MÁSTER UNIVERSITARIO EN INGENIERÍA EN TECNOLOGÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

ANÁLISIS DE VIABILIDAD Y DISEÑO DE LA NUEVA METODOLOGÍA DE ENSEÑANZA DE PROGRAMACIÓN BASADA EN TECNOLOGÍAS DE CONTROL DE VERSIONES (NME-CV)

Estudiante *Martín García-Cuevas, Eder*
Director *Ferro Vázquez, Armando*
Departamento *Ingeniería de Comunicaciones*

Curso académico *2021/2022*

Bilbao, 2 de marzo de 2022

Resumen trilingüe

Como consecuencia de la necesidad de corregir las actividades que cada estudiante debe resolver como parte del proceso de evaluación, en 2018 se realizó un estudio técnico para la creación del sistema NME-CV, con el que evaluar automáticamente las actividades de las asignaturas de programación. En este documento, se parte de los resultados de dicho estudio para realizar un análisis técnico de viabilidad que permita hallar una solución técnica para el sistema, aportando una arquitectura software detallada. Para realizar este proceso, se realizan una serie de prototipos demostradores con las que tomar decisiones con criterio.

Ebaluazio-prozesuaren barruan ikasle bakoitzak ebatzi behar dituen jardueren zuzenketaren ondorioz, 2018an NME-CV sistema sortzeko azterketa teknikoa egin zen programazio-gaien jarduerak automatikoki ebaluatzeko. Dokumentu honetan, aipatutako azterketaren emaitzetatik abiatu da sistemarako irtenbide tekniko bat aurkitzea ahalbidetzen duen bideragarritasun teknikoaren analisia egiteko, software-arkitektura zehatza eskainiz. Prozesu hori aurrera eramateko, zenbait prototipo egiten dira, irizpideekin erabakiak hartzeko.

As a result of the need for correcting the exercises each student must undertake as part of the evaluation process, a technical study for the creation of the NME-CV system was executed in 2018, with which to automatically evaluate the exercises in programming subjects. In this document, the result of such study is the beginning to carry out a technical feasibility analysis that allows finding a technical solution for the system, contributing with a detailed software architecture. For the accomplishment of this process, a series of demonstrator prototypes will be developed, with whom informed decisions will be taken.

Índice

Resumen trilingüe I
Índice II
Índice de figuras y tablas V
Índice de figuras V
Índice de tablas V
Tablas de acrónimos VII
Acrónimos generales VII
Acrónimos propios VII
1. Introducción y contexto 8
2. Objetivos y alcance 10
3. Beneficios 11
3.1. Beneficios técnicos 11
3.2. Económicos 11
3.3. Socioculturales 12
4. Especificaciones generales 13
4.1. Especificaciones generales de los procesos del sistema 13
4.2. Especificaciones generales de la arquitectura 14
4.3. Especificaciones generales sobre apartado de docencia 14
4.4. Especificaciones generales sobre la seguridad del sistema 15
5. Análisis de viabilidad técnica 16
5.1. Problemáticas 16
5.1.1. Problemática I: Gestión de repositorios 16
5.1.2. Problemática II: Despliegue del contexto de pruebas 17
5.1.3. Problemática III: Gestión del entorno académico 18
5.1.4. Problemática IV: Interfaz de gestión de los usuarios 19
5.2. Alternativas 20
5.2.1. Alternativas a la Problemática I: Gestión de los repositorios 20
5.2.2. Alternativas a la Problemática II: Despliegue del contexto de pruebas 23
5.2.3. Alternativas a la Problemática III: Gestión de entorno académico 24
5.2.4. Alternativas a la Problemática IV: Interfaz de gestión de los usuarios 26
5.3. Criterios de evaluación 28

5.3.1. Criterios comunes 28
5.3.2. Criterios específicos de la Problemática I: Gestión de los repositorios 29
5.3.3. Criterios específicos de la Problemática II: Despliegue del contexto de pruebas 31
5.3.4. Criterios específicos de la Problemática III: Gestión del entorno académico 32
5.3.5. Criterios específicos de la Problemática IV: Interfaz de gestión de los usuarios 34
5.4. Decisión 35
5.4.1. Solución de la Problemática I: Gestión de los repositorios 35
5.4.2. Solución de la Problemática II: Despliegue del contexto de pruebas 35
5.4.3. Solución de la Problemática III: Gestión del entorno académico 36
5.4.4. Solución de la Problemática IV: Interfaz de gestión de los usuarios 36
6. Análisis Funcional 37
6.1. Entornos de aplicación y usuarios potenciales 37
6.2. Análisis y especificaciones funcionales 38
6.2.1. Especificaciones funcionales de la forja 38
6.2.2. Especificaciones funcionales de los contextos y aplicación de evaluación 38
6.2.3. Especificaciones funcionales sobre el modelo de datos 38
6.2.4. Especificaciones funcionales de los aspectos docentes 38
6.2.5. Especificaciones funcionales de integración 38
7. Diseño 43
7.1. Integración de los módulos 43
7.2. Arquitectura 44
7.2.1. Comunicación entre módulos 45
7.2.2. Interfaces de usuarios con el sistema 46
7.3. Descripción detallada de los módulos 48
7.3.1. Descripción detallada del Módulo Repositorio 49
7.3.2. Descripción detallada del Módulo Evaluador 53
7.3.3. Descripción detallada del Módulo Gestor 56
7.3.4. Descripción detallada del Módulo Datos 59
7.3.5. Descripción detallada del Módulo Docencia 62
7.4. Despliegue de la arquitectura 64
7.5. Conclusiones del diseño 65

8. Plan de trabajo y presupuesto 66
8.1. Plan de trabajo 66
8.1.1. Descripción de paquetes de trabajo 67
8.1.2. Cronograma y resumen del plan de trabajo 73
8.2. Presupuesto 84
8. Conclusiones 86
9. Referencias 87
ANEXO I: Análisis y especificaciones funcionales detalladas 88
Especificaciones funcionales de la forja 88
Especificaciones funcionales de los contextos y aplicación de evaluación 92
Especificaciones funcionales sobre el modelo de datos 93
Especificaciones funcionales de los aspectos docentes 95
Especificaciones funcionales de integración 97
ANEXO II: Lista de prototipos desarrollados 100
ANEXO III: Funcionalidades a futuro 103
Pruebas ajenas al funcionamiento del código 103
Clonación de issues y wikis 103
Movimiento de tarjetas 104
Protección de modificación del fichero de pruebas 104
Salta automático de branches 105
ANEXO IV: Partida de paquetes de trabajo desglosada 106

Índice de figuras y tablas

Índice de figuras

Figura 1. Módulos del sistema NME-CV 43
Figura 2. Arquitectura del sistema NME-CV 45
Figura 3. Comunicación entre módulos 46
Figura 4. Usuarios del sistema y módulos que utilizan 48
Figura 5. Arquitectura con diseño detallado 49
Figura 6. Diseño interno del MRe 50
Figura 7. Diseño interno del MEv 54
Figura 8. Diseño interno del MGe 56
Figura 9. Diseño interno del MDa 60
Figura 10. Diseño interno del MDo 63

Índice de tablas

Tabla 1. Acrónimos generales VII
Tabla 2. Acrónimos propios VII
Tabla 3. Resumen de especificaciones generales 13
Tabla 4. Resumen de especificaciones funcionales de la Problemática I 17
Tabla 5. Resumen de especificaciones funcionales de la Problemática II 18
Tabla 6. Resumen de especificaciones funcionales de la Problemática III 19
Tabla 7. Resumen de especificaciones funcionales de la Problemática IV 20
Tabla 8. Tabla de decisión de la Problemática I 35
Tabla 9. Tabla de decisión de la Problemática II 36
Tabla 10. Tabla de decisión de la Problemática III 36
Tabla 11. Tabla de decisión de la Problemática IV 36
Tabla 12. Resumen de las especificaciones funcionales de la forja 39
Tabla 13. Resumen de las especificaciones funcionales de los contextos y aplicación de evaluación 39
Tabla 14. Resumen de las especificaciones funcionales del modelo de datos 40
Tabla 15. Resumen de las especificaciones funcionales de aspectos docentes 41
Tabla 16. Resumen de las especificaciones funcionales de integración 42
Tabla 17. Tareas del PT0 67
Tabla 18. Unidades de entrega del PT0 67
Tabla 19. Tareas del PT1 68
Tabla 20. Unidades de entrega del PT1 68
Tabla 21. Tareas del PT2 69
Tabla 22. Unidades de entrega del PT2 69
Tabla 23. Tareas del PT3 69
Tabla 24. Unidades de entrega del PT3 70
Tabla 25. Tareas del PT4 70

Tabla 26. Unidades de entrega del PT4 70
Tabla 27. Tareas del PT5 71
Tabla 28. Unidades de entrega del PT5 71
Tabla 29. Tareas del PT6 72
Tabla 30. Unidades de entrega del PT6 72
Tabla 31. Tareas del PT7 72
Tabla 32. Unidades de entrega del PT7 72
Tabla 33. Hitos del proyecto 73
Tabla 34. Cronograma del PT0 74
Tabla 35. Cronograma del PT1 75
Tabla 36. Cronograma del PT2 76
Tabla 37. Cronograma del PT3 77
Tabla 38. Cronograma del PT4 78
Tabla 39. Cronograma del PT5 79
Tabla 40. Cronograma del PT6 80
Tabla 41. Cronograma del PT7 (junto con los hitos) 81
Tabla 42. Resumen de tareas con desglose de horas por figura 82
Tabla 43. Resumen de unidades entregables 83
Tabla 44. Costes horarios de cada figura 84
Tabla 45. Partida de paquetes de trabajo 84
Tabla 46. Partida de amortizaciones 85
Tabla 47. Partida de gastos 85
Tabla 48. Presupuesto total 85
Tabla 49. Partida de paquetes de trabajo desglosada 107

Tablas de acrónimos

Acrónimos generales

Acrónimo	Significado
API	***A**pplication **P**rogramming **I**nterface*
CD	***C**ontinuous **D**elivery*
CI	***C**ontinuous **I**ntegration*
CRUD	***C**reate, **R**etrieve **U**pdate and **D**elete*
CSV	***C**omma **S**eparated **V**alue*
DNI	***D**ocumento **N**acional de **I**dentidad*
EIB	***E**scuela de **I**ngeniería de **B**ilbao*
HTTP	***H**yperText **T**ransfer **P**rotocol*
IVA	***I**mpuesto sobre el **V**alor **A**ñadido*
JDBC	***J**ava **D**ata**B**ase **C**onnectivity*
JSON	***J**ava**S**cript **O**bject **N**otation*
LMS	***L**earning **M**anagement **S**ystem*
MVC	***M**odel **V**iew **C**ontroller*
OAuth	***O**pen **A**uthorization*
REST	***RE**presentational **S**tate **T**ransfer*
UPV/EHU	***U**niversidad del **P**aís **V**asco/**E**uskal **H**erriko **U**nibertsitatea*
URL	***U**niform **R**esource **L**ocator*
VCS	***V**ersion **C**ontrol **S**ystem*

Tabla 1. Acrónimos generales

Acrónimos propios

Acrónimo	Significado
IJ	***I**ngeniero **J**unior*
IS	***I**ngeniero **S**énior*
MDa	***M**ódulo **D**atos*
MDo	***M**ódulo **D**ocencia*
MEv	***M**ódulo **E**valuador*
MGe	***M**ódulo **G**estor*
MRe	***M**ódulo **R**epositorio*
NME-CV	***N**ueva **M**etodología de **E**nseñanza de programación basada en tecnologías de **C**ontrol de **V**ersiones*
SC	***S**ubsistema **C**ódigo*
SD	***S**ubsistema **D**ocumentación*
SI	***S**ubsistema **I**ntegración*
SU	***SU**pervisor*
TE	***TÉ**cnico*
UsuAd	***U**suario **A**dministrador*
UsuDo	***U**suario **D**ocente*
UsuEs	***U**suario **E**studiante*
UsuMa	***U**suario **M**antenimiento*

Tabla 2. Acrónimos propios

1. Introducción y contexto

Tras la instauración del Plan Bolonia en el año 2007, el profesorado de la UPV/EHU (*Universidad del País Vasco/Euskal Herriko Unibertsitatea*), al igual que en otras muchas universidades, se vio obligado a añadir una serie de actividades (prácticas, ejercicios…) con las que realizar una evaluación continua sobre el trabajo de sus estudiantes. Debido a esto, se aumentó la carga de trabajo docente en cuanto a la corrección de actividades se refiere. Esto se traduce en que, en las asignaturas de programación, cada estudiante (o grupo de estudiantes) debe desarrollar códigos que luego son evaluados por el cuerpo docente. Para una correcta evaluación, no basta con leer el código, sino que también es preciso probar que se compila y se ejecuta correctamente.

En el contexto la EIB (*Escuela de Ingenieros de Bilbao*), perteneciente a dicha universidad, esta problemática se ha tratado de resolver con el desarrollo del programa Monitor, utilizado en la asignatura de Arquitectura de Sistemas de Información perteneciente al tercer curso del grado de Ingeniería en Tecnología de Telecomunicación, mediante el cual se evalúan automáticamente las actividades que se desarrollan durante el curso (incluyendo exámenes). Además, esta herramienta es utilizable no sólo dentro de la EIB, sino que sus estudiantes pueden hacer uso de ellas desde sus casas, permitiéndoles trabajar en la resolución de las actividades en cualquier momento. Pese a que la herramienta es útil en dicho contexto, presenta la carencia de ser poco exportable a otras asignaturas.

Los problemas de evaluación no sólo suceden en las universidades, sino que cualquier centro educativo que disponga de una evaluación continua en sus asignaturas de programación se encuentra ante las mismas problemáticas. Incluso, saliendo de la educación reglada, los cursos impartidos de forma online, cada vez más presentes, están en la misma situación, con el agravante de que una misma persona puede tener que encarnar la figura del docente ante cientos de estudiantes, haciendo directamente imposible la tarea de evaluación continua.

En el caso concreto de las asignaturas de programación, la corrección de actividades (y, con ello, el seguimiento de la evaluación continua) puede resolverse mediante el uso de herramientas y tecnologías que se utilizan en entornos profesionales, como son los VCS (*Version Control System*), las forjas, y las aplicaciones de CI (*Continuous Integration*)[1]. Combinando estas tecnologías con los LMS (*Learning Management System*), se ideó la Nueva Metodología de Enseñanza de programación basada en tecnologías de Control de Versiones, acortado como sistema NME-CV.

Como primer paso, en 2018, se realizó un estudio técnico con el que se estableció la metodología, definiendo las tecnologías que la harían posible y proponiendo una arquitectura de referencia sobre la que construir, a futuro, una implementación del sistema. Siendo esta implementación el objetivo final de todo el proyecto relacionado con el sistema NME-CV, no es suficiente con realizar el estudio técnico, sino que es necesario dar más pasos intermedios antes de llegar a él.

[1] Estas herramientas y plataformas serán explicadas más en detalle en anotaciones posteriores de apartados más técnicos, pues se considera que su explicación no corresponde a este tipo de apartados.

Por tanto, surge la necesidad de dar el siguiente paso y realizar un análisis técnico de viabilidad con el que ahondar aún más en los resultados del estudio, obteniendo con ello una imagen más clara de la forma exacta en la que implementar las tecnologías y plataformas previamente estudiadas.

2. Objetivos y alcance

El objetivo principal de este trabajo es la realización de un **análisis técnico de viabilidad y la propuesta de diseño de una arquitectura software que permita identificar una solución técnica completa para la implantación del sistema NME-CV** y que en sí mismas forman una parte fundamental en el conocimiento que el alumnado debe adquirir para el ejercicio de su profesión en el ámbito del desarrollo software.

En la búsqueda de este objetivo principal, se plantea el desarrollo de algunos objetivos secundarios cuya realización ayudarán a enfocar el trabajo y conseguir resultados relevantes de cara a validar la implantación final de la propuesta realizada. Como objetivos secundarios, se definen los siguientes:

1. **Realización de prototipos demostradores**. Para poder analizar la viabilidad de la solución y poder detallar mejor la arquitectura propuesta, se han realizado prototipos de funcionalidades concretas que demuestran que es posible implementar dichas funciones. De esta manera, también se consigue tener una idea más clara del esfuerzo que supone implementar una solución u otra, lo que se traduce en una mejor toma de decisión a la hora de escoger entre diversas alternativas.
2. **Desarrollo de una maqueta de despliegue del servicio**. Se presentará en el diseño una arquitectura completa que permite identificar los subsistemas que la componen, su distribución en la infraestructura de la red y la asignación de servicios y funcionalidades necesarias. Esto permitirá identificar qué servicios se deben desplegar y de qué manera se instalarían en las máquinas correspondientes.
3. **Validación de la solución propuesta en el contexto de la EIB**. En el trabajo se considera la aplicabilidad de las soluciones propuestas al entorno de formación de las asignaturas prácticas de programación que se imparten en la EIB. No quiere decir que sólo deba ser aplicable a ese contexto. Debería ser utilizable en el mayor número de escenarios posible, pero, al menos, es obligatorio que pueda utilizarse en ese.

Una vez completado el análisis de viabilidad y el diseño, el siguiente paso sería implementar el sistema en un entorno controlado, usándose en una o dos asignaturas para así validar definitivamente el diseño y pulir detalles. Por tanto, al final de este trabajo, debe llegarse al punto en el que el diseño esté listo para planear dicho paso.

3. Beneficios

A continuación, se detallan los beneficios que supone el desarrollo de este trabajo. Para una mejor organización, los beneficios se dividen en tres grupos: técnicos, económicos y sociales.

3.1. Beneficios técnicos

En este subapartado se van a destacar los beneficios de carácter técnico que se entiendan como más relevantes fruto de la realización de este trabajo.

B.TECN.1 Disponibilidad de prototipos

Al desarrollar los prototipos demostradores que ayuden a la toma de decisiones, también se están dando los primeros pasos hacia la consecución de los desarrollos que sean necesarios para implementar la solución. De esta forma, ya se está adelantando trabajo que hay que realizar a posteriori.

B.TECN.2 Disponibilidad del diseño detallado

Al disponer del diseño detallado de la solución del proyecto NME-CV, es posible empezar con los desarrollos y la planificación necesaria para poder implementar las primeras pruebas piloto que validen definitivamente la solución. Así, se siguen dando pasos hacia la implementación final.

B.TECN.3 Desarrollo de aplicaciones de integración

Para poder llegar a la solución, es necesario realizar ciertos desarrollos de integración, ya sea para integrar plataformas como tal o para integrarlas en un entorno académico. Si bien estos códigos están pensados para ser usados en el sistema NME-CV, algunas de sus partes (o prototipos descartados) podrían ser reutilizados en otros escenarios, de forma que sirvan de primer paso para nuevos proyectos.

3.2. Económicos

Los trabajos realizados permiten obtener beneficios de carácter económico que se destacan en este subapartado.

B.ECON.1 Mejor ajuste presupuestario

El realizar prototipos en este trabajo supone hacerse una mejor idea de los recursos que son necesarios consumir para poder implementar el sistema. Así, dando este paso intermedio, se puede realizar un mejor ajuste de este consumo de recursos, reduciendo costes en etapas posteriores.

B.ECON.2 Mayores matriculaciones

Al mejorar el proceso educativo mediante este sistema, se mejora también la imagen del centro frente a futuros estudiantes. De esta manera, se consigue un mayor número de matriculaciones, con la consecuente ganancia económica.

B.ECON.3 Explotación de los desarrollos

Como se ha mencionado anteriormente, es necesario realizar ciertos desarrollos para poder dar solución al sistema. Tanto la solución en general como estos desarrollos pueden ser explotados comercialmente al implementarlo en otros centros educativos, ofreciendo servicios de soporte y mantenimiento con los que generar ingresos recurrentes.

3.3. Socioculturales

Desde el punto de vista social se pueden destacar también importantes beneficios aprovechando que esta arquitectura da soporte al desarrollo de una nueva metodología de formación en un área de tanto potencial como es la programación software.

B.SOCI.1 Mejora del proceso educativo

Gracias a este trabajo, se dan avances en la implementación de un sistema que consigue mejorar el proceso educativo al dotar de nuevas herramientas a la evaluación continua de prácticas de programación. Sumado al uso de tecnologías utilizadas en entornos profesionales, el alumnado recibe una formación más completa que le permitirá desarrollar mejores programas y hacer un mejor uso de estas herramientas.

B.SOCI.2 Mejor gestión del tiempo

Al usarse procesos automatizados, el equipo docente ve reducida su cargar de trabajo en lo que a correcciones se refiere, de forma que puedan dedicar este tiempo al resto de sus labores, mejorándolas. Además, el alumnado también se ve beneficiado al reducirse el tiempo que necesita para conocer los fallos en su código y mejorarlo, potenciando su aprendizaje.

B.SOCI.3 Solución utilizable desde múltiples lugares

El sistema NME-CV está pensado para utilizarse desde cualquier punto, de forma que cada estudiante pueda seguir usándolo desde su casa, permitiendo el trabajo telemático. Aunque esto se planteó en un principio para ajustarse a los requerimientos del Plan Bolonia, también puede ser utilizado en situaciones de clases telemáticas como las que han tenido que realizarse como consecuencia de la pandemia de COVID-19, mejorando la educación en dichos escenarios.

4. Especificaciones generales

En este apartado, se definen cuáles son las especificaciones con las que se encara el desarrollo del trabajo. En este caso, no se habla de especificaciones que se relacionen con funcionalidades concretas de los módulos del sistema (éstas se comentarán más adelante), sino que se pone el foco en aquellas de carácter más general, haciendo referencia al sistema completo más que a una de sus partes en concreto.

Las especificaciones generales se muestran agrupadas en los siguientes subapartados, según el aspecto al que hagan referencia, utilizando un código para cada una de ellas

A modo de resumen, se añade la siguiente tabla en la que se agrupan todas las especificaciones generales junto con el esquema de formación de códigos.

Especificaciones Generales (**G**)	Procesos del Sistema (**PROC**)	Automatización de los Procedimientos (**1**)
	Arquitectura (**ARQT**)	Arquitectura Adaptable a Varias Soluciones (**1**)
		Alta Independencia entre Módulos (**2**)
		Menor Despliegue Posible (**3**)
	Apartado de Docencia (**DOCN**)	Eficiencia del Trabajo del Grupo Docente (**1**)
	Seguridad del Sistema (**SECU**)	Privacidad de la Información (**1**)
		Protección Frente a Plagios (**2**)

Tabla 3. Resumen de especificaciones generales

4.1. Especificaciones generales de los procesos del sistema

El primer grupo de especificaciones generales a describir es el que hace referencia a cómo deben ser los procesos en el sistema. En concreto, se ha definido una única especificación general.

G.PROC.1 Automatización de los procedimientos

Para que el sistema tenga sentido y pueda satisfacer las necesidades para las que ha sido creado, es capital que automatice la mayor parte de sus procedimientos. En concreto, se pone el foco en el proceso de evaluación de las actividades, tanto en su cálculo como en su registro.

Se quiere insistir en que esto no quiere decir que sea el único procedimiento a automatizar. En general, todos los procesos que puedan se puedan realizar de manera automática deben ser automatizados.

4.2. Especificaciones generales de la arquitectura

En lo que arquitectura se refiere, la solución debe cumplir con las especificaciones generales descritas a continuación.

G.ARQT.1 Arquitectura adaptable a varias soluciones

Aunque en este documento se aporta una solución concreta, utilizando ciertas plataformas en detrimento de otras, la arquitectura sobre la que se sostiene la solución debe poder adaptarse al uso de otras plataformas. Por ello, en medida de lo posible, debe diseñarse pensando en las funciones que deben cumplir los módulos y todos los componentes que los conforman en lugar de cómo encajan las plataformas en concreto.

G.ARQT.2 Alta independencia entre módulos

Para que puedan desarrollarse nuevas versiones de la arquitectura, los módulos que la conforman deben diseñarse de forma que sean lo más independientes posible entre sí. De esta forma, si en un futuro fuera necesario rediseñar internamente un módulo, esto pueda hacerse de la forma más sencilla posible. Para poder llegar a ello, las interfaces entre los módulos deben estar bien definidas, teniendo claro cómo se comunican entre ellos.

Además, cuando llegue el momento de desarrollarlos, debe tenerse en cuenta que todos los módulos que requieran de diseños internos deben realizar este proceso teniendo en cuenta estos mismos principios de independencia, de forma que se puedan alterar de la forma más sencilla posible.

G.ARQT.3 Menor despliegue posible

Como se ha comentado anteriormente, en este trabajo se da un paso adicional para llegar al objetivo final que es la implementación del sistema. Por tanto, a la hora de diseñar la solución, no debe perderse de vista cómo influye la selección de la solución en el despliegue del sistema, valorando el impacto que tiene cada decisión en este proceso en aras de simplificarlo, abaratando costes.

4.3. Especificaciones generales sobre apartado de docencia

En este subapartado, se describen cuáles son las especificaciones generales que se relacionan directamente con los aspectos de docencia. En concreto, se han definido una especificación general.

G.DOCN.1 Eficiencia del trabajo del grupo docente

El origen de este proyecto es el de simplificar la labor de corrección de actividades relacionas con la evaluación continua durante la impartición de una asignatura de programación, reduciendo el tiempo que debe dedicar el profesorado a estas tareas. Por tanto, no tendría sentido diseñar una solución en la que el cuerpo docente de la asignatura requiriera de realizar labores demasiado complejas o tediosas. Así pues, se debe premiar que el sistema no les ponga trabas, permitiendo que trabajen de forma más eficiente.

4.4. Especificaciones generales sobre la seguridad del sistema

Por último, se encuentran las especificaciones generales relativas a la seguridad que debe implementar el sistema NME-CV para considerar que funciona correctamente.

G.SEGU.1 Privacidad de la información

Al querer ser usado en un entorno educativo, en este sistema se está almacenando información considerada sensible, como es la relación entre una persona y su DNI o la calificación que dicha persona ha obtenido durante la resolución de una actividad. Por tanto, en general, el sistema debe asegurar que cada información sólo llega a las personas autorizadas a consultarla.

G.SEGU.2 Protección frente a plagios

Uno de los aspectos clave del uso de forjas es la facilidad que presenta a la hora de compartir el código desarrollado por un usuario con el resto del mundo. En un entorno educativo, esto se considera una debilidad, pues abre la puerta a que algunos estudiantes pueda plagiar las actividades realizadas por otras personas. Por tanto, el sistema debe introducir medidas que, al menos, no introduzcan vulnerabilidades frente a los plagios.

5. Análisis de viabilidad técnica

A continuación, se va a discutir sobre la viabilidad técnica del proyecto, presentando una serie de problemáticas encontradas a la hora de plantear el diseño de la solución y planteando cómo resolver las mismas. Se va a llegar a estar resoluciones al escoger una de las alternativas que se van a plantear para cada problemática, utilizando para ellos una serie de criterios de selección.

Cabe destacar que el proceso para llegar a la solución no se ha limitado al plano teórico, sino que, como se ha comentado anteriormente, se han realizado pruebas a través de pequeños prototipos, lo que ha contribuido no sólo a demostrar la posibilidad de desarrollar dicha solución, sino que también ha dado una idea del esfuerzo que requiere implantarla.

5.1. Problemáticas

A continuación, se enuncian las problemáticas identificadas para el diseño de la solución. En ellas, se van a enunciar las especificaciones funcionales que deben cumplir sus soluciones para ser consideradas como tal.

Las problemáticas que se presentan son las siguientes:

- Gestión de repositorios
- Despliegue del contexto de pruebas
- Gestión del entorno académico
- Interfaz de gestión de los usuarios

5.1.1. Problemática I: Gestión de repositorios

Al buscar una solución basada en tecnologías de control de versiones, los alumnos deben utilizar repositorios en los que alojar los códigos para la resolución de actividades. A su vez, para poder hacer el seguimiento de sus avances, esos repositorios tienen que ser accesibles por el profesorado responsable de manera sencilla, por lo que se necesita una plataforma que albergue todos los repositorios de los alumnos, esto es, se necesita una forja[2].

En el mercado existen varias forjas que podrían cumplir estos requisitos de forma sencilla. Sin embargo, además de lo básico, la forja que se vaya a utilizar debe cumplir una serie de especificaciones funcionales.

Para empezar, para poder dotar de automatización a la metodología, es necesario que esta plataforma pueda detectar y notificar los eventos que ocurran en los repositorios, enviando la información suficiente para conocer, como mínimo, qué evento ha ocurrido y en qué repositorio, además de toda la información posible al respecto. **Esto constituye la especificación funcional F.FORJ.EVE Detección de eventos**.

[2] Plataforma web en la que distintas personas suben códigos desarrollados por ellos mismos, de forma que puedan compartirlo con usuarios finales o con otros desarrolladores que colaboren en el proyecto. Además, estas plataformas utilizan software VCS para poder gestionar las distintas versiones del código alojado.

En relación con esto último, no basta con únicamente detectar los eventos y recopilar su información, sino que también es necesario poder ejecutar acciones contra los repositorios de manera remota y automatizada, por lo que la plataforma debe disponer de alguna interfaz que permita introducir cambios en los repositorios de manera programática. **Esto constituye la especificación funcional F.FORJ.COM Interfaz de comunicación**.

Otro de los pilares del sistema es la posibilidad de realizar pruebas automatizadas a los códigos que suben los alumnos para resolver las actividades, utilizando para ello una aplicación de CI. Por tanto, es necesario que la forja escogida sea compatible con una de estas aplicaciones, ya sea utilizando una de terceros o alguna que esté integrada en la propia plataforma. **Esto constituye la especificación funcional F.FORJ.ICO Compatibilidad con aplicaciones de CI**.

Por último, no hay que perder el foco de que este sistema debe utilizarse en un entorno académico, con lo que ello conlleva. Por ejemplo, debe poner facilidades para poder crear un repositorio con el mismo contenido a un conjunto de alumnos, independientemente del número de alumnos que lo formen. **Esto constituye la especificación funcional F.FORJ.EDU Adecuación al entorno académico**.

A modo de resumen, se aporta la siguiente tabla con las especificaciones funcionales que debe cumplir.

