Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática

DEFINICIÓN DE PROCESOS

Grado en Ingeniería Informática – Ingeniería del Software Evolución y Gestión de la Configuración

EGC2425-RaboDeToro-hub-1 2024 – 2025

Índice

1. Resumen ejecutivo **2**
2. Contenido **3**
a. Integración continua **3**
Definición y Objetivos 3
Herramientas y Tecnologías Utilizadas 3
Flujo de Trabajo 3
Beneficios Observados 4
Entorno de Desarrollo y Virtualización 5
b. Construcción **6**
Configuración del Entorno y Dependencias: 6
Análisis de Calidad y Linting: 6
Compilación y Preparación del Código: 6
Ejecución de Pruebas Automáticas: 6
Integración Continua y Verificación en Render: 6
Empaquetado y Despliegue Final: 6
c. Cambios **8**
Gestión de Cambios en el Proyecto 8
Ventajas de este Enfoque 9
d. Código **10**
Organización del Código 10
e. Despliegue **13**
Configuración de CI/CD: 13
Contenerización con Docker: 13
Gestión de Dependencias: 13
Estrategias de Despliegue: 13
Monitoreo y Logs: 14
f. Pipeline del proyecto **14**
1. Validación de Commits (commit.yml) 14
2. Análisis de Calidad del Código (codacy.yml y codacy2.yml) 15
3. Pruebas Automatizadas (test.yml) 16
4. Renderizado y Despliegue(render.yml) 16
5. Integración con Docker y Vagrant 16
6. Despliegue final 17
g. Conductas y sanciones **17**

1. Resumen ejecutivo

El proyecto **RaboDeToro-Hub-1** tiene como objetivo proporcionar una plataforma que integre modelos en formato UVL, permitiendo a los usuarios acceder a conjuntos de datos públicos de manera eficiente y organizada. Este proyecto se desarrolla bajo los principios de Ciencia Abierta, promoviendo la accesibilidad y la colaboración entre comunidades de usuarios.

A lo largo del desarrollo, se han implementado mejoras clave en la usabilidad y funcionalidad de la plataforma, como la incorporación de filtros avanzados, la creación de comunidades de usuarios y la funcionalidad de recuperación de contraseñas. Además, se ha optimizado la integración con herramientas externas como Zenodo, utilizando una implementación local (Fakenodo) para garantizar la sostenibilidad y el rendimiento.

El documento de definición de procesos detalla las metodologías y estándares seguidos durante el desarrollo, incluyendo integración continua, control de versiones y pruebas automatizadas, asegurando la calidad y escalabilidad del producto. Este enfoque garantiza que la plataforma sea adaptable a las necesidades cambiantes de los usuarios y las tendencias tecnológicas.

2. Contenido

a. Integración continua

Definición y Objetivos

La integración continua (CI, por sus siglas en inglés) es una práctica esencial en el desarrollo de software moderno que busca garantizar la calidad y la estabilidad del código mediante la integración frecuente de los cambios realizados por el equipo de desarrollo. En el proyecto **RaboDeToro-Hub-1**, la integración continua se ha implementado para automatizar la compilación, pruebas y despliegue, reduciendo el tiempo necesario para identificar errores y mejorar la colaboración entre los miembros del equipo.

El objetivo principal de la CI en este proyecto es:

- Garantizar que cada cambio en el código sea probado y validado automáticamente.
- Reducir el riesgo de integración tardía y conflictos de código.
- Proveer retroalimentación rápida sobre el estado del sistema después de cada cambio.

Herramientas y Tecnologías Utilizadas

Para implementar la integración continua en **RaboDeToro-Hub**, se han utilizado las siguientes herramientas:

- **GitHub Actions:** Automatización de flujos de trabajo para ejecutar pruebas, construir el proyecto y realizar despliegues en entornos de desarrollo y producción.
- **Docker**: Contenerización del entorno de desarrollo y pruebas para garantizar la consistencia entre diferentes configuraciones locales y en el servidor.
- **Codacy**: Análisis estático del código para garantizar la calidad y adherencia a los estándares definidos.
- **Pytest**: Herramienta utilizada para ejecutar pruebas unitarias e integrales.
- **Render**: Plataforma de despliegue continuo utilizada para publicar la aplicación web tras la validación del pipeline.

