GameCraft: Una plataforma de integración continua para videojuegos basada en microservicios

Trabajo de Fin de Máster realizado por

Iván Martínez Mateu

Dirigido por

Prof. Dr. Federico Peinado Gil

Máster en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Curso académico	*Convocatoria*	*Calificación*
2017 - 2018	**Septiembre 2018**	**9 - Sobresaliente**

Autorización de difusión

El alumno abajo firmante, matriculado en el Máster en Ingeniería Informática de la Facultad de Informática, y el profesor abajo firmante, director del presente Trabajo Fin de Máster, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "GameCraft: Una plataforma de integración continua para videojuegos basada en microservicios", realizado durante el curso académico 2017-2018 bajo la dirección de Federico Peinado Gil en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en internet y garantizar su preservación y acceso a largo plazo.

Fdo.: Iván Martínez Mateu

Fdo.: Federico Peinado Gil

Agradecimientos

Cuando se innova, se corre el riesgo de cometer errores. Es mejor admitirlo rápidamente y continuar con otra innovación

Steve Jobs

Quería expresar mi más sincero agradecimiento a mi director, Federico, por ayudarme y guiarme durante este proyecto, a mi "familia", en el sentido más amplio de la palabra, por haberme apoyado y animado a lo largo de todos estos años y a todos los desarrolladores del mundo que han participado en el desarrollo de las librerías, herramientas y tecnologías usadas en el proyecto, ya que sin ellos, GameCraft no podría haber llegado al estado final de su desarrollo. A todos ellos, gracias.

Resumen

Hoy día uno de los paradigmas más establecidos consiste en facilitar la colaboración entre los desarrolladores del proyecto automatizando muchos de los pasos necesarios para generar la versión final de una aplicación software. El modelo de integración continua surge como parte de este esfuerzo, no sólo para acelerar el proceso sino para garantizar un software de mayor calidad, detectando fallos y otros problemas de manera anticipada. Sin embargo, a día de hoy, todavía muchos proyectos de videojuegos se desarrollan utilizando procedimientos obsoletos que ocasionan retrasos.

Estudios recientes señalan que la industria del videojuego podría beneficiarse del uso de metodologías y herramientas que tuviesen en cuenta las peculiaridades del largo y complejo proceso de desarrollo y publicación de un videojuego. Esto causaría impacto especialmente en estudios de desarrolladores independientes de videojuegos que actualmente no utilizan entornos profesionales de integración, entrega y despliegue continuo de sus productos, y encuentran dificultades para crear un flujo de trabajo razonable.

Por ello, este trabajo propone *GameCraft*, una plataforma de integración continua orientada a la producción de videojuegos. El código ha sido desarrollado en abierto utilizando *Java* y *Spring Boot* y las primeras pruebas de esta herramienta con desarrolladores han tenido un razonable éxito. Finalmente se ha publicado una primera versión que permite a cualquier equipo montar de forma gratuita este sistema, y así poder realizar la integración continua de manera profesional y adaptada a sus necesidades.

Palabras clave

integración, continua, videojuegos, código abierto, plataforma, entrega, microservicios

Abstract

Nowadays one of the most established paradigms is to facilitate the collaboration between the developers of a project automating many of the necessary steps to generate the final version of a software application. The continuous integration model emerges as part of this effort, not only to accelerate the process but to guarantee a higher quality software, detecting failures and other problems in advance. However, to this day, many video game projects are still developed using obsolete procedures that often cause delays and operational failures.

Recent studies indicate that the video game industry could benefit from the use of methodologies and tools that take into account the peculiarities of the long and complex process of development and publication of a video game. This would cause impact especially in studios of independent game developers who are not using professional continuous integration, delivery and deployment environment, and find it difficult to create a reasonable workflow.

Therefore, this work proposes *GameCraft*, a continuous integration platform oriented to the production of video games. Code has been developed as open source using *Java* and *Spring Boot* framework and first tests of this tools with real developers have obtained a reasonable success. Finally it has been published a first version that allows any team to install this system and its web interface for free, and, therefore, perform continuous integration in a professional way, adapted to their needs.

Title

"GameCraft: A continuous integration platform for video games based on microservices".

Keywords

continuous, integration, video games, open, source, platform, delivery, microservices.

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Mi trabajo es un juego, un juego muy serio.

Maurits Cornelis Escher

Resumen: Este capítulo presenta los conceptos de integración, entrega y despliegue continuo y explica la motivación del proyecto. Además, se presentan los objetivos a abordar con este trabajo, los entregables esperados y se proporciona una visión de alto nivel de *GameCraft*, la herramienta software desarrollada.

En el ciclo de vida de un proyecto software, el producto pasa por diferentes procesos como pueden ser análisis, diseño, desarrollo, pruebas y publicación. Lo común cuando se está trabajando en un proyecto industrial es realizar miles de pruebas diferentes en múltiples dispositivos con diferentes versiones de una misma aplicación y publicarlo también en múltiples mercados de aplicaciones y plataformas de testeo para usuarios finales. Realizar manualmente todas estas tareas para un equipo sería realmente tedioso además de contraproducente para la propia empresa ya que implica un alto coste económico y de oportunidades debido al tiempo que se necesita emplear.

Por estas razones, las empresas necesitan automatizar los procesos seguidos para mejorar en calidad final del producto reduciendo costes y aumentando la calidad del software producido. De aquí surge el concepto de *integración, entrega y despliegue continuo*. La idea es que, una vez los miembros del equipo de trabajo suben los cambios realizados en el proyecto a un repositorio central, se ejecutan de forma automática una serie de tareas y pruebas que buscan obtener los siguientes beneficios:

- Encontrar y arreglar fallos de programación con mayor rapidez.
- Mejorar la calidad del software producido.

- Reducir el tiempo que se tarda en validar y publicar nuevas versiones.

La dificultad para instalar y configurar un entorno de integración, entrega y despliegue continuo y su compleja integración con los flujos de trabajo en los equipos de desarrollo del ámbito particular de los videojuegos son los dos problemas principales que se plantean resolver en este trabajo.

Para hacer frente a esta problemática, se propone la creación de una plataforma software de código abierto capaz de abstraer al usuario de los detalles de bajo nivel de configuración de un entorno de integración, entrega y despliegue continuo, así como de algunas de las dificultades de su utilización. El desarrollo de la plataforma utiliza el lenguaje de programación *Java* y el framework *Spring Boot* para ofrecer una arquitectura basada en microservicios, de manera que esta pueda escalar horizontalmente de forma sencilla de acuerdo a la carga de trabajo recibida.

1.1. Objetivos

El objetivo principal de este proyecto es el siguiente:

- **(Obj1)**. Crear una infraestructura para la integración continua enfocada al desarrollo de videojuegos, fácil de instalar, configurar y usar por estos equipos, y con posibilidad en un futuro de incorporar funcionalidades específicas para realizar pruebas interactivas.

Para alcanzar este objetivo principal, se establecen las siguientes actividades a realizar en la metodología:

- **(Act1)**. Estudiar y analizar el paradigma de la integración continua, sus herramientas y, en general, su relevancia en el proceso de desarrollo, testeo y publicación de aplicaciones.
- **(Act2)**. Identificar el papel de la integración continua dentro de los equipos de desarrollo de un estudio de videojuegos.
- **(Act3)**. Averiguar las características principales que debe tener una plataforma de integración, entrega y despliegue continuo para ser útil de manera específica en el ámbito de los videojuegos.
- **(Act4)**. Desarrollar la plataforma mencionada como sistema escalable y de alta disponibilidad, basado en microservicios para que se adapte fácilmente a cualquier carga de trabajo.

Las plataformas de integración continua de este tipo que existen actualmente tienen algunas carencias. Algunas son gratuitas pero difíciles de administrar y mantener, como por ejemplo *Jenkins* o *Hudson*. Otras tienen

versiones de pago, que suelen ser muy manejables y ofrecen atractivas características, pero su licencia es tan costosa que en ocasiones resultan inasumibles por los estudios de desarrollo de videojuegos más pequeños, como ocurre con *Unity Cloud Build* o *Visual Studio Team Services*.

Gracias a la propuesta de este trabajo de crear una plataforma gratuita y de código abierto específica para proyectos de videojuegos, se espera conseguir un aumento de la productividad y una reducción de los costes en los estudios de desarrollo independiente. Además, el diseño de la herramienta cuenta con una arquitectura de microservicios, para mejorar su escalabilidad horizontal y, por tanto, adaptarse de forma natural a muy distintos volúmenes de trabajo. La interfaz de usuario también será trabajada para simplificar su uso, pensando en administradores que podrían no ser expertos en programación ni tener experiencia en integración continua. A diferencia de otros sistemas como *Jenkins*, donde la curva de aprendizaje es elevada por la gran cantidad de opciones que existen en la plataforma, la cantidad de funciones y opciones disponibles estará limitada a las estrictamente necesarias para el desarrollo y la publicación de videojuegos. Se trata de reducir el tiempo requerido para aprender a configurar y mantener este entorno, de manera que se pueda compilar, probar, analizar el código y publicar el producto de la manera más sencilla y rápida posible.

Por tanto, el resultado principal de este trabajo será el lanzamiento de este nuevo proyecto software enfocado a la comunidad de desarrolladores de videojuegos, comparable al menos desde el punto de vista de la interfaz con otros muy populares, como *Unity Cloud Build*, compatible con los sistemas operativos más utilizados (*GNU/Linux*, *macOS* y *Windows*), y con unos requisitos de recursos de cómputo asequibles y bien determinados.

1.2. Alcance del proyecto

Para que el lector se haga una idea del alcance del proyecto se muestra a continuacion el árbol de características de *GameCraft*, la plataforma de integración continua resultante de este trabajo.

El árbol de características, introducido por primera vez en el año 1990 (David Olson (2013)), organiza las principales características de un sistema en grupos que capturan visualmente información sobre la funcionalidad del sistema en alto nivel. Este modelo, considerado próximo al diagrama de descomposición funcional en algunas metodologías, puede contener características funcionales (aspectos relacionados con el hardware y el software de un sistema), características no funcionales (aspectos relacionados con el rendimiento, la internacionalización o el mantenimiento del sistema). y otros parámetros como el coste de desarrollo del sistema o los diferentes niveles de detalle que refleja una característica.

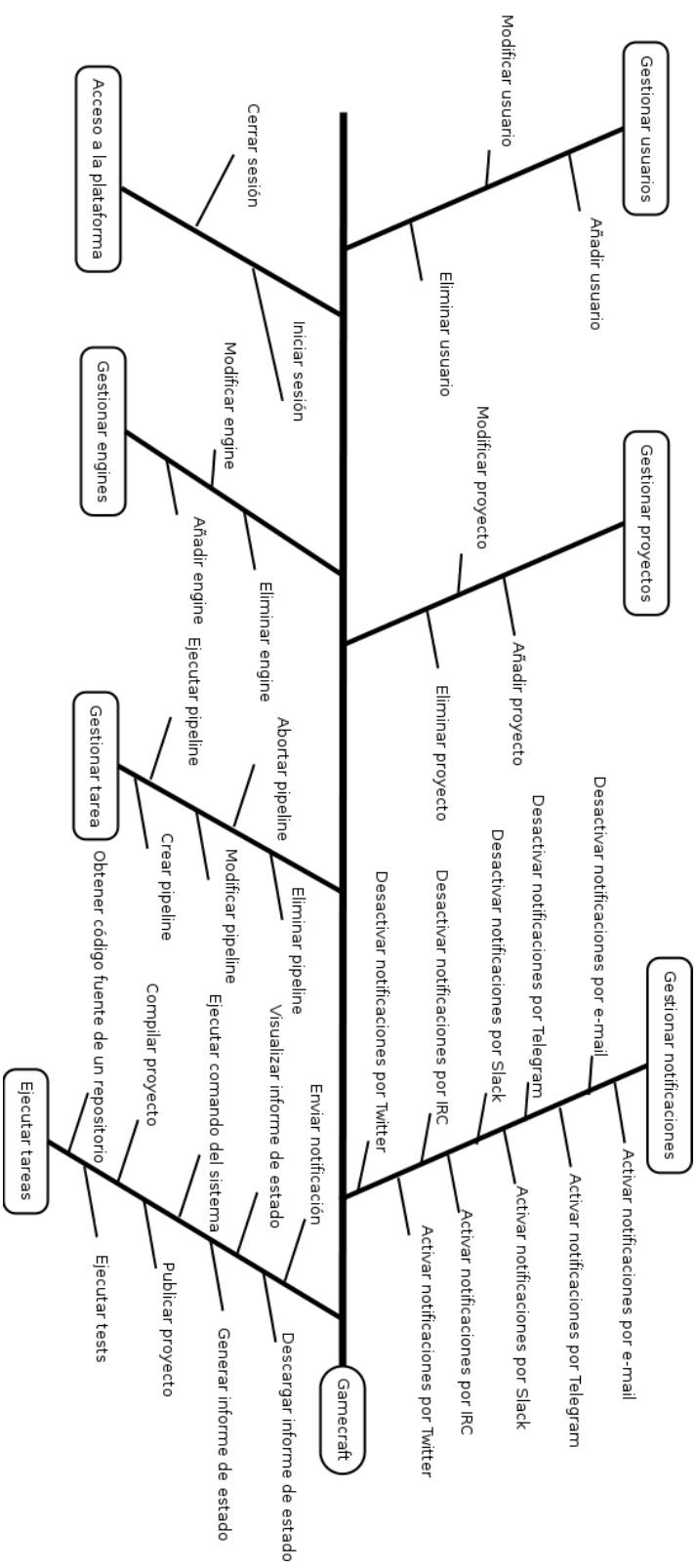

Figura 1.1: Árbol de características de *GameCraft*

En el Anexo D se explican brevemente las características recogidas en el árbol de características de la Figura 1.1.

Es necesario reseñar también las limitaciones actuales de alcance que este proyecto tiene y que se deberían ir corrigiendo en versiones futuras que se vayan haciendo. Cabe destacar que este proyecto es una prueba de concepto, es decir, una propuesta de cómo se ha llevado la arquitectura basada en microservicios para solucionar un problema evidente en el desarrollo de videojuegos. Debido a la complejidad técnica del uso de varias tecnologías en el proyecto, es necesario contar con, al menos, una o dos personas con conocimiento elevado en informática (perfiles *senior*) y en diseño de arquitectura para mantener la plataforma o entender porque los microservicios fallan para poder tratar de hacerlos funcionar lo antes posible. Además, la plataforma es capaz de compilar y ejecutar pruebas automatizadas de código y evaluación de calidad de la misma, pero todavía no es posible ejecutar automáticamente *playtestings*, ni es capaz de evaluar adecuadamente si los ficheros *multimedia* que se incorporan en el videojuego (como modelos 3D o ficheros de audio) consumen demasiados recursos que pudieran ralentizar el rendimiento general del videojuego, por ejemplo. También, la plataforma no es compatible con repositorios que no sean de *Github*, por lo que los proyectos de videojuegos que utilicen *SVN* o *BitBucket* como repositorios, no podrán hacer uso de la plataforma ni es posible llevar a cabo publicaciones automatizadas en plataformas de aplicaciones como *App Store* o *Google Play* y plataformas de testeo para usuarios finales.

1.3. Entregables

Como entregables resultantes de este trabajo se pone a disposición del lector un CD-ROM con una copia digital de esta misma memoria. Además, en los apéndices se proporciona un manual de instalación que ayudará al usuario en el proceso de despliegue de la plataforma en un servidor privado y un manual de usuario que proporciona una descripción de las características y funcionalidades principales ofrecidas por la plataforma.

El repositorio donde se ha almacenado el código fuente de este proyecto se encuentra en el siguiente enlace: `https://github.com/iMartinezMateu/gamecraft`.

1.4. Estructura del documento

En esta sección se describe la estructura del documento de modo que sirva de orientación al lector. La presente memoria se divide en los siguientes capítulos:

- **Capítulo 1. Introducción**. Este capítulo presenta los conceptos de

integración, entrega y despliegue continuo y explica la motivación del proyecto. Además, se presentan los objetivos a abordar con este trabajo, los entregables esperados y se proporciona una visión de alto nivel de *GameCraft*, la herramienta software desarrollada.

- **Capítulo 2. Estado de la técnica**. Este capítulo explica el ciclo de vida del software y por qué es importante la integración, entrega y despliegue continuo tanto en un proyecto software genérico como en el desarrollo de un videojuego. Además, se analiza el estado de la técnica en integración continua, tras obtener información de las herramientas similares a *GameCraft* que existen en el mercado y ver cuales son las características de cada una. Por último, se realiza una comparación de todas las herramientas analizadas y de *GameCraft*.

- **Capítulo 3. Metodología y gestión del proyecto**. Este capítulo aborda con detalle las metodologías y tecnologías utilizadas en el desarrollo y presenta una planificación temporal de todo el proyecto en forma de diagrama de Gantt.

- **Capítulo 4. Desarrollo del sistema**. Este capítulo contiene el grueso de la contribución de este trabajo, exponiéndose todo lo relativo a las distintas fases del proceso de desarrollo software del sistema, desde la idea inicial a la implementación, pasando por el análisis de los requisitos, la arquitectura y el diseño de la plataforma.

- **Capítulo 5. Pruebas y resultados**. El capítulo documenta las pruebas realizadas para evaluar el desempeño de la plataforma y también las consultas y los resultados obtenidos tras evaluarla y obtener realimentación con usuarios.

- **Capítulo 6. Conclusiones**. En este capítulo se extraen las conclusiones del proyecto. Además se plantea el posible trabajo futuro que la comunidad de código abierto puede continuar realizando sobre esta.

Además la memoria cuenta con una serie de apéndices para las traducciones al inglés de la Introducción y de las Conclusiones, así como un glosario con la terminología utilizada, manuales y otra documentación.

Capítulo 2

Estado de la técnica

Una buena idea es algo que no resuelve
un solo problema, sino que puede
resolver múltiples problemas a la vez.

Shigeru Miyamoto

Resumen: Este capítulo explica el ciclo de vida del software y por qué es importante la integración, entrega y despliegue continuo tanto en un proyecto software genérico como en el desarrollo de un videojuego. Además, se analiza el estado de la técnica en integración continua, tras obtener información de las herramientas similares a *GameCraft* que existen en el mercado y ver cuales son las características de cada una. Por último, se lleva a cabo una comparación entre todas las herramientas analizadas y *GameCraft*.

Como advierten Duvall et al. (2007) la integración de módulos o componentes software no supone un desafío nuevo para el mundo de la Informática. La integración no suele ser problemática cuando el proyecto sólo lo lleva a cabo un programador, pero cuando la complejidad del proyecto aumenta, aparece la necesidad de integrar todas las partes desarrolladas con más frecuencia y de asegurarse periódicamente de que el conjunto funciona de manera correcta. Esperar a que se termine el desarrollo para llevar a cabo la integración de todas sus partes produce como mínimo retrasos en los proyectos. El paradigma de la integración, entrega y despliegue continuo trata de resolver esta clase de problemas.

Podemos definir la integración, entrega y despliegue continuo esencialmente como una serie de prácticas a realizar durante el proceso de desarrollo de software donde los miembros del equipo integran su trabajo frecuentemente, al menos una vez al día. Cada integración es verificada por una construcción automatizada del proyecto, incluyendo la ejecución de tests y análisis

de código, para detectar los problemas de integración rápidamente. Gracias a la integración continua, un equipo puede desarrollar software de calidad y detectar fallos de cohesión entre los diferentes módulos o componentes muy rápidamente. Esto quiere decir que:

- Todos los desarrolladores deben enviar el código al repositorio común al menos una vez al día.
- Siempre se genera un producto que ha sido testeado al 100 %.
- Se obtienen informes y métricas del desempeño del equipo que pueden ser utilizadas para mejorar la productividad.

En esencia, la integración, la entrega y el despliegue continuo[1], tal y como consideran Humble y Farley (2010), actúa como un indicador de fiabilidad, ya que asegura de todos los cambios pequeños introducidos a lo largo de la vida del desarrollo del proyecto son plenamente compatibles con el resto del software. Además de incrementar la confianza de los equipos, reduce la cantidad de recursos humanos necesarios en los proyectos, porque se consiguen automatizar tareas tediosas que en principio deberían ser realizadas varias veces al día, cada vez que el software cambia.

El escenario típico (representado gráficamente en la Figura 2.1) comienza cuando el desarrollador sube los cambios del código fuente al repositorio común. Periódicamente la plataforma de integración, entrega y despliegue continuo está analizando el repositorio para detectar si se ha producido un cambio.

Por ello, en cuanto se han remitido dichos cambios (*commit*) al repositorio, la plataforma los detecta y procede a descargar la última versión de todo el código fuente del proyecto. Una vez descargado este código, la plataforma ejecuta los *scripts* de construcción de proyecto. En estos *scripts* se indica qué pasos se deben seguir para compilar y ejecutar los tests del proyecto (por ejemplo, si es necesario descargar dependencias, utilizar alguna directiva de precompilación (*flag*), etc.)

El resultado de la ejecución de estos *scripts* se enviará por correo electrónico o por otro método de comunicación a los integrantes del equipo de desarrollo para informarles del estado de la construcción automática, es decir, si se ha producido un error o si por el contrario todo marcha bien.

La plataforma de integración, entrega y despliegue continuo siempre está analizando el repositorio en busca de nuevos cambios y en caso de que se produzcan, repetirá estos pasos relatados anteriormente.

[1] En esta memoria se usan indistintamente los término íntegración continuaé íntegración, entrega y despliegue continuo'. Aunque el segundo sea algo más amplio que el primero, en cuanto a paradigma hablamos esencialmente de lo mismo.

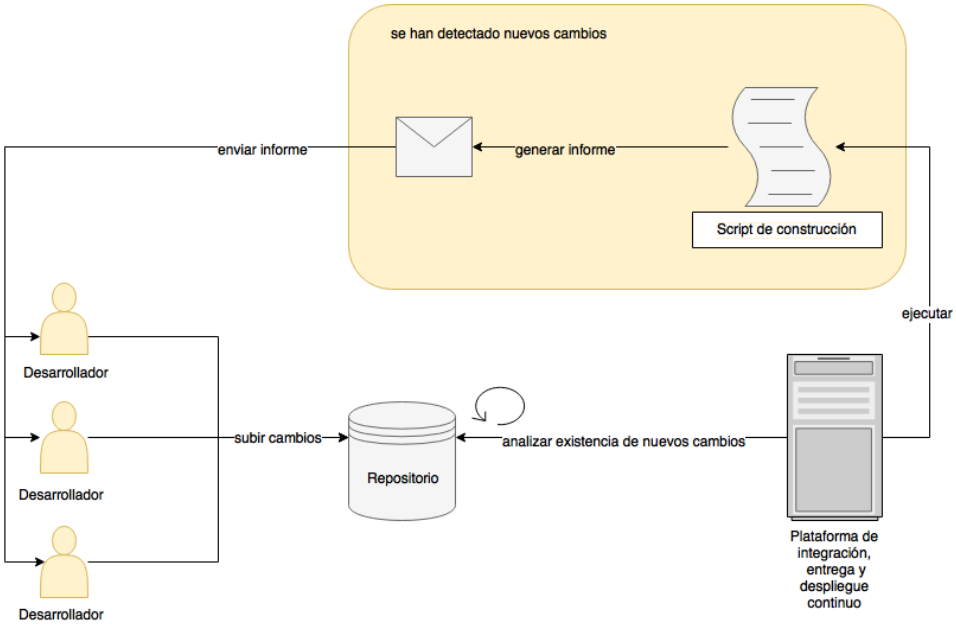

Figura 2.1: Componentes de una plataforma de integración, entrega y despliegue continuo.

2.1. Integración continua en el videojuego

El ámbito del desarrollo de videojuegos es un subproblema de desarrollar *software* según Scacchi (2004) y Stacey y Nandhakumar (2004). Lo peculiar de los proyectos de videojuegos es la variedad de talento que interviene en el proceso de creación y desarrollo (no sólo participan programadores, sino diseñadores, artistas, músicos, gente de localización... y hasta responsables de marketing), personas con diferente nivel de conocimiento que a menudo no suelen estar concienciados con los procesos de integración continúa, pero que el desconocimiento de la misma puede resultar también un problema ya que en un videojuego, un fallo no sólo se puede producir por un error en el código, sino también por un fichero de audio corrupto (pudo haber sido incluido al videojuego sin haber sido escuchado antes) o un modelo 3D cuya calidad es excesivamente alta y provoca que el juego no gestione correctamente los recursos para representarlo visualmente por pantalla. Además, el desarrollador de videojuegos está llamado a optimizar recursos con un enorme esfuerzo: el juego suele ocupar mucho y consumir muchos recursos (lo cual afecta claramente al tema de su alojamiento y a la forma de trabajar con él) y dinámicamente, carga y descarga recursos muy pesados a tiempo real. Además de los controles de calidad y pruebas de *UX* normales, los videojuegos también requieren de *playtesting*, es decir, comprobar que son divertidos, no demasiado difíciles para superar ciertos niveles, que la dificultad se ajusta

gradualmente, etc.

Al emplear pruebas automatizadas como parte del proceso de desarrollo de un videojuego, que suelen ser aplicaciones software bastantes complejos, es común que la ejecución de determinados tests lleva bastante tiempo. Si los desarrolladores tienen que ejecutar estas pruebas en sus propias máquinas, se mostrarán reacios a realizarlas, ya que van a ralentizar su trabajo, y es posible que lleguen a no realizarse, siendo absolutamente inútiles.

La solución a este problema es dedicar una máquina, debidamente configurada, para ejecutar las pruebas automatizadas. Dicha máquina es la que regularmente debe analizar el repositorio donde se encuentra el código fuente del videojuego para comprobar si los desarrolladores han subido cambios. Si en efecto se han encontrado cambios recientes, el código se revisa y se compila y las pruebas se ejecutan. Finalmente, como dijimos, el sistema, quien lleva un estricto control de versiones, envía un informe que contiene los resultados de la compilación y el testeo al equipo de desarrolladores.

Los beneficios en usar la integración, entrega y despliegue continuo en el desarrollo de los videojuegos son claros: ahorra tiempo de producción a la empresa, permitiendo que los desarrolladores tengan más recursos computacionales para trabajar de forma productiva. Además, dado que ya no tienen que preocuparse por la ejecución de las pruebas automatizadas, siempre se tiene la certeza de que estas se van a ejecutar correctamente, ya que si apareciera código fuente con errores provocaría que los sistemas de integración, entrega y despliegue continuo emitieran un informe alertando al equipo de desarrolladores.

La introducción de pruebas automatizadas como parte de la integración, entrega y despliegue continuo en los procesos de desarrollo de los videojuegos mejora la eficiencia de los equipos, obteniéndose videojuegos más depurados, estables y con mayor calidad. Se reduce la presión y la carga de trabajo en los equipos al prescindir del gran esfuerzo que supone hacer las pruebas manuales, y también ayuda a encontrar errores desde el principio del proceso de desarrollo.

Para que el proceso de integración continua sea eficiente, es necesario que los desarrolladores cambien sus hábitos típicos de desarrollo. Los desarrolladores deben subir los cambios al repositorio común muy frecuentemente, dar prioridad a arreglar errores que evitan la construcción del proyecto frente a otras tareas, desarrollar tests que se puedan verificar y validar siempre antes de subir o nada más obtener código fuente que no tiene fallos de programación.

De acuerdo a Pascarella et al. (2018), las conclusiones que obtienen justifican crear una Ingeniería del Software orientada a videojuegos:

- Hace falta otra ingeniería de requisitos, que siga mejor el proceso e incorpore lo que desean los usuarios finales (poner mayor énfasis en la *retroalimentación*).

- El código de los videojuegos es menos extensible y se reutiliza menos, por lo que, aumentan los costes y repercute en mayores tiempos de desarrollo y *testing* y en un mayor grado de repetición de tareas manuales que si estuviéramos desarrollando una aplicación informática.
- La prevención y corrección de errores no puede limitarse al código, sino a los múltiples y variados ficheros que utilizan los videojuegos para representar gráficos en la pantalla, leer configuraciones, almacenar datos, etc.
- El *testeo* actual en el mundo de los videojuegos todavía es muy manual, es necesario crear herramientas que permiten generar y replicar partidas jugadas, incluso generar datos de prueba automáticamente, para automatizar los campos de prueba que se realizan sobre el videojuego.

Subir cambios frecuentemente

Uno de los principios fundamentales de esta paradigma es integrar tan frecuentemente como sea posible. En cuanto hay alguna novedad, esta debe figurar en el repositorio común, pues de lo contrario, la integración tardará más tiempo en lograrse y los desarrolladores no podrán familiarizarse ni hacer uso de las últimas mejoras con que cuenta el proyecto.

Para subir cambios frecuentemente, se puede seguir algunas técnicas como:

- **Hacer pequeños cambios**: No hay que realizar cambios a varios componentes al mismo tiempo. Hay que elegir una tarea pequeña, desarrollar los tests y la funcionalidad, ejecutar los tests y luego subir el *commit* al repositorio.
- **Subir el *commit* después de cada tarea**: Partiendo de la base de que las tareas se han dividido para que puedan acabarse en unas pocas horas, subir el *commit* cuando se haya completado totalmente la tarea.

Hay que evitar que los miembros del equipo envíen los *commits* en un mismo momento del día ya que es frecuente obtener errores de compilación debido a los conflictos que se producen por los cambios introducidos entre varias personas al proyecto. Cuanto más tiempo se tarde en integrar los cambios de una persona con el resto, más probabilidad de que la integración falle debido a conflictos de código.

No subir código fuente con fallos de programación

En un proyecto no se debe enviar al repositorio código que no funcione debidamente, por lo que es vital que se compruebe antes, que el código haya sido probado y testeado localmente en la máquina del desarrollador como

paso previo a subirla al repositorio. La última mitigación para este riesgo es tener un *script* de compilación bien estructurado que compile y pruebe el código de manera repetible. Es necesario convertir esto en una parte crucial de la práctica de desarrollo que sigue el equipo, llegando incluso a realizar compilaciones privadas (similares a las del proceso de integración) antes de enviar código al repositorio.

Una buena forma de detectar posibles fallos de programación es analizando la calidad del código. Este análisis nunca se realiza sobre los programas en ejecución, sino que se realiza directamente sobre el código fuente del proyecto, detectando por ejemplo código duplicado, vulnerabilidades de seguridad, fallos de programación y uso de APIs obsoletas.

Existen herramientas que facilitan la automatización este tipo de análisis (como *Sonarqube* [2] y envían informes al equipo de desarrollo con los problemas detectados y las medidas correctivas que se deben adoptar para aumentar la calidad del código producido, aunque también se pueden llevar a cabo análisis 'manuales'realizados por otros miembros del equipo.

- El análisis automático se basa en un software que detecta problemas y vulnerabilidades en el código fuente utilizando unas reglas predefinidas. Al ser llevado a cabo por una máquina, su duración es de unos pocos minutos aunque tiene la capacidad de analizar mucho más código y más rápido que en un análisis manual.

- El análisis manual se centra en analizar apartados propios de la aplicación, como por ejemplo, si el diseño de la arquitectura lógica ha sido correcto. Al ser llevado a cabo por una o varias personas, su duración puede llegar a durar unas horas, por lo que es conveniente no abusar de este tipo de análisis pues podrían ocasionar retrasos en la finalización del proyecto.

Cabe destacar que ambos tipos de análisis son complementarios ya que el análisis automático se centra en la sintaxis y la semántica del código mientras que el análisis manual se ocupa de facetas de más alto nivel, como por ejemplo, la forma que se ha seguido para estructurar la aplicación en módulos o componentes. La simbiosis de ambos tipos permite identificar problemas antes, repercutiendo en una mejora tanto en el desarrollo como en el mantenimiento del código fuente.

Arreglar errores que evitan la construcción del proyecto

Un error en la construcción del proyecto es cualquiera que impide que el ejecutable pueda ser utilizado de forma satisfactoria, desde un error de compilación, un test fallido, un problema con la base de datos o un mal

[2] https://www.sonarqube.org

despliegue. En un entorno de integración, entrega y despliegue continuo, estos problemas deben ser arreglados inmediatamente.

Desarrollar tests que puedan ser automatizados

El proceso de construcción de un proyecto debe intentar ser automatizado en su totalidad. Con el objetivo de ejecutar tests para un entorno de integración, entrega y despliegue continuo, los tests también tienen que estar automatizados. Armazones populares como *Nunit*[3] o *Junit*[4] ya proporcionan mecanismos eficaces para ejecutar estos tests de forma automática.

A pesar de lo beneficioso que son estos tests automatizados, todavía hay muchos aspectos en el desarrollo del videojuego que no pueden ser probados automáticamente. Lógicamente es difícil probar automáticamente si un juego es divertido o no, o si será del agrado del público objetivo. Para desarrollar tests automatizados para videojuegos, es por tanto necesario seguir las siguientes pautas:

- Implementar casos de prueba para detectar errores antes de comenzar a corregirlos. Contar con estos casos de prueba nos asegurará que los errores no vuelvan a aparecer en versiones futuras del juego.

- Usar pruebas de estrés para verificar la estabilidad del código. Si el juego se mantiene estable en condiciones extremas, por ejemplo, cuando se generan 5000 personajes no controlables y se destruyen a cada segundo, o cuando se recarga un escenario 300 veces seguidas, esto reduce la probabilidad de que los jugadores experimenten luego errores de ejecución cuando realicen algo inusual en el juego.

- Implementar tests automatizados en los módulos o componentes más importantes, es decir, en aquellos fragmentos de código más complejos y más utilizados.

- Concentrar los esfuerzos en probar las diferentes partes de que se compone el juego cuando parezca que no es posible realizar una prueba funcional a alto nivel. Por ejemplo, es posible que no se pueda verificar si funciona correctamente la inteligencia artificial de forma automática pero sí es posible probar si un enemigo alcanza el estado *muerto* cuando sus puntos de vida llegan a 0.

- Mantener las pruebas tan simples como sea posible. La facilidad de mantenimiento y la extensibilidad de las mismas son factores cruciales.

[3] http://nunit.org
[4] http://junit.org

Verificar y validar que todos los test desarrollados se ejecutan correctamente

En un entorno automatizado, todos los tests desarrollados en el proyecto han de ser superados con éxito. Todo el mundo acepta que el código que no compila no va a funcionar, por el mismo motivo el código que tiene errores en sus tests tampoco va a funcionar. Aceptar código cuyos tests poseen errores es peligroso y nos puede conducir a la producción de software de mala calidad.

2.2. Beneficios y costes

A continuación se muestra un resumen de los beneficios que aporta la integración, entrega y despliegue continuo, según Humble y Farley (2010):

- **Aumentar la calidad del producto**: Al integrar el código de forma continua, el riesgo de que haya errores y el tiempo en solucionarlos, si los hay, disminuye. Esto se traduce en un aumento de la calidad del producto, ya que hay más probabilidades de éxito a la hora de solucionar los fallos de programación.

- **Automatizar procesos repetitivos**: Los procesos manuales, cuando se realizan frecuentemente, ocasionan una pérdida de tiempo y puede convertirse en una fuente inagotable de problemas. Automatizar estos procesos mediante *scripts* hace que las personas no necesiten perder el tiempo en tener que realizarlos a mano.

- **Lograr un aumento de la productividad y la motivación del equipo**: Al tener un mayor control sobre el desarrollo y del estado del proyecto, la productividad del equipo aumenta puesto que los desarrolladores solo tienen que ocuparse de programar. El entorno ya se encarga de integrar, compilar, testear y medir la calidad de forma automática, evitando que ningún miembro del equipo ocupe su tiempo en realizar estas tareas de forma manual.

En términos de negocio, la combinación de estos tres beneficios aporta a la empresa lo siguiente:

- Se reducen riesgos y gastos generales en todo el proceso de desarrollo y despliegue.

- Aumenta la reputación de la empresa, al poder asegurar la alta calidad del software producido por sus equipos de desarrollo.

A continuación se muestra un listado de todas los costes que se han de asumir en la integración, entrega y despliegue continuo:

- Se requiere un inversión de tiempo y conocimiento para instalar y configurar la plataforma de forma adecuada para poder llevar a cabo los procesos de integración, entrega y despliegue continuo sobre los proyectos que lleva la empresa.
- Se requiere una inversión de dinero para poder instalar y desplegar la plataforma en un servidor con las características hardware necesarias para poder ejecutar los procesos de integración, entrega y despliegue continuo sobre los proyectos que lleva la empresa.
- Se ha de asumir el sobrecoste de mantener el sistema de integración, entrega y despliegue continuo.

2.3. Herramientas disponibles

En esta sección se va a realizar un análisis de las herramientas más comúnmente utilizadas por los equipos de desarrollo y, al final, se muestra una tabla comparativa para clarificar las principales características de cada una.

2.3.1. Jenkins

Jenkins es una plataforma de integración continua popular y de código abierto y gratuita desarrollado en Java y surgió como *fork* de *Hudson* después de una disputa entre la comunidad y *Oracle*. Jenkins permite automatizar el proceso de desarrollo de software mediante la integración continua y la entrega continua. Es un servidor que se despliega en *servlets*, como por ejemplo *Apache Tomcat*. Tiene soporte para descargar el código fuente de los siguientes repositorios: *CVS*, *Subversion*, *Git*, *Mercurial* y *Perforce*, y puede ejecutar proyectos basados en *Apache Ant*, *Gradle* y *Apache Maven*, así como comandos en la Terminal. Las compilaciones pueden ser lanzadas por distintas formas, por ejemplo, cuando alguien realiza un *commit* en un repositorio o porque ha sido programado mediante un sistema similar a *cron*. La funcionalidad de *Jenkins* puede ser extendida por *plug-ins*. Su principal desventaja es la complejidad, ya que se necesita tiempo para ajustarla adecuadamente para cada proyecto. Además el modelo servidor-esclavos de *Jenkins* produce los siguientes problemas a la hora de escalar horizontalmente:

- Cada esclavo tiene que ejecutar el *pipeline* desde 0, lo que típicamente involucra clonar el repositorio y realizar el compilado para evitar que las compilaciones incrementales fallen.
- *Jenkins* tiene un rendimiento pobre en modelos escalonados que implican muchos esclavos.

GameCraft ofrecerá una interfaz más sencilla, alineada con el uso que se le va a dar en el mundo del desarrollo de los videojuegos, pues su objetivo

principal es ser una plataforma para automatizar procesos en este ámbito. Además, la arquitectura de microservicios de la plataforma a desarrollar ofrecerá una escalabilidad conforme a la demanda que se está produciendo, de tal forma que todas las instancias desplegadas de los microservicios estarán siendo usadas y cuando una instancia deje de ser usada, entonces, será eliminada.

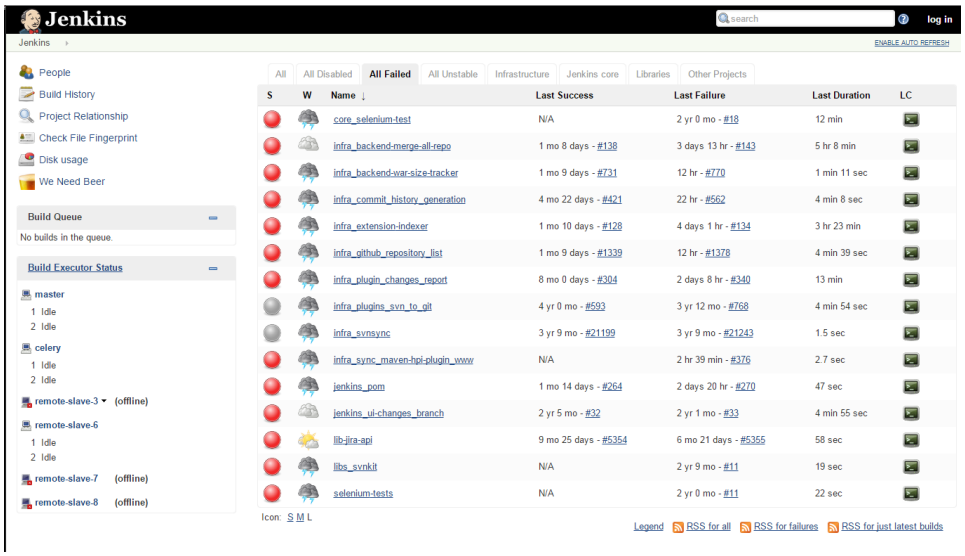

Figura 2.2: Interfaz de usuario de *Jenkins*
Imagen obtenida desde `https://testeandosoftware.com/jenkins-servidor-de-integracion-continua-gratuito/`

2.3.2. Circle CI

Circle CI es una plataforma de integración continua y despliegue en la nube. Similar a *Jenkins* pero ofreciendo una interfaz de usuario más sencilla, permite automatizar el proceso de desarrollo de software mediante la integración continua y la entrega continua. Las compilaciones se lanzan mayoritariamente cuando alguien realiza un *commit* en un repositorio.

Las principales desventajas de *Circle CI* son:

- Sólo ofrece la posibilidad de desplegar la plataforma de integración continua en entornos *GNU/Linux Ubuntu* y *MacOS*, descartando *Windows*.
- Solo soporta un conjunto limitado de lenguajes (*Go*, *Haskell*, *Java*, *PHP*, *Python*, *Ruby* y *Scala*) y el usuario necesita instalar *plug-ins* para añadir compatibilidad con nuevos lenguajes.
- No es gratuito.

GameCraft será de código abierto y gratuito, y tendrá soporte para los lenguajes de programación y los motores de videojuegos más populares. Además, la plataforma de integración continua a desarrollar será compatible tanto con *Windows* como *GNU/Linux* y *MacOS*, por lo que se podrán generar versiones de un mismo juego para estas tres plataformas, al contrario que con *CircleCI*, que debido a que no posee soporte con *Windows*, solo sería posible generar versiones del juego para *MacOS* y para *GNU/Linux*. Algo que sin duda es un obstáculo importante cuando hablamos de desarrollo de videojuegos.

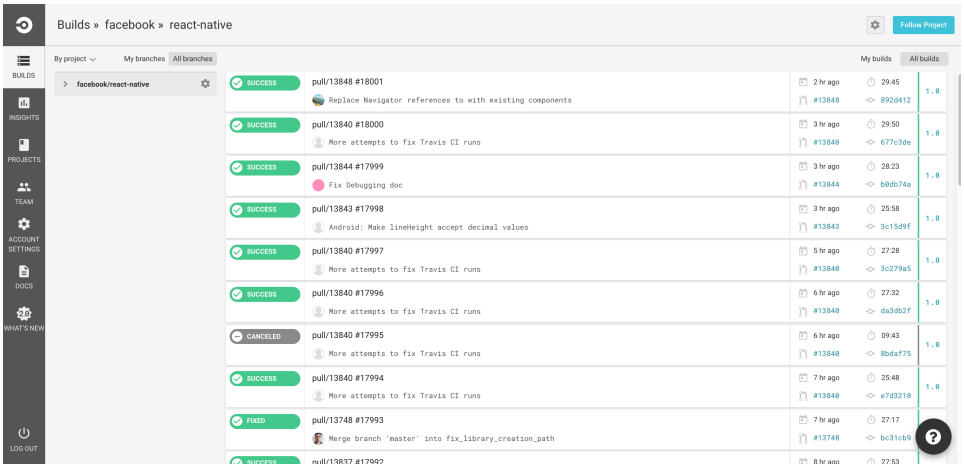

Figura 2.3: Interfaz de usuario de *Circle CI*
Imagen obtenida desde `https://github.com/marketplace/circleci`

2.3.3. Travis CI

Travis CI es una plataforma de integración continua en la nube, similar a *CircleCI*, enfocada sobre todo a ofrecer una interfaz de usuario sencilla donde el usuario no tenga que perder mucho tiempo en preparar el entorno para su proyecto. Las compilaciones se lanzan mayoritariamente cuando alguien realiza un *commit* pero también se puede configurar para que se realicen de forma periódica a través del uso de una sentencia *cron*.

