



**Escuela Superior
de Ingeniería y Tecnología**
Universidad de La Laguna

Máster Universitario en Ingeniería Informática

Trabajo Fin de Máster

GH2GR: Github Classroom como fuente de datos para GoRace

GH2GR: Github Classroom as a data source for GoRace

Claudio Néstor Yanes Mesa

La Laguna, 6 de Julio de 2024

D. **Casiano Rodríguez León**, con N.I.F. 42020072S profesor Catedrático de Universidad del área de Lenguajes y Sistemas Informáticos, adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A N

Que la presente memoria titulada:

"GH2GR: Github Classroom como fuente de datos para GoRace"

ha sido realizada bajo nuestra dirección por D. **Claudio Néstor Yanes Mesa**, con N.I.F. 79.087.556-D.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 6 de julio de 2024.

Agradecimientos

A Casiano Rodríguez León por su tutela en este proyecto, así como por su tiempo, ayuda y paciencia.

A Mercedes Ruiz y Alejandro Calderón por su inestimable ayuda y por su fantástica plataforma en la que he podido basar este proyecto.

A la Universidad de La Laguna por haberme provisto de una enseñanza de calidad que me ha permitido llegar hasta este punto.

Finalmente a mi familia por apoyarme en el transcurso de mi vida académica.

Licencia



© Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-SinObraDerivada 4.0 Internacional.

Resumen

Este trabajo investiga el creciente interés por la gamificación como método para dinamizar el aprendizaje dentro del campo de las Tecnologías de la Información (TI). Presentamos GH2GR, un programa de software que actúa como puente entre GitHub Classroom, una plataforma de enseñanza prevalente para asignaturas de TI, y GoRace, una plataforma de gamificación extensible. Mientras que los enfoques tradicionales de gamificación pueden sin duda ayudar a mejorar el compromiso de los estudiantes, GoRace proporciona un conjunto de herramientas más rico en características y adaptable. GH2GR agiliza la incorporación de las mecánicas de GoRace en los flujos de trabajo existentes de GitHub Classroom. Esta innovación permite a los profesores gamificar sus cursos de TI sin necesidad de utilizar herramientas adicionales. Aprovechando GH2GR, los instructores pueden introducir el poder motivacional de la gamificación dentro del entorno familiar y establecido de GitHub Classroom, minimizando la interrupción de las prácticas de enseñanza actuales. Este artículo explora el diseño y el desarrollo de GH2GR, evaluando su potencial para mejorar la participación de los estudiantes y los resultados del aprendizaje en la enseñanza de TI.

Palabras clave: Ludificación, Gamificación, GoRace, Github Classroom

Abstract

This work investigates the growing interest in gamification as a method to invigorate learning within the Information Technology (IT) field. We introduce GH2GR, a software program that acts as a bridge between GitHub Classroom, a prevalent teaching platform for IT subjects, and GoRace, an extensible gamification platform. While traditional gamification approaches can undoubtedly enhance student engagement, GoRace provides a more feature-rich and adaptable toolkit. GH2GR streamlines the incorporation of GoRace mechanics into existing GitHub Classroom workflows. This innovation empowers professors to seamlessly gamify their IT courses without requiring extensive use of additional tools. By leveraging GH2GR, instructors can introduce the motivational power of gamification within the familiar and established environment of GitHub Classroom, minimizing disruption to current teaching practices. This thesis explores the design and development of GH2GR, evaluating its potential to improve student engagement and learning outcomes in IT education.

Keywords: Gamification, GoRace, Github Classroom

Índice general

1	Introducción	1
1.1	Introducción	1
1.2	Gamificación	2
1.3	Github	2
1.4	Github Classroom	2
1.5	GitHub API y GitHub API Classroom	3
1.6	GitHub Apps	3
1.7	GitHub Actions	4
1.8	GoRace	4
1.9	GoRace API	5
2	Objetivos, estado del arte y metodología de trabajo	6
2.1	Objetivos	6
2.2	Estado del arte	7
2.3	Metodología de trabajo	7
	2.3.1 Planificación y Revisión de Sprints	7
	2.3.2 Comunicación y Coordinación	8
	2.3.3 Pruebas y evaluación de los usuarios	9
3	Diseño del sistema e implementación	10
3.1	Traducción entre GoRace y Github Classroom	10
3.2	Diseño a alto nivel del sistema	12
3.3	Sobre la GitHub App	14
3.4	Sobre la GitHub Action	15
3.5	GH2GR Server en detalle	16
	3.5.1 Estructura del código	17
	3.5.2 Los tipos abstractos de datos principales	19
	3.5.3 La API de gestión	21
	3.5.4 La base de datos	22
	3.5.5 El método de construcción y despliegue	22
	3.5.6 Sobre Gh2GR Client	23
4	Modo de empleo	25
4.1	Crear y configurar la GitHub App	25
4.2	Construir y distribuir el servidor	28
	4.2.1 Distribución mediante fichero	30
	4.2.2 Distribución mediante un registro de imágenes OCI	30

4.3	Configurar el servidor	31
4.4	Añadir usuarios y crear credenciales	35
4.5	Construcción del cliente	36
4.6	Configurar el cliente	36
4.7	Añadir una carrera	38
4.8	Añadir una actividad y prepararla para su uso	40
5	Evaluación en un entorno real	43
5.1	Retroalimentación de los alumnos	44
6	Conclusiones y líneas futuras	46
6.1	Conclusiones	46
6.2	Líneas futuras	47
7	Conclusions and future lines of work	48
7.1	Conclusions	48
7.2	Future lines of work	49
	Bibliografía	50
	Glosario	55
A	Códigos	57
A.1	Esquema OpenAPI de la API de gestión	57

Índice de Figuras

3.1	Ejemplo de pruebas automatizadas de evaluación agrupadas en Github Classroom. En la imagen se muestran tres grupos: FUND, OBJ y MAN. . . .	11
3.2	Diagrama que muestra los componentes a alto nivel que interactúan en el funcionamiento de GH2GR. Los círculos representan componentes, mientras que las flechas indican que un componente (el que origina la flecha) puede iniciar comunicación con otro (al que apunta la flecha).	12
3.3	Diagrama de flujo de los eventos que ocurren tras la realización un <i>push</i> . Leyenda: La figura de un hombre representa un actor, un rectángulo representan un componente. Las flechas solidas representan una operación síncrona, las punteadas una asíncrona. El ensañamiento de la línea de la vida significa que el componente está trabajando en el periodo ensanchado.	13
3.4	Pantalla de instalación de la GitHub App en una organización.	15
4.1	Perfil de una organización en GitHub con el botón que dirige a los ajustes resaltado.	25
4.2	Barra lateral de los ajustes de la organización mostrando la opción para ir a las GitHub Apps.	26
4.3	Botón para crear una nueva GitHub App.	26
4.4	Formulario para la creación de la GitHub App con los campos ha cambiar resaltados en amarillo.	27
4.5	Botón para crear la clave privada.	27
4.6	Clave privada creada y el navegador ofreciendo su descarga.	28
4.7	Lugar donde encontrar el App ID de nuestra GitHub App	28
4.9	Salida del comando de ayuda de Gh2GR Server.	29
4.8	Ejecución del comando de construcción de GH2GR Server. Parte de la salida omitida por brevedad.	29
4.10	Importado y exportado de una imagen OCI.	30
4.11	Botón para crear un repositorio en Docker Hub	31
4.12	Formulario de creación	32
4.13	Iniciar sesión en Docker Hub mediante Podman.	32
4.14	Salida del comando para subir la imagen a Docker Hub	33
4.15	Descarga de la imagen OCI y posterior salida del comando de ayuda de Gh2GR Server.	33
4.16	Comando para ejecutar GH2GR Server	34
4.17	Salida del comando que agrega un usuario a GH2GR Server.	35
4.18	Salida del comando para crear las credenciales de un usuario de GH2GR Server.	35
4.19	Salida del comando principal de GH2GR Client.	36
4.20	Inicio de sesión en GitHub desde GH2GR Client.	37
4.21	Inicio de sesión en GH2GR.	37

4.22 Mensaje indicándonos que nuestro token de GitHub se ha enviado con éxito a nuestro usuario en GH2GR Server.	38
4.23 Crear un JWT en GoRaceAdmin.	39
4.24 Ventana que muestra el valor del JWT en GoRaceAdmin.	39
4.25 Permitir una IP para la API de GoRace en GoRaceAdmin.	39
4.26 Formulario para añadir la carrera a GH2GR.	39
4.27 Instrucciones para descargar el <i>roster</i> de GitHub Classroom.	40
4.28 Comando que envía la traducción entre nombre de usuario de GitHub del alumnado y sus correos electrónicos.	40
4.29 Formulario para la creación de un nuevo <i>assignment</i> en GitHub Classroom.	41
4.30 Comando de gh2gr generate.	42

Índice de Tablas

5.1 Resultado de la encuesta de satisfacción del alumnado con GH2GR. El número de la pregunta corresponde con su posición en la lista del apartado 5.1. 45

Capítulo 1

Introducción

1.1. Introducción

En el panorama en rápida evolución de la enseñanza de las tecnologías de la información (TI), la ludificación o gamificación ha surgido como una poderosa estrategia para potenciar el aprendizaje mediante la incorporación de elementos de diseño de juegos en contextos no lúdicos [1]. En este proyecto, nuestro objetivo es proporcionar una alternativa que simplifique el desarrollo de experiencias de ludificación para la enseñanza en el sector TI. Para este fin, nos valdremos de dos herramientas preexistentes: GoRace y Github Classroom.

GoRace está diseñada para fomentar un nivel más profundo de compromiso y motivación entre los estudiantes proporcionando un entorno flexible y dinámico que se adapta a diversas necesidades educativas. A diferencia de las plataformas tradicionales, GoRace ofrece un conjunto de herramientas que permiten a los educadores crear una experiencia de aprendizaje más envolvente que sea a la vez divertida y eficaz.

Paralelamente, GitHub Classroom se ha consolidado como una plataforma sólida dentro del ecosistema de GitHub, ampliamente adoptada por los educadores para difundir cursos de TI y gestionar las entregas de tareas de los estudiantes. Su integración con la plataforma GitHub aprovecha herramientas y flujos de trabajo familiares, lo que la convierte en un activo inestimable para la educación en TI.

Para salvar la brecha entre estas dos plataformas, este trabajo presenta GH2GR, una novedosa solución de software que integra a la perfección GitHub Classroom con GoRace. GH2GR automatiza el proceso de gamificación dentro de GoRace para las asignaturas de TI que usan GitHub, minimizando las herramientas adicionales requeridas por los educadores. Esta integración permite a los profesores seguir utilizando las herramientas a las que están acostumbrados mientras aprovechan los beneficios de la gamificación a través de GoRace.

Las secciones siguientes profundizarán en los fundamentos teóricos de la gamificación en la educación, la arquitectura y las características de GH2GR, el papel de GitHub Classroom en la educación de TI, y la funcionalidad y el impacto real de GH2GR.

1.2. Gamificación

La ludificación (o gamificación) es la incorporación estratégica de elementos y principios de diseño de juegos en contextos no lúdicos con la intención de mejorar la experiencia de usuario y el compromiso de estos [2].

En el contexto educativo, la gamificación puede incrementar notablemente el compromiso y la motivación de los alumnos. Al convertir las tareas de aprendizaje en experiencias lúdicas, se favorece la participación activa de los estudiantes y una mejor retención de la información. Además, este método promueve la sensación de logro y avance a través de sistemas de recompensas y reconocimientos [3, 4, 5].

1.3. Github

El desarrollo de software es un proceso iterativo, caracterizado por el perfeccionamiento y la adaptación constantes. En este proceso es fundamental el control de versiones, que permite a los desarrolladores hacer un seguimiento de los cambios realizados en el código, volver a versiones anteriores si es necesario y colaborar eficazmente en los proyectos. Una de las herramientas más populares para el control de versiones es Git, un sistema de control de versiones distribuido (DVCS) que permite trabajar sin conexión y fusionar fácilmente el código de múltiples colaboradores [6].

En este contexto, GitHub surge como una plataforma basada en la nube que aprovecha las funcionalidades de Git. GitHub ofrece una interfaz fácil de usar para que los desarrolladores almacenen sus repositorios de código, realicen un seguimiento de los cambios y colaboren en los proyectos. Estas funcionalidades van más allá del simple control de versiones y ofrecen funciones como el seguimiento de incidencias, herramientas de gestión de proyectos y wikis para la documentación del código, todo ello centralizado en una única plataforma. Esto fomenta un entorno de colaboración en el que los desarrolladores pueden compartir código, hacer un seguimiento de los errores y trabajar juntos sin problemas [7].

1.4. Github Classroom

GitHub Classroom es una herramienta de enseñanza diseñada para facilitar la gestión y organización de aulas digitales y deberes. Permite crear tareas para estudiantes que pueden ser individuales o de grupo, establecer plazos de entrega y monitorear las asignaciones desde el panel del profesor. Además, GitHub Classroom ofrece múltiples funciones que simplifican tareas como dar retroalimentación, evaluar trabajos e integrar herramientas educativas existentes [8].

Con GitHub Classroom, se pueden establecer pruebas que califiquen automáticamente el trabajo de los estudiantes cada vez que ellos actualicen su repositorio de la asignación. Cuando se habilita la autoevaluación, GitHub Actions [9] ejecuta los comandos para su prueba de autoevaluación en un entorno Linux que contiene el código más reciente del

estudiante. GitHub Classroom crea los flujos de trabajo necesarios para GitHub Actions. Por lo que no es necesario saber trabajar con estas [10].

GitHub Classroom expone un cómodo panel de control, donde el profesorado puede configurar pruebas utilizando un *framework* de pruebas, ejecutando un comando personalizado, escribiendo pruebas de entrada/salida o combinar diferentes métodos de prueba [10].

1.5. GitHub API y GitHub API Classroom

La API de GitHub es un conjunto de herramientas y protocolos que permite a los desarrolladores interactuar con GitHub mediante programación. En esencia, proporciona una forma para que las aplicaciones de software se comuniquen con los servidores de GitHub y realicen diversas operaciones, como recuperar información del repositorio, gestionar incidencias y pull requests, e interactuar con cuentas de usuario [11].

El propósito de la API de GitHub es permitir a los desarrolladores crear aplicaciones e integraciones que aprovechen la funcionalidad y los datos de GitHub. Esto puede abarcar desde la automatización de tareas rutinarias, como la creación y gestión de repositorios, hasta la creación de herramientas sofisticadas que analicen el código, realicen un seguimiento del progreso del proyecto o integren GitHub con otros servicios.

Además, GitHub ofrece una API que facilita la comunicación con su herramienta GitHub Classroom. A diferencia de la API estándar de GitHub, la API de Classroom no permite modificar los datos de Classroom, pero sí proporciona acceso a la consulta de una amplia gama de información, como las tareas de una clase, las entregas realizadas y las calificaciones de cada entrega [12].

1.6. GitHub Apps

Aunque GitHub destaca en el control de versiones y el fomento de la colaboración, sus capacidades pueden mejorarse aún más a través de GitHub Apps. Se trata de herramientas personalizadas que se integran a la perfección con las funcionalidades de la plataforma a través de API y *webhooks*. Las GitHub Apps funcionan con un sistema de permisos de grano fino. Esto permite a los usuarios controlar meticulosamente los datos y acciones accesibles a la aplicación, reforzando la seguridad y la confianza del usuario. Además, GitHub Apps utiliza *tokens* de corta duración, lo que minimiza la superficie potencial de ataque [13].

Las funcionalidades de GitHub Apps son amplias. Pueden operar dentro de la propia plataforma, automatizando tareas como abrir incidencias (issues), comentar *pull requests* y gestionar proyectos. Aún más importante, las GitHub Apps pueden reaccionar a eventos que ocurren en la plataforma y desencadenar acciones fuera de ella. Esto abre las puertas a una amplia gama de integraciones y personalizaciones, fomentando un flujo de trabajo de desarrollo más ágil.

1.7. GitHub Actions

Aprovechando la ubicuidad de GitHub en el panorama del desarrollo de software, GitHub Actions ofrece un sólido motor de automatización del flujo de trabajo integrado directamente en la plataforma. Esto elimina la necesidad de servidores CI/CD externos o integraciones complejas. Los desarrolladores pueden definir flujos de trabajo personalizados activados por eventos como un push a una rama o la creación de un *pull request*. Estos flujos de trabajo orquestan una serie de acciones pre-construidas o scripts personalizados, automatizando tareas que antes eran manuales. Esto puede abarcar la construcción y prueba de código, la racionalización de los despliegues, e incluso la incorporación de análisis de revisión de código [9].

El verdadero poder de GitHub Actions reside en su extensibilidad. Existe un rico ecosistema de acciones pre-construidas, que cubren una amplia gama de funcionalidades - desde la construcción y pruebas con frameworks populares hasta el escaneo de seguridad y despliegue en varias plataformas en la nube. Esta amplia biblioteca permite a los desarrolladores elegir las acciones que mejor se adapten a las necesidades de su proyecto, en lugar de reinventar la rueda. Además, GitHub Actions permite a los desarrolladores crear y compartir acciones personalizadas, fomentando un entorno de colaboración en el que la comunidad puede contribuir y beneficiarse de flujos de trabajo reutilizables.

1.8. GoRace

GoRace [14] es una suite de aplicaciones web que permite crear automáticamente una solución web a medida para gamificar cualquier dominio. Se integra fácilmente con los sistemas, herramientas o sitios de una empresa, negocio, entorno educativo, etc. y aprovecha las actividades que realizan los participantes para proporcionarles una experiencia única que impulsa el compromiso, la motivación y el éxito de los resultados finales de acuerdo con los objetivos empresariales.

Inspirado en las dinámicas de juego, mecánicas y componentes de conocidos videojuegos, GoRace involucra a los participantes en una experiencia de gamificación basada en la narrativa. En una experiencia GoRace, los participantes se trasladan a un mundo virtual basado en la época de la mitología griega, donde participarán en una legendaria carrera olímpica. El objetivo de la carrera es alcanzar la inmortalidad llegando el primero a la meta. Para ello, los participantes deben cumplir sus actividades en la vida real para obtener distintas recompensas del juego, como puntos de distancia y monedas virtuales, e interactuar con otros elementos del mundo virtual, como una tienda virtual, para comprar y decidir cuándo y cómo utilizar una amplia variedad de objetos de fantasía que pueden tener efectos positivos o negativos en el progreso de un participante en particular o de todos los participantes, y que permiten a los participantes avanzar en la carrera.