Flujo de Trabajo

El flujo de trabajo de integración continua para este proyecto sigue los siguientes pasos:

1. **Control de Versiones**:
   Todos los cambios realizados al código se integran a través de ramas en el repositorio principal de GitHub. Las ramas siguen un esquema basado en GitFlow:

- `develop`: Contiene el código en desarrollo.
- `main`: Contiene la versión estable del proyecto.
- Ramas feature (`feature/nombre_feature`): Usadas para el desarrollo de nuevas funcionalidades.
- Ramas de `hotfix`: se utilizan para solucionar errores que se encuentren en producción. Se crean a partir de la rama main y se fusionan en las ramas main y develop una vez que el error está solucionado.

2. **Ejecución Automática del Pipeline**:
   Cada vez que se crea un **Pull Request** o se realiza un **push** en las ramas monitoreadas, se desencadena un flujo de integración continua en GitHub Actions. Este flujo incluye:
   - **Paso 1**: Instalación de dependencias y configuración del entorno virtual.
   - **Paso 2**: Ejecución de pruebas unitarias y de integración mediante `pytest`.
   - **Paso 3**: Análisis de la calidad del código con **Codacy**.
   - **Paso 4**: Generación de artefactos (builds) listos para despliegue en contenedores Docker.
   - **Paso 5**: Despliegue automatizado en entornos de prueba o producción mediante **Render**.

3. **Validación del Código**:
   Solo se permite la integración de código en `develop` o `main` si:
   - Todas las pruebas han sido superadas.
   - No se han encontrado vulnerabilidades críticas en el análisis estático.
   - El equipo ha realizado una revisión manual del Pull Request (Code Review).
   - Se pasan todos los Workflows del proyecto satisfactoriamente.

Beneficios Observados

- **Detección temprana de errores:** Gracias a la ejecución automática de pruebas, los errores se detectan antes de que se integren a las ramas principales.
- **Mayor calidad del código:** La integración de herramientas como Codacy asegura que el código cumpla con estándares de calidad y mejores prácticas.
- **Despliegue rápido y seguro:** El uso de Docker y Render permite desplegar la aplicación de manera rápida y confiable en entornos controlados.
- **Colaboración eficiente:** El equipo puede trabajar en paralelo en diferentes ramas sin temor a conflictos de integración tardía.

Entorno de Desarrollo y Virtualización

Para facilitar la ejecución de la aplicación en diferentes entornos, se ha implementado un sistema de virtualización usando Vagrant. Este enfoque permite ejecutar la aplicación en una máquina virtual sin necesidad de configuraciones locales adicionales. Para ello, se ha creado un archivo Vagrantfile que, al ejecutarse mediante el comando vagrant up, realiza la creación de una máquina virtual y la ejecución de los scripts de aprovisionamiento.

El aprovisionamiento se realiza utilizando Ansible y ciertos scripts personalizados debido a incompatibilidades con el script original. Este proceso automatizar tareas como:

- Creación de la máquina virtual.
- Actualización del sistema.
- Instalación de MariaDB y configuración del entorno.
- Instalación de las dependencias del proyecto.
- Instalación de Rosemary y prueba de la aplicación.

