Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería del Software

Desarrollo de una propuesta para la gamificación de aprendizaje de SQL

Alumno:
Helio Fernández Abad

Tutores:
Diego García Álvarez
Miguel Ángel Martínez Prieto

...

Agradecimientos

Quiero dedicar unas frases para agradecer a todos los que me han ayudado en este trayecto.

En primer lugar, a mi familia: mi madre, mi padre y mi hermano, que siempre han estado conmigo incondicionalmente. Su cariño y apoyo es lo que me ha ayudado a dar paso tras paso.

A mis amigos, que han estado ahí tanto para compartir los buenos momentos como para oír y ayudar en los malos, no dudando nunca en apoyarme.

Y por último, pero no menos importante a mi tutor, Diego. Quien me ha ayudado y guiado durante esta última etapa de la carrera.

Resumen

SQL (*Structured Query Language*) es el lenguaje de gestión de bases de datos más usado en la actualidad [37], por ello, resulta natural que también se emplee con frecuencia como introducción a las bases de datos dentro de los estudios de informática. A pesar de tener gran uso y presencia dentro de la educación, la enseñanza de SQL también se enfrenta a muchos de los retos comúnmente presentes en la educación, como en el caso de la enseñanza universitaria, como la falta de motivación en los estudiantes, el bajo nivel de implicación y similares.

Una solución a estos problemas, o al menos una medida para reducir sus efectos, es la **gamificación**, que consiste en el uso de componentes típicos de juegos fuera de contextos lúdicos para, en el caso de la enseñanza, ayudar y facilitar el aprendizaje.

Este **T**rabajo de **F**in de **G**rado (**TFG**) nace de ambos factores, la relevancia de SQL y los beneficios de una correcta gamificación. Por todo esto, el objetivo de este TFG es la creación de una aplicación que proporcione soporte y sirva como apoyo a la enseñanza a nivel universitario de SQL, concretamente MySQL, de forma gamificada. Y esto lo hace en forma de una plataforma web que proporciona a sus usuarios una vía para crear y gestionar pseudo-aventuras conversacionales en las que se incorpora SQL como parte de su narrativa, así como la vía para poder jugar a dichas aventuras.

Palabras clave:
Gamificación, Función-Vista o Ruta, Nivel, Bloque, Test, SQLAlchemy, Flask, Docker, Contenedor de Evaluación.

Abstract

SQL (Structured Query Language) is the most widely used database management language nowadays [37], and thus, it's often used as an introduction to databases in any informatic/computer-science related curricula.
However, despite its extensive use and presence in the educative environment (at least within computer-science courses and so on), teaching SQL also has to face against many of the usual challenges found in any type of educational settings (and especially at higher levels such as the university settings), challenges such as student's lack of motivation, low levels of engagement and so on.
One solution to these issues (or at least a way to mitigate their effects) is **gamification**, which is based in the usage of elements traditionally exclusive to games and video-games outside of ludic/recreational settings in order to, in the teaching environment, aid and facilitate the learning experience.

This Bachelor's Thesis (BT) is born out of both of those factors, the relevance of SQL and the benefits of an adequate gamification. Due to the prior facts, this BT aims to create an application that provides support and acts as an aid to the teaching of SQL at college level, concretely MySQL, and in a gamified manner. It does so in the shape of a web platform that provides to its users both, a way to create and manage pseudo-conversational adventures where SQL is incorporated into their narrative, as well as providing the means by which to play said "adventures.

Keywords:
Gamification, View-Function or Route, Level, Block, Test, SQLAlchemy, Flask, Docker, Evaluation Container.

Índice general

Agradecimientos **III**

Resumen **V**

Abstract **VII**

Lista de figuras **XV**

Lista de tablas **XVII**

1. Introducción **1**

1.1. Contexto 1

1.2. Motivación 3

1.3. Objetivos 6

1.3.1. Objetivos Académicos 6

1.4. Estructura de la memoria 7

2. Estado del arte **9**

2.1. Gamificación 9

2.1.1. Diversión 10

2.1.2. Motivación 11

2.1.3. Compromiso 16

2.1.4. Flujo 16
2.1.5. Tipos de usuarios 17
2.1.6. Elementos de la gamificación: 21
2.1.7. Tipos de gamificación según Werbach y Hunter 24
2.1.8. Gamificación, ventajas e inconvenientes 24
2.1.9. Gamificación a lo largo de la historia 29

3. Planificación **31**

3.1. Metodología del proyecto 31
3.2. Análisis de Riesgos 32
3.3. Planificación 36
3.4. Presupuesto 37

4. Especificación de Requisitos **39**

4.1. Descripción del sistema 39
4.2. Requisitos 44
4.2.1. Requisitos Funcionales 44
4.2.2. Requisitos No-Funcionales 45
4.3. Requisitos de Información 46
4.4. Modelos de Casos de Uso 48
4.5. Especificación de Casos de Uso 50

5. Análisis **61**

5.1. Modelo de Dominio 61
5.2. Modelo de Análisis 63
5.2.1. Clases de Análisis 64
5.2.2. Realización de Casos de Uso de Análisis 65
5.2.3. Secuencia: Transcurso del Nivel - CU.Jugar un nivel 66

5.2.4. Secuencia: Evaluar respuesta - CU.Jugar un nivel 69

5.2.5. Secuencia: Preparar un nivel jugable - CU.Administrar Nivel, Administrar Bloque, Administrar Test 70

6. Diseño **73**

6.1. Arquitectura lógica del sistema 73

6.2. Arquitectura física del sistema 80

6.3. Patrones de diseño principales 81

6.3.1. *MVT* 82

6.3.2. *Page Controller* 83

6.3.3. *Template View* 84

6.3.4. *Data Mapper* 86

6.3.5. *Singleton* 86

6.3.6. *Decorator* 87

6.4. Realización de Caso de Uso en Diseño 87

7. Tecnologías y Herramientas **93**

7.1. Introducción 93

7.2. Herramientas de desarrollo 93

7.2.1. Visual Studio Code 93

7.2.2. Docker Desktop 94

7.2.3. Git 96

7.2.4. Google Chrome 96

7.3. Herramientas de diseño, modelado y documentación 96

7.3.1. Visual Paradigm 96

7.3.2. Overleaf 96

7.3.3. Crixet 96

7.4. Herramientas de comunicación 96

7.4.1. Microsoft Teams 97
7.4.2. Microsoft Outlook 97
7.5. Tecnologías utilizadas 97
7.5.1. Python 3 97
7.5.2. Flask 97
7.5.3. SQLAlchemy 97
7.5.4. Flask-SQLAlchemy 98
7.5.5. MySQL 98
7.5.6. mysql-connector-python 98
7.5.7. Jinja2 98
7.5.8. WTForms 98
7.5.9. Flask-WTF 98
7.5.10. Flask-Login 99
7.5.11. Flask-Bcrypt / bcrypt 99
7.5.12. itsdangerous 99
7.5.13. docker (Python SDK) 99
7.5.14. sqlglot 99
7.5.15. Werkzeug 99

8. Implementación **101**

8.1. Estructura del proyecto 102
8.2. Arquitectura física a nivel de implementación 112
8.3. Detalles de implementación 118
8.3.1. Docker-Compose 118
8.3.2. SQLAlchemy 119
8.3.3. MySQL-connector 121
8.3.4. Werkzeug 121

9. Pruebas **123**

9.1. Pruebas Unitarias 123

9.2. Pruebas de Sistema 124

9.2.1. Prueba *Registrar usuario* 125

9.2.2. Prueba *Iniciar sesión* 126

9.2.3. Prueba *Cerrar sesión* 127

9.2.4. Prueba *Acceder al menú de niveles* 128

9.2.5. Prueba *Jugar un nivel* 129

9.2.6. Prueba *Administrar nivel* 130

9.2.7. Prueba *Administrar bloque* 131

9.2.8. Prueba *Administrar Test* 132

10. Conclusiones **133**

10.1. Conclusiones 133

10.2. Trabajo Futuro 134

A. Resumen de enlaces adicionales **137**

Bibliografía **139**

Lista de Figuras

2.1. Diagrama mostrando la relación entre dificultad y habilidad y como afecta al flujo. Gráfica extraída de https://cinegamification.com/storytelling-gamification/the-flow-theory/. 17

2.2. Modelo de cuadrantes reflejando la taxonomía de Bartle. Imagen extraída de la entrada en español del siguiente artículo de la Wikipedia https://w.wiki/7mp7. 18

2.3. Clasificación de usuarios según su disposición a jugar (Marczewski). Imagen extraída del blog de Marczewski:https://www.gamified.uk/user-types/ . 19

2.4. Modelo de cuadrantes de los tipos motivados intrínsecamente (Marczewski). Imagen extraída del blog de Marczewski:https://www.gamified.uk/user-types/ 21

2.5. Pirámide de elementos de los juegos y su jerarquía. Imagen extraída del documento [68]. 21

3.1. Figura que representa la matriz de riesgo del proyecto. 32

4.1. Diagrama que refleja la jerarquia de los usuarios del sistema. 48

4.2. Diagrama de casos de uso en los que participa el actor Profesor. 49

4.3. Diagrama de casos de uso en los que participa el actor Estudiante. 49

5.1. Diagrama de clases del modelo de dominio del sistema. 63

5.2. Diagrama de secuencia reflejando la primera parte del escenario de un usuario jugando un nivel hasta completarlo. 67

5.3. Diagrama de secuencia reflejando la segunda parte del escenario de un usuario jugando un nivel hasta completarlo. 68

5.4. Diagrama de secuencia reflejando el proceso mediante el cual el sistema evalúa una respuesta de jugador. 69

5.5. Diagrama de secuencia reflejando la primera parte del proceso mediante el cual un Profesor crea y prepara un nivel para que esté listo para ser jugado. . 71

5.6. Diagrama de secuencia reflejando la segunda parte del proceso mediante el cual un Profesor crea y prepara un nivel para que esté listo para ser jugado. . 72

6.1. Diagrama de Paquetes que representa la organización de capas del sistema, reflejando la arquitectura lógica del mismo. 77

6.2. Diagrama de despliegue de diseño del sistema. 81

6.3. Diagrama representando el patrón de diseño Model-View-Template. 82

6.4. Diagrama representando el patrón de diseño *Page Controller*. 83

6.5. Diagrama representando el patrón de diseño *Template View*. 84

6.6. Diagrama representando el patrón de diseño Data Mapper. 86

6.7. Diagrama de secuencia de diseño reflejando la primera parte del proceso de evaluación de la respuesta de un jugador. 89

6.8. Diagrama de secuencia de diseño reflejando la segunda parte del proceso de evaluación de la respuesta de un jugador. 90

6.9. Diagrama de secuencia de diseño reflejando la tercera parte del proceso de evaluación de la respuesta de un jugador. 91

6.10. Diagrama de secuencia de diseño reflejando la última parte del proceso de evaluación de la respuesta de un jugador. 92

8.1. Diagrama de despliegue detallado, refleja el hardware del sistema. 115

Lista de Tablas

3.1. Riesgos de nivel Insignificante del Plan de Contingencia del proyecto. 33

3.2. Riesgos de nivel Menor del Plan de Contingencia del proyecto. 33

3.3. Riesgos de nivel Moderado del Plan de Contingencia del proyecto. 34

3.4. Riesgos de nivel Alto y Crítico del Plan de Contingencia del proyecto. 35

3.5. Tabla mostrando la estimación de las horas dedicadas a las distintas fases del desarrollo del proyecto. 36

4.1. Especificación del caso de uso "Registrarse en la aplicación". 50

4.2. Especificación del caso de uso "Iniciar sesión". 51

4.3. Especificación del caso de uso "Cerrar sesión". 52

4.4. Especificación del caso de uso "Jugar Nivel". 54

4.5. Especificación del caso de uso "Administrar Nivel". 56

4.6. Especificación del caso de uso "Administrar Bloque". 58

4.7. Especificación del caso de uso "Administrar Test". 60

9.1. Prueba del caso de uso "Registrar usuario". 125

9.2. Prueba del caso de uso "Iniciar sesión". 126

9.3. Prueba del caso de uso "Cerrar sesión". 127

9.4. Prueba del caso de uso "Acceder al menú niveles". 128

9.5. Prueba del caso de uso "Jugar un nivel". 129

9.6. Prueba del caso de uso "Administrar nivel". 130

9.7. Prueba del caso de uso "Administrar bloque". 131

9.8. Prueba del caso de uso "Administrar test". 132

Capítulo 1

Introducción

1.1. Contexto

La enseñanza, a cualquier nivel, presenta una serie de desafíos (extensión del contenido, nivel de dificultad del mismo, etc.) y, de entre ellos, este TFG se enfoca en aquellos relacionados con **el interés del estudiante**. Esta problemática se resume en "**¿Cómo conseguir que los estudiantes se interesen por lo que se está enseñando?**". La primera idea que le puede venir a alguien a la cabeza al plantearse esa pregunta es el contenido en sí, él "**¿Qué se enseña?**", sin embargo, en el caso de este TFG, **SQL** no es algo que se pueda redefinir. No es posible reinventar SQL para convertirlo en algo que no es, además tampoco se dispone de muchas opciones enfocándose en cambiar las partes de SQL que se enseñan.

A las dificultades propias de SQL se suman los desafíos típicos de la enseñanza, presentes independientemente de lo que se esté enseñando. Estos son factores como el grado de abstracción de lo que se pretende enseñar, la dificultad de lo que se quiere enseñar, la cercanía o familiaridad de cada estudiante con los conceptos a enseñar, e incluso aspectos totalmente externos a lo que se esté enseñando como pueden ser circunstancias familiares, estado anímico y psicológico del estudiante, incluso situaciones tan banales como que un estudiante esté saturado tras muchos años de estudios.

Si a las dificultades comunes de la enseñanza se le suman las intrínsecas de SQL, resulta que, en este escenario, la siguiente mejor opción es enfocarse en "**el cómo**", centrarse en la manera en que SQL es enseñado. Es en este punto donde entra la **Gamificación** (o **Ludificación**) en juego.

La **Gamificación** es una técnica de enseñanza cuyo objetivo es precisamente generar interés en los estudiantes o, al menos, fomentar que, entre aquellos que presentan ya cierto interés en lo que se va a enseñar, este interés aumente. El término gamificación surge en 2002 [67], cuando el programador informático **Nick Pelling** lo acuña mientras estaba diseñando una interfaz de usuario *game-like* (la traducción literal es "similar o parecido a juego") para ser usada en dispositivos electrónicos de ámbito comercial (máquinas expendedoras,

cajeros electrónicos...). Como concepto, la gamificación existe desde mucho antes de 2002, pudiendo considerarse la creación de los "*Boy Scouts*" en 1908 [17] como el primer evento documentado en el que se aplicaron, aunque igual de manera no intencional, las bases de lo que es la gamificación. Aunque los primeros indicios de gamificación registrados datan sobre 1908 y el término como tal no se acuñó hasta 2002, no fue hasta la década de 2010 que la gamificación no solo empezó a ser vista como técnica a nivel general, sino que fue en esa época cuando su popularidad se disparó, particularmente en 2011 [17].

Más adelante se profundizará sobre qué es y en qué consiste la gamificación. Por ahora, se definirá la gamificación como la incorporación de elementos de juegos en actividades no lúdicas para aumentar la motivación y fomentar el interés y participación de los participantes en dichas actividades. Esta técnica resulta ideal para el proyecto, ya que la incorporación de esos elementos a la enseñanza de SQL permite obtener los beneficios antes mencionados siguiendo prácticas y pautas probadas a lo largo de los años, con el beneficio adicional de la abundancia, actualmente, de documentación al respecto y recursos sobre ello *online*.

La idea u objetivo del proyecto es combinar **SQL** y **Gamificación** en forma de una aplicación educativa y gamificada, en la que se permita a **Profesores** crear una serie de niveles, que funcionan como una mezcla entre los denominados *librojuegos* [10] y una *aventura conversacional* [7] (o, dependiendo de a quién se pregunte, una aventura gráfica [8][1]). No solo crearlos, sino también configurarlos/editarlos al gusto y eliminarlos. Por otro lado, la aplicación permitirá también a los **Estudiantes** jugar a estos niveles, los cuales contienen puntos de las distintas historias que se desarrollan en cada nivel, en los que al jugador le serán planteadas preguntas sobre MySQL, las cuales debe resolver para poder continuar/avanzar en el nivel.

[1] El motivo de este comentario es que, tradicionalmente, las aventuras conversacionales o *text-based adventures* son juegos en los que la interacción entre el jugador y el juego es **totalmente** vía teclado, ningún otro periférico participa, mientras que las aventuras gráficas **sí** que involucran al ratón y hacen participar al aspecto gráfico, sin embargo, tradicionalmente, las aventuras gráficas como *point and click* y similares, incluyen imágenes que reflejan el entorno en que se encuentra el personaje, y no son escenarios tan simples como en los de esta aplicación, en los que el ratón participa sencillamente para interactuar con el servicio web, y es que, al ser una aplicación gamificada y **no** un videojuego, no hay una clara distinción entre si sería más correcto decir que se asemeja a una aventura conversacional, o a una aventura gráfica.

1.2. Motivación

FreeWheeling Travelers (FWT) es una actividad gamificada desarrollada por el profesor y tutor *Miguel Ángel Martínez Prieto* y *Jorge Silvestre*, que plantea al estudiante una narrativa de gestión y realización de viajes a través de una supuesta aplicación llamada también FreeWheeling Travelers utilizando SQL. Por otro lado, *CodeRunner* [27] es un *plugin* de *Moodle* que permite, entre otras cosas, la evaluación y calificación automática de preguntas preparadas para los estudiantes.

El tema original planteado para este TFG iba a ser extender el proyecto inicial FWT más allá de ser una actividad aislada, buscando su implementación en Moodle a través del antes mencionado CodeRunner, tratando de incorporar con la mayor fidelidad posible las características originales de FWT (incluyendo la base de datos compartida entre todos los participantes) al entorno Moodle y añadiendo las funcionalidades de CodeRunner para permitir la evaluación automática de las respuestas de los estudiantes sin necesitar involucrar a un profesor para corregirlas. **Sin embargo**, tras un periodo considerable de tiempo y desarrollo de dicho proyecto, se alcanzó la conclusión que el tema, por las limitaciones tanto de Moodle como del *plugin* CodeRunner, más factores como el tiempo que resultaría necesario para ello y la formación necesaria, resultaba inviable como TFG si se quisiese implementar como se planeó (incluso ignorando la cantidad excesiva de tiempo que sería necesario para que el proyecto alcanzase una envergadura suficiente, no se podía garantizar al 100 % que fuese posible desarrollar tal actividad) y que, tratar de replantear el proyecto como un proyecto de investigación sobre el porqué no resultaba viable, el resultado podría no ser suficiente en cuanto a contenido y complejidad como para su presentación como TFG. Por ello, ante la recomendación del tutor, se cambió el tema del TFG.

El tema final del proyecto, todavía influenciado por el antes mencionado FWT, sigue siendo crear una actividad gamificada que sirva como refuerzo a la enseñanza de SQL a nivel universitario; sin embargo, en vez de hacerlo desde Moodle mediante CodeRunner, pasa a materializarse como una aplicación web basada en Python y Flask. Se han mantenido de FWT el concepto de los viajes, emulando cada actividad un escenario distinto en el que el estudiante va de vacaciones a un destino nuevo, pero sin forzar la temática viajera. En el producto final, el profesor tiene libertad para escribir y desarrollar cualquier ambientación que desee gracias a que los **Niveles** (equivalentes a los viajes de FWT) son altamente configurables y personalizables y, entre otras múltiples opciones, permiten al usuario escribir bloque a bloque (siendo **Bloque** el nombre dado a la unidad o componente básico del que está compuesto cada **Nivel**) cualquier narrativa o historia, siendo el usuario además el responsable de garantizar que la trama (tanto la global del nivel como sus distintas ramificaciones en función de las decisiones del jugador a lo largo del nivel) sea coherente, consistente y bien hilada.

También se ha heredado de FWT (no de la actividad original, sino del tema inicial del TFG sobre extender FWT) el objetivo de que la aplicación web sea capaz de evaluar autónomamente y sin participación humana las respuestas o soluciones de los estudiantes. El sistema debe ser capaz de identificar si cada respuesta es correcta o errónea y, en caso de detectar una respuesta errónea, proporcionar al alumno una retroalimentación clara sobre el motivo del error, para que pueda corregir su respuesta hasta dar con una solución.

Cabe destacar que este último aspecto se ha extendido aún más allá de lo planeado

originalmente, ya que este proyecto no solo evalúa de forma autónoma y programática, sino que, frente a implementaciones típicas de sistemas de evaluación automática, que suelen estar basadas en *expresiones regulares* y simplemente comparan el *input* del alumno con una o varias respuestas predefinidas, escogidas arbitrariamente por el autor de la pregunta, y declaran la respuesta correcta o incorrecta según esa coincidencia, sin verificar si realmente resuelve el problema planteado por la pregunta, **en este proyecto** se evalúan las respuestas ejecutando cada propuesta para, acto seguido, comprobar los efectos de la propuesta y así poder ciertamente determinar si la respuesta proporcionada soluciona el problema planteado en la pregunta o no.

Este método de evaluación también permite ajustar el grado de rigor con el que se desea evaluar las respuestas a las distintas preguntas: según cuántos efectos se estén buscando y cuáles sean los que el sistema espera encontrar, se puede regular no solo la certeza con la que el sistema califica una respuesta como correcta o incorrecta, sino que permite también, cuando es posible solucionar el problema de varias formas distintas, gestionar que soluciones son las buscadas y cuáles no.

A continuación se va a proporcionar un ejemplo de lo que podría ser una pregunta cualquiera de un nivel creado en la aplicación, aunque para ser breve no se va a escribir el enunciado siguiendo ninguna narrativa o temática (en un nivel o pregunta real en la aplicación, la pregunta debería hacerse siguiendo la temática e hilo narrativo del nivel al que pertenece[2]):
Disponemos de una tabla Nombre_tabla con columnas X, Y y Z. Escribe un statement SQL que haga que dicha tabla contenga una entrada con los valores valor_x, valor_y, valor_z.

Este ejemplo básicamente quiere que se añada una entrada con unos atributos concretos a la tabla especificada e, inmediatamente, la primera solución que suele venir a alguien a la cabeza es mediante un statement `INSERT INTO Nombre_tabla (X, Y, Z) VALUES (valor_x, valor_y, valor_z);`. Y esto sería correcto, ya que, en base a lo que pide la pregunta, esa respuesta añadiría a la tabla una fila más, y los valores de los atributos de esa fila coinciden con los solicitados por el enunciado de la pregunta. Sin embargo, **no es la única solución** correcta, un *statement* `UPDATE Nombre_tabla SET X=valor_x, Y=valor_y, Z=valor_z;` resultaría en que **todas** las filas de la tabla coincidiesen con las solicitadas por el enunciado, y aunque la primera sensación al leer esto es que esa respuesta es incorrecta, ese **no** es el caso, la respuesta, literalmente, ha hecho lo pedido por el enunciado, que es que en la tabla especificada haya una entrada con los datos proporcionados.

El ejemplo demuestra que, una misma pregunta, puede ser solucionada de formas distintas (dos en el ejemplo) y con dicho ejemplo se busca aclarar lo **especial** del proyecto. En un sistema como los mencionados al principio, de evaluación automática basada en *expresiones regulares*, si el autor de la pregunta del ejemplo olvida incluir un *statement* `UPDATE Nombre_tabla...;` cualquier alumno cuya respuesta sea mediante la operación `UPDATE` será informado por el sistema que su respuesta es incorrecta cuando, en realidad, **no lo es**. Esto es un problema bastante grave, ya que: significa que el sistema no ha sido capaz de reconocer una respuesta correcta, al indicar al alumno que su respuesta es incorrecta cuando no es cierto significa que el sistema, aunque no de manera intencionada, ha **empeorado** los

[2]Se comentará más en detalle a lo largo del documento, pero las preguntas son instancias de los antes mencionados **Bloques** que componen cada nivel.

conocimientos sobre SQL del alumno, ya que este, a partir de dicha evaluación, va a creer, incorrectamente, que `UPDATE` no solucionaría el problema planteado, por último, en función de otros factores como el estado de ánimo del alumno y con que frecuencia se encuentre con situaciones en las que se declara su propuesta como incorrecta cuando es correcta[3], se puede causar el efecto contrario al deseado y reducir la motivación del alumno y su interés en la actividad y en SQL.

El sistema propuesto en este proyecto **no es inmune al error humano**. Sin embargo, su estrategia de evaluación se basa en los efectos que produce cada respuesta al ejecutarse, lo cual introduce una **diferencia clave** respecto a los enfoques basados en *expresiones regulares*. En un sistema convencional, las respuestas del alumno se comparan directamente con una lista predefinida de cadenas de texto: si el *input* coincide con alguna de ellas, se considera correcto, sin analizar qué sucede al ejecutarlo.

En cambio, en el sistema propuesto en este proyecto, el **autor de la pregunta** tiene también la responsabilidad de indicar, durante la creación de la misma, **qué efectos** deben observarse para que una respuesta se considere **válida** y cuáles deben evitarse. De este modo, antes de incorporar la pregunta al sistema, el autor **debe pensar explícitamente en cómo va a diseñar la evaluación**, definiendo con claridad los criterios basados en resultado y no solo en texto. Gracias a ello, resulta mucho **más difícil** que una solución técnicamente correcta sea evaluada como incorrecta, salvo que el autor de la pregunta, de manera deliberada, establezca como no deseados los efectos causados por las otras soluciones.

Además, este mecanismo de evaluación permite ofrecer al alumno información detallada sobre su error. Al comprobar qué efecto falta o qué efecto inesperado aparece tras la ejecución, el sistema puede proporcioar *feedback* específico sobre el error detectado en la respuesta del alumno. Esto facilita que el alumno entienda en qué se ha equivocado y pueda corregir su respuesta según corresponda.

En resumen, la motivación de este proyecto parte de la premisa de que la gamificación, bien aplicada, resulta beneficial al aprendizaje y que, un sistema como el propuesto, que ofrece a los docentes las herramientas necesarias para diseñar actividades gamificadas, controlando tanto la flexibilidad sobre qué se pregunta, qué se considera respuesta correcta y el nivel de detalle del *feedback*, como la precisión con la que se evalúan esas respuestas, en conjunto con los beneficios de una correcta gamificación, resultaría extremadamente beneficial para la enseñanza de SQL.

Además, el sistema proporciona a los alumnos la vía para jugar a estos niveles en forma de retos interactivos de modo que los estudiantes perciban la actividad no como una obligación, sino como una experiencia entretenida y motivadora. De este modo, se potencia la enseñanza de SQL y se facilita la incorporación de elementos de gamificación en cualquier curso.

Por último, habiendo sido expuestas las peculiaridades de este proyecto, así como las razones detrás de su diseño, hay que mencionar los siguientes dos puntos:

- Como autor de una pregunta, es importante procurar que, en escenarios en los que se podría solucionar el problema planteado de varias formas distintas, el enunciado del

[3] Independientemente de si el alumno sabe o, al menos, cree que su respuesta **es correcta**.

problema refleje adecuadamente cuál, de entre las posibles soluciones, es la deseada.

- Por si no hubiese quedado claro, la aplicación no solo sirve como vía para que los profesores puedan gestionar estos niveles o actividades, sino también es la plataforma mediante la cual los alumnos podrán experimentar dichas actividades.

1.3. Objetivos

1. Desarrollar una aplicación web Python-Flask que permita a usuarios con las credenciales y conocimientos suficientes (Profesores) crear actividades gamificadas que el resto de usuarios puedan jugar, así como la vía para que el resto de usuarios puedan jugarlos. Incorporando estas actividades (niveles) preguntas sobre SQL para poner a prueba los conocimientos de estos sobre lo enseñado en clase.
2. Estudiar sobre gamificación para su incorporación al proyecto.
3. Estudiar el funcionamiento y uso del micro-framework **Flask**.
4. Conseguir que la actividad resultante despierte el interés de los alumnos y promueva la participación e implicación de los mismos.
5. Garantizar la correcta evaluación de los estudiantes (a prueba de trampas).

Antes de continuar conviene aclarar que esta aplicación no está diseñada para ser usada como medio de enseñanza de SQL, sino que busca apoyar y asistir como complemento a las sesiones magistrales impartidas en las respectivas asignaturas en las que los profesores enseñarán SQL a sus alumnos.

En otras palabras, la aplicación **no** pretende enseñar SQL a los usuarios, sino que está pensada para actuar como una ayuda y un apoyo a las asignaturas en cuyos contenidos se estudie SQL, de forma que, a medida que los estudiantes tratan de superar los niveles, se familiaricen con el uso de SQL. Además, el tener que enfrentarse a las preguntas les permite: poner sus conocimientos a prueba, corregir aspectos que pueden no haber entendido del todo y, en general, ayuda a afianzar y asentar lo aprendido en las lecciones.

1.3.1. Objetivos Académicos

1. Aprender a utilizar la herramienta **Docker** que permite simular un entorno con acceso a múltiples servidores sin requerir los gastos asociados a disponer de servidores físicos, muy útil en este caso por ser la aplicación, no un producto real, sino un proyecto académico.
2. Profundizar los conocimientos sobre SQL.
3. Profundizar los conocimientos sobre servicios Web.
4. Que el producto final del proyecto satisfaga los requisitos de seguridad.

1.4. Estructura de la memoria

Este documento se estructura de la siguiente forma:

Capítulo 1 Introducción: En este capítulo se presenta el contexto y la motivación del proyecto, se define el problema que se aborda, los objetivos generales y específicos, y se explica la estructura global de la memoria.

Capítulo 2 Estado del arte: En este capítulo se revisan y analizan los trabajos, artículos, teorías y herramientas más relevantes relacionados con el tema del Proyecto. En el caso de este TFG, el contenido del capítulo se centra en la gamificación como técnica o herramienta para incrementar el interés y motivación.

Capítulo 3 Planificación: En este capítulo se describe la metodología seguida durante el desarrollo del trabajo, se muestra el Plan de Contingencia del proyecto, la estimación de tiempo dedicado a completar el proyecto y por último una estimación del coste que podría haber acarreado el desarrollo del proyecto si se hubiese llevado a cabo en el entorno laboral.

Capítulo 4 Especificación de Requisitos: En este capítulo se proporciona una descripción más detallada del sistema, se detallan los requisitos funcionales, no-funcionales y de información del proyecto y finalmente se exponen los casos de uso del proyecto junto con la especificación de cada uno de ellos.

Capítulo 5 Análisis: En este capítulo se despliega el modelo de análisis del proyecto. Se muestra el modelo de dominio del proyecto y los diagramas de secuencia de los tres eventos más relevantes del proyecto.

Capítulo 6 Diseño: En este capítulo se despliega el modelo de diseño del proyecto. Se describen la arquitectura lógica y física del sistema, los patrones de diseño más relevantes y finalmente un diagrama de secuencia detallado.

Capítulo 7 Tecnologías y Herramientas: En este capítulo se listan las herramientas y tecnologías involucradas en el desarrollo del proyecto. Proporcionando una breve descripción de cada una de ellas junto con una mención de su función y papel en el sistema.

Capítulo 8 Implementación: En este capítulo se presenta cómo se ha materializado el proyecto, estructura del código, división de la funcionalidad entre los componentes junto a fragmentos del código como ejemplo.

Capítulo 9 Pruebas: En este capítulo se describen las pruebas realizadas para comprobar la aplicación desarrollada. Cada una de estas descripciones incluye no solo la prueba realizada, sino el caso de uso asociado y el resultado de la prueba.

Capítulo 10 Conclusiones: En este capítulo se evalúan los objetivos alcanzados y aquellos puntos del resultado que podrían mejorarse o extenderse como trabajo de futuro.

Capítulo 2

Estado del arte

A continuación se presentan algunos de los conceptos básicos sobre los que se desarrolla este proyecto:

2.1. Gamificación

La ludificación, o *gamificación*, consiste en la aplicación de elementos, mecánicas y dinámicas propias de los juegos en un entorno no lúdico, con el objetivo de potenciar la motivación, implicación, interés y eficacia en la consecución de determinados objetivos.

La *gamificación* aprovecha la predisposición humana al juego, buscando que los individuos asocien la sensación de jugar con actividades orientadas a alcanzar metas específicas, como el ejercicio físico, la adquisición de habilidades o el aprendizaje. Sin embargo, este proceso se lleva a cabo sin llegar a crear un juego propiamente dicho. En caso contrario, se trataría de desarrollo tradicional de juegos (cuando el único objetivo es la diversión) o de *serious games* (juegos serios) cuando existe un contexto no exclusivamente lúdico. Aunque ambos conceptos comparten similitudes, no son equivalentes a una actividad gamificada.

Una forma sencilla de diferenciar ambos conceptos es la siguiente: un juego serio es un juego que incorpora contenido educativo, es decir, una actividad en la que se integra material educativo en un juego. Por el contrario, una actividad gamificada es una actividad no lúdica (como puede ser un test o una tarea laboral) a la que se añaden elementos de juegos. Así, la *gamificación* implica la incorporación de elementos de juego en una actividad no lúdica, mientras que un juego serio es un juego diseñado explícitamente para la enseñanza o el aprendizaje.

La *gamificación* es un concepto complejo en el que intervienen múltiples disciplinas (psicología, economía, diseño y desarrollo de juegos[1]), en la que se presentan, tanto elementos

[1]Esta última no es literal, ya que gamificar no es lo mismo que crear un juego; sin embargo, comparten

de naturaleza menos tangible y más psicológica y/o emocional, asociados a la naturaleza humana, como elementos técnicos, propios del diseño y desarrollo de juegos. En concreto, se van a presentar los conceptos de: **diversión**, **motivación**, **compromiso**, **flujo** y **los tipos de usuarios**.

Al final del capítulo se detallan las ventajas y desventajas de la gamificación y una breve sección sobre la historia de la gamificación.

2.1.1. Diversión

La diversión constituye uno de los puntos clave de la *gamificación*, aunque a menudo tiende a ser ignorada en la práctica, lo que puede derivar en intentos fallidos de implementación. Esta situación se debe, en parte, a una visión superficial de la *gamificación*, entendida únicamente como la utilización de elementos, mecánicas y dinámicas de juegos en actividades no lúdicas. Esta perspectiva trivializa el complejo proceso de diseño exitoso de juegos, asumiendo que basta con añadir dichos elementos para lograr una gamificación adecuada. En estos casos, los sistemas resultantes no solo pueden ser poco atractivos, sino incluso menos motivadores que la actividad original.

La importancia de la diversión se pone de manifiesto al considerar que, en última instancia, los juegos funcionan porque resultan divertidos. Un sistema gamificado se construye sobre la predisposición humana al juego, buscando generar la sensación de juego en un contexto no lúdico y, con ello, provocar diversión.

Nicole Lazzaro define en *4 keys 2 fun* [15], en el ámbito de los juegos y la *gamificación*, cuatro tipos de diversión [66]:

- **Diversión fácil** (*easy fun*): Diversión producida por la novedad y la vivencia de nuevas experiencias. Fomenta la exploración, la imaginación y la creatividad. Un ejemplo fuera del ámbito de la *gamificación* serían las sensaciones experimentadas al pasear y descubrir cosas nuevas. Este tipo de diversión depende en gran medida de la autonomía y libertad otorgadas al usuario, así como del grado de control ejercido sobre el mismo.
- **Diversión difícil** (*hard fun*): Diversión asociada a retos y dificultades, especialmente al proceso de adquirir y mejorar habilidades y conocimientos para superar dichos retos, incluyendo la satisfacción derivada de superarlos.
- **Diversión con personas** (*people fun*): Diversión que surge de la interacción, cooperación, competición y relaciones sociales en general. Facilitar la creación de lazos sociales en los sistemas gamificados resulta útil para incrementar la implicación y el compromiso de los usuarios.
- **Diversión seria** (*serious fun*): Este tipo de diversión se asocia con sensaciones de calma y relajación, y se presenta en actividades que influyen en el individuo tanto dentro como fuera del sistema gamificado, haciendo que el usuario perciba que está

pasos y elementos en forma de dinámicas, mecánicas y componentes de juegos. No obstante, el producto final de una gamificación correcta no puede ser un juego.

generando un cambio. Según Nicole Lazzaro en *4 keys 2 fun*, la diversión seria proviene de jugar con un propósito, inducida por sensaciones de alegría y tranquilidad asociadas al aprendizaje voluntario, el crecimiento personal o la adquisición de mejores hábitos. Por este motivo, muchas actividades que en otros contextos serían monótonas o tediosas resultan amenas y, en cierta medida, entretenidas o relajantes cuando se realizan por voluntad propia. Algunos ejemplos de diversión seria son:

- Participar en un curso para aprender un idioma que no es necesario para el trabajo o por obligación, sino por interés personal, lo que genera emociones positivas.
- Colaborar en un banco de alimentos u otros eventos caritativos, donde la satisfacción proviene de contribuir a un cambio positivo, aunque no haya un beneficio personal inmediato.
- En videojuegos del género MMO (*Massively Multiplayer Online*), es común realizar tareas repetitivas al inicio, pero los jugadores las llevan a cabo con entusiasmo porque persiguen un objetivo propio, como mejorar su equipamiento o rango, lo que genera sensaciones positivas asociadas a la diversión seria.

Nicole Lazzaro indica que cada uno de los cuatro tipos de diversión corresponde a una emoción humana fundamental: la diversión fácil se asocia a la curiosidad y el asombro; la diversión difícil, a la frustración y al *fiero*[2] experimentado al superar un desafío; la diversión con personas, al entretenimiento y la alegría; y la diversión seria, a la expectación, anticipación y alivio.

Por último, resulta relevante mencionar la paradoja de la diversión obligatoria [75], que parte de la base de que los juegos resultan divertidos por la espontaneidad y sorpresa asociadas, así como por el hecho de que se realizan de manera voluntaria. En el caso de la *gamificación*, por su propia naturaleza, las actividades pueden resultar entretenidas, pero no surgen del deseo de los participantes, sino que se organizan e imponen desde fuera (por ejemplo, por la dirección de una empresa). Esto puede generar respuestas mixtas entre los participantes, ya que algunos lo aceptan y disfrutan, mientras que otros pueden ignorarlo o incluso sentirse incómodos o molestos, lo que resulta contraproducente. Para evitarlo, es fundamental contar con el consentimiento de los participantes, de modo que perciban la iniciativa como algo voluntario y de interés, y no como una imposición o una muestra de control por parte del sistema.

2.1.2. Motivación

La motivación puede definirse como el conjunto de factores internos y externos que impulsan a un individuo a actuar de una determinada manera. Kevin Werbach y Dan Hunter [74] comparan la motivación en las personas con la inercia en los objetos: las personas presentan una inercia que debe ser superada para que actúen, y la motivación es la fuerza que permite superar dicha inercia.

[2] *Fiero* es un término italiano que significa orgullo asociado a la sensación de placer o satisfacción ante los logros, posesiones, conexiones y similares de uno mismo.

Antes de continuar, conviene aclarar que el contenido de esta sección proviene, principalmente, del libro escrito por los antes mencionados autores [74].

Para comprender la motivación y su papel en la gamificación, es útil considerar dos teorías psicológicas fundamentales: el conductismo y el cognitivismo.

Conductismo: El conductismo sostiene que el comportamiento humano consiste en respuestas ante estímulos externos, sin analizar los procesos internos que llevan a una determinada reacción. Con la aparición del *condicionamiento operante*, se introduce la importancia de las consecuencias: en función de si las consecuencias de una acción son favorables o desfavorables, el individuo aprende a repetir o evitar ciertos comportamientos. Así, la motivación puede ser reforzada mediante recompensas o premios, y también mediante castigos, aunque el conductismo se centra principalmente en los refuerzos positivos.

Este proceso de aprendizaje del comportamiento se basa en tres puntos clave: la observación de las acciones y respuestas de los participantes, el refuerzo de estímulos para fomentar asociaciones entre acciones y consecuencias, y los bucles de realimentación, en los que se observa la respuesta del individuo ante la retroalimentación recibida. Por ello, la retroalimentación y el uso de recompensas tienen gran importancia en la gamificación, ya que permiten guiar el comportamiento de los usuarios hacia acciones deseadas.

El conductismo clasifica las recompensas en varios niveles:

- **Tangibles e intangibles**: Las recompensas tangibles son materiales (por ejemplo, sueldos o regalos), mientras que las intangibles son inmateriales (elogios, aumentos de estatus, logros digitales).
- **Esperadas e inesperadas**: Las recompensas pueden ser conocidas de antemano por el usuario o entregarse por sorpresa, lo que afecta a su impacto motivador.
- **Contingentes**: Según su relación con la tarea, pueden ser no contingentes (no dependen de ninguna acción), contingentes a la participación, a la finalización o al rendimiento en una tarea.

Además, el conductismo introduce la noción de *calendario de recompensas* (*reward schedule*), es decir, las reglas que determinan cuándo y con qué frecuencia se otorgan recompensas. Las principales categorías según el momento en que se otorgan son:

- **Recompensas continuas**: Se otorgan cada vez que se cumple una condición, de forma constante y regular. Tienden a perder su efecto motivador rápidamente, ya que el individuo se acostumbra a recibirlas.
- **Recompensas de ratio fijo**: Se entregan cada cierto número de veces que se cumple la condición (por ejemplo, cada 3 tareas completadas). Mantienen el interés durante más tiempo, pero también pueden volverse predecibles.
- **Recompensas de intervalo fijo**: Se otorgan tras pasar un periodo de tiempo determinado, independientemente de la cantidad de tareas realizadas (por ejemplo, cada 10 minutos de actividad).

- **Recompensas variables**: Se entregan de manera impredecible, sin un patrón fijo, lo que puede aumentar el interés y la motivación, especialmente si se combinan con elementos competitivos y no competitivos, esperados e inesperados.

Las recompensas variables, que combinan diferentes criterios, suelen ser las más efectivas para mantener la motivación, ya que introducen novedad y sorpresa. Sin embargo, un uso exclusivo de recompensas competitivas puede generar monotonía y desmotivación en quienes no suelen obtenerlas, mientras que la ausencia de competición puede resultar poco estimulante para quienes buscan retos. Por ello, se recomienda emplear una combinación de múltiples tipos de recompensas para maximizar su efecto motivador.

En resumen, el conductismo aporta las bases para el uso de la retroalimentación y las recompensas en la gamificación, aunque su enfoque es limitado al no considerar factores internos como emociones y deseos.

