



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



TFG del Grado en Ingeniería Informática

Asistente de prácticas ágiles
para repositorios en GitHub
Documentación Técnica



Presentado por Lucas Olmedo Díez
en Universidad de Burgos — Junio de 2025
Tutor: Carlos López Nozal

Índice general

Índice general	i
Índice de figuras	iii
Índice de tablas	v
Apéndice A Plan de Proyecto Software	1
A.1. Introducción	1
A.2. Planificación temporal	1
A.3. Estudio de viabilidad	13
Apéndice B Especificación de Requisitos	19
B.1. Introducción	19
B.2. Objetivos generales	19
B.3. Catálogo de requisitos	20
B.4. Especificación de requisitos	21
Apéndice C Especificación de diseño	35
C.1. Introducción	35
C.2. Diseño de datos	35
C.3. Diseño arquitectónico	38
C.4. Diseño procedimental	44
Apéndice D Documentación técnica de programación	51
D.1. Introducción	51
D.2. Estructura de directorios	51
D.3. Manual del programador	52

D.4. Compilación, instalación y ejecución del proyecto	54
D.5. Pruebas del sistema	55
Apéndice E Documentación de usuario	57
E.1. Introducción	57
E.2. Requisitos de usuarios	57
E.3. Instalación	58
E.4. Manual del Usuario	59
Apéndice F Anexo de sostenibilización curricular	67
F.1. Introducción	67
F.2. Competencias de Sostenibilidad en el proyecto	68
F.3. Conclusión	69
Bibliografía	71

Índice de figuras

A.1. Gráfico Burndown del Sprint 3 - Comparación temporal	2
A.2. Ejemplo de una tarea vista en el tablero Kanban de Zube	3
A.3. Ejemplo de un Sprint en Zube	3
A.4. Ejemplo del estado de un sprint visto en el tablero Kanban	4
A.5. Ejemplo de comentarios en Zube del tutor y alumno en una tarea del proyecto	5
A.6. Gráfico Burndown del Sprint 0 - Kick-off	7
A.7. Gráfico Burndown del Sprint 1 - Primer prototipo de la aplicación	8
A.8. Gráfico Burndown del Sprint 2 - Comparación temporal	9
A.9. Imagen de la primera versión de la interfaz de la aplicación	10
A.10.Imagen de la interfaz tras la maquetación del Sprint 3 - Diseño de la estética de la interfaz de usuario	10
A.11. Gráfico Burndown del Sprint 3 - Diseño de la estética de la interfaz de usuario	11
A.12. Gráfico Burndown del Sprint 4 - Despliegue continuo	13
 B.1. Diagrama de casos de uso general	26
C.1. Diagrama Entidad - Relación de la aplicación del proyecto	36
C.2. Diagrama Entidad - Relación UML de la aplicación del proyecto	36
C.3. Diagrama del diseño arquitectónico	39
C.4. Diagrama del directorio modules del frontend	40
C.5. Diagrama del caso de uso CU1	44
C.6. Diagrama del caso de uso CU2	45
C.7. Diagrama del caso de uso CU4	47
C.8. Diagrama del caso de uso CU5	48
C.9. Diagrama del caso de uso CU7	49

E.1. Inicio de sesión del usuario	59
E.2. Registro del usuario	59
E.3. Configuración del usuario	61
E.4. Validación de URLs de repositorios	62
E.5. Configuración del análisis de los repositorios	63
E.6. Gestión de grupos de repositorios	64
E.7. Evaluación de buenas prácticas ágiles	65
E.8. E6: Detalle de las prácticas ágiles	65
E.9. Comparación con repositorios de referencia	66

Índice de tablas

A.1. Costes de <i>personal</i>	14
A.2. Costes de <i>hardware</i>	14
A.3. Costes totales del asistente inteligente para prácticas ágiles	15
B.1. CU-1 Creación y acceso a la cuenta.	27
B.2. CU-2 Configuración de la cuenta.	28
B.3. CU-3 Introducción y validación de URLs de repositorios.	29
B.4. CU-4 Configurar análisis del repositorio	30
B.5. CU-5 Gestionar grupos de repositorios de comparación	31
B.6. CU-6 Visualizar evaluación de buenas prácticas ágiles	32
B.7. CU-7 Comparar con repositorios de referencia	33
B.8. CU-8 Navegar por la aplicación	34

Apéndice A

Plan de Proyecto Software

A.1. Introducción

El Plan de Proyecto Software describe el proceso de desarrollo de la aplicación web documentada en esta memoria. Describe los procesos seguidos durante la planificación y su seguimiento para el desarrollo continuo del proyecto. Para la planificación del proyecto se ha usado la aplicación **Zube**, que permite un seguimiento de las *issues* del repositorio de GitHub que implementa diferentes prácticas ágiles.

Este apartado del Anexo describirá, además, la viabilidad del proyecto, que representa recursos y costes valorados para el mismo, tanto humanos como materiales. Este punto de vista de carácter económico es necesario para conocer los límites del proyecto y saber en qué medida sus objetivos entran dentro de los mismos. Para ello se calculará una aproximación de los fondos que requeriría un trabajador que desarrolle el proyecto, y se valorarán los posibles riesgos surgidos durante el desarrollo y cómo actuar frente a ellos.

Con el enfoque estructurado desarrollado este plan de la documentación, el equipo de desarrollo será capaz de cumplir con los objetivos establecidos del proyecto y entregar un resultado satisfactorio dentro de los límites calculados.

A.2. Planificación temporal

Como se ha mencionado anteriormente, la planificación temporal del proyecto se ha llevado a cabo con el uso de la herramienta de las *issues* y

milestones de GitHub, y la herramienta de **Zube** basando todo el desarrollo en tareas. Cada tarea representa una Issue de GitHub y representan una historia de usuario o pequeña tarea interna. Las tareas están bien estructuradas y clasificadas con etiquetas y *milestones* que indican con qué parte de la estructura del proyecto están relacionadas, y a qué fase del desarrollo pertenecen, respectivamente.

Un *sprint*, por su parte, representa una fase del desarrollo del software. En la planificación de este proyecto la mayoría de los sprints se han calculado con una duración de dos semanas y han abarcado entre 10 y 18 tareas o *issues*.

Además, **Zube** incluye varias herramientas de agilidad que favorecen el seguimiento del desarrollo de la aplicación como los Gráficos *burnup* y *burndown*, que representan la evolución del desarrollo de tareas realizadas en cada *sprint*. Más adelante en esta misma sección se pueden visualizar ejemplos de los gráficos *burndown* resultado de los *sprints*. En la Figura A.11 se muestra un ejemplo del gráfico *burndown* correspondiente al Sprint 3 - Comparación temporal.



Figura A.1: Gráfico Burndown del Sprint 3 - Comparación temporal

A continuación se explicará cómo se ha desarrollado, por medio de *sprints*, la planificación temporal del proyecto, siguiendo sus respectivas tareas y representando el flujo seguido para la creación y puesta en producción de la aplicación, así como la redacción de esta documentación del proyecto.

Cada tarea comprendía una historia de usuario o una tarea interna de pequeño tamaño, cuya completación comprende alrededor de un día de duración. En la Figura A.2 se puede observar un ejemplo de tarea en el tablero Kanban de Zube.

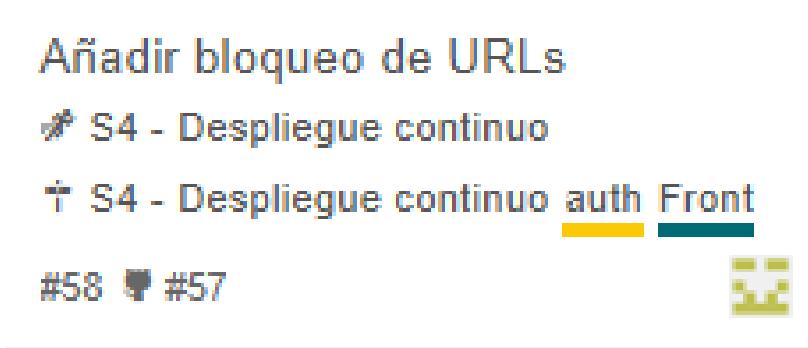


Figura A.2: Ejemplo de una tarea vista en el tablero Kanban de Zube

Los *sprints*, por su parte, han tenido una duración media de quince días, a excepción del primero, pues requirió de varias semanas acordar las bases teóricas y objetivos principales de la aplicación. La Figura A.3 muestra un ejemplo de cómo se visualiza un sprint en Zube.



Figura A.3: Ejemplo de un Sprint en Zube

La gestión de las tareas de cada *sprint* ha sido posible gracias a la herramienta del tablero KanBan, que permite gestionar de forma ágil y visual el proceso de desarrollo en el que se encuentra cada tarea. Una tarea se crea y pasa a estado *ready*, preparada para ser comenzada. Una vez esta se empieza a desarrollar, la tarea pasa al estado *in progress*. Una vez terminado el progreso pasará al estado de *in review*, donde se revisará nuevamente el contenido de la tarea para verificar que sea correcto y esté bien implementado, y, en caso positivo, finalizar en el estado *done*, como se puede ver en la siguiente imagen.

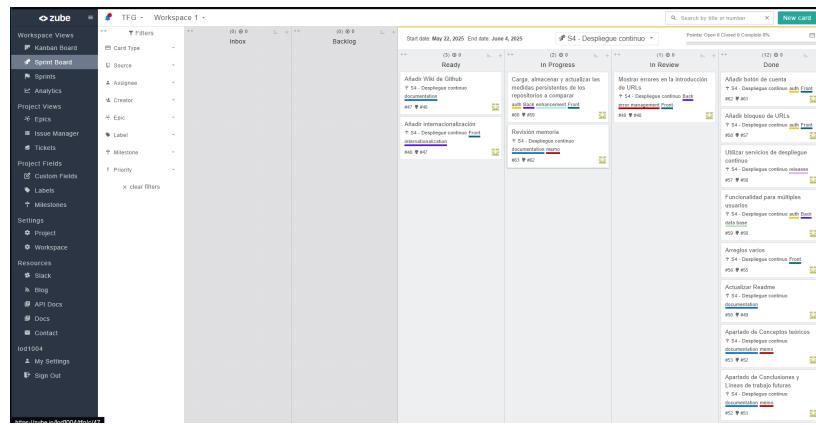


Figura A.4: Ejemplo del estado de un sprint visto en el tablero Kanban

La herramienta de Zube no sólo ha permitido el seguimiento del desarrollo del proyecto en tareas, sino que también ha facilitado la comunicación entre el tutor y el estudiante autor de este TFG, por medio de los comentarios disponibles en cada tarea. Al terminar cada tarea se ha puesto un comentario en la misma indicando los resultados de las actividades asociadas, en ocasiones con imágenes representativas, y el commit que los subió al repositorio del proyecto. Esto ha permitido al tutor exponer su valoración del resultado de la tarea, puntos a mejorar y/o aspectos que necesiten cambios, como se puede ver en la figura A.5

Timeline

Show: comments events

 Carlos López Nozal · May 28
¡Buen diseño gráfico!
Hay un pequeño error tipográfico "Vorsion Control" vs "Version Control". Si quieres tener una versión en inglés de la imagen para internacionalizar la aplicación puedes cambiar solo el título "Agile practices wizard for Github repositories"

 Iod1004 changed the category from In Progress to Done May 28
 Iod1004 closed this Card May 28

 Iod1004 · May 28



 Iod1004 - May 28
Añadida página de inicio con un logo y título al frontEnd que servirá también para iniciar sesión, registrarse, etc.
Commit: c69e3087f19eb1738dcb9cb0597074777d3792c2

 Iod1004 changed the category from Ready to In Progress May 27
 Iod1004 added the label auth May 25
 Iod1004 added this Card to Sprint S4 - Despliegue continuo May 22
 Iod1004 changed the category from Inbox to Ready May 22

Figura A.5: Ejemplo de comentarios en Zube del tutor y alumno en una tarea del proyecto

Organización en *sprints*

Para finalizar esta sección se van a desarrollar individualmente los *sprints* definidos y seguidos durante el desarrollo software. Estos *sprints* han sido pensados y creados durante el propio desarrollo en función del estado en el que se encontraba el proyecto, y los objetivos más adecuados que debían ser implementados en cada momento.

Sprint 0 - Kick-off (1/03/2025 - 03/04/2025)

Este sprint fue el primero y más largo de todos, pues abarcaría no solo el comienzo del desarrollo del proyecto, sino también la planificación del mismo.