De esta manera, el entorno queda listo para ser accesible en localhost:5000. Cuando se termine de trabajar con la máquina virtual, los comandos vagrant halt y vagrant destroy permitirán apagar y eliminar la máquina, respectivamente.

b. Construcción

El proyecto **RaboDeToroHub1** tiene un proceso de construcción el cual garantiza la calidad y la estabilidad del código antes de su despliegue. A continuación, se describen los detalles clave de este proceso:

Configuración del Entorno y Dependencias:

El proyecto utiliza herramientas de construcción que aseguran que todas las dependencias necesarias se instalen y configuren en el entorno de desarrollo y producción. Al ser un proyecto de Python, el archivo requirements.txt o configuraciones equivalentes permiten la instalación automática de librerías necesarias para la aplicación, facilitando la configuración en distintos entornos.

Análisis de Calidad y Linting:

Antes de proceder con la construcción, se ejecutan herramientas de análisis de calidad de código (como Codacy, ya integrado en este proyecto) para detectar errores y problemas de estilo en el código. Codacy realiza una revisión estática que ayuda a los desarrolladores a identificar y corregir problemas antes de que el código se compile o se ejecute.

Compilación y Preparación del Código:

Aunque en proyectos de Python no se realiza una compilación estricta, se

asegura que el código esté en un estado listo para ejecutarse sin errores. En esta etapa, también se validan módulos y se configuran scripts de inicio que establecen los parámetros necesarios para que la aplicación funcione correctamente.

Ejecución de Pruebas Automáticas:

El proyecto incluye pruebas automáticas que se ejecutan después de la construcción para validar que los componentes críticos del sistema funcionen según lo esperado. Estas pruebas pueden incluir pruebas unitarias y de integración. Al ejecutar las pruebas de manera automatizada, se garantiza que cualquier error en el código se detecte y se pueda corregir antes del despliegue.

Integración Continua y Verificación en Render:

Después de completar las pruebas, la integración continua a través de Render permite un despliegue automatizado. Render asegura que la última versión del proyecto se implemente en un entorno accesible, donde se pueden realizar pruebas adicionales en un entorno real.

Empaquetado y Despliegue Final:

Una vez superadas las pruebas y la revisión de calidad, el código se prepara para el despliegue. Render gestiona este proceso de empaquetado y despliegue, de modo que la aplicación esté lista y accesible para los usuarios o para su integración en entornos de producción.

c. Cambios

La gestión de cambios se organiza cuidadosamente mediante el uso de ramas, pull requests (PRs) y otras prácticas de control de versiones. Esto permite mantener un flujo de trabajo ordenado y facilita la colaboración entre los desarrolladores, asegurando que el código se integre de manera segura y controlada. A continuación, se describe cómo se implementa este proceso:

Gestión de Cambios en el Proyecto

1. **Ramas**:

El proyecto utiliza ramas para organizar y aislar las diferentes etapas de desarrollo. Las ramas más comunes incluyen:

- **main**: La rama principal, que contiene la versión estable del proyecto y que no debe modificarse directamente.
- **develop**: Esta rama suele ser la base para el desarrollo y la integración de nuevas características antes de que se fusionen en main.
- **feature/***: Ramas creadas para desarrollar nuevas funcionalidades. Cada nueva característica o mejora se desarrolla en una rama feature, lo cual permite a los desarrolladores trabajar en paralelo sin afectar la estabilidad de main o develop.

- **hotfix/***: Ramas destinadas a resolver errores específicos. Cada corrección se trabaja en una rama hotfix, asegurando que los cambios necesarios se apliquen de forma aislada y controlada.

También se pueden generar en casos especiales otras ramas las cuales tengan una utilidad relevante a la hora de implementar o definir algún elemento o concepto dentro de la aplicación o para facilitar la gestión del cambio.

Esta estructura de ramas facilita la organización y el seguimiento de cada cambio, permitiendo que el equipo desarrolle nuevas características o corrija errores sin interferir con el trabajo de los demás.

2. Pull Requests (PRs):

Cada cambio importante o nueva funcionalidad en el proyecto se introduce mediante un pull request. Un PR permite que los desarrolladores propongan cambios en una rama y soliciten una revisión antes de fusionarlos con develop o main.