1.9. GoRace API

GoRace se refiere a las herramientas externas a GoRace que se emplean en la estrategia de ludificación como recursos del dominio. Estos recursos deben conectarse con GoRace a través de su API. Los maestros de gamificación deben adaptar sus recursos de dominio para transmitir, mediante la GoRaceAPI, la información relacionada con las actividades de sus participantes conforme a la estrategia de ludificación. Mediante esta configuración, GoRace se integra con los recursos del dominio, permitiendo así que el dominio se conecte con GoRace para el envío y procesamiento automático de los logros de los participantes en sus actividades de la vida real.

Capítulo 2

Objetivos, estado del arte y metodología de trabajo

2.1. Objetivos

El presente trabajo tiene como objetivo principal explorar las oportunidades de implementación de la gamificación en el ámbito de la enseñanza de programación y otras especialidades de TI, centrándose específicamente en el uso de la plataforma GoRace. Se busca aprovechar las ventajas reportadas de la gamificación en este sector para mejorar la experiencia de aprendizaje y fomentar la participación activa de los estudiantes [15, 16].

Para lograr este propósito, se plantean los siguientes objetivos específicos:

1. **Desarrollar un recurso de dominio para GoRace:** Se diseñará y desarrollará un recurso específico para GoRace que permita la extracción de información de las pruebas automáticas creadas por los profesores utilizando Github Classroom. Este recurso facilitará la integración de GoRace en entornos educativos, aprovechando las herramientas familiares para el profesorado y simplificando la implementación de actividades gamificadas.
2. **Evaluar la efectividad y la facilidad de implementación del recurso:** Se realizarán pruebas piloto del recurso desarrollado en entornos educativos reales para evaluar su eficacia en la mejora del proceso de enseñanza-aprendizaje y su facilidad de uso para profesores y estudiantes.

Al alcanzar estos objetivos, se espera contribuir al avance de la gamificación en la enseñanza de programación y áreas afines, proporcionando a los educadores una herramienta práctica y efectiva para mejorar la motivación y el compromiso de los estudiantes en el proceso de aprendizaje.

2.2. Estado del arte

Actualmente, hay diversas herramientas diseñadas para ludificar el aprendizaje en el ámbito de la programación, tales como UDPiler [15] o Dungeons & Developers [17], entre otras. Sin embargo, hasta la fecha de este documento, no se ha identificado ninguna otra herramienta que utilice las ventajas de Github Classroom. Asimismo, no se ha hallado ningún recurso de dominio comparable para GoRace con una funcionalidad similar.

También hemos investigado otras herramientas de gamificación de ámbito más general, similares al papel de GoRace, como BuncBall [18] y MEdit4CEP-Gam [19], pero no hemos hallado soluciones que posibiliten enriquecer la experiencia de gamificación con pruebas automáticas de código, y mucho menos que permitan aprovechar las herramientas proporcionadas por Github Classroom para tal propósito.

Al comparar la solución propuesta en este documento con las alternativas, observamos que, aunque puede que no esté tan integrada en el apartado jugable de la experiencia como una solución específica como Dungeons & Developers, es definitivamente más versátil que sus competidores. Debido a la naturaleza genérica de las pruebas automáticas que soporta, puede ser adaptada para cualquier propósito, lo cual no se puede afirmar de Dungeons & Developers ni de UDPiler.

2.3. Metodología de trabajo

El desarrollo de este proyecto siguió una metodología inspirada en AGILE, caracterizada por su flexibilidad, naturaleza iterativa y énfasis en la comunicación y la retroalimentación periódicas. El uso de sprints cortos, de una semana de duración, facilitó ciclos de desarrollo rápidos y la reevaluación frecuente de los objetivos y avances del proyecto.

En este apartado se describen los procesos y prácticas específicos adoptados a lo largo del ciclo de vida del proyecto.

2.3.1. Planificación y Revisión de Sprints

Sprints Semanales

El proyecto se organizó en sprints semanales. Cada sprint comenzaba con una reunión entre el director del TFM y el estudiante, en la cual se realizaban las siguientes actividades:

1. **Revisión del Sprint Anterior:** Evaluación del trabajo completado durante el sprint anterior. Esto incluía evaluar los logros en comparación con los objetivos del sprint y discutir cualquier desafío encontrado.
2. **Establecimiento de Objetivos:** Definición de objetivos y planificación para el

próximo sprint. Esto implicaba identificar tareas clave, establecer prioridades y delinear entregables.

Diseño Iterativo y Toma de Decisiones

Durante estas reuniones, era habitual cuestionar y revisar decisiones de diseño anteriores. A medida que nuestra comprensión del dominio del proyecto evolucionaba, adaptábamos nuestro enfoque en consecuencia. Este proceso iterativo se vio facilitado por la estructura flexible del proyecto, que nos permitió sustituir las soluciones anteriores por otras mejoradas sin grandes trastornos.

2.3.2. Comunicación y Coordinación

Interacción con la UCA

La coordinación con los responsables de GoRace, es decir, el personal de la Universidad de Cádiz (UCA), o más en concreto con la doctora Mercedes Ruiz y el doctor Alejandro Calderón, se realizaba según fuera necesario. Estas interacciones no se programaban regularmente, sino que se iniciaban a través de hilos de correo electrónico cuando se estimaba necesario. Esta comunicación ad-hoc aseguraba que abordáramos los problemas de manera oportuna mientras manteníamos el impulso del proyecto y minimizábamos las molestias que podíamos causar al equipo de la UCA.

Ambos doctores mencionados anteriormente fueron muy veloces con sus respuestas y estuvieron siempre disponibles para ayudarnos a entender cualquier detalle de GoRace, solucionar cualquier problema y preparar las distintas carreras. Por todo esto quiero aprovechar la oportunidad para darles las gracias personalmente.

Falta de Soporte Directo de GitHub Classroom

Optamos por no contactar al equipo de GitHub Classroom en busca de soporte o cualquier tipo de ayuda, asumiendo que nuestras solicitudes tendrían escasas posibilidades de prosperar a corto plazo. Esta decisión hizo necesario un enfoque reactivo a los cambios implementados por el equipo de GitHub Classroom durante el desarrollo de este proyecto. Aunque la mayoría de estos cambios no entraban en conflicto con nuestro desarrollo, debemos mencionar que una alteración significativa que afectaba el formato de un fichero del cual dependíamos ocurrió justo antes de una prueba programada con alumnos reales. Esta incidencia estuvo a punto de imposibilitar dicha prueba, echando por tierra todo el trabajo dedicado a la preparación de la misma. Afortunadamente, se pudo encontrar un parche que nos permitió seguir adelante con la prueba sin que los alumnos percibieran error alguno.

2.3.3. Pruebas y evaluación de los usuarios

Los primeros usuarios y evaluadores de nuestro trabajo fueron los alumnos matriculados en la asignatura "Procesadores de Lenguajes" del grado de Ingeniería Informática de la Universidad de La Laguna (ULL). Estos estudiantes interactuaron con nuestra demo, que había sido integrada en sus tareas curriculares. Tras su participación, recogimos sus comentarios a través de una encuesta para evaluar la efectividad y usabilidad del proyecto.

La demostración se desarrolló alrededor de tareas pre-planeadas para los estudiantes, con énfasis en cumplir con plazos estrictos para no perturbar el cronograma del curso. La coordinación se facilitó por el hecho de que el director del proyecto también servía como profesor coordinador del curso, asegurando una integración y alineación fluidas con los requisitos académicos.

La prueba se desarrolló en torno a tareas que ya se habían planificadas de antemano para los estudiantes y formaban parte de su currículum. La preparación de la misma se hizo con énfasis en cumplir con plazos estrictos para no perturbar el cronograma del curso. Se dieron instrucciones al alumnado de como participar que pueden encontrarse en junto a la descripción de la tarea en los apuntes de la asignatura <https://ull-pl.vercel.app/labs/left-side>.

Capítulo 3

Diseño del sistema e implementación

El diseño de este proyecto surge de un proceso iterativo en el que se han ido descubriendo los requerimientos poco a poco y trabajando para satisfacerlos de la forma más efectiva posible. Esto se ha traducido en un diseño simple, que no intenta solucionar problemas teóricos que nunca se darán en la práctica, a la vez que ofrece una base extensible capaz de adaptarse a cualquier nuevo requerimiento que pudiera surgir para quien desee extender su funcionalidad. De esta manera, hemos logrado crear una aplicación sencilla de operar para los usuarios y fácil de gestionar para sus administradores.

En este capítulo nos adentraremos en el diseño de GH2GR, comenzando con una visión general de todo el sistema, para luego analizar sus componentes en detalle.

3.1. Traducción entre GoRace y Github Classroom

GH2GR es una aplicación que se encarga de trasladar los eventos ocurridos en una plataforma a otra. Así pues, lo primero que debemos discutir sobre su diseño es cómo traduce e interpreta los recursos y eventos de las plataformas con las que interactúa y cómo realiza la traducción de una a la otra. Para esto, necesitamos entender los términos que utiliza cada plataforma.

En GoRace, cada experiencia didáctica, ya sea una asignatura, un curso, un torneo, etc., se empaqueta en lo que ellos denominan una *carrera*. Las carreras son contenedores de nivel superior y están completamente aisladas entre sí. Dentro de las carreras, los participantes pueden completar actividades para avanzar. GoRace evalúa el desempeño de cada participante en una actividad mediante los valores obtenidos en las *variables* de dicha *actividad*. GoRace descompone cada actividad en variables, que son básicamente contenedores de puntuaciones con un rango posible de valores. Por ejemplo, podríamos tener una variable que admite valores del 0 al 10 y otra del 0 al 100. Gracias a estos rangos, es posible determinar cuán satisfactorio es el valor almacenado en la variable, pues no representa lo mismo un 10 en la primera variable, que indicaría un éxito absoluto, que en la segunda variable, donde sería un resultado paupérrimo.

Por otro lado, GitHub Classroom encapsula cada experiencia en lo que llama *classrooms*







[MAN] manyargs	 
[OBJ] objects	 
[FUND] operations	 
[FUND] recursive	 
[FUND] scope	 
[FUND] string	 

Figura 3.1: Ejemplo de pruebas automatizadas de evaluación agrupadas en Github Classroom. En la imagen se muestran tres grupos: FUND, OBJ y MAN.

o aulas. A su vez, estas clases contienen *assignments* o tareas que, como su nombre indica, son las distintas tareas que el docente asigna a los alumnos. Dentro de cada tarea, el docente puede definir cuantas pruebas automatizadas de evaluación considere necesarias, asignando una puntuación a cada una de ellas. Posteriormente, cuando los alumnos hagan sus entregas, GitHub Classroom sumará las puntuaciones de cada prueba superada por el alumno y le presentará el total junto al máximo posible si hubiera superado todas las pruebas.

Ahora, al trasladar el avance de los alumnos desde GitHub Classroom hasta GoRace, encontramos natural traducir las *classrooms* en carreras y los *assignments* en actividades. Efectivamente, esta fue la decisión tomada, ya que consideramos que de esta forma resultaría más intuitivo para los usuarios. No obstante, aunque podría parecer tentador relacionar las pruebas automatizadas de evaluación con las variables de GoRace, decidimos optar por otra alternativa. Proceder de esa manera resultaría en una pérdida de expresividad, ya que las variables de GoRace admiten un rango de valores, mientras que las pruebas automatizadas son binarias; estas pruebas solo pueden ser superadas o no serlo, y proporcionarán su puntuación únicamente si son superadas.

Para resolver este dilema, GH2GR define un método que permite crear múltiples agrupaciones de pruebas automatizadas de evaluación dentro de una misma actividad a través de un prefijo en el nombre de la prueba con el formato de [Nombre Variable], como se puede ver en la figura 3.1. Son estos grupos entonces los que se asignan a las variables de GoRace. Cada grupo tiene un rango definido de $[0, N]$, siendo N la suma de las puntuaciones de cada una de las pruebas del grupo. El valor que cada alumno obtendrá en cada grupo será la suma de las puntuaciones de todas las pruebas que haya superado.

Ahora que hemos expresado los grupos en los mismos términos que las variables de GoRace, es sencillo ver cómo es posible una traducción sin pérdida de expresividad. Pues tan solo requerimos hacer una regla de tres para transformar el valor del rango del grupo de pruebas al de la variable de GoRace. El único problema de este acercamiento es que, a la fecha de la escritura de este documento, no es posible extraer usando la API de GoRace cuál es el rango de la variable. Para bordear este problema, GH2GR asume que las variables de GoRace tienen un rango de cero a cien $[0, 100]$. Si bien con este compromiso se pierde un poco de flexibilidad, consideramos que ofrece expresividad

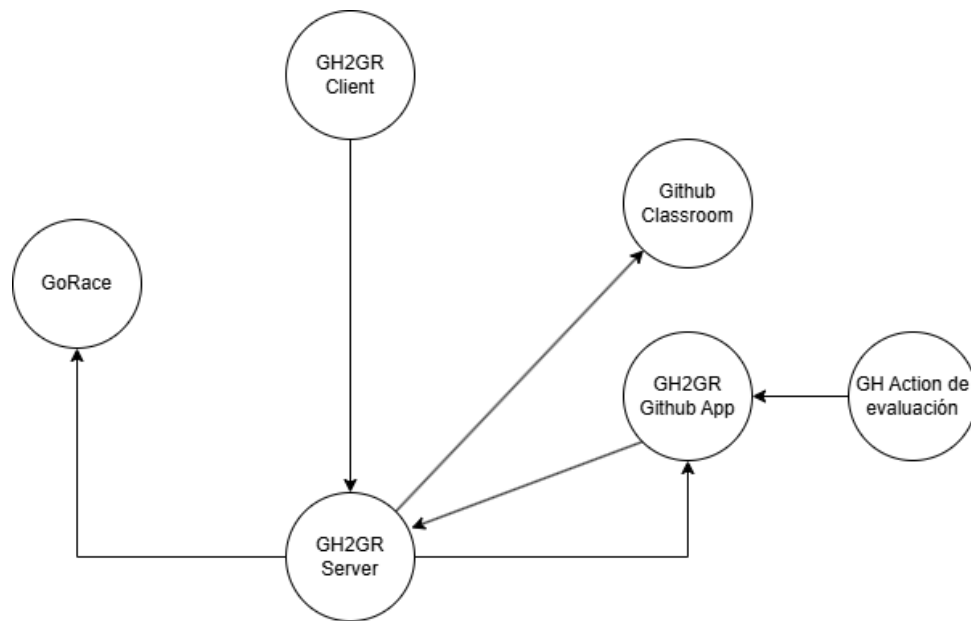


Figura 3.2: Diagrama que muestra los componentes a alto nivel que interactúan en el funcionamiento de GH2GR. Los círculos representan componentes, mientras que las flechas indican que un componente (el que origina la flecha) puede iniciar comunicación con otro (al que apunta la flecha).

suficiente para cualquier aplicación.

3.2. Diseño a alto nivel del sistema

En una experiencia que utiliza GH2GR podemos encontrar los siguientes componentes base:

- **GH2GR Server:** Es el componente principal de GH2GR. Este componente se encarga de extraer los resultados desde GitHub y brindarlos a GoRace. Adicionalmente, ofrece una API para su gestión.
- **GitHub App:** Este constructo nos proporciona las credenciales para interactuar con la API de Github y así poder acceder a los recursos que se esconden detrás de esta.
- **GitHub Classroom:** Alberga toda la información relativa a las tareas, agrupaciones de alumnos, etc. Desde aquí el profesorado crea las distintas tareas y define las pruebas automatizadas de evaluación.
- **GitHub Action de evaluación:** Esta *Action* se ejecuta automáticamente cuando un alumno publica algún cambio en su repositorio, ejecutando las pruebas de automatizadas de evaluación definidas por el profesor y ofreciendo los resultados en un formato de fácil lectura para GH2GR Server.
- **GH2GR Client:** Utilidad de línea de comandos que permite al profesor gestionar la integración por medio de la API de GH2GR Server y generar el *workflow* [9] para la Github Action de evaluación.

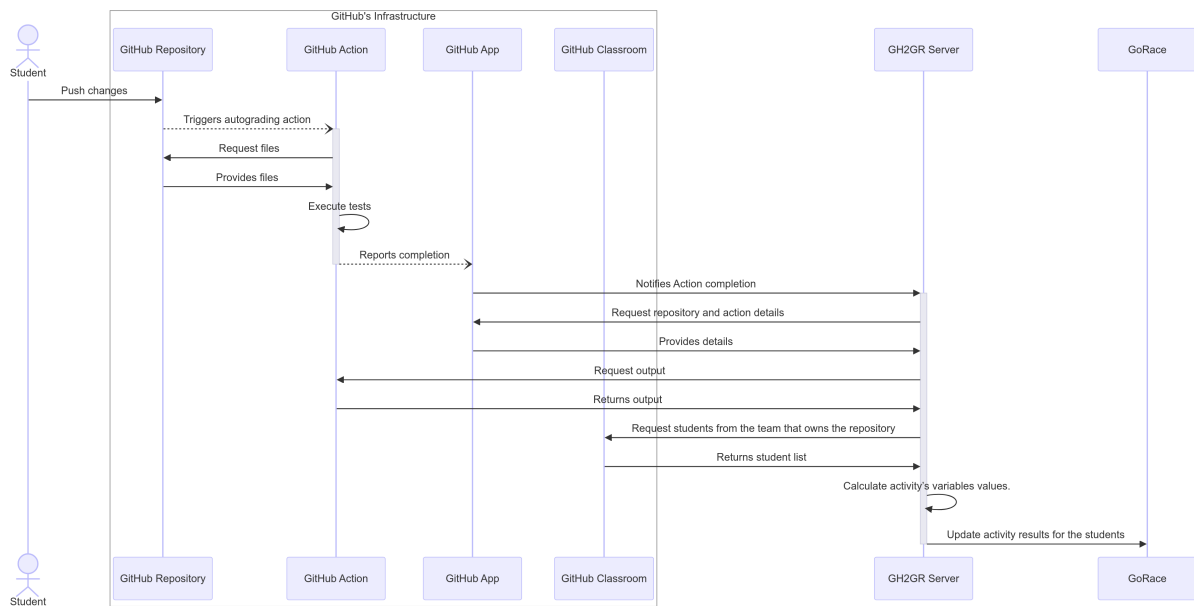


Figura 3.3: Diagrama de flujo de los eventos que ocurren tras la realización un *push*. Leyenda: La figura de un hombre representa un actor, un rectángulo representan un componente. Las flechas solidas representan una operación síncrona, las punteadas una asíncrona. El ensañamiento de la línea de la vida significa que el componente está trabajando en el periodo ensanchado.