Cognitivismo: Frente al conductismo, el cognitivismo analiza el comportamiento humano considerando los procesos internos, como deseos, emociones y sentimientos. Esta teoría busca comprender por qué determinados estímulos o refuerzos provocan ciertas reacciones. El cognitivismo introduce la distinción entre recompensas extrínsecas, externas al individuo, como premios o reconocimientos, e intrínsecas, internas, como el interés o la satisfacción personal.

Según Gabe Zichermann y Christopher Cunningham [76], las recompensas extrínsecas pueden satisfacer cuatro deseos principales, conocidos como *SAPS* (*Status, Access, Power, Stuff*): estatus, acceso, poder y bienes materiales. Sin embargo, el uso excesivo de recompensas extrínsecas puede reducir la motivación intrínseca, especialmente si son tangibles, esperadas o relacionadas con el poder. Por ejemplo, recompensas materiales o de poder pueden llegar a ser desmotivadoras si desplazan el interés genuino por la actividad.

Las recompensas intrínsecas, por el contrario, son aquellas que surgen del propio individuo y no requieren de incentivos externos. En estos casos, la motivación proviene del disfrute o interés en la propia actividad, sin tener en cuenta las consecuencias externas.

Del cognitivismo surge la *Self-Determination Theory* (**SDT**) de Deci y Ryan [18], que sostiene que la motivación intrínseca es más eficaz y sostenible, y que los individuos tienen un deseo interno de crecer y desarrollarse. Esta teoría distingue entre la ausencia de motivación, la motivación extrínseca (impulsada por factores externos) y la motivación intrínseca (impulsada por el interés o satisfacción en la propia actividad).

Estados y tipos de motivación. A partir de estas teorías, se pueden distinguir tres estados principales en el espectro de la motivación:

1. **Sin motivación**: La ausencia de motivación es un problema en sistemas gamificados, ya que indica que es necesario replantear el diseño o incluso cuestionar la viabilidad de gamificar ciertas actividades. Las causas pueden ser diversas, como la falta de percepción de utilidad o la sensación de incapacidad para alcanzar los objetivos.
2. **Motivación extrínseca**: El factor motivador proviene del exterior del individuo, como recompensas, castigos o normas. Es el tipo de motivación más frecuente, especialmente

en el ámbito laboral, y puede clasificarse en varios subtipos según el grado de autonomía percibido por el individuo, siguiendo la *Organismic Integration Theory* (*OIT*), una subteoría de la SDT:

- **Regulación externa**: La acción se realiza por motivos totalmente externos, como recompensas o castigos, con mínima voluntariedad.
- **Introyección**: El comportamiento se adapta a lo solicitado, pero el individuo no lo considera propio; suele estar asociado a la autoestima o al estatus.
- **Identificación**: El individuo reconoce el valor de la acción y la realiza porque la considera útil o importante, aunque sigue existiendo cierta influencia externa.
- **Integración**: El individuo actúa porque sabe que la acción le es beneficiosa, y las causas del comportamiento son principalmente internas, aunque el objetivo sigue siendo externo.

La integración se acerca a la motivación intrínseca, pero se diferencia en que el principal motivo sigue siendo la obtención de un beneficio externo, aunque se haya interiorizado.

3. **Motivación intrínseca**: Se produce cuando la acción se realiza por el interés o disfrute que genera la propia actividad, sin considerar sus consecuencias externas. Es menos frecuente, pero más eficaz y sostenible a largo plazo.

En la práctica, el comportamiento humano puede estar influido por múltiples motivadores, tanto extrínsecos como intrínsecos, cuya intensidad varía según el individuo y el contexto. Por ejemplo, dos personas pueden aceptar una oferta de trabajo por motivos distintos: una por el interés en la ubicación y otra por el salario. Además, la motivación puede cambiar con el tiempo y en función de las circunstancias.

Ambos tipos de motivación pueden coexistir y reforzarse mutuamente, aunque en muchos contextos, como el laboral, la motivación extrínseca suele ser predominante. Por ejemplo, alguien que trabaja en lo que le apasiona puede disfrutar de su trabajo (motivación intrínseca) y, al mismo tiempo, valorar el salario que recibe (motivación extrínseca). Sin embargo, no siempre ambos tipos estarán presentes, y en ocasiones puede ser necesario limitar ciertos motivadores extrínsecos si reducen la motivación intrínseca.

Diversos estudios en los ámbitos académico y profesional han demostrado que la motivación intrínseca tiende a generar mejores resultados, mayor satisfacción y un rendimiento más eficiente que la motivación puramente extrínseca. No obstante, fomentar la motivación intrínseca no siempre es sencillo, ya que depende en gran medida de los gustos, intereses y la relación personal del individuo con la tarea. Por el contrario, los motivadores extrínsecos suelen ser más fáciles de implementar, ya que se basan en recompensas o consecuencias externas que, en general, son percibidas como positivas o negativas por la mayoría de las personas.

La gamificación busca precisamente fomentar la aparición de motivadores intrínsecos, utilizando elementos y dinámicas de juego para hacer que las tareas sean más atractivas y satisfactorias. Sin embargo, es más fácil motivar a través de recompensas externas que transformar una tarea poco atractiva en algo intrínsecamente motivador, aunque los beneficios a largo plazo son mayores cuando la motivación es intrínseca. No obstante, existen actividades que, por su naturaleza monótona o poco atractiva, resultan muy difíciles de transformar

en experiencias intrínsecamente motivadoras, por ejemplo, la declaración de la renta o tareas repetitivas en una fábrica. En estos casos, puede ser necesario recurrir a motivadores extrínsecos para incentivar la participación.

A la hora de gamificar un sistema, es recomendable analizar tanto la naturaleza de la actividad como las características de los usuarios, utilizando el sentido común y la lógica para seleccionar las estrategias más adecuadas. No es imprescindible realizar estudios exhaustivos, pero sí es útil contar con información básica sobre los usuarios y el contexto para evitar aplicar medidas que puedan resultar ineficaces o incluso contraproducentes. Es fundamental entender que la motivación no es un fenómeno exclusivo ni estático: una misma actividad puede resultar intrínsecamente motivadora para una persona y, al mismo tiempo, estar acompañada de motivadores extrínsecos.

Por último, la teoría de la autodeterminación (SDT) identifica tres necesidades humanas básicas que suelen estar presentes en las actividades intrínsecamente motivadas:

- **Competencia**: La sensación de ser capaz y eficaz al interactuar con el entorno, como aprender una nueva habilidad o superar un reto.
- **Relación**: El deseo de establecer conexiones sociales y sentirse parte de un grupo o comunidad.
- **Autonomía**: La necesidad de sentir que se tiene control sobre las propias acciones y decisiones.

Estas necesidades pueden estar presentes en mayor o menor medida, y satisfacerlas favorece la aparición de motivación intrínseca. Sin embargo, no siempre se manifiestan todas a la vez, y en ocasiones puede predominar una sobre las demás.

Además de las necesidades básicas identificadas por la SDT, algunos autores proponen organizar los motivadores en diferentes niveles o capas, según la naturaleza de la necesidad que satisfacen. También existe el concepto de **capas de motivación** (*Layers of Motivation*), que organiza los motivadores en función de la naturaleza, no de la necesidad que satisfacen, sino de la necesidad a la que responden:

- **Núcleo (Core)**: Incluye las necesidades básicas del ser humano, como la salud, la seguridad o el alimento.
- **Emocional (Emotional)**: Engloba necesidades emocionales como la autonomía, el propósito o la maestría.
- **Trivial**: Agrupa elementos que, aunque no son fundamentales para el bienestar del individuo, pueden servir como incentivos adicionales, como medallas, logros, bonificaciones o tablas de clasificación.

El orden de estas capas indica la prioridad que se debe dar a la hora de satisfacerlas: primero las necesidades básicas, luego las emocionales y, finalmente, los elementos triviales. En gamificación, aunque los elementos triviales pueden ser útiles para motivar, es más efectivo y sostenible centrarse en satisfacer las necesidades de las capas más profundas.

2.1.3. Compromiso

El **compromiso** (*engagement*) de los usuarios con un sistema gamificado es un factor clave para su éxito, independientemente del contexto (laboral, educativo, comercial, etc.). Un mayor compromiso se traduce en beneficios como una mejora a la productividad y rentabilidad, un incremento a la fidelidad de los clientes y una mayor tasa de éxito[3] dependiendo de la naturaleza el contexto en el que se aplique[4].

Conviene mencionar que motivación y compromiso no son lo mismo, suelen ir de la mano, pero **son conceptos independientes** y es posible estar comprometido sin motivación o viceversa. Aun así, ambos suelen incrementar tras una gamificación bien implementada.

Para medir el nivel de compromiso de un usuario con un sistema, se pueden considerar cinco factores principales:

1. **Recencia**: tiempo desde la última interacción del usuario con el sistema.
2. **Frecuencia**: promedio de interacciones en un periodo concreto.
3. **Duración**: cuánto tiempo duran esas interacciones.
4. **Viralidad**: grado en que el sistema se propaga y es conocido y/o empleado por nuevos usuarios.
5. **Calificaciones**: sistemas de puntuación que reflejan la opinión y compromiso de los usuarios.

No siempre resultan útiles todos los factores, por ejemplo, la viralidad no tiene sentido medirla cuando se ha gamificado una tarea interna de una empresa, los empleados responsables deben utilizar el sistema, no existe una propagación, ya que todos conocen el sistema por ser la herramienta de trabajo.

2.1.4. Flujo

El flujo o flow, según el psicólogo Mihaly Csikszentmihalyi, es el estado mental óptimo para realizar una actividad, donde el individuo está completamente inmerso. Alcanzar el *flow* depende del equilibrio entre dificultad y habilidad: si la tarea es demasiado difícil surge ansiedad, si es demasiado fácil, aburrimiento. Además, es necesario tener en cuenta que la personalidad de cada individuo puede hacer a este más o menos propenso a sentir ansiedad y/o aburrimiento. La siguiente figura muestra la relación entre dificultad y habilidad y cómo afecta al flujo:

[3] Éxito académico.

[4] Una correcta gamificación proporciona beneficios, independientemente del contexto, sin embargo, en función del contexto, la forma que tomen esos beneficios es distinta.

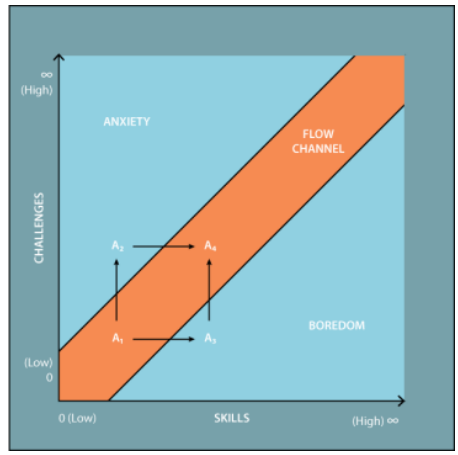

Figura 2.1: Diagrama mostrando la relación entre dificultad y habilidad y como afecta al flujo. Gráfica extraída de https://cinegamification.com/storytelling-gamification/the-flow-theory/.

En la gráfica de la Figura 2.1, el eje Y representa la dificultad y el eje X las habilidades. El área naranja es el espacio de flow; las zonas azules representan ansiedad (dificultad muy superior a la habilidad) y aburrimiento (habilidad muy superior a la dificultad). En la práctica, el usuario fluctúa entre estos estados, y mantener un flow constante es muy difícil, ya que requiere ajustar continuamente la dificultad y conocer las capacidades y personalidad de cada usuario. Por tanto, el objetivo es evitar que los usuarios pasen demasiado tiempo en zonas de ansiedad o aburrimiento, ya que esto reduce motivación y compromiso. Por ejemplo, la ansiedad puede motivar al principio, pero si se mantiene, desanima; el aburrimiento puede ser un respiro, pero si se prolonga, reduce el interés. Los estados A1-A4 de la gráfica representan el flujo ideal y los desvíos habituales. En conclusión, para alcanzar el flow es necesario un equilibrio entre habilidades y desafíos, objetivos claros y feedback inmediato.

2.1.5. Tipos de usuarios

Existen dos taxonomías de usuarios ampliamente reconocidas en gamificación[5]: la de Bartle [12] y la de Marczewski [45].

Taxonomía de Bartle: Basada en el escrito de Richard Bartle (1996), originalmente para juegos multijugador online (MMORPGs y MUDs), pero aplicable también a juegos de un jugador. Divide a los jugadores en cuatro tipos:

- **Triunfador (*Achiever*)**: Prefiere interactuar con el mundo del juego, busca maestría, prestigio, completar logros y desafíos. Le atraen los porcentajes de completitud y la competición y/o cooperación para obtener recompensas. Es el tipo de jugador que más atraído se ve por sistemas de logros y similares.

[5]Existen más clasificaciones y múltiples hibridaciones, pero estas dos son las más reconocidas.

- **Explorador (*Explorer*)**: Prefiere interactuar con el mundo del juego, disfruta explorando e investigando el entorno, prioriza la narrativa y la sensación de descubrimiento. Suele rechazar juegos con poca narrativa o con limitaciones de tiempo.
- **Socializador (*Socializer*)**: Se enfoca en la interacción con otros jugadores y usa el juego como medio para socializar. Prefiere juegos multijugador con amplias posibilidades de interacción, aunque también puede disfrutar juegos de un jugador con narrativa y diálogos extensos.
- **Asesino (*Killer*)**: Busca la acción contra otros jugadores, disfruta la competición y el conflicto directo. Prefiere juegos multijugador competitivos, como *shooters* o *MMORPGs*, y busca la victoria por cualquier método.

Figura 2.2: Modelo de cuadrantes reflejando la taxonomía de Bartle. Imagen extraída de la entrada en español del siguiente artículo de la Wikipedia https://w.wiki/7mp7.

Esta clasificación sigue siendo relativamente útil en la actualidad, porque se basa en lo que motiva y atrae a los jugadores. Sin embargo, no es ideal para la gamificación al tomar, precisamente, lugar fuera del entorno lúdico. El propio Bartle reconoce que su modelo no se traduce bien fuera de los MMOs y menos aún en sistemas no lúdicos.

Taxonomía de Marczewski: Andrzej Marczewski desarrolló una clasificación orientada a sistemas gamificados, basada en motivaciones. Su **Player and User Types Hexad** [45] distingue seis tipos:

- **Cuatro tipos motivados intrínsecamente:**
  - **Socializadores**: Buscan interactuar y establecer relaciones, motivados por la necesidad de relación. Asociados a la diversión con personas.
  - **Espíritus Libres**: Valoran la autonomía y la libertad de acción, disfrutan explorando y creando. Asociados a la diversión fácil (novedad, curiosidad, creatividad).
  - **Triunfadores**: Buscan logros y superación personal, motivados por la maestría y competencia. Asociados a la diversión difícil.

  - **Filántropos**: Buscan propósito y sentido, disfrutan ayudando a otros y contribuyendo al bien común. Asociados a la diversión sería.

- **Jugador (*Player*)**: Motivado por recompensas extrínsecas, actúa como los tipos intrínsecos, pero **solo** si hay recompensa. Subtipos: *Self-Seeker*, *Consumer*, *Networker*, *Exploiter*.
- **Disruptor**: Busca provocar cambios en el sistema, con subtipos beneficiosos como el *Influencer* o el *Improver*, y otros dañinos, como el *Griefer* y el *Destroyer*.

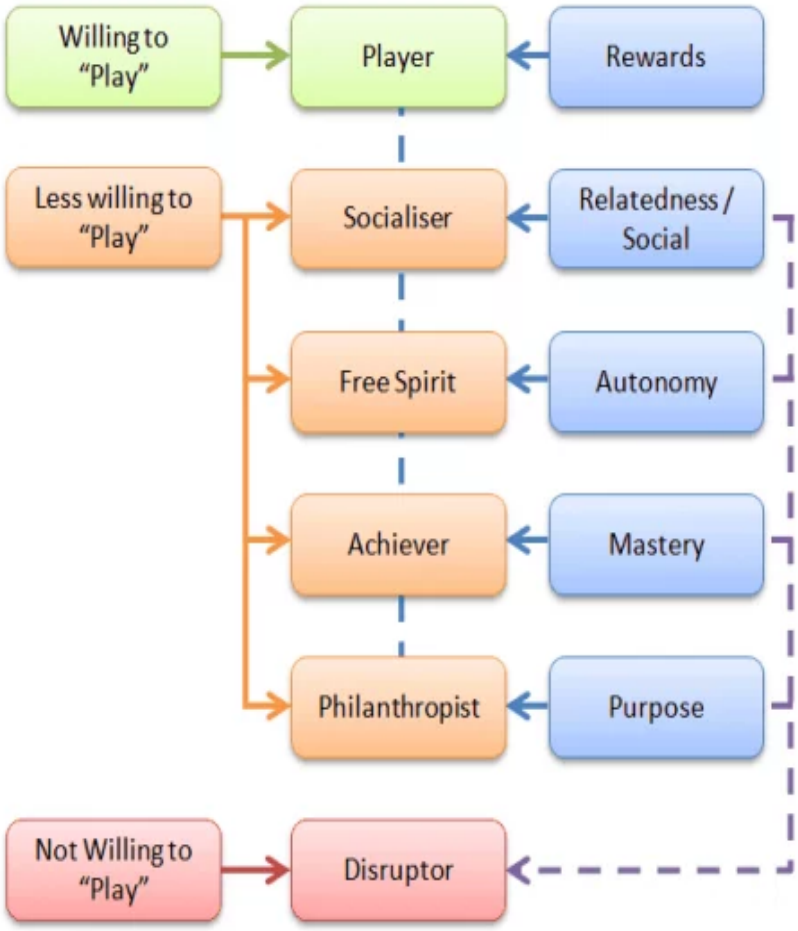

Figura 2.3: Clasificación de usuarios según su disposición a jugar (Marczewski). Imagen extraída del blog de Marczewski:https://www.gamified.uk/user-types/

Marczewski introduce el modelo **RAMP** (Relación, Autonomía, Maestría, Propósito), combinando la SDT y el modelo de Daniel Pink, como motivadores clave para sistemas gamificados.

El modelo de cuadrantes de Marczewski para los tipos intrínsecos es similar al de Bartle, pero adaptado a usuarios y sistemas en vez de jugadores y mundos de juego.

Antes de definir los dos tipos adicionales (Jugador y Disruptor), Marczewski aclara que los ve más como grupos de subtipos, pero es práctico tratarlos como tipos individuales. Los subtipos de Jugador actúan como los tipos intrínsecos pero motivados por recompensas, y los Disruptores pueden ser beneficiosos o dañinos según su orientación.

- **Jugadores (*Players*)**: Motivados por recompensas como dinero, bienes, estatus, etc. Coinciden en cuanto a naturaleza con los tipos intrínsecos de usuario, y actúan igual que ellos, sin embargo, solo cuando hay una recompensa.

  Presenta los subtipos: *Self-Seeker* que busca el beneficio propio, *Consumer* que participa buscando recompensas, *Networker* que busca contactos y oportunidades y *Exploiter* que explora los límites del sistema para obtener recompensas.

  Como se ha mencionado, el mayor beneficio lo genera la motivación intrínseca, y por ello, es recomendable centrarse, a la hora de diseñar el sistema, en hacerlo para los usuarios de tipos intrínsecos, sin embargo, aunque los **Jugadores** son usuarios que inicialmente se unen al sistema buscando recompensas, pueden acabar transformándose en usuarios de tipos intrínsecos, por lo que es también recomendable dedicar, aunque en menor medida, recursos, a la hora de diseñar el sistema, en favorecer esa transformación.

- **Disruptores (*Disruptors*)**: Usuarios que no quieren jugar según las reglas del sistema, pero interactúan buscando provocar cambios. Subtipos:

  - ***Griefer***: Busca arruinar la experiencia de otros usuarios.
  - ***Destroyer***: Busca destruir el sistema, hackeando o explotando vacíos en las reglas.
  - ***Influencer***: Influye en otros usuarios para cambiar el sistema, si se le proporciona asistencia y apoyo, puede traer beneficios.
  - ***Improver***: Busca mejorar el sistema, detectando problemas y proponiendo soluciones.

  Los Disruptores pueden ser muy beneficiosos o extremadamente dañinos, por lo que es fundamental gestionarlos correctamente. Conviene aclarar que este último tipo de usuarios es poco frecuente.

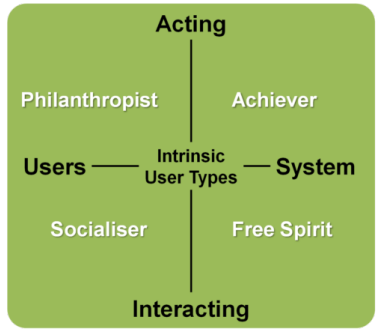

Figura 2.4: Modelo de cuadrantes de los tipos motivados intrínsecamente (Marczewski). Imagen extraída del blog de Marczewski:https://www.gamified.uk/user-types/

2.1.6. Elementos de la gamificación:

Los elementos de los juegos se organizan en una pirámide de tres niveles: **Dinámicas** (cima, mayor abstracción), **Mecánicas** (nivel intermedio) y **Componentes** (base, mayor concreción). Existe un cuarto elemento, la **Experiencia**, que rodea la pirámide y representa la integración emocional y funcional de todos los elementos.

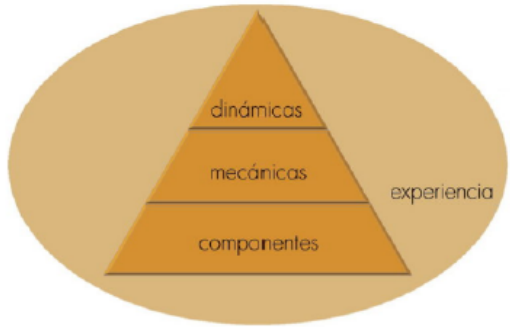

Figura 2.5: Pirámide de elementos de los juegos y su jerarquía. Imagen extraída del documento [68].

Las **dinámicas** son directrices globales o patrones que orientan el sistema gamificado. Incluyen:

- **Restricciones**: Limitaciones que generan desafío e interés.
- **Emociones**: Despertar emociones como curiosidad, competitividad, alegría, relajación.
- **Narrativa**: Historia coherente que guía el progreso del usuario y se integra con las tareas.
- **Progresión**: Sensación de avance y desarrollo personal.

- **Relaciones entre usuarios**: Cooperación y/o competición, generando vínculos sociales y emociones asociadas.
- **Estado/Estatus**: Búsqueda de reconocimiento, prestigio y fama.

Las dinámicas son difíciles de implementar directamente, ya que requieren procesos complejos y afectan a todo el sistema. Son las pautas que guían la evolución del sistema y los usuarios.

Las **mecánicas** son los procesos básicos que impulsan la acción y generan compromiso (*engagement*). Incluyen:

1. **Desafíos/Retos**: Tareas que sacan al usuario de su zona de confort sin llegar a la frustración.
2. **Oportunidad/Azar**: Elementos aleatorios, como recompensas variables.
3. **Cooperación**: Trabajo en equipo para alcanzar metas compartidas.
4. **Competición**: Competir por ser el mejor, con cuidado de no generar efectos negativos.
5. **Realimentación (Feedback)**: Información clara, concisa y rápida sobre el progreso y los errores.
6. **Recompensas**: Premios por acciones o comportamientos, con diferentes esquemas de entrega.
7. **Estados de victoria**: Objetivos o condiciones que determinan el éxito, empate o derrota.

Otras mecánicas importantes son la recopilación de recursos, las transacciones e intercambios, los turnos...

Los **componentes** son elementos concretos y específicos. Son instancias de las dinámicas y mecánicas. Los tipos principales de componentes son:

1. **Logros**: Objetivos definidos, visibles u ocultos, que fomentan el progreso.
2. **Avatares**: Representaciones visuales del usuario.
3. **Insignias**: Medallas que reflejan logros y progreso.
4. **Batallas de jefe**: Desafíos especialmente difíciles, habitualmente al final de una etapa.
5. **Coleccionables**: Objetos acumulables que satisfacen el deseo de completitud, previamente se mencionó que los logros resultan especialmente efectivos con los usuarios de tipo *Achiever*, los coleccionables son también extremadamente efectivos para ese tipo de usuarios.
6. **Combates**: Desafíos definidos, normalmente de corta duración.

7. **Desbloqueo de contenido**: Acceso a nuevas partes del sistema tras alcanzar objetivos determinados.
8. **Regalos**: Posibilidad de compartir recursos con otros.
9. **Tablas de clasificación**: Ranking público de usuarios según su desempeño.
10. **Niveles**: Etapas en que se organiza el progreso.
11. **Puntos**: Representación numérica del progreso.
12. **Misiones**: Conjunto de desafíos con objetivos y recompensas.
13. **Gráficos sociales**: Representaciones de las relaciones sociales.
14. **Equipos**: Grupos de usuarios con objetivos comunes.
15. **Bienes virtuales**: Activos del sistema con valor percibido o monetario.

No es necesario usar todos los componentes; cada sistema debe seleccionar los más adecuados.

Antes de acabar con los elementos del juego, múltiples fuentes mencionan un **cuarto tipo de elemento** de los juegos que es la **Experiencia**. Esta queda muy bien explicada en un artículo escrito por los profesores de la Universidad de Valladolid Alma María Pisabarro Marrón y Carlos Enrique Vivaracho PascualPisabarro Marrón y Vivaracho [68]. En dicho artículo, además de hablar de la pirámide en que se estructuran dinámicas, mecánicas y componentes, con los componentes en la base y las dinámicas en la cima, describen la **experiencia** como un cuarto elemento. Este elemento está **fuera** de la pirámide, **rodeando** a la misma. La consideran como "**el elemento que aglutina el juego y lo hace sentir real**" y "**las respuestas emocionales durante el desarrollo del juego**", remarcando que **la suma de estos cuatro elementos no hace un juego, sino que estos deben estar integrados para que el juego funcione**, en otras palabras, no es tan simple como meter elementos de los listados en estos últimos apartados y decir que el sistema está gamificado, hay que estudiar la viabilidad de cada uno de ellos para la situación en la que te encuentras, o lo que es lo mismo, para tu sistema, y, una vez seleccionados, juntarlos e integrarlos correctamente.

Finalmente, quedan tres puntos sobre los que se debe hablar, estos son tres elementos que innegablemente forman parte fundamental del juego, pero que no acaban de poderse asignar a ninguna de las tres categorías previas (dinámicas, mecánicas y componentes). Estos elementos son:

1. **Voluntariedad**: El juego debe ser voluntario para ser efectivo. En gamificación, donde la participación puede ser obligatoria, es esencial fomentar la motivación y el disfrute.
2. **Resolución de problemas**: Aprender y resolver problemas es intrínsecamente entretenido y debe formar parte de la experiencia gamificada.
3. **Equilibrio entre estructura y libertad**: Es fundamental encontrar un equilibrio entre reglas y libertad del usuario. Un exceso de reglas limita la experiencia, pero la ausencia de estructura puede desorientar.

2.1.7. Tipos de gamificación según Werbach y Hunter

Tras una descripción detallada sobre la gamificación y sus componentes, en esta sección se presentan brevemente los tipos de gamificación establecidos por Werbach y Hunter:

1. **Gamificación interna**: Aplicada dentro de una organización para mejorar la motivación y el compromiso de una comunidad ya definida (empleados, estudiantes, etc.), integrándose con las estructuras y objetivos existentes.

   Los usuarios ya pertenecen a una comunidad y comparten cultura, intereses y entorno y la gamificación debe alinearse con las estructuras de gestión y recompensas ya existentes.

2. **Gamificación externa**: Orientada al marketing y la relación con clientes, busca mejorar el compromiso, la lealtad y la identificación de los clientes con la marca o producto. Sus efectos se producen fuera del sistema, en su entorno.

3. **Gamificación de cambio de comportamiento**: Busca fomentar hábitos beneficiosos a nivel social (vida saludable, ahorro, etc.), promovida por organizaciones sin ánimo de lucro o gobiernos, aunque también puede tener fines comerciales. Un ejemplo de ello es **Keas**, una startup que promueve la salud y el bienestar de empleados mediante gamificación.

Estos tipos no son los únicos, pero son los más reconocidos en el ámbito de los negocios[6].

2.1.8. Gamificación, ventajas e inconvenientes

Ahora que se ha definido qué es la gamificación y sus tipos, se presentan las ventajas e inconvenientes que conlleva su uso en el ámbito educativo, comenzando por las **ventajas**:

1. Motivación: El principal beneficio de la gamificación es su potencial motivador. Uno de los factores más importantes a la hora de afrontar tareas complejas, tediosas o difíciles es la motivación, y los juegos resultan ser una de las herramientas de motivación más potentes que existen[7]. Es de estos juegos de donde la gamificación toma distintos elementos para motivar con esa misma efectividad a los individuos. La gamificación aporta muchos beneficios (de los cuales varios derivan de esta capacidad de motivar), pero el principal, sin lugar a dudas, es la motivación.

2. *Engagement* o compromiso: Otro de los principales puntos a favor de la gamificación es su capacidad de generar compromiso en los usuarios con el sistema gamificado. Una correcta gamificación permite que aquello que se ha gamificado pueda satisfacer

[6] Que es el ámbito para el que se escribió el libro de Werbach y Hunter.

[7] Extraído de la página 25 del libro de Werbach y Hunter [74]: *motivation is at the heart of sustained behavior change, and games are among the most powerful motivational tools*, que en español quiere decir que "la motivación se encuentra en el núcleo del cambio de comportamiento sostenido y los juegos se encuentran entre las herramientas motivadoras más poderosas".

las condiciones que hacen que los juegos resulten entretenidos e interesantes para los seres humanos, aunque no necesariamente al mismo nivel o intensidad[8]. Esto resulta de gran utilidad en la educación, incrementando el compromiso de los estudiantes con el aprendizaje y aumentando su implicación con el contenido impartido y las tareas mediante las que se imparte. Un ejemplo de las formas en que esto se lleva a cabo es la narrativa; una narrativa interesante y bien planteada hace que los usuarios se impliquen con la trama, sintiéndose participantes de ella y generando interés por su desarrollo.

3. Como consecuencia de los dos puntos anteriores se produce un incremento en la eficacia de la enseñanza, ya que el aprendizaje resulta más significativo y atractivo para los estudiantes, permitiendo una mejor retención en la memoria a lo largo del tiempo e incrementando también el rendimiento académico, al estar los estudiantes no solo motivados a aprender, sino que también muestran interés hacia lo que se les va a enseñar.

4. La gamificación genera un espacio o entorno similar al de un juego, pero sin estar tan aislado del mundo real, de manera que el usuario es consciente de que no está jugando[9]. Este espacio promueve la experimentación y la innovación, fomentando no solo la búsqueda de soluciones a los problemas, sino la búsqueda de mejores soluciones. Todos estos factores resultan muy positivos para la enseñanza y benefician en gran medida a los estudiantes.

5. Las mecánicas de cooperación y competición, utilizadas adecuadamente, despiertan y fomentan sentimientos altruistas y de camaradería, contribuyendo a que los usuarios se consideren o sientan parte de una comunidad, y motivan a los individuos a participar en el sistema, ya sea por apoyar a sus compañeros o por superar a sus oponentes.

6. Permite en la mayoría de los casos una retroalimentación de información constante (incluso en tiempo real), tanto para profesores como para estudiantes.

7. Dependiendo de cómo se gamifique, se puede posibilitar que cada estudiante avance a su ritmo; con niveles de dificultad incrementales y retroalimentación adecuada se permite a los estudiantes progresar a su ritmo, obteniendo mejores resultados al final.

8. El uso de componentes como los niveles, puntos, *leaderboards* y logros facilita la medición de los resultados, tanto para los profesores como para los alumnos, haciendo a estos últimos más conscientes de su propio progreso.

9. Otro beneficio que tienen los niveles y logros es que proporcionan, en cierta forma, a los estudiantes una lista de objetivos a alcanzar, sirviendo como una especie de guía que pueden usar para no moverse a ciegas por el sistema.

10. Aunque, como se ha comentado varias veces, la gamificación no crea un juego y no tiene como objetivo convertir el sistema en algo con lo que el usuario pueda jugar, sí consigue que resulte más entretenido e interesante para el usuario, al menos más que si no se hubiera gamificado.

[8] Esto no es algo negativo, no se busca crear un juego, sino aprovechar factores asociados a ellos. Se gamifica precisamente para ayudar y facilitar el alcance de los objetivos del sistema sin que la importancia de estos se vea reducida por la búsqueda de diversión.

[9] Esto no significa que no pueda divertirse o disfrutar de la tarea, pero sí que debe ser consciente de que no se trata de un juego y que debe tomarse en serio.

11. Al presentarse frecuentemente desafíos o retos en forma de resolución de problemas, junto con la posibilidad de progresar a su propio ritmo, los estudiantes se vuelven más autónomos.

12. Resultados: Una correcta gamificación es un proceso muy complejo que depende de múltiples variables; no obstante, no se puede negar que cuando se implementa adecuadamente resulta beneficiosa. Actualmente, no es raro que en la enseñanza se organicen actividades ambientadas mediante narrativa, el uso de *leaderboards* y sistemas de puntos para organizar pequeñas competiciones entre estudiantes, e incluso la implementación de logros (por ejemplo, en la asignatura de Computación Paralela del grado de Ingeniería Informática de la Universidad de Valladolid se utilizan en las prácticas *leaderboards*, puntos y logros para motivar a los estudiantes), estando presentes también en niveles menos avanzados de educación (ya sea mediante gráficas y animaciones que acompañan a la lección, o cuestionarios con componentes del tipo *une el elemento de la columna A con su correspondiente pareja de la columna B* o cualquiera de las posibles formas que puede tomar la gamificación).

A continuación se presentan los **inconvenientes**:

1. La gamificación tiende a cuantificar, reducir y clasificar, lo cual en el ámbito educativo puede resultar perjudicial, ya que pueden perderse matices del contenido a enseñar o incluso reducirse la visión general del tema a puntos muy concretos.

2. Ciertos elementos de la gamificación, como los puntos y las *leaderboards*, pueden provocar que individuos particularmente competitivos ignoren los objetivos del sistema gamificado, centrándose únicamente en ganar. Por ello, es necesario gestionar adecuadamente las emociones que se pretende despertar en los usuarios.

3. En línea con el punto anterior, esta competitividad extrema o el deseo de adquirir estatus pueden llevar a los usuarios a hacer trampas, lo cual resulta totalmente inaceptable en la educación y la enseñanza.

4. Un uso excesivo de mecánicas de competición y sus correspondientes recompensas puede provocar que los individuos no ganadores acaben desmotivados e incluso se consideren aislados del resto de la comunidad. Esto también puede ocurrir con el uso incorrecto de *leaderboards* y puntos.

5. La gamificación es un proceso complejo y difícil de dominar, al igual que lo son el diseño y la creación de juegos. Otro de los aspectos que hereda del proceso de diseño y creación de juegos es que incluso aquellos individuos que se consideran expertos pueden fallar sin que exista una explicación clara al respecto.

6. La gamificación no es aplicable en cualquier contexto y es posible que, en contextos en los que sí se pueda aplicar, no sea recomendable por no compensar los posibles beneficios los costes del proceso[10].

[10]No necesariamente costes monetarios, puede tratarse de tiempo para llevarlo a cabo, mano de obra, desventajas generadas, etc.

7. La sobre-gamificación puede provocar que no se le otorgue la importancia merecida a la actividad gamificada. Por ejemplo, una actividad gamificada organizada para extender y afianzar los conocimientos de los alumnos sobre un cierto tema, al estar sobregamificada, puede ser vista por los estudiantes como un juego y no como una lección, resultando en que estos no le den importancia a lo que se les intenta enseñar, concentrándose únicamente en divertirse y haciendo que la actividad resulte una pérdida de tiempo. No debe confundirse con el primer inconveniente mencionado, ya que en el primero implica una simplificación o incorrecta presentación del contenido a enseñar, mientras que aquí se hace referencia a la posibilidad de que los usuarios (estudiantes en este caso) no otorguen la importancia debida a un contenido que no tiene por qué estar mal explicado.

8. La gestión indebida de recompensas y logros puede provocar la llamada *hedonic treadmill* o rueda hedónica, fenómeno ampliamente estudiado en el campo de la psicología, que en lo relativo a la gamificación se refiere a la tendencia de los usuarios a dar por sentado el recibir recompensas por cada logro que alcancen o por cada tarea que completen, acostumbrándose a ser recompensados constantemente y esperando cada vez recompensas mayores. Llegados a este punto, las recompensas resultan cada vez menos motivadoras y satisfactorias para los usuarios, pudiendo llegar al extremo de que algunos no realicen las tareas si no van a recibir una recompensa a cambio. Esto resulta muy perjudicial, ya que se trata de sistemas gamificados, no de juegos cuya única función es la diversión; los deberes y formaciones o lecciones (en la educación) o las tareas del trabajo deben llevarse a cabo.

9. Otro inconveniente común en la gamificación surge por dar demasiada importancia al estatus como recompensa en el sistema. Aunque puede resultar un elemento muy motivador, como se ha mencionado en el punto 4, aquellos individuos que no están ganando o que no ascienden en la *leaderboard* pueden sentirse desmotivados. Aspectos como saber o pensar que no se va a llegar a lo más alto pueden resultar desmotivadores, además de que no todos los usuarios son receptivos a este tipo de motivación y no es raro que haya individuos que no busquen adquirir ese reconocimiento o estatus.
   Estas situaciones son especialmente comunes en aquellos sistemas en los que se gamifica centrándose en usar motivadores extrínsecos, en vez de emplear estos como refuerzo para generar motivación intrínseca en los usuarios.

10. Existe un intenso debate social en torno al uso indebido de la gamificación, que incluso llega a cuestionar si, en nuestra sociedad, el ser humano está obligado a trabajar para subsistir. No obstante, el alcance de este trabajo se limita al análisis de la gamificación en entornos locales sin elevarlo al nivel de toda una cultura educativa o corporativa.

    El origen del problema reside en que la gamificación, en su esencia, consiste en emplear ciertos estímulos (los elementos de juego) para provocar en los usuarios reacciones específicas (emociones y motivaciones) que les impulsen a comportarse de la manera deseada: mayor implicación en su aprendizaje, incremento del interés por los contenidos, y similares.

    Desde un punto de vista neutral, no sería objetivamente problemático si esos estímulos generan reacciones "neutras" o positivas: recompensas, sistemas más entretenidos, fomento de motivadores intrínsecos como el deseo de mejorar, etc. Por ejemplo, un estudiante universitario podría no sentir verdadera pasión por su carrera, sino simplemente

aspirar a aumentar sus oportunidades laborales. No está forzado a elegir esa vía: podría optar por otra titulación, por formación profesional o incluso incorporarse directamente al mercado de trabajo. Sin embargo, una gamificación bien diseñada haría que todo el proceso resultase más ameno, y aun cuando, tras finalizar sus estudios, esa carrera no se convierta en su vocación, el estudiante habría disfrutado de numerosas ventajas sin haber sido obligado a actuar en contra de sus deseos o valores.

El verdadero problema aparece cuando la gamificación recurre a emociones negativas como el miedo, rivalidad exacerbada, una competitividad excesiva y similares, para conseguir sus fines. Algunos ejemplos son las penalizaciones o castigos (por ejemplo, amenazas de reducción salarial injusta) o la implementación de sistemas de puntos y *leaderboards* tan competitivos que deterioran las relaciones entre los participantes y generan estrés o temor a quedar en puestos bajos del ranking. Situaciones como estas alimentan el debate sobre la explotación y la manipulación mediante la gamificación. Un caso especialmente relevante tuvo lugar en los lavaderos de varios hoteles de Disneyland en Anaheim (California), donde los empleados denominaron *electronic whip* o látigo electrónico [36] al sistema de puntos, clasificaciones y pantallas que mostraban en tiempo real el rendimiento propio y el de los compañeros. Esta constante comparación creó un clima de tensión tan fuerte que algunos trabajadores renunciaban incluso a sus descansos para ir al baño, temiendo por la seguridad de su empleo. Con independencia de las intenciones de quienes diseñaron aquel sistema, este ejemplo ilustra de forma muy clara por qué la gamificación, especialmente la basada en emociones negativas, genera dudas y desconfianza, ya que en ella se observan con bastante claridad los peligros de la explotación y la manipulación.

Antes de pasar al siguiente apartado, conviene hacer dos comentarios sobre este último inconveniente de la explotación y manipulación.

El primero es que, aunque ciertamente el caso del *electronic whip* resulta demasiado limitado como para considerarse una gamificación propiamente dicha (consiste únicamente en una *leaderboard* y los rankings y métricas/puntuaciones intrínsecas a dichas *leaderboards*), no resulta complicado pensar en formas de expandir dicho sistema, premiando a los puestos más altos del ranking y castigando a los más bajos, por ejemplo.

El segundo comentario es respecto aquellas opiniones que presentan la gamificación como una herramienta para explotar a los usuarios. Esas ideas en general son excesivas y exageradas, ya que, a pesar de que la gamificación ciertamente puede ser utilizada de forma que favorezca situaciones moralmente incorrectas, lo mismo puede decirse de cualquier otra disciplina, como la química, que puede emplearse tanto para crear medicamentos y productos beneficiosos como para fabricar venenos y contaminantes.

En resumen, la gamificación es una herramienta que puede generar resultados positivos, pero, al igual que cualquier otra herramienta, su uso indebido puede causar daños.

2.1.9. Gamificación a lo largo de la historia

A continuación se presentan algunos ejemplos representativos de la aplicación de la gamificación a lo largo de la historia, con el objetivo de ilustrar la evolución y el impacto de esta estrategia en diferentes contextos, tanto antes como después de la formalización del término.

El término *gamificación* es un anglicismo derivado de *gamification*, acuñado en 2002 por Nick Pelling [67]. Sin embargo, la utilización de elementos de juego en contextos no lúdicos es anterior a la aparición del término. Un ejemplo temprano se encuentra en 1896, cuando la compañía Sperry & Hutchinson introdujo los *Green Stamps*, un sistema de sellos de recompensa que los minoristas entregaban a sus clientes, quienes podían canjearlos posteriormente por productos [17]. Este sistema incentivaba la fidelidad y la participación de los consumidores mediante mecanismos propios de los juegos, como la acumulación de puntos y la obtención de recompensas.

A principios del siglo XX, el movimiento Scout, fundado en 1908 por Robert Baden-Powell, constituye uno de los primeros ejemplos de gamificación aplicada a la educación y el desarrollo personal [16]. En este movimiento, los jóvenes reciben insignias y medallas a medida que adquieren habilidades y participan en actividades, lo que fomenta la motivación, el aprendizaje y la superación personal a través de la consecución de logros y el reconocimiento social. El sistema de insignias y progresión por niveles dentro de los Scouts permite a los participantes visualizar su avance, establecer metas y recibir recompensas tangibles por su esfuerzo, elementos que posteriormente serían reconocidos como fundamentales en la gamificación moderna.