Los PRs son una parte fundamental del proceso de revisión de código, ya que permiten que otros miembros del equipo revisen el código, hagan sugerencias y detecten posibles errores o mejoras antes de que el cambio se integre en una rama principal.

El uso de PRs ayuda a asegurar la calidad y coherencia del código en el proyecto. Cada PR suele incluir una descripción detallada de los cambios y, en algunos casos, pruebas que demuestren que la nueva funcionalidad o corrección funciona como se espera.

3. Revisión y Aprobación de Cambios:

Antes de fusionar un PR en develop o main, al menos uno de los miembros del equipo debe revisar y aprobar los cambios. Esta revisión puede incluir pruebas de funcionamiento, verificación de estilo y consistencia, así como asegurar que los cambios cumplen con los estándares del proyecto.

Los revisores pueden hacer comentarios en el PR, sugiriendo ajustes o cambios adicionales. Una vez que el autor del PR aborda estos comentarios, el PR puede ser aprobado y fusionado en la rama de destino.

Integración de Cambios en Ramas Principales:

Después de que los PRs han sido aprobados y probados en develop, los cambios se pueden fusionar en main cuando el equipo esté listo para lanzar una nueva versión estable del proyecto. Este proceso asegura que solo el código completamente probado y revisado llegue a la versión estable.

Automatización y Control de Calidad:

Con la integración de Codacy y Render, cada cambio propuesto en un PR pasa automáticamente por análisis de calidad y pruebas antes de ser revisado por un miembro del equipo. Esto ayuda a detectar problemas de calidad, errores o vulnerabilidades en una etapa temprana del ciclo de desarrollo.

Ventajas de este Enfoque

Este enfoque estructurado permite al equipo mantener un flujo de trabajo eficiente y seguro, reduciendo el riesgo de errores y facilitando la cooperación entre desarrolladores. Al utilizar ramas, pull requests y revisiones de código, el proyecto se beneficia de un control riguroso de cambios, manteniendo alta la calidad del código y permitiendo que cada nueva funcionalidad o corrección se integre de manera organizada y controlada.

d. Código

La gestión y organización del código son esenciales para asegurar su calidad, mantenibilidad y escalabilidad. A continuación, se detallan las prácticas y estándares de código usados en este repositorio.

Organización del Código

Estructura de Archivos y Directorios:

El proyecto está organizado en directorios que separan claramente los diferentes componentes y módulos de la aplicación. Esta estructura facilita la navegación y permite a los desarrolladores encontrar rápidamente el código relevante para cualquier tarea.

Los directorios suelen incluir:

- app: Contiene el núcleo de la aplicación, con la lógica principal del sistema.
- core: Incluye archivos fundamentales y configuraciones esenciales para el funcionamiento del proyecto.
- docs: Documentación adicional que proporciona instrucciones y detalles técnicos para que los desarrolladores comprendan el código y el proyecto en general.
- migrations: Archivos de migración de base de datos, en caso de que el proyecto requiera persistencia de datos, facilitando la actualización y modificación de la estructura de datos.

Estándares de código:

El proyecto sigue un estándar de codificación para mantener la consistencia y calidad del código. Este estándar cubre aspectos como el formato, el uso de nombres de variables y funciones, y la organización de módulos y clases.

Codacy, una herramienta de análisis de calidad de código está integrada en el repositorio y ayuda a asegurar que el código cumpla con las mejores prácticas y estándares de calidad. Codacy realiza análisis automáticos para detectar problemas de estilo, vulnerabilidades de seguridad y posibles errores lógicos.

La política de commits en el proyecto se basa en la claridad, estructura y estandarización mediante la convención **Conventional Commits**. Los puntos clave incluyen:

1. **Estructura del mensaje de commit**:
    - Línea de asunto: breve y descriptiva, en modo imperativo, con un máximo de 50 caracteres y sin punto final.
    - Línea en blanco entre el asunto y el cuerpo.

