Políticas de desarrollo

Arriaza Arriaza, Daniel	**Morato Navarro, Juan Carlos**
Barrera García, Ismael	**Morato Navarro, Olegario**
Calero López, Marina	**Robles Russo, Eduardo**
Galeano de Paz, Guillermo	**Roldán García, Miguel Ángel**
González Castillero, Rafael	**Romero González, Juan**
Márquez Sierra, María	**Urquijo Martínez, Álvaro**
Márquez Soldán, María	

Grupo	4	**Entregable**	DP
Repositorio	https://github.com/Cohabify/Cohabify		

Tabla de contenidos

Resumen ejecutivo

En el siguiente documento se detallan las políticas que serán utilizadas para mantener una homogeneidad en la gestión del repositorio y del código.

Se estipula que los commits deben ser atómicos y siguiendo una plantilla, lo que mejora la legibilidad.

En cuanto a la política de branching,se seguirá el estándar de gitflow, con ramas de feature, de develop o de release, entre otros.

Las issues serán catalogadas por tipos, y se tomarán medidas según este.

Las normas de código buscan reducir la complejidad y la deuda técnica de este, así como reducir el código spaghetti.

Por último, el versionado de la aplicación será la definición de los estados estables de la aplicación, por lo que se utilizará el sistema vX.Y.Z.

1. Políticas de desarrollo

1.1 Política de commits

A continuación, se detallan una serie de características que debe tener cualquier commit de nuestro repositorio, buscando que estos sean homogéneos y facilitando la lectura y la rápida interpretación.

1.1.1 Normas de commits

Commits Atómicos: Cada commit debe representar un cambio atómico y significativo en el código. Esto facilita la revisión de código y la identificación de errores.

Mensajes Descriptivos: Los mensajes de commit deben ser descriptivos y claros, explicando qué cambios se realizaron y por qué. Esto ayuda a entender el propósito de cada cambio sin necesidad de revisar el código.

Frecuencia Regular: Se debe animar a realizar commits de forma regular y frecuente, en lugar de acumular muchos cambios en un solo commit. Esto facilita la colaboración y reduce el riesgo de conflictos.

Revisión de Código: Antes de hacer un commit en el branch principal, se debe realizar una revisión de código por parte de al menos un compañero de equipo que no haya participado en la creación del commit. Esto ayuda a mantener la calidad del código y a identificar posibles problemas antes de fusionar los cambios.

Integración Continua: Utilizar herramientas de integración continua para automatizar pruebas y despliegues. Los commits deben pasar estas pruebas antes de fusionarse en el branch principal.

Resolución de Conflictos: En caso de conflictos durante la fusión de branches, se debe resolver de manera rápida y efectiva, comunicando cualquier cambio importante al equipo.

1.1.2 Plantilla de commits

tipo: asunto
<Línea en blanco>
cuerpo

Donde:

Tipos de commits

- feat (nueva funcionalidad)
- fix (corrección de bugs)
- research (incorporación de código experimental, puede ser no funcional)
- refactor (refactorización de código)
- docs (actualización de documentación)
- test (incorporación o modificación de tests)
- conf (modificación de archivos de configuración)

Asunto

Consiste en una breve descripción del problema que se ha tratado y que debe de comenzar con un verbo en participio.

Cuerpo (opcional)

Se utilizará en caso de que el asunto no sea suficientemente descriptivo.

Ejemplo
conf : Actualizado docker-compose.yml

1.2 Política de branching

A la hora de elegir una política de branching, se optará por utilizar Git Flow, debido a las ventajas que esta puede brindar a nuestra aplicación, además de nuestra familiaridad con el método. Dicha política se basa en la idea de tener distintos tipos de branches para gestionar las diferentes etapas del ciclo de vida del desarrollo de software. Los tipos de ramas de este estándar son:

Branch Master (Main):

- En Gitflow, el branch master (o main como se le llama en algunas convenciones) es el branch principal y estable. Este branch debería contener sólo el código que está listo para ser desplegado en producción.
- Beneficio para el proyecto: Proporciona una línea base estable para la versión actual del producto, lo que garantiza que la versión desplegada en producción sea confiable y funcional.

Branch Develop:

- El branch develop es donde se integran todas las características completadas y se realiza la mayoría del desarrollo.
- Beneficio para el proyecto: Ofrece un entorno centralizado para la integración continua y la colaboración del equipo. Permite realizar pruebas de integración tempranas y mantener un código base consistente y en evolución.

Branches Feature:

- Cada nueva característica o historia de usuario se desarrolla en su propio branch de feature, que se deriva de develop.
- Beneficio para el proyecto: Permite un desarrollo paralelo de múltiples características sin interferencias entre sí. También facilita la revisión de código y la realización de pruebas específicas para cada característica antes de su integración en el branch develop.

Branches Release:

- Cuando se está preparando una nueva versión para el lanzamiento, se crea un branch de release desde develop. En este branch se realizan las últimas pruebas y ajustes antes del lanzamiento.
- Beneficio para el proyecto: Proporciona un entorno controlado para la preparación del lanzamiento, lo que permite corregir errores de último minuto y realizar pruebas finales sin interferir con el desarrollo en curso en develop.