Se definió qué aplicación se desarrollaría, sus bases teóricas, el *abstract* y los objetivos principales. Contiene las siguientes 9 tareas:

1. **Definir título y descripción del proyecto:** Consistió en, tras terminar de decidir los objetivos y bases del proyecto, redactar el título y descripción iniciales del mismo.
2. **Introducción del documento:** Redacción inicial del apartado de introducción de la memoria, planteando el contexto y motivación del proyecto.
3. **Revisión del resumen:** Corrección y mejora del resumen del proyecto, afinando su redacción y adecuación a los objetivos propuestos.
4. **Anexos:** Preparación de la estructura base de los anexos que acompañarán a la memoria del proyecto.
5. **Mockups:** Diseño preliminar de la interfaz de usuario para visualizar cómo se estructuraría la aplicación de forma gráfica.
6. **Definir requisitos funcionales:** Identificación de las funciones clave que debía cumplir la aplicación para alcanzar sus objetivos.
7. **Rama de pruebas:** Creación de una rama en el repositorio destinada a pruebas y experimentación durante el desarrollo que se usaría en este sprint.
8. **Estructura del front:** Primer esqueleto del frontend de la aplicación, estableciendo la organización interna de los componentes básicos.
9. **Estructura del back:** Configuración inicial del backend y definición de su arquitectura básica.



Figura A.6: Gráfico Burndown del Sprint 0 - Kick-off

Sprint 1 - Primer prototipo de la aplicación (8/04/2025 - 22/04/2025)

Instaladas las herramientas básicas del proyecto y asentadas las ideas del mismo, comienza el desarrollo con un pequeño prototipo de la aplicación. Se perseguían los siguientes objetivos: conectar el backend con el frontend, definir las reglas que se evaluarán, continuar con la documentación y realizar los primeros análisis de datos.

1. **Conexión Front - Back:** Establecimiento de la comunicación entre el frontend y el backend.
2. **Recogida de Repositorio vía URL:** Implementación de una funcionalidad para introducir un repositorio mediante su URL.
3. **Guardar repositorio en base de datos:** Añadir la funcionalidad de almacenamiento de repositorios para su análisis posterior.
4. **Recoger número de Issues abiertas y cerradas:** Desarrollo de una métrica base para evaluar las *issues* del repositorio.
5. **Mostrar número de Issues:** Visualización en la interfaz del número de *issues*, como parte de las estadísticas presentadas.
6. **Evaluar mensajes de los commits:** Análisis del contenido de los mensajes de los commits para detectar buenas prácticas.

7. **Evaluar Issues en el tiempo:** Cálculo de métricas temporales sobre la evolución de los *issues*.
8. **Evaluar Pull Requests:** Análisis de la actividad de las *pull requests* como indicador de colaboración.
9. **Evaluar Releases:** Inclusión de información sobre las versiones liberadas del proyecto.
10. **Evaluar Acciones:** Estudio de las GitHub Actions empleadas como parte del flujo de trabajo.
11. **Definir reglas:** Redacción y codificación de las reglas de evaluación de buenas prácticas basadas en metodología ágil.

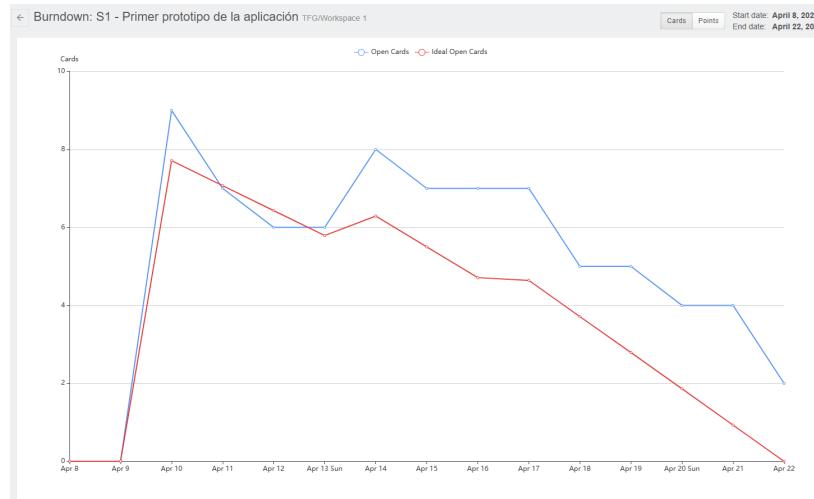


Figura A.7: Gráfico Burndown del Sprint 1 - Primer prototipo de la aplicación

Sprint 2 - Comparación temporal (24/04/2025 - 08/05/2025)

La aplicación ya obtiene estadísticas básicas. En este sprint se centró en implementar una comparación temporal entre repositorios, añadir control de errores y documentar la evolución temporal de los proyectos.

1. **Crear funciones de las reglas:** Implementación concreta de las funciones para evaluar las reglas definidas.
2. **Mostrar cumplimiento de reglas:** Inclusión en el frontend de indicadores que muestran si se cumplen las reglas.

3. **Extraer intervalos de tiempo del repositorio:** Implementar la funcionalidad de los intervalos de tiempo para comparar la actividad del repositorio.
4. **Selección de intervalos de tiempo:** Interfaz para que el usuario pueda seleccionar intervalos de tiempo a usar en el análisis.
5. **Añadir estadísticas temporales:** Implementación de métricas que reflejen la evolución cada cierto número de días.
6. **Añadir estadísticas restantes:** Inclusión de estadísticas que aún no habían sido implementadas.
7. **Estadísticas de participantes:** Incorporación de datos sobre los miembros del repositorio.
8. **Ordenar Base de datos:** Refactorización y optimización de la estructura de almacenamiento de repositorios y medidas de calidad de proceso.
9. **Añadir control de errores:** Añadir gestión de errores para situaciones como problemas de red o repositorios no válidos.
10. **Documentar sobre la evolución de proyectos en el tiempo:** Redacción teórica en la memoria sobre las fases de un proyecto ágil.

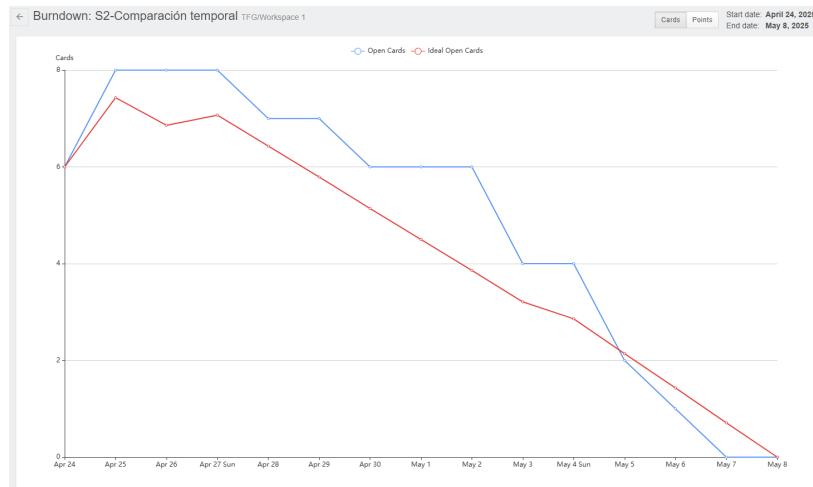


Figura A.8: Gráfico Burndown del Sprint 2 - Comparación temporal

Sprint 3 - Diseño de la estética de la interfaz de usuario (08/05/2025 - 21/05/2025)

Con una versión funcional disponible, se procedió a mejorar la experiencia de usuario y seguir avanzando en la documentación del proyecto. El principal objetivo era maquetar la interfaz de usuario para que esta se pareciera al *mockup* diseñado en el Sprint 0 - KickOff.

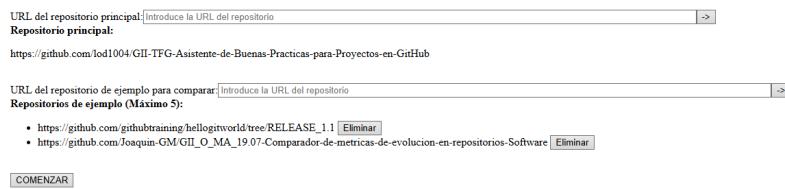


Figura A.9: Imagen de la primera versión de la interfaz de la aplicación



Figura A.10: Imagen de la interfaz tras la maquetación del Sprint 3 - Diseño de la estética de la interfaz de usuario

Como se puede apreciar en las figuras A.9 y A.10, este Sprint supuso un cambio significado para el diseño de la interfaz de usuario, y el *look and feel* de la aplicación.

1. **Maquetación del FrontEnd:** Rediseño visual del frontend para mejorar la experiencia y estética.
2. **Nuevas estadísticas y revisión:** Mejora de algunas métricas, añadiendo de algunas nuevas y validación de los resultados obtenidos.

3. **Añadir Spinner:** Mejora visual mediante un componente que indica la carga de información.
4. **Implementar logs:** Sistema de registro de eventos para facilitar el mantenimiento.
5. **Información sobre resultados de las reglas:** Mostrar explicaciones detalladas de cada regla y su resultado.
6. **Apartado de "Trabajos Relacionados":** Redacción del apartado que contextualiza el trabajo con investigaciones previas.
7. **Apartado de ".^spectos relevantes del desarrollo":** Inclusión en la memoria de las decisiones clave durante el desarrollo.
8. **Apartado de ".ºbjetivos del proyecto":** Definición clara de los objetivos concretos que se querían alcanzar.
9. **Apartado de "Técnicas y herramientas":** Descripción de las herramientas usadas y su justificación.
10. **Primera Release del proyecto:** Publicación de la primera versión oficial v0.1.0 de la aplicación.



Figura A.11: Gráfico Burndown del Sprint 3 - Diseño de la estética de la interfaz de usuario

Sprint 4 - Despliegue continuo (22/05/2025 - 09/06/2025)

En este último sprint se trabajó en el despliegue de la aplicación, correcciones finales, soporte para múltiples usuarios y finalización de la memoria.

1. **Utilizar servicios de despliegue continuo:** Implementación de un flujo de despliegue automático tras cada actualización.
2. **Mostrar errores en la introducción de URLs:** Validación del input del usuario para evitar URLs no válidas.
3. **Añadir internacionalización:** Soporte para diferentes idiomas en la interfaz de usuario.
4. **Añadir página de inicio:** Creación de una landing page para la aplicación que serviría como formulario de login.
5. **Cambiar título y nomenclaturas:** Revisión de nombres de variables y componentes para mayor claridad.
6. **Actualizar Readme:** Actualización de la documentación principal del repositorio con instrucciones claras.
7. **Añadir Wiki de Github:** Creación de una wiki complementaria al README para documentar el uso.
8. **Añadir usuarios y login:** Implementación básica de autenticación de usuarios.
9. **Funcionalidad para múltiples usuarios:** Adaptación del backend para permitir análisis separados por usuario.
10. **Añadir botón de cuenta:** Crear un botón de apertura del menú de usuario.
11. **Revisión memoria:** Revisión general del contenido escrito de la memoria.
12. **PDF completo de la memoria:** Generación del documento final completo en formato PDF.
13. **Apartado de Conclusiones y Líneas de trabajo futuras:** Reflexión final sobre el proyecto y posibles continuaciones en la memoria.
14. **Apartado A de los anexos:** Redacción del apartado A de los anexos.

15. **Revisión del apartado B de los anexos:** Revisión del apartado B de los anexos.
16. **Apartado C de los anexos:** Redacción del apartado C de los anexos.
17. **Apartado D de los anexos:** Redacción del apartado D de los anexos.
18. **Apartado E de los anexos:** Redacción del apartado E de los anexos.
19. **Apartado F de los anexos:** Redacción del apartado F de los anexos.
20. **Anexos (finales):** Inclusión de todos los anexos restantes con los elementos complementarios.
21. **Test eficiencia v.0.2.0:** Evaluación de la eficiencia del sistema tras la última versión.
22. **Arreglos varios:** Correcciones menores y ajustes visuales o funcionales.

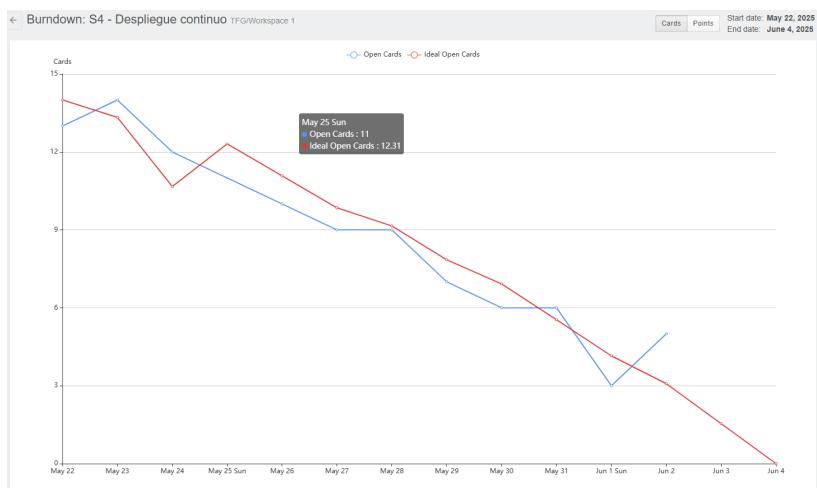


Figura A.12: Gráfico Burndown del Sprint 4 - Despliegue continuo

A.3. Estudio de viabilidad

Este apartado recoge un análisis detallado sobre la viabilidad del proyecto desde una doble perspectiva: económica y legal. Se examinan los recursos necesarios para el desarrollo de la aplicación web **Asistente de prácticas ágiles para repositorios en GitHub** en equipos de desarrollo que

utilizan GitHub, así como los requisitos y restricciones legales que podrían condicionar su puesta en marcha. Dado su enfoque académico, automatizado y extensible, el proyecto se plantea como una herramienta útil tanto para entornos académicos como profesionales, y su desarrollo es asumido por un único responsable técnico.