    - Cuerpo opcional: ajustado a un máximo de 72 caracteres por línea, proporcionando detalles adicionales si es necesario.
    - Pie opcional: puede incluir referencias a issues con el formato `Refs #número-issue`.
2. **Uso de prefijos en el asunto**:
    - **fix**: Corrección de errores. Ejemplo: `fix: Corregir error en inicio de sesión`.
    - **feat**: Nueva funcionalidad. Ejemplo: `feat: Añadir función de búsqueda avanzada`.
    - **docs**: Cambios en la documentación. Ejemplo: `docs: Actualizar guía de instalación`.
    - **style**: Cambios de estilo que no afectan la funcionalidad. Ejemplo: `style: Corregir formato en código`.
    - **test**: Adición o corrección de pruebas. Ejemplo: `test: Añadir pruebas para función X`.
3. **Requisitos adicionales**:
    - Los commits deben ser claros, relevantes y seguir el estándar definido para facilitar el seguimiento del historial del proyecto.
    - La convención busca mejorar la colaboración y el entendimiento entre los miembros del equipo.

Política de Issues

La política de Issues se centra en la organización, priorización y documentación detallada de las tareas. Los aspectos principales son:

1. **Estructura de las Issues**:
    - **Título**: Resumen claro del objetivo de la tarea.
        - Work Items: Nombrados como `WI-nombre del workitem`.
        - Tareas técnicas: Nombradas con el nombre específico de la tarea.
    - **Descripción**: Detalle de la funcionalidad o problema, en español.
    - **Assignees**: Responsable de ejecutar la tarea.
    - **Labels**: Clasificación según prioridad y tipo.
    - **Project:** Se vincula al **Project** correspondiente.
2. **Etiquetas disponibles**:
    - **Prioridad**:
        - `Low priority`: Resolución no urgente.
        - `Medium priority`: Necesita resolverse pronto.
        - `High priority`: Resolución inmediata.
    - **Tipo**:
        - `Add`: Añadir al código.
        - `Bug`: Solución de errores.
        - `Documentation`: Cambios en documentación.
        - `CI`/`CD`: Tareas relacionadas con integración o despliegue continuo.

        - `Test`: Verificación de funcionalidad.
        - `Workflows`: Implementar nuevos workflows.
        - `Fakenodo`: Tareas relacionadas con Fakenodo.
3. **Relación con ramas y milestones**:
    - Cada Issue está vinculada a una rama de desarrollo específica y al milestone correspondiente (M1, M2, M3).
4. **Estados:**
    - **Todo:** Tareas que aún no se han comenzado, están pendientes y generalmente se encuentran en la etapa inicial de planificación.
    - In Progress: Tareas en las que el equipo está trabajando activamente, están en proceso de desarrollo o implementación.
    - Test: Estas issues están siendo probadas, están en fase de verificación para asegurar su calidad y funcionalidad.
    - In Review: Tareas que están en proceso de revisión por parte del equipo.
    - Done: Estas issues han sido completadas y marcadas como finalizadas.

Esta política garantiza una gestión eficiente de tareas y una priorización clara, mejorando la colaboración y el seguimiento en el equipo.

Gestión de Issues con Plantillas:
Para estandarizar la creación y gestión de issues, el proyecto utiliza plantillas predefinidas que garantizan la uniformidad y claridad en cada tarea registrada. Estas plantillas se encuentran en el directorio .github/ISSUE_TEMPLATE del repositorio y están diseñadas para abordar diferentes tipos de tareas:

- **issue_work_item_template.yml:** Para la definición de Work Items, describiendo objetivos, prioridad, y resultados esperados.
- **issue_technical_task_template.yml:** Para tareas técnicas específicas, detallando dependencias y resultados esperados.

Además, el archivo **`config.yml`** configura cómo se presentan y gestionan estas plantillas dentro del repositorio. Este archivo permite:

- **Definir la disponibilidad de las plantillas:** Determina qué plantillas se muestran al crear una nueva issue.
- **Ofrecer opciones personalizadas:** Permite que los colaboradores seleccionen una plantilla específica o inicien con una plantilla en blanco si es necesario.
- **Organizar las plantillas por tipo:** Mejora la experiencia al categorizar las plantillas según su propósito.

