Metodología de gestión de configuración.

Índice

1. Normas de Codificación
2. Políticas de Mensajes de Commits
3. Estructura de Repositorios y Ramas Predeterminadas
4. Estrategia de Ramificación
    - 4.1 Cómo Desarrollar Ramas de Funcionalidades
    - 4.2 Cómo Preparar Versiones
    - 4.3 Cómo Corregir Errores en Producción
5. Revisiones
6. Políticas de Versionamiento
7. Definición de "Hecho"
8. Gestión de Documentos Generados Durante el Proyecto
9. Funcionalidad de iTop (CMDB)
    - 9.1 Objetivo y alcance de la CMDB
    - 9.2 Estructura de la CMDB

1. Normas de Codificación

1. **Nomenclatura de Clases:** Deben comenzar con una letra mayúscula y usar camelCase. ***Ejemplo: MiClase***
2. **Nomenclatura de Variables:** Deben comenzar con una letra minúscula y usar camelCase. ***Ejemplo: miVariable***
3. **Nomenclatura de Métodos:** Deben comenzar con una letra minúscula y usar camelCase. ***Ejemplo: miMetodo()***
4. **Nomenclatura de Constantes:** Deben estar en mayúsculas y usar guiones bajos para separar las palabras. ***Ejemplo: MI_CONSTANTE***

5. **Nomenclatura de Documentos:** Deben comenzar con una letra mayúscula y usar camelCase con extensión .md. ***Ejemplo: MiDocumento.md***
6. **Indentación:** Será de una tabulación.
7. **Longitud de Línea:** Se recomienda que las líneas de código no superen los 80 caracteres de longitud para mejorar la legibilidad.
8. **Comentarios:** Se deben incluir comentarios descriptivos en el código para explicar su funcionamiento, especialmente en partes complejas o críticas. Los comentarios deben estar en inglés y seguir un estilo claro y conciso.
9. **Importaciones:** Se deben evitar las importaciones de paquetes completos y solo importar las clases específicas necesarias. Además, las importaciones deben agruparse y ordenarse de manera lógica.
10. **Formato de Archivos:** Los archivos Java deben tener una estructura clara y consistente, con una declaración de paquete, seguida de importaciones y luego la definición de clases y métodos.

---

2. Políticas de Mensajes de Commits

Los mensajes de commits seguirán la siguiente estructura:

> ***[Tipo]: Descripción en inglés***

Tipos :

- **[feat]** : Indica que el commit está introduciendo una nueva característica o funcionalidad al proyecto. (Código)
- **[edit]** : Indica que se han realizado modificaciones o ediciones sin introducir nuevas características o funcionalidades. (Código)
- **[fix]** : Indica que se ha corregido un error o un problema específico en el código. (Código)
- **[wip]** : Indica que el trabajo está en curso ("Work In Progress") y aún no está completo. (Código)
- **[docs]** : Indica que los cambios realizados están relacionados con la documentación del proyecto. (Documentación)
- **[refactor]** : Indica que se han realizado cambios en el código con el objetivo de mejorar su estructura, claridad, rendimiento o mantenibilidad sin modificar su comportamiento funcional externo. (Código)

---

3. Estructura de Repositorios y Ramas Predeterminadas

Al crear un repositorio en GitHub.com, se establece automáticamente una rama predeterminada. Esta rama es la primera que se muestra cuando alguien visita el repositorio y es la rama inicial que Git verifica localmente al clonar el repositorio. Por defecto, GitHub nombra esta rama predeterminada como "main" en cualquier repositorio recién creado.
Además, creamos una rama llamada "develop", que actuará como la rama principal para cada sprint. Esta rama se actualizará a medida que se completen y revisen los issues. La rama "main" permanecerá sin modificaciones hasta que se complete el sprint, momento en el que todo el contenido de la rama "develop" se fusionará en ella.

---

4. Estrategia de Ramificación

4.1 Cómo Desarrollar Ramas de Funcionalidades

Para cada issue tendrá una rama propia y será implementada en su propia rama, así es fácil saber qué código implementa qué issue, con solo buscar el número de issue en las ramas. Las ramas seguirán el siguiente formato :

> *Feature/<descripción del issue>/<número de issue>*

Una vez que se haya implementado y revisado correctamente el issue definido, se realizará un pull-request a la rama develop.