Especificaciones Funcionales (**F**)	Forja (**FORJ**)	Detección de Eventos (**EVE**)
		Interfaz de Comunicación (**COM**)
		Compatibilidad con Aplicaciones de CI (**ICO**)
		Adecuación al Entrono Académico (**EDU**)

Tabla 4. Resumen de especificaciones funcionales de la Problemática I

5.1.2. Problemática II: Despliegue del contexto de pruebas

Otro aspecto para tener en cuenta a la hora de diseñar el sistema es que, para poder utilizar correctamente una aplicación de CI para automatizar las pruebas al código desarrollado por el grupo de estudiantes, es necesario utilizar un contexto en el que éstas se lleven a cabo. Para ello, éste debe estar alojado en una máquina que tenga acceso a la forja, pues, antes de iniciar las pruebas, debe poder descargar el repositorio que va a evaluar, además de la batería de pruebas a realizar. No es necesario que sea accesible por otros módulos del sistema. Esta problemática pone el foco en la forma que van a tomar los contextos, esto es, con qué tecnologías se va a desplegar.

Para poder completarlo, ese despliegue debe cumplir varias especificaciones funcionales, siendo una de ellas que esos contextos deben ser también compatibles con las herramientas necesarias para la realización de pruebas unitarias que demuestren el funcionamiento del código a evaluar. Más allá de estas, también deberían ser compatibles con herramientas que permitan la realización de otros tipos de pruebas. Un ejemplo de este tipo de pruebas son las pruebas para detectar plagios, con las que se

buscaría perseguir a cualquier estudiante que haya podido plagiar su trabajo con el de otro. **Esto constituye la especificación funcional F.EVAL.PRU Compatibilidad con múltiples tipos de pruebas**.

Otra de las especificaciones funcionales es que, sea cual sea la forma en la que se despliegue, tiene que asegurar que es compatible con las necesidades de cualquier asignatura. Por ejemplo, una asignatura en la que se programe en lenguaje C debe tener, obligatoriamente, un programa que sea capaz de compilar dicho código (por ejemplo, gcc), mientras que una en la que se utilice el lenguaje Java debe tener instalado el intérprete del lenguaje. La solución que se escoja debe ser consciente tanto de la existencia de estos escenarios como de las necesidades que tienen, pudiendo servir a su función de probar el código. **Esto constituye la especificación funcional F.EVAL.ASI Compatibilidad con múltiples asignaturas**.

A modo de resumen, se aporta la siguiente tabla con las especificaciones funcionales que debe cumplir.

Especificaciones Funcionales (**F**)	Evaluador (**EVAL**)	Compatibilidad con Múltiples Tipos de Pruebas (**PRU**)
		Compatibilidad con Múltiples Asignaturas (**ASI**)

Tabla 5. Resumen de especificaciones funcionales de la Problemática II

5.1.3. Problemática III: Gestión del entorno académico

Para que este sistema sea adecuado a un entorno académico, es necesario que se almacene información de interés en dicho entorno. Por ejemplo, es necesario que se guarde información sobre qué estudiantes hacen uso del sistema, en qué grupos se encuentran en qué asignatura o el resultado obtenido tras la resolución de una actividad. Esto supone, por tanto, la necesidad de disponer de algún modelo de datos capaz de recoger toda esta información.

Uno de los pilares de este modelo de datos debe ser la figura del estudiante, que debe quedar correctamente registrada en el modelo de datos. Este perfil debe poder identificar unívocamente al estudiante y poder relacionarlo con el resto de los elementos del sistema. Además, también el modelo de datos debe ubicarlo en los grupos y/o asignaturas correctas, de forma que el cuerpo docente tenga claro quiénes son sus estudiantes. **Esto constituye la especificación funcional F.DATO.EST Registro del perfil de estudiantes**.

Otro de los puntos importantes en cuanto a la gestión del entorno académico es el registrar la evaluación que han obtenido cada estudiante al resolver las actividades propuestas por el equipo docente. Hay que tener en cuenta que esto no se refiere a la evaluación final, sino que se habla de las evaluaciones de cada actividad, por lo que, junto a ello, es necesario almacenar los pesos de cada actividad en la evaluación final, de forma que ésta se pueda calcular una vez finalice la asignatura. **Esto constituye la especificación funcional F.DATO.EVA Registro de resultados de evaluación**.

Además, debe tenerse en cuenta que la información que se almacena en el modelo de datos es información sensible, ya que contiene información personal de estudiantes, sus evaluaciones o, incluso, puede llegar a contener usuarios y contraseñas. Por tanto, los datos almacenados deben estar correctamente securizados, de forma que se eviten filtraciones de esta información sensible. **Esto constituye la especificación funcional F.DATO.SEC Seguridad de los datos almacenados**.

A modo de resumen, se aporta la siguiente tabla con las especificaciones funcionales que debe cumplir.

Especificaciones Funcionales (**F**)	Modelo de Datos (**DATO**)	Registro del Perfil de Estudiantes (**EST**)
		Registro de Resultados de Evaluación (**EVA**)
		Seguridad de los Datos Almacenados (**SEC**)

Tabla 6. Resumen de especificaciones funcionales de la Problemática III

5.1.4. Problemática IV: Interfaz de gestión de los usuarios

Por último, se plantea una problemática que, en cierto modo, se relaciona con la anterior. En la tercera problemática, se exponía la necesidad de almacenar información relevante para el entorno académico. En este punto, lo que se plantea es cómo debe ser la interfaz que consulte y actualice dicha información para poder mostrársela a los usuarios.

Para poder adecuarse al sistema, esta interfaz debe ser capaz de mostrar la lista de estudiantes asignados a cargo de cada docente. Esta lista debe contener la información suficiente para poder identificar a cada estudiante. Cabe destacar también que ésta debe mostrar la información filtrada, facilitando la labor docente. **Esto constituye la especificación funcional F.DOC.LIS Lista de estudiantes**.

Otro de los puntos importantes que debe cumplir esta interfaz de consulta es que debe poder comunicar a cualquier estudiante la evaluación que ha obtenido al resolver las actividades propuestas por el equipo docente, así como el resultado de sus intentos de resolverla. Esta información sólo debería poder ser accedida por la persona evaluada y por el equipo docente responsable, de forma que se asegure que nadie ajeno a la evaluación tenga acceso a ella. Además de la calificación numérica, también sería interesante que cada estudiante recibiera un mensaje con el resultado de cada prueba, de forma que vea de manera sencilla el resultado de sus intentos. **Esto constituye la especificación funcional F.DOC.EVL Anuncio de resultados de evaluación**.

Por último, la interfaz de consulta debe disponer de un acceso rápido hacia los repositorios de cada estudiante. El motivo de este requerimiento es que, si se debe tratar con grupos grandes, el profesorado vería muchos repositorios en su menú de la forja, pudiendo identificar los repositorios fijándose en el nombre. Si hubiera un enlace que relacionara a cada estudiante con sus repositorios, podría hacer su trabajo de forma más eficiente. **Esto constituye la especificación funcional F.DOC.REE Acceso rápido a repositorios de estudiantes**.

A modo de resumen, se aporta la siguiente tabla con las especificaciones funcionales que debe cumplir.

Especificaciones Funcionales (**F**)	Aspectos Docentes (**DOCE**)	Lista de Estudiantes (**LIS**)
		Anuncio de Resultados de Evaluación (**EVL**)
		Acceso Rápido a Repositorios de Estudiantes (**REE**)
		Registro de Información (**REG**)

Tabla 7. Resumen de especificaciones funcionales de la Problemática IV

5.2. Alternativas

A continuación, se plantean las alternativas que pueden dar solución a las problemáticas planteadas, teniendo en cuenta que éstas deben cumplir con las especificaciones funcionales definidas.

5.2.1. Alternativas a la Problemática I: Gestión de los repositorios

Para resolver esta problemática, se plantea escoger entre dos forjas disponibles en Internet: GitLab[3] y GitHub[4]. Se trata de dos forjas de características muy similares y que utilizan tecnologías parecidas, aunque, como se verá más adelante, difieren en uno de los aspectos clave de la problemática.

[3] Puede encontrarse en la URL https://gitlab.com.
[4] Puede encontrarse en la URL https://github.com.

A.I.1 GitLab

La primera de las alternativas a estudiar es GitLab.

Ésta dispone de tecnología de webhooks[5]. Tal y como está implementada, permite configurar los repositorios de tal manera que, ante la ocurrencia de ciertos eventos, se envíe un mensaje HTTP POST a una URL previamente configurada. Este mensaje contiene un JSON con información relativa no sólo al evento en sí, sino también al repositorio en el que ocurrió, incluyendo toda la información del usuario que ha provocado el evento, el nombre del repositorio y su identificador único entre otros. Esto supone el cumplimiento de la especificación funcional **F.FORJ.EVE**.

En esta forja existe también una API REST mediante la cual un usuario con los permisos adecuados puede realizar acciones de manera programática sobre sus repositorios. Ejemplos de estas acciones son la creación y eliminación de repositorios, la configuración de webhooks en los mismos o la adición y eliminación de usuarios del repositorio (incluyendo la posibilidad de asignar roles). Además, al tratarse de REST, sólo se necesita de acceso al repositorio a través de HTTP, lo que permite la comunicación desde cualquier equipo conectado a Internet. Utilizando esta API es posible, por tanto, modificar el repositorio cualquier estudiante de cara a poder automatizar procesos, por lo que se cumple la especificación funcional **F.FORJ.COM**.

Para poder cumplir con la especificación funcional **F.FORJ.ICO**, GitLab dispone de una aplicación de CI integrada en la propia plataforma. Ésta utiliza la aplicación GitLab CI/CD, que se basa en la codificación de pruebas en un fichero YAML denominado .gitlab-ci.yml, ubicado en la raíz del repositorio. La codificación de las pruebas se realiza utilizando unas directivas especiales definidas por GitLab.

En cuanto al contexto en el que se realizan las pruebas, éstas se ejecutan en unos denominados runners. Éstos están provistos por GitLab y disponen de varios “ejecutores”, que no son otra cosa que la forma que toman los runners, esto es, si son contenedores Docker, entornos de Linux, de Windows, etc. Cabe destacar que GitLab provee de la herramienta GitLab Runner, un software para poder desplegar runners propios y no depender de GitLab.

Por último, para poder cumplir con la especificación **F.FORJ.EDU**, GitLab permite instalar instancias propias ajenas a la alojada en la URL https://gitlab.com. Esto significa que el centro educativo podría alojar una de estas instancias en una de sus máquinas, lo que supondría mantener un control absoluto sobre la plataforma. Esto es interesante de cara a poder a adaptar la configuración de la web y la gestión de los usuarios. Se podría, por ejemplo, limitar las opciones de registro, sería más fácil encontrar usuarios y repositorios al haber menos volumen de ambos, etc. El gran problema de esto es que sería necesario realizar un mantenimiento exhaustivo de la

[5] Callbacks dentro de una página o aplicación web que se resuelven mediante HTTP. Esto quiere decir que, en dicha página o aplicación web, existe un componente software capaz de detectar la ocurrencia de un evento en el servicio y procesarlo (a través de un callbacks) enviando, a una dirección previamente configurada, un mensaje HTTP POST cuyo cuerpo contiene información relativa al evento.

plataforma, y se necesitaría de una máquina que albergara la web, además del dominio. Incluso, para poder crear los contextos para GitLab CI/CD (no habría runners ya configurados) sería necesario instalar la aplicación GitLab Runner en una máquina diferente a la que albergue la instancia de GitLab, ya que, según la documentación oficial, el no hacerlo traería consigo problemas de rendimiento. Otro de los aspectos positivos de disponer de una implementación propia es que no existe ningún límite de uso de CI, como es el caso de otras forjas.

Cabe destacar que utilizar la instancia oficial de GitLab no es una opción, ya que carece de un sistema capaz de organizar a los alumnos en grupos mientras se mantiene la privacidad de los repositorios. También carece de una herramienta que permita crear repositorios de estudiantes de forma automatizada, lo que obligaría a hacer pequeños desarrollos para dotarle de dicha funcionalidad y que cumpla definitivamente con esta especificación.

A.I.2 GitHub

La otra alternativa que se plantea es GitHub.

GitHub es bastante parecida a GitLab y dispone también de tecnología de webhooks y una API REST para modificar los repositorios desde el exterior, por lo que también cumple con las especificaciones funcionales **F.FORJ.EVE** y **F.FORJ.COM**.

Igualmente, también dispone de una aplicación de CI integrada en la propia forja. Se trata de GitHub Actions, que tiene un funcionamiento muy similar al de GitLab CI/CD. En este caso, lo que se ejecuta son todos los ficheros YAML ubicados en el directorio .github/actions (ubicado a su vez en la raíz del repositorio). Esta vez, no existe el concepto de runners como tal, aunque las pruebas se ejecutan en unas máquinas provistas por GitHub en unos contextos personalizables, que bien pueden ser también contenedores Docker o entornos Linux.

La gran diferencia entre ambas aplicaciones radica en que GitHub Actions abarca más allá de los procesos de CI/CD, pudiendo ejecutar todo tipo de acciones como respuesta a múltiples eventos del sistema. Sin embargo, en base al resto de necesidades del sistema, se prevé utilizar GitHub Actions únicamente como aplicación de CI, por lo que se consideran iguales. Se considera por tanto que también cumple con la especificación funcional **F.FORJ.ICO**.

Vistas todas las similitudes entre ambos, se pasa a comentar la gran diferencia que presentan, que se encuentra a la hora de resolver la especificación **F.FORJ.EDU**, y es que, para poder integrar las plataformas en el entorno académico, deben tomarse rumbos diferentes según la plataforma que se decida a usar.

Para empezar, GitHub no permite la instalación de una instancia separada a diferencia de lo que ocurría en GitLab, por lo que se pierde el control total sobre la forja. Sin embargo, sí que dispone de un sistema de organizaciones, que no es otra cosa que un conjunto de cuentas agrupadas, de tal forma que una persona de esa organización puede guardar repositorios tanto públicos como privados dentro de la organización. Así, se podrían mantener la lista de estudiantes y sus

repositorios para las actividades propuestas por el centro educativo en un mismo lugar. En lo negativo, al no poder instalar una instancia propia, no es posible saltarse el límite de 2000 minutos de CI al mes por cada organización que establece GitHub en su plan gratuito, por lo que habría que contratar alguno de los planes de pago. Aun así, cabe destacar que existen algunas rebajas y planes pensados para entornos educativos.

Este, sin embargo, no es el único punto para destacar de GitHub. Desde hace unos años, GitHub ha apostado por meterse en los entornos educativos, a través de la herramienta GitHub Classroom. Ésta permite definir una clase (*classroom*) como una subdivisión de una organización y asignarle tareas (*assignment*). Estas tareas no son otra cosa que un fork[6] de un repositorio base, dando a cada estudiante (o grupo de estudiantes) repositorios en los que trabajar sus actividades. También dispone de una interfaz gráfica que permite encontrar fácil los alumnos inscritos en una classroom y enlaces a sus repositorios.

5.2.2. Alternativas a la Problemática II: Despliegue del contexto de pruebas

Para esta segunda problemática, se plantean dos soluciones diferentes: utilizar un contexto en el que se instalen todas las herramientas para poder dar servicio a las asignaturas o desplegar contenedores específicos para cada caso.

A.II.1 Contexto completo

La primera de las alternativas consiste en desplegar un contexto único que contenga todas y cada una de las herramientas necesarias para la evaluación del código de cualquier asignatura que vaya a utilizar el sistema NME-CV. Esto supone disponer de una solución ad-hoc instalada en una única máquina física.

Al ser una solución ad-hoc y configurada desde cero, se asegura que se van a poder instalar todas las herramientas que se requieran en las diferentes asignaturas, sea para resolver las pruebas unitarias u otro tipo de pruebas. Por tanto, se cumplen ambas especificaciones funcionales (**F.EVAL.PRU** y **F.EVAL.ASI**), por lo que se considera a esta solución como una válida para la problemática.

Sin embargo, se trata de una solución que presenta una gran desventaja. Según el funcionamiento de las dos aplicaciones de CI descritas en la Problemática I, esto supondría que la adquisición, instalación, gestión y mantenimiento de la máquina, y por ende del contexto, recaería totalmente en el centro, lo que incrementaría considerablemente los costes de la implementación del proyecto. Estos costes vendrían a cambio de disponer del control absoluto sobre dicho contexto y simplificar la labor del equipo docente, pues no se debería encargar de ninguna de estas tareas.

[6] En GitHub (al igual que en otras forjas), se denomina fork a un proceso que consiste en clonar el contenido de otro repositorio en uno nuevo, aunque el repositorio pertenezca a otro usuario. La idea tras esta función es que una persona pueda partir del trabajo de otro usuario para realizar su propia versión del código (por ejemplo, para añadir una funcionalidad), pudiendo a posteriori integrarse dentro del repositorio original. De esta forma, se favorece la colaboración entre usuarios.

A.II.2 Separación en contenedores

La otra alternativa consiste en cambiar la máquina dedicada por el uso de contenedores. En esta alternativa, ya no hay un único contexto, sino que hay un conjunto de ellos que se ajustan a las necesidades puntuales bien sea de una asignatura al completo o de una actividad concreta dentro de ella.

En este caso, para poder satisfacer las especificaciones funcionales, debe ponerse el ojo en los ficheros Dockerfile que se definen dentro de los contenedores Docker. Los ficheros Dockerfile son aquellos en los que un usuario puede partir de la imagen creada por otro y alterarla de varias formas. El ejemplo a destacar entre estas alteraciones es la instalación de software adicional a la imagen base. Por tanto, sabiendo que ambas aplicaciones de CI lo permiten, se podrían generar imágenes Docker ajustadas a cada caso mediante los ficheros Dockerfile, por lo que se cumplen las especificaciones funcionales **F.EVAL.PRU** y **F.EVAL.ASI**.

Si bien es cierto que se pierde el control que se tenía en la anterior alternativa, se consigue eliminar los costes de adquisición, instalación, gestión y mantenimiento de disponer de una máquina dedicada, pues ambas aplicaciones de CI dan la posibilidad de gestionar los contenedores en la propia forja. Esto simplifica mucho el despliegue de los contextos de pruebas. Sin embargo, existe la gran desventaja de que, si el contexto debe ajustarse a cada caso, es el cuerpo docente el responsable de generar dichos contextos, aumentando así la complejidad de uso del sistema para ellos.

5.2.3. Alternativas a la Problemática III: Gestión de entorno académico

Para la resolución de esta problemática, se plantean dos soluciones en forma de alternativa. En concreto, la primera de ellas hace referencia a la realización de un desarrollo propio para todo el modelo de datos, mientras que la segunda plantea un modelo híbrido, en el que se aproveche un modelo de datos existente que se combine con el desarrollo de un modelo de datos más pequeño que integre el sistema NME-CV con ese modelo de datos escogido. En cualquier caso, se plantea que ese modelo de datos tome forma de base de datos relacional.

A.III.1 Desarrollo propio

La primera de las alternativas consiste en hacer un desarrollo propio del modelo de datos. Para ello, la base de datos a plantear debe cubrir todos los aspectos involucrados en el sistema.

Poniendo atención a las especificaciones funcionales que se han definido junto al problema, se aprecia que los dos pilares fundamentales del modelo de datos son el perfil de estudiante y la evaluación que obtienen por resolver sus actividades. Por tanto, hay que diseñar y desarrollar toda la estructura de tablas y relaciones para poder proveer de esta información.

Para el caso de la especificación **F.DATO.EST**, se deben incluir, además de las tablas de estudiante en sí, tablas de asignatura que puedan identificar en cuáles de las que utilizan este sistema se han matriculado, tablas de grupo que indiquen en qué grupo está cada estudiante en cada asignatura

o tablas de docentes para que se pueda conocer qué docente es responsable de un cierto grupo de estudiantes. Además de estos datos, también debería conectarse con tablas que almacenen la información que se ha obtenido durante el proceso de resolución de actividades, de forma que se pueda obtener su evaluación.

En el fondo, esto último significa que debe relacionarse con la parte de la estructura encargada de que se cumpla con la especificación funcional **F.DATO.EVA**. Para poder cumplirla, es necesario guardar toda la información de los avances de cada estudiante en la asignatura, esto es, la evaluación de cada una de las actividades, pudiendo después calcular y registrar la nota final de la parte evaluada mediante este sistema. Para poder guardar esta información, es necesario guardar antes la información de las actividades en sí de forma jerarquizada. Así, se podría plasmar el disponer, por un lado, de actividades de tipo asignatura que registraran la evaluación a nivel de asignatura y otras de más bajo nivel (como prácticas o ejercicios) que guardaran la evaluación de cada una de las partes junto a su peso en la evaluación final.

Para poder cumplir con la última de las especificaciones funcionales, la **F.DATO.SEC**, sería necesario implementar medidas de seguridad en la base de datos. Una de las medidas a tomar sería el control de acceso, de forma que sólo los usuarios autorizados puedan consultar o modificar datos.

Como puede apreciarse, el desarrollo de una solución ad-hoc como esta supone una gran ventaja en cuanto a integración, ya que se puede diseñar e implementar a medida del sistema de forma que se eviten procesos intermedios para integrar soluciones de terceros. Sin embargo, como gran desventaja, presenta la necesidad de realizar el desarrollo de un modelo de datos que abarca muchas tablas y conceptos, por lo que se trataría de un desarrollo largo, complejo y, con ello, costoso, más teniendo en cuenta las labores de mantenimiento a posteriori.

A.III.2 Modelo de datos híbrido

La otra alternativa que se propone es la de aprovechar algún sistema ya existente que pudiera hacer estas gestiones e integrarse con el resto de la arquitectura.

Para poder implementar esta solución, se pone el foco sobre los LMS[7], y en concreto sobre Moodle. Moodle tiene ya integrados recursos como disponer de una lista de estudiantes inscritos y poder proponerles actividades evaluables. En casos como el de la UPV/EHU, todos los estudiantes son dados de alta al principio del cuatrimestre por la administración de la universidad. Además, Moodle dispone de una API consumible mediante REST (entre otras tecnologías) desde la que se puede obtener la lista de estudiantes de un curso, así como escribir (y, con ello,

[7] Learning Management System. Se trata de aplicaciones software que permiten realizar una gestión integral de un curso educativo. En ellos, se crea un curso administrado por un profesoral que se registran unos alumnos. Para poder hacer esta gestión, cada LMS dispone de una serie de recursos que permiten replicar de forma online el funcionamiento de un aula. Por ejemplo, en el caso de Moodle, es posible crear cuestionarios que corrige y evalúa automáticamente, foros en los que resolver dudas de la asignatura, entregas que luego evalúa el profesor, etc.

almacenar) evaluaciones en las actividades. Por tanto, esta solución pasaría por reaprovechar el propio modelo de datos de la instancia de Moodle instalada en la universidad.

Pudiendo acceder a las listas de estudiantes, se cumpliría con la especificación funcional **F.DATO.EST**, pues las entradas ya estarían registradas y se podría acceder a las mismas. Por otro lado, esta API hace que se cumpla también con la especificación **F.DATO.EVA**, pues se podrían crear actividades dentro de Moodle que representen las actividades a resolver y escribir automáticamente su calificación en las mismas. Además, para resolver el problema de los pesos, se pueden aprovechar herramientas ya disponibles en Moodle.

Cabe destacar que, en este escenario, seguiría siendo necesario desarrollar una base de datos, sólo que, en esta ocasión, en lugar de necesitarse para hacer el seguimiento de la asignatura, se necesitaría como parte del proceso de integración (por ejemplo, para relacionar a cada estudiante con su cuenta en la forja). Por tanto, se trataría de una base de datos más sencilla que la planteada en la otra alternativa.

Por tanto, este caso supone una alternativa que, a costa de añadir complejidad al proceso de integración de plataformas por añadir una más, simplifica mucho el desarrollo y mantenimiento del modelo de datos, pues gran parte recae, en el caso de la EIB, en eGela[8] y en las labores de mantenimiento ya existentes.

5.2.4. Alternativas a la Problemática IV: Interfaz de gestión de los usuarios

En la resolución de la última problemática, se plantean también dos alternativas. Al igual que en la anterior, la primera consiste en realizar un desarrollo propio que se ajuste a las necesidades de esta problemática para así darle solución, mientras que la otra trata de buscar una solución con la que se puedan aprovechar soluciones de terceros ya existentes que, combinadas, puedan dar solución al problema, aunque sean necesarios pequeños desarrollos.

[8] Implementación de Moodle utilizada en la UPV/EHU.

A.IV.1 Interfaz propia

Como se ha introducido, en la primera alternativa se propone desarrollar una interfaz propia que permita agrupar todas las consultas en un único lugar, el cual debe ser accesible tanto por cualquier docente como por cualquier estudiante.

A la hora de diseñar esta interfaz, es necesario tener en cuenta que debe cumplir con las especificaciones funcionales que se han definido junto a la problemática. Esto se traduce en que, como mínimo, debe disponer de vistas que permitan mostrar las listas de estudiantes de cada docente, sus repositorios y que mostrara a cada estudiante su evaluación, cuidando que no se vista por otras personas. Esto supone también que, para poder acceder a la información, sea necesario desarrollar una vista de login, controlando el acceso.

Mostrar esas vistas trae consigo el diseño completo de las capas inferiores. Por tanto, es necesario desarrollar toda la lógica de negocio que soportaría las consultas a la base de datos y que rellenara la vista, mostrando la información pertinente. Junto a ello, también sería necesario desarrollar la forma en la que se va a comunicar esta lógica de negocio con la base de datos.

Juntando estas piezas, se cumplen las tres especificaciones funcionales (**F.DOCE.LIS**, **F.DOCE.EVL** y **F.DOCE.REE**) descritas.

En cuanto a ventajas y desventajas de esta alternativa, se destaca que presenta el inconveniente de que es necesario realizar el desarrollo completo, con el consumo de recursos que ello supone, tanto para el desarrollo en sí como para el posterior mantenimiento. A cambio, lo que se obtiene es una solución ajustada al sistema que no requiere de trabajo de integración adicional y que se ajusta exactamente a las necesidades. Además, al ser una interfaz única, se consigue la comodidad que supone disponer de un único punto en el que disponer de toda la información.

A.IV.2 Interfaz híbrida

En la segunda alternativa, se busca ahorrarse el desarrollo que supone la primera e integrar plataformas y herramientas de terceros como interfaz. En concreto, se plantea utilizar una combinación de Moodle, GitHub y GitHub Classroom.

Empezando con Moodle, la plataforma ya realiza la separación de asignaturas, permitiendo incluir a cada estudiante en aquellas en las que esté matriculado y mostrando en cada asignatura una lista completa de estudiantes. Ya con esto se cumple con la especificación **F.DOCE.LIS**. Además, como se ha mencionado en las alternativas de la anterior problemática, ya dispone de formas de representar y evaluar las actividades dentro de una asignatura. También es posible asignar unos pesos a estas actividades, de forma que luego se pueda calcular una evaluación final. Teniendo en cuenta cada estudiante sólo puede ver sus calificaciones, se cumple con gran parte de la especificación **F.DOCE.EVL**, aunque queda un detalle.

Para poder resolver el envío del mensaje con el resultado de cada prueba, se puede utilizar GitHub, escribiendo issues[9] en los repositorios en los que resuelven las actividades. Garantizando que cada estudiante sólo pueda acceder a sus repositorios, se cumple completamente con la especificación funcional **F.DOCE.EVL** mediante la adición de esta funcionalidad.

Finalmente, para resolver la especificación **F.DOCE.REE**, se puede utilizar la propia interfaz de GitHub Classroom. En ella, separada por grupos, se puede encontrar una lista que relaciona la identificación de una persona (su DNI, por ejemplo) con su cuenta en GitHub y con un enlace al repositorio en el que está resolviendo una actividad. Así, se dispone de una forma rápida y sencilla de acceder a la información deseada.

En comparación con la otra alternativa, se trata del caso contrario. Con esta segunda alternativa se consigue reducir el desarrollo y mantenimiento, pues sólo es necesario realizar pequeñas tareas de integración para el anuncio del resultado de la prueba. Para el resto de los aspectos, sólo se utilizan funcionalidades ya desarrolladas y fácilmente accesibles, por lo que el consumo de recursos se reduce drásticamente. Sin embargo, el precio a pagar es que deben utilizarse tres interfaces simultáneamente, lo que elimina la comodidad que suponía disponer de una interfaz única.

5.3. Criterios de evaluación

En este subapartado, se describen cuáles son los criterios con los que se van a evaluar las anteriores alternativas. Además de los criterios específicos para cada caso, se definen también una serie de criterios comunes con los que se van a evaluar todas las posibles soluciones de las problemáticas.

Cabe destacar que estos criterios se evalúan en base a unos indicadores obtenidos a través de las experiencias realizadas con prototipos, según lo definido en los objetivos del trabajo[10].

5.3.1. Criterios comunes

Los primeros criterios que se van a exponer son los que se van a aplicar a todas las problemáticas, pues se consideran transversales en la toma de decisiones. Los pesos de los criterios comunes son distintos para cada problemática, definiéndose durante la explicación de criterios específicos.

C.COMU.1 Facilidad del proceso de desarrollo e implementación

El primero de los criterios comunes hace referencia al coste en esfuerzo que supondría el desarrollo y/o la implementación de la alternativa. Se considera un criterio importante, ya que una mayor complejidad se acaba traduciendo en un mayor coste económico a la hora de afrontar la solución.

[9] Los issues son un componente dentro de los repositorios de GitHub (aunque también es usado en otras forjas) que permite a otros usuarios escribir mensajes en el repositorio, favoreciendo la comunicación entre desarrolladores y usuarios u otros desarrolladores.

[10] En el Anexo II, se encuentra la lista de prototipos desarrollados, con una breve explicación. En dicho anexo también puede consultarse a qué prototipo corresponde cada código.