- **GoRace:** Es la plataforma de ludificación. Recibe las actualizaciones de GH2GR Server y las utiliza para avanzar la experiencia lúdica según los diseños del profesorado.

Para entender mejor cual el rol de cada una de las partes, es conveniente explicar que ocurre desde que un alumno realiza un cambio a su repositorio hasta que el resultado de este se ve reflejado en GoRace (véase la Figura 3.3):

Lo primero que ocurre cuando un alumno sube sus cambios a su repositorio en GitHub es que la plataforma ejecuta la Github Action de evaluación. Esta, a su vez,

1. Lee el fichero de configuración que creó GitHub Classroom cuando el profesor definió las pruebas automáticas de evaluación.
2. Tras consultar el fichero, la Action ejecuta las pruebas una por una y produce como salida un JSON detallando cada prueba ejecutada, su puntuación de ser superada y si la prueba en cuestión fue superada o no.
3. Además, esta salida añade algunos detalles como la carrera a la que pertenece, la actividad y el identificador del *assignment* de GitHub Classroom al que pertenece.

La GitHub App está configurada de tal forma que monitorea la ejecución de las Github Actions. Así, a través de un *webhook*, es decir, una llamada HTTP, es capaz de avisar a GH2GR Server de este evento, otorgándole, además, contexto como el identificador de la ejecución y el repositorio donde ocurrió.

Con esa información, el servidor

1. Utiliza sus credenciales de la App de GitHub para acceder a la salida de la Github Action.
2. Una vez ha leído el JSON, accede a la API de GitHub Classroom y busca a todos los alumnos relacionados con el repositorio, ya que GitHub Classroom permite que las actividades se realicen en equipo y, en tal caso, asigna un solo repositorio a todos los miembros.
3. Luego, agrupa las pruebas en las distintas variables a partir del texto entre corchetes que se usa como prefijo en cada prueba.
4. Una vez agrupadas, se calcula la puntuación máxima de cada grupo sumando todos los puntos que pueden dar las pruebas que los componen y cuántos se han obtenido por pruebas superadas.
5. Es ahora cuando el servidor aplica una regla de tres para ajustar el resultado al rango de las variables en GoRace, que, como se comentó antes, se espera que sea de cero a cien.
6. Finalmente, el GH2GR Server actualiza el resultado de la actividad con los valores de las variables obtenidos para todos los alumnos partícipes del repositorio.

3.3. Sobre la GitHub App

Como ya se comentó con anterioridad, la GitHub App no es una aplicación per se, pues no es algo que podemos programar. En su lugar, es un constructo que ha creado GitHub para agrupar y dar nombre a los permisos y credenciales que necesita una aplicación para interactuar con la API de GitHub y para escuchar los eventos que puedan surgir en la plataforma.

A fin de que la aplicación esté autorizada para acceder a los recursos (ej. Repositorios, ejecuciones de GitHub Actions, etc.) necesarios para su correcto funcionamiento, la GitHub App debe ser “instalada”, utilizando la terminología de la propia GitHub, en la organización en GitHub utilizada para crear el *classroom*. En términos más comunes, la GitHub App tiene que ser autorizada para acceder a los recursos que se esconden bajo el manto de la organización. Véase la Figura 3.4.

Como buena práctica de seguridad, se ha tratado de reducir los permisos que solicita la GitHub App asociada a GH2GR al mínimo imprescindible. Es por esto que nuestra GitHub App no requiere de ningún permiso de escritura y solo requiere acceso de lectura de los siguientes permisos:

- **Metadata:** Permite listar los repositorios, colaboradores y otros datos superficiales. Es un permiso básico que GitHub obliga a todas las *apps* a solicitar.
- **Actions:** Permite acceder a información de las GitHub Actions: cuáles hay en un repositorio, cuándo se lanzan, cuándo terminan, etc. No permite acceder a la salida de una ejecución de una GitHub Action en particular.

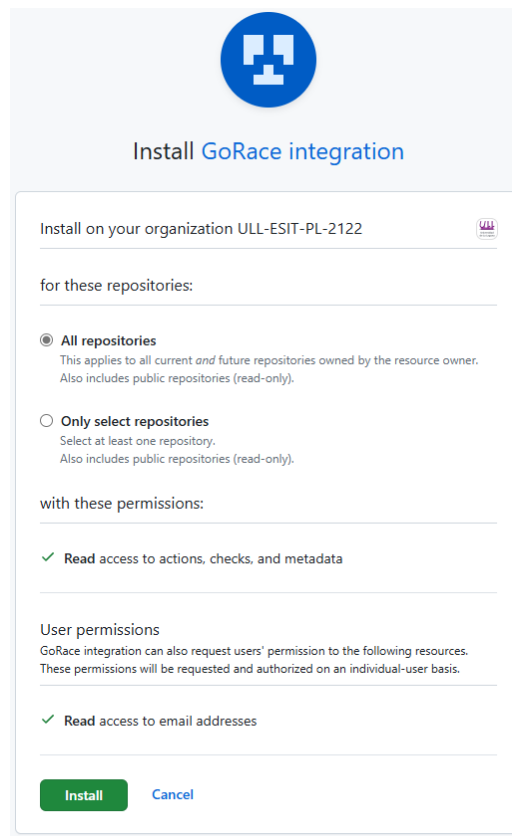


Figura 3.4: Pantalla de instalación de la GitHub App en una organización.

- **Checks:** Permite acceder a la salida de las GitHub Actions.

Además de permitir acceder a la API de GitHub, la GitHub App nos permite ser avisados de eventos, como, por ejemplo, cuando se completa la ejecución de una GitHub Action, a través de un mecanismo de *webhook*. Un *webhook* es un mecanismo de mensajería en tiempo real empleado para automatizar la comunicación entre aplicaciones. Cuando se produce un evento predefinido en un sistema de origen (por ejemplo, una nueva entrada de datos o la finalización de una tarea), se activa una solicitud HTTP que envía los datos relevantes (*payload*) a una URL designada en un sistema receptor. Esto permite que las aplicaciones reaccionen rápidamente a los eventos sin necesidad de un sondeo constante, lo que fomenta un flujo de trabajo más eficiente [20].

3.4. Sobre la GitHub Action

La GitHub Action es un programa autocontenido que se ejecuta en la infraestructura de GitHub cuando un alumno sube un cambio al repositorio. Este programa, de hecho, es un *fork* del que ya utilizaba la propia GitHub Classroom [21].

La Action está escrita en TypeScript [22] y está diseñada para ejecutarse sobre NodeJS [23]. Tras leer el fichero *autograding.json*, la Action ejecuta las pruebas de forma secuencial, ejecutando el comando que se especifica en cada una y comparando la salida obtenida con la esperada para superar la prueba. Las comprobaciones que se pueden

hacer son variadas, pues se puede:

- observar el código de salida [24].
- comparar la salida estándar esperando una salida en concreto.
- que la salida contenga un fragmento de texto concreto.
- que la salida satisfaga una expresión regular.

Tras ejecutar las pruebas y determinar su resultado, la GitHub Action produce como salida un documento JSON con la siguiente estructura:

- **activity**: Nombre de la actividad en GoRace a la que alude el repositorio.
- **assignmentId**: El identificador del *assignment* en GitHub Classroom. Es un valor numérico entero.
- **race**: El nombre de la carrera en GoRace.
- **results**: Vector conteniendo los resultados de cada una de las pruebas.
 - **name**: El nombre de la prueba automatizada de evaluación.
 - **failed**: Si la prueba se ha pasado con éxito o no. Valor booleano, verdadero si la prueba ha fallado o falso si se ha superado con éxito.
 - **points**: Cuántos puntos vale la prueba automatizada de evaluación. Valor numérico entero.

Se ha escogido utilizar JSON porque es un formato de datos ligero y de texto plano, lo que facilita su lectura y escritura tanto para humanos como para máquinas. Su estructura, basada en pares clave-valor y vectores, es intuitiva y directa, lo que simplifica la serialización y deserialización. Estas características han posibilitado que JSON se transforme en un lenguaje omnipresente que cuenta con utilidades para ser procesado en casi cualquier lenguaje de programación [25]. Estos puntos hacen que JSON sea un lenguaje ideal para la tarea que tenemos entre manos, pues nos permite fácilmente revisar la salida del programa para detectar cualquier posible fallo a simple vista y no nos limita en las tecnologías que podemos usar en el resto del sistema. Además, el soporte para JSON se encuentra integrado de forma nativa en NodeJS [26].

3.5. GH2GR Server en detalle

El componente de servidor de GH2GR es el corazón de la aplicación y donde se concentra la mayor parte del trabajo. Es responsable de hacer la transformación real de la información obtenida de GitHub y traducirla para que GoRace la pueda entender.

A este componente lo denominamos servidor porque su función principal es servir como un servidor HTTP que recibe los eventos del *webhook* de la GitHub App y presenta una API de gestión.

El *webhook* está diseñado para seguir la especificación de GitHub. Es decir, se trata de un *endpoint*, es decir, una URL, que recibe peticiones HTTP de tipo POST con la información del evento en el cuerpo de la petición codificada en formato JSON.

Adicionalmente, el servidor espera que en la petición HTTP se encuentre la cabecera "X-Hub-Signature-256". Esta cabecera debe contener una firma HMAC [27] utilizando el hash SHA-256 [28] del contenido del cuerpo de la petición firmado con un secreto establecido en la configuración de GitHub y proporcionado a GH2GR Server a la hora de su arranque.

Si el servidor no es capaz de verificar la validez de la firma de la petición, la petición al completo será descartada y el servidor retornará una respuesta inmediata con el código HTTP 400 *Bad Request*, indicando que la petición es inválida y el servidor se niega a procesarla [29]. Esto se hace para evitar que un atacante que haya descubierto la URL del *endpoint* sea capaz de engañar al sistema enviando eventos falsos.

Lo anterior resume el diseño del *webhook*, pero se puede encontrar más información sobre la información que recibe el *webhook* en la documentación de GitHub del evento de *workflow_run* [30] y sobre la validación en la sección de la documentación dedicada a la validación del *webhook* [31].

En cuanto a la API de gestión, hablaremos de ella más en detalle en el apartado 3.5.3.

3.5.1. Estructura del código

GH2GR Server está escrito en el lenguaje de programación Go [32]. Esto afecta profundamente a cómo está estructurado, pues la estructura del proyecto está en parte moldeada por las funcionalidades, limitaciones y el paradigma del lenguaje, así como por lo que le resulta idiomático.

Go, también conocido como Golang, es un lenguaje de programación compilado de tipo estático diseñado por Google. Notable por su simplicidad y eficiencia, Go proporciona un potente modelo de concurrencia a través de *goroutines* y canales, lo que permite el desarrollo de aplicaciones altamente concurrentes y paralelas [33]. Además, cuenta con un colector de basura que simplifica la gestión de memoria [34]. El lenguaje fue elegido para este proyecto por las siguientes razones (sin orden específico):

- su simplicidad,
- la familiaridad del alumno con este,
- su popularidad en herramientas que interactúan con GitHub, incluyendo la propia GitHub CLI [35],
- la multitud de librerías disponibles para este fin, y
- su enfoque en concurrencia que permite desarrollar un servidor eficiente capaz de gestionar múltiples peticiones de forma simultánea de manera sencilla.

Go es un lenguaje ligeramente estricto a la hora de organizar código. En él, múltiples archivos en una misma carpeta conforman un *paquete*. Dentro de un mismo paquete no existe una barrera que separe el código privado del público, pudiendo cualquier archivo acceder a todas las funciones de cualquier otro. No obstante, cuando hablamos de acceder a lo definido en un paquete externo, Go sí ofrece una separación clara entre lo público, al cual podremos acceder, y lo privado, cuyo acceso nos será prohibido por el compilador. Para acceder a la interfaz pública de otro paquete, tendremos que importarlo, es decir, declarar que queremos hacer uso del mismo, marcándolo como una dependencia. Aquí encontramos una de las peculiaridades del lenguaje, pues *el grafo de dependencias de paquetes debe ser acíclico*. En otras palabras, un paquete no puede importar ningún paquete que intente importarlo, o que sus dependencias intenten importarlo.

Finalmente, debemos discutir el paradigma de programación del lenguaje, ya que, por supuesto, esto también afecta a cómo estructuramos nuestro código. En primer lugar sigue el paradigma CSP (procesos secuenciales que se comunican [36]) que facilita la ejecución concurrente. En segundo lugar es complicado hacer un corte claro y fino del paradigma al que pertenece. Go es principalmente un lenguaje imperativo con elementos de orientación a objetos sin llegar a serlo puramente [37], y también cuenta con ciertos elementos más propios de la programación funcional [38].

Ahora que hemos terminado de discutir cómo afecta el lenguaje a la organización del código, podemos pasar a hablar de los principios que han guiado la estructura de este. Dicha estructura está fuertemente inspirada por el modelo MVC [39], contando con un controlador que responde a los eventos externos y un modelo que engloba la lógica de negocio. Además, tiene una influencia de la arquitectura hexagonal [40], ya que se han diseñado capas que aíslan a la lógica de negocio y evitan que dependa de recursos externos, aprovechando al máximo el Principio de Inversión de la Dependencia [41].

El paquete principal de GH2GR es el paquete **models**. Este paquete contiene los tipos abstractos de datos que se utilizan para contener la información en toda la aplicación, el código que se encarga de validar sus valores y las operaciones sobre estos datos que no requieren de ningún recurso externo.

A continuación, nos encontramos con el paquete **services**. Este paquete alberga las operaciones o procesos que debe realizar la aplicación. Por ejemplo, si decimos que GH2GR permite actualizar la información que tiene el servidor sobre una carrera, en este paquete nos encontraremos una función que encapsula esta funcionalidad. Esta función se encargará de revisar que la carrera en cuestión existe, que el docente que está intentando hacer la modificación es dueño de la carrera y de asegurarse de que la modificación es válida. No obstante, este paquete también se puede ver como una extensión de *models* pues contiene solo lógica de negocio. Tampoco se le permite a este paquete tener dependencias externas más allá de *models*, por lo que se vale de interfaces definidas por sí mismo o por *models* para interactuar con recursos externos. El motivo de que este sea un paquete separado de *models* es tan sencillo como el deseo de evitar que *models* acabe albergando demasiado código. Esta es una ocurrencia tan habitual que a esta pequeña variación de MVC se le ha apodado como MVCS [42].

El siguiente gran paquete del que tenemos que hablar es el paquete **controllers**. Podemos ver este paquete como una capa de traducción entre nuestra API HTTP (apartado

3.5.3) y las operaciones que tenemos definidas en *services*. Así pues, en este paquete hallaremos todo el código responsable de manejar rutas, verbos y cabeceras HTTP, así como de la serialización y deserialización de los distintos documentos JSON que representan la información de nuestros modelos.

Los paquetes que restan son paquetes más utilitarios, con un nombre que identifica el propósito de cada paquete, siguiendo las recomendaciones de estilo de Go [43]. Ejemplos de estos son el paquete **database**, que tiene el código que interactúa directamente con la base de datos; el paquete **cli**, que contiene utilidades para construir la interfaz de línea de comandos; el paquete **gorace**, que traduce las partes relevantes de la API HTTP de GoRace a una API Go; el paquete **github**, que cumple una función similar al paquete *gorace*, pero para GitHub y GitHub Classroom, etc.

3.5.2. Los tipos abstractos de datos principales

Ahora examinaremos cuáles son los ADT principales del programa, que sirven para almacenar y transportar la información más importante entre las distintas capas de la aplicación.

El primero que nos encontramos es el que representa la información que necesitamos saber sobre la carrera en GoRace, adecuadamente llamado *Race*. Este tipo cuenta con los siguientes campos:

- **ID**: Tipo entero. Identificador interno de la carrera.
- **JWT**: Token de acceso para la API de GoRace.
- **Name**: Tipo texto. El nombre de la carrera.

El primer campo es un simple identificador numérico, que sirve para distinguir de forma inequívoca una carrera en cuestión del resto. Por su lado, el campo de nombre es un simple apodo para la carrera, pensado para que los usuarios puedan identificar rápidamente la carrera sin tener que lidiar con identificadores numéricos. Por último, el campo de JWT contiene el token proporcionado por GoRace que autoriza a interactuar con la carrera por medio de su API.

Luego tenemos el ADT de *User*, que refleja a un usuario directo de GH2GR-Server, es decir, un docente. Los campos que lo componen son los siguientes:

- **ID**: Tipo entero. Identificador interno para el usuario.
- **Username**: Tipo texto. El nombre del usuario.
- **GithubToken**: El token de GitHub del usuario.

El campo de ID cumple el mismo rol que el que ya hemos visto. El *Username* es el nombre de usuario único con el cual se puede identificar al docente. Finalmente, el *GithubToken* es un token personal de GitHub proporcionado por el docente. Este dato es necesario ya

que la API de GitHub Classroom no puede ser accedida por GitHub Apps propiamente, sino que requiere de un token generado por el usuario para la aplicación [12]. Así pues, como este dato está asociado estrictamente a un usuario, se almacena con los datos propios de este.

Ahora echemos un vistazo a los ADT que representan los datos que esperamos obtener de GitHub. Primero, tenemos el tipo `GithubInput`, que cuenta con los siguientes campos:

- **Activity:** Tipo texto. El nombre de la actividad correspondiente en GoRace.
- **Race:** Tipo texto. El nombre de la carrera correspondiente en GoRace.
- **TestResults:** Tipo vector de `ScoreInput`. Los distintos resultados de las pruebas de evaluación automatizadas.
- **AssignmentId:** Tipo entero. El identificador de la tarea en GitHub Classroom.

Como podemos comprobar, esta es la entrada esperada cuando obtenemos la salida de la GitHub Action. Por tanto, los datos que aquí tenemos son los necesarios para ponernos en contexto de qué tarea estamos puntuando, más los resultados que nos proporcionan los valores para calcular la puntuación. Es propio que ahora revisemos el tipo `ScoreInput`, que contiene los siguientes campos:

- **Name:** Tipo texto. El nombre de una prueba de evaluación automatizada.
- **Points:** Tipo entero sin signo. La cantidad de puntos que proporciona la prueba.
- **Failed:** Tipo booleano. Representa si la prueba ha sido superada (valor falso) o fallida (valor verdadero).

Ahora solo nos resta explorar un pequeño ADT llamado `GithubUserMapping`, que viene a solventar el problema de que

- GitHub principalmente nos habla de los usuarios usando nombres de usuario, mientras que
- GoRace lo hace hablando de correos electrónicos.