El uso combinado de estas plantillas y la configuración proporcionada por **`config.yml`** asegura que cada issue esté bien documentada, organizada y estructurada, facilitando la asignación, el seguimiento y la colaboración entre los miembros del equipo.

Política de Ramas

La política de ramas en el proyecto sigue la estrategia **GitFlow**, que organiza el desarrollo mediante ramas específicas para gestionar de manera eficiente las funcionalidades, correcciones y versiones del proyecto. Los principales puntos son:

1. **Ramas Principales**:
   - **`main`**: Contiene la versión estable y en producción del proyecto. Es la rama que siempre debe estar funcional y lista para ser desplegada.
   - **`develop`**: Rama de desarrollo que contiene el código en progreso. Aquí se integran las nuevas funcionalidades y correcciones antes de fusionarlas en `main`.
2. **Ramas de Soporte**:
   - **`feature/*`**: Ramas creadas para desarrollar nuevas funcionalidades. Se crean a partir de `develop` y, una vez completada la funcionalidad, se fusionan de nuevo en `develop`. Esto permite a los desarrolladores trabajar en paralelo sin afectar la estabilidad de las ramas principales.
   - **`hotfix/*`**: Ramas destinadas a corregir errores que afectan la producción. Se crean a partir de `main` y se fusionan tanto en `main` como en `develop` para asegurar que los cambios se apliquen en ambas ramas.
3. **Flujo de Trabajo**:
   - Las ramas **feature** se crean para cada nueva característica y se nombran de manera descriptiva (por ejemplo, `feature/login-functionality`).
   - Una vez que se ha completado una característica, se realiza un **pull request** (PR) para fusionarla en `develop`, donde se revisa y valida antes de ser integrada.
   - **`hotfix`** se usa solo para correcciones urgentes en producción, con un flujo similar al de las ramas `feature`.
4. **Frecuencia de Integración**:
   - La integración de cambios en `main` desde `develop` ocurre al final de cada **milestone**. Esto garantiza que solo el código completamente probado y validado se fusione en la versión estable del proyecto.

Esta estructura permite un desarrollo organizado, controlado y con un manejo adecuado de los cambios, asegurando la estabilidad y calidad del proyecto durante todo el ciclo de vida del desarrollo.

Buenas Prácticas:

Cada módulo y función incluye comentarios y documentación interna que explican su propósito y funcionamiento. Esto es fundamental para el mantenimiento del proyecto, ya que permite a otros desarrolladores entender rápidamente el propósito de cada fragmento de código.

e. Despliegue

Para el despliegue del proyecto es importante seguir un enfoque que facilite la integración continua (CI) y el despliegue continuo (CD) para mantener el flujo de desarrollo estable y actualizado. Aquí algunos puntos clave para el despliegue de este tipo de proyecto:

Configuración de CI/CD:

Utilizar herramientas de CI/CD como GitHub Actions, Travis CI o Jenkins para automatizar las pruebas y despliegues. Configura un archivo de flujo de trabajo en `.github/workflows` para definir los pasos de integración y despliegue.

Contenerización con Docker:

Implementar un `Dockerfile` y, si es necesario, un `docker-compose.yml` para definir los servicios y sus dependencias. Esto facilita la creación de entornos de desarrollo y producción replicables. Verifica que las dependencias estén correctamente instaladas y configuradas.

Gestión de Dependencias:

Define y actualiza las dependencias en un archivo como `requirements.txt` (Python) o un archivo específico según el lenguaje. Esto asegura que las dependencias sean las mismas en cada despliegue.