Las principales desventajas de *Travis CI* son:

- No es gratuito.
- No permite un elevado grado de personalización, y para preparar el entorno, se necesita instalar *plug-ins* de terceros.

El hecho de que *GameCraft* siga una arquitectura de microservicios va a facilitar la posibilidad de integrar nuevas características a la plataforma. Tan solo sería necesario desarrollar un nuevo servicio con la nueva funcionalidad implementada y desplegarlo, sin necesidad de tener que parar o reiniciar toda

la plataforma en el momento en que se está desplegando el nuevo servicio (*hot swap*).

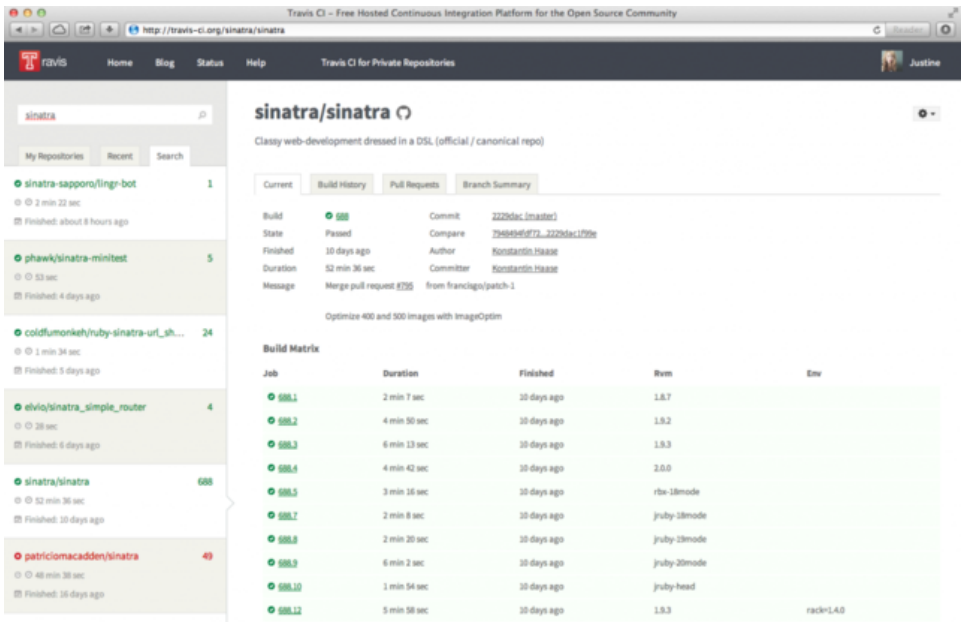

Figura 2.4: Interfaz de usuario de *Travis CI*
Imagen obtenida desde `https://bitnami.com/stack/travisci`

2.3.4. Unity Cloud Build

Unity Cloud Build es una plataforma de integración continua en la nube ofrecida directamente para los usuarios del entorno de desarrollo *Unity*. A diferencia de las otras plataformas comentadas anteriormente, esta solución sí está pensada directamente para ser utilizada por los desarrolladores de videojuegos. Su funcionamiento es bastante sencillo: primero se elige el proyecto de *Unity* que se quiere utilizar en la plataforma; posteriormente, el usuario debe introducir la URL del repositorio donde está almacenado el proyecto; cada vez que alguien haga un nuevo *commit*, *Unity Cloud Build* compilará y testeará el proyecto. Las desventajas de *Unity Cloud Build* son:

- No es gratuito. Para utilizar este servicio es necesario pagar por una licencia Plus (35 dólares al mes) o Professional (125 dolares al mes). Este servicio no se puede utilizar con la licencia Personal gratuita.
- No se puede personalizar ni añadir nuevas funciones a través de *plug-ins*.
- Sólo es compatible con *Unity* y no se puede utilizar para juegos desarrollados con otro entorno de desarrollo, lo que es un factor limitante decisivo.

- No permite analizar la calidad del código.

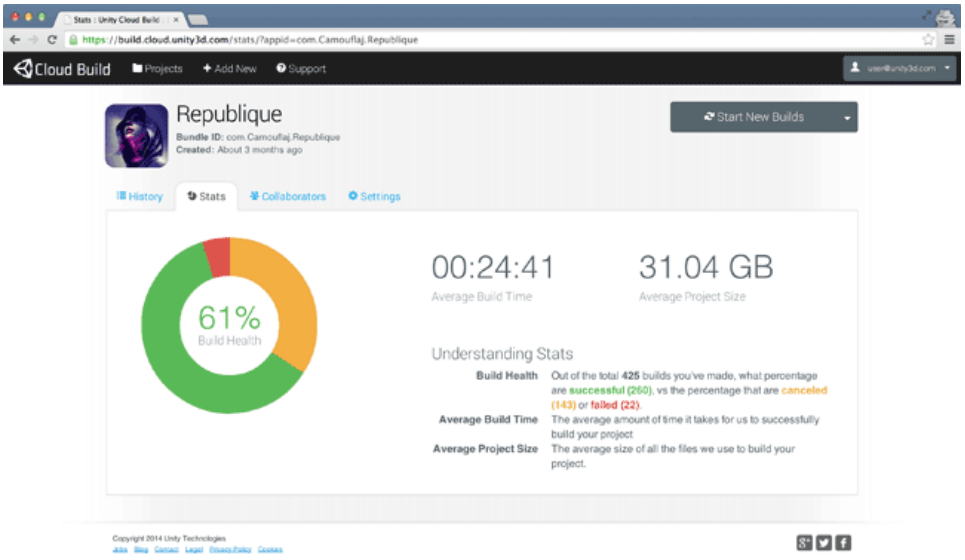

Figura 2.5: Interfaz de usuario de *Unity Cloud Build*
Imagen obtenida desde
https://www.gamasutra.com/view/news/223799/Unity_rolls_out_version_46_opensource_initiative_Unity_Cloud_Build.php

2.3.5. Microsoft Team Foundation Server

Microsoft Team Foundation Server es un paquete de plataformas software encaminadas a facilitar la gestión de proyectos software de una empresa. *Microsoft Team Foundation Server* posee los siguientes módulos:

- Control de versiones de código fuente.
- Servidor de integración continua.
- Registro de incidencias.
- Gestor de requisitos y tareas.
- Gestor de recursos humanos

A diferencia de las otras plataformas comentadas anteriormente, *Microsoft Team Foundation Server* se centra en ofrecer una herramienta que sea de ayuda tanto a los desarrolladores como a los gestores de proyecto. El acceso a *TFS* se realiza desde *Visual Studio* a través de un *plug-in* incorporado, pero además es posible acceder a los datos a través de una página web propia o

a través de la integración con *Excel* y *Microsoft Project*. *Team Foundation Server* no es simplemente una herramienta para el control de versiones de código fuente: su alcance va mucho más allá de lo que abordamos en este trabajo, facilitando la gestión completa del ciclo de vida de la aplicación, desde la fase de diseño hasta las pruebas, pasando por la integración continua o la calidad del código.

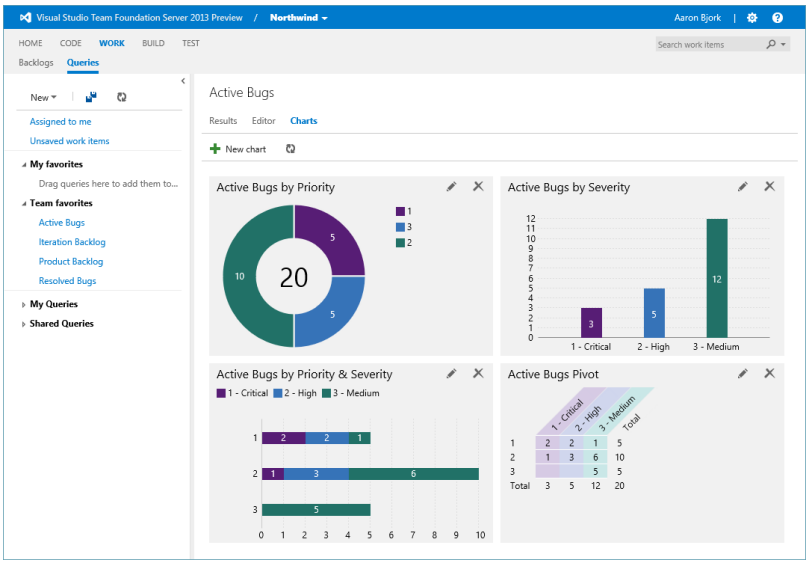

Figura 2.6: Interfaz de usuario de *TFS*
Imagen obtenida desde `https://www.developpez.com/actu/61143/`

2.3.6. Comparativa entre las herramientas

En la tabla 2.1 se muestra una comparativa de las características de las herramientas que se han analizado anteriormente y puede observarse como *GameCraft* contempla proveer la gran mayoría de ellas:

	Jenkins	**Circle**	**Travis**	**Unity Cloud**	**TFS**	**GameCraft**
Soporta *macOS*	✓	✓	✓	✓		✓
Soporta *GNU/Linux*	✓	✓	✓	✓		✓
Soporta *Windows*	✓			✓	✓	✓
Gratuito	✓					✓
Código abierto	✓					✓
Internacionalización	✓			✓	✓	
Integración con plugins	✓	✓	✓		✓	
Soporte al engine *Unity*				✓		✓
Soporte al engine *Unreal*						✓
Soporte al engine *Monogame*						✓
Soporte al engine *Libgdx*						✓
Soporte a otros engines	✓	✓	✓		✓	✓
Análisis de calidad del código	✓	✓	✓		✓	

Tabla 2.1: Comparación de las plataformas de integración continua analizadas

Capítulo 3

Metodología y gestión del proyecto

Medir el progreso de un programa por el número de líneas de código, es como medir la construcción de una aeronave por su peso.

Bill Gates

Resumen: Este capítulo aborda con detalle las metodologías y tecnologías utilizadas en el desarrollo y presenta una planificación temporal de todo el proyecto en forma de diagrama de Gantt.

Para desarrollar, implementar y desplegar este proyecto, aparte de recurrir a un modelo de desarrollo iterativo e incremental, se han utilizado las siguientes tecnologías y librerías.

Para la parte trasera (*back*) del sistema, se ha utilizado el lenguaje de programación *Java*[1], junto con el armazón *Spring Boot*[2] y el uso de *Maven*[3] para la gestión de dependencias.

Para la parte delantera (*front*), que será una interfaz web, se ha utilizado el lenguaje *HTML*[4], junto con *Bootstrap*[5] y *JQuery*[6].

Para el almacenamiento y recuperación de datos, se ha confiado en el uso de la base de datos *MySQL*[7], en *ElasticSearch*[8] para proporcionar búsquedas

[1] `https://www.java.com/es/`
[2] `http://spring.io/projects/spring-boot`
[3] `https://maven.apache.org/`
[4] `https://www.w3.org/html/`
[5] `https://getbootstrap.com/`
[6] `https://jquery.com/`
[7] `https://www.mysql.com/`
[8] `https://www.elastic.co/`

más rápidas utilizando índices y en *Hazelcast*[9] para el caché de peticiones.

El microservicio de registro[10] y el microservicio de enrutamiento [11] es trabajo ya realizado por la comunidad y que se ha incorporado en este proyecto. Estos proyectos poseen licencia abierta, y por tanto, no hay ningún problema en usarlos siempre y cuando se mencione este hecho.

Para el despliegue de la plataforma, se ha utilizado el popular sistema *Docker*.

3.1. Tecnologías y herramientas empleadas

3.1.1. *Java*

Java es un lenguaje de programación orientado a objetos, cuya principal finalidad es la de ser utilizado para desarrollar aplicaciones multiplataforma, es decir, aplicaciones que comparten el mismo código pero que pueden ser ejecutadas en *Windows*, *GNU/Linux* y *MacOS*.

GameCraft ha sido desarrollado utilizando *Java 8*, por lo que es necesario tener dicha versión o superior para poder desplegar la plataforma en una máquina y poder hacer uso de ella.

3.1.2. *Spring Boot*

Spring Boot (Walls (2016)) es un armazón para el desarrollo de aplicaciones web, de código abierto, para ser utilizado junto al lenguaje Java e incluye los siguientes módulos, listos para ser utilizados de forma sencilla por la aplicación:

- Programación orientada a aspectos: permite implementar preocupaciones transversales.
- Autenticación y autorización: procesos de seguridad configurables que soportan una gama de estándares, protocolos, herramientas y prácticas a través del subproyecto *Spring Security*.
- Convención sobre la configuración: en el módulo *Spring Roo* se ofrece una solución rápida de desarrollo de aplicaciones para aplicaciones empresariales basadas en *Spring*.
- Acceso a datos: trabaja con sistemas de administración de bases de datos relacionales en la plataforma *Java* usando *Java Database Connectivity (JDBC)* y herramientas de mapeo relacional de objetos y con bases de datos *NoSQL*

[9] `https://hazelcast.com/`
[10] `https://github.com/jhipster/jhipster-registry`
[11] `https://github.com/jhipster/jhipster-sample-app-gateway`

- *Inversión del contenedor de control*: configuración de los componentes de la aplicación y gestión del ciclo de vida de los objetos Java, realizada principalmente a través de la inyección de dependencias

- *Mensajería*: registro configurativo de objetos de escucha de mensajes para el consumo transparente de colas de mensajes a través de *Java Message Service (JMS)*.

- Modelo-vista-controlador: un marco basado en *HTTP* y *servlet* para aplicaciones web y servicios web *RESTful* (transferencia de estado de representación).

- Implementación para hacer llamadas a procedimientos remoto de objetos *Java*.

- *Gestión de transacciones*: unifica varias *API* de gestión de transacciones y coordina transacciones para objetos Java.

- *Testing*: clases que facilitan bastante la realización de pruebas de unidad y de integración.

GameCraft utiliza *Spring* como elemento básico para la arquitectura de microservicios en los siguientes aspectos: para la implementación de seguridad mediante el uso de *tokens*, para la exposición de *endpoints*, para la autenticación y creación de usuarios y para acceder y guardar los datos en cada una de las bases de datos que corresponden a cada microservicio.

3.1.3. *Maven*

Maven es una herramienta creada en el año 2001, de código abierto, para simplificar el proceso de construcción de proyectos. *Maven* describe el proyecto de software a construir, sus dependencias de otros módulos, y el orden de construcción de los elementos. Viene con objetivos ya definidos para realizar ciertas tareas, como la compilación del código y su empaquetado.

GameCraft utiliza *Maven* para construir los microservicios.

3.1.4. *Bootstrap*

Bootstrap es un armazón de código abierto para el diseño de páginas web. Contiene elementos de diseño como formularios, botones y menús de navegación , así como extensiones de JavaScript adicionales. A diferencia de muchos armazones web, solo se ocupa del desarrollo front-end.

GameCraft utiliza *Bootstrap* para el diseño de la interfaz de la página web que se despliega en la plataforma.

3.1.5. *jQuery*

jQuery es una biblioteca multiplataforma que proporciona nuevas funciones y operaciones *JavaScript* para interactuar con los elementos *HTML* de la página, manipular el árbol *DOM*, manejar eventos, desarrollar animaciones y agregar interacciones.

GameCraft utiliza *jQuery* para el desarrollo de las funcionalidades de los componentes presentes en la interfaz de la página web que se despliega en la plataforma.

3.1.6. *MySQL*

MySQL es un sistema de gestión de bases de datos de tipo relacional gratuito que ofrece alta velocidad en lectura de datos, lo que lo hace ideal para aplicaciones web. Los datos almacenados se archivan en tablas separadas, en vez de que todo se junte en un mismo sitio, ganando velocidad y flexibilidad. Las tablas se conectan por relaciones definidas para hacer posible combinar datos almacenados en varias tablas.

GameCraft utiliza *MySQL* como medio principal para almacenar toda la información de la plataforma (usuarios, motores, notificadores, etc.).

3.1.7. *Liquibase*

Liquibase[12] es una herramienta que permite rastrear, gestionar y aplicar cambios en los esquemas de las bases de datos. Ofrece un histórico de cambios realizados, por lo que todas las modificaciones realizadas en la base de datos quedan registradas, posibilitando así además poder volver atrás en caso de que una modificación afecte negativamente a la integridad de la base de datos.

GameCraft utiliza *Liquibase* para registrar todos los cambios que se van haciendo a las bases de datos.

3.1.8. *ElasticSearch*

ElasticSearch (Gheorghe et al. (2015)) es un servidor que provee de un motor de búsqueda de texto completo, escalable, distribuido y accesible a través de *endpoints RESTful*, devolviendo siempre respuestas en formato *JSON*. Tiene una gran velocidad de respuesta y al ser distribuido, lo hace la opción ideal para combinar un sistema de búsqueda como este dentro de una arquitectura de microservicios.

GameCraft utiliza *ElasticSearch* para proporcionar los sistemas de búsqueda cada vez que se quiere obtener información de una entidad almacenada en la base de datos (un usuario en concreto, un motor, etc.)

[12] `https://www.liquibase.org/`

3.1.9. *Hazelcast*

Hazelcast (Raible (2016)) es una herramienta capaz de distribuir un conjunto de datos entre distintos nodos de forma balanceada y ofrece tolerancia a fallos, es decir, si un nodo se cae no se produce pérdida de datos, puesto que además de su partición de datos los nodos se reparten una copia de seguridad del resto de nodos.

GameCraft utiliza *HazelCast* para implementar un sistema de *caché* para las peticiones que se realizan a los *endpoint* de los microservicios.

3.1.10. *Logback*

Logback[13] es una librería para *Java* que ofrece un sistema de *logging*, es decir, ofrece la forma de crear informes de estado que son útiles para depurar y auditar el estado de una aplicación en un momento determinado.

GameCraft utiliza *Logback* para enviar mensajes de estado que se muestran tanto en la Terminal como en un archivo de texto de cada uno de los microservicios desplegados.

3.1.11. Microservicio de registro

El microservicio de registro (Raible (2016)) tiene tres objetivos principales:

- Es un servidor *Eureka*, cuyo propósito es descubrir la dirección IP de los microservicios y ubicarlos en la red.
- Es un servidor de *Spring Cloud Config*, cuyo propósito es centralizar y distribuir la configuración inicial para que cada microservicio pueda funcionar correctamente.
- Proporciona un panel de administración para obtener métricas del estado de cada microservicio.

Todas las funcionalidades anteriores se han empaquetado dentro del microservicio de registro, y es un elemento importante en *GameCraft* ya que sin él, el resto de microservicios no podrían funcionar correctamente.

3.1.12. Microservicio de enrutamiento

GameCraft utiliza el microservicio de enrutamiento (Raible (2016)) para hacer posible el enrutamiento HTTP y el balanceo de carga entre los diferentes microservicios, mantener la calidad de servicio, ofrecer seguridad y documentación API para todos los microservicios. Al igual que el microservicio de

[13] `https://logback.qos.ch/`

registro, el microservicio de enrutamiento tiene un panel de administración donde además se puede acceder a las bases de datos registradas por cada microservicio y manipularlas, gestionar los usuarios registrados en la plataforma, etc. Este microservicio utiliza el armazón *Zuul Proxy* para posibilitar todas las funciones anteriormente descritas.

3.1.13. *Swagger*

Swagger[14] es un armazón de código abierto que facilita la labor de diseñar y documentar los *endpoints* de los servicios web. Con *Swagger* se puede describir, producir, consumir y visualizar *APIs*, por lo que es interesante para los desarrolladores, pero también desde el punto de vista de un tercero que pudiese consumir tu API, o un usuario que busca información sobre un error, etc.

GameCraft utiliza *Swagger* para documentar todos los *endpoints* expuestos por los microservicios aunque su acceso solo es posible realizarlo a través del microservicio de enrutamiento y no a través de *Gamecraft UI*.

3.1.14. *Docker*

La idea básica de *Docker* (Gallagher (2015)) es la de construir contenedores que almacenan tanto la aplicación que el usuario quiere distribuir como las librerías y otras dependencias que esa aplicación necesita para funcionar correctamente. Gracias a que los contenedores son ligeros y portables que las máquinas virtuales, al no necesitar instalar un *hypervisor* ni un sistema operativo para que puedan funcionar, facilitan mucho las etapas de desarrollo, testing , integración, packaging...

Docker permite meter en un contenedor ("una caja", algo auto contenido, cerrado) todas las dependencias que se necesita para ejecutar una aplicación (*Java*, *Maven*, ...) y la propia aplicación. Así, una persona puede mover ese contenedor a cualquier máquina que tenga instalado *Docker* y ejecutar la aplicación sin tener que hacer nada más, ni preocuparme de qué versiones de software tiene instalada esa máquina ya que dentro del contenedor estarán todas las librerías y cosas que necesita dicha aplicación para funcionar correctamente.

GameCraft utiliza *Docker* para ofrecer una forma más fácil de desplegar la plataforma, en vez de invocar uno a uno todos los microservicios. Gracias a *Docker*, se puede desplegar toda la plataforma con un único comando.

[14] `https://swagger.io/tools/open-source/getting-started/`

3.1.15. *Ribbon*

Netflix Ribbon[15] es una biblioteca que proporciona principalmente algoritmos de balanceo de carga de procesos, un servidor de detección de servicios en la red y un mecanismo de detección dinámica de si hay servidores dentro de la red que están activos y en ejecución y los que no lo están.

GameCraft utiliza *Netflix Ribbon* para el balanceo de carga entre los diferentes microservicios, para de esta forma, poder ser escalable horizontalmente dependiendo de la demanda que tengan los microservicios.

3.2. Planificación temporal

Se ha planificado el proyecto de forma incremental, dividiendo el tiempo total en varios incrementos. Cada incremento estará dividido en seis etapas: análisis, diseño, implementación, prueba, despliegue y documentación.

En los incrementos iniciales se analizan todas las posibles alternativas que podrían ser útiles a la hora de implementar el producto y se han documentado para, en un futuro, determinar la viabilidad de cada una de ellas. Posteriormente, en los incrementos intermedios se analiza la alternativa elegida y se desarrolla un prototipo. Después de comprobar la viabilidad del prototipo y la alternativa, se desarrolla el software final.

En las últimas etapas, el esfuerzo se concentró en dejar documentado todo el proceso y realizar las pruebas pertinentes.

Las etapas recorridas aparecen descritas a continuación:

- **Estudio de la técnica y análisis de requisitos y componentes**. Durante esta etapa, se intenta explicar lo que debería hacer el software para satisfacer las necesidades de los usuarios que lo utilizarán y se indica cual es la interfaz de usuario más adecuada para el programa. La captura, análisis y especificación de requisitos es una parte crucial: de esta etapa depende en gran medida que el software producido tenga la calidad esperada. Estos requisitos se determinan tomando en cuenta las necesidades a satisfacer del usuario final.

- **Diseño de componentes**. Se decide como funcionará, de forma general, el software sin entrar en detalles e incorporando consideraciones de la implementación tecnológica. Generalmente se realiza en base a diagramas que permiten describir las interacciones entre las entidades y se diseñarán los componentes que darán respuesta a las funcionalidades del software.

- **Implementación de componentes**. En esta etapa, se programa el código para el producto, materializando así el resultado de las etapas

[15] `https://github.com/Netflix/ribbon/wiki`

anteriores.

- **Prueba de componentes**. Consiste en comprobar que los componentes diseñados funcionen correctamente en el momento en que se van implementando en el sistema. Cada vez que se completa el desarrollo de cada módulo del software, se probará de manera integral para comprobar que el programa funciona correctamente y cumple con los objetivos estipulados

- **Prueba de aplicación y despliegue**. Consiste en comprobar que el software, con todos sus componentes ya implementados y probados, realice correctamente las tareas indicadas en la especificación de los requisitos. Si las pruebas resultan exitosas, entonces se puede comenzar a desplegar la aplicación a los usuarios finales.

- **Documentación**. En esta etapa, de naturaleza trasversal, se genera la documentación del propio desarrollo del software y de la gestión del proyecto, pasando por modelos UML, diagramas de casos de uso, pruebas, manuales de usuario y técnicos, etc. con el objetivo de que dicha documentación pueda ser fácilmente alterada ante posibles correcciones y ampliaciones al sistema.

Dado que durante el proceso surgió una necesidad no contemplada de mejorar el proyecto desarrollado para hacerlo más usable y solucionar fallos de programación, el periodo contemplado inicialmente en la estimación no se ha cumplido. Se han necesitado un par de meses más para las etapas de diseño e implementación de los componentes, y por este motivo, el proyecto se ha alargado hasta ser finalmente cerrado en septiembre de 2018.

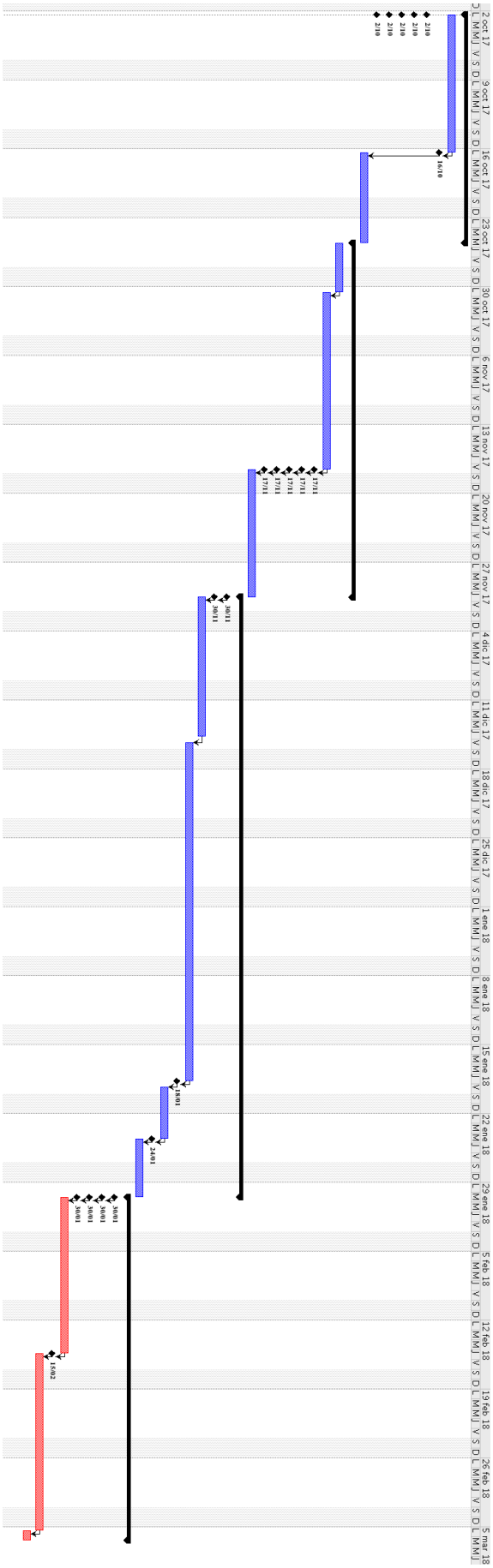

Figura 3.1: Diagrama de Gantt

Capítulo 4

Desarrollo del sistema

Antes de que el software pueda ser reutilizable primero debe ser utilizable

Ralph Johnson

Resumen: Este capítulo contiene el grueso de la contribución de este trabajo, exponiéndose todo lo relativo a las distintas fases del proceso de desarrollo software del sistema, desde la idea inicial a la implementación, pasando por el análisis de los requisitos, la arquitectura y el diseño de la plataforma.

En este capítulo se documenta el proceso de desarrollo de *GameCraft*, su arquitectura, los actores que participan en el sistema y la especificación formal de sus requisitos.

4.1. Actores del sistema

Los actores representan el papel jugado por un usuario o por cualquier otro sistema exterior que interactúa con *GameCraft*. En las tablas siguientes figuran los actores del sistema.

ACT-01	**Anónoimo**
Descripción	Este actor representa a la persona que se encuentra en la pantalla de inicio de sesión pero todavía no se ha autenticado con su cuenta.

Tabla 4.1: ACT-01: Anónoimo

ACT-02	**Usuario**
Descripción	Este actor representa a la persona que va a acceder a la plataforma para ejecutar los *pipelines* y visualizar los informes de ejecución.

Tabla 4.2: ACT-02: Usuario

ACT-03	**Administrador**
Descripción	Este actor representa a la persona que tiene privilegios para configurar y acceder a toda la plataforma. El administrador puede manipular los usuarios, los notificadores, los motores, los proyectos y los *pipelines*.

Tabla 4.3: ACT-03: Administrador

ACT-04	**Notificador**
Descripción	Este actor representa un notificador que se utilizará para enviar notificaciones a través de correos electrónicos, mensajes por *Telegram*, *IRC* o *Slack* y *tuits*.

Tabla 4.4: ACT-04: Notificador

ACT-05	**Engine**
Descripción	Este actor representa el motor, o entorno de desarrollo del videojuegos, para llevar a cabo la invocación de su compilador y generar los binarios del proyecto.

Tabla 4.5: ACT-05: Engine

4.2. Casos de uso

4.2.1. Introducción

Los diagramas de casos de uso representan el comportamiento que el sistema debe presentar utilizando un lenguaje sencillo y no demasiado técnico. Cada caso de uso se centra en describir cómo alcanzar una única meta o tarea. A raíz de esto, un caso de uso se puede entender como una interacción entre actor y sistema en respuesta a un evento. El diagrama de casos de uso de *GameCraft* aparece en la Figura 4.2.

Los casos de uso se relacionan entre sí a través de varios tipos de relaciones:

- *Comunica («communicates»).* Asociación entre un actor y un caso de uso para describir la involucración en dicho escenario. Se suele representar con una línea continua.

- *Usa («uses» o «include».* Inclusión del comportamiento de un escenario de un caso de uso en otro caso de uso. Se representa con una línea discontinua y con la etiqueta *«include»*.

- *Extiende («extends»).* Denota que un caso de uso es una especialización de otro caso de uso. Se representa con una línea discontinua y con la etiqueta *«extends»*.

Además, un actor puede heredar los casos de uso que comunican con otro actor mediante el uso de la notación de generalización que se representa con una flecha continua que sube desde el actor «hijo» al «padre». Para representar los límites del sistema con el exterior, se suelen encuadrar todos los casos de uso dentro de una caja.

En la Figura 4.1 se puede ver una representación gráfica de las notaciones usadas.

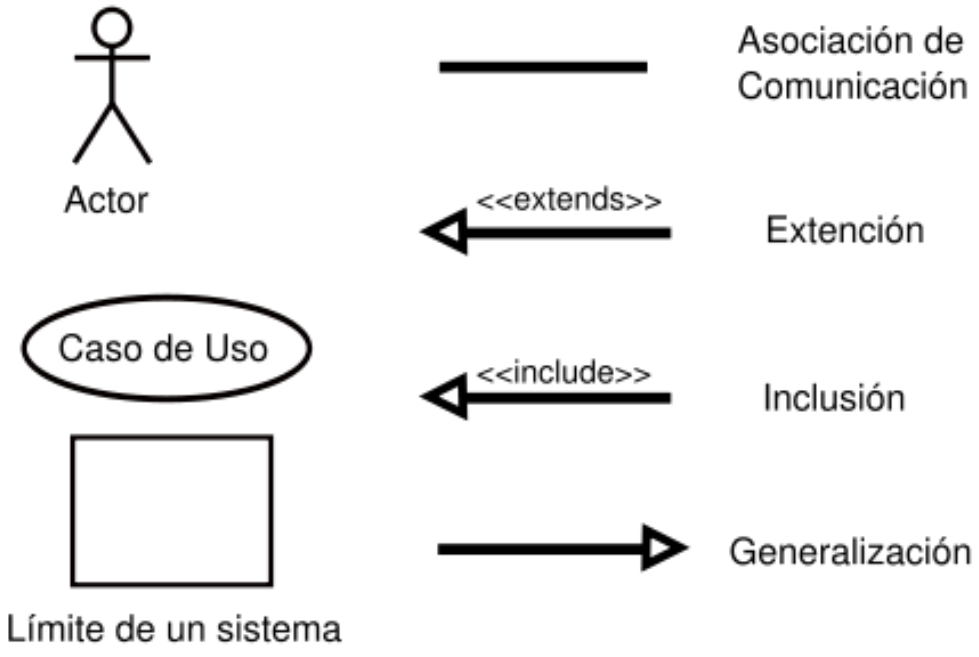

Figura 4.1: Notación de casos de uso
Imagen obtenida desde `https://es.wikipedia.org/wiki/Caso_de_uso`

4.2.2. Diagrama

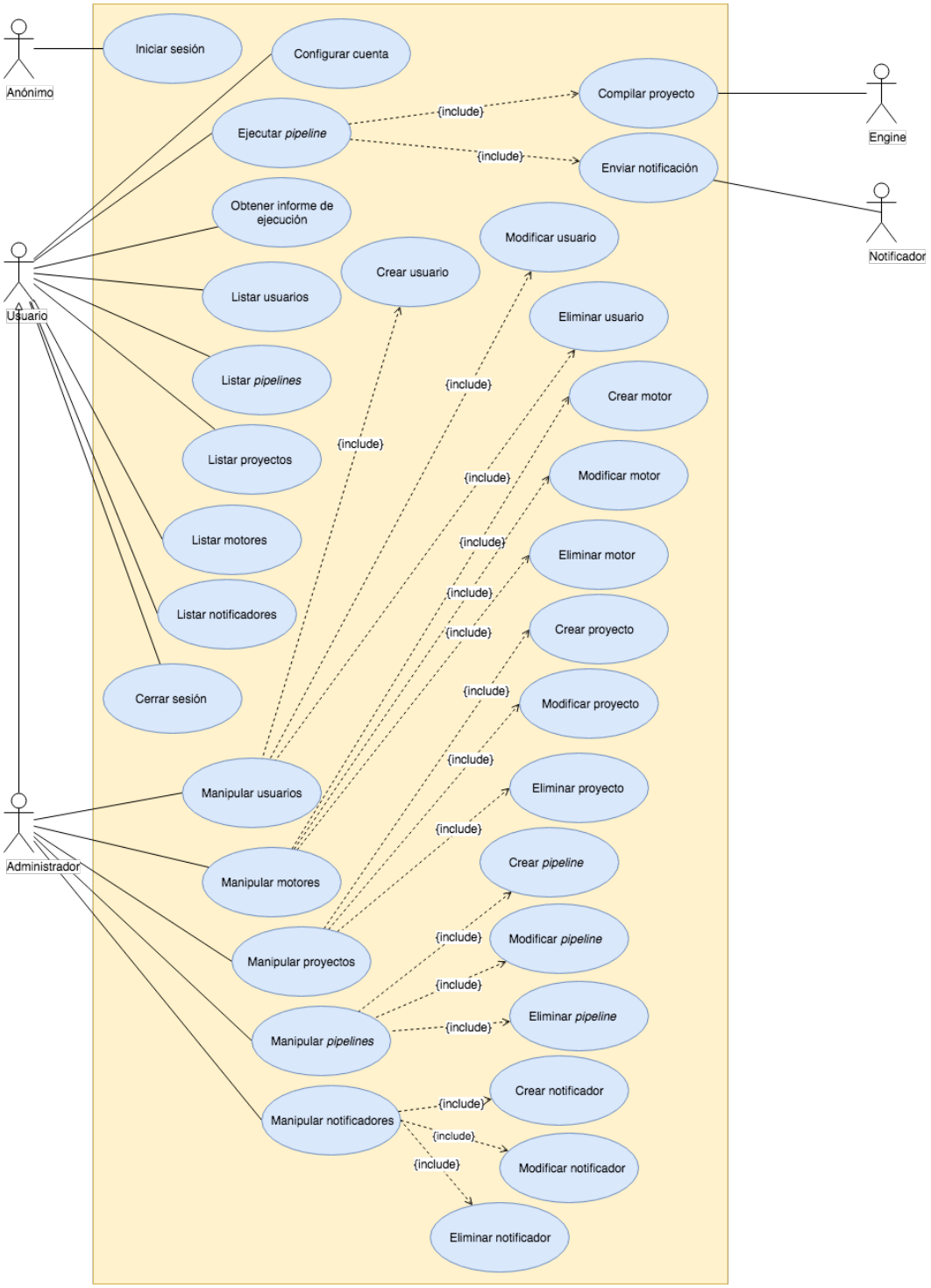

Figura 4.2: Diagrama de casos de uso de GameCraft

4.2.3. Especificación

CU-01	**Iniciar sesión**
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 35.- Iniciar sesión en el sistema.
Precondición	Ninguna.
Descripción	El sistema debe permitir a un usuario poder acceder con una cuenta de usuario existente en la base de datos del sistema.
Secuencia Normal	1. El sistema pregunta al usuario por el nombre de usuario y la contraseña. 2. El usuario introduce su nombre de usuario. 3. El usuario introduce su contraseña. 4. El usuario confirma la acción haciendo clic en el botón «*Log in*». 5. El sistema comprueba la validez de los datos introducidos. 6. El sistema redirige al usuario a la página principal en caso de que la validez anterior haya sido exitosa.
Postcondición	El sistema debe haber producido un token JWT que se utilizará en cada petición que se haga a la *API* de los microservicios y debe redigirir al usuario a la página principal de la plataforma.
Excepción	Puede darse el caso de que la cuenta de usuario no exista o se haya introducido incorrectamente, hecho por el cual se mostrará un error y no dejará al usuario iniciar sesión.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.6: CU-01: Iniciar sesión

CU-02	**Configurar cuenta**
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 02.- Modificar usuario.
Precondición	Haber iniciado la sesión con una cuenta válida.
Descripción	El sistema debe permitir a un usuario poder modificar los datos personales de la cuenta con la que está iniciando sesión: nombre, apellidos, correo electrónico y contraseña.
Secuencia Normal	1. El sistema pregunta al usuario por los nuevos datos que quiere modificar. 2. El usuario introduce su nombre de pila. 3. El usuario introduce sus apellidos. 4. El usuario introduce su dirección de correo electrónico. 5. El usuario introduce su nueva contraseña. 6. El sistema comprueba la validez de los datos introducidos. 7. El sistema almacena los nuevos datos introducidos en caso de que la validez anterior haya sido exitosa.
Postcondición	El sistema debe haber almacenado en la base de datos la nueva información introducida de la cuenta de usuario que ha iniciado sesión.
Excepción	En caso de que la validez no haya sido satisfactoria, el sistema no modificará los datos almacenados en la base de datos y mostrará un error.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.7: CU-02: Configurar cuenta

CU-03	**Ejecutar *pipeline***
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 30.- Ejecutar *pipeline* de trabajo.
Precondición	Haber iniciado la sesión con una cuenta válida, que exista al menos un *pipeline* creado, un proyecto creado y un notificador creado.
Descripción	El sistema debe permitir a un usuario poder ejecutar un *pipeline*, un flujo de tareas que se irán ejecutando secuencialmente para poder compilar y producir un videojuego con éxito.
Secuencia Normal	1. El usuario selecciona el proyecto donde se encuentra el *pipeline* a ejecutar. 2. El usuario selecciona el *pipeline* que quiere ejecutar. 3. El usuario envía la solicitud de ejecución del *pipeline* al sistema. 4. El sistema procesa la solicitud y ejecuta las tareas del *pipeline*. 5. El sistema va recopilando los mensajes de estado de cada una de las tareas. 6. Tras la finalización de todas las tareas, el sistema recopila todos los mensajes de estado y envía una notificación a través del notificador elegido en el pipeline.
Postcondición	El sistema debe haber ejecutado todas las tareas descritas en el *pipeline* con éxito.
Excepción	En caso de que existe un error en el *pipeline*, el sistema devuelve un mensaje de error al usuario.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.8: CU-03: Ejecutar *pipeline*

CU-04	**Compilar proyecto**
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 40.- Ejecutar motor.
Precondición	Haber iniciado la sesión con una cuenta válida, que exista al menos un *pipeline* creado, un proyecto creado, un motor creado y un notificador creado.
Descripción	El motor debe compilar el proyecto seleccionado, con los argumentos proporcionados por el usuario.
Secuencia Normal	1. El usuario selecciona el proyecto donde se encuentra el *pipeline* a ejecutar. 2. El usuario selecciona el *pipeline* que quiere ejecutar. 3. El usuario envía la solicitud de ejecución del *pipeline* al sistema. 4. El sistema procesa la solicitud y ejecuta la tarea de compilación, entre otras tareas, invocando al motor.
Postcondición	El sistema debe haber generado el binario correspondiente a la compilación realizada.
Excepción	En caso de que existe un error en la compilación, el sistema devuelve un mensaje de error al usuario.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.9: CU-04: Compilar proyecto

CU-05	**Enviar notificación**
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 09.- Enviar notificación por correo electrónico. ▪ RQF 13.- Enviar notificación por *Telegram*. ▪ RQF 17.- Enviar notificación por *Slack*. ▪ RQF 21.- Enviar notificación por *IRC*. ▪ RQF 25.- Enviar notificación por *Twitter*.
Precondición	Haber iniciado la sesión con una cuenta válida, que exista al menos un *pipeline* creado, un proyecto creado, un motor creado y un notificador creado.
Descripción	El notificador debe enviar una notificación al usuario informando acerca del estado de ejecución del *pipeline*, si ha fallado en algún punto en concreto o si todo se ha ejecutado correctamente.
Secuencia Normal	1. El usuario selecciona el proyecto donde se encuentra el *pipeline* a ejecutar. 2. El usuario selecciona el *pipeline* que quiere ejecutar. 3. El usuario envía la solicitud de ejecución del *pipeline* al sistema. 4. El sistema procesa la solicitud y ejecuta las tareas del *pipeline*. 5. El sistema va recopilando los mensajes de estado de cada una de las tareas. 6. Tras la finalización de todas las tareas, el sistema recopila todos los mensajes de estado y envía una notificación a través del notificador elegido en el pipeline.
Postcondición	El sistema debe haber enviado una notificación.
Excepción	En caso de que existe un error en el envío de la notificación, el sistema devuelve un mensaje de error al usuario.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.10: CU-05: Enviar notificación

CU-06	**Obtener informe de ejecución**
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 32.- Generar informe del estado de la ejecución de un *pipeline*. ▪ RQF 33.- Visualizar informe del estado de la ejecución de un *pipeline*. ▪ RQF 34.- Descargar informe del estado de la ejecución de un *pipeline*.
Precondición	Haber iniciado la sesión con una cuenta válida y que exista, al menos, un *pipeline* que se haya ejecutado por primera vez.
Descripción	El sistema debe haber generado un informe de ejecución del *pipeline*, conteniendo los mensajes de estado de cada una de las tareas ejecutadas y el estado final de la ejecución, es decir, si se ha ejecutado correctamente o no.
Secuencia Normal	1. El usuario selecciona el proyecto donde se encuentra el *pipeline* a ejecutar. 2. El usuario selecciona el *pipeline* que quiere ejecutar. 3. El usuario envía la solicitud de ejecución del *pipeline* al sistema. 4. El sistema procesa la solicitud y ejecuta las tareas del *pipeline*. 5. El sistema va recopilando los mensajes de estado de cada una de las tareas. 6. Tras la finalización de todas las tareas, el sistema recopila todos los mensajes de estado y lo almacena para su posterior recuperación por parte del usuario.
Postcondición	El sistema debe haber generado correctamente el informe de ejecución.
Excepción	En caso de que existe un error en la generación del informe, el sistema devuelve un mensaje de error al usuario.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.11: CU-06: Obtener informe de ejecución

