



Universidad
Rey Juan Carlos

GRADO EN INGENIERÍA EN SISTEMAS AUDIOVISUALES
Y MULTIMEDIA

Curso Académico 2022/2023

Trabajo Fin de Grado

HERRAMIENTA PARA LA CREACIÓN DE
GYMKHANAS SOBRE MODELADO

Autor : Jorge De Pablo Martínez

Tutor : Dr. Gregorio Robles Martínez

Trabajo Fin de Grado

Herramienta para la creación de Gymkhanas sobre Modelado.

Autor : Jorge De Pablo Martínez

Tutor : Dr. Gregorio Robles Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el día de octubre de 2022,
siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de octubre de 2022

*Dedicado a
mi señora madre.*

Agradecimientos

Como no podía ser de otra manera, primero quiero agradecer a mi familia el apoyo y la ayuda constante ofrecida durante mis años de universidad. Especialmente a mi madre, que ha disfrutado viendo como su hijo ha logrado culminar sus objetivos.

Por otro lado, teniendo en cuenta que la gran mayoría del personal docente que he tenido en la ETSIT ha mostrado una actitud sobresaliente conmigo, quiero destacar mi especial agradecimiento a Gregorio, gracias al cual es posible este TFG, a Damián que me inspiró para centrar mi carrera profesional, sin olvidarme del resto que ha dejado huella en mi vida como Eva, Roberto, Inma, Pedro, Javier, Enrique. . .

No quiero olvidarme de las amistades que he hecho durante este trayecto, entre las cuales nos hemos ayudado y apoyado en momentos difíciles y con las que he celebrado y disfrutado de nuestros logros: Rafa, Julia, Irene, Antonio, Juanjo, Victor, Willy. . .

En general a todo estudiante y docente comprometido, que aportan su granito de arena para mejorar y aportar conocimiento de fácil acceso al resto del mundo que, al fin y al cabo, es la idea en torno a la cual gira este proyecto. Gracias.

Resumen

El lenguaje de modelado unificado es, en la actualidad, el lenguaje más utilizado para el modelado de software. No obstante, los diagramas y aspectos conceptuales del mismo pueden utilizarse para otros entornos, como el educativo.

Este Trabajo de Fin de Grado tiene como objetivo desarrollar una aplicación web a través de la cual, usuarios de tempranas edades puedan familiarizarse con el lenguaje unificado de modelado. Se trata de una aplicación web diseñada para el entorno educativo. La idea principal es poder dotar a un equipo docente de una herramienta para que sea utilizada por los alumnos, los cuales tendrán que avanzar a través de *gymkhanas*, juegos y retos relacionados con diagramas que sigan las reglas del lenguaje de modelado unificado.

La aplicación se ha desarrollado prácticamente en su totalidad con tecnologías *OpenSource* o, en su defecto, sobre software propietario de empresas, pero que cuentan con planes gratuitos para propuestas estudiantiles o pequeñas pruebas de concepto. El acceso a la aplicación web y al código fuente que la conforma debe ser totalmente gratuito. Entre esas tecnologías hay que destacar Python y Django, junto con Bootstrap, HTML, CSS, y Git para el control de versiones. Se ha desplegado en la red de Internet gracias a Heroku y, por tanto, se puede tener acceso a *Gymkhana App* desde cualquier parte del mundo, en cualquier momento del tiempo.

Summary

The Unified Modeling Language is currently the most widely used language in software modeling. However, the diagrams and conceptual aspects of it can be used for other environments, such as education.

This final thesis of the degree aims to develop a web application for teaching and student use through which students of early ages can learn and become familiar with activities that imply in some way understanding and analyzing diagrams to extract information from them. All of this will be attempted through gymkhanas and games to somehow attract the interest of younger users.

The application has been developed practically in its entirety with *OpenSource* technologies, or failing that, with company software but that offer free (as in gratis) plans for student proposals and/or small proofs of concept and, of course, access to it must be totally free (as in gratis). These technologies include Python and Django, along with Bootstrap, HTML, CSS, and git for version control. It has been deployed on the internet thanks to Heroku and therefore Gymkhana App can be accessed from anywhere in the world, at any moment of time.

Índice general

1. Introducción	1
1.1. Contexto	1
1.2. Estructura de la memoria	2
2. Objetivos	5
2.1. Objetivo general	5
2.2. Objetivos específicos	5
3. Estado del arte	7
3.1. UML: Unified Modeling Language	7
3.1.1. Diagramas Estructurales	9
3.1.2. Diagramas de Comportamiento	10
3.2. Python	11
3.3. Django	12
3.3.1. Modelo Vista Controlador (MVC)	13
3.4. Git	14
3.5. Bootstrap	14
3.6. Heroku	15
3.6.1. Heroku PostgreSQL	16
3.7. SQLite 3	16
3.8. PostgreSQL	17
3.9. Visual Studio Code	18
3.10. WSL 2	19
3.11. Diagrams.net	20

4. Diseño e implementación	21
4.1. Arquitectura general	21
4.2. Módulo Vistas-Controlador	21
4.3. Módulo Templates-Vistas	24
4.3.1. Internacionalización	26
4.4. Módulo Models-Modelo	27
4.5. Despliegue del software	29
5. Resultados	33
5.1. Acceso al proyecto	33
5.2. Página principal	33
5.3. Menú de juegos	33
5.4. Página de los retos	35
5.5. Inicio de sesión	35
5.6. Perfil del usuario	38
5.7. Subida de retos	38
5.8. Creación de juegos	41
6. Experimentos y validación	43
6.1. Validación sobre un entorno WSL.	43
6.2. Validación en la plataforma Heroku.	44
7. Conclusiones	45
7.1. Consecución de objetivos	45
7.2. Aplicación de lo aprendido	46
7.3. Lecciones aprendidas	47
7.4. Trabajos futuros	48
A. Anexo I: Ejemplos de Diagramas UML	51
A.1. Diagrama de clases	51
A.2. Diagrama de objetos	51
A.3. Diagrama de despliegue	51
A.4. Diagrama de componentes	53

<i>ÍNDICE GENERAL</i>	XI
A.5. Diagrama de paquetes	53
A.6. Diagrama de estructura compuesta	55
A.7. Diagrama de actividad	55
A.8. Diagrama de casos de uso	55
A.9. Diagrama de máquina de estados	58
A.10. Diagrama de interacción	58
Bibliografía	61

Índice de figuras

3.1. Jerarquía de diagramas en UML 2.2 (como diagrama de clases).	8
3.2. Top uso de lenguajes de programación alojados en GitHub	11
3.3. Estructura de las relación modelo-vista-controlador.	13
4.1. Diagrama entidad-relación de la aplicación Ghymkhana App.	29
5.1. Página principal de la aplicación.	34
5.2. Menú de juegos de la aplicación.	36
5.3. Ejemplo de un reto.	37
5.4. Página de inicio de sesión.	38
5.5. Perfil del usuario.	39
5.6. Formulario de subida de retos.	40
5.7. Formulario de creación de juegos.	41
6.1. Gráfico de Git del repositorio.	44
A.1. Ejemplo de diagrama de clases.	52
A.2. Ejemplo de diagrama de objetos.	52
A.3. Ejemplo de diagrama de despliegue.	53
A.4. Ejemplo diagrama de componentes.	54
A.5. Ejemplo diagrama de paquetes.	54
A.6. Ejemplo diagrama de estructura compuesta.	55
A.7. Ejemplo diagrama de actividad.	56
A.8. Ejemplo diagrama casos de uso.	57
A.9. Ejemplo diagrama de máquina de estados.	58

A.10. Ejemplo diagrama de colaboración	59
A.11. Ejemplo diagrama de secuencia	59
A.12. Ejemplo diagrama de tiempos.	60

Capítulo 1

Introducción

Esta memoria recoge el trabajo realizado durante el desarrollo de *Gymkhana App*. Se trata de una aplicación web diseñada y orientada para que niños y niñas puedan disfrutar de sencillos juegos a la vez que aprender conceptos del lenguaje de modelado unificado.