En la actualidad, la gamificación se ha extendido a numerosos ámbitos, tanto laborales como educativos y personales. Un ejemplo destacado en el entorno empresarial es el Language Quality Game de Microsoft, diseñado para mejorar la calidad de las traducciones mediante la participación voluntaria de empleados en tareas lúdicas [17]. Este juego interno permitía a los empleados identificar y corregir errores en traducciones de software, compitiendo y colaborando para alcanzar mejores resultados. La iniciativa no solo mejoró la calidad lingüística de los productos, sino que también incrementó la implicación y satisfacción de los trabajadores al transformar una tarea rutinaria en una experiencia motivadora y competitiva.

Por otro lado, Duolingo representa un caso paradigmático de gamificación orientada al aprendizaje de idiomas. Esta plataforma utiliza puntos, niveles, recompensas, retos diarios y sistemas de progresión visual para incentivar la práctica constante y el progreso de los usuarios [17]. A través de mecánicas como las rachas de días consecutivos, los logros y la competencia amistosa entre usuarios, Duolingo logra mantener altos niveles de compromiso y motivación en el aprendizaje autónomo. La experiencia de usuario se ve enriquecida por la retroalimentación inmediata y la posibilidad de comparar el propio rendimiento con el de otros, elementos que han demostrado ser eficaces para fomentar la constancia y el interés en el aprendizaje.

Estos ejemplos ilustran cómo la gamificación, tanto antes como después de la formalización del término, ha demostrado ser una estrategia eficaz para potenciar la implicación, la motivación y el aprendizaje en contextos muy diversos. La evolución y expansión de la gamificación a lo largo del tiempo evidencian su relevancia y adaptabilidad como herramienta

para el cambio y la mejora en múltiples facetas de la experiencia humana.

Capítulo 3

Planificación

En este capítulo se explica en detalle la planificación del proyecto. Primero se abordará el enfoque adoptado para el desarrollo del proyecto. Después se muestra la *Matriz de Riesgos* del proyecto seguida por su *Plan de contingencia*, a continuación se expone la tabla con la estimación de tiempo invertido en el proyecto y, por último, una estimación del presupuesto potencial que podría incurrir el desarrollo del proyecto si se hubiese llevado a cabo en el entorno laboral y no en el académico.

3.1. Metodología del proyecto

El **P**roceso **U**nificado de **D**esarrollo de **S**oftware (USDP) o **Proceso Unificado** [22] (***U**nified **P**rocess* en inglés) es un marco iterativo e incremental que busca enfocar y organizar el desarrollo de software mediante casos de uso, arquitectura clara y gestión de riesgos. En cada una de las iteraciones están involucrados flujos de trabajo[1] de especificación de requisitos, análisis, diseño, implementación y pruebas. De ahí viene el nombre que le ha sido asignado a los distintos capítulos y secciones de este documento.

Establece seis *buenas prácticas*[2] que ayudan a que, en general, mejore la calidad del producto software y el proceso **se adapte** a las necesidades del proyecto.

Cabe mencionar que los requisitos funcionales del proyecto no se cerraron hasta una etapa más avanzada del desarrollo; se establecieron y se trabajó en base a ellos, pero no se denegó la posibilidad de incorporar nuevos requisitos o modificar los ya establecidos hasta etapas posteriores. Por tanto, el proyecto también adopta la metodología del **Modelo de Prototipos**.

[1] El término utilizado para estos flujos de trabajo en *Unified Process* es *disciplines*, que significa disciplinas en español.

[2] Estas son: desarrollo iterativo, gestión de requisitos, una arquitectura basada en componentes, el uso de UML, control de cambios y la revisión constante de la calidad del proyecto.

3.2. Análisis de Riesgos

La Matriz de Riesgos establece el denominado *Nivel de Calor* de cada suceso, en función de "**cómo de probable es que ocurra**" y "**si ocurre, cómo de severas son las consecuencias**" y en base a estos dos **factores** el correspondiente *Nivel de Calor* es asignado. La nomenclatura usada para estos niveles de calor representa cinco grados del mismo, en orden incremental de probabilidad de que ocurra el suceso y daño o impacto en el proyecto si ocurre: *Insignificante*, *Menor*, *Moderado*, *Importante* y *Catastrófico*.

Se entiende por **riesgo** a la posibilidad de que ocurra un evento que, al materializarse, pueda afectar negativamente a los objetivos del proyecto debido a sus consecuencias e impacto. Se denomina **suceso** a la materialización del riesgo, el riesgo es un evento que **puede suceder**, mientras que el suceso es un riesgo que **ha sucedido**.

Antes de presentar los componentes en los que se reflejan los detalles de esta sección, conviene mencionar que uno de los sucesos tiene un nivel de calor que no sigue los niveles de calor establecidos por la matriz de la Figura 3.1, este suceso es "***No es posible alcanzar la fecha de entrega del proyecto por motivos de fuerza mayor***" y el motivo por el que se le ha asignado Riesgo Catastrófico a pesar de que su par Probabilidad/Severidad es Baja-Alta, es porque su probabilidad es **muy baja** y su severidad es **muy alta**, pero al ser el único suceso tan extremo se ha preferido mencionarlo aquí y en el propio plan de contingencia frente a añadir un caso aparte a la matriz de riesgo.

		SEVERIDAD		
		BAJA	**MEDIA**	**ALTA**
PROBABILIDAD	**BAJA**	*Riesgo Insignificante (MB - Muy Bajo)*	*Riesgo Menor (B - Bajo)*	*Riesgo Moderado (M - Medio)*
	MEDIA	*Riesgo Menor (B - Bajo)*	*Riesgo Moderado (M - Medio)*	*Riesgo Importante (A - Alto)*
	ALTA	*Riesgo Moderado (M - Medio)*	*Riesgo Importante (A - Alto)*	*Riesgo Catastrófico (MA - Muy Alto)*

Figura 3.1: Figura que representa la matriz de riesgo del proyecto.

				Detalles	
Riesgo	*Nivel de Calor*	*Detalles*	*Plan de Contingencia (+ Mitigación)*	*Probabilidad*	*Severidad*
Baja/Enfermedad del Tutor	Riesgo Insignificante (MB)	Asumiendo que no sea una baja de larga extensión ni se deba a causas muy graves.	Cambiar las fechas de reuniones y revisiones para adaptarse al contratiempo. Reunirse con frecuencia para aumentar la posibilidad de que, en caso de que se de el evento, los "objetivos" a solucionar con la reunión sean los menos posibles.	Baja	Baja
Pérdida de datos "persistentes" como los de la base de datos de la aplicación	Riesgo Insignificante (MB)	Por la naturaleza del proyecto, durante el desarrollo de la aplicación, los datos "persistentes" almacenados en la/s base/s de datos de la aplicación son fácilmente replicables y no cruciales.	Mantener copias separadas de dichos datos o de imágenes de estos. Asegurarse de tomar las medidas adecuadas para evitar la corrupción de datos. En caso de que ocurra el suceso, determinar la envergadura de la perdida de datos y reemplazar los afectados.	Baja	Baja

Tabla 3.1: Riesgos de nivel Insignificante del Plan de Contingencia del proyecto.

				Detalles	
Riesgo	*Nivel de Calor*	*Detalles*	*Plan de Contingencia (+ Mitigación)*	*Probabilidad*	*Severidad*
Problemas o imposibilidad de acceso/uso a los recursos de trabajo	Riesgo Menor (B)	Dependiendo de la severidad y duración del evento, causas de fuerza mayor que extiendan a mas de "un par de horas" incrementarian el nivel de calor.	Tratar de dar con medios/vias alternativas de acceso a los materiales de trabajo. Mantener todos los aspectos del proyecto actualizados. Mantener el equipo de trabajo en un buen estado de mantenimiento. Determinar la causa del evento y, si es posible, aplicar soluciones temporales. Documentar el suceso.	Baja	Media
Problemas de compatibilidad de tecnologías implicadas	Riesgo Menor (B)		Examinar las opciones disponibles y tratar de escoger las "más compatibles" entre si. Asegurarse de que sea posible la compatibilidad mediante pruebas de integración. Si el evento es solucionable, tomar las medidas necesarias. Si el problema no se puede arreglar, buscar una alternativa comparable a la que no se pueda utilizar e incorporarla.	Media	Baja

Tabla 3.2: Riesgos de nivel Menor del Plan de Contingencia del proyecto.

				Detalles	
Riesgo	*Nivel de Calor*	*Detalles*	*Plan de Contingencia (+ Mitigación)*	*Probabilidad*	*Severidad*
Perdida/Corrupción del "progreso" de la aplicación del proyecto, perdida de versión más reciente, etc.	Riesgo Moderado (M)		Incorporar lo más rápido posible un sistema de control de versiones (Git). Seguir las "Buenas Prácticas" de control de versiones y guardado de datos. Disponer de copias de versión en bases de datos "de respaldo". Determinar la gravedad de la pérdida de datos, que información se ha visto afectada. Tratar de recuperar los datos afectados.	Baja	Alta
Conocimientos de las tecnologías y herramientas involucradas insuficientes / Falta de experiencia	Riesgo Moderado (M)		Planificación rigurosa del desarrollo del proyecto. Ser lo mas preciso y realista en las estimaciones de tiempo y esfuerzo necesarios para cada tarea. Hacer un estudio en profundidad sobre el dominio del proyecto para entender los puntos principales de este y las "tecnologías" involucradas. Acceder a cursos/tutoriales sobre estos elementos para adquirir dichos conocimientos y ponerlos lo antes posible en práctica. Si no se pudiese alcanzar el nivel de conocimiento necesario, buscar soluciones mediante tecnologías/herramientas alternativas.	Alta	Media
Retraso en cualquiera de las fases por factores externos	Riesgo Moderado (M)	Inevitable, se refiere a factores como situaciones de naturaleza familiar, contratiempos del día a día.	La única medida de mitigación para este tipo de sucesos es tratar de aplicar las medidas de mitigación del resto de sucesos de la mejor manera posible para que si ocurre el evento, no sea tan potencialmente perjudicial. Cambiar las fechas de reuniones y plazos en función de la extensión y gravedad del factor externo.	Media	Media

Tabla 3.3: Riesgos de nivel Moderado del Plan de Contingencia del proyecto.

				Detalles	
Riesgo	*Nivel de Calor*	*Detalles*	*Plan de Contingencia (+ Mitigación)*	*Probabilidad*	*Severidad*
Retraso/Incumplimiento de plazos.	Riesgo Importante (A)	Muy probable al ser este el primer proyecto de esta escala realizado por el Estudiante. Dependiendo de la "etapa" del desarrollo del proyecto la severidad del suceso puede variar.	Planificación rigurosa del desarrollo del proyecto. Ser lo mas preciso y realista en las estimaciones de tiempo y esfuerzo necesarios para cada tarea. Comprobar periódicamente el estado de progreso del proyecto frente al planeado. Planificar dando márgenes de tiempo para disponer de tiempo extra en caso de que ocurra algún contratiempo. Si ocurre el evento, adaptar la planificación y alcance del proyecto, según sea necesario, para compensar y recuperar "el tiempo perdido".	Alta	Media
Baja/Incapacidad del Alumno	Riesgo Importante (A)	Depende de la severidad de la enfermedad/incapacitación. Depende de la fecha en la que ocurra.	Mantener hábitos saludables. Evitar riesgos innecesarios. Plantear pausar temporalmente hábitos que pudiesen resultar en lesiones como deporte y similar durante el transcurso del proyecto. Reasignar plazos y reuniones una vez el Alumno se haya recuperado. Si ocurriese durante las últimas etapas del proyecto, plantear reducir el alcance del proyecto de la forma menos "perjudicial".	Media	Alta
Cambios en los requisitos del proyecto	Riesgo Importante (A)	Depende de la fase/etapa del desarrollo del proyecto en la que ocurra.	Registrar cada potencial cambio y determinar su impacto en el proyecto. Realizar un "análisis de requisitos" adecuado para reducir la posibilidad de que sea necesario añadir, eliminar y/o cambiar requisitos en fases posteriores del desarrollo del proyecto. Procurar establecer "márgenes" de tiempo para situaciones "no planeadas" como estas. Dar prioridad a los cambios más "cruciales" que a los de menor importancia.	Alta	Media
Mala planificación	Riesgo Importante (A)	Dependiendo de en que fase/etapa del proyecto la planificación haya sido "mala" la severidad puede variar entre Baja y Media.	Comunicarse frecuentemente con el Tutor y consultar con el decisiones importantes. Tratar de hacer las estimaciones de tiempo lo mas realistas posibles. Realizar un análisis exhaustivo de los requisitos del proyecto para reducir la posibilidad de que la mala planificación sea debida a un análisis insuficiente. Determinar la severidad del suceso y rediseñar/ajustar el plan en base a ello.	Alta	Media
No es posible alcanzar la fecha de entrega del proyecto por motivos de fuerza mayor	Riesgo Catastrófico* (MA)	Caso especial que se desvía ligeramente de la Matriz de Riesgo. Se trata de un evento poco probable pero tiene la "máxima" severidad posible.	Aplicar todas las medidas de mitigación para el resto de eventos para tratar de evitar que esto ocurra. Si ocurre en primera convocatoria, optar a defender en la segunda convocatoria. Revisar opciones con el Tutor.	Baja	Alta*

Tabla 3.4: Riesgos de nivel Alto y Crítico del Plan de Contingencia del proyecto.

3.3. Planificación

Antes de continuar, es importante mencionar que las horas reflejadas en la Tabla 3.5 son una aproximación hecha estableciendo como máximo total de horas para el proyecto **300 horas**, ya que el peso del TFG es de 12 créditos lo que corresponde a, aproximadamente, 25 horas por crédito.

Una vez calculado el total de horas a dedicar al proyecto, se le asignó a cada una de las distintas actividades en las que el proyecto fue dividido la cantidad de horas que se consideró adecuada.

Tiempo invertido en el desarrollo del proyecto	
Actividad/Tarea	***Horas***
Aprendizaje de concepto y uso de herramientas y tecnologias involucradas.	**40**
Busqueda y especificación de Requisitos.	**10**
Análisis del proyecto.	**25**
Diseño del Proyecto.	**35**
Implementación del Proyecto. Escritura e implementación del código.	**115**
Experimentación/Pruebas.	**25**
Finalizar documentación/memoria del Proyecto.	**30**
Otros	**20**
Total	**300**

Tabla 3.5: Tabla mostrando la estimación de las horas dedicadas a las distintas fases del desarrollo del proyecto.

La entrada ***Otros*** representa al conjunto variado de actividades que forman parte del proceso de desarrollo del proyecto, pero que son de menor importancia, aspectos como actualizaciones del equipo, mantenimiento, control de versiones, hacer copias de seguridad y similar.

3.4. Presupuesto

La *estimación del presupuesto* se hará en base al tiempo dedicado a cada paso en la estimación de tiempo y en base al salario promedio actual del oficio al que se considera debería ser responsable de la labor en cuestión. La estimación representa un escenario ficticio en el cual el desarrollo del proyecto no se realizó en un entorno académico, sino dentro del ámbito laboral por dos empleados de una empresa cualquiera: un **Programador** y un **Analista Software**.

Como aclaración, las horas dedicadas al *aprendizaje* y *otros*[3] no se van a tener en cuenta a la hora de calcular el presupuesto, ya que, a diferencia de los salarios y precios de licencias de herramientas, la *formación* requerida para el proyecto no se puede estimar con tanta certeza por la presencia de variables como la posibilidad de que el empleado al que se le asigne la labor ya tenga los conocimientos necesarios. En caso de ser necesaria formación adicional, el presupuesto varía en función de la intensidad de la formación, la duración, cuánto tiempo pueda llevar al empleado adquirir dichos conocimientos, etc. Por ese motivo, en la estimación de este documento no se ha incluido un coste de formación.

Los datos sobre los sueldos se han obtenido de la web tecnoempleo [69] que se trata de un portal de empleo *online* específicamente para ofertas de trabajo en el campo de la Informática y las Telecomunicaciones. Según indica la propia web[4] estos promedios se obtienen en base a las ofertas presentes en la misma web, por lo que dependiendo del momento en que se acceda a dicho portal, los datos y, por lo tanto, el presupuesto calculado a través de ellos puede ser distinto del presente en el documento. Una vez aclarado lo necesario, se procede con el presupuesto:

- La cantidad de horas trabajadas por el **Programador** son aproximadamente 130 (105 de escritura e implementación de código[5]).
- La del **Analista Software** son 110.
- Según *Tecnoempleo* el salario promedio de un **Programador** es de **41 100 €** al año o, aproximadamente, **19.78 €** por hora.
- Según *Tecnoempleo* el salario promedio de un **Analista Software** es de **38 800 €** al año o, aproximadamente, **18.65 €** por hora.

Esto implica un presupuesto en **mano de obra** de **4 622.9 €**. A esto se le sumaría el coste de la licencia Estándar de **Visual Paradigm**, que sería un añadido de **16.54 €** si la licencia de un mes fuese suficiente o **49.63 €** si se prefiriese la licencia de tres meses. Este último añadido sería en función del reparto de días laborables del analista.

[3] Esta entrada representa cualquier otra actividad de menor importancia que haya ocurrido durante el transcurso del desarrollo del proyecto, actualizaciones, mantenimiento, márgenes...

[4] En el enlace proporcionado, al final del listado de salarios hay una anotación.

[5] Las 10 restantes de esa actividad son las que le lleva al Analista la parte correspondiente a la implementación en la documentación.

Capítulo 4

Especificación de Requisitos

En este capítulo se presentan, de forma detallada, las funcionalidades, objetivos y requisitos del sistema. Establecer estos aspectos en una etapa temprana ayuda en gran medida al desarrollo del proyecto, ya que permite disponer de una guía para el proceso de desarrollo para múltiples matices del proceso como pautas a seguir. También sirve de guía a la hora de determinar qué **es necesario** y qué **no lo es**, además es siempre uno de los primeros pasos cuando el desarrollo del proyecto se lleva a cabo siguiendo el *Modelo de Prototipos*. En el caso de este proyecto, como se menciona en el apartado anterior, se ha utilizado como metodología el **Proceso Unificado**, por lo que disponer lo antes posible de la especificación para el proyecto resulta extremadamente beneficioso. Cabe destacar que esta especificación **no** es la lista de instrucciones para desarrollar el proyecto, sí que sirve **conceptualmente** como una base para dicho proceso, pero **no** se trata de algo que deba seguirse al pie de la letra, ya que, en proyectos de esta naturaleza hay múltiples variables y factores que no pueden preestablecerse.

Este capítulo presenta una descripción (textual) detallada de la aplicación, el listado de requisitos **Funcionales**, **No-Funcionales** y **de Información** del sistema, seguido por el **Modelo de Casos de Uso**, que en este caso son **dos** (el motivo de esto se explicará en la sección correspondiente), y finalmente, la especificación de los Casos de Uso.

4.1. Descripción del sistema

FreeWheeling Travelers es, como ya se ha mencionado anteriormente, un sistema que busca actuar como una ayuda y asistencia a la enseñanza de **SQL**. Hace esto basándose en la **Gamificación**, en las denominadas *Aventuras conversacionales* [7] [1], *Aventuras Gráficas* [8] y un género relativamente conocido de libros llamado "*Choose Your Own Adventure*" (**CYOA**) o en español "*Escoge Tu Propia Aventura*" cuyo nombre viene de la serie de libros [9] con el mismo nombre que, a pesar de no ser el primer ejemplo del género, sí que es la que

[1] También denominadas Aventuras de texto.

popularizó el género. Sin entrar en detalles, el concepto de este tipo de libros es que están escritos en segunda persona para que el lector adopte el punto de vista del protagonista y que cada libro narra una trama principal. Sin embargo, en vez de narrar solo una historia, el libro contiene múltiples variaciones de dicha trama general, con distintos desenlaces y etapas.

El objetivo de este tipo de narrativas es ofrecer una experiencia de lectura personalizada. A medida que el lector avanza en la trama, se encuentra con puntos en los que el protagonista debe tomar una decisión. En esos escenarios, se indica al lector a que página debe dirigirse en función de la opción que haya escogido. Este mecanismo resulta en el desarrollo de diferentes subtramas o historias alternativas, de modo que, cada libro del género no presenta una única historia lineal, sino un conjunto de relatos posibles diseñados en torno a una trama o temática común. Gracias a ello, la narrativa experimentada por cada lector varía en función de las decisiones que hayan tomado, coincidiendo exclusivamente cuando estos escojan las mismas alternativas.

El sistema se ha desarrollado en base a ese mecanismo. Este sistema permite a usuarios identificados (*loggeados*) en el sistema, dependiendo de si el usuario es un **Profesor** o un **Estudiante**, administrar y gestionar (incluye crear, configurar, modificar y eliminar) **Niveles** compuestos por **Bloques** con distintos **roles** y que, bajo las condiciones apropiadas, pueden a su vez contener **Tests** o[2] **jugar** a dichos niveles respectivamente.

Para facilitar la comprensión del sistema es útil asociar las entidades y elementos del sistema con ejemplos del escenario de los libros CYOA, los usuarios Profesor representan a los **autores del libro**, los usuarios Estudiante los **lectores**, cada **Nivel** es un **libro** compuesto totalmente por **Bloques**, que son las **páginas**, conectados entre sí en tal orden que cada vez que el estudiante, jugando el nivel, llega a una bifurcación (instancia de Bloque cuyo atributo **rol** tiene como valor Bifurcación) equivale a un lector llegando a uno de esos puntos en los que se debe tomar una decisión, y en base a esa decisión, el lector debe saltar a una página determinada. En este caso, cuando el estudiante, interactuando con la interfaz del sistema, escoge la opción que desea, el sistema se encarga de enviar al estudiante al bloque (página) correspondiente. Esto se repite hasta que el lector llega al final de la historia o, en el caso del proyecto, hasta que el estudiante alcanza un bloque que no enlaza con ningún otro bloque. Una vez llegado a ese punto, igual que en el caso del libro se considera que el lector ha terminado su historia, el estudiante ha completado el nivel[3] y el sistema actúa acorde.

Hasta aquí se ha podido observar la clara inspiración en el sistema en CYOA y, por proximidad, con las aventuras gráficas y conversacionales, pero no se ha detallado aún donde encaja la **Gamificación** (más allá de los intrínsecos al mecanismo de los libros CYOA) ni el apoyo a la enseñanza de SQL. Esto es porque **ambos** conceptos vienen juntos en la forma de los bloques Pregunta, que son instancias de Bloque cuyo rol es Pregunta. Antes de continuar explicando por qué la gamificación y enseñanza (apoyo) de SQL la introducen estas instancias de bloque, es momento de explicar que es exactamente el rol de un bloque.

[2]En realidad los usuarios Profesor tienen acceso a **toda** la funcionalidad que ofrece la aplicación, así que no solo son capaces administrar niveles, sino que también pueden jugarlos, sin embargo, se ha utilizado "o" para destacar que los estudiantes **no** pueden administrar niveles, solo jugarlos.

[3]Aquí con completado nos referimos a que, al igual que en el caso del libro no es que el lector se haya leído todas las páginas del libro, sino que ha leído al completo una de las variaciones de la historia narrada en el libro. Lo mismo pasa con el nivel, con completado no nos referimos a que haya jugado todos los bloques del nivel, sino a que ha completado una de las tramas al completo.

Rol es uno de los múltiples atributos de la clase **Bloque** y señala el papel del bloque en el **Nivel** al que pertenece. Esta información es utilizada tanto para la administración de niveles como durante la ejecución de un nivel. En el caso de gestión, cuando un profesor crea y configura un bloque, lo que el sistema hace es cambiar los valores de los atributos del Bloque correspondiente. No todos los datos del Bloque son siempre relevantes. En función del rol de este, al sistema solo le interesa un set específico de atributos, por ello, está diseñado para, durante la configuración de un bloque, habilitar solo los campos relevantes, ocultando la información no pertinente y facilitando el proceso.

Por otro lado, a la hora de jugar a un nivel, igual que los libros se leen página a página, los niveles se juegan bloque a bloque, y es el rol de cada bloque (el valor del atributo rol de la instancia de bloque que el estudiante está jugando actualmente) lo que indica al sistema cómo debe utilizar dicho bloque. En otras palabras, el rol de un bloque es lo que permite al sistema, mientras se está jugando un nivel, determinar como se juega ese bloque y cuál es el procedimiento a seguir. Solo hay tres tipos de rol que puede tener un bloque en el sistema, son mutuamente excluyentes (solo puede ser uno de los tres, un bloque no puede tener dos roles) y son:

1. Narrativa: Este rol es el único que no se ha mencionado aún. Es el rol más simple de todos, ya que los bloques Narrativa son el equivalente al resto de páginas del libro, no tienen función otra más que presentar al jugador partes de la trama del nivel.

2. Bifurcación: Se ha mencionado antes, este rol significa que el bloque representa un cruce de caminos, es uno de los puntos en los que la trama se separa en diversas variaciones y le indica al sistema que, al contrario que el resto de bloques, que solo pueden preceder a otro bloque (o ninguno si se trata de un final de trama), son los únicos bloques que apuntan a más de un bloque, y al igual que las tomas de decisiones en los libros, apuntan a tantos bloques como opciones sean proporcionadas al jugador. Este rol es lo que ha permitido replicar el mecanismo de los libros CYOA en el proyecto, ya que le indica al sistema que la continuación bloque actual no es una fija, y que debe, en función de la opción que escoja el jugador mientras juega a dicho bloque, cuando sea momento de enviar al jugador al bloque siguiente, deberá enviar al jugador al bloque correspondiente a la opción que haya escogido. Es aquí donde se materializa como tal dicha mecánica porque, mientras que el resto de roles solo tienen **un único** bloque siguiente (o **ninguno** si el bloque representa el final de una trama), los bloques bifurcación no tienen una única continuación, sino que tienen tantas como opciones se le presenten en el bloque al jugador, y cada una de esas opciones tiene un bloque correspondiente, por lo que se podría decir que los bloques bifurcación apuntan a tantos bloques como opciones le den al jugador y dividen la historia del nivel en tantas variantes como opciones. De esta forma, el sistema imita a los libros, ya que en las páginas de toma de decisiones (los bloques bifurcación) la continuación de la trama no está necesariamente en la página a continuación de la actual, mientras que en el resto de páginas (el resto de roles), la continuación de la trama desarrollada en la página `X` está en la página `X+1`.

3. Pregunta: Este **rol**, o más concretamente los bloques con este rol, permiten incorporar al sistema tanto **Gamificación**, como la funcionalidad de refuerzo a la enseñanza de SQL. Este rol como tal no tiene un equivalente directo en el ejemplo de los libros, pero

se puede pensar en los bloques con este rol como en los puntos de la trama de mayor tensión donde el protagonista se enfrenta a retos y desafíos.

Resulta que los términos **retos** y **desafíos** son algo muy frecuente en el ámbito de la gamificación y como plantea **Tamás Németh** mediante citas a otras eminencias en el campo de la gamificación, "*Levels, being defined steps in player progression, contain at least one challenge. Subsequent levels usually grow in complexity and feature new opponents, but also unlock new items or abilities*" [47] que, en español, significa " Los niveles, al ser pasos definidos en la progresión de un jugador, contienen al menos un desafío" motivo por el cual se puede decir que los desafíos, resultan interesantes e incluso divertidos al ser humano.

Existen múltiples evidencias de la relación entre los desafíos y el interés en los humanos, se observa en programas de la tele como *Saber y Ganar* o *PasaPalabra*, en los que el conocimiento de los participantes se pone a prueba en un entorno en el que solo pueden beneficiarse (no hay riesgo de pérdida), y en juegos de mesa como *Trivial* y *Scattergories*.

Como se ha dicho, enfrentarse a algo que pone a uno a prueba (un desafío), pero no en un entorno en el que hay un riesgo real (un juego) resulta entretenido a la mayoría, y en este proyecto, el entorno lo constituyen los niveles y los desafíos las preguntas de los bloques del nivel. O dicho de otra forma, para el ser humano, el poner a prueba los conocimientos propios, bajo las condiciones adecuadas, es una actividad entretenida, que tiene valor por sí misma y, por ende, resulta intrínsecamente interesante y este sistema facilita todo ello.

Está cualidad de los desafíos es la que se incorpora al sistema mediante los bloques **Pregunta** (incorporandose así tanto los aspectos de gamificación como de la función de refuerzo a la enseñanza de SQL). Estos bloques contienen en sus campos:

- La pregunta o problema SQL que el alumno debe lograr superar para poder avanzar por el nivel.
- El contexto suficiente con el que el estudiante debería de ser capaz de idear una solución al problema.
- Los recursos que permiten al sistema preparse para poder llevar a cabo el proceso de evaluación de respuestas a dicha pregunta.
- Los detalles del bloque que permiten al sistema comprobar si los resultados de la respuesta del alumno corresponden a los deseados o no. O en otras palabras, si los efectos de la propuesta del estudiante son evidencia de haber solucionado el problema planteado o no.

Merece la pena recordar que es responsabilidad del autor de una pregunta diseñarla adecuadamente y proporcionar al sistema los datos correspondientes.

La **pregunta** es, intrínsecamente, una forma de desafío y si a esto se le suma el resto de la experiencia que es jugar un nivel (la trama coherente y que se desarrolla bloque a bloque, las temáticas de cada nivel, la toma de decisiones...) resulta evidente que el sistema ciertamente está gamificado, y al diseñar las preguntas de forma que pongan a prueba los conocimientos del jugador sobre SQL, sirve como refuerzo a las lecciones sobre SQL dadas en las distintas asignaturas que cursa el estudiante. Este punto se extiende más aún, ya que el sistema no solo evalúa las respuestas informando al jugador

sobre si su respuesta es correcta o incorrecta, sino que gracias a la forma en que lo hace, si la respuesta es incorrecta, el sistema proporciona *feedback* al jugador, indicando a este cuál puede ser el problema de su respuesta, pero sin mostrar directamente la solución correcta.

Al hablar sobre los bloques Pregunta se menciona que contienen los datos que indican al sistema como **prepararse** para evaluar respuestas a la pregunta del bloque, pero no se ha mencionado aún como es capaz el sistema de evaluar estas respuestas. La clave para esto es un elemento del sistema que se ha mencionado previamente, los **Tests**. Los Tests disponen de unos atributos que contienen la información que indica al sistema exactamente qué debe hacer para examinar un aspecto de la respuesta del jugador, cómo debe interpretar los resultados de dicho examen y, en caso de que la respuesta no supere el test, la información sobre cuál puede ser el motivo por el que la respuesta es incorrecta (evidentemente esta información se define en base a que es exactamente lo que el test examina). Funciona de tal forma que solo los bloques Pregunta pueden contener instancias de Test y el conjunto de todas las instancias de Test de una pregunta constituye la batería de pruebas de la pregunta, la idea es que el autor de la pregunta cree para esta tantos Tests como considere necesarios para garantizar que la respuesta del jugador a la pregunta es correcta, aprovechando el alto grado de detalle que permite el sistema a la hora de configurar tanto la pregunta como sus tests, para que el conjunto de Preguntas y Tests sean tan flexibles o estrictos como el autor desee.

A nivel conceptual, el sistema determina si la respuesta es correcta o no ejecutándola en un entorno aislado (**Contenedores de Evaluación** y sus bases de datos **MySQL**) en el que dispone de los datos de la pregunta necesarios para llevar a cabo la evaluación. Tras la ejecución, los tests definidos para la pregunta se encargan de buscar evidencias de error (y/o de solución), es decir, los antes mencionados efectos de la respuesta. El sistema ejecuta estos tests y, dependiendo de si estos han captado evidencias de respuesta errónea o no, el sistema declara la respuesta del alumno correcta o incorrecta.

Conviene mencionar que, a medida que la pregunta se vuelve más específica, existen más posibilidades de que una respuesta sea incorrecta[4]. Por ejemplo, si la pregunta solicita al jugador que redacte un *statement* SQL para introducir una fila específica en una tabla específica, un solo examen que consista en seleccionar una entrada que coincida con la fila del enunciado resulta suficiente para determinar si la respuesta del jugador es correcta o no (para determinar si el código escrito por el alumno es capaz de realizar lo solicitado en el enunciado). No obstante, si la pregunta especifica que debe realizar lo solicitado, pero sin modificar el número de filas en la tabla (antes podía solucionarse mediante un `INSERT` o un `UPDATE`, ahora solo mediante `UPDATE`), entonces el primer examen que verifica si la entrada en cuestión está presente en la tabla ya no es suficiente por sí mismo y resulta necesario un segundo examen que verifique si la entrada en cuestión está presente en la tabla.

En resumen, el sistema somete a la respuesta del jugador al conjunto de todos los tests creados para la pregunta a la que responde, siendo declarada la respuesta como correcta **si y solo si**, esta supera **todos** los tests, mientras que, en el instante en que no supere uno de los tests, el sistema declara la respuesta como incorrecta, finaliza el proceso de evaluación

[4]Cuanto más específica es la pregunta, más específica es la solución y por ello, propuestas más ambiguas que solucionaban el problema cuando este no era tan específico, dejan de ser viables como solución.

y proporciona al jugador *feedback* correspondiente al test que haya fallado, de esta forma el jugador puede leer ese *feedback*, pensar en una respuesta nueva en base a los detalles proporcionados por el sistema y responder de nuevo, repitiéndose este bucle tantas veces como intentos necesite el jugador hasta dar con una respuesta correcta.

Para terminar la descripción del sistema, cabe destacar que aunque los aspectos más importantes del proyecto son la creación de estos niveles, el poder jugarlos y el que el sistema sea capaz de evaluar las respuestas de los jugadores sin intervención humana (más allá de la adecuada configuración de la pregunta y sus tests) y por ello la descripción se ha enfocado en ellos, el sistema también proporciona los medios para que los usuarios puedan, registrarse en el sistema, iniciar sesión, escoger que nivel quieren jugar, moverse por las páginas de la web de forma relativamente sencilla e intuitiva y poder cerrar sesión una vez terminen.

4.2. Requisitos

El *Glosario de Estándares de la IEEE para Terminología de Ingeniería del Software* [41] establece que un requisito es:

1. *Condición o capacidad que el sistema, o un componente del sistema, debe tener o cumplir para satisfacer un contrato, estándar u otro documento impuesto formalmente.*
2. *Condición o capacidad que un usuario necesita para poder resolver un problema o alcanzar un objetivo.*
3. *Una representación documentada de una instancia de los anteriores dos puntos.*

Una vez establecido esto, se procede a listar los distintos requisitos del sistema.

4.2.1. Requisitos Funcionales

Se entiende por **requisito funcional** a cualquier requisito que defina lo que el sistema **debe hacer**.

RF.1) El sistema debe permitir a los distintos usuarios identificarse e iniciar sesión.

RF.2) El sistema debe permitir a los usuarios cerrar sesión.

RF.3) El sistema debe permitir y dar los medios a los profesores para crear, editar y eliminar Niveles.

RF.4) El sistema debe permitir y dar los medios a los profesores para crear, configurar y eliminar Bloques.

RF.5) El sistema debe permitir y dar los medios a los profesores para crear, configurar y eliminar Tests.

RF.6) El sistema debe permitir a los usuarios jugar y explorar los distintos caminos de los que están compuestos los niveles.

RF.7) El sistema debe ser capaz de guardar el progreso de los distintos usuarios.

RF.8) El sistema debe ser capaz de reconocer cuando unas credenciales no son válidas e impedir el inicio de sesión.

RF.9) El sistema debe almacenar las respuestas correctas de los estudiantes a las distintas preguntas de los niveles.

RF.10) El sistema debe ser capaz de enfrentarse y gestionar errores y excepciones.

RF.11) El sistema debe ser capaz de gestionar el acceso de los usuarios a los niveles.

RF.12) El sistema debe ser capaz de evaluar correctamente las respuestas de los estudiantes.

RF.13) El sistema debe ser capaz de gestionar los servicios dados a los usuarios y limitar estos en función del tipo de usuario del que se trate.

RF.14) El sistema permite a los profesores no solo diseñar niveles jugables por los estudiantes, sino que además permite incorporar en estos distintas narrativas y/o temáticas con las que atraer el interés de los jugadores.

RF.15) El sistema debe permitir incorporar múltiples hilos narrativos y/o desenlaces en un mismo nivel.

RF.16) El sistema debe proporcionar a los usuarios con rol Profesor una plataforma para crear y configurar niveles en los que se desarrolle una historia de temática y narrativa al gusto del autor en la que múltiples posibles hilos narrativos y desenlaces puedan experimentarse en base a las decisiones tomadas por el jugador durante el transcurso del nivel.

4.2.2. Requisitos No-Funcionales

Se entiende por **requisito no-funcional** a cualquier requisito que defina, no las funcionalidades del sistema, sino las características de este. Es decir, definen **no** el qué deben hacer, sino **cómo** deben de hacerlo, como debe comportarse el sistema.

RNF.1) El usuario interactúa con el sistema vía servicio web.

RNF.2) El sistema debe garantizar la seguridad de la información de los usuarios.

RNF.3) El sistema debe poder gestionar la información de múltiples usuarios.

RNF.4) Únicamente los usuarios tipo-admin (usuarios profesor) tienen acceso a los servicios de creación, configuración y borrado de Niveles, Bloques y Tests.

RNF.5) El acceso a los niveles dependerá del rol del usuario, los profesores tendrán acceso a todos los niveles del sistema mientras que los estudiantes solo a los niveles marcados como listos para ser jugados.

RNF.6) El sistema debe operar mediante bases datos MySQL.

RNF.7) Cada usuario del sistema dispondrá de un identificador único.

RNF.8) El sistema debe funcionar de forma consistente y estable.

RNF.9) El programa estará desarrollado en python.

RNF.10) La aplicación estará desarrollada usando el micro-framework Flask.

RNF.11) La utilización de los servicios del sistema pensados para estudiantes no requerirá formación específica más allá de conocimientos sobre SQL (que son proporcionados por las lecciones).

RNF.12) Los componentes visuales del sistema presentarán un formato, fuente y tamaño de letra que facilite la visualización de los mismos.

RNF.13) La interfaz del sistema será clara y de fácil utilización.

RNF.14) Por la naturaleza de la aplicación, el sistema debe estar diseñado de forma que sea capaz de gestionar, protegerse y/o limitar dentro de lo posible los intentos y posibles efectos de agresores.

RNF.15) El sistema debe resultar intuitivo a la hora de ser utilizado.

RNF.16) El sistema debe proporcionar contexto e información adicional en aquellos puntos que puedan resultar más complejos.

RNF.17) La documentación, así como el contenido de la aplicación, estará escrito en Español.

4.3. Requisitos de Información

Se entiende por **requisito de información** (o requisito de datos) a una categoría específica de requisito que define la información que debe manejar un sistema. Se diferencia de los otros tipos de requisitos en el hecho de que se centra exclusivamente en los datos en sí. A diferencia de los requisitos funcionales (qué debe hacer el sistema) y no funcionales (cómo debe comportarse), los requisitos de información se centran en los datos como tal.

RI.1) El sistema debe disponer de las credenciales de todos los usuarios del sistema.

RI.2) El sistema debe disponer de los datos sobre los distintos bloques creados en el sistema.

RI.3) El sistema debe disponer las características y la configuración de todos los niveles del sistema.

RI.4) El sistema debe disponer de los datos sobre los distintos tests diseñados para cada bloque **Pregunta** del sistema.

RI.5) El sistema debe disponer de las respuestas correctas dadas por cada usuario.

RI.6) El sistema guardará información sobre los usuarios del mismo, entre ellos: identificador, clave, alias, tipo de usuario, etc.

RI.7) El sistema dispondrá de sets de datos ficticios, así como de comandos, en forma de código MySQL, **almacenado** como *string*, creados por los Profesores como parte de los atributos de ciertas instancias de Bloque y de Test.

RI.8) El sistema guardará información sobre las etapas y preguntas del sistema, tales como: identificadores, narrativa, autoría, visibilidad y otros elementos de su configuración.

RI.9) El sistema guardará información sobre las distintas clases de su dominio: **Progreso**, **Nivel**, **Respuesta**, **Usuarios**, **Bloque** y **Test**.

RI.10) El sistema debe de disponer de acceso a datos como la fecha y hora actuales.

4.4. Modelos de Casos de Uso

Antes de mostrar los diagramas de casos de uso del sistema, conviene aclarar el motivo por el que se han diseñado dos diagramas y no uno. El motivo es simple, algunos de los servicios que el sistema debe proporcionar **no** deben ser proporcionados a cualquier usuario, servicios como "Crear niveles" o "Crear bloques" solo deben estar disponibles para Profesores, es por esto que, conceptualmente se habla de dos actores principales a pesar de que estos generalicen de un Usuario **base**. El usuario Profesor tiene acceso a todos los servicios proporcionados por el sistema, mientras que el usuario Estudiante **no** puede acceder a los servicios relacionados con la creación, modificación y borrado de ningún componente del sistema.

En la Figura 4.1 se refleja esta jerarquia entre los tipos de actor.

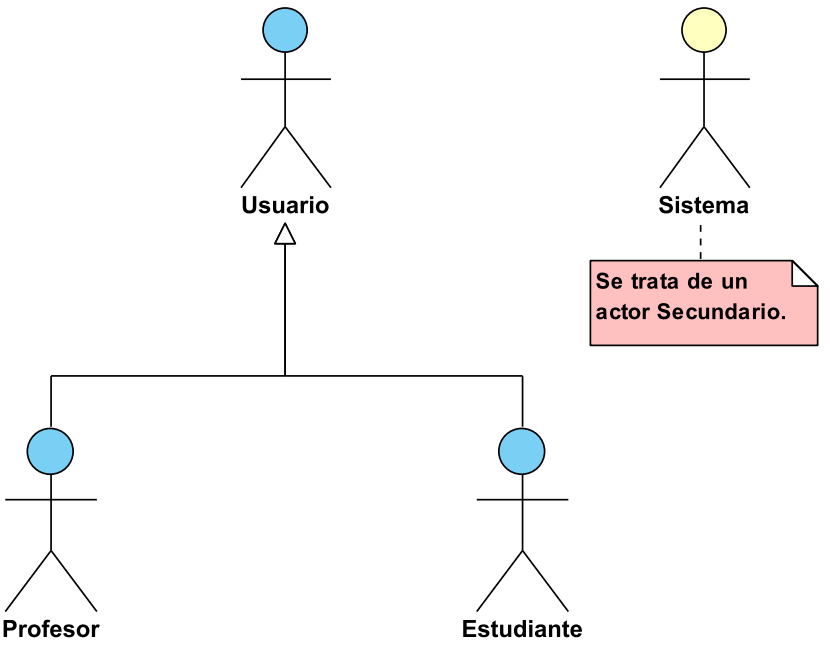

Figura 4.1: Diagrama que refleja la jerarquia de los usuarios del sistema.