Y aunque sería posible preguntar por los correos electrónicos a GitHub, esto representaría la necesidad de que los usuarios autorizaran personalmente a la GitHub App a acceder a este detalle y, aun así, sería problemático, pues una cuenta de GitHub permite tener múltiples correos asociados [44] y habría que determinar cuál es el adecuado.

Para esquivar este problema, este tipo representa una asociación que ha hecho el docente de cada nombre de usuario de GitHub del alumnado con un correo electrónico que se usará en GoRace. Por tanto, este tipo se compone de esos dos campos:

- **Username:** Nombre de usuario en GitHub.
- **Email:** Correo electrónico a usar en GoRace.

3.5.3. La API de gestión

La API de gestión permite a un cliente con las credenciales de un docente añadir **carreras** bajo su cargo y listar, actualizar y eliminar las carreras que tiene asociadas. Además, le permite crear las **traducciones** de los nombres de usuario de GitHub de los alumnos a correos electrónicos que usar en GoRace. Por último, también le permite consultar y modificar sus datos de usuario en GH2GR-Server.

Notablemente, están exentas de la API las operaciones de crear o eliminar usuarios docentes, pues esto se considera una operación de administrador y, por tanto, solo se pueden realizar a través de la interfaz de línea de comandos de GH2GR-Server directamente.

La API tiene un diseño HTTP REST-like (similar a REST sin llegar a serlo). Pues tal como pide REST [45], la API tiene un diseño cliente/servidor, carece de estado, puede ser *cacheable* y puede ser utilizado de forma transparente a través de distintas capas como cachés, servidores proxy o balanceadores de carga.

El factor que hace que la API sea REST-like y no completamente REST se encuentra en el uso de la denominada “interfaz uniforme”. Si bien la API cumple con ella en la mayor parte de su extensión, estando orientada a recursos identificados por URI, utilizando adecuadamente los verbos HTTP y expresando los errores de forma semántica utilizando los mecanismos de HTTP, existen unas pocas operaciones que se presentan en un estilo más propio del RPC, pues hacía más claro su uso. Además, esta funcionalidad ni siquiera se publicita mediante hipermedia, pues se decidió en contra de usar esta para mantener una API sencilla.

Aunque se haya optado por no hacer una API completamente REST, sí que se han hecho esfuerzos para que la API sea legible tanto por humanos como por máquinas. A fe de esto, todos los recursos se han modelado utilizando JSON-Schema [46].

Además, todos los errores son representados con el código de estado HTTP más adecuado según la RFC 9110 [47], acompañado de un cuerpo que detalla el problema en formato de *problem details*, que ofrecen una forma estándar de especificar de forma inequívoca un problema y ofrecer toda la información sobre este que se considere relevante en un formato legible por personas y consumible por máquinas [48].

Como culmen de este esfuerzo, se ha documentado la totalidad de la API en el formato OpenAPI 3.1, un formato que permite tanto a humanos como a ordenadores descubrir y comprender las capacidades de un servicio sin necesidad de acceder al código fuente, documentación adicional o inspección del tráfico de red. Cuando se define correctamente a través de OpenAPI, un consumidor puede entender e interactuar con el servicio remoto con una cantidad mínima de lógica de implementación, de forma similar a lo que las descripciones de interfaces han hecho por la programación de bajo nivel [49].

En otras palabras, la API está documentada en un formato que permite la generación automática tanto de páginas de documentación para el consumo humano, como de SDK para interactuar con ella directamente desde el código de cualquier aplicación cliente. Esta especificación en formato OpenAPI se encuentra en el anexo A.1 de este documento.

Dado que ya se encuentra anexa a este documento una documentación completa de la API, no se explorará más sobre ella en este apartado. Aunque sí se mencionará que su diseño fue altamente influido por ADT, descrito en el apartado anterior, que sirvió como la base para los recursos sobre los que se modela la API.

3.5.4. La base de datos

Como ya se comentó anteriormente, en el diseño de GH2GR-Server, la base de datos no es un componente irremplazable y, de hecho, puede ser sustituida por cualquier otro mecanismo con una cantidad moderada y bien acotada de trabajo. No obstante, es cierto que el rol que cumple es crucial: almacenar de forma permanente los datos y permitir recuperarlos de forma eficiente. Para este trabajo, GH2GR-Server solo se ha dotado de la capacidad de usar un sistema gestor de bases de datos.

El sistema gestor de bases de datos que utiliza la aplicación es SQLite en su versión 3. El motivo de esta elección es que, al ser un DBMS que está integrado en el propio binario de la aplicación y se ejecuta en el mismo proceso que esta, solo necesita como añadido un lugar donde guardar un fichero [50]. Esto la convierte en la candidata ideal para mantener la línea de diseño orientada a la simplicidad a la hora de administrar y poner en funcionamiento GH2GR-Server. Además de esta ya deseable característica, SQLite nos ofrece un buen rendimiento y transacciones ACID sobre una interfaz SQL [50], que nos permite modelar cómodamente las relaciones en los datos que almacenamos [51].

3.5.5. El método de construcción y despliegue

Como ya se comentó anteriormente, GH2GR-Server está escrito en Go, un lenguaje de programación famoso por su sencilla compilación y por su habilidad de producir un binario sin dependencias externas, facilitando la distribución [52]. No obstante, GH2GR-Server utiliza las funcionalidades de CGo, una utilidad que permite llamar a los compiladores de C para utilizar código programado en este lenguaje [53]. Dado que utilizamos estos compiladores, necesitamos que estén presentes a la hora de construir nuestro programa y, si deseamos compilar para otra plataforma, también necesitaremos tener las capacidades para compilar este lenguaje para la máquina objetivo.

Para reducir la cantidad de software que es necesario para construir el programa y además contar con un método de despliegue estándar, se ha decidido utilizar contenedores OCI, también conocidos por muchos como contenedores Docker [54]. Estos permiten definir un entorno replicable, con todas las dependencias, que posibilita la construcción con un solo comando, produciendo no solo el binario, sino que también lo acompaña con todo el entorno que necesita para ser ejecutado correctamente dentro de un *runtime* de contenedores Linux OCI. Esto significa que puede ejecutarse no solo en casi cualquier máquina Linux [55, 56], sino también en máquinas Windows y Mac OS [57, 58, 59].

Las ventajas que nos traen los contenedores no solo se limitan a lo anterior, y es que, usando las palabras del *Docker Captain* Adrian Mouat [60]:

Los desarrolladores pueden crear software localmente, sabiendo que se ejecutará de forma idéntica independientemente del entorno del host, ya sea un rack en el departamento de TI, el portátil de un usuario o un clúster en la nube. Los ingenieros de operaciones pueden concentrarse en las redes, los recursos y el tiempo de actividad, y pasar menos tiempo configurando entornos y luchando contra las dependencias del sistema.

Los contenedores son una encapsulación de una aplicación con sus dependencias. A primera vista, parecen simplemente una forma ligera de máquinas virtuales (VM): al igual que una VM, un contenedor contiene una instancia aislada de un sistema operativo (SO), que podemos utilizar para ejecutar aplicaciones.

Sin embargo, los contenedores tienen varias ventajas que permiten casos de uso que son difíciles o imposibles con las máquinas virtuales tradicionales:

- *Los contenedores comparten recursos con el sistema operativo anfitrión, lo que los hace un orden de magnitud más eficientes. Los contenedores pueden iniciarse y detenerse en una fracción de segundo. Las aplicaciones que se ejecutan en contenedores incurren en una sobrecarga mínima o nula en comparación con las aplicaciones que se ejecutan de forma nativa en el SO anfitrión.*
- *La portabilidad de los contenedores tiene el potencial de eliminar toda una clase de errores causados por cambios sutiles en el entorno de ejecución; incluso podría poner fin al viejo refrán de los desarrolladores de "ipero si funciona en mi máquina!".*
- *La naturaleza ligera de los contenedores permite a los desarrolladores ejecutar docenas de contenedores al mismo tiempo, lo que hace posible emular un sistema distribuido listo para la producción. Los ingenieros de operaciones pueden ejecutar muchos más contenedores en una sola máquina host que utilizando solo máquinas virtuales.*
- *Los contenedores también tienen ventajas para los usuarios finales y los desarrolladores, aparte de la implantación en la nube. Los usuarios pueden descargar y ejecutar aplicaciones complejas sin necesidad de dedicar horas a cuestiones de configuración e instalación o preocuparse por los cambios necesarios en su sistema. A su vez, los desarrolladores de dichas aplicaciones pueden evitar preocuparse por las diferencias en los entornos de usuario y la disponibilidad de dependencias.*

(...) the purpose of a container is to make applications portable and self-contained.

Por Adrian Mouat en Using Docker [61], traducción propia.

3.5.6. Sobre Gh2GR Client

El cliente de GH2GR es una pequeña herramienta de línea de comando que permite a un docente interactuar de forma cómoda con el servidor de GH2GR y realizar todas las

operaciones necesarias para preparar la integración entre una tarea en GitHub Classroom y GoRace.

Se escogió que este cliente fuera una herramienta de terminal y no contara con interfaz gráfica, porque se llegó a la opinión de que las tareas que había que realizar con el cliente se desempeñaban de forma más eficiente a través de una interfaz de línea de comandos, abriendo, además, la posibilidad de ser usado en *scripts* de *shell*. Otro factor importante es que, al estar orientada esta herramienta a docentes de temática TI, se espera que sus usuarios cuenten con un grado de familiaridad con la línea de comandos.

A nivel de su desarrollo, la aplicación cliente está programada en Go, al igual que el servidor, y, de hecho, se encuentra en un mono-repositorio con él [62], por lo que toma prestadas partes del código de GH2GR Server. Más allá de esto, la aplicación está diseñada alrededor de subcomandos, cada uno de ellos permitiendo realizar una de las funciones de la aplicación. El código que lo hace posible se encuentra en un archivo dedicado a dicho subcomando, separándose el código que es común entre subcomandos en pequeños paquetes utilitarios que facilitan su uso en múltiples lugares, siguiendo el principio DRY [63].

Capítulo 4

Modo de empleo

En este capítulo, ofrecemos una guía completa para configurar y desplegar la integración entre GitHub Classroom y GoRace.

El capítulo está estructurado para guiar al usuario a través de cada paso esencial, comenzando por la preparación de la aplicación GitHub, pasando por el proceso de construcción de los componentes cliente y servidor, y concluyendo con el despliegue de todo el sistema. Se asume, tan solo, que el lector ya tiene una carrera configurada en GoRace y una *classroom* creada en GitHub Classroom.

4.1. Crear y configurar la GitHub App

Se procederá a detallar cómo crear la GitHub App y cómo configurarla propiamente para ser usada por GH2GR. No obstante, la interfaz de GitHub Classroom está en constante evolución, por lo que es posible que ciertos procedimientos no se realicen tal y como se explica en este texto si suficiente tiempo ha transcurrido desde el momento en el que se redactó. Por este motivo, es aconsejable no usar esta guía como única referencia y también acudir a la documentación oficial de GitHub.

Lo primero que tiene que hacer es ir a la página de perfil de la organización en GitHub con la que creó su *classroom*. Una vez ahí, diríjase a los ajustes de la misma.

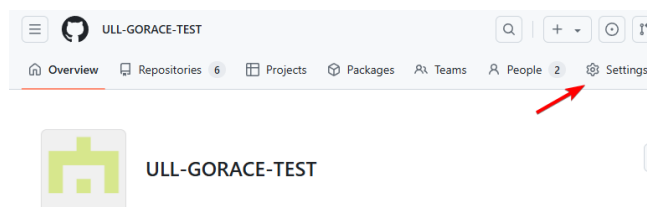


Figura 4.1: Perfil de una organización en GitHub con el botón que dirige a los ajustes resaltado.

Una vez en los ajustes, localice la barra lateral izquierda y descienda hasta su final. Ahí encontrará un desplegable con los ajustes de desarrollo. Tras abrirlo, debe pinchar

en la opción de GitHub Apps.

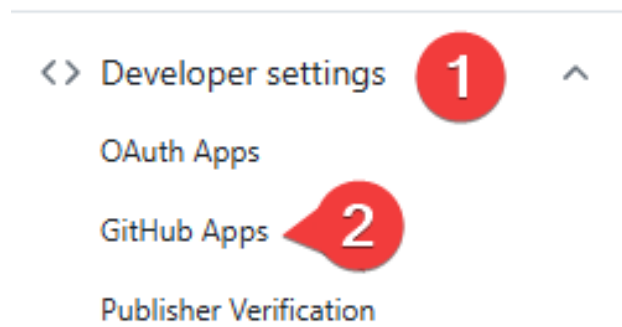


Figura 4.2: Barra lateral de los ajustes de la organización mostrando la opción para ir a las GitHub Apps.

En la página en la que nos encontramos, pulsaremos el botón que dice “New GitHub App”.

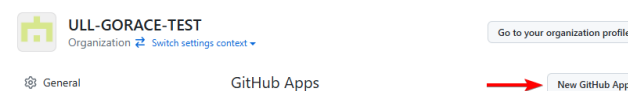


Figura 4.3: Botón para crear una nueva GitHub App.

En el formulario que se nos presenta, introduciremos un nombre que nos parezca adecuado y haremos lo mismo para la dirección de nuestra *homepage*. En “Callback URL” introduciremos “`http://127.0.0.1/callback`”, desmarcaremos las opciones de “Expire user authorization tokens” y “Request user authorization (OAuth) during installation” si están seleccionadas, y marcaremos la opción de “Enable Device Flow”.

En el apartado de *webhook* estableceremos la URL a la dirección de la máquina que va a servir GH2GR Server, añadiéndole el protocolo HTTP o HTTPS según corresponda y añadiendo “/hook” como sufijo. Así pues, si el dominio es “`gh2gr.my-domain.example`” y sirve el contenido por medio de HTTP, introduciríamos la dirección “`http://gh2gr.my-domain.example`”.

También en el apartado de *webhook* especificaremos un secreto, que será un texto aleatorio y seguro que debemos recordar para más adelante.

En el apartado de permisos, dentro de los permisos de repositorio, dotaremos a la aplicación de acceso de lectura a “Actions”, “Checks” y “Metadata”. Luego, más abajo, indicaremos que nos queremos suscribir al evento de “Workflow job”.

Concluiremos decidiendo si queremos que nuestra aplicación solo pueda ser instalada en nuestra organización o si puede ser instalada en cualquiera. Esta decisión es libre, pero se debe tener en cuenta que la aplicación debe estar instalada en todas las organizaciones donde se quiera usar GH2GR. Una vez elegido, pulsaremos el botón de “Create GitHub App”.

Figura 4.4: Formulario para la creación de la GitHub App con los campos ha cambiar resaltados en amarillo.

Figura 4.5: Botón para crear la clave privada.

Si todo sale bien, ya se habrá creado nuestra GitHub App. Pero nuestro trabajo todavía no ha acabado, pues necesitamos generar la clave secreta que permitirá a GH2GR Server actuar como nuestra GitHub App. Para generar dicha clave, descenderemos en la página que nos ha llevado GitHub hasta llegar al apartado de "Private Keys". Cuando lleguemos a él, pulsaremos el botón verde que dice "Generate a private key". En cuanto lo pulsemos, se generará una clave y el navegador intentará descargarla, tal como se puede apreciar en la figura 4.6. Es importante que la guardemos en un lugar seguro, pues la necesitaremos más adelante y no puede volver a ser descargada. No obstante, si la perdemos, siempre podemos pulsar otra vez el botón de "Generate a private key" (que se habrá desplazado ligeramente más arriba) y, posteriormente, eliminar la clave anteriormente creada. Esto último tiene que hacerse en este orden, pues GitHub no permite eliminar la única clave privada.

Otra cosa que debemos hacer mientras estamos por esta página es apuntar el "App ID" que aparece al principio de la página tal y como se puede ver en la figura 4.7. Necesitaremos este identificar para configurar nuestro servidor en el apartado 4.3.

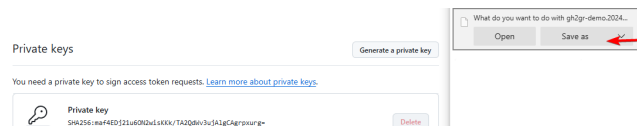


Figura 4.6: Clave privada creada y el navegador ofreciendo su descarga.

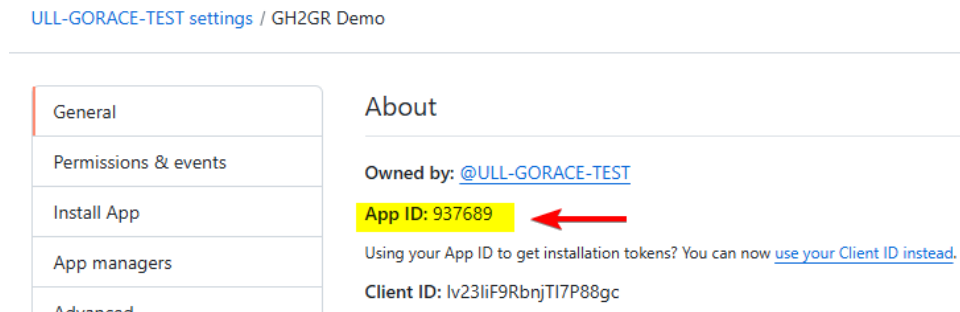


Figura 4.7: Lugar donde encontrar el App ID de nuestra GitHub App

4.2. Construir y distribuir el servidor

Como ya se comentó en el apartado 3.5.5, GH2GR Server se construye utilizando contenedores OCI. Por lo tanto, el único software que necesitamos tener instalado en el equipo con el que queremos construir el servidor es un motor de contenedores, específicamente uno capaz de crear imágenes OCI a partir de archivos Containerfile. En los ejemplos a continuación se usará Podman [56]. Sin embargo, esto es una preferencia personal y puede ser sustituido por cualquier otro, como Docker [64] o Containerd (nerdctl) [65], por nombrar algunos ejemplos. De hacerlo, se deberán ajustar los comandos a ejecutar para satisfacer cualquier motor que se haya preferido usar. En el caso de Docker y Nerdctl, los comandos son iguales a los de Podman, por lo que solo es necesario cambiar "podman" por "docker" o "nerdctl", según corresponda. Si se desea usar otro motor, descubrir qué comando utilizar se deja como ejercicio al lector.