Los indicadores de esta problemática se basan en los diferentes prototipos realizados, aplicando, en cada caso, los resultados de los suyos. Aun así, se define la siguiente escala de evaluación como la estándar para todos los casos:

1) Complicación de desarrollo excesiva. Se requiere de gente muy especializada en el ámbito. Implementación muy compleja.
2) Cierta dificultad de desarrollo. Se requiere de gente bien formada, aunque sin una gran especialización. La implementación es compleja pero abarcable.
3) Necesidad de desarrollo e implementación moderados. Desarrollos laboriosos pero sencillos en el fondo. La implementación es abarcable.
4) Baja necesidad de desarrollo, limitándose a desarrollos de integración. Poco o nulo despliegue.
5) Nula necesidad de desarrollo e implementación.

C.COMU.2 Coste económico ajeno al desarrollo

Si en el anterior criterio se evaluaba el coste relativo al desarrollo, en este se evalúan el resto de los costes, como el coste de uso de un servicio de compra de material. Cabe destacar que este criterio no se va a considerar tan importante en comparación a otros, ya que se está valorando la viabilidad técnica para el sistema y buscando un diseño más detallado. Si el objetivo fuera empezar con una explotación comercial a gran escala, este criterio ganaría mucho más peso.

Los indicadores para este criterio se han obtenido a partir de búsquedas del precio de los distintos costes atribuidos a cada solución. La escala de evaluación para este caso es:

1) Es necesario realizar un desembolso económico mayor a 1000€.
2) Es necesario realizar un desembolso económico entre 500€ y 1000€.
3) Es necesario realizar un desembolso económico entre 100€ y 500€.
4) Es necesario realizar un desembolso económico menor a 100€.
5) La solución puede implementarse de forma gratuita. No se requiere de ningún gasto económico ajeno al desarrollo.

5.3.2. Criterios específicos de la Problemática I: Gestión de los repositorios

Después de los criterios comunes, se enumeran y definen los criterios específicos con los que se pone solución a la problemática de gestión de los repositorios.

C.REPO.1 Capacidad de comunicación (20%)

El primero de los criterios mide las facilidades y prestaciones de las que dispone la forja para poder comunicarse con el exterior, ya sea mediante la detección de eventos (envío de webhooks) o mediante el uso de la interfaz de comunicación (API REST). Por tanto, de forma combinada, se mide la cantidad de eventos que puede detectar la forja (así como la facilidad de hallar en los webhooks la información de interés para el trabajo) y la sencillez que presentan las API REST de las forjas a estudiar a la hora de ser utilizadas e implementadas (así como la oferta existente en cuanto a librerías de implementación de las APIs).

Mediante el desarrollo de los prototipos P.OYEN y P.ESCR (junto con su integración las diferentes versiones de los prototipos P.AVEV y P.INTG), se han obtenido indicadores con los que definir la siguiente escala de evaluación:

1) Pocos eventos para detectar o pocas funcionalidades de la API. Los eventos apenas disponen de la información suficiente. No existen librerías de implementación.
2) Existen los eventos detectables y funciones de la API suficientes. Eventos con poca información. Variedad de librerías de implementación muy limitada.
3) Existen los eventos detectables y funciones de la API suficientes. Eventos con algo más de información. Poca variedad de librerías de implementación.
4) Amplia variedad de eventos detectables y funciones de la API. Eventos con una cantidad razonable de información. Amplia variedad de librerías de implementación, pero mal documentadas.
5) Amplia variedad de eventos detectables y funciones de la API. Eventos con mucha información. Amplia variedad de librerías de implementación bien documentadas.

C.REPO.2 Potencia de la aplicación de CI (20%)

Como se ha descrito durante el enunciado de la problemática, es necesario que las forjas dispongan de aplicaciones de CI para la evaluación del código. Este criterio mide la potencia de las aplicaciones a utilizar en las forjas, incluyendo si están o no integradas en ellas.

En aras de obtener indicadores para este criterio, se ha desarrollado el prototipo P.EVAL en varias versiones. La escala de evaluación para este criterio es:

1) Necesidad de utilizar una aplicación de pago ajena a la forja. Apenas ofrece las opciones más básicas.
2) Necesidad de utilizar una aplicación gratuita ajena a la forja. Ofrece opciones básicas, pero funciona de forma compleja.
3) Aplicación fácilmente integrable en la forja. Uso complejo o con pocas posibilidades.
4) Aplicación integrada en la forja. Uso complejo o con posibilidades moderadas.
5) Aplicación integrada en la forja. Uso sencillo y potente.

C.REPO.3 Capacidad de integración en el entorno académico (20%)

Por último, este criterio mide las facilidades que aporta la forja para ser implementada en un entorno académico.

Para poder evaluar correctamente este criterio, se ha desarrollado las primeras dos versiones del prototipo P.CCRE y el prototipo P.INGL, obteniendo la siguiente escala de evaluación:

1) No dispone de ninguna clase de herramienta o funcionalidad que permita adecuar la forja a un entorno académico. Necesidad de hacer un desarrollo completo.
2) Dispone de alguna de las funcionalidades, pero es necesario realizar el desarrollo completo de la mayoría de ellas.

3) Dispone de la mayoría de las funcionalidades, pero es necesario realizar el desarrollo completo de alguna de ellas.
4) Si bien es cierto que requiere el desarrollo de alguna pequeña herramienta, se ajusta muy bien a las necesidades de un entorno académico.
5) Dispone de un conjunto de herramientas integradas que permiten adecuarse perfectamente a las necesidades de un entorno académico.

Los pesos asignados a los criterios comunes son **30%** para el criterio **C.COMU**.1 y **10%** para el criterio **C.COMU.2**.

5.3.3. Criterios específicos de la Problemática II: Despliegue del contexto de pruebas

Los siguientes criterios a enunciar son los relativos a la segunda problemática, referida a la forma que toma el despliegue de los contextos donde se prueba el código desarrollado para la resolución de actividades.

C.EVAL.1 Adaptabilidad a las asignaturas (25%)

Según las especificaciones funcionales definidas a la hora de describir la problemática, el despliegue del contexto debe permitir adecuarse a las distintas asignaturas que hacen uso del sistema NME-CV. Por tanto, este criterio se crea para evaluar la sencillez que ofrece la alternativa a la hora de ofrecer esta adaptabilidad, tanto en términos de instalación como de mantenimiento del contexto.

A través de los indicadores obtenidos al desarrollar el prototipo P.EVAL se ha llegado a la siguiente escala de evaluación:

1) El contexto apenas se adapta a la asignatura. Instalar nuevas herramientas supone realizar un trabajo previo exhaustivo. Es necesario realizar mucho mantenimiento.
2) El contexto se adapta bien a la asignatura. Instalar nuevas herramientas supone una labor tediosa para una persona técnica, pero no es especialmente compleja. Es necesario realizar mucho mantenimiento.
3) El contexto se adapta bien a la asignatura. Instalar nuevas herramientas supone una labor tediosa para una persona técnica, pero no es especialmente compleja. No es necesario realizar mucho mantenimiento.
4) El contexto se adapta perfectamente a la asignatura. Resulta sencillo implementar nuevas herramientas. No es necesario realizar mucho mantenimiento.
5) El contexto se adapta perfectamente a la asignatura. Resulta sencillo implementar las herramientas. Las labores de mantenimiento son escasas o inexistentes.

C.EVAL.2 Facilidad de uso por parte de docente (35%)

Puesto que dentro del origen del sistema NME-CV también está la simplificación de la labor docente, no tendría sentido utilizar una solución demasiado complicada para este grupo. Por tanto, con este criterio se quiere medir lo manejable que puede ser para el profesorado utilizar y desplegar los contextos de prueba.

En este criterio, también se utilizado el prototipo P.EVAL para obtener la siguiente escala de evaluación:

1) El profesorado debe ser formado para convertirse en expertos en alguno de los aspectos del despliegue de los contextos. Debe realizar tareas adicionales muy complejas para el despliegue.
2) El profesorado debe recibir una pequeña formación para desplegar los contextos. Debe realizar tareas adicionales de complejidad moderada para el despliegue.
3) El profesorado debe recibir una pequeña formación para desplegar los contextos. Debe realizar muchas tareas adicionales sencillas para el despliegue.
4) No se requiere de una formación específica. El profesorado debe realizar pocas tareas adicionales sencillas para el despliegue.
5) No se requiere de ningún trabajo adicional por parte del profesorado (más allá de la definición de pruebas y herramientas necesarias).

Los pesos asignados a los criterios comunes son **30%** para el criterio **C.COMU.1** y **10%** para el criterio **C.COMU.2**.

5.3.4. Criterios específicos de la Problemática III: Gestión del entorno académico

En la tercera problemática, se plantea qué forma toma el modelo de datos que almacena la información del entorno académico para su gestión. Para escoger una solución entre las alternativas propuestas, se han definido sendos criterios.

C.DATO.1 Facilidad de operación sobre el modelo de datos (30%)

A través de este criterio, se evalúa cómo se opera sobre la base de datos en términos de sencillez, tanto para los módulos como para los usuarios que tengan que hacerlo.

En este caso, se han desarrollador los prototipos P.DBPR, P.APDB y P.MAPI[11] con las que se ha obtenido la siguiente escala de evaluación:

1) La gestión del modelo de datos es complicada e implica la creación de muchos formularios y herramientas para ello. Los módulos deben disponer de excesivas entidades para poder manejarla.
2) La gestión del modelo es difícil y se necesitan muchos formularios y herramientas complejas para para ello. Los módulos deben disponer de muchas entidades para poder manejarla.
3) La gestión del modelo tiene una dificultad modera y se necesitan ciertos formularios y herramientas complejas para ello. Los módulos necesitan pocas entidades para poder manejarla.
4) La gestión del modelo es sencilla, pudiéndose gestionar con una menor cantidad de formularios y herramientas sencillas. Los módulos apenas necesitan entidades para manejarla.
5) La gestión del modelo es muy sencilla, pudiéndose gestionar con pocos formularios y herramientas sencillas. Los módulos apenas necesitan entidades para manejarla.

C.DATO.2 Necesidades de mantenimiento (25%)

El trabajo por realizar sobre el modelo de datos no termina con su implementación. Es necesario realizar procesos de mantenimiento para asegurarse de que sigue funcionando correctamente durante el uso y que los errores que puedan ocurrir no echen por tierra todo el trabajo. De esta forma, se enuncia el criterio que mide la complejidad y/o cantidad de labores de mantenimiento a realizar sobre el modelo de datos.

A través de los prototipos del criterio anterior, también se han obtenido los indicadores para la escala de evaluación de este criterio, que es la siguiente:

1) Mantenimiento excesivo e inabarcable. Se requiere de un equipo dedicado en exclusiva a ello.
2) Mantenimiento complejo. Se requiere de una persona dedicada en exclusiva a ello.
3) Mantenimiento moderado. Se requiere de una persona que compagine sus labores con las de mantenimiento.

[11] Revisando la lista de prototipos descrita en el Anexo II, se puede apreciar que existen los prototipos P.MDIR y P.GDAT, que hacen referencia a un modelo de datos creado a partir de ficheros CSV y XML. Esto corresponde a una versión en la cual no se optaba todavía por una base de datos relacional y se exploró cómo utilizar ese tipo de ficheros. Esta opción fue finalmente desechada en favor de las bases de datos y la problemática se ha enfocado más a lo que supone usar esta tecnología, obviando el intento de usar los ficheros. Por tanto, no se consideran los resultados de esos prototipos para la obtención de indicadores.

4) Mantenimiento sencillo. Se requiere de una persona que compagine sus labores con las de mantenimiento en las pocas situaciones que se requiera.
5) Mantenimiento escaso o inexistente.

Los pesos asignados a los criterios comunes son **35%** para el criterio **C.COMU.1** y **10%** para el criterio **C.COMU.2**.

5.3.5. Criterios específicos de la Problemática IV: Interfaz de gestión de los usuarios

Por último, se listan los criterios con los que tomar la decisión de solución sobre la problemática relativa a la interfaz a través de la cual los usuarios acceden a sus datos.

C.DOCE.1 Facilidad de uso por parte de los usuarios

Tal y como se ha mencionado durante la definición del criterio C.EVAL.2 sobre la facilidad de uso por parte del docente, el sistema NME-CV pretende, entre otras cosas, simplificar la labor docente. Además de a este grupo, el sistema también mostrar sencillez al alumnado que cursa sus asignaturas utilizando este sistema. Por tanto, con este criterio se mide la facilidad de uso que encuentran los usuarios durante la utilización de la interfaz.

Basándose de una parte de los resultados de los prototipos P.APDB y P.MAPI y combinándolos con pequeñas experiencias con las interfaces de GitHub, se ha obtenido la siguiente escala de evaluación:

1) Los usuarios deben utilizar guías complejas para entender cómo visualizar la información.
2) Los usuarios deben utilizar guías de complejidad moderada para entender cómo visualizar la información.
3) Los usuarios deben utilizar guías sencillas para entender cómo visualizar la información.
4) Los usuarios no requieren de guías para entender cómo visualizar la información, aunque si necesitarán aprender a manejarse correctamente.
5) Los usuarios no requieren de guías para entender cómo visualizar la información y pueden utilizar la interfaz de forma intuitiva.

C.DOCE.2 Necesidades de mantenimiento

Este criterio se define por el mismo motivo y de la misma forma que el criterio C.DATO.2, aplicado esta vez a la interfaz de gestión de usuarios.

En base a los mismos indicadores del criterio C.DOCE.1, se ha definido una escala de evaluación idéntica a la que se definió en el criterio C.DATO.2. Recordándola:

1) Mantenimiento excesivo e inabarcable. Se requiere de un equipo dedicado en exclusiva a ello.
2) Mantenimiento complejo. Se requiere de una persona dedicada en exclusiva a ello.
3) Mantenimiento moderado. Se requiere de una persona que compagine sus labores con las de mantenimiento.

4) Mantenimiento sencillo. Se requiere de una persona que compagine sus labores con las de mantenimiento en las pocas situaciones que se requiera.
5) Mantenimiento escaso o inexistente.

Los pesos asignados a los criterios comunes son **35%** para el criterio **C.COMU.1** y **10%** para el criterio **C.COMU.2**.

5.4. Decisión

Una vez realizadas todas las definiciones, sólo queda tomar la decisión de qué alternativa se escoge en cada problemática, basando esa decisión en los criterios comunes y específicos.

5.4.1. Solución de la Problemática I: Gestión de los repositorios

A continuación, se muestra la tabla de decisión de solución para a problemática relativa a la forja a utilizar.

	C.COMU.1 (30%)	***C.COMU.2 (10%)***	***C.FORJ.1 (20%)***	***C.FORJ.2 (20%)***	***C.FORJ.3 (20%)***	***TOTAL***
A.FORJ.1	2	4	5	5	3	**3'6**
A.FORJ.2	5	1	5	5	5	**4'6**

Tabla 8. Tabla de decisión de la Problemática I

Por tanto, la decisión que se toma es la de utilizar la forja de GitHub, pues, aun siendo dos alternativas parecidas, el disponer de GitHub Classroom y deshacerse del mantenimiento de la instancia propia, acaba por simplificar tanto el desarrollo de integración como el despliegue. Todo esto se consigue pese a tener un coste mucho mayor[12].

Esta decisión también supone la aparición del primer módulo del sistema: el **Módulo Repositorio**, el cual alberga la forja. Por tanto, será el punto de la arquitectura en el que el grupo de estudiantes suban sus códigos para ser evaluados.

5.4.2. Solución de la Problemática II: Despliegue del contexto de pruebas

En cuanto a la problemática sobre el despliegue del contexto de pruebas, se ha utilizado la siguiente tabla de decisión para hallar la solución.

[12] En este punto, se quiere puntualizar una cosa. Siendo más exactos, el coste del plan de GitHub mínimo para poder hacer viable el sistema es de 252$ por mes y usuario. En un entorno universitario, esto dispararía los costes de forma absurda, por lo que la solución de GitHub, en verdad, no sería viable. Sin embargo, GitHub se muestra superior en el resto de los aspectos y permite hacer implementaciones de forma más sencilla y potente. Además, dispone de planes para educación, ofreciendo rebajas y servicios adicionales a universidades, todo ello tras ponerse en contacto y negociar con el equipo comercial de GitHub. Por tanto, según esas negociaciones, se podría llegar a un precio que devolviera la viabilidad. Es por eso por lo que se ha decidido seguir adelante con GitHub, aprovechando también la mayor sencillez para demostrar la viabilidad técnica y de plantear el diseño (uno que, como se verá más adelante, es capaz de soportar GitLab también) y cumplir con los objetivos del trabajo. Aun así, debe tenerse en cuenta que, cuando se llegue a la fase de la explotación económica, si GitHub no puede hacerse viable, habría que migrar a GitLab, aprovechando en dicho proceso que también se han realizado prototipos demostradores para dicha forja.

	C.COMU.1 (30%)	C.COMU.2 (10%)	C.EVAL.1 (25%)	C.EVAL.2 (35%)	TOTAL
A.EVAL.1	3	4	2	5	**3'55**
A.EVAL.2	5	5	5	3	**4'3**

Tabla 9. Tabla de decisión de la Problemática II

En vista del resultado, se opta por utilizar contenedores. Esta tecnología se ha mostrado superior al uso de una máquina propia en todos los aspectos salvo en la facilidad para el profesorado. Por tanto, el profesorado tendrá que ser formado en el uso de contenedores y de cómo gestionarlos para la evaluación del código a través de CI.

En este momento, se define el segundo módulo del sistema, que será aquél que recoja los contextos de evaluación y la aplicación de CI. Este módulo es denominado como **Módulo Evaluador**.

5.4.3. Solución de la Problemática III: Gestión del entorno académico

Para resolver la tercera problemática, sólo queda realizar la tabla de decisión que permita conocer cuál es la alternativa que mejor se ajusta a los criterios.

	C.COMU.1 (35%)	C.COMU.2 (10%)	C.DATO.1 (30%)	C.DATO.2 (25%)	TOTAL
A.DATO.1	2	4	2	2	**2'2**
A.DATO.2	4	4	4	3	**3'75**

Tabla 10. Tabla de decisión de la Problemática III

En este caso, claramente gana la alternativa de utilizar un modelo mixto, gracias a su amplia reducción de dificultad de desarrollo y mantenimiento. Además, esta solución también supone una gestión más sencilla del modelo, lo que se traduce en menor probabilidad de errores.

Por tanto, sabiendo la forma que toma el modelo de datos, se puede definir ya el **Módulo Datos**, que encapsula dicho modelo para ser utilizado por el resto de los módulos del sistema.

5.4.4. Solución de la Problemática IV: Interfaz de gestión de los usuarios

En esta última problemática, la tabla de decisión toma la forma que puede verse a continuación.

	C.COMU.1 (35%)	C.COMU.2 (10%)	C.DOCE.1 (30%)	C.DOCE.2 (25%)	TOTAL
A.DOCE.1	3	4	4	3	**3'4**
A.DOCE.2	4	4	2	5	**3'65**

Tabla 11. Tabla de decisión de la Problemática IV

Finalmente, en esta última problemática la alternativa de utilizar una interfaz híbrida vence pese a ser menos intuitiva para los usuarios. Se debe a que, aun así, anula casi totalmente el desarrollo y lo que ello conlleva. Por tanto, el sistema utilizará esas plataformas para mostrar la información.

Aunque luego se requieran de más funcionalidades para definirlo completamente, esta problemática inicia la definición del **Módulo Docencia**, donde se recogerán los elementos para poder mostrar la información de docencia.

6. Análisis Funcional

Una vez conocidas las soluciones a las principales problemáticas del proyecto, se procede a analizar cuáles deberían ser las funcionalidades concretas que deberían cumplir cada uno de los módulos del sistema. De esta forma, se pretende finalizar teniendo la lista definitiva de módulos que compondrán el sistema para, así, poder completar el diseño de la arquitectura en apartados posteriores.

6.1. Entornos de aplicación y usuarios potenciales

Para poder realizar este análisis, previamente, se valora cuáles son los entornos en los que puede aplicarse este sistema y qué usuarios habría en el sistema. De esta manera, se podrán terminar de evaluar cuáles son las funcionalidades que debe cumplir el sistema.

El principal entorno de aplicación del sistema y para el que se ha planteado es para el ámbito académico, en especial, para el universitario. Como se ha mencionado en el apartado "Objetivos y alcance", la solución propuesta debe ser aplicable en el contexto de la EIB, por lo que se ha partido de la base de que se utilizan los recursos allí disponibles. Sin embargo, esto no quiere decir que no pueda ser aplicable en otros entornos académicos (sean o no universitarios), pues según esas mismas especificaciones generales la solución debe ser lo suficientemente abierta para poder adaptarse a otros entornos (incluso, a otras herramientas y plataformas diferentes a la solución que se aporta en este documento).

Además de la educación reglada, este sistema también puede utilizarse en cursos que se impartan de forma on-line, para los que puede resultar especialmente beneficioso al simplificar las labores de unos docentes que pueden tener miles de personas inscritas simultáneamente.

Más allá del propio entorno, el sistema se ha desarrollado pensando en dar soluciones más eficientes a la acumulación de corrección de prácticas derivado de realización de una evaluación continua. Aun así, nada impide que este sistema pudiera llegar a ser utilizado en ejercicios más contenidos o, incluso, en exámenes.

En cuanto a los usuarios que van a hacer uso del sistema, se identifican los siguientes:

- Usuario Docente (UsuDo): Representa a cualquier docente de una asignatura que va a proponer a sus estudiantes resolver una o varias actividades a través del sistema planteado en este proyecto.
- Usuario Estudiante (UsuEs): Representa a cualquier estudiante de cualquier curso académico que debe realizar las actividades propuestas por un UsuDo.
- Usuario Administrador (UsuAd): Representa a una persona responsable de la administración del sistema.
- Usuario de Mantenimiento (UsuMa): Representa a una persona responsable de la instalación y mantenimiento de los módulos del sistema que requieran de un desarrollo. Entre sus labores, destaca la corrección de errores o la instalación de actualizaciones.

6.2. Análisis y especificaciones funcionales

Se realiza ahora el análisis funcional del sistema a través de la identificación de las especificaciones funcionales a cumplir para poder satisfacer las necesidades del sistema y cumplir los objetivos del proyecto. Para ello, es necesario recapitular sobre algunas funcionalidades de las que se hablaron en el apartado *Análisis de la viabilidad técnica*, de forma que se pueda terminar de dar forma a los módulos que se definieron en ese momento[13].

6.2.1. Especificaciones funcionales de la forja

Las especificaciones funcionales de la forja son las que se recogen en la tabla 12.

6.2.2. Especificaciones funcionales de los contextos y aplicación de evaluación

Las especificaciones funcionales de los contextos y aplicación de evaluación son las que se recogen en la tabla 13.

6.2.3. Especificaciones funcionales sobre el modelo de datos

Las especificaciones funcionales del modelo de datos son las que se recogen en la tabla 14.

6.2.4. Especificaciones funcionales de los aspectos docentes

Las especificaciones funcionales de los aspectos docentes son las que se recogen en la tabla 15.

6.2.5. Especificaciones funcionales de integración

Una vez revisadas todas las especificaciones funcionales de los grupos previamente vistos, se procede a definir aquellas relativas a la integración de los módulos que conformarán el sistema. Para realizar este proceso, se pone el foco en las necesidades de comunicación entre los módulos y la gestión de éstas, así como de los procesos de automatización que deberían existir.

Las especificaciones funcionales de integración son las que se recogen en la tabla 16.

[13] Pese a haber realizado el análisis, en este subapartado sólo se van a proveer las tablas resumen que contienen todas las especificaciones funcionales junto con sus códigos. La descripción de las funcionalidades puede encontrase en el Anexo I.

Especificaciones Funcionales (**F**)	Forja (**FORJ**)	Detección de Eventos (**EVE**)	Detección del Evento de Creación de Repositorios (**1**)
			Detección del Evento de Evaluación (**2**)
			Detección del evento de finalización de prueba concreta (**3**)
		Interfaz de Comunicación (**COM**)	Configuración del Detector de Eventos (**1**)
			Escritura de Resultados de Evaluación en Repositorios (**2**)
			Ejecución de Acciones en Nombre de Docente (**3**)
		Compatibilidad con Aplicaciones de CI (**ICO**)	Máxima Integración con la Forja (**1**)
			Mínimo Despliegue para la Aplicación (**2**)
		Adecuación al Entrono Académico (**EDU**)	Creación de Repositorios para Todos el Grupo de Estudiantes (**1**)
			Repositorio de Referencia (**2**)
			Privacidad de los Repositorios (**3**)
			Organización de Estudiantes en Grupos (**4**)
			Separación entre Prácticas Dependientes e Independientes (**5**)

Tabla 12. Resumen de las especificaciones funcionales de la forja

Especificaciones Funcionales (**F**)	Evaluador (**EVAL**)	Compatibilidad con Múltiples Tipos de Pruebas (**PRU**)	Soporte para Pruebas Opcionales (**1**)
			Instalación Sencilla de Herramientas Generales (**2**)
		Compatibilidad con Múltiples Asignaturas (**ASI**)	Instalación de Herramientas para Manejar Diferentes Lenguajes (**1**)

Tabla 13. Resumen de las especificaciones funcionales de los contextos y aplicación de evaluación

Especificaciones Funcionales (**F**)	Modelo de Datos (**DATO**)	Registro del Perfil de Estudiantes (**EST**)	Registro Automático de Estudiantes (**1**)
			Identificación de Asignatura y Grupo (**2**)
			Perfil de Docente (**3**)
			Relación con la Cuenta en la Forja (4**)**
		Registro de Resultados de Evaluación (**EVA**)	Asociación de Cada Evaluación con su Resolución (**1**)
			Cálculo de la Evaluación Final (**2**)
			Relación entre Asignaturas y Actividades con la Forja (**3**)
		Seguridad de los Datos Almacenados (**SEC**)	Control de Acceso al Modelo de Datos (**1**)
			Consulta y Modificación en Nombre de Docente (**2**)
			Almacenamiento de Información Sensible (**3**)

Tabla 14. Resumen de las especificaciones funcionales del modelo de datos

Especificaciones Funcionales (**F**)	Aspectos Docentes (**DOCE**)	Lista de Estudiantes (**LIS**)	Lista Separada por Asignaturas (**1**)
		Anuncio de Resultados de Evaluación (**EVL**)	Privacidad de los Resultados de Evaluación (**1**)
		Acceso Rápido a Repositorios de Estudiantes (**REE**)	Relación entre Estudiante y Repositorio en Grupo Concreto (**1**)
		Registro de Información (**REG**)	Registro de Información Adicional de Estudiantes (**1**)
			Registro del Mensaje de Resultado de Prueba (**2**)
			Registro de Tokens (**3**)
			Registro de Asignaturas y Actividades (**4**)

Tabla 15. Resumen de las especificaciones funcionales de aspectos docentes

Especificaciones Funcionales (**F**)	Integración (**INTE**)	Comunicación con el Módulo Repositorio (**REP**)	Recepción de Eventos en los Repositorios (**1**)
			Filtrado de la Información de los Eventos (**2**)
			Ejecución de Acciones a Través de Interfaz de Comunicación (**3**)
		Gestión Docente (**DOC**)	Comunicación con el Modelo de Datos (**1**)
			Uso de la Interfaz de Comunicación (**2**)
			Gestión de las Transacciones con la Base de Datos (**3**)
		Gestión de la Automatización del Sistema (**AUT**)	Automatización de la Evaluación (**1**)
			Automatización de la Configuración de Eventos (**2**)
			Acceso a Tokens de Identificación de Docentes (**3**)

Tabla 16. Resumen de las especificaciones funcionales de integración

7. Diseño

Una vez realizado los análisis de viabilidad técnica y funcionales, con los que se han conocido los módulos que van a conforma el sistema, se procede a describir la solución final que se ha tomado para poder cumplir con todas las especificaciones, tanto generales como funcionales.

En esta solución, se propone una arquitectura completa y detallada formada por dichos módulos. Se entiende por arquitectura completa y detallada una que no sólo muestre los módulos y sus comunicaciones, sino que describa también el funcionamiento interno del módulo, esto es, qué componentes lo conforman y cómo se comunican entre ellos. Para lograr este objetivo, se parte del estudio técnico con el que se definió el proyecto, utilizando las tecnologías allí descritas y basándose en la arquitectura de referencia provista.

A su vez, se van a organizar los módulos en varios subsistemas, que no son otra cosa que grupos de módulos con funciones relacionadas.

7.1. Integración de los módulos

En apartados anteriores, se han ido definiendo varios de los módulos que conforman la arquitectura de la solución del proyecto NME-CV. Sin embargo, para que este sistema funcione correctamente, falta un último módulo que sea capaz de gestionar la integración del resto, satisfaciendo las especificaciones funcionales de integración listadas durante el análisis funcional. Se define entonces el denominado como **Módulo Gestor**.

Este módulo consiste en el conjunto de herramientas necesarias para poder interactuar con el Módulo Repositorio (recogiendo los eventos que ocurran y realizando acciones sobre el repositorio pertinente) y almacenar en el modelo de datos cualquier información relevante sobre lo que haya ocurrido en el Módulo Repositorio, incluyendo escribir las evaluaciones de las actividades en la plataforma del Módulo Docencia. Es, por tanto, el módulo que lleva el peso de la automatización del sistema, decidiendo qué acciones tomar en cada momento e integrando el resto de los módulos.

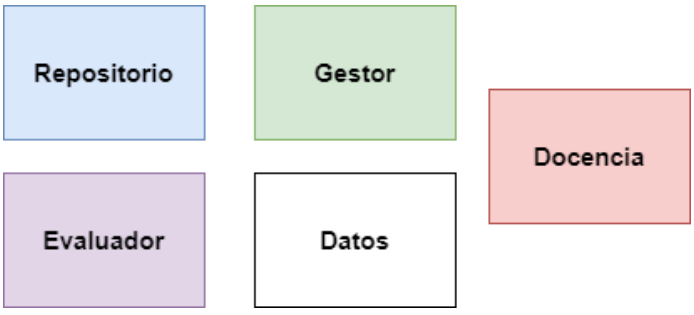

Figura 1. Módulos del sistema NME-CV

Finalmente, cabe destacar que el único módulo del sistema con el que no es necesario que se comunique es con el Módulo Evaluador, ya que éste es gestionado íntegramente por la aplicación de CI del Módulo Repositorio. Además, como el único dato que interesa de dicho módulo es el resultado de las evaluaciones (tanto el general como el de cada parte específica) y éste puede ser obtenido a

través de los webhooks del Módulo Repositorio, se considera que es suficiente con comunicarse con este último.