CU-07	**Listar usuarios**
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 41.- Listar usuarios.
Precondición	Haber iniciado la sesión con una cuenta válida y que exista al menos una cuenta creada.
Descripción	El sistema debe devolver al usuario un listado con todas las cuentas creadas en el sistema.
Secuencia Normal	1. El usuario solicita el listado completo de usuarios creados en la plataforma. 2. El sistema procesa la solicitud y devuelve el listado.
Postcondición	El sistema debe haber devuelto un listado con todos los usuarios creados.
Excepción	Ninguna.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.12: CU-07: Listar usuarios

CU-08	**Listar *pipelines***
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 42.- Listar *pipelines*.
Precondición	Haber iniciado la sesión con una cuenta válida y que exista al menos un proyecto creado y un *pipeline* creado.
Descripción	El sistema debe devolver al usuario un listado con todos los *pipelines* creados que corresponden a un proyecto en concreto.
Secuencia Normal	1. El usuario selecciona un proyecto para el cual quiere ver todos sus *pipelines* creados. 2. El usuario solicita el listado completo de *pipelines* creados en la plataforma para el proyecto en concreto. 3. El sistema procesa la solicitud y devuelve el listado.
Postcondición	El sistema debe haber devuelto un listado con todos los *pipelines* creados.
Excepción	Ninguna.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.13: CU-08: Listar *pipelines*

CU-09	**Listar proyectos**
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 43.- Listar proyectos.
Precondición	Haber iniciado la sesión con una cuenta válida y que exista al menos un proyecto creado.
Descripción	El sistema debe devolver al usuario un listado con todos los proyectos creados.
Secuencia Normal	1. El usuario solicita el listado completo de proyectos creados en la plataforma. 2. El sistema procesa la solicitud y devuelve el listado.
Postcondición	El sistema debe haber devuelto un listado con todos los proyectos creados.
Excepción	Ninguna.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.14: CU-09: Listar proyectos

CU-10	**Listar motores**
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 44.- Listar motores.
Precondición	Haber iniciado la sesión con una cuenta válida y que exista al menos un motor creado.
Descripción	El sistema debe devolver al usuario un listado con todos los motores creados.
Secuencia Normal	1. El usuario solicita el listado completo de motores creados en la plataforma. 2. El sistema procesa la solicitud y devuelve el listado.
Postcondición	El sistema debe haber devuelto un listado con todos los motores creados.
Excepción	Ninguna.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.15: CU-10: Listar motores

CU-11	**Listar notificadores**
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 45.- Listar notificadores de correo electrónico. ▪ RQF 46.- Listar notificadores de *Slack*. ▪ RQF 47.- Listar notificadores de *Telegram*. ▪ RQF-48.- Listar notificadores de *IRC*. ▪ RQF-49.- Listar notificadores de *Twitter*.
Precondición	Haber iniciado la sesión con una cuenta válida y que exista al menos un notificador creado.
Descripción	El sistema debe devolver al usuario un listado con todos los notificadores creados.
Secuencia Normal	1. El usuario solicita el listado completo de notificadores creados en la plataforma. 2. El sistema procesa la solicitud y devuelve el listado.
Postcondición	El sistema debe haber devuelto un listado con todos los notificadores creados.
Excepción	Ninguna.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.16: CU-11: Listar notificadores

CU-12	**Cerrar sesión**
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 36.- Cerrar sesión.
Precondición	Haber iniciado la sesión con una cuenta válida.
Descripción	El sistema debe permitir al usuario cerrar la sesión que abrió anteriormente con una cuenta válida, para evitar así usos no autorizados de su cuenta.
Secuencia Normal	1. El usuario, que previamente ha iniciado sesión, solicita al sistema la petición de cierre de sesión. 2. El sistema procesa la solicitud y desconecta al usuario, redirigiendolo a la página de inicio de sesión.
Postcondición	El sistema debe haber desconectado al usuario de la plataforma.
Excepción	Ninguna.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.17: CU-12: Cerrar sesión

CU-13	**Crear usuario**
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 01.- Añadir usuario.
Precondición	Haber iniciado la sesión con una cuenta de administración válida.
Descripción	El sistema debe permitir a un administrador crear una nueva cuenta de usuario.
Secuencia Normal	1. El administrador accede a la página de Usuarios y hace clic en el botón de creación. 2. El sistema devuelve un formulario con los datos a rellenar. 3. El administrador envía la petición de creación de una nueva cuenta de usuario con los datos rellenados en el formulario. 4. El sistema procesa la solicitud y valida los datos. 5. El sistema introduce los datos en la base de datos.
Postcondición	Un nuevo usuario es creado.
Excepción	Si la validez de los datos no ha sido exitosa, el sistema devuelve un mensaje de error.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.18: CU-13: Crear usuario

CU-14	**Modificar usuario**
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 02.- Modificar usuario.
Precondición	Haber iniciado la sesión con una cuenta de administración válida y que exista, al menos, un usuario creado.
Descripción	El sistema debe permitir al administrador modificar una cuenta de usuario existente.
Secuencia Normal	1. El administrador accede a la página de Usuarios y hace clic en el botón de modificación sobre el usuario que quiere editar. 2. El sistema devuelve un formulario con los datos a rellenar. 3. El administrador envía la petición de modificación de la cuenta de usuario con los datos rellenados en el formulario. 4. El sistema procesa la solicitud y valida los datos. 5. El sistema introduce los datos en la base de datos.
Postcondición	Un usuario es modificado en el sistema.
Excepción	Si la validez de los datos no ha sido exitosa, el sistema devuelve un mensaje de error.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.19: CU-14: Modificar usuario

CU-15	**Eliminar usuario**
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 03.- Eliminar usuario.
Precondición	Haber iniciado la sesión con una cuenta de administración válida y que exista, al menos, un usuario creado.
Descripción	El sistema debe permitir al administrador poder eliminar una cuenta existente.
Secuencia Normal	1. El administrador accede a la página de Usuarios y hace clic en el botón de supresión sobre el usuario que quiere eliminar. 2. El sistema procesa la solicitud y elimina los datos de la base de datos.
Postcondición	El usuario es eliminado.
Excepción	Ninguna.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.20: CU-15: Eliminar usuario

CU-16	**Crear motor**
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 37.- Añadir motor.
Precondición	Haber iniciado la sesión con una cuenta de administración válida.
Descripción	El sistema debe permitir a un administrador crear un nuevo motor.
Secuencia Normal	1. El administrador accede a la página de *Engines* y hace clic en el botón de creación. 2. El sistema devuelve un formulario con los datos a rellenar. 3. El administrador envía la petición de creación de un nuevo motor con los datos rellenados en el formulario. 4. El sistema procesa la solicitud y valida los datos. 5. El sistema introduce los datos en la base de datos.
Postcondición	Un nuevo motor es creado.
Excepción	Si la validez de los datos no ha sido exitosa, el sistema devuelve un mensaje de error.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.21: CU-16: Crear motor

CU-17	**Modificar motor**
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 38.- Modificar motor.
Precondición	Haber iniciado la sesión con una cuenta de administración válida y que exista, al menos, un motor creado.
Descripción	El sistema debe permitir al administrador modificar un motor existente.
Secuencia Normal	1. El administrador accede a la página de *Engines* y hace clic en el botón de modificación sobre el motor que quiere editar. 2. El sistema devuelve un formulario con los datos a rellenar. 3. El administrador envía la petición de modificación del motor con los datos rellenados en el formulario. 4. El sistema procesa la solicitud y valida los datos. 5. El sistema introduce los datos en la base de datos.
Postcondición	Un motor es modificado en el sistema.
Excepción	Si la validez de los datos no ha sido exitosa, el sistema devuelve un mensaje de error.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.22: CU-17: Modificar motor

CU-18	**Eliminar motor**
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 39.- Eliminar motor.
Precondición	Haber iniciado la sesión con una cuenta de administración válida y que exista, al menos, un motor creado.
Descripción	El sistema debe permitir al administrador poder eliminar un motor existente.
Secuencia Normal	1. El administrador accede a la página de *Engines* y hace clic en el botón de supresión sobre el motor que quiere eliminar. 2. El sistema procesa la solicitud y elimina los datos de la base de datos.
Postcondición	El motor es eliminado.
Excepción	Ninguna.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.23: CU-18: Eliminar motor

CU-19	**Crear proyecto**
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 04.- Añadir proyecto.
Precondición	Haber iniciado la sesión con una cuenta de administración válida.
Descripción	El sistema debe permitir a un administrador crear un nuevo proyecto.
Secuencia Normal	1. El administrador accede a la página de proyectos y hace clic en el botón de creación. 2. El sistema devuelve un formulario con los datos a rellenar. 3. El administrador envía la petición de creación de un nuevo proyecto con los datos rellenados en el formulario. 4. El sistema procesa la solicitud y valida los datos. 5. El sistema introduce los datos en la base de datos.
Postcondición	Un nuevo proyecto es creado.
Excepción	Si la validez de los datos no ha sido exitosa, el sistema devuelve un mensaje de error.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.24: CU-19: Crear proyecto

CU-20	**Modificar proyecto**
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 05.- Modificar proyecto.
Precondición	Haber iniciado la sesión con una cuenta de administración válida y que exista, al menos, un proyecto creado.
Descripción	El sistema debe permitir al administrador modificar un proyecto existente.
Secuencia Normal	1. El administrador accede a la página de proyectos y hace clic en el botón de modificación sobre el proyecto que quiere editar. 2. El sistema devuelve un formulario con los datos a rellenar. 3. El administrador envía la petición de modificación del proyecto con los datos rellenados en el formulario. 4. El sistema procesa la solicitud y valida los datos. 5. El sistema introduce los datos en la base de datos.
Postcondición	Un proyecto es modificado en el sistema.
Excepción	Si la validez de los datos no ha sido exitosa, el sistema devuelve un mensaje de error.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.25: CU-20: Modificar proyecto

CU-21	**Eliminar proyecto**
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 06.- Eliminar proyecto.
Precondición	Haber iniciado la sesión con una cuenta de administración válida y que exista, al menos, un proyecto creado.
Descripción	El sistema debe permitir al administrador poder eliminar un proyecto existente.
Secuencia Normal	1. El administrador accede a la página de proyectos y hace clic en el botón de supresión sobre el proyecto que quiere eliminar. 2. El sistema procesa la solicitud y elimina los datos de la base de datos.
Postcondición	El proyecto es eliminado.
Excepción	Ninguna.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.26: CU-21: Eliminar proyecto

CU-22	**Crear *pipeline***
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 27.- Añadir *pipeline* de trabajo.
Precondición	Haber iniciado la sesión con una cuenta de administración válida.
Descripción	El sistema debe permitir a un administrador crear un nuevo *pipeline*.
Secuencia Normal	1. El administrador selecciona el proyecto sobre el cual quiere crear un *pipeline* en la página de proyectos. 2. El administrador hace clic en el botón de creación de *Pipeline*. 3. El sistema devuelve un formulario con los datos a rellenar. 4. El administrador envía la petición de creación de un nuevo *pipeline* con los datos rellenados en el formulario. 5. El sistema procesa la solicitud y valida los datos. 6. El sistema introduce los datos en la base de datos.
Postcondición	Un nuevo *pipeline* es creado.
Excepción	Si la validez de los datos no ha sido exitosa, el sistema devuelve un mensaje de error.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.27: CU-22: Crear *pipeline*

CU-23	**Modificar *pipeline***
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 28.- Modificar *pipeline* de trabajo.
Precondición	Haber iniciado la sesión con una cuenta de administración válida y que exista, al menos, un *pipeline* creado.
Descripción	El sistema debe permitir al administrador modificar un *pipeline* existente.
Secuencia Normal	1. El administrador selecciona el proyecto sobre el cual quiere modificar un *pipeline* en la página de proyectos. 2. El administrador hace clic en el botón de modificación sobre el *pipeline* que quiere editar. 3. El sistema devuelve un formulario con los datos a rellenar. 4. El administrador envía la petición de modificación del *pipeline* con los datos rellenados en el formulario. 5. El sistema procesa la solicitud y valida los datos. 6. El sistema introduce los datos en la base de datos.
Postcondición	Un *pipeline* es modificado en el sistema.
Excepción	Si la validez de los datos no ha sido exitosa, el sistema devuelve un mensaje de error.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.28: CU-23: Modificar *pipeline*

CU-24	**Eliminar *pipeline***
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 29.- Eliminar *pipeline* de trabajo.
Precondición	Haber iniciado la sesión con una cuenta de administración válida y que exista, al menos, un *pipeline* creado.
Descripción	El sistema debe permitir al administrador poder eliminar un proyecto existente.
Secuencia Normal	1. El administrador selecciona el proyecto sobre el cual quiere eliminar un *pipeline* en la página de proyectos. 2. El administrador hace clic en el botón de supresión sobre el *pipeline* que quiere eliminar. 3. El sistema procesa la solicitud y elimina los datos de la base de datos.
Postcondición	El *pipeline* es eliminado.
Excepción	Ninguna.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.29: CU-24: Eliminar *pipeline*

CU-25	**Crear notificador**
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 10.- Configurar notificaciones por correo electrónico. ▪ RQF 14.- Configurar notificaciones por *Telegram*. ▪ RQF 18.- Configurar notificaciones por *Slack*. ▪ RQF 22.- Configurar notificaciones por *IRC*. ▪ RQF 26.- Configurar notificaciones por *Twitter*.
Precondición	Haber iniciado la sesión con una cuenta de administración válida.
Descripción	El sistema debe permitir a un administrador crear un nuevo notificador.
Secuencia Normal	1. El administrador accede a la página de notificadores y hace clic en el botón de creación. 2. El sistema devuelve un formulario con los datos a rellenar. 3. El administrador envía la petición de creación de un nuevo notificador con los datos rellenados en el formulario. 4. El sistema procesa la solicitud y valida los datos. 5. El sistema introduce los datos en la base de datos.
Postcondición	Un nuevo notificador es creado.
Excepción	Si la validez de los datos no ha sido exitosa, el sistema devuelve un mensaje de error.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.30: CU-25: Crear notificador

CU-26	**Modificar notificador**
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 10.- Configurar notificaciones por correo electrónico. ▪ RQF 14.- Configurar notificaciones por *Telegram*. ▪ RQF 18.- Configurar notificaciones por *Slack*. ▪ RQF 22.- Configurar notificaciones por *IRC*. ▪ RQF 26.- Configurar notificaciones por *Twitter*.
Precondición	Haber iniciado la sesión con una cuenta de administración válida y que exista, al menos, un notificador creado.
Descripción	El sistema debe permitir al administrador modificar un notificador existente.
Secuencia Normal	1. El administrador accede a la página de notificadores y hace clic en el botón de modificación sobre el notificador que quiere editar. 2. El sistema devuelve un formulario con los datos a rellenar. 3. El administrador envía la petición de modificación del notificador con los datos rellenados en el formulario. 4. El sistema procesa la solicitud y valida los datos. 5. El sistema introduce los datos en la base de datos.
Postcondición	Un notificador es modificado en el sistema.
Excepción	Si la validez de los datos no ha sido exitosa, el sistema devuelve un mensaje de error.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.31: CU-26: Modificar notificador

CU-27	**Eliminar notificador**
Dependencias	*Depende de los siguientes requisitos:* ▪ RQF 08.- Desactivar notificaciones por correo electrónico. ▪ RQF 12.- Desactivar notificaciones por *Telegram*. ▪ RQF 16.- Desactivar notificaciones por *Slack*. ▪ RQF 20.- Desactivar notificaciones por *IRC*. ▪ RQF 24.- Desactivar notificaciones por *Twitter*.
Precondición	Haber iniciado la sesión con una cuenta de administración válida y que exista, al menos, un notificador creado.
Descripción	El sistema debe permitir al administrador poder eliminar un notificador existente.
Secuencia Normal	1. El administrador accede a la página de notificadores y hace clic en el botón de supresión sobre el notificador que quiere eliminar. 2. El sistema procesa la solicitud y elimina los datos de la base de datos.
Postcondición	Un notificador es eliminado del sistema.
Excepción	Ninguna.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 4.32: CU-27: Eliminar notificador

4.3. Requisitos

La especificación de requisitos consiste en proporcionar una descripción general del funcionamiento que el sistema a desarrollar va a tener durante su ciclo de vida. En lugar de utilizar una aproximación formal, al estar dirigida a un equipo de desarrollo multidisciplinar se utiliza un lenguaje algo más informal, de forma que sea fácilmente comprensible para todas los perfiles involucrados en comprender el uso de esta plataforma.

En esta sección se muestran todos los requisitos del proyecto, divididos por tipo.

4.3.1. Requisitos funcionales

Un requisito funcional es aquel que describe una funcionalidad del software. *GameCraft* cuenta con los siguientes requisitos funcionales.

RQF-01	**Añadir usuario**
Descripción	El sistema debe permitir a un administrador crear cuentas de usuario para que las personas puedan poder usar las funcionalidades de la plataforma.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.33: RQF-01: Añadir usuario

RQF-02	**Modificar usuario**
Descripción	El sistema debe permitir a un administrador modificar cuentas de usuario en el momento en que una persona quiera modificar algún detalle de su cuenta.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.34: RQF-02: Modificar usuario

RQF-03	**Eliminar usuario**
Descripción	El sistema debe permitir a un administrador eliminar cuentas de usuario que, por alguna razón en concreto, ya no sean necesarias de almacenar.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.35: RQF-03: Eliminar usuario

RQF-04	**Añadir proyecto**
Descripción	El sistema debe permitir a un administrador crear un proyecto de desarrollo de un videojuego sobre el cual se irán ejecutando las diferentes etapas de un *pipeline*.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.36: RQF-04: Añadir proyecto

RQF-05	**Modificar proyecto**
Descripción	El sistema debe permitir a un administrador modificar los detalles de un proyecto de desarrollo de un videojuego sobre el cual se irán ejecutando las diferentes etapas de un *pipeline*.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.37: RQF-05: Modificar proyecto

RQF-06	**Eliminar proyecto**
Descripción	El sistema debe permitir a un administrador eliminar un proyecto de desarrollo de un videojuego sobre el cual se irán ejecutando las diferentes etapas de un *pipeline*.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.38: RQF-06: Eliminar proyecto

RQF-07	**Activar notificaciones por correo electrónico**
Descripción	El sistema debe permitir al administrador automatizar el envío de correos electrónicos cuando sucede un evento dentro de la plataforma en lo que concierne al estado final de compilación de un proyecto.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.39: RQF-07: Activar notificaciones por correo electrónico

RQF-08	**Desactivar notificaciones por correo electrónico**
Descripción	El sistema debe permitir al administrador desactivar el envío de correos electrónicos cuando sucede un evento dentro de la plataforma en lo que concierne al estado final de compilación de un proyecto.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.40: RQF-08: Desactivar notificaciones por correo electrónico

RQF-09	**Enviar notificación por correo electrónico**
Descripción	El sistema debe enviar un correo electrónico a una dirección de correo electrónica proporcionada por el administrador cuando sucede un evento dentro de la plataforma en lo que concierne al estado final de compilación de un proyecto.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.41: RQF-09: Enviar notificación por correo electrónico

RQF-10	**Configurar notificaciones por correo electrónico**
Descripción	El sistema debe permitir al administrador introducir los parámetros de conexión a una cuenta de correo electrónico válida: servidor SMTP, usuario, contraseña, puerto y uso de protocolo seguro.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.42: RQF-10: Configurar notificaciones por correo electrónico

RQF-11	**Activar notificaciones por *Telegram***
Descripción	El sistema debe permitir al administrador automatizar el envío de mensajes por *Telegram* cuando sucede un evento dentro de la plataforma en lo que concierne al estado final de compilación de un proyecto.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.43: RQF-11: Activar notificaciones por *Telegram*

RQF-12	**Desactivar notificaciones por *Telegram***
Descripción	El sistema debe permitir al administrador desactivar el envío de mensajes por *Telegram* cuando sucede un evento dentro de la plataforma en lo que concierne al estado final de compilación de un proyecto.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.44: RQF-12 Desactivar notificaciones por *Telegram*

RQF-13	**Enviar notificación por *Telegram***
Descripción	El sistema debe enviar un mensaje de *Telegram* a un usuario o grupo de *Telegram* definido por el administrador cuando sucede un evento dentro de la plataforma en lo que concierne al estado final de compilación de un proyecto.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.45: RQF-13: Enviar notificación por *Telegram*

RQF-14	**Configurar notificaciones por *Telegram***
Descripción	El sistema debe permitir al administrador introducir los parámetros de conexión a una cuenta de *Telegram* válida: nombre de la cuenta y token de autenticación.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.46: RQF-14: Configurar notificaciones por *Telegram*

RQF-15	**Activar notificaciones por *Slack***
Descripción	El sistema debe permitir al administrador automatizar el envío de mensajes por *Slack* cuando sucede un evento dentro de la plataforma en lo que concierne al estado final de compilación de un proyecto.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.47: RQF-15: Activar notificaciones por *Slack*

RQF-16	**Desactivar notificaciones por *Slack***
Descripción	El sistema debe permitir al administrador desactivar el envío de mensajes por *Slack* cuando sucede un evento dentro de la plataforma en lo que concierne al estado final de compilación de un proyecto.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.48: RQF-16: Desactivar notificaciones por *Slack*

RQF-17	**Enviar notificación por *Slack***
Descripción	El sistema debe enviar un mensaje a un usuario o grupo de *Slack* definido por el administrador cuando sucede un evento dentro de la plataforma en lo que concierne al estado final de compilación de un proyecto.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.49: RQF-17: Enviar notificación por *Slack*

RQF-18	**Configurar notificaciones por *Slack***
Descripción	El sistema debe permitir al administrador introducir los parámetros de conexión a una cuenta de *Slack* válida: nombre de la cuenta y token de autenticación.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.50: RQF-18: Configurar notificaciones por *Slack*

RQF-19	**Activar notificaciones por *IRC***
Descripción	El sistema debe permitir al administrador automatizar el envío de mensajes por *IRC* cuando sucede un evento dentro de la plataforma en lo que concierne al estado final de compilación de un proyecto.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.51: RQF-19: Activar notificaciones por *IRC*

RQF-20	**Desactivar notificaciones por *IRC***
Descripción	El sistema debe permitir al administrador desactivar el envío de mensajes por *IRC* cuando sucede un evento dentro de la plataforma en lo que concierne al estado final de compilación de un proyecto.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.52: RQF-20: Desactivar notificaciones por *IRC*

RQF-21	**Enviar notificación por *IRC***
Descripción	El sistema debe enviar un mensaje a un canal de *IRC* definido por el administrador cuando sucede un evento dentro de la plataforma en lo que concierne al estado final de compilación de un proyecto.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.53: RQF-21: Enviar notificación por *IRC*

RQF-22	**Configurar notificaciones por *IRC***
Descripción	El sistema debe permitir al administrador introducir los parámetros de conexión a una cuenta de *IRC* válida: nombre de la cuenta y token de autenticación.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.54: RQF-22: Configurar notificaciones por *IRC*

RQF-23	**Activar notificaciones por *Twitter***
Descripción	El sistema debe permitir al administrador automatizar el envío de *tuits* cuando sucede un evento dentro de la plataforma en lo que concierne al estado final de compilación de un proyecto.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.55: RQF-23: Activar notificaciones por Twitter

RQF-24	**Desactivar notificaciones por Twitter**
Descripción	El sistema debe permitir al administrador desactivar el envío de *tuits* cuando sucede un evento dentro de la plataforma en lo que concierne al estado final de compilación de un proyecto.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.56: RQF-24: Desactivar notificaciones por *Twitter*

RQF-25	**Enviar notificación por *Twitter***
Descripción	El sistema debe enviar un *tuit* cuando sucede un evento dentro de la plataforma en lo que concierne al estado final de compilación de un proyecto.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.57: RQF-25: Enviar notificación por *Twitter*

RQF-26	**Configurar notificaciones por *Twitter***
Descripción	El sistema debe permitir al administrador introducir los parámetros de conexión a una cuenta de *Twitter* válida: nombre de la cuenta y token de autenticación.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.58: RQF-26: Configurar notificaciones por *Twitter*

RQF-27	**Añadir *pipeline* de trabajo**
Descripción	El sistema debe permitir al administrador poder crear una secuencia de tareas que serán ejecutadas dentro de un proyecto. Una tarea es un determinado comando que se ejecutará internamente en el sistema, por ejemplo, obtención del código fuente desde un repositorio o la compilación y ejecución de tests.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.59: RQF-27: Añadir *pipeline* de trabajo

RQF-28	**Modificar *pipeline* de trabajo**
Descripción	El sistema debe permitir al administrador poder modificar una secuencia de tareas que serán ejecutadas dentro de un proyecto. Una tarea es un determinado comando que se ejecutará internamente en el sistema, por ejemplo, obtención del código fuente desde un repositorio o la compilación y ejecución de tests.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.60: RQF-28: Modificar *pipeline* de trabajo

RQF-29	**Eliminar *pipeline* de trabajo**
Descripción	El sistema debe permitir al administrador poder eliminar una secuencia de tareas que serán ejecutadas dentro de un proyecto. Una tarea es un determinado comando que se ejecutará internamente en el sistema, por ejemplo, obtención del código fuente desde un repositorio o la compilación y ejecución de tests.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.61: RQF-29: Eliminar *pipeline* de trabajo

RQF-30	**Ejecutar *pipeline* de trabajo**
Descripción	El sistema debe permitir al administrador poder ejecutar una secuencia de tareas dentro de un proyecto. Una tarea es un determinado comando que se ejecutará internamente en el sistema, por ejemplo, obtención del código fuente desde un repositorio o la compilación y ejecución de tests.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.62: RQF-30: Ejecutar *pipeline* de trabajo

RQF-31	**Abortar ejecución de *pipeline* de trabajo**
Descripción	El sistema debe permitir al administrador poder abortar la ejecución de una secuencia de tareas dentro de un proyecto. Una tarea es un determinado comando que se ejecutará internamente en el sistema, por ejemplo, obtención del código fuente desde un repositorio o la compilación y ejecución de tests.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.63: RQF-31: Abortar ejecución de *pipeline* de trabajo

RQF-32	**Generar informe del estado de la ejecución de un *pipeline***
Descripción	El sistema debe poder generar un informe que contenga información sobre la ejecución de un *pipeline*. El informe contiene la salida generada por los propios comandos internos de sistema que se han ido ejecutando y el tiempo empleado.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.64: RQF-32: Generar informe del estado de la ejecución de un *pipeline*

RQF-33	**Visualizar informe del estado de la ejecución de un *pipeline***
Descripción	El sistema debe permitir a un usuario o administrador visualizar un informe que contenga información sobre la ejecución de un *pipeline*. El informe contiene la salida generada por los propios comandos internos de sistema que se han ido ejecutando y el tiempo empleado.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.65: RQF-33: Visualizar informe del estado de la ejecución de un *pipeline*

RQF-34	**Descargar informe del estado de la ejecución de un *pipeline***
Descripción	El sistema debe permitir a un usuario o administrador descargar en su ordenador un informe que contenga información sobre la ejecución de un *pipeline*. El informe contiene la salida generada por los propios comandos internos de sistema que se han ido ejecutando y el tiempo empleado. El informe se descargará en formato PDF.
Importancia	Media.
Prioridad	Media.

Tabla 4.66: RQF-34: Descargar informe del estado de la ejecución de un *pipeline*

RQF-35	**Iniciar sesión en el sistema**
Descripción	El sistema debe permitir a una persona autenticarse utilizando su cuenta de usuario y poder acceder a las funcionalidades de la plataforma.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.67: RQF-35: Iniciar sesión en el sistema

RQF-36	**Cerrar sesión**
Descripción	El sistema debe permitir a un usuario cerrar su sesión abierta previamente con su cuenta, para evitar accesos no autorizados.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.68: RQF-36: Cerrar sesión

RQF-37	**Añadir motor**
Descripción	El sistema debe permitir al administrador añadir la referencia a una herramienta de desarrollo y creación de videojuegos, utilizada para desarrollar el proyecto. En el momento de compilar y ejecutar los tests, se invocará a dicha herramienta a través del uso de comandos de sistema.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.69: RQF-37: Añadir motor

RQF-38	**Modificar motor**
Descripción	El sistema debe permitir al administrador modificar la referencia a una herramienta de desarrollo y creación de videojuegos, utilizada para desarrollar el proyecto. En el momento de compilar y ejecutar los tests, se invocará a dicha herramienta a través del uso de comandos de sistema.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.70: RQF-38: Modificar motor

RQF-39	**Eliminar motor**
Descripción	El sistema debe permitir al administrador eliminar la referencia a una herramienta de desarrollo y creación de videojuegos, utilizada para desarrollar el proyecto. En el momento de compilar y ejecutar los tests, se invocará a dicha herramienta a través del uso de comandos de sistema.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.71: RQF-39: Eliminar motor

RQF-40	**Ejecutar motor**
Descripción	El sistema debe permitir al administrador procesar y ejecutar la referencia a una herramienta de desarrollo y creación de videojuegos, utilizada para desarrollar el proyecto. En el momento de compilar y ejecutar los tests, se invocará a dicha herramienta a través del uso de comandos de sistema.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.72: RQF-40: Ejecutar motor

RQF-41	**Listar usuarios**
Descripción	El sistema debe ofrecer un listado de todos los usuarios que han sido creados.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.73: RQF-41: Listar usuarios

RQF-42	**Listar *pipelines***
Descripción	El sistema debe ofrecer un listado de todos los *pipelines* que han sido creados.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.74: RQF-42: Listar *pipelines*

RQF-43	**Listar proyectos**
Descripción	El sistema debe ofrecer un listado de todos los proyectos que han sido creados.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.75: RQF-43: Listar proyectos

RQF-44	**Listar motores**
Descripción	El sistema debe ofrecer un listado de todos los motores que han sido creados.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.76: RQF-44: Listar motores

RQF-45	**Listar notificadores de correo electrónico**
Descripción	El sistema debe ofrecer un listado de todos los notificadores de correos electrónicos que han sido creados.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.77: RQF-45: Listar notificadores de correo electrónico

RQF-46	**Listar notificadores de *Slack***
Descripción	El sistema debe ofrecer un listado de todos los notificadores de *Slack* que han sido creados.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.78: RQF-46: Listar notificadores de *Slack*

RQF-47	**Listar notificadores de *Telegram***
Descripción	El sistema debe ofrecer un listado de todos los notificadores de *Telegram* que han sido creados.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.79: RQF-47: Listar notificadores de *Telegram*

RQF-48	**Listar notificadores de *IRC***
Descripción	El sistema debe ofrecer un listado de todos los notificadores de *IRC* que han sido creados.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.80: RQF-48: Listar notificadores de *IRC*

RQF-49	**Listar notificadores de *Twitter***
Descripción	El sistema debe ofrecer un listado de todos los notificadores de *Twitter* que han sido creados.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.81: RQF-49: Listar notificadores de *Twitter*

4.3.2. Requisitos no funcionales

Los requisitos no funcionales son aquellos que describen propiedades que el producto debe tener, y en ningún momento expresan funcionalidad. En este proyecto se van a desarrollar los siguientes requisitos no funcionales:

NFR-01	**Multiplataforma**
Descripción	El sistema debe poder desplegarse sin problemas en servidores con *Windows*, *GNU/Linux* y *macOS*.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.82: NFR-01: Multiplataforma

NFR-02	**Consistencia de los datos**
Descripción	El sistema no debe corromper los ficheros fuente ni los elementos de los proyectos que se descarguen desde los repositorios, evitando en todo momento que se produzcan errores a la hora de compilar o testear el proyecto.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.83: NFR-02: Consistencia de los datos

NFR-03	**Escalabilidad horizontal**
Descripción	El sistema debe garantizar una buena escalabilidad horizontal siguiendo una arquitectura de microservicios. El sistema es escalable horizontalmente si y sólo si la plataforma se puede descomponer en módulos o servicios y cada módulo de la plataforma es independiente del resto de módulos De esta forma, se pueden crear o eliminar instancias de módulos dependiendo del uso y la demanda de los usuarios a la plataforma.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.84: NFR-03: Escalabilidad horizontal

NFR-04	**Interfaz de usuario**
Descripción	El sistema debe tener una interfaz de usuario simple y adaptada a las pantallas de los dispositivos móviles. La interfaz debe carecer de más de 2 cuadros de diálogo consecutivos y recurrir a elementos visuales que permitan al usuario de un simple vistazo obtener información (como utilizar el color rojo en botones de supresión, por ejemplo).
Importancia	Alta.
Prioridad	Alta.

Tabla 4.85: NFR-04: Interfaz de usuario

4.3.3. Requisitos de información

Los requisitos de información describen la información que debe almacenar y gestionar el sistema para dar soporte a diferentes procesos de negocio. Deben identificar el *concepto relevante* sobre el que se debe guardar información así como qué *datos específicos* relativos al concepto son importantes para cumplir los objetivos del sistema.

En este proyecto se van a desarrollar los siguientes requisitos de información:

IR-01	**Usuario**
Descripción	El sistema debe almacenar en una base de datos *SQL* la información relacionada con la cuenta de un usuario para que pueda ser utilizada para acceder al resto de funcionalidades de la plataforma. La información que se almacena acerca de un usuario es la siguiente: identificador del usuario, nombre de la cuenta, nombre y apellidos de la persona, dirección de correo electrónico, idioma y rol (usuario o administrador).
Importancia	Alta.
Prioridad	Alta.

Tabla 4.86: IR-01: Usuario

IR-02	**Proyecto**
Descripción	El sistema debe almacenar en una base de datos *SQL* la información relacionada con los proyectos: identificador del proyecto, nombre del proyecto, descripción del proyecto y web del proyecto.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.87: IR-02: Proyecto

IR-03	**Pipeline**
Descripción	El sistema debe almacenar en una base de datos *SQL* la información relacionada con cada paso que se hace en el *pipeline*: identificación del sistema de repositorio de obtención de código fuente y detalles del mismo (dirección del repositorio, rama, usuario y contraseña), identificador del proyecto, identificador del motor a utilizar en el proceso de compilación y testeo, identificación del sistema de notificación a utilizar y detalles del receptor de las notificaciones.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.88: IR-03: Pipeline

IR-04	Informe del estado
Descripción	El sistema debe almacenar la información relacionada con la ejecución de un *pipeline* sobre un proyecto determinado: información generada por el conector del repositorio, información generada por el compilador, información generada por el analizador de código y métricas acerca del tiempo consumido, fecha de inicio y de fin de la compilación.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.89: IR-04: Informe del estado

IR-05	Notificación
Descripción	El sistema debe generar un mensaje tras haberse ejecutado un *pipeline* sobre un proyecto determinado. El mensaje informará si el proceso se ha realizado correctamente o si, por el contrario, han habido errores. Se proporciona un enlace directo al informe generado.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.90: IR-05: Notificación

IR-06	Motor
Descripción	El sistema debe almacenar información relacionada acerca de una herramienta de desarrollo de videojuegos. La información a almacenar será la siguiente: identificador del motor, nombre del motor, descripción del motor, ruta de instalación del motor y argumentos a añadir en el momento de ejecutar el motor.
Importancia	Alta.
Prioridad	Alta.

Tabla 4.91: IR-06: Motor

4.4. Arquitectura

La arquitectura expresa cuáles son los componentes lógicos que participan en la aplicación y la relación entre ellos.

Con el objetivo de dotar a *GameCraft* de una escalabilidad horizontal, el desarrollo del sistema se ha centrado en el uso de microservicios. Una arquitectura basada en microservicios es un enfoque donde una aplicación esta modularizada en un grupo de pequeños servicios, independientes entre sí, comunicándose a través de peticiones *HTTP* a sus *API*. Al ser servicios independientes, su código puede ser desplegado y retirado sin afectar a los demás microservicios, es decir, que un cambio en el módulo de notificaciones de correo electrónico en *GameCraft* no afectará al módulo de invocación de *pipelines*, por ejemplo. Además, este hecho permite al desarrollador entender mejor el sistema ya que la lógica de negocio está bien separada, permite

hot-swapping (habilidad para poder hacer cambios en caliente de un servicio sin afectar a la disponibilidad del resto de la plataforma) y es más fácil de escalar a nivel de *software* ya que se evita replicar toda la aplicación, sino sólo aquellos módulos que están procesando un gran volumen de peticiones.

La arquitectura física del proyecto queda representada gráficamente mediante la Figura 4.3.

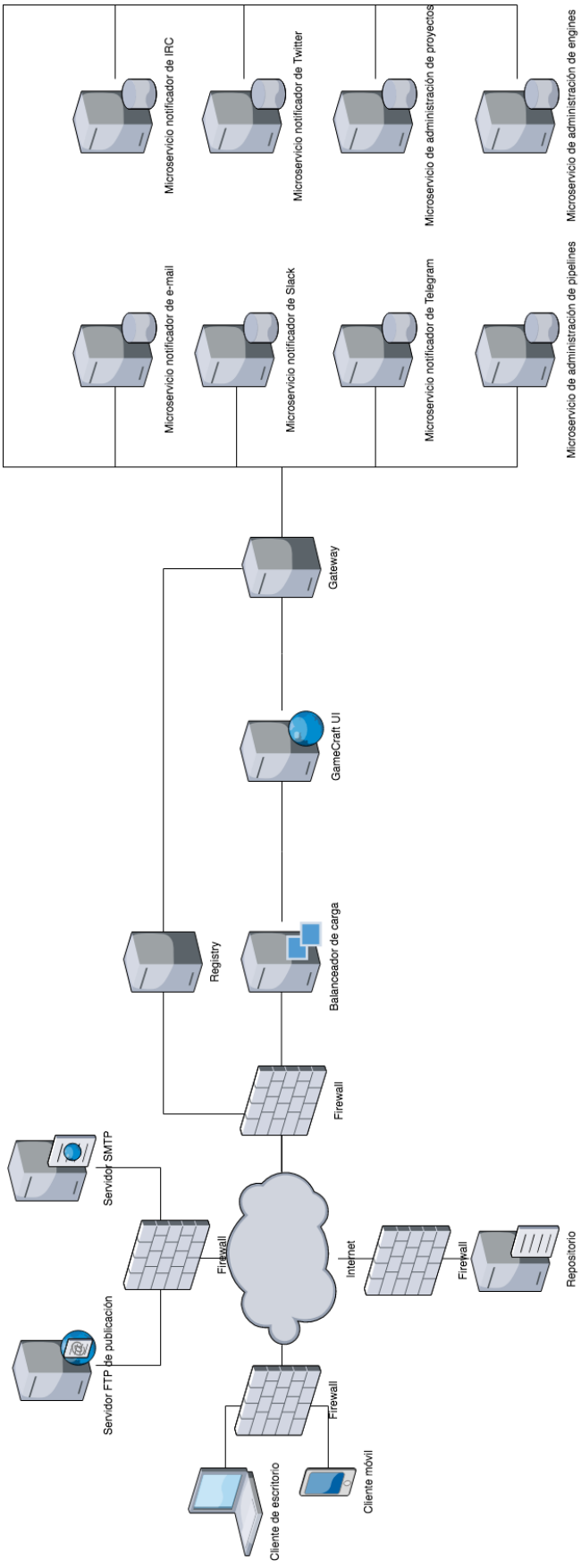

Figura 4.3: Arquitectura física de *GameCraft*

La arquitectura lógica del proyecto queda representada gráficamente mediante la Figura 4.4.

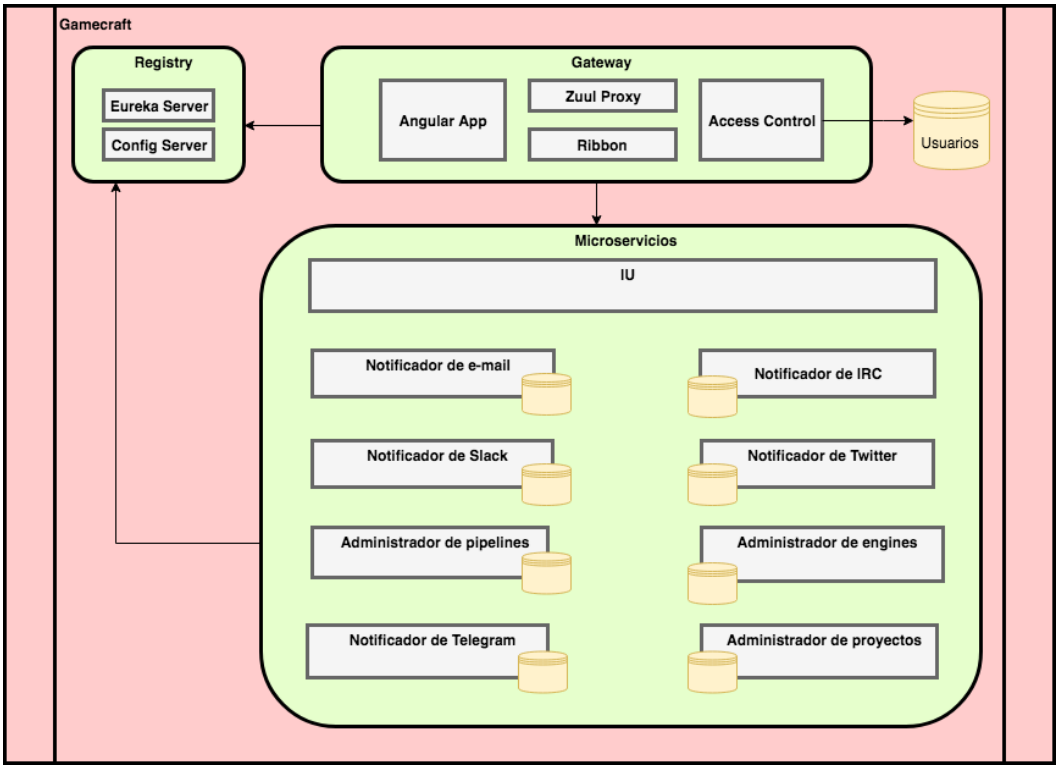

Figura 4.4: Arquitectura lógica de *GameCraft*

Cuando todos los microservicios se van desplegando, proceden a registrarse en el microservicio de registro, para que el sistema tenga conocimiento de cuántos microservicios hay levantados y cuántos microservicios no están levantados.

El usuario accede a través de la interfaz de usuario de la plataforma, que es un microservicio más de la arquitectura. Para poder manejar los datos procedentes de otros microservicios desde la interfaz de usuario, las peticiones primero se encaminan hacia el microservicio de encaminamiento, que se encarga de validar que la petición procede de un usuario autenticado, y redirige la petición al microservicio adecuado. De acuerdo con la arquitectura de microservicios, cada microservicio debe ser independiente, y por tanto, este es el motivo de que cada uno tenga bases de datos independientes. Siempre que se quiere obtener un dato, por ejemplo, de la base de datos de proyectos, hay que hacer uso de la *API Rest* diseñada en el microservicio de proyectos.

4.5. Diagramas de secuencia

En esta sección se adjuntan algunos diagramas de secuencia correspondientes a los procesos más importantes de la plataforma.