Viabilidad económica

Costes de personal

El desarrollo ha sido asumido por una sola persona con perfil de estudiante de ingeniería informática. De acuerdo con el XVIII Convenio Colectivo Estatal de Empresas de Consultoría y Tecnologías de la Información [?], se estima un coste bruto mensual en torno a 2.000 €. Para el desarrollo completo del proyecto se contemplan hasta 4 meses de trabajo.

Concepto	Coste
Salario mensual	1.500€
Seguridad Social y retenciones	500€
Total 4 meses	8.000€

Tabla A.1: Costes de *personal*

Coste de hardware

Se presupone la utilización de un equipo de sobremesa ya disponible con prestaciones suficientes para ejecutar herramientas de desarrollo, e instalar aplicaciones y herramientas de programación y puesta en producción, así como entornos de inferencia local. A efectos contables, se considera una amortización del equipo a 3 años.

Concepto	Coste	Coste amortizado
Ordenador de desarrollo	500€	113€
Total	500€	113€

Tabla A.2: Costes de *hardware*

Coste de conectividad

El uso continuo de internet es fundamental, tanto para descargar dependencias como para utilizar APIs de inferencia o conectarse a servicios de terceros como GitHub[3], Vercel[8] o Render[6]. Se estima un coste medio mensual de 35€.

- **Total en conectividad (4 meses):** 140€

Coste de software

Todo el software utilizado en el desarrollo es de código abierto o con licencias gratuitas para su uso individual: Visual Studio Code, GitHub, Vercel (usando entornos públicos gratuitos) y Render. El sistema operativo empleado es Windows 10 Home, de un coste de alrededor de 10 euros y ampliamente compatible con las herramientas mencionadas.

Coste estimado: 10€

Auditoría y supervisión académica

Se reserva un coste temporal asociado al seguimiento del proyecto por parte del tutor, que incluye la corrección del código, revisión de entregables y reuniones de orientación técnica.

Concepto	Coste
Mano de obra	8.000€
Hardware amortizado	113€
Conectividad	140€
Software Windows 10	10€
Revisión académica	750€
Total	9.306€

Tabla A.3: Costes totales del asistente inteligente para prácticas ágiles

Sostenibilidad económica del proyecto

Una vez completado el desarrollo inicial, el coste de mantenimiento es bajo, especialmente si se utilizan modelos locales. La monetización del asistente podría basarse en:

- **Licencia freemium:** acceso básico gratuito y funciones avanzadas bajo suscripción.
- **Integración en GitHub Marketplace:** cobro por instalación en organizaciones.
- **Modelo educativo:** cesión gratuita a universidades con apoyo institucional.

Viabilidad legal

Licencias del software utilizado

Se ha garantizado el uso exclusivo de herramientas de código abierto o con licencias permisivas. Esto incluye tanto las librerías públicas usadas en el frontend y el backend como herramientas de despliegue continuo como Vercel o Render.

- Librerías de Angular y Node.js.
- Integraciones continuas con GitHub, Vercel y Render.

Términos de uso de las APIs de terceros

Al utilizar servicios de inferencia por API (En este caso GitHub API) deben respetarse las políticas de uso responsable, límites de almacenamiento de datos y uso no comercial sin licencia explícita.

Requisitos:

- Consentimiento del usuario en el tratamiento de repositorios.
- Prohibición del reentrenamiento sobre inputs sin autorización.

Protección de datos

En caso de desplegar el asistente en organizaciones reales, será necesario cumplir con la normativa de protección de datos como el RGPD:

- Anonimización de inputs del usuario.
- Consentimiento explícito para recoger métricas.
- Política de privacidad visible.

Restricciones de publicación

Si el asistente se publica en plataformas como GitHub Marketplace o se distribuye como SaaS, deberá incluir:

- Condiciones de uso claras.
- Política de privacidad.
- Declaración de uso de APIs externas.

Checklist legal

1. Verificación de licencias de todas las librerías.
2. Inclusión de política de privacidad si se recoge algún dato.
3. Revisión de los términos de cada API externa usada.
4. Consentimiento de usuario informado en contextos reales.
5. Uso responsable de la inferencia (sin outputs automatizados sin revisión humana en contextos críticos).

Cumpliendo estas condiciones, la legalidad de la aplicación está asegurada y no representa impedimentos para su desarrollo ni distribución futura.

Apéndice B

Especificación de Requisitos

B.1. Introducción

Este apéndice recoge los requisitos funcionales de la aplicación propuesta en este proyecto para evaluar la adopción de prácticas ágiles en repositorios de GitHub utilizados por estudiantes para realizar diversos trabajos como, por ejemplo, Trabajos de Fin de Grado (TFG). Se incluyen los casos de uso posibles que podrá tener un usuario de la aplicación desarrollada, junto a tablas y diagramas que ayuden a comprender los mismos. Los casos de uso comprenderán cierto número de requisitos funcionales y no funcionales, listados y explicados más adelante en este apartado para especificar claramente todas las formas que tiene el usuario de interactuar con el software propuesto. Además se incluirá el listado de objetivos generales del proyecto para ayudar a contextualizar tanto los casos de uso como los requisitos funcionales.

B.2. Objetivos generales

La visión general de este proyecto abarca los objetivos descritos a continuación:

- **Análisis de métricas de calidad de proceso de software de los repositorios:** La aplicación web analiza detalladamente todas las métricas de calidad de proceso y características que pueden resultar útiles de un repositorio durante el proceso de desarrollo de software.

- **Ajuste temporal del análisis:** Consiste en la posibilidad de ajustar distintos parámetros temporales del análisis para elegir en qué etapas del desarrollo se analizarán los repositorios.
- **Comparación de las métricas de calidad de proceso obtenidas de los repositorios de referencia:** La aplicación es capaz de, dadas todas las medidas de calidad de proceso de los repositorios que el usuario ha elegido como referencia, compararlas para ofrecer al usuario una visualización clara de los diferentes valores de estas medidas en los distintos repositorios.
- **Prácticas ágiles y medidas de repositorios:** La aplicación utiliza las definiciones de prácticas ágiles recogidas en [1] para presentar los resultados al usuario tras analizar el repositorio.
- **Asistencia a estudiantes:** Se pueden utilizar otros TFGs públicos realizados en la Universidad de Burgos como casos de estudio, introduciéndolos en la aplicación y que estos sirvan como referencia para los estudiantes a la hora de realizar sus propios TFG.

B.3. Catálogo de requisitos

Requisitos funcionales

A continuación se puede observar un listado de todos los requisitos funcionales considerados para la aplicación realizada en este proyecto.

- **RF-01** – Creación una cuenta para usar la aplicación.
- **RF-02** – Entrada al perfil creado iniciando sesión.
- **RF-03** – Salida del perfil creado cerrando sesión.
- **RF-04** – Cambio de la contraseña del perfil de la cuenta creada.
- **RF-05** – Cambio del idioma de la interfaz de usuario de la aplicación.
- **RF-06** – Introducción y validación de URLs de repositorios.
- **RF-07** – Elección del número de días para realizar el cálculo de las medidas de calidad temporales.
- **RF-08** – Elección del ámbito temporal de extracción de medidas de calidad de proceso de los repositorios.

- **RF-09** – Elección del tipo de intervalos de tiempo a usar para extraer las medidas de calidad de proceso de los repositorios introducidos.
- **RF-10** – Ajuste de los intervalos de tiempo específicos para elegir de qué momento en el tiempo de vida de los repositorios se extraerán las medidas de calidad de proceso.
- **RF-11** – Carga de grupos de repositorios usados como fuente de referencia en la aplicación con anterioridad.
- **RF-12** – Borrado de grupos de repositorios usados como fuente de referencia en la aplicación con anterioridad.
- **RF-13** – Visualización de la evaluación de la aplicación de prácticas ágiles por parte del repositorio a evaluar.
- **RF-14** – Visualización de la comparación de los valores numéricos de las medidas de calidad de proceso entre el repositorio a analizar y los repositorios de referencia.

Requisitos no funcionales

Continuando los requisitos del proyecto, más adelante se encuentra un listado de todos los requisitos no funcionales considerados para la aplicación realizada en este proyecto.

- **RNF-01** Modularidad
- **RNF-02** Usabilidad
- **RNF-03** Reusabilidad
- **RNF-04** Separación entre frontend y backend
- **RNF-05** Validaciones informativas
- **RNF-06** Navegación intuitiva

B.4. Especificación de requisitos

Requisitos funcionales

A continuación se describen de forma detallada los requisitos funcionales del sistema.

- *RF-01 – Creación de una cuenta de usuario El sistema debe permitir a los usuarios crear una cuenta personal mediante un formulario de registro. Esta funcionalidad es necesaria para garantizar la personalización de la experiencia, el almacenamiento de configuraciones y el acceso a funcionalidades avanzadas, como la carga de repositorios de referencia o la consulta del historial de análisis realizados; además de garantizar seguridad y control de acceso a la aplicación.
- *RF-02 – Inicio de sesión Los usuarios registrados deben poder iniciar sesión en el sistema para acceder a sus perfiles y funcionalidades asociadas mediante un nombre de usuario único y una contraseña que se almacenará encriptada para garantizar la seguridad del almacenaje de perfiles de usuario en la base de datos.
- *RF-03 – Cierre de sesión Los usuarios pueden cerrar sesión de forma segura al finalizar su uso, tanto para finalizar de forma segura su actividad o cambiar de cuenta por diferentes motivos como el uso de varias cuentas personalizadas con diferentes ajustes y/o grupos de repositorios almacenados.
- *RF-04 – Cambio de contraseña El sistema debe permitir a los usuarios modificar la contraseña de su cuenta en cualquier momento, con el fin de mantener la seguridad y el control sobre su acceso. El proceso debe incluir medidas de validación para evitar contraseñas débiles o inseguras.
- *RF-05 – Cambio de idioma de la interfaz La interfaz de la aplicación debe ser multilingüe, permitiendo a los usuarios seleccionar el idioma que prefieran entre las opciones disponibles. Esto contribuye a la internacionalización del software, mejora la accesibilidad y facilita su uso en contextos internacionales o multilingües.
- *RF-06 – Introducción y validación de URLs de repositorios El sistema debe permitir a los usuarios introducir la URL de uno o varios repositorios públicos de GitHub. La aplicación validará que las URLs introducidas sean correctas, que los repositorios estén disponibles y que se pueda acceder a ellos para su análisis para informar al usuario de que está introduciendo repositorios válidos y de manera correcta.
- *RF-07 – Selección del número de días para métricas temporales El sistema debe permitir al usuario definir un número de días a considerar como ventana temporal para el cálculo de medidas de calidad de proceso relacionadas con medias. Esta opción permite realizar análisis

adaptados a periodos de mayor o menor longitud de actividad dentro del proyecto.

- *RF-08 – Selección del ámbito temporal del análisis Los usuarios podrán definir si los repositorios se analizarán al completo, lo cual abarcaría todo su tiempo de vida desde el primer *commit* hasta el último, o por intervalos de tiempo ajustables.
- *RF-09 – Selección del tipo de intervalos de tiempo El sistema debe ofrecer la posibilidad de seleccionar el tipo de intervalo temporal para el análisis, ya sean intervalos relativos, que abarquen cuartos de la vida del repositorio, o intervalos absolutos que equivalgan a meses de vida. Esta opción facilita la visualización de tendencias y eficiencia de trabajo en distintas fases del desarrollo de software.
- *RF-10 – Ajuste manual de intervalos de tiempo Además de seleccionar el tipo de intervalo, el usuario podrá definir manualmente los intervalos exactos de tiempo que desea analizar (Por ejemplo el primer y segundo cuarto, lo cual es equivalente a la primera mitad del repositorio para los intervalos relativos, o del tercer al sexto mes de vida del repositorio para los intervalos absolutos). Esta funcionalidad avanzada permite focalizar el análisis en momentos clave del desarrollo del proyecto.
- *RF-11 – Carga de grupos de repositorios de referencia La aplicación debe permitir al usuario cargar grupos de repositorios previamente definidos como referencia. Esto permite al usuario repetir comparaciones o realizar comparaciones muy similares evitando el tiempo de extracción de medidas de calidad de proceso.
- *RF-12 – Eliminación de grupos de referencia cargados* El usuario también debe tener la posibilidad de eliminar grupos de repositorios de referencia previamente cargados, con el fin de mantener organizada su área de trabajo y adaptar el análisis a nuevos contextos o criterios de comparación.
- *RF-13 – Visualización de la evaluación de prácticas ágiles Una vez realizado el análisis, el sistema debe mostrar una evaluación del repositorio analizado en función de su cumplimiento de buenas prácticas ágiles. Esta evaluación debe representarse de forma clara y comprensible en forma de reglas que indiquen visualmente en qué aspectos se usan de forma sólida las metodologías ágiles y en qué aspectos hay opción de mejora.