7.2. Arquitectura

Como solución, se propone una arquitectura compuesta por cinco módulos repartidos entre tres subsistemas, además de una base de datos. Como se ha visto anteriormente, los módulos son:

- **Módulo Repositorio (MRe)**. Módulo que contiene la forja (en concreto para esta solución, GitHub) a la que los UsuEs subirán el código con el que resolver sus actividades.
- **Módulo Evaluador (MEv)**. Módulo encargado de la evaluación del código, por lo que se forma de la aplicación de CI junto con los contextos en los que se prueba. Para esta solución, como se verá más adelante, se utiliza GitHub Actions, que corresponde con la aplicación instalada por defecto en la forja escogida.
- **Módulo Gestor (MGe)**. Módulo que contiene la lógica necesaria para la integración de las distintas partes de la arquitectura, siendo el corazón de la arquitectura. Debido a esto, en este módulo no se implementa una solución de terceros, sino que es necesario realizar un desarrollo propio para poder introducir al sistema todas las herramientas de integración y automatización. Este desarrollo se realiza en Java, aprovechando el framework Spring.
- **Módulo Datos (MDa)**. Módulo encargado de almacenar toda la información relevante para el sistema NME-CV. Tiene un diseño híbrido, pues está formado por dos bases de datos relacionales diferentes. Por un lado, en esté módulo se ubica la base de datos de Moodle (DB_M), y por ende el productor de la API REST de dicha plataforma, y, por otro, la base de datos propia que contiene la información de integración de los módulos (DB_P).
- **Módulo Docencia (MDo)**. Módulo encargado de mostrar a los usuarios la información contenida en el MDa, así como de crear y modificar parte de la información (en concreto, toda aquella información que deba ser introducida manualmente y que no esté relacionada con la obtenida durante el funcionamiento del sistema). De la misma forma que el MDa disponía de un diseño híbrido, este módulo también lo tiene, pues por un lado tiene la propia aplicación de Moodle, para gestionar la DB_M y mostrar su información, y, por otro, una aplicación propia para manejar la DB_P.

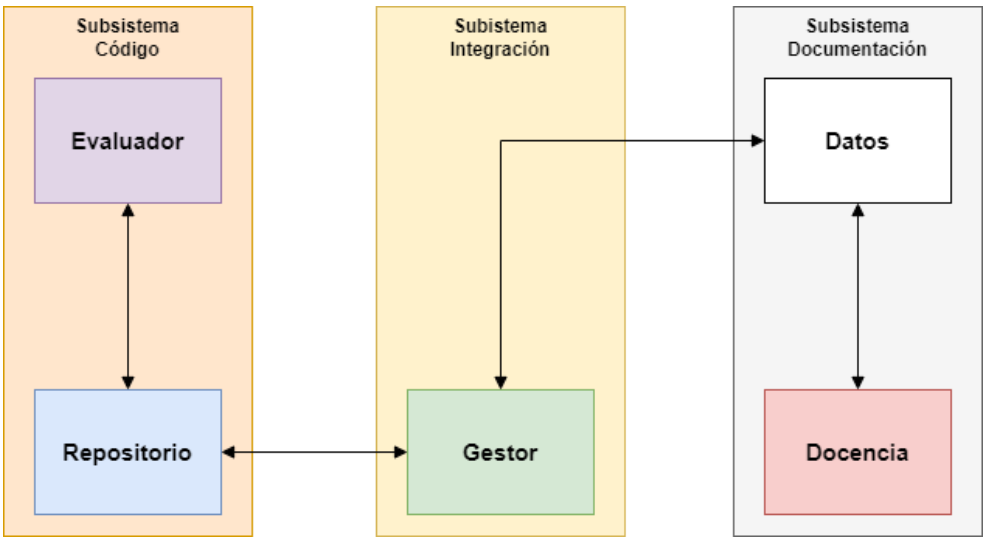

Figura 2. Arquitectura del sistema NME-CV

Los módulos que cumplen funciones relacionadas se agrupan en subsistemas para una mejor organización. En concreto, se distinguen tres:

- **Subsistema Código (SC)**. Es el subsistema que contiene los módulos que interactúan directamente con el código que desarrollan los UsuEs para completar las actividades. Está formado por los módulos MRe y MEv.
- **Subsistema Integración (SI)**. Se trata del subsistema que contiene el módulo encargado de la integración del resto de módulos, esto es, del MGe.
- **Subsistema Documentación (SC)**. Es el subsistema que contiene los módulos cuyas funciones se relacionan con la gestión y muestra de la información de docencia generada en el sistema. Por tanto, en sus módulos se va a registrar información acerca de los UsuEs y su evaluación de las actividades que completen utilizando el sistema NME-CV. Está formado por los módulos MDo y MDa.

7.2.1. Comunicación entre módulos

Para el correcto funcionamiento del sistema, los módulos deben comunicarse entre sí, para las que se utilizarán ciertas tecnologías en cada caso. Las comunicaciones que se dan en el sistema son:

- **Repositorio-Evaluador**. Se comunican utilizando la aplicación de CI instalada en el MRe. De esta forma, se consigue evaluar el código que los UsuEs suben al MRe en los contextos que conforman el MEv. Por tanto, la comunicación hacia el MEv va a consistir en la descarga del contenido del repositorio, mientras que lo que se va a enviar hacia el MRe es el resultado de las pruebas.
- **Repositorio-Gestor**. La comunicación entre estos dos módulos se realiza mediante dos tecnologías que funcionan a través del protocolo HTTP como son webhooks y API REST. En ambos casos, las tecnologías están implementadas dentro la forja, siendo el MGe el cliente de ambas tecnologías. En concreto, el MGe recibe los webhooks que el MRe envía como respuesta a los eventos que ocurren dentro (una nueva evaluación, por ejemplo) y ejecuta

acciones en él consumiendo la API REST provista (la creación de una issue que muestre el resultado de ésta).

- **Gestor-Datos**. El MGe debe ser capaz de poder realizar acciones de lectura y escritura (CRUD) sobre las bases de datos del MDa. Debido a la naturaleza del MDa, esta comunicación debe realizarse tanto mediante HTTP (para acceder a la DB_M desde la API REST) como mediante JDBC (para acceder a la DB_P desde Java).
- **Docencia-Datos**. Al igual que el MGe, el MDo debe poder realizar las operaciones de tipo CRUD sobre las bases de datos del MDa, por lo que la comunicación se realiza mediante los mismos protocolos y utilizando las mismas tecnologías.

Como resumen, se añade la siguiente figura, en la que se muestran cuáles son las diferentes comunicaciones entre los módulos.

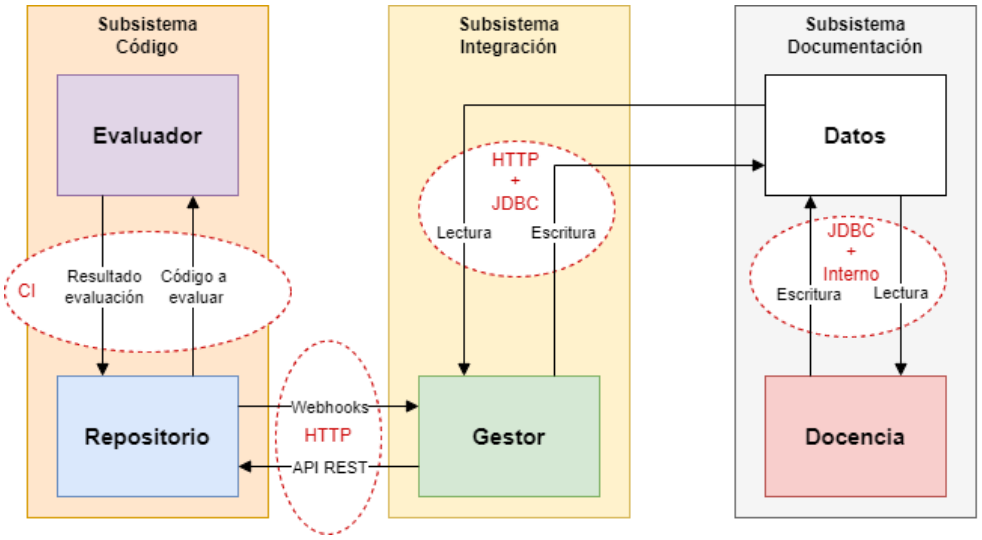

Figura 3. Comunicación entre módulos

7.2.2. Interfaces de usuarios con el sistema

En el apartado del análisis funcional, se describieron cuáles eran los diferentes usuarios que pueden encontrarse en el sistema. A continuación, se describe cómo interactúan dichos usuarios con la arquitectura, definiendo sus interfaces con el sistema y las labores que deben realizar en ellos.

- **UsuEs**

  Los UsuEs disponen de varias interfaces con el sistema, siendo la principal el MRe. Esto se debe a que es allí donde deberán subir sus códigos para resolver las actividades propuestas por los UsuDo, detonando el proceso de evaluación. Pero no sólo será usada para subir códigos, sino que se pretende que los UsuEs puedan encontrar en este módulo los resultados de las pruebas realizadas al código, de forma que sepan si han superado o no las pruebas.

  La otra interfaz que utilizan los UsuEs es el MDo, que usarán para poder consultar las calificaciones de sus actividades, así como acceder a la evaluación final de la parte de la asignatura evaluada mediante este sistema.

- **UsuDo**

  En cuanto a los UsuDo, también disponen de varias interfaces, siendo la principal el MDo. Desde esa interfaz, los UsuDo pueden primeramente consultar la información generada por los UsuEs durante su uso en el sistema, destacando la evaluación de sus actividades. Para ello, deben asegurarse de que crean las actividades de Moodle que requieran para cumplir las actividades que se van a seguir mediante el sistema NME-CV y configurar sus pesos para así poder calcular la evaluación final. También deben utilizar este módulo en su parte de registro, de forma que puedan añadir la información requeridas a las bases de datos del MDa.

  El MRe supone otra interfaz de uso del sistema para los UsuDo. En este módulo, disponen del código subido por sus UsuEs, lo que le permite poder supervisar los avances de estos usuarios durante el desarrollo de las actividades de la asignatura. Además de esto, como se verá más adelante, deberán usar el módulo para crear los repositorios de referencia a partir de los cuales se generarán los repositorios de los UsuEs, utilizando después GitHub Classroom (ubicado en este módulo) para dicho proceso de generación. También deben utilizar este módulo para el proceso de pruebas al código, introduciendo en los repositorios de referencia los ficheros de pruebas al código y generando los contextos en los que se realizará este proceso. Esto último también será descrito más en detalle en apartados posteriores. Por último, deben utilizar esta interfaz para generar los tokens que sean necesarios para que el sistema pueda ejecutar acciones en su nombre.

- **UsuAd**

  El UsuAd sólo dispone de una interfaz, que es la que le proporciona el MDo. Esto se debe a que, al utilizar Moodle en dicho módulo, el UsuAd debe cerciorarse de que todos los UsuEs se encuentran en sus correspondientes asignaturas al principio del curso o del cuatrimestre. Además, teniendo el control de Moodle, deben proveer a los UsuDo de sus tokens de Moodle y debe crear los servicios de su API REST (se explicará más adelante).

- **UsuMa**

  En general, los UsuMa deben poder acceder a todos los módulos en los que deba hacerse un desarrollo, de forma que pueda realizar en ellos las labores de mantenimiento. En concreto, estos módulos son el MGe, el MDa y el MDo. En todos ellos, la interfaz es igual: deben poder acceder por consola a las máquinas que alberguen dichos módulos de cara a poder realizar sus tareas.

Como resumen, se provee de la siguiente figura, en la que se ubican a los diferentes usuarios y su relación con las interfaces que encuentran en los módulos.

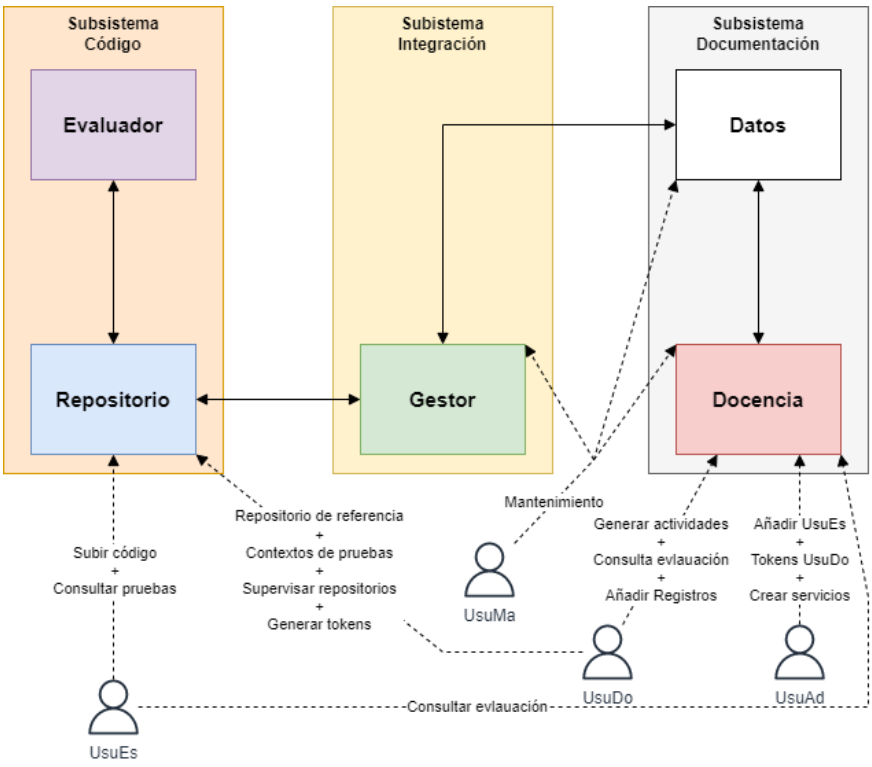

Figura 4. Usuarios del sistema y módulos que utilizan

7.3. Descripción detallada de los módulos

Hasta ahora, los módulos se han visto de forma superficial, como cajas negras que cumplen las funciones que se les hayan asignado. En este apartado, se pasa a describir cómo se han diseñado internamente para que cumplan con ellas, definiendo los compontes que conforman los módulos.

La imagen completa del sistema, incluyendo la descripción interna de los módulos puede verse en la siguiente figura.

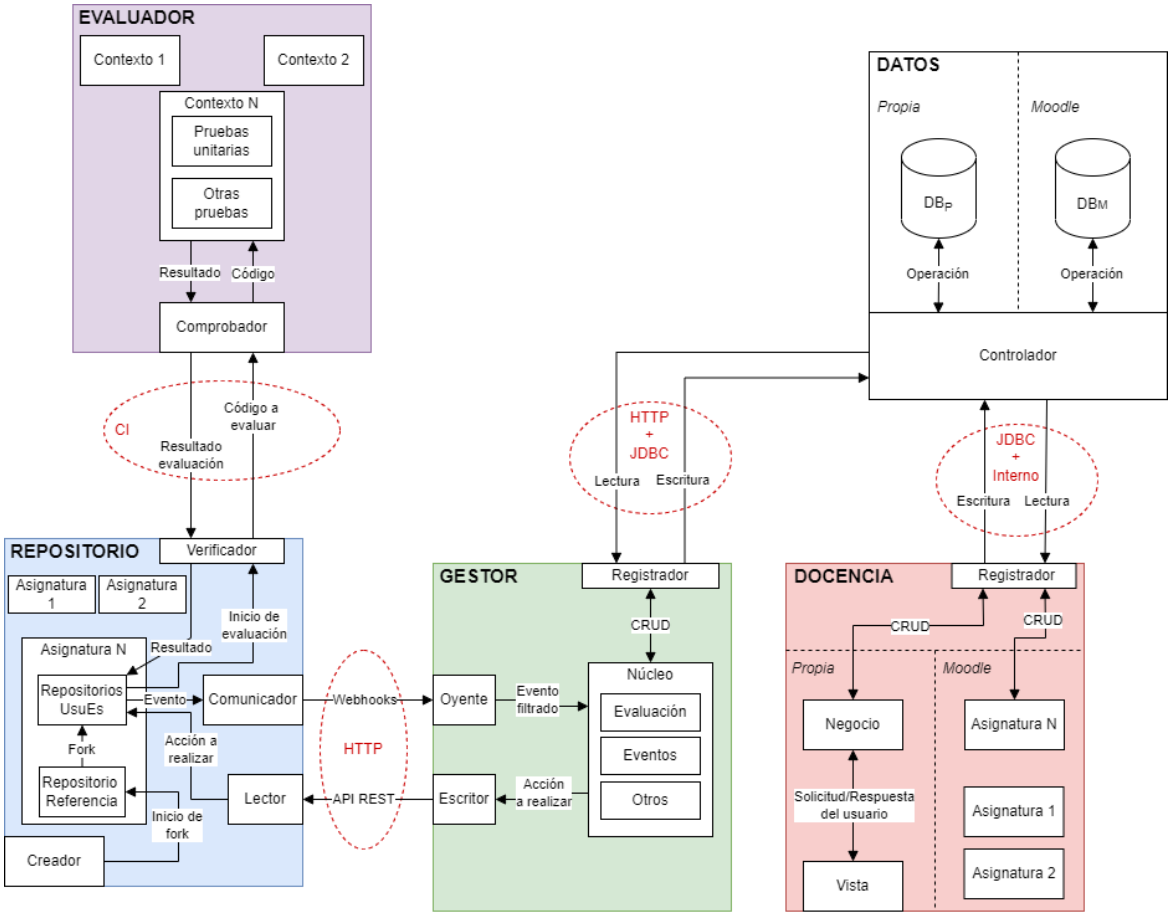

Figura 5. Arquitectura con diseño detallado

Para explicar la figura superior, se realiza la descripción detallada de cada módulo en los siguientes subapartados.

7.3.1. Descripción detallada del Módulo Repositorio

El primero de los módulos que se explora internamente es el MRe. Éste está considerado como uno de los módulos principales de la arquitectura, pues los eventos que ocurran en él son los que van a desencadenar los procesos que alteran el resto. Como se ha mencionado anteriormente, el MRe es el módulo que alberga la forja, siendo GitHub la escogida en esta solución.

Internamente, el MRe se puede ver como un conjunto de componentes independientes que representan una serie de asignaturas, en concreto, a aquellas que utilizan el sistema NME-CV como parte del proceso de seguimiento de la evaluación continua. A su vez, estos componentes de asignatura se forman a partir del conjunto de repositorios de estudiantes que cursan la asignatura en cuestión.

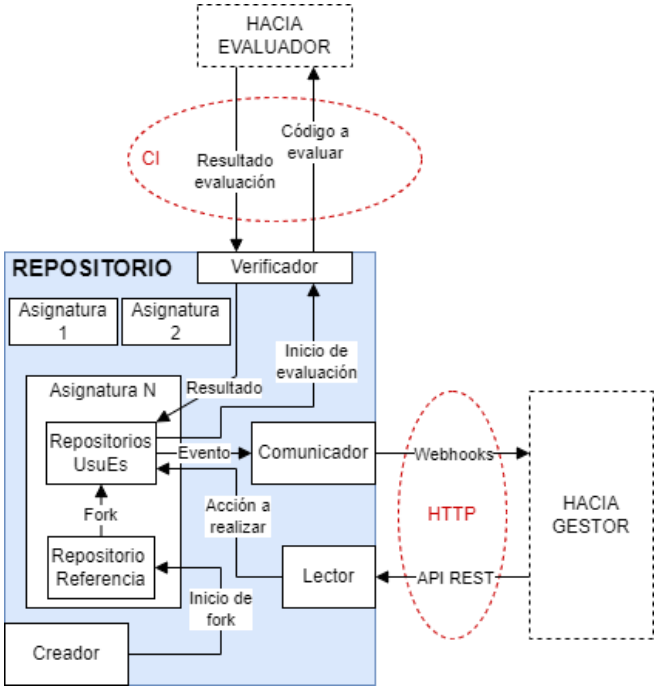

Figura 6. Diseño interno del MRe

Además de los componentes de asignaturas, existen otros que corresponden con las interfaces de este módulo con el resto del sistema. A saber, son los componentes Creador, Comunicador, Lector y Verificador.

El primero de los componentes para estudiar es el Creador. Su labor es la de crear los repositorios de los UsuEs a partir de cierto repositorio de referencia, por lo que supone que representa el uso de GitHub Classroom. Recordando lo comentado anteriormente, los repositorios de los UsuEs se generan a través de un proceso de fork de un cierto repositorio de referencia. Esto supone que, dentro de los repositorios que conforman cada componente de asignatura, existe una jerarquía, siendo el repositorio de referencia el repositorio padre y los repositorios de los UsuEs los repositorios hijos. La existencia de este componente "Creador" satisface la especificación funcional **F.FORJ.EDU.1**.

Antes de seguir explicando el resto de los componentes, es necesario hacer una pequeña pausa para explicar mejor cómo se integra GitHub Classroom en el sistema. Para poder realizar el fork, no basta sólo con indicar el repositorio de referencia, sino que es necesario disponer de una organización de GitHub y, dentro de ella, formar una subdivisión conocida como "Classroom". Estas classroom están formadas por grupos de usuarios que ya estén registrados dentro de la organización.

De entre las múltiples formas en las que se podrían distribuir las organizaciones y classrooms, se opta por la siguiente: se utiliza una única organización que represente a todo el centro educativo (por ejemplo, para el caso de la EIB sería una única organización para todos los UsuEs de la escuela) y dentro se formarían los classrooms para cada grupo de prácticas de cada asignatura. De esta manera se cumple con la especificación funcional **F.FORJ.EDU.4**.

En cuanto a la cantidad de repositorios de referencia que deben usarse en cada asignatura, no se especifica un número fijo, sino que deberían usarse todos los que se requieran para la impartición de la asignatura. La forma que deben tener estos repositorios de referencia (y, con ello, la forma en la que se cumple con la especificación funcional **F.FORJ.EDU.2**) se especificará más adelante en este mismo apartado.

Otro elemento a tener en cuenta durante el uso de GitHub Classroom es que se puede elegir el nivel de visibilidad de los repositorios que se crean. Mientras que el repositorio de referencia debe ser público obligatoriamente, los repositorios creados a partir de él sí que pueden ser privados. Al existir la posibilidad de crearlos de forma privada, se consigue cumplir con la especificación **F.FORJ.EDU.3**. Cabe destacar que, aunque exista la posibilidad de crear repositorios públicos para los UsuEs, se recomienda que los repositorios sean privados por defecto para mejorar la protección en el sistema, reservando la opción de repositorios públicos sólo a casos concretos.

Finalizando con la explicación de GitHub Classroom, cabe destacar que, como se mencionó en puntos anteriores, dispone de una interfaz en la que le muestra al UsuDo una vista en la que tiene relacionados a cada UsuEs con la dirección URL del repositorio en el que está resolviendo la práctica, disponiendo de una lista por cada grupo. Por tanto, con esto se cumple con la especificación **F.DOCE.REE.1**[14].

Volviendo a los componentes, el Comunicador representa al componente encargado de gestionar la comunicación de los eventos a través de webhooks, siendo, en este caso, el emisor de éstos. Este subcomponente es necesario para poder cumplir con las especificaciones funcionales del grupo **F.FORJ.EVE** y se soluciona mediante el propio motor de webhooks implementado dentro de los repositorios de GitHub. En concreto, hay que asegurarse de que todos los repositorios de los UsuEs se suscriban[15] a los eventos *Workflow jobs* (para detectar la finalización de cada una de las pruebas individuales al código) y *Workflow runs* (para detectar la finalización de la batería de pruebas). De esta manera, se cumplen las especificaciones funcionales **F.FORJ.EVE.3** y **F.FORJ.EVE.2** respectivamente.

Para poder completar la especificación **F.FORJ.EVE.1**, se debe utilizar el evento *Repositories* que eventos relacionados con los repositorios en general, aunque se va a poner el foco en el evento de creación de repositorios. Así, detectando esta creación (que se daría durante el proceso de fork del repositorio de referencia), se podrían configurar los webhooks en dichos repositorios, cumpliendo con la especificación. Sin embargo, hay que tener en cuenta que, para poder detectar estos eventos, el

[14] Si bien es cierto que se trata de una especificación no relacionada con la forja sino con los aspectos docentes, se considera que la solución que se aporta con GitHub Classroom es suficiente. Por tanto, no se considera necesario resolver este punto en el MDo, esto es, el módulo que engloba el resto de las especificaciones funcionales de tipo F.DOCE.

[15] Los webhooks en GitHub funcionan de la siguiente manera: en cada repositorio, existe una lista cerrada de eventos que el repositorio es capaz de escuchar para enviar un webhook cuando este ocurra. Sin embargo, los webhooks no están escuchando ningún evento por defecto, sino que es necesario escoger alguno de los eventos de esa lista, esto es, suscribirse a alguno de ellos.

webhook no debe configurarse a nivel de repositorio, sino a nivel de organización. Por tanto, este webhook debe configurarse en la organización que representa a todo el centro educativo.

Por otro lado, el Lector representa al productor de la API REST, comunicándose con el MGe a través de ésta. Mediante este subcomponente, se cumplen con las especificaciones del grupo F.FORJ.COM. En concreto, se plantea utilizar la función *Create an issue* dentro de la sección de "Issues". Gracias a esta, se pueden escribir issues con cualquier contenido dentro de los repositorios de los UsuEs desde el MGe. El disponer de esta función hace que se cumpla con la especificación funcional **F.FORJ.COM.2**.

Para poder configurar los detectores de eventos en los repositorios (esto es, los webhooks), es necesario utilizar la función *Create a repository webhook* de la sección *Webhooks*. Según esta función, se pueden crear webhooks dentro de los repositorios, seleccionando los eventos que se quieren escuchar, la dirección URL a la que se envían y el secret token que se va a utilizar para su autenticación. Con todo ello, se cumple con la especificación **F.FORJ.COM.1**.

Se procede a hacer otra pequeña pausa antes de terminar de explicar los componentes para explicar un aspecto relacionado con el uso de la API REST. Como se ha mencionado anteriormente, se favorece el uso de repositorios privados para los UsuEs. En estas situaciones, es obligatorio utilizar algún modo de autenticación para poder realizar cualquier acción sobre el repositorio, de forma que se pueda comprobar que quien realiza la acción tiene autorización dentro del repositorio.

En el caso de GitHub, la única forma de realizar esto actualmente es mediante el uso de tokens OAuth[16]. Cada usuario puede crear varios tokens OAuth con diversos permisos de uso de la API REST. Siguiendo la especificación F.FORJ.COM.3, cada UsuDo debe disponer de su propio token OAuth para poder realizar las acciones. Por tanto, en GitHub, previo a utilizar el sistema NME-CV, cada UsuDo ha tenido que crear uno de estos tokens indicando como *scope* (alcance, esto es, los permisos otorgados) *repo* y *repo_hook*, pues son la autorización para utilizar las funciones necesarias. Una vez creados, deben almacenarse dentro de este sistema de forma segura, aunque esto se verá más adelante.

De esta forma, cada UsuDo dispondría de su propio token OAuth, aunque, para poder usarlos, sería necesario que estuvieran dentro de los repositorios de los UsuEs, para así poder realizar las acciones. Esto se resuelve de manera sencilla, pues a la hora de crear los repositorios de los UsuEs mediante GitHub Classroom se asigna a cada UsuEs a su repositorio, pero también se añade al UsuDo que ha creado la actividad a cada uno de los repositorios. Así, el UsuDo ya podría utilizar su token OAuth para modificar los repositorios, cumpliendo con la especificación **F.FORJ.COM.3**.

El siguiente de los componentes a describir es el Verificador, que representa al agente del MEv dentro del MRe. Este agente se encarga de comunicarse, a través de la aplicación de CI, con el MEv para la

[16] Se trata de una tecnología para que el usuario de una API pueda identificarse a la hora de consumir una función de ésta. Se trata de una cadena de texto sin un formato particular que se da de alta en la API (donde se relaciona con un usuario y se le asignan unos permisos) y que se almacena en el cliente. Después, en cada petición HTTP, se envía el token dentro de la cabecera y, con ello, se consigue la identificación. Para que este sistema funcione, tanto el almacenaje como el envío deben ser seguros, de forma que se evite la intercepción del token.

evaluación del código de los UsuEs. En concreto, se decide utilizar la aplicación de CI integrada en GitHub: GitHub Actions. Esta aplicación está disponible para todos los repositorios de GitHub y es gestionada íntegramente por GitHub, por lo que se cumple el escenario ideal de las especificaciones funcionales **F.FORJ.ICO.1** y **F.FORJ.ICO.2**. Se reserva la explicación detallada de la aplicación y de su uso para el apartado que detalla el MEv.

Finalmente, aunque ya se ha especificado la necesidad de disponer de un repositorio de referencia, falta por concretar la forma que toma de cara a poder cumplir con la especificación funcional F.FORJ.EDU.2. En general, cada asignatura debe disponer de todos los repositorios de referencia que sean necesarios para poder completar todas las actividades de la asignatura. Sin embargo, como se ha mencionado durante la definición de la especificación, se premia la reducción de repositorios de referencia, de forma que sean más fáciles de gestionar.

En aras de conseguir reducir el número y cumplir con el ideal, se propone como solución el utilizar un único repositorio de referencia que representa la totalidad de la asignatura, utilizando para ello las branches del repositorio. Así, se plantea que cada branch represente una actividad diferente de la asignatura (preferiblemente, quedándose a nivel de práctica para no tener demasiadas branches), de forma que los UsuEs salten de una a otra según terminen sus prácticas. El hacerlo de esta manera permite también cumplir con la especificación **F.FORJ.EDU.5**, pues se utiliza el mismo repositorio para todas las prácticas, aunque utilizando diferentes branches.

Por tanto, los UsuDo deberían generar al principio del curso dicho repositorio de referencia junto con todas las branches necesarias, incluyendo en cada una de ellas el fichero para evaluar dicha práctica. De esta manera, se cumple con la especificación funcional **F.FORJ.EDU.2**.

