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

En este documento se establecerá de manera clara y sintética cuáles serán los miembros del grupo y cuáles serán los acuerdos que regirán al mismo, incluyendo sanciones y acuerdos. Este documento deberá estar firmado por todos los integrantes del equipo y solo será revisado y modificado de manera excepcional y con el acuerdo de todos los integrantes.

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 se aplicará de la siguiente manera: Se creará un código limpio, utilizando el estilo de escritura camelCase (por ejemplo, `int variableEjemplo`), con funciones atómicas que realicen una única tarea. El código estará comentado de manera precisa y necesaria para facilitar su comprensión. Además, los nombres de las variables deben ser descriptivos y justificados (por ejemplo, `int ruedasDeCoche` es aceptable, mientras que `int rdc` no lo es).

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 *main* 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.

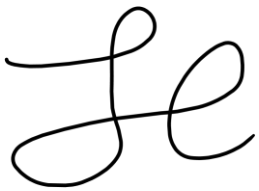
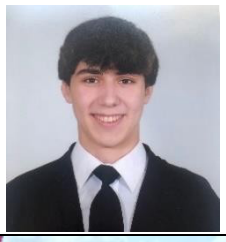
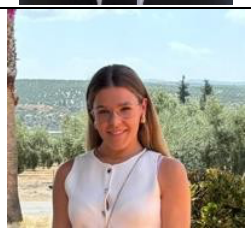
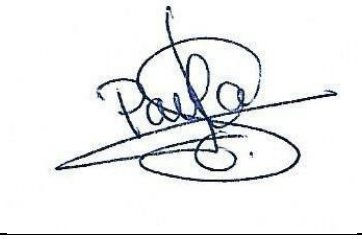
E. Política de Pull Requests

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.
4. El miembro del equipo revisará de nuevo el código y, si todo está correcto,aprobará la pull request.

F. 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