Branches Hotfix:

- Si surge un problema crítico en producción que requiere una solución inmediata, se crea un branch de hotfix desde master. Una vez corregido el problema, los cambios se fusionan tanto en master como en develop.
- Beneficio para el proyecto: Permite abordar problemas urgentes en producción de manera rápida y controlada, mientras se mantiene la estabilidad del branch develop para el desarrollo continuo.

1.3 Política de versionado

Se ha decidido utilizar el esquema de versionado vX.Y.Z porque proporciona una forma clara y consistente de identificar y comunicar las versiones de nuestro software en desarrollo. En este esquema:

El número de versiones se divide en tres partes: X, Y y Z.

- El número X indica la versión principal o mayor, que cambia cuando se realizan cambios significativos que pueden afectar la compatibilidad hacia atrás o introducir nuevas características importantes.
- El número Y representa la versión secundaria o menor, que se incrementa con cada nueva funcionalidad agregada o mejora que no rompe la compatibilidad con versiones anteriores.
- El número Z corresponde a la versión de corrección o revisión, que se incrementa con cada corrección de errores o parches que no alteran la funcionalidad existente.

1.4 Política de issues

Una incidencia se define como cualquier evento o situación que afecta al desarrollo, implementación o funcionamiento del software, requiriendo atención para su resolución. En nuestro proyecto de búsqueda de compañeros de piso, identificamos tres tipos de incidencias:

1. Tipo Incremento: Estas incidencias representan los diferentes incrementos funcionales planteados durante el desarrollo del proyecto.

2. Tipo Conflicto: Se detectan al fusionar ramas durante una solicitud de extracción (pull request). Si se encuentran conflictos durante este proceso, se crea una incidencia y se espera que el revisor y el desarrollador de la funcionalidad resuelvan estos conflictos (por ejemplo, "Conflicto-nombre_de_la_rama").

3. Tipo Error: Estas incidencias no están relacionadas con el código en sí, sino con aspectos del proyecto. Por lo general, estas incidencias tienen pasos específicos para su resolución, que deben ser documentados en la propia incidencia (por ejemplo, "Error-nombre_descriptivo").

4. Tipo Bug: Se detectan después de fusionar cambios en la rama principal y se refieren a problemas específicos en el código (por ejemplo, "Bug-XXXXX").

Cada incidencia se clasifica según su prioridad (baja, media o alta) en función de la urgencia de su resolución.

En cuanto al método de resolución de incidencias, cada una pasa por tres estados: Detectada, En Resolución y Cerrada. Utilizamos GitHub Project para gestionar estas incidencias, con cada columna representando un estado y las issues representando las incidencias. Cada incidencia se asigna a un responsable y se le asignan tareas para el tipo de incidencia y su prioridad.

Los pasos para resolver cada tipo de incidencia son los siguientes:

- **Incremento:** Se analizan los módulos afectados, se crean las incidencias correspondientes, se trabajan las funcionalidades y se cierran al fusionar los cambios en la rama principal.
- **Conflicto:** Se detecta el conflicto, se crea la incidencia, se resuelven los conflictos y se cierra la incidencia.
- **Error:** Se detecta el error, se crea la incidencia con los pasos para solucionarlo, y se mantiene abierta hasta que se solucione.
- **Bug:** Se detecta el bug, se crea la incidencia, se crea una rama para solucionarlo, se trabaja en la solución y se cierra la incidencia al fusionar los cambios en la rama principal.

1.5 Políticas de código

Con respecto a la realización de código, seguiremos estas pautas con el objetivo de conseguir claridad y eficiencia a la hora de desarrollar nuestra aplicación:

- **Nombres claros y descriptivos:** Los nombres que usaremos para variables, funciones y clases serán significativos y describirán su uso. Evitaremos abreviaturas complejas y nombres poco explicativos que puedan crear confusión a la hora de la lectura.

- **Funciones y métodos pequeños y específicos:** A la hora de crear funciones, estas serán cortas y se centrarán en una tarea específica en la medida de lo posible para conseguir una mayor legibilidad del código, además de facilitar el trabajo de entendimiento del mismo por parte del resto de compañeros del equipo. En el caso de tener funciones demasiado largas, se valorará el dividirlas en otras más cortas.

- **Comentarios útiles y concisos:** Solo se añadirán comentarios en lugares en los que sea necesario aclarar el propósito del código comentado para evitar llenar líneas de texto que no aporta nada.

- **Formato consistente:** Para esta política se enumeran ciertas prácticas que ayudarán a mantener el código legible y homogéneo:
    - Usaremos la sangría de forma adecuada y homogénea a lo largo del proyecto.
    - Mantendremos el espaciado de forma consistente.

- **Evita la duplicación de código (DRY):** Trataremos de mantener el código repetido al mínimo posible. Encapsularemos la mayor parte lógica repetida en funciones, clases o módulos reutilizables para facilitar el trabajo de desarrollo.

- **Refactorización constante:** En el caso de encontrarnos con un posible elemento refactorizable en el código, este se valorará con el objetivo de analizar si su refactorización producirá un cambio significativo en la limpieza del código.