1.1. Contexto

Gymkhana App es una aplicación web con simples juegos que consisten en la resolución de retos asociados a diagramas UML (lenguaje de modelado unificado, de las siglas en inglés *Unified Modeling Language*), o extracción de las respuestas a través del análisis de estos.

La idea surge bajo la premisa de fomentar y ayudar en el desarrollo de habilidades como el seguimiento de flujo, pensamiento lógico, simplificación y análisis de situaciones complejas y la resolución de problemas mediante la división de estos en un conjunto de actividades estructurales que juntas forman una solución.

Los diferentes retos consisten en la resolución de sencillos enigmas que se resuelven combinando sus habilidades con diagramas UML. El nivel elegido en esta primera versión está orientado para niños de edades tempranas. También se quiere ofrecer la posibilidad de ser utilizado por el equipo docente para que estos mismos o cualquier contribuidor pueda crear, diseñar y aportar nuevos retos y juegos que puedan ser utilizados por toda la comunidad de la aplicación.

El objetivo final de este proyecto trata de poder aportar una infraestructura en forma de aplicación de acceso público y muy sencilla de entender y manejar por los usuarios con el objetivo de divulgar y enseñar conceptos de UML y que, a la vez, los más jóvenes se familiaricen

con estos últimos a través de los sencillos juegos y retos que contiene Gymkhana App.

1.2. Estructura de la memoria

La memoria de este trabajo está estructurada de la siguiente manera:

- **Capítulo 1. Introducción.** Hace una descripción básica del proyecto y se exponen las ideas principales del mismo. Aparte, también especifica la estructura de la memoria.
- **Capítulo 2. Objetivos.** En este capítulo se describe el objetivo principal de este proyecto y también objetivos más específicos e hitos a alcanzar durante el desarrollo de la aplicación.
- **Capítulo 3. Estado del arte.** Capítulo dedicado a las tecnologías utilizadas durante el desarrollo de la aplicación. Se hace una pequeña introducción de estas y se explican características básicas y propiedades que hacen que se pueda y tenga sentido utilizarlas en el proyecto.
- **Capítulo 4. Diseño e implementación.** Se hace una descripción más profunda y técnica de la arquitectura del proyecto a nivel interno, explicando detalladamente, con ayuda de diagramas, el diseño de la aplicación y desgranando los módulos que la componen.
- **Capítulo 5. Resultados.** Pequeño recorrido por la interfaz de usuario de la aplicación donde se muestra, con imágenes, los pasos a seguir para iniciar, resolver o subir nuevos retos en Gymkhana App.
- **Capítulo 6. Experimentos y validación.** Describe la metodología a seguir, los experimentos y casos de test que se han desarrollado durante el proyecto para validar los resultados del mismo.
- **Capítulo 7. Conclusiones.** Capítulo dedicado a repasar los objetivos cumplimentados durante el desarrollo del proyecto, los conocimientos aplicados y aprendidos durante el mismo y, por último, estudiar posibles líneas de trabajo futuras para el proyecto.

- **Apéndice A. Ejemplos de diagramas UML.** En este anexo se describen brevemente ejemplos de cada uno de los diagramas UML que se explican en la sección 3.1. También se añaden imágenes de cada uno de ellos con el fin de aportar información gráfica.

Capítulo 2

Objetivos

2.1. Objetivo general

El objetivo de este trabajo es desarrollar una aplicación web sencilla, limpia y clara, enfocada para su uso por niños que aprendan y se familiaricen con el lenguaje de modelado unificado y sus tipos de diagramas. También tendrá un apartado que te permita subir juegos y actividades que se integren en la misma plataforma, por tanto, también permite el uso por parte de un equipo docente.

Con esta aplicación web un equipo docente debería ser capaz de poder crear una serie de gymkhanas en forma de juegos y retos basados en diagramas UML para que estudiantes de edades tempranas sean capaces de manejar, entender y poder moverse por la interfaz de la aplicación de manera sencilla.

2.2. Objetivos específicos

El proyecto general se ha desglosado en los siguientes objetivos y tareas más específicas:

- Diseñar una aplicación web sencilla y limpia sobre *localhost* que permita resolver retos compuestos por una pregunta relacionada directamente con un diagrama embebido en una imagen sobre la web y una respuesta.
- Permitir agrupar cuantos retos se deseen en juegos o gymkhanas y crear un menú de juegos donde puedas seleccionar de 1 a n juegos.

- Hacer accesible la aplicación en diferentes idiomas, a través del sistema de internacionalización i18n. De esta manera, podemos disponer de este proyecto tanto en castellano como en inglés de manera muy sencilla y ofreciendo a otros desarrolladores o contribuidores poder traducir la aplicación a otros idiomas de una manera estandarizada.
- Crear un sistema de gestión de usuarios que gestione directamente la base de datos de la aplicación, que permita a estos usuarios la posibilidad de subir contenido a la aplicación en forma de retos y, además, puedan agrupar estos retos y/o los creados por otros usuarios, a juegos nuevos que se añadirán al panel principal común.
- Añadir sistema de puntuación a los retos, siendo estos puntos acumulables en los usuarios, y con un control de retos ya superados por un usuario a la hora de la asignación de puntos.
- Despliegue de la aplicación y migración de base de datos a un portal web público para que tener acceso a la misma desde cualquier parte del mundo.

Capítulo 3

Estado del arte

La aplicación web en la que se basa este proyecto se ha desarrollado con diferentes tecnologías gratuitas y de código abierto (*Open Source*), pero la piedra angular sobre la que se apoya todo el proyecto es sobre el lenguaje de modelado unificado.

3.1. UML: Unified Modeling Language

El lenguaje de modelado unificado, también conocido por sus siglas en inglés UML [6], es el lenguaje de modelado de software más utilizado actualmente.

El modelado de software es una parte esencial en todo el proceso de desarrollo del software. Al igual que en el plano análogo de la construcción de rascacielos, los planos, mapas del sitio y modelos físicos cumplen un papel esencial en la finalización con éxito del proyecto, en el desarrollo de software el lenguaje de modelado unificado es necesario para asegurarse de que la funcionalidad del producto final es completa y correcta, de que satisface las necesidades del usuario final y que el diseño del programa admite los requisitos de escalabilidad, solidez, seguridad, expansibilidad y otras importantes características de un software.

Se puede definir como un lenguaje gráfico que tiene como finalidad documentar, construir, especificar y, en definitiva, visualizar un sistema. Así mismo, también ofrece un estándar con el que definir un sistema o modelo.

Es importante destacar que UML no es un lenguaje de programación ni es programación estructurada, pues UML es un lenguaje de modelado con el que se definen métodos y procesos, es decir, el lenguaje que describe el modelo.