Para construir la aplicación, nos dirigiremos a la raíz del repositorio de código y ejecutaremos el comando `podman build -t <usuario>/gh2gr-server -f Containerfile.server .`, sustituyendo "<usuario>" por nuestro propio nombre de usuario. Con la ejecución de ese comando, todas las dependencias necesarias serán descargadas, por lo que necesitaremos una conexión a internet, y se construirá una imagen OCI con el binario de GH2GR Server y el entorno mínimo para poder ejecutarlo. Si todo ha salido bien, deberíamos ver una salida similar a la figura 4.8.


```

> podman run --rm claudio4/gh2gr-server help
The server responsible for the integration between GoRace and GitHub

Usage:
  gh2gr-server [flags]
  gh2gr-server [command]

Available Commands:
  add-user          Adds a user to the database
  completion        Generate the autocompletion script for the specified shell
  create-user-token Creates a token to identify a user against the server
  help              Help about any command
  openapi           Print the OpenAPI spec

Flags:
  -a, --address string      Address to listen on
  -i, --app-id int          GitHub App ID
  -d, --database-path string Path to the database file (default "./gh2gr.sqlite")
  -k, --github-key-file string Path to the github private key file
  -s, --github-web-secret string Secret used to verify webhook calls coming from github
  -h, --help                help for gh2gr-server
  --jwt-secret string       Secret used to sign JWT tokens
  -l, --log-level string    Log level (debug, info, warn, error) (default "info")
  -p, --port uint16         Port to listen on (default 8050)

Use "gh2gr-server [command] --help" for more information about a command.

```

Figura 4.9: Salida del comando de ayuda de Gh2GR Server.

```

> podman build -t claudio4/gh2gr-server -f Containerfile.server .
[1/2] STEP 1/6: FROM docker.io/library/golang:1 AS builder
# Snipped
Successfully tagged localhost/claudio4/gh2gr-server:latest
f671b427df73b4075a315a8503bade7b91fa4e664a9a1fb8391891189e2e05e1

```

Figura 4.8: Ejecución del comando de construcción de GH2GR Server. Parte de la salida omitida por brevedad.

Podemos asegurarnos de que todo ha funcionado correctamente tratando de ejecutar el programa. Para simplificar la prueba, solo llamaremos al subcomando de ayuda, que debe imprimirnos el texto de ayuda del programa. Esto lo haremos ejecutando `podman run --rm <usuario>/gh2gr-server help`. Con dicho comando se creará un contenedor efímero que ejecutará la ayuda de GH2GR Server y luego será eliminado. Si no se nos presenta ningún problema, deberíamos tener una salida igual a la de la figura 4.9.

Ahora que ya tenemos una imagen funcional del servidor, lo que resta es saber cómo poder transferirla a la máquina servidor que vaya a ser la encargada de servir GH2GR Server. Una opción completamente válida es simplemente construir la imagen directamente en la máquina objetivo y ahorrarse el tener que moverla, ya que no se requiere más software especializado en la máquina que el motor de contenedores que necesita para poder ejecutar la imagen OCI de todas formas. Aun válida, solo contar con esta opción es restrictivo, por eso a continuación se van a enseñar dos métodos para poder distribuir la imagen.

```
> podman image save claudio4/gh2gr-server -o gh2gr-server.tar
Copying blob f144bb4c7c7f done
Copying blob 49626df344c9 done
Copying blob 945d17be9a3e done
Copying blob 4d049f83d9cf done
Copying blob af5aa97ebe6c done
Copying blob ac805962e479 done
Copying blob bbb6cacb8c82 done
Copying blob 2a92d6ac9e4f done
Copying blob 1a73b54f556b done
Copying blob f4aee9e53c42 done
Copying blob b336e209998f done
Copying blob 642674950c6f done
Copying blob 9c72aa2ec06c done
Copying blob ff6128b8bcec done
Copying config f671b427df done
Writing manifest to image destination
```

```
> docker image load --input gh2gr-server.tar
f144bb4c7c7f: Loading layer [=====] 327.7kB/327.7kB
49626df344c9: Loading layer [=====] 40.96kB/40.96kB
945d17be9a3e: Loading layer [=====] 2.390MB/2.390MB
4d049f83d9cf: Loading layer [=====] 1.536kB/1.536kB
af5aa97ebe6c: Loading layer [=====] 2.56kB/2.56kB
ac805962e479: Loading layer [=====] 2.56kB/2.56kB
bbb6cacb8c82: Loading layer [=====] 2.56kB/2.56kB
2a92d6ac9e4f: Loading layer [=====] 1.536kB/1.536kB
1a73b54f556b: Loading layer [=====] 10.24kB/10.24kB
f4aee9e53c42: Loading layer [=====] 3.072kB/3.072kB
b336e209998f: Loading layer [=====] 238.6kB/238.6kB
642674950c6f: Loading layer [=====] 13.80MB/13.80MB
9c72aa2ec06c: Loading layer [=====] 5.988MB/5.988MB
ff6128b8bcec: Loading layer [=====] 18.9MB/18.9MB
Loaded image: localhost/claudio4/gh2gr-server:latest
```

- (a) Ejecución del comando de exportado de la imagen OCI en Podman. (b) Ejecución del comando de importado de la imagen OCI. en Docker

Figura 4.10: Importado y exportado de una imagen OCI.

4.2.1. Distribución mediante fichero

Es posible exportar nuestra imagen OCI a un único fichero tar, el cual luego podremos transferir a donde necesitemos utilizando cualquier método de transferencia de archivos. Para lograr esto, solo tenemos que ejecutar el comando `podman image save <usuario>/gh2gr-server -o gh2gr-server.tar`. Tras ver una salida como la de la figura 4.10a, contaremos con un archivo "gh2gr-server.tar" que podemos llevar a la máquina destino.

Una vez en la máquina destino, debemos importar la imagen antes de poder usarla. Esto lo haremos con el comando `podman image load --input gh2gr-server.tar`, suponiendo que nos encontramos en la misma carpeta que el fichero tar. Tras unos segundos, ya deberíamos contar con la imagen en nuestro sistema y estará lista para ser utilizada. Si queremos asegurarnos, podemos volver a hacer la misma comprobación ejecutando el comando de ayuda que hicimos tras construir la imagen.

4.2.2. Distribución mediante un registro de imágenes OCI

Un registro de imágenes OCI es un repositorio especializado encargado de almacenar y distribuir imágenes OCI [66]. Esto lo hacen siguiendo la especificación de distribución OCI. La principal ventaja que tienen es que los motores de contenedores que implementan la especificación de distribución OCI son capaces de buscar en ellos y descargarse las imágenes adecuadas automáticamente.

El primer paso es escoger qué registro utilizar, ya que existen múltiples servicios en línea que podemos usar. También, por supuesto, podemos crear el nuestro propio, pero esto se sale del alcance de este documento. Para esta guía, utilizaremos Docker Hub, pero una vez más, el lector puede escoger cualquiera que le satisfaga más.

Para subir una imagen a Docker Hub, primero tendremos que registrarnos. Una vez registrado, iremos a la página principal y pincharemos en el botón de "Create repository"

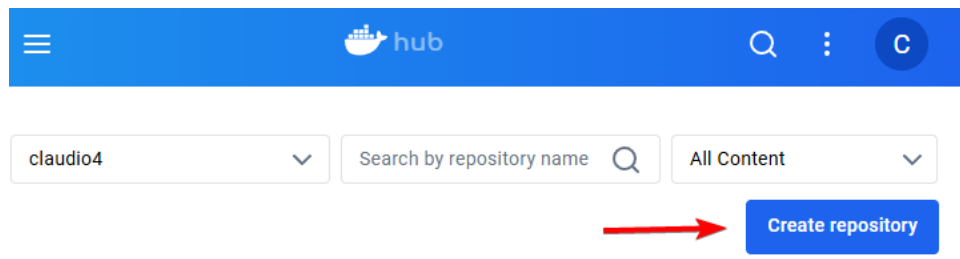


Figura 4.11: Botón para crear un repositorio en Docker Hub

(fig. 4.11). Tras pulsarlo, nos preguntará por el nombre del repositorio y por su visibilidad. Lo llamaremos "gh2gr-server" y procederemos a crearlo con la visibilidad que más nos convenza. Una vez creado, podemos volver a la terminal.

Lo primero que tendremos que hacer en la línea de comando es iniciar sesión en Docker Hub desde Podman, si no lo hemos hecho ya. Para ello, basta con ejecutar `podman login docker.io` e introducir nuestras credenciales cuando nos las pregunte (Fig. 4.13). Ahora pasaremos a reetiquetar nuestra imagen para que esté prefijada con el dominio "docker.io", señalando así el repositorio en donde debe ser buscada. Para ello, ejecutamos `podman tag <usuario>/gh2gr-server docker.io/<usuario>/gh2gr-server:latest`. Es normal que este último comando no produzca salida, de hecho, esto indica que la operación se ha realizado con éxito.

Ya estamos listos para subir la imagen al registro, hito que lograremos ejecutando `podman push docker.io/<usuario>/gh2gr-server:latest`. Si vemos algo similar a la figura 4.14, significará que todo ha ido de forma correcta y nuestra imagen ya se encuentra en el registro.

Ahora podemos dirigirnos a nuestra máquina objetivo y simplemente tratar de utilizar la imagen como si ya estuviera ahí, pues el motor de contenedores se encargará de obtenerla por nosotros. Por tanto, si ejecutamos `podman run -rm docker.io/<usuario>/gh2gr-server help`, se descargará la imagen y aparecerá el texto de ayuda tal y como muestra la figura 4.15.

4.3. Configurar el servidor

Ahora que ya tenemos la imagen de GH2GR Server en nuestro servidor, ha llegado la hora de configurarlo para que comience a trabajar correctamente. Lo primero que tendremos que hacer es crear una carpeta para albergar los ficheros de GH2GR Server en el lugar que nos resulte más conveniente. Una vez creada, colocaremos en ella la clave privada que obtuvimos en el apartado 4.1.

GH2GR Server requiere que le configuremos, al menos, la GitHub App ID (obtenida al final del apartado 4.1), la ruta de la clave secreta, el secreto para el *webhook*, que también configuramos en el apartado 4.1, y un secreto aleatorio para firmar nuestros JWT. Podemos proporcionar estos detalles como parámetros, tal y como podemos ver en la figura 4.9, o podemos usar variables de entorno. En caso de usar estas últimas,


Create repository


Namespace

claudio4

Repository Name *

gh2gr-server







Short description

A short description to identify your repository. If the repository is public, this description is used to index your content on Docker Hub and in search engines, and is visible to users in search results.

Visibility

Using 0 of 1 private repositories. [Get more](#)

☒ **Public** 
Appears in Docker Hub search results

☐ **Private** 
Only visible to you

Cancel

Create

Figura 4.12: Formulario de creación

```
> podman login docker.io
Username: claudio4
Password:
Login Succeeded!
```

Figura 4.13: Iniciar sesión en Docker Hub mediante Podman.

```

> podman push docker.io/claudio4/gh2gr-server:latest
Getting image source signatures
Copying blob 49626df344c9 done |
Copying blob f144bb4c7c7f done |
Copying blob af5aa97ebe6c done |
Copying blob 945d17be9a3e done |
Copying blob ac805962e479 done |
Copying blob 4d049f83d9cf done |
Copying blob bbb6cacb8c82 done |
Copying blob 2a92d6ac9e4f done |
Copying blob 1a73b54f556b done |
Copying blob f4aee9e53c42 done |
Copying blob b336e209998f done |
Copying blob 642674950c6f done |
Copying blob ff6128b8bcec done |
Copying blob 9c72aa2ec06c done |
Copying config f671b427df done |
Writing manifest to image destination

```

Figura 4.14: Salida del comando para subir la imagen a Docker Hub

```

> podman run --rm docker.io/claudio4/gh2gr-server help
Trying to pull docker.io/claudio4/gh2gr-server:latest...
Getting image source signatures
Copying blob 50d2b2cd6788 done |
Copying blob ffd8cfdbafe done |
# Snipped
Copying config f671b427df done |
Writing manifest to image destination
The server responsible for the integration between GoRace and GitHub

Usage:
  gh2gr-server [flags]
  gh2gr-server [command]

Available Commands:
  add-user          Adds a user to the database
  completion        Generate the autocompletion script for the specified shell
  create-user-token Creates a token to identify a user against the server
  help              Help about any command
  openapi           Print the OpenAPI spec

Flags:
  -a, --address string      Address to listen on
  -i, --app-id int          GitHub App ID
  -d, --database-path string Path to the database file (default "./gh2gr.sqlite")
  -k, --github-key-file string Path to the github private key file
  -s, --github-web-secret string Secret used to verify webhook calls coming from github
  -h, --help                help for gh2gr-server
      --jwt-secret string   Secret used to sign JWT tokens
  -l, --log-level string    Log level (debug, info, warn, error) (default "info")
  -p, --port uint16         Port to listen on (default 8050)

Use "gh2gr-server [command] --help" for more information about a command.

```

Figura 4.15: Descarga de la imagen OCI y posterior salida del comando de ayuda de Gh2GR Server.

el nombre de cada variable será igual al del parámetro (en su versión no acortada) en mayúsculas, sustituyendo los guiones por barras bajas (_) y precediéndolo de GH2GR_. Así pues, si quisiéramos configurar el App ID, usaremos la variable GH2GR_APP_ID. También es importante comentar que se pueden usar a la vez parámetros y variables de entorno para configurar los distintos ajustes. Si una misma opción se configurara por ambos métodos, prevalecería el valor escogido mediante parámetros.

Para lanzar nuestro contenedor correctamente configurado, usaríamos el comando de la figura 4.16. Si hemos hecho todo correctamente, deberíamos ver un mensaje indicándonos que el servidor HTTP se ha iniciado en el puerto 8050.

```
podman run -p 8050:8050 -v "/path/to/gh2gr-folder:/data" \
-e GH2GR_APP_ID=937689 \
-e GH2GR_GITHUB_KEY_FILE=/data/key.pem \
-e GH2GR_GITHUB_WEB_SECRET=webhook-secret \
-e GH2GR_JWT_SECRET=very-random-secret \
-e GH2GR_DATABASE_PATH=/data/gh2gr.sqlite \
<usuario>/gh2gr-server
```

Figura 4.16: Comando para ejecutar GH2GR Server

Desglosemos rápidamente el comando:

- **podman run:** Lanzamos un contenedor con Podman.
- **-p 80:8050:** Enlazamos el puerto 80 de nuestra máquina con el puerto 8050 del contenedor.
- **-v /path/to/gh2gr-folder:/data:** Montamos la carpeta que creamos al inicio en la ruta /app dentro del contenedor.
- **-e GH2GR_APP_ID=937689:** Establecemos el GitHub App ID con una variable de entorno.
- **-e GH2GR_GITHUB_KEY_FILE=/data/key.pem:** Indicamos la ruta a nuestra clave privada mediante una variable de entorno. Como la ruta es relativa al interior del contenedor, usamos el /data que montamos antes.
- **-e GH2GR_GITHUB_WEB_SECRET=webhook-secret:** Configuramos el secreto del *webhook* con una variable de entorno.
- **-e GH2GR_JWT_SECRET=very-random-secret:** Configuramos el secreto de nuestros JWT con una variable de entorno.
- **-e GH2GR_DATABASE_PATH=/data/gh2gr.sqlite:** Indicamos dónde se debe guardar la base de datos mediante una variable de entorno. La ruta tiene que ubicarse dentro de /data; de lo contrario, la base de datos no persistirá si el contenedor es borrado.
- **<usuario>/gh2gr-server:** La imagen que debe ejecutar Podman, en concreto, la que creamos nosotros.

```
> podman run --rm -v "/path/to/gh2gr-folder:/data" -e GH2GR_DATABASE_PATH=/data/gh2gr.sqlite
claudio4/gh2gr-server add-user pruebal
User pruebal added with ID 1
```

Figura 4.17: Salida del comando que agrega un usuario a GH2GR Server.

```
> podman run --rm -v "/path/to/gh2gr-folder:/data" -e GH2GR_JWT_SECRET=very-random-secret -e GH2GR_DATABASE_PATH=/data/gh2gr.sqlite claudio4/gh2gr-server create-user-token pruebal
```

Figura 4.18: Salida del comando para crear las credenciales de un usuario de GH2GR Server.

Llegados a este punto, ya tenemos nuestro servidor prácticamente listo. Lo único que nos queda es buscar un método para persistir la ejecución del comando y asegurarnos de que el puerto 80 esté expuesto a través de internet. Dado que el cómo conseguir esto depende mucho del entorno en el que se ha decidido ejecutar el contenedor, se ha dejado el resolver estas cuestiones como un ejercicio para el lector.

4.4. Añadir usuarios y crear credenciales

Ahora vamos a proceder a añadir usuarios docentes para que puedan usar GH2GR Server. Esta acción la haremos desde la línea de comandos de la máquina que sirve GH2GR Server.

El único detalle que necesitaremos conocer de ellos es el nombre de usuario. Una vez nos hemos decidido por uno, podemos agregarlo ejecutando

```
podman run -rm -v /path/to/gh2gr-folder:/data -e
GH2GR_DATABASE_PATH=/data/gh2gr.sqlite <usuario>/gh2gr-server
add-user prueba1
```

El comando nos responderá con el ID del usuario, como se puede ver en la figura 4.17.

Ahora que hemos añadido al usuario, sería conveniente generarle unas credenciales para que pueda iniciar sesión. Esto lo podemos hacer con el comando

```
podman run -rm -v /path/to/gh2gr-folder:/data -e
GH2GR_JWT_SECRET=very-random-secret -e GH2GR_DATABASE_PATH=/data/gh2gr.sqli
<usuario>/gh2gr-server create-user-token pruebal
```

El comando nos devolverá inmediatamente el token que el usuario necesita para iniciar sesión desde GH2GR Client, tal y como se puede ver en la figura 4.18.

```

> ./gh2gr
A CLI to interact with the GoRace to Github integration.

Usage:
  gh2gr [command]

Available Commands:
  auth          This command handles auth with the GoRace API and Github
  completion    Generate the autocompletion script for the specified shell
  generate       Generate action for template repository
  help          Help about any command
  race          Manage races in the GH2GR server

Flags:
  --config string  Set the config file location
  -h, --help      help for gh2gr

Use "gh2gr [command] --help" for more information about a command.

```

Figura 4.19: Salida del comando principal de GH2GR Client.

4.5. Construcción del cliente

El cliente es una aplicación más sencilla que el servidor y no requiere de CGO, así que nos aventuraremos a compilarla directamente sin usar contenedores. Esto significa que necesitamos tener instalado el SDK de Go en nuestra máquina. Las instrucciones varían según el sistema operativo, pero son sencillas y están disponibles en la página oficial del proyecto [67].