Más allá del Profesor y el Estudiante, puede considerarse al conjunto de la aplicación junto con su base de datos como un actor secundario denominado Sistema. Una vez aclarado esto, a continuación, en las Figuras 4.2 y 4.3 se muestran los diagramas casos de uso de cada actor.

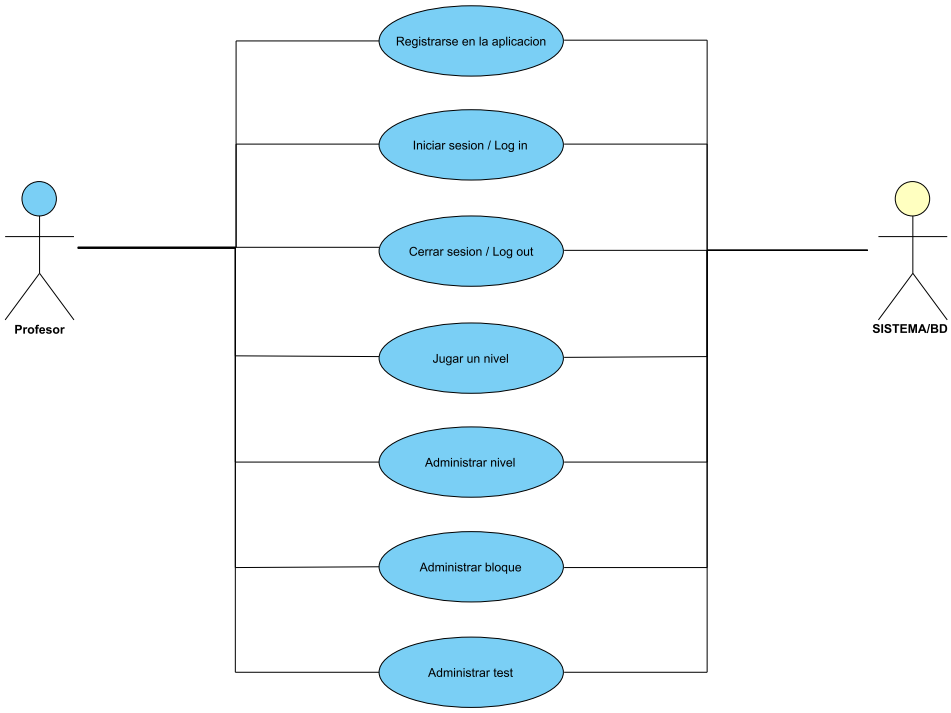

Figura 4.2: Diagrama de casos de uso en los que participa el actor Profesor.

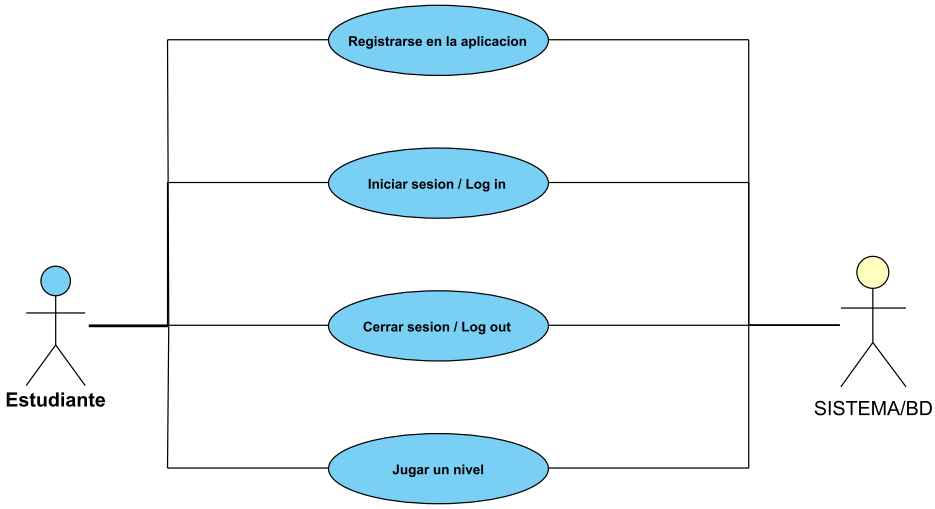

Figura 4.3: Diagrama de casos de uso en los que participa el actor Estudiante.

4.5. Especificación de Casos de Uso

Un caso de uso es "una secuencia de acciones realizadas por un sistema que produce un resultado observable de valor a un actor particular" [43]. Por ello, la especificación de un caso de uso es la descripción detallada de un conjunto de acciones o actividades que pueden ser realizadas dentro del sistema. El objetivo de la identificación y descripción de los casos de uso es definir: las posibles acciones más relevantes que forman parte del sistema y como este se debe comportar durante su realización. Para ello se especificará el flujo normal del caso de uso, la realización exitosa de una acción o actividad, y aquellos flujos alternativos ante imposibilidades de realización o fallos, tanto del sistema, como de los actores, a la hora de realizar cada uno de los casos de uso.

En esta sección se describirán los casos de uso identificados en las Figuras 4.2 y 4.3.

Caso de uso:	Registrarse en la aplicación.
Descripción:	El caso de uso permitirá a un usuario sin identificar crear una nueva cuenta de usuario en la aplicación, registrándose en la misma.
Actores:	Profesor, Estudiante, Sistema.
Precondiciones:	El Actor no ha iniciado aún sesión.
Postcondiciones:	El usuario queda registrado en el sistema.

Secuencia normal

1. El caso de uso comienza cuando un Actor (Usuario) solicita Registrarse.

2. El sistema solicita al Actor los datos necesarios para crear una nueva instancia de Usuario.

3. El Actor proporciona los datos.

4. El sistema comprueba si los datos proporcionados por el Actor son **válidos** y están disponibles.

5. El sistema crea la nueva instancia de Usuario.

6. El sistema redirige al Actor a la página de *Login*, finalizando el Caso de Uso.

Excepciones

0) El Actor podrá abortar el caso de uso en cualquier momento, quedando el caso de uso sin efecto.

2.a), **4.a)**, **5.a)**, **6.a)** Si el sistema no puede realizar la acción, muestra un mensaje de error y el caso de uso finaliza.

4.b) Si los datos proporcionados no son válidos (o no han sido proporcionados) o no están disponibles, el sistema avisa al Actor, no se inicia sesión y el caso regresa al paso 2.

Tabla 4.1: Especificación del caso de uso "Registrarse en la aplicación".

Caso de uso:	Iniciar sesión / Log in.
Descripción:	Caso de uso mediante el cual el Actor puede identificarse en el sistema para acceder al resto de funciones.
Actores:	Estudiante, Profesor, Sistema.
Precondiciones:	El Actor no ha iniciado sesión aún.
Postcondiciones:	El sistema inicia una nueva sesión de usuario.

Secuencia normal

1. El caso de uso comienza cuando un Actor solicita iniciar sesión y el sistema dirige al Actor a la página correspondiente.

2. El Actor proporciona sus credenciales.

3. El sistema comprueba las credenciales proporcionadas para verificar que el usuario este dado de alta.

4. El sistema inicia la nueva sesión de usuario, finalizando el Caso de Uso.

Excepciones

3.a), **4.a)** Si el sistema no puede realizar la acción, se muestra un mensaje de error y el caso de uso finaliza.

3.b) Si las credenciales proporcionadas no son las de un usuario registrado en el sistema, se informa al Actor de dicho hecho y el proceso es cancelado, siendo el Actor redirigido por el sistema al paso 2 de la secuencia.

Tabla 4.2: Especificación del caso de uso "Iniciar sesión".

Caso de uso:	Cerrar sesión.
Descripción:	El caso de uso permitirá a los actores cerrar sesión en el sistema.
Actores:	Estudiante, Profesor.
Precondiciones:	El Actor tiene una sesión activa en el sistema.
Postcondiciones:	El sistema cierra la sesión y lleva al Actor a la página correspondiente.
Secuencia normal	
1. El caso de uso comienza cuando un Actor solicita **Cerrar sesión**.	
2. El sistema cierra la sesión del usuario en cuestión y redirige a este a la página correspondiente, finalizando el Caso de Uso.	

Tabla 4.3: Especificación del caso de uso "Cerrar sesión".

Caso de uso:	Jugar un nivel.
Descripción:	Caso de uso que representa el bucle de pasos involucrados en el proceso por el que un usuario del sistema juega un nivel de su elección hasta completarlo.
Actores:	Estudiante, Profesor, Sistema.
Precondiciones:	El Actor ha iniciado sesión, existe **al menos** una instancia de nivel disponible para jugar en el sistema.
Postcondiciones:	Se registraran las respuestas correctas proporcionadas por el usuario. Si es la primera vez que el usuario completa el nivel, se registra el suceso.

Secuencia normal

1. El caso de uso comienza cuando el Actor solicita jugar un nivel.

2. El sistema redirige al Actor a la página del primer bloque del nivel seleccionado.

3. El primer bloque (o *bloque raíz*) de un nivel siempre es un bloque de narrativa, así que el Actor lee la narrativa del bloque y solicita continuar.

4. El sistema redirige al Actor al bloque siguiente.

5. En función del rol del bloque el Actor actúa acorde: ***Narrativa***-**5.a**, ***Bifurcación***-**5.b** o ***Pregunta***-**5.c**.

6. Cuando el Actor solicita continuar, en función de si el bloque actual es un final de nivel o no, el sistema actúa de dos formas distintas:

6.1. Si el bloque actual **no** es un final del nivel, el sistema redirige al Actor al bloque siguiente, y la secuencia regresa al paso **4**.

6.2. Si el bloque actual **es** un final del nivel el sistema actúa acorde.

7. Al alcanzar un final de nivel el sistema:

7.1. Congratula al Actor por superar el nivel.

7.2. Comprueba si es la primera vez que el usuario supera el nivel jugado y, en caso de serlo, registra el suceso.

7.3. Redirige al Actor a la página correspondiente, finalizando el Caso de Uso.

Caminos alternativos

5.a) El Actor juega el bloque **Narrativa** leyendo la narrativa y solicitando continuar.

5.b) El Actor juega el bloque **Bifurcación** tomando la decisión que desee y solicitando continuar.

5.c) El Actor juega el bloque **Pregunta** respondiendo correctamente, solicitando la evaluación de su respuesta y, tras ello, solicitando continuar.

6.a) Si el bloque actual es un bloque **Bifurcación**, el sistema redirige al Actor al bloque correspondiente a la decisión seleccionada.

Excepciones

0) El Actor podrá abortar el caso de uso, quedando este sin efecto.

3.a) Si el nivel solo tiene un bloque, el sistema considerará el nivel completado en cuanto el Actor solicite continuar, saltando directamente al paso **7**.

6.a) Si el Actor responde incorrectamente a la pregunta, el sistema informa al Actor de este hecho, proporciona el feedback correspondiente y la secuencia regresa al paso **5.c**.

Tabla 4.4: Especificación del caso de uso "Jugar Nivel".

Caso de uso:	Administrar Nivel.
Descripción:	Este caso de uso encapsula las tres operaciones, que un Actor Profesor puede llevar a cabo mediante la aplicación, que pertenecen a la administración de niveles: **Crear** un nivel, **Configurar** o **Editar** un nivel y **Eliminar** un nivel. El caso de uso describe un escenario en el que el Actor crea un nuevo nivel, lo configura y finalmente lo elimina, **sin embargo**, cualquiera de estos tres eventos puede realizarse por sí mismo siempre y cuando las condiciones sean las adecuadas.
Actores:	Profesor, Sistema.
Precondiciones:	La sesión activa del Actor es un usuario Profesor. Si se desea tan solo **configurar** o **eliminar** un nivel, es necesario que exista **al menos** un nivel en el sistema.
Postcondiciones:	Las acciones del Actor se verán reflejadas en el sistema.

Secuencia normal

1. El caso de uso comienza cuando el Actor solicita crear un nuevo nivel.

2. El sistema solicita al Actor los datos necesario para crear un nuevo nivel.

3. El Actor proporciona los datos.

4. El sistema comprueba si los datos proporcionados son válidos y están disponibles, en caso de serlo, crea el nuevo nivel.

El subcaso **Crear nivel** finalizaría en este punto.

5. El Actor solicita configurar un nivel, indicando el nivel que desea configurar.

6. El sistema redirige al Actor a la página correspondiente.

7. El **Actor** modifica la configuración del nivel al gusto.

8. El sistema comprueba los cambios propuestos por el Actor y, si son válidos, los registra.

El subcaso **Configurar nivel** finalizaría en este punto.

9. El Actor solicita eliminar un nivel, indicando el nivel que desea eliminar.

10. El sistema informa al Actor de los detalles de la acción que ha solicitado y solicita una segunda confirmación.

11. El Actor confirma de nuevo.

12. El sistema elimina el nivel seleccionado y redirige al Actor a la página correspondiente, finalizando el Caso de Uso.

El subcaso **Eliminar nivel** finalizaría en este punto.

Caminos alternativos

1) Como se ha indicado, el caso de uso actual está compuesto por tres casos menores, el escenario representado es una secuencia de los tres, uno tras otro, sin embargo, siempre y cuando se cumplan las precondiciones correspondientes, cualquiera de las tres puede ser realizada por sí sola, sin realizar las otras dos.

1.a) Si el Actor solo desease cambiar la configuración del nivel o eliminar el nivel, el caso de uso empezaría en los pasos **5** o **9** respectivamente.

Excepciones

0) El Actor podrá cancelar el caso de uso, quedando este sin efecto.

0.a) Lo establecido en la excepción anterior se aplica también a los casos en los que solo se fuese a realizar uno de los tres subcasos.

4.a) Si los datos proporcionados por el Actor no son válidos o no están disponibles, el sistema **no** crea el nuevo nivel, informa al Actor y el caso de uso regresa al paso **2**.

8.a) Si las modificaciones propuestas por el Actor no son válidas, el sistema **no** registra los cambios propuestos, informa al Actor y el caso de uso regresa al paso **7**.

Tabla 4.5: Especificación del caso de uso "Administrar Nivel".

Caso de uso:	Administrar Bloque.
Descripción:	Este caso de uso encapsula las tres operaciones, que un Actor Profesor puede llevar a cabo mediante la aplicación, que pertenecen a la administración de bloques: **Crear** un bloque, **Configurar** o **Editar** un bloque y **Eliminar** un bloque. El caso de uso describe un escenario en el que el Actor crea un nuevo bloque, lo configura y finalmente lo elimina, **sin embargo**, cualquiera de estos tres eventos puede realizarse por sí mismo siempre y cuando las condiciones sean las adecuadas.
Actores:	Profesor, Sistema.
Precondiciones:	La sesión activa del Actor es un usuario Profesor. Si se desea tan solo **configurar** o **eliminar** un bloque, es necesario que exista **al menos** un bloque en el sistema.
Postcondiciones:	Las acciones del Actor se verán reflejadas en el sistema.

Secuencia normal

1. El caso de uso comienza cuando el Actor solicita crear un nuevo bloque.

2. El sistema solicita al Actor los datos necesario para crear un nuevo bloque.

3. El Actor proporciona los datos.

4. El sistema comprueba si los datos proporcionados son válidos y están disponibles, en caso de serlo, crea el nuevo bloque.

El subcaso **Crear Bloque** finalizaría en este punto.

5. El Actor solicita configurar un bloque, indicando el bloque que desea configurar.

6. El sistema redirige al Actor a la página correspondiente.

7. El **Actor** modifica la configuración del bloque al gusto.

8. El sistema comprueba los cambios propuestos por el Actor y, si son válidos, los registra.

El subcaso **Configurar bloque** finalizaría en este punto.

9. El Actor solicita eliminar un bloque, indicando el bloque que desea eliminar.

10. El sistema informa al Actor de los detalles de la acción que ha solicitado y solicita una segunda confirmación.

11. El Actor confirma de nuevo.

12. El sistema elimina el bloque seleccionado y redirige al Actor a la página correspondiente, finalizando el Caso de Uso.

El subcaso **Eliminar bloque** finalizaría en este punto.

Caminos alternativos

1) Como se ha indicado, el caso de uso actual está compuesto por tres casos menores, el escenario representado es una secuencia de los tres, uno tras otro, sin embargo, siempre y cuando se cumplan las precondiciones correspondientes, cualquiera de las tres puede ser realizada por sí sola, sin realizar las otras dos.

1.a) Si el Actor solo desease cambiar la configuración del bloque o eliminar el bloque, el caso de uso empezaría en los pasos **5** o **9** respectivamente.

Excepciones

0) El Actor podrá cancelar el caso de uso, quedando este sin efecto.

0.a) Lo establecido en la excepción anterior se aplica también a los casos en los que solo se fuese a realizar uno de los tres subcasos.

4.a) Si los datos proporcionados por el Actor no son válidos o no están disponibles, el sistema **no** crea el nuevo bloque, informa al Actor y el caso de uso regresa al paso **2**.

8.a) Si las modificaciones propuestas por el Actor no son válidas, el sistema **no** registra los cambios propuestos, informa al Actor y el caso de uso regresa al paso **7**.

Tabla 4.6: Especificación del caso de uso "Administrar Bloque".

Caso de uso:	Administrar Test.
Descripción:	Este caso de uso encapsula las tres operaciones, que un Actor Profesor puede llevar a cabo mediante la aplicación, que pertenecen a la administración de tests: **Crear** un test, **Configurar** o **Editar** un test y **Eliminar** un test. El caso de uso describe un escenario en el que el Actor crea un nuevo test, lo configura y finalmente lo elimina, **sin embargo**, cualquiera de estos tres eventos puede realizarse por sí mismo siempre y cuando las condiciones sean las adecuadas.
Actores:	Profesor, Sistema.
Precondiciones:	La sesión activa del Actor es un usuario Profesor. Si se desea tan solo **configurar** o **eliminar** un test, es necesario que exista **al menos** un test en el sistema.
Postcondiciones:	Las acciones del Actor se verán reflejadas en el sistema.

Secuencia normal

1. El caso de uso comienza cuando el Actor solicita crear un nuevo test.

2. El sistema solicita al Actor los datos necesario para crear un nuevo test.

3. El Actor proporciona los datos.

4. El sistema comprueba si los datos proporcionados son válidos y están disponibles, en caso de serlo, crea el nuevo test.

El subcaso **Crear test** finalizaría en este punto.

5. El Actor solicita configurar un test, indicando el test que desea configurar.

6. El sistema redirige al Actor a la página correspondiente.

7. El **Actor** modifica la configuración del test al gusto.

8. El sistema comprueba los cambios propuestos por el Actor y, si son válidos, los registra.

El subcaso **Configurar test** finalizaría en este punto.

9. El Actor solicita eliminar un test, indicando el test que desea eliminar.

10. El sistema informa al Actor de los detalles de la acción que ha solicitado y solicita una segunda confirmación.

11. El Actor confirma de nuevo.

12. El sistema elimina el test seleccionado y redirige al Actor a la página correspondiente, finalizando el Caso de Uso.

El subcaso **Eliminar test** finalizaría en este punto.

Caminos alternativos

1) Como se ha indicado, el caso de uso actual está compuesto por tres casos menores, el escenario representado es una secuencia de los tres, uno tras otro, sin embargo, siempre y cuando se cumplan las precondiciones correspondientes, cualquiera de las tres puede ser realizada por sí sola, sin realizar las otras dos.

1.a) Si el Actor solo desease cambiar la configuración del test o eliminar el test, el caso de uso empezaría en los pasos **5** o **9** respectivamente.

Excepciones

0) El Actor podrá cancelar el caso de uso, quedando este sin efecto.

0.a) Lo establecido en la excepción anterior se aplica también a los casos en los que solo se fuese a realizar uno de los tres subcasos.

4.a) Si los datos proporcionados por el Actor no son válidos o no están disponibles, el sistema **no** crea el nuevo test, informa al Actor y el caso de uso regresa al paso **2**.

8.a) Si las modificaciones propuestas por el Actor no son válidas, el sistema **no** registra los cambios propuestos, informa al Actor y el caso de uso regresa al paso **7**.

Tabla 4.7: Especificación del caso de uso "Administrar Test".

Capítulo 5

Análisis

Este capítulo se dedica a la exposición del Modelo de Análisis del sistema. Se van a mostrar, el **Modelo de Dominio**, en el que se reflejan los elementos principales del sistema y las relaciones lógicas (conexiones) entre ellos para así disponer de una representación del dominio del problema. Esto se hará mediante el diagrama de clases. Tras acabar con el Modelo de Dominio se plantea el **Modelo de Análisis**.

5.1. Modelo de Dominio

La Figura 5.1 muestra el diagrama de clases del Modelo de Dominio. En él se pueden observar las siguientes clases:

- Usuarios: Representa al usuario genérico del sistema.
  - Profesor: Representa a los usuarios del sistema con el rol de Profesor.
  - Estudiante: Representa a los usuarios del sistema con el rol de Estudiante, no tienen acceso a todas las funcionalidades del sistema.
- Nivel: Representa los niveles que son gestionados por los Profesores y jugados por los Estudiantes.
- Bloque: Representa a la unidad de la que los niveles están compuestos. En función de su rol son operados por el sistema de una u otra forma. Se explicaron en más detalle en la sección 4.1.
- Test: Representa cada uno de los tests utilizados para evaluar distintos aspectos de las respuestas a la pregunta del Bloque Pregunta asociado. Su asociación con la clase **Bloque** se debe a que las instancias de Test pertenecen a instancias de Bloque, sin embargo, no es una relación de composición, y esto solo ocurre si la instancia de Bloque

tiene como rol Pregunta, si el rol del bloque **no** es Pregunta entonces no es posible crear Tests para dicha instancia.

- Respuesta: Clase asociación entre Usuario y Bloque, representa cada una de las respuestas **correctas** que el Usuario Asociado ha dado a la pregunta del Bloque asociado. Solo se crean instancias de las respuestas correctas, así que es necesario que el sistema haya evaluado la respuesta y la haya declarado correcta.
- Progreso: Clase asociación entre Usuario y Nivel, constituye la información sobre que Usuario ha completado que Nivel.
- Roles: Es un enumerado (**Enum**) que define los tres posibles roles que pueden tener los bloques.

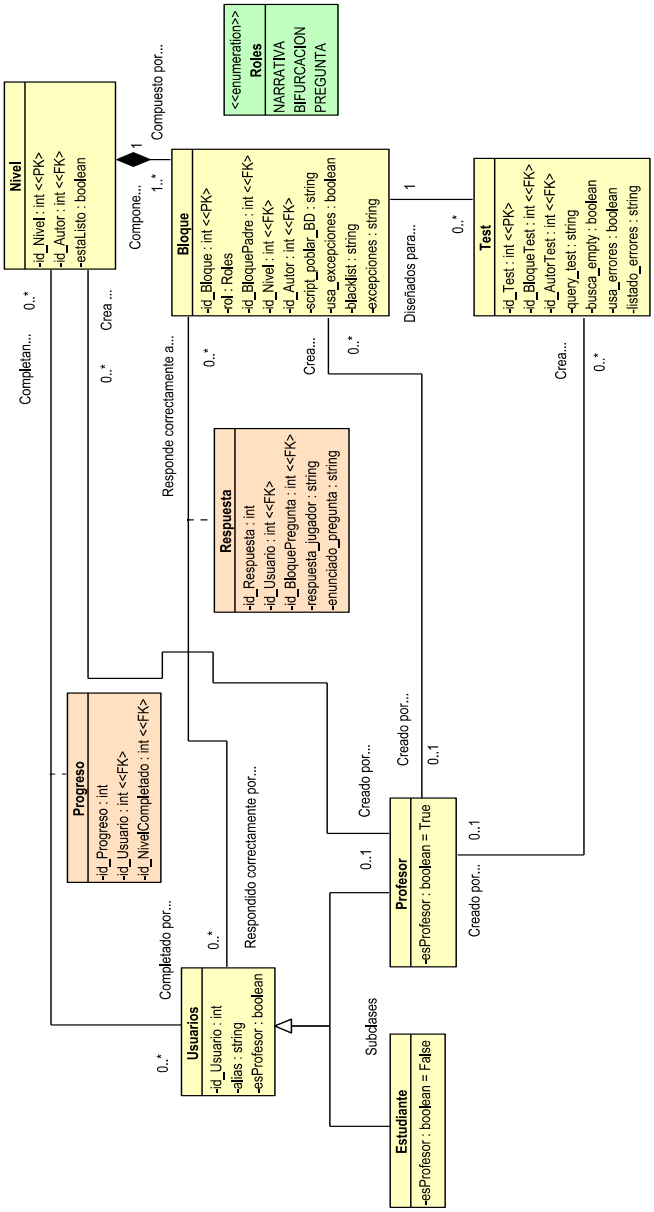

Figura 5.1: Diagrama de clases del modelo de dominio del sistema.

5.2. Modelo de Análisis

El Modelo de Análisis está formado por las **Clases de Análisis** 5.2.1 y la **Realización de los Casos de Uso** 5.2.2.

5.2.1. Clases de Análisis

En este apartado se extiende lo planteado en el Modelo de Dominio 5.1 en la sección previa. Se muestra una vista detallada de las distintas clases y entidades del sistema, describiéndose todos los atributos y responsabilidades de estas:

- **Usuarios**: Representa al usuario genérico del sistema.
  - Atributos: `id_Usuario`, `clave`, `alias`, `esProfesor`, `is_active`.
  - Responsabilidades: Autenticarse en el sistema, registrar progreso y respuestas.
- **Estudiante**: Representa a los usuarios del sistema con rol de Estudiante.
  - Atributos: Las mismas que Usuarios, pero el valor de **esProfesor** debe ser *False*.
  - Responsabilidades: Las responsabilidades del usuario genérico, acceder a los niveles disponibles, jugar a los niveles.
- **Profesor**: Representa a los usuarios del sistema con rol de Profesor.
  - Atributos: Las mismas que Usuarios, pero el valor de **esProfesor** debe ser *True*.
  - Responsabilidades: Las responsabilidades del usuario genérico, acceder a los niveles disponibles, jugar a los niveles, administrar niveles, bloques y tests del sistema.
- **Nivel**
  - Atributos: `id_Nivel`, `id_Autor`, `nombre_nivel`, `descripcion_nivel`, `estaListo`.
  - Responsabilidades: Contener bloques (preguntas, narrativas y bifurcaciones), gestionar la configuración y estado del nivel, establecer si el nivel esta listo para ser jugado.
- **Bloque**
  - Atributos: `id_Bloque`, `nombre_Bloque`, `contexto`, `rol`, `id_BloquePadre`, `id_Nivel`, `id_Autor`, `opcion`, `script_poblar_BD`, `usa_excepciones`, `blacklist`, `excepciones`, `genera_result_set`, `script_obtener_results`.
  - Responsabilidades: Constituir la unidad de construcción de los niveles (preguntas, narrativas, bifurcaciones), gestionar la lógica de navegación entre los bloques que componen un nivel, contener y proveer el datos necesarios para la evaluación, gestionar la configuración del bloque.
- **Test**
  - Atributos: `id_Test`, `nombre_Test`, `id_BloqueTest`, `id_AutorTest`, `explicacion`, `query_test`, `busca_empty`, `usa_errores`, `listado_errores`, `error_feedback`.
  - Responsabilidades: Definir y constituir los criterios de evaluación de las respuestas a cada pregunta, ejecutar y validar los tests sobre la respuesta del usuario, proporcionar *feedback* específico en función al test que no haya sido superado.
- **Respuesta**

  - Atributos: `id_Respuesta`, `id_Usuario`, `id_BloquePregunta`, `respuesta_raw`, `respuesta_n enunciado`, `fecha_hora`.
  - Responsabilidades: Registrar la respuesta correcta de un usuario a una pregunta, almacenar la cadena tal y como ha sido escrita por el jugador, almacenar la versión normalizada (estandarizada) de la respuesta, asociar la respuesta al usuario autor de la respuesta y al bloque cuya pregunta responde.

- **Progreso**
  - Atributos: `id_Progreso`, `fecha`, `id_Usuario`, `id_NivelCompletado`.
  - Responsabilidades: Asociar los niveles superados con el usuario que los ha superado, registrar la fecha de la primera vez que un usuario supera un nivel específico.

- **Evaluador**: Clase Controlador que representa al conjunto de operaciones, interacciones y llamadas que constituyen el servicio de evaluación de respuestas.
  - Atributos: Al representar a un conjunto de operaciones no tiene atributos persistentes; usa *helpers* y la lógica de evaluación. Adicionalmente, para operar necesita disponer de `id_Usuario` del jugador que solicita la evaluación de su respuesta, el `id_Bloque` del bloque correspondiente a la pregunta y la `respuesta` que se debe evaluar.
  - Responsabilidades: Orquestar el proceso de evaluación de respuestas, orquestar la validación de las respuestas, gestionar la retroalimentación de los resultados de la evaluación.

5.2.2. Realización de Casos de Uso de Análisis

En esta sección se muestra la realización de los tres casos más relevantes del proyecto. Esto se lleva a cabo mediante tres Diagramas de Secuencia ya que permiten mostrar el flujo de eventos e interacciones que constituyen cada uno de los tres escenarios escogidos. No obstante, antes de continuar, es importante mencionar que estos tres casos **no** son definidos como tres casos de uso individuales en el Modelo de Casos de Uso 4.4 y, por lo tanto, tampoco coinciden con tres Especificaciones de Casos de Uso 4.5.

La razón de esto es que el contexto, nivel de detalle y matices que se busca mostrar con el Modelo de Casos de Uso y las Especificaciones de Casos de Uso, no son los mismos que en el caso de la Realización de Casos de Uso de Análisis. Los casos de uso del Modelo de Casos de Uso se crean directamente en base a los requisistos del sistema, por ello las secciones correspondientes al Modelo de Casos de Uso 4.4 y a la Especificación de Casos de Uso 4.5 pertenecen al capítulo correspondiente a la Especificación de Requisitos. A ese nivel, los casos de uso del sistema se establecen únicamente teniendo en cuenta los requisitos del sistema. Sin embargo, el capítulo de Análisis corresponde a una etapa más avanzada del desarrollo del proyecto, por ello al establecer los tres casos de uso más importantes a **nivel** de Análisis, los requisitos del sistema dejan de ser el único factor a tener en cuenta y factores como las relaciones entre los elementos del sistema y una vista más detallada de cúal es el valor que el sistema proporciona a los actores de este, ganan importancia.

Por esos motivos, en este proyecto, en la Realización de Casos de Uso de Análisis se representan **dos escenarios** que corresponden a los servicios principales del proyecto: "Crear niveles para jugar" y "Jugar niveles". A nivel de requisitos del sistema, estos dos servicios corresponden a **cuatro** casos de uso: **Jugar un nivel** 4.4, **Administrar nivel** 4.5, **Administrar bloque** 4.6 y **Administrar test** 4.7, siendo estos tres últimos los necesarios para crear un nivel que pueda ser jugado.

Como ha sido mencionado, a nivel de Análisis, se tienen en cuenta no solo los requisitos del sistema, sino también qué valor ven los actores en el sistema y las relaciones entre los elementos de este, sin embargo, esto no significa que los requisitos del sistema y los objetivos del proyecto dejen de ser importantes, lo que quiere decir que, aunque a nivel de un Actor del sistema, el valor innato del sistema proviene de los servicios para crear y jugar niveles, estos dos servicios no representan plenamente el objetivo principal de este TFG que es **gamificar la enseñanza de SQL**, sin embargo, se puede plantear el escenario en el que un usuario juega a un nivel hasta completarlo como **dos** casos de uso: **Transcurso del nivel** 5.2.3, que representa al conjunto de acciones mediante las cuales el Actor avanza a lo largo de un nivel hasta alcanzar el final, y **Evaluar respuesta** 5.2.4, que representa el conjunto de acciones mediante las cuales el sistema, cada vez que un usuario alcanza una pregunta durante la partida, evalúa la respuesta, declara si es correcta o no, y proporciona el resultado al usuario.

En resumen, se van a presentar tres Diagramas de Secuencia mediante los que se desarrollan dos escenarios ficticios de uso del sistema por los actores del mismo: un usuario juega a un nivel hasta completarlo y un Profesor crea y prepara un nivel al completo para que pueda ser jugado por los usuarios del sistema. Estos dos escenarios corresponden a tres casos de uso a nivel de Análisis: **Transcurso del nivel** y **Evaluar respuesta** que corresponden a jugar un nivel hasta completarlo, y **Preparar un nivel jugable** que corresponde a crear y preparar un nivel al completo. Por último, la correspondencia entre los casos de uso de nivel de Análisis y los casos de uso del nivel de Especificación de Requisitos es la siguiente: los casos de análisis **Transcurso del nivel** 5.2.3 y **Evaluar respuesta** 5.2.4 corresponden, ambos, al servicio especificado en la Tabla 4.4 correspondiente al caso de uso **Jugar un nivel** del Modelo de Casos de Uso, mientras que el caso de análisis **Preparar un nivel jugable** 5.2.5 corresponde a los casos de uso **Administrar nivel**, **Administrar bloque** y **Administrar test** cuyas especificaciones son las tablas 4.5, 4.6 y 4.7 respectivamente.

5.2.3. Secuencia: Transcurso del Nivel - CU.Jugar un nivel

Este es el primero de los dos diagramas de secuencia que representan el escenario de un usuario jugando a un nivel hasta completarlo. Este diagrama en concreto representa el movimiento entre los distintos bloques que componen el nivel y la lógica asociada a cada bloque, pero sin reflejar el proceso mediante el cual el sistema evalúa la respuesta del jugador para determinar si esta es correcta o incorrecta. La especificación del caso de uso correspondiente es la Tabla 4.4.

En favor de su legibilidad, el diagrama está dividido en dos figuras: Figura 5.2 y Figura 5.3.

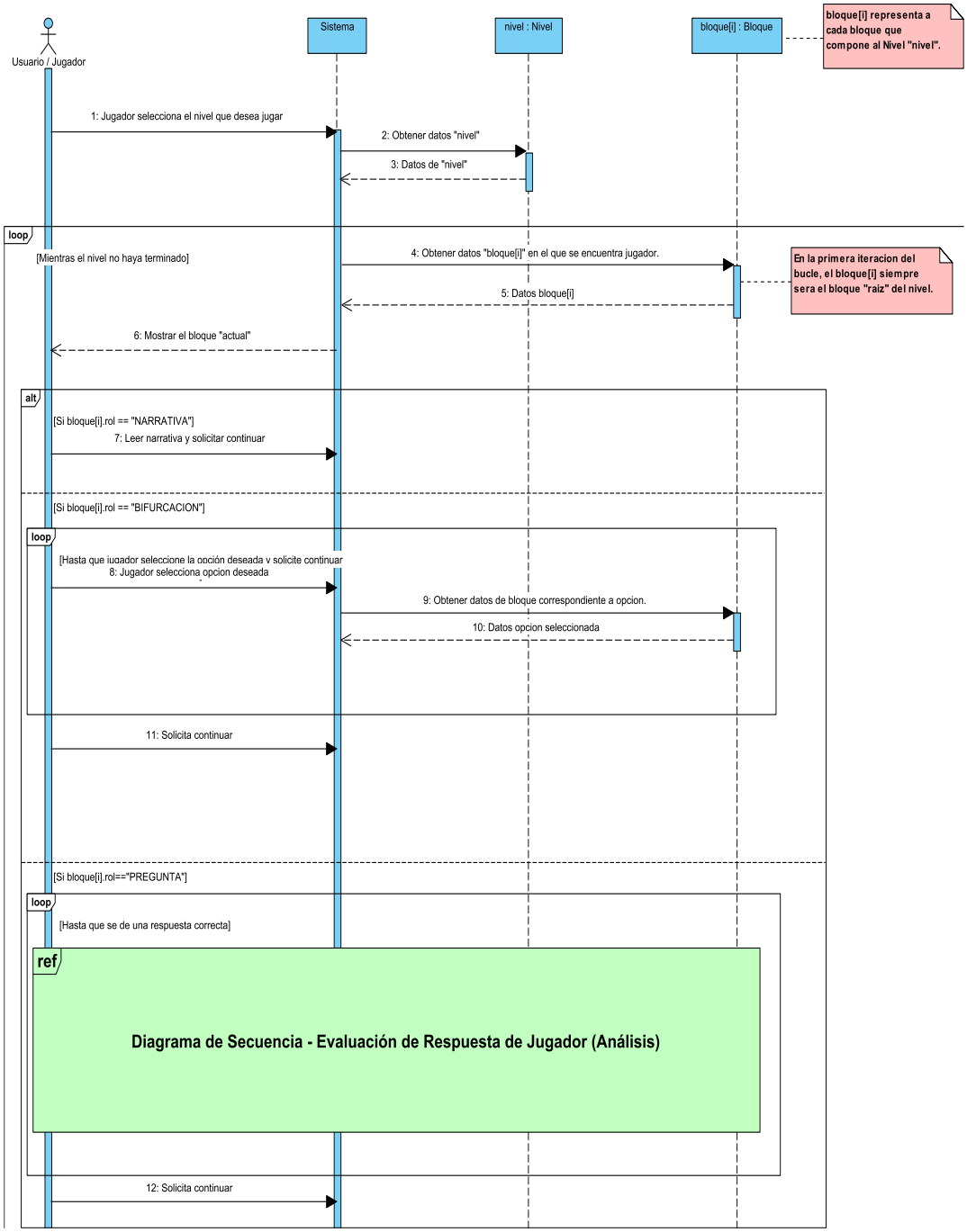

Figura 5.2: Diagrama de secuencia reflejando la primera parte del escenario de un usuario jugando un nivel hasta completarlo.

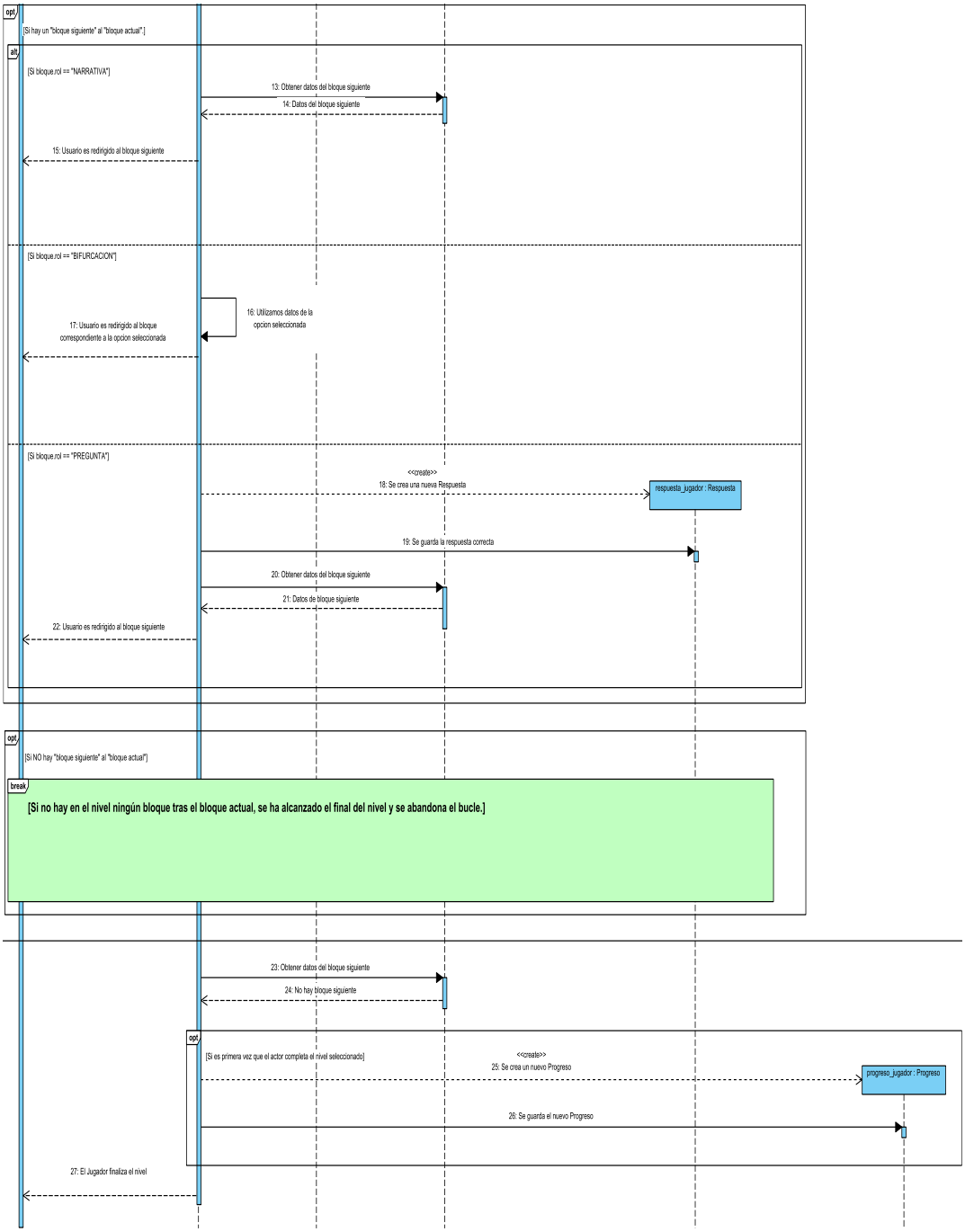

Figura 5.3: Diagrama de secuencia reflejando la segunda parte del escenario de un usuario jugando un nivel hasta completarlo.

5.2.4. Secuencia: Evaluar respuesta - CU.Jugar un nivel

El segundo de los diagramas de secuencia. Refleja el proceso mediante el cual el sistema, al recibir una solicitud de evaluación, lleva a cabo el proceso mediante el cual determina si la respuesta proporcionada por el jugador responde correctamente o no la pregunta en cuestión. La especificación del caso de uso correspondiente es la Tabla 4.4.