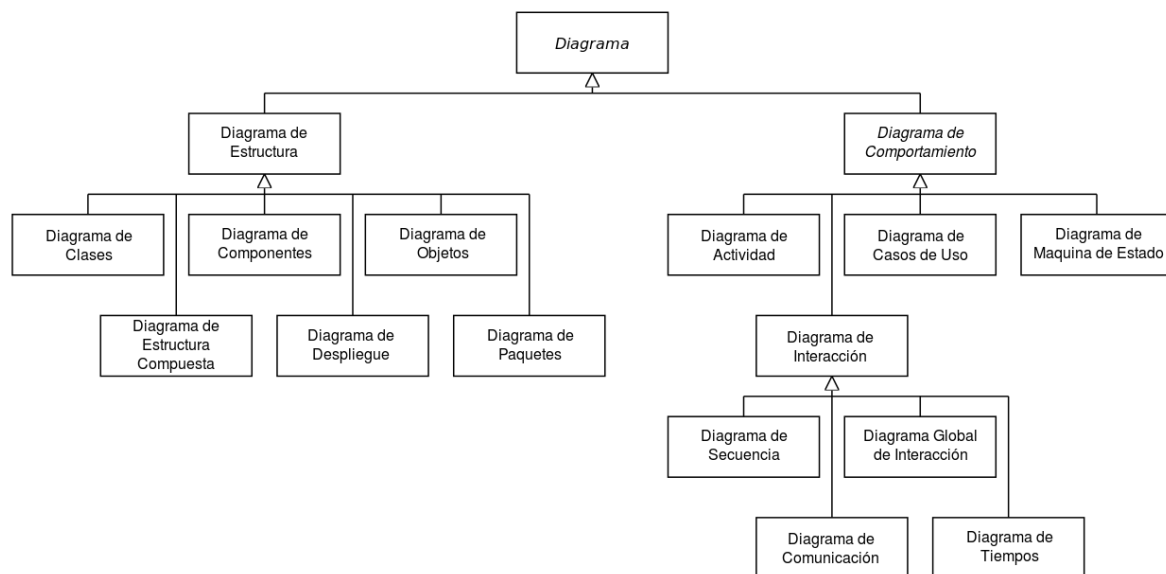


Figura 3.1: Jerarquía de diagramas en UML 2.2 (como diagrama de clases).

Desde que fue presentado el primer estándar UML en 1997 (UML 1.1) hasta día de hoy ha evolucionado. En el momento de redactar esta memoria, la OMT¹, organización que respalda UML, tiene formalmente publicada la versión 2.5.1, que es sobre la cual este proyecto se ha desarrollado. Naturalmente, entremedias, han aparecido varias versiones menores, las cuales se han ido actualizando y corrigiendo hasta la versión que se utiliza actualmente. Desde el año 2004, UML es un estándar aprobado por la ISO (ISO/IEC 19501:2005 Information technology — Open Distributed Processing — Unified Modeling Language) y en el año 2012 se actualizó la norma dando lugar, a la ISO/IEC 19505-1, la última versión disponible en este momento.

En UML se definen distintos tipos de diagramas, los cuales muestran diferentes aspectos del modelo representado. Los dos tipos principales de diagramas son: diagramas *estructurales* y diagramas de *comportamiento*. En la Figura 3.1 se puede ver un “metadiagrama” donde están ordenados jerárquicamente los diferentes tipos de diagrama. De la misma manera, en el Apéndice A, se dejan ejemplos brevemente explicados de cada uno de los tipos de diagramas que se comentan a continuación.

¹Object Management Group <https://www.omg.org/>

3.1.1. Diagramas Estructurales

Los diagramas estructurales muestran la estructura estática de los objetos en un sistema, es decir representan los elementos independientes del tiempo en un sistema. Los diagramas estructurales no muestran los detalles del comportamiento dinámico de un sistema, para ello existen los diagramas de comportamiento, sin embargo, sí que pueden mostrar las relaciones entre los comportamientos definidos en la estructura.

A continuación, se describe brevemente cada uno de los diagramas pertenecientes a la familia de los diagramas estructurales que se muestran en la Figura 3.1.

Diagrama de clases Este tipo de diagramas proporciona los mecanismos para definir la estructura de un sistema estático. Muestra las clases del sistema, sus atributos, métodos y las relaciones entre los objetos.

Diagrama de despliegue Esta clase de diagrama muestra la arquitectura de ejecución de un sistema, incluyendo los entornos de ejecución de hardware o software y el middleware que los conecta. Principalmente ayudan a entender y modelar la topología hardware de un sistema y cómo se despliega.

Diagrama de componentes Los diagramas de componentes representan un sistema software dividido en componentes y muestra las dependencias entre los componentes. Son realmente útiles para ver qué componentes pueden compartirse entre sistemas o entre diferentes partes de un sistema.

Diagrama de objetos Este tipo de diagramas ofrece una vista parcial o completa de los objetos de un sistema. Es un gráfico de instancias que incluye objetos y datos. Estos diagramas están ligados a los diagramas de clases y muestra el estado del sistema en un punto determinado del tiempo.

Diagrama de paquetes Representa las dependencias entre los paquetes que componen un sistema. Muestra como este está dividido en agrupaciones lógicas y las dependencias entre esas agrupaciones. Estos diagramas muestran la descomposición de la jerarquía lógica de un sistema.

Diagrama de estructura compuesta Esta clase de diagrama muestra la estructura interna de una clase y las relaciones e interacciones entre las distintas partes de esta. Muestran el rol definido que tiene cada elemento y la estructura compuesta como el conjunto de esos elementos interconectados.

3.1.2. Diagramas de Comportamiento

Los diagramas de comportamiento muestran el comportamiento dinámico de un sistema. Se puede describir como la serie de cambios que pueden existir en un sistema en un tiempo extraordinario.

Diagrama de actividad También conocidos como diagrama de flujo, contienen la representación gráfica de un algoritmo o proceso. Representan flujos de trabajo paso a paso utilizando símbolos para representar cada uno de estos y flechas que conectan los símbolos para marcar el flujo de ejecución.

Diagrama de casos de uso Los diagramas de casos de usos describen los diferentes tipos de interacción que tiene alguien o algo con un determinado sistema, especificando el tipo de comunicación y comportamiento de este mediante la interacción con los usuarios y/u otros sistemas.

Diagrama de interacción Este tipo de diagrama describe al detalle un determinado escenario de casos de uso. Ilustran la interacción entre el conjunto de objetos que cooperan en la realización de un proceso. También se pueden utilizar para representar secuencias ordenadas dentro de un sistema. Según las últimas versiones de UML este, diagrama se puede clasificar en cuatro tipos principales de diagramas.

- Diagrama de colaboración.
- Diagrama de secuencia.
- Diagrama de tiempos.
- Diagrama global de interacciones.

