo anterior github ofrece métricas de aportaciones y avances que permitían un seguimiento y recuento de las aportaciones de cada miembro del equipo.

A partir de entonces el desarrollo pudo seguir con facilidad hasta el último día del proyecto sin necesidad de realizar más cambios en la organización del repositorio.

El enlace a la organización para más detalles:
https://github.com/CudeClarke