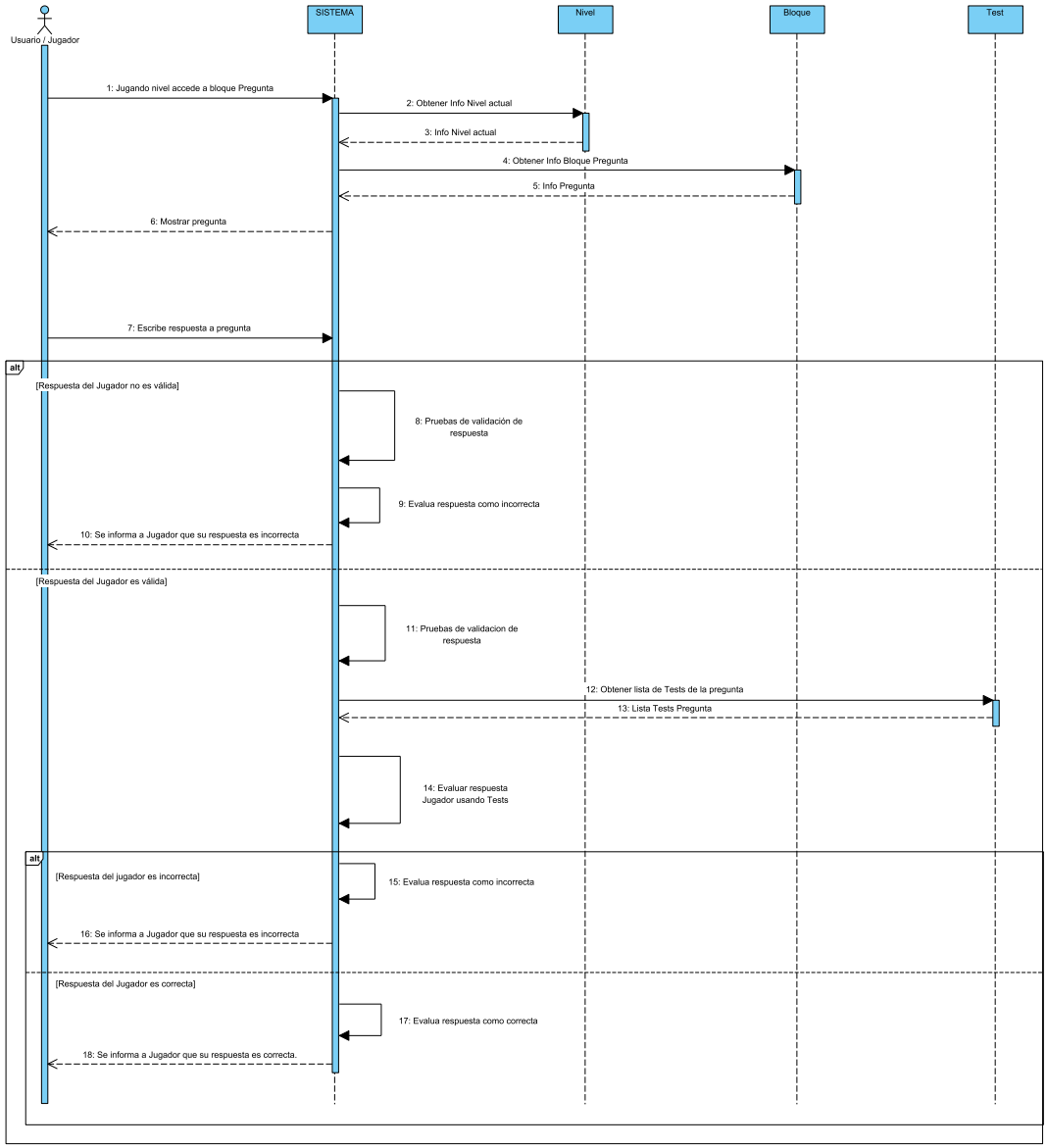

Figura 5.4: Diagrama de secuencia reflejando el proceso mediante el cual el sistema evalúa una respuesta de jugador.

5.2.5. Secuencia: Preparar un nivel jugable - CU.Administrar Nivel, Administrar Bloque, Administrar Test

Refleja el escenario de un Profesor utilizando el sistema para crear un nivel y hacerlo completamente jugable. Creando y configurando los bloques que componen el nivel y, en el caso de los bloques pregunta, los tests de estos.

Tal y como ha sido explicado en la Sección 5.2.2, el caso, a nivel de Análisis, representado en esta secuencia, está compuesto por los casos de uso (a nivel de Especificación de Requisitos): **Administrar nivel** 4.5, **Administrar bloque** 4.6 y **Administrar test** 4.7.

En favor su legibilidad, el Diagrama de Secuencia esta dividido en dos figuras: Figura 5.5 y Figura 5.6

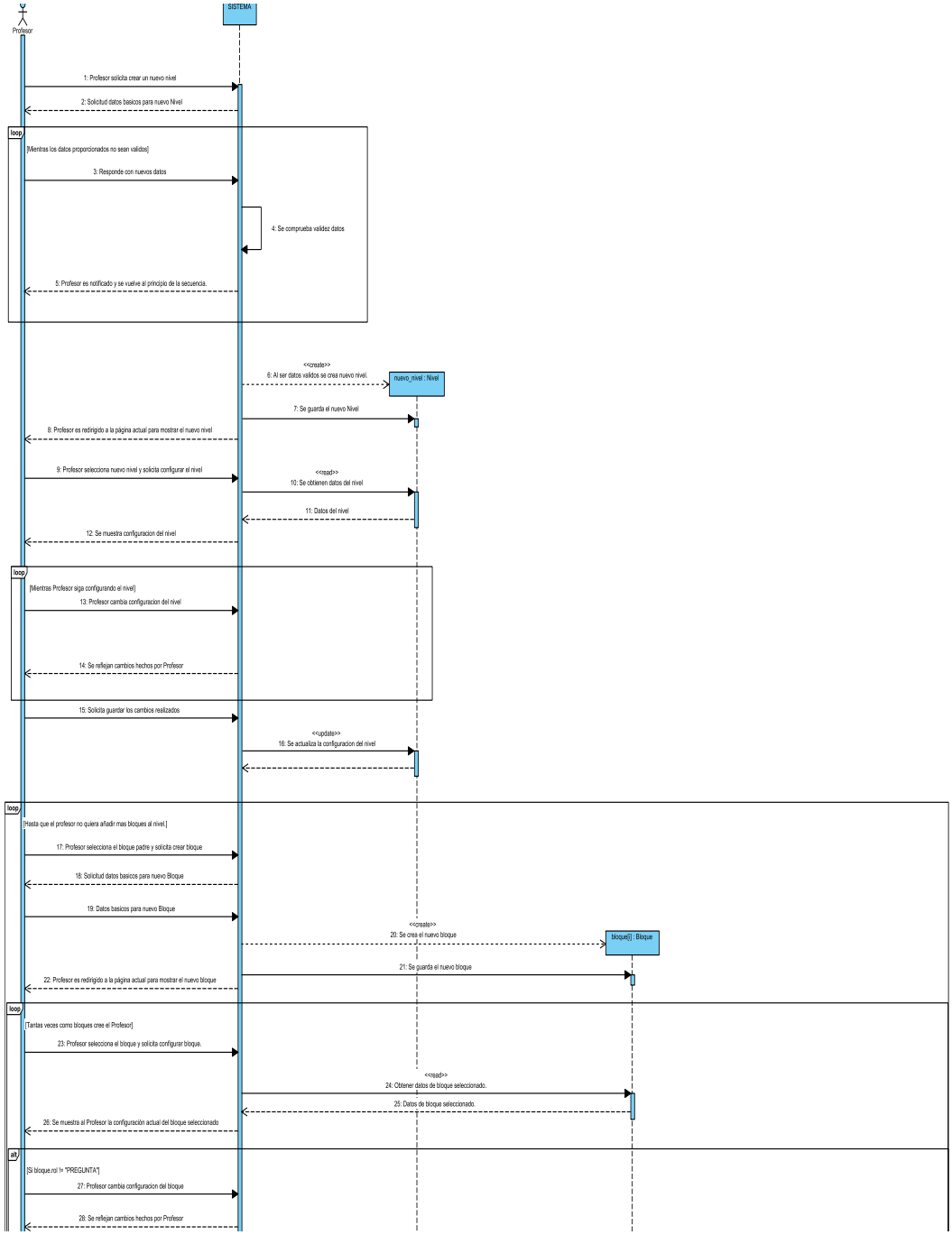

Figura 5.5: Diagrama de secuencia reflejando la primera parte del proceso mediante el cual un Profesor crea y prepara un nivel para que esté listo para ser jugado.

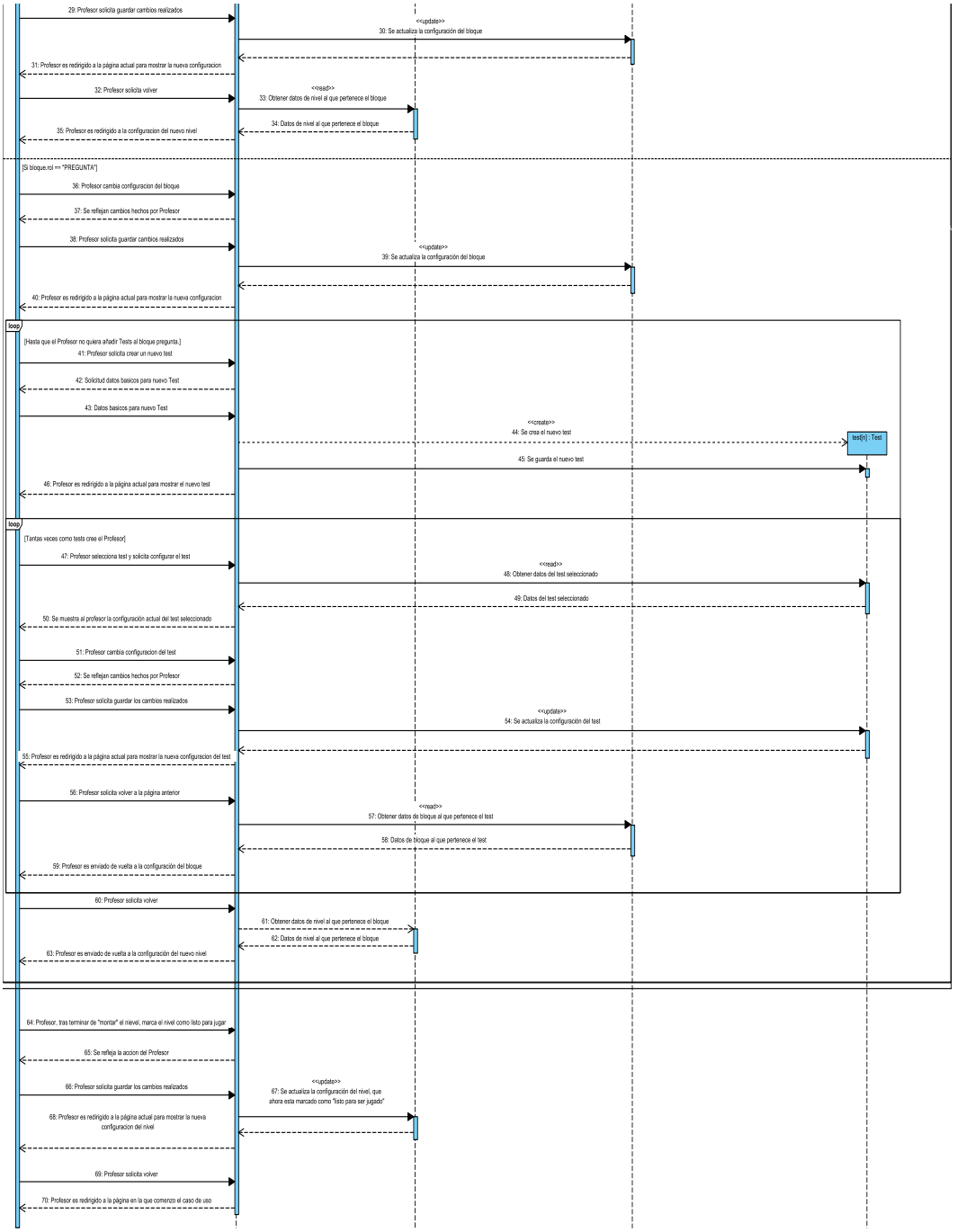

Figura 5.6: Diagrama de secuencia reflejando la segunda parte del proceso mediante el cual un Profesor crea y prepara un nivel para que esté listo para ser jugado.

Capítulo 6

Diseño

Este capítulo está dedicado al *workflow* de Diseño del sistema. A lo largo del capítulo se presentan las arquitecturas lógica y física del sistema, mediante un Diagrama de Paquetes y un Diagrama de Despliegue respectivamente. Acto seguido, se proporciona un listado de los principales patrones de diseño utilizados en el proyecto y, finalmente, la realización, mediante un Diagrama de Secuencia UML, del caso de uso más relevante del sistema.

6.1. Arquitectura lógica del sistema

El patrón arquitectónico seleccionado para la arquitectura **lógica** del sistema es el patrón **Capas**. Este patrón consiste en organizar los componentes del sistema (o aplicación) en una serie de niveles, o **capas**, lógicos y físicos conectados entre si, repartiendose los componentes del sistema entre las capas de forma que cada capa tenga un rol y una responsabilidad específica dentro del sistema [72].

Cada capa representa un nivel distinto de abstracción y están ordenadas jerárquicamente de forma que: las capas superiores dependen de los servicios que las capas inferiores proporcionan y cada capa, al representar su propio nivel de abstracción, el funcionamiento de las capas que la rodean es extremadamante reducido. En otras palabras, al representar cada capa su propio grado o nivel de abstracción, cada capa sabe únicamente lo mínimo necesario sobre la capa de la que depende (la capa que tiene por debajo) y no sabe nada sobre las capas a las que da servicio (las que tiene por encima).

Como el punto anterior puede resultar complejo, se va a plantear un ejemplo: En un escenario en el que un sistema se organiza en tres capas, Alfa, Beta y Gamma, ordenadas en ese mismo orden de forma que Alfa es la más alta y Gamma la más baja. En este ejemplo, Alfa no da servicio a ninguna otra capa y depende de los servicios de Beta, Beta da servicio a Alfa y depende de los servicios de Gamma, y Gamma da servicio a Beta y no depende de ninguna otra capa. El conocimiento que tiene cada capa sobre el resto de capas es el mínimo

necesario para funcionar, esto quiere decir que capas como Alfa y Beta, que dependen de servicios proporcionados por otras capas, conocen únicamente lo necesario para poder recibir ese servicio del que dependen, mientras que capas como Gamma que **no dependen** de ninguna otra capa no tienen conocimiento alguno sobre el resto de capas pues para ofrecer servicio no es necesario saber esos detalles, solo hacen falta para recibir un servicio.

Esta arquitectura busca distribuir (separar) las responsabilidades del sistema[1] de forma que estas no se acumulen en una única capa, mejorando la modularidad del sistema y permitiendo que se puedan hacer cambios en una capa sin que estos tengan efectos en las capas que la rodean. Las ventajas [11] de esta arquitectura **no** son únicamente el favorecer la modularidad del sistema y minimizar las dependencias y el acoplamiento.

Se ha explicado que el concepto básico de esta arquitectura es organizar los componentes del sistema en distintas capas, cada una con un rol y responsabilidad en concreto, esto significa que los componentes del sistema con responsabilidades relacionadas y/o similares se reparten entre las capas en función de la responsabilidad de cada capa, de esa forma, una capa cuya responsabilidad es "la gestión de los clientes" estará constituida por todos los componentes del sistema que están involucrados en conseguir que se puedan gestionar los clientes del sistema. Organizar el sistema de esa forma facilita el desarrollo del proyecto ya que, en cierta forma, divide las responsabilidades del sistema en unidades más pequeñas y comprensibles. Esto junto con la mejora a la modularidad significa que esta arquitectura favorece también la escalabilidad y facilita el mantenimiento de la aplicación. Por último y no menos importante, cada capa es, en realidad, una unidad funcional del sistema, compuesta por todos los componentes necesarios para una proporcionar correctamente una función específica, esto significa que, dependiendo de la cantidad de capas y como de específicas sean las responsabilidades que se asignan a cada capa, la estructura de capas fomenta la reusabilidad. El ejemplo de la capa responsable de la gestión de los clientes demuestra también esta ventaja, una capa con esas cualidades puede ser reutilizada, si las condiciones son adecuadas, en una aplicación web, en una aplicación móvil, etc.

Antes de continuar conviene mencionar los siguientes detalles:

- La arquitectura o patrón de capas puede organizar el sistema en capas lógicas y físicas, pero no significa que siempre se haga en ambos niveles, de hecho, en el caso de este proyecto, la arquitectura de capas solo es lógica.

- El patrón capas (arquitectura de capas) es considerado uno de los patrones de arquitectura software más populares de la actualidad [46]. Es tal su popularidad que hay divisiones de capas conocidas en el entorno de la informática a nivel global, siendo las más típicas las variaciones de tres y cuatro capas en las que dependiendo del caso el sistema se organiza en: **Capa de Presentación**, **Capa de Servicios**, **Capa de Dominio** y **Capa de Acceso a Datos**. Dependiendo del sistema hay veces que se establecen las cuatro capas y otras en las que se establecen solo tres, descartando una de las capas de los extremos (Presentación o Acceso a Datos) o combinando las capas contiguas entre si, a veces incluso recibiendo esta capa combinada un nombre distinto.

[1] El conjunto de todos los servicios y funciones que se esperan del sistema, es decir, todo aquello que el sistema debe ser capaz de hacer y proveer.

Estas estructuras son muy populares por ser aplicables a una extensa cantidad de sistemas, sin embargo, tal es su popularidad que a la hora de buscar información online, existen fuentes que presentan las capas antes mencionadas como intrínsecas al patrón capas, y ese **no** es el caso. El patrón capas, como se menciona anteriormente en esta sección, consiste sencillamente en la organización de los componentes del sistema en capas, agrupando en cada capa los componentes en función de su responsabilidad y rol, el patrón no declara que responsabilidades deben asignarse a cada capa y tampoco el número de capas que deben establecerse.

- Dependiendo no solo de las necesidades y características de cada sistema o proyecto, sino también de lo que se busque mejorar aplicando el patrón y de lo que los responsables del proyecto consideren oportuno, el grado de detalle de la responsabilidad de cada capa [2], la cantidad y el orden de capas, todos son aspectos que pueden variar de un sistema a otro.
- Se explica de manera más detallada al final de esta sección pero en beneficioso saber que aunque el nombre de las capas en las que se ha organizado este sistema son **Presentación**, **Servicios** y **Dominio+Acceso a Datos** y es también común en sistemas con arquitectura de capas combinar las capas de Dominio y Acceso a Datos, en este caso la última capa no ha sido dada ese nombre por el motivo usual.

Habiéndose explicado el patrón de arquitectura de capas, sus matices y sus beneficios, se procede a exponer la arquitectura lógica del sistema, el cual se divide en:

- Un **Front-End** formado por las plantillas de la carpeta *templates* del proyecto. Cada una de ellas se ha diseñado teniendo en cuenta las peculiaridades de Flask.
- Un **Back-End** basado en **Flask**, como micro-framework de desarrollo de páginas web, y, como lenguaje de programación, **Python**.

El **Front-End** está compuesto en su totalidad por las plantillas de la aplicación, como se ha mencionado antes, están todas juntas en su propio módulo y como se puede apreciar en la Figura 6.1, constituyen la **Capa de Presentación** del sistema.

Si se presta algo más de atención a dicha figura, se puede observar que, como tal, en el diagrama no se las denomina plantillas (*template* en Inglés) sino *Template-view* seguido del nombre de la vista. Esto es porque, aunque a nivel físico, los componentes donde se definen las interfaces del sistema son las plantillas, el diagrama de la figura en cuestión es el *Diagrama de Paquetes*[3] a nivel lógico y, desde el punto de vista de la arquitectura **Lógica**

[2] Se trata de algo muy detallado o la responsabilidad es más general, por ejemplo, establecer como responsabilidad de una capa "Gestionar operaciones de cálculo en el sistema" frente a "Gestionar operaciones de división en el sistema".

[3] Como aclaración, **Front-End** y **Capa de Presentación** son conceptos distintos, sin embargo, en la práctica, en gran parte de arquitecturas web modernas, se refieren, ambas, a la capa de interfaz de usuario, solo que la primera viene de una división del sistema entre **lo que el usuario puede ver y con lo que puede interactuar** (*Front-End*) y lo que no está a la vista del usuario y por ello no puede interactuar con ello (*Back-End*), mientras que el patrón Capas busca separar el sistema en múltiples capas en base a su responsabilidad, separa, a nivel de diseño lógico, las partes del sistema en grupos dependiendo de que responsabilidad tengan, siendo usualmente la capa más externa la de Presentación.

del sistema, los componentes de la capa de Presentación no son las distintas plantillas, pues estas son la definición de la interfaz, sino que la capa de Presentación son las *Template-View* correspondientes a cada una de las plantillas **físicas**, ya que, a nivel lógico, mientras que la plantilla es la definición de la interfaz, la *Template-View* representa a la propia interfaz definida por la plantilla y, por ello, constituye la unidad básica de la **Capa de Presentación**. **Template-View** es un término que se ha asignado por el patrón de diseño del mismo nombre, y del que, como patrón, se va a hablar más en detalle en la sección 6.3.3 de este mismo capítulo.

El **Back-End** del sistema, hablando a nivel de diseño lógico, engloba a las Capas de Servicios, Dominio y Acceso a Datos, es decir, todas las capas menos la de Presentación. Como se observa en la Figura 6.1, las capas de Dominio y Acceso a Datos están juntas en una misma capa, esto se debe a que el sistema utiliza **SQLAlchemy** 7.5.3 para simplificar las interacciones entre la aplicación y su base de datos.

SQLAlchemy es un *ORM* (Object-Relational Mapper) y por ello el patrón **Data Mapper** es intrínseco a su uso. Se explica dicho patrón más en detalle en la sección 6.3.4 de este capítulo, pero de momento basta con saber que, en el sistema, SQLAlchemy se encarga de traducir los datos de la aplicación al formato de la base de datos y viceversa, esto es representado por los *Data Mapper* que se aprecian en el Diagrama de Paquetes, dentro de las capas de Dominio y Acceso a Datos. Esa necesidad de traducción resulta en un flujo constante de datos entre la aplicación y su BD que es el motivo por el que en el diagrama de Capas del sistema, las capas de Dominio y Acceso a Datos están combinadas en una única capa[4] debido a que en este sistema el *acoplamiento* [2] entre dichas capas es demasiado como para que sea apropiado presentarlas como capas separadas.

[4]Normalmente, cada capa es su propia entidad y suelen presentar entre ellas una dependencia unidireccional en la que gracias al bajo grado de acoplamiento, se puedan separar en capas que únicamente tienen una dependencia débil con la capa que tienen inmediatamente debajo. Sin embargo, en nuestro caso, Data Mapper resulta en que las capas de Dominio y Acceso a Datos dependan en mayor medida la una de la otra, de ahí que se hayan combinado en la misma capa.

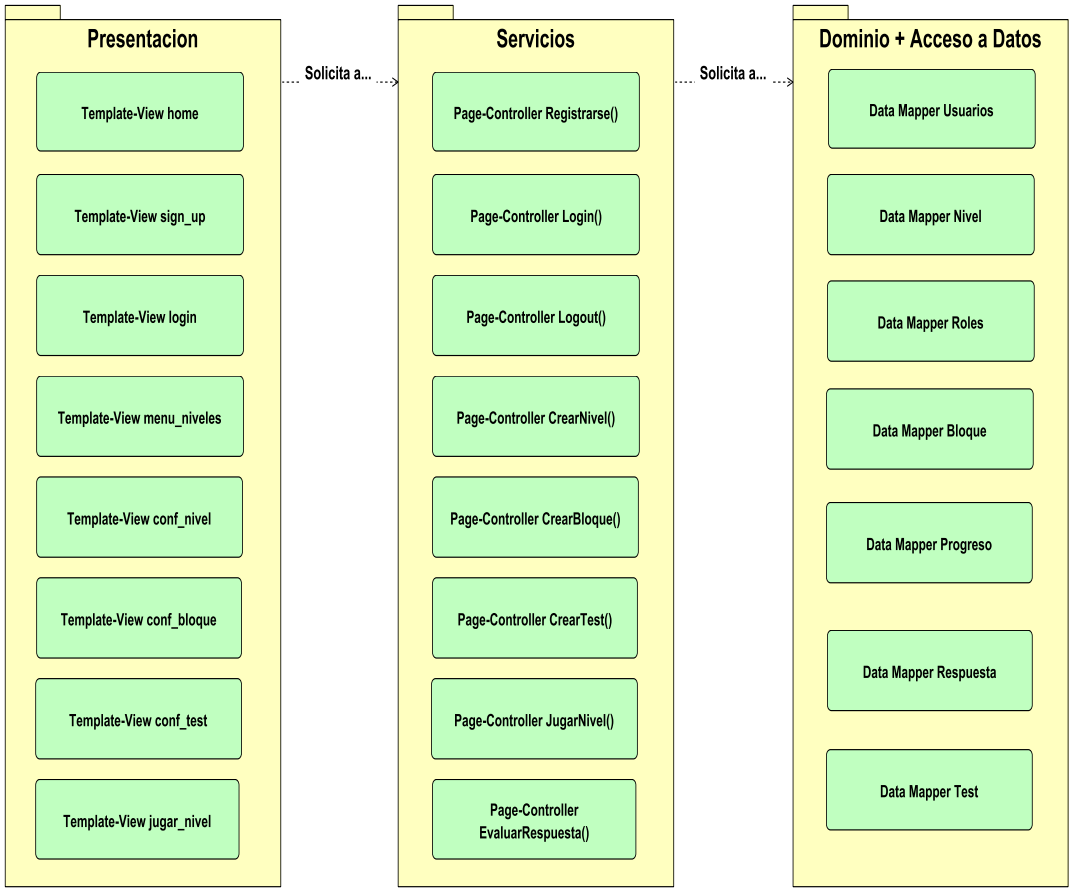

Figura 6.1: Diagrama de Paquetes que representa la organización de capas del sistema, reflejando la arquitectura lógica del mismo.

Aunque ya se ha referenciado previamente, el diagrama de la Figura 6.1 es el Diagrama de Paquetes del sistema, y divide a la aplicación en **tres capas**:

1. **Presentación**: Tradicionalmente, es la capa en la que se sitúan los elementos responsables de la Interfaz de Usuario (IU o *UI*, en inglés) o en otras palabras, donde el usuario interactúa con la aplicación [3]. Cabe destacar que, dependiendo de como se implemente, en aplicaciones **Flask** ambos ***Template View*** 6.3.3 y ***Page Controller*** 6.3.2 son asignados a la capa de **Presentación** si las **Rutas** (*endpoints* o funciones-vista) Flask son utilizadas **exclusivamente** como ***controladores*** [1] y se delega la lógica de la aplicación (las funcionalidades que proporciona el sistema) a funciones de apoyo, actuando la ruta meramente como gestor de las plantillas y, puntualmente, se encarga de orquestar la lógica de la aplicación.

   En el caso de este sistema se ha sido más directo y la mayoría[5] de rutas tienen ambas

[5]Salvo la lógica de evaluación, en este caso la ruta sí que actúa como orquestador del proceso, ya que la

responsabilidades, gestionar las interfaces que se renderizan y también llevar a cabo la lógica de aplicación. Esto quiere decir que en este sistema el acoplamiento entre las capas de Presentación y Servicios es superior al caso habitual, ya que la capa de Servicios no solo actúa como controlador, sino que es también responsable de ciertos servicios del sistema que forman parte de la lógica de negocio, resultando en que, para ciertos escenarios, la lógica de negocio donde se define como debe reaccionar el sistema a las acciones del usuario venga directamente no de la Capa de Dominio sino de la Capa de Servicio. Esto significa que la dependencia y frecuencia de interacción entre la Capa de Presentación y la Capa de Servicios de este proyecto es más fuerte que lo habitual, pero no lo suficientemente fuerte como para considerar adecuado juntar ambas capas en una misma[6].

2. **Servicios**: Tradicionalmente, esta capa orquesta los distintos casos de uso de la aplicación: recibe las peticiones de la capa de Presentación, aplica reglas transversales como autenticación, autorización o transacciones, y delega la ejecución de la lógica de negocio al corazón del sistema. O, en otras palabras, se encarga de que la aplicación proporcione los servicios que debe proporcionar. Como se ha mencionado en el punto anterior, usualmente el diagrama de Capas no tendría *Page Controller* (representando a las rutas) en esta capa, pero en el caso de este proyecto, por el uso que se ha dado a Flask, las rutas se encargan, además de su función usual de controlador, también de la lógica de aplicación (negocio), por lo que, para reflejar de la forma más realista posible la arquitectura lógica del sistema, los *Page Controller* se han colocado en la capa de Servicios.

3. **Dominio + Acceso a Datos**: Finalmente, como se ha mencionado previamente, debido a la presencia e influencia de SQLAlchemy en este sistema, se ha considerado necesario combinar las capas de **Dominio** y **Acceso a Datos**. En concreto, la causa de esto es que, cuando se usa Flask-SQLAlchemy 7.5.4, SQLAlchemy utiliza por defecto el modelo declarativo, lo que significa que las clases definidas en los modelos del sistema definen tanto el modelo Python como el esquema de la tabla MySQL correspondiente, es decir, no solo definen los objetos de negocio sino que también son lo que permite a SQLAlchemy *mapear* (enlazar) los objetos Python con las correspondientes filas de las tablas MySQL. El Listing 6.1 muestra un **ejemplo** de cómo se define una clase cualquiera mediante el modelo declarativo de SQLAlchemy. El parámetro `Base` proporcionado a la definición de la clase indica a SQLAlchemy que dicha clase Usuario es una clase del dominio del sistema, y que no es tan solo una clase Python sino que también debe haber una Tabla correspondiente en la base de datos. En el ejemplo, se define un modelo llamado Usuario que tiene dos atributos, `id` y `nombre`, y gracias a la sintaxis utilizada en la definición de estos atributo, se consigue que la clase Usuario presente un formato que Python puede interpretar a la vez que se establece el esquema de la tabla correspondiente. Como se ve en el ejemplo, la línea `id = Column(Integer,`

cantidad de acciones, operaciones y en general la lógica necesaria para evaluar una respuesta es demasiado compleja y extensa como para que sea apropiado que la ruta se encargue plenamente de ello. El resto de funcionalidades del sistema son llevadas a cabo principalmente por su correspondiente ruta.

[6]El caso de las capas de Dominio y Acceso a Datos es diferente, ya que la presencia de SQLAlchemy resulta en que a nivel conceptual no tenga sentido diferenciar ambas capas ya que los componentes de información del sistema (Clases y Modelos) fluctuan constantemente entre un formato de datos compatible con Python y otro compatible con MySQL, perteneciendo los primeros a la capa de Dominio y los segundos a la de Acceso a Datos.

`primary_key=True)` utiliza `Column` con lo que no solo establece que el atributo `id` del modelo Usuario en python (es importante no olvidad que esta definición tiene lugar dentro de la aplicación Python). sino que también define la **columna** correspondiente en la tabla. En resumen, debido a SQLAlchemy, cuando se define un modelo en la aplicación, se está definiendo también la tabla correspondiente en MySQL. Esta función doble resulta en que, las clases Python tengan dos roles: definir los objetos de negocio del sistema o en otras palabras, definir la estructura lógica de datos del sistema, que es responsabilidad de la capa de Dominio, y permitir a SQLAlchemy *mappear* (enlazar) el modelo Python con la tabla MySQL correspondiente, que es responsabilidad de la capa de Acceso a Datos. Esto significa que, debido a SQLAlchemy, las clases y modelos, que tradicionalmente son componentes de la capa de Dominio, en este escenario forman también parte de la capa de Acceso a Datos, motivo por el cúal se ha considerado adecuado combinar ambas capas en una. Esto se refleja en la Figura 6.1 mediante las entidades *Data Mapper* del interior de la capa combinada, habiendo una entidad por cada par Clase (Python) y Tabla (MySQL) del sistema.

```
class Usuario(Base):
    __tablename__ = 'usuarios'
    id      = Column(Integer, primary_key=True)
    nombre = Column(String(50), nullable=False)
```

Listing 6.1: *Snippet* con un ejemplo de definicion de modelo declarativo.

Antes de terminar la sección, se considera necesario aclarar un último aspecto de la última capa descrita. Ha sido mencionado varias veces que, en este proyecto, la mayor parte de la lógica de negocio esta definida en las rutas Flask. Estas son las responsables de controlar las interfaces del sistema por lo que pertenecen, tradicionalmente, a la capa de servicios. Sin embargo, en este sistema también son responsables de gran parte [7] de la lógica de negocio. Esto puede llevar a que se combinen las capas de Servicios y Dominio, **sin embargo**, como, lo antes mencionado sobre SQLAlchemy, resulta en un alto grado de acoplamiento entre las capas de Dominio y Acceso a Datos, se tomó la decisión de mantener las capas de Servicios y Dominio separadas y combinar las de Dominio y Acceso a Datos. También conviene aclarar que, el motivo por el que la capa mantiene el nombre de **Dominio** es porque, aunque no contiene la lógica de negocio del sistema, al menos no la mayor parte, sigue siendo responsable de la estructura lógica del sistema, y sus componentes **se corresponden** con las clases del dominio del sistema por ello, al seguir siendo responsable de la definición de las clases del dominio del sistema, sigue teniendo el nombre de capa de Dominio.

[7] Esto se debe a que, en gran medida, la lógica de negocio del sistema es relativamente ligera, sin embargo, como se explica con más detalle en el capítulo de Implementación, la lógica de evaluación **no** esta definida en la ruta correspondiente, sino que esta ruta actua como orquestador del proceso, pero la lógica como tal esta definida en los helpers de evaluación. Asi que gran parte de la lógica de negocio del sistema esta en las rutas, pero no toda.

6.2. Arquitectura física del sistema

Con **arquitectura física** se está hablando de la descripción del hardware del sistema, sobre como sus componentes están conectados entre sí y sobre como se despliega el software desarrollado sobre el hardware. La Figura 6.2 es un diagrama de Despliegue (de nivel de Diseño) del sistema. En el diagrama puede apreciarse que el sistema se puede dividir en tres-cuatro componentes generales:

1. **Cliente**: Conectado con la aplicación mediante HTTPS. Como la aplicación está basada en Flask el cliente sencillamente se trata de cualquier solicitud de acceso a la web del servicio.

2. **Aplicación Flask**: Contiene los ficheros y archivos del proyecto, toda la lógica de la aplicación, las clases, formularios, plantillas y rutas. Como se puede ver en el diagrama, a nivel de diseño, se observan tres paquetes principales (bastante similares a las vistas en el Diagrama de Paquetes 6.1) que encapsulan las interfaces, rutas y modelos de la aplicación. A diferencia de en el diagrama de paquetes en el que, al representar la arquitectura **lógica**, el contenido de cada paquete (aunque estos paquetes representaban a las capas lógicas del sistema) referenciaba a patrones y no al componente físico correspondiente, aquí se observa que las plantillas que constituyen la interfaz del sistema son archivos HTML, las rutas siguen siendo funciones Python y los modelos son clases Python con la peculiaridad de que utilizan también la clase declarativa `db.Model` de SQLAlchemy [30] diferenciándolas de una clase Python estándar, salvo la clase Roles, la cual es un Enum asociado a la clase Bloque. Por último, pero no menos importante, la aplicación Flask también contiene las carpetas correspondientes a los dos tipos de base de datos involucrados en el sistema, cada carpeta contiene los scripts que deben ser ejecutados en cada base de datos y la configuración de cada base de datos (su esqueleto).

3. **Bases de datos**: En el sistema se pueden observar **dos** tipos de base de datos, la llamada FWT que es la base de datos de la aplicación es decir, la base de datos en la que se almacenan los datos necesarios por el sistema para funcionar, y la otra es EVAL_BD, que representa a las bases de datos utilizadas meramente como entorno de ejecución para que el sistema pueda evaluar las respuestas de los usuarios a las distintas preguntas de sus niveles[8]. Como se ha mencionado antes y además puede apreciarse en el diagrama, estas bases de datos son desplegadas utilizando el contenido de las carpetas **BD Aplicación** y **BD Evaluación** de la **Aplicación Flask**, que es la que hace uso de dichas bases de datos.

Es importante no confundirse, este diagrama de despliegue es a nivel de diseño, en el capítulo de implementación se encuentra el diagrama de despliegue a nivel de implementación o lo que es lo mismo, físico, del sistema, en él se da una explicación más detallada sobre como se ha implementado el sistema y sus componentes a nivel físico.

[8]Estas preguntas y niveles son parte del contenido que se almacena en la base de datos de la aplicación.

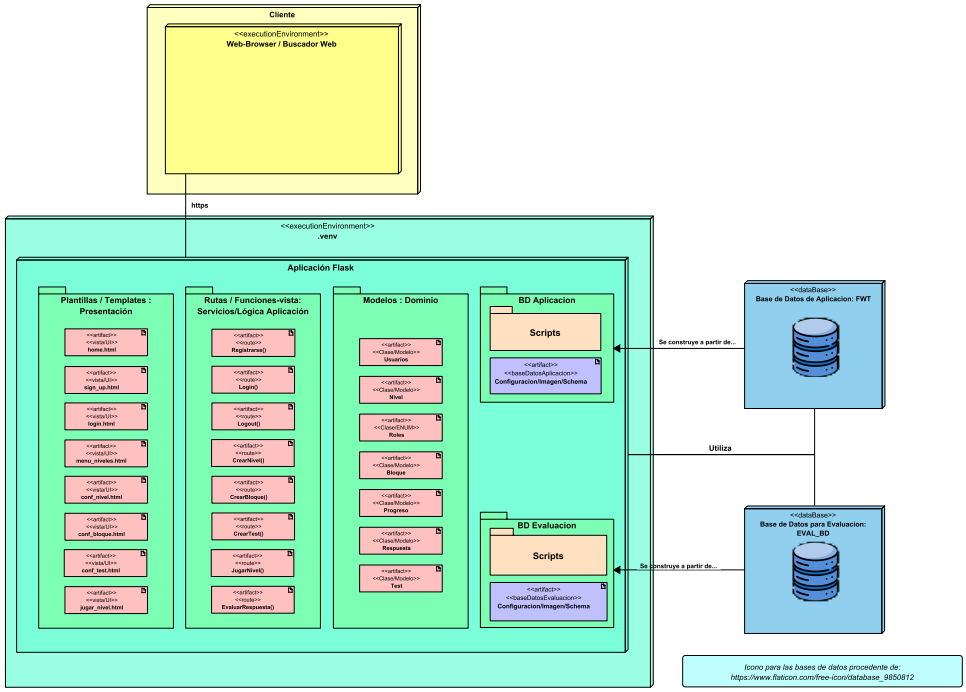

Figura 6.2: Diagrama de despliegue de diseño del sistema.

6.3. Patrones de diseño principales

Los principales patrones de diseño presentes en el sistema son:

- **MVT** (*Model-View-Template*)
- **Page Controller**
- **Template View**
- **Data Mapper**
- **Singleton**
- **Decorator**

6.3.1. *MVT*

El patrón MVT 6.3 es una arquitectura típica de sistemas web donde los **Modelos** (*Models*) gestionan los datos y la lógica de datos, las **Vistas** (**Rutas** o, siendo más concretos, Funciones-Vista) procesan las peticiones del usuario, preparan los datos y gestionan la lógica de negocio y las **Plantillas** (Templates) renderizan la interfaz [19]. Esta separación resulta en un código modular, mantenible y escalable.
Como se ha mencionado previamente, en esta aplicación las rutas también se encargan de parte de la lógica de aplicación, pero no limitandose a orquestar esta como es el caso habitual, sino que las rutas de esta aplicación definen explicitamente parte de la lógica de negocio. Por ello, la **vista** en este sistema está más involucrada aún con la lógica de negocio.
Merece la pena comentar que Flask es un microframework y carece de modelos, pero como en este proyecto se utiliza SQLAlchemy, gracias a los modelos que este proporciona (los antes mencionados modelos declarativos) se puede considerar que se sigue un patrón MVT.

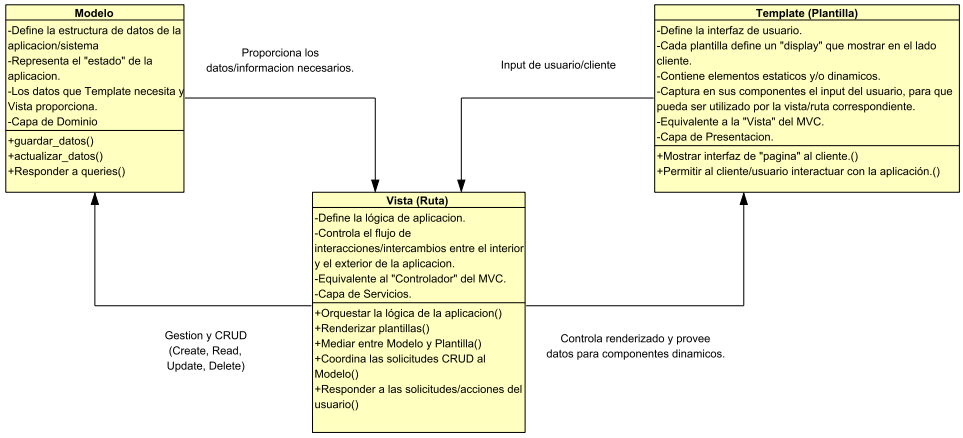

Figura 6.3: Diagrama representando el patrón de diseño Model-View-Template.

6.3.2. *Page Controller*

Este patrón asigna un controlador a cada página o *endpoint*, gestionando todas las peticiones y coordinando la respuesta para esa página [20] . De esta forma, se centraliza la lógica de cada página, haciendo que coordinar el flujo a través de las páginas de la web sea más sencillo y haciendo más cómodo el mantenimiento del código.
Cada vez que un cliente alcanza una página de la aplicación, se invoca al método (la ruta) asignado a dicha URL, esta atiende la petición HTTP, determinando si es necesario hacer algún cambio a los modelos (datos) o solicitar información a los mismos en función de que vista se quiera mostrar.
Este patrón da muy buenos resultados en conjunto con el patrón **View Template**, y es común a la hora de usarlos en conjunto el delegar la lógica de aplicación a un *helper* asociado al Page Controller (la definición de Page Controller es la misma que la de ruta) es decir, a la ruta. En este sistema no se ha hecho esta delegación, por eso se ha mencionado antes que, en este proyecto, las rutas definen explícitamente parte de la lógica de negocio.