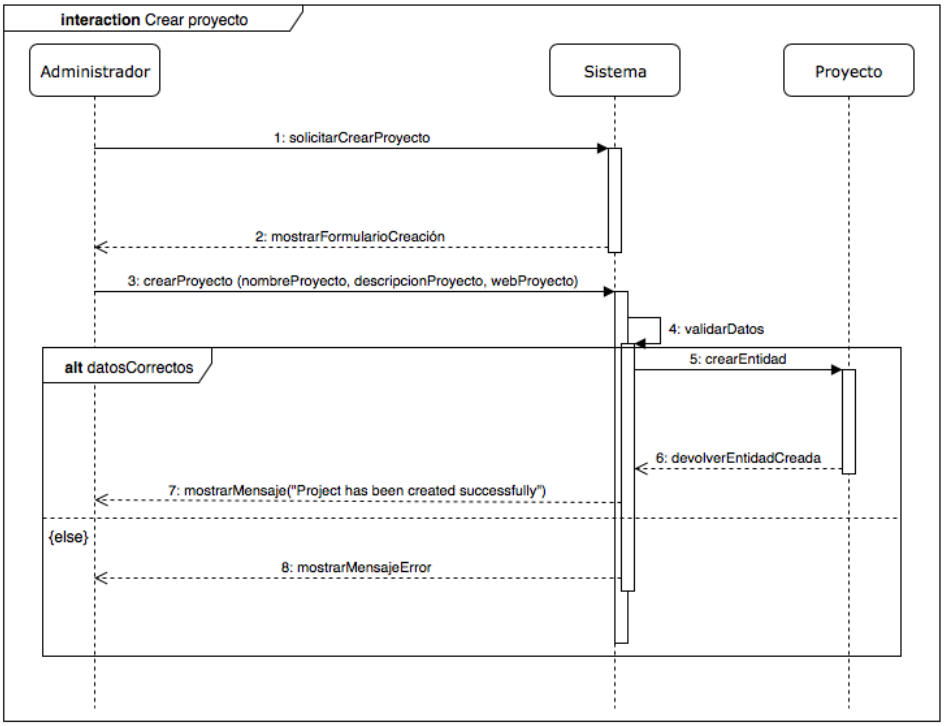

Figura 4.5: Diagrama de secuencia para la funcionalidad «Crear proyecto»

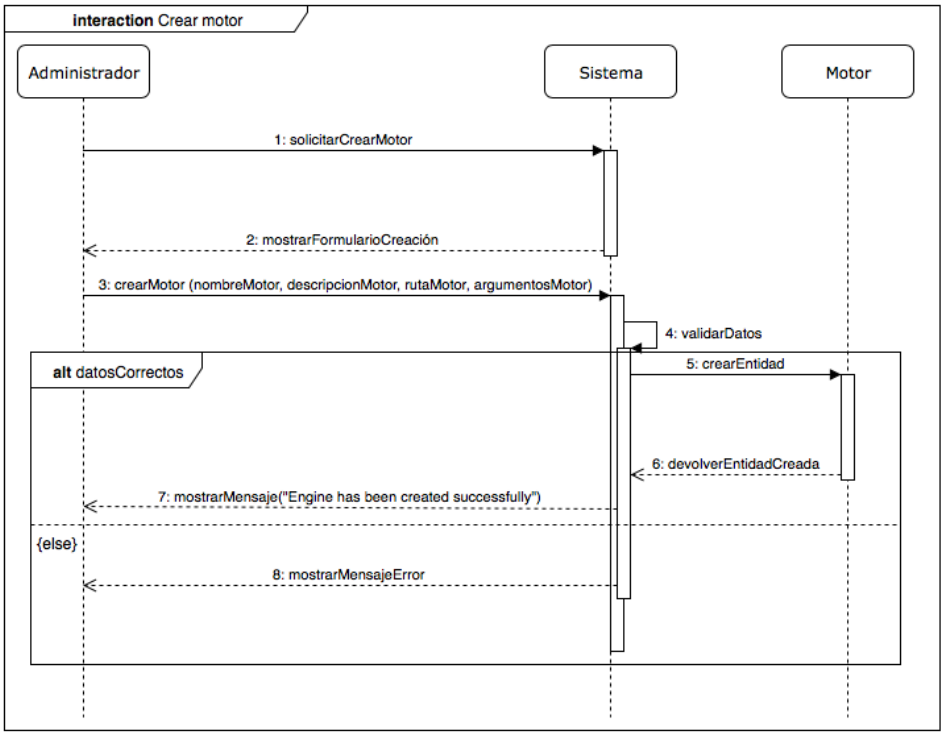

Figura 4.6: Diagrama de secuencia para la funcionalidad «Crear motor»

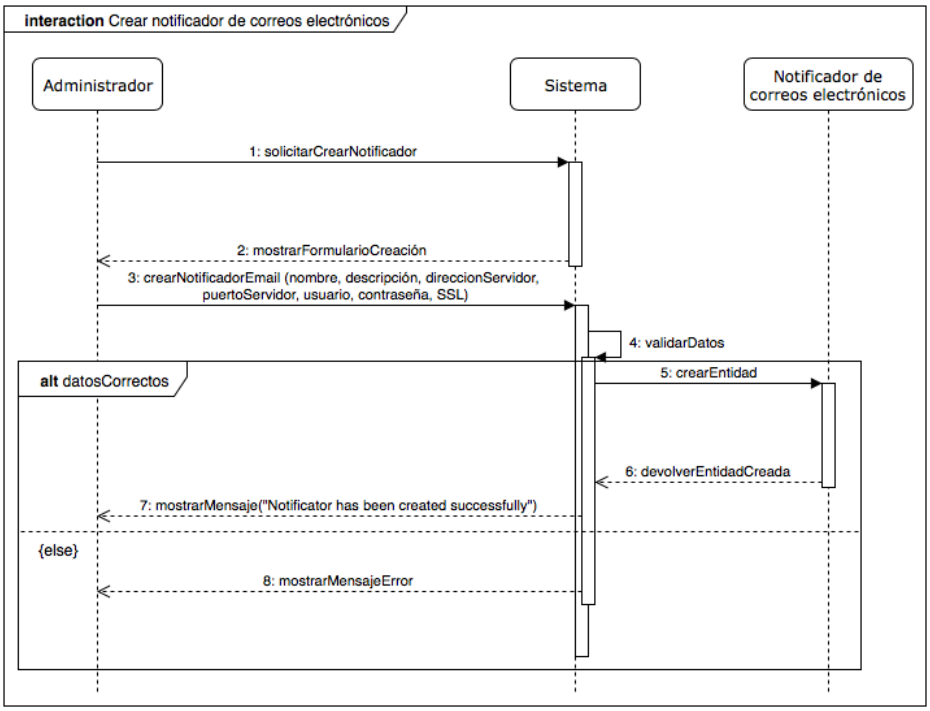

Figura 4.7: Diagrama de secuencia para la funcionalidad «Crear notificador de correos electrónicos»

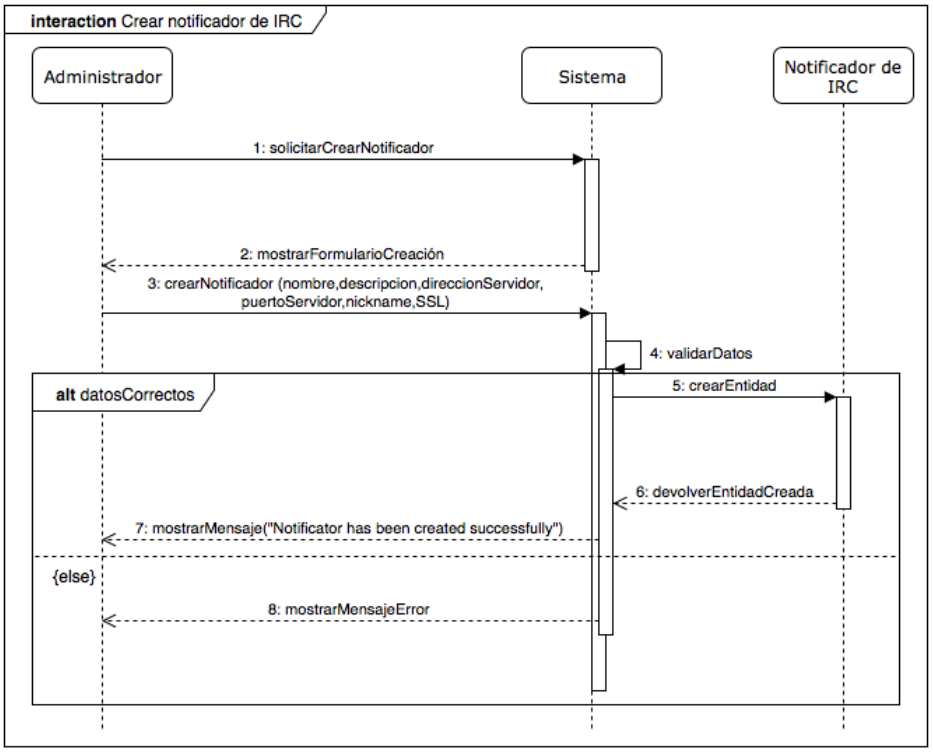

Figura 4.8: Diagrama de secuencia para la funcionalidad «Crear notificador de *IRC*»

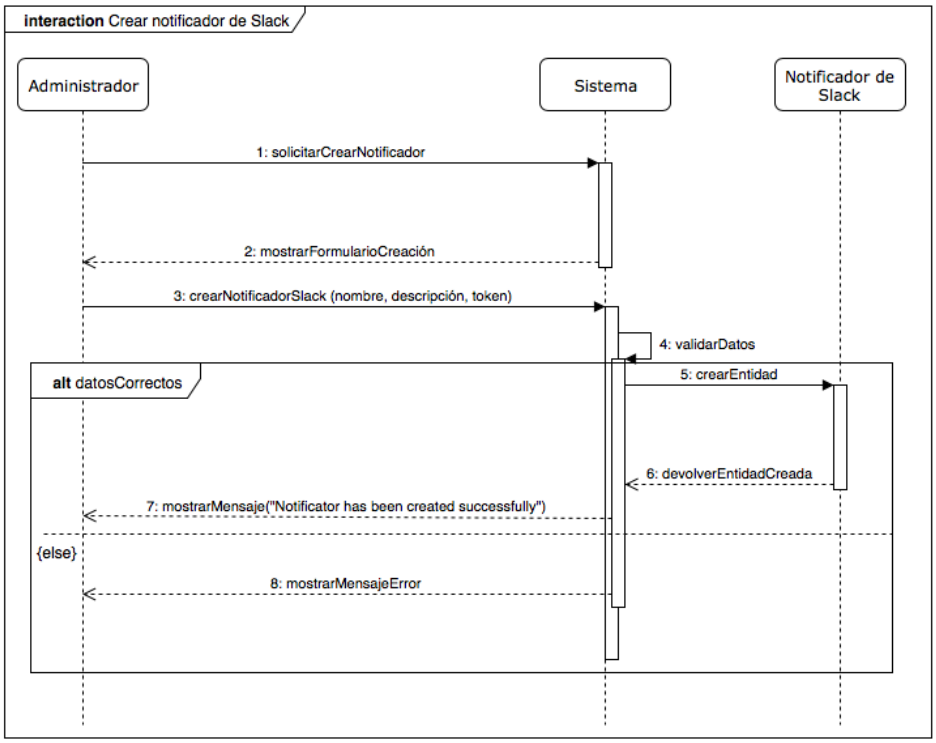

Figura 4.9: Diagrama de secuencia para la funcionalidad «Crear notificador de *Slack*»

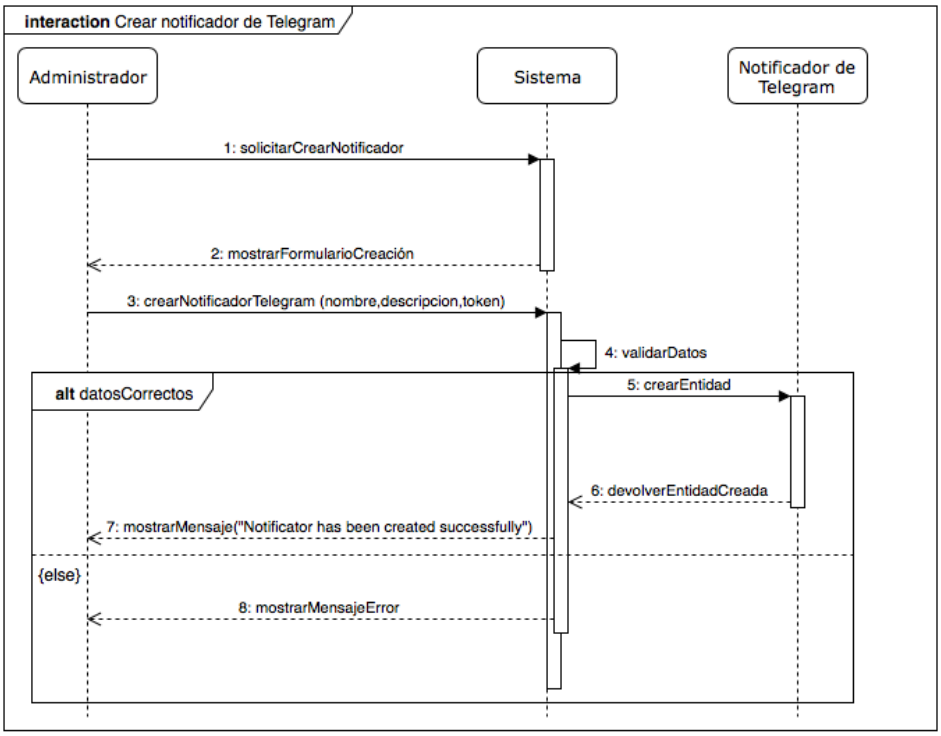

Figura 4.10: Diagrama de secuencia para la funcionalidad «Crear notificador de *Telegram*»

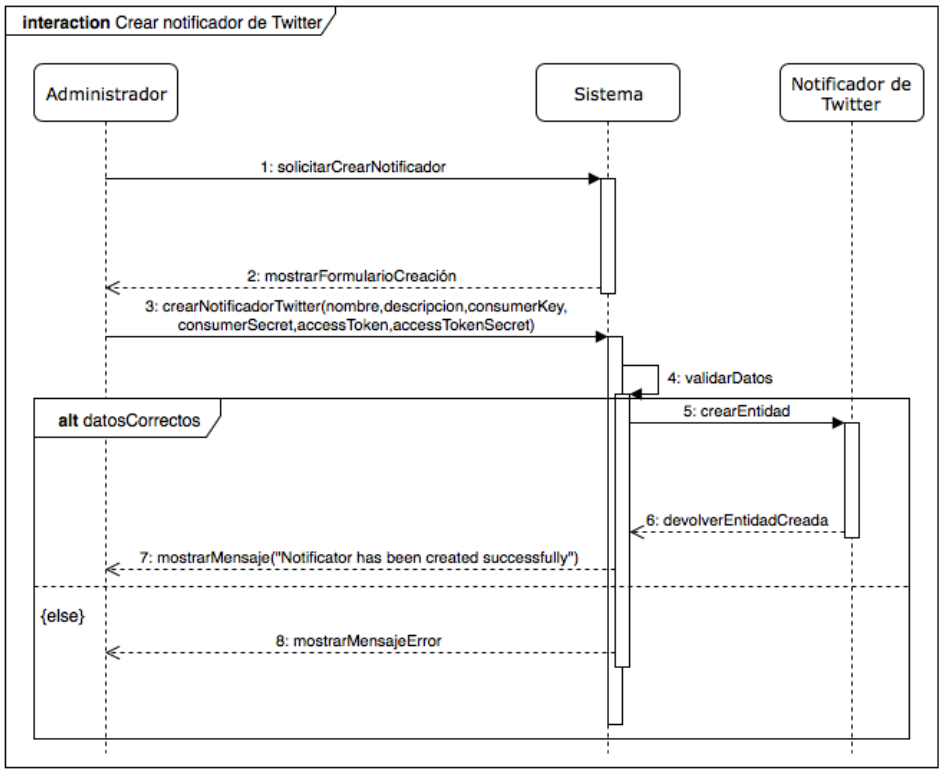

Figura 4.11: Diagrama de secuencia para la funcionalidad «Crear notificador de *Twitter*»

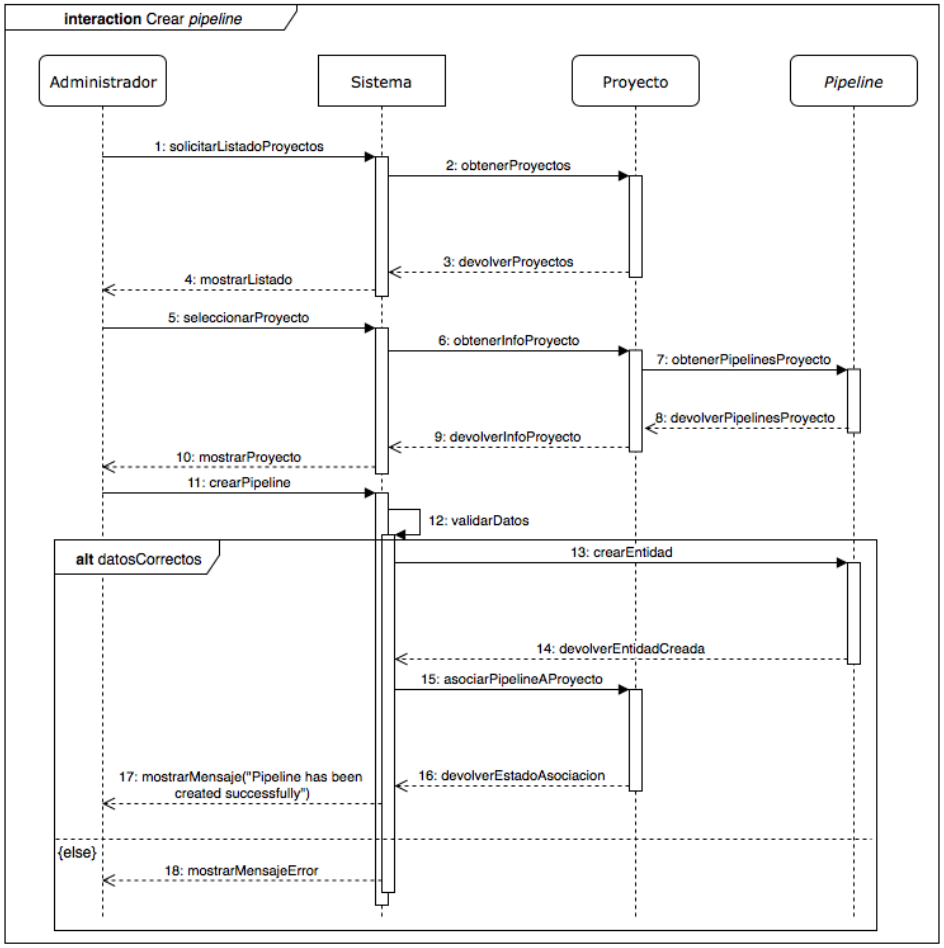

Figura 4.12: Diagrama de secuencia para la funcionalidad «Crear *pipeline*»

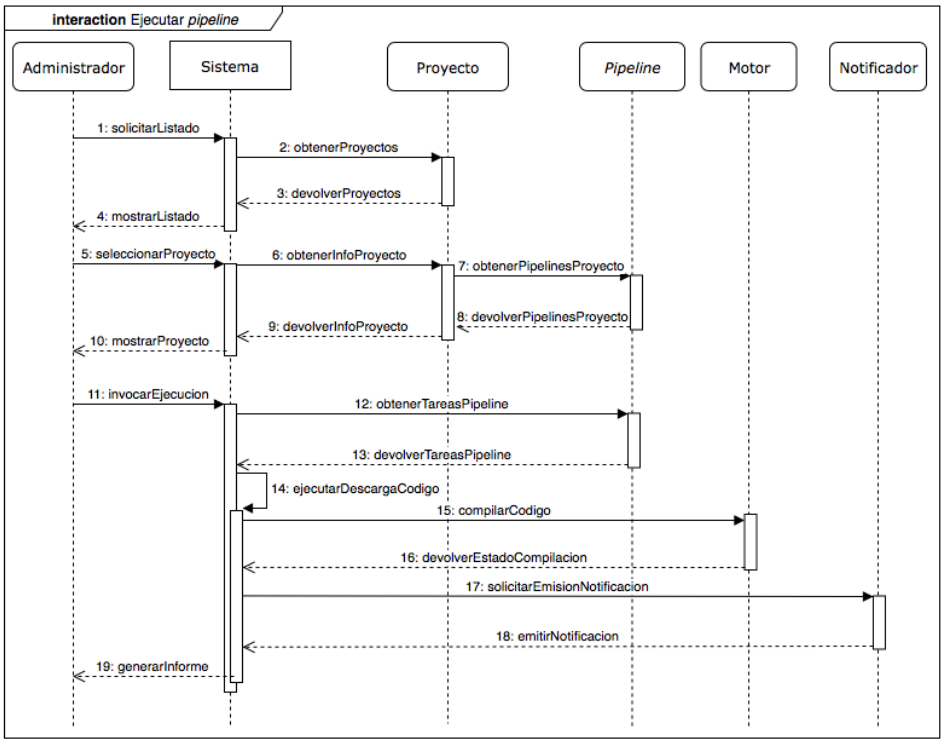

Figura 4.13: Diagrama de secuencia para la funcionalidad «Ejecutar *pipeline*»

4.6. Bocetos de diseño de la interfaz de usuario

En esta sección se adjuntan los bocetos que se realizaron como etapa anterior al diseño y desarrollo de la interfaz actual que presenta el sistema.

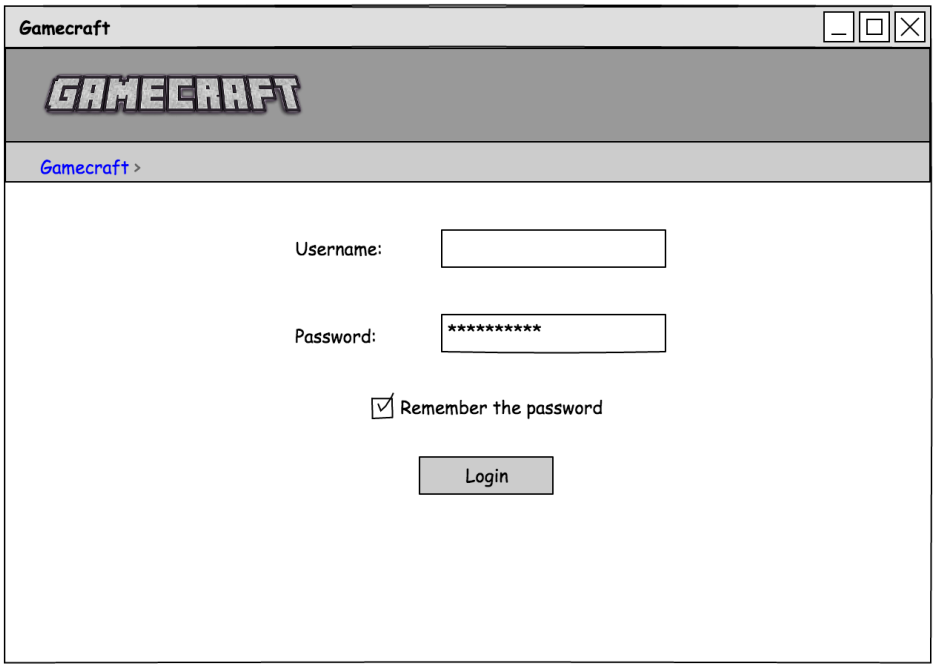

Figura 4.14: Boceto de la pantalla de inicio de sesión

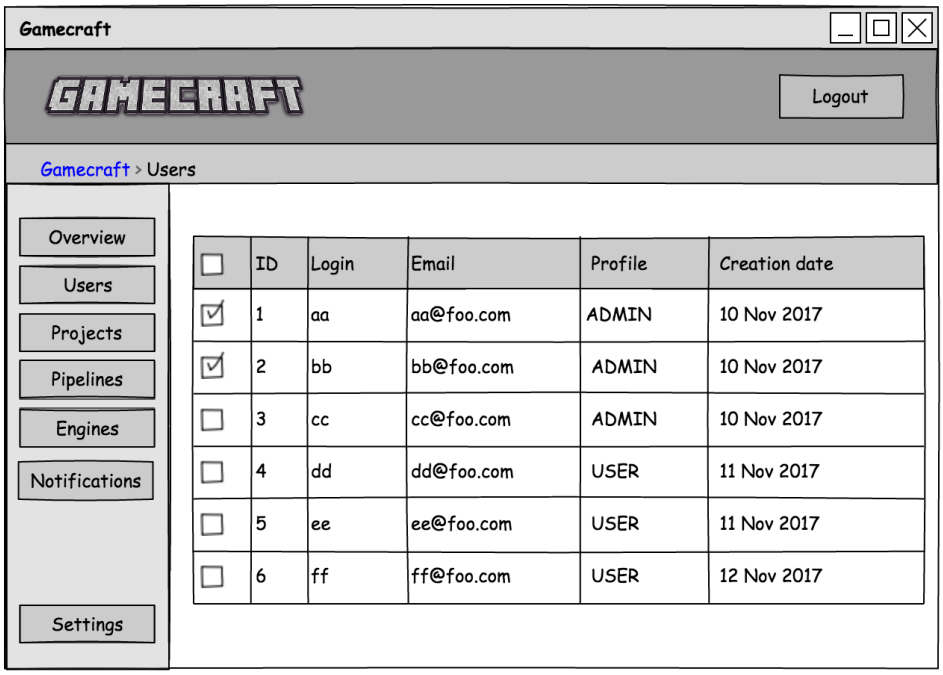

Figura 4.15: Boceto de la pantalla de usuarios

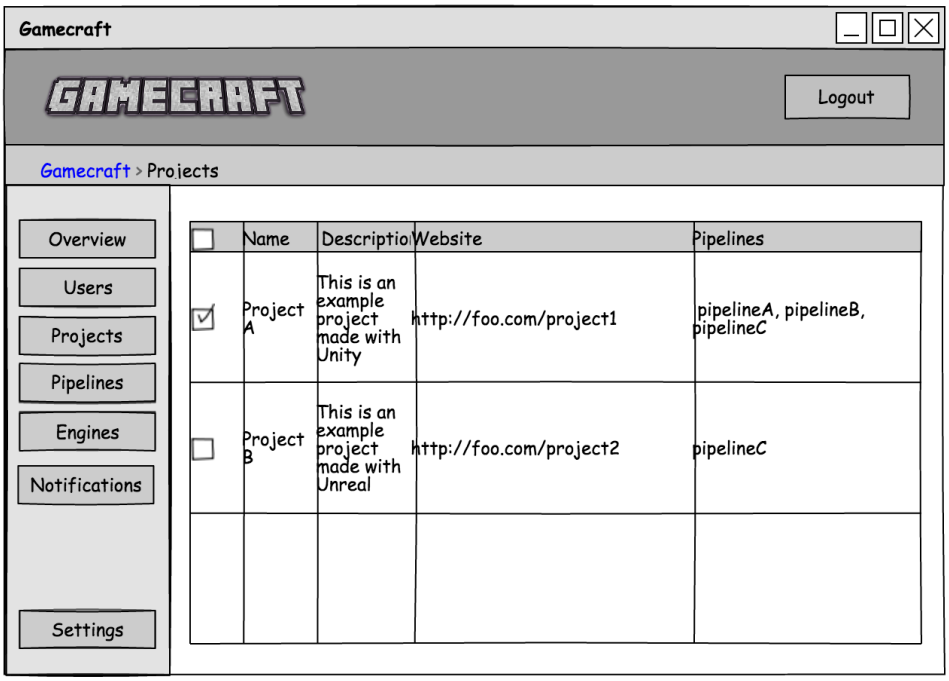

Figura 4.16: Boceto de la pantalla de proyectos

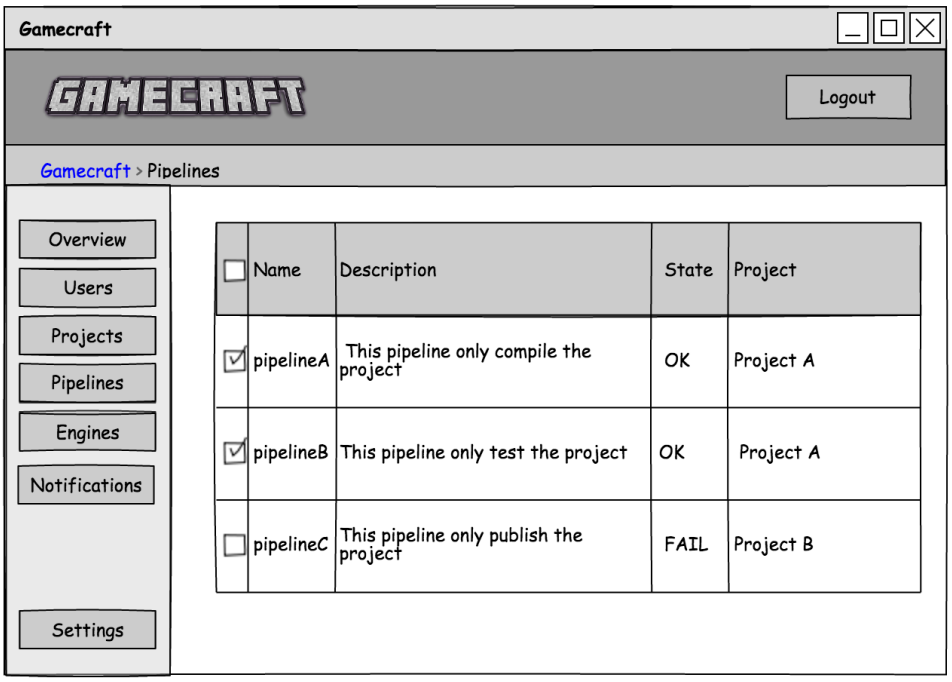

Figura 4.17: Boceto de la pantalla de *pipelines*

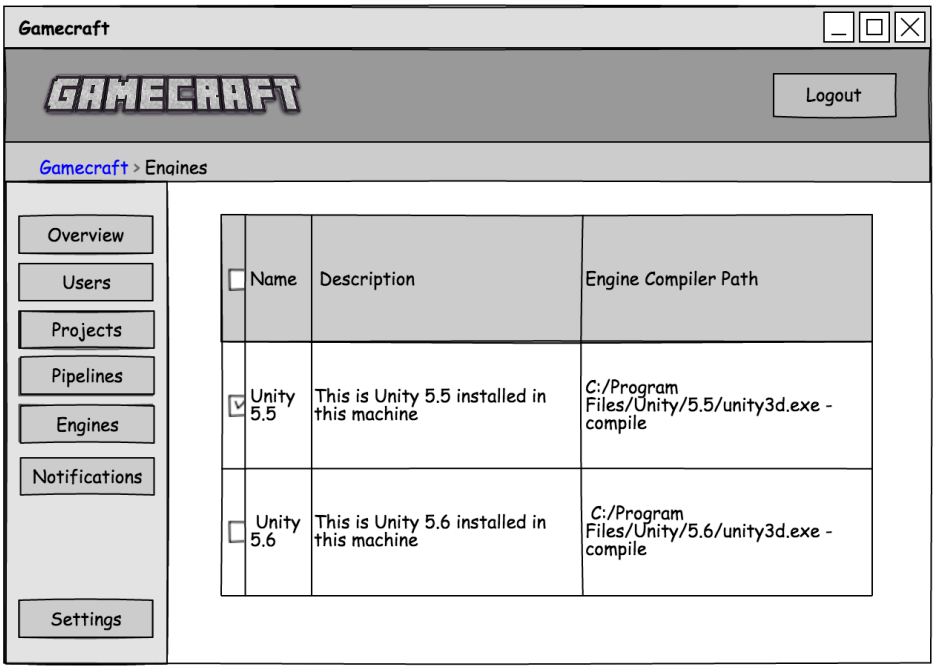

Figura 4.18: Boceto de la pantalla de motores

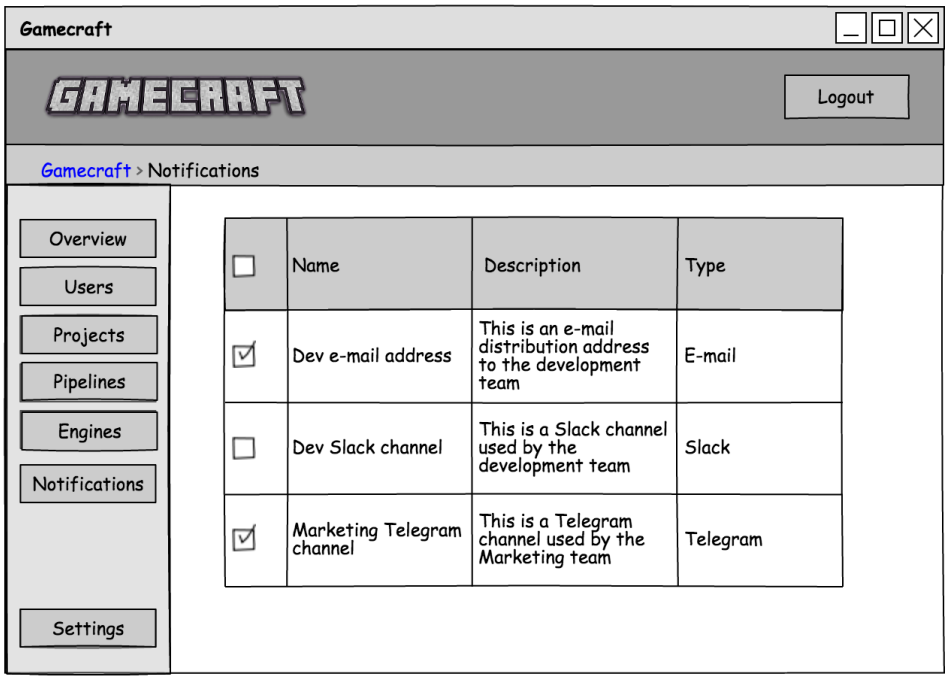

Figura 4.19: Boceto de la pantalla de notificadores

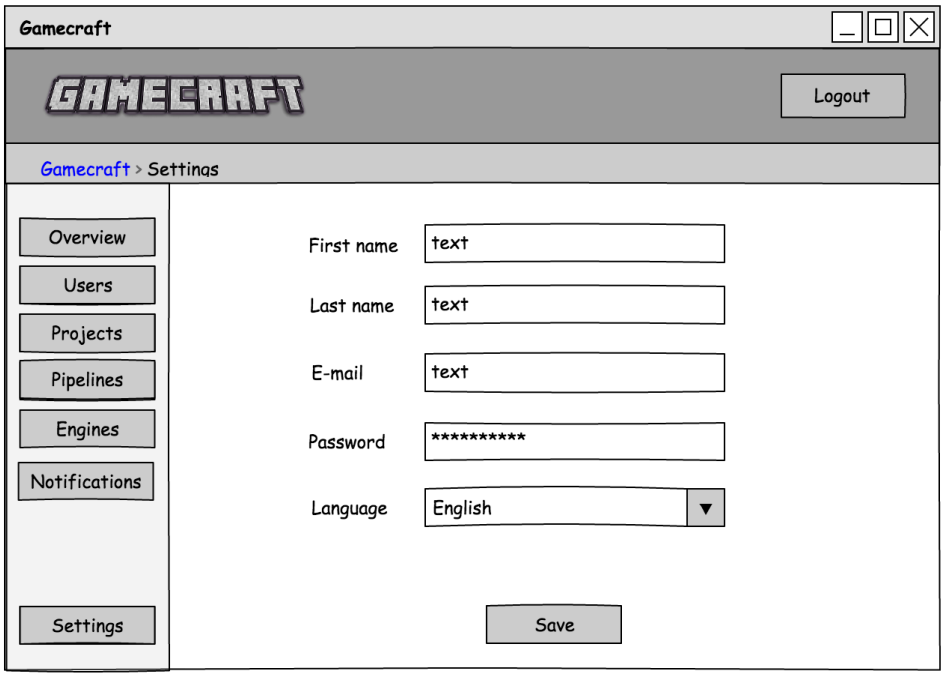

Figura 4.20: Boceto de la pantalla de preferencias del usuario

Capítulo 5

Pruebas y resultados

Dar ejemplo no es la principal manera de influir sobre los demás; es la única manera.

Albert Einstein

Resumen: El capítulo documenta las pruebas realizadas para evaluar el desempeño de la plataforma y también las consultas y los resultados obtenidos tras evaluarla y obtener realimentación con usuarios.

En esta etapa del documento se reflejan las pruebas realizadas al proyecto. Para ello se muestran los resultados en forma de tablas, gráficas e imágenes donde se describe cuantitativa y cualitativamente el funcionamiento de la aplicación y se realiza un análisis crítico de los resultados con el objetivo de decidir si el sistema está preparado para su distribución y, por tanto, es válido para que otras personas puedan hacer uso de él.

Concebimos dos tipos de pruebas:

- **Pruebas de caja blanca**: se realizan sobre las funciones internas de un módulo, componente, clase, etc.
- **Pruebas de caja negra**: comprueban que los requisitos funcionales se han respetado y cumplido. Es decir, permite obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa.

Las pruebas de caja blanca se han ido realizando a la vez que se ha ido implementando el proyecto. En cambio, el diseño de las pruebas de caja negra, que se van a presentar a continuación, se creó en la fase de análisis, sin embargo hemos decidido mostrar toda la información relativa a pruebas unificada en esta sección.

CP - 01	**Creación de un usuario**
Propósito	Comprobar si el proceso de creación de un usuario finaliza con éxito, esto es, el sistema crea una cuenta de usuario con la información proporcionada para que pueda ser utilizado para poder acceder a la plataforma.
Prerrequisito	Ninguno.
Datos de entrada	Cuenta de usuario.
Pasos	1. El usuario hace clic en la opción de Usuarios en el menú superior. 2. El usuario hace clic en el botón de «Crear usuario». 3. El usuario rellena el formulario y confirma la creación haciendo clic en el botón de aceptación.
Resultado esperado	Una cuenta de usuario es creada satisfactoriamente.
Resultado obtenido	Correcto.

Tabla 5.1: CP-01: Creación de un usuario

CP - 02	**Modificación de un usuario**
Propósito	Comprobar si el proceso de modificación de un usuario finaliza con éxito, esto es, el sistema modifica una cuenta de usuario con la información proporcionada.
Prerrequisito	Ninguno.
Datos de entrada	Cuenta de usuario.
Pasos	1. El usuario hace clic en la opción de Usuarios en el menú superior. 2. El usuario hace clic en el botón de «Actualizar usuario». 3. El usuario rellena el formulario y confirma la creación haciendo clic en el botón de aceptación.
Resultado esperado	Los detalles de la cuenta de usuario son modificados con éxito de acuerdo a la información suministrada.
Resultado obtenido	Correcto.

Tabla 5.2: CP-02: Modificación de un usuario

CP - 03	**Supresión de un usuario**
Propósito	Comprobar si el proceso de supresión de un usuario finaliza con éxito, esto es, la cuenta deja de existir en la plataforma y no podrá ser utilizada para poder acceder a ella.
Prerrequisito	Ninguno.
Datos de entrada	Ninguno.
Pasos	1. El usuario hace clic en la opción de Usuarios en el menú superior. 2. El usuario hace clic en el botón de «Eliminar usuario».
Resultado esperado	La cuenta de usuario es eliminada del sistema.
Resultado obtenido	Correcto.

Tabla 5.3: CP-03: Supresión de un usuario

CP - 04	**Creación de un proyecto**
Propósito	Comprobar si el proceso de creación de un proyecto finaliza con éxito, esto es, el programa debería ser capaz de crear un proyecto con la información necesaria.
Prerrequisito	Ninguno.
Datos de entrada	Proyecto.
Pasos	1. El usuario hace clic en la opción de Proyectos en el menú superior. 2. El usuario hace clic en el botón de «Crear proyecto». 3. El usuario rellena el formulario y confirma la creación haciendo clic en el botón de aceptación.
Resultado esperado	Un proyecto es creado en la plataforma sin problemas.
Resultado obtenido	Correcto.

Tabla 5.4: CP-04: Creación de un proyecto

CP - 05	**Modificación de un proyecto**
Propósito	Comprobar si el proceso de modificación de un proyecto finaliza con éxito, esto es, el programa debería ser capaz de modificar un proyecto con la información necesaria.
Prerrequisito	Que exista un proyecto creado.
Datos de entrada	Proyecto.
Pasos	1. El usuario hace clic en la opción de Proyectos en el menú superior. 2. El usuario hace clic en el botón de «Actualizar proyecto». 3. El usuario rellena el formulario y confirma la actualización haciendo clic en el botón de aceptación.
Resultado esperado	Un proyecto es modificado con los nuevos datos introducidos.
Resultado obtenido	Correcto.

Tabla 5.5: CP-05: Modificación de un proyecto

CP - 06	**Supresión de un proyecto**
Propósito	Comprobar si el proceso de supresión de un proyecto finaliza con éxito, esto es, el programa debería ser capaz de eliminar un proyecto.
Prerrequisito	Que exista un proyecto creado.
Datos de entrada	Proyecto.
Pasos	1. El usuario hace clic en la opción de Proyectos en el menú superior. 2. El usuario hace clic en el botón de «Eliminar proyecto».
Resultado esperado	Un proyecto es eliminado del sistema.
Resultado obtenido	Correcto.

Tabla 5.6: CP-06: Supresión de un proyecto

CP - 07	**Configurar notificación por correo electrónico**
Propósito	Comprobar si el proceso de configuración de notificaciones por correo electrónico funciona correctamente, esto es, el programa debería ser capaz de permitir al usuario añadir la posibilidad de proporcionar los datos de acceso a un servidor de correo electrónico y que el sistema los almacene para, posteriormente, cuando se ejecute un pipeline, poder mandar los resultados de la compilación por este método de comunicación.
Prerrequisito	Ninguno.
Datos de entrada	Nombre del notificador de correo electrónico para distinguirlo de otros que hubieran en el sistema, breve descripción del notificador, servidor SMTP, usuario, contraseña y puerto del servidor y uso o n de SSL para la comunicación.
Pasos	1. El usuario hace clic en la opción de Notificaciones en el menú superior. 2. El usuario hace clic en el botón de «Email». 3. El usuario hace clic en el botón de «Crear nuevo notificador». 4. El usuario rellena el formulario y confirma la creación haciendo clic en el botón de guardar cambios.
Resultado esperado	Un notificador es creado con la configuración proporcionada y almacenada en la base de datos.
Resultado obtenido	Correcto.

Tabla 5.7: CP-07: Configurar notificación por correo electrónico

CP - 08	**Configurar notificación por *Telegram***
Propósito	Comprobar si el proceso de configuración de notificaciones por *Telegram* funciona correctamente, esto es, el programa debería ser capaz de permitir al usuario añadir la posibilidad de proporcionar los datos de acceso a una cuenta de *Telegram* y que el sistema los almacene para, posteriormente, cuando se ejecute un pipeline, poder mandar los resultados de la compilación por este método de comunicación.
Prerrequisito	Ninguno.
Datos de entrada	Nombre del notificador de *Telegram* para distinguirlo de otros que hubieran en el sistema, breve descripción del notificador y token de la cuenta de *Telegram*.
Pasos	1. El usuario hace clic en la opción de Notificaciones en el menú superior. 2. El usuario hace clic en el botón de «*Telegram*». 3. El usuario hace clic en el botón de «Crear nuevo notificador». 4. El usuario rellena el formulario y confirma la creación haciendo clic en el botón de guardar cambios.
Resultado esperado	Un notificador es creado con la configuración proporcionada y almacenada en la base de datos.
Resultado obtenido	Correcto.

