Acta fundacional

Evaluación y Gestión de la Configuración

track-hub-2

Fecha: 15/10/2025

Elaborado por:
Adolfo Borrego González
German Sanchez Carmona
Roberto Muñoz Perez
Jesús Salas Muñoz
José Egido Carnerero
Enrique Pérez Milla

Índice

1. Introducción

Este Acta Fundacional establece de forma clara la forma de trabajar del equipo para todo el proyecto. Define cómo nos organizamos, nos comunicamos y decidimos el flujo de trabajo (WIs, ramas, commits, versionado), los criterios de calidad y CI/CD, el manejo de incidencias y el protocolo de resolución de conflictos, así como medidas graduales en caso de incumplimientos.

2. Política de ramas

Para realizar una correcta gestión de las ramas durante el desarrollo del proyecto se establecerá la siguiente estructura de ramas:

- Rama ***main***: Esta será la rama que contendrá la última versión completamente operativa de nuestros cambios en forma de release. Esta rama debe permanecer siempre estable y sin fallos.
- Rama ***trunk***: En esta rama se realizarán las fusiones de las distintas actividades con el fin de evitar la introducción de errores fatales en la rama *main*.
- Ramas ***feature/<WI-<número>-<nombre_identificativo>***: Estas ramas se crearán a partir de la rama develop para el desarrollo de las diferentes funcionalidades a implementar.
- Rama **hotfix/<*nombre_identificativo*>**: Correcciones urgentes para incidencias que surjan en la rama *main* (en producción).
- Rama ***bugfix/<nombre_identificativo>***: Correcciones para incidencias que ocurran en la rama *trunk* (en preproducción).

3. Política de commits

Para la política de commits utilizaremos la siguiente estructura:

`<type>(<scope>): <subject> <body> <footer>`

Los posibles `<type>` que se han establecido son:

- **feat** → nuevas funcionalidades o nuevas integraciones en las mismas.
- **fix** → arreglo de bugs y fallos.
- **docs** → adición o cambios en la documentación.
- **refactor** → refactorización del código.
- **test** → nuevos tests, cambios o su refactorización.
- **chore** → tareas que no incumben al código de la aplicación, como pueden ser dependencias, configuración, etc.

El `<scope>` es opcional, sirve para identificar en el caso de haber cambiado un único archivo cuál es la localización del mismo. En caso de no tener `<scope>` se eliminarán los paréntesis que lo contienen.

El `<subject>` es el título del commit, con una descripción breve del cambio.

El `<body>` es opcional, es la descripción del cambio realizado y la causa del cambio.

El `<footer>` es opcional, se usa para hacer referencia a las issues, como por ejemplo a la hora de cerrarlas (Closes #123).

4. Política de versionado

Este proyecto utiliza un sistema de control de versiones semántico (Semantic Versioning), siguiendo el formato:

`MAJOR.MINOR.PATCH`

- `MAJOR`. Se incrementa cuando se realizan cambios incompatibles con versiones anteriores (breaking changes) en la API o en el comportamiento público del software.
    - Eliminación o modificación de endpoints.
    - Cambios en estructuras de datos.
    - Ajustes que requieran intervención manual para actualizar.
- `MINOR`. Se incrementa cuando se añaden nuevas funcionalidades de forma compatible con versiones anteriores.
    - Nuevas funciones.
    - Opciones o mejoras no disruptivas.
- `PATCH`. Se incrementa cuando se aplican correcciones de errores o mejoras menores que no alteran la compatibilidad ni introducen nuevas funcionalidades.
    - Fix de bugs.
    - Optimización de código.
    - Ajustes en la documentación.

5. Ciclo CI/CD

Usando lo aprendido en las sesiones de teoría de relaciones a este punto, y siguiendo la estructura del ciclo CI/CD que podemos ver en el repositorio oficial del proyecto uvlhub (enlace), esbozamos nuestro ciclo mínimo de la siguiente forma:

- Integración Continua:
  - **commits**: se ejecuta en *push* y *pull_request* para comprobar que el formato de los mensajes de commit se ajuste al definido en el apartado de *Política de commits*, en nuestro caso *Conventional commits*.
  - **linter**: se ejecuta en *push* y *pull_request y* lanza flake8 para revisar el estilo y calidad del código.
  - **tests**: se ejecuta en *push* y *pull_request* en *main* y lanza pytest para los tests unitarios y de integración.
- Despliegue continuo:
  - **render**: se ejecuta en *push* y *pull_request* y realiza el despliegue en render.com.

6. Gestión de conflictos

Se distinguen los siguientes tipos de conflictos:

- **Participación irregular**: algún miembro del equipo no trabaja lo suficiente en comparación al resto, no muestra el suficiente interés o no atiende a las comunicaciones del equipo. Si se vuelve un problema recurrente y no está justificado por causa mayor, se comunicará debidamente al profesor para gestionar la situación de la mejor manera posible.

- **Choques de opinión**: algunos miembros tienen puntos de vista muy distintos acerca de algún tema del trabajo y entran en conflicto. Se tratará de llegar a un acuerdo entre las partes o en su defecto, se acudirá al profesor para que actúe de mediador.

- **Trabajo pobre debido a diferencias en habilidad**: si alguna tarea ha sido asignada a un miembro del equipo que no tiene completamente adquiridos los conocimientos necesarios para su resolución, se le intentará en la medida de lo posible que los miembros con mejor dominio que él en dicha tarea aporten asistencia y apoyo. No califica como este conflicto si el miembro que tenga que realizar la tarea no muestra el suficiente interés por solucionarla y no ha hecho al menos trabajo de investigación previo por su cuenta.

7. Gestión de incidencias

Con respecto a las incidencias, en el momento donde un miembro del grupo se encuentre con un bug durante el desarrollo, este deberá crear una issue en GitHub, indicando en esta el error producido y la sucesión de pasos que ha llevado a esa situación, por último, se establecerá un nivel de prioridad y de urgencia de la incidencia en cuestión, seguida de una breve justificación de ambos. Está Issue deberá ser resuelta y tratada siguiendo los pasos vistos en las clases de teoría:

1.Informar de la incidencia → 2.Reproducir → 3.Diagnosticar → 4.Arreglar → 5.Analizar

8. Firmas

Miembro	Firma
Adolfo Borrego González	
German Sanchez Carmona	
Roberto Muñoz Perez	
Jesús Salas Muñoz	
José Egido Carnerero	
Enrique Pérez Milla	