Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática

Devising A Project

BunquetBuddy

Gestión del código fuente

Grado en Ingeniería Informática – Ingeniería del Software

Ingeniería del Software y Práctica Profesional

Curso 2023 – 2024

Fecha	Versión
16/02/2024	1.0

Grupo de prácticas: 8
Alberto Benitez Morales
Álvaro Carrera Bernal
Álvaro Navarro Rivera
Álvaro Jose Sanchez Flores
Artemio Rodriguez Asensio
Eduardo de Bustamante Lucena
Fernando Barroso Barroso
Francisco Jose Vargas Castro
Gonzalo Santigo Martín
Guillermo Alonso Pacheco Rodrigues
Jaime Caballero Hernandez
Javier Nunes Ruiz
Javier Rodríguez Cordero
Juan Martínez Cano
Marco Antonio Roca Rodríguez
Mario Sanchez Naranjo
Pablo Martínez Valladares

Control de Versiones

Fecha	Versión	Descripción
16/02/2024	1.0	Creación del documento
18/02/2024	1.1	Revisión del documento

Gestión del código fuente

En el proyecto en el que participamos, compuesto por 17 personas, es crucial establecer una sólida política de gestión del código fuente y del repositorio. Esto asegurará la eficiencia, la colaboración y la calidad del código producido. A continuación, detallamos nuestras decisiones y prácticas clave.

Política de ramas: Git Flow

Hemos optado por adoptar la metodología Gitflow para la gestión de ramas. Gitflow es un modelo de branching que facilita el desarrollo paralelo de características, versiones y hotfixes. En resumen, este flujo se compone de las siguientes ramas principales:

1. **Master (Main):** Representa la rama de producción estable. El código aquí refleja siempre una versión estable y lista para ser implementada en producción.
2. **Develop:** Esta rama es donde se integran todas las características completadas. Se considera la rama "en desarrollo" y sirve como base para todas las nuevas características.

Además de estas ramas principales, se crean ramas de características, hotfixes y versiones para desarrollar funcionalidades, corregir errores y gestionar versiones del software.

Todas las ramas deberán ser nombradas en inglés.

Para dejar claro el funcionamiento veamos un ejemplo de creación de ramas para cada una de las posibles casuísticas.

- Creación de una rama para una nueva característica:
    - feature/<<nombre_nueva_caracterísitca>>
        - feature/search_catering
        - feature/book_catering
- Creación de una rama de corrección:
    - hotfix/<<nombre_de_corrección>>
        - hotfix/fix_security_vulnerability
- Creación de una rama de publicación:
    - release/<<número_versión>>
        - release/1.0.0

Política de versionado

El proyecto seguirá la convención de versionado semántico para facilitar la comprensión de la versión. El número de versión se representará como x.y.z, donde:

- **X, número de versión principal:** Aumentará cuando se realicen cambios estructurales, es decir, cuando se introduce una nueva arquitectura, funcionalidades principales o cambios que rompen la compatibilidad.
    - *Ejemplos*: 1.0, 2.0, 3.0.

- **Y, número de versión secundaria:** Aumentará cuando se introduzcan nuevas características que no afecten la compatibilidad con las versiones anteriores.
    - *Ejemplos*: 1.1, 1.2, 2.1.
- **Z, número de la versión de parche:** Aumenta cuando se corrigen errores, se optimiza el rendimiento o se realizan pequeñas mejoras.
    - *Ejemplos*: 1.0.1, 1.2.2, 2.1.3.

Aquí podemos ver un ejemplo de la evolución del versionado:

I. *1.0.0*: Versión inicial del software.
II. *1.1.0*: Se añade una nueva función sin afectar la compatibilidad con la versión 1.0.0.
III. *1.2.1*: Se corrige un error menor en la versión 1.1.0.
IV. *2.0.0*: Se introduce una nueva arquitectura que rompe la compatibilidad con versiones anteriores.
V. *2.1.0*: Se añade una nueva función a la versión 2.0.0 sin afectar la compatibilidad.
VI. *2.1.3*: Se corrige un error de seguridad en la versión 2.1.0.