Si ya tenemos el SDK instalado, podemos construir la aplicación simplemente yendo a la raíz del repositorio y ejecutando `go build -o gh2gr cmd/gh2gr-cli/main.go`. Go se encargará de descargar las dependencias externas (cosa que solo hace la primera vez, pero requiere conexión a internet) y compilará nuestro binario con el nombre “gh2gr”. Podemos probar que funciona correctamente ejecutándolo con `./gh2gr`. Nos debería devolver una pantalla como la de la Figura 4.19.

4.6. Configurar el cliente

Ahora que ya tenemos el servidor funcionando y el cliente compilado, lo que nos resta es configurar el cliente, y eso realmente se traduce en iniciar sesión con él tanto en GitHub como en GH2GR Server.

Para iniciar sesión en GitHub, ejecutaremos el comando `gh2gr auth github login`. El comando nos dará un código y nos pedirá que pulsemos *Enter* para continuar al navegador (Figura 4.20a). Al pulsarlo, se nos llevará a una página de GitHub donde tendremos que iniciar sesión, si no lo hemos hecho ya, e introducir el código que nos proporcionó GH2GR Client al ejecutarlo (Figura 4.20b). Tras introducirlo, nos pedirá que confirmemos si efectivamente queremos permitir a GH2GR Client acceder a nuestra

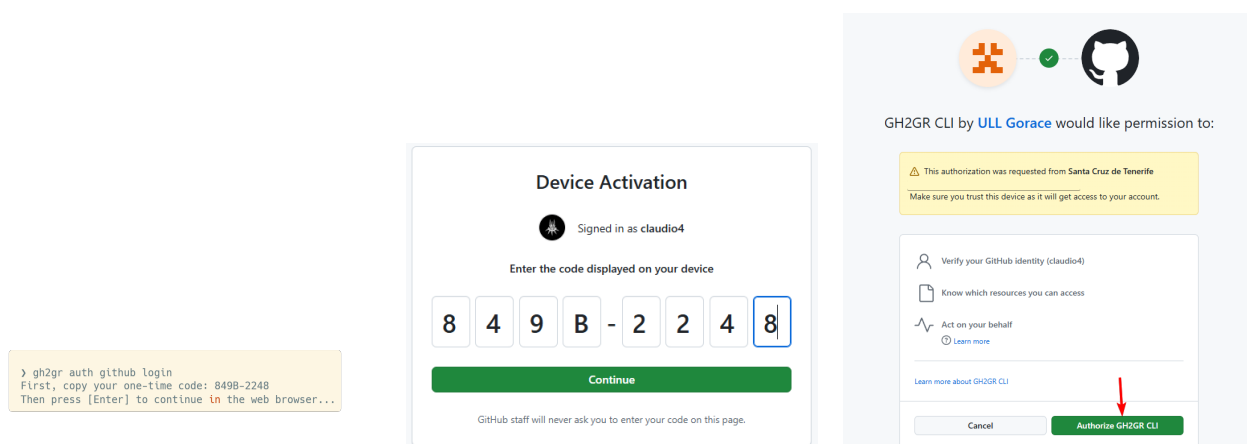


Figura 4.20: Inicio de sesión en GitHub desde GH2GR Client.

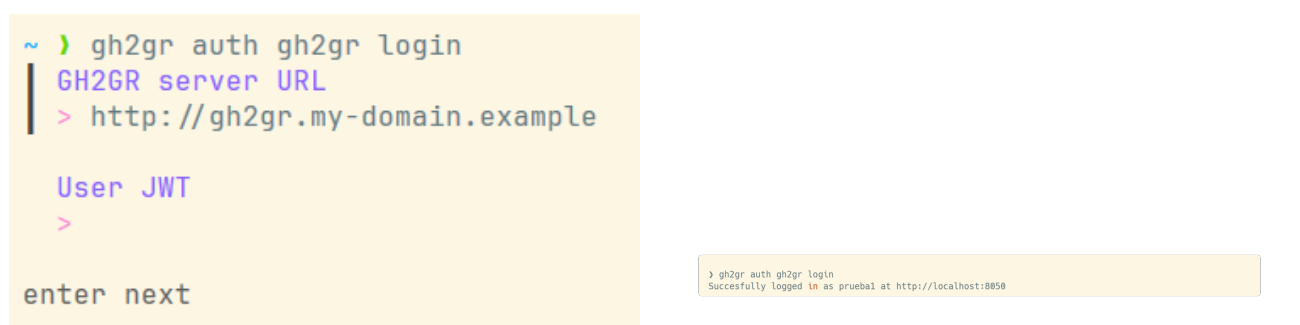


Figura 4.21: Inicio de sesión en GH2GR.

cuenta (Figura 4.20c). Responderemos que sí. Después, veremos que GH2GR Client nos imprime nuestro código de acceso y sale. No es necesario que guardemos este token, ya que la propia aplicación lo ha guardado en su configuración.

Antes de continuar, me gustaría hacer un apunte rápido para responder a la pregunta de dónde guarda el programa su configuración. La respuesta es que, si está definida la variable de entorno “XDG_CONFIG_HOME”, la guardará en un subdirectorio de la ruta que la variable especifique llamado “gh2gr”. Si la variable no está configurada, se guardará en “/.config” en sistemas Linux, en “/Library/Application Support” en sistemas Mac OS y en “%LOCALAPPDATA%” en sistemas Windows.

Nuestro siguiente paso será iniciar sesión en GH2GR Server. Para eso, ejecutaremos el comando `gh2gr auth gh2gr login`. Nos aparecerá un pequeño formulario (Figura 4.21a) donde nos preguntará la dirección de nuestro servidor y el token que generamos en el apartado 4.4. Podemos usar *Tab* y *Shift+Tab* para movernos abajo y arriba en el formulario, respectivamente. Cuando demos *Enter* en el último campo del formulario, el programa mostrará un símbolo de carga mientras se comunica con el servidor. Si todo va correctamente, nos dirá que hemos iniciado sesión exitosamente, como en la Figura 4.21b.

Lo que vamos a hacer ahora no es realmente parte de la configuración del cliente, pero sí de nuestra cuenta de usuario. Vamos a enviar el token que obtuvimos al iniciar sesión en GitHub a nuestra cuenta en GH2GR Server, para permitir que este acceda a GitHub Classroom en nuestro nombre. Para ello, ejecutaremos el comando `gh2gr auth gh2gr send-github-token`. El programa nos responderá inmediatamente avisándonos de que ha enviado el token tal y como se lo hemos pedido.

```
> gh2gr auth gh2gr send-github-token  
Your GitHub token has been successfully sent to your user prueba1 at http://gh2gr.my-domain.example
```

Figura 4.22: Mensaje indicándonos que nuestro token de GitHub se ha enviado con éxito a nuestro usuario en GH2GR Server.

4.7. Añadir una carrera

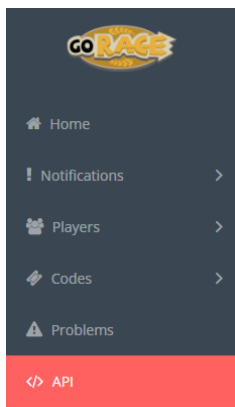
Ahora vamos a añadir una carrera a GH2GR. Antes de poder añadir nada, necesitaremos tener una carrera ya creada en GoRace, como ya se comentó en la introducción de este capítulo.

Lo primero que tenemos que hacer es obtener el JWT que nos autoriza a modificar la carrera. A continuación, se darán instrucciones de cómo proceder, pero es mejor consultar la documentación oficial de GoRace, pues, al contrario que este documento, esta puede ser actualizada si la interfaz o el procedimiento sufre algún cambio.

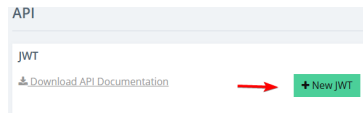
Primero nos dirigiremos a GoRaceAdmin y en la barra lateral pulsaremos la opción de API (fig. 4.23a). Ahora pulsaremos el botón de nuevo JWT (fig. 4.23b). Tras pulsarlo, nos mostrará una confirmación (fig. 4.23c) y, al confirmarlo, nuestro JWT se habrá creado y podremos pulsar el icono de la lupa para ver su valor (fig. 4.23d). Al pulsarla, veremos una ventana como en la figura 4.24 que nos dará el valor de nuestro token. Es importante que lo tengamos a mano, pues lo necesitaremos en poco tiempo.

Una vez cerrado el diálogo de la figura 4.24, aprovecharemos que seguimos en el apartado de API de GoRaceAdmin para añadir la IP de nuestro servidor a la lista blanca de IP que pueden interactuar con la API de GoRace. Para ello, pulsaremos el botón verde que dice “New IP” (fig. 4.25a). Al pulsarlo, nos aparecerá una ventana (fig. 4.25b) que nos pedirá que introduzcamos la IP. Es importante resaltar que tenemos que introducir la IP del servidor que ejecuta GH2GR Server, no la de ningún cliente. Una vez introducida, pulsaremos en añadir. Y listo, podemos comprobar que efectivamente nuestra IP se ha añadido a la lista de IP permitidas, tal y como se ve en la figura 4.25c.

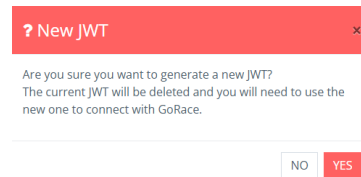
Ahora ya podemos añadir la carrera al GH2GR. Para ello, nos dirigiremos a la terminal donde tengamos nuestro cliente y ejecutamos `gh2gr race add`. Al hacerlo, nos aparecerá un formulario (fig. 4.26) para que añadamos el nombre de la carrera y el JWT que acabamos de generar. Introducimos ambos datos y pulsamos *Enter*. Tras unos instantes, nos dirá que nuestra carrera se ha creado con éxito.



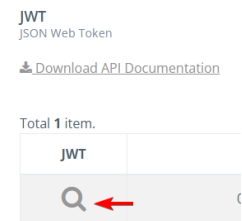
(a) Opción de API en la barra lateral de GoRaceAdmin.



(b) Botón para añadir un JWT en GoRaceAdmin.



(c) Ventana de confirmación para la creación de un nuevo JWT en GoRaceAdmin.



(d) Botón para revelar el JWT en GoRaceAdmin.

Figura 4.23: Crear un JWT en GoRaceAdmin.

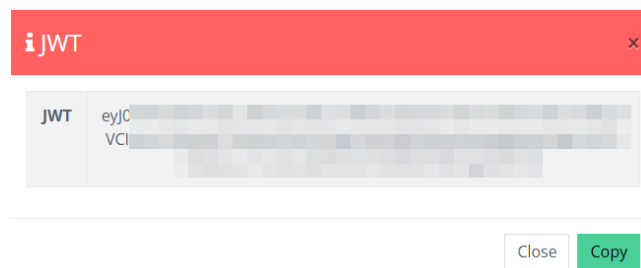
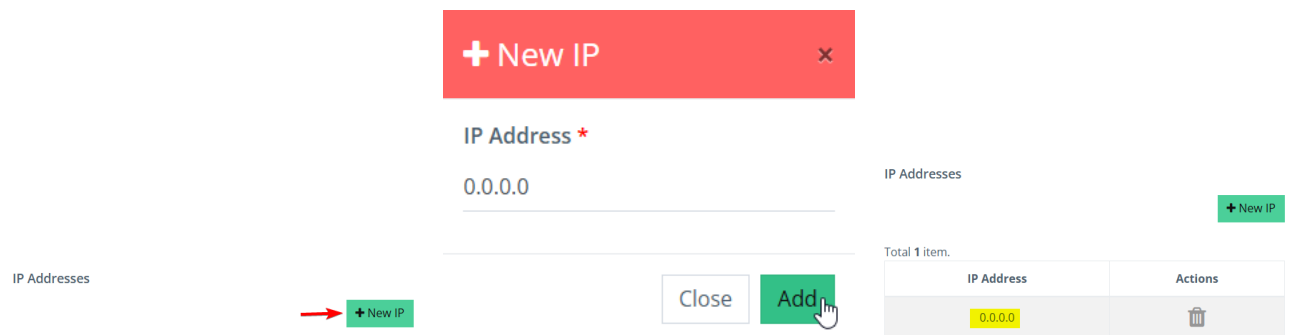


Figura 4.24: Ventana que muestra el valor del JWT en GoRaceAdmin.



(a) Botón para añadir una IP a la lista blanca.

(b) Diálogo para introducir la IP a añadir.

(c) Lista de IP permitidas tras adición.

Figura 4.25: Permitir una IP para la API de GoRace en GoRaceAdmin.

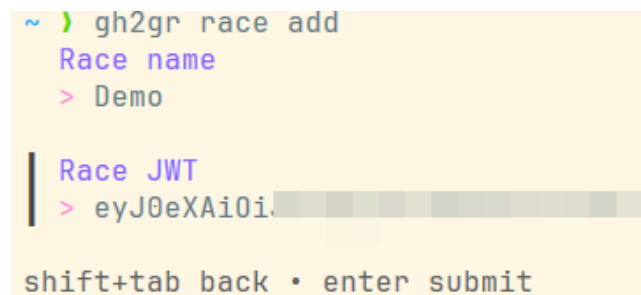


Figura 4.26: Formulario para añadir la carrera a GH2GR.

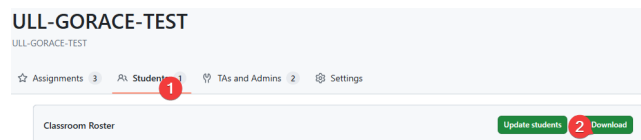


Figura 4.27: Instrucciones para descargar el *roster* de GitHub Classroom.

```
> gh2gr race send-roster demo classroom_roster.csv
Github usernames sent successfully
```

Figura 4.28: Comando que envía la traducción entre nombre de usuario de GitHub del alumnado y sus correos electrónicos.

Ahora que nuestra carrera ya está añadida, deberíamos añadir las relaciones entre los nombres de usuario de GitHub de los estudiantes y su correo electrónico. Dado que probablemente necesitemos añadir múltiples alumnos y hacerlo uno por uno puede ser tedioso, GH2GR Client acepta un CSV con todos los alumnos que queramos añadir. Como es un CSV común, podemos generarlo con el *software* que más nos convenga, siempre que pongamos los correos electrónicos en la columna “*identifier*” y los nombres de usuario en la columna “*github_username*”. El porqué de estos nombres de columnas no es casual, y es que si usamos la función de *roster* de GitHub Classroom, podremos simplemente descargarlo como se muestra en la figura 4.27 y utilizarlo directamente.

Cuando ya tengamos nuestro CSV, nos dirigiremos a la carpeta que lo albergue y ejecutaremos `gh2gr race send-roster <nombre-carrera><fichero.csv>`. También podemos usar un guion (-) como el nombre del fichero si queremos que el programa lo lea de la entrada estándar. La terminal nos responderá que se han enviado correctamente.

Un efecto secundario de enviar estas relaciones es que el servidor preregistrará a todos los correos que le hayamos otorgado en nuestra carrera en GoRace. Además, es importante comentar que podemos hacer sin ningún tipo de problema envíos parciales del *roster* y estos se irán añadiendo sin problema. Además, si proporcionamos una dirección de correo electrónico distinta para un nombre que ya había sido enviado, GH2GR actualizará la traducción para usar este nuevo correo de forma silenciosa.

4.8. Añadir una actividad y prepararla para su uso

Lo último que nos resta por explicar es cómo añadir una actividad a GH2GR y prepararla para que pueda ser usada por el alumnado.

Antes de continuar, recordemos que, como siempre, se recomienda consultar la documentación oficial de GitHub además de esta guía, pues este documento corre el riesgo de quedarse desactualizado.

Lo primero que tenemos que hacer es añadir el *assignment* en GitHub Classroom. Para ello, vamos a nuestro *classroom* y pulsamos el botón de “New assignment” (fig. 4.29a).

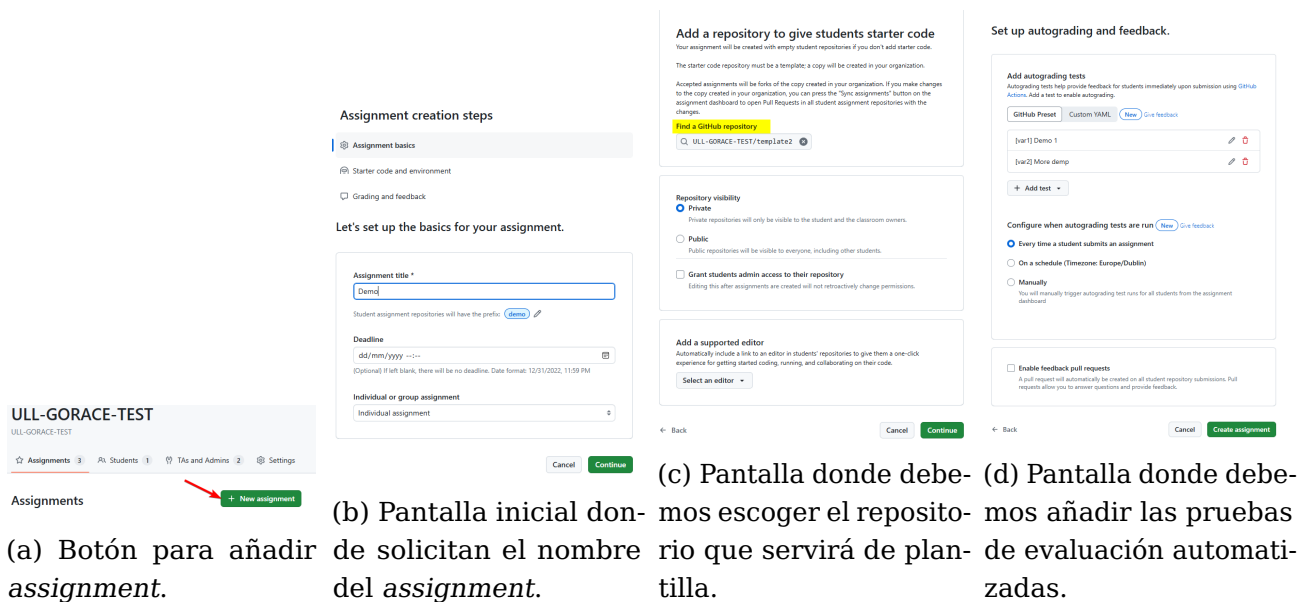


Figura 4.29: Formulario para la creación de un nuevo *assignment* en GitHub Classroom.