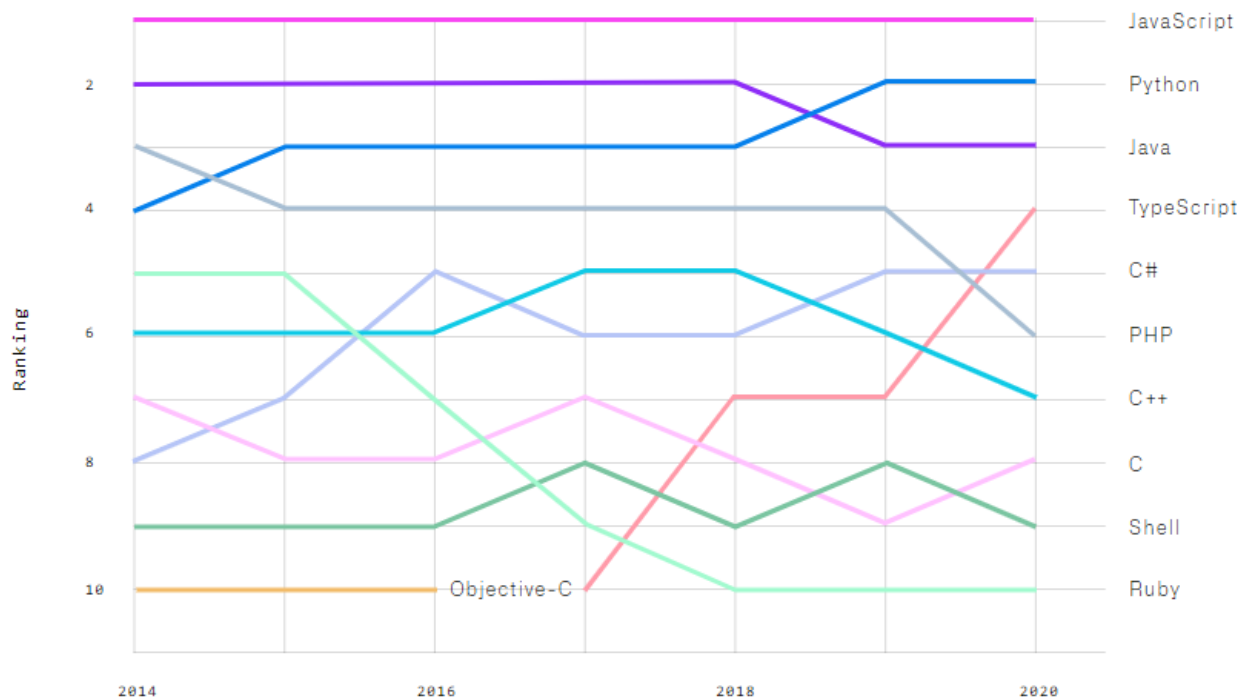


Figura 3.2: Top uso de lenguajes de programación alojados en GitHub

Al igual que el resto de diagramas, se hace una introspección más profunda en cada uno de estos diagramas en el Apéndice A.

Diagrama de máquina de estados Un diagrama de máquina de estados, conocidos en ocasiones como diagrama de estados, modela el comportamiento de un objeto. Especifican la secuencia de eventos que sufre este determinado objeto durante su tiempo de ejecución.

3.2. Python

Python [4] es un lenguaje de programación interpretado, dinámico y multiplataforma con licencia de código abierto². Su máxima es hacer un lenguaje que sea fácilmente legible y con una empinada curva de aprendizaje. Python fue creado a principios de la década de los 90 por Guido van Rossum en los Países Bajos, como curiosidad, saber que le debe su nombre gracias a la afición que tenía su creador por el grupo de humoristas británicos Monty Python.

²Python Software Foundation License: <https://docs.python.org/3/license.html>

Dentro de sus características nos encontramos con que Python es:

- Interpretado: No es necesario que sea procesado por el compilador, se detectan errores en tiempo de ejecución.
- Multiparadigma: Soporta tanto programación funcional, como programación orientada a objetos y también programación imperativa.
- Tipado dinámico: Las variables se comprueban en tiempo de ejecución.
- Flexible: No es obligatorio definir ni asignar variables antes de usarlas, es posible omitir parámetros, y su única manera de definir la estructura del código es mediante indentación.
- Multiplataforma: Puede ejecutarse tanto en Linux, como Windows como macOS.

La versatilidad y fácil acceso a este lenguaje ha provocado que en los últimos años se haya vuelto muy relevante, haciendo que sea uno de los principales y más importantes lenguajes de programación usados hoy en día. Como se puede ver en la Figura 3.2, actualmente su última versión estable es Python 3.9, pero este proyecto ha sido desarrollado usando la versión 3.7.3, ya que es una de las más populares y con más usuarios en activo. También cuenta con una versión anterior muy popular, Python 2, que es considerada desactualizada (*deprecated*) desde el 1 de enero de 2020.

3.3. Django

Django [1] es un framework de desarrollo web gratuito, de código abierto y escrito en Python. Fue lanzado en Julio de 2005 y comparte objetivos generales con Python buscando fundamentalmente poder desarrollar sitios web complejos de manera sencilla y accesible para el mayor número de personas posibles. Django funciona bajo Python, es decir, que para poder trabajar con Django es necesario tener instalado Python.

Un framework, traducéndolo al castellano, viene a ser un marco de trabajo, una especie de ecosistema formado por un conjunto de herramientas, librerías y buenas prácticas para crear aplicaciones, en este caso para desarrollar aplicaciones web.

El framework de Django en concreto nos permite crear sitios web con un cierto grado de complejidad de manera rápida y sencilla. Muchas de las tareas al realizar sitios web suelen ser

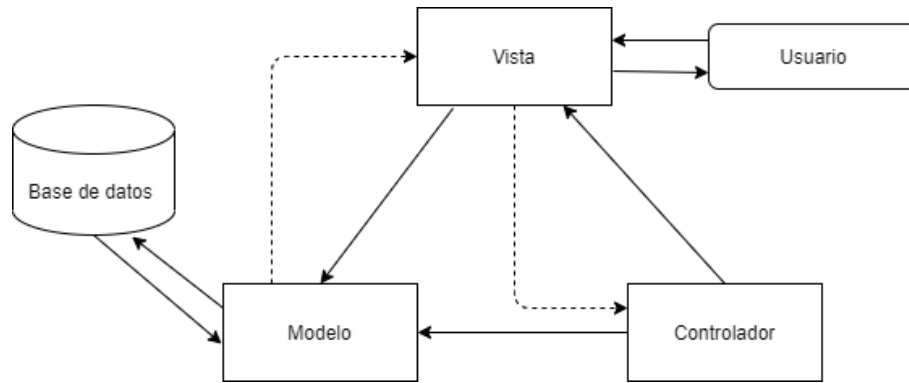


Figura 3.3: Estructura de las relación modelo-vista-controlador.

repetitivas, pesadas y comunes en la mayoría de casos, Django viene a facilitar la realización de estas tareas.

3.3.1. Modelo Vista Controlador (MVC)

Modelo-vista-controlador es un patrón de arquitectura de software, este patrón consiste en dividir la arquitectura del software en tres grandes módulos: modelo, vista y controlador. Como se ve en la figura 3.3, en esta figura, las líneas sólidas muestran una relación directa entre los elementos y las rayadas una asociación indirecta.

- **Modelo:** Se encarga de gestionar los datos, normalmente de obtener información de una base de datos.
- **Vista:** Se encarga de mostrar la información al usuario y las interacciones de este con el sistema.
- **Controlador:** Gestiona todas las comunicaciones que existen entre la vista y el modelo.

Esta división en módulos tiene como ventaja que hace las aplicaciones más funcionales, sostenibles y escalables. Otros muchos grandes frameworks de desarrollo web usan este modelo y, Django, básicamente también, pero llaman a su modelo de manera diferente. La filosofía sigue siendo la del modelo-vista-controlador pero se llama Model Template View, lo que hace es sustituir las vistas por un módulo que llama Template, o plantilla en castellano, el controlador de Django viene a ser el módulo Views y el Model sigue siendo el modelo.

3.4. Git

Git [9] es un sistema de control de versiones gratuito y de código abierto, diseñado para manejar cualquier tipo de proyecto, tanto en pequeños proyectos como en los muy grandes y complejos, funcionando con gran velocidad y eficiencia. Vino desarrollado por la mano de Linus Torvalds, famoso por iniciar el desarrollo del kernel del sistema operativo Linux.

Git es fácil de aprender y ocupa poco espacio, con un rendimiento increíblemente rápido. Su objetivo es facilitar el desarrollo y mantenimiento de código fuente tanto de forma individual o como de trabajo en equipo. Te permite crear varias ramas de desarrollo y provee de herramientas para poder unir estas ramas, además de guardar un registro de todos los cambios que sufran los archivos.

Este software es multiplataforma se puede instalar tanto en Linux, como en Windows y macOS. Se puede utilizar cualquier tipo de lenguaje de programación sobre el archivo del que se desea hacer control de versiones, ya que Git no se fija en el contenido sino en la diferencia de este contenido a lo largo del tiempo y en las diferentes ramas.