Política de commits: Conventional Commits

Para mantener la coherencia y la comprensión del historial de cambios, hemos decidido adoptar el estándar de Conventional Commits. Este enfoque define un conjunto de reglas para crear mensajes de commit significativos y estructurados, que incluyen un prefijo indicativo del tipo de cambio ("feat" para una nueva característica, "fix" para una corrección de errores, etc.) seguido de una descripción concisa del cambio.

Para dejar claro el funcionamiento veamos algunos ejemplos de cómo realizar un commit para cada una de las posibles casuísticas.

- Commit para una nueva funcionalidad:
    - feat:<<título>> [breve descripción opcional]
    - feat: Add menu management feature
- Commit para una corrección:
    - fix:<<título>> [breve descripción opcional]
    - fix: Fix event booking functionality
- Commit referentes a documentación.
    - doc:<<título>> [breve descripción opcional]
    - doc: Update installation guide with new dependencies
- Commit referentes a refactorización:
    - refactor:<<título>> [breve descripción opcional]
    - refactor: Improve report generation code
- Commit referentes a la realización de test:
    - test:<<título>> [breve descripción opcional]
    - Add tests for booking confirmation feature

GitHub Actions

GitHub Actions te permite automatizar tareas repetitivas en tu proyecto, como ejecutar pruebas unitarias, realizar integraciones continuas, implementar automáticamente el código y enviar notificaciones. Esto asegurará que cada cambio propuesto pase por una serie de pruebas automatizadas antes de ser integrado en las ramas principales.

Algunos ejemplos de automatización con GitHub Actions incluyen:

- **Pruebas unitarias:** Asegurar que cada cambio de código pase las pruebas antes de ser integrado.
- **Integración continua:** Compilar y empaquetar el código automáticamente después de cada cambio.
- **Despliegue automático:** Implementar el código nuevo en un servidor de prueba o producción sin necesidad de intervención manual.
- **Notificaciones:** Enviar correos electrónicos o mensajes de Slack al equipo sobre el estado de las pruebas y las implementaciones.

Revisión de Pull Requests

Dado que es fundamental mantener la integridad y la calidad del código, hemos establecido un proceso de revisión de Pull Requests riguroso. Se configurará github de manera que la rama develop y la rama main (ramas principales) queden protegidas evitando que se puedan hacer push directamente sobre ellas.
Al menos una persona revisará todas las Pull Requests dirigidas a la rama Develop, mientras que al menos dos personas revisarán las Pull Requests dirigidas a la rama Main (Master), antes de su fusión. Esta revisión se centrará en la funcionalidad, calidad del código, pruebas y cumplimiento de estándares.

Gestión de Issues

Utilizaremos el sistema de seguimiento de problemas de GitHub (Issues) para gestionar y priorizar las tareas, errores y solicitudes de características. Cada problema se clasificará con etiquetas que indiquen su prioridad (por ejemplo, "alta", "media", "baja"), su tipo (por ejemplo, "bug", "mejora", "nueva característica") y en función del subgrupo al que pertenezca la tarea (“subgrupo 1”, “subgrupo 2”, “subgrupo 3”, “subgrupo 4”).
Se creará una plantilla que facilite la redacción de las issues en función de su naturaleza facilitando la rápida compresión y la estandarización de este proceso.

GitHub Projects.

Para el seguimiento de requisitos e issues, utilizaremos el tablero Kanban de GitHub. Este tablero nos permitirá visualizar el flujo de trabajo de manera clara, desde la creación de una issue hasta su resolución. Podremos mover las issues a través de diferentes columnas que representarán estados como "To Do", "In Progress", "In Review" y "Done", facilitando así la coordinación y el seguimiento del trabajo realizado.