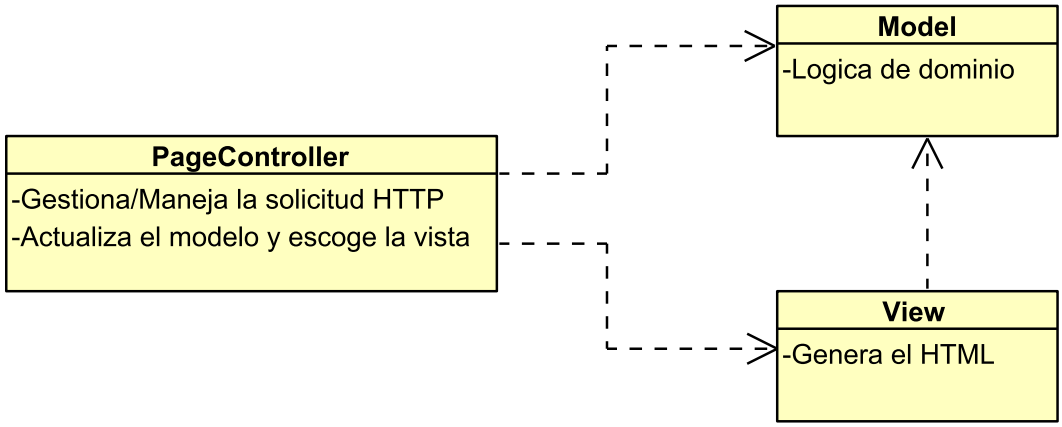

Figura 6.4: Diagrama representando el patrón de diseño *Page Controller*.

6.3.3. *Template View*

Template View es un patrón que se basa en introducir elementos dinámicos en HTML estático [70], más concretamente en insertar en la definición de las plantillas HTML *Placeholders* que en Español viene a significar un objeto que está ahí exclusivamente para guardar el sitio, en otras palabras, se introduce en la definición de la plantilla HTML un objeto cuya única función es la de ser reemplazado durante la ejecución por los elementos dinámicos. Esto es posible gracias a Jinja (motor de plantillas por defecto de Flask) que permite reemplazar a la hora de renderizar la plantilla estos *Placeholders* por el dato, obtenido dinámicamente, correspondiente, esto pueden ser valores de atributos de clases del sistema, resultados de funciones, etc.
Volviendo a lo comentado en el patrón anterior, el *Page Controller*, es decir, la ruta (función-vista) es la responsable de proporcionar a la plantilla estos datos dinámicos junto con la orden de renderizado.

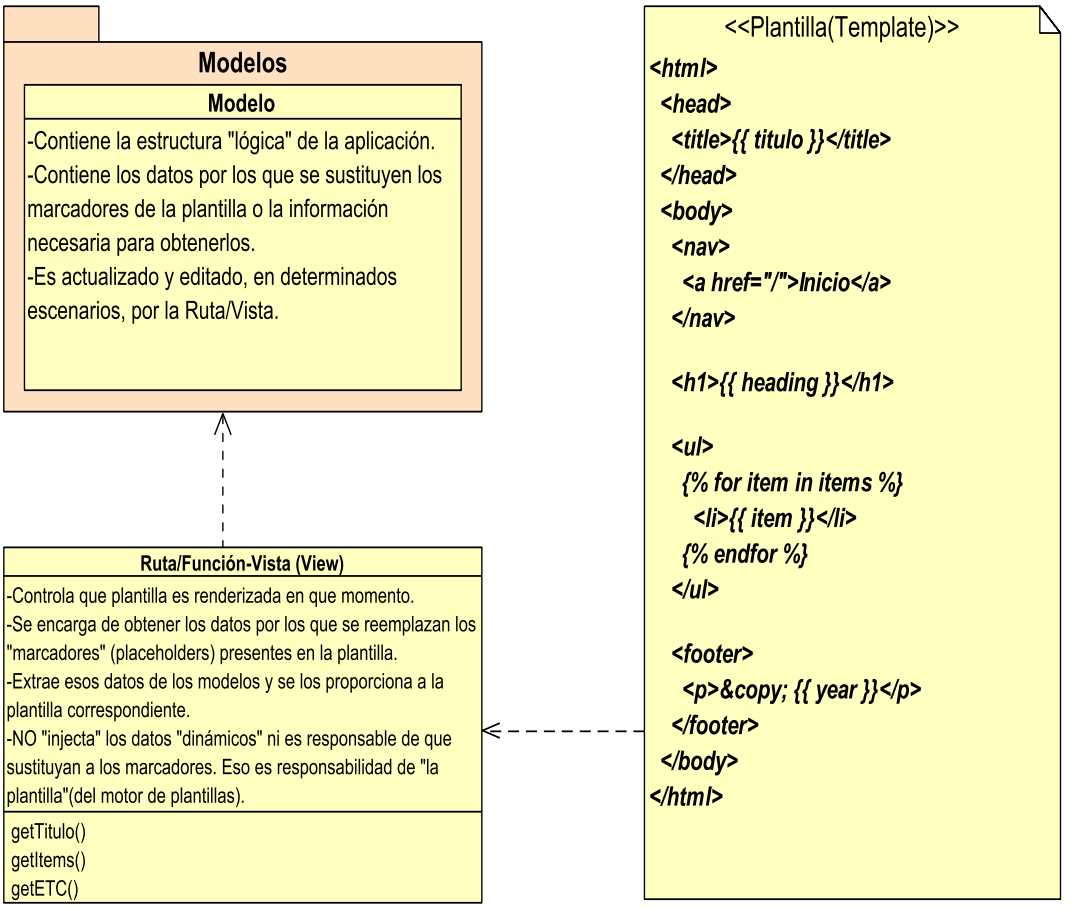

Figura 6.5: Diagrama representando el patrón de diseño *Template View*.

El patrón *Template View* no es uno para el que sea sencillo idear un diagrama con el que reflejar todas sus características y matices, por eso el diagrama para dicho patrón, en la Figura 6.5, está inspirado en el utilizado en el artículo sobre **Template-View** escrito en la web de Martin Fowler [70]. En el artículo, el intermediario entre la plantilla y el/los modelo/s donde se encuentran los datos dinámicos que la plantilla requiere es un objeto *Helper*, que representa a cualquier tipo de entidad que se encargue de responder a las necesidades de información de la plantilla y de obtenerlas de los modelos del sistema, en este sistema esa responsabilidad recae sobre las rutas de cada página (*URL*) del servicio web así que por eso en vez de llamar a la clase *Helper* se ha especificado que se trata de la ruta que corresponda. Antes de pasar al siguiente patrón se listan una serie de aclaraciones para garantizar que las limitaciones del diagrama no hayan resultado en que se haya entendido erróneamente el patrón:

- La lógica que **puede** estar definida en las plantillas gracias a los *marcadores* (*placeholders*) debe ser **exclusivamente** lógica de presentación, es decir, estos marcadores solo pueden contener instrucciones que afecten a la propia interfaz, al **cómo** se muestra el contenido definido en la plantilla, pero **no** pueden contener lógica que busque alterar **qué** se muestra en la plantilla ni el **por qué** se muestra lo que se muestra.

- Los *placeholders* son marcadores HTML definidos en la estructura de la plantilla mediante los delimitadores de Jinja [25], que son simplemente una serie de combinaciones de caracteres predefinidas en **Jinja** y que sirven precisamente para indicar al motor de plantillas que el contenido de dichas marcas (incluyendo los caracteres que delimitan el marcador) debe ser reemplazado por los aquellos elementos dinámicos que no estaban disponibles durante la definición de la plantilla. En el ejemplo de plantilla del diagrama; *{ título }*, *{ heading }*, *{ % for item in items %}*, *{{ item }}*, *{ % endfor %}* y *{{ year }}*, **todos** son ejemplos de marcadores (placeholders) JINJA y como se puede apreciar pueden sencillamente corresponder a un objeto, pero también pueden representar instrucciones como el caso de `for item in items` en el que el marcador no simplemente pretende ser reemplazado por un objeto, sino que el marcador contiene lógica que debe ejecutarse en base a información que no está aún disponible, en este caso un iterable llamado *items* que una vez disponible permitirá al motor de plantillas Jinja reemplazar ese marcador por un bucle que itere por cada entrada de *items*.

- Por si no resultase evidente, los datos que la ruta proporciona a la plantilla son precisamente aquellos por los que la plantilla va a reemplazar los marcadores.

- Por último, en el diagrama se indica que el Modelo es actualizado y editado por la Vista en determinados escenarios, con esto se quiere decir que aunque es cierto que la vista puede actualizar los modelos, esto es solo bajo condiciones controladas como que reciba un formulario que debe procesar y/o el método de la solicitud HTTP pasa a ser **POST**.

6.3.4. *Data Mapper*

Este patrón establece una capa, el *Data Mapper*, dedicada a separar los objetos en memoria del esquema de la base de datos, mapeando entre ambos [42]. En otras palabras, el patrón *Data Mapper* consiste en establecer un intermediario entre la representación en memoria (*in-memory*) de datos, es decir, los objetos del dominio del sistema, y el almacenamiento de estos datos, es decir, la persistencia, como podria ser una base de datos relacional (el caso en este proyecto).
Curiosamente, de este patrón se dice que favorece la separación de la lógica de negocio y la de acceso a datos, y, teniendo en cuenta lo establecido en el apartado dedicado a la capa lógica de este sistema en la que se combinan las capas de Dominio y de Acceso a Datos 3, este hecho es más evidente aún.
SQLAlchemy ORM, usado aquí, es un *Data Mapper* clásico: las clases modelo (como Nivel, Bloque) se mapean a tablas y el ORM traduce entre objetos Python y filas SQL. Como se ha mencionado en los apartados anteriores, SQLAlchemy logra esto gracias al modelo declarativo, mediante el cúal es posible definir una misma entidad una clase Python y su tabla MySQL, una vez la relación entre la clase y la tabla está establecida, SQLAlchemy sencillamente utiliza un objeto *mapper* que mediante los datos y metadatos de los pares **Clase/Tabla** establece una relación entre cada instancia de la clase Python y la fila correspondiente de la tabla MySQL.

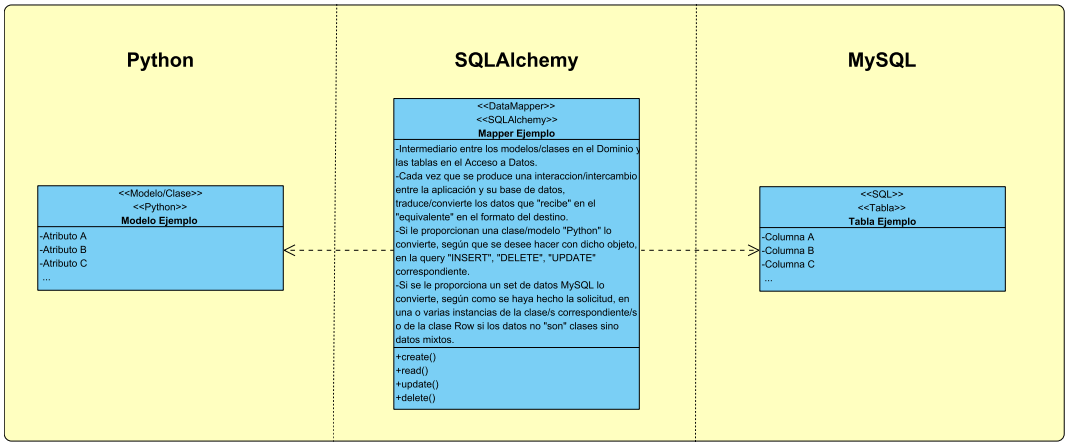

Figura 6.6: Diagrama representando el patrón de diseño Data Mapper.

6.3.5. *Singleton*

El patrón *Singleton* es un patrón de diseño creacional que consiste en garantizar que una clase tenga una única instancia, y proporcionar un punto de acceso global a dicha instancia, de esta forma permite controlar y regular el acceso a dicho recurso, ya que la única vía de acceso es el *singleton*, reduce también problemas relativos a conflictos debidos a diferencias entre instancias de la clase y similar [21].
El ejemplo principal en este proyecto es la instancia db, presente en la Figura 8.1, que es la instancia de la clase SQLAlchemy, esta permite control total sobre las interacciones entre la app y su base de datos durante la ejecución entera, ya que esta instancia se crea

durante el despliegue (inicialización) de la aplicación y se establece como parámetro de la propia instancia de aplicación, de esta forma la única instancia de SQLAlchemy para toda la aplicación es esa instancia db lo que garantiza que, independientemente desde que punto del código se acceda a la base de datos, siempre va a ser la misma base de datos.

6.3.6. *Decorator*

El patrón *Decorator* permite añadir comportamiento a funciones o clases de forma dinámica sin modificar su definición (su código fuente). Facilita la reutilización de código, pudiendo definir funciones más genéricas y reutilizables, ya que posibilita el añadir los aspectos más específicos mediante decoradores [13].
Este patrón es intrínseco a Flask, ya que lo que establece una función como **Ruta** de una URL de la web es precisamente el decorador `nombre_blueprint.route(/URL, methods=[...])`.

6.4. Realización de Caso de Uso en Diseño

Finalmente, para ejemplificar el conjunto total del modelo de diseño se ha creado un diagrama de secuencia que representa la realización del que se ha considerado el caso más relevante e interesante del sistema, que es **Evaluar una respuesta de un jugador**.
Este proceso técnicamente no es un caso de uso por sí mismo, sino que forma parte del caso de uso **Jugar un nivel**, especificado en la Tabla 4.4, sin embargo, ya en el capítulo de Analisis, en la sección de Realización de Casos de Uso5.2.2, se comentó que, aunque desde un punto de vista humano, el Caso de Uso sería todo el proceso de un usuario jugando un nivel hasta completarlo, ese proceso está compuesto por dos subsecuencias bastante complejas, especialmente el proceso de evaluar una respuesta, que es muy complejo, y por eso se ha decidido que lo adecuado es, a la hora de detallar la realización, a nivel de diseño, del caso de uso **más** importante del proyecto, actuar de la misma forma que se actuó en el capítulo de Análisis. En el capítulo de Análisis, en la sección 5.2.2, el caso de uso **Jugar un nivel** fue dividido en dos secuencias y diagramas distintos, una para reflejar el movimiento entre los bloques de un nivel 5.2.3 y la otra para reflejar el proceso de evaluación, por parte del sistema, de la respuesta de un jugador 5.2.4.
En esta sección se ha actuado de la misma forma y **Jugar un nivel** es considerado un escenario compuesto por dos secuencias, una que refleja el movimiento entre los bloques del nivel y otra que refleja el proceso de evaluación de la respuesta del jugador a una pregunta del nivel, siendo esta última la más importante y por lo tanto, la cual se desarrolla en la realización.
De nuevo, este diagrama de secuencia representa la realización del escenario en el que un jugador ha solicitado que se evalúe su respuesta a una pregunta del nivel. Empieza con el jugador alcanzando un Bloque Pregunta, escribiendo su respuesta y solicitando que sea evaluada y termina cuando el servidor regresa con el resultado de la solicitud de evaluación y se lo muestra al jugador. Como la secuencia es considerablemente extensa y el diagrama puede resultar complejo, se describe de forma resumida el flujo de la secuencia de manera textual:
El **Jugador**, mientras juega a un nivel del sistema, alcanza un bloque con rol **Pregunta**, el sistema, como ha hecho en el resto de bloques del nivel, renderiza la plantilla correspondiente. Tras esto, el Jugador interactúa con la interfaz para proporcionar su respuesta a la pregunta y pulsar el botón que dispara el proceso de evaluación. Tras esto, el sistema prepara el paquete

de datos con la información necesaria para **evaluar la respuesta** del jugador y, mediante una solicitud asíncrona, ordena la evaluación de la respuesta en cuestión.
Esto invoca a la ruta responsable del proceso de evaluación la cual se encarga de orquestar una serie de llamadas a funciones auxiliares[9] para, paso a paso, evaluar la respuesta del jugador y regular el estado de la evaluación. A nivel de código, el proceso presenta múltiples casos alternativos y excepciones dependiendo no solo de si la respuesta del jugador es correcta sino de otros muchos factores como la configuración del Bloque cuya pregunta está siendo respondida, el que dicho bloque se haya configurado correctamente, etc. Por todo esto, y para no entrar en detalles no relevantes a nivel de Diseño, el diagrama muestra la secuencia del llamado *Happy Path*, es decir, el escenario en el que no se produce ninguna excepción y se sigue totalmente la secuencia normal.
Continuando donde se había dejado, una vez el paquete de datos llega a la ruta responsable de la evaluación, el proceso se lleva a cabo mediante la gestión, por parte del sistema, de los siguientes eventos en orden: Se realiza la primera prueba de validación, se recoge información necesaria para la segunda prueba de validación, se realiza la segunda prueba de validación, se procede a crear y desplegar el entorno MySQL necesario para evaluar la respuesta del jugador[10], una vez el entorno está listo, se realiza la última prueba de validación, asumiendo que esta se supere, se alcanza un punto crítico del proceso, la ejecución de la respuesta del jugador en el entorno creado. En este punto, una vez se ha ejecutado la respuesta del jugador, se hace `commit` para poder regresar a este estado cuando sea necesario. Tras esto se entra en un bucle en el que el sistema, con la lista de **Tests** correspondiente a la pregunta siendo respondida, debe ejecutar cada uno de esos tests, uno a uno, contra el entorno de evaluación, en el que están almacenados los datos en los que se perciben los efectos de la respuesta del jugador, de esta forma, cada test examina un aspecto de los efectos de la respuesta del jugador para, de esta forma, poder determinar si la respuesta es correcta (hace lo solicitado en el enunciado de la pregunta) o no. El sistema analiza cada instancia de Test para determinar como interpretar los resultados de su ejecución (y así saber si se ha superado o fallado el test), si el test es superado, el sistema hace `rollback` del entorno para descartar cualquier posible modificación que el test pudiese haber causado a la base de datos de evaluación y se procede a realizar el test siguiente.
El ciclo continúa hasta que:

1. No se supere un test, en cuyo caso no se siguen ejecutando tests aunque hubiese aún pendientes, pues fallar un test significa que la respuesta es incorrecta.

2. Se superen **todos** los tests correspondientes al bloque de la pregunta. Si este es el caso, cuando se supera el último test y se abandona el bucle, la respuesta es declarada como correcta.

Una vez se ha evaluado la respuesta, el sistema o en concreto, la ruta de evaluación, invoca al set de funciones encargadas de limpiar el entorno de datos creado para la evaluación y, una vez terminada la limpieza, se genera un paquete de datos con los resultados de la evaluación.

[9] Sencillamente funciones *puras* que no actúan como rutas, sino que se encargan de las partes del proceso de evaluación demasiado complejas como para que fuese razonable hacerlas responsabilidad de la ruta de evaluación.

[10] Esto incluye, crear el contenedor **Docker** que albergará la base de datos **MySQL** para evaluación, crear los objetos **MySQL-connector** que permiten al sistema comunicarse directamente con la base de datos de evaluación sin involucrar al contenedor que lo contiene y finalmente, poblar la base de datos de evaluación con los datos correspondientes a la pregunta que se va a responder.

Estos datos son enviados de vuelta al jugador (Cliente), donde se procesan dichos resultados y son mostrados al jugador, acabando la secuencia.
Debido a la complejidad y extensión del diagrama, para que este siguiese siendo legible, ha sido necesario partir el diagrama entre múltiples figuras. Estas figuras son: **Primera Parte** 6.7, **Segunda Parte** 6.8, **Tercera Parte** 6.9 y **Parte Final** 6.10.

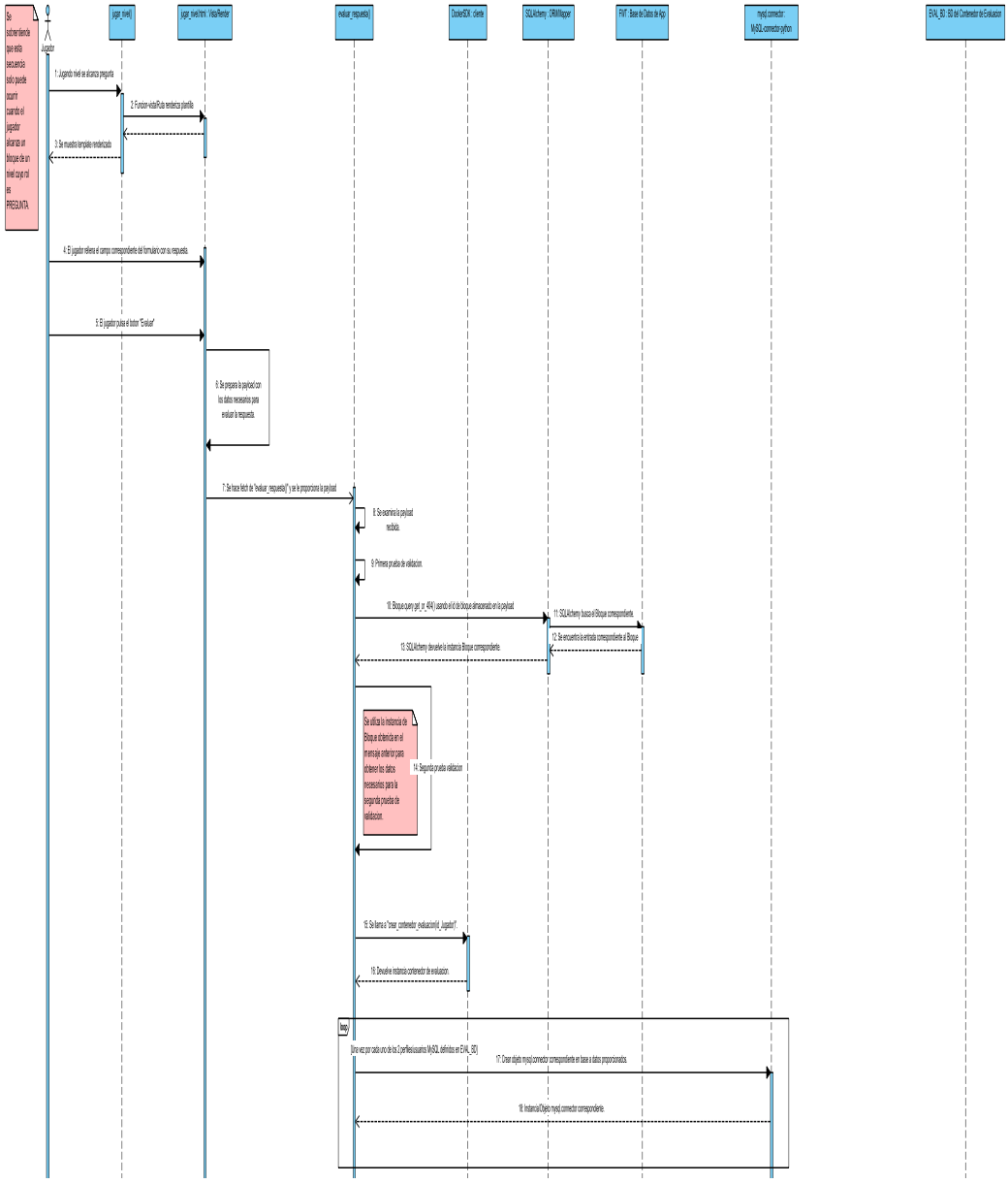

Figura 6.7: Diagrama de secuencia de diseño reflejando la primera parte del proceso de evaluación de la respuesta de un jugador.

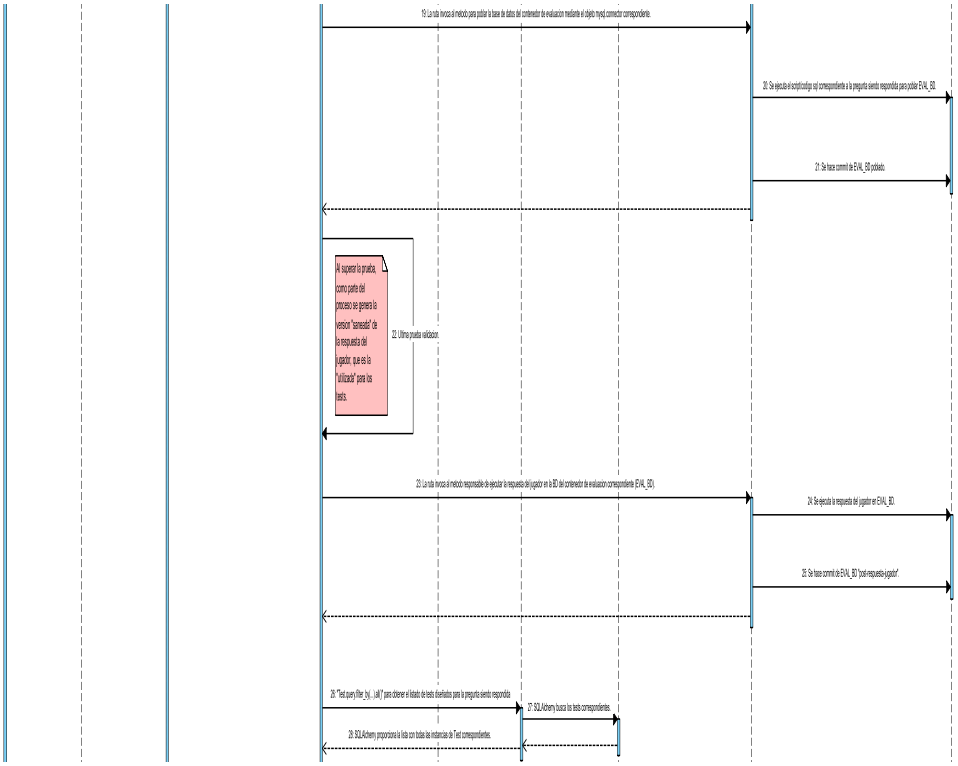

Figura 6.8: Diagrama de secuencia de diseño reflejando la segunda parte del proceso de evaluación de la respuesta de un jugador.

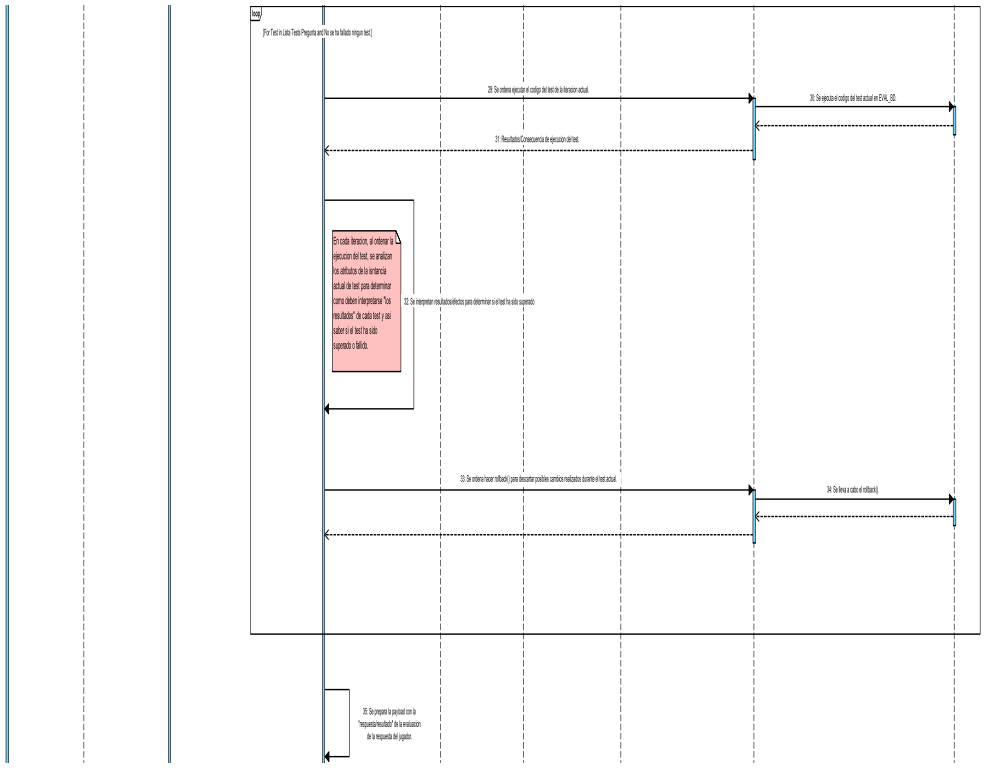

Figura 6.9: Diagrama de secuencia de diseño reflejando la tercera parte del proceso de evaluación de la respuesta de un jugador.

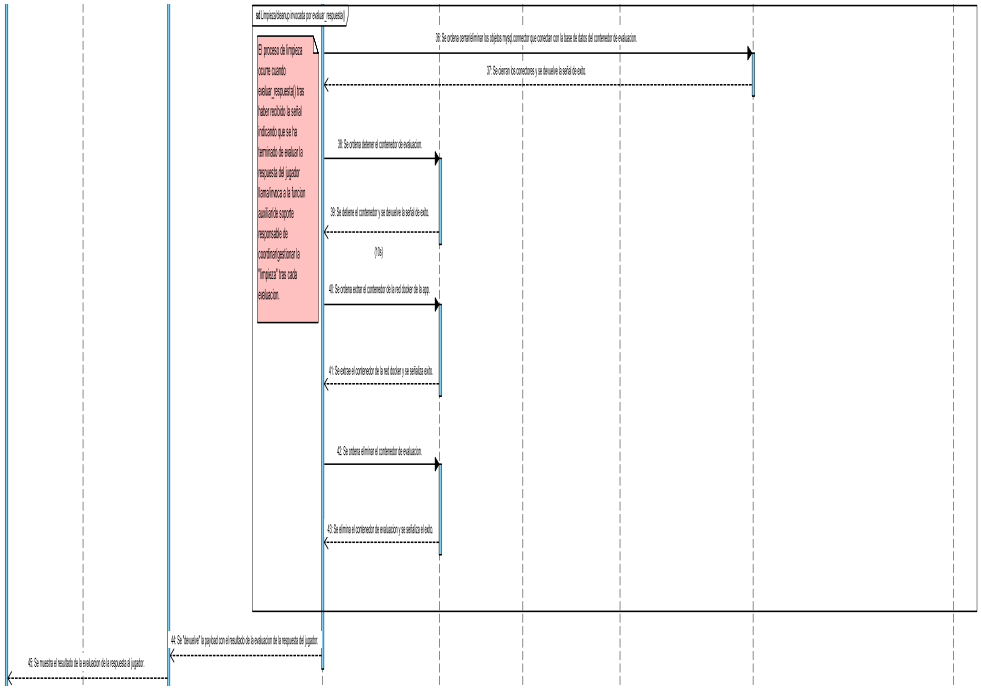

Figura 6.10: Diagrama de secuencia de diseño reflejando la última parte del proceso de evaluación de la respuesta de un jugador.

Capítulo 7

Tecnologías y Herramientas

7.1. Introducción

Durante el transcurso de este TFG, se han utilizado numerosas tecnologías, herramientas y extensiones, que han permitido alcanzar el fin del proyecto, facilitando el desarrollo, construcción y gestión del mismo.

7.2. Herramientas de desarrollo

En esta sección se presentan las herramientas involucradas en el proceso de desarrollo de la aplicación, tanto a nivel de implementación de la aplicación, como de su funcionalidad.
Se incluyen también en esta sección las herramientas de gestión por estar su uso tan ligado al proceso de desarrollo.

7.2.1. Visual Studio Code

Visual Studio Code o **VSCode** [48] es un editor de código extremadamente popular, considerado entre los 5 más utilizados en la actualidad. Esto se debe a su versatilidad y comodidad de uso gracias a su sistema de extensiones, lo que permite a VSCode ser, de base, un editor relativamente ligero, pero también altamente configurable y extendible gracias al, valga la redundancia, sistema de extensiones. En el caso de este proyecto, las extensiones que merece la pena destacar son:

Python

VSCode dispone de una extensión para Python [34], gracias a la cual VSCode da soporte adicional a la programación en Python, proporcionando, entre otros, un servicio de *debugging* (*Python Debugger*) y soporte para entornos virtuales como el utilizado por la aplicación.

SQLTools

Extensión [35] para gestionar bases de datos **SQL** y ejecutar *queries* desde VSCode. Incluye múltiples subextensiones como si se tratase de un paquete.

Jinja

Extensión [32] que mejora en general la experiencia de uso y trabajo con plantillas Jinja mediante VSCode, incorpora utilidades como: resaltar la sintaxis, detectar errores y la más útil de ellas, tiene total integración con la extensión Python, incrementando la cohesión y fluidez entre ambos. Es particularmente útil, ya que Jinja es el motor de plantillas utilizado por defecto por Flask, que como se explicará a continuación, es el *framework*[1] utilizado en el proyecto.

Docker

Extensión [31] que permite gestionar contenedores y redes Docker, así como imágenes Dockerfile desde VSCode, siempre y cuando Docker Desktop esté instalado en el sistema. De forma similar a SQLTools, es más bien un conjunto de extensiones y no una única extensión.

Live Preview

Esta última extensión [33] no tiene el peso que las otras tienen en el resultado final, pero resulta de gran utilidad a la hora de trabajar en plantillas para servicios web, ya que permite generar una *preview* de archivos HTML en tiempo real lo cual resulta extremadamente útil a la hora de diseñar estas plantillas, ya que permite no necesitar desplegar continuamente la aplicación para comprobar el aspecto y el efecto de los cambios hechos a cada plantilla.

7.2.2. Docker Desktop

Docker es una plataforma de software que permite crear, desplegar y ejecutar aplicaciones utilizando contenedores. Estos contenedores son entornos ligeros, portables y aislados que incluyen todo lo necesario para ejecutar una aplicación, elementos como: el código, las dependencias, la configuración. De esta forma, facilitan la portabilidad y la consistencia entre diferentes entornos.
En este proyecto, Docker **no** es utilizado para contener y desplegar la aplicación, sino que se usa Docker para desplegar y gestionar los entornos de datos involucrados en el correcto funcionamiento de la aplicación. Estos son la base de datos de la aplicación, llamada **FWT** como referencia al nombre original del proyecto, y los múltiples entornos, también MySQL, creados, utilizados y eliminados, dinámicamente, por el sistema durante el proceso de evaluación de respuestas. Estos entornos de evaluación siempre reciben el mismo nombre[2], que es **EVAL_BD**.
Tanto la base de datos de la aplicación, como las bases utilizadas para la ejecución y evaluación de las respuestas de los usuarios del sistema, son desplegadas en sus correspondientes contenedores Docker. En el caso de la base de datos de la aplicación, este contenedor persiste independientemente de si la propia aplicación esta desplegada o no, mientras que los contenedores de los entornos de evaluación, los denominados **Contenedores de Evaluación**, solo existen mientras la aplicación esta en funcionamiento, es más, cada contenedor no es construido hasta el momento en el que la aplicación lo necesita, y una vez creado hace uso del contenedor inmediatamente, para realizar la solicitud de evaluación recibida y, una

[1] Técnicamente es un *microframework*.

[2] Pueden ser dados el mismo nombre porque, los contenedores donde se despliegan, como se ha mencionado al principio de la explicación, **aíslan** el contenido del contenedor, de forma que no hay problemas de nombrado ya que los distintos entornos MySQL de evaluación están aislados entre sí.

vez la respuesta ha sido evaluada, el contenedor (y la base de datos MySQL de su interior) es eliminado, de forma que los contenedores de evaluación solo pueden existir mientras la aplicación está en funcionamiento.
En resumen, **Docker** se utiliza, en este proyecto, para desplegar la base de datos de la aplicación (solo si por algún motivo la base no esta ya en funcionamiento), gestionar las interacciones entre la aplicación y su base de datos, y para, mediante los contenedores de evaluación, poder llevar a cabo la evaluación de respuestas en entornos aislados, limpios y controlados, imposibilitando que, en caso de sufrir un ataque de un usuario mal intencionado, los efectos de la respuesta dañina puedan tener efecto en el exterior del contenedor. Todo esto con el beneficio adicional de un uso bajo de recursos del *host* y un proceso de uso relativamente intuitivo.
Antes de proceder con la descripción de **Docker Desktop** es conveniente hacer las siguientes dos anotaciones:

1. En el proyecto, Docker también es utilizado para desplegar una red Docker, necesaria para que sea posible a la aplicación alcanzar los contenedores Docker de su entorno. Se habla de ello en más detalle en el apartado 7.2.2 de este mismo capítulo.

2. Las bases MySQL mencionadas previamente son eso, bases de datos, los nombres asignados a los contenedores en los que se despliegan son: **c_BD** (contenedor_Base de Datos) para el contenedor de la base de datos de la aplicación, mientras que, para los contenedores de evaluación se utiliza un patrón mediante la línea `nombre_contenedor = f''exec_id_usuario_container''` donde `nombre_contenedor` es la variable en la que se almacena el nombre de la instancia de contenedor de evaluación que esta siendo creada e `id_usuario` es el identificador del usuario del sistema que ha solicitado que su respuesta sea evaluada, de esta forma, a cada contenedor de evaluación se le nombra como *exec_ identificador del cliente_ container*.

Docker Desktop [23] es una herramienta de interfaz gráfica (*Graphical User Interface* o *GUI*) que proporciona una interfaz con la que interactuar para gestionar y manejar contenedores, redes, volúmenes e imágenes Docker. En otras palabras, es lo que incorpora y permite hacer uso de Docker dentro del *host* donde se despliegue la aplicación del proyecto.
Docker Desktop también incluye el propio Motor Docker (*Docker Engine*) así como su propia *Máquina Virtual* (Virtual Machine o VM) necesaria porque Docker, como tal, es una herramienta basada en Linux, por lo que para funcionar en Windows, como ocurre en este proyecto, necesita disponer de su propia VM. En conjunto, proporciona un entorno intuitivo, cómodo y sencillo mediante el cual desarrollar y gestionar componentes Docker. En este caso, sin embargo, su función es principalmente proporcionar y activar y desactivar el *Docker Engine*, ya que sin el *Docker Engine* no es posible ni crear, ni gestionar ni ejecutar contenedores en ninguna máquina, es decir, es el núcleo sin el cual no es posible ningún tipo de funcionalidad Docker.

Docker Compose

Es como tal un componente de Docker Desktop. Como herramienta, sirve para definir y desplegar aplicaciones multi-contenedor, ya que permite especificar en un único archivo todos los contenedores implicados y su configuración y desplegar e iniciar todos ellos mediante un único comando.

En el caso de este proyecto, se usa para desplegar y activar la Red Docker (*Docker Network*) que permite comunicar la aplicación con los distintos contenedores Docker de su entorno, así como para desplegar y activar el contenedor Docker en el que se almacena la base de datos de la aplicación.

7.2.3. Git

Un sistema de control de versiones distribuido [54]. Diseñado para registrar y gestionar cada cambio realizado en los archivos de un proyecto y manteniendo un historial de las versiones anteriores, permitiendo examinar, comparar e incluso regresar a dichas versiones con relativa facilidad. Se ha utilizado desde que se comenzó a desarrollar la aplicación.
Cabe mencionar que el uso de Git ha sido a mediante **GitHub** [55]

7.2.4. Google Chrome

Se trata del navegador utilizado para realizar todas las pruebas durante el desarrollo de la aplicación [56].

7.3. Herramientas de diseño, modelado y documentación

Esta sección contiene las herramientas que se han utilizado para crear los distintos gráficos, modelos, diagramas y documentos del proyecto (incluida esta propia memoria).

7.3.1. Visual Paradigm

Herramienta que permite el modelado [40] de la gran mayoría de diagramas que se puedan necesitar de un proyecto. Se ha utilizado para crear todos los diagramas **UML** del proyecto.

7.3.2. Overleaf

Overleaf [39] es una plataforma en la nube diseñada para la edición de documentos LaTeX. Su uso está muy extendido en la comunidad académica por estar diseñado teniendo en cuenta el proceso científico, de ahí que sea muy utilizada en universidades y otros entornos académicos similares. Este documento ha sido creado desde Overleaf.

7.3.3. Crixet

Herramienta similar a Overleaf pero totalmente gratuita [49].
Se utilizó para reemplazar a Overleaf cuando, por el tamaño del documento, se superó el límite de tiempo para compilación permitido a las cuentas gratis de Overleaf.

7.4. Herramientas de comunicación

Esta sección contiene las herramientas utilizadas para comunicarse con el Tutor durante el transcurso del TFG.

7.4.1. Microsoft Teams

Microsoft Teams [60] es una plataforma de colaboración para equipos que combina chat, videollamadas, reuniones en línea, almacenamiento en la nube y la posibilidad de integrar otras aplicaciones de Microsoft 365. La mayoría de reuniones se han realizado vía Teams.

7.4.2. Microsoft Outlook

Microsoft Outlook [59] es un cliente de correo electrónico y gestor de información personal. Es muy utilizado a nivel global y al ser la plataforma del correo electrónico de estudiante, ha sido uno de los principales medios a través de los cuales se ha mantenido la comunicación con el tutor.

7.5. Tecnologías utilizadas

En esta sección se describen las principales tecnologías empleadas en el desarrollo de la aplicación, tanto a nivel de desarrollo del código como de la funcionalidad intrínseca. Son las listadas en el archivo requirements.txt de la aplicación, y se van a listar solo las más importantes.

7.5.1. Python 3

Python 3 [62] es el lenguaje principal de la aplicación. Destaca por su sintaxis legible, su amplia biblioteca estándar y su gran comunidad, lo que lo hace ideal para desarrollo web rápido y scripting.

7.5.2. Flask

Flask [24] es un *microframework* flexible para Python que proporciona el esqueleto de la aplicación web: enrutado, manejo de peticiones y fácil integración con extensiones para autenticación, ORM, formularios y similar.

7.5.3. SQLAlchemy

SQLAlchemy [63] es una biblioteca Python compuesta por dos componentes principales:

1. SQL Toolkit: El *Core* (núcleo) que permite gestionar bases de datos SQL, así como construir y ejecutar *statements* de forma programática, pero sin descartar la posibilidad de ejecutar SQL puro.

2. ORM (Object-Relational Mapper): Se encarga de **mapear** las clases (modelos) Python y sus atributos a las correspondientes tablas y columnas en la base de datos SQL y gestiona toda transacción de información entre la base de datos y la aplicación para básicamente traducir la información proveniente de una fuente, en la instancia correspondiente usando el formato del destino. En otras palabras, se encarga de mapear las clases Python a las Tablas SQL y traducir cada instancia o atributo Python a la correspondiente tabla o columna SQL y viceversa.[3]

[3]Se explica de manera más detallada más adelante, pero debido a esto SQLAlchemy actúa como un intermediario entre las capas de Dominio y Acceso a Datos.

En este proyecto se usa SQLAlchemy **exclusivamente** a la hora de interactuar con la base de datos **de la aplicación**.

7.5.4. Flask-SQLAlchemy

Extensión [58] que integra SQLAlchemy en Flask, simplificando la configuración, gestión de sesiones y transacciones para operaciones de base de datos dentro de la aplicación.