Estrategias de Despliegue:

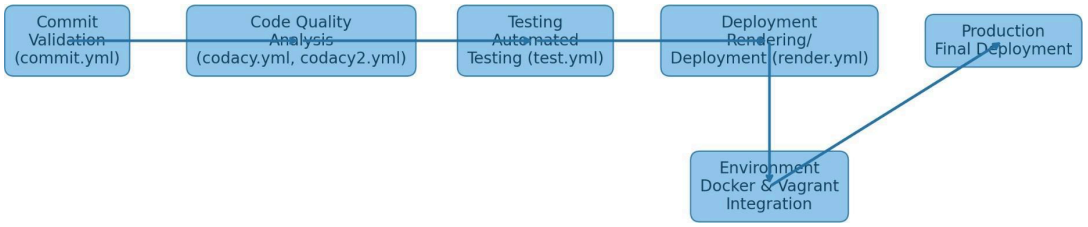
Dependiendo de los requisitos, puedes optar por despliegues en entornos de prueba (staging) antes del despliegue en producción. Servicios como AWS, Google Cloud, o Heroku pueden facilitar este proceso. Define las variables de entorno necesarias para la conexión con bases de datos u otros servicios externos en el entorno de despliegue.

Monitoreo y Logs:

Implementar soluciones de monitoreo para rastrear el rendimiento y errores del sistema en tiempo real. Herramientas como Prometheus o servicios de logging pueden integrarse para mejorar la visibilidad del sistema.

f. Pipeline del proyecto

El pipeline de este proyecto está diseñado para garantizar la calidad del código, automatizar pruebas y simplificar el despliegue en entornos virtualizados. Utiliza workflows en GitHub Actions, integraciones con herramientas de análisis de código y contenedores para entornos consistentes. A continuación, se describe cada etapa del pipeline.



1. **Validación de Commits (commit.yml)**

Este workflow llamado **Commit Syntax Validation** tiene como propósito garantizar que los mensajes de commit cumplan con un formato predefinido.

Eventos que lo disparan:

- Pull Requests: Cuando se crean o actualizan pull requests hacia las ramas main o develop.
- Push: Cuando se realizan commits directamente en las ramas main o develop.

Reglas de Validación

- **Ignorar commits de merge**:
  - Los commits cuyo mensaje comienza con "Merge" son ignorados.
- **Formato del mensaje**:
  - El mensaje debe separar el asunto (primera línea) del cuerpo (contenido restante) con una línea en blanco.
- **Asunto del mensaje**:
  - **No vacío**: El asunto debe estar presente.
  - **Longitud máxima**: El asunto debe tener un máximo de 50 caracteres.
  - **Sin punto final**: El asunto no debe terminar con un punto.
  - **Prefijos obligatorios**: El asunto debe comenzar con uno de los siguientes prefijos:
    - feat: para nuevas funcionalidades.
    - fix: para correcciones de errores.
    - docs: para cambios en la documentación.
    - style: para cambios que no afectan la lógica (formato, espacios, etc.).
    - test: para agregar o modificar pruebas.

Este workflow garantiza un historial de commits limpio y bien estructurado, lo que mejora la colaboración, la trazabilidad y la calidad del proyecto.

2. **Análisis de Calidad del Código (codacy.yml y codacy2.yml)**

- **Codacy.yml**

Este workflow llamado **Codacy CI** sube reportes de cobertura de pruebas a Codacy para garantizar el seguimiento de la calidad del código.

Eventos que lo activan:

- Push: A las ramas main y develop.
- Pull Requests: Hacia las ramas main y develop

Tareas principales:

- Prepara el entorno ejecutando el job en ubuntu-latest y levanta un servicio MySQL5.7 con las credenciales necesarias.
- Obtiene el código fuente, instala python 3.12 y las dependencias
- Crea la carpeta uploads y ejecuta pruebas usando pytest.
- Genera un reporte de cobertura en formato XML y lo sube a Codacy.

- **Codacy2.yml**

Este workflow llamado **Codacy CI-2** amplía la validación con análisis estático del código, cobertura de pruebas y validaciones adicionales en pull requests.