- *RF-14 – Comparación con repositorios de referencia La aplicación debe permitir comparar los valores de todas las métricas extraídas del repositorio analizado con los correspondientes valores de los repositorios de referencia seleccionados. Esta comparación se visualizará mediante listas que ayuden a respaldar la evaluación del uso de prácticas ágiles en los repositorios.

Requisitos no funcionales

A continuación se describen de forma detallada los requisitos no funcionales del sistema.

- *RNF-01 – Modularidad La aplicación debe estar diseñada utilizando una arquitectura modular basada en componentes reutilizables. Debe ser compatible con patrones de diseño modernos y fácil de escalar o modificar por partes sin afectar al conjunto total del software. Esta estructura permite que futuras modificaciones o ampliaciones (como la adición de nuevas métricas o funcionalidades) se integren de forma controlada y sencilla.
- *RNF-02 – Usabilidad La interfaz debe ser intuitiva, clara y fácil de usar. Los formularios deben presentar instrucciones comprensibles, validaciones dinámicas y retroalimentación inmediata ante errores o acciones del usuario. Se prioriza un diseño simple y directo que permita a usuarios no técnicos comprender la aplicación sin curva de aprendizaje.
- *RNF-03 – Reusabilidad El código debe seguir principios como DRY (Don't Repeat Yourself) y SOLID, en particular el principio Open/Closed, que facilita la extensión de funcionalidades sin modificar el núcleo existente. La organización del código promueve la reutilización de servicios, componentes y lógica, favoreciendo la eficiencia en el desarrollo y la reducción de errores.
- *RNF-04 –Separación entre frontend y backend El sistema debe mantener una separación estricta entre las capas de presentación (frontend) y lógica de negocio (backend), utilizando servicios HTTP para el intercambio de datos de forma asíncrona. Esto mejora la escalabilidad y permite gestionar respuestas y errores de forma controlada desde la interfaz.

- *RNF-05 – Validaciones informativas Los formularios deben incorporar validación reactiva utilizando sistemas de formularios y alertas, proporcionando retroalimentación inmediata al usuario en caso de errores o datos incompletos. Esto reduce la frustración del usuario y mantiene informado al mismo, y además mejora la precisión y seguridad de los datos introducidos.

- *RNF-06 – Navegación intuitiva La aplicación debe organizar sus funcionalidades de forma jerárquica, utilizando tabs, rutas y botones de navegación claros que guíen al usuario durante el análisis. Los componentes de navegación deben ser reutilizables y consistentes en toda la aplicación para facilitar el acceso rápido a las diferentes secciones.

Casos de uso

En esta sección se presentan los diferentes casos de uso que encapsulan los requisitos previamente definidos, con el fin de cumplir los objetivos funcionales y no funcionales establecidos para la aplicación. Cada uno de estos casos de uso describe una interacción específica entre los actores del sistema (en el caso de esta aplicación: el usuario, el backend y el frontend) y sus funcionalidades principales, proporcionando así una visión clara de cómo se espera que los usuarios interactúen con la aplicación. Esto permite no solo una mejor comprensión de las funcionalidades esenciales del sistema, sino también una base sólida para futuras etapas de diseño e implementación. La Figura B.1 ilustra de manera general todos los casos de uso identificados, organizando visualmente las relaciones entre los actores principales y las funcionalidades del sistema, lo cual facilita su análisis y validación.

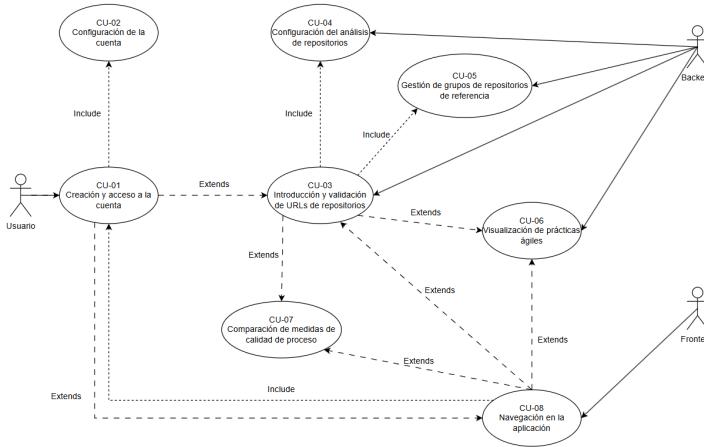


Figura B.1: Diagrama de casos de uso general

CU-1	Creación y acceso a la cuenta
Versión	1.0
Autor	Lucas Olmedo Diéz
Requisitos asociados	RF-01, RF-02
Descripción	Permite al usuario crear una cuenta y acceder a la aplicación mediante inicio de sesión.
Precondición	El usuario debe introducir un usuario único y una contraseña válida
Acciones	<ol style="list-style-type: none"> 1. El usuario accede a la página de registro o inicio de sesión. 2. El usuario introduce los datos necesarios (usuario y contraseña). 3. El sistema verifica la validez de los datos. 4. El sistema registra al usuario o permite el acceso según corresponda.
Postcondición	El usuario ha creado una cuenta o ha accedido a su perfil.
Excepciones	Datos incorrectos o cuenta inexistente/ya registrada.
Importancia	Alta

Tabla B.1: CU-1 Creación y acceso a la cuenta.

CU-2	Configuración de la cuenta
Versión	1.0
Autor	Lucas Olmedo Diéz
Requisitos asociados	RF-02, RF-03
Descripción	Permite al usuario cambiar la contraseña y modificar el idioma de la aplicación.
Precondición	El usuario debe haber iniciado sesión.
Acciones	<ol style="list-style-type: none">1. El usuario accede a la sección de configuración.2. Selecciona cambiar contraseña o idioma.3. El sistema valida y aplica los cambios.
Postcondición	Se actualizan las preferencias del usuario.
Excepciones	
Importancia	Media

Tabla B.2: CU-2 Configuración de la cuenta.

CU-3	Introducción y validación de URLs de repositorios
Versión	1.0
Autor	Lucas Olmedo Diéz
Requisitos asociados	RF-04
Descripción	Permite al usuario introducir URLs de repositorios GitHub y valida su formato y accesibilidad.
Precondición	El usuario ha iniciado sesión.
Acciones	<ol style="list-style-type: none">1. El usuario introduce una o varias URLs.2. El sistema valida la estructura y el acceso.3. Se informa al usuario del resultado.
Postcondición	Se aceptan o rechazan las URLs introducidas.
Excepciones	URL malformada o repositorio no accesible.
Importancia	Alta

Tabla B.3: CU-3 Introducción y validación de URLs de repositorios.

CU-4	Configuración del análisis de repositorios
Versión	1.0
Autor	Lucas Olmedo Díez
Requisitos asociados	RF-05, RF-06, RF-07, RF-08
Descripción	Permite al usuario configurar cómo se analizarán los datos del repositorio: intervalos absolutos o relativos, número de días para calcular las métricas temporales, períodos de los intervalos.
Precondición	El usuario debe haber elegido la opción de ".Analizar por intervalos de tiempo".
Acciones	<ol style="list-style-type: none"> 1. El usuario selecciona si desea usar fechas absolutas o relativas. 2. Ajusta manualmente los intervalos de fechas específicos. 3. Elige el tipo de intervalos para segmentar la información. 4. (Opcional) Se cambia el número de días usado para obtener métricas de calidad temporales.
Postcondición	El sistema ha almacenado la configuración de análisis para ser utilizada en el cálculo de métricas.
Excepciones	Intervalos inválidos, número de días no válido, configuración incompleta.
Importancia	Alta

Tabla B.4: CU-4 Configurar análisis del repositorio

CU-5	Gestión de grupos de repositorios de referencia
Versión	1.0
Autor	Lucas Olmedo Díez
Requisitos asociados	RF-09, RF-10
Descripción	Permite al usuario cargar o borrar grupos de repositorios previamente guardados para usarlos como referencia comparativa en los análisis.
Precondición	El usuario debe haber hecho algún análisis previo con el que se guardaron repositorios de referencia.
Acciones	<ol style="list-style-type: none"> 1. El usuario accede a la sección de gestión de repositorios guardados. 2. Selecciona un grupo de repositorios guardado previamente y lo carga. 3. (Opcional) Elimina uno o más grupos guardados si lo desea.
Postcondición	Los grupos seleccionados han sido cargados o eliminados exitosamente.
Excepciones	No hay grupos guardados disponibles o error en la carga/borrado.
Importancia	Media

Tabla B.5: CU-5 Gestionar grupos de repositorios de comparación

CU-6	Visualización de prácticas ágiles
Versión	1.0
Autor	Lucas Olmedo Díez
Requisitos asociados	RF-11
Descripción	Permite al usuario visualizar el grado de adopción de buenas prácticas ágiles evaluadas automáticamente por el sistema a partir de los datos del repositorio.
Precondición	Se han introducido URLs válidas y parámetros del análisis correctos.
Acciones	<ol style="list-style-type: none"> 1. El sistema muestra los resultados del análisis de buenas prácticas en formato gráfico y textual. 2. El usuario puede consultar qué reglas se cumplen, cuáles no, y por qué.
Postcondición	El usuario ha accedido a la información evaluada y entendible de las prácticas ágiles del repositorio.
Excepciones	No se han podido calcular algunas reglas por errores en algún repositorio.
Importancia	Alta

Tabla B.6: CU-6 Visualizar evaluación de buenas prácticas ágiles

CU-7	Comparación de medidas de calidad de proceso
Versión	1.0
Autor	Lucas Olmedo Díez
Requisitos asociados	RF-12
Descripción	Permite comparar visualmente las métricas del repositorio analizado con las de los repositorios de referencia cargados.
Precondición	El repositorio analizado y al menos un grupo de referencia, junto a sus medidas de calidad deben estar cargados.
Acciones	<ol style="list-style-type: none"> 1. El usuario accede a la sección de comparación. 2. El sistema genera y muestra listas comparativas de las métricas seleccionadas. 3. El usuario interpreta la comparación en función de los valores visualizados.
Postcondición	El usuario obtiene una visión comparativa clara del estado de su proyecto.
Excepciones	Error al cargar los datos de referencia o métricas incomparables.
Importancia	Alta

Tabla B.7: CU-7 Comparar con repositorios de referencia

CU-8	Navegar por la aplicación
Versión	1.0
Autor	Lucas Olmedo Díez
Requisitos asociados	RF-11, RF-03
Descripción	Permite al usuario desplazarse cómodamente por las diferentes secciones de la aplicación (login, análisis, comparación, configuración, etc.), garantizando una experiencia de usuario fluida.
Precondición	El usuario debe haber iniciado sesión. La aplicación debe estar desplegada correctamente.
Acciones	<ol style="list-style-type: none"> 1. El usuario interactúa con los tabs y botones de navegación disponibles. 2. El sistema muestra el contenido correspondiente sin errores.
Postcondición	El usuario ha accedido correctamente a las diferentes funcionalidades sin fricciones.
Excepciones	Error de carga, o comportamiento inesperado.
Importancia	Alta

Tabla B.8: CU-8 Navegar por la aplicación

Apéndice C

Especificación de diseño

C.1. Introducción

En este apartado del anexo se van a documentar todos los aspectos relacionados al diseño del proyecto, y como consecuencia, de la aplicación desarrollada en el mismo. Se describirán los componentes y entidades que forman el proyecto, y cómo estos se relacionan entre sí, con el objetivo de detallar cómo logran implementar los objetivos del proyecto ya mencionados.

La manera principal de explicar los componentes, datos y entidades del diseño del proyecto será la de varios diagramas que ayuden a comprender el funcionamiento y relaciones que existen entre los mismos, incluyendo sus atributos y características.

C.2. Diseño de datos

En esta sección se describirá el diseño de los datos del sistema. Para ello se detallará una serie de diagramas entidad-relación que detallarán cómo se relacionan estos entre sí, mostrando los atributos más influyentes de cada uno.