Tabla 5.8: CP-08: Configurar notificación por *Telegram*

CP - 09	**Configurar notificación por *Slack***
Propósito	Comprobar si el proceso de configuración de notificaciones por *Slack* funciona correctamente, esto es, el programa debería ser capaz de permitir al usuario añadir la posibilidad de proporcionar los datos de acceso a una cuenta de *Slack* y que el sistema los almacene para, posteriormente, cuando se ejecute un pipeline, poder mandar los resultados de la compilación por este método de comunicación.
Prerrequisito	Ninguno.
Datos de entrada	Nombre del notificador de *Slack* para distinguirlo de otros que hubieran en el sistema, breve descripción del notificador y token de la cuenta de *Slack*.
Pasos	1. El usuario hace clic en la opción de Notificaciones en el menú superior. 2. El usuario hace clic en el botón de «*Slack*». 3. El usuario hace clic en el botón de «Crear nuevo notificador». 4. El usuario rellena el formulario y confirma la creación haciendo clic en el botón de guardar cambios.
Resultado esperado	Un notificador es creado con la configuración proporcionada y almacenada en la base de datos.
Resultado obtenido	Correcto.

Tabla 5.9: CP-09: Configurar notificación por *Slack*

CP - 10	**Configurar notificación por *IRC***
Propósito	Comprobar si el proceso de configuración de notificaciones por *IRC* funciona correctamente, esto es, el programa debería ser capaz de permitir al usuario añadir la posibilidad de proporcionar los datos de acceso a un servidor de *IRC* y que el sistema los almacene para, posteriormente, cuando se ejecute un pipeline, poder mandar los resultados de la compilación por este método de comunicación.
Prerrequisito	Ninguno.
Datos de entrada	Nombre del notificador de *IRC* para distinguirlo de otros que hubieran en el sistema, breve descripción del notificador, dirección del servidor de *IRC*, puerto del servidor de *IRC*, *nickname* de la cuenta que se utilizará dentro del servidor *IRC* y uso o no de *SSL* para la comunicación.
Pasos	1. El usuario hace clic en la opción de Notificaciones en el menú superior. 2. El usuario hace clic en el botón de «*IRC*». 3. El usuario hace clic en el botón de «Crear nuevo notificador». 4. El usuario rellena el formulario y confirma la creación haciendo clic en el botón de guardar cambios.
Resultado esperado	Un notificador es creado con la configuración proporcionada y almacenada en la base de datos.
Resultado obtenido	Correcto.

Tabla 5.10: CP-10: Configurar notificación por *IRC*

CP - 11	**Configurar notificación por *Twitter***
Propósito	Comprobar si el proceso de configuración de notificaciones por *Twitter* funciona correctamente, esto es, el programa debería ser capaz de permitir al usuario añadir la posibilidad de proporcionar los datos de acceso a una cuenta de *Twitter* y que el sistema los almacene para, posteriormente, cuando se ejecute un pipeline, poder mandar los resultados de la compilación por este método de comunicación.
Prerrequisito	Ninguno.
Datos de entrada	Nombre del notificador de *Twitter* para distinguirlo de otros que hubieran en el sistema, breve descripción del notificador, *consumer key* y *consumer secret* de la cuenta de Twitter y *access token* y *token secret* de la cuenta de *Twitter*.
Pasos	1. El usuario hace clic en la opción de Notificaciones en el menú superior. 2. El usuario hace clic en el botón de «*Twitter*». 3. El usuario hace clic en el botón de «Crear nuevo notificador». 4. El usuario rellena el formulario y confirma la creación haciendo clic en el botón de guardar cambios.
Resultado esperado	Un notificador es creado con la configuración proporcionada y almacenada en la base de datos.
Resultado obtenido	Correcto.

Tabla 5.11: CP-11: Configurar notificación por *Twitter*

CP - 12	**Creación de un motor**
Propósito	Comprobar si el proceso de creación de un motor finaliza con éxito, esto es, el programa debería ser capaz de crear una referencia a un motor de desarrollo de videojuegos instalada localmente en la máquina. La referencia es almacenada en la base de datos del sistema.
Prerrequisito	Ninguno.
Datos de entrada	Nombre del motor, breve descripción, ruta donde se encuentra instalado el motor y argumentos del motor a ejecutar durante la compilación.
Pasos	1. El usuario hace clic en la opción de *Engines* en el menú superior. 2. El usuario hace clic en el botón de «Crear nuevo motor». 3. El usuario rellena el formulario y confirma la creación haciendo clic en el botón de guardar cambios.
Resultado esperado	Un motor es creado en el sistema con los datos proporcionados por el usuario.
Resultado obtenido	Correcto.

Tabla 5.12: CP-12: Creación de un motor

CP - 13	**Modificación de un motor**
Propósito	Comprobar si el proceso de modificación de un motor finaliza con éxito, esto es, el programa debería ser capaz de modificar una referencia a un motor de desarrollo de videojuegos instalada localmente en la máquina con los datos que el usuario le haya proporcionado.
Prerrequisito	Que exista un motor creado en el sistema.
Datos de entrada	Nombre del motor, breve descripción, ruta donde se encuentra instalado el motor y argumentos del motor a ejecutar durante la compilación.
Pasos	1. El usuario hace clic en la opción de *Engines* en el menú superior. 2. El usuario hace clic en el botón de «Actualizar motor». 3. El usuario rellena el formulario y confirma la actualización haciendo clic en el botón de guardar cambios.
Resultado esperado	Un motor es modificado en el sistema con los datos proporcionados por el usuario.
Resultado obtenido	Correcto.

Tabla 5.13: CP-13: Modificación de un motor

CP - 14	**Supresión de un motor**
Propósito	Comprobar si el proceso de supresión de un motor finaliza con éxito, esto es, el programa debería ser capaz de eliminar un motor previamente creado para que no pueda ser usado más en otros motores.
Prerrequisito	Que exista un motor creado en el sistema.
Datos de entrada	*Engine*.
Pasos	1. El usuario hace clic en la opción de *Engines* en el menú superior. 2. El usuario hace clic en el botón de «Eliminar motor».
Resultado esperado	Un motor es eliminado del sistema.
Resultado obtenido	Correcto.

Tabla 5.14: CP-14: Supresión de un motor

CP - 15	**Creación de un *pipeline***
Propósito	Comprobar si el proceso de creación de un *pipeline* finaliza con éxito, esto es, el programa debería ser capaz de crear un *pipeline* con la información proporcionada por el usuario en el formulario. El *pipeline* es almacenado en la base de datos del sistema.
Prerrequisito	Ninguno.
Datos de entrada	Nombre del *pipeline*, breve descripción del *pipeline*, dirección y credenciales (usuario y contraseña) para acceder al repositorio del proyecto, motor a utilizar para la compilación del proyecto, dirección y credenciales (usuario y contraseña) para subir el resultado de la compilación a un servidor *FTP*, token de acceso para subir el resultado de la compilación a *Dropbox*, notificador a utilizar para enviar el resultado final y programación del *pipeline* (si se va a ejecutar periódicamente, como una tarea *cron*, o se hará uso de *Webhook*) .
Pasos	1. El usuario hace clic en la opción de Proyectos en el menú superior. 2. El usuario hace clic en el botón de *Pipelines* para el proyecto que quiere trabajar. 3. El usuario hace clic en el botón de «Crear *pipeline*». 4. El usuario rellena el formulario y confirma la inserción haciendo clic en el botón de guardar cambios.
Resultado esperado	Un *pipeline* es creado en el sistema con los datos proporcionados por el usuario.
Resultado obtenido	Correcto.

Tabla 5.15: CP-15: Creación de un *pipeline*

CP - 16	**Modificación de un *pipeline***
Propósito	Comprobar si el proceso de modificación de un *pipeline* finaliza con éxito, esto es, el programa debería ser capaz de modificar un *pipeline* con la información proporcionada por el usuario en el formulario. El *pipeline* es almacenado en la base de datos del sistema.
Prerrequisito	Que exista un *pipeline* creado en el sistema.
Datos de entrada	Nombre del *pipeline*, breve descripción del *pipeline*, dirección y credenciales (usuario y contraseña) para acceder al repositorio del proyecto, motor a utilizar para la compilación del proyecto, dirección y credenciales (usuario y contraseña) para subir el resultado de la compilación a un servidor *FTP*, token de acceso para subir el resultado de la compilación a *Dropbox*, notificador a utilizar para enviar el resultado final y programación del *pipeline* (si se va a ejecutar periódicamente, como una tarea *cron*, o se hará uso de *Webhook*).
Pasos	1. El usuario hace clic en la opción de Proyectos en el menú superior. 2. El usuario hace clic en el botón de *Pipelines* para el proyecto que quiere trabajar. 3. El usuario hace clic en el botón de «Modificar pipeline». 4. El usuario rellena el formulario y confirma la modificación haciendo clic en el botón de guardar cambios.
Resultado esperado	Un *pipeline* es actualizado en el sistema con los nuevos datos proporcionados por el usuario.
Resultado obtenido	Correcto.

Tabla 5.16: CP-16: Modificación de un *pipeline*

CP - 17	**Supresión de un *pipeline***
Propósito	Comprobar si el proceso de supresión de un *pipeline* finaliza con éxito, esto es, el programa debería ser capaz de eliminar un *pipeline* creado en el sistema para que no pueda ser invocado más veces en el futuro.
Prerrequisito	Que exista un *pipeline* creado en el sistema.
Datos de entrada	*Pipeline*.
Pasos	1. El usuario hace clic en la opción de Proyectos en el menú superior. 2. El usuario hace clic en el botón de *Pipelines* para el proyecto que quiere trabajar. 3. El usuario hace clic en el botón de «Eliminar *pipeline*».
Resultado esperado	El *pipeline* es eliminado del sistema.
Resultado obtenido	Correcto.

Tabla 5.17: CP-17: Supresión de un *pipeline*

CP - 18	**Ejecución de un *pipeline***
Propósito	Comprobar si el proceso de ejecución de un *pipeline* finaliza con éxito, esto es, el programa debería ser capaz de poder ejecutar un *pipeline* creado previamente en el sistema y de generar la información de salida que posteriormente será vista al usuario para que sepa como ha ido la compilación. Cuando finaliza el *pipeline*, se envía una notificación utilizando el notificador elegido en el proceso de creación del *pipeline*.
Prerrequisito	Que exista un *pipeline* en el sistema y funcione correctamente.
Datos de entrada	Pipeline.
Pasos	1. El usuario hace clic en la opción de Proyectos en el menú superior. 2. El usuario hace clic en el botón de *Pipelines* para el proyecto que quiere trabajar. 3. El usuario hace clic en el botón de «Ejecutar *pipeline*».
Resultado esperado	El *pipeline* es ejecutado y la información de salida es mostrada por pantalla. Se envía una notificación para comunicar si el *pipeline* ha funcionado correctamente o, si por el contrario, ha fracasado.
Resultado obtenido	Correcto.

Tabla 5.18: CP-18: Ejecución de un *pipeline*

5.1. Evaluación teórica de la propuesta

Durante el desarrollo de este proyecto, se llevó a cabo la difusión de una encuesta[1] a varios desarrolladores de videojuegos para que diesen su opinión sobre la integración continua y sobre la propuesta de *GameCraft*, con el objetivo de tomar contacto con ellos y obtener más información sobre lo que conocen y valoran los usuarios sobre las características de las herramientas de integración continua. Los 17 usuarios que completaron esta encuesta trabajan en empresas de videojuegos, o forman parte de estudios independientes o en algunos casos son profesores o estudian carreras universitarias relacionadas con el sector.

Los resultados de la encuesta se adjuntan a continuación, agrupando todas las respuestas obtenidas en cada una de las preguntas de la encuesta:

- **Q1 - Tengo experiencia usando herramientas de integración continua en el trabajo.**

[1]Una versión en línea de la encuesta se puede obtener en el siguiente enlace: `https://docs.google.com/forms/d/1UljfP9vSApvScdg8iJBqTTE7OkZXv-e_w25z9RbhQ4U`

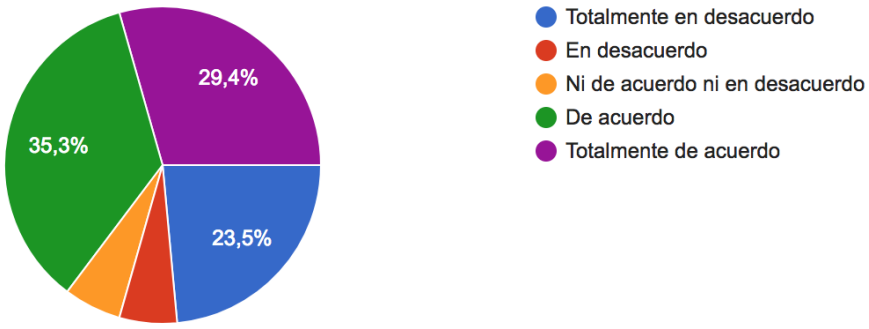

Figura 5.1: Resumen de las respuestas obtenidas a la primera pregunta de la encuesta

- **Q2 - ¿Qué herramientas de integración continua has utilizado?**
  - SVN, Git.
  - SVN, Git.
  - Mercurial, Git y Perforce.
  - En mi trabajo en el estudio de videojuegos ninguna, en el otro trabajo me han llegado a explicar el uso de Jenkins.
  - Jenkins pero solo para probar. Mi empresa no usa herramientas de este tipo por desgracia.
  - TFS.
  - MercuryEngine (motor propio de Mercury).
  - AgileCraft, Scrum, JIRA.
  - Bitbucket, con cliente TortoiseHG para mercurial y SourceTree para Git.
  - SVN, Git, Perforce, Mercurial.
  - Git, SVN, Mercurial.
  - Jenkins, Gitlab CI.
  - Perforce, tortoise svn.
  - Perforce.
  - Jenkins pipeline.
  - Ninguna.
  - AppVeyoy y TravisCI con GitHub.
- **Q3 - Considero que la integración continua es un asunto exclusivo de los programadores.**

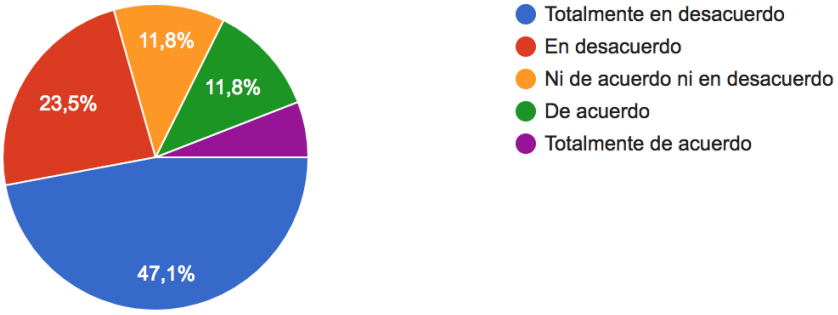

Figura 5.2: Resumen de las respuestas obtenidas a la tercera pregunta de la encuesta

- **Q4 - Considero que mi empresa actual da importancia a la integración continua.**

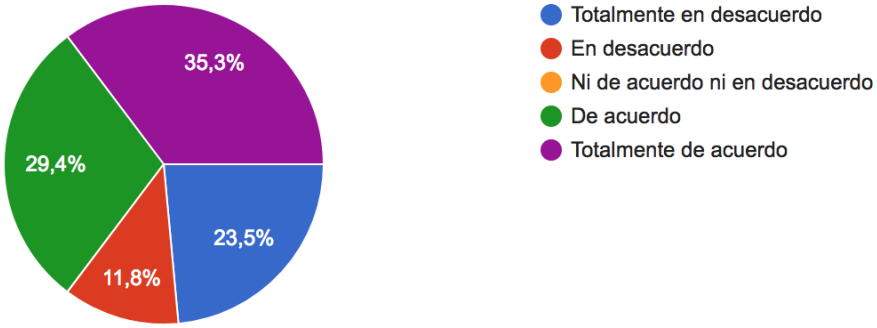

Figura 5.3: Resumen de las respuestas obtenidas a la cuarta pregunta de la encuesta

- **Q5 - Considero que la integración continúa debería ser más importante en mi empresa actual.**

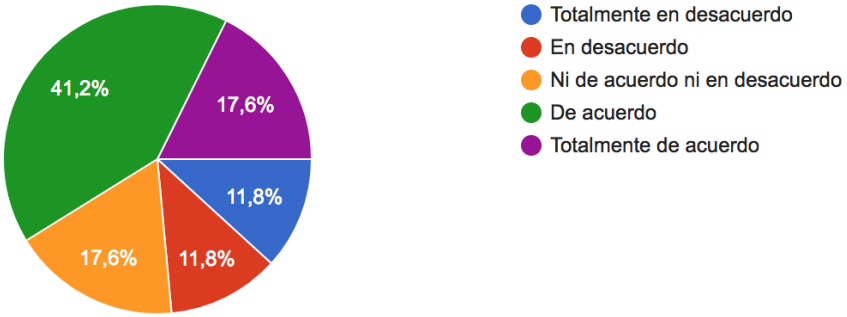

Figura 5.4: Resumen de las respuestas obtenidas a la quinta pregunta de la encuesta

- **Q6 - En el uso profesional de las herramientas de integración continua, ¿cuales son los principales obstáculos y dificultades?**
  - El trabajo con archivos binarios y las herramientas, a veces, poco intuitivas.
  - Mentalizar a TODO el equipo de que cualquier cosa que se suba al repositorio debe ser probada, aún si no hay herramientas de integración continua. Entender bien cómo se puede aplicar la integración contínua en videojuegos de cierto tamaño porque no lo veo nada claro más allá de que çompila, luego está OK.º, a lo sumo, pruebas unitarias tontorronas para métodos sueltos de cierta complejidad.
  - No querer aprender por parte de mis superiores.
  - TFS lo da para tontos, no sé.
  - Se requiere de un equipo potente -y dedicado exclusivamente a esta tarea- para que las pruebas se ejecuten con la rapidez necesaria. A medida que el proyecto crece, cada ciclo requiere más tiempo. Por lo que es útil que la herramienta acepte diferentes parámetros como hacer build o re-build del proyecto, poder precompilar scripts como los .lua, seleccionar solo ciertas secciones de código para la integración continua (por ejemplo X niveles de un juego en vez de todos), etc. Por la misma causa que el punto anterior, puede llegar a ser necesario tener 2 máquinas dedicadas a integración continua, cada una encargándose de una parte diferente (o incluso tener varias integraciones continuas en cada máquina). La integración continua puede llegar a ser una responsabilidad del equipo de *QA*. Pero esto requiere que el responsable tenga conocimientos técnicos para interpretar los errores y aumentar la eficiencia de este proceso.
  - Dificultad de Implantación, cuestionables ventajas.
  - Curva de aprendizaje para los no técnicos. Problemas de *merges*.
  - Integración de archivos binarios.

- La adaptación de un usuario primerizo al uso de herramientas como estas. Normalmente suelen generar problemas durante cierto tiempo. A veces pueden generar caos por no realizar correctamente el protocolo exigido.
- El aprendizaje de los novatos.
- *Merging* y *branching*.
- El uso de lenguajes *script* sin compilador causa muchas iteraciones hasta que el *script* funciona
- Entender la filosofía y aplicarlo de forma continua.
- En nuestra experiencia ha sido la instalación y configuración inicial del servidor.
- My Spanish is not good, but I can read. The main difficulties is that developers do not understand the importance of CI, and not willing to write unit tests.
- La comunicación.
- Setup para multiples plataformas "no estandard".

- **Q7 - ¿Qué características y facilidades valorarías más en una hipotética herramienta de integración continua?**
  - Que sea gratis. (7 votos)
  - Que sea de código abierto. (4 votos)
  - Que se integre fácilmente con *Unity* o *Unreal*. (8 votos)
  - Que esté bien documentado. (9 votos)
  - **Que sea fácil de instalar y configurar. (11 votos)**
  - Que sea fácilmente escalable. (8 votos)
  - Que esté tanto en español como en inglés. (3 votos)
  - Que ofrezca servicios adicionales para la mejora de productividad como analizador de calidad de código. (4 votos)
  - Que su funcionalidad pueda ampliarse mediante *plug-ins*. (7 votos)
  - Otro: Multiplataforma. (1 voto)
  - Otro: Interfaz sencilla. (1 voto)
  - Otro: Que sea fácil de usar y visualmente atractivo. (1 voto)
  - Otro: Compatible con Docker. (1 voto)

- **Q8 - ¿Qué aspectos y funcionalidades valoras más en las herramientas de integración continua que ya has utilizado?**
  - Soporte para *macOS*.(7 votos)
  - Soporte para *GNU/Linux*. (4 votos)
  - **Soporte para *Windows*. (12 votos)**
  - Que sea gratuito. (7 votos)
  - Que sea de código abierto. (2 votos)

- Que esté traducido en varios idiomas. (1 voto)
- Que pueda integrarse con *plug-ins* de terceros. (3 votos)
- Que tenga soporte nativo al engine *Unity*. (4 votos)
- Que tenga soporte nativo al engine *Unreal*. (4 votos)
- Que tenga soporte nativo al engine *Monogame*. (0 votos)
- Que tenga soporte nativo al engine *Libgdx*. (0 votos)
- Que cuente con un buen analizador de calidad de código. (4 votos)
- Que tenga buena documentación. (5 votos)
- Que sea fácil de instalar. (10 votos)
- Que sea fácil de configurar. (13 votos)
- Otro: Que sean muy adaptables al proyecto en desarrollo. (1 voto)
- Otro: Poder excluir fácilmente archivos a sincronizar. (1 voto)
- Otro: Que sea fácil de usar. (1 voto)

- **Q10 - Por favor, escribe aquí cualquier comentario, aclaración o sugerencia que nos quieras hacer llegar (¡nos será de mucha ayuda!)**

  - Creo que diferenciaría en el estudio a estudios *indies* de "no *indies*za sea por tamaño, financiación actual, etc. para ver cómo de importante es, incluso siendo *indie*, dedicar tiempo a temas como éste.
  - Supongo que enfocándolo al desarrollo de videojuegos abarcaréis otros campos menos específicos (y más fáciles de testear)
  - El repositorio artístico necesita muchos *Gb* que suelen superar los planes gratuitos. Por falta de tiempo no hemos implementado, ni análisis de código, ni un compilador automático o testeos.
  - Solo somos dos en nuestra empresa, no trabajamos en proyectos conjuntos y aún así utilizamos Git para llevar un control de estados de lo que vamos haciendo. Es tremendamente útil en todos los casos.
  - Habiendo ya varios sistemas de *CI/CD open source* como *jenkins*, *gitlab CI*, *Travis*, etc... ¿Por qué crear otro más?
  - What is the advantage against other products ?

  Del análisis de las respuesta a la encuesta realizada se pueden extraer varias ideas:

  - Algunos usuarios no saben distinguir realmente lo que es una herramienta de integración continua, poniendo herramientas y tecnologías que no son tales como *Git*, *SVN* o *Bitbucket* en Q2. Esto se debe a que herramientas de este tipo no se aplican en muchas organizaciones y empresas, y por tanto, los usuarios no tienen claro qué responder ya que no han conocido o no han tenido la oportunidad de utilizar estas herramientas y ver el beneficio que generan tanto a la empresa en costes temporales y económicos como a los trabajadores.

- Existe una mayoría que considera que la integración continua es un asunto que también incumbe al resto de equipos de la empresa y no solo al de desarrollo. Muchas plataformas como *Jenkins* poseen interfaces de usuarios que no han sido diseñadas específicamente para ser utilizadas por personas que carecen de conocimientos informáticos, por lo que, en muchas ocasiones, cuando es necesario configurar o navegar por los menús, estas se pierden con facilidad al no comprender lo que se muestra en la pantalla. *GameCraft* trata de resolver esto al ofrecer una interfaz de usuario sin utilizar un lenguaje demasiado técnico y sin abrumar al usuario con un gran conjunto de opciones, las cuales muchas de ellas no son de utilidad para las necesidades de la mayoría de proyectos.
- Existe una mayoría que piensa que la integración continua debería ser más importante en el lugar de trabajo. Esto permite afirmar que la gente es consciente de algunas de las ventajas que el uso de estas herramientas puede ofrecer al equipo y posiblemente no dudarían en recomendar herramientas como Jenkins o la propia *GameCraft* a sus superiores si eso les ayuda en el trabajo.
- La mayoría de gente pide que una herramienta de este tipo sea fácil de instalar y configurar, lo que confirma y justifica una de las prioridades de diseño de *GameCraft*.
- Algunas personas encuestadas no vieron las ventajas de *GameCraft* frente al resto de alternativas que existen en el mercado. Si bien, es cierto que se pueden utilizar herramientas como *Jenkins* o *Travis CI* para poder tratar proyectos de videojuegos, su dificultad en la instalación y configuración, el hecho de que muchas de las soluciones existentes en el mercado son de pago y que no haya una solución desarrollada a partir de una arquitectura de microservicios, hacen que *GameCraft* sea una alternativa sólida frente a herramientas ya asentadas. Además, *GameCraft* es un proyecto gratuito y de código abierto, por lo que puede ser adaptado para cualquier entorno de trabajo sin mucha complejidad.

5.2. Evaluación práctica con usuarios

El objetivo de la evaluación práctica con usuarios es comprobar la funcionalidad y usabilidad de la plataforma con un conjunto relevante y lo más realista posible de usuarios finales para asegurar que el proyecto pueda ser usado sin problemas en entornos de producción. El principal objetivo es asegurar hasta qué punto se cumplen los requisitos y funcionalidades pedidas como, por ejemplo, eficiencia o usabilidad. El segundo objetivo es propor-

cionar retroalimentación general a los equipos de desarrollo e investigación para que puedan llevar a cabo mejoras de forma más rápida.

La validación se centrará en la implementación del proyecto (en las funcionalidades que los usuarios ven de la plataforma). Los resultados se documentan aquí para que puedan ser tratados en futuros hitos del proyecto.

En un escenario general, los beneficios que se esperan de usar esta plataforma son:

- Incrementar productividad del usuario debido a la utilización de la plataforma en la compilación de varios proyectos.
- Reducir la complejidad y el tiempo de desarrollo.
- Proporcionar una herramienta que pueda ser utilizado tanto por desarrolladores independientes como para empresas en sus proyectos, en entornos de producción real.

El criterio de evaluación se basará en los requisitos funcionales desde el punto de vista del usuario que va a utilizar la plataforma. Las variables más importantes se listan a continuación junto con una breve descripción:

- Productividad del usuario: tiempo dedicado para configurar la plataforma, tiempo dedicado para compilar un proyecto, calidad del resultado, tasa de error.
- Experiencia personal del usuario con la plataforma: satisfacción del usuario, curva de aprendizaje, interactividad.
- Aceptación: voluntad de adoptar la plataforma, tasa de adopción entre los usuarios.

Basándose en estas variables, la plataforma será juzgada de acuerdo al efecto observado en el comportamiento de los usuarios y en su experiencia personal. Los resultados de cada una de las pruebas se obtienen a través de un cuestionario completado por las personas que se ofrecen a probar el proyecto. En total, han sido 8 desarrolladores los que han realizado esta evaluación.

5.2.1. Escenario de usuario

La interfaz de usuario de la plataforma ha sido diseñada para cumplir con el objetivo de conseguir que sea una interfaz simple de usar y versátil. La página principal, una vez iniciada la sesión, ofrece de un simple vistazo todas las funcionalidades implementadas en la plataforma y que pueden ser utilizadas por la plataforma:

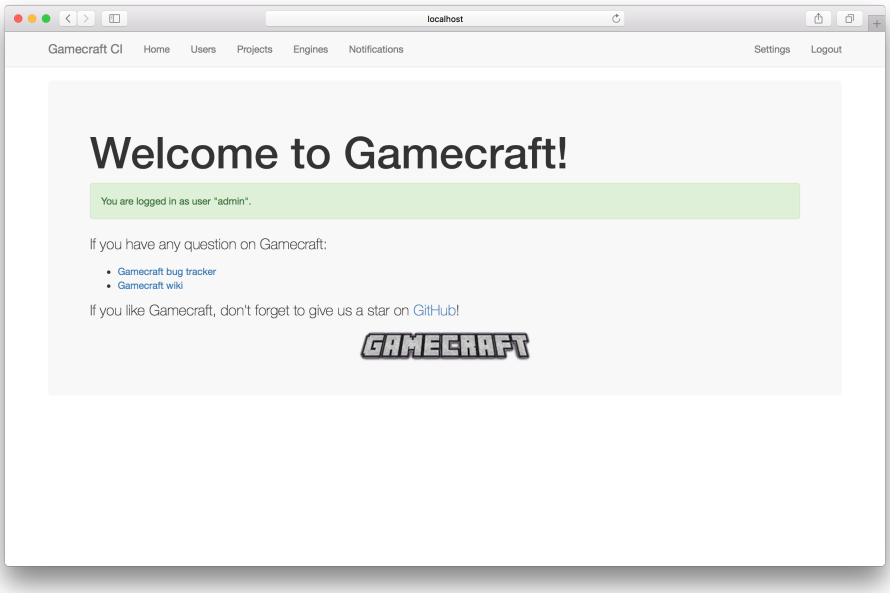

Figura 5.5: Pantalla principal de Gamecraft

A continuación, se explica cada una de las opciones del menú superior:

- *Home*. Se muestra la pantalla que se puede ver en la Figura F.2.
- *Users*. Se muestran todos los usuarios creados y activos que pueden acceder a la plataforma. Desde esta pantalla, se pueden crear nuevos usuarios, modificar los ya existentes o eliminarlos.
- *Projects*. Se muestran todos los proyectos creados y activos que están siendo procesados por la plataforma. Desde esta pantalla, se pueden crear nuevos proyectos, modificar los ya existentes o eliminarlos. Además, desde esta pantalla se pueden acceder a los *pipelines* creados por cada proyecto existente en la plataforma.
- *Engines*. Se muestran todos los motores creados y activos que están siendo utilizados por la plataforma para las tareas de compilación. Desde esta pantalla, se pueden crear nuevos *pipelines*, modificar los ya existentes o eliminarlos.
- *Notifications*. Se muestran todos los notificadores creados y activos que están siendo utilizados por la plataforma para comunicar eventos al usuario. Al abrir esta opción en el menú, se mostrará un desplegable con las siguientes opciones:
    - *E-mail*: Se muestran todos los notificadores de correo electrónico creados y activos que están siendo utilizados por la plataforma.

Desde esta pantalla, se pueden crear nuevos notificadores, modificar los ya existentes o eliminarlos.

- *Twitter*: Se muestran todos los notificadores de *Twitter* creados y activos que están siendo utilizados por la plataforma. Desde esta pantalla, se pueden crear nuevos notificadores, modificar los ya existentes o eliminarlos.
- *Telegram*: Se muestran todos los notificadores de *Telegram* creados y activos que están siendo utilizados por la plataforma. Desde esta pantalla, se pueden crear nuevos notificadores, modificar los ya existentes o eliminarlos.
- *Slack*: Se muestran todos los notificadores de *Slack* creados y activos que están siendo utilizados por la plataforma. Desde esta pantalla, se pueden crear nuevos notificadores, modificar los ya existentes o eliminarlos.
- *IRC*: Se muestran todos los notificadores de *IRC* creados y activos que están siendo utilizados por la plataforma. Desde esta pantalla, se pueden crear nuevos notificadores, modificar los ya existentes o eliminarlos.

- *Settings*. Se muestran las opciones del usuario. Desde esta pantalla se puede cambiar el nombre de usuario, la contraseña y otros datos personales de la cuenta de usuario con la que se ha iniciado sesión.
- *Logout*. Cierra la sesión de la cuenta con la que se ha accedido anteriormente.

Con la plataforma en funcionamiento, el usuario puede empezar a compilar automáticamente proyectos.

Acción 1 para los usuarios: Iniciar sesión utilizando la cuenta de administración.

GameCraft utiliza un sistema de autenticación basado en cuentas de usuario para verificar la identidad de las personas. Una persona sin cuenta no puede acceder a la plataforma y debería ponerse en contacto con el administrador de la misma para que le cree una cuenta de usuario. En esta parte de la evaluación, vamos a proceder a iniciar la sesión usando la cuenta de administración.

1. Acceder a la página principal de *GameCraft*. Tras desplegar la plataforma, está disponible en la siguiente dirección: `http://localhost:8000/`.
2. Introducir en el campo de *Username*: `admin`.
3. Introducir en el campo de *Password*: `admin`.

4. Hacer clic en el botón `Log in`.

Acción 2 para los usuarios: Cambiar la contraseña de la cuenta de administración.

Cambiar la contraseña de la cuenta de administración es la primera tarea después de desplegar la plataforma, ya que cualquier persona que tuviera conocimiento de este documento, sabrá de antemano cual es el usuario y contraseña de esta cuenta. En esta parte de la evaluación, vamos a proceder a cambiar la contraseña para evitar accesos no autorizados que pudieran poner en riesgo la integridad del sistema.

1. Una vez iniciada la sesión y en la página principal, en el menú superior, hacer clic en `Settings`.
2. En el campo de `Password` introducir la nueva contraseña.
3. Hacer clic en el botón `Save settings`.
4. Hacer clic en el botón `Logout`.
5. Introducir en el campo de *Username*: `admin`.
6. Introducir en el campo de *Password*: la nueva contraseña.
7. Hacer clic en el botón `Log in`.

Acción 3 para los usuarios: Crear un proyecto.

En *GameCraft*, un proyecto corresponde a un videojuego que es necesario compilar de forma automática dentro de la plataforma. Por ello, para poder aprovechar al máximo la plataforma, es necesario crear un proyecto sobre el cual crearemos los sucesivos *pipelines* que se irán ejecutando a diferentes intervalos de tiempo e irán compilando el código fuente del videojuego en cuestión. En esta parte de la evaluación, vamos a proceder a crear un proyecto para, más adelante, crear un *pipeline* y ejecutarlo para ver el resultado.

1. Una vez iniciada la sesión y en la página principal, en el menú superior, hacer clic en `Projects`.
2. Hacer clic en el botón `Create new project`.
3. Introducir en el campo de *Project Name*: `Example videogame`.
4. Introducir en el campo de *Project Description*: `This is an example to see how GameCraft works`.
5. Introducir en el campo de *Project Webpage*: `http://www.foo.com`.
6. Hacer clic en el botón `Save changes`.

Acción 4 para los usuarios: Crear un *motor*.

Un *motor* es el binario encargado de compilar y ejecutar los tests para un proyecto en concreto. Un *motor* podría ser el compilador `gcc`, por ejemplo, si estuviésemos hablando de un videojuego desarrollado en C++ o del propio ejecutable de *Unity*. En esta parte de la evaluación, vamos a proceder a crear una referencia a la versión de *Unity* que está instalada localmente en la máquina de demostración.

1. En la página principal, en el menú superior, hacer clic en `Engines`.
2. Hacer clic en el botón `Create new engine`.
3. Introducir en el campo de *Engine Name*: `Unity3D`.
4. Introducir en el campo de *Engine Description*: `Reference to a local installation of Unity3D`.
5. Introducir en el campo de *Engine Compiler Path*: `/Applications/Unity/Unity.app/Contents/MacOS/Unity`.
6. Introducir en el campo de *Engine Compiler Arguments*: `-batchmode -projectPath $TMPDIR -executeMethod Build.Build_iOS_Device -quit -logFile devstdout`.
7. Hacer clic en `Save changes`.

Acción 5 para los usuarios: Crear un notificador de Slack.

Un notificador es el encargado de entablar las comunicaciones a entidades externas a la plataforma. Un notificador se utiliza cuando se quiere saber el estado de ejecución de un *pipeline* de forma rápida, sin necesidad de acceder a la plataforma. En *GameCraft* se da soporte a varios notificadores diferentes aunque en esta parte de la evaluación, vamos a crear un notificador que enviará mensajes privados de *Slack* en concreto para comunicar el estado de la ejecución del *pipeline* que ejecutaremos más adelante.

1. En la página principal, en el menú superior, hacer clic en `Notifications`.
2. Hacer clic en `Slack`.
3. Hacer clic en el botón `Create new notificator`.
4. Introducir en el campo de *Notificator Name*: `Slack`.
5. Introducir en el campo de *Notificator Description*: `Use Slack to deliver updates`.
6. Introducir en el campo de *Slack Token* un token válido de una cuenta de Slack. En la sección F.5.1.3 existe una guía para obtener este dato.

7. Hacer clic en `Save changes`.

Acción 6 para los usuarios: Crear un *pipeline*

Un *pipeline* es una secuencia de tareas que se irán ejecutando secuencialmente por la plataforma para poder compilar con éxito un proyecto. Para poder crear, modificar y eliminar *pipelines* es necesario haber creado antes un proyecto. En esta parte de la evaluación, vamos a proceder a crear un *pipeline* en el proyecto creado en la acción anterior.

1. En la página principal, en el menú superior, hacer clic en `Projects`.
2. En la página de listado de proyectos, en la fila que hace referencia al proyecto creado en esta evaluación, hacer clic en el botón de más a la derecha de la columna *Operations*.
3. Hacer clic en `Create new pipeline`.
4. Introducir en el campo de *Pipeline Name*: `Example pipeline`.
5. Introducir en el campo de *Pipeline Description*: `This pipeline will compile the project once there are any update in the repo`.
6. Introducir en el campo de *Github Repository Username*: tu usuario de Github.
7. Introducir en el campo de *Github Repository Password*: tu contraseña de Github.
8. Introducir en el campo de *Github Repository Address*: la dirección del repositorio donde se encuentra el videojuego.
9. Introducir en el campo de *Github Repository Branch*: la rama donde se encuentra el código del videojuego.
10. Seleccionar en el campo de *Engine name*: el *motor* creado anteriormente.
11. Introducir en el campo de *FTP Address*: `localhost`.
12. Introducir en el campo de *FTP Port*: `22`.
13. Introducir en el campo de *FTP Username*: `demo`.
14. Introducir en el campo de *FTP Password*: `demodemo`.
15. Seleccionar en el campo de *Notificator Type*: `Slack`.
16. Seleccionar en el campo de *Notificator name*: el notificador creado anteriormente.

17. Seleccionar en el campo de *Schedule Type*: `WebHook`.

18. Hacer clic en `Save changes`.

Acción 7 para los usuarios: Crear un nuevo usuario con rol "Usuario".

Hasta ahora hemos utilizado la cuenta de administración para configurar la plataforma pero es conveniente que los miembros del equipo de desarrollo tengan una cuenta de acceso a la plataforma para que puedan comprobar el estado y controlar las ejecuciones de los *pipelines*. En esta parte de la evaluación, vamos a proceder a crear una cuenta de usuario sin permisos de administración.

1. En la página principal, en el menú superior, hacer clic en `Users`.

2. Hacer clic en `Create new user`.

3. Rellenar los campos de *Username*, *First Name*, *Last Name*,*E-mail* y *Password* con los datos que se quieran.

4. Seleccionar en el campo de *Role*: `User`.

5. Hacer clic en `Save changes`.

Acción 8 para los usuarios: Iniciar sesión utilizando la cuenta de usuario.

Una vez creado el usuario en la acción anterior, vamos a probar que realmente se ha creado y está funcionando accediendo con él en la plataforma. En esta parte de la evaluación, cerraremos sesión con la cuenta de administración y volveremos a abrir sesión con la cuenta de usuario.

1. Hacer clic en el botón `Log out`.

2. Introducir en el campo de *Username*: el nombre de usuario creado anteriormente.

3. Introducir en el campo de *Password*: la contraseña de la cuenta creada anteriormente.

4. Hacer clic en el botón `Log in`.

Acción 9 para los usuarios: Listar proyectos como usuario normal.

Una vez iniciada la sesión con la cuenta de usuario, vamos a acceder a la página de proyectos para apreciar las diferencias que existen cuando accedimos a esta página con la cuenta de administración.

1. En la página principal, en el menú superior, hacer clic en `Projects`.

2. La principal diferencia con lo que aparecía antes usando la cuenta de administración son las operaciones, ya que antes podíamos crear, editar y borrar y ahora solo podemos visualizar los *items* creados y acceder a los *pipelines* que el administrador ha creado dentro de cada uno de los proyectos.

Acción 10 para los usuarios: Listar *motores* como usuario normal.

Una vez iniciada la sesión con la cuenta de usuario, vamos a acceder a la página de *motores* para apreciar las diferencias que existen cuando accedimos a esta página con la cuenta de administración.

1. En la página principal, en el menú superior, hacer clic en `Engines`.

2. La principal diferencia con lo que aparecía antes usando la cuenta de administración son las operaciones, ya que antes podíamos crear, editar y borrar y ahora solo podemos visualizar los *items* creados.

Acción 11 para los usuarios: Ejecutar manualmente un *pipeline*

Como parte final de esta evaluación, se va a proceder a ejecutar el *pipeline* creado anteriormente. Tras la ejecución, vamos a ver como se conectan los diferentes microservicios de la plataforma para recuperar y enviar información y como se ha compilado con éxito el proyecto de ejemplo.

1. En la página principal, en el menú superior, hacer clic en `Projects`.

2. En la página de listado de proyectos, en la fila que hace referencia al proyecto creado en esta evaluación, hacer clic en el botón de más a la derecha de la columna *Operations*.