7.5.5. MySQL

Sistema de gestión de base de datos relacional [61]. En este proyecto los dos tipos de bases de datos involucrados son MySQL, y corresponden a la base de datos de la aplicación y las bases de datos de los contenedores de evaluación, sin embargo, solo el primer ejemplo es utilizado para almacenar información, las bases de datos de los contenedores de evaluación únicamente son utilizadas como entorno MySQL, es decir, no se utilizan por ser capaces de almacenar datos, sino por ser entornos MySQL en los que es posible ejecutar *queries* y *statements*.

7.5.6. mysql-connector-python

Driver (o *controlador*) oficial de MySQL para Python [38], imprescindible para conectar la aplicación directamente tanto a las bases de datos de los contenedores de evaluación como, indirectamente, a la propia base de datos de la aplicación. En este último caso es indirectamente porque, aunque la comunicación entre la aplicación y su base de datos se hace mediante SQLAlchemy, el propio SQLAlchemy necesita un *driver* para poder conectarse a MySQL y en nuestro caso, se ha escogido **mysql-connector-python** como *driver*, ya que en entornos ORM da mejores resultados (más eficiente y rápido) que otros *drivers* como PyMySQL. Por eso se utiliza el término, indirectamente, porque la aplicación interactúa con su base de datos mediante SQLAlchemy, pero SQLAlchemy sí que depende de mysql-connector.

7.5.7. Jinja2

Motor de plantillas moderno e intuitivo que permite incrustar expresiones y lógica tipo Python en archivos HTML [4]. Como se mencionó antes, es el motor utilizado por Flask por defecto y por ello, igual que necesitamos la extensión, necesitamos la librería.

7.5.8. WTForms

Librería para definir, validar y renderizar formularios como clases Python, compatible con múltiples tipos de campo y validadores [65].

7.5.9. Flask-WTF

Extensión de WTForms para Flask que añade protección CSRF y facilita la gestión de formularios usando el contexto de peticiones y sesiones de Flask [53].

7.5.10. Flask-Login

Gestión de sesiones de usuario en Flask: autenticación, persistencia de sesión y control de acceso a rutas protegidas [52].

7.5.11. Flask-Bcrypt / bcrypt

Bibliotecas para *hashing* y verificación de contraseñas con el algoritmo bcrypt, resistente a ataques de fuerza bruta [51].

7.5.12. itsdangerous

Librería para firmar criptográficamente datos (cookies, tokens, sesiones), garantizando que no se puedan alterar sin invalidar la firma [57].

7.5.13. docker (Python SDK)

SDK que interactúa con la API de *Docker Engine*; permite crear, gestionar y eliminar contenedores desde el propio código Python, esencial para instancias de MySQL aisladas [50].

7.5.14. sqlglot

Parser, *transpiler* y optimizador de SQL en Python [44]. Fue incorporado a la aplicación por no necesitar un entorno MySQL para funcionar, lo que permite utilizarlo para inspeccionar la propia respuesta del jugador, el *input*, para determinar si es *parseable* como SQL o no sin necesitar desplegar un entorno SQL.
En otras palabras, es utilizado por la aplicación para garantizar que la respuesta del jugador al menos sea *parseable* como SQL antes de tratar de ejecutarla para su evaluación, de esta forma la aplicación evita gastar recursos en crear, montar y desplegar un contenedor Docker de evaluación con su correspondiente base de datos solo para que la respuesta del jugador dispare un error, ya que no es SQL (no se puede *parsear* como SQL) y por ende cuando la aplicación trate de ejecutarla para su evaluación, dispare un error al haber tratado de ejecutar como query (como si fuese SQL) algo que no es SQL. Si la respuesta supera esa prueba de *parseo* (junto con otras que se mencionarán más adelante), entonces se considera que es SQL válido y se procede con su evaluación.

7.5.15. Werkzeug

Werkzeug [64] es un conjunto o, más concretamente, colección, de bibliotecas, también denominado como un *toolkit* enfocado a la creación de aplicaciones (servicios) web compatibles con **WSGI** (*Web Server Gateway Interface*) [26]. **WSGI** se traduce, en Español, a Interfaz de Puesta de Enlace de Servidor Web y es de forma resumida, una **especificación** que describe como un servidor web se comunica con las aplicaciones web y como estas aplicaciones web pueden comunicarse entre ellas para procesar solicitudes de servicio en el ámbito de **Python**. En otras palabras, es una especificación que define una interfaz estándar de comunicación entre servidores web y aplicaciones o *frameworks* web escritas en **Python**.
El motivo por el que se mencionan **Werkzeug** y **WSGI** es que estos son cruciales para **Flask**, ya que, como se indica en la propia documentación de Flask [28], Flask depende de

dicho *toolkit*. De hecho, en la documentación de WSGI se habla también de la relación entre WSGI, Werkzeug y Flask [29]. En resumen, **Werkzeug** es un *toolkit* para WSGI, en concreto es un set de herramientas para la creación de aplicaciones web **compatibles** con WSGI y el *framework* (técnicamente *microframework*) utilizado para este proyecto, Flask, depende de dicho *toolkit* para su funcionamiento adecuado.

Capítulo 8

Implementación

En este capítulo se presenta la estructura de directorios del proyecto, así como la arquitectura física del sistema a nivel de implementación para explicar como se despliega la aplicación y por último se explicará de forma más detallada el uso que se ha dado en la aplicación a las tecnologías y/o herramientas más cruciales a la hora de alcanzar los objetivos de la aplicación.
Cabe mencionar que aunque la aplicación al final funciona mediante la ejecución de tests en entornos controlados para determinar los efectos de la respuesta del jugador y así comprobar si la respuesta hace lo solicitado o no, este no fue siempre el caso. Inicialmente, el **tema** del **TFG** era bastante distinto, se iba a crear una actividad en el entorno educativo **Moodle** utilizando un plugin llamado **CodeRunner** para tratar de incorporar lo que fue inicialmente la actividad **FreeWheeling Travelers** dentro de Moodle, utilizando CodeRunner para incorporar un sistema de evaluación automático que permitiese la evaluación de las respuestas de los alumnos sin necesitar aumentar la carga de trabajo de los profesores responsables de la actividad. Lamentablemente, se alcanzó un punto en dicho tema en el que no era posible avanzar y tampoco se disponía de suficiente como para justificar mantener el tema del proyecto, tras esto se decidió el tema actual.
También han ocurrido cambios durante el propio desarrollo del tema actual, inicialmente la evaluación de respuestas iba a realizarse mediante **RegEx**, sin embargo, se consideró ser demasiado impráctico e ilógico, pues implicaría introducir como requisito, en una aplicación pensada para actuar como refuerzo a la enseñanza de SQL, que los individuos a los que se les hiciera responsables de dirigir la actividad (en el supuesto en que la aplicación del proyecto se llevase a producción), tener conocimientos no solo sobre SQL sino también los suficientes sobre RegEx como para que esto no limitase su capacidad de usar la aplicación para crear niveles.
Otro cambio en los planes, relativamente interesante, es respecto a como se iba a garantizar que, durante la fase de ejecución de los tests para evaluar la respuesta de un jugador, los resultados de un test anterior no pudiesen influenciar los resultados del test siguiente. Como se ha mencionado en el capítulo anterior, actualmente se lleva a cabo mediante el uso de las instrucciones `commit()` y `rollback()` para registrar el estado de la base de datos del contenedor de evaluación en el momento de la llamada a la instrucción y descartar todos los cambios hecho desde la última versión registrada respectivamente, de esta forma, se hace `commit` justo tras ejecutar la respuesta del jugador en el entorno de evaluación para que la versión registrada sea la requerida para el correcto funcionamiento de los tests y acto seguido,

cada vez que se finaliza un test de la batería de pruebas correspondiente a la pregunta que se esté respondiendo, se ejecuta `rollback` para limpiar entre cada test.
Inicialmente, se iba a ejecutar un *script* entre cada test, para obtener un listado de las tablas existentes en el entorno de evaluación para, acto seguido, eliminarlas y ejecutar de nuevo el *script* del Bloque Pregunta correspondiente para poblar el entorno de evaluación con los datos base y, acto seguido, ejecutar de nuevo la respuesta del jugador. De esta forma se alcanzaría de nuevo el estado deseado del entorno de evaluación, sin embargo, durante el proceso de desarrollo, se optó por usar el procedimiento actual por ser mucho más sencillo, intuitivo y fácil de mantener.

8.1. Estructura del proyecto

Como puede observarse en la representación textual de la estructura de archivos del **proyecto**, el proyecto está moderadamente modularizado. La distribución en directorios se ha hecho en base a lo considerado como buena práctica así como en función de la naturaleza de los contenidos.
Inmediatamente a nivel de la raíz del proyecto (**ROOT**) están las carpetas `app`, `contenedores_eval`, `db`, `logs` y `static` así como los ficheros `config.py`, `requirements.txt` y `run.py`.

- `app`: Esta carpeta contiene la aplicación Flask en sí, todo lo relativo a su lógica, sus modelos, formularios, etc. El contenido de esta carpeta es lo que debería ser considerado como el servicio web o la aplicación.

- `contenedores_eval`: Esta carpeta contiene a su vez otros dos elementos, su carpeta *scripts* en la que está almacenado el *script* SQL que se copia en el *entrypoint* del entorno MySQL del contenedor de evaluación durante la inicialización de este y luego, al mismo nivel que *scripts*, se encuentra el archivo `Dockerfile` específico de los contenedores de evaluación, diseñado para generar la imagen a partir de la cual la aplicación construye los **contenedores de evaluación** y la base de datos MySQL de su interior. El conjunto "contenedor + base de datos" es lo que ha estado siendo referido como entorno de evaluación.

  Por último, volviendo al *script* `perfiles_permisos.sql`, este es un *script* SQL en el que se declaran el nombre que le es asignado a la base de datos MySQL del interior del contenedor de evaluación **y** los usuarios MySQL **PREP** y **EVAL**, a los que se le han proporcionado y retirado los permisos concretos de tal forma que **PREP** se puede considerar un usuario tipo **Admin**, con permiso para ejecutar cualquier tipo de operación y **EVAL** un usuario extremadamente restringido, con muchas limitaciones en lo que pueden y no pueden hacer, de esta forma, el sistema dispone de dos perfiles perfectos para ejecutar cualquier código seguro proporcionado por usuarios de fiar, es decir, por **Profesores** y un perfil perfecto para ejecutar código de fuentes no fiables, es decir, los Estudiantes de tal forma que se pueda garantizar que. si dichas fuentes tratan de hacer algo que podría ser dañino para el sistema, gracias a las restricciones impuestas sobre el perfil **EVAL**, no se permitirá su ejecución. En otras palabras, este *script* es el que establece los perfiles tipo-admin y restringido que el sistema utiliza para ejecutar código escrito por los profesores y código escrito por los estudiantes respectivamente.

- `db`: Al igual que el archivo `contenedores_eval` contiene una carpeta de *scripts* y un Dockerfile, solo que en este caso, el Dockerfile está diseñado para generar la imagen del contenedor Docker en el que se contendrá la propia base de datos de la aplicación (tiene como nombre **FWT**). Lo mismo pasa con la carpeta `scripts`, en el caso de `db` el *script* de la carpeta se llama `db.sql` y lo que este *script* contiene es la definición del esquema de la base de datos de la aplicación, inicializando esta si no existiese y asignándola un nombre.

- `logs`: Como el nombre indica, esta carpeta contiene los logs generados por la aplicación durante su funcionamiento.

- `static`: Esta carpeta contiene los elementos estáticos del sistema, aunque en este caso no contiene más que dos imágenes (PNG) utilizadas en las interfaces de alguna de las páginas de la web.

- `config.py`: Archivo que contiene la configuración de la aplicación Flask. Contiene información necesaria por toda la aplicación para que esta funcione correctamente.

- `requirements.txt`: Fichero de texto en el que están listadas las dependencias de la aplicación. Es de interés a la hora de desplegar la aplicación en otros dispositivos, ya que permite instalar todas las librerías de las que depende la aplicación Flask rápidamente.

- `run.py`: Archivo python responsable de la inicialización del proyecto. Su ejecución dispara la creación de la instancia de aplicación así como la ejecución del archivo `docker-compose.yml` del que se habla más en detalle en su entrada de la lista.

Con esto queda definido el nivel superior del sistema, los principales componentes del proyecto y los archivos necesarios para el despliegue de este, ahora se exponen los detalles de la estructura de directorios de la aplicación **Flask**:

- `auth`: El primero de los **Blueprints Flask [71]** del sistema. En Flask, los *blueprints* [73] son conceptos utilizados para crear componentes de aplicaciones y facilitar patrones comunes dentro de una misma aplicación **y** entre aplicaciones. Se puede pensar en un *blueprint* como en una microaplicación Flask, es una entidad en la que se agrupan rutas, plantillas, lógica de negocio, archivos estáticos e incluso gestores de errores relacionados. No obstante, es importante recordar que **no** son aplicaciones como tal, son como aplicaciones de tamaño reducido, pero no son aplicaciones por si mismas.

  El término *Blueprint* se utiliza en inglés para referirse al plano con el que se crea algo, son documentos donde se proporcionan las instrucciones para fabricar algo mediante dibujos. Por ello, puede resultar más adecuado pensar en cada *blueprint* de una aplicación Flask como los elementos con los que construir un aspecto específico de la aplicación Flask.

  El enfoque que se ha seguido con los Blueprints ha sido usarlos para separar la lógica de aplicación en función de su naturaleza, de ahí que haya tres blueprints en el proyecto, `auth` para la lógica de autenticación, `evaluación` para la lógica responsable de evaluar las respuestas de los jugadores y `main` para el resto de responsabilidades del sistema. Contiene los archivos `__init__.py` y `routes.py` que son, respectivamente, el componente que convierte a la carpeta `auth` en un paquete que puede ser importado por el resto del sistema y el archivo python donde están definidas las **rutas** en las que

se desarrolla la lógica de autenticación del sistema y, si son necesarios, junto con las rutas están definidas sus funciones de apoyo.

- `evaluación`: La segunda **Blueprint Flask**, como su nombre indica contiene lo relativo a evaluar las respuestas de los jugadores. A diferencia de las otras dos Blueprints (*auth* y *main*), no contiene solo sus respectivos archivos `__init__.py` y `routes.py`, evaluar las respuestas de los jugadores es demasiado complejo como para que sea apropiado que la lógica de aplicación relacionada esté definida en las rutas, por eso, **únicamente** en el caso de evaluación, hay un tercer archivo `helpers_evaluacion.py`, en el que están definidas todas las funciones con la lógica necesaria para realizar completamente el proceso de evaluación, preparar el entorno de evaluación (contenedor, objetos MySQL-connector, poblar entorno con el *dataset custom* de la pregunta), ejecutar la respuesta del jugador, realizar los tests y limpiar o encargarse del entorno de evaluación para liberar todos los recursos involucrados antes de la próxima solicitud de evaluación. En el caso de evaluación, `routes.py` solo tiene una ruta, la cual se encarga de coordinar y orquestar las llamadas al las funciones del antes mencionado `helpers_evaluacion.py`.

- `main`: Tercer y última **Blueprint**, sigue el mismo formato que auth, en el que contiene únicamente sus correspondientes `__init__.py` y `routes.py`. Este *blueprint* contiene el resto de la lógica de aplicación.

- `forms`: Carpeta en la que se encuentra el fichero `formularios.py`, en el cual están definidas las distintas **FlaskForm** que constituyen los formularios renderizados en las interfaces gracias a los cuales el sistema puede recibir información proporcionada por los usuarios.

- `models`: Carpeta que contiene el archivo `modelos.py` donde están definidas las clases de la aplicación y, por ello, la estructura de datos del sistema.

- `templates`: Carpeta donde se almacenan las plantillas que definen la interfaz de cada página de la aplicación web. Como puede apreciarse en la figura, salvo *home.html* el resto de las plantillas pueden relacionarse fácilmente con los distintos casos de uso de la aplicación.

- `utils`: Carpeta que contiene los denominados *utils* que vienen a ser los archivos donde está definida la funcionalidad de apoyo que resulta útil a nivel general por toda la aplicación. En concreto contiene:

  - `decorators.py`: Archivo en el que se define el único *decorator* no predefinido del sistema, `profesor_required` diseñado para incorporar a las rutas que no deban ser accedidas por usuarios estudiante la lógica para garantizar que, incluso si lograsen acceder a dichas rutas, sean inmediatamente bloqueados.
  - `lanzar_cliente.py`: Archivo con el código para crear la instancia de Cliente Docker que es utilizada a lo largo de toda la aplicación. Este es otro ejemplo de aplicación del patrón *Singleton*.
  - `lanzar_docker.py`: Archivo en el que está definida la función `setup_docker(cliente_docker)`, esta función es una de las invocadas durante la ejecución de `run.py`, se invoca inmediatamente **antes** de la llamada

a `create_app(configuración)` (la función responsable de crear la instancia de aplicación Flask) y su responsabilidad es invocar al ya mencionado `docker-compose.yml` y tras ello comprobar que la red Docker se haya desplegado debidamente así como de que el contenedor Docker en el que se encuentra la base de datos de la aplicación (llamado `c_BD` por **c**ontenedor **B**ase de **D**atos) este en funcionamiento.

- `logging_config.py`: Archivo en el que se configura y pone en funcionamiento el sistema de *logging* de la aplicación para que, en caso de que ocurran errores durante la ejecución de esta, se registren debidamente para que los responsables dispongan de los detalles sobre qué errores han ocurrido y en que punto de la ejecución.
- `validacion_respuestas.py`: Archivo en el que se definen las tres pruebas de validación que cada respuesta de jugador debe superar antes de que el sistema trate de evaluar la respuesta como tal.

  Estas pruebas consisten, de forma resumida, en parsear la respuesta como SQL usando **sqlglot** y comprobar si la respuesta contiene comentarios (primera prueba), comprobar si la respuesta contiene algún término no permitido (*blacklisted*) por la configuración del bloque de la pregunta a la que la respuesta responde (segunda prueba de validación) y la tercera y última, que consiste en tratar de obtener, mediante el comando `EXPLAIN` de MySQL, la respuesta del jugador, de esta forma se comprueban ambos, que la respuesta sea compatible con la sintaxis de **MySQL** y que el contenido de la respuesta corresponde a las limitaciones de la aplicación (*statements* DML a excepción de `SELECT`).

  Merece especial atención la segunda prueba de validación, la cual mediante el uso de **dos patrones regEx** en conjunto con los atributos de la instancia de bloque cuya pregunta está siendo respondida, permite controlar que elementos de la sintaxis SQL se le permite utilizar en su respuesta a los jugadores e incluso prohibir el uso de uno (o varios) de esos comandos **salvo** que estos vayan seguidos por otra palabra en específico, por ejemplo, es posible prohibir el uso de cualquier variante de `TABLE` y permitir, como excepción, el uso de `ALTER TABLE`, es decir, en este escenario, cualquier respuesta que contenga el string `TABLE` excepto si va precedido por `ALTER` (en el caso de las excepciones, como siempre son grupos de dos palabras, los espacios están incluidos en la excepción, por lo que al configurarlos es importante no añadir más espacios de los necesarios), es decir, si la respuesta contiene `DROP TABLE` o `CREATE TABLE`, la respuesta es declarada incorrecta sin siquiera alcanzar la etapa de Testing, sin embargo, si el string es `ALTER TABLE`, no se considera como una infracción.

  A continuación se muestran los *snippets* con la definición de los dos patrones **regEx** utilizados por la segunda prueba de validación:

```
# Patrón para "capturar" cada instancia de lo que es considerado en la
aplicación una "palabra singular válida" en el string proporcionado.
PATRON_SINGLE_WORD = re.compile(r"""
    (?:
        # Opción 1: la "incidencia" es el primer elemento de una subquery,
        el "(" va "pegado" a dicha palabra y para ser válido tiene que ir
        seguido por un espacio.

        (?<=\()           # Lookbehind: "(" a la izquierda.
        [A-Za-z]+        # Captura: La palabra detectada.
        (?=\s)          # Lookahead: Espacio (" ") a la derecha.

      |

        # Opción 2: Esta vez la "incidencia" es el último elemento de la
        subquery, el ")" está inmediatamente tras la palabra y, para ser
        valido, tiene que ir precedida por un espacio.

        (?<=\s)          # Lookbehind: Espacio (" ") a la izquierda.
        [A-Za-z]+        # Captura: La palabra detectada.
        (?=\))          # Lookahead: ")" a la derecha.

      |

        # Opción 3: La "incidencia" es el primer elemento del "string",
        por ende no tiene nada a su izquierda (límite del string), para
        ser válido tiene que ir seguida por:
        # Un espacio o ";" ya que simboliza el fin del "statement".
        No obstante, no se debería encontrar ";" porque SQLGlot
        los retira al parsear.

        (?<=^)          # Lookbehind: "Comienzo" del string a la izquierda,
        es decir, el "match" es el primer elemento del string y no tiene
        nada a su izquierda.
        [A-Za-z]+       # Captura: La palabra detectada.
        (?=\s|;)       # Lookahead: Espacio (" ") o semicolon (";")
        a la derecha.

      |

        # Opción 4: Esta última opción abarca 3 posibilidades, que la
        "incidencia" sea el último elemento del string y, por tanto, sea
        el "fin del string" y no haya nada a su derecha
        # y tenga que haber un espacio a su izquierda para ser válida o
        que sea sencillamente una palabra separada por espacios de la
        palabra anterior y posterior o el último elemento
        "del statement".
```

```
            (?<=\s)         # Lookbehind: Espacio (" ") a la izquierda.
            [A-Za-z]+       # Captura: La palabra detectada.
            (?=\s|;|$)    # Lookahead: Espacio (" ") o semicolon (";") o
            "nada" (fin del string) a la derecha.
        )
    """, re.VERBOSE)
```

```
# Patron para "capturar" cada instancia de lo que se considera en la
aplicación una "palabra compuesta valida" en el string proporcionado.
# Las "palabras compuestas" PARA NUESTRA APLICACIÓN son el conjunto
de 2 palabras "individuales" separadas por exactamente UN espacio
entre ellas. Asi que la "unidad" palabra compuesta
# es una secuencia de dos palabras con un espacio entre ellas.
PATRON_COMPOUND_WORD = regex.compile(r"""
    (?:
        # Opción 1: La palabra compuesta detectada es el primer elemento
        de una subquery, igual que en "single word" el "(" está a la
        izquierda de la
        # captura (a la izquierda de la primera palabra del "compuesto")
        y para ser válido debe tener un espacio a la derecha
        (a la derecha de la segunda palabra del "compuesto").

        (?<=\()             # Lookbehind: "(" a la izquierda.
        [A-Za-z]+\s[A-Za-z]+  # Captura: La "palabra compuesta" detectada.
        Cada captura es un conjunto de dos palabras separadas por un espacio.
        (?=\s)              # Lookahead: Espacio (" ") a la derecha.

      |

        # Opción 2: La palabra compuesta detectada es el último elemento
        de una subquery, igual que en "single word", el ")" está a la
        derecha de la captura
        # (a la derecha de la segunda palabra del "compuesto") y para ser
        válido debe tener un espacio a la izquierda de la captura
        (a la izquierda de la primera palabra del "compuesto").

        (?<=\s)             # Lookbehind: Espacio (" ") a la izquierda.
        [A-Za-z]+\s[A-Za-z]+  # Captura: La "palabra compuesta" detectada.
        Cada captura es un conjunto de dos palabras separadas por un espacio.
        (?=\))              # Lookahead: ")" a la derecha.

      |

        # Opción 3: La palabra compuesta detectada es el primer elemento del
        string, igual que en "single word", a la izquierda de la captura está
        vacio
        # (a la izquierda de la primera palabra del "compuesto") y para ser
        válido debe tener un espacio o un semicolon a la derecha
        (a la derecha de la segunda palabra del "compuesto").

        (?<=^)             # Lookbehind: "Inicio del string" a
        la izquierda.
        [A-Za-z]+\s[A-Za-z]+  # Captura: La "palabra compuesta" detectada.
        Cada captura es un conjunto de dos palabras separadas por un espacio.
```

```
            (?=\s|;)          # Lookahead: Espacio (" ") o semicolon (";")
            a la derecha.

          |

            # Opción 4: La palabra compuesta detectada está entre 2
            espacios o es el es el ultimo elemento del string o es
            el último elemento del "statement", igual que en
            "single word", a la izquierda de la captura hay un espacio
            (a la izquierda de la primera palabra del "compuesto") y a la
            derecha puede haber un espacio, un semicolon o
            "nada/vacio/fin del string".

            (?<=\s)            # Lookbehind: Espacio ("") a la izquierda.
            [A-Za-z]+\s[A-Za-z]+  # Captura: La "palabra compuesta" detectada.
            Cada captura es un conjunto de dos palabras separadas por
            un espacio.
            (?=\s|;|$)        # Lookahead: Espacio (" ") o semicolon (";") o
            "Nada/Fin del string" a la derecha.
        )
    """, regex.VERBOSE)
```

- `__init__.py`: Archivo python que establece la carpeta `app` como paquete que se puede importar en otros puntos de la aplicación y permite la correcta ejecución del código de esta carpeta.
- `docker-compose.yml`: Archivo de configuración en formato YAML para Docker Compose. Sirve para definir y orquestar múltiples contenedores (Docker), redes y volúmenes a la vez.
- `extensions.py`: Archivo python en el que se inicializan las extensiones involucradas en el correcto funcionamiento de la aplicación. En concreto estas son:

  ```
  db = SQLAlchemy()
  bcrypt = Bcrypt()
  login_manager = LoginManager()
  csrf = CSRFProtect()
  ```

  Este fragmento de código es de hecho el mejor ejemplo en el sistema del patrón **Singleton**, descrito en la Sección 6.3.5, en especial `db`, pues esta es la instancia de SQLAlchemy que se utiliza globalmente por la aplicación para garantizar que la comunicación con la base de datos de la aplicación siempre se hace mediante `db`.

Árbol de directorios

de la aplicación del proyecto

```
ROOT/
    ├── app/
    │   ├── auth/
    │   │   ├── __init__.py
    │   │   └── routes.py
    │   ├── evaluacion/
    │   │   ├── __init__.py
    │   │   ├── helpers_evaluacion.py
    │   │   └── routes.py
    │   ├── main/
    │   │   ├── __init__.py
    │   │   └── routes.py
    │   ├── forms/
    │   │   └── formularios.py
    │   ├── models/
    │   │   └── modelos.py
    │   ├── templates/
    │   │   ├── conf_bloque.html
    │   │   ├── conf_nivel.html
    │   │   ├── conf_test.html
    │   │   ├── home.html
    │   │   ├── jugar_nivel.html
    │   │   ├── login.html
    │   │   ├── menu_niveles.html
    │   │   └── sign_up.html
    │   ├── utils/
    │   │   ├── decorators.py
    │   │   ├── lanzar_cliente.py
    │   │   ├── lanzar_docker.py
    │   │   ├── logging_config.py
    │   │   └── validacion_respuestas.py
    │   ├── __init__.py
    │   ├── docker-compose.yml
    │   └── extensions.py
    ├── contenedores_eval/
    │   ├── scripts/
    │   │   └── perfiles_permisos.sql
    │   └── Dockerfile
    ├── db/
    │   ├── scripts/
    │   │   └── db.sql
    │   └── Dockerfile
    ├── static/
    │   └── img/
    ├── logs/
    ├── config.py
    ├── requirements.txt
    └── run.py
```

8.2. Arquitectura física a nivel de implementación

En esta sección se explica en detalle la arquitectura física del proyecto, es decir, los componentes que lo conforman y la manera en que se conectan y despliegan.
La Figura 8.1 muestra el diagrama de despliegue a nivel de implementación del proyecto. Inmediatamente se aprecia que es muy similar al diagrama de despliegue de diseño 6.2, aunque este no solo es más detallado y refleja con mayor fidelidad la realidad del proyecto, sino que se puede apreciar la presencia de nodos nuevos: **Máquina Host**, **Docker**, **Contenedor BD Aplicacion: c_BD** y **Contenedor Temporal BD Evaluacion**. El hecho por el que estos elementos no estaban presentes en el otro diagrama es sencillamente que representan detalles que no son relevantes para el modelo de diseño, pero como es evidente, sí que lo son para el modelo de implementación y por ello forman parte de la descripción que se va a dar a continuación.
Lo primero y más importante es entender que absolutamente **todo** se ejecuta y despliega en la **Máquina Host**, no hay ningún servidor dedicado, todo se ejecuta en el mismo dispositivo. Es importante tener esto claro, ya que aunque pueda parecer que la **Aplicación Flask**, su base de datos **FWT** y los distintos entornos de evaluación creados dinámicamente por la aplicación estén en dispositivos distintos, ese no es el caso. La razón por la que parece que sí que se dispone de servidores dedicados es precisamente **Docker** o, más en concreto, **los contenedores Docker**.
Un **Contenedor Docker** es en realidad un entorno software ligero y autónomo (*self-contained*) que empaqueta una aplicación y sus dependencias en una misma imagen (el Dockerfile y su contenido y la imagen generada a partir del Dockerfile), sin embargo, al desplegar el contenedor, este se ejecuta como un set de procesos aislados[1] en el *host* (sistema anfitrión) o, como ocurre en este escenario, ya que el sistema operativo sobre el que se ha desarrollado la aplicación es Windows 10 y Docker está basado en Linux, la máquina virtual **auxiliar** de Docker Desktop, esta máquina virtual es una VM Linux y forma parte de Docker Desktop precisamente porque Docker Desktop es la herramienta diseñada para el uso de Docker en entornos Windows, y crea un nuevo *namespace* (espacio de nombres) independiente para el contenedor (se crea un nuevo espacio de nombres para cada contenedor) para así conseguir que todo lo que esté dentro del contenedor solo perciban los ficheros, procesos e interfaces de red que le hayan sido asignados, quedando el resto de elementos del entorno del *host*, es decir, todo lo que queda fuera del contenedor, como si no existiesen, en otras palabras, Docker consigue, creando para cada contenedor su propio *namespace* no solo que el interior del contenedor tenga su propio contexto, sino que, **para los elementos dentro del contenedor**, el exterior no exista o, siendo más específicos, no se pueda ver.

Es por todo esto que **parece** que los contenedores, y por ende, las bases de datos, sean dispositivos externos, distintos del *host* en el que se ejecuta la aplicación, sin embargo, este no es el caso y es importante tenerlo en cuenta, ya que los **recursos** que utilicen la aplicación Flask, los contenedores Docker y las bases de datos dentro de dichos contenedores, **todos** vienen de la máquina *host* en la que se despliega el proyecto.
Una vez esto está claro, se procede con la explicación del diagrama:

[1] Con aislados se quiere decir que, dentro del sistema operativo del *host*, estos procesos se ejecutan en un entorno aislado del resto para que no puedan interferir con el resto de procesos del sistema operativo del *host*, o en otras palabras, para que los procesos dentro del entorno aislado no puedan interferir con nada que ocurra fuera de dicho entorno.

- Como se ha mencionado antes, todo está desplegado en la máquina *host*.

- Se puede apreciar la separación en base a funcionalidad en las distintas carpetas del nodo **Aplicación Flask**, reflejando la estructura de directorios que se ha mostrado al principio de este capítulo.

- Las bases de datos utilizadas por el proyecto vienen de dos Dockerfile distintos, uno diseñado para contener la base de datos MySQL de la aplicación y otro para crear entornos MySQL **temporales**, utilizados no para almacenar datos, sino para poder llevar a cabo lo necesario para evaluar las respuestas de los jugadores (para ejecutar los *scripts* de población, las respuestas de los jugadores y los tests de cada pregunta, es necesario disponer de un entorno MySQL en el que ejecutar dichos códigos SQL) sin poner en peligro a la aplicación ni a su propia base de datos.
  Se utiliza el adjetivo temporal porque, como ya se ha ido mencionando a lo largo del documento, estos contenedores (entornos) de evaluación son **eliminados** completamente tras haber cumplido su función, para regular el consumo de recursos del programa y cumplir con la restricción impuesta en la aplicación por la que no se permite a los jugadores hacer una solicitud de evaluación cuando ya hay una evaluación solicitada por ellos mismos en proceso.
  En el Listing 8.1 se muestra el snippet de la función responsable de crear estos contenedores de evaluación.

- Todo este sistema Docker requiere que la aplicación disponga de una Red Docker desplegada en la que poder establecer los contenedores, ya que esta Red es la que permite que estos entornos aislados se comuniquen entre ellos y/o con el exterior.

- Como se refleja en el diagrama, el archivo `docker-compose.yml` es el responsable de desplegar la Red Docker que utiliza la aplicación así como el contenedor **c_BD** y la base de datos de su interior **FWT** (la base de datos de la aplicación).

- Las carpetas `/db` y `/contenedores_eval` contienen los archivos necesarios para garantizar que los contenedores están adecuadamente configurados, cada una contiene el Dockerfile del que se genera la imagen a partir de la cual se crea el contenedor de la base de datos y los contenedores de evaluación respectivamente, mientras que las carpetas *scripts* contienen los archivos SQL que se copian en el *entrypoint* de los contendores para que sean ejecutados durante la inicialización de estos. Estas carpetas como tal no se usan frecuentemente, ya que solo son utilizadas cuando es necesario montar la imagen correspondiente para poder crear los contenedores, normalmente las imágenes estarán disponibles, pero si por algún motivo hubiesen sido eliminadas, el programa está diseñado para encargarse de ello automáticamente, en el caso de la imagen para **c_BD** docker-compose está diseñado para, si no está disponible ni el contendor ni su imagen, construir una imagen del Dockerfile correspondiente y con ella crear el contenedor c_BD, mientras que en el caso de los contenedores de evaluación, el código de la función responsable de crear estas instancias está definido para, antes de tratar de crear la instancia, comprobar que la imagen correspondiente esté disponible y, si esta no existe, la crea. En el fragmento de código 8.1 se muestra el snippet de dicha función.

- Para acceder a las bases de datos de los contenedores no es suficiente con tener desplegada la Red Docker, esta permite contactar con el contenedor como tal, pero no con

su contenido. Para esto se toman dos enfoques distintos dependiendo de cuál sea el contenedor a cuyo contenido se desea acceder:

- **Base de datos de la aplicación**: Como se ha ido mencionando a lo largo del documento, la base de datos de la aplicación (y su contenedor) es estática, en cuanto a que, durante el transcurso del servicio de la aplicación, no se elimina ni el contenedor ni la base de datos. Si ocurre es por acción externa de algún individuo con acceso físico a los archivos y componentes del proyecto, que ha eliminado el contenedor y su base de datos manualmente.
  Esta estabilidad de la base de datos permite utilizar **SQLAlchemy** para abstraer la mayor parte de la interacción entre la aplicación y la base de datos.
  SQLAlchemy está diseñado para buscar, en la configuración de la aplicación, una serie de valores y parámetros que le indican las credenciales de la base de datos, la dirección de esta y similar.
- **Bases de evaluación**: En el caso de las bases de evaluación no se puede utilizar SQLAlchemy debido a la volatilidad de estas. SQLAlchemy no está diseñado para trabajar con entornos SQL fugaces que, como ocurre en la aplicación, no existen mientras la aplicación no está en funcionamiento. Por esto se utiliza directamente **MySQL-connector**. Para que esto pueda llevarse a cabo, cada vez que la aplicación crea una instancia de contenedor de evaluación, además de crearla directamente en la red Docker del sistema, crea la instancia de forma que un puerto predefinido de la misma queda expuesto, acto seguido se invoca a una función responsable de averiguar que puerto del lado de la aplicación ha sido mapeado al puerto del contenedor y, una vez se dispone de esa información, se crean dos instancias MySQL-connector, una para el usuario MySQL **PREP** y otra para el usuario **EVAL**, de esta forma, se dispone de dos vías directas entre la aplicación y la base de datos del contenedor de evaluación.
  En el Listing 8.1 en el que se aprecia el código mediante el que se crean los contenedores de evaluación, se puede apreciar como, al crear el contenedor, se hace de forma que su puerto **3306** quede expuesto, que es lo que, como se mencionó previamente, permite una vía de acceso a través del aislamiento que Docker establece entre el interior del contenedor y el exterior.

- Como punto final de este apartado, es importante comprender que si alguno de estos elementos falta, la comunicación entre los elementos del sistema no es posible.

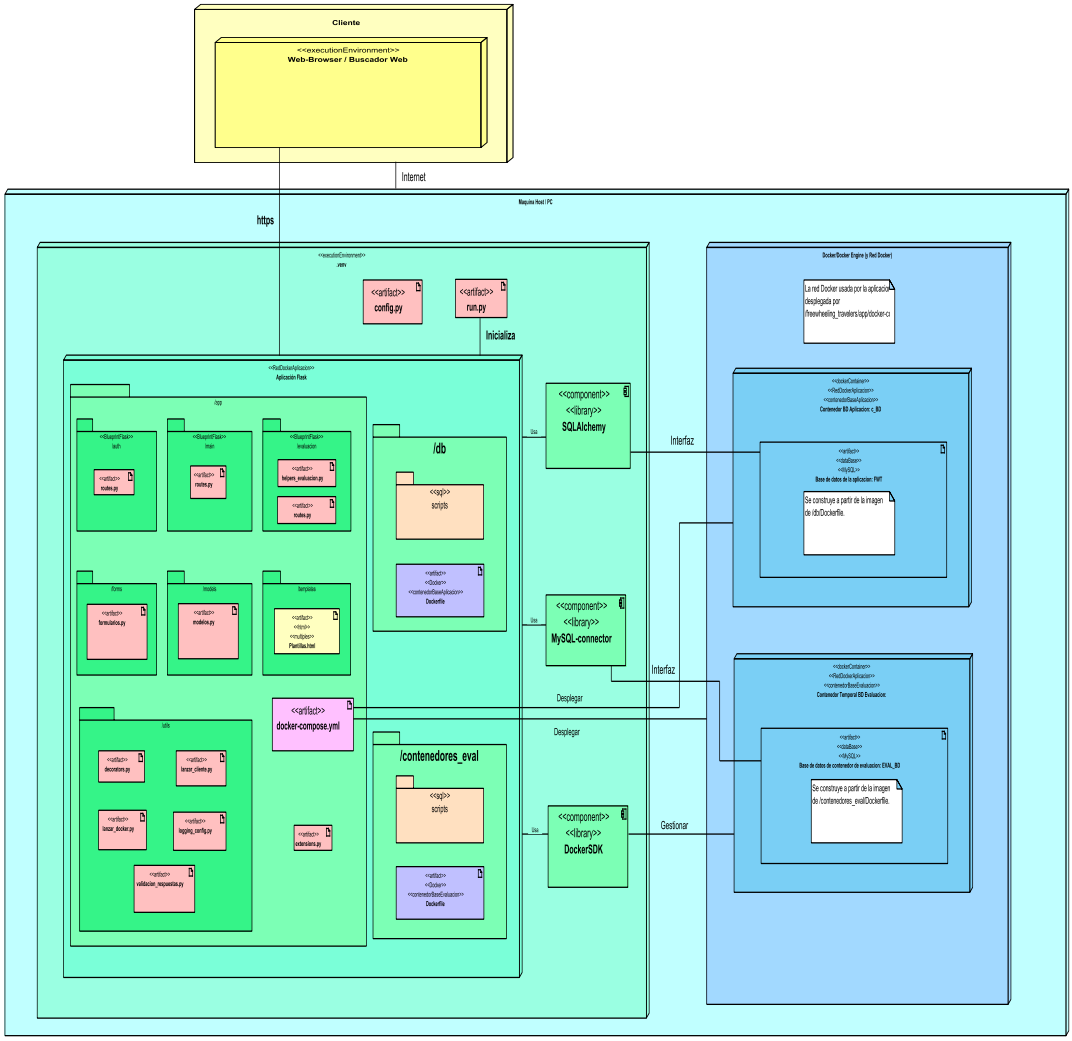

Figura 8.1: Diagrama de despliegue detallado, refleja el hardware del sistema.

```
def crear_contenedor_evaluacion(id_usuario):
    cliente_docker = current_app.config['DOCKER_CLIENT']
    image_name = current_app.config['MYSQL_EXEC_IMAGE']
    nombre_red = current_app.config['DOCKER_NETWORK_NAME']
    nombre_contenedor = f"exec_{id_usuario}_container"

    limite_intentos = 10
    delay_intentos = 2

    contenedores_existentes =
        cliente_docker.containers.list(all=True)
    if any(contenedor.name == nombre_contenedor for contenedor in
        contenedores_existentes):
        logger.info(f"Ya hay un contenedor {nombre_contenedor} en
            uso en la aplicacion, espera un poco y si este mensaje
            sigue apareciendo cierra el programa y contacta con un
            administrador.")
        mensaje_error = f"Ya existe un contenedor a tu nombre en
            el sistema. Evaluar respuestas puede llevar un poco de
            tiempo.\nEspera un poco para que la aplicacion pueda
            terminar de evaluar tu respuesta antes de volver a
            solicitar que se evalue.\nSi tras unos minutos no has
            recibido respuesta o tu solicitud anterior ya fue
            respondida antes de la actual, informa al tutor."
        return (False, mensaje_error)

    try:
        cliente_docker.images.get(image_name)
        print(f"Imagen '{image_name}' ya existe, no se vuelve a
            construir.")
    except ImageNotFound:
        print(f"Imagen '{image_name}' no encontrada.
            Construyendo...")
        try:
            image, logs = cliente_docker.images.build(
                path="contenedores_eval/",
                dockerfile="Dockerfile",
                tag=image_name,
                rm=True,
                pull=False
            )

            for chunk in logs:
                if 'stream' in chunk:
                    print(chunk['stream'].strip())
            print(f"Imagen '{image_name}' construida con ID
                {image.id}.")
        except BuildError as e:
            print("Error al construir la imagen:", e)
            raise
```

```
try:
    contenedor = cliente_docker.containers.run(
        image_name,
        name=nombre_contenedor,
        network=nombre_red,
        ports={'3306/tcp': None},
        detach=True
    )
    contenedor.reload()
    print(f"Se ha creado el contenedor Docker:
        {contenedor.name}.")

    for intento in range(limite_intentos):
        print(f"Va a comenzar el intento n {intento+1} de
            averiguar el puesto host.")
        contenedor.reload()
        port_bindings =
            contenedor.attrs['NetworkSettings']['Ports'].
            get('3306/tcp')
        if port_bindings:
            puerto_host = port_bindings[0]['HostPort']
            print(f"Se ha averiguado el puerto host en el
                intento {intento+1}. Es: {puerto_host}")
            break
        time.sleep(delay_intentos)
    else:
        print(f"Se ha accedido al 'else' del pooling para
            obtener el puerto host, eso quiere decir que no se
            ha logrado asignar un puerto del host al puerto
            3306 del contenedor.")
        mensaje_error = f"Seguimos con problemas de se ha
            expuesto 3306 pero no se le ha mapeado nada."
        return (False, mensaje_error)

    return (True, contenedor, puerto_host)
except docker.errors.APIError as e:
    print(f"Error al tratar de crear un contenedor para
        llevar a cabo la evaluacion. Error = {e}.")
    logger.error(f"Error durante la creacion de contenedor
        para la solicitud del usuario {id_usuario}. Error =
        {e}.")
    logger.error("AQUI!")
    raise
```

Listing 8.1: *Snippet* del método responsable de crear las instancias de **Contenedor de Evaluación**.