Los repositorios de código que usan Git como control de versiones no necesariamente tienen que estar alojados en un repositorio de Internet, pero lo más habitual y práctico es que sí lo esté. Existen numerosas plataformas que te permiten alojar repositorios de código fuente e interactuar con las herramientas de Git como GitLab o Gogs, que son de código abierto y gratuitas o sitios como CodeComit que pertenece a AWS³ y, por tanto, es de dominio privado. En este proyecto durante su desarrollo se ha utilizado Git para el control de versiones alojando el repositorio en GitHub, que es el otro sitio más de código abierto para alojar proyectos, es gratuita y típicamente se almacenan los repositorios de forma pública.

3.5. Bootstrap

Bootstrap [7] es un framework de código abierto y multiplataforma que contiene bibliotecas y herramientas para el diseño de aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, menús de navegación y otros elementos basados en HTML y CSS, así como extensiones de JavaScript adicionales. Fue desarrollado por Mark Otto y Hacob Thornton

³Amazon Web Services

como marco de trabajo para fomentar las herramientas internas de Twitter, en 2011 se publicó como código abierto bajo una licencia MIT License.

Bootstrap tiene soporte relativo para HTML5 y CSS3, pero es compatible con la mayoría de los navegadores web. Desde su versión 2.0 soporta diseños web adaptables. Esto significa que el diseño de la web se ajusta dinámicamente en función del display del que disponga el dispositivo donde se esté visualizando, esta tecnología también se conoce como “responsive” cuando hablamos de desarrollo web.

Este framework web sólo se ocupa del desarrollo del “front-end”, es decir de la parte que ve e interactúa el usuario directamente. Integrando este framework con el modelo-vista-controlador de Django, explicado anteriormente en la sección 3.3.1 de este documento podemos enmarcarlo dentro del módulo Template de Django, pudiendo dividir así la parte funcional y práctica del proyecto de la parte del diseño relegándola en Bootstrap.

3.6. Heroku

Heroku [2] es una plataforma que ofrece servicios de computación en la nube que soportan distintos lenguajes de programación. Fue desarrollada en 2007 con el objetivo de soportar la únicamente el lenguaje de programación Ruby, pero poco a poco extendió su soporte para otros lenguajes como Node, Java, PHP y Python entre otros. También soporta servicios de bases de datos tanto MongoDB como Redis o PostgreSQL, tanto como parte de la plataforma como servicio independiente. Además de esto, tiene integración con Git, que, dentro de Heroku, será quien maneje los repositorios de las aplicaciones subidas por los usuarios.

El modelo de arquitectura de Heroku se basa en Dynos, que son unidades de capacidad de cómputo basadas en contenedores Linux que hay dentro de la plataforma. Cada Dyno está aislado del resto por lo que se pueden desplegar entornos y procesos en cada uno de estos sin que se vean afectados otros. Esto otorga a estas pequeñas máquinas unitarias las siguientes características:

- Elasticidad y crecimiento: La cantidad de Dynos se puede cambiar en cualquier momento.
- Tamaño: Puedes elegir diferentes unidades de memoria y procesamiento en cada máquina.
- Routing: Internamente se conoce la ubicación de los Dynos y, por tanto, redirigen el

tráfico eficientemente.

- Seguimiento: Existe un manejador de Dynos, el cual está continuamente monitorizando sus servicios, en caso de que alguno falle este nodo es eliminado y levantado nuevamente.
- Distribución y redundancia. Estas unidades de procesamiento al estar aisladas, implica que, si existen fallos en la infraestructura de una de ellas, las otras no se verán afectadas y en consecuencia tampoco sus servicios.

Heroku, a diferencia de otras tecnologías vistas en este documento, no es un servicio puramente gratuito, ya que es propiedad de Salesforce, una gran empresa estadounidense de software bajo demanda. No obstante, su unidad de cómputo más básica es de uso gratuito. Con una de estas unidades o Dynos es suficiente para experimentar, poder trabajar en pequeñas aplicaciones y en pruebas de concepto. Todas estas características convierten a este servicio en un entorno perfecto para desplegar una aplicación web pequeña o mediana de manera gratuita, para que esté disponible en cualquier parte del mundo.

3.6.1. Heroku PostgreSQL

Heroku ofrece la base de datos PostgreSQL, de código abierto, en forma de Heroku Postgres.

Heroku Postgres [3] es una de las soluciones que más usan los desarrolladores en los proyectos de la plataforma. Ofrece todas las ventajas de utilizar PostgreSQL como escalabilidad, acceso a un sistema de administración, seguridad, una alta concurrencia y otras características que se detallan de manera más profunda en la sección 3.8

Los planes de precio que Heroku ofrece a sus usuarios para este servicio pasan por una gran variedad de precios, pero cuenta con un plan gratuito para desarrolladores que permite probarlo sin ningún riesgo con un tamaño y características más que suficientes para pequeños proyectos o pruebas de concepto sencillas.

3.7. SQLite 3

SQLite [11] es un sistema de gestión de bases de datos relacional de código abierto. A diferencia de los sistemas de gestión de bases datos cliente-servidor, la biblioteca SQLite es

una parte integral del programa que enlaza, y no un proceso independiente del mismo. Esto hace que la latencia sea más reducida y las comunicaciones entre procesos y funciones son más eficientes.

SQLite nació en el año 2000, fue creado por D. Richard Hipp y su objetivo era diseñar un sistema de gestión de bases de datos relacional que permitiera ejecutar programas sin la necesidad de instalar un sistema de gestión de bases de datos.

Sus primeras versiones se caracterizaban por contener la base de datos en un espacio en disco relativamente pequeño y está contenida en un solo archivo. En su versión 3, SQLite admite bases de datos de hasta 2 Terabytes de tamaño.

Este sistema de gestión de bases de datos soporta funciones SQL definidas por los usuarios, y tiene otras ventajas como una amplia compatibilidad con diferentes plataformas como Windows, Linux, Android, MacOS e iOS. También es compatible con casi todos los lenguajes de programación y proporciona APIs útiles para ellos. Por defecto, es la base de datos que viene configurada para proyectos de Django, por su ligereza, independencia de paquetes externos y la eficiencia de sus procesos es ideal para pruebas de concepto en proyectos pequeños, pero es difícil de escalar y al no presentar el estándar cliente-servidor tiene limitaciones en cuanto al formato y la sintaxis, y no presenta muchas funciones de seguridad.

Este tipo de base de datos es la seleccionada por defecto al comenzar proyectos en Django. Es por eso que se optó en un primer momento desarrollar el proyecto con este tipo de base de datos. Más adelante, en un punto avanzado del desarrollo, fue necesario implementar la compatibilidad con una base de datos de tipo PostgreSQL para el despliegue del proyecto en Heroku. Actualmente, este software es compatible con ambos sistemas de gestión de bases de datos.

3.8. PostgreSQL

PostgreSQL [10], también llamado Postgres, es un sistema de gestión de bases de datos relacional orientado a objetos y de código abierto. Este proyecto de código abierto no es gestionado por una persona o empresa, sino que lo mantienen una comunidad de desarrolladores que trabajan de forma altruista y libre, el PGDG.⁴

⁴PostgreSQL Global Development Group

Entre las características principales de este sistema de gestión de bases de datos se encuentran:

- **Alta concurrencia:** Permiten que varios usuarios o procesos accedan a la misma tabla sin necesidad bloqueos.
- **Amplia variedad de tipos nativos:** PostgreSQL provee nativamente soporte para diferentes tipos de datos como texto ilimitado, direcciones IP o MAC o CIDR, arrays, números de precisión arbitraria e incluso figuras geométricas (con variedad de funciones asociadas). Adicionalmente, permite que los usuarios puedan crear sus propios tipos de datos, que se pueden indexar perfectamente a una tabla.
- **Otras características deseables en un gestor de bases de datos,** como claves foráneas (*foreign keys*), disparadores, herencia de tablas, integridad transaccional, seguridad y escalabilidad.