Al pulsarlo, nos preguntará un título (fig. 4.29b), que podemos escoger libremente, y otras opciones que también somos libres de configurar como nos plazca. En la siguiente pantalla, es vital que escojamos un repositorio bajo nuestro control que usaremos como plantilla para el repositorio de los estudiantes (fig. 4.29c); el resto de valores somos libres de configurarlos a nuestro gusto. En la última pantalla (fig. 4.29d), debemos añadir las pruebas de evaluación automatizadas que consideremos, recordando que debemos agruparlas por las variables de la actividad de GoRace, usando un prefijo de “[nombre variable]” para este fin, tal y como se muestra en la figura 3.1 y se comenta en el apartado 3.1. También es importante añadir una puntuación a cada una de las pruebas. Tras acabar con las pruebas, podemos confirmar para crear el *assignment*.

Ahora simplemente tenemos que generar el fichero de *workflow* que provocará que se active la GitHub Action cuando los alumnos actualicen sus repositorios. Para ello, nos vamos a la raíz de nuestro repositorio plantilla y ejecutamos `gh2gr generate`. GH2GR nos dará una lista de las *classrooms* (fig. 4.30a) a las que nuestro usuario de GitHub tiene acceso, escogeremos la que sea adecuada. Ahora nos listará los *assignments* dentro de ese *classroom* (fig. 4.30b), una vez más, escogeremos el correspondiente. A continuación, nos pedirá que escojamos una carrera de las que tenemos registradas en GH2GR (fig. 4.30c). Tras escoger carrera, nos pedirá que escribamos el nombre de la actividad en GoRace (fig. 4.30d) y después nos preguntará dónde debe guardar el archivo de *workflow* (fig. 4.30e). Si nos habíamos colocado en la raíz del repositorio plantilla como se indicó antes, la ruta debería ser correcta; si no, aquí tiene una oportunidad para ajustarla. Al pulsar *Enter*, la aplicación generará el archivo en la ruta indicada, creando también cualquier carpeta que no existiera pero fuera parte de la ruta y nos imprimirá un mensaje de éxito (fig. 4.30f).

Con esto, solo tendremos que hacer *commit* y *push*, y ya estará todo listo para que nuestros alumnos puedan disfrutar de la integración entre GitHub Classroom y GoRace.

```

~ > gh2gr generate
Select a classroom
ULL-ESIT-PL-2122
ULL-ESIT-PL-2324
ULL-ESIT-PL-2122-classroom-7baae6
> ULL-GORACE-TEST

```

(a) Menú de selección de *classroom*.

```

~ > gh2gr generate
Select a race
> Demo

```

(c) Menú de selección de carrera.

```

~ > gh2gr generate
Workflow directory
Where to save the workflow file?
> /home/claudio4/.github/workflow

```

(e) Entrada para el nombre de la ruta de guardado.

```

~ > gh2gr generate
Select an assignment
Act1
Actividad2
new-autograding
> Demo

```

(b) Menú de selección de *assignment*.

```

~ > gh2gr generate
Go race assignment name
> demo

```

(d) Entrada para el nombre de la actividad.

```

~ > gh2gr generate
Workflow file created at /home/claudio4/.github/workflow/gorace.yml

```

(f) Salida confirmando el guardado del fichero.

Figura 4.30: Comando de gh2gr generate.

Capítulo 5

Evaluación en un entorno real

Como parte de este TFM se puso a prueba GH2GR en un entorno real. En concreto, se utilizó para lúdificar la actividad de *left-side* de la asignatura de Procesadores de Lenguajes del grado de Ingeniería Informática de la ULL [68]. En esta experiencia participaron treinta y siete alumnos, realizando una sola actividad compuesta de cinco variables y doce pruebas de evaluación automatizadas.

Para esta prueba, el GH2GR Server se ejecutó en un VPS hospedado en la nube de Oracle, más concretamente en su centro de datos de Newport, Reino Unido. En este servidor permaneció ejecutándose GH2GR Server sin interrupción durante el periodo de una semana que duró la carrera. Aunque no estuvo expuesto directamente, sino detrás de un proxy inverso, en concreto detrás de un servidor Caddy [69].

En el transcurso de la prueba solo se registró una incidencia, aunque incluso esta es difícil de argumentar que fuera realmente culpa de GH2GR. El problema surgió a la hora de ejecutar las pruebas automáticas de evaluación, pues el código de los alumnos no lograba superar las pruebas dentro de la GitHub Action, al contrario de lo que ocurría en los ordenadores de los propios alumnos. La causa del problema resultó ser que la GitHub Action establecía la variable de entorno `"FORCE_COLOR"`, variable que causaba que el código de los alumnos se comportara de formas inesperadas. Que esta variable estuviera configurada era una simple herencia del código original de la acción de autoevaluación de GitHub Classroom sobre el que se basó nuestra GitHub Action y no cumple ningún propósito en nuestra versión modificada. Por este motivo, se decidió eliminar la variable y con eso se solucionó el problema.

La incidencia se solucionó rápidamente y solo se tiene constancia de que haya afectado a dos alumnos especialmente diligentes.

Más allá de lo comentado, no hubo ningún problema, así que se consideró que la prueba fue un rotundo éxito.

Me gustaría aprovechar una vez más para agradecer al equipo de la UCA, con énfasis especial para el doctor Alejandro Calderón Sánchez, por el soporte prestado y la celeridad del mismo, que permitió que esta prueba se llevara a cabo en el ajustado marco de tiempo con el que se contaba.

5.1. Retroalimentación de los alumnos

Tras el experimento, se les sirvió al alumnado una encuesta de satisfacción para observar qué tan contentos (o descontentos) estaban con GH2GR. Elaborar esta encuesta no fue sencillo, pues es complicado preguntar sobre un elemento como GH2GR que hace todo lo posible por ser invisible para los alumnos y transferir la información de una plataforma a la otra con la menor fanfarria posible. Además, había que diseñar una encuesta extremadamente corta y sencilla de rellenar, pues ya habíamos pasado una solicitud al alumnado a petición de GoRace y nos temíamos que una encuesta demasiado larga podría causar rechazo en unos fatigados estudiantes.

La encuesta al final se resolvió en siete afirmaciones con las que los alumnos podían expresar cuán de acuerdo estaban usando un número natural del 1 al 5, representando 1 “Completamente en desacuerdo” y 5 “Completamente de acuerdo”. Las preguntas fueron las siguientes:

1. Mi trabajo en el repositorio de Github se reflejaba rápidamente en GoRace.
2. La Github Action de “GoRace Workflow” otorgaba resultados correctos, mientras que la Github Action de Classroom denominada “Autograding Tests” no lo hacía. (Visite el apartado de Actions de su repositorio en el lado izquierdo si tiene dudas acerca de esta pregunta.)
3. Mi trabajo en el repositorio de Github se reflejaba sin problemas en GoRace.
4. La presencia de la integración con GoRace no ha complicado mi labor a la hora de completar el ejercicio.
5. Observar cómo mi trabajo en el repositorio de Github influye en mi posición en GoRace me motiva a seguir adelante con el desarrollo del ejercicio.
6. La presencia de la integración con GoRace ha hecho más satisfactorio el desarrollo de mi tarea.
7. La integración de GoRace mejora la experiencia de utilizar exclusivamente Github Classroom.

La encuesta recibió tan solo cinco respuestas, es decir, un 13 % de los participantes. Con un espacio muestral tan bajo, cualquier conclusión que se pueda sacar es irrelevante; aun así, por el ejercicio de ello, vamos a intentar analizar los datos de todas formas.

La tabla 5.1 muestra la media y desviación típica de cada pregunta. De estos datos podemos concluir que el alumnado considera que GH2GR es rápido, no encuentra problemas en el paso de datos a GoRace, ni considera que la presencia de GH2GR haya complicado su labor. Además, reportan que el contar con su trabajo reflejado en GoRace mejora su motivación y hace la experiencia más satisfactoria.

Desde luego, la opinión que revela la encuesta es satisfactoria, pero se debe recordar que los participantes son demasiado pocos como para dar ningún peso a estas conclusiones.

Nº Pregunta	1	2	3	4	5	6	7
Media	4.40	3.80	4.60	4.80	4.20	4.60	4.40
Desviación media	0.96	0.64	0.64	0.32	0.32	0.48	0.48

Cuadro 5.1: Resultado de la encuesta de satisfacción del alumnado con GH2GR. El número de la pregunta corresponde con su posición en la lista del apartado 5.1.

Capítulo 6

Conclusiones y líneas futuras

6.1. Conclusiones

Este trabajo ha demostrado el importante potencial de la gamificación para mejorar la enseñanza de las TI centrándose en el desarrollo y la integración del programa GH2GR. Se ha demostrado que la gamificación potencia el aprendizaje en el campo de las TI, ofreciendo una experiencia educativa más atractiva y eficaz.

El objetivo principal de este proyecto fue el desarrollo de GH2GR, un programa de software diseñado para unir Github Classroom, una plataforma ampliamente utilizada para la educación en TI, con GoRace, una plataforma de gamificación establecida y probada. GH2GR proporciona una forma automatizada y fluida de incorporar la gamificación a los cursos de TI, requiriendo un mínimo esfuerzo adicional por parte de los educadores. Esta integración permite a los profesores seguir utilizando sus herramientas habituales mientras mejoran sus cursos con los beneficios de la gamificación a través de GoRace.

El desarrollo y la implantación de GH2GR han sido un éxito, ya que el programa ha funcionado a la perfección y ha demostrado ser una herramienta fiable para los educadores. Este éxito técnico valida el diseño y la ejecución de GH2GR, confirmando su practicidad y utilidad en entornos educativos del mundo real.

Los primeros comentarios de los usuarios de GH2GR han sido en general favorables, lo que pone de manifiesto la buena acogida y el impacto potencial de la herramienta. Sin embargo, el escaso número de comentarios recogidos hasta la fecha impide extraer conclusiones definitivas sobre su eficacia a largo plazo y su aplicabilidad generalizada.

En conclusión, este trabajo pone de relieve el papel potenciador de la gamificación en la enseñanza de las TI a través del desarrollo de GH2GR. El programa representa un avance significativo en la integración de la gamificación con las herramientas educativas existentes, y promete mejorar el compromiso de los estudiantes y los resultados del aprendizaje.

6.2. Líneas futuras

El futuro de este proyecto está altamente influenciado por las plataformas que integra, por lo que es complicado dictaminar cuáles son las líneas de trabajo futuras más interesantes sin saber cómo evolucionarán estas dos plataformas.

Tal y como están las cosas ahora mismo, la línea de trabajo principal parece ser la de aumentar la base de usuarios y poner a prueba la aplicación en carreras más completas para así poder ver cuál es el impacto real de GH2GR y comprobar si de verdad supone un beneficio positivo para los cursos de TI que lo implementen.

Otra línea de trabajo interesante sería adaptarlo para ofrecerlo como un SaaS, evitando que el profesorado tenga que gestionar su propio servidor. Otro giro interesante, si el proyecto llegara a interesar a las buenas gentes de la UCA, sería una mayor integración con GoRace y quizás ofrecerlo como una integración por defecto en la que el servidor ya esté desplegado en la infraestructura de GoRace.

Capítulo 7

Conclusions and future lines of work

7.1. Conclusions

This thesis has demonstrated the significant potential of gamification in enhancing IT education by focusing on the development and integration of the GH2GR program. Gamification has been shown to empower learning in the IT field, offering a more engaging and effective educational experience.

The primary focus of this thesis was the development of GH2GR, a software program designed to bridge Github Classroom, a widely-used platform for IT education, with GoRace, an established and proven gamification platform. GH2GR provides an automated and seamless way to incorporate gamification into IT courses, requiring minimal additional effort from educators. This integration allows professors to continue using their familiar tools while enhancing their courses with the benefits of gamification through GoRace.

The development and implementation of GH2GR have been successful, with the program functioning flawlessly and proving to be a reliable tool for educators. This technical success validates the design and execution of GH2GR, confirming its practicality and utility in real-world educational settings.

Initial feedback from users of GH2GR has been generally favourable, highlighting the positive reception and potential impact of the tool. However, the limited amount of feedback collected thus far precludes definitive conclusions regarding its long-term effectiveness and widespread applicability.

In conclusion, this thesis emphasizes the empowering role of gamification in IT education through the development of GH2GR. The program represents a significant advancement in integrating gamification with existing educational tools, promising to enhance student engagement and learning outcomes.

7.2. Future lines of work

The future of this project is highly influenced by the platforms it integrates, so it is difficult to say what the most interesting future lines of work are without knowing how these two platforms will evolve.

As things stand at the moment, the main line of work seems to be to increase the user base and test the application in more complete races in order to see what the real impact of GH2GR is and to check if it really means a positive benefit for the IT courses that implement it.

Another interesting line of work would be to adapt it to offer it as a SaaS, avoiding teachers having to manage their own server. Another interesting twist, if the project were to be of interest to the good people at UCA, would be to further integrate it with GoRace and perhaps offer it as a default integration where the server is already deployed on the GoRace infrastructure.

Bibliografía

- [1] B. Marín, J. Frez, J. Cruz-Lemus, and M. Genero, "An empirical investigation on the benefits of gamification in programming courses," *ACM Trans. Comput. Educ.*, vol. 19, nov 2018.
- [2] S. Deterding, M. Sicart, L. Nacke, K. O'Hara, and D. Dixon, "Gamification. using game-design elements in non-gaming contexts," in *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, (New York, NY, USA), p. 2425–2428, Association for Computing Machinery, 2011.
- [3] C.-H. Tu, L. E. Sujo-Montes, and C.-J. Yen, *Gamification for Learning*, p. 203–217. Springer International Publishing, June 2014.
- [4] G. Barata, S. Gama, J. Jorge, and D. Gonçalves, "Improving participation and learning with gamification," in *Proceedings of the First International Conference on Gameful Design, Research, and Applications*, Gamification '13, ACM, Oct. 2013.
- [5] M. Lister, "Gamification: The effect on student motivation and performance at the post-secondary level," *Issues and Trends in Educational Technology*, vol. 3, Dec. 2015.
- [6] "What is version control? — about.gitlab.com." <https://about.gitlab.com/topics/version-control/>. [Accessed 14-05-2024].
- [7] "About GitHub and Git - GitHub Docs — docs.github.com." <https://docs.github.com/en/get-started/start-your-journey/about-github-and-git>. [Accessed 14-05-2024].
- [8] "About GitHub Classroom - GitHub Docs." <https://docs.github.com/en/education/manage-coursework-with-github-classroom/get-started-with-github-classroom/about-github-classroom>. [Accessed 03-05-2024].
- [9] "Understanding GitHub Actions - GitHub Docs." <https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions>. [Accessed 03-05-2024].
- [10] "Use autograding - GitHub Docs." <https://docs.github.com/en/education/manage-coursework-with-github-classroom/teach-with-github-classroom/use-autograding>. [Accessed 03-05-2024].

- [11] "About the REST API - GitHub Docs — docs.github.com." <https://docs.github.com/en/rest/about-the-rest-api/about-the-rest-api?apiVersion=2022-11-28>. [Accessed 14-05-2024].
- [12] "REST API endpoints for GitHub Classroom - GitHub Docs — docs.github.com." <https://docs.github.com/en/rest/classroom/classroom?apiVersion=2022-11-28>. [Accessed 14-05-2024].
- [13] "About creating GitHub Apps - GitHub Docs — docs.github.com." <https://docs.github.com/en/apps/creating-github-apps/about-creating-github-apps/about-creating-github-apps>. [Accessed 14-05-2024].
- [14] M. Trinidad, A. Calderón, and M. Ruiz, "Gorace: A multi-context and narrative-based gamification suite to overcome gamification technological challenges," *IEEE Access*, vol. 9, pp. 65882–65905, 2021.
- [15] B. Marín, J. Frez, J. Cruz-Lemus, and M. Genero, "An empirical investigation on the benefits of gamification in programming courses," *ACM Trans. Comput. Educ.*, vol. 19, nov 2018.
- [16] Z. Zhan, L. He, Y. Tong, X. Liang, S. Guo, and X. Lan, "The effectiveness of gamification in programming education: Evidence from a meta-analysis," *Computers and Education: Artificial Intelligence*, vol. 3, p. 100096, 2022.
- [17] "Dungeons & Developers." <http://www.dungeonsanddevelopers.com/>. [Accessed 07-05-2024].
- [18] "About Bunchball | BI WORLDWIDE." <https://www.biworldwide.com/about-us/about-bunchball/>. [Accessed 07-05-2024].
- [19] A. Calderón, J. Boubeta-Puig, and M. Ruiz, "Medit4cep-gam: A model-driven approach for user-friendly gamification design, monitoring and code generation in cep-based systems," *Information and Software Technology*, vol. 95, pp. 238–264, 2018.
- [20] "What is a webhook? — redhat.com." <https://www.redhat.com/en/topics/automation/what-is-a-webhook>. [Accessed 19-06-2024].
- [21] "GitHub - education/autograding: GitHub Education Auto-grading and Feedback for GitHub Classroom — github.com." <https://github.com/education/autograding>. [Accessed 19-06-2024].
- [22] "JavaScript With Syntax For Types. — typescriptlang.org." <https://www.typescriptlang.org/>. [Accessed 19-06-2024].
- [23] "Node.js — Run JavaScript Everywhere — nodejs.org." <https://nodejs.org/en>. [Accessed 19-06-2024].
- [24] I. of Electrical, E. Engineers, and T. O. Group, *exit(1p) — POSIX Programmer's Manual — Linux manual page*, 2018.
- [25] "JSON — json.org." <https://www.json.org/json-es.html>. [Accessed 19-06-2024].