8.3. Detalles de implementación

Por último, antes de finalizar el capítulo, se van a explicar algunos detalles respecto a las tecnologías involucradas en el proyecto que no han sido mencionados hasta ahora:

8.3.1. Docker-Compose

Como se mencionó en la sección de herramientas y tecnologías, en este sistema **Docker-Compose** se utiliza para, durante el proceso de inicialización del proyecto, desplegar la **Red Docker** que permitirá la comunicación entre la aplicación y su base de datos (el contenedor Docker que la contiene) y servirá de entorno en el que la aplicación podrá crear y desplegar (cuando sea el momento) los **contenedores de Evaluación**. También se encarga de desplegar el antes mencionado contenedor de la base de datos de la aplicación.

Cabe mencionar que aunque la red Docker del sistema es necesaria para el correcto funcionamiento de los contenedores de Evaluación y que esta es desplegada por **Docker-Compose**, Compose **no** está involucrado en la creación de los contenedores de evaluación, estos son responsabilidad completa de la aplicación Flask.

8.3.2. SQLAlchemy

Más allá de lo que se ha mencionado en el capítulo de herramienta y tecnologías respecto a como SQLAlchemy actúa como un *ORM* (*Data Mapper*) y facilita la comunicación entre la aplicación y su base de datos, es también responsable de poblar la base de datos de la aplicación. Como se ha mencionado previamente es un **Data Mapper** y se encarga de mapear los modelos definidos en Python y generar el equivalente en formato MySQL en la base de datos (las tablas y sus columnas y filas).

Esto empieza con la línea:

```
db.create_all()
```

Dentro del archivo __init__.py de la carpeta app se ejecuta esa línea, con ello lo que se hace es, en resumen, ordenar a SQLAlchemy crear en la base de datos (en el lado MySQL) las tablas correspondientes a cada clase definida en el archivo `modelos.py` a la que se la haya establecido el parámetro `db.Model` con el que se declara a SQLAlchemy que la clase asociada es una tabla.

A continuación se muestra un snippet con la definición de la clase **Bloque** de la aplicación:

```
class Bloque(db.Model):
    __tablename__="bloque"

    # Atributos de la clase.
    id_Bloque = db.Column(db.Integer, primary_key=True)

    nombre_Bloque = db.Column(db.String(20), nullable=False)

    contexto = db.Column(db.Text, nullable=False, default='Narrativa de la
    instancia del bloque.')

    rol = db.Column(db.Enum(Roles), nullable=False)

    id_BloquePadre = db.Column(db.Integer, db.ForeignKey('bloque.id_Bloque'),
    nullable=True)

    id_Nivel = db.Column(db.Integer, db.ForeignKey('nivel.id_Nivel'),
    nullable=False)

    id_Autor = db.Column(db.Integer, db.ForeignKey('usuario.id_Usuario',
    ondelete='SET NULL'), nullable=True)

    opcion = db.Column(db.String(30), nullable=False, default='comun')

    script_poblar_BD = db.Column(db.Text, nullable=False,
    default='Scripts para base de datos temporal en caso de
    que el bloque sea de rol pregunta.')
```

```
usa_excepciones = db.Column(db.Boolean, nullable=False, default=False)

blacklist = db.Column(db.Text, nullable=False, default='CREATE,ALTER,DROP,
TRUNCATE,RENAME,COMMENT,DESCRIBE,EXPLAIN,GRANT,REVOKE,SET ROLE,
START TRANSACTION,COMMIT,ROLLBACK,SAVEPOINT,RELEASE SAVEPOINT,LOCK TABLES,
UNLOCK TABLES,SHOW,USE,SET,FLUSH')

excepciones = db.Column(db.Text, nullable=False, default='')

script_obtener_results = db.Column(db.Text, nullable=False,
default='Si la pregunta no se resuelve mediante statements que generen
por si mismos un result set escribe aqui la query que genere un result
set que "capture" el set de datos en el que DEBERIAN percibirse los
"efectos" de la respuesta del jugador en caso de ser correcta.')

# Relaciones de la clase.
dir_nivel = db.relationship('Nivel', back_populates='bloques')

autorBloque = db.relationship('Usuarios', back_populates='bloques_creados')

bloquePadre = db.relationship('Bloque', remote_side=[id_Bloque],
backref=db.backref('hijos', cascade='all, delete-orphan'))

respuestas = db.relationship('Respuesta', back_populates='pregunta')

tests = db.relationship('Test', back_populates='bloque',
passive_deletes=True)

def __init__(self, **kwargs):
    super(Bloque, self).__init__(**kwargs)
    if not self.nombre_Bloque:
        self.nombre_Bloque = f"{self.id_Nivel}-{self.id_Autor}-RAIZ"
```

En el snippet se puede apreciar tanto el antes mencionado `db.Model` como el proceso mediante el que se define qué tipo de datos será la columna correspondiente a cada atributo, qué restricciones se aplicarán y similar.

8.3.3. MySQL-connector

A lo largo del documento se han comentado distintos aspectos de MySQL-connector y su uso en la aplicación, pero se va a aprovechar esta sección para agruparlos. MySQL-connector es crucial para el correcto funcionamiento del sistema por dos motivos principales:

1. Proporciona a la aplicación el protocolo de comunicación con entornos MySQL, SQLAlchemy no dispone de su propio protocolo, así que es necesario obtenerlo de otra fuente como, en el caso de este proyecto, MySQL-connector.

2. En el caso de los contenedores de evaluación, ya que la volatilidad de estos imposibilita el uso de SQLAlchemy para conectar a sus bases de datos, se utiliza directamente MySQL-connector como interfaz de bajo nivel.

8.3.4. Werkzeug

Como ya se mencionó en la correspondiente entrada del capítulo de tecnologías y herramientas, **Werkzeug** es un conjunto de librerías (o *toolkit*) enfocado en **WSGI** para Flask-Python. En otras palabras, Werkzeug es una librería de utilidad (*utility library*) para Python que permite y facilita crear aplicaciones web Python **compatibles** con la **especificación** WSGI. Sí que conviene aclarar que Werkzeug **no es** un *framework web* como tal, pero muchos *frameworks web* populares dependen de Werkzeug, por ejemplo **Flask**, **Quart** y **FastAPI**.

Como se acaba de explicar, Flask está basado en Werkzeug, es decir, depende de forma innata del *toolkit*. Werkzeug proporciona a Flask, o visto de otra forma, Flask lo utiliza para gestionar las solicitudes (peticiones) y respuestas HTTP, gestionar el enrutamiento y, de gran importancia, Werkzeug es la fuente del servidor de desarrollo en el que se despliega la aplicación Flask[2].

En resumen, **Werkzeug** es un componente **crucial** del proyecto, ya que es lo que permite a Flask, gestionar las solicitudes y respuestas HTTP, que el sistema de enrutamiento funcione[3] y proporciona el servidor sobre el que se está desplegando la aplicación actualmente (por no haber abandonado la etapa de desarrollo).

Como curiosidad, el nombre *Werkzeug* le fue asignado por significar esta palabra herramienta en Alemán, para destacar el propósito de Werkzeug como "caja de herramientas para desarrollo web".

[2]Por la naturaleza del proyecto como Trabajo de Fin de Grado no se ha considerado necesario incorporar un servidor WSGI dedicado, ya que para el alcance planeado para el proyecto el servidor de desarrollo proporcionado por Werkzeug es más que suficiente.

[3]Si se desea entrar en detalles, Werkzeug es el responsable de, en el enrutamiento, cada vez que llega una solicitud, determinar cuál de todas las rutas (funciones con el decorador **@routes**) es la que corresponde con la URL en cuestión y, una vez encontrada, proporciona la información relevante a Flask para que este invoque a la ruta en cuestión.

Capítulo 9

Pruebas

Se define como **pruebas de software** al proceso mediante el cual se evalúa y verifica que un producto o aplicación *software* hace lo que debe hacer y la calidad con que es capaz de hacerlo [6]. En otras palabras, cada uno de los distintos procesos para evaluar la calidad y correcto funcionamiento de cualquier *software* es una prueba de software.

Consisten en examinar distintos aspectos del *software* que se esté probando para verificar que su comportamiento se ajuste a los requisitos establecidos y también para buscar errores y variaciones en el comportamiento establecido[1] para mejorar la calidad del producto *software*. Se llevan a cabo ejecutando la aplicación bajo condiciones y escenarios específicos con el objetivo de poder probar el correcto funcionamiento de la aplicación en distintos niveles de detalle, tamaño y alcance[2].

Existen muchos tipos o niveles distintos de prueba de *software*, pero en este Capítulo, por su relevancia durante el desarrollo de este proyecto en concreto, se van a definir y detallar las **pruebas unitarias** y las **pruebas de sistema**.

9.1. Pruebas Unitarias

Según *IBM*, las **pruebas unitarias** son aquellas pruebas de software que validan que cada unidad de software funciona según lo esperado, entendiendo como **unidad** al componente comprobable más pequeño de una aplicación. En otras palabras, son las pruebas mediante las cuales se verifica el correcto funcionamiento de cada función, método y similares (la unidad más pequeña de código).

Se dice de este tipo de pruebas que son "de bajo nivel" [14] porque verifican las unidades de

[1] Con el fin de corregir estos errores y desviaciones.

[2] Con esto se está hablando de aspectos como la granularidad, el aislamiento y similares. No se consideran del mismo nivel una prueba unitaria en la que el elemento puesto a prueba es la unidad lógica más pequeña y se examina su funcionamiento por sí mismo, con el menor grado de dependencias posible, que una prueba de integración cuyo objetivo es precisamente comprobar que dos componentes distintos no tengan problema alguno en coexistir en el sistema y sus interacciones sean las diseñadas.

código más pequeñas y lo hacen en un entorno con el máximo grado posible de aislamiento, de forma que el único elemento involucrado en la prueba es la propia unidad o, dicho de otra forma, de forma que se garantice la menor dependencia posible de elementos externos (cualquier elemento que no sea la unidad a prueba).

Por estos motivos, este tipo de pruebas son muy numerosas, frecuentes y, en la mayoría de los casos, superan en cantidad al resto de pruebas de software. De hecho, es tal la cantidad de pruebas unitarias que se llevan a cabo en proyectos de dimensiones como este TFG, que tratar de documentar todas y cada una de ellas resulta extremadamente laborioso y, en un documento como el actual, de muy poca utilidad. Por ello, aunque no se han documentado las pruebas unitarias de este proyecto, sí que se desea mencionar que, durante la implementación de la lógica de la aplicación, es decir, durante la definición del código de la aplicación, se realizaron varias de estas pruebas cada vez que se incorporaba al sistema una de estas **unidades lógicas**.

9.2. Pruebas de Sistema

Se definen las **pruebas de sistema** como las pruebas de software que verifican que la totalidad del sistema, una vez este está plenamente integrado, se comporta tal y como es esperado, cumpliendo con sus requisitos y especificaciones [5]. En otras palabras, son el conjunto de pruebas realizadas una vez se ha desarrollado una versión completa[3] (en cuanto a funcionalidad, requisitos y especificaciones) para verificar que el sistema funciona tal y como debe.

Durante el desarrollo de la aplicación se prestó especial atención al diseño de la lógica, antes de implementar el código, para garantizar que la lógica diseñada fuese **sólida**, **coherente** y se ajustase perfectamente a los **Casos de Uso** establecidos para el proyecto, de forma que, una vez se incorporase al proyecto, no hiciese falta modificar o corregir la lógica por no cumplir con los requisitos impuestos por los Casos de Uso.

Gracias a dar tal importancia al diseño apropiado de la lógica, a la hora de probar la funcionalidad de la aplicación a medida que la lógica correspondiente a los distintos casos de uso era incorporada al sistema, la cantidad de pruebas necesarias fue mínima, ya que, al dar tal prioridad al diseño de la lógica, al completar la implementación de los distintos casos de uso, estos funcionaban tal y como se establece en la especificación de requisitos y por ello, ninguna de las pruebas de sistema documentadas[4] obtuvo un resultado distinto al esperado.

Merece la pena mencionar que, tal y como establece la definición de **prueba de sistema**, estas pruebas se llevan a cabo cuando **todos** los componentes del sistema están integrados plenamente en el sistema, es decir, una vez el desarrollo de la aplicación ha finalizado, no obstante, aunque las pruebas de sistema hayan dado los resultados deseados y no haya sido necesario modificar la lógica de la aplicación, conviene tener en cuenta que, como ha sido mencionado en la sección anterior, durante la implementación de los segmentos del sistema que encapsulan a cada uno de los casos de uso establecidos, se realizaron numerosas **pruebas unitarias** para garantizar el correcto funcionamiento.

[3] Aquí **completa** se refiere a que se trata de una versión del sistema que proporciona toda la funcionalidad planeada, no quiere decir que sea la versión final de dicho sistema.

[4] Una prueba por cada caso de uso

Las pruebas de sistema mediante las que los casos de uso del proyecto son puestos a prueba es especifican en las siguientes tablas: Tabla 9.1, Tabla 9.2, Tabla 9.3, Tabla 9.4, Tabla 9.5, Tabla 9.6, Tabla 9.7 y Tabla 9.8.

9.2.1. Prueba *Registrar usuario*

Caso de uso a prueba:	Registrarse en la aplicación.
Prueba realizada:	Se ha desplegado la aplicación y utilizado el servicio web para crear **dos** cuentas nuevas de usuario, una de **Usuario Profesor** y otra de **Usuario Estudiante**.
Resultado Esperado:	En el sistema hay dos usuarios nuevos (reflejado en la base de datos de la aplicación) cuyos atributos coinciden con los proporcionados al sistema.
Resultado Obtenido:	Se crearon los dos usuarios nuevos sin problema.
Trazabilidad:	Prueba de sistema del caso de uso 4.1.

Tabla 9.1: Prueba del caso de uso "Registrar usuario".

9.2.2. Prueba *Iniciar sesión*

Caso de uso a prueba:	Iniciar sesión.
Prueba realizada:	Se ha tratado de iniciar sesión utilizando las credenciales de instancias de ambos tipos de usuario.
Resultado Esperado:	Se inicia sesión y el usuario es dirigido a la vista home correspondiente a la de un usuario identificado.
Resultado Obtenido:	Se alcanza la página home para un usuario identificado.
Trazabilidad:	Prueba de sistema del caso de uso 4.2.

Tabla 9.2: Prueba del caso de uso "Iniciar sesión".

9.2.3. Prueba *Cerrar sesión*

Caso de uso a prueba:	Cerrar sesión en la aplicación.
Prueba realizada:	Con la aplicación desplegada y la sesión iniciada, se ha tratado de cerrar sesión.
Resultado Esperado:	Se cierra la sesión del usuario, siendo llevado de vuelta a la página home, pero a la versión para usuarios aún por identificarse.
Resultado Obtenido:	Se cerró sesión y se alcanzó la antes mencionada página.
Trazabilidad:	Prueba de sistema del caso de uso 4.3.

Tabla 9.3: Prueba del caso de uso "Cerrar sesión".

9.2.4. Prueba *Acceder al menú de niveles*

Esta prueba **no** está relacionada con un caso de uso. El sistema no proporciona los mismos servicios a un Profesor que a un Estudiante y, a la hora de acceder al listado de niveles del sistema, aunque ambos disfrutan de ese servicio, este no es proporcionado de la misma forma. Los profesores tienen acceso a todos los servicios que el sistema es capaz de proporcionar, y a la hora de acceder al listado de niveles, se les muestran **todos** los niveles del sistema, independientemente de si estos están listos para ser usados o no. Mientras que, a los estudiantes, al no disponer de acceso a los servicios relacionados con la manipulación y gestion de niveles y sus componentes, pudiendo únicamente jugar a aquellos niveles listos para ser jugados, el sistema solo muestra a los estudiantes los niveles cuyo atributo `estaListo` tiene como valor `True`.

Aunque la relevancia de este evento no es suficiente como para considerlo un caso de uso, es un aspecto del sistema que ha sido explícitamente puesto a prueba y por ello, documentado.

Caso de uso a prueba:	Acceder al menú niveles.
Prueba realizada:	Consiste en dos pruebas, se ha iniciado sesión como un usuario Estudiante y un usuario Profesor (primero se hizo la primera iteración y luego la segunda), y se ha tratado de acceder al menú de niveles con ambos usuarios.
Resultado Esperado:	Se alcanza la página del menú de niveles donde, si el usuario es un Estudiante, el listado de niveles solo muestra aquellos marcados como *listo para jugar*, y en el caso del usuario Profesor, **todos** los niveles del sistema.
Resultado Obtenido:	Ambos usuarios han logrado alcanzar el menú de niveles, y los niveles mostrados son los correspondientes al rol del usuario.
Trazabilidad:	Prueba de sistema del correcto funcionamiento a la hora de mostrar a los usuarios el listado de niveles. No existe caso de uso asociado.

Tabla 9.4: Prueba del caso de uso "Acceder al menú niveles".

9.2.5. Prueba *Jugar un nivel*

Caso de uso a prueba:	Jugar un nivel(hasta completarlo).
Prueba realizada:	Se ha creado un nivel listo para ser jugado con 4 ramificaciones y se ha confirmado que la configuración era correcta, tras esto, usando tanto un usuario Profesor como uno Estudiante, se ha tratado de jugar el nivel hasta completarlo, la prueba se ha repetido otras 3 veces para garantizar que las tres ramas restantes funcionan también debidamente.
Resultado Esperado:	El *gameplay* del nivel es, tal como se planeó y se puede jugar hasta completarlo.
Resultado Obtenido:	Se pueden jugar los niveles tal y como se planea.
Trazabilidad:	Prueba de sistema del caso de uso 4.4.

Tabla 9.5: Prueba del caso de uso "Jugar un nivel".

9.2.6. Prueba *Administrar nivel*

Caso de uso a prueba:	Administrar un nivel.
Prueba realizada:	Se ha creado, configurado y eliminado una instancia de Nivel. Se ha comprobado el contenido de la base de datos entre cada acción para verificar que se crea, modifica y elimina la instancia en cuestión.
Resultado Esperado:	Las acciones realizadas con la/s instancia/s de nivel se llevan a cabo correctamente.
Resultado Obtenido:	El sistema de administración de niveles funciona perfectamente.
Trazabilidad:	Prueba de sistema del caso de uso 4.5.

Tabla 9.6: Prueba del caso de uso "Administrar nivel".

9.2.7. Prueba *Administrar bloque*

Caso de uso a prueba:	Administrar un bloque.
Prueba realizada:	Se ha creado, configurado y eliminado una instancia de Bloque. Se ha comprobado el contenido de la base de datos entre cada acción para verificar que se crea, modifica y elimina la instancia en cuestión.
Resultado Esperado:	Las acciones realizadas con la/s instancia/s de bloque se llevan a cabo correctamente.
Resultado Obtenido:	El sistema de administración de bloques funciona perfectamente.
Trazabilidad:	Prueba de sistema del caso de uso 4.6.

Tabla 9.7: Prueba del caso de uso "Administrar bloque".

9.2.8. Prueba *Administrar Test*

Caso de uso a prueba:	Registrarse en la aplicación.
Prueba realizada:	Se ha creado, configurado y eliminado una instancia de Test. Se ha comprobado el contenido de la base de datos entre cada acción para verificar que se crea, modifica y elimina la instancia en cuestión.
Resultado Esperado:	Las acciones realizadas con la/s instancia/s de test se llevan a cabo correctamente.
Resultado Obtenido:	El sistema de administración de tests funciona perfectamente.
Trazabilidad:	Prueba de sistema del caso de uso 4.7.

Tabla 9.8: Prueba del caso de uso "Administrar test".

Capítulo 10

Conclusiones

En este capítulo se presentan las conclusiones alcanzadas durante el transcurso del TFG, y el apartado de **Trabajo Futuro** donde se habla de posibles aspectos de mejora y extensión de la aplicación del proyecto si se decidiese llevarlo a producción.

10.1. Conclusiones

Alineadas con los objetivos, las principales conclusiones de este trabajo son:

- El estudio sobre la gamificación se llevó a cabo mediante múltiples fuentes. Los que proporcionaron una mejor visión conceptual de la gamificación fueron los libros [18] y [74], especialmente, el último, que resultó de hecho una lectura interesante.
- Todos los objetivos relacionados con la aplicación han sido alcanzados, la aplicación es plenamente funcional y los relativos a aprender a utilizar las herramientas y profundizar conocimientos fueron intrínsecos al desarrollo de la aplicación.
- Los requisitos de seguridad también han sido alcanzados, la aplicación no solicita datos personales de los usuarios, así que no es necesario preocuparse en ese aspecto, y luego información crucial como las contraseñas de cada cuenta están debidamente encriptadas.
- El objetivo de conseguir que la actividad resulte interesante para los alumnos no ha sido validado, ya que lamentablemente no ha sido posible hacer una demostración para ser probada. Aunque desde el punto de vista del autor del documento, que es un estudiante, parece razonable considerar el objetivo como alcanzado, ya que, en su experiencia personal, los sistemas basados en *quizzes* (o cuestionarios) siempre le han resultado entretenidos.
- Por último, el objetivo sobre garantizar la correcta evaluación se considera alcanzado ya que, siempre y cuando los autores de las distintas preguntas respeten las limitaciones

del sistema, la aplicación es capaz de determinar si la respuesta es correcta o incorrecta, incluso proporcionando *feedback* cuando es incorrecta.

Las limitaciones en cuestión se resumen en la siguiente frase: La pregunta debe estar diseñada de forma que su problema se solucione mediante *statements* **DML**. Conviene mencionar que `SELECT`, aunque es frecuentemente mencionado junto al resto de operaciones **DML** (`CREATE`, `DELETE`, `UPDATE`...), es en realidad una operación **DQL**. En caso de que sea necesaria la aclaración: **DML** viene de *Data Management Language* o Lenguaje de Gestion de Datos, mientras que **DQL** viene de *Data Query Language* o Lenguage de Consulta de Datos.

10.2. Trabajo Futuro

Durante el desarrollo del proyecto se han podido identificar áreas de mejora del proyecto y secciones en las que se podría ampliar la funcionalidad de la aplicación. Por restricciones de tiempo y de recursos, estas mejoras no se han podido llevar a cabo y por ello quedaron fuera del alcance del proyecto, sin embargo, se van a mencionar en caso de que se desease, en el futuro, llevar el proyecto a producción y no tratarlo como un mero prototipo.

- La aplicación incorpora aspectos de gamificación suficientes como para obtener sus beneficios, sin embargo, existen muchos más componentes de la gamificación que podrían haber sido incorporados al proyecto y, potencialmente, habrían generado mayores beneficios aún. Componentes como; sistemas de puntuación y *leaderboard*, aunque no es del todo recomendable en una actividad de soporte, ya que, especialmente la *leaderboard*, pueden tener efectos contraproducentes en estudiantes de naturaleza menos competitiva. También se podría considerar algún tipo de chat global para poder comunicarse con otros jugadores, un **sistema de logros** sería especialmente efectivo, ya que, por un lado, en colaboración con el sistema de chat podría utilizarse para despertar el interés de los estudiantes más competitivos si los logros de otros usuarios pudiesen verse, de esta forma los estudiantes competitivos se ven incentivados en tratar de conseguir todos los logros posibles, o los más raros, mientras que los usuarios menos competitivos siguen recibiendo el refuerzo positivo de los logros obtenidos durante el transcurso natural de los niveles de la aplicación. Este último punto vería aún más beneficios si se incluyesen logros ocultos para los cuales no se revela la condición de obtención y despiertan la curiosidad de los usuarios. Como se ha dicho, los ejemplos son múltiples y podrían incluirse más, pero se va a cerrar el apartado hablando sobre uno que estuvo planeado desde etapas tempranas del proyecto, pero que al final no fue posible incorporar. Se trata de proporcionar un avatar detallado a los usuarios.

  En la implementación actual, la idea es que los profesores escriban la narrativa de los niveles en segunda persona, para que los jugadores se pongan en la piel del protagonista de la historia y facilitar la inmersión de estos en la trama, sin embargo, esto podría llevarse más allá permitiendo a los usuarios personalizar su perfil con avatares *customizables* (personalizables) y escribiendo detalles sobre ellos mismos como gustos y objetivos y podría incorporarse al proyecto un mecanismo para que, en los niveles, el protagonista de ellos sea dicho avatar, sin embargo, idear la lógica y código para hacer funcionar tal sistema quedaba muy fuera del alcance del proyecto.

- También relacionado con la gamificación, pero relacionado con el dominio del diseño gráfico, la aplicación se beneficiaría de algún sistema para poder incorporar imágenes en el fondo de las preguntas (como mejora de nivel básico si se pudiese hacer la aplicación tener el nivel visual de una aventura gráfica sería mejor aún) relacionadas con los eventos que transcurren en la trama del nivel, esto mejoraría la experiencia de juego, sin embargo, los conocimientos del autor del proyecto en ese campo no son los suficientes como para incorporarlo a tiempo.

- Un aspecto que, si se quisiera dar uso oficial a la aplicación, habría que cambiar es el actual sistema de creación de cuentas, ya que por utilidad como proyecto TFG, cualquier usuario puede crearse una cuenta, en la práctica habría que cambiar la lógica de la aplicación para que solo los profesores fuesen capaces de crear y eliminar usuarios mientras que los estudiantes solo podrían iniciar y cerrar sesión. A nivel de implementación sería suficiente desplazar la funcionalidad de registrarse como usuario (o crear nuevos usuarios) al entorno en el que el usuario tiene que haberse identificado, añadir una funcionalidad de eliminación de usuarios y aplicar a ambas funcionalidades el *decorator* `@profesor_required` para garantizar que los usuarios sin identificar no puedan crear cuentas y que esta funcionalidad solo esté disponible para usuarios identificados cuya cuenta tiene como valor del atributo **esProfesor** `True`. Sin embargo, esto quiere decir que las cuentas de Profesor, o al menos la primera de ellas, deberían introducirse en el sistema directamente desde la base de datos mediante algún tipo de herramienta apta para ello, como **MySQL Workbench**.

- Otro aspecto que se podría extender es el espectro de preguntas que se pueden hacer mediante la aplicación. Como se ha mencionado antes, actualmente la aplicación solo funciona si la pregunta se soluciona mediante respuestas **DML** puras, pero lo ideal sería poder evaluar todo el espectro, incluyendo **DDL**, **DCL** y **DQL**. Respecto a **DQL** o su principal representante, `SELECT`, se podría incorporar un sistema mediante el cual, mediante un *script*, se lean los resultados y metadatos de la ejecución de la respuesta del estudiante, para poder insertar, en el entorno de evaluación, una tabla que coincida con los elementos seleccionados por la respuesta del jugador. Si se le da siempre un nombre predeterminado a dicha tabla entonces ya sería suficiente pues, mediante algún atributo en la instancia de Bloque a la que corresponda la pregunta, se podría señalar al sistema que, en vez de ejecutar los tests sobre la base de datos, se van a realizar en concreto sobre la tabla con el `SELECT` del estudiante, y una vez establecido, el profesor tan solo tendría que diseñar los tests teniendo eso en cuenta.

  Para incorporar la posibilidad de hacer preguntas sobre **DDL** y **DCL**, técnicamente, bastaría con cambiar el funcionamiento del sistema de filtrado del proyecto. Actualmente, existen dos puntos del sistema en los que se filtran operaciones indeseadas, la correspondiente prueba de validación y durante el despliegue de los contenedores de evaluación, ya que el script que define la base de datos de dichos contenedores esta diseñado para que el usuario **EVAL**, el usado para ejecutar las respuestas de los usuarios, no tenga permitido realizar la mayoría de las operaciones potencialmente peligrosas. Sería necesario unificar estos sistemas de filtrado en un único sistema, de forma que los filtros se establezcan para cada pregunta. De esta forma, sería simple diseñar los tests para este tipo de preguntas, pues bastaría con aplicar la misma lógica a la hora de examinar las respuestas, comprobando si la respuesta del estudiante hace el cambio solicitado o crea la estructura deseada. Aunque esto también incrementa las responsa-

bilidades de los autores, ya que deben ahora también prestar especial cuidado a la hora de seleccionar que permisos son proporcionados a los estudiantes.

Es necesario considerar el riesgo innato que implica el permitir a los jugadores tener privilegios DDL y DCL,especialmente DCL, ya que con `GRANT` y `REVOKE` el jugador puede proporcionarse más privilegios, así como retirarlos.

- El aspecto visual del proyecto podría mejorarse. La calidad de las interfaces del sistema, al menos en medida de estética, no es uno de los objetivos del proyecto.
- Para llevar el proyecto a producción, sería necesario buscar un servidor WSGI dedicado, al menos para alojar la base de datos de la aplicación, ya que en la implementación actual se utiliza el servidor nativo de **Werkzeug**, pero este no es apto una vez pasada la etapa de desarrollo. Habría que comparar datos para escoger la mejor opción, ya que existen varias alternativas como **uWSGI** y **Gunicorn**, siendo esta última especialmente popular cuando se trabaja con Flask.

Apéndice A

Resumen de enlaces adicionales

Los enlaces útiles de interés en este Trabajo Fin de Grado son:

- Repositorio del código: `https://github.com/helfern/proyecto-tfg`.

Bibliografía

[1] *¿Qué es el patrón Modelo Vista Controlador?* Mar. de 2022. URL: https://www.geeksforgeeks.org/software-engineering/mvc-framework-introduction/#components-of-mvc.

[2] *Artículo de GeeksforGeeks sobre acoplamiento.* Jul. de 2018. URL: https://www.geeksforgeeks.org/software-engineering-coupling-and-cohesion/.

[3] *Artículo de GeeksforGeeks sobre patrones de arquitectura software.* Oct. de 2023. URL: https://www.geeksforgeeks.org/design-patterns-architecture/.

[4] *Artículo de IBM sobre Jinja2.* Ene. de 2025. URL: https://www.ibm.com/docs/es/gdp/11.4.0?topic=sdk-jinja2-templates.

[5] *Artículo de IBM sobre las pruebas de sistema.* 11 de abr. de 2025. URL: https://www.ibm.com/think/topics/system-testing.

[6] *Artículo de IBM sobre las pruebas de software.* Ago. de 2021. URL: https://www.ibm.com/es-es/think/topics/software-testing.

[7] *Artículo de Wikipedia sobre Aventura Conversacional.* Page Version ID: 164272677. Dic. de 2024. URL: https://es.wikipedia.org/w/index.php?title=Aventura_conversacional&oldid=164272677.

[8] *Artículo de Wikipedia sobre Aventura Gráfica.* Page Version ID: 166020234. Mar. de 2025. URL: https://es.wikipedia.org/w/index.php?title=Aventura_gr%5C%C3%5C%A1fica&oldid=166020234.

[9] *Artículo de Wikipedia sobre Choose Your Own Adventure.* Page Version ID: 1294727533. Jun. de 2025. URL: https://en.wikipedia.org/w/index.php?title=Choose_Your_Own_Adventure&oldid=1294727533.

[10] *Artículo de Wikipedia sobre Librojuegos.* Page Version ID: 167441054. Mayo de 2025. URL: https://es.wikipedia.org/w/index.php?title=Librojuego&oldid=167441054.

[11] *Artículo online en el que se explica el patrón capas y sus ventajas y desventajas.* Nov. de 2024. URL: https://dev.to/yasmine_ddec94f4d4/understanding-the-layered-architecture-pattern-a-comprehensive-guide-1e2j.

[12] *Artículo online sobre la taxonomía de usuarios de Bartle.* URL: https://drimify.com/es/recursos/tipos-jugadores-bartle-tipo-jugador-es/.

[13] *Artículo sobre el patrón Decorator.* URL: https://refactoring.guru/design-patterns/decorator.

[14] Atlassian. *Artículo de Atlassian sobre los tipos de pruebas de software.* URL: https://www.atlassian.com/es/continuous-delivery/software-testing/types-of-software-testing.

[15] *Blog de Nicole Lazzaro, autora de The 4 Keys 2 Fun.* URL: https://www.nicolelazzaro.com/the4-keys-to-fun/.

[16] Matt Bullock. *Who started gamification?* Mar. de 2023. URL: https://spinify.com/blog/gamification-history/.

[17] Harry Cloke. *The History of Gamification: From the Beginning to Right Now.* Ago. de 2019. URL: https://www.growthengineering.co.uk/history-of-gamification/.

[18] Edward L. Deci y Richard M. Ryan. *Intrinsic motivation and self-determination in human behavior.* Plenum Press, 1985.

[19] *Descripción del patrón MVT, aplicable tanto a Django como a Flask si este último se usa junto a un Data Mapper como SQLAlchemy.* Sep. de 2019. URL: https://www.geeksforgeeks.org/python/django-project-mvt-structure/.

[20] *Descripción del patrón Page Controller.* Dic. de 2024. URL: https://dev.to/xxzeroxx/php-design-patterns-page-controller-34f2.

[21] *Descripción del patrón Singleton.* Abr. de 2016. URL: https://www.geeksforgeeks.org/system-design/singleton-design-pattern/.

[22] *Descripción del Proceso Unificado.* URL: https://www.open.edu/openlearn/science-maths-technology/approaches-software-development/content-section-3.3.

[23] *Documentación de Docker y Docker-Desktop.* Abr. de 2025. URL: https://docs.docker.com/desktop/.

[24] *Documentación de Flask 3.1.x.* URL: https://flask.palletsprojects.com/en/stable/.

[25] *Documentación de Jinja, sección en la que se presentan los delimitadores por defecto de Jinja.* URL: https://jinja.palletsprojects.com/en/stable/templates/#:~:text=delimiters.%5C%20The%5C%20default%5C%20Jinja%5C%20delimiters%5C%20are%5C%20configured%5C%20as%5C%20follows%5C%3A.

[26] *Documentación de WSGI, en concreto la definición de qué es WSGI.* URL: https://wsgi.readthedocs.io/en/latest/what.html.

[27] *Enlace a la web de CodeRunner.* URL: https://coderunner.org.nz/.

[28] *Enlace al segmento de la documentación de Flask en la que se indica la dependencia del microframework con el "toolkit"WSGI Werkzeug.* URL: https://flask.palletsprojects.com/en/stable/#:~:text=Flask%5C%20depends%5C%20on%5C%20the%5C%20Werkzeug%5C%20WSGI%5C%20toolkit.

[29] *Enlace al segmento de la documentación de WSGI en la que se habla de la relación entre este, Flask y Werkzeug.* URL: https://wsgi.readthedocs.io/en/latest/frameworks.html#:~:text=Flask%5C%20is%5C%20a%5C%20microframework%5C%20for%5C%20Python%5C%20based%5C%20on%5C%20Werkzeug%5C%2C%5C%20Jinja%5C%202%5C%20and%5C%20good%5C%20intentions.

[30] *Enlace que explica el uso de la clase declarativa db.Model.* URL: https://flask-sqlalchemy.readthedocs.io/en/stable/models/.

[31] *Entrada del VSCode Marketplace de la extensión Docker.* URL: https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-docker.

[32] *Entrada del VSCode Marketplace de la extensión Jinja.* URL: https://marketplace.visualstudio.com/items?itemName=wholroyd.jinja.

[33] *Entrada del VSCode Marketplace de la extensión Live Preview.* URL: https://marketplace.visualstudio.com/items?itemName=ms-vscode.live-server.

[34] *Entrada del VSCode Marketplace de la extensión Python.* URL: https://marketplace.visualstudio.com/items?itemName=ms-python.python.

[35] *Entrada del VSCode Marketplace de la extensión SQLTools.* URL: https://marketplace.visualstudio.com/items?itemName=mtxr.sqltools.

[36] Allen Frederick E. *Disneyland uses "Electronic whip" on employees.* Oct. de 2011. URL: https://www.forbes.com/sites/frederickallen/2011/10/21/disneyland-uses-electronic-whip-on-employees/.

[37] FutureLearn. *What is SQL and what is it used for?* 2023. URL: https://www.futurelearn.com/info/blog/what-sql-used-for.

[38] *Guía de instalación y uso de mysql-connector-python.* URL: https://dev.mysql.com/doc/connector-python/en/.

[39] J Hammersley J. & Lees-Miller. *Overleaf.* 2012. URL: https://es.overleaf.com/.

[40] *Herramienta de modelado de diagramas UML.* URL: https://www.visual-paradigm.com/.

[41] "IEEE Standard Glossary of Software Engineering Terminology". En: *ANSI/ IEEE Std 729-1983* (1983), págs. 1-40. DOI: 10.1109/IEEESTD.1983.7435207.

[42] Hossein Nejati Javaremi. *Artículo donde se describe el patrón Data Mapper.* Abr. de 2025. URL: https://hosseinnejati.medium.com/what-is-the-data-mapper-pattern-219ce6484e28.

[43] Philippe Kruchten. *Definición de Caso de Uso. Versión online del libro The Rational Unified Process: An Introduction.* Ene. de 2000. URL: https://flylib.com/books/en/1.561.1.66/1/?/#:~:text=A%5C%20use%5C%20case%5C%20is%5C%20a%5C%20sequence%5C%20of%5C%20actions%5C%20a%5C%20system%5C%20performs%5C%20that%5C%20yields%5C%20an%5C%20observable%5C%20result%5C%20of%5C%20value%5C%20to%5C%20a%5C%20particular%5C%20actor..

[44] Toby Mao. *Repositorio GitHub de sqlglot, creado por Toby Mao.* Abr. de 2025. URL: https://github.com/tobymao/sqlglot.

[45] Andrzej Marczewski. *Hexad: A player type framework for Gamification Design.* Ene. de 2022. URL: https://www.gamified.uk/user-types/.

[46] Nick Morgan. *What Are The 10 Most Common Software Architecture Patterns?* Ago. de 2023. URL: https://medium.com/@the_nick_morgan/what-are-the-10-most-common-software-architecture-patterns-faa4b26e8808.

[47] Tamás Németh. *Página web en la que se listan y clasifican los mecanísmos de los juegos que resultan de utilidad para los objetivos de la gamificación.* 2015. URL: https://ludus.hu/en/gamification/game-elements/#:~:text=their%5C%20friends%5C%E2%5C%80%5C%99%5C%20scores.-,LEVELS,new%5C%20opponents%5C%2C%5C%20but%5C%20also%5C%20unlock%5C%20new%5C%20items%5C%20or%5C%20abilities%5C%20(Becker%5C%2C%5C%202009).,-CHALLENGES%5C%20/%5C%20QUESTS.

[48] *Página oficial de Visual Studio Code.* URL: https://code.visualstudio.com/.

[49] *Página web de Crixet.* URL: https://crixet.com/.

[50] *Página web de DockerSDK, también llamado Docker-Py.* URL: https://docker-py.readthedocs.io/en/stable/.

[51] *Página web de Flask-bcrypt.* URL: https://flask-bcrypt.readthedocs.io/en/1.0.1/.

[52] *Página web de Flask-Login.* URL: https://flask-login.readthedocs.io/en/latest/.

[53] *Página web de Flask-WTF.* URL: https://flask-wtf.readthedocs.io/en/1.2.x/.

[54] *Página web de Git.* 2025. URL: https://git-scm.com/.

[55] *Página web de GitHub.* 2025. URL: https://github.com/.

[56] *Página web de Google Chrome.* URL: https://www.google.com/intl/es_es/chrome/.

[57] *Página web de itsdangerous.* URL: https://itsdangerous.palletsprojects.com/en/stable/.

[58] *Página web de la extensión Flask-SQLAlchemy.* URL: https://flask-sqlalchemy.readthedocs.io/en/stable/.

[59] *Página web de Microsoft Outlook.* URL: https://www.microsoft.com/es-es/microsoft-365/outlook/email-and-calendar-software-microsoft-outlook.

[60] *Página web de Microsoft Teams.* URL: https://www.microsoft.com/es-es/microsoft-teams/group-chat-software.

[61] *Página web de MySQL.* URL: https://www.mysql.com/.

[62] *Página web de Python.* URL: https://www.python.org/downloads/.

[63] *Página web de SQLAlchemy.* URL: https://www.sqlalchemy.org/.

[64] *Página web de Werkzeug.* URL: https://werkzeug.palletsprojects.com/en/stable/.

[65] *Página web de WTForms.* URL: https://wtforms.readthedocs.io/en/3.2.x/.

[66] Pepe Pedraz. *Las llaves de la diversión. – Pensamiento lúdico.* es. URL: https://www.alaluzdeunabombilla.com/2016/08/23/las-llaves-de-la-diversion/.

[67] Nick Pelling. *The (short) prehistory of "gamification":* 2011. URL: https://nanodome.wordpress.com/2011/08/09/the-short-prehistory-of-gamification/.

[68] Alma María Pisabarro Marrón y Carlos E. Vivaracho. "Gamificación en el aula: gincana de programación". En: *ReVisión* 11.1 (2018). ISSN: 1989-1199.

[69] *Portal de Empleo con datos sobre salarios promedio.* Abr. de 2025. URL: https://www.tecnoempleo.com/tecnocalculadora.php.

[70] *Referencia utilizada para ejemplo de diagrama del patrón Template-View.* URL: `https://martinfowler.com/eaaCatalog/templateView.html`.

[71] *Referencia utilizada para entender que es una Blueprint en Flask.* URL: `https://realpython.com/flask-blueprint/`.

[72] Mark Richards. *Enlace al primer capítulo del libro Software Architecture Patterns de la plataforma de aprendizaje online O'Reilly.* Feb. de 2015. URL: `https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html`.

[73] *Sección de la web de Flask en la que se describe el concepto Blueprint y sus usos.* URL: `https://flask.palletsprojects.com/en/stable/blueprints/`.

[74] Kevin Werbach y Dan Hunter. *For the win: How game thinking can revolutionize your business.* Wharton Digital Press, 2012.

[75] Contract Workplaces. *Cara y ceca de la gamificación.* Jul. de 2018. URL: `https://www.worktechacademy.com/cara-y-ceca-de-la-gamificacion/`.

[76] Gabe Zichermann y Christopher Cunningham. *Gamification by design: Implementing game mechanics in web and mobile apps.* .ºReilly Media, Inc.", 2011.