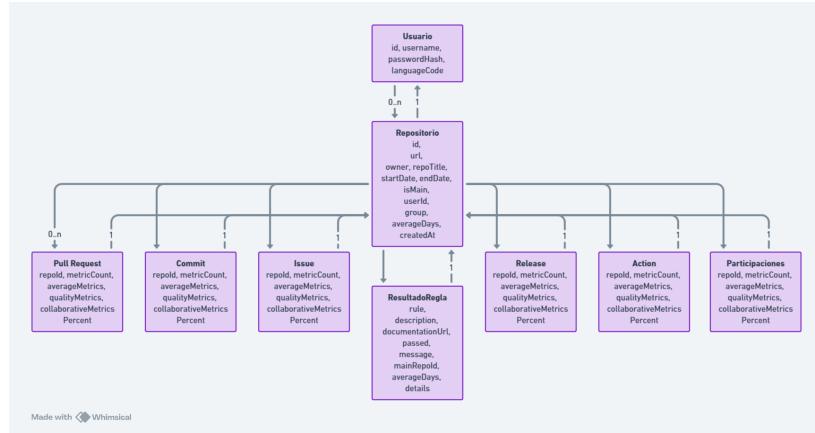


Figura C.1: Diagrama Entidad - Relación de la aplicación del proyecto

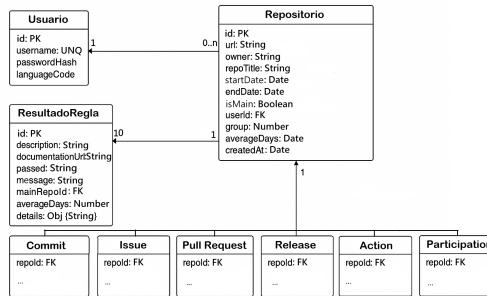


Figura C.2: Diagrama Entidad - Relación UML de la aplicación del proyecto

Entidades del Sistema

- **Usuario**: Representa a los usuarios de la aplicación.
 - **id**: Identificador único del usuario.
 - **username**: Nombre de usuario, único en el sistema.
 - **passwordHash**: Contraseña encriptada.
 - **languageCode**: Código del idioma que utiliza el usuario.

- **Repository:** Contiene los datos de un repositorio analizado.
 - **id:** Identificador único del repositorio analizado.
 - **url:** Dirección URL del repositorio.
 - **owner:** Nombre del propietario del repositorio.
 - **repoTitle:** Título o nombre del repositorio.
 - **startDate:** Fecha de inicio del análisis.
 - **endDate:** Fecha de finalización del análisis.
 - **isMain:** Indica si es el repositorio principal o de referencia.
 - **userId:** Identificador del usuario propietario del repositorio.
 - **group:** Número de grupo comparativo de referencia (si aplica).
 - **averageDays:** Número de días promedio utilizado en métricas de media.
 - **createdAt:** Fecha de creación del análisis.
- **ResultadoRegla:** Almacena el resultado del análisis de buenas prácticas.
 - **rule:** Nombre de la regla evaluada.
 - **description:** Descripción de la regla.
 - **documentationUrl:** Enlace a la documentación de la regla.
 - **passed:** Indica si la regla fue superada.
 - **message:** Comentario asociado al resultado.
 - **mainRepoId:** Repositorio principal evaluado.
 - **averageDays:** Días promedio utilizados en la evaluación.
 - **details:** Medidas que evalúa la regla y su evaluación
- **Métricas (*Commit, Issue, Pull Request, Release, Action, Participaciones*):** Cada una de estas entidades representa métricas agregadas de su tipo específico, siempre vinculadas a un único repositorio mediante la clave **repoId**.
 - **repoId:** Repositorio al que pertenece la métrica.
 - **metricCount:** Total numérico de la métrica (Por ejemplo *commitCount*).
 - **averageMetrics:** Promedio de aparición o cierre de la métrica (Por ejemplo *averageClosedIssues*).

- **qualityMetrics**: Atributos que reflejan medidas de calidad de proceso que dependen del tipo de métrica (Por ejemplo el porcentaje de issues con imágenes, o el número de ficheros workflow ejecutados con éxito).
- **collaborativeMetricsPercent**: Porcentaje de instancias de la métrica realizadas de forma colaborativa.

Relaciones entre Entidades

- **Usuario - Repositorio**:
 - Un **Usuario** posee **0..n** instancias de **Repositorio**.
 - Un **Repositorio** pertenece a exactamente un **Usuario**.
- **Repositorio - ResultadoRegla**:
 - Un **Repositorio (principal)** genera **10** resultados de **ResultadoRegla** (Uno por cada una de las 10 reglas definidas).
 - Un **ResultadoRegla** se asocia a exactamente un **Repositorio (principal)**.
- **Repositorio - Métrica**:
 - Un **Repositorio** tiene exactamente una instancia de métricas para cada uno de los siguientes: **Commit, Issue, Pull Request, Release, Action, Participaciones**.
 - Cada una de estas métricas pertenece a un solo **Repositorio**.

C.3. Diseño arquitectónico

El diseño de la arquitectura de la aplicación se divide en 2 partes fundamentales: un frontend de Angular dividido en componentes y servicios, y un backend de Node js que usa controladores para ejecutar funciones y modelos para la base de datos. La separación del frontEnd y el backend garantiza la separación de responsabilidades, lo que se ve reforzado por los componentes del frontend y los controladores del backend, que separan aún más las responsabilidades internamente. Los componentes y controladores siguen el principio *Open-Closed* de programación, permitiendo añadir nuevas funcionalidades sin necesidad de alterar el resto del código (Por ejemplo añadir nuevas pantallas o reglas).

Esta arquitectura y sus divisiones en directorios se muestran de una forma más detallada y visual en la siguiente figura:

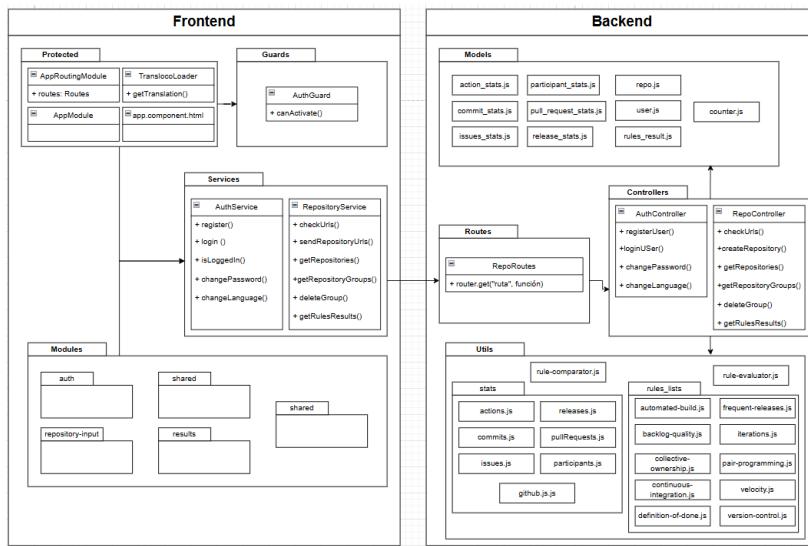


Figura C.3: Diagrama del diseño arquitectónico

A continuación se detallará en mayor medida las funcionalidades de cada parte tanto del backend como el frontend

■ Frontend:

- **Guards:** Incluye el fichero `auth.guard.guard.ts`, que se encarga de reforzar la seguridad de acceso a la aplicación bloqueando accesos a las distintas URLs sin haber iniciado sesión
- **Modules:** Contiene casi toda la lógica del frontend, incluyendo todos los componentes `.ts`, `.html` y `.css` de cada parte de la página web, encargándose estos de definir la lógica, estructura de la página y estilos de maquetación de la misma, respectivamente. Se puede apreciar una mejor representación de su estructura en la figura siguiente. Se compone de las siguientes carpetas:
 - **auth:** Da forma a la pantalla de inicio y usa sus componentes hijos para cambiar entre formulario de registro, inicio de sesión o cambio de contraseña.
 - **repository-input:** Se encarga de la pantalla donde el usuario introduce los repositorios y configura el análisis.

- **results:** Muestra la pantalla de los resultados de las prácticas ágiles.
- **statistics:** Muestra la pantalla de comparación de métricas de proceso, que se compone de 2 listas comparativas entre repositorios.
- **shared:** Contiene componentes que usan varias pantallas, como lo son el *header* y el *footer*.
- **Protected:** Incluye los módulos y componentes más importantes del código, como las definiciones de las rutas, las librerías y componentes a usar, el fichero html principal y el cargador de la librería de traducciones "Transloco".
- **Services:** Incluye los ficheros auth.service.ts y repository.service.ts, que utilizarán peticiones HttpClient para llamar a las funciones correspondientes de los controladores del backend y utilizar sus funciones.

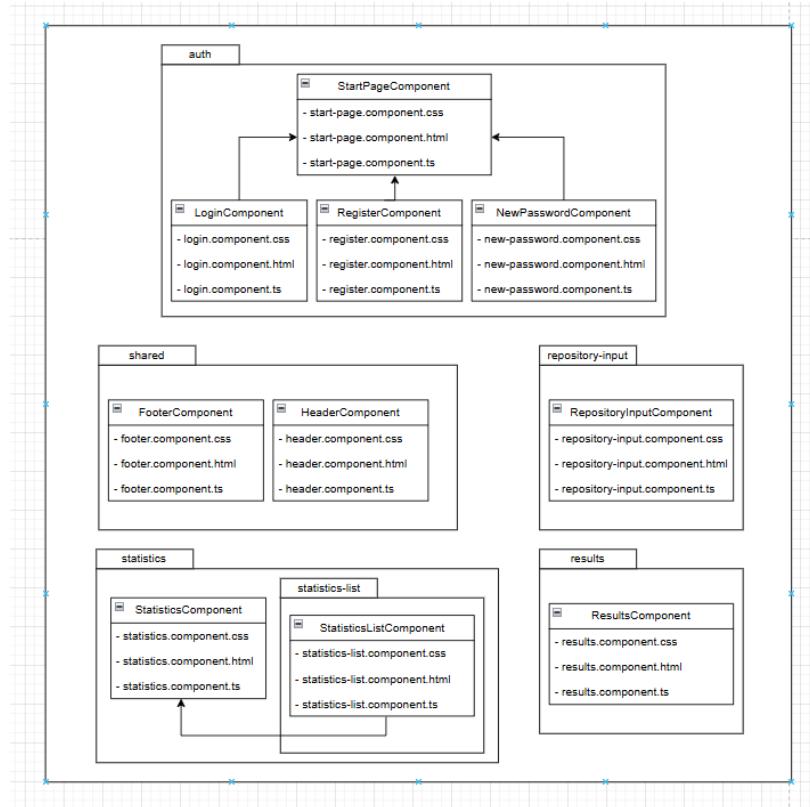


Figura C.4: Diagrama del directorio **modules** del frontend

- **Backend:**

- **controllers:** Se compone de authController.js y repoController.js, que son los ficheros con las funciones principales que definen la lógica general del código relacionado con la autentificación de usuarios y el manejo de repositorios, respectivamente. Para ello utilizarán los modelos definidos en "Models" las funciones menores definidas en "utils"
- **routes:** Su fichero repoRoutes.js define las rutas que usarán los servicios del frontend para acceder a las funciones de los controllers.
- **models:** Contiene los ficheros que definen cada tabla de la base de datos a través de la librería mongoose.
- **utils:** Contiene las funciones secundarias necesarias que usará repoController.js. Hay un fichero y función pequeña para cada tabla de métricas (*commits*, *issues*, etc), junto a un fichero github.js que obtiene datos del repositorio importantes como el título y autor; y para cada práctica ágil definida (Version control, Iterations, etc), además de algunas funciones de ayuda comorule-comparator.js, que compara las medidas de calidad de proceso entre el repositorio principal y los repositorios de referencia para evaluar las reglas

La particularidad de haber dividido el código entre frontend y backend refuerza enormemente la flexibilidad y la escalabilidad del diseño, así como la legibilidad y mantenimiento del código.

Patrones de diseño

No toda la flexibilidad, escalabilidad y robustez se deben sólo a la estructura interna del código, sino también a los patrones de diseño [2] aplicados. Estos patrones mejoran de la calidad del código y la reutilización de componentes, y refuerzan la legibilidad y mantenibilidad, así como una comunicación más clara entre desarrolladores o, en el caso del desarrollo de este proyecto, la comunicación entre el alumno autor y el tutor del TFG.

Para el diseño de este proyecto se han tenido en cuenta los principios SOLID [4] que, junto a ciertos patrones de diseño concretos, han sido de especial utilidad en una arquitectura como lo es la ya descrita de este proyecto. A continuación se detallan los patrones y principios de diseño más relevantes considerados:

- **Patrón fachada:** Este patrón se aplica especialmente en la comunicación entre el frontend y el backend a través de los servicios (AuthService, RepositoryService). Estos actúan como una "fachada" que simplifica el acceso a múltiples funciones complejas del backend.
 - Facilita la interacción del frontend con el backend al ocultar la complejidad de las peticiones HTTP y lógica interna.
 - Mejora la cohesión y disminuye el acoplamiento entre componentes.
 - Permite modificar el backend sin afectar al frontend, siempre que la fachada (servicio) mantenga su interfaz.
 - Aumenta la reutilización y testeo de código, al centralizar la lógica de acceso a datos.
- **Principio Abierto / Cerrado:** Este principio se observa en los controladores y módulos, los cuales están diseñados para permitir extensiones (nuevas rutas, pantallas, reglas) sin necesidad de modificar el código ya existente.
 - **Beneficio:** Facilita la adición de nuevas funcionalidades de forma segura.
 - **Beneficio:** Reduce el riesgo de introducir errores al mantener el código existente intacto.
 - **Beneficio:** Mejora la mantenibilidad y escalabilidad del sistema.
 - **Beneficio:** Favorece el desarrollo ágil, permitiendo iteraciones rápidas.
- **Patrón Inyección de Dependencias:** Angular, por diseño, utiliza inyección de dependencias para los servicios (AuthService, RepositoryService, HttpClient, etc.), permitiendo declarar dependencias en los constructores.
 - **Beneficio:** Aumenta la modularidad del código y promueve el bajo acoplamiento.
 - **Beneficio:** Mejora la reutilización y mantenimiento del código.
 - **Beneficio:** Permite una mayor flexibilidad en la configuración de servicios.
- **Principio de Separación de responsabilidades:** Toda la arquitectura se basa en una clara división de responsabilidades: frontend vs backend, componentes vs servicios, controladores vs modelos, etc.

