Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática

ACTA FUNDACIONAL

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

EGC-2425-RaboDeToro

-hub-1 Curso 2024 –

2025

17-10-2024

Índice

1. Resumen ejecutivo

El presente documento establece de manera clara y sintética las normas, acuerdos y procedimientos que regirán el trabajo del equipo durante el desarrollo del proyecto **RaboDeToro-hub-1**. Este proyecto tiene como objetivo principal desarrollar un sistema funcional y de alta calidad, asegurando la estabilidad, escalabilidad y alineación con los hitos definidos:

- Hito 1 (M1): Sistema funcionando y pruebas (23/10/2024)
- Hito 2 (M2): Sistema funcionando y con incrementos (13/11/2024)
- Hito 3(M3): Entrega de proyectos y defensas (18/12/2024)

Para alcanzar los objetivos, el equipo acuerda seguir una estructura organizativa y regirse por las políticas establecidas en este documento.

Este documento será firmado por todos los integrantes del equipo y será revisado o modificado con el acuerdo de todos los miembros.

2. Contenido

A. Conductas y sanciones

A continuación, se tratará el tema de acuerdos y sanciones para poder mantener el orden dentro del equipo:

> Cualquier integrante que no esté colaborando lo suficiente recibirá un aviso, hasta un máximo de 3 avisos. Al llegar al límite de avisos, se contactará con el coordinador para que lo tenga en cuenta al evaluar.
>
> Se intentará mantener siempre una actitud positiva dentro del entorno de equipo y con la mayor comunicación posible para mantener una buena coordinación.
>
> Cualquier conflicto que ocurra deberá resolverse de manera constructiva. Si esto no sucede, se avisará automáticamente al coordinador, lo que podría resultar en la expulsión del compañero del equipo.
>
> El cumplimiento de la política de commits y la política de código es absolutamente necesario. Se penalizará con un aviso a quienes ignoren reiteradamente estas políticas de forma consciente.

B. Política de commits

En este apartado se establecen las políticas que se seguirán para los mensajes de commits en el proyecto:

1. **Separar el asunto del cuerpo con una línea en blanco**

   Correcto:

   ```
   Añadir nueva función de inicio de sesión

   Este commit añade una nueva función de inicio de sesión que permite a los usuarios iniciar sesión con su correo electrónico.
   ```

   Incorrecto:

   ```
   Añadir nueva función de inicio de sesión. Este commit añade una nueva función de inicio de sesión que permite a los usuarios iniciar sesión con su correo electrónico.
   ```

2. **Limitar la línea del asunto a 50 caracteres**

   Correcto:

   ```
   Añadir nueva función de inicio de sesión
   ```

   Incorrecto:

   ```
   Añadir una nueva función de inicio de sesión que permite a los usuarios iniciar sesión con su correo electrónico.
   ```

3. **Poner en mayúsculas la línea del asunto**

   Correcto:

   ```
   Añadir nueva función de inicio de sesión
   ```

   Incorrecto:

   ```
   añadir nueva función de inicio de sesión
   ```

4. **No terminar la línea del asunto con un punto**

   Correcto:

   ```
   Añadir nueva función de inicio de sesión
   ```

   Incorrecto:

   ```
   Añadir nueva función de inicio de sesión.
   ```

5. **Usar el modo imperativo en la línea del asunto**

   Correcto:

   ```
   Añadir nueva función de inicio de sesión
   ```

   Incorrecto:

   ```
   Añadida nueva función de inicio de sesión
   ```

6. **Ajustar el cuerpo a 72 caracteres**

   Correcto:

   ```
   Este commit añade una nueva función de inicio de
   sesión que permite a los usuarios iniciar sesión con
   su correo electrónico.
   ```

   Incorrecto:

   ```
   Este commit añade una nueva función de inicio de
   sesión que permite a los usuarios iniciar sesión con
   su correo electrónico y su contraseña.
   ```

Adicionalmente, vamos a seguir *Conventional Commits* a la hora de realizar commits a lo largo del proyecto.

Esta convención se basa en el uso de prefijos de tipo en los mensajes de commit para indicar el tipo de cambio que se ha realizado. Estos prefijos son verbos en inglés y en infinitivo que intenta describir el tipo de cambio que se ha realizado. Los tipos de cambio más comunes y que utilizaremos son los siguientes:

- **fix**: cuando se corrige un error en el código base

  Ejemplo:

  ```
  fix: Corregir error en el inicio de sesión
  ```

- **feat**: cuando se añade una nueva funcionalidad al código

  base Ejemplo:

  ```
  feat: Añadir nueva función de inicio de sesión
  ```

- **docs**: cuando se modifica la

  documentación Ejemplo:

  ```
  docs: Actualizar README
  ```

- **style**: cuando se realizan cambios que no afectan al significado del código (espacios en blanco, formato, punto y coma que falta, etc.)