Eventos que lo activan:

- Push: A cualquier rama.
- Pull Requests: En los eventos opened, synchronize, y reopened.

Tareas principales:

- Prepara el entorno de forma similar a la anterior, genera análisis de código con flake8 y genera un reporte en HTML.
- Ejecuta las pruebas y reportes de cobertura al igual que codacy.yml pero también genera un reporte HTML de cobertura y guarda como artefactos en el workflow.
- Realiza validaciones de las pull requests : las pull requests deben tener al menos una etiqueta asignada y al menos un Assignes.

Este workflow amplía las capacidades del anterior, añadiendo análisis estático del código con Flake8, validaciones en pull requests, y almacenamiento de artefactos.

Ambos contribuyen a garantizar un código de alta calidad y trazabilidad.

3. **Pruebas Automatizadas (test.yml)**

Este workflow llamado **Run tests** se encarga de ejecutar un conjunto de pruebas automatizadas utilizando pytest para garantizar la funcionalidad del proyecto en las ramas principales de desarrollo.

Eventos que lo activan:

- Push: Cada vez que hay un cambio en las ramas main o develop.
- Pull Requests: Cuando se crean o actualizan pull requests dirigidas a las ramas main o develop.

Tareas principales:

- Prepara un entorno de prueba basado en Ubuntu con un contenedor MySQL.
- Configura el entorno Python, instala dependencias y ajusta las configuraciones necesarias.
- Ejecuta pruebas automatizadas con pytest, ignorando pruebas específicas que no son relevantes (como las de Selenium).

Este workflow es una pieza clave para mantener la calidad del proyecto.

4. **Renderizado y Despliegue(render.yml)**

Este workflow llamado Deploy to Render realiza el despliegue en render después de ejecutar pruebas automatizadas al realizar cambios en la rama main.

Eventos que lo activan:

- Push: Cada vez que se realiza un push en la rama main.
- Pull Requests: Cuando se crean o actualizan pull requests hacia la rama main.

Funcionalidad:

- Verifica que el código sea funcional mediante pruebas automatizadas antes de intentar el despliegue.
- Si las pruebas son exitosas, realiza un despliegue automatizado en Render utilizando una URL de hook configurada como un secreto.

Este workflow asegura un flujo confiable desde las pruebas hasta el despliegue en producción.

5. **Integración con Docker y Vagrant**

El proyecto utiliza Docker y Vagrant para virtualizar entornos. Estas herramientas permiten:

- Crear entornos consistentes para pruebas y desarrollo.
- Ejecutar el proyecto en un entorno que simula producción (localhost:5000).
- Simplificar la configuración de dependencias y bases de datos como MariaDB.

6. **Despliegue final**

La última etapa del pipeline es el despliegue en el entorno de producción. Esta fase asegura:

- Que solo se despliegue código que haya pasado todas las etapas anteriores.
- Que el entorno de producción sea estable y seguro.

g. Conductas y sanciones

La política de **Conductas y Sanciones** establece normas claras para asegurar un ambiente de trabajo colaborativo y profesional en el equipo de desarrollo de **RaboDeToro-Hub**. Se espera que todos los miembros mantengan una actitud positiva, se comprometan activamente con las tareas asignadas y respeten las políticas del proyecto, como la de commits, ramas y pull requests.

En caso de incumplimiento:

1. **Primer aviso**: El miembro recibe un aviso formal.
2. **Segundo aviso**: Se realiza una reunión para discutir el problema.
3. **Tercer aviso**: Se evalúa la expulsión del equipo si el comportamiento persiste.

El proceso de resolución de conflictos promueve una actitud constructiva, buscando soluciones colaborativas. Si los conflictos no se resuelven internamente, el coordinador tomará la decisión final.

Esta política garantiza que el equipo trabaje de manera ordenada y respetuosa, contribuyendo al éxito del proyecto.