4.2 Cómo Preparar Versiones

Para cada release crearemos una rama propia. Las ramas seguirán el siguiente formato :

> *Release/<versión del proyecto>*

La rama main se actualizará cada vez que se haga una release.

4.3 Cómo Corregir Errores en Producción

Para arreglar los bugs inesperados en medio de la producción, existirá una rama llamada hotfix. Hotfix es similar a la rama releases solo que será utilizada para resolver bugs críticos en una versión o errores no planeados en la producción.

---

5. Revisiones.

Antes de fusionar cualquier cambio de cualquier rama en la rama `develop`, es necesario crear un pull request. Este pull request debe ser revisado y aceptado por una persona distinta a quien lo ha propuesto, lo que garantiza una revisión imparcial. Durante esta revisión, cualquier fal

lo detectado será señalado mediante comentarios, y el autor del pull request deberá corregirlos lo antes posible. Todo el equipo de desarrollo está comprometido a priorizar la revisión de pull requests para evitar retrasos en su aceptación o acumulación de cambios pendientes.

---

6. Políticas de Versionamiento.

Vamos a seguir el formato de versiones X.Y.Z. La primera versión será la 1.00.
Cuando se lanza una versión mayor (major release), incrementaremos el número `X` de la versión y, si existen, pondremos a cero los números `Y` y `Z`.

> *Release/2.0.0*

Para las versiones menores (minor release), mantendremos el número `X`, aumentaremos `Y` y resetearemos `Z`, que es la versión de parche (patch).

> *Release/2.1.0*

Finalmente, cuando lanzamos una versión de parche (patch release), incrementamos `Z`.

> *Release/2.1.1*

---

7. Definición de "Hecho".

Los criterios que seguiremos para considerar completada y lista para entregar una historia de usuario o una tarea son los siguientes:

- Se ha escrito y revisado el código correspondiente o la documentación por al menos un miembro del equipo.
- El código o documento cumple con los estándares de codificación establecidos.
- La tarea ha sido revisada y aprobada por el Product Owner o el cliente para asegurarse de que cumple con sus expectativas y requisitos.
- Se han corregido todos los defectos identificados durante las pruebas, y se ha verificado que las correcciones sean efectivas.
- El equipo está de acuerdo en que la tarea cumple con todos los criterios definidos y se considera completa y lista para ser entregada.

---

8. Gestión de Documentos Generados Durante el Proyecto.

Los documentos se crearán directamente en el editor de texto de GitHub en la rama `doc`, y se almacenarán en la carpeta del proyecto `/docs/sprintX`, donde "X" representa el número del sprint actual.

---

9. Funcionalidad de iTop (CMDB)

9.1 Objetivo y alcance de la CMDB

La principal funcionalidad de una CMDB (Configuration Management Database) para una organización es proporcionar un repositorio centralizado y estructurado sobre los elementos de configuración (CI). Estos pueden incluir Hardware, Software, documentación, y cualquier otro elementos necesario.

La CMDB ayuda a mantener un inventario actualizado de todos los activos de TI de la organización, incluyendo servidores, equipos de red, dispositivos de almacenamiento, software, licencias y otros recursos. También facilita la gestión de cambios al proporcionar información detallada sobre los elementos de configuración afectados por un cambio propuesto, permitiendo una evaluación de impacto más precisa y una mejor planificación de cambios. Ayuda a identificar rápidamente los elementos de configuración afectados por incidentes o problemas, lo que facilita la resolución de problemas y la restauración del servicio. Además, proporciona datos precisos y actualizados, lo que ayuda a los equipos de gestión a tomar decisiones informadas sobre la inversión en tecnología, la optimización de recursos y la planificación estratégica.

9.2 Estructura de la CMDB

Nuestra organización esta compuesta por cinco miembros, los usuarios esta asociado a la unidad organizativa correspondiente, definiendo quien tiene acceso a que recursos y funciones dentro de iTop. A continuación, se dejara un enlace a la lista de configuración de elementos por cada miembro de la organización. Se ha establecido una nomenclatura a la hora de introducir los elementos para que faciliten la comprensión y búsqueda de los elementos por todos los miembros.

> *nombredellemento_uvusdelmiembro*

Listado de elementos de configuración