- **Beneficio:** Mejora la legibilidad del código al estar organizado por función.
 - **Beneficio:** Permite a varios desarrolladores trabajar en paralelo en diferentes áreas del sistema.
 - **Beneficio:** Reduce errores al evitar dependencias innecesarias entre módulos.
 - **Beneficio:** Favorece la reutilización de módulos y componentes en otros proyectos.
- **Patrón Controlador:** Utilizado en el backend para organizar la lógica relacionada con diferentes dominios (AuthController, RepoController). Cada controlador es responsable de orquestar las funciones relacionadas con una entidad del sistema.
- **Beneficio:** Centraliza la lógica de cada parte del sistema, facilitando su entendimiento y modificación.
 - **Beneficio:** Permite escalar el sistema con facilidad al añadir nuevos controladores según se necesiten.
 - **Beneficio:** Mejora la trazabilidad de errores
 - **Beneficio:** Favorece el desacoplamiento entre la lógica de negocio y la lógica de acceso a datos.
- **Patrón decorador:** En Angular se emplea el patrón decorador mediante anotaciones como @Component, @Injectable, @Input, @Output, que permiten añadir información a las clases, propiedades o métodos sin modificar directamente su implementación.
- **Beneficio:** Permite extender funcionalidades de manera flexible sin modificar el código original.
 - **Beneficio:** Aumenta la legibilidad y organización del código, permitiendo declarar claramente el propósito de cada clase o propiedad.
 - **Beneficio:** Mejora la reutilización de componentes a través de la configuración por propiedades decoradas (@Input(), usado en **StatisticsListComponent**).

C.4. Diseño procedimental

En esta sección se describirá el flujo a seguir en las tareas más relevantes que se hacen en la aplicación mediante los **Casos de Uso (CU)** definidos previamente

CU1 - Creación y acceso a la cuenta

Este caso de uso permite al usuario crear una cuenta en la aplicación o acceder mediante inicio de sesión. Está dirigido a nuevos usuarios o a aquellos que desean autenticarse. El sistema valida la información antes de registrar o permitir el acceso como se puede ver en la figura C.5)

Pasos que sigue el CU:

1. El usuario accede a la página de registro o inicio de sesión.
2. Introduce su usuario y contraseña.
3. El sistema verifica los datos ingresados.
4. Según el caso, se crea una nueva cuenta o se otorga acceso.

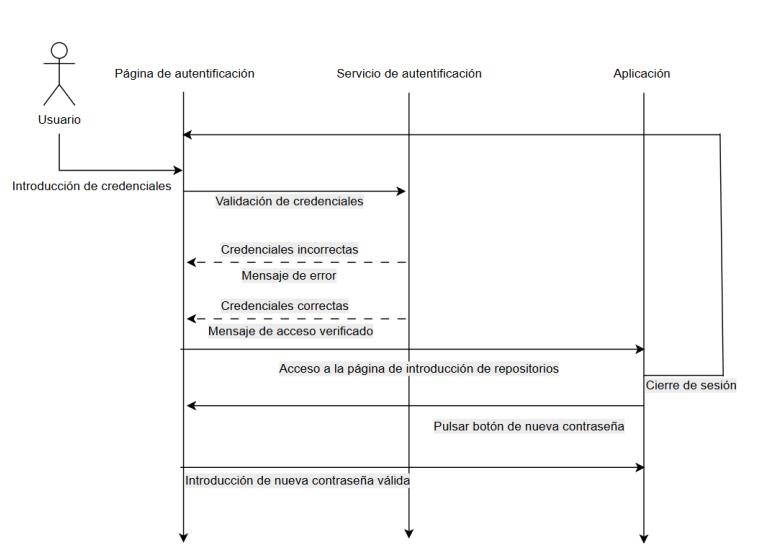


Figura C.5: Diagrama del caso de uso CU1

CU2 - Configuración de la cuenta

Permite al usuario modificar su configuración de cuenta, como el idioma o la contraseña. Es necesario haber iniciado sesión previamente. El sistema valida y aplica los cambios, tal como se muestra en la figura C.6.

Pasos que sigue el CU:

1. El usuario accede a la sección de configuración.
2. Selecciona si desea cambiar la contraseña o el idioma.
3. El sistema valida los datos ingresados y aplica los cambios.

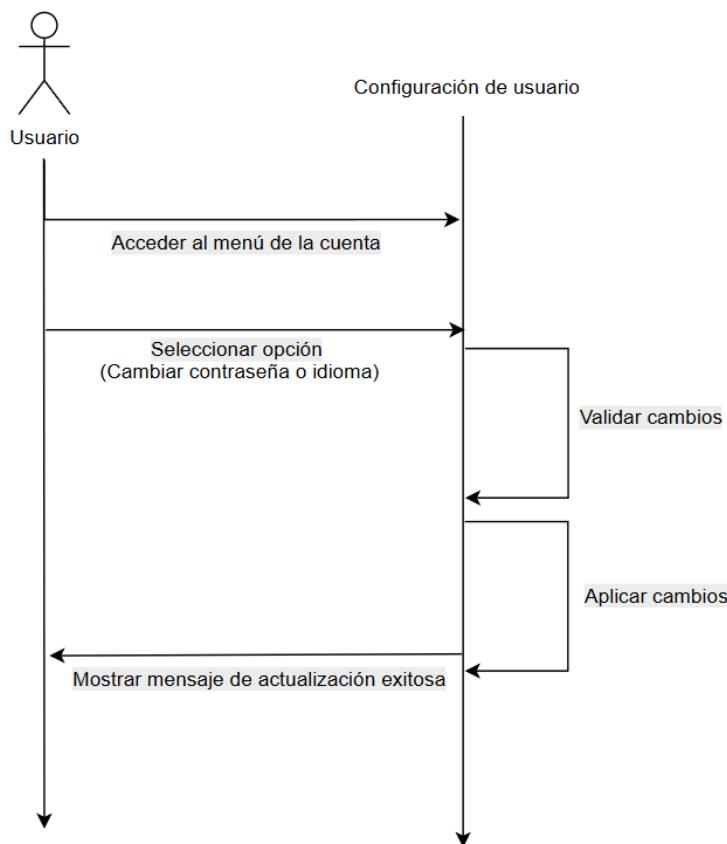


Figura C.6: Diagrama del caso de uso CU2

CU3 - Introducción y validación de URLs de repositorios

Este caso de uso permite al usuario introducir una o más URLs de repositorios GitHub para su posterior análisis. El sistema se encarga de validar tanto la estructura como el acceso. La figura C.9 describe un flujo de introducción y validación de URLs de repositorios para su posterior visualización.

Pasos que sigue el CU:

1. El usuario introduce una o varias URLs.
2. El sistema valida su formato y accesibilidad.
3. Si las URLs no son correctamente validadas saltará un mensaje de error. En caso contrario, el análisis se iniciará cuando el usuario pulse el botón..

CU4 - Configuración del análisis de repositorios

Este caso de uso permite definir cómo se analizarán los datos de los repositorios previamente introducidos. El usuario puede personalizar fechas, tipos de intervalos y métricas. Cada paso incluye validaciones, y al final se habilita la opción de comparar (ver figura C.7).

Pasos que sigue el CU:

1. El usuario elige el número de días sobre el cual se basarán las medidas de calidad que reflejan medias estadísticas.
2. El usuario elige entre analizar los repositorios al completo o por intervalos de tiempo.
3. El usuario elige en qué intervalos de tiempo realizar el análisis.
4. Se validan las entradas y se habilita el botón *Comparar*.

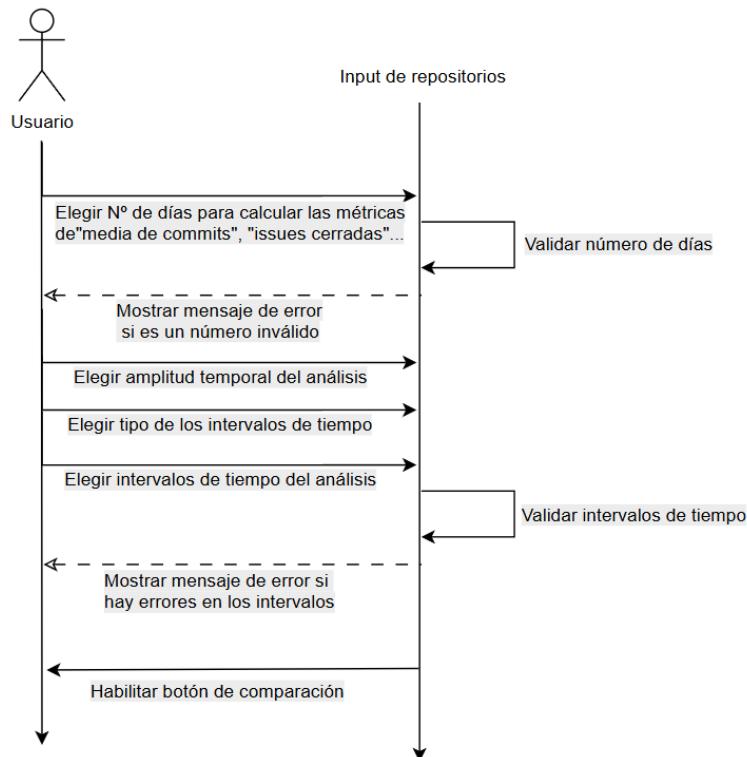


Figura C.7: Diagrama del caso de uso CU4

CU5 - Gestión de grupos de repositorios de referencia

Este caso de uso permite gestionar grupos de repositorios previamente guardados como referencia comparativa. El usuario puede cargarlos o eliminarlos, siempre y cuando tenga al menos un grupo guardado. (Ver figura C.8)

Pasos que sigue el CU:

1. El usuario accede a la gestión de grupos de repositorios.
2. Selecciona uno para cargarlo.
3. Opcionalmente elimina uno o más grupos.

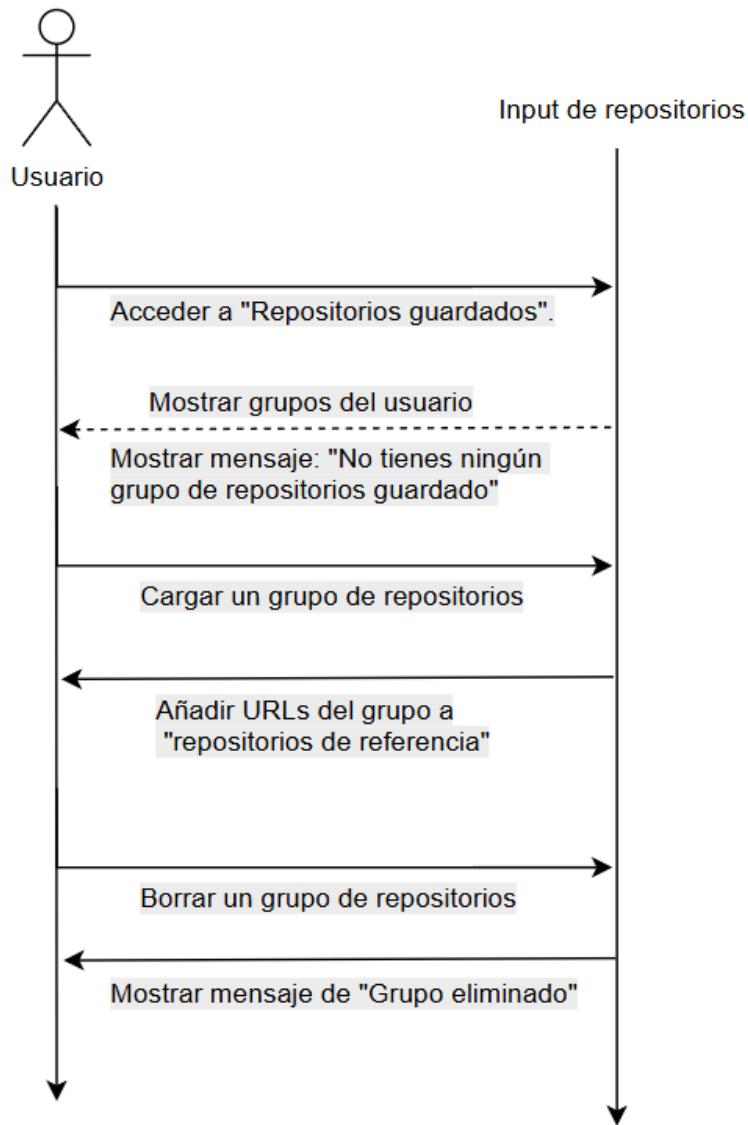


Figura C.8: Diagrama del caso de uso CU5

CU6 - Visualización de prácticas ágiles

Permite al usuario visualizar los resultados de un análisis automático de prácticas ágiles basado en las medidas de calidad de proceso extraídas de los repositorios. Se presentan los resultados en forma gráfica y textual, indicando qué prácticas se cumplen, en qué medida, y en qué aspectos hay lugar para mejoras. Esto puede observarse en la figura C.9, junto a la posibilidad de

que el usuario cambie de pantalla y vuelva a la de este caso de uso o pase a la del CU7.