7.3.2. Descripción detallada del Módulo Evaluador

El siguiente módulo que detallar es el MEv, esto es, módulo que evalúa el código. Se considera un módulo complementario del MRe, pues sólo le sirve a él, descargando el código subido para después notificar el resultado al propio MRe, que debe encargarse de cómo gestionar los siguientes pasos.

Detallando el funcionamiento interno del MEv, éste puede verse como un conjunto de contextos independientes, siendo cada uno de ellos los que van a utilizarse, en principio, en una asignatura concreta. Cabe destacar que, por mucho que se plantee así, nada impide a un UsoDo reutilizar el contexto utilizado en una asignatura para otra, siempre y cuando aquél que se reutilice ya cumpla con todos los requerimientos de la nueva asignatura. En estos casos, se recomienda hacer las mayores reutilizaciones posibles para trabajar de forma más eficiente, aunque se deja al criterio del centro la política concreta de uso de contextos.

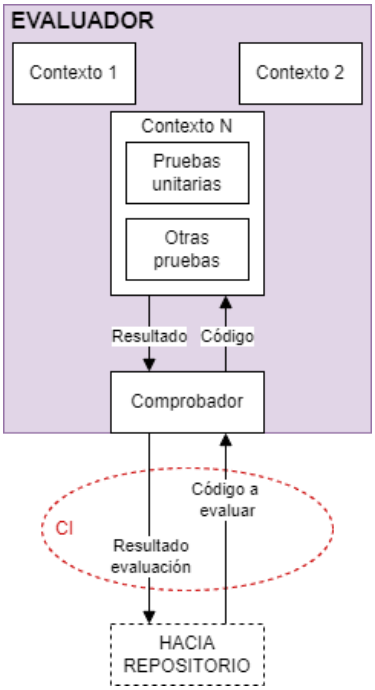

Figura 7. Diseño interno del MEv

A su vez, cada uno de estos contextos está formado por una serie de subcomponentes que representan los diferentes conjuntos de pruebas que pueden realizarse. No se considera que exista ningún conjunto de pruebas que sea estrictamente necesario realizar, pues se considera que es labor de los diferentes UsuDo saber qué aspectos son importantes evaluar en cada actividad y cuáles no. Sin embargo, sí que se recomienda que, como mínimo, se realicen pruebas unitarias al código para demostrar su correcto funcionamiento, por lo que sí que aparece representado como subcomponente aparte de Otras pruebas.

Antes de continuar con la explicación de otro componente, es el momento de explicar más en profundidad cómo se define este proceso de pruebas y cómo se generan los contextos.

Como se ha mencionado anteriormente, la aplicación de CI que se va a utilizar, GitHub Actions, utiliza unos ficheros YAML ubicados dentro de los repositorios (en concreto, dentro del directorio /.github/actions/) para definir qué pruebas se van a realizar y en qué condiciones. La definición y control de éstas se realiza mediante una serie de directivas especiales definidas por GitHub[17]. A la batería completa de pruebas (esto es, a la totalidad del fichero) se la conoce como *workflow* que, a través de la directiva *jobs*, se divide en un conjunto de jobs, que no son otra cosa que un conjunto de pruebas concretas (conocidas como *steps*, nombre que toma también la directiva que las define).

En última instancia, los steps no son otra cosa que la ejecución de comandos dentro del contexto que, en caso de ocasionar algún error (esto es, que tengan un código de retorno diferente a 0 en el caso de

[17] La sintaxis de estos ficheros YAML puede encontrarse en el siguiente enlace: https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions.

los sistemas operativos de la familia GNU/Linux), provocan la finalización del job (marcándolo como erróneo) y, con ello, la finalización con estado de error de todo el workflow. Según lo definido en la especificación funcional F.EVAL.PRU.1, esto no debe funcionar así, sino que deben poder definirse pruebas que, de no superarse, no detengan la ejecución de pruebas. Revisando la documentación, se encuentra que añadiendo la directiva *continue-on-error* a un job, se permite que continúe la ejecución del workflow pese a que dicho job falle. Así, si cada UsuDo define, en cada caso, qué pruebas son mínimas para superar la actividad y cuáles son opcionales para obtener una mejor calificación, se cumple con la especificación **F.EVAL.PRU.1**.

Como aclaración, cabe destacar que se puede configurar la ejecución de jobs de tal manera que algunos se ejecuten en paralelo y otros de manera secuencial. En general, para poder llevar un orden, se recomienda que los jobs deben ejecutarse de forma secuencial.

Para finalizar con la explicación de GitHub Actions, queda comentar cómo es el proceso de definición de contextos. Por defecto, las pruebas se ejecutan en máquinas virtuales de Ubuntu provistas por GitHub. Sin embargo, es posible definir unos contextos dentro de dichas máquinas en forma de contenedores Docker personalizados.

Para ello, y para que puedan ser utilizados por varios repositorios (que sería el caso de este sistema, donde los repositorios de los UsuEs todos utilizan el mismo contexto), se puede crear un repositorio público en GitHub formado por un fichero Dockerfile que define completamente el contenedor y cualquier cantidad de scripts que añadir al contexto para su posterior ejecución. Así, se define completamente el contexto de pruebas.

Por tanto, el UsuDo debería crear todos los contextos (junto con los scripts necesarios para realizar las pruebas pertinentes) necesarios mediante estos repositorios. Al ser estos usuarios los que conocen cuáles son las necesidades de la asignatura y su evaluación y teniendo el control del Dockerfile, podrían instalar todas las herramientas (generales o básicas para el manejo del código) que necesiten, cumpliendo así con las especificaciones **F.EVAL.PRU.2** y **F.EVAL.ASI.1**.

La selección de estos contextos se realiza en los ficheros YAML de los repositorios, a través de la directiva “uses” que se coloca dentro de un job específico. Combinándolo con la directiva *run*, se puede también ejecutar un script concreto dentro del contexto como step del job.

Así, la gestión del MEv recae completamente en los UsuDo, pues, debido a su rol en la enseñanza, son estos usuarios los que conocen la forma exacta en la que se quieren evaluar las actividades. La gestión consiste en definir previamente los contextos que van a utilizar para evaluar la asignatura, así como las pruebas a realizar a cada una de las actividades, separando aquellas obligatorias de las opcionales. Definidos los contextos, los UsuDo crean los repositorios públicos que los contienen, introduciendo en ellos los Dockerfile y scripts necesarios. Finalmente, añaden a los repositorios de referencia los ficheros YAML, definiendo el workflow separado en jobs, referenciando en cada caso al contexto y a los steps en concreto a realizar, teniendo en cuenta de si se trata o no de un job opcional.

Volviendo a la descripción del diseño interno del módulo, el otro componente a explicar es el Comprobador. Esta es la interfaz de comunicación con el MRe, recibiendo el código a evaluar que le envía su contraparte en el MRe (el componente Verificador) y devolviéndole el resultado de la evaluación. Para ello, según reciba el código debe enviárselo al contexto adecuado y esperar a recibir de él es resultado para reenviarlo al MRe.

7.3.3. Descripción detallada del Módulo Gestor

El siguiente módulo a detallar es el MGe. Se trata del único módulo que conforma el SI y, al albergar toda la lógica de automatización e integración, se convierte en el corazón del sistema NME-CV, permitiendo que éste pueda funcionar. Debido a esto, debe desarrollarse completamente.

El desarrollo debe realizarse teniendo en cuenta el diseño interno del módulo, que está conformado por varios componentes independientes que realizan funciones concretas y se comunican entre sí para poder dar solución a las especificaciones funcionales de automatización descritas anteriormente. En concreto, se distinguen los componentes Oyente, Escritor, Registrador y Núcleo.

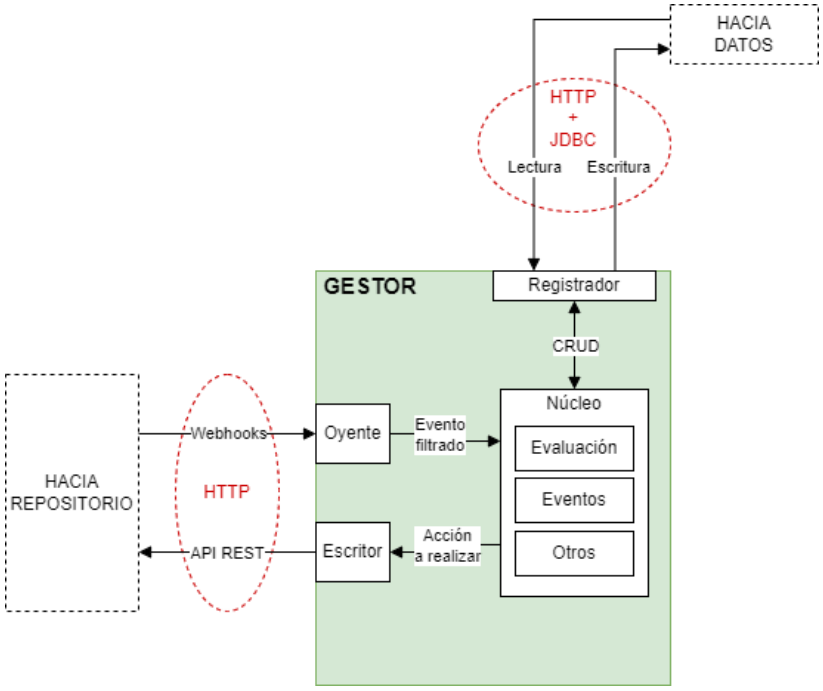

Figura 8. Diseño interno del MGe

El primero de los componentes que se estudia es el Oyente, que gestiona la integración de los webhooks en el sistema. Por un lado, éste hace las veces de receptor de webhooks, siendo el punto al que éstos se envían desde el MRe. Además de recibirlos como tal, también los filtra, obteniendo de toda la información que se envía en el JSON que forma el webhooks sólo aquella que sea relevante.

Resolver la primera de las funciones supone que el código debe configurarse de tal manera que sea capaz de servir las peticiones que lleguen a cierta URL (aquella que sea marcada como destino en la configuración de los webhooks en el MRe), iniciando el proceso de filtrado. Para ello, en el framework

de Spring se define el concepto de Controlador, que se refiere a componentes de Spring capaces de gestionar peticiones HTTP de distinto tipo. Por tanto, el Oyente se crea como un controlador Spring que sirva a la URL destino de los webhooks. Así, se cumple con la especificación **F.INTE.REP.1**.

Definido el controlador, es necesario introducir en él código capaz de realizar el filtrado de eventos. Para realizar este proceso, es necesario seguir distintos pasos:

1. El primer paso es comprobar el secret token del webhook. El secret token es una cadena de texto arbitraria que debe definirse a la hora de configurar el webhook y accesible por el receptor del webhook (esto es, por el Oyente). Cuando se envía el webhook, en la cabecera del mensaje HTTP se envía la cabecera personalizada X-GitHub-Token. Comparando ambos tokens, se puede identificar que el origen del webhook es el MRe y no es otro webhook cualquiera.
2. Comprobado el origen, el siguiente paso es comprobar qué evento ha generado el webhook, pues según el tipo de evento que sea, debe realizarse una acción u otra con dicho evento, además de que, según el evento, la información importante con la que quedarse varía (por ejemplo, en los eventos de evaluación es necesario quedarse con el resultado de la misma, información que ni siquiera existe cuando se manda un evento de creación de repositorios). El tipo de evento puede encontrarse en la cabecera HTTP personalizada X-GitHub-Event que se envía en cada webhook.
3. Comprobado el origen y el tipo de evento, sólo queda filtrar el webhook para quedarse con la información requerida y ejecutar la acción correspondiente del Núcleo.

Con la implementación del código que sea capaz de seguir estos tres pasos, se cumple con la especificación **F.INTE.REP.2**.

El siguiente de los componentes que se estudia es el Escritor, que es la contraparte del Lector dentro del MRe. El Escritor se encarga de convertir las acciones que deben realizarse en el MRe en funciones concretas (o series de funciones) de la API REST, consumiéndolas. Por tanto, este módulo debe conocer exactamente la forma en la que se deben utilizar las funciones de dicha API, de forma que se puede generar un cliente de ésta.

Existen varias opciones para implementar este cliente. La más directa, es utilizar las funciones Java que permiten utilizar consumir directamente las funciones de una API REST, escribiendo la URL exacta con todos los parámetros. Sin embargo, existe una forma más sencilla de implementarlo: utilizar librearías de terceros que implementen el cliente. Estas librerías están disponibles en la propia documentación de GitHub y convierten las llamadas a las URL específicas en métodos del lenguaje (Java en este caso). Implementando una de estas librerías se puede simplificar considerablemente el desarrollo del Escritor[18]. Al implementar la librería, se cumple con la especificación funcional **F.INTE.REP.3**.

[18] En concreto, para las pruebas, se ha utilizado la librería GitHub Jcabi que puede encontrarse en el enlace https://github.jcabi.com/.

El otro componente interfaz que se encuentra dentro del módulo es el Registrador, que hace referencia al componente encargado de gestionar la comunicación con las bases de datos, esto es, con el MDa. En general, realiza las operaciones de CRUD sobre ambas bases de datos, pero utilizando dos tecnologías diferentes, pues, como se ha visto anteriormente, el acceso a cada una se realiza mediante una tecnología diferente.

Para acceder a la DB_P, es necesario utilizar JDBC, esto es, la API de conexión con bases de datos relacionales desde Java. Para esta API existen varias implementaciones que pueden ser instaladas y configuradas de forma sencilla a través de Spring, indicando el URI de JDBC de la base de datos y credenciales. De esta forma, se consigue conectar el MGe a la DB_P.

Para conectarse a la DB_M, sin embargo, es necesario consumir su API REST (la cual se comentará más en profundidad cuando se hable del resto de módulos). Para ello, al igual que ocurría durante el desarrollo del componente Escritor, es necesario desarrollar el cliente para dicha API. En este caso, no se han encontrado librerías para Java [19] que implementen clientes, por lo que sería necesario desarrollar dicho cliente.

Combinando estas dos tecnologías, se consigue acceder a ambas bases de datos, cumpliendo con las especificaciones **F.INTE.DOC.1** y **F.INTE.DOC.2**. Además, como deben guardarse los tokens de identificación en la base de datos, el garantizar este acceso hace que también se cumpla con la especificación **F.INTE.AUT.3**.

El último de los componentes a describir es el Núcleo, llamado así por ser, precisamente, el componente principal dentro del MGe. Contiene la lógica con la que poder implementar los procesos automatizados, reaccionando a los eventos que llegan a él a través del Oyente y accionando mecanismos del Escritor y del Registrador para realizar el proceso que corresponda.

Por tanto, internamente, puede pensarse en este componente como un conjunto separado de funciones relacionadas con un aspecto a automatizar. En vista de las funciones a automatizar que se han visto con anterioridad, el diseñó actual dispone de un subcomponente para la evaluación y otro para la configuración de webhooks.

Siguiendo el funcionamiento de Spring, cada uno de estos subcomponentes sería un Servicio de Spring, de forma que, entre otras cosas, se pueda gestionar las transacciones con la base de datos, lo que hace cumplir con la especificación funcional **F.INTE.DOC.3**. Además, separándolos en distintos Servicios, se consigue un diseño más modular, en el que cada uno de los procesos no dependa del otro, lo que permite realizar modificaciones en cada uno de ellos de forma sencilla. Este diseño debe respetarse también si, a futuro, se añaden nuevas funcionalidades.

[19] En la documentación de la API REST de Moodle, se referencia a una librería con la que implementar un cliente en Java. Sin embargo, actualmente el código no se encuentra disponible debido a errores de funcionamiento en el código, por lo que no se ha podido probar. Con el suficiente tiempo, quizá vuelva a subirse, pero, ante la incertidumbre, se opta por asumir que no se va a tener acceso a la misma, evaluando así el peor caso.

En cuanto a la forma en la que se implementas esos subcomponentes:

- Para el subcomponente de automatización de la evaluación, hay que partir de la base de que al Núcleo llega la información filtrada del evento, que debe contener la identificación del UsuEs a evaluar, el nombre del repositorio al que ha subido el código y los resultados de evaluación. Con dicha información, se debe actuar de una manera u otra en función de si se ha superado o no pues, si el workflow es correcto (aunque haya jobs que no), se debe registrar la evaluación en la DB_M, utilizando la API. Además, debe escribir el mensaje de evaluación en el repositorio del UsuEs, utilizando al Escritor.
- Para el subcomponente de configuración de webhooks, en este caso, basta con que se conozca el repositorio sobre el que hay que actuar. Al ser siempre los mismos webhooks, en el código de MGe ya se podrían encontrar los webhooks a los que suscribirse, por lo que valdría con saber en qué repositorio hay que configurarlos.

Conviene recordar en este punto que, para realizar estas acciones, es necesario que el Núcleo acceda mediante el Registrador a los tokens de identificación de los UsuEs, ya sean los de GitHub o los de Moodle.

Combinando todo el diseño visto hasta, se aprecia que se cumple con las especificaciones funcionales **F.INTE.AUT.1** y **F.INTE.AUT.2**.

7.3.4. Descripción detallada del Módulo Datos

El siguiente módulo es el primero que se va a analizar dentro del SD, y no es otro que el MDo. Este módulo contiene el modelo de información al completo del sistema, donde va a almacenarse toda la información requerida para poder realizar el seguimiento de los avances de los UsuEs y los procesos automáticos del sistema.

Echando un vistazo al diseño interno del módulo, lo primero que se aprecia es que está formado por dos regiones diferentes. Estas regiones no son otra cosa que una separación entre los dos bloques que conforman el módulo, pues, como ya se ha mencionado con anterioridad, el módulo se forma a partir de dos bases de datos relacionales implementadas de formas diferentes: DB_P y DB_M, ubicadas en la Región Propia y la Región Moodle respectivamente. Además de estas bases de datos, existe un componente común denominado Controlador para realizar el control de acceso a las bases de datos.

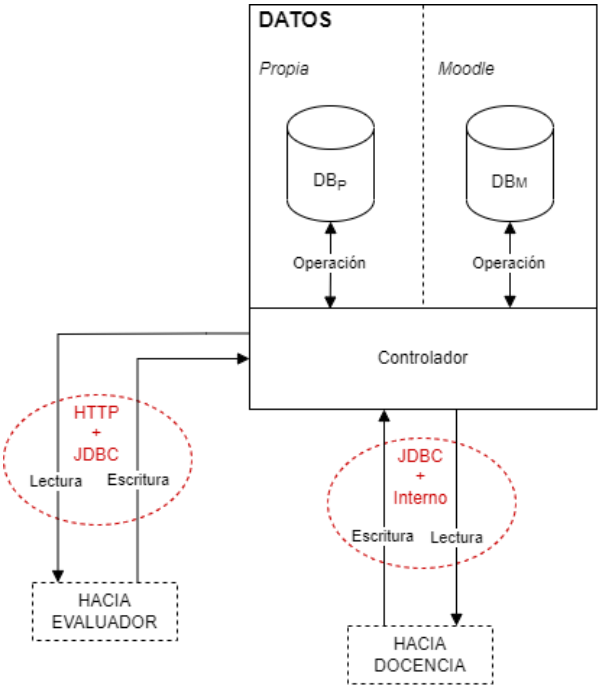

Figura 9. Diseño interno del MDa

Empezando por el componente común, éste representa a los elementos que son necesarios para realizar un control de acceso sobre las bases de datos, evitando así que usuarios no autorizados accedan a las bases de datos. La forma en la que esto se cumple es distinta según la base de datos, realizando una correcta configuración en la DB_P y aprovechando los propios sistemas de Moodle en la DB_M. Así, se cumple con la especificación funcional **F.DATO.SEC.1**.

En cuanto a las bases de datos, la primera que se va a estudiar es la DB_M. Como ya se ha mencionado anteriormente, esta base de datos es la que utiliza la instancia de Moodle para funcionar, por lo que no requiere de ninguna clase de desarrollo, sino que se integra con el resto del sistema. Al estar pensada para un LMS, esta base de datos ya soporta conceptos como asignaturas, actividades, evaluaciones o listas de estudiantes. Por tanto, esta base de datos es clave para aportar al sistema la gestión de los aspectos docentes, cumpliendo con las especificaciones funcionales relacionadas con el almacenaje de información de dicho carácter.

Debido a que la instancia de Moodle es gestionada por la administración del centro académico, que añade a cada estudiante a las aulas virtuales de las asignaturas en las que está matriculado (como es el caso de la EIB), ya se resuelven las especificaciones funcionales **F.DATO.EST.1** y **F.DATO.EST.2** sólo por utilizar dicha plataforma.

Dentro de los aspectos docentes a almacenar, está también la evaluación (tanto de actividades como final) de los UsuEs. Esto también se resuelve mediante la DB_M, que dispone de los mecanismos para almacenar dicha información. La diferencia con el caso de las listas de estudiantes radica en que, en este caso, los UsuDo sí que realizan ciertas acciones y configuraciones sobre las aulas virtuales de Moodle para poder dar soporte a la gestión de la evaluación. En concreto, deben crear actividades en

dicha plataforma que representen las que se quieren evaluar y seguir a través del sistema NME-CV. Estas actividades son evaluables desde Moodle e, incluso, se puede añadir una funcionalidad de evaluación final que realice la suma ponderada de las actividades.

La existencia de estos elementos de evaluación es suficiente para cumplir con la especificación funcional **F.DATO.EVA.2** y también podría considerarse suficiente para cumplir con la F.DATO.EVA.1. Sin embargo, para mantener los principios de automatización del sistema NME-CV, esta evaluación se escribe en las actividades de Moodle de forma automatizada. Para ello, se hace uso de la API REST de Moodle introducida previamente.

La API REST de Moodle es un servicio instalado en todas las instancias de Moodle, aunque, a diferencia de la API REST de GitHub, es necesario activarla y no tiene todas las funciones activas por defecto. En su lugar, dispone de un amplio conjunto de funciones que se recogen en servicios creados por el UsuAd. Para ello, el UsuAd crea el servicio en el panel de administración de Moodle, añadiendo sólo las funciones que sean de interés para el sistema NME-CV. Para poder cumplir con la especificación **F.DATO.EVA.1**, la función a utilizar es *core_grades_update_grades*, con la que se puede calificar una actividad de Moodle. Por tanto, con esa función, se puede añadir la evaluación obtenida por un UsuEs a la DB_M, cumpliendo con dicha especificación funcional.

Antes de continuar con la siguiente base de datos, cabe destacar que, para poder utilizar esta API de forma segura, Moodle implementa unos tokens con los que ejecutar acciones en nombre de un usuario, de la misma forma que ocurría con los tokens OAuth de GitHub. Estos tokens se relacionan con un servicio de la API REST y se dan de alta en el panel de administración de Moodle por un usuario que tenga permisos de administración (a diferencia de lo que ocurre en GitHub, que cada usuario da de alta sus propios tokens), por lo que debe ser realizado por el UsuAd. Por tanto, este usuario crea y envía los tokens a los UsuDo para que pueda ser registrado por ellos, cumpliendo así con la especificación funcional **F.DATO.SEC.2**.

La otra base de datos que se encuentra en este módulo es la DB_P. Esta hace referencia a una base de datos relacional de desarrollo propio que contiene toda la información no puede proveer la DB_M, de forma que se integre con el resto de los módulos. Esta información es, sobre todo, aquella que pueda relacionar a los UsuEs de la DB_M con sus cuentas en GitHub y los repositorios en los que resuelve cada actividad de cada práctica.

Para ello, la DB_P contiene las tablas necesarias para esta relación, obteniendo la lista de estudiantes de cada asignatura de Moodle a través de la función *core_enrol_get_enrolled_users* (que debe incluirse en el servicio de la API REST) y relacionándolo con la lista de estudiantes y asignaturas que introduzca el UsuDo desde el módulo Docencia (como se explicará más adelante). Así, se cumple con las especificaciones funcionales **F.DATO.EST.4** y **F.DATO.EVA.3**.

El otro aspecto a tener en cuenta en esta base de datos es que, en aras de que se relacione a cada UsuDo con su información y poder realizar procesos de control de acceso desde el MDo (se explicará más adelante), es necesario que exista un perfil de docente dentro de la DB_P. Este perfil contiene tanto

la información que identifique al UsuDo como su información de login. Al existir esta última, hay que tener en cuenta que es necesario almacenar contraseñas, que deberán estar almacenadas de forma segura. Para ello, se almacena la contraseña tras pasar por un proceso de hash en lugar de la contraseña en claro. Con esto se finaliza con el cumplimiento de las especificaciones funcionales del modelo datos, con las especificaciones **F.DATO.EST.3** y **F.DATO.SEC.3** cumplidas.

7.3.5. Descripción detallada del Módulo Docencia

El último módulo a explicar es el MDo, considerado el módulo principal del SD. Como ya se ha mencionado al introducir el módulo, está dividido entre una implementación de Moodle y una aplicación de desarrollo propio, por lo que este módulo también requiere de un cierto desarrollo.

Internamente, el módulo se encuentra dividido en dos regiones: Propia y Moodle. Cada una de ellas hace referencia a la aplicación que representa y dispone de un diseño diferente para cada caso. Mientras que el primero basa sus componentes en los necesarios para el despliegue de una aplicación web, el segundo se ve internamente como el conjunto independiente de asignaturas que utilizan el sistema NME-CV, esto es, como el reflejo de los componentes del de asignatura del MRe, pero con una orientación docente en este caso. En cualquier caso, ambas regiones acaban comunicando con un mismo componente interfaz llamado Registrador, cuya función es idéntica a la descrita en el componente homónimo del MGe, por lo que no se va a explicar en profundidad este componente[20].

[20] Como breve apunte, aunque la comunicación con la DB_P se haga con JDBC, hay que tener en cuenta que la comunicación con la DB_M la realiza Moodle directamente, sin necesidad de utilizar la API REST que se utilizaba en el caso del MGe. Por tanto, el Registrador no se implementaría exactamente igual que en dicho módulo, pues se eliminaría el cliente de la API REST. Sin embargo, funcionalmente se consideran iguales puesto que, sea cual sea la tecnología, ambos componentes gestionan ambas bases de datos.

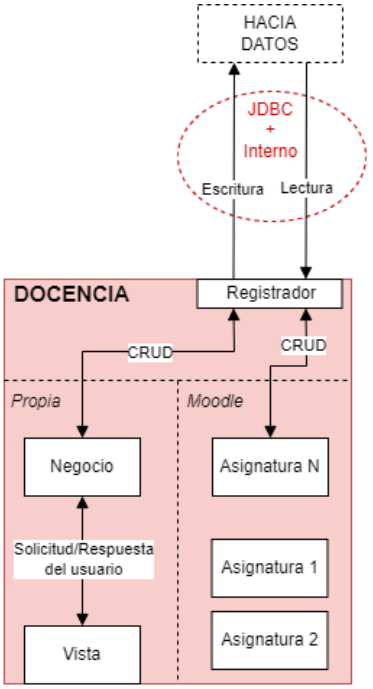

Figura 10. Diseño interno del MDo

Empezando por la región Propia, como se ha mencionado, dispone de una aplicación web de desarrollo propio que debe cumplir con las necesidades de registros en la DB_P, en concreto con aquellos que requieran la intervención de una persona al no generarse automáticamente durante el funcionamiento del sistema. Éstos son los descritos en las tres especificaciones funcionales de registros de información de aspecto docente: F.DOCE.REG.1, F.DOCE.REG.2 y F.DOCE.REG.3.

Existen muchas formas de diseñar esta aplicación, pero, al tratarse de una aplicación web que debe permitir a varios usuarios poder modificar y gestionar una base de datos relacional, se plantea utilizar el patrón de diseño MVC en tres capas (capa de presentación, capa de negocio y capa de datos), utilizando para ello el framework Spring. Gracias a esta decisión, se simplifica el despliegue de la aplicación y se facilitan procesos tan importantes en estos escenarios como es la implementación de un contenedor de transacciones que pueda gestionar las transacciones con la base de datos, el login en la aplicación o el servicio de múltiples tipos de vista y gestión de pantallas de error.

Puesto que, según se ha definido, el componente registrador cumpliría con las funciones propias de la capa de datos, la región se conforma con los componentes Vista y Negocio, que hacen referencia a las capas que implementan. Por tanto, para cumplir con las especificaciones funcionales previamente descritas, se implementa una vista que permita a un UsuDo subir una lista de UsuEs en formato CSV[21] (cumpliendo con la especificación funcional **F.DOCE.REG.1**), otra que permita introducir el texto con

[21] La elección de este formato no es casual, sino que se aprovecha que, desde GitHub Classroom puede descargarse un fichero CSV que contiene la lista que relaciona a los UsuDo con su nombre de cuenta en GitHub y el repositorio en el que resuelve la actividad. Por tanto, el UsuDo descargaría ese fichero y lo subiría a la base de datos a través de esta vista.

el que contestar a las pruebas de código de los UsuDo (cumpliendo con la especificación funcional **F.DOCE.REG.2**), otra en la que un UsuDo registre sus tokens (cumpliendo con la especificación funcional **F.DOCE.REG.3**) y otra que registre las asignaturas de cada UsuDo y las actividades que la conforman (cumpliendo la especificación funcional **F.DOCE.REG.4**). El componente Negocio incluye los métodos necesarios para recibir la información del componente Vista y registrarlo, no sin antes realizar las comprobaciones necesarias sobre la información que se va a registrar y la que ya existe en la base de datos.

Aunque no corresponda a ninguna especificación funcional, también se destaca la necesidad de una vista que muestre un formulario de login para que el UsuDo se identifique antes de poder utilizar la aplicación, controlando el acceso a la misma. El componente Negocio dispone de los métodos para permitir este login.

En cuanto a la región de Moodle, ésta se encuentra formada por la implementación de la plataforma homónima[22], compuesta por las asignaturas a las que se realiza el seguimiento mediante el sistema NME-CV. En esta solución, se considera que no debe utilizarse la implementación de Moodle sólo para su uso en el sistema, sino que se integra su uso con el que ya se le diera en el entorno académico. Esto quiere decir que, en el caso de la EIB, no se despliega una nueva implementación de Moodle ni se añaden nuevas asignaturas, sino que se sigue utilizando aquello de lo que ya se dispone.