Las ventajas que ofrece este gestor de base de datos son básicamente seguridad e integridad en términos generales en la BD⁵, un buen sistema de transacciones y respaldos y una conexión al sistema de gestión de base datos. Desafortunadamente, no cuenta con gestor de errores, por tanto, es difícil conocer el estado de los errores o corregirlos.

Como ya vimos en el 3.6, PostgreSQL cuenta con un paquete propio integrado en Heroku que permite utilizar este sistema de gestión de bases de datos junto con el almacenamiento de los mismos en servicios virtualizados de manera gratuita en algunos de sus planes.

3.9. Visual Studio Code

Visual Studio Code [12] es un editor de código fuente, lanzado en 2015 y desarrollado por Microsoft. Es un editor multiplataforma compatible con Windows, Linux, MacOS y ahora cuenta también con una versión web.

Soporta prácticamente cualquier lenguaje de programación conocido, resaltando la sintaxis de este de manera personalizada, ofrece sugerencias de sentencias y variables, finalización inteligente de código, integración con la shell y un gestor de archivos interno. Incluye soporte

⁵Base de Datos

para depuración y control de versiones integrado con Git. Además, gracias a la gran comunidad de desarrolladores, existen infinidad de *plugins* muy fácilmente instalables que añaden funcionalidades realmente útiles a la hora de programar, depurar, integrar o desplegar código. Es un software gratuito y de código abierto pero la descarga oficial está bajo software privativo de Microsoft.

Visual Studio Code recopila datos de uso y los envía a Microsoft, aunque esto se puede desactivar, y por la naturaleza open-source de la aplicación se puede ver exactamente qué clase de datos se recopilan y envían. Los datos pueden ser compartidos entre Microsoft, sus filiales y las autoridades según lo firmado conformemente en la declaración de privacidad.

3.10. WSL 2

Windows Subsystem for Linux (WSL) [5] es una capa de compatibilidad entre sistemas desarrollada por Microsoft con el objetivo de poder correr ejecutables de Linux nativamente en Windows 10 y Windows Server. En 2019 se lanza la versión 2 de WSL incluyendo cambios importantes, como el uso real del kernel de Linux.

WSL ofrece una interfaz que simula el kernel de Linux sobre Windows sin necesidad de instalar un SO⁶ en particiones de disco y sin necesidad de crear una máquina virtual que tenga que emular hardware. Establece un espacio de usuario GNU⁷ donde instalar un sistema base como Ubuntu, Debian o Kali Linux. Dicho entorno no contará con interfaz gráfica(a no ser que tenga ayuda de aplicaciones gráficas como un servidor X11) sino que se accederá a ella mediante una Shell de tipo Bash⁸.

Para acceder al hardware y al sistema de archivos, se hace uso directo del sistema de archivos en Windows y partes del hardware como el hardware de red e incluso su infraestructura como los puertos. Este uso compartido del sistema hace que haya gran interoperabilidad entre ambos. Esto permite incluso desarrollar software sobre Linux, en un sistema Windows con sus aplicaciones propias como Visual Studio, hacer que ejecute el código en un entorno Linux con

⁶Sistema Operativo

⁷GNU es un sistema operativo de tipo Unix, así como una gran colección de programas informáticos que componen al sistema, desarrollado por y para el Proyecto GNU y auspiciado por la Free Software Foundation.

⁸Bash (Bourne-again shell) es una popular interfaz de usuario de línea de comandos, específicamente un Shell de Unix; así como un lenguaje de scripting

sus respectivos paquetes de software y acceder al programa o aplicación desarrollado a través de un puerto de red en un navegador de Windows con relativa facilidad.

3.11. Diagrams.net

Diagrams.net [8], anteriormente conocido como draw.io, es un software de código abierto de dibujo gráfico, pensado especialmente para la creación y dibujado de diagramas.

Está desarrollado en HTML5 y JavaScript, es multiplataforma y está disponible en aplicación web⁹. Aparte de ser muy intuitiva y fácil de manejar, no requiere inicio de sesión o registro. Permite exportar los diagramas o dibujos a gran cantidad de formatos como .PNG, .PDF, .SVG y .JPEG. Se puede integrar con servicios de almacenamiento *onCloud* como Google Drive, Dropbox o OneDrive (para estas funcionalidades sí es necesario registrarse) y cuenta con un complemento para integrar la aplicación web en Visual Studio Code, lo cual simplifica el poder hacer diagramas UML paralelamente al desarrollo de un software.

⁹<https://app.diagrams.net/>

Capítulo 4

Diseño e implementación

4.1. Arquitectura general

La aplicación web `Gymkhana App` se ha desarrollado en Django con su estructura típica basada en el modelo-vista-controlador, que se explica más detalladamente en la sección 3.3.1. Dentro de este modelo, la vista será la capa que se presente y con la que interactúa el usuario, compuesta por los diferentes tipos de plantillas o Templates como los conoce Django. El modelo es quien se encarga de gestionar los datos, es quien tiene comunicación directa con la base de datos y maneja toda la información que entra y sale de esta. El controlador es quien maneja las comunicaciones entre los dos anteriores módulos y el único que tiene conexión directa con el resto de los módulos, se encarga de recibir las peticiones del módulo de vistas, gestionar internamente estas peticiones y decidir qué hacer en base a la información recogida consultando en cualquier caso los datos que considere necesarios del modelo.

4.2. Módulo Vistas-Controlador

Este módulo es el encargado de recibir y gestionar las peticiones recibidas desde el módulo Templates, gestionarlas, consultar con el módulo Models y preparar las respuestas para el usuario. Django dispone de herramientas muy útiles para la gestión y respuesta de peticiones web, de tal manera que definir las funciones del servidor cada vez que un usuario accede a una URL y generar las respuestas es limpio, sencillo, escalable. Las URL y como se gestionan las respuestas se define a continuación:

- **/ (Home):** Como muchas otras aplicaciones que existen, cuando se introduce la URL con el nombre completo del dominio sin añadir ningún directorio tras este, se muestra la página principal o home, donde, simplemente, se pueden ver dos botones grandes. Uno de ellos permite iniciar sesión y registrarte, y el otro, iniciar Gymkhana App.
- **/start:** Cuando accedes a la aplicación desde el home te lleva automáticamente a esta dirección, donde primero este módulo comprueba si existen juegos disponibles en la base de datos, y si es así, los muestra. El usuario puede elegir uno de estos juegos y enviar una petición GET con el identificador del juego elegido.
- **/challenge:** Cuando se accede a este directorio, se recoge la petición GET del usuario con el juego seleccionado, y si vienes de un reto anterior o acaba de empezar el juego. Así podemos elegir cuál de los diferentes retos que componen un juego mostrar, siempre comprobando que todos los datos existan en la base de datos para evitar posibles errores. Una vez elegido el reto a mostrar se le muestra el reto donde el usuario debe introducir la respuesta de este para poder avanzar.
- **/response:** Una vez el usuario ha rellenado la respuesta al reto se envía una petición tipo GET con la información contenida del reto al que pertenece, el juego, si ha habido un reto del mismo juego anterior y la respuesta al reto. En caso de ser correcta el servidor redirigirá al usuario al nuevo reto, si se está jugando a un juego con varios retos, o a una pantalla en la que indica que ha finalizado correctamente el juego y desde ahí el usuario podrá regresar al menú principal para poder elegir otros juegos, o repetir el mismo. También en cada response se analizará si el usuario que está jugando está registrado y le añadirá a ese usuario la puntuación de dicho reto, pero solo en caso de que no lo haya completado anteriormente. Por otro lado, si la respuesta no es correcta, enviará al usuario a una pantalla donde se indica que la respuesta es incorrecta y te dejará volver a intentar el reto o salir al menú de juegos.
- **/accounts:** Esta dirección funciona de una manera un poco diferente. Para el inicio de sesión, modificación de contraseñas y cierre de la misma en la aplicación se ha decidido utilizar un módulo ya implementado de Django que gestione correctamente estas peticiones. Este módulo se le conoce como “Django’s authentication system” y está dentro de

la configuración por defecto de Django. Una vez accedemos a esta dirección el usuario deberá ingresar su nombre de usuario y contraseña para poder continuar. En caso de ser correcto redirigirá al usuario a su perfil. Cuando el usuario decida cerrar la sesión se le redirigirá al home de la aplicación.