Pasos que sigue el CU:

1. El sistema muestra los resultados del uso de prácticas ágiles del análisis en forma gráfica y textual.
2. El usuario revisa qué prácticas de agilidad se cumplen, cuáles no, y los motivos.

CU7 - Comparación de medidas de calidad de proceso

Este caso de uso permite al usuario comparar visualmente las métricas de calidad de su repositorio analizado con aquellas de uno o más repositorios de referencia previamente cargados. El sistema genera listas comparativas para facilitar la interpretación. Esta funcionalidad, junto a las de los casos de uso CU3 y CU6 se detalla gráficamente en la figura ?? siguiendo un flujo detallado del proceso de validación de URLs de repositorios para después ver los resultados.

Pasos que sigue el CU:

1. El usuario accede a la sección de comparación.
2. El sistema genera y muestra listas comparativas con las métricas seleccionadas del repositorio actual y los de referencia.
3. El usuario analiza los resultados para evaluar el estado de su proyecto.

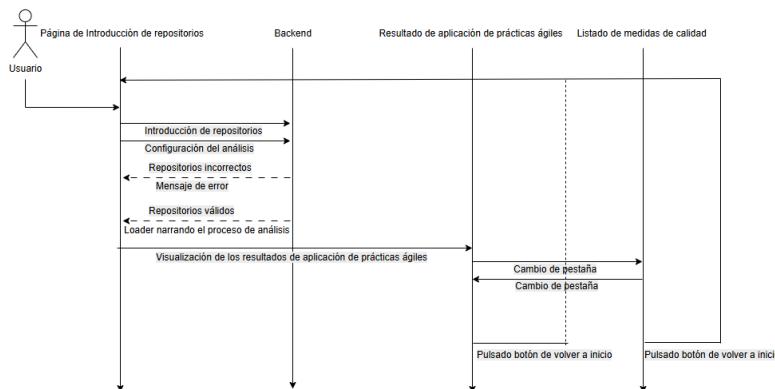


Figura C.9: Diagrama del caso de uso CU7

Apéndice D

Documentación técnica de programación

D.1. Introducción

Este apartado del anexo detalla una guía técnica completa orientada a desarrolladores software que quieran trabajar o utilizar la aplicación de Asistente de prácticas ágiles para repositorios en GitHub. Se incluye una guía para clonar el entorno de desarrollo, compilar y ejecutar la aplicación, detallando los pasos necesarios a realizar en la configuración del proyecto para lograrlo.

D.2. Estructura de directorios

A continuación se detalla la estructura de ficheros de todo el proyecto, incluyendo tanto el código fuente de la aplicación desarrollada como la documentación, y ficheros readme.

- /: Directorio raíz. Contiene el resto de directorios, junto al fichero .gitignore y el fichero readme.md principal del proyecto.
- **/Memoria/**: Directorio que contiene la documentación del proyecto. En su interior se encuentran:
 - Directorio tex/ con los archivos L^AT_EX que componen la memoria principal y el anexo.

- Contiene el fichero L^AT_EX principal de la memoria y el fichero L^AT_EX principal del anexo
 - Ficheros PDF generados correspondientes a la memoria y al anexo.
 - Carpeta /img con las imágenes utilizadas en los documentos.
 - Un archivo .gitignore específico para esta carpeta.
 - Un fichero README.md con información sobre la plantilla utilizada para redactar la documentación.
- /src/: Carpeta principal que contiene el código fuente de la aplicación web desarrollada. Dentro de este directorio se encuentran:
- La carpeta frontend/, que contiene el código de la interfaz de usuario.
 - La carpeta backend/, que incluye la lógica de servidor y la API, y los modelos de las tablas de la base de datos.
 - Una copia de la carpeta node_modules/ con las dependencias necesarias.
 - Un archivo README.md que proporciona una guía sobre la organización del código y cómo distinguir entre el backend y el frontend.

D.3. Manual del programador

Esta sección tiene el objetivo de ser utilizada por desarrolladores de software, programadores o personas interesadas en trabajar en el proyecto para guiar a los mismos a la hora de configurar el entorno de desarrollo y compilar y ejecutar el código fuente para poder probar el mismo. Para ello se numeran los siguientes puntos a tener en cuenta.

Entorno de desarrollo

Para configurar el entorno de desarrollo del proyecto, se sugiere utilizar las herramientas listadas a continuación:

- **Visual Studio Code:** Se trata de un editor de código fuente gratuito, ligero y versátil. Es una herramienta útil para escribir, editar, depurar y gestionar código en una amplia variedad de lenguajes de programación.

Se caracteriza por su flexibilidad, extensibilidad y capacidad para integrarse con diversas herramientas y extensiones.

- **Copilot (Opcional):** Es una extensión de *Visual Studio Code* que asiste al programador con una inteligencia artificial a través de sugerencias de código, lo que acelera mucho el proceso de desarrollo.
- **MongoDB Compass:** Una aplicación de escritorio desde la cual gestionar la conexión y contenidos de la base de datos ubicada en **Mongo DB Atlas**[5].
- **GitHubDesktop:** Un entorno gráfico de GitHub de gran utilidad gracias a la información visual que brinda sobre los ficheros del repositorio modificados, añadidos o borrados. Ofrece una gran ayuda para hacer *commits*, *pull requests* y *merges* de calidad.

Obtención del código fuente

El código fuente del proyecto está disponible en el repositorio en GitHub <https://github.com/lod1004/GII-TFG-Asistente-de-practicas-agiles-para-repositorios-en-GitHub>. Es necesario clonar el proyecto para trabajar en él de forma activa. A continuación se describen dos formas para lograrlo:

- Descargar e instalar **GitHub Desktop** desde <https://desktop.github.com/> si se prefiere clonar el proyecto gráficamente.
- Clonar el repositorio del proyecto desde GitHub de una de las siguientes formas:
 - **Opción 1 - Gráfica (recomendada para usuarios sin experiencia con Git):** Abrir GitHub Desktop, hacer clic en "*File > Clone repository*", pegar la URL del repositorio y elegir una ubicación en el disco local.
 - **Opción 2 - Terminal:** Ejecutar `git clone URL-del-repositorio` en una terminal ubicada en la carpeta donde se desea clonar el repositorio.

Pasos para configurar el entorno

Una vez clonado el repositorio en una carpeta local, para instalar y configurar el entorno del proyecto correctamente se recomienda seguir los pasos numerados a continuación:

1. Descargar e instalar **Visual Studio Code** desde <https://code.visualstudio.com/>.
2. Descargar e instalar **Node.js** desde <https://nodejs.org/en/download>.
Se recomienda la versión LTS más reciente.
3. Instalar Angular CLI ejecutando en una terminal uno de los siguientes comandos:
 - De forma global: **npm install -g @angular/cli@19.2.3**
 - O bien, de forma local en la carpeta del proyecto: **npm install @angular/cli@19.2.3**
4. Descargar e instalar **MongoDB Compass** desde <https://www.mongodb.com/products/compass>, para visualizar y gestionar la base de datos de forma gráfica.

Además del entorno base, Visual Studio Code permite instalar algunas extensiones que pueden facilitar significativamente el desarrollo, mejorar la productividad y proporcionar soporte inteligente durante la programación. Una de las más destacadas es la ya mencionada extensión de Copilot, una extensión basada en inteligencia artificial que sugiere automáticamente líneas completas de código y funciones enteras a partir del contexto del proyecto y de los comentarios escritos por el programador. Esto resulta especialmente útil en entornos colaborativos o cuando se trabaja con estructuras repetitivas. También se sugiere instalar extensiones como ESLint, que ayuda a mantener un estilo de código coherente y libre de errores comunes, Prettier para el formateo automático del código, y Angular Language Service, que proporciona autocompletado y detección de errores específicos de Angular en tiempo real. Para entornos backend con Node.js y MongoDB, extensiones como MongoDB for VS Code permiten realizar consultas directamente desde el editor y visualizar colecciones, lo cual es muy conveniente durante el desarrollo y depuración. Estas herramientas integradas en el editor permiten una experiencia de desarrollo más fluida, profesional y eficiente.

D.4. Compilación, instalación y ejecución del proyecto

Una vez configurado el entorno del proyecto y los servicios externos, se detallan los siguientes pasos a seguir necesarios para compilar e instalar lo necesario para poder ejecutar la aplicación.

1. Desde la raíz del repositorio clonado, navegar a la carpeta **src/backend** y ejecutar los siguientes comandos en una terminal:
 - **npm install** para instalar las dependencias del servidor.
 - En el fichero .env del back se debe incluir un token de GitHub para poder hacer las peticiones a través de la API. Se puede ver más información sobre los tokens de GitHub en el siguiente enlace: <https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens>.
 - También puede ser necesario añadir una conexión de MongoDB Atlas o, en su defecto, dejar la que ya está y usar la base de datos existente.
 - Ejecutar el backend con **node server.js**, según lo especificado en la documentación del proyecto.
2. En una nueva terminal, navegar a **src/frontend** y ejecutar:
 - **npm install** para instalar las dependencias del cliente.
 - **ng serve -o** para iniciar la aplicación Angular en el navegador y que la pestaña se abra de forma automática.
 - **Importante** asegurarse de que los servicios del frontend (AuthService y RepositoryService) utilicen el environment correcto. ('../../environments/environment' para ejecutar en local o '../../environments/environment.prod' para acceder al backend subido en el despliegue continuo en **Render**)

D.5. Pruebas del sistema

El sistema fue sometido a diversas pruebas para garantizar su correcto funcionamiento, principalmente manuales, por parte del autor, tutor, y usuarios externos. Debido a la ventaja de división de código en componentes de Angular, la mayoría de los cambios en la aplicación consistían en añadir código, y raramente modificaban lo hecho previamente, lo que ahorró mucho tiempo de pruebas. Sin embargo, hay dos tipos de pruebas relevantes que se realizaron durante el desarrollo del sistema.

Prueba general del sistema

La prueba general principal usada para comprobar el funcionamiento completo del sistema fue la de comparar un repositorio consigo mismo. Esta

prueba garantiza que tanto la recopilación de métricas como la lógica de comparación funcionan de forma coherente.

Según la lógica interna de la aplicación, al comparar un repositorio consigo mismo en los mismos intervalos de tiempo, se espera que los resultados se comporten de la siguiente forma:

- Las medidas de calidad de proceso deben ser idénticas en ambos repositorios, dado que se trata del mismo repositorio y el análisis se realiza bajo las mismas condiciones y fechas.
- En la evaluación de prácticas ágiles, todas las medidas aplicables deben considerarse superadas, ya que no hay diferencias entre los repositorios comparados. Las únicas medidas que podrían no superarse son aquellas que no se aplican al repositorio (por ejemplo, por falta de datos), ya que estas se contabilizan como no completadas, incluso si en el repositorio comparado tampoco se aplican.

Esta prueba es útil no solo para verificar que el sistema se comporta como se espera en condiciones ideales, sino también para validar la integridad de los datos obtenidos desde GitHub, el cálculo de métricas y la aplicación de las reglas de evaluación automatizadas.

Pruebas con SonarQube

Además de la lógica interna de evaluación y comparación, el proyecto hace uso de **SonarQube**[7], integrado en el repositorio de GitHub como plataforma de análisis de calidad del código. SonarQube permite detectar de forma automatizada problemas como errores de seguridad, código duplicado, malas prácticas y código que no sigue los estándares recomendados.

La integración con SonarQube ha resultado esencial para mantener un código de alta calidad y detectar posibles refactorizaciones necesarias de forma rápida y visual.

Gracias a esta herramienta se ha podido reforzar la seguridad, mantenibilidad y fiabilidad del proyecto, así como evitar la duplicación de código y las posibles vulnerabilidades que puedan surgir durante el proceso de desarrollo software.