3. En la fila del *pipeline* creado anteriormente, hacer clic en el botón verde de la columna *Operations*

5.2.2. Resultados de la evaluación práctica

Los resultados de esta segunda encuesta se adjuntan a continuación, agrupando todas las respuestas obtenidas por los usuarios en cada una de las preguntas:

- **Q1 - Tengo experiencia en el uso de herramientas de integración continua.**

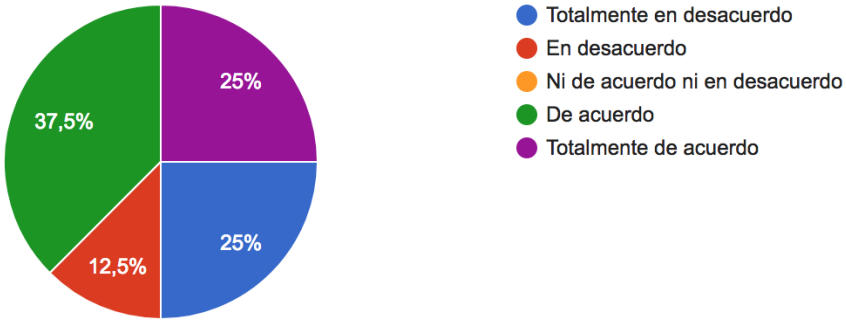

Figura 5.6: Resumen de las respuestas obtenidas a la primera pregunta de la encuesta

- **Q2- ¿Qué herramientas de integración continua has utilizado?**
  - Solamente usé *Jenkins* en una empresa hace tiempo que no se dedicaba al desarrollo de videojuegos.
  - Nunca he utilizado una herramienta así.
  - No conocía este mundo, y ahora que me lo han enseñado, me resulta bastante interesante.
  - He usado *Circle* y actualmente uso *TFS*.
  - Uso principalmente *Jenkins* y *Travis*.
  - No he usado ninguna herramienta.
  - *Jenkins*.
  - *Jenkins CI*.
- **Q3 - Considero que la integración continua es importante en el lugar donde trabajo.**

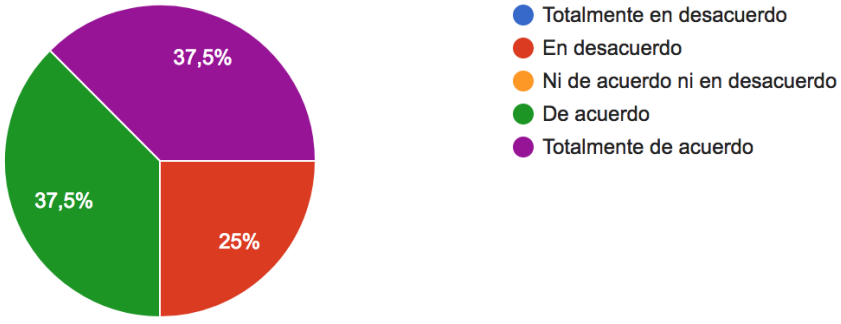

Figura 5.7: Resumen de las respuestas obtenidas a la tercera pregunta de la encuesta

- **Q4 - Considero que la integración continua debería ser más importante en el lugar donde trabajo.**

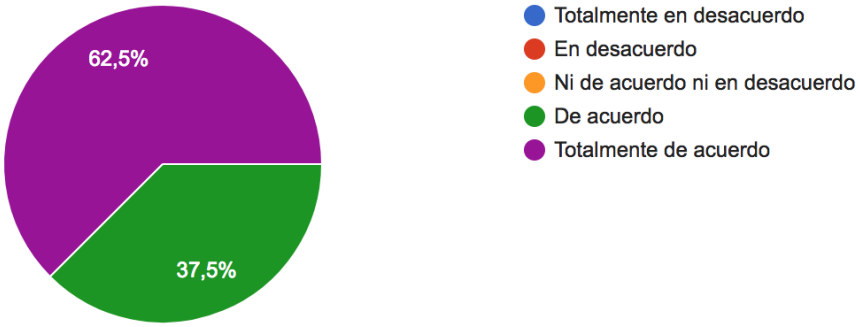

Figura 5.8: Resumen de las respuestas obtenidas a la cuarta pregunta de la encuesta

- **Q5 - Considero que la integración continua es muy importante para mi.**

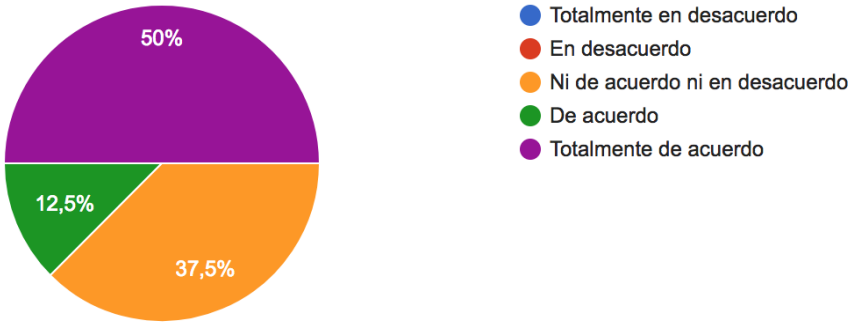

Figura 5.9: Resumen de las respuestas obtenidas a la quinta pregunta de la encuesta

- **Q6 - ¿Qué tipo de facilidades y funcionalidades espero a la hora de utilizar una plataforma de integración continua como *GameCraft*?**

  - Que sea fácil de usar, ya que *Jenkins* aunque era muy potente, al principio asusta por la cantidad de opciones. También me gustaría ver una integración con *Slack*, que es lo que utilizamos en la empresa para comunicaciones internas.
  - Que pueda compilar bien mis proyectos y que no sea muy difícil de configurar.

- Que no de muchos dolores de cabeza a la hora de utilizarlo en proyectos grandes.
- Que no requiera crear un archivo de texto para programar el pipeline como ocurre en otro software.
- Que sea flexible y pueda adaptarse a cualquier proyecto, sea pequeño o grande.
- Que sea fácil de configurar y usar para alguien con poca o 0 experiencia.
- Que sea fácil de usar y no de muchos dolores de cabeza.
- Me gustaría ver una interfaz de usuario sencilla, sin una gran variedad de opciones que en muchos casos no llego a utilizar ni la mitad.

- **Q7 - Considero que *GameCraft* ha cumplido mis expectativas anteriores.**

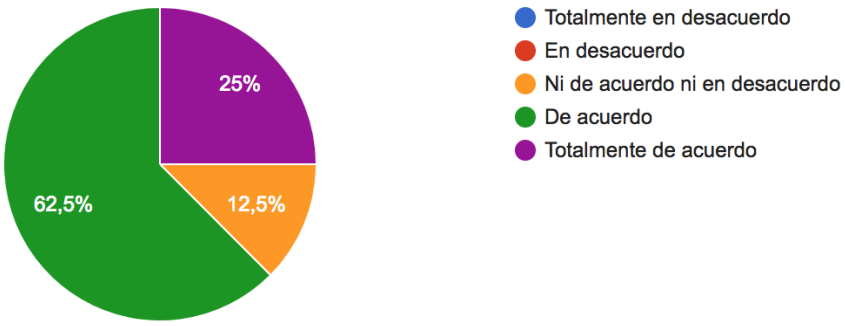

Figura 5.10: Resumen de las respuestas obtenidas a la séptima pregunta de la encuesta

- **Q8 - De forma más detallada, ¿cuáles han sido las razones principales por las cuales considero que *GameCraft* ha cumplido o no mis expectativas anteriores?**

  - Me ha sido muy fácil de configurar y utilizar, aunque echaba de menos que se pudiera cambiar de idioma (solo he podido hacer que se muestre en inglés y no en español). Creo que aunque le faltan muchas funcionalidades para competir de tu a tu con *Jenkins* y otras herramientas que llevan mucho tiempo, va por el buen camino.
  - Se nota que es una plataforma hecha al uso para desarrolladores de videojuegos y no da mucha libertad al usuario de poder configurar cosas de la plataforma. Quizás, esto es una ventaja, ya que ayuda a reducir la cantidad de opciones que yo veo en la pantalla y me hace más fácil usar una herramienta de este tipo, sin haber tenido experiencia en usar otras.
  - Aunque hemos podido probar *GameCraft* en proyectos no muy grandes, echo en falta ver como se desenvuelve la plataforma en proyectos más

grandes. También echo en falta mayor libertad a la hora de configurar varios parámetros en los *pipelines*, como si tiene *Jenkins*: posibilidad de eliminar o no el entorno con cada compilación y poder ejecutar un *script bash* antes, durante y después de un pipeline son dos cosas que echo más en falta.

- *GameCraft* es fácil de instalar y configurar, en pocos clics he ejecutado un pipeline y todo fue bien.
- Me ha faltado ver como se comporta ante proyectos grandes en la demo hecha.
- Es muy fácil de usar, solamente tardé 10 minutos en tener todo el entorno funcionando.
- Su sencillez en la configuración y uso frente a *Jenkins* que es más complejo.
- Me ha gustado su interfaz de usuario, se nota el empeño del autor en hacerla amigable y sencilla.

- **Q9 - *GameCraft* es fácil de instalar.**

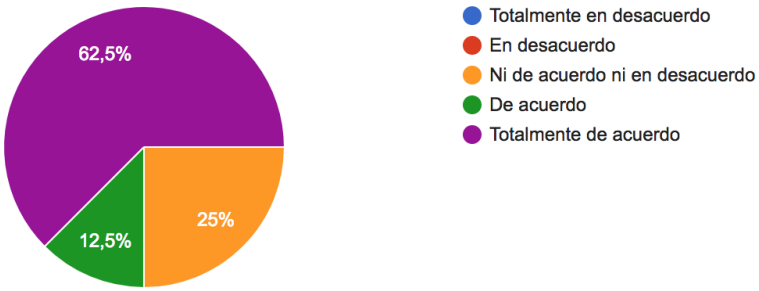

Figura 5.11: Resumen de las respuestas obtenidas a la novena pregunta de la encuesta

- **Q10 - *GameCraft* es fácil de configurar.**

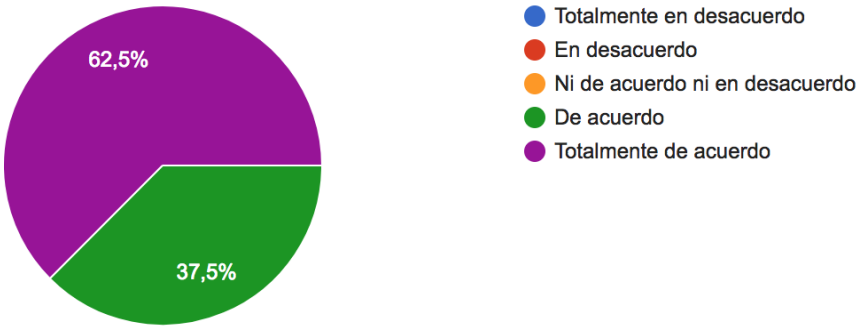

Figura 5.12: Resumen de las respuestas obtenidas a la décima pregunta de la encuesta

- **Q11 - *GameCraft* es fácil de usar.**

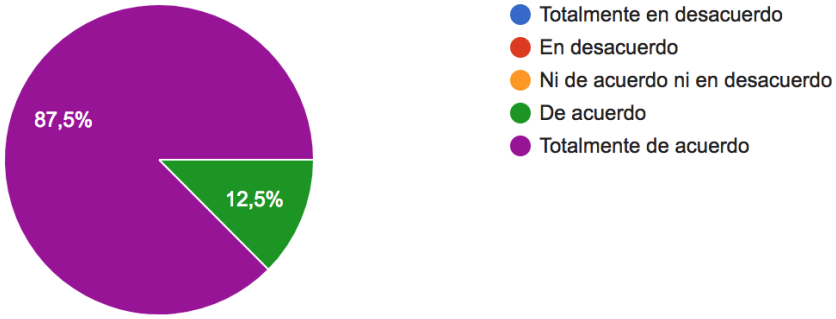

Figura 5.13: Resumen de las respuestas obtenidas a la undécima pregunta de la encuesta

- **Q12 - *GameCraft* reduce costes y optimiza tiempo.**

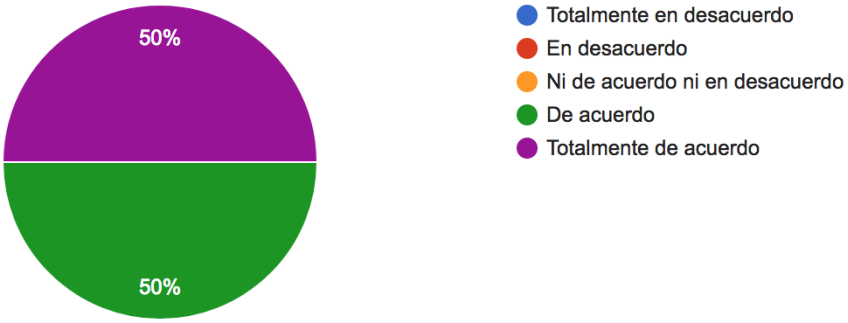

Figura 5.14: Resumen de las respuestas obtenidas a la duodécima pregunta de la encuesta

- **Q13 - *GameCraft* mejora la calidad del videojuego producido.**

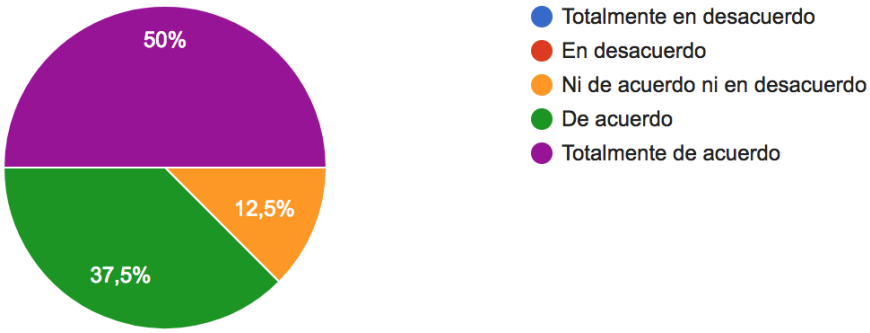

Figura 5.15: Resumen de las respuestas obtenidas a la decimotercera pregunta de la encuesta

- **Q14 - Si pudiera cambiar una cosa de *GameCraft*, ¿qué cambiaría y por qué?**
  - No cambiaría nada.
  - A lo mejor se podría hacer que la empresa pudiera personalizar la plataforma para hacerla más suya, reemplazar el logo de *GameCraft* por el de la empresa, poder añadir logos a los proyectos para reconocer a simple vista los videojuegos que se están compilando, añadir un sistema de ayuda dentro de la plataforma sin necesidad de tener que leer el documento.

- Quizás aunque no se cómo, hacer un asistente visual para poder desplegar la plataforma en un equipo o *cluster* y no recurrir a *Docker* o a un *script* en la Terminal para hacerlo, ya que eso suele alejar a usuarios que no tienen perfil técnico.
- Aunque el uso de *Docker* está bien para desplegar, a lo mejor usar un asistente como si tiene *Jenkins* haría más sencillo la instalación de la plataforma. Por lo demás, lo encuentro todo bien
- No se, quizás añadiría en la página principal visualuzación de gráficas de proyectos totales compilados y fallidos, tiempo en línea... Sería muy interesante poder ampliar las funciones de la plataforma a través de *plug-ins*.
- Añadiría soporte para el idioma español aunque no es muy importante.

- **Q15 - Qué es lo que más me gusta de *GameCraft*?**
  - Su facilidad de uso.
  - Está muy bien orientado para conseguir resultados rápidamente gracias a la facilidad de uso de la interfaz.
  - Su interfaz de usuario.
  - Tiene una interfaz de usuario bastante sencilla que puede ser interesante para aquellos que no quieren trastear mucho y quieren algo que se pueda poner en marcha en un par de minutos.
  - Que no es necesario perder mucho tiempo en configurarlo para compilar proyectos.
  - Su interfaz de usuario y sencillez.
  - Poder integrarlo con mis proyectos de forma sencilla y usar los notificadores para avisarme del estado de los *pipelines*.
- **Q16 - Me gustaría recomendar *GameCraft* a otras empresas y organizaciones.**

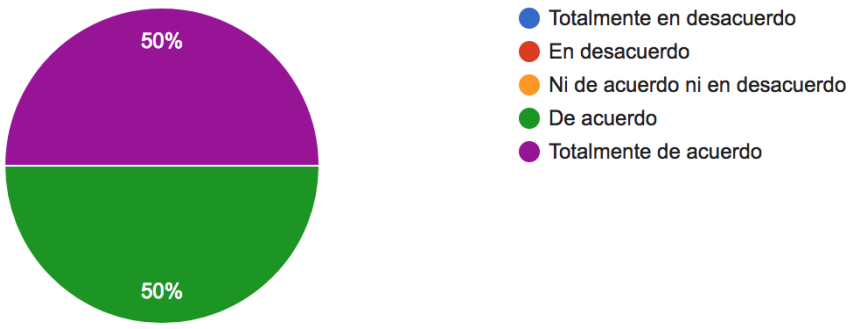

Figura 5.16: Resumen de las respuestas obtenidas a la decimosexta pregunta de la encuesta

- **Q17 - ¿Algún comentario, pregunta o duda?**
  - Soy un desarrollador independiente, y aunque creo que esta herramienta sería más útil en equipos de mediano y gran tamaño, trataré de utilizarlo más a menudo.
  - El proyecto es interesante y además es gratuito, mola :)

El objetivo de este capítulo es describir la evaluación realizada, tanto en el plano teórico como con usuarios finales que abordan escenarios específicos de este proyecto. Se han detallado los criterios y mecanismos de evaluación, esencialmente cuestionarios que combinan preguntas cerradas y abiertas, para verificar si los requisitos del proyecto y las necesidades de los usuarios son cumplidos por los resultados.

Del análisis de las respuesta a la encuesta realizada se pueden extraer varias ideas:

- Los usuarios consideraron que el uso de *GameCraft* puede ser muy importante ya que ayudaría a producir un videojuego con mayor calidad y ayuda además a reducir tiempos y coste.
- La mayoría de usuarios alabaron la facilidad de configuración y uso de la plataforma frente a otras características que ofrece. Este resultado coincide con el *feedback* obtenido en la séptima pregunta de la encuesta inicial, a pesar de ser usuarios diferentes los que probaron *GameCraft*.
- La mayoría de usuarios consideran que *GameCraft* ha cumplido sus expectativas, pero todavía hay margen para mejorar. Entre las características demandadas, se puede destacar facilitar el despliegue a través de un asistente, mayor personalización de la plataforma en lo referente a la interfaz desde el panel de administración sin ser necesario modificar el código fuente y presencia de gráficas para ver de un vistazo el estado de ejecución de todos los *pipelines*.
- Como conclusión final y desde la experiencia de lo que ocurrió en los campos de prueba, se puede afirmar que la realimentación general fue positiva, las funcionalidades principales fueron bien recibidas por los usuarios, y los ejercicios realizados y su utilidad resultó adecuada.

5.3. Artículo de investigación

Para documentar parte de los resultado de este proyecto, se ha realizado un artículo científico que ha sido aceptado en la 5ª edición del Congreso de la Sociedad Española para las Ciencias del Videojuego (COSECiVi 2018). El artículo lleva por título *Una propuesta de integración continua especializada para desarrollo de videojuegos* (Mateu y Peinado (2018)) y este es su resumen:

Los estudios de desarrollo de videojuegos invierten mucho esfuerzo en mejorar la calidad de sus productos, beneficiándose significativamente de toda metodología y herramienta que les permita garantizar la integridad de sus procesos de trabajo con el menor coste posible. Desde hace anos la industria del software cuenta con paradigmas y tecnologıas que facilitan el trabajo a sus equipos de desarrollo, automatizando la mayorıa de las tareas que anteriormente eran tediosas y originaban retrasos y errores. Este es el caso de la integración continua, una practica de desarrollo en la que todos los miembros del equipo integran sus contribuciones como mínimo una vez al día, siendo cada integración verificada y probada exhaustivamente de forma automática para detectar lo antes posible cualquier conflicto. Tras detectar que esta práctica no ha calado lo suficiente entre los desarrolladores independientes de videojuegos, se propone cambiar la situación mediante el proyecto de construcción de una plataforma de integración continua especializada para el desarrollo, el testeo y la publicación de videojuegos. Como resultado se obtiene la primera versión funcional de una herramienta basada en microservicios web que se presenta como libre, gratuita y de fácil manejo para los desarrolladores.

Capítulo 6

Conclusiones

Tened el valor de seguir vuestro corazón e intuición, porque de alguna manera ya sabéis lo que realmente queréis llegar a ser. Todo lo demás es secundario.

Steve Jobs

Resumen: En este capítulo se extraen las conclusiones del proyecto. Además se plantea el posible trabajo futuro que la comunidad de código abierto puede continuar realizando sobre esta.

En este proyecto se ha planteado la necesidad de mejorar el proceso de desarrollo de un videojuego mediante el desarrollo de una plataforma de integración, entrega y despliegue continuo que contribuya a la mejora de la calidad del producto desarrollado. Uno de los principales problemas que existen en muchas empresas de desarrollo es que se llevan a cabo tareas demasiado repetitivas y que consumen mucho tiempo al desarrollador, tiempo que podría estar invirtiendo para solucionar fallos de programación y depurar el código. Con el desarrollo de este proyecto, se ha intentado paliar este problema al proporcionar una herramienta que se encargue de automatizar estas tareas.

Podemos decir que el objetivo principal de este proyecto se ha alcanzado:

- **(Obj1)**. Crear una infraestructura para la integración continua enfocada al desarrollo de videojuegos, fácil de instalar, configurar y usar por estos equipos, y con posibilidad en un futuro de incorporar funcionalidades específicas para realizar pruebas interactivas.

Las actividades se recuerdan ahora y a continuación y se matiza el nivel de consecución de cada una de ellas:

- **(Act1)**. Estudiar y analizar el paradigma de la integración continua, sus herramientas y, en general, su relevancia en el proceso de desarrollo, testeo y publicación de aplicaciones.
- **(Act2)**. Identificar el papel de la integración continua dentro de los equipos de desarrollo de un estudio de videojuegos.
- **(Act3)**. Averiguar las características principales que debe tener una plataforma de integración, entrega y despliegue continuo para ser útil de manera específica en el ámbito de los videojuegos.
- **(Act4)**. Desarrollar la plataforma mencionada como sistema escalable y de alta disponibilidad, basado en microservicios para que se adapte fácilmente a cualquier carga de trabajo.

Durante el trabajo ciertamente se realizó un estudio de la técnica para determinar las principales características que otras herramientas similares ofrecían a los usuarios. Posteriormente, se llevó a cabo un análisis de aquellas funcionalidades que no estaban presentes y que se podrían implementar en este proyecto. Se identificó que los desafíos a los que había que enfrentarse eran el ofrecer una escalabilidad viable que pudiera adaptarse a la carga de cualquier trabajo, una facilidad de puesta en marcha y configuración de la plataforma y una eficiencia económica, tanto a la hora de invertir tiempo en configurar la plataforma, como en el propio hecho de poder adquirir y usar la plataforma.

Después de haber acreditado las características del producto y de haber llevado a cabo un análisis del proceso de desarrollo de software y clarificar de esta forma cuales serían los beneficios de implementar una herramienta de este tipo en las empresas, se envió una encuesta a varias empresas y organizaciones para obtener realimentación sobre cómo trabajaban y valorar el efecto que una herramienta de este tipo provocaría en la organización.

Como parte de los resultados se observó que aunque es posible que algunas empresas no quieran usar esta herramienta, bien porque ya estén usando desarrollos propios o no estén dispuestos a invertir el esfuerzo necesario para aprender una nueva herramienta, muchas otras empresas, organizaciones y desarrolladores independientes serán más abiertas para probar e integrar *GameCraft* en sus procesos de desarrollo. Se ha comprobado en los últimos años que aplicar una herramienta de integración continua al desarrollo de un videojuego es viable desde el punto de vista económico, ya que se está reduciendo el número de tareas repetitivas que una o varias personas tendrían que estar ejecutando, escalable, ya que es capaz de adaptarse a proyectos de cualquier tamaño y tolerante a fallos, ya que la arquitectura de microservicios permite la redundancia de la plataforma en varias instancias de un mismo servicio, por lo que si falla una instancia está disponible otra igual. Al no tener un proyecto grande, no se ha podido probar el comportamiento

del servidor y de la plataforma y tomar medidas de latencia y rendimiento, algo que más adelante se debería llevar a cabo.

En definitiva, este proyecto pone de manifiesto que la *integración continua* es una alternativa viable para contribuir a la mejora de la calidad y a la reducción de costes en la producción de software, y que a pesar de las ventajas que ofrece, existen muchas organizaciones y empresas que por su reducido tamaño o su poco conocimiento en este área no aplican. Con la herramienta *GameCraft* se pone a disposición de todos una plataforma de gratuita, con una interfaz de usuario sencilla y adaptada a los procesos de desarrollo de videojuegos, escalable a cualquier tamaño de un proyecto y de código abierto, por lo que puede adaptarse a las necesidades de cualquier desarrollador.

6.1. Trabajo futuro

Las posibilidades para continuar mejorando este proyecto son variadas. A continuación, se muestran algunas ideas que podrían servir para continuar con la trayectoria iniciada en este trabajo.

En primer lugar, una de las posibilidades de continuación es la de añadir soporte para que terceros puedan ampliar la funcionalidad de esta plataforma. Es decir, desarrollar un sistema de *plugins* que pudiera extender la funcionalidad de la herramienta y poder así ser más versátil y adaptable a cualquier entorno con otro tipo de necesidades que no han podido ser cubiertas inicialmente en este proyecto.

En segundo lugar, aunque la interfaz de usuario sea compatible con dispositivos móviles, estaría bastante bien crear aplicaciones nativas para *smartphones* y *tablets*. Sería muy cómodo que desde el propio dispositivo que se esté testeando el juego se pudiera orquestar una compilación a través de *GameCraft* y recibir la nueva versión del juego en el dispositivo sin tener que pasar por tener que administrar la plataforma desde un ordenador para poder llevar a cabo la compilación.

Por último, otra de las posibilidades sería incorporar nuevas formas para analizar la calidad de código. Por ejemplo, se podría utilizar la inteligencia artificial, y más en concreto el aprendizaje automático, para poder detectar errores de programación con precisión. *Commit Assistant*[1], que es una herramienta desarrollada por *Ubisoft*, permite detectar los errores de programación antes de que el juego llegue a los consumidores, contribuyendo así a reducir los tiempos de desarrollo y conseguir sacar al mercado juegos más pulidos.

Con esta clase de avances se conseguiría una mayor aceptación por parte de los usuarios a esta plataforma, pues verían a la herramienta como una

[1] https://www.youtube.com/watch?v=I5C4FUvDyCc

tecnología indispensable para poder mejorar enormemente la calidad de sus proyectos sin invertir mucho tiempo ni esfuerzo.

Bibliografía

Y así, del mucho leer y del poco dormir,
se le secó el celebro de manera que vino
a perder el juicio.

Miguel de Cervantes Saavedra

David Olson. GNU General Public License — BAwiki, a Reference and Blog for Business Analysts. 2013. Online [`http://www.bawiki.com/wiki/techniques/feature-tree/`]; accedido el 12 de Octubre de 2017.

Duvall, P., Matyas, S. M. y Glover, A. *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley Signature Series. Addison-Wesley, Upper Saddle River, NJ, 2007. ISBN 978-0-321-33638-5.

Gallagher, S. *Mastering Docker*. Packt Publishing, 2015. ISBN 1785287036, 9781785287039.

Gheorghe, R., Hinman, M. L. y Russo, R. *Elasticsearch in Action*. Manning Publications Co., Greenwich, CT, USA, 1st edición, 2015. ISBN 1617291625, 9781617291623.

Humble, J. y Farley, D. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 1st edición, 2010. ISBN 0321601912, 9780321601919.

Mateu, I. M. y Peinado, F. Una propuesta de integración continua especializada para desarrollo de videojuegos. *V Congress of the Spanish Society for Video Game Science*, 2018.

Pascarella, L., Palomba, F., Di Penta, M. y Bacchelli, A. How is video game development different from software development in open source? En *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, páginas 392–402. ACM, New York, NY, USA, 2018. ISBN 978-1-4503-5716-6.

Raible, M. *The JHipster Mini-Book*. Lulu.com, 2016. ISBN 132963814X, 9781329638143.

Scacchi, W. Free and open source development practices in the game community. *IEEE Softw.*, vol. 21(1), páginas 59–66, 2004. ISSN 0740-7459.

Shahin, M., Babar, M. A. y Zhu, L. Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, vol. 5, páginas 3909–3943, 2017. ISSN 2169-3536.

Stacey, P. y Nandhakumar, J. Managing the development process in a games factory : A temporal perspective. 2004.

Walls, C. *Spring Boot in Action*. Manning Publications Co., Greenwich, CT, USA, 1st edición, 2016. ISBN 1617292540, 9781617292545.

Apéndice A

Introduction

In the life cycle of a software project, the product goes through different processes such as analysis, design, development, testing and publication. The common thing when working on a project is to perform thousands of different tests on multiple devices with different versions of the same video game and publish it in various application markets and testing platforms for end users. Manually performing all these tasks for a team would be really tedoius and counterproductive to the company as it involves a high economic and opportunity cost due to the time it takes to employ.

For these reasons, companies need to automate the processes followed to improve the final quality of the product by reducing costs and increasing the quality of the softwae produced. From this arises the concept of *continuous integration, delivery and deployment*. The idea is that members of a team, by uploading the changes they make in the project to a central repository, automatically run a series of tasks and tests in order to achieve the following objectives:

- Find and fix bugs faster.
- Improve the quality of the produced video game.
- Reduce the tame it takes to validate and publish new versions.

The difficulty in installing and configuring a continuous integration, delivery and deployment environment and its integration with the workflows in the development teams are two problems that should be resolved in this Master's thesis.

To solve this problem, it is proposed to design a free and open source software platform capable of abstracting the user from the configuration of a continuous integration, deilvery and deployment environment in video games'development and thus to improve the quality of the software produced. The platform will be developed using the *Java* programming language

and the use of the *Spring Boot* framework to provide a microservice-based architecture, thus ensuring that the platform can scale horizontally easily according to the workload.

A.1. Objectives

The main objective of this project is:

- **(Obj1)**. Create an infrastructure for continuous integration focused on videogame development, easy to install, easy to configure and easy to use, and with the possibility of incorporating specific functionalities to perform interactive tests.

To achieve this main objective **Obj1**, the following activities should be done:

- **(Act1)**. Study and analyze the paradigm of continuous integration, its tools and, in general, its relevance in the process of development, testing and publication of applications.
- **(Act2)**. Identify the role of continuous integration within the development teams of a video game studio.
- **(Act3)**. Find out the main characteristics that a continuous integration platform should have to be specifically useful in the field of videogames.
- **(Act4)**. Develop the mentioned platform as a scalable and highly available system, based on microservices so that it adapts easily to any workload.

Below is a detailed description of each of these objectives. The main objective to be achieved with this project is to facilitate the task of continuous integration within development teams in a video game company. Systems of this type that currently exist are either free and difficult to manage and maintain (*Jenkins*, *Hudson*, ...) or are paid like *Unity Cloud Build*, whose licenses are so costly that they are inasumable by many video games companies. Thanks to the use of the free open source platform that is going to be developed, it is expected to increase productivity and reduce costs in companies. In addition, the project will follow a microservice architecture, so it will be easily scalable horizontally and, therefore, will have the facililty to adapt to any workload. Finally, the user interface will be designed in such a way that its use is as simple as possible for those who manage the continuous integration environment. Unlike other free systems like *Jenkins* where the learning curve is higher because of the large number of options that exist in the platform, and being designed specifically for video games, we will try to reduce the number of functions and options to the most strictly necessary for

the development and publication of video games with the aim of not taking much time or difficulty to configure and maintain the environment.

The achievement of the previous objective will materialize in a new software product that will allow developers to compile, test, analyze the quality of the code and publish their video games in a simple and fast way. The software will have an interface similar to that offered by other continuous integration platforms that are more popular like *Unity Cloud Build*. The new product will be able to run on the most used operating systems (*GNU/-Linux*, *macOS* and *Windows*), establishing affordable and well determined computing resources.

A.2. Project scope

For the reader, to get an idea of the scope of the project, can see below the feature tree of *GameCraft*, the continuous integration platform resulting from this work.

The feature tree, introduced for the first time in 1990 (David Olson (2013)), is a model that organizes the main characteristics of a system into groups of characteristics that capture the scope of the project. Thus, at a glance you can get information on what a system will do at a high level. This type of models can contain functional features (aspects related to the hardware and software of a system), non-functional features (aspects related to performance, internationalization or system maintenance) or parameters such as the cost of developing the system or the different levels of detail of a characteristic. The feature tree may be similar to the functional decomposition diagrams, depending on the methodology that was followed to create them.

It is also necessary to review the current limitations of scope that this project has and that should be corrected in future versions that are being made. It should be noted that this project is a proof of concept, that is, a proposal of how the architecture based on microservices has taken to solve an obvious problem in the development of videogames. Due to the technical complexity of the use of several technologies in the project, it is necessary to have at least one or two people with high knowledge in computer science and in software architecture design to maintain the platform or understand because microservices fail and try to make them work as soon as possible. In addition, the platform is capable of compiling and executing automated code tests and quality evaluation of it, but it is not yet possible to execute *playtestings* automatically, nor is it able to adequately evaluate whether the *multimedia* assets are incorporated into the video game (such as 3D models or audio files) consume too many resources that could slow down the overall performance of the game, for example. Also, the platform does not support repositories other than *Github*, so videogame projects that use *SVN* or *Bit-*

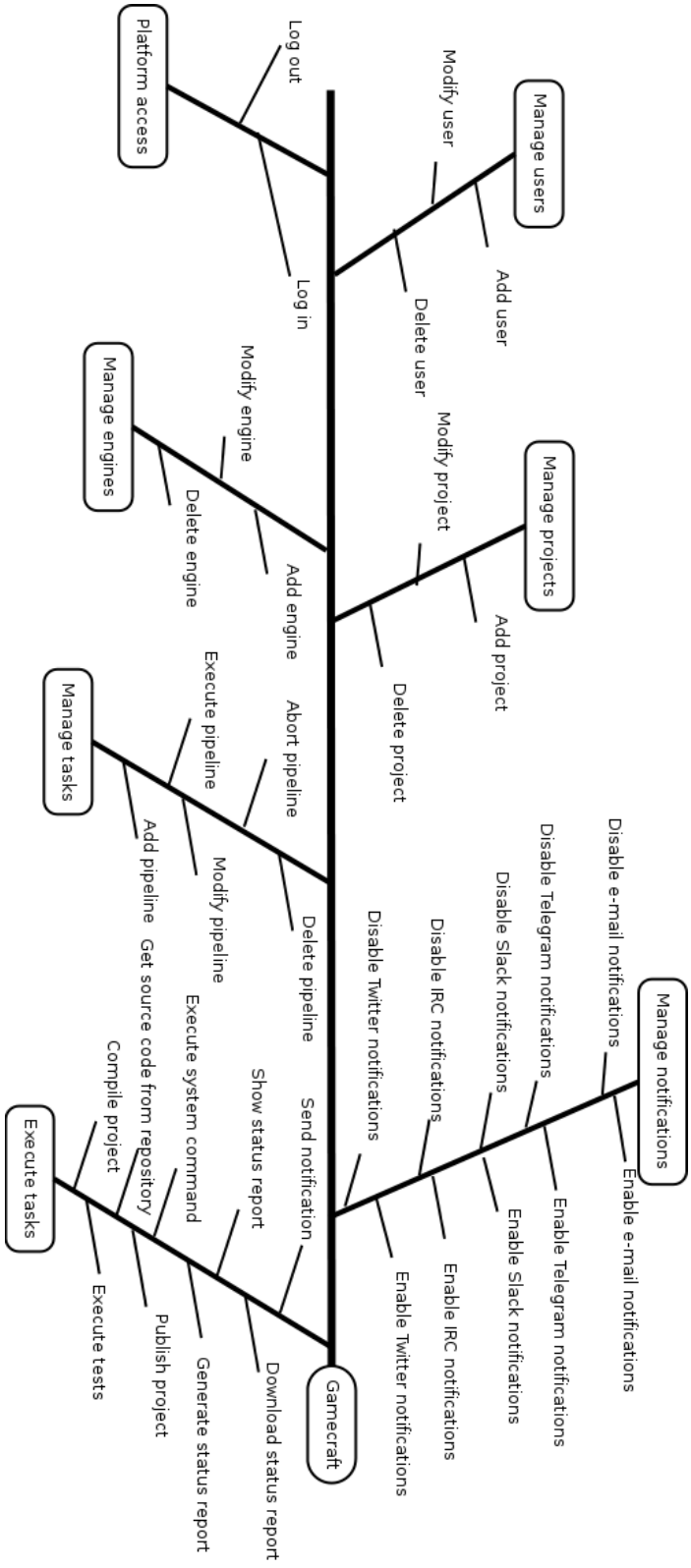

Figura A.1: Feature tree of GameCraft

Bucket as repositories, will not be able to use the platform. Also, it is not possible to carry out automated publications on application platforms such as *App Store* or *Google Play* and testing platforms for end users.

The following will briefly explain the characteristics collected in the feature tree shown in the Figure A.1:

- **Manage users**: This feature refers to manipulation operations on users who will access the platform.
  - *Add user*: Create a new user in the platform with the necessary data so that a person can access it.
  - *Modify user*: Modify an existing user in the platform with the new data.
  - *Delete user*: Remove an existing user in the platform so that the person cannot access it anymore.
- **Manage projects**: This feature refers to manipulation operations on the projects from which the stages of the *pipeline*s will be executed:
  - *Add project*: Create a new project in the platform with the necessary data so that the *pipeline*s can be executed on them.
  - *Modify project*: Modify an existing project in the platform with the new data.
  - *Delete project*: Remove an existing project in the platform.
- **Manage notifications**: This feature refers to the activation and deactivation operations of notifications that will be sent via different social platforms whenever an event occurs within the platform:
  - *Enable e-mail notifications*: Activates notifications by e-mail, so that when an event occurs on the platform, an e-mail will be sent to the provided e-mail address.
  - *Disable e-mail notifications*: Disable e-mail notifications, so that when an event occurs on the platform, an e-mail will not be sent to the provided e-mail address.
  - *Enable Telegram notifications*: Activates notifications by Telegram, so that when an event occurs on the platform, a message will be sent to the provided Telegram account.
  - *Disable Telegram notifications*: Disable Telegram notifications, so that when an event occurs on the platform, a message will not be sent to the provided Telegram account.
  - *Enable Slack notifications*: Activates notifications by Slack, so that when an event occurs on the platform, a message will be sent to the provided Slack channel.

- *Disable Slack notifications*: Disable Slack notifications, so that when an event occurs on the platform, a message will not be sent to the provided Slack channel.
- *Enable IRC notifications*: Activates notifications by IRC, so that when an event occurs on the platform, a message will be sent to the provided IRC channel.
- *Disable IRC notifications*: Disable IRC notifications, so that when an event occurs on the platform, a message will not be sent to the provided IRC channel.
- *Enable Twitter notifications*: Activates notifications by Twitter, so that when an event occurs on the platform, a message will be sent using the provided Twitter account.
- *Disable Twitter notifications*: Disable Twitter notifications, so that when an event occurs on the platform, a message will not be sent using the provided Twitter account.

- **Platform access**: This feature refers to the access by the users to the platform:
  - *Log in*: Access the platform using an existing user account
  - *Log out*: Log out on the platform.
- **Manage engines**: This feature refers to manipulation operations on engines to be executed by the platform:
  - *Add engine*: Create a new reference to an engine installed in the machine so the platform can use it to compile the projects.
  - *Modify engine*: Modify an existing reference to an engine installed in the machine with the new data.
  - *Delete engine*: Remove an existing reference to an engine installed in the machine.
- **Manage tasks**: This feature refers to manipulation operations on the *pipeline*s that are going to be executed in the platform:
  - *Add pipeline*: Create a new *pipeline* with the different steps or tasks that have to be executed sequentially.
  - *Modify pipeline*: Modify the steps or tasks of a *pipeline* already created in the platform
  - *Delete pipeline*: Delete an existing *pipeline* in the platform
  - *Execute pipeline*: Execute the steps or tasks defined in the *pipeline* in the exact order in which they were added within the *pipeline*.

  - *Abort pipeline*: Interrupt the execution of the steps or tasks of a *pipeline* while it is being launched. All steps or tasks that have already been executed will be reverted.

- **Execute tasks**: This feature refers to each of the steps that can be executed within a *pipeline* sequentially:

  - *Get source code from repository*: Download the source code of a project stored in a repository. The download can be done manually or when a change is detected.
  - *Compile project*: Compile the source code of the project using an engine previously configured in the platform.
  - *Execute tests*: Executes the tests that the platform has detected in the source code and verifies if they are correct or not.
  - *Publish project*: Upload the binaries that have been generated as a result of the compilation to an storage platform.
  - *Execute system command*: Execute a system command. The execution is done natively to the operating system that is installed, so the amount of commands available and their result may vary depending on whether you use Windows, MacOS or GNU / Linux.
  - *Generate status report*: Generate a report that contains information about the execution status of a *pipeline*.
  - *Show status report*: Show a status report that has been generated previously.
  - *Download status report*: Download a status report that has been generated previously in PDF format.
  - *Send notification*: Send a notification using one of the following technologies that have been previously configured: e-mail, Telegram, Slack, IRC, Twitter...

A.3. Description of the product to be delivered

As deliverables resulting from this work, a CD-ROM with a digital copy of this thesis is made available. In addition, the appendices provide an installation manual that will help the user in the process of deploying the platform on a private server and a user manual that provides a description of the main features and functionalities offered by the platform.

The source code of this project can be found at the following repository: `https://github.com/iMartinezMateu/gamecraft`.

A.4. Document organization

In this section, the structure of the document will be described. This document is divided into the following chapters:

- **Chapter 1. Introduction**. This chapter presents the concepts of continuous integration and explains the motivation of the project. In addition, the objectives to be addressed with this work, the expected deliverables and a high-level view of *GameCraft*, the developed software tool, are presented.
- **Chapter 2. State of the technique**. This chapter explains the life cycle of the software and why it is important to integrate, deliver and deploy continuously both in a generic software project and in the development of a videogame. In addition, the state of the art in continuous integration is analyzed, after obtaining information from tools similar to *GameCraft* that exist in the market and see what are the characteristics of each one. Finally, a comparison between all analyzed tools and *GameCraft* is made.
- **Chapter 3. Methodology and project management**. This chapter deals in detail with the methodologies and technologies used in the development and presents a temporal planning of the entire project in the form of a Gantt chart.
- **Chapter 4. System development**. This chapter contains the bulk of the contribution of this work, exposing everything related to the different phases of the software development process, from the initial idea to the implementation, through the analysis of requirements, architecture and design of the platform.
- **Chapter 5. Tests and results**. The chapter documents the tests performed to evaluate the performance of the platform and also the queries and results obtained after evaluating it and obtaining feedback with users.
- **Chapter 6. Conclusions**. In this chapter the conclusions of the project are extracted. In addition, there is a possible future work that the open source community can continue to do on it.

In addition, the thesis has a series of appendices for the English translations of the Introduction and Conclusions chapters, as well as a glossary with the terminology used, manuals and other documentation.

Apéndice B

Conclusions

This project has raised the need to improve the process of developing a video game through the development of a tool for integration, delivery and continuous deployment that will contribute to the improvement of the quality of the product developed. One of the main problems that exist in many development companies is that repetitive tasks takes too much time to complete them, and this time can be spent to solve programming flaws and debug the code. With the development of this project, this problem has been attempted to be solved by providing a tool that is responsible for automating these tasks.

We can say that the main objective of this project has been reached:

- **(Obj1)**. Create an infrastructure for continuous integration focused on videogame development, easy to install, easy to configure and easy to use, and with the possibility of incorporating specific functionalities to perform interactive tests.

The activities are remembered now and the level of achievement of each one is qualified below:

- **(Act1)**. Study and analyze the paradigm of continuous integration, its tools and, in general, its relevance in the process of development, testing and publication of applications.
- **(Act2)**. Identify the role of continuous integration within the development teams of a video game studio.
- **(Act3)**. Find out the main characteristics that a continuous integration platform should have to be specifically useful in the field of videogames.
- **(Act4)**. Develop the mentioned platform as a scalable and highly available system, based on microservices so that it adapts easily to any workload.

During the work a study of the technique was carried out to determine the main characteristics that other similar tools offered to the users. Subsequently, an analysis of those functionalities that were not present and that could be implemented in this project was carried out. The challenges that had to be faced were to offer a viable scalability that could be adapted to the load of any job, a facility of setting up and configuring the platform and an economic efficiency, both when investing time in configuring the platform, as well as in the acquire and use the platform.