  Ejemplo:

  ```
  style: Corregir errores de formato
  ```

- **test**: cuando se añaden pruebas faltantes o se corrigen pruebas existentes

  Ejemplo:

  ```
  test: Añadir pruebas para la función de inicio de sesión
  ```

Cuando se realice un commit, si este tiene asignado un issue en *Github*, se deberá añadir el número del issue en el pie del commit de la siguiente manera:

```
Refs #numero-issue
```

En conclusión, los mensajes de commit seguirán la siguiente estructura:

```
<prefijo>: <asunto>
     <linea en
  blanco> [cuerpo
     opcional]
 <linea en blanco>
 [pie opcional]
```

C. Política de código

La política de código en este proyecto sigue las convenciones estándar de Python, aplicando un estilo limpio y consistente para garantizar la legibilidad y mantenibilidad. Se utilizará **snake_case** para nombres de funciones, métodos y variables (por ejemplo, user_id o get_user_data), mientras que los nombres de clases seguirán el formato **PascalCase** (por ejemplo, UserRepository). Las funciones y métodos deben ser modulares y cumplir una única responsabilidad, facilitando la comprensión y el mantenimiento del código. Además, los nombres de las variables y funciones serán descriptivos y autoexplicativos, evitando abreviaturas ambiguas (por ejemplo, total_users es preferible a tu).

D. Política de ramas

En nuestro proyecto seguiremos la estrategia de ramificación conocida como *GitFlow* para gestionar las ramas del proyecto. Distinguiremos dos tipos de ramas:

Ramas principales:

- **main**: será la rama principal del proyecto y contendrá el código que se encuentra en producción, y contiene la última versión estable del proyecto.
- **develop**: será la rama de desarrollo del proyecto y contendrá el código que se encuentra en desarrollo, y contiene la última versión en desarrollo del proyecto.

Ramas de soporte: Paralelamente a las ramas principales, se crearán ramas de soporte para realizar tareas específicas, nos ayudarán a facilitar el seguimiento de nuevas features.

Las ramas de soporte se dividen en dos tipos:

> **Ramas de 'features'**: se utilizan para desarrollar nuevas funcionalidades del proyecto. Se crean a partir de la rama *develop* y se fusionan en ella una vez que la funcionalidad está completa.
> **Ramas de 'hotfix'**: se utilizan para solucionar errores que se encuentren en producción. Se crean a partir de la rama *develop* y se fusionan en las ramas *main* y *develop* una vez que el error está solucionado.

Cómo desarrollar ramas de 'feature'

Cuando se vaya a desarrollar una nueva funcionalidad, se creará una rama de *feature* a partir de la rama *develop* con el siguiente comando:

```
git checkout -b feature/nombre-de-la-funcionalidad/numero-issue develop
```

Una vez que la funcionalidad esté completada:

Crear una pull request de nuestra rama de *feature* a la rama *develop*, incluyendo el nombre del issue al final del nombre de la pull request.

```
git push origin feature/nombre-de-la-funcionalidad/numero-issue
```

Para revisar la pull request y fusionarla en la rama *develop*, se seguirá el proceso de revisión por pares.

E. Frecuencia de Merge develop → main

En nuestro proyecto, hemos definido una política clara para la integración de cambios en la rama principal (main). El proceso de merge desde la rama de desarrollo (develop) hacia main se llevará a cabo una vez por milestone. No obstante, si todo el equipo está de acuerdo, se podrá realizar el merge en cualquier otro momento, ya sea antes o después de un milestone, siempre que el trabajo desarrollado haya sido revisado y validado en su totalidad.

Esta práctica asegura que cada integración se realice con consenso, garantizando la estabilidad y funcionalidad del proyecto. Solo después de completar estas verificaciones, se procederá a la integración de los cambios en **main**, consolidando así los avances alcanzados. Esta estrategia fomenta

un desarrollo estructurado y minimiza posibles errores en la rama principal.

F. Política de Pull Request

Todas las Pull Requests del proyecto deben seguir la siguiente estructura:

- **Título:** el título o nombre de la pull request debe estar formado por:

```
[fix/feat/test/docs/style]: [Asunto en modo imperativo]
```

- **Descripción:** Debe tener una pequeña descripción sobre lo que se ha realizado en dicha pull.
- **Etiquetas:** Se asignan etiquetas relevantes (e.g., documentation, bug, workflows) para clasificar y facilitar la comprensión de la naturaleza del cambio.
- **Milestones: S**e asocian con un milestone específico (e.g., M3) para vincular el cambio con el ciclo de desarrollo correspondiente.
- **Revisores:** El autor de la Pull Request asigna un revisor del equipo, quien se encarga de verificar el código y proporcionar feedback.
- **Asignaciones**: El autor también puede autoasignarse para señalar su responsabilidad en el cambio.
- **Project:** Pull Request relevante se vincula al **Project** correspondiente, lo que permite rastrear su estado dentro del flujo de trabajo.