- **/profile:** Si te registras en la aplicación exitosamente podrás acceder a tu perfil de usuario a través de una petición GET en esta dirección. Básicamente aquí se muestra la información del usuario, como el nombre o el acumulado de sus puntos. También se podrá acceder a /start para comenzar a jugar. Desde este punto el usuario que ha iniciado sesión puede acceder a los módulos para crear retos y juegos
- **/create challenge:** En esta dirección primero comprueba que el usuario esté registrado y si es así se le muestra un formulario con los campos necesarios que componen un reto. Para poder continuar desde este punto habrá que haber rellenado correctamente todos los datos del formulario y pulsar el botón de subir reto.
- **/upload challenge:** La activación de botón mencionado en el apartado anterior enviara los datos del formulario mediante una petición POST. En este punto se revisa que los campos requeridos hayan sido rellenados correctamente y en caso afirmativo se guarda este reto en base de datos.
- **/create game:** Al igual que para crear un reto, se accede a este módulo desde la página del usuario, una vez se haya registrado. En este caso, una vez se haya accedido se presenta un formulario para poder crear un juego. Como los juegos son simples colecciones de retos, aquí se le muestra al usuario un listado con los retos subidos para que seleccione uno o varios retos. La única forma de avanzar desde este punto será pulsando el botón de subir juego con el formulario correctamente cumplimentado.
- **/upload challenge** Para terminar la subida de un juego se envían los campos del formulario contenidos en una petición POST que será la se espera recibir aquí. En caso de que todos los campos requeridos existan y sean correctos subirá el juego a la base de datos de la aplicación Gymkhana App.

Django cuenta con muchas funciones y respuestas que vienen ya integradas, como por ejemplo poder responder a peticiones que provocan errores genéricos como tipo HTTP 404 (Not

Found), entre otros. De esta manera todas las peticiones que se hacen al servidor en caso de no encontrar la información requerida en la base de datos la aplicación captura este error y devuelve un ERROR 404 genérico de Django, que aparte hace terminar el resto de procesos en curso, haciendo a la aplicación mucho más robusta ante errores.

4.3. Módulo Templates-Vistas

El módulo de Templates es el que forma la interfaz de usuario de la aplicación. Cada vista de la aplicación, o página que el usuario acaba visualizando. Está compuesta por un template o plantilla HTML sobre la que se aplica un estilo con CSS. Trabajar con plantillas en Django hace que la aplicación sea escalable y el código quede ordenado. En muchos casos la plantilla es reutilizable si se parametrizan adecuadamente, a su vez, la capa de estilo está muy bien aislada gracias a Bootstrap, de manera que se puede separar muy bien la parte programática del desarrollo de la parte encargada del diseño y el estilo evitando así tener que aplicar estilo a cada una de las páginas implicadas en el desarrollo.

La aplicación de *Gymkhana App* se compone de las siguientes views: home, start, challenge, succes y wrong, login, profile, create challenge y create game

Home Es la página principal que el usuario ve cuando accede a la web. Se limita a una presentación sencilla y limpia del proyecto, con un botón bien visible al usuario que será el que le permita avanzar y adentrarse en *Gymkhana App*.

Start Esta página actúa de forma de menú y muestra los juegos que hay disponibles en la aplicación, los lista ordenadamente de manera descendientes, y cada juego tiene un botón con el que el usuario puede interactuar, pulsando el mismo se accede a dicho juego.

Challenge En esta página muestra un único reto, que se compone de un nombre o título, una pregunta y una imagen con un diagrama UML que debe estar relacionado con el reto. Esta vista tiene también un campo que el usuario debe rellenar con la respuesta planteada al reto y un botón para enviar la petición que más tarde procesará el módulo Vistas-Controlador.

Success Una vez enviada la respuesta a un reto, si esta es correcta te redirigirá a esta página donde puedes ver un texto dando la enhorabuena al usuario. Dependiendo del juego te mostrará un botón para volver al menú de juegos o al siguiente reto si el juego estaba compuesto por varios.

Wrong En contraposición a la anterior plantilla, esta mostrará al usuario un comentario haciéndole saber que la respuesta no es correcta, pero también animándolo a que lo vuelva a intentar. Esta plantilla solo mostrará dos botones, uno para volver a intentar el reto que ha fallado y otro para volver al menú de juegos en caso de que el usuario desista y de por perdido el reto.

Login Tiene la estructura de cualquier página de inicio de sesión tradicional, donde el usuario deberá introducir su nombre y contraseña para acceder. Como diferencial al resto de sitios web, Gymkhana App no ofrece en esta página la opción de crear un usuario, ya que un usuario registrado tiene la capacidad de hacer inyecciones en la base de datos y subir contenidos como imágenes. Por tanto y por motivos de seguridad, la potestad de creación de usuarios queda relegada solo al administrador de este proyecto que, en todo caso, proporcionara las credenciales de registro a quien considere necesario.

Profile Una vez el usuario esté registrado tendrá acceso a esta página, donde podrá ver información del usuario, como su nombre en la aplicación y los puntos que ha obtenido. Además, desde este punto tendrá acceso a las dos siguientes views: create challenge y create game

Create Challenge En esta página se podrá ver un formulario con los campos que debe rellenar un usuario para poder subir el reto. Los campos son los siguientes:

- **nombre:** Nombre o título deseado para el reto.
- **pregunta:** Será la cuestión a la que los jugadores deban responder para poder completar el desafío.
- **solución:** La respuesta o solución correcta que se espera por parte del jugador para poder completar el reto.

- **tipo de diagrama:** En este campo se despliega una lista con todos los tipos de diagramas UML para que el usuario que está subiendo el juego seleccione el tipo de diagrama al que pertenece el usado en el reto.
- **imagen:** Un campo que permite al usuario abrir el explorador de archivos de su sistema operativo para poder subir una imagen, en concreto se debe subir la imagen del diagrama UML asociado al reto.
- **puntos:** Valor que se le quiere dar en puntos al reto.

Create Game Al igual que la anterior esta página mostrará un formulario que se deber rellenar para poder crear juegos.

- **título:** Nombre o título deseado para el juego.
- **retos:** Se muestra una lista con todos los retos subidos a la plataforma, tanto los creados por ese usuario como los creados por otros. El usuario que cree un juego puede elegir uno o tantos como considere para formar un juego.

Todas estas páginas han sido construidas sobre una base genérica común que usan todas las plantillas. Esta base común permite contener la información de todas las vieews mencionadas anteriormente, además de la cabecera o “header” y el pie de página o “footer”. En la cabecera hay dos enlaces ocultos que redirigirán al usuario bien a la página principal de la aplicación Gymkhana App, o a la web oficial de la organización UML. En el pie de página se muestra información relativa a la ETSIT como localización y enlaces ocultos para poder visitar la cuenta de Twitter, y la web oficial de la ETSIT. También aparece la licencia de uso de la plantilla de Bootstrap elegida

4.3.1. Internacionalización

Django también ofrece de manera relativamente sencilla la posibilidad de añadir internacionalización a las plantillas. Este proceso consiste en traducir todo el texto que aparece en las plantillas.