After having accredited the characteristics of the product and having carried out an analysis of the software development process and to clarify in this way what would be the benefits of implementing a tool of this type in the companies, a survey was sent to several companies and organizations to obtain *feedback* of how they worked and assess the effect that a tool of this type would have on the organization.

Although it is possible that some companies do not want to use this tool since they may be using their own developments and would not be willing to invest the effort in learning a new tool, many other companies, organizations and independent developers will show a more open behavior to try integrate *GameCraft* in their development processes. It has been proven that applying a tool of continuous integration to the development of a videogame can be viable from the economic point of view, since it is reducing the number of repetitive tasks that one or several people would have to be executing; scalable, since it is able to adapt to projects of any size and tolerant to failures, because of the architecture of microservices that allows to create several instances of the same service, so if one instance fails, an equal one is available. By not having a large project, it has not been possible to test the behavior of the server and the platform and to take measures of latency and performance and this is something that in the future should be carried out.

In short, this project demonstrates that *continuous integration* is a viable alternative to contribute to the improvement of quality and to the reduction of costs in the production of software, and that in spite of the advantages it offers, there are many organizations and companies that due to their small size or little knowledge in this area do not apply. With *GameCraft*, a free platform is made available to everyone, with a simple user interface adapted to videogame development processes, scalable to any size of a project and open source, so the tool can be adapted to the needs of any developer

B.1. Future work

The possibilities to continue improving this project are varied. Here are some interesting ideas that could continue with the line of this work.

In the first place, one of the possibilities of continuation is to add support so that third parties can expand the functionality of this platform. That

is, develop a system of plug-ins that could extend the functionality of the platform and thus be more versatile and adaptable to any environment with other types of needs that could not be covered initially in this project.

Secondly, although the user interface is compatible with mobile devices, it would be good enough to create native applications for smartphones and tablets. It would be very comfortable that from the device that is being tested the game could be orchestrated a compilation through GameCraft and receive the new version of the game on the device without having to go through having to manage the platform from a computer to be able to carry out the compilation.

Finally, another possibility would be to incorporate new ways to analyze the quality of code. For example, artificial intelligence could be used, and more specifically machine learning, to be able to detect programming errors with precision. Commit Assistant [1], which is a tool developed by Ubisoft, allows detecting programming errors before the game reaches consumers, thus helping to reduce the development times and get to market more polished games.

With these advances, a greater acceptance by users of this platform would be achieved, since they would see the tool as indispensable in order to improve the quality of their projects without investing much time or effort.

[1] `https://www.youtube.com/watch?v=I5C4FUvDyCc`

Apéndice C

Glosario

Término	Definición
Fork	Una **bifuración** (*fork en inglés* es la creación de un proyecto en una dirección alternativa del proyecto principal tomando su código fuente. Como resultado de esta acción se pueden generar proyectos diferentes que cubren necesidades distintas al del proyecto inicial.
Servlet	Un **servlet** es una tecnología que permite crear aplicaciones web dinámicas sobre las clases de Java.
Apache Tomcat	**Apache Tomcat** es un proyecto de código abierto de Apache que implementa un contenedor de servlets capaz de recibir peticiones de páginas web y redireccionar las peticiones a un objeto Servlet.
CVS	**Concurrent Versions System** es un sistema centralizado de control de versiones que mantiene el registro del trabajo y permite que distintos desarrolladores colaboren en un proyecto. **CVS** utiliza una arquitectura cliente-servidor: un servidor guarda cada cambio que se realiza en el proyecto y los clientes se conectan al servidor para sacar una copia completa del proyecto.
Subversion	**Apache Subversion** es un sistema centralizado de control de versiones de código abierto cuyo funcionamiento se asemeja al de un sistema de ficheros. Utiliza el concepto de revisión para guardar los cambios producidos en el repositorio, intentando optimizar en todo caso el uso del espacio en el disco duro y permite que varias personas pueden modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones, fomentando la colaboración e incluso deshacer los cambios realizados.
Git	**Git** es un sistema distribuido de control de versiones diseñado por Linux Torvalds diseñado para mantener la eficiencia y la mantenibilidad de versiones de aplicaciones cuando tienen una gran cantidad de código fuente.
Mercurial	**Mercurial** es un sistema distribuido de control de versiones multiplataforma de código abierto diseñado para ofrecer una gran escalabilidad y rendimiento en la gestión de archivos tanto de texto como binarios y con capacidades avanzadas de ramificación e integración, todo ello manteniendo sencillez conceptual.
Perforce	**Perforce** es un sistema de control de versiones comercial, propietario y centralizado que funciona en modo cliente/servidor. El servidor gestiona una base de datos central que contiene varios repositorios y los clientes se conectan a esas bases de datos para obtener los ficheros del repositorio y enviarlos agrupados en listas de cambios. Ofrece gran robustez en la gestión de archivos tanto de texto como binarios.
Apache Ant	**Apache Ant** es una herramienta de código abierto que permite automatizar tareas mecánicas y repetitivas durante la fase de compilación y construcción de proyectos Java.
Gradle	**Gradle** es una herramienta de automatización para la construcción y compilación de proyectos que hereda características de Apache Ant y Apache Maven. Utiliza un lenguaje sencillo y claro y dispone de una gran flexibilidad que permite trabajar con ella utilizando otros lenguajes y no solo Java. Dispone por otro lado de un sistema de gestión de dependencias potente y estable.

Tabla C.1: Glosario

Término	Definición
Apache Maven	**Apache Maven** es una herramienta de gestión y construcción de proyectos Java similar a Apache Ant pero tiene un modelo de configuración más simple basado en un formato XML. Maven utiliza un Project Object Model (POM) para describir el proyecto de software a construir, dependencias y componentes externos y el orden de construcción de los elementos. Además, tiene la posibilidad de ser usado en red y poder dinámicamente descargar plugins de un repositorio.
Cron	**Cron** es un administrador de procesos en segundo plano y disponible para sistemas Unix que los ejecuta a intervalos regulares (por ejemplo, cada minuto, día o semana). Los procesos que deben ejecutarse y la hora en la que deben hacerlo se especifican en el fichero `crontab`.
Hot swap	**Hot swap** hace referencia a la capacidad para instalar o sustituir un componente hardware o software sin necesidad de detener o alterar la operación normal del programa o de la computadora donde se aloja.
PDF	**PDF** (sigla del inglés *Portable Document Format*, "formato de documento portátil") es un formato de almacenamiento para documentos y/o formularios digitales independiente de plataformas de software o hardware muy extendido en Internet.

Tabla C.2: Glosario

Apéndice D

Características del sistema

A continuación, se proporciona una breve explicación de las características recogidas en la Figura 1.1:

- **Gestionar usuarios**: Esta característica se refiere a las operaciones de manipulación de usuarios que accederán a la plataforma.
  - *Añadir usuario*: Crea un nuevo usuario en la plataforma con los datos necesarios para que una persona pueda acceder a ella.
  - *Modificar usuario*: Modifica un usuario existente en la plataforma con nuevos datos.
  - *Eliminar usuario*: Elimina un usuario existente de la plataforma de tal forma que la persona no pueda acceder más con ella.
- **Gestionar proyectos**: Esta característica se refiere a las operaciones de manipulación en los proyectos sobre los cuales se ejecutarán las etapas de los *pipelines*:
  - *Añadir proyecto*: Crea un nuevo proyecto en la plataforma con los datos necesarios sobre los cuales los *pipelines* puedan ser ejecutados sobre ellos.
  - *Modificar proyecto*: Modifica un proyecto existente en la plataforma con nuevos datos.
  - *Eliminar proyecto*: Elimina un proyecto existente en la plataforma.
- **Gestionar notificaciones**: Esta característica se refiere a las operaciones de activación y desactivación de las notificaciones que se enviarán a través de diferentes plataformas sociales cada vez que ocurra un evento dentro de la plataforma:
  - *Activar notificaciones por e-mail*: Activa las notificaciones por correo electrónico, de tal forma que cuando ocurra un evento en

la plataforma, un correo electrónico será enviado a la dirección de correo proporcionada.

- *Desactivar notificaciones por e-mail*: Desactiva las notificaciones por correo electrónico, de tal forma que cuando ocurra un evento en la plataforma, no se envíe un correo electrónico a la dirección de correo proporcionada.
- *Activar notificaciones por Telegram*: Activa las notificaciones por *Telegram*, de tal forma que cuando ocurra un evento en la plataforma, se enviará un mensaje a la cuenta de *Telegram* proporcionada.
- *Desactivar notificaciones por Telegram*: Desactiva las notificaciones por *Telegram*, de tal forma que cuando ocurra un evento en la plataforma, no se envíe un mensaje a la cuenta de *Telegram* proporcionada.
- *Activar notificaciones por Slack*: Activa las notificaciones por *Slack*, de tal forma que cuando ocurra un evento en la plataforma, se enviará un mensaje al canal de *Slack* proporcionado.
- *Desactivar notificaciones por Slack*: Desactiva las notificaciones por *Slack*, de tal forma que cuando ocurra un evento en la plataforma, no se envíe un mensaje al canal de *Slack* proporcionado.
- *Activar notificaciones por IRC*: Activa las notificaciones por *IRC*, de tal forma que cuando ocurra un evento en la plataforma, se enviará un mensaje al canal de *IRC* proporcionado.
- *Desactivar notificaciones por IRC*: Desactiva las notificaciones por *IRC*, de tal forma que cuando ocurra un evento en la plataforma, no se envíe un mensaje al canal de *IRC* proporcionado.
- *Activar notificaciones por Twitter*: Activa las notificaciones por *Twitter*, de tal forma que cuando ocurra un evento en la plataforma, se enviará un mensaje utilizando la cuenta de *Twitter* proporcionada.
- *Desactivar notificaciones por Twitter*: Desactiva las notificaciones por *Twitter*, de tal forma que cuando ocurra un evento en la plataforma, no se envíe un mensaje utilizando la cuenta de *Twitter* proporcionada.

- **Acceso a la plataforma**: Esta característica se refiere al acceso de los usuarios a la plataforma:

  - *Iniciar sesión*: Accede a la plataforma utilizando una cuenta de usuario existente.
  - *Cerrar sesión*: Cierra la sesión en la plataforma.

- **Gestionar *engines***: Esta característica se refiere a la manipulación de referencias a *engines* instaladas localmente en la máquina para poder usarlas en las tareas de compilación:
  - *Crear engine*: Crea una nueva referencia a un *engine* instalado en la máquina para que la plataforma pueda utilizarla y compilar los proyectos.
  - *Modificar engine*: Modifica una referencia a un *engine* instalado en la máquina.
  - *Eliminar engine*: Elimina una referencia a un *engine* instalado en la máquina para que no se pueda utilizar más.
- **Gestionar tareas**: Esta característica se refiere a las operaciones de manipulación en los *pipelines* que van a ser ejecutados en la plataforma:
  - *Crear pipeline*: Crea un nuevo *pipeline* con los diferentes pasos o tareas que tienen que ejecutarse secuencialmente.
  - *Modificar pipeline*: Modifica los pasos o tareas de un *pipeline* ya creado en la plataforma.
  - *Eliminar pipeline*: Elimina un *pipeline* existente en la plataforma.
  - *Ejecutar pipeline*: Ejecuta cada paso o tarea definida en el *pipeline* en el orden exacto en que fueron añadidos al *pipeline*.
  - *Abortar pipeline*: Interrumpe la ejecución de los pasos o tareas de un *pipeline* mientras está en ejecución. Todos los pasos o tareas que ya han sido ejecutados serán revertidos.
- **Ejecutar tareas**: Esta característica se refiere a cada uno de los pasos o tareas que conforman un *pipeline* y que serán ejecutados secuencialmente:
  - *Obtener código fuente de un repositorio*: Descarga el código fuente del proyecto almacenado en un repositorio. La descarga puede realizarse manualmente o cada vez que se detecta un cambio o *commit*.
  - *Compilar proyecto*: Compilar el código fuente del proyecto utilizando un *engine* configurado previamente en la plataforma.
  - *Ejecutar tests*: Ejecuta los tests que la plataforma ha detectado en el código fuente y detecta si son correctos o no.
  - *Publicar proyecto*: Sube los binarios que han sido generados como resultado de la compilación del proyecto a una plataforma de almacenamiento.

- *Ejecutar comando del sistema*: Ejecuta un comando de sistema. La ejecución se realiza de forma nativa al sistema operativo que hay instalado, por lo que la cantidad de comandos disponibles y su resultado difiere de si se está usando *Windows*, *MacOS* o *GNU/Linux*.
- *Generar informe de estado*: Genera un informe de estado que contiene información acerca del estado de la ejecución de un *pipeline*.
- *Visualizar informe de estado*: Muestra un informe de estado que ya ha sido generado previamente.
- *Descargar informe de estado*: Descarga un informe de estado que ya ha sido generado previamente en formato *PDF*.
- *Enviar notificación*: Envía una notificación utilizando una de las siguientes tecnologías que han sido previamente configuradas: correo electrónico, *Telegram*, *Slack*, *IRC*, *Twitter*...

Apéndice E

Manual de despliegue

En este anexo se suministran los pasos a seguir para desplegar GameCraft.

Antes de empezar con el proceso de instalación, asegúrate de que el sistema cumpla con los siguientes requisitos:

	Requisitos mínimos	Requisitos recomendados
Procesador	Procesador de 32 bits (x86) o 64 bits (x64) con un núcleos a 1.8 gigahercios (GHz) o más.	Procesador de 32 bits (x86) o 64 bits (x64) con dos núcleos a 2.6 gigahercios (GHz) o más.
Memoria RAM	4 gigabyte (GB)	8 gigabytes (GB)
Espacio en disco duro	Espacio disponible en disco rígido de 30 GB	Espacio disponible en disco rígido de 100 GB

Tabla E.1: Requisitos hardware del proyecto.

Para desplegar GameCraft, se puede llevar a cabo de dos formas totalmente diferentes: de forma manual o utilizando Docker.

E.1. Procedimiento manual

Para desplegar la plataforma mediante este procedimiento, se necesita tener instalado previamente `Java 8` y `MySQL`.

Posteriormente a esto, para cada carpeta que conforma cada microservicio que compone la plataforma, existirá un JAR. Desde la Terminal, hay que ejecutar el archivo anterior mediante el comando: `java -jar microservicio.jar`. Repetir para cada microservicio. Se recomienda utilizar el argumento `-Xmx512m` para ajustar la cantidad de memoria que cada microservicio utiliza a 512 MB de RAM.

La plataforma se puede acceder a través del navegador web en la dirección `http://localhost:8000/`.

E.2. Docker

GameCraft se suministra con un fichero de Docker Compose que permite desplegar toda la plataforma a través de un único comando y sin necesidad de instalar ninguna dependencia en el sistema. Para desplegar la plataforma utilizando Docker, es necesario abrir la Terminal, situarse en la carpeta del proyecto y ejecutar el comando: `docker-compose up .`

La plataforma se puede acceder a través del navegador web en la dirección `http://localhost:8000/`.

Apéndice F

Manual de usuario

Una vez desplegada la plataforma, el usuario puede acceder a ella a través de su navegador web en la dirección `http://localhost:8000/`. Aparecerá la pantalla de inicio de sesión F.1. La cuenta de administrador por defecto es la siguiente; el nombre de usuario es *admin* y la contraseña es *admin*. Más adelante, se puede cambiar la contraseña para hacer más dificil la entrada de personas no autorizadas al panel de administración. Si por el contrario se quiere iniciar sesión como usuario, existe una cuenta creada de ejemplo; el nombre de usuario es *user* y la contraseña es *user*.

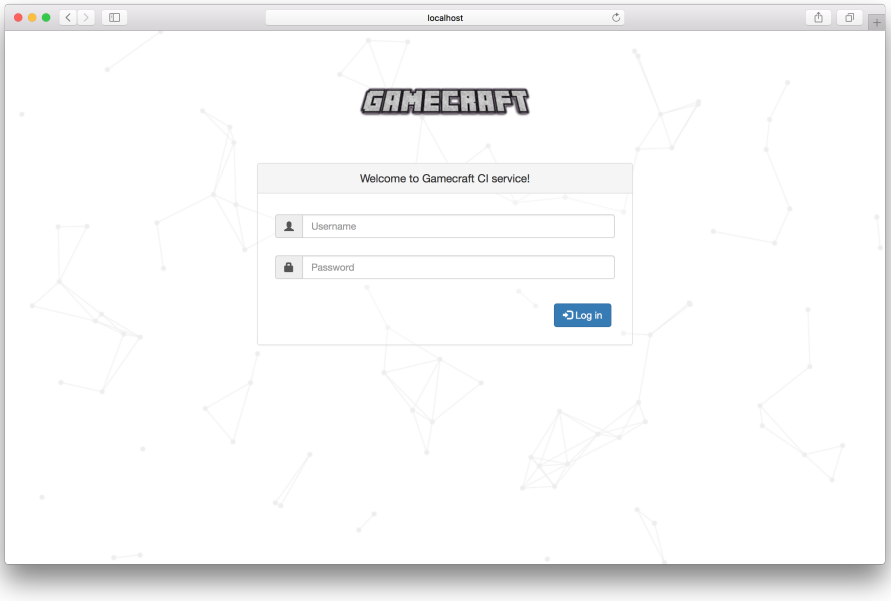

Figura F.1: Pantalla de inicio de sesión

Una vez iniciada sesión, ya sea como administrador o como usuario, esta-

remos ante la pantalla principal de la plataforma, representada en la Figura F.2. Desde aquí, podemos diferenciar la forma de navegación que se irá repetiendo en cada una de las siguientes pantallas que se vea de la plataforma. En la parte superior de la misma, se encuentra el menú de navegación donde se puede acceder al resto de módulos de la plataforma y en la parte central, se encuentra el canvas de visualización, donde se irá mostrando la información relevante de cada uno de los módulos de Gamecraft.

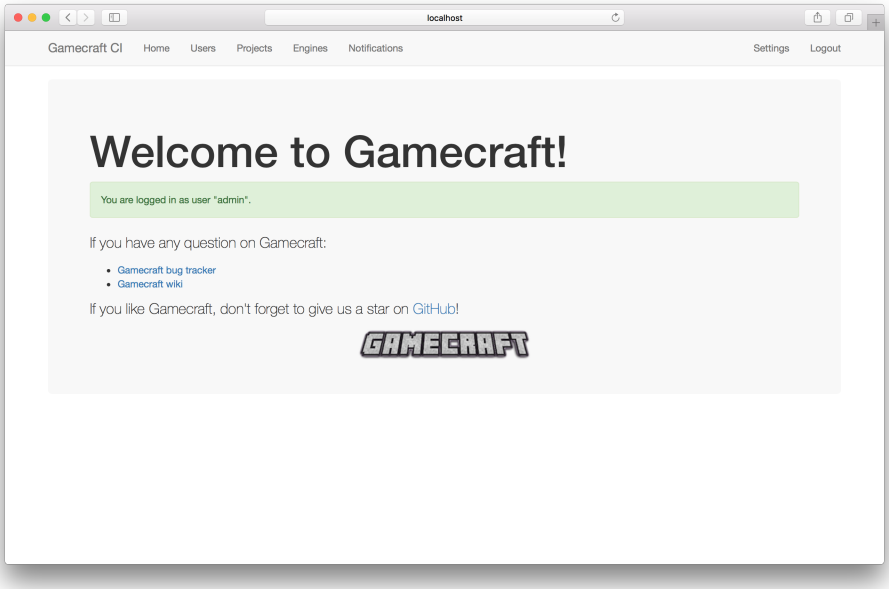

Figura F.2: Pantalla principal de Gamecraft

A continuación, se explica cada una de las opciones del menú superior:

- *Home*. Se muestra la pantalla que se puede ver en la Figura F.2.
- *Users*. Se muestran todos los usuarios creados y activos que pueden acceder a la plataforma. Desde esta pantalla, se pueden crear nuevos usuarios, modificar los ya existentes o eliminarlos.
- *Projects*. Se muestran todos los proyectos creados y activos que están siendo procesados por la plataforma. Desde esta pantalla, se pueden crear nuevos proyectos, modificar los ya existentes o eliminarlos. Además, desde esta pantalla se pueden acceder a los pipelines creados por cada proyecto existente en la plataforma.
- *Engines*. Se muestran todos los motores creados y activos que están siendo utilizados por la plataforma para las tareas de compilación.

Desde esta pantalla, se pueden crear nuevos pipelines, modificar los ya existentes o eliminarlos.

- *Notifications*. Se muestran todos los notificadores creados y activos que están siendo utilizados por la plataforma para comunicar eventos al usuario. Al abrir esta opción en el menú, se mostrará un desplegable con las siguientes opciones:

  - *E-mail*: Se muestran todos los notificadores de correo electrónico creados y activos que están siendo utilizados por la plataforma. Desde esta pantalla, se pueden crear nuevos notificadores, modificar los ya existentes o eliminarlos.
  - *Twitter*: Se muestran todos los notificadores de Twitter creados y activos que están siendo utilizados por la plataforma. Desde esta pantalla, se pueden crear nuevos notificadores, modificar los ya existentes o eliminarlos.
  - *Telegram*: Se muestran todos los notificadores de Telegram creados y activos que están siendo utilizados por la plataforma. Desde esta pantalla, se pueden crear nuevos notificadores, modificar los ya existentes o eliminarlos.
  - *Slack*: Se muestran todos los notificadores de Slack creados y activos que están siendo utilizados por la plataforma. Desde esta pantalla, se pueden crear nuevos notificadores, modificar los ya existentes o eliminarlos.
  - *IRC*: Se muestran todos los notificadores de IRC creados y activos que están siendo utilizados por la plataforma. Desde esta pantalla, se pueden crear nuevos notificadores, modificar los ya existentes o eliminarlos.

- *Settings*. Se muestran las opciones del usuario. Desde esta pantalla se puede cambiar el nombre de usuario, la contraseña y otros datos personales de la cuenta de usuario con la que se ha iniciado sesión.

- *Logout*. Cierra la sesión de la cuenta con la que se ha accedido anteriormente.

F.1. Cambiar la contraseña de usuario

Para cambiar la contraseña con la que se ha iniciado sesión, el usuario tiene que acceder a la pantalla Settings desde la pantalla principal representada en la Figura F.2.

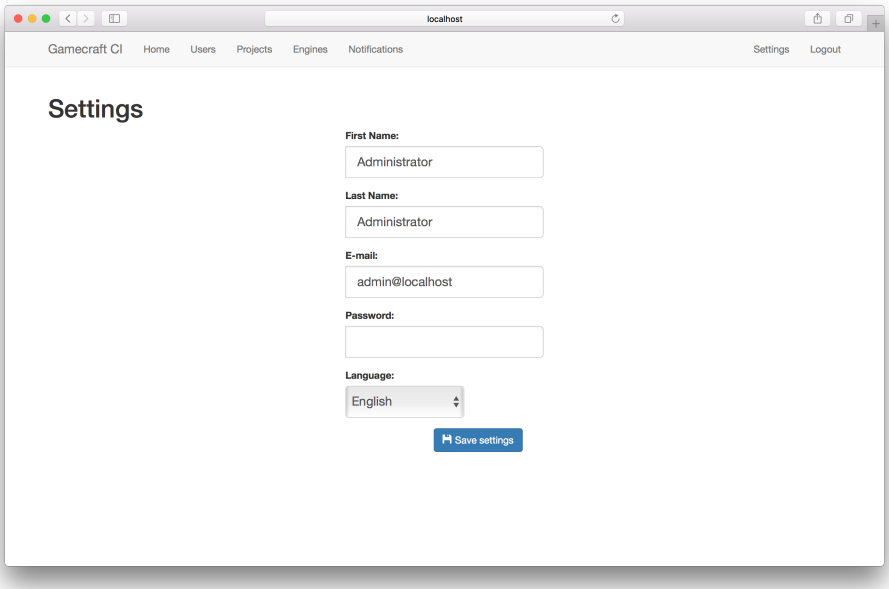

Figura F.3: Pantalla de configuración

Desde esta pantalla representada en la Figura F.3, rellenar el campo de *Password* con la nueva contraseña que se quiera introducir y guardar los cambios haciendo clic en el botón *Save changes*. La contraseña se ha cambiado y la siguiente vez que queramos acceder a Gamecraft, tendremos que introducir esta nueva contraseña en vez de la anterior.

F.2. Manipular usuarios de la plataforma

Para poder crear, modificar y eliminar usuarios que acceden a la plataforma, desde la pantalla inicial representada en la Figura F.2, acceder a la opción *Users* del menú principal.

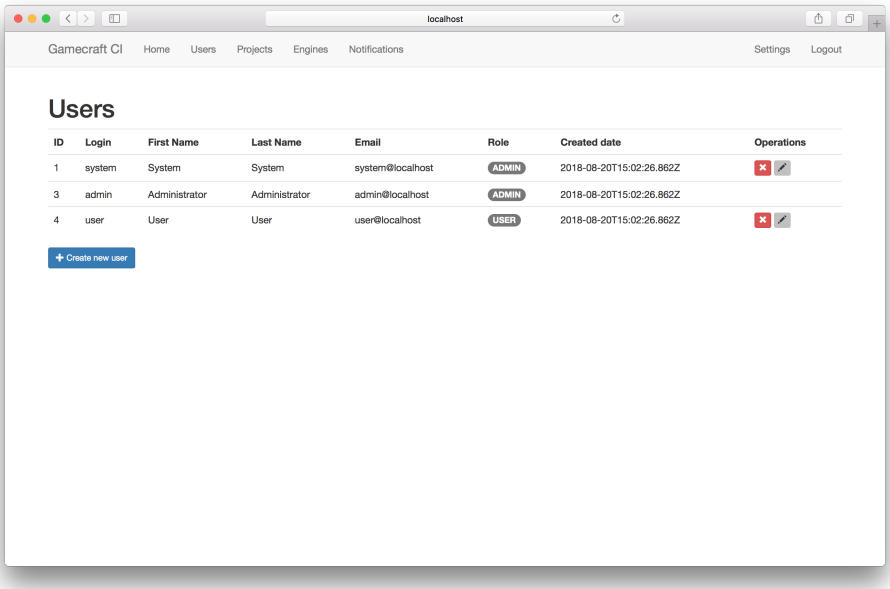

Figura F.4: Pantalla de usuarios

Desde esta pantalla, se pueden visualizar todos los usuarios que están registrados en la plataforma, así como cierta información relevante como, por ejemplo: si es usuario o administrador, su correo electrónico o la fecha de creación de la cuenta.

F.2.1. Creación de usuarios

Para crear un usuario, es necesario hacer clic en el botón azul que aparece debajo de la tabla de usuarios titulado *Create new user*.

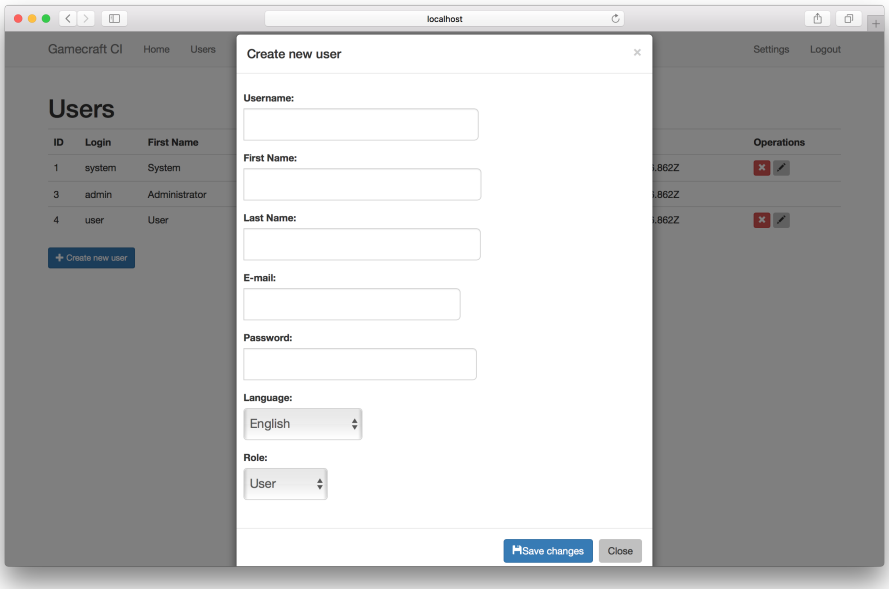

Figura F.5: Pantalla de creación de usuarios

La pantalla de creación de usuarios, representada en la Figura F.5, permite añadir un usuario proporcionando la información requerida. Una vez se ha completado el formulario, el usuario debe hacer clic en el botón azul *Save changes* para crear el usuario con los datos proporcionados.

A continuación, se proporciona una explicación de cada uno de los campos del formulario:

- *Username*. Nombre de la cuenta de usuario a crear. Se utilizará para iniciar sesión en la pantalla representada en la Figura F.1.
- *First Name*. Nombre de la persona que utilizará la cuenta de usuario a crear.
- *Last Name*. Apellidos de la persona que utilizará la cuenta de usuario a crear.
- *E-mail*. Dirección de correo electrónico de la cuenta de usuario a crear.
- *Password*. Contraseña de la cuenta de usuario a crear.
- *Language*. Determina el idioma en el que se visualizará las cadenas de texto de la plataforma.
- *Role*. Determina el rango de privilegios que el usuario tendrá dentro de la plataforma, es decir, desde aquí se puede elegir si el usuario contará con permisos de administración o no.

F.2.2. Modificación de usuarios

Para modificar un usuario, es necesario hacer clic en el botón gris del lápiz que aparece en la fila del usuario que se quiere modificar dentro de la tabla de usuarios. No se puede modificar desde esta pantalla la cuenta de usuario que se está utilizando para acceder a la plataforma. Si se quiere modificar esta cuenta de usuario, es necesario hacerlo desde la pantalla *Settings*, tal y como aparece en la sección F.1.

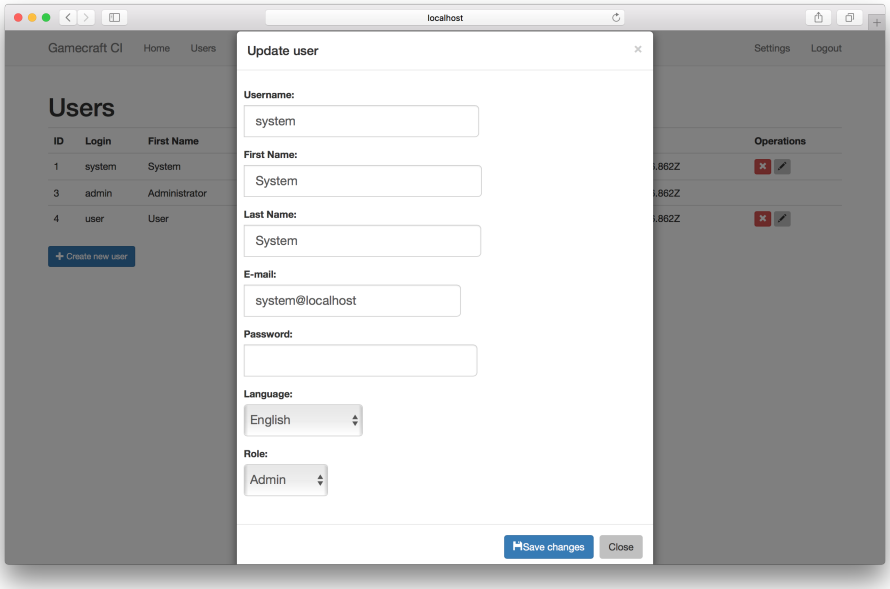

Figura F.6: Pantalla de modificación de usuarios

A continuación, se proporciona una explicación de cada uno de los campos del formulario:

- *Username*. Nombre de la cuenta de usuario a crear. Se utilizará para iniciar sesión en la pantalla representada en la Figura F.1.
- *First Name*. Nombre de la persona que utilizará la cuenta de usuario a crear.
- *Last Name*. Apellidos de la persona que utilizará la cuenta de usuario a crear.
- *E-mail*. Dirección de correo electrónico de la cuenta de usuario a crear.
- *Password*. Contraseña de la cuenta de usuario a crear.
- *Language*. Determina el idioma en el que se visualizará las cadenas de texto de la plataforma.

- *Role*. Determina el rango de privilegios que el usuario tendrá dentro de la plataforma, es decir, desde aquí se puede elegir si el usuario contará con permisos de administración o no.

F.2.3. Supresión de usuarios

Para eliminar un usuario, basta con hacer clic en el botón rojo que aparece en la fila del usuario que queremos eliminar en la tabla de usuarios. No se puede eliminar desde esta pantalla la cuenta de usuario que se está utilizando para acceder a la plataforma. La única forma de eliminar esta cuenta de usuario sería accediendo desde otra cuenta que tuviera permisos de administración.

F.3. Manipular proyectos de la plataforma

Para poder crear, modificar y eliminar proyectos que serán procesados por la plataforma, desde la pantalla inicial representada en la Figura F.2, acceder a la opción *Projects* del menú principal.

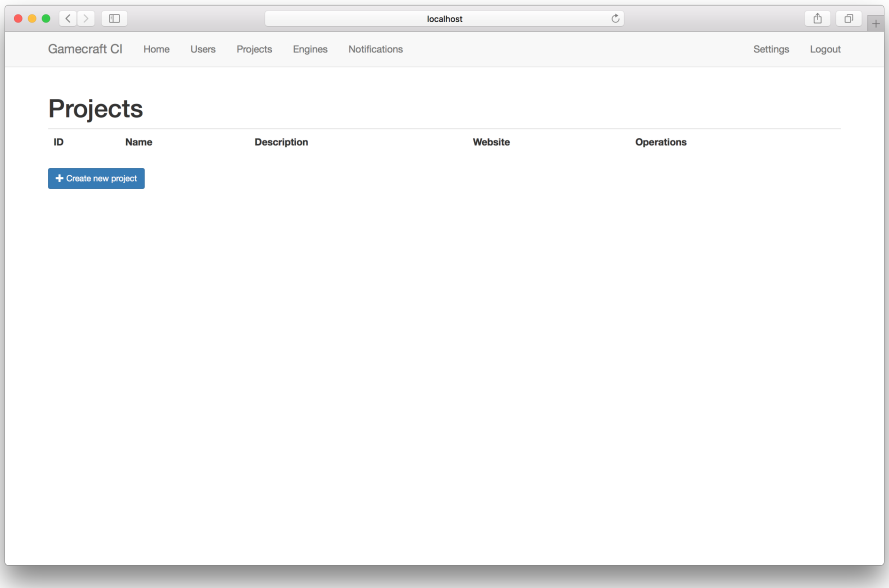

Figura F.7: Pantalla de proyectos

Desde esta pantalla, se pueden visualizar todos los proyectos que están registrados en la plataforma.

F.3.1. Creación de proyectos

Para crear un proyecto, es necesario hacer clic en el botón azul que aparece debajo de la tabla de proyectos titulado *Create new project*.

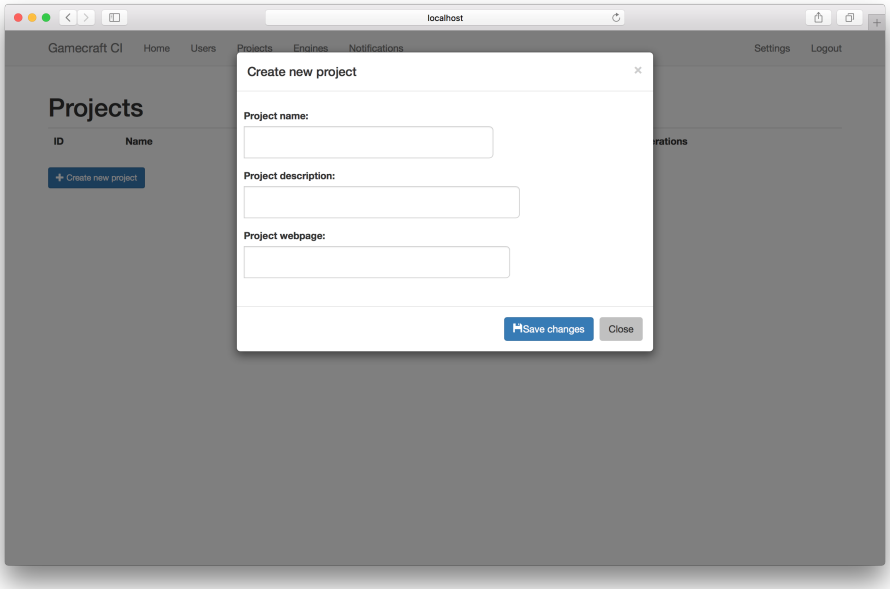

Figura F.8: Pantalla de creación de proyectos

La pantalla de creación de proyectos, representada en la Figura F.8, permite añadir un proyecto proporcionando la información requerida. Una vez se ha completado el formulario, el usuario debe hacer clic en el botón azul *Save changes* para crear el proyecto con los datos proporcionados.

A continuación, se proporciona una explicación de cada uno de los campos del formulario:

- *Project name*. Nombre del proyecto o videojuego a procesar por la plataforma.
- *Project description*. Breve descripción del proyecto o videojuego a procesar por la plataforma.
- *Project webpage*. Página web del proyecto o videojuego a procesar por la plataforma.

F.3.2. Modificación de proyectos

Para modificar un proyecto, es necesario hacer clic en el botón gris del lápiz que aparece en la fila del proyecto que se quiere modificar dentro de la tabla de proyectos.

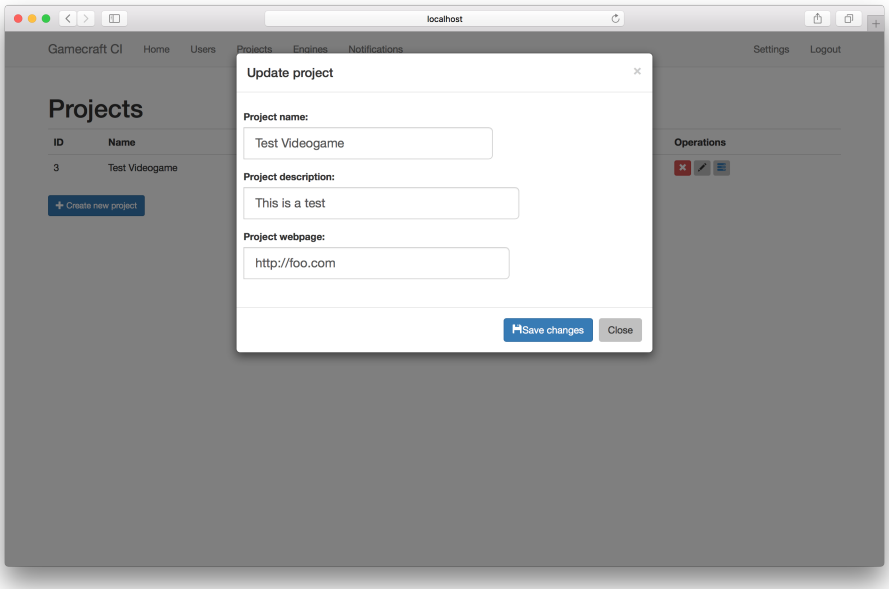

Figura F.9: Pantalla de modificación de proyectos

A continuación, se proporciona una explicación de cada uno de los campos del formulario:

- *Project name*. Nombre del proyecto o videojuego a procesar por la plataforma.
- *Project description*. Breve descripción del proyecto o videojuego a procesar por la plataforma.
- *Project webpage*. Página web del proyecto o videojuego a procesar por la plataforma.

F.3.3. Supresión de proyectos

Para eliminar un proyecto, basta con hacer clic en el botón rojo que aparece en la fila del proyecto que queremos eliminar en la tabla de proyectos. Si se elimina un proyecto, todos los pipelines asociados a dicho proyecto serán también eliminados.

F.4. Manipular motores de la plataforma

Un motor es el binario encargado de compilar y ejecutar los tests para un proyecto determinado. Un motor podría ser el compilador *gcc* por ejemplo si estuviesemos hablando de un videojuego desarrollado en C++ o del propio

ejecutable de Unity3D. Para poder crear, modificar y eliminar motores que serán utilizados en las etapas de compilación de los *pipelines*, desde la pantalla inicial representada en la Figura F.2, acceder a la opción *Engines* del menú principal.

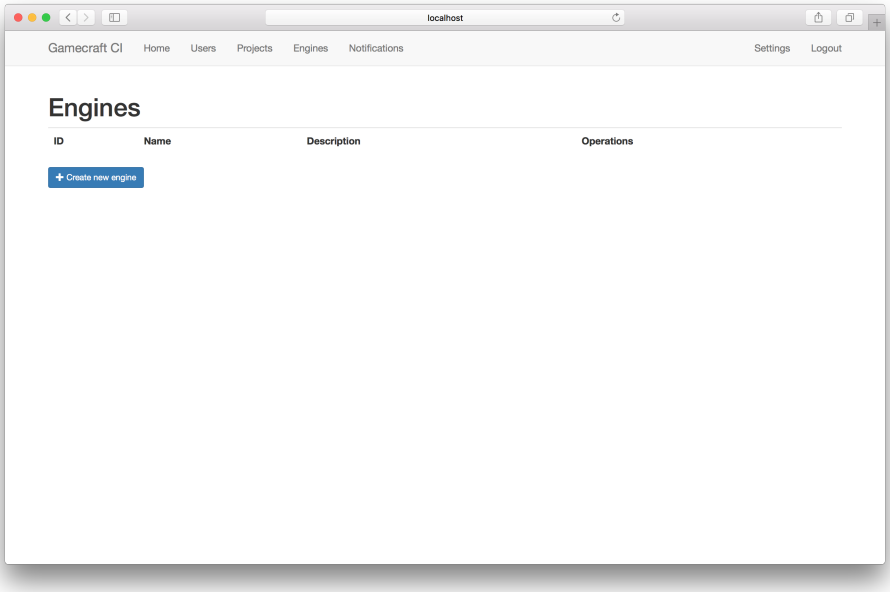

Figura F.10: Pantalla de motores

F.4.1. Creación de motores

Para crear un motor, es necesario hacer clic en el botón azul que aparece debajo de la tabla de motores titulado *Create new engine.*

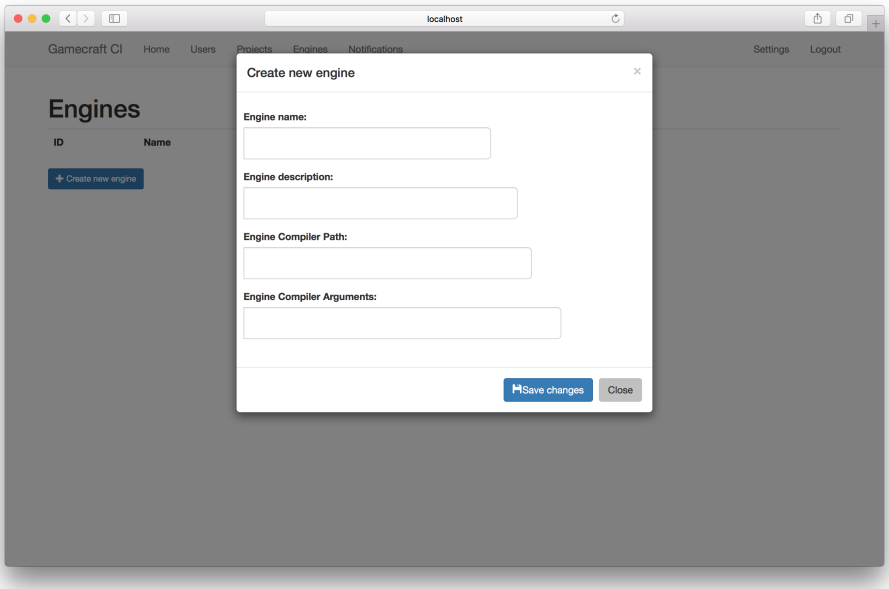

Figura F.11: Pantalla de creación de motores

A continuación, se proporciona una explicación de cada uno de los campos del formulario:

- *Engine name*. Nombre del motor a utilizar por la plataforma.
- *Engine description*. Breve descripción del motor a utilizar por la plataforma.
- *Engine Compiler Path*. Ruta de instalación del binario del motor que se encargará de realizar la compilación.
- *Engine Compiler Arguments*. Argumentos a suministrar a la hora de ejecutar el motor desde la ruta proporcionada en el campo anterior.