Procedimiento

Cada cambio del código que se realice en el proyecto será revisado por al menos un miembro del equipo. Para ello, se seguirá el siguiente proceso:

1. El autor de la pull request solicitará una revisión a un miembro del equipo.
2. El miembro del equipo revisará el código y realizará los comentarios que considere necesarios.
3. El autor de la pull request realizará los cambios necesarios en el código, en el caso de tener un comentario negativo.
4. El miembro del equipo revisará de nuevo el código y, si todo está correcto, aprobará la pull request.

G. Política de Issues

En nuestro proyecto todas las Issues seguirán las siguientes pautas:

- **Estructura**: la definición de las Issues se basa en:
  - **Título**: breve texto que engloba el objetivo principal de la Issue en español.
    Todas la relativas a un WorkItem serán nombradas como `WI-nombre del workitem.`
    Todas las relativas a las tareas técnicas serán nombradas por `nombre de la tarea específica.`
  - **Descripción**: Texto que describe la Issue y su funcionalidad en español.
  - **Assignes**: Usuario de Github de la persona encargada del desarrollo de la Issue.
  - **Project:** Issue relevante se vincula al **Project** correspondiente, lo que permite rastrear su estado dentro del flujo de trabajo.
  - **Labels**: etiquetas de prioridad y tipo de Issues. Las etiquetas definidas en el proyecto son:
    - **Prioridad**:

    ➔ Low priority: Puede resolverse en cualquier momento.
    ➔ Medium priority: Necesita resolverse pronto.
    ➔ High priority: Necesita resolverse primero

    - **Tipos**:

    ➔ Add: Agregar al código principal.
    ➔ Bug: Algo no está funcionando.
    ➔ CI (Continuous Integration: Ayuda a la integración continua.
    ➔ CD (Continuous Deployment): Ayuda al despliegue continuo.
    ➔ Documentation: Mejoras o adiciones a la documentación.
    ➔ Fakenodo: Elemento de trabajo con implementación de fakenodo.
    ➔ Hard: Elemento de trabajo con implementación difícil.
    ➔ Low: Elemento de trabajo con implementación sencilla.
    ➔ Medium: Elemento de trabajo con implementación media.
    ➔ Test: Verificación de funcionalidad.
    ➔ Workflows: Implementar nuevos workflows de GitHub
  - **Development**: Indica la rama en la que se ha desarrollado la Issue.
  - **Milestone**: Indica el Milestone al que pertenece (M1, M2, M3)
  - **Estados:** Las issues dentro de los proyectos se gestionan a

través de un flujo definido por diferentes columnas que representan los estados del trabajo.

- **Todo:** Representa las tareas que aún no se han comenzado. Estas issues están pendientes y generalmente se encuentran en la etapa inicial de planificación.
- **In Progress:** Indica las tareas en las que el equipo está trabajando activamente. Estas issues están en proceso de desarrollo o implementación.
- **Test:** Estas issues están siendo probadas, lo que implica que el desarrollo está completo y están en fase de verificación para asegurar su calidad y funcionalidad.
- **In Review:** Representa las tareas que están en proceso de revisión por parte del equipo. Esto incluye revisiones de código o revisiones generales para garantizar que cumplen con los estándares del proyecto.
- **Done:** Estas issues han sido completadas y marcadas como finalizadas. No requieren más trabajo y han pasado todas las etapas anteriores con éxito.

H. Documentos

Toda la documentación del proyecto se gestionará en el directorio /docs dentro del repositorio, incluyendo:

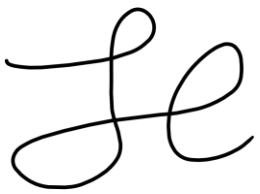
- Acta fundacional del proyecto.
- Presentación de los Works ítems.
- Definición de procesos.

Además, tenemos un readme que proporciona una visión general, incluyendo tecnologías usadas, instrucciones de instalación y enlaces clave como el despliegue del proyecto. Es el punto de partida recomendado para nuevos usuarios o desarrolladores.

También estará habilitada la wiki de GitHub, donde estará la descripción del proyecto y el diario del equipo. Enlace: `https://github.com/EGC2425-RaboDeToro-hub/RaboDeToro-hub-1.wiki.git`

I. Miembros del equipo

Por último**,** se establecerán quienes son los integrantes del grupo junto a su foto y firma, mostrando acuerdo con todo lo mencionado en este documento:

Nombre	Foto	Firma
Juan Antonio Moreno Moguel (Coordinador)		
José María Morgado Prudencio		
Virginia Mesa Pérez		
Paula Sánchez Gómez		
Mario Reyes Apresa		
Paula María Suárez Linares