- [26] T. C. 39, “ECMAScript® 2023 language specification,” standard, Ecma International, June 2023.
- [27] D. H. Krawczyk, M. Bellare, and R. Canetti, “HMAC: Keyed-Hashing for Message Authentication.” RFC 2104, Feb. 1997.
- [28] Q. H. Dang, *Secure Hash Standard*. National Institute of Standards and Technology, July 2015.
- [29] R. T. Fielding and J. Reschke, “Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content.” RFC 7231, June 2014.
- [30] “Webhook events and payloads - GitHub Docs — docs.github.com.” https://docs.github.com/en/webhooks/webhook-events-and-payloads#workflow_run. [Accessed 19-06-2024].
- [31] “Validating webhook deliveries - GitHub Docs — docs.github.com.” <https://docs.github.com/en/webhooks/using-webhooks/validating-webhook-deliveries>. [Accessed 19-06-2024].
- [32] “The Go Programming Language — go.dev.” <https://go.dev/>. [Accessed 19-06-2024].
- [33] “What’s so great about Go? - Stack Overflow — stackoverflow.blog.” <https://stackoverflow.blog/2020/11/02/go-golang-learn-fast-programming-languages/>. [Accessed 20-06-2024].
- [34] “A Guide to the Go Garbage Collector - The Go Programming Language — tip.golang.org.” <https://tip.golang.org/doc/gc-guide>. [Accessed 20-06-2024].
- [35] “GitHub - cli/cli: GitHub’s official command line tool — github.com.” <https://github.com/cli/cli>. [Accessed 20-06-2024].
- [36] C. A. R. Hoare, *Communicating Sequential Processes*. Prentice-Hall International Series in Computer Science, Englewood Cliffs, New Jersey: Prentice-Hall International, 1985.
- [37] “Frequently Asked Questions (FAQ) - The Go Programming Language — go.dev.” https://go.dev/doc/faq#Is_Go_an_object-oriented_language. [Accessed 20-06-2024].
- [38] “Codewalk: First-Class Functions in Go - The Go Programming Language — go.dev.” <https://go.dev/doc/codewalk/functions/>. [Accessed 20-06-2024].
- [39] “Trygve/MVC — folk.universitetetioslo.no.” <https://folk.universitetetioslo.no/trygver/themes/mvc/mvc-index.html>. [Accessed 20-06-2024].
- [40] “Hexagonal architecture — alistair.cockburn.us.” <https://alistair.cockburn.us/hexagonal-architecture/>. [Accessed 20-06-2024].
- [41] R. C. Martin, *Agile software development, principles, patterns, and practices*. Alan Apt series, Upper Saddle River, NJ: Pearson, Oct. 2002.

- [42] A. Kumar, “An Introduction to MVCS Architecture.” <https://quantiphi.com/an-introduction-to-mvcs-architecture/>. [Accessed 27-06-2024].
- [43] S. Ajmani, “Package names - The Go Programming Language — go.dev.” <https://go.dev/blog/package-names>, 2015. [Accessed 27-06-2024].
- [44] “REST API endpoints for emails - GitHub Docs — docs.github.com.” <https://docs.github.com/en/rest/users/emails?apiVersion=2022-11-28#list-email-addresses-for-the-authenticated-user>. [Accessed 02-07-2024].
- [45] R. T. Fielding, “Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST) — ics.uci.edu.” https://ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, 2000. [Accessed 02-07-2024].
- [46] “JSON Schema - 2020-12 Release Notes — json-schema.org.” <https://json-schema.org/draft/2020-12/release-notes>, 2020. [Accessed 03-07-2024].
- [47] R. T. Fielding, M. Nottingham, and J. Reschke, “HTTP Semantics.” RFC 9110, June 2022.
- [48] M. Nottingham, E. Wilde, and S. Dalal, “Problem Details for HTTP APIs.” RFC 9457, July 2023.
- [49] “OpenAPI Specification v3.1.0 | Introduction, Definitions, & More — spec.openapis.org.” <https://spec.openapis.org/oas/v3.1.0>. [Accessed 03-07-2024].
- [50] “About SQLite — sqlite.org.” <https://www.sqlite.org/about.html>. [Accessed 03-07-2024].
- [51] A. Beaulieu, *Learning SQL*. Sebastopol, CA: O’Reilly Media, 2 ed., May 2009.
- [52] “Command-line Interfaces (CLIs) - The Go Programming Language — go.dev.” <https://go.dev/solutions/clis#key-benefits>. [Accessed 03-07-2024].
- [53] “cgo command - cmd/cgo - Go Packages — pkg.go.dev.” <https://pkg.go.dev/cmd/cgo>. [Accessed 03-07-2024].
- [54] “FAQ - Open Container Initiative — opencontainers.org.” <https://opencontainers.org/faq/#what-has-docker-done-to-help-create-this-founda> [Accessed 03-07-2024].
- [55] “moby/README.md at moby/moby — github.com.” <https://github.com/moby/moby/blob/81ccfd44e4fa451d18bdc538ff731b11b4064040/README.md>, 2022. [Accessed 04-07-2024].
- [56] “What is Podman? Podman documentation — docs.podman.io.” <https://docs.podman.io/en/v5.1.1/>. [Accessed 04-07-2024].
- [57] “Docker Desktop — docker.com.” <https://www.docker.com/products/docker-desktop/>. [Accessed 04-07-2024].

- [58] “Introduction | Rancher Desktop Docs — docs.rancherdesktop.io.” <https://docs.rancherdesktop.io/>. [Accessed 04-07-2024].
- [59] “Introduction | Podman Desktop — podman-desktop.io.” <https://podman-desktop.io/docs/intro>. [Accessed 04-07-2024].
- [60] “Adrian Mouat — docker.com.” <https://www.docker.com/captains/adrian-mouat/>. [Accessed 04-07-2024].
- [61] A. Mouat, *Using docker*. O’Reilly Media, Inc., 2015.
- [62] “GitHub - claudio4/gorace-gh-integration — github.com.” <https://github.com/claudio4/gorace-gh-integration>.
- [63] A. Hunt and D. Thomas, *The pragmatic programmer*. Boston, MA: Addison Wesley, Oct. 1999.
- [64] “Docker Engine overview — docs.docker.com.” <https://docs.docker.com/engine/>. [Accessed 05-07-2024].
- [65] “containerd/nerdctl: contaiNERD CTL - Docker-compatible CLI for containerd, with support for Compose, Rootless, eStargz, OCIcrypt, IPFS, ... — github.com.” <https://github.com/containerd/nerdctl>. [Accessed 05-07-2024].
- [66] O. C. Initiative, “Open Container Initiative Distribution Specification,” standard, Open Container Initiative, Feb. 2024.
- [67] “Download and install - The Go Programming Language — go.dev.” <https://go.dev/doc/install>. [Accessed 05-07-2024].
- [68] “Nextra: the next docs builder — ull-pl.vercel.app.” <https://ull-pl.vercel.app/>. [Accessed 06-07-2024].
- [69] C. W. Server, “Welcome - Caddy Documentation — caddyserver.com.” <https://caddyserver.com/docs/>. [Accessed 06-07-2024].
- [70] “Evolution of HTTP - HTTP | MDN — developer.mozilla.org.” https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP#post-http2_evolution. [Accessed 18-06-2024].
- [71] “What is a URL? - Learn web development | MDN — developer.mozilla.org.” https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_URL. [Accessed 19-06-2024].

Glosario

- ACID** Atomicity, Consistency, Isolation and Durability ó Atomicidad, Consistencia, Aislamiento y Durabilidad. 22
- ADT** Abstract data type ó Tipo Abstracto de Dato. 19, 20, 22
- API** Application Programming Interface ó Interfaz de Programación de Aplicaciones. 3, 5, 12, 14, 16–22, 38
- CI/CD** Integración Continua/Despliegue Continuo. 4
- CSP** Communicating Sequential Processes ó Comunicación de Procesos Secuenciales. 18
- CSV** Comma-Separated Values ó Valores Separados por Comas. 40
- DBMS** Database Management System ó Sistema de Gestión de Bases de Datos. 22
- DRY** Don't Repeat Yourself ó No Te Repitas. 24
- HMAC** Hash-based Message Authentication Code ó Código de Autenticación de Mensaje basado en Hash. 17
- HTTP** HyperText Transfer Protocol ó Protocolo de Transferencia de Hipertexto. Es el protocolo utilizado para servir las páginas web y hoy en día se usa para compatir todo tipo de contenido.[70]. 13, 15–19, 21, 26, 34
- HTTPS** HyperText Transfer Protocol Secure ó Protocolo de Transferencia de Hipertexto Seguro. 26
- IP** Internet Protocol ó Protocolo de Internet. 38
- IT** Information Technology ó Tecnologías de la Información. 49
- JSON** JavaScript Object Notation ó Notación de Objeto de JavaScript. 13, 14, 16, 17, 19, 55
- JWT** JSON Web Token. 19, 31, 34, 38
- MVC** Model-View-Controller ó Modelo-Vista-Controlador. 18
- MVCS** Model-View-Controller-Service ó Modelo-Vista-Controlador-Servicio. 18
- OCI** Open Container Initiative. 22, 28–30

REST Representational State Transfer ó Transferencia de Estado Representacional. 21

RPC Remote Procedure Call ó Llamada a Procedimiento Remoto. 21

SaaS Software as a Service ó Software como un Servicio. 47, 49

SDK Software Development Kit ó Kit de Desarrollo de Software. 21

SQL Structured Query Language ó Lenguaje de Consulta Estructurada. 22

TFM Trabajo de Fin de Máster. 7, 43

TI Tecnologías de la Información. 1, 6, 24, 47

UCA Universidad de Cádiz. 8, 43, 47, 49

ULL Universidad de La Laguna. 9, 43

URI Uniform Resource Identifier ó Identificador de Recursos Uniforme. 21

URL Uniform Resource Locator ó Localizador de Recursos Uniforme. Es la dirección de un recurso único en Internet.[71]. 15, 17

VPS Virtual Private Server ó Servidor Virtual Privado. 43

Apéndice A

Códigos

A.1. Esquema OpenAPI de la API de gestión

```
1 components:
2   schemas:
3     ErrorDetail:
4       additionalProperties: false
5       properties:
6         location:
7           description: Where the error occurred, e.g. 'body.items[3].tags' or
              ↳ 'path.thing-id'
8           type: string
9         message:
10          description: Error message text
11          type: string
12        value:
13          description: The value at the given location
14          type: object
15      ErrorModel:
16        additionalProperties: false
17        properties:
18          $schema:
19            description: A URL to the JSON Schema for this object.
20            format: uri
21            readOnly: true
22            type: string
23          detail:
24            description: A human-readable explanation specific to this occurrence
              ↳ of the problem.
25          examples:
26            - Property foo is required but is missing.
27            type: string
28        errors:
29          description: Optional list of individual error details
30          items:
31            $ref: "#/components/schemas/ErrorDetail"
32          type: array
33      instance:
34        description: A URI reference that identifies the specific occurrence
              ↳ of the problem.
```

```

35     examples:
36       - https://example.com/error-log/abc123
37     format: uri
38     type: string
39   status:
40     description: HTTP status code
41     examples:
42       - 400
43     format: int64
44     type: integer
45   title:
46     description: A short, human-readable summary of the problem type. This
47     ↪ value should not change between occurrences of the error.
48     examples:
49       - Bad Request
50     type: string
51   type:
52     default: about:blank
53     description: A URI reference to human-readable documentation for the
54     ↪ error.
55     examples:
56       - https://example.com/errors/example
57     format: uri
58     type: string
59   type: object
60   GithubUserMapping:
61     additionalProperties: false
62     properties:
63       email:
64         description: The email address that the github user will use in GoRace
65         examples:
66           - user@example.tld
67         format: email
68         type: string
69       username:
70         description: The github username of the student
71         examples:
72           - studentuser1
73         type: string
74     required:
75       - email
76       - username
77     type: object
78   Race:
79     additionalProperties: false
80     properties:
81       $schema:
82         description: A URL to the JSON Schema for this object.
83         format: uri
84         readOnly: true
85         type: string
86       id:
87         examples:
88           - 1
89         format: int64
90         readOnly: true
91         type: integer
92       jwt:

```

```

91     description: The JWT from the GoRace Admin app used to access the race
    ↪ data
92     examples:
93         -
    ↪ eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCIsI
94     type: string
95     writeOnly: true
96     name:
97     description: The name of the race. It does not need to match the name
    ↪ in the GoRace application.
98     examples:
99         - race1
100     maxLength: 20
101     minLength: 3
102     pattern: "[\\p{L}-_\\d]+"
103     type: string
104     required:
105         - name
106         - id
107     type: object
108 User:
109     additionalProperties: false
110     properties:
111         $schema:
112             description: A URL to the JSON Schema for this object.
113             format: uri
114             readOnly: true
115             type: string
116         githubToken:
117             description: The GitHub token of the user generated by CLI app.
118             examples:
119                 - ghp_1234567890abcdef1234567890abcdef12345678
120             pattern: gh[a-z]_[[:alnum:]]{36}
121             type: string
122             writeOnly: true
123         id:
124             examples:
125                 - 1
126             format: int64
127             type: integer
128         username:
129             description: The nickname of the user. Can contain any character
    ↪ except control characters.
130             examples:
131                 - user1
132             maxLength: 20
133             minLength: 3
134             pattern: "\\P{C}+"
135             type: string
136     required:
137         - username
138         - id
139     type: object
140 updateRaceJWTRequestBody:
141     additionalProperties: false
142     properties:
143         $schema:
144             description: A URL to the JSON Schema for this object.

```

```

145     format: uri
146     readOnly: true
147     type: string
148   jwt:
149     description: Refer to Race.JWT
150     examples:
151       -
152         ↪ eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCIsI
153     type: string
154     writeOnly: true
155   required:
156   - jwt
157   type: object
158   securitySchemes:
159     bearer:
160       bearerFormat: JWT
161       scheme: bearer
162       type: http
163   info:
164     title: GH2GR API
165     version: 0.1.0
166     openapi: 3.1.0
167     paths:
168       /races:
169         get:
170           description: Returns all races owned by the current user
171           operationId: list-races
172           responses:
173             "200":
174               content:
175                 application/json:
176                   schema:
177                     items:
178                       $ref: "#/components/schemas/Race"
179                     type: array
180                   description: OK
181             default:
182               content:
183                 application/problem+json:
184                   schema:
185                     $ref: "#/components/schemas/ErrorModel"
186                   description: Error
187           security:
188             - bearer: []
189           summary: List races
190           tags:
191             - races
192         post:
193           description: Creates a new race, making the current user the owner
194           operationId: create-race
195           requestBody:
196             content:
197               application/json:
198                 schema:
199                   $ref: "#/components/schemas/Race"
200             required: true
201           responses:
202             "201":

```



```

202     content:
203       application/json:
204         schema:
205           $ref: "#/components/schemas/Race"
206       description: Created
207     default:
208       content:
209         application/problem+json:
210           schema:
211             $ref: "#/components/schemas/ErrorMessage"
212       description: Error
213     security:
214       - bearer: []
215     summary: Add a race
216     tags:
217       - races
218 /races/{race-id}:
219 delete:
220   description: Remove a race from the database. All associated github
221   ↳ user name mappings will be deleted by it. Only an owner of the given
222   ↳ race can delete it.
223   operationId: delete-race
224   parameters:
225     - example: 42
226       in: path
227       name: race-id
228       required: true
229       schema:
230         description: The ID of the race to be deleted.
231         examples:
232           - 42
233         format: int64
234         type: integer
235   responses:
236     "204":
237       description: No Content
238     default:
239       content:
240         application/problem+json:
241           schema:
242             $ref: "#/components/schemas/ErrorMessage"
243       description: Error
244     security:
245       - bearer: []
246     summary: Delete a race
247     tags:
248       - races
249 get:
250   description: Retrieves the race with the given ID. The current user must
251   ↳ be an owner of the race
252   operationId: get-race
253   parameters:
254     - example: 42
255       in: path
256       name: race-id
257       required: true
258       schema:
259         description: The ID of the race that you wish to retrieve.

```

```

257     examples:
258       - 42
259     format: int64
260     type: integer
261 responses:
262   "200":
263     content:
264       application/json:
265         schema:
266           $ref: "#/components/schemas/Race"
267       description: OK
268     default:
269       content:
270         application/problem+json:
271           schema:
272             $ref: "#/components/schemas/ErrorModel"
273       description: Error
274     security:
275       - bearer: []
276     summary: Get a race
277     tags:
278       - races
279 /races/{race-id}/update-jwt:
280   put:
281     description: Updates the GoRace JWT token of a race you own.
282     operationId: update-race-jwt
283     parameters:
284       - example: 42
285         in: path
286         name: race-id
287         required: true
288         schema:
289           description: The ID of the race to be deleted.
290           examples:
291             - 42
292           format: int64
293           type: integer
294     requestBody:
295       content:
296         application/json:
297           schema:
298             $ref: "#/components/schemas/updateRaceJWTInputBody"
299       required: true
300     responses:
301       "204":
302         description: No Content
303         default:
304           content:
305             application/problem+json:
306               schema:
307                 $ref: "#/components/schemas/ErrorModel"
308             description: Error
309         security:
310           - bearer: []
311         summary: Update Race JWT
312         tags:
313           - races
314 /races/{race-id}/user-mappings:

```

```

315 post:
316   description: Map Github users to the email addresses to be used in GoRace.
    ↪ It sends preregistration emails to the email addresses. Sending a
    ↪ username that was previously associated with an email, will overwrite
    ↪ the previous value
317   operationId: add-github-user-mapping
318   parameters:
319     - example: 42
320       in: path
321       name: race-id
322       required: true
323       schema:
324         description: The ID of the race that will receive the github user
    ↪ mappings
325         examples:
326           - 42
327         format: int64
328         type: integer
329   requestBody:
330     content:
331       application/json:
332         schema:
333           items:
334             $ref: "#/components/schemas/GithubUserMapping"
335             type: array
336           required: true
337   responses:
338     "204":
339       description: No Content
340     default:
341       content:
342         application/problem+json:
343           schema:
344             $ref: "#/components/schemas/ErrorModel"
345           description: Error
346   security:
347     - bearer: []
348   summary: Add Github user mappings to a race
349   tags:
350     - github-user-mappings
351 /user:
352 get:
353   description: Retrieves the user data associated with the auth token used
    ↪ to make the request.
354   operationId: get-current-user
355   responses:
356     "200":
357       content:
358         application/json:
359           schema:
360             $ref: "#/components/schemas/User"
361           description: OK
362     default:
363       content:
364         application/problem+json:
365           schema:
366             $ref: "#/components/schemas/ErrorModel"
367           description: Error

```

```

368 security:
369   - bearer: []
370 summary: Get current user
371 tags:
372   - current-user
373 put:
374   description: Updates the user data associated with the auth token used to
    ↪ make the request.
375   operationId: update-current-user
376   requestBody:
377     content:
378       application/json:
379         schema:
380           $ref: "#/components/schemas/User"
381     required: true
382   responses:
383     "204":
384       description: No Content
385     default:
386       content:
387         application/problem+json:
388           schema:
389             $ref: "#/components/schemas/ModelError"
390       description: Error
391 security:
392   - bearer: []
393 summary: Update current user
394 tags:
395   - current-user

```