F.4.2. Modificación de motores

Para modificar un motor, es necesario hacer clic en el botón gris del lápiz que aparece en la fila del motor que se quiere modificar dentro de la tabla de motores.

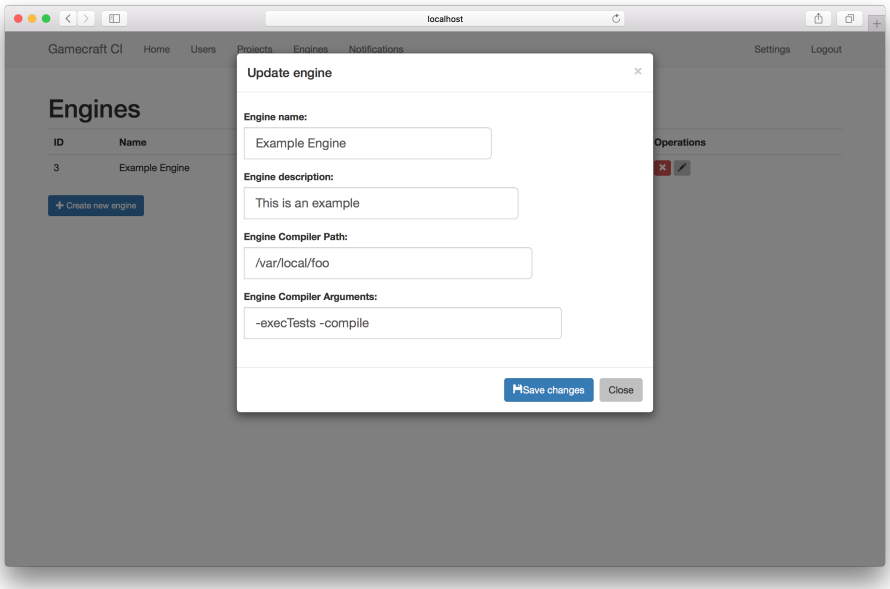

Figura F.12: Pantalla de modificación de motores

A continuación, se proporciona una explicación de cada uno de los campos del formulario:

- *Engine name*. Nombre del motor a utilizar por la plataforma.
- *Engine description*. Breve descripción del motor a utilizar por la plataforma.
- *Engine Compiler Path*. Ruta de instalación del binario del motor que se encargará de realizar la compilación.
- *Engine Compiler Arguments*. Argumentos a suministrar a la hora de ejecutar el motor desde la ruta proporcionada en el campo anterior.

F.4.3. Supresión de motores

Para eliminar un motor, basta con hacer clic en el botón rojo que aparece en la fila del motor que queremos eliminar en la tabla de motores.

F.5. Manipular notificadores de la plataforma

Un notificador es el encargado de entablar las comunicaciones a entidades externas a la plataforma. Un notificador se utiliza cuando se quiere saber el estado de ejecución de un pipeline de forma rápida, sin necesidad de acceder

a la plataforma. En Gamecraft se da soporte a varios notificadores diferentes aunque la forma de configurar cada una de ellas es similar. Para poder crear, modificar y eliminar notificadores, desde la pantalla inicial representada en la Figura F.2, acceder a la opción *Notifications* del menú principal y a continuación elegir la forma de comunicación que se quiere utilizar (correo electrónico, tweet, mensaje en Slack, mensaje en Telegram o mensaje en IRC).

F.5.1. Creación de notificadores

F.5.1.1. Correo electrónico

Para crear un notificador de correo electrónico, es necesario hacer clic en el botón azul que aparece debajo de la tabla de notificadores titulado *Create new notificator*.

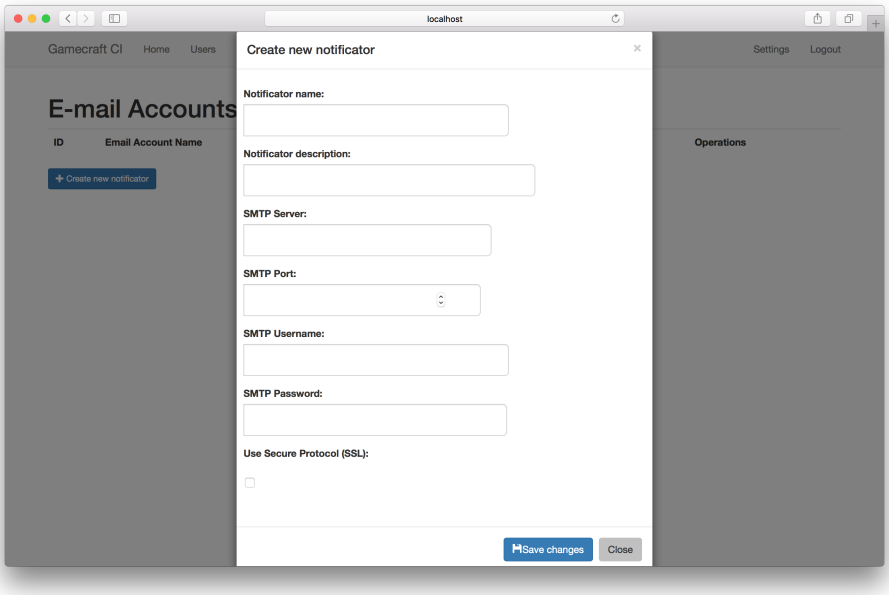

Figura F.13: Pantalla de creación de notificadores por correo electrónico

A continuación, se proporciona una explicación de cada uno de los campos del formulario:

- *Notificator name*. Nombre del notificador a utilizar por la plataforma.
- *Notificator description*. Breve descripción del notificador a utilizar por la plataforma.
- *SMTP Server*. Dirección del servidor de envío de correo electrónico.

- *SMTP Port*. Puerto del servidor de envío de correo electrónico.
- *SMTP Username*. Usuario del servidor de envío de correo electrónico.
- *SMTP Password*. Contraseña del servidor de envío de correo electrónico.
- *Use Secure Protocol (SSL)*. Utilizar el protocolo seguro SSL para conectar con el servidor de envío de correo electrónico (requerido para algunos servidores).

F.5.1.2. Twitter

Para crear un notificador de Twitter, es necesario hacer clic en el botón azul que aparece debajo de la tabla de notificadores titulado *Create new notificator*.

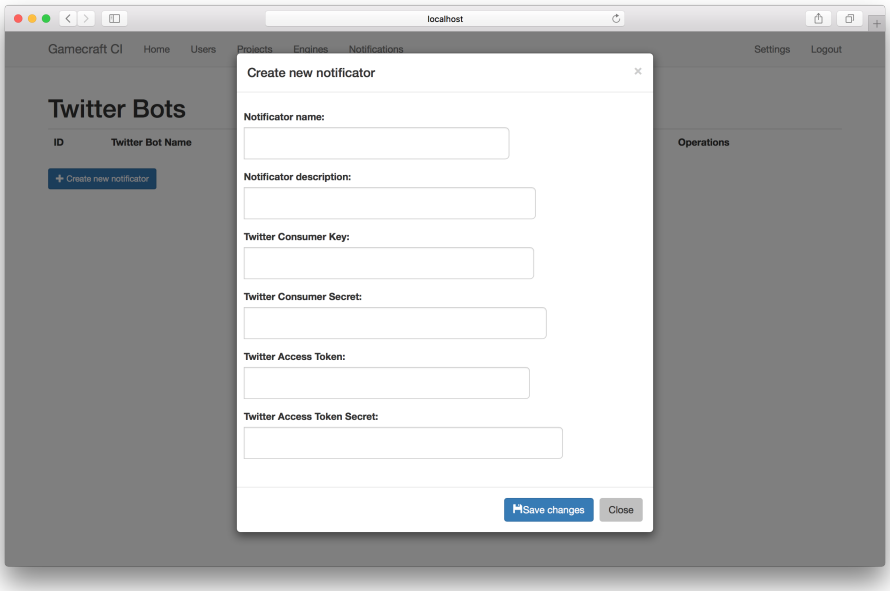

Figura F.14: Pantalla de creación de notificadores por Twitter

Es necesario previamente haber creado una aplicación en Twitter para poder utilizar su API con una cuenta de desarrollador adecuada. Para poder crear una aplicación y así obtener las claves y tokens necesarios para rellenar el formulario, es necesario seguir los siguientes pasos recogidos en las siguientes páginas: `https://developer.twitter.com/en/docs/basics/authentication/guides/access-tokens.html` y `https://support.yapsody.com/hc/en-us/articles/360003291573`.

A continuación, se proporciona una explicación de cada uno de los campos del formulario:

- *Notificator name*. Nombre del notificador a utilizar por la plataforma.
- *Notificator description*. Breve descripción del notificador a utilizar por la plataforma.
- *Twitter Consumer Key*. Clave de consumidor, necesaria para poder enviar tweets por esta plataforma. La aplicación debe tener permisos para enviar tweets.
- *Twitter Consumer Secret*. Token de consumidor, necesaria para poder enviar tweets por esta plataforma. La aplicación debe tener permisos para enviar tweets.
- *Twitter Access Token*. Clave de acceso a la cuenta de desarrollador, necesaria para poder enviar tweets por esta plataforma.
- *Twitter Access Token Secret*. Token de acceso a la cuenta de esarrollador, necesaria para poder enviar tweets por esta plataforma.

F.5.1.3. Slack

Para crear un notificador de Slack, es necesario hacer clic en el botón azul que aparece debajo de la tabla de notificadores titulado *Create new notificator*.

Es necesario previamente haber creado una aplicación en Slack. Para ello, es necesario seguir los siguientes pasos:

1. Acceder a `https://api.slack.com/apps?new_app=1`
2. Hacer clic en el botón *Create New App*.
3. Elegir un nombre de aplicación y la organización donde se quiere utilizar la aplicación. Dentro de la pantalla de gestión de la aplicación, hacer clic en *OAuth & Permissions*.
4. En la sección de *Scopes*, deben estar seleccionados los siguientes: `bot` y `chat:write:bot`.
5. Una vez pedida la autorización al administrador de la organización, debe aparecer en la parte superior el token *OAuth Access Token*.
6. Posteriormente en el menú de la izquierda, hacer clic en *Bot Users*.
7. Crear un nuevo bot con el nombre y el usuario por defecto que se quiera.

8. Posteriormente en el menú de la izquierda, hacer clic en *OAuth & Permissions*.

9. Apuntar en cualquier sitio el *Bot User OAuth Access Token*. Este token es el que posteriormente se añadirá dentro del microservicio para crear el enlace entre la plataforma y la aplicación de Slack.

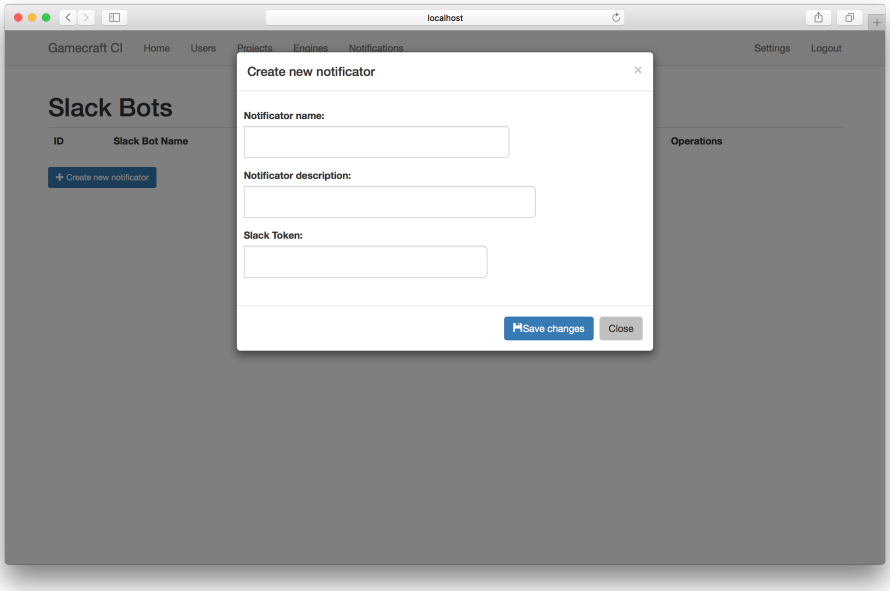

Figura F.15: Pantalla de creación de notificadores por Slack

A continuación, se proporciona una explicación de cada uno de los campos del formulario:

- *Notificator name*. Nombre del notificador a utilizar por la plataforma.
- *Notificator description*. Breve descripción del notificador a utilizar por la plataforma.
- *Slack Token*. Es el token de autenticación (Auth token) de la cuenta de bot creada en la aplicación de Slack.

F.5.1.4. Telegram

Para crear un notificador de Telegram, es necesario hacer clic en el botón azul que aparece debajo de la tabla de notificadores titulado *Create new notificator*.

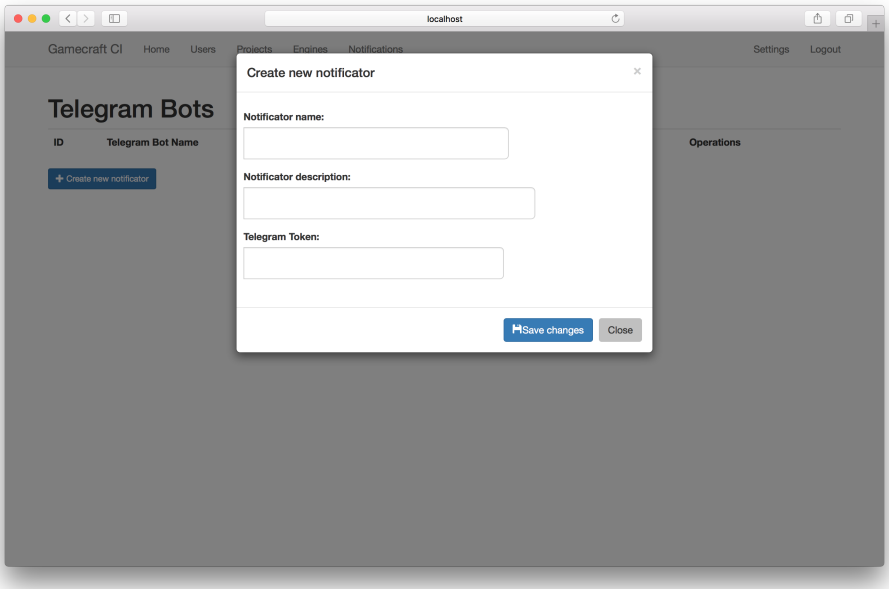

Figura F.16: Pantalla de creación de notificadores por Telegram

A continuación, se proporciona una explicación de cada uno de los campos del formulario:

- *Notificator name*. Nombre del notificador a utilizar por la plataforma.
- *Notificator description*. Breve descripción del notificador a utilizar por la plataforma.
- *Telegram Token*. Es el token de autenticación (Auth token) de la cuenta de bot creada en la aplicación de Telegram.

Para obtener el token del bot de Telegram a utilizar en Gamecraft, es necesario seguir los siguientes pasos:

1. Mediante Telegram, mandar el mensaje `/newbot` al *BotFather* (@BotFather).
2. Posteriormente el propio bot te preguntará por el nombre que quieres para tu bot. Importante, tiene que acabar en Bot.
3. Ahora es el turno de configurar la privacidad de tu bot. Escribimos `/setprivacy` y posteriormente, el nombre de tu bot mencionándolo por su nombre "@Bot". El *BotFather* te responderá con las opciones y puedes hacer que sólo atienda a mensajes que lo mencionen o que empiecen por un "/" con el modo `ENABLED`. O recibir cualquier mensaje del grupo si marcamos la opción `DISABLED`.

4. Para obtener el token del bot y poder incorporarlo al formulario, escribir el mensaje `/token`.

F.5.1.5. IRC

Para crear un notificador de IRC, es necesario hacer clic en el botón azul que aparece debajo de la tabla de notificadores titulado *Create new notificator*.

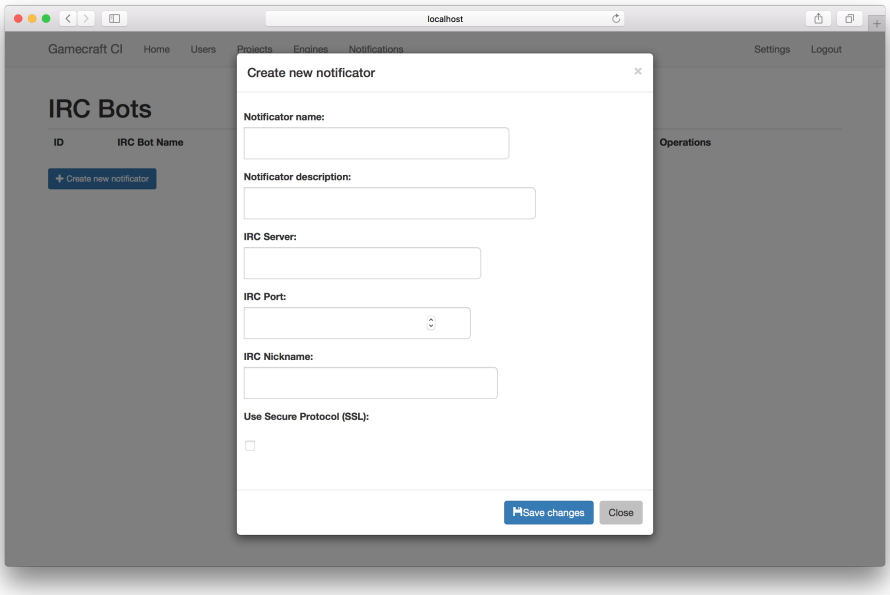

Figura F.17: Pantalla de creación de notificadores por IRC

A continuación, se proporciona una explicación de cada uno de los campos del formulario:

- *Notificator name*. Nombre del notificador a utilizar por la plataforma.
- *Notificator description*. Breve descripción del notificador a utilizar por la plataforma.
- *IRC Server*. Dirección del servidor de IRC para establecer la conexión.
- *IRC Port*. Puerto del servidor de IRC para establecer la conexión.
- *IRC Nickname*. Usuario del servidor de IRC para enviar mensajes.
- *Use Secure Protocol (SSL)*. Utilizar el protocolo seguro SSL para conectar con el servidor de IRC (requerido para establecer conexión con algunos servidores).

F.5.2. Modificación de notificadores

F.5.2.1. Correo electrónico

Para modificar un notificador de correo electrónico, es necesario hacer clic en el botón gris del lápiz que aparece en la fila del notificador que se quiere modificar dentro de la tabla de notificadores.

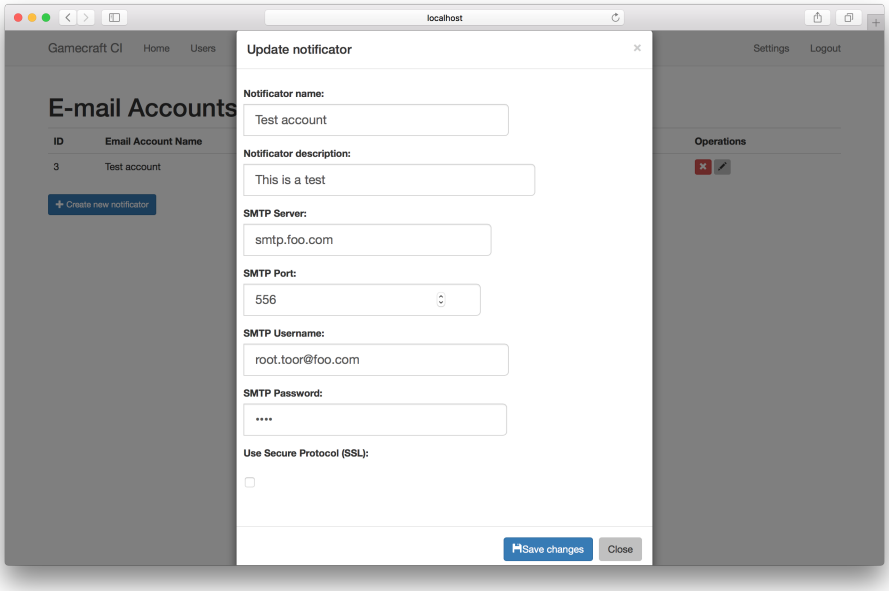

Figura F.18: Pantalla de modificación de notificadores por e-mail

A continuación, se proporciona una explicación de cada uno de los campos del formulario:

- *Notificator name*. Nombre del notificador a utilizar por la plataforma.
- *Notificator description*. Breve descripción del notificador a utilizar por la plataforma.
- *SMTP Server*. Dirección del servidor de envío de correo electrónico.
- *SMTP Port*. Puerto del servidor de envío de correo electrónico.
- *SMTP Username*. Usuario del servidor de envío de correo electrónico.
- *SMTP Password*. Contraseña del servidor de envío de correo electrónico.
- *Use Secure Protocol (SSL)*. Utilizar el protocolo seguro SSL para conectar con el servidor de envío de correo electrónico (requerido para algunos servidores).

F.5.2.2. Twitter

Para modificar un notificador de Twitter, es necesario hacer clic en el botón gris del lápiz que aparece en la fila del notificador que se quiere modificar dentro de la tabla de notificadores.

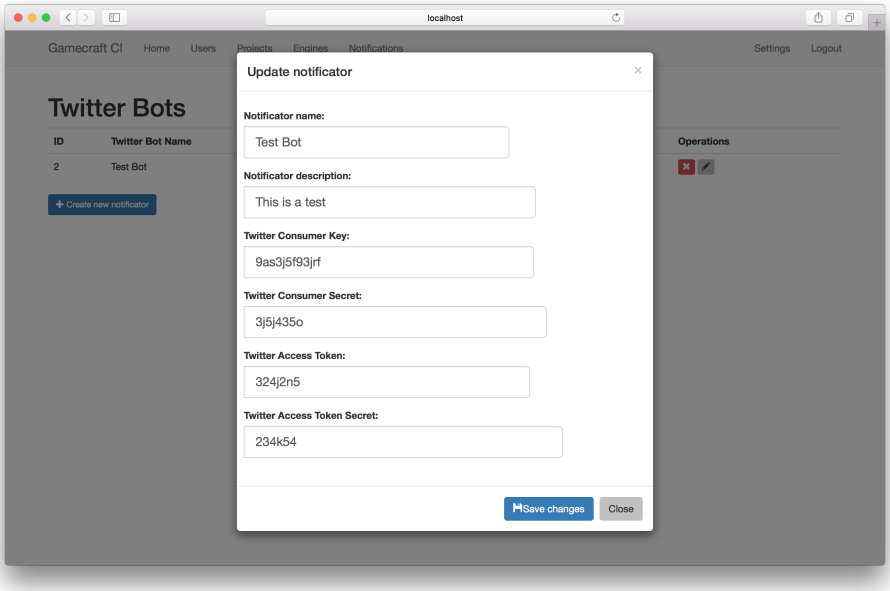

Figura F.19: Pantalla de modificación de notificadores por Twitter

A continuación, se proporciona una explicación de cada uno de los campos del formulario:

- *Notificator name*. Nombre del notificador a utilizar por la plataforma.
- *Notificator description*. Breve descripción del notificador a utilizar por la plataforma.
- *Twitter Consumer Key*. Clave de consumidor, necesaria para poder enviar tweets por esta plataforma. La aplicación debe tener permisos para enviar tweets.
- *Twitter Consumer Secret*. Token de consumidor, necesaria para poder enviar tweets por esta plataforma. La aplicación debe tener permisos para enviar tweets.
- *Twitter Access Token*. Clave de acceso a la cuenta de desarrollador, necesaria para poder enviar tweets por esta plataforma.
- *Twitter Access Token Secret*. Token de acceso a la cuenta de esarrollador, necesaria para poder enviar tweets por esta plataforma.

F.5.2.3. Slack

Para modificar un notificador de Slack, es necesario hacer clic en el botón gris del lápiz que aparece en la fila del notificador que se quiere modificar dentro de la tabla de notificadores.

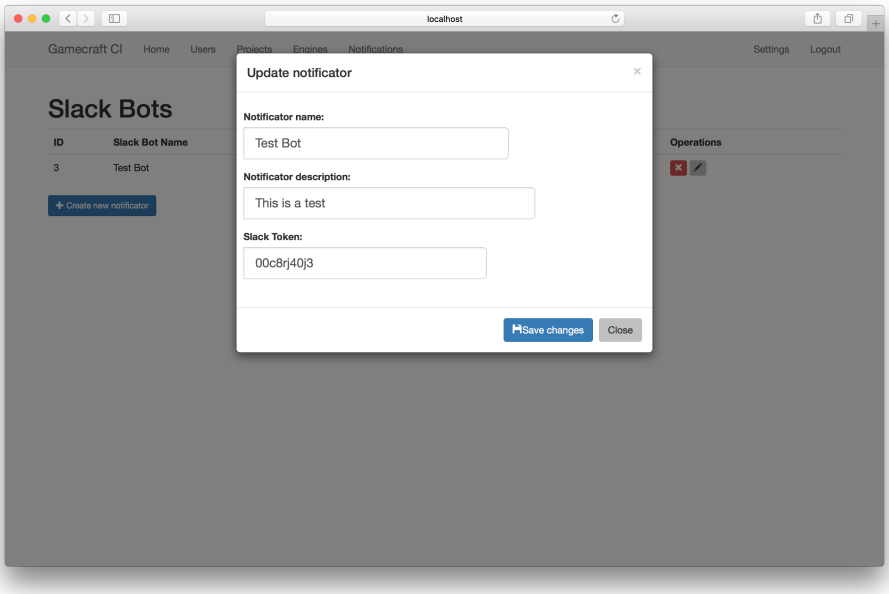

Figura F.20: Pantalla de modificación de notificadores por Slack

A continuación, se proporciona una explicación de cada uno de los campos del formulario:

- *Notificator name*. Nombre del notificador a utilizar por la plataforma.
- *Notificator description*. Breve descripción del notificador a utilizar por la plataforma.
- *Slack Token*. Es el token de autenticación (Auth token) de la cuenta de bot creada en la aplicación de Slack.

F.5.2.4. Telegram

Para modificar un notificador de Telegram, es necesario hacer clic en el botón gris del lápiz que aparece en la fila del notificador que se quiere modificar dentro de la tabla de notificadores.

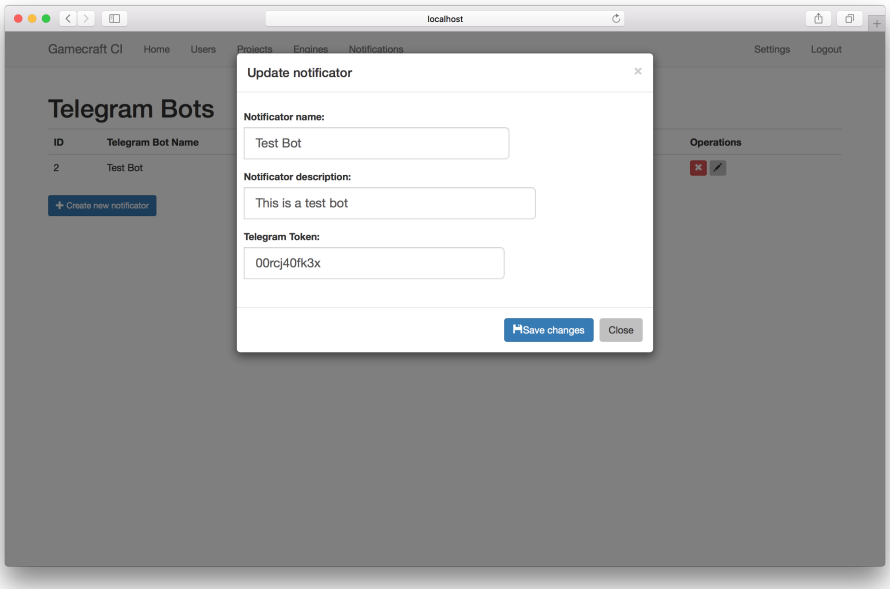

Figura F.21: Pantalla de modificación de notificadores por Telegram

A continuación, se proporciona una explicación de cada uno de los campos del formulario:

- *Notificator name*. Nombre del notificador a utilizar por la plataforma.
- *Notificator description*. Breve descripción del notificador a utilizar por la plataforma.
- *Telegram Token*. Es el token de autenticación (Auth token) de la cuenta de bot creada en la aplicación de Telegram.

F.5.2.5. IRC

Para modificar un notificador de IRC, es necesario hacer clic en el botón gris del lápiz que aparece en la fila del notificador que se quiere modificar dentro de la tabla de notificadores.

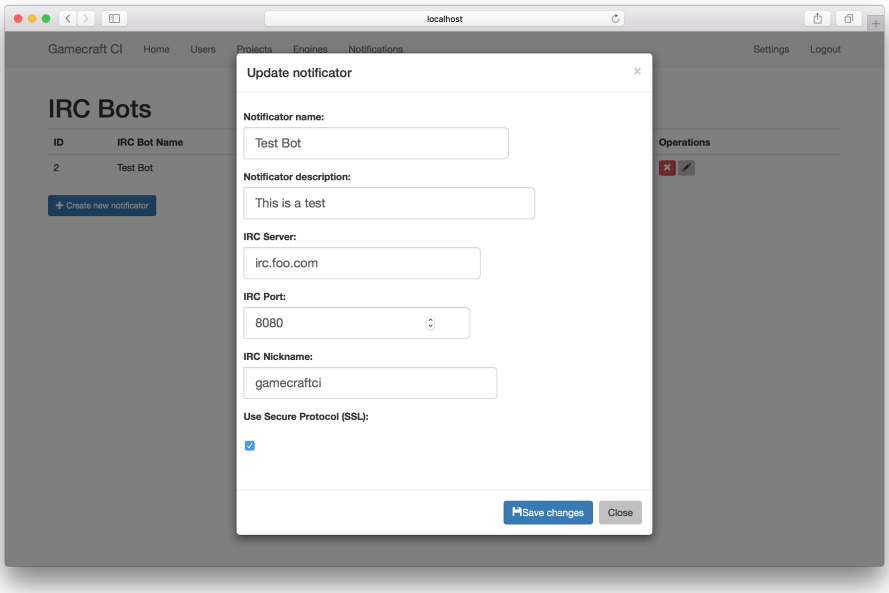

Figura F.22: Pantalla de modificación de notificadores por IRC

A continuación, se proporciona una explicación de cada uno de los campos del formulario:

- *Notificator name*. Nombre del notificador a utilizar por la plataforma.
- *Notificator description*. Breve descripción del notificador a utilizar por la plataforma.
- *IRC Server*. Dirección del servidor de IRC para establecer la conexión.
- *IRC Port*. Puerto del servidor de IRC para establecer la conexión.
- *IRC Nickname*. Usuario del servidor de IRC para enviar mensajes.
- *Use Secure Protocol (SSL)*. Utilizar el protocolo seguro SSL para conectar con el servidor de IRC (requerido para establecer conexión con algunos servidores).

F.5.3. Supresión de notificadores

Para eliminar un notificador, basta con hacer clic en el botón rojo que aparece en la fila del notificador que queremos eliminar en la tabla de notificadores.

F.6. Manipular pipelines de la plataforma

Un *pipeline* es una secuencia de pasos que se irán ejecutando secuencialmente por la plataforma para poder compilar con éxito un proyecto. Para poder crear, modificar y eliminar pipelines que serán procesados por la plataforma, desde la pantalla inicial representada en la Figura F.2, acceder a la opción *Projects* del menú principal. Una vez en esta pantalla, centraremos nuestra visión en el proyecto (dentro de la tabla de proyectos) que queremos procesar en la plataforma a través de un pipeline y accederemos a la pantalla de pipelines haciendo clic en el botón situado a la más derecha dentro de la columna de *Operations*.

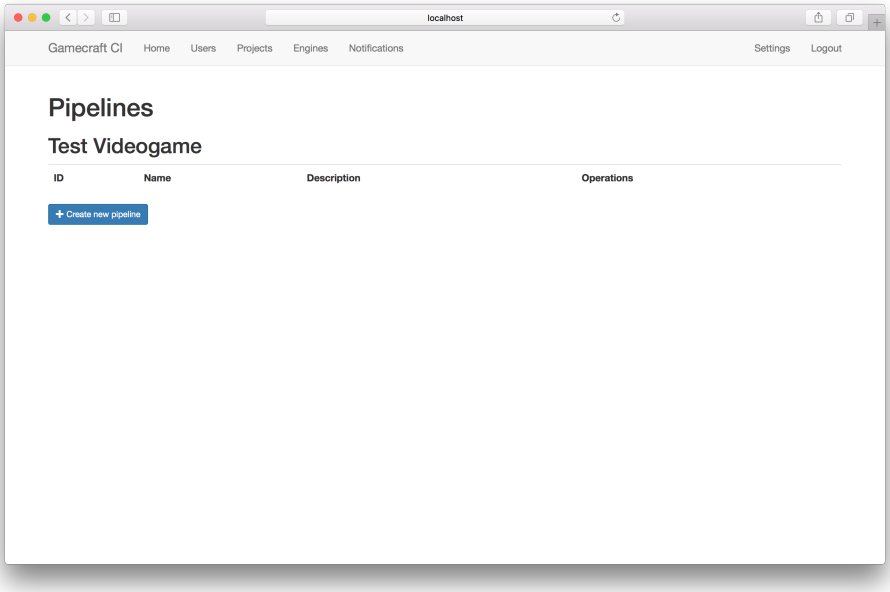

Figura F.23: Pantalla de *pipelines* del proyecto *Test Videogame*

F.6.1. Creación de pipelines

Para crear un *pipeline*, es necesario hacer clic en el botón azul que aparece debajo de la tabla de *pipelines* titulado *Create new pipeline*.

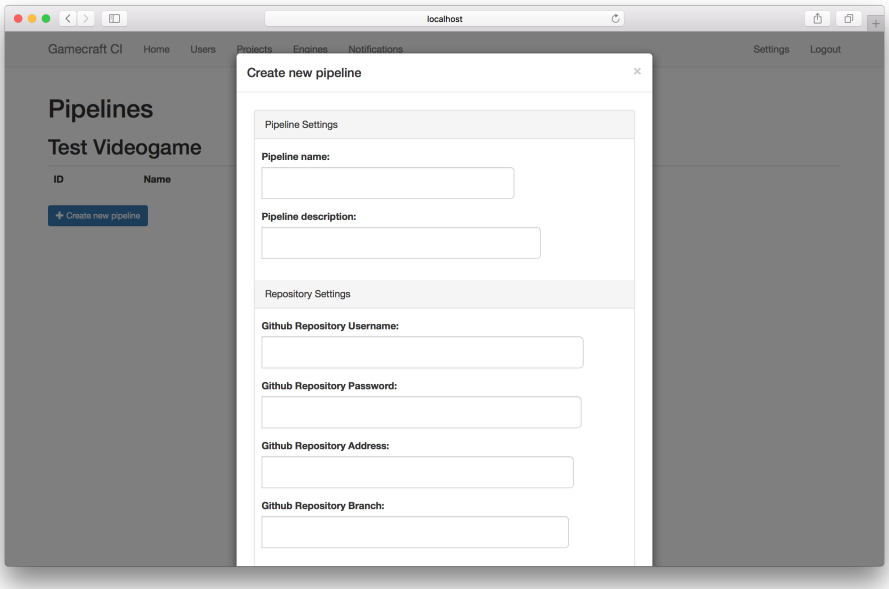

Figura F.24: Pantalla de creación de *pipelines* del proyecto *Test Videogame*

A continuación, se proporciona una explicación de cada uno de los campos del formulario:

- *Pipeline name*. Nombre del *pipeline* a procesar por la plataforma.
- *Pipeline description*. Breve descripción del *pipeline* a procesar por la plataforma.
- *Github Repository Username*. Nombre de usuario de la cuenta de Github para obtener el código fuente del repositorio.
- *Github Repository Password*. Contraseña de la cuenta de Github para obtener el código fuente del repositorio.
- *Github Repository Address*. Dirección del repositorio de Github para obtener el código fuente.
- *Github Repository Branch*. Rama donde se encuentra el código fuente del proyecto dentro del repositorio.
- *Engine Name*. Nombre del motor a utilizar para compilar el proyecto.
- *Engine Compiler Path*. Ruta del motor seleccionado. Esta información se obtiene directamente del módulo de *Engines*.
- *Engine Compiler Arguments*. Argumentos del motor seleccionado. Esta información se obtiene directamente del módulo de *Engines*.

- *FTP Address*. Dirección del servidor FTP donde se subirá el resultado de la compilación. Si la compilación ha sido exitosa, se subira los binarios del videojuego compilado.

- *FTP Port*. Puerto del servidor FTP donde se subirá el resultado de la compilación.

- *FTP Username*. Usuario del servidor FTP donde se subirá el resultado de la compilación.

- *FTP Password*. Contraseña del servidor FTP donde se subirá el resultado de la compilación.

- *Dropbox App Key*. Clave de acceso de aplicación para poder almacenar el resultado de la compilación en una cuenta de Dropbox.

- *Dropbox Token*. Token para poder almacenar el resultado de la compilación en una cuenta de Dropbox.

- *Notificator Type*. Tipo de notificador a utilizar en cada estado del pipeline. A elegir entre correo electrónico, Twitter, Telegram, IRC o Slack.

- *Notificator Name*. Nombre del notificador a utilizar. La información de conexión del notificador elegido se obtiene directamente del módulo *Notifications*.

- *Schedule Type*. El pipeline puede ser ejecutado periodicamente a través de una regla *cron* o únicamente cuando se produce un cambio en el repositorio a través de Webhook.

- *Schedule Cronjob*. Si se ha elegido en el punto anterior la opción *Cronjob*, aqui se ha de insertar la sentencia *cron* que indique la frecuencia de ejecución del pipeline.

F.6.2. Modificación de pipelines

Para modificar un *pipeline*, es necesario hacer clic en el botón del lápiz que aparece en la fila del *pipeline* que se quiere modificar dentro de la tabla de *pipelines*.

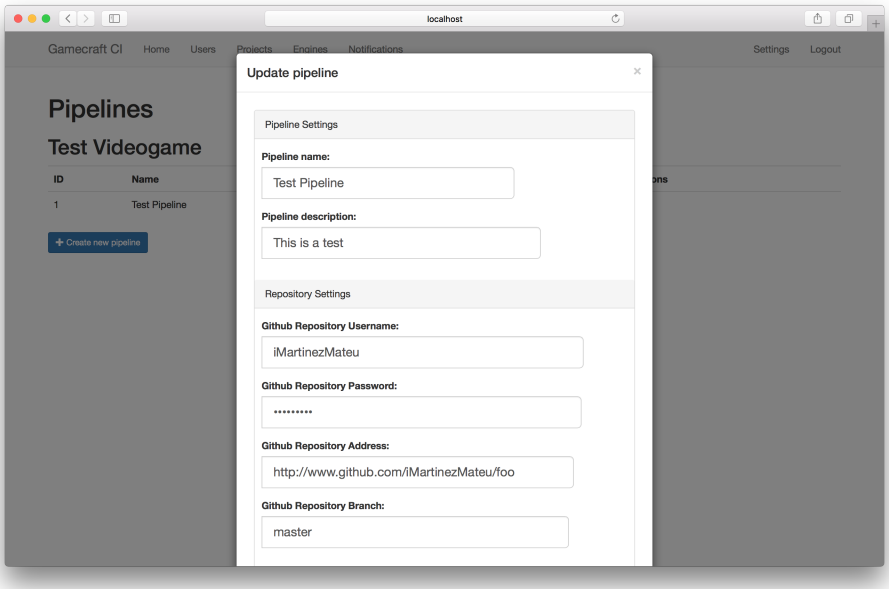

Figura F.25: Pantalla de modificación de *pipelines* del proyecto *Test Videogame*

A continuación, se proporciona una explicación de cada uno de los campos del formulario:

- *Pipeline name*. Nombre del *pipeline* a procesar por la plataforma.
- *Pipeline description*. Breve descripción del *pipeline* a procesar por la plataforma.
- *Github Repository Username*. Nombre de usuario de la cuenta de Github para obtener el código fuente del repositorio.
- *Github Repository Password*. Contraseña de la cuenta de Github para obtener el código fuente del repositorio.
- *Github Repository Address*. Dirección del repositorio de Github para obtener el código fuente.
- *Github Repository Branch*. Rama donde se encuentra el código fuente del proyecto dentro del repositorio.
- *Engine Name*. Nombre del motor a utilizar para compilar el proyecto.
- *Engine Compiler Path*. Ruta del motor seleccionado. Esta información se obtiene directamente del módulo de *Engines*.

- *Engine Compiler Arguments*. Argumentos del motor seleccionado. Esta información se obtiene directamente del módulo de *Engines*.
- *FTP Address*. Dirección del servidor FTP donde se subirá el resultado de la compilación. Si la compilación ha sido exitosa, se subira los binarios del videojuego compilado.
- *FTP Port*. Puerto del servidor FTP donde se subirá el resultado de la compilación.
- *FTP Username*. Usuario del servidor FTP donde se subirá el resultado de la compilación.
- *FTP Password*. Contraseña del servidor FTP donde se subirá el resultado de la compilación.
- *Dropbox App Key*. Clave de acceso de aplicación para poder almacenar el resultado de la compilación en una cuenta de Dropbox.
- *Dropbox Token*. Token para poder almacenar el resultado de la compilación en una cuenta de Dropbox.
- *Notificator Type*. Tipo de notificador a utilizar en cada estado del pipeline. A elegir entre correo electrónico, Twitter, Telegram, IRC o Slack.
- *Notificator Name*. Nombre del notificador a utilizar. La información de conexión del notificador elegido se obtiene directamente del módulo *Notifications*.
- *Schedule Type*. El pipeline puede ser ejecutado periodicamente a través de una regla *cron* o únicamente cuando se produce un cambio en el repositorio a través de Webhook.
- *Schedule Cronjob*. Si se ha elegido en el punto anterior la opción *Cronjob*, aqui se ha de insertar la sentencia *cron* que indique la frecuencia de ejecución del pipeline.

F.6.3. Supresión de pipelines

Para eliminar un pipeline, basta con hacer clic en el botón rojo que aparece en la fila del *pipeline* que queremos eliminar en la tabla de *pipelines*.

F.6.4. Ejecución de pipelines

Para ejecutar manualmente un *pipeline* y comprobar que funciona correctamente, basta con hacer clic en el botón verde que aparece en la fila del *pipeline* que queremos ejecutar en la tabla de *pipelines*. Aparecerá un cuadro de diálogo con la información de salida y la posibilidad de almacenar este contenido en un archivo una vez termine de ejecutarse el *pipeline*.