ISPP 2024

Metodologías de desarrollo

Repositorio: https://github.com/ispp-2324-ocial

Miembros:

- Adrián Romero Flores
- Ramón J. Guerrero Romero
- Aitor Rodríguez Dueñas
- Fernando Jesús Fernández
- Francisco Manuel Villalobos Páez
- Horacio García Lergo
- Juan José Gómez Borrallo
- Iñigo Ruíz Marchueta
- Jesús Zambrana Guerra
- Eduardo Pizarro López
- Paula Peña Fernández
- José Carlos Ortiz Gutiérrez
- Carlos Varela Soult
- Raúl Montalbán Martín

GRUPO 2
FECHA: 20/02/2024 Versión 1.0

Índice

Historial de versiones

Fecha	Versión	Descripción	Entrega
16/02/2024	V1.0	Metodologías de Desarrollo y descripción de políticas	DP
18/02/2024	V1.1	Adición de Integración Continua y Correcciones	DP

1. Introducción

En este documento se describen las políticas y procedimientos que seguirá el grupo 2 para el desarrollo del proyecto de la aplicación 'Ocial' para la asignatura de ISPP. Se detallará tanto la gestión del código como la gestión de las incidencias y se mostrarán las plantillas necesarias para ello.

2. Gestión del Código

La gestión del código se llevará a cabo en GitHub con dos repositorios distintos, uno para Backend y otro para el Frontend. Las políticas en ambos repositorios serán las mismas:

2.1 Política de Ramas

Para gestionar el flujo de trabajo en el repositorio se ha optado por utilizar Gitflow, separando el desarrollo de cada funcionalidad en su rama feature correspondiente. Cuando la funcionalidad está lista y sus pruebas han sido también desarrollados, se lleva a develop (base para el desarrollo continuo y la integración de nuevas características) los cambios mediante pull-request revisadas por al menos el revisor de los cumplimientos de las políticas y/o algún desarrollador extra. Una vez todas las funcionalidades del Sprint están en develop, se llevan siguiendo la misma mecánica a master (rama principal del proyecto) y luego a la rama release (rama exclusiva para preparar las versiones del proyecto), para su despliegue en remoto. Las ramas master, develop y release está protegidas frente a push directos con reglas de GitHub.

2.2 Política de Commits

Se usará Convetional Commits, es decir, el título del commit (no más de 50 caracteres) viene precedido por un tag según lo que se intente subir al remoto y posteriormente una descripción opcional (no más de 80 caracteres). La estructura pues sería la siguiente:

```
git commit -m "tag: título commit" -m "Descripción commit"
```

Algunas de estos tags:

- **fix**: Utilizado cuando se realiza una corrección de errores.
- **feat**: Se emplea cuando se agrega una nueva característica o funcionalidad.
- **docs**: Usado para cambios relacionados con la documentación.
- **style**: Se utiliza para cambios que no afectan el comportamiento del código, como cambios en el formato o estilo de código.
- **refactor**: Utilizado para cambios en el código que no corrigen errores ni añaden nuevas funcionalidades, pero mejoran la estructura o legibilidad del código.
- **test**: Se emplea para agregar o modificar pruebas unitarias o de integración.
- **chore**: Utilizado para cambios en el proceso de compilación o tareas de mantenimiento.
- **perf**: Usado para mejoras de rendimiento.
- **revert**: Se emplea cuando se revierte un commit previo.

2.3 Política de Versiones

Se usará el versionado semántico X.Y.Z donde:

- X: Será cada cambio importante en la aplicación no tiene por qué coincidir con los deliberarles y/o Sprints.
- Y: Será las features añadidas al proyecto
- Z: Serán arreglos de errores (fixes).

Cada apartado incrementa en 1.

3. Gestión de Incidencias

Para detallar la gestión de incidencias lo primero es definir las incidencias en el contexto de nuestro proyecto. Con incidencias nos referimos a cualquier evento o situación que no está planeado y sea interesante implementar porque pueda afectar positivamente al desarrollo, la calidad o sea necesaria su implementación en el software para el correcto funcionamiento de este.

Las incidencias se gestionarán mediante las issues en GitHub. Para proceder debe tener claro el alcance de esta incidencia, es decir, de que tipo es (configuración, bug, funcionalidad) y a qué aspectos del proyecto afecta (módulos a los que afecta). Esta clasificación se realizará mediante los tags definidos en el proyecto GitHub. A todo esto, hay que incluir los desarrolladores a quienes se asigne la incidencia y se deben crear ramas asociadas a estas issues (por ejemplo “feature/01-Nuevo-campo-formulario”). Adicionalmente a esto en la descripción de la issue se debe rellenar la siguiente plantilla.

```
# Incidencia – XX (donde XX es la ID de la Incidencia)

## Descripción de la incidencia
[Descripción detallada de la incidencia, incluyendo cualquier información
relevante como contexto, comportamiento esperado vs. observado, etc.]

## Alcance de la incidencia
[Indica hasta dónde se extiende esta incidencia, si afecta a otras áreas
del proyecto, módulos, funcionalidades, etc.]

## Replicación de la incidencia
### Pasos para replicar:
1. [Paso 1]
2. [Paso 2]
3. [Paso 3]
...
[Incluye todos los pasos necesarios para replicar la incidencia]

### Comportamiento esperado
[Describe qué debería ocurrir en lugar de la incidencia]

### Comportamiento observado
[Describe qué ocurre realmente cuando se siguen los pasos de replicación]

## Solución propuesta
[Propuesta para solucionar la incidencia. Esto puede incluir cambios de
código, configuración, o cualquier otra acción necesaria para resolver el
problema]
```

4. Aseguramiento de la Calidad

Para asegurar la calidad del proyecto se ha definido un cierto umbral mínimo de cobertura de pruebas de un 70%. Esta cobertura se medirá con la herramienta SonarCloud, en la que se revisarán los *bad smells* y malas prácticas en el código.

En cuanto al diseño de las pruebas se centrarán sobre todo en la realización de pruebas funcionales por la parte de Backend y de pruebas de vistas en la parte de Frotend. Eventualmente se podrán diseñar algunas pruebas de carga y rendimiento.

5. Integración Continua

Dentro de los repositorios de Github por cada pull-request se realizan previamente ciertas GitHub Actions:

- Una acción para revisar el etiquetado.
- GitHub CodeQL: para realizar análisis exhaustivos en busca de vulnerabilidades de seguridad, errores de programación y otros problemas potenciales en el código.
- Una acción para el escaneo de dependencias para vulnerabilidades.
- Una acción para el despliegue a CloudFlare Pages (para alojar la web).
- Una acción para la construcción del contenedor.
- Una acción para la publicación de comentario con el estado del despliegue.
- Lint: Para revisar los conventional commits y que la estructura propuesta se ha seguido en estos.
- Una acción que ejecuta todos los tests.
- Una acción que comprueba los tipos.

Luego cuando se proceda pasar el código a master se añaden estas acciones:

- Una acción para el despliegue a Cloudflare.
- Una acción para la construcción y despliegue del contenedor.

Sumado a esto en el repositorio de Backend se añade una acción con la herramienta Black que formatea el código de Python, modificando el estilo del código para hacerlo consistente y legible de manera automática.

6. Bibliografía

Intencionadamente en blanco.