Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática

Integración Continua (Continuous Integration)

Grado en Ingeniería Informática – Ingeniería del Software

Evolución y Gestión de la Configuración

Curso 2024 – 2025

Fecha	Versión
10/11/2024	1.0.

Proyecto: **innosoft-diplomas-1**

Equipo de trabajo
Aragón Sánchez, Alejandro
Chico Castellano, Álvaro
Guillén Fernández, David
Jiménez Osuna, Álvaro
Linares Barrera, Jaime
López Oliva, Ángela

Control de Versiones

Fecha	Versión	Descripción
10/11/2024	1.0.	Creación, elaboración y firma del documento

Índice de contenido

1. Introducción

El presente documento tiene como finalidad definir de manera clara y precisa los acuerdos y directrices que regirán la implementación de la Integración Continua (CI) en el desarrollo del proyecto **innosoft-diplomas-1**. Dichos acuerdos buscan establecer un flujo de trabajo automatizado que garantice la calidad del código y el correcto funcionamiento del sistema a lo largo del ciclo de vida del desarrollo.

2. Herramientas utilizadas

El proyecto **innosoft-diplomas-1** emplea un enfoque automatizado para la integración continua, utilizando las siguientes herramientas y plataformas:

- **Plataforma Utilizada**
  Se ha configurado un **flujo de trabajo** en **GitHub Actions** que se activa cada vez que se realiza un **push** o un **pull request** en la rama preproduction. Esta configuración asegura un ciclo de validación continuo y eficiente, permitiendo al equipo mantener la estabilidad del proyecto a medida que se desarrollan nuevas funcionalidades.

- **GitHub Actions**
  Para asegurar la calidad del desarrollo, se ha integrado **GitHub Actions** como herramienta principal para la **Integración Continua (CI)**. Esta plataforma permite que cada cambio realizado en el código sea **validado automáticamente** antes de ser incorporado a la rama preproduction. Esto garantiza que los errores sean detectados de forma temprana, evitando que afecten al entorno de producción.

3. Flujo de Trabajo

El proyecto **innosoft-diplomas-1** sigue un flujo de trabajo estructurado para la Integración Continua (CI) mediante **GitHub Actions**, con el objetivo de garantizar la calidad del código y la estabilidad del sistema antes de su despliegue. El flujo consta de las siguientes etapas:

1. **Instalación de Dependencias**
   Se instalan todas las dependencias necesarias para el proyecto utilizando el archivo requirements.txt. Esto asegura que el entorno esté correctamente configurado para las etapas posteriores.
2. **Linting y Análisis de Código**
   Se lleva a cabo un análisis exhaustivo del código utilizando herramientas como **flake8**. Esta etapa asegura el cumplimiento de los **estándares de calidad** y ayuda a detectar errores de estilo, mejorando la mantenibilidad del proyecto.
3. **Ejecución de Pruebas Automatizadas**
   - **Pruebas Unitarias**: Validan la funcionalidad de componentes individuales del sistema.
   - **Pruebas de Integración**: Verifican la interacción entre diferentes módulos para asegurar que el sistema funcione de forma coherente. Ambas pruebas se ejecutan para garantizar que los módulos críticos funcionen correctamente antes de ser desplegados.
4. **Despliegue Condicional**
   Si todas las pruebas y análisis previos son **exitosos**, el flujo de trabajo procede al **despliegue automático** en un entorno seguro (Render). Esto permite que el equipo asegure la calidad y estabilidad del sistema antes de su entrega final.

4. Buenas practicas

Para asegurar un desarrollo eficiente y mantener la calidad del código en el proyecto **innosoft-diplomas-1**, el equipo sigue un conjunto de buenas prácticas que garantizan la correcta implementación de la Integración Continua (CI) y el trabajo colaborativo:

1. **Commits Frecuentes y Descriptivos**
   Realizar commits de forma regular con **mensajes claros y detallados** que describan los cambios realizados. Esto facilita el seguimiento del historial de versiones y la resolución de problemas.

2. **Uso de Ramas por Funcionalidad (feature/)**
   - Crear ramas con el prefijo **feature/** para nuevas funcionalidades (por ejemplo, feature/generar-pdf, feature/validaciones-formulario).
   - Utilizar **fix/** para corrección de errores y **hotfix/** para solucionar problemas críticos en producción.
   - Una vez finalizada la implementación, abrir un **Pull Request (PR)** para su revisión y fusión con la rama preproduction.

3. **Convención de Commits** Para mantener un historial de commits organizado y comprensible, utilizamos una convención de **etiquetas** en los mensajes de commit:
   - **[feat]** - Para nuevas funcionalidades:
     - Ejemplo: [feat] Implementar generación automática de diplomas
   - **[fix]** - Para corrección de errores:
     - Ejemplo: [fix] Corregir error en la generación del PDF
   - **[docs]** - Para documentación:
     - Ejemplo: [docs] Actualizar README con instrucciones de despliegue
   - **[style]** (opcional) - Para cambios en el estilo y formato del código:
     - Ejemplo: [style] Ajustar indentación en views.py
   - **[refactor]** (opcional) - Para refactorización de código:
     - Ejemplo: [refactor] Simplificar la lógica de validación en forms.py

   Mantener los mensajes **claros y concisos**, y realizar **commits pequeños y frecuentes** en lugar de un gran commit al finalizar una tarea.

4. **Revisión de Pull Requests**
   Todo nuevo código debe ser **revisado por al menos un miembro del equipo** antes de su integración. Esto ayuda a detectar errores y asegurar que los estándares de calidad se cumplan.

5. **Ejecución Local de Pruebas Antes del Push**
   Antes de realizar un push al repositorio, ejecutar **todas las pruebas localmente** para asegurarse de que no haya errores que rompan el flujo de CI.

6. **Mantener el Archivo requirements.txt Actualizado**
   Asegurarse de que todas las **dependencias del proyecto** estén actualizadas en el archivo requirements.txt para que el pipeline de CI funcione correctamente.

7. **Documentación del Código**
   Incluir **comentarios en funciones y clases importantes** para que el equipo pueda comprender y mantener el código de manera eficiente.

8. **Eliminar Ramas Obsoletas**
   Una vez que una rama feature/ ha sido fusionada y desplegada correctamente, **eliminarla** para mantener el repositorio limpio y organizado.

9. **Monitoreo y Retroalimentación Continua**
   Revisar periódicamente el estado del pipeline de **GitHub Actions** y ajustar las configuraciones según sea necesario para mejorar la eficiencia del flujo de trabajo. Actuar rápidamente ante cualquier notificación de error para minimizar tiempos de inactividad.

5. Resultados esperados

Con la implementación de la **Integración Continua (CI)** y la **automatización del despliegue**, se espera lograr los siguientes beneficios para el proyecto **innosoft-diplomas-1**:

- **Reducción de riesgos asociados a procesos manuales**: La automatización asegura que las implementaciones sean consistentes y estén libres de errores humanos, lo que minimiza fallos inesperados en producción.
- **Replicación fiel entre entornos**: El entorno de producción será una **réplica exacta** del entorno de desarrollo, garantizando que todo funcione correctamente antes del despliegue final.
- **Proceso ágil de implementación**: La automatización permite **implementaciones rápidas y eficientes**, reduciendo el tiempo necesario para actualizar el sistema y liberando al equipo de tareas manuales.
- **Actualizaciones fluidas y sin interrupciones**: Al minimizar la posibilidad de errores durante el despliegue, se garantiza que las actualizaciones se realicen de manera **continua y sin interrupciones** para los usuarios.