Para completar la integración y dar solución a las especificaciones funcionales, las asignaturas se completan con la creación de actividades evaluables que, tras ser registradas, sirven como lugar en el que almacenar las evaluaciones de los UsuEs. Las evaluaciones quedan registradas en sus asignaturas, de forma que cada estudiante sólo puede ver su evaluación, asegurando el cumplimiento de la especificación funcional **F.DOCE.EVL.1**. Utilizando además la funcionalidad de calificación final de Moodle, que suma las calificaciones de cada actividad, se consigue también la evaluación final de cada UsuEs.

En cuanto a la especificación **F.DOCE.LIS.1**, basta con el uso de Moodle para cumplirla, pues ya trae incorporada una vista con la lista completa de estudiantes dentro de la asignatura. Al poder consultarla, ya se da la especificación por cumplida.

7.4. Despliegue de la arquitectura

Descritos los módulos, lo siguiente es describir cómo se realiza el despliegue de la arquitectura. El despliegue propuesto es sencillo ya que, al haber aprovechado infraestructura ya existente, se eliminan muchas necesidades de este despliegue. De esta forma, se despliegan únicamente aquellos elementos que es necesario desarrollar.

Concretando más, hay que realizar despliegues para los módulos MGe, MDo y MDa, aunque el primero de ellos es el único del que hay que realizar un despliegue completo, al ser el único módulo con

[22] En el caso de la EIB, esta implementación es eGela.

desarrollo completo. De los otros dos módulos, sólo es necesario desplegar las partes desarrolladas, esto es, las regiones Propias de ambos módulos[23].

En aras de reducir la cantidad de máquinas necesarias, se propone realizar el despliegue utilizando una única máquina que contenga los tres módulos. Para ello, se definen los puertos por los que acceder a las interfaces y eligiendo el 80 para aplicación del MDo y uno arbitrario para el Oyente del MGe. No es necesario que la DB_P sea accesible desde el exterior, pues los módulos que deben comunicarse con ella están en la misma máquina.

Como detalle final, cabe destacar que la máquina debe estar correctamente securizada para evitar problemas de seguridad con la misma. El plan para evitar dichos problemas se deja al criterio de cada centro académico.

7.5. Conclusiones del diseño

Finalmente, como conclusión del diseño que se aporta en este documento, se quiere mencionar que, aunque se haya hecho hincapié en plataformas concretas y en su aplicación en el contexto de la EIB, esta solución es lo suficientemente abierta como para ser aplicada en otros escenarios con otras plataformas que dispongan de las funcionalidades requeridas (aunque podría traer desarrollos adicionales). Esto se debe a que se ha realizado la definición de las especificaciones funcionales y de los componentes que conforman los módulos en los que no se ha tenido tanto en cuenta la plataforma a usar como lo que esta realiza.

Por ejemplo, esta misma arquitectura soporta que se utilice GitLab en lugar de GitHub[24], teniendo para ello en cuenta que habría que realizar el desarrollo del componente Creador del MRe y que habría que realizar el despliegue de las instancias de GitLab y de los runners que sirven de contexto de pruebas. En el MGe, también sería necesario modificar los componentes Oyente y Escritor para adaptarlos a las especificaciones de los webhooks en GitLab y de su API REST. Como puede apreciarse, estas modificaciones no se relacionan con la definición de los módulos o de sus componentes en sí, sino con la implementación de los mismos. Se concluye, por tanto, la posibilidad de aplicar la arquitectura a varios escenarios.

[23] Al no existir una instancia global de Moodle, sino que se basa en instancias propias, se podría argumentar es necesario que exista un despliegue de esta plataforma y, por tanto, que es necesario desplegar los dos módulos al completo. Sin embargo, no se considera así, pues se parte de la base de que el centro educativo ya dispone de una instancia de Moodle (como es el caso de la EIB) y sólo se está reaprovechando de lo que ya se dispone, por lo que no sería necesario desplegarlo en concreto para el sistema. Si el centro educativo no dispusiera de la plataforma (o cualquiera con las mismas características), sí que sería necesario realizar el despliegue.

[24] Durante buena parte del desarrollo de este trabajo, se valoró utilizar GitLab en lugar de GitHub, pues, al momento de empezar, GitLab ya disponía de una aplicación de CI integrada, mientras que GitHub la incluiría más adelante. Parte de la definición de los módulos y de los prototipos demostradores se hicieron teniendo en mente utilizar dicha plataforma, por lo que, de paso, se demostró que podría ser utilizada con otras plataformas.

8. Plan de trabajo y presupuesto

En este apartado se muestra el plan de trabajo que habría sido necesario para realizar este trabajo junto con su presupuesto. Para ello, además de los resultados en sí, se aporta la descripción de los paquetes de trabajo y sus tareas, así como un cronograma con la organización.

Para ambos apartados, se han definido 4 figuras involucradas en el desarrollo:

- **Ingeniero Sénior (IS)**. Esta figura representa a un ingeniero de más experiencia, encargado principalmente de plantear diseños de más alto nivel y de supervisar parte del trabajo del resto de figuras. Además, también se responsabiliza de la validación de los módulos del sistema.
- **Ingeniero Junior (IJ)**. Esta figura representa a un ingeniero de menos experiencia, encargado principalmente de plantear diseños de más bajo nivel y del resto de labores de ingeniería.
- **Técnico (TE)**. Esta figura representa a una persona de perfil más técnico, responsable de los desarrollos y su validación. Aun así, participa como apoyo en otras partes del trabajo.
- **Supervisor (SU)**. Esta figura representa a la persona encargada de supervisar los avances del trabajo. Su trabajo se reduce a participar en reuniones de seguimiento.

8.1. Plan de trabajo

Como se ha mencionado anteriormente, el plan de trabajo es, en verdad, un hipotético. Es el plan de trabajo que se hubiera realizado si se hubiera dispuesto de los conocimientos que se tienen actualmente. En la realidad, el proyecto ha pasado por varias fases diferentes, pues, para empezar, durante casi todo el desarrollo del trabajo, se planteó el uso de GitLab, volviendo a GitHub sólo en las fases finales. Por ello, se invirtió mucho tiempo en realizar investigaciones, prototipos y experiencias sobre una plataforma que, finalmente, fue descartada, descartando con ella los prototipos o necesitando nuevas versiones. Además, la existencia del a API de Moodle (que desencadenó en el uso de Moodle, modificando la idea que se tenía en un principio sobre todo el SD) también fue descubierta hacia el final del desarrollo, por lo que sólo se ha podido comprobar lo más básico para demostrar que la solución es viable. La fecha de inicio tampoco coincide con la real, pues, debido a motivos laborales, el trabajo se ha visto completamente parado en un par de ocasiones.

Aun así, se considera de mayor interés plantear así el trabajo, pues, si se hiciera con el plan real, se vería una amalgama de tareas que no llevan a ningún sitio o que se dilatan en el tiempo de manera absurda.

Completada la aclaración, se vuelve al plan de trabajo propuesto en este documento. Según éste, se necesitan 1902 horas para completar el todo el trabajo, empezando con la formación en septiembre de 2019 y finalizando con la entrega del documento final al principio de marzo de 2022. Los meses de agosto se consideran como no hábiles.

En los siguientes subapartados, se desglosa el plan de trabajo en varios paquetes y actividades, finalizando con el cronograma que muestre cómo se reparten las tareas.

8.1.1. Descripción de paquetes de trabajo

Para organizar el trabajo, se han definido una serie de paquetes de trabajo que agrupan tareas relacionadas. Además, cada paquete de trabajo produce una unidad de entrega, que representa al conjunto de entregables de cada tarea del paquete de trabajo. Estos entregables pueden ser de tipos muy diferentes (documentos, maquetas, diseños…).

En concreto, los paquetes de trabajo son los siguientes:

PT0 Investigación y formación tecnológica

Este primer paquete de trabajo engloba todas tareas relacionadas con la formación de las figuras para que puedan acometer con las tareas que vendrán en paquetes de trabajo posteriores. Este proceso consiste en la realización de tutoriales o cursos con los que puedan familiarizarse con las tecnologías que tendrán que utilizar a futuro. La designación de 0 deriva precisamente de eso, porque se considera que este paquete de trabajo como uno que es necesario para la organización interna pero que no se ofertaría a un cliente final, pues es labor de formación del equipo. La duración del paquete es de 136 horas, repartidas entre septiembre y noviembre de 2019.

Este paquete produce como unidades de entrega las guías de uso de las tecnologías, que deberán realizarse para guiar los diseños y desarrollos posteriores.

En las siguientes tablas, se muestran las tareas y las unidades de entrega que conforma este paquete de trabajo.

Código	*Tarea*
T001	Uso avanzado de aplicación de CI
T002	Recepción y filtrado de webhooks
T003	Implementaciones de clientes de API REST de GitHub
T004	Uso de API REST de Moodle

Tabla 17. Tareas del PT0

Código	*Tarea*
UE001	Guía de uso de GitHub Actions con ejemplos
UE002	Guía de uso y recepción de webhooks en GitHub con ejemplos
UE003	Guía de uso de API REST de GitHub con ejemplos
UE004	Guía de uso de API REST de Moodle con ejemplos

Tabla 18. Unidades de entrega del PT0

PT1 Definición de requerimientos y especificaciones

Entrando ya el PT1, se encuentra el paquete en el que se realiza la definición de las problemáticas que se encuentran a la hora de buscar una solución técnica al sistema y las especificaciones que ésta deberá cumplir. Por tanto, en este paquete de trabajo se realizan los primeros pasos en el desarrollo tanto del análisis de viabilidad técnica como del diseño final propuesto. Cabe destacar que en este paquete de trabajo todavía no se pone solución a las problemáticas, sino que el trabajo se limita a la identificación de alternativas. Antes de hallar la solución, es necesario seguir con el procedimiento hasta desarrollar los prototipos demostradores. Es por ello por lo que en

este paquete de trabajo también se desarrolla el plan de pruebas con el que se tomará la decisión final. La duración del paquete es de 276 horas repartidas entre diciembre de 2019 y junio de 2020.

Este paquete de trabajo produce como unidades de entrega la identificación de problemáticas y alternativas, así como el plan de pruebas con el que especificar y desarrollar los prototipos.

En las siguientes tablas, se muestran las tareas y las unidades de entrega que conforma este paquete de trabajo.

Código	*Tarea*
T101	Problemáticas
T102	Identificación de alternativas y criterios
T103	Especificaciones de diseño
T104	Especificaciones funcionales
T105	Definición del plan de pruebas

Tabla 19. Tareas del PT1

Código	*Tarea*
UE101	Definición de problemáticas de diseño
UE102	Definición de alternativas y criterios
UE103	Listado preliminar de las especificaciones de diseño
UE104	Listado preliminar de las especificaciones funcionales
UE105	Plan de pruebas

Tabla 20. Unidades de entrega del PT1

PT2 Diseño de la arquitectura

Este paquete de trabajo comienza cuando ya ha pasado el suficiente tiempo en el anterior como para tener una idea clara de cuáles son las problemáticas encontradas y la mayoría de las especificaciones a cumplir, por lo que ya se dispone de la información suficiente para iniciar el desarrollo de la arquitectura que soporte la solución final (aunque este paquete sólo se centra en el alto nivel). A la par que se empieza con este proceso, también es necesario comenzar a definir los prototipos de validación para cumplir con las especificaciones, por lo que dicha tarea se incluye en este paquete de trabajo. Precisamente, el desarrollo de estos prototipos es lo que va a marcar el final del diseño, pues, hasta que no se haya alcanzado cierto punto de del desarrollo de prototipos, no se pueden terminar de tomar decisiones sobre el diseño de alto nivel de la arquitectura. La duración del paquete es de 258 horas, repartidas entre abril de 2020 y mayo de 2021.

Este paquete produce como unidades entregables las definiciones de los distintos aspectos de la arquitectura, además del listado de prototipos a realizar.

En las siguientes tablas, se muestran las tareas y las unidades de entrega que conforma este paquete de trabajo.

Código	*Tarea*
T201	Módulos que conforman el sistema
T202	Comunicaciones entre módulos
T203	Usuarios en cada módulo
T204	Tablas y requerimientos de la base de datos
T205	Definición de prototipos de validación

Tabla 21. Tareas del PT2

Código	*Tarea*
UE201	Definición de la estructura básica de la arquitectura
UE202	Definición de comunicación entre módulos
UE203	Definición de interfaces de usuarios con el sistema
UE204	Estructura básica y definición de requerimientos de la base de datos
UE205	Listado de prototipos de validación a desarrollar

Tabla 22. Unidades de entrega del PT2

PT3 Diseño interno de los módulos

Durante el diseño de alto nivel, ya se puede empezar con el siguiente paso, que es diseñar internamente los módulos que conforman la arquitectura, de forma que se pueda completar la solución propuesta. En este paquete de trabajo, se definen como tareas el diseño interno de cada módulo, teniendo en cuenta que este no se puede dar por cerrado hasta que el diseño esté completado. Para ello, es necesario que los prototipos estén desarrollados y aprobados, pues las decisiones que se tengan que tomar para las problemáticas son las que van a terminar de definir el diseño a este nivel. Aprovechando estas labores de diseño, también se pasa a especificar formalmente los prototipos, diseñándolos con exactitud ahora que se conocen todos los niveles del diseño. Este paquete tiene una duración de 320 horas repartidas entre mayo de 2020 y enero de 2022, siendo el paquete que más se dilata en el tiempo.

Este paquete produce como unidades entregables las definiciones completas de cada módulo de la arquitectura y de los prototipos demostradores que desarrollar.

En las siguientes tablas, se muestran las tareas y las unidades de entrega que conforma este paquete de trabajo.

Código	*Tarea*
T301	Módulo Repositorio
T302	Módulo Evaluador
T303	Módulo Gestor
T304	Relaciones y atributos de tablas de base de datos
T305	Módulo Datos
T306	Módulo Docencia
T307	Especificación formal de prototipos

Tabla 23. Tareas del PT3

Código	*Tarea*
UE301	Definición completa del Módulo Repositorio
UE302	Definición completa del Módulo Evaluador
UE303	Definición completa del Módulo Gestor
UE304	Diagrama EER de la base de datos
UE305	Definición completa del Módulo Datos
UE306	Definición completa del Módulo Docencia
UE307	Definición completa de los prototipos

Tabla 24. Unidades de entrega del PT3

PT4 Desarrollo y depuración de prototipos

Cuando ya se tienen las primeras versiones de los diseños internos de los módulos de la arquitectura y alguno de los prototipos formalmente especificados, se puede comenzar con el proceso de desarrollo de los prototipos. Este paquete de trabajo se define como el central del plan de trabajo. El desarrollo final del desarrollo de los prototipos es lo que marca la propuesta final de diseño, desarrollo que sólo finalizará una vez se validen, de forma que se de pie al realizar varias iteraciones hasta llegar a la solución óptima. Además del desarrollo de los prototipos, es necesario comenzar con una primera tarea que prepare los entornos de desarrollo. Este paquete tiene una duración de 476 horas repartidas entre noviembre de 2020 y enero de 2022.

Este paquete produce como unidades entregables los resultados que se obtienen al desarrollar los prototipos.

En las siguientes tablas, se muestran las tareas y las unidades de entrega que conforma este paquete de trabajo.

Código	*Tarea*
T401	Instalación de servicios y preparación de entornos
T402	Prototipo de contexto de evaluación sencilla (P.EVAL)
T403	Prototipo de reporte de prueba en repositorio (P.INTEG)
T404	Prototipo de cliente de API REST de Moodle (P.MAPI)
T405	Prototipo de implementación de base de datos (P.DBPR)

Tabla 25. Tareas del PT4

Código	*Tarea*
UE401	Guía de uso del entorno de prototipos
UE402	Resultados del prototipo de contexto de evaluación sencilla
UE403	Resultados del prototipo de reporte de prueba en repositorio
UE404	Resultados del prototipo de cliente de API REST de Moodle
UE405	Resultados del prototipo de implementación de base de datos

Tabla 26. Unidades de entrega del PT4

PT5 Integración de prototipos

En este paquete de trabajo se agrupan las tareas en las que, partiendo de los desarrollos de prototipos, éstos se integran entre sí, en busca de acabar montando una pequeña maqueta del sistema completo que demuestre al menos una de las funcionalidades del sistema completo. De

esta manera, se consigue terminar de definir el diseño interno de los módulos, puliendo las interfaces de comunicación entre ellos. Precisamente por eso, este paquete también es necesario para finalizar con el desarrollo de prototipos, por lo que se empieza durante ese proceso, integrándolos por subsistemas a los que pertenece el prototipo. Además, mediante la maqueta, se demuestra también que la arquitectura puede ser integrada, demostrando su viabilidad. La duración de este paquete es de 162 horas repartidas entre febrero de 2021 y enero de 2022.

Este paquete produce como unidades entregables las diferentes maquetas que se forman como consecuencia del proceso de integración entre módulos.

En las siguientes tablas, se muestran las tareas y las unidades de entrega que conforma este paquete de trabajo.

Código	*Tarea*
T501	Integración del Subsistema Código
T502	Integración del Subsistema Docencia
T503	Integración con el Subsistema Integración

Tabla 27. Tareas del PT5

Código	*Tarea*
UE501	Maqueta del Subsistema Código
UE502	Maqueta del Subsistema Docencia
UE503	Maqueta final

Tabla 28. Unidades de entrega del PT5

PT6 Validación de la solución adoptada

Para poder dar por finalizado los prototipos y, con ello, el análisis de viabilidad técnica y el diseño, es necesario llegar al punto en el que éstos se validen. Este paquete de trabajo recoge las tareas de validación de los distintos módulos, poniendo fin a su diseño y a la integración entre ellos pues, al fin y al cabo, al validar un prototipo se está validando el diseño de los distintos módulos. Si se dispone del diseño validado de todos los módulos, también se dispone de la información suficiente para la selección de la solución a las problemáticas, por lo que también se incluye dicha tarea. Este paquete es, por tanto, el último del desarrollo del trabajo (sin contar con la documentación del mismo), poniendo fin a todos los paquetes de trabajo que hayan quedado abiertos. La duración de este paquete es de 172 horas repartidas entre julio de 2021 y enero de 2022.

Este paquete produce como unidades entregables los informes que contienen el diseño completo de los módulos, así como el análisis de viabilidad técnica completado.

En las siguientes tablas, se muestran las tareas y las unidades de entrega que conforma este paquete de trabajo.

Código	*Tarea*
T601	Validación del Módulo Repositorio
T602	Validación del Módulo Evaluador
T603	Validación del Módulo Gestor
T604	Validación del Módulo Datos
T605	Validación del Módulo Docencia
T606	Selección de soluciones a las problemáticas

Tabla 29. Tareas del PT6

Código	*Tarea*
UE601	Informe de diseño final del Módulo Docencia
UE602	Informe de diseño final del Módulo Repositorio
UE603	Informe de diseño final del Módulo Evaluador
UE604	Informe de diseño final del Módulo Gestor
UE605	Informe de diseño final del Módulo Datos
UE606	Análisis de viabilidad técnica

Tabla 30. Unidades de entrega del PT6

PT7 Documentación y gestión

El último paquete de trabajo recoge las tareas de documentación del trabajo, incluyendo la realización del presupuesto al inicio del trabajo. Este paquete empieza con el PT1 y es el último en acabar, pues las labores de documentación están presentes durante todo el trabajo y sólo se pone fin una vez se entrega el documento final. La duración de este paquete es 102 horas repartidas entre diciembre de 2019 y marzo de 2022.

Este paquete produce como unidades entregables el presupuesto y el documento final del trabajo.

En las siguientes tablas, se muestran las tareas y las unidades de entrega que conforma este paquete de trabajo.

Código	*Tarea*
T701	Realización del presupuesto
T702	Documentación del proyecto y entregables

Tabla 31. Tareas del PT7

Código	*Tarea*
UE801	Presupuesto del trabajo
UE802	Documento final del trabajo

Tabla 32. Unidades de entrega del PT7

Además de los paquetes de trabajo, también se han definido los siguientes hitos que se recogen en la tabla inferior.

Código	*Hito*	*Fecha*
H1	Inicio de la formación	Inicio de septiembre de 2019
H2	Inicio del trabajo	Inicio de diciembre de 2019
H3	Inicio del desarrollo de prototipo	Inicio de diciembre de 2020
H4	Integración del SC	Final del septiembre de 2021
H5	Integración del SD	Final del noviembre de 2021
H6	Maqueta completa	Final de enero de 2022
H7	Fin de desarrollo, diseño y análisis de viabilidad técnica	Final de enero de 2022
H8	Final del trabajo	Inicio de marzo de 2022

Tabla 33. Hitos del proyecto

8.1.2. Cronograma y resumen del plan de trabajo

En este subapartado, se aporta el cronograma con la organización de las tareas a realizar cada mes. Así mismo, se aportan dos tablas resumen:

- Una que contiene las horas dedicadas a cada tarea y paquete de trabajo, desglosando las horas por figuras y marcando en negrita en cada tarea qué figura es la responsable. Al final de la tabla, también se aporta cuántas horas en total dedica cada figura y el total de horas dedicadas.
- En la otra tabla, se encuentre el resumen de las unidades entregables asociadas a cada paquete de trabajo y tarea.

Debido al amplio tamaño del cronograma y de las tablas, estas se muestran en las siguientes páginas. El cronograma se muestra en tablas separadas por cada paquete de trabajo para facilitar su lectura.

No hábil

No hábil

		2019				2020												2021												2022		
		Sep	*Oct*	*Nov*	*Dic*	*Ene*	*Feb*	*Mar*	*Abr*	*May*	*Jun*	*Jul*	*Ago*	*Sep*	*Oct*	*Nov*	*Dic*	*Ene*	*Feb*	*Mar*	*Abr*	*May*	*Jun*	*Jul*	*Ago*	*Sep*	*Oct*	*Nov*	*Dic*	*Ene*	*Feb*	*Mar*
PT0	**Investigación y formación tecnológica**																															
T001	Uso avanzado de aplicación de CI																															
T002	Recepción y filtrado de webhooks																															
T003	Implementaciones de clientes de API REST de GitHub																															
T004	Uso de API REST de Moodle																															

Tabla 34. Cronograma del PT0

No hábil

No hábil

		2019				2020											2021												2022		
		Sep	*Oct*	*Nov*	*Dic*	*Ene*	*Feb*	*Mar*	*Abr*	*May*	*Jun*	*Jul*	*Ago*	*Sep*	*Oct*	*Nov*	*Dic*	*Ene*	*Feb*	*Mar*	*Abr*	*May*	*Jun*	*Jul*	*Ago*	*Sep*	*Oct*	*Nov*	*Dic*	*Ene*	*Feb*	*Mar*
PT1	**Definición de requerimientos y especificaciones**																														
T101	Problemáticas																														
T102	Identificación de alternativas y criterios																														
T103	Especificaciones de diseño																														
T104	Especificaciones funcionales																														
T105	Definición del plan de pruebas																														

Tabla 35. Cronograma del PT1

No hábil

No hábil

		2019				2020												2021												2022		
		Sep	*Oct*	*Nov*	*Dic*	*Ene*	*Feb*	*Mar*	*Abr*	*May*	*Jun*	*Jul*	*Ago*	*Sep*	*Oct*	*Nov*	*Dic*	*Ene*	*Feb*	*Mar*	*Abr*	*May*	*Jun*	*Jul*	*Ago*	*Sep*	*Oct*	*Nov*	*Dic*	*Ene*	*Feb*	*Mar*
PT2	**Diseño de la arquitectura**																															
T201	Módulos que conforman el sistema																															
T202	Comunicación entre módulos																															
T203	Usuarios en que hacen uso de cada módulo																															
T204	Tablas y requerimientos de la base de datos																															
T205	Definición de prototipos de validación																															

Tabla 36. Cronograma del PT2

		2019				2020											2021											2022		
		Sep	*Oct*	*Nov*	*Dic*	*Ene*	*Feb*	*Mar*	*Abr*	*May*	*Jun*	*Jul*	*Ago*	*Sep*	*Oct*	*Nov*	*Dic*	*Ene*	*Feb*	*Mar*	*Abr*	*May*	*Jun*	*Jul*	*Ago*	*Sep*	*Oct*	*Nov*	*Dic*	*Ene*	*Feb*	*Mar*
PT3	**Diseño interno de los módulos**																															
T301	Módulo Repositorio																															
T302	Módulo Evaluador																															
T303	Módulo Gestor																															
T304	Relaciones y atributos de tablas de base de datos																															
T305	Módulo Datos																															
T306	Módulo Docencia																															
T307	Especificación formal de prototipos																															

No hábil

No hábil

Tabla 37. Cronograma del PT3

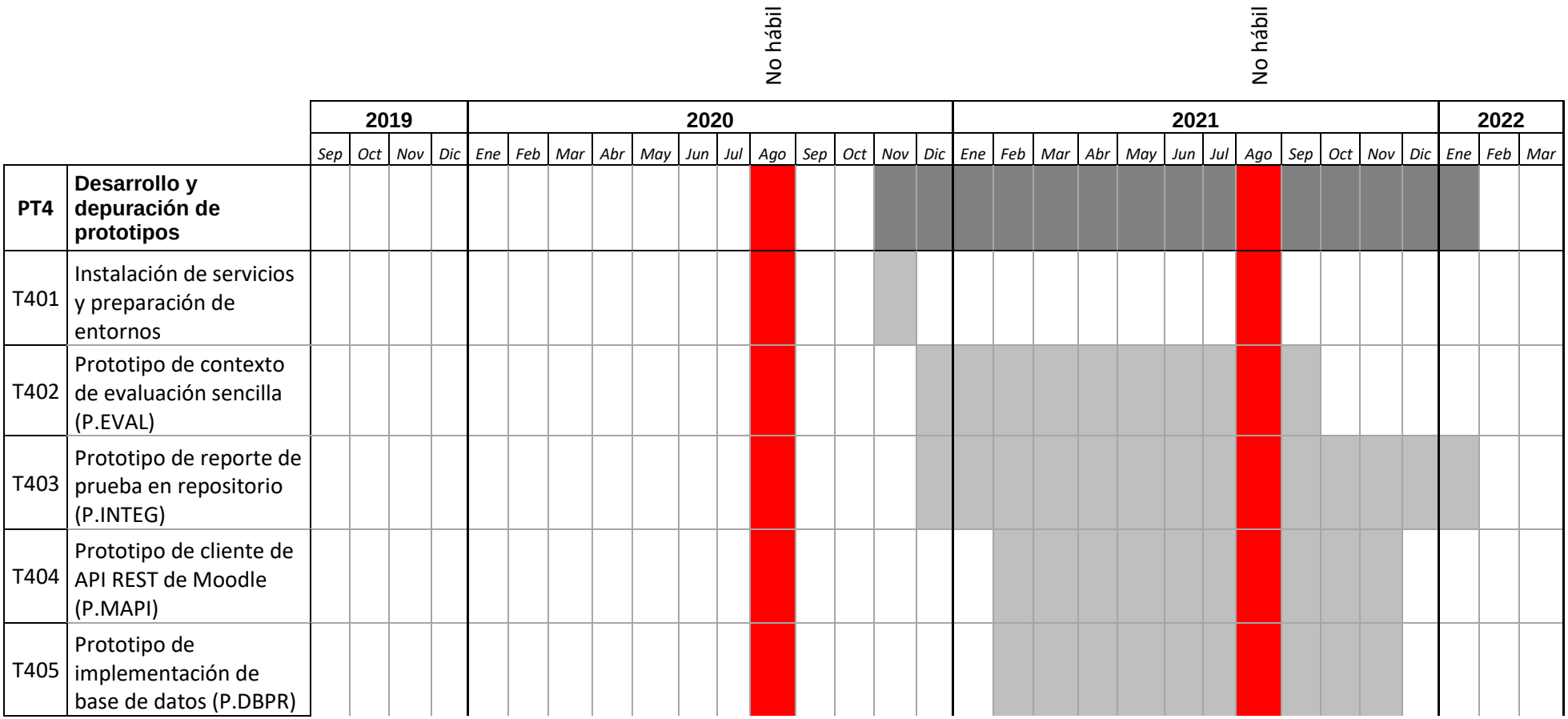

Tabla 38. Cronograma del PT4

No hábil

No hábil

		2019				2020												2021												2022		
		Sep	*Oct*	*Nov*	*Dic*	*Ene*	*Feb*	*Mar*	*Abr*	*May*	*Jun*	*Jul*	*Ago*	*Sep*	*Oct*	*Nov*	*Dic*	*Ene*	*Feb*	*Mar*	*Abr*	*May*	*Jun*	*Jul*	*Ago*	*Sep*	*Oct*	*Nov*	*Dic*	*Ene*	*Feb*	*Mar*
PT5	**Integración de prototipos**																															
T501	Integración del Subsistema Código																															
T502	Integración del Subsistema Docencia																															
T503	Integración con el Subsistema Integración																															

Tabla 39. Cronograma del PT5

No hábil

No hábil

		2019				2020												2021												2022		
		Sep	*Oct*	*Nov*	*Dic*	*Ene*	*Feb*	*Mar*	*Abr*	*May*	*Jun*	*Jul*	*Ago*	*Sep*	*Oct*	*Nov*	*Dic*	*Ene*	*Feb*	*Mar*	*Abr*	*May*	*Jun*	*Jul*	*Ago*	*Sep*	*Oct*	*Nov*	*Dic*	*Ene*	*Feb*	*Mar*
PT6	**Validación de la solución adoptada**																															
T601	Validación del Módulo Repositorio																															
T602	Validación del Módulo Evaluador																															
T603	Validación del Módulo Gestor																															
T604	Validación del Módulo Datos																															
T605	Validación del Módulo Docencia																															
T606	Selección de soluciones a las problemáticas																															

Tabla 40. Cronograma del PT6

		2019				2020											2021											2022		
		Sep	*Oct*	*Nov*	*Dic*	*Ene*	*Feb*	*Mar*	*Abr*	*May*	*Jun*	*Jul*	*Ago*	*Sep*	*Oct*	*Nov*	*Dic*	*Ene*	*Feb*	*Mar*	*Abr*	*May*	*Jun*	*Jul*	*Ago*	*Sep*	*Oct*	*Nov*	*Dic*	*Ene*	*Feb*	*Mar*
PT7	**Documentación y gestión**																															
T701	Realización del presupuesto																															
T702	Documentación del proyecto y entregables																															
		H1			H2												H3									H4		H5		H6		H8
																														H7		