Apéndice E

Documentación de usuario

E.1. Introducción

Este apartado del anexo de Asistente de prácticas ágiles para repositorios en GitHub tiene como objetivo proporcionar a los usuarios las instrucciones necesarias para utilizar la aplicación web desarrollada en el proyecto. Está orientado a usuarios que hayan decidido usar la aplicación para evaluar el uso de prácticas ágiles en sus repositorios, así como observar las diferentes medidas de calidad de proceso que puedan ayudar a visualizar el progreso del desarrollo de software. Se describirán los requisitos recomendados de los usuarios, los pasos para la instalación y, al final, una guía de usuario que explica cómo realizar los distintos casos de uso aprovechando las funcionalidades disponibles en la aplicación.

E.2. Requisitos de usuarios

Para poder utilizar la aplicación web, se requiere del dispositivo del usuario los requisitos listados a continuación:

Permisos requeridos del usuario

La aplicación web requiere únicamente de acceso a internet para ser utilizada tanto para registrarse e iniciar sesión como para analizar y comparar repositorios. No se requiere de una dirección de correo electrónico o una cuenta de GitHub para utilizar los servicios de la aplicación, pero se debe tener en cuenta que, aunque los repositorios a analizar deben ser obligatoriamente **públicos** (lo cual deja a entender que el propietario accede

a que se obtenga información del repositorio), se accederá a los contenidos de estos para proceder con la comparación, incluyendo:

- **Información de *issues*:** Número de *issues*, fechas, título, descripción, etiquetas, imágenes, autores y personas asignadas.
- **Información de *commits*:** Número de *commits*, fechas, título, descripción, y autores.
- **Información de *releases*:** Número de *releases* y *tags*, fechas, título, descripción y autores.
- **Información de *pull requests*:** Número de *pull requests*, fechas, título, descripción, etiquetas, imágenes, autores y personas asignadas.
- **Información de ficheros *workflow* de GitHub Actions:** (Número de ficheros, ejecuciones exitosas y frecuencia de ejecución).
- **Información de los participantes del repositorio:** Nombre de los participantes y su actividad en el repositorio.

Requisitos de Hardware recomendados

- Se recomienda utilizar un computador para usar la aplicación, pues está pensada para ser utilizada en entornos de desarrollo de software, por lo que no se sugiere acceder desde un dispositivo móvil.
- Sistema operativo funcional con navegadores web (Firefox, Google Chrome, Opera, etc).

E.3. Instalación

Debido al despliegue continuo de la aplicación web, no es necesario realizar ninguna instalación para acceder a las funcionalidades de la misma. Basta con acceder a la siguiente URL con el despliegue final de la aplicación: <https://gi-tfg-asistente-de-practicas-agiles-para-repositorio-f3z9mn3lt.vercel.app/>

En caso de querer configurar, instalar y ejecutar el código fuente en formato local, se dispone de una explicación detallada por pasos en el apartado anterior de este documento.

E.4. Manual del Usuario

En esta sección se detallan todas las funcionalidades disponibles en la aplicación paso por paso para ser utilizadas por los usuarios de manera efectiva.

Creación y acceso a la cuenta

Permite al usuario crear una cuenta mediante la opción de registro y acceder a la aplicación mediante inicio de sesión.



Figura E.1: Inicio de sesión del usuario



Figura E.2: Registro del usuario

- El usuario accede a la página de registro o inicio de sesión.

- El usuario introduce los datos necesarios (usuario y contraseña).
- El sistema verifica la validez de los datos.
- El sistema registra al usuario o permite el acceso según corresponda.

Configuración de la cuenta

Permite al usuario cambiar la contraseña y modificar el idioma de la aplicación.



Figura E.3: Configuración del usuario

- El usuario puede cambiar el idioma de la interfaz entre inglés y español mediante el ícono de la bandera
- El usuario pulsa la opción de cambio de contraseña e introduce los datos necesarios (usuario, contraseña actual y nueva contraseña).
- El sistema verifica la validez de los datos.
- El sistema cambia la contraseña del usuario e inicia sesión automáticamente.

Introducción y validación de URLs de repositorios

Permite al usuario introducir URLs de repositorios GitHub y valida su formato y accesibilidad.



Figura E.4: Validación de URLs de repositorios

- El usuario introduce una URL de repositorio a analizar y una o varias URLs de repositorios de referencia.
- El sistema valida las urls.

Configuración del análisis de repositorios

El usuario configura cómo se analizarán los datos del repositorio: número de días para calcular las métricas temporales, análisis completo o por intervalos, usar intervalos absolutos o relativos y los períodos de los intervalos de tiempo del análisis.

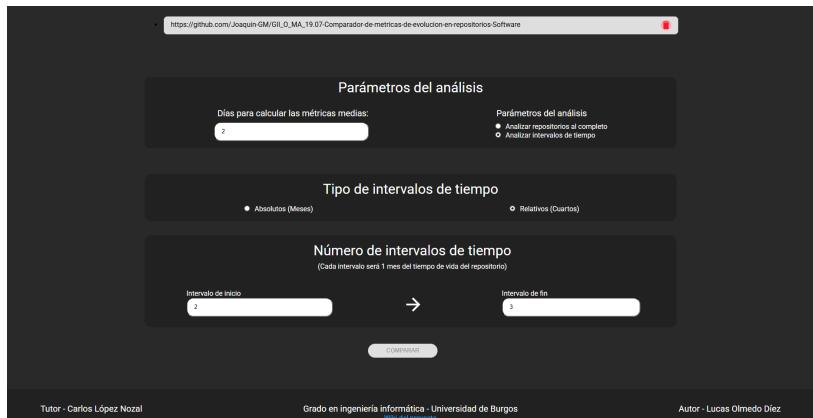


Figura E.5: Configuración del análisis de los repositorios

- El usuario selecciona si analizar los repositorios al completo o analizarlos por intervalos.
- Ajusta el número de días para calcular las métricas de medias.
- Elige el tipo de intervalos entre absolutos o relativos.
- Se eligen los intervalos del análisis.

Gestión de grupos de repositorios de referencia

Permite al usuario cargar o borrar grupos de repositorios previamente guardados para usarlos como referencia comparativa en los análisis.



Figura E.6: Gestión de grupos de repositorios

- El usuario accede a la sección de gestión de repositorios guardados.
- Selecciona un grupo de repositorios guardado previamente y lo carga.
- (Opcional) Elimina uno o más grupos guardados si lo desea.

Visualización de prácticas ágiles

Permite al usuario visualizar el grado de adopción de buenas prácticas ágiles evaluadas automáticamente por el sistema a partir de los datos del repositorio.



Figura E.7: Evaluación de buenas prácticas ágiles

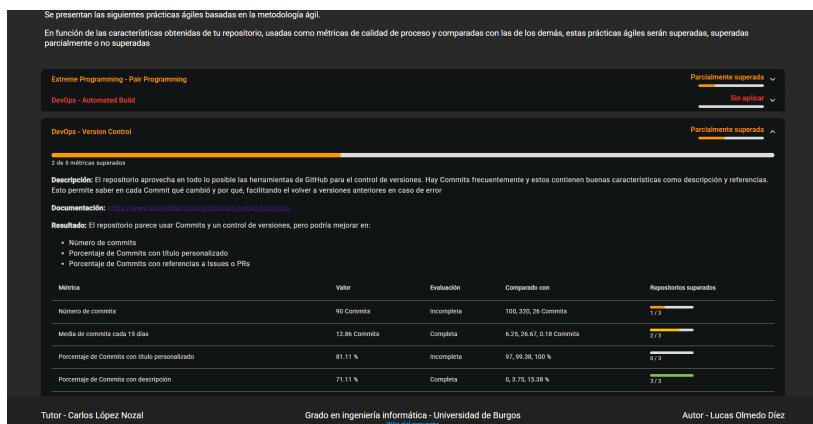


Figura E.8: E6: Detalle de las prácticas ágiles

- El sistema muestra los resultados del análisis de buenas prácticas en formato gráfico y textual.
- El usuario puede seleccionar cada práctica ágil para ver los detalles de su evaluación y comparación entre repositorios.

Comparación de medidas de calidad de proceso

Permite comparar visualmente las métricas del repositorio analizado con las de los repositorios de referencia cargados.



Figura E.9: Comparación con repositorios de referencia

- El usuario accede a la sección de comparación.
 - El sistema genera y muestra listas comparativas de las métricas seleccionadas.
 - El usuario puede elegir qué repositorio de referencia mostrar para compararlo.

Apéndice F

Anexo de sostenibilización curricular

F.1. Introducción

Durante el desarrollo de este TFG, centrado en la creación de una aplicación web para el análisis automatizado de prácticas ágiles y métricas de calidad de proceso en proyectos alojados en GitHub, he aplicado diversas competencias de sostenibilidad de forma transversal. Este trabajo ha sido una oportunidad para reflexionar sobre cómo el desarrollo de software puede contribuir de manera activa al cumplimiento de los Objetivos de Desarrollo Sostenible (*Objetivos de Desarrollo Sostenible (ODS)*).

El análisis del desarrollo de software, combinado con el uso de las prácticas ágiles permite fomentar procesos de mejora continua que reduzcan el desperdicio de recursos humanos y computacionales, a la vez que promueven modelos colaborativos, transparentes y sostenibles. La herramienta desarrollada contribuye a una mayor conciencia en torno a la eficiencia y responsabilidad dentro de los equipos de desarrollo de software.

F.2. Competencias de Sostenibilidad en el proyecto

Adquisición de conocimiento de competencias de sostenibilidad

A través de este proyecto, he sido capaz de relacionar el desarrollo tecnológico con los retos globales de sostenibilidad. La aplicación se enfoca en facilitar el análisis reflexivo de proyectos de software, lo que permite a los equipos evaluar sus prácticas y alinear sus metodologías con objetivos más amplios como la eficiencia, la justicia laboral (mediante la mejora del clima de equipo), o el acceso equitativo al conocimiento abierto.

Sostenibilidad al tomar decisiones de proyecto

He priorizado el uso eficiente de recursos en la arquitectura del sistema, optando por tecnologías *open-source*, integraciones ligeras con la API de GitHub para minimizar el consumo energético y computacional. Se evita la persistencia innecesaria de datos, lo que contribuye al uso responsable de almacenamiento y procesamiento en servidores.

Fomento de la participación colectiva

Este trabajo promueve el uso de métricas abiertas y replicables para evaluar proyectos en GitHub, lo que incentiva la participación colaborativa en comunidades de desarrollo de software. Al facilitar la evaluación colectiva de buenas prácticas, la herramienta ayuda a fortalecer ecosistemas de software más transparentes, resilientes y responsables.

Ética del proyecto

Durante el desarrollo se ha seguido un enfoque ético que evita la recopilación innecesaria de datos personales, se promueve el análisis de repositorios públicos y se fomenta el uso responsable de la tecnología como instrumento para mejorar la transparencia y la calidad en el trabajo en equipo, sin fomentar modelos competitivos insostenibles o presiones laborales indebidas.

Conciencia sobre Sostenibilidad en el Desarrollo de Software

Una parte clave del proyecto ha sido su valor educativo. La herramienta actúa como recurso de concienciación para estudiantes y desarrolladores que buscan mejorar sus prácticas en el desarrollo de software y planificación para el mismo. Fomenta la reflexión sobre el ciclo de vida de los proyectos de software, sus tareas y componentes, la calidad de sus procesos, y su sostenibilidad a largo plazo.

F.3. Conclusión

El desarrollo de este proyecto ha fortalecido mi comprensión de cómo el software no es únicamente una herramienta técnica, sino un elemento transformador con capacidad de influir en dimensiones sociales y económicas. Aplicar competencias de sostenibilidad en el diseño, implementación y propósito de esta herramienta me ha hecho más consciente de las decisiones éticas y técnicas que tomamos como desarrolladores de software. Estoy convencido de que estos tipos de soluciones tecnológicas son fundamentales para alcanzar un desarrollo sostenible.

Bibliografía

- [1] Agile Alliance. Subway map to agile practices, 2023.
- [2] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
- [3] GitHub. Github docs: Documentation for github usage and apis. <https://docs.github.com>, 2024. [Documentación oficial; consultado el 9-junio-2025].
- [4] Robert C. Martin. Solid principles of object-oriented design. <https://web.archive.org/web/20240101000000/http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOOD>, 2024. [Archivo web del sitio de Uncle Bob; consultado el 9-junio-2025].
- [5] MongoDB Inc. Mongodb atlas: The multi-cloud database service. <https://www.mongodb.com/cloud/atlas>, 2024. [Sitio web oficial; consultado el 9-junio-2025].
- [6] Render. Render: The modern cloud for application developers. <https://render.com>, 2024. [Sitio web oficial; consultado el 9-junio-2025].
- [7] SonarSource. Sonarqube: Continuous inspection of code quality. <https://www.sonarsource.com/products/sonarqube/>, 2024. [Sitio web oficial; consultado el 9-junio-2025].
- [8] Vercel. Vercel: Develop. preview. ship. <https://vercel.com>, 2024. [Sitio web oficial; consultado el 9-junio-2025].