No hábil

No hábil

Tabla 41. Cronograma del PT7 (junto con los hitos)

Código	Nombre	Horas IS	Horas IJ	Horas TE	Horas SU	Horas totales
PT0	***Investigación y formación tecnológica***	***0***	***80***	***56***	***0***	***136***
T001	Uso avanzado de aplicación de CI	0	**32**	8	0	40
T002	Recepción y filtrado de webhooks	0	16	**16**	0	32
T003	Implementaciones de clientes de API REST de GitHub	0	16	**16**	0	32
T004	Uso de API REST de Moodle	0	16	**16**	0	32
PT1	***Definición de requerimientos y especificaciones***	***132***	***112***	***20***	***12***	***276***
T101	Problemáticas	**20**	16	4	2	42
T102	Identificación de alternativas y criterios	16	**32**	4	4	56
T103	Especificaciones de diseño	**24**	16	4	2	46
T104	Especificaciones funcionales	**32**	16	4	2	54
T105	Definición del plan de pruebas	**40**	32	4	2	78
PT2	***Diseño de la arquitectura***	***112***	***92***	***40***	***14***	***258***
T201	Módulos que conforman el sistema	**24**	24	8	4	60
T202	Comunicación entre módulos	16	**16**	8	2	42
T203	Usuarios en que hacen uso de cada módulo	**16**	16	8	2	42
T204	Tablas y requerimientos de la base de datos	**24**	12	8	2	46
T205	Definición de prototipos de validación	**32**	24	8	4	68
PT3	***Diseño interno de los módulos***	***128***	***144***	***28***	***20***	***320***
T301	Módulo Repositorio	16	**16**	4	2	38
T302	Módulo Evaluador	16	**16**	4	2	38
T303	Módulo Gestor	24	**32**	4	4	64
T304	Relaciones y atributos de tablas de base de datos	16	**32**	4	4	56
T305	Módulo Datos	8	**8**	4	2	22
T306	Módulo Docencia	16	**16**	4	2	38
T307	Especificación formal de prototipos	32	**24**	4	4	64
PT4	***Desarrollo y depuración de prototipos***	***64***	***168***	***224***	***20***	***476***
T401	Instalación de servicios y preparación de entornos	0	0	**12**	0	12
T402	Prototipo de contexto de evaluación sencilla (P.EVAL)	0	40	**40**	4	84
T403	Prototipo de reporte de prueba en repositorio (P.INTEG)	32	52	**84**	8	176
T404	Prototipo de cliente de API REST de Moodle (P.MAPI)	16	36	**40**	4	96
T405	Prototipo de implementación de base de datos (P.DBPR)	16	40	**48**	4	108
PT5	***Integración de prototipos***	***36***	***72***	***48***	***6***	***162***
T501	Integración del Subsistema Código	8	**8**	8	2	26
T502	Integración del Subsistema Docencia	8	**24**	16	2	50
T503	Integración con el Subsistema Integración	20	**40**	24	2	86
PT6	***Validación de la solución adoptada***	***64***	***96***	***0***	***12***	***172***
T601	Validación del Módulo Repositorio	**8**	16	0	2	26
T602	Validación del Módulo Evaluador	**8**	16	0	2	26
T603	Validación del Módulo Gestor	**16**	24	0	2	42
T604	Validación del Módulo Datos	**8**	16	0	2	26
T605	Validación del Módulo Docencia	**16**	16	0	2	34
T606	Selección de soluciones a las problemáticas	**8**	8	0	2	18
PT7	***Documentación y gestión***	***28***	***48***	***20***	***6***	***102***
T701	Realización del presupuesto	8	8	0	2	18
T702	Documentación del proyecto y entregables	20	40	20	4	84
	HORAS TOTALES DEL TRABAJO	***564***	***812***	***436***	***90***	***1902***

Tabla 42. Resumen de tareas con desglose de horas por figura

Código	Unidad Entregable	PT/T Relacionada
UE0	***Guías de uso de las diferentes tecnologías***	***PT0***
UE001	Guía de uso de GitHub Actions con ejemplos	T001
UE002	Guía de uso y recepción de webhooks en GitHub con ejemplos	T002
UE003	Guía de uso de API REST de GitHub con ejemplos	T003
UE004	Guía de uso de API REST de Moodle con ejemplos	T004
UE1	***Identificación de problemáticas y alternativas***	***PT1***
UE101	Definición de problemáticas de diseño	T101
UE102	Definición de alternativas y criterios	T102
UE103	Listado preliminar de las especificaciones de diseño	T103
UE104	Listado preliminar de las especificaciones funcionales	T104
UE105	Plan de pruebas	T105
UE2	***Definición de la arquitectura***	***PT2***
UE201	Definición de la estructura básica de la arquitectura	T201
UE202	Definición de comunicación entre módulos	T202
UE203	Definición de interfaces de usuarios con el sistema	T203
UE204	Estructura básica y definición de requerimientos de la base de datos	T204
UE205	Listado de prototipos de validación a desarrollar	T205
UE3	***Definiciones completas de los módulos***	***PT3***
UE301	Definición completa del Módulo Repositorio	T301
UE302	Definición completa del Módulo Evaluador	T302
UE303	Definición completa del Módulo Gestor	T303
UE304	Diagrama EER de la base de datos	T304
UE305	Definición completa del Módulo Datos	T305
UE306	Definición completa del Módulo Docencia	T306
UE307	Definición completa de los prototipos	T307
UE4	***Resultados de prototipos***	***PT4***
UE401	Guía de uso del entorno de prototipos	T401
UE402	Resultados del prototipo de contexto de evaluación sencilla	T402
UE403	Resultados del prototipo de reporte de prueba en repositorio	T403
UE404	Resultados del prototipo de cliente de API REST de Moodle	T404
UE405	Resultados del prototipo de implementación de base de datos	T405
UE5	***Maquetas basadas en prototipos***	***PT5***
UE501	Maqueta del Subsistema Código	T501
UE502	Maqueta del Subsistema Docencia	T502
UE503	Maqueta final	T503
UE6	***Informes de diseño final***	***PT6***
UE601	Informe de diseño final del Módulo Docencia	T601
UE602	Informe de diseño final del Módulo Repositorio	T602
UE603	Informe de diseño final del Módulo Evaluador	T603
UE604	Informe de diseño final del Módulo Gestor	T604
UE605	Informe de diseño final del Módulo Datos	T605
UE606	Análisis de viabilidad técnica	T606
UE7	***Documentación final***	***PT7***
UE701	Presupuesto del trabajo	T701
UE702	Documento final del trabajo	T702

Tabla 43. Resumen de unidades entregables

8.2. Presupuesto

A continuación, se muestra el presupuesto necesario para acometer el proyecto. Cabe destacar que este presupuesto se ha valorado en base a lo que se cobraría a un cliente por el trabajo realizado en el plan anterior, por lo que el PT0 no se ha presupuestado.

La primera de las partidas es referida a cada paquete de trabajo. Para calcularla, se tiene en cuenta las horas dedicadas a la realización del paquete (junto con el desglose de horas descrito anteriormente) junto con los costes por hora de dedicación de cada una de las figuras que intervienen en el trabajo[25].

Figura	*Coste (€/h)*
IS	60
IJ	40
TE	20
SU	30

Tabla 44. Costes horarios de cada figura

Concepto	*Nº de horas*	*Total*
PT1	276	13.160,00€
PT2	258	11.620,00€
PT3	320	14.600,00€
PT4	476	15.640,00€
PT5	162	6.180,00€
PT6	172	8.040,00€
PT7	102	4.180,00€
	Subtotal	***73.420,00€***

Tabla 45. Partida de paquetes de trabajo

La siguiente partida recoge las amortizaciones imputables al trabajo. En esta partida, se imputa la amortización de un servidor que se compra para dar servicio a un grupo de investigación, compartiendo el uso del servidor entre varios proyectos entre los que se incluye este trabajo. Dicho servidor es el que alberga los módulos de desarrollo propio a desplegar. Los datos para calcular la amortización del servidor son los siguientes:

- **Precio del servidor**: 7680€
- **Vida útil**: 4 años (48 meses)
- **Número de proyectos**: 10

Por tanto, para amortizarlo, se llega a que sería necesario cobrar 16€/mes a cada proyecto que utilice el servidor. Teniendo en cuenta que el servidor sería necesario durante 15 meses (desde noviembre de 2020 hasta enero de 2022, esto es, la duración del PT4), la partida de amortización toma la forma que puede verse en la siguiente tabla.

[25] Consultar el Anexo IV para un mayor desglose de esta partida.

Concepto	Precio de uso mensual	Uso mensual	Total
Servidor	16€/mes	15 meses	240,00€
		Subtotal	***240,00€***

Tabla 46. Partida de amortizaciones

La última partida recoge el resto de los gastos realizados que, en este caso, corresponden con el uso continuado de Amazon Web Services, utilizado para experiencias piloto antes de utilizar el servidor de la anterior partida.

Concepto	Total	
Amazon Web Services	50,00€	
	Subtotal	***50,00€***

Tabla 47. Partida de gastos

Definidas las partidas, sólo queda calcular el presupuesto final, al que hay que añadir el 21% de IVA.

Concepto	Coste total	
Paquetes de trabajo	73.420,00€	
Amortizaciones	240,00€	
Gastos	50,00€	
	Subtotal	***73.710,00€***
IVA (21%)	15.479,10€	
	TOTAL	***89.189,10€***

Tabla 48. Presupuesto total

8. Conclusiones

En el estudio técnico anterior no sólo se definió el sistema NME-CV, sino que, además, se definieron qué tecnologías la harían posible, aportando una arquitectura de referencia. Sin embargo, antes de poder continuar con el proyecto, era necesario profundizar más en lo anteriormente propuesto.

A lo largo de este documento, se ha presentado el análisis técnico de viabilidad realizado al sistema NME-CV, cuyo resultado se considera positivo al haber llegado a una solución que resuelve las problemáticas y cumple con las especificaciones generales y funcionales que se ha propuesto. Para este análisis, a diferencia de lo que se hizo en el estudio, se han realizado pequeños prototipos que han permitido tomar decisiones basadas en los resultados de las experiencias, aportando criterio a las mismas.

Además, junto con el análisis, se ha diseñado una arquitectura mucho más detallada y completa de aquella de referencia aportada en el estudio técnico. Esta arquitectura, además, ya no es sólo aplicable a unas plataformas y contexto en concreto, sino que puede aprovecharse en otros escenarios en los que se haya decidido utilizar otras plataformas. Además, el disponer de la arquitectura se traduce en progreso para la implementación, que ahora toma una forma mucho más específica y ve sus primeros pasos cumplidos gracias a los prototipos, que pueden ser reutilizados como base para el desarrollo final de los módulos.

Todo esto supone un gran avance para el proyecto, que se encuentra ahora ante las puertas de la realización de un piloto funcional que demuestre sus bondades y encamine al sistema hacia su implementación final. Llegar a ese punto aporta beneficios técnicos y económicos, es cierto, pero no se debe perder el foco que, más importante que esos, la implementación de este sistema trae consigo importantes beneficios sociales, influyendo en la mejora del proceso educativo, formando a mejores profesionales y allanando el terreno tanto a estudiantes como docentes.

9. Referencias

https://docs.github.com/en/developers/webhooks-and-events/webhooks/about-webhooks
https://docs.github.com/en/rest
https://docs.github.com/en/actions
https://docs.gitlab.com/ee/user/project/integrations/webhooks.html#issues-events
https://docs.gitlab.com/ee/api/
https://docs.gitlab.com/ee/ci/
https://about.gitlab.com/install/
https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions
https://github.jcabi.com/
https://github.com/gitlab4j/gitlab4j-api
https://docs.moodle.org/dev/Web_service_API_functions
https://docs.moodle.org/dev/Creating_a_web_service_client
https://docs.moodle.org/all/es/Usando_servicios_web
https://www.udemy.com/course/gitlab-ci-pipelines-ci-cd-and-devops-for-beginners/
https://www.udemy.com/course/desarrollo-web-con-spring-framework-4-de-cero-a-ninja/
https://www.udemy.com/course/test-unitarios-con-junit-curso-de-introduccion/
https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/ResponseEntity.html
https://www.baeldung.com/spring-rest-http-headers
https://www.baeldung.com/spring-response-entity
https://docs.docker.com/
https://www.oauth.com/oauth2-servers/access-tokens/

ANEXO I: Análisis y especificaciones funcionales detalladas

En este anexo, se detallan las especificaciones funcionales que se han enumerado en el apartado de *Análisis funcional*.

Especificaciones funcionales de la forja

En el apartado de *Análisis de viabilidad* técnica, se han definido varias especificaciones funcionales que deben ser cumplidas por la forja. En concreto se definieron las siguientes:

- F.FORJ.EVE Detección de eventos (sobre la necesidad de detectar los eventos que ocurran en los repositorios de los UsuEs).
- F.FORJ.COM Interfaz de comunicación (sobre la necesidad de poder interactuar de forma externa y programática con los repositorios de los UsuEs en aras de dotar de automatización al sistema).
- F.FORJ.ICO Compatibilidad con aplicaciones de CI (sobre la necesidad de poder utilizar aplicaciones de CI dentro de los repositorios para así evaluar el código de los UsuEs).
- F.FORJ.EDU Adecuación al entrono académico (sobre la necesidad de que la forja presente herramientas para facilitar su uso en entornos académico).

Tomando estas especificaciones funcionales, es posible definir unas de más bajo nivel. Para ello, es necesario identificar las funciones concretas que son necesarias para el correcto funcionamiento del sistema y la integración de módulos.

Empezando por el grupo de detección de eventos, se pueden definir qué eventos en concreto deben escuchar los webhooks de GitHub para que funcione el sistema. En concreto, se han identificado los siguientes:

F.REPO.EVE.1 Detección del evento de creación de repositorios

Tal y como funciona GitHub Classroom, el repositorio que se crea para los UsuEs no es otra cosa que el resultado de realizar un fork sobre el repositorio de referencia provisto por el UsuDo. El problema radica en que, cuando se realiza un fork en GitHub, lo único que se clona de un repositorio a otro es el contenido, no la configuración, dentro de la cual se encuentran los eventos a los que escuchan los webhooks para ser lanzados. Tenerlos correctamente configurados es clave para el correcto funcionamiento del sistema. Por tanto, es necesario encontrar una forma automatizada para que los repositorios de los UsuEs se creen con ellos configurados.

Para comprender la solución propuesta para este problema, primero es necesario mencionar la existencia de los webhooks de organización que ofrece GitHub. Éstos no son otra cosa que los mismo webhooks que se utilizan en los repositorios de GitHub, pero aplicados a una organización (como las que son necesarias para que funcione GitHub Classroom). Así, lo que se escucha no son los eventos ocurridos dentro de un repositorio, sino que se escuchan los que ocurren dentro de la propia organización.

Con esto en mente, se debería configurar la organización que se utilice para GitHub Classroom para que escuchara los eventos de creación de repositorios. Así, se podría lanzar, a través de la interfaz de comunicación que constituye la API REST de GitHub, acciones para configurar los webhooks en los repositorios creados a través de GitHub Classroom.

F.REPO.EVE.2 Detección del evento de evaluación

Otro de los eventos que es necesario escuchar para que funcione el sistema, y que por tanto constituyen una especificación funcional de este grupo, es el evento que produce haber finalizado la evaluación del código que haya subido un UsuEs a su repositorio. Este evento es necesario para poder detectar un nuevo intento de resolución por parte del UsuEs y para almacenar su el resultado de la evaluación, de forma que, a posteriori, pueda calcular su calificación final.

F.REPO.EVE.3 Detección del evento de finalización de prueba concreta

En relación con la anterior especificación, sería interesante que se detectara no sólo la evaluación completa del código, si no también qué pruebas en concreto se han superado y cuáles no. Detectar este evento supone poder conocer los detalles de la realización de las pruebas, pudiendo usar a posteriori esta información para realizar una mejor evaluación de los UsuEs.

En cuanto al grupo de especificaciones de la interfaz de comunicación, se pueden obtener más especificaciones funcionales especificando qué funciones en concreto se deben poder realizar mediante la API REST que presenta GitHub. Las funciones identificadas son:

F.REPO.COM.1 Configuración del detector de eventos

La primera de las funciones que deben realizarse a través de la API que se ha identificado es una que se ha introducido con anterioridad. En concreto, se ha mencionado en la descripción de la especificación funcional F.FORJ.EVE.1.

En dicha especificación, se ha mencionado que se necesitaba escuchar el evento de creación de repositorios para poder configurar correctamente los webhooks de los repositorios de los UsuEs. Esta acción se realiza a través de la API REST de GitHub, por lo que se define esta especificación funciona F.FORJ.COM.1 como la necesidad de que a través de una o más funciones de la API REST pueda ajustarse la configuración de los repositorios de los UsuEs de tal forma que sus webhooks escuchen los eventos descritos en las especificaciones funcionales del grupo de la detección de eventos.

F.REPO.COM.2 Escritura de resultados de evaluación en repositorios

Según se ha descrito en la solución de la cuarta problemática del apartado de *Análisis de viabilidad técnica*, GitHub va a ser utilizado por los UsuEs como interfaz para consultar los resultados de las pruebas de evaluación del código que han subido a sus repositorios. Para ello, se utiliza el sistema de issues de GitHub, escribiendo el resultado en uno de ellos.

Teniendo en cuenta esto, se plantea como especificación funcional la posibilidad de escribir issues en los repositorios de los alumnos en nombre del profesor que lleven consigo el resultado (positivo o negativo) de la evaluación de código. Además de la propia evaluación, cabe destacar que la issue deberá contener a su vez información relativa al momento en el que se evaluó, siendo, en su forma más simple, un sello de tiempo (*timestamp*) del momento de escritura de la issue.

F.REPO.COM.3 Ejecución de acciones en nombre de docente

Según el funcionamiento de la API REST de GitHub, para realizar acciones sobre los repositorios de GitHub, es necesario autenticarse de alguna manera, de forma que se sepa quién es el usuario que ha realizado la acción. Esto es importante, por ejemplo, cuando se manejan repositorios privados, pues es necesario saber si el usuario tiene autorización para realizar dicha acción.

En el sistema NME-CV, estas acciones con la API REST deben realizarse en nombre de uno de los UsuDo responsables, utilizando para ello alguna clase de token que lo identifique. Esto se debe a que tener un único token para todo un grupo resulta más sencillo de almacenar, gestionar y revocar. Estos tokens deben poder generarse desde el Módulo Repositorio y ser asociados al usuario de la forja del UsuDo correspondiente.

Siguiendo con el grupo de la aplicación de CI, se pueden especificar las características que debe tener la aplicación de CI utilizada. En concreto, se han identificado:

F.REPO.ICO.1 Máxima integración con la forja

La aplicación de CI que se utilice para la evaluación del código de los UsuEs debe poder ser capaz de integrarse con la forja escogida (GitHub en este caso) de la mayor manera posible. Esto es así para evitar añadir todavía más elementos e interfaces al sistema.

Lo ideal sería que los resultados de las pruebas pudieran consultarse directamente en la forja sin tener que pasar por otras plataformas.

F.REPO.ICO.2 Mínimo despliegue para la aplicación

En relación con lo anterior, sería necesario que el utilizar la aplicación de CI no supusiera un gran despliegue para poder albergarla. En este caso, lo ideal sería poder ahorrarse el despliegue de la aplicación, utilizando para ello una aplicación cuyo despliegue lo haya realizado un tercero y permita su uso externo.

Finalizando con las especificaciones funcionales del grupo de la forja ya descritas en anteriores apartados, se encuentra el grupo de adecuación al entorno académico, dentro del cual se han identificado las siguientes especificaciones:

F.REPO.EDU.1 Creación de repositorios para todo el grupo de estudiantes

Lo que se pide a la forja es que sea capaz de crear varios repositorios que sean a su vez forks de uno de referencia para poder crear los repositorios de los UsuEs de la forma más dinámica posible. Esto se debe a las necesidades de un entorno académico, en el que todos los UsuEs deben disponer de repositorios propios pero idénticos al del resto que puedan usar para resolver los mismos problemas. Aunque esto mismo se puede resolver de forma manual, dado el volumen de UsuEs que puede haber en un mismo grupo, debería poder hacerse de formas más sencillas, de cara a evitar errores y a hacer más eficiente el trabajo de los UsuDo.

F.REPO.EDU.2 Repositorio de referencia

En relación con la anterior especificación, es necesario que, previamente, los UsuDo hayan creado un repositorio de referencia sobre el que poder realizar un fork que cree los repositorios de los UsuEs. Este repositorio debe poder utilizarse para resolver al menos una de las actividades de la asignatura, aunque se premia el poder utilizar un mismo repositorio de referencia para representar varias actividades, de forma que se reduzca el volumen de repositorios requeridos para impartir una asignatura, simplificando así su gestión. En el caso ideal, un repositorio de referencia representaría la totalidad de una asignatura.

F.REPO.EDU.3 Privacidad de los repositorios

Al tratarse de repositorios aplicados en un entorno académico, se debe asegurar que exista la posibilidad de que éstos sean privados, de tal forma que se restrinja el acceso a un repositorio al UsuEs propietario, además de al UsuDo responsable. De esta manera, se impide que uno UsuEs pueda plagiar el trabajo de otro usando los repositorios.

Cabe destacar que esto no supone la obligación de que los todos los repositorios de UsuEs deban ser privados, pues algún UsuDo podría preferir utilizar repositorios públicos según su método didáctico. Más bien, la especificación se limita a que exista la posibilidad de que puedan ser privados, para así cumplir con la especificación general sobre protección frente a copias.

F.REPO.EDU.4 Organización de estudiantes en grupos

Otra especificación de este grupo es la necesidad de que todos los UsuEs se encuentren organizados dentro de algún grupo en la forja. El motivo por el que esto es necesario es porque facilita el encontrar a otros estudiantes dentro de la forja.

F.REPO.EDU.5 Separación entre prácticas dependientes e independientes

A la hora de enseñar programación, existe la posibilidad de que se desarrollen prácticas independientes (esto es, que no tienen una continuidad) o dependientes (esto es, que cada una empieza donde acaba la anterior). Así como en el primer caso se podrían llegar a utilizar repositorios separados para cada práctica, en el segundo debería utilizarse el mismo

repositorio para así poder aprovechar todas las ventajas de los VCS. El sistema debe poder dar soporte a ambos.

Especificaciones funcionales de los contextos y aplicación de evaluación

Concretadas las especificaciones funcionales referidas a la forja, se pasa ahora a comentar las relativas al funcionamiento de la aplicación de CI y los contextos de pruebas. Del apartado anterior, se conocen:

- F.EVAL.PRU Compatibilidad con múltiples tipos de pruebas (sobre la necesidad de que el contexto sea capaz de utilizar herramientas que permitan pruebas más allá de las unitarias).
- F.EVAL.ASI Compatibilidad con múltiples asignaturas (sobre la necesidad de que el contexto se adapte a las necesidades de cada asignatura a evaluar).

En cuanto a las especificaciones relativas a los tipos de pruebas, se han definido las siguientes especificaciones funcionales de más bajo nivel:

F.EVAL.PRU.1 Soporte para pruebas opcionales

A la hora de probar el código en un entorno profesional, tiene sentido que se considere a toda la batería fallida cuando una de las pruebas falla, pues, de cara a poder publicar la build, el programa debe funcionar correctamente. Sin embargo, en un entorno académico, esto no tiene tanto sentido, pues la evaluación puede ser algo más que un resultado binario (10 si funciona, 0 si no).

De cara a poder realizar una evaluación al código subido por los UsuEs de una forma que se adecúe más a un entorno académico, sería interesante que la aplicación de CI tuviera soporte para poder implementar pruebas opcionales. Esto supondría separar, por un lado, las pruebas que se consideran mínimos para dar por aprobada la actividad y pruebas que, de no superarse, no bloquean al UsuEs en sus avances por la asignatura. Por tanto, estas pruebas funcionarían de tal manera que, de superarse, supongan un aumento en la calificación de la actividad pero que, de no hacerlo, no se consideraría un suspenso por parte del UsuEs.

F.EVAL.PRU.2 Instalación sencilla de herramientas generales

A veces no basta con las herramientas que vienen instaladas en los contenedores para poder realizar las pruebas al código, sino que es necesario instalar herramientas adicionales. Si sólo se realizan pruebas unitarias, puede que sí sea suficiente, pero si se quiere acceder a otros tipos (como la comprobación de estilo), es capital tener la posibilidad de instalar herramientas adicionales en los contenedores. Por ello, es necesario que en el diseño existan formas sencillas de instalar estas herramientas.

Indagando más en la compatibilidad con múltiples asignaturas, se pueden definir unas nuevas dentro de ella, obtenidas al analizar las necesidades del contexto para adecuarse a múltiples asignaturas.

F.EVAL.ASI.1 Instalación de herramientas para manejar diferentes lenguajes

Para que esta arquitectura pueda ser utilizada en varias asignaturas, debe tenerse en cuenta, primeramente, que el contexto debe disponer de todas las herramientas necesarias para poder manejar el código de los UsuEs. Como en cada asignatura puede estar utilizándose un lenguaje de programación distinto, debe asegurarse de que existe una manera sencilla de instalar las herramientas necesarias para cada uno de ellos. Ejemplos de estas herramientas son compiladores o intérpretes.

Especificaciones funcionales sobre el modelo de datos

En cuanto a las especificaciones sobre el modelo de datos planteado en el apartado anterior, se han recogido las siguientes especificaciones funcionales en apartados anteriores:

- F.DATO.EST Registro del perfil de estudiantes (sobre la necesidad de agilizar el proceso de registro de estudiantes en el modelo de datos).
- F.DATO.EVA Registro de resultados de evaluación (sobre la necesidad de almacenar la información suficiente para poder calcular la evaluación de las actividades por cada estudiante).
- F.DATO.SEC Seguridad de los datos almacenados (sobre la necesidad de proteger la información sensible contenida).

Buscando especificaciones funcionales de más bajo nivel en el grupo sobre el registro del perfil de estudiantes, se ha encontrado lo siguiente.

F.DATO.EST.1 Registro automático de estudiantes

La primera de las especificaciones funcionales dentro de este grupo se centra en la necesidad de que el perfil de UsuEs se rellene de la forma más automatizada posible, esto es, sin que haya necesidad de introducirlos uno a uno en el sistema. Esta especificación se define para seguir cumpliendo con las especificaciones generales que hablan de automatización y eficiencia, pues la introducción uno a uno supondría atentar contra ellas.

F.DATO.EST.2 Identificación de asignatura y grupo

El perfil de UsuEs debe contener no sólo la información suficiente para identificar a la persona concreta, sino que también debe contener aquella que permita al sistema ubicarlo dentro del contexto académico.

Esta información es, primeramente, qué asignaturas que utilicen el sistema NME-CV está cursando. Gracias a esta información, el UsuDo de una cierta asignatura podría acceder únicamente a los UsuEs de su asignatura, facilitando su búsqueda.

Otra información relevante para el contexto académico es en qué grupo de prácticas se encuentra en cada asignatura. Ésta debe existir por el mismo motivo que la información de asignatura, pero aplicada a los distintos grupos de prácticas en los que se pueden dividir los UsuEs.

F.DATO.EST.3 Perfil de docente

Hasta ahora, se ha centrado la atención en la gestión de perfiles de estudiantes en la base de datos. Para poder gestionarlos correctamente, es necesario que coexistan perfiles de docente, de forma que se pueda relacionar a los UsuEs con su UsuDo responsable en cada caso.

F.DATO.EST.4 Relación con la cuenta en la forja

Por último, el perfil de UsuEs debe contener también el nombre de la cuenta que utiliza en la forja. Esto se debe a que, cuando lleguen los webhooks con información relativa a un evento, se va a enviar también dicho nombre de usuario. Esta información, al ser única en GitHub, puede ser utilizada para identificar a un usuario concreto, si se ha relacionado previamente con un perfil de estudiante concreto.

En resumen, cuando se quiera identificar al usuario que ha originado un evento en la forja, se recogería el nombre de usuario registrado en la información del evento y éste se compararía con los registrados para los distintos UsuEs, de forma que se encuentre cuál lo ha generado.

En cuanto al grupo sobre la evaluación se han encontrado nuevas especificaciones funcionales al definir qué información de evaluación en concreto debe almacenarse. Las especificaciones funcionales internas son las indicadas a continuación.

F.DATO.EVA.1 Asociación de cada evaluación con su resolución

La primera de las especificaciones funcionales de este grupo consiste en que el modelo de datos debe estar diseñado de tal manera que se pueda identificar correctamente la resolución de una actividad por parte de un UsuEs con la evaluación que le corresponde. Esto es necesario de cara a poder realizar el seguimiento de los avances de un UsuEs durante el desarrollo de una asignatura, sabiendo qué actividades ha completado y con qué evaluación.

F.DATO.EVA.2 Cálculo de la evaluación final

Otra de las especificaciones funcionales a cumplir dentro de este grupo es que debe contener la información suficiente para poder calcular información final al final del curso. Para ello, es necesario conocer no sólo la información comentada en la especificación funcional F.DATO.EVA.1, sino que también debe ser capaz de almacenar los pesos de cada actividad (pues podrían ser distintos) y relacionarlos con cada actividad.

De esta manera, el proceso del cálculo de evaluación final consistiría en recuperar las evaluaciones registradas en las actividades resueltas por los UsuEs y sumarlas aplicando a cada una de ellas el peso correspondiente.

F.DATO.EVA.3 Relación entre asignaturas y actividades con la forja

Para poder calcular la evaluación final, es necesario que el modelo de datos permita relacionar las asignaturas y sus actividades con sus representaciones dentro de la forja. Por tanto, es necesario diseñar dicha relación.

Por último, en cuanto al grupo sobre la seguridad del modelo de datos, se han definido las siguientes especificaciones funcionales.

F.DATO.SEC.1 Control de acceso al modelo de datos

En el modelo de datos se almacena información considerada sensible, como es el caso de la evaluación. Ninguna persona que no sea un UsuDo o el propio UsuEs debería poder acceder a dicha información, por lo que una falta de control de acceso al modelo provocaría que, por ejemplo, otros UsuEs accedieran a esa información. Por tanto, se debe garantizar que nadie accede a dicha información sensible.

F.DATO.SEC.2 Consulta y modificación en nombre de docente

De la misma forma que en la especificación funcional F.FORJ.COM.3 se definía la necesidad de realizar acciones sobre los repositorios en nombre de un UsuDo, el sistema también requiere de realizar acciones sobre la base de datos de Moodle en nombre de los UsuDo. Así, debe existir alguna clase de token que, de forma segura, permita realizar estas acciones.

F.DATO.SEC.3 Almacenamiento de información sensible

En estas bases de datos se almacena información sensible, como los token o información de login en las aplicaciones de la aplicación de docencia. Así, la base de datos debe garantizar que estas se almacenan de forma segura, para evitar filtraciones.

Especificaciones funcionales de los aspectos docentes

El último de los grupos de especificaciones funcionales que se ha visto en el apartado de "Análisis de viabilidad técnica" han sido las relacionadas con la interfaz de los usuarios. En concreto, se han visto:

- F.DOCE.LIS Lista de estudiantes (sobre la necesidad de disponer de una lista ordenada de los UsuEs asignados a uno o varios UsuDo).
- F.DOCE.EVL Anuncio de resultados de evaluación (sobre la necesidad de que cada UsuEs pueda acceder a su evaluación).
- F.DOCE.REE Acceso rápido a repositorios de estudiantes (sobre la necesidad de que los UsuDo puedan acceder de forma sencilla a los repositorios de los UsuEs).

Desgranando el primero de los grupos, se pueden definir nuevas especificaciones funcionales en base a las necesidades de esta lista de UsuEs.

F.DOCE.LIS.1 Lista separada por asignaturas

La lista de estudiantes debe estar separada por asignaturas, de tal forma que la lista aparezca ya filtrada y cada UsuDo vea las listas de UsuEs de sus asignaturas. Esto facilita la gestión por parte del UsuDo, que encuentra a sus estudiantes de forma rápida y sencilla.

En cuanto a las nuevas especificaciones funcionales del grupo sobre el anuncio de la evaluación, se han definido las siguientes.

F.DOCE.EVL.1 Privacidad de los resultados de evaluación

Sean cuales sean los resultados de evaluación de un UsuEs, el sistema debe garantizar que sólo las personas interesadas (el UsuEs concreto y cualquier UsuDo responsable de la asignatura) tengan acceso a la misma. Esto se hace para proteger la privacidad de cada UsuEs, pues se considera tal calificación como confidencial.

El último de los grupos a estudiar es del grupo del acceso rápido a repositorios, donde se ha definido la siguiente especificación funcionales:

F.DOCE.REE.1 Relación entre estudiante y repositorio en grupo concreto

Para poder realizar el acceso rápido, es necesario que el UsuDo pueda acceder a una lista que le relacione a cada estudiante (a través de cualquier identificador) con la URL hacia su repositorio. Esta relación tiene que aparecer de forma ordenada, debiendo estar filtrada por grupo (identificando con ello la asignatura).

Cabe destacar que la definición de los anteriores grupos se hizo durante la definición de una problemática que, pese a estar relacionado con la docencia, no abarcaba la definición completa de las funcionalidades que se deben cumplir para cubrir en su totalidad los aspectos docentes. Para poder cumplirlo, se define ahora un nuevo grupo de especificaciones funcionales.

- F.DOCE.REG Registro de información (sobre la necesidad de poder registrar información en el sistema de forma manual).

Dentro del primer grupo se encuentran varias especificaciones funcionales de menor nivel.

F.DOCE.REG.1 Registro de información adicional de estudiantes

Para el funcionamiento del sistema, debe existir cierta información adicional de los UsuEs además de la almacenada en Moodle. En concreto, se necesita que se guarde qué repositorio están utilizando en qué asignatura, de forma que a la hora de evaluar se pueda saber a ciencia cierta en qué asignatura se están evaluando.

F.DOCE.REG.2 Registro del mensaje de resultado de prueba

Entre las especificaciones funcionales de apartados anteriores, se ha visto la necesidad de poder enviar un mensaje en forma de issue a los repositorios de los UsuEs que contenga la información relativa al resultado de la ejecución de una prueba. Esta especificación dicta que debe existir una manera de poder registrar la forma que toma ese mensaje en cada asignatura, de forma que, por ejemplo, se soporte tener mensajes en diferentes idiomas.

F.DOCE.REG.3 Registro de tokens

De la misma forma, se deben poder registrar todos los tokens que se utilicen para que otros módulos del sistema puedan realizar acciones en nombre de un UsuDo.

F.DOCE.REG.4 Registro de asignaturas y actividades

Para poder realizar la evaluación en el módulo de eGela, es necesario que la base de datos propia pueda relacionar cada evaluación que le llega con la asignatura y actividad a la que pertenece. Por tanto, es necesario poder registrar esta información para su posterior relación.

Especificaciones funcionales de integración

Una vez revisadas todas las especificaciones funcionales de los grupos previamente vistos, se procede a definir aquellas relativas a la integración de los módulos que conformarán el sistema. Para realizar este proceso, se pone el foco en las necesidades de comunicación entre los módulos y la gestión de éstas, así como de los procesos de automatización que deberían existir.

Por tanto, se definen los siguientes grupos de especificaciones funcionales:

- F.INTE.REP Comunicación con el Módulo Repositorio (sobre la necesidad de recibir y filtrar los eventos ocurridos en los repositorios y poder realizar acciones sobre ellos).
- F.INTE.DOC Gestión docente (sobre la necesidad gestionar la información contenida en el bloque formado por el Módulo Docencia y el modelo de datos).
- F.INTE.AUT Gestión de la automatización del sistema (sobre la necesidad de dotar de la mayor automatización posible al sistema NME-CV).

Con la definición completa de los grupos internos de las especificaciones funciones de integración, se pasa ahora a concretar las de más bajo nivel que conforman dichos grupos.

En el grupo de comunicación con el Módulo Repositorio, se pueden definir más especificaciones si se atiende a cómo debe producirse esta comunicación.

F.INTE.REP.1 Recepción de eventos en los repositorios

Primeramente, para saber qué hacer en los repositorios de los UsuEs, es necesario que el sistema sea capaz de recibir y filtrar los eventos que ocurran en ellos, a través de los webhooks que ofrece la plataforma de GitHub. Para ello, es necesario que exista un puerto

abierto y accesible a través de Internet para que el Módulo Repositorio pueda mandar los eventos ocurridos.

F.INTE.REP.2 Filtrado de la información de los eventos

No basta sólo con recibir los webhooks, sino que también hay que ser capaz de filtrarlos a varios niveles. Para empezar, el elemento de integración debe ser capaz de quedarse únicamente con aquellos webhooks que provengan de los UsuEs en GitHub. Por tanto, debe descartar cualquier mensaje que no sea un webhook y que, por mucho que lo sea, no tenga por origen un UsuEs.

Una vez recibido, el sistema debe ser capaz de filtrar la información recibida para quedarse únicamente con la información que sea de utilidad. Esta información se reduce al nombre de usuario del UsuEs que ha generado el evento, el repositorio en el que ha ocurrido y, en general, la información clave del evento. Esta información clave es, por ejemplo, el resultado de una prueba en los eventos de prueba concreta o el resultado de la batería completa en los eventos de evaluación.

F.INTE.REP.3 Ejecución de acciones a través de interfaz de comunicación

Como se ha visto anteriormente, con la información obtenida de los eventos, es necesario que el sistema ejecute ciertas acciones sobre los repositorios de los UsuEs. Esto se realiza a través de la API REST de GitHub y, por tanto, el sistema debe ser capaz de utilizar esta API para ejecutar las funciones necesarias, utilizando para ello la información proveniente de los eventos ocurridos y notificados a través de webhooks.

En cuanto al grupo sobre gestión docente, se han identificado las especificaciones funcionales descritas a continuación.

F.INTE.DOC.1 Comunicación con el modelo de datos

El sistema tiene que disponer de las tecnologías necesarias para poder comunicarse con el modelo de datos de la arquitectura. Esto supone que debe disponer no sólo de la tecnología necesaria para el establecimiento, sino que también debe generar las entidades necesarias en el código para poder manejarla.

F.INTE.DOC.2 Uso de la interfaz de comunicación

Parte del modelo de datos se encuentra dentro de Moodle. Esto supone que, para poder registrar la información de esa parte (la evaluación, por ejemplo), debe utilizar su API REST, por lo que debe disponer del cliente necesario para realizar estas acciones.

F.INTE.DOC.3 Gestión de las transacciones con la base de datos

A la hora de modificar la base de datos, es necesario que exista alguna forma de gestionar las transacciones con ella. De no hacerlo, se corre el riesgo de que la base de datos quede

en un estado inconsistente, al haber realizado sólo una parte de los cambios que iban a haberse realizado.

Por último, en cuanto al grupo sobre la automatización del sistema, se pueden definir las siguientes especificaciones funcionales.

F.INTE.AUT.1 Automatización de la evaluación

Dentro de los procesos a automatizar, se encuentra la automatización de la evaluación de los UsuEs. Todo el proceso desde evaluación debe ser automático, incluyendo:

- Detección de un nuevo intento
- Grabado del resultado de las pruebas
- Anuncio del resultado de las pruebas
- Cálculo de la evaluación para cada actividad
- Anuncio de la evaluación de cada actividad

Esto supone utilizar tanto las tecnologías para poder hacer los cálculos como gestionar la comunicación con el resto de los módulos del sistema.

F.INTE.AUT.2 Automatización de la configuración de eventos

Como se ha mencionado con anterioridad, es necesario que se configuren los webhooks en los repositorios de los UsuEs según éstos se vayan creando. Este es otro de los aspectos a automatizar dentro de la solución, recabando la información de en qué repositorios hay que actuar y configurando en ellos los webhooks.

F.INTE.AUT.3 Acceso a tokens de identificación de docentes

Para poder asegurar la automatización, es necesario que el elemento del sistema responsable de ella tenga un acceso seguro a los tokens que se utilizan para realizar acciones con las API REST de GitHub y Moodle. El uso de los tokens también tiene que ser seguro, de forma que se eviten su filtración.

ANEXO II: Lista de prototipos desarrollados

En este anexo, se listan y definen brevemente los prototipos desarrollados durante el trabajo. A cada prototipo se le asigna un código de nombre al que posteriormente se le añade el código VX (donde X es un número positivo mayor a 0) que identifica las distintas versiones del prototipo.

- **Contexto de evaluación sencilla (P.EVAL)**. Este prototipo corresponde con la definición de un contexto de pruebas sencillo, en el que se realiza una prueba sencilla aun código sencillo. Sus versiones son:
  - **P.EVAL.V1**. Versión en GitLab. Comprueba si el código se compila correctamente.
  - **P.EVAL.V2**. Igual que el anterior añadiendo la comprobación del comportamiento del código ante varias entradas.
  - **P.EVAL.V3**. Versión en GitHub. Comprueba lo misma que el anterior caso, pero para un código más sencillo.
- **Implementación de GitLab (P.INGL)**. Este prototipo corresponde con la forma en la que se despliega una instancia propia de GitLab. Tiene una única versión:
  - **P.INGL.V1**. Despliegue de la instancia de GitLab mediante un contenedor Docker según la documentación oficial.
- **Modelo de datos en directorios (P.MDIR)**. Este prototipo corresponde con la implementación de un modelo de datos que almacene a los UsuEs de forma agrupada que utiliza una estructura de directorios con fichero XML y CSV. Tiene una única versión:
  - **P.MDIR.V1**. Estructura de directorios en sistema Linux. Dispone de varios niveles (asignatura, grupo, subgrupo) con ficheros de configuración XML y listas de estudiantes en CSV.
- **Gestión de ficheros de datos (P.GDAT)**. Este prototipo corresponde con la forma en la que se gestiona el modelo de datos anterior. Tiene una única versión:
  - **P.GDAT.V1**. Gestión de ficheros XML y CSV, completando el path hacia los distintos ficheros del modelo de forma automática en base a la información introducida.
- **Oyente (P.OYEN)**. Prototipo del componente Oyente del MGe. Sus versiones son:
  - **P.OYEN.V1**. Detección del evento de finalización de pipeline de pruebas en GitLab utilizando un servlet de Java.
  - **P.OYEN.V2**. Igual que el anterior añadiendo los eventos de issue y wiki.
  - **P.OYEN.V3**. Igual que P.OYEN.V1 (los eventos de issue y wiki fueron eliminados) en un controlador de Spring.
  - **P.OYEN.V4**. Adaptación del anterior prototipo a GitHub.
- **Escritor (P.ESCR)**. Prototipo del componente Escritor del MGe. Sus versiones son:
  - **P.ESCR.V1**. Utiliza la API de GitLab para escribir issues en los repositorios.
  - **P.ESCR.V2**. Añade creación y configuración de repositorios al anterior.
  - **P.ESCR.V3**. Añade escritura de wikis al anterior.
  - **P.ESCR.V4**. Adapta P.ESCR.V2 (la escritura de wikis fue eliminada) al anterior.

- **Aviso de evaluación (P.AVEV)**. Avisa del resultado de una evaluación escribiendo una issues en el repositorio. Tiene una única versión
    - **P.AVEV.V1**. Escribe la issue con el resultado y la cierra.
- **Creación y configuración de repositorios (P.CCRE)**. Prototipo que incluye la creación y configuración de repositorios de UsuEs para la realización de actividades. Sus versiones son:
    - **P.CCRE.V1**. Creación y configuración de repositorios para GitLab. Apodado como GitLab Classroom, pues pretende dar los primeros pasos para convertirse en el equivalente de GitHub Classroom. Recoge la información de ficheros de datos (P.MDIR y P.GDAT).
    - **P.CCRE.V2**. Modifica el anterior para no leer datos desde ficheros (necesitaría recoger los datos desde una base de datos).
    - **P.CCRE.V3**. Adaptación a GitHub. Al disponer de GitHub Classroom, sólo se aprovecha la configuración de webhooks.
- **Clonación de issues y wikis (P.CLON)**. Clonación de issues y wikis en el repositorio de referencia[26]. Tiene una única versión:
    - **P.CLON.V1**. Demostración de la funcionalidad.
- **Módulo de Integración (P.INTEG)**. Diferentes versiones de implementaciones del módulo de integración (esto es, el que finalmente se ha convertido en el MGe), integrando otros prototipos mencionados anteriormente. Parte de este prototipo hace referencia al prototipo de reporte de prueba en repositorio mencionado en el plan de trabajo. Sus versiones son:
    - **P.INTG.V1**. Integración completa para escritura de issues de respuesta de evaluación en GitLab. Engloba los siguientes prototipos:
        - P.OYEN.V1
        - P.ESCR.V1
        - P.AVEV.V1
    - **P.INTG.V2**. Añade al anterior las funcionalidades de GitLab Classroom. Engloba los siguientes prototipos:
        - P.OYEN.V1
        - P.ESCR.V2
        - P.AVEV.V1
        - P.CCRE.V1
    - **P.INTG.V3**. Añade al anterior la funcionalidad de clonación de issues y wikis. Engloba los siguientes prototipos:
        - P.OYEN.V2
        - P.ESCR.V3
        - P.AVEV.V1
        - P.CCRE.V1
        - P.CLON.V1

[26] Ver Anexo III.

- **P.INTG.V4**. Adaptación del anterior a revisiones de la arquitectura y a Spring. Elimina la función de clonación. Engloba los siguientes prototipos:
  - P.OYEN.V3
  - P.ESCR.V3
  - P.AVEV.V1
  - P.CCRE.V2
- **P.INTG.V5**. Adaptación del anterior a GitHub. Engloba los siguientes prototipos:
  - P.OYEN.V4
  - P.ESCR.V4
  - P.AVEV.V1
  - P.CCRE.V3

- **Implementación de base de datos (P.DBPR)**. Prototipo del modelo de datos utilizando una base de datos[27]. Sus versiones son:
  - **P.DBPR.V1**. Diseño de tablas básicas.
  - **P.DBPR.V2**. Parte del anterior añadiendo parte del seguimiento del desarrollo de una asignatura, utilizando tablas recurrentes.
  - **P.DBPR.V3**. Parte del anterior, simplificando el seguimiento y añadiendo nuevas tablas.
- **Aplicación de gestión de base de datos (P.APDB)**. Prototipo para la gestión de la base de datos del prototipo anterior. Tiene una única versión:
  - **P.APDB.V1**. Gestión de tablas básicas, utilizando Angular y TypeScript.
- **Cliente de API REST de Moodle (P.MAPI)**. Prototipo de cliente para consultar la base de datos de Moodle a partir de su API REST. Tiene una única versión:
  - **P.MAPI.V1**. Llamadas a funciones de interés directamente.

[27] No se han llegado a realizar prototipos completos que combine esta base de datos con la de Moodle.

ANEXO III: Funcionalidades a futuro

En este anexo, se quiere enumerar y describir una serie de funcionalidades que, si bien no se han añadido a la solución final, se han valorado en algún momento del desarrollo. Incluso, en algunos casos, se han llegado a desarrollar prototipos que, si bien se han mostrado exitosos, la funcionalidad que había detrás no ha llegado a implementarse por otros motivos.

Cabe destacar que, en cualquier caso, estas funcionalidades podrían ser añadidas en futuras versiones de la solución.

Pruebas ajenas al funcionamiento del código

Según cómo se definen las pruebas a ejecutar en el MEv, nada impide ejecutar un script que, en lugar de comprobar el funcionamiento del código, pueda comprobar otros aspectos, como, por ejemplo, que cumpla con unas ciertas normas de estilo (que el código no supere cierta longitud, que cumpla con ciertas convenciones…). Este tipo de pruebas es especialmente interesante en un entorno académico, pues, como parte del proceso educativo, es importante no sólo enseñar cómo hacer que el código funcione, sino que también deben inculcarse buenas prácticas de programación. Gracias a ellas, se consiguen crear códigos más eficientes y fáciles de modificar para versiones futuras.

Esta funcionalidad es viable dentro del diseño, pero no se han estudiado suficiente las herramientas que lo harían posible como para haberlo añadido al diseño final.

Pensando de forma más ambiciosa, se podría plantear incluso una prueba que fuera capaz de detectar indicios de plagio en el código que se está evaluando. Para ello, debería accederse al código subido por otros estudiantes del mismo curso (o, incluso, de cursos anteriores) y compararlo en busca de dichos indicios. En el entorno académico, esta prueba sería de un gran interés, pues facilitaría la labor del cuerpo docente y disuadiría al alumnado de intentarlo, obligándoles a realizar un trabajo original.

Esta funcionalidad también es viable en cuanto a arquitectura se refiere. Sin embargo, presenta varios problemas. El primero de ellos es que es necesario encontrar la forma de descargar los códigos con los que comparar. El segundo, que, al tener que resolver la misma práctica, los códigos pueden acabar pareciéndose mucho entre sí, presentando muchos falsos positivos. No se añadió al diseño final al no haberse estudiado en profundidad la forma de implementarla.

Clonación de issues y wikis

En este diseño, no se aprovecha para nada la capacidad social que presenta GitHub a través de issues. A través de estas, se pueden resolver dudas, mediante la escritura de issues que contengan dudas a ser resueltas por el docente al cargo. Sin embargo, los grupos de estudiantes pueden llegar a ser muy grandes, por lo que la tarea de ir repositorio a repositorio comprobando si hay dudas podría llegar a ser una tarea demasiado tediosa.

Para resolver esto, se puede tomar la solución de invitar a los UsuEs a escribir sus dudas en el repositorio de referencia de la asignatura, de forma que los UsuDo sólo tengan que consultar dicho repositorio. Sin embargo, de cara a mejorar el sistema, en su momento se propuso la idea de permitir a los UsuDo escribir issues en sus propios repositorios, para después clonarlos en el repositorio de referencia.

De la misma forma, también se propuso hacer lo mismo con las wikis, donde los UsuEs podrían generar pequeñas guías que ayudaran al resto de estudiantes a resolver las actividades. Estas wikis también podrían ser escritas en los repositorios de los UsuEs y clonados al repositorio de referencia.

Esta funcionalidad llegó a ser probada en el prototipo P.CLON.V1, incluido dentro del prototipo P.INTG.V3, para GitLab. Sin embargo, no se valoró en la solución final, pues, aunque se consiguió una funcionalidad básica, el prototipo sólo se desarrolló en GitLab y no se terminó de desarrollar, al no considerarse una funcionalidad importante para el sistema.

Movimiento de tarjetas

Otra de las funcionalidades que se valoró añadir fue la posibilidad de realizar el seguimiento de los avances del UsuEs a través del movimiento de tarjetas siguiendo la metodología Kanban. Según los planes, estas tarjetas representarían conceptos que va asimilando el UsuEs o actividades que va completando, de forma que, durante sus progresos en la asignatura, se movieran automáticamente.

En su momento, se descubrió que esta metodología era aplicable en GitLab a través de la funcionalidad de boards. Las boards no son otra cosa que tablas donde se muestran las issues del repositorio ordenadas según varios criterios, siendo uno de ellos las etiquetas que se les asignan. Por tanto, se valoró crear una especie de “etiquetas de sistema” llamadas “Sin empezar”, “Empezado” y “Completado” y, al momento de crear el repositorio, crear una serie de issues que correspondieran a las tarjetas de la metodología Kanban con la etiqueta “Sin empezar”. A medida que el UsuEs fuera avanzando en las prácticas, se modificaría la etiqueta, cambiándola por la etiqueta de sistema correspondiente. De esta manera, disponiendo de un board configurado para ver esas etiquetas, lo que se simularía es que las tarjetas se mueven solas a medida que el UsuEs alcanza sus objetivos de aprendizaje.

Sin embargo, aunque sí se realizaron pequeñas experiencias, nunca se llegó a desarrollar un prototipo de la función y, además, al cambiar a GitHub, quedó pendiente reevaluar cómo encajaría en dicha forja la funcionalidad. Por tanto, la funcionalidad no se añadió al diseño final.

Protección de modificación del fichero de pruebas

Quizá la más importante de las funcionalidades que quedó por definir con exactitud es aquella que puede evitar que un UsuEs modifique el fichero de pruebas para falsear sus resultados. Esta funcionalidad no fue añadida ante la falta de prototipos desarrollados para comprobar la mejor de las opciones, aunque se barajaban dos en concreto:

- En git, existe la función de git lock, que permite eliminar los permisos de escritura de un fichero en un repositorio. Se sabe que este sistema funciona dentro de un repositorio local, pero no se ha investigado lo suficiente como para saber si los cambios permanecen una vez el fichero se sube a una forja y se clona.
- Otra opción, más rudimentaria, es detectar también el evento de commit, y tratar de desplegar alguna función de la API que detecte qué ficheros han sido modificados, para avisar al UsuDo si el fichero de pruebas era uno de ellos.

Finalmente, al no haber llegado a una solución concreta, se optó por no implementar nada, de forma que el UsuDo deberá comprobar el estado del fichero de pruebas o las modificaciones de ficheros de cada commit, de forma que pueda detectar si el UsuEs ha hecho trampas.

En cualquier caso, la arquitectura actual soportaría sin problemas esta funcionalidad ya que, como mucho, sería necesario añadir nuevos elementos al Oyente, Escritor y Núcleo del MGe.

Salta automático de branches

Finalmente, otra función que se valoró, pero nunca se llegó a una propuesta específica fue la opción de que, cuando un UsuEs terminara con una actividad, se modificara el repositorio para que la rama principal fuera la de la siguiente práctica, facilitando el uso de ramas. Nunca se exploró ni valoró debidamente esta opción, quedando en una idea peregrina. De todos modos, se quiere dejar constancia de que una vez se pensó por si en una futura implementación se encontrara la forma de hacerlo.

ANEXO IV: Partida de paquetes de trabajo desglosada

Para más información acerca de la partida del presupuesto referente al coste de los paquetes de trabajo, a continuación, se muestran unas tablas en la que se desglosa cada paquete, mostrando el coste de cada actividad. Además, también se muestra el coste de cada paquete de trabajo y actividad por cada figura.

		Coste IS	Coste IJ	Coste TE	Coste SU	Coste Total
PT0	***Investigación y formación tecnológica***	***0 €***	***3.200,00 €***	***1.120,00 €***	***0 €***	***4.320,00 €***
T001	Uso avanzado de aplicación de CI	0 €	1.280,00 €	160,00 €	0 €	1.440,00 €
T002	Recepción y filtrado de webhooks	0 €	640,00 €	320,00 €	0 €	960,00 €
T003	Implementaciones de clientes de API REST de GitHub	0 €	640,00 €	320,00 €	0 €	960,00 €
T004	Uso de API REST de Moodle	0 €	640,00 €	320,00 €	0 €	960,00 €
PT1	***Definición de requerimientos y especificaciones***	***7.920,00 €***	***4.480,00 €***	***400,00 €***	***360,00 €***	***13.160,00 €***
T101	Problemáticas	1.200,00 €	640,00 €	80,00 €	60,00 €	1.980,00 €
T102	Identificación de alternativas y criterios	960,00 €	1.280,00 €	80,00 €	120,00 €	2.440,00 €
T103	Especificaciones de diseño	1.440,00 €	640,00 €	80,00 €	60,00 €	2.220,00 €
T104	Especificaciones funcionales	1.920,00 €	640,00 €	80,00 €	60,00 €	2.700,00 €
T105	Definición del plan de pruebas	2.400,00 €	1.280,00 €	80,00 €	60,00 €	3.820,00 €
PT2	***Diseño de la arquitectura***	***6.720,00 €***	***3.680,00 €***	***800,00 €***	***420,00 €***	***11.620,00 €***
T201	Módulos que conforman el sistema	1.440,00 €	960,00 €	160,00 €	120,00 €	2.680,00 €
T202	Comunicación entre módulos	960,00 €	640,00 €	160,00 €	60,00 €	1.820,00 €
T203	Usuarios en que hacen uso de cada módulo	960,00 €	640,00 €	160,00 €	60,00 €	1.820,00 €
T204	Tablas y requerimientos de la base de datos	1.440,00 €	480,00 €	160,00 €	60,00 €	2.140,00 €
T205	Definición de prototipos de validación	1.920,00 €	960,00 €	160,00 €	120,00 €	3.160,00 €
PT3	***Diseño interno de los módulos***	***7.680,00 €***	***5.760,00 €***	***560,00 €***	***600,00 €***	***14.600,00 €***
T301	Módulo Repositorio	960,00 €	640,00 €	80,00 €	60,00 €	1.740,00 €
T302	Módulo Evaluador	960,00 €	640,00 €	80,00 €	60,00 €	1.740,00 €
T303	Módulo Gestor	1.440,00 €	1.280,00 €	80,00 €	120,00 €	2.920,00 €
T304	Relaciones y atributos de tablas de base de datos	960,00 €	1.280,00 €	80,00 €	120,00 €	2.440,00 €
T305	Módulo Datos	480,00 €	320,00 €	80,00 €	60,00 €	940,00 €
T306	Módulo Docencia	960,00 €	640,00 €	80,00 €	60,00 €	1.740,00 €
T307	Especificación formal de prototipos	1.920,00 €	960,00 €	80,00 €	120,00 €	3.080,00 €
PT4	***Desarrollo y depuración de prototipos***	***3.840,00 €***	***6.720,00 €***	***4.480,00 €***	***600,00 €***	***15.640,00 €***
T401	Instalación de servicios y preparación de entornos	0 €	0 €	240,00 €	0 €	240,00 €
T402	Prototipo de contexto de evaluación sencilla (P.EVAL)	0 €	1.600,00 €	800,00 €	120,00 €	2.520,00 €
T403	Prototipo de reporte de prueba en repositorio (P.INTEG)	1.920,00 €	2.080,00 €	1.680,00 €	240,00 €	5.920,00 €
T404	Prototipo de cliente de API REST de Moodle (P.MAPI)	960,00 €	1.440,00 €	800,00 €	120,00 €	3.320,00 €
T405	Prototipo de implementación de base de datos (P.DBPR)	960,00 €	1.600,00 €	960,00 €	120,00 €	3.640,00 €
PT5	***Integración de prototipos***	***2.160,00 €***	***2.880,00 €***	***960,00 €***	***180,00 €***	***6.180,00 €***
T501	Integración del Subsistema Código	480,00 €	320,00 €	160,00 €	60,00 €	1.020,00 €
T502	Integración del Subsistema Docencia	480,00 €	960,00 €	320,00 €	60,00 €	1.820,00 €
T503	Integración con el Subsistema Integración	1.200,00 €	1.600,00 €	480,00 €	60,00 €	3.340,00 €
PT6	***Validación de la solución adoptada***	***3.840,00 €***	***3.840,00 €***	***0 €***	***360,00 €***	***8.040,00 €***
T601	Validación del Módulo Repositorio	480,00 €	640,00 €	0 €	60,00 €	1.180,00 €
T602	Validación del Módulo Evaluador	480,00 €	640,00 €	0 €	60,00 €	1.180,00 €
T603	Validación del Módulo Gestor	960,00 €	960,00 €	0 €	60,00 €	1.980,00 €
T604	Validación del Módulo Datos	480,00 €	640,00 €	0 €	60,00 €	1.180,00 €
T605	Validación del Módulo Docencia	960,00 €	640,00 €	0 €	60,00 €	1.660,00 €
T606	Selección de soluciones a las problemáticas	480,00 €	320,00 €	0 €	60,00 €	860,00 €
PT7	***Documentación y gestión***	***1.680,00 €***	***1.920,00 €***	***400,00 €***	***180,00 €***	***4.180,00 €***
T701	Realización del presupuesto	480,00 €	320,00 €	0 €	60,00 €	860,00 €
T702	Documentación del proyecto y entregables	1.200,00 €	1.600,00 €	400,00 €	120,00 €	3.320,00 €

Tabla 49. Partida de paquetes de trabajo desglosada