En este proyecto, el idioma original que se eligió para desarrollar las plantillas fue el inglés, pero con el proceso de internacionalización también está disponible en castellano. La idea básica

de la internacionalización o i18n es marcar aquellas cadenas de texto que deben ser traducidas. Estas cadenas se guardan en un diccionario, donde se añade la traducción al idioma o idiomas deseados, de esta manera cuando un usuario seleccione un idioma, mostrará estas cadenas de texto marcadas en el lenguaje que le corresponda al usuario.

En el diccionario, aparte de todas las cadenas de texto que se muestran en las diferentes plantillas, también están guardadas las respuestas a los retos. Esta es la única manera de poder procesar las respuestas de los usuarios en diferentes idiomas y comprobar que coinciden con la respuesta de un reto concreto.

4.4. Módulo Models-Modelo

Este módulo es el que gestiona la información de la aplicación Gymkhana App, toda la información de la aplicación está contenida en una base de datos. El gestor de base de datos utilizado es, en este caso, SQLite versión 3. Este sistema de base de datos es un sistema de base de datos relacional y es por defecto el que utiliza Django, no obstante, se permite cambiar a otros modelos fácilmente. Se ha decidido utilizar este gestor precisamente por la sencillez y compatibilidad con Django. Con SQLite la base de datos completa se encuentra en un solo archivo totalmente autocontenido, sin dependencias externas, además, admite concurrencia, es decir, que permite que múltiples procesos tengan el archivo de base de datos abierto y puedan leer a la vez. En la figura 4.1 se puede ver el diagrama entidad-relación que define la base de datos de la aplicación, el proyecto de Django tiene más tablas aparte de las que se muestran, pero el resto son propias de la arquitectura de Django así que no nos centraremos en ellas. Las tablas que componen la base de datos de la aplicación se describen a continuación:

- **Users:** En esta tabla se almacenan los usuarios registrados en Gymkhana App. Dispone de los siguientes campos:
 - **name:** Indica el nombre del usuario.
 - **email:** La dirección de correo electrónico con la que se registra el usuario.
 - **admin:** Un campo que indica si este usuario tiene privilegios de administrador o no.
 - **points:** Campo numérico entero que contiene la puntuación del usuario.

- **challenges passed:** Un relación Many To Many con la tabla Challenges, que registra que Challenges ha superado ya el usuario.
 - **created at:** Campo en formato fecha del momento en el que se crea el usuario.
 - **updated at:** Campo en formato fecha que indica el momento que se actualiza la información del usuario.
- **Diagrams:** Esta tabla contiene los tipos de diagramas que pueden aparecer en los retos, estos tipos de diagramas se limitan a los mencionados en el apartado 3.1 de este documento.
 - **name:** Indica el nombre general del tipo de diagrama UML.
 - **description:** La descripción del tipo de diagrama asociado que se puede extender hasta diez mil caracteres.
 - **Challenges:** Tabla que almacena los retos de los que se componen los juegos que maneja la aplicación.
 - **name:** Nombre que se le da a cada reto.
 - **question:** Pregunta clave sobre el reto específico.
 - **awwser:** Respuesta a la pregunta clave.
 - **image:** Campo tipo imagen que contiene el Diagrama UML al que está asociado el reto.
 - **diagram type:** Identificador único que referencia el tipo de diagrama de la tabla Diagrams al que está asociado el reto.
 - **creator:** Usuario que ha creado este reto.
 - **created at:** Campo de tipo date que registra el momento en el que se creó el reto.
 - **points:** Campo tipo numérico entero que contiene la puntuación del reto.
 - **Games:** Tabla que almacena los distintos juegos, estos juegos son principalmente una colección de uno o varios retos.
 - **title:** Nombre o título del juego.

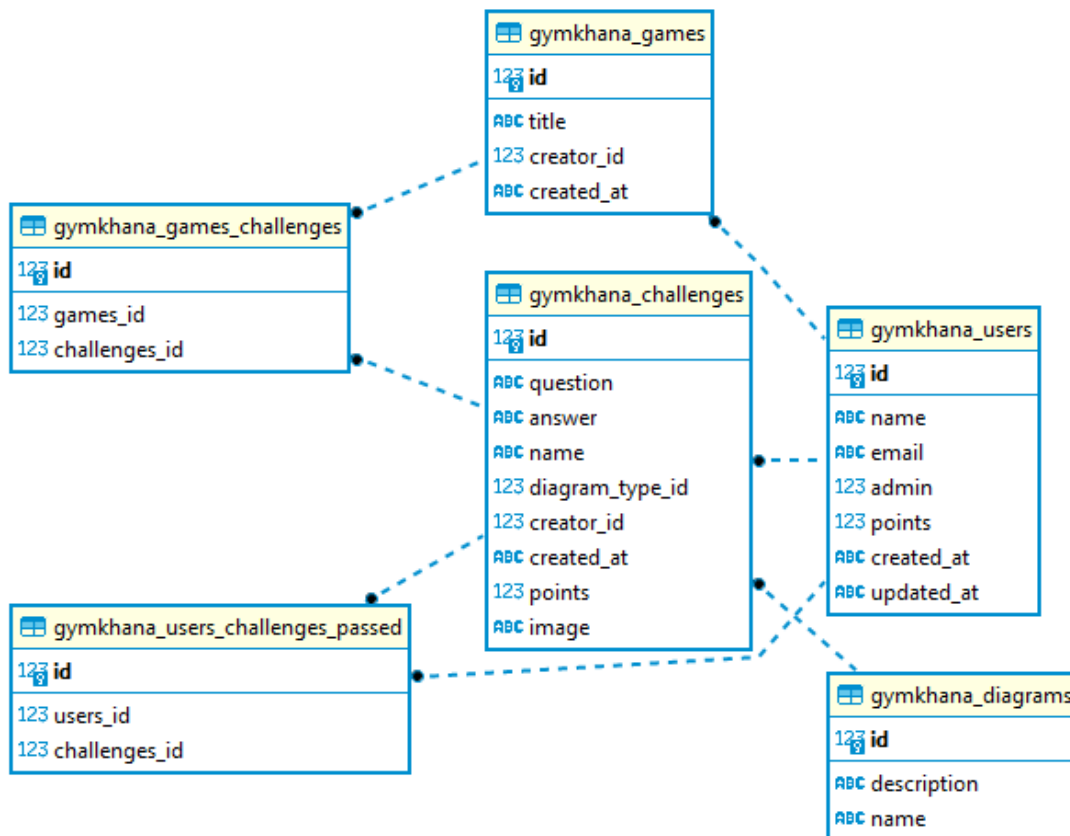


Figura 4.1: Diagrama entidad-relación de la aplicación Ghymkhana App.

- **challenges:** Uno o varios retos de los contenidos en la tabla Challenges.
- **creator:** Usuario que ha creado el juego.

4.5. Despliegue del software

Con el objetivo de poder acceder a Gymkhana App desde cualquier lugar con acceso a la red de Internet, este proyecto se ha desplegado en un servidor de acceso público. Este despliegue se ha realizado en los servidores de Heroku, se puede encontrar más información acerca de este servicio en la sección 3.6.

Heroku ofrece varias maneras de gestionar el despliegue de código en sus servidores; con un proyecto Git, a través de Docker¹ y con otros servicios de integración populares entre desa-

¹Herramienta de automatización de despliegue de software en contenedores que se ha hecho muy popular en estos últimos años entre los desarrolladores.

rolladores. En este proyecto se ha desplegado en Heroku a través de Git.

Para este proceso de despliegue es por tanto necesario tener instalado en la máquina desde la que se quiere desplegar el proyecto los paquetes de Git y Heroku CLI. Además de hacer ciertas variaciones en el proyecto y añadir archivos necesarios para la autoconfiguración en las máquinas remotas de Heroku. Estas modificaciones se explican a continuación:

Heroku necesita saber el lenguaje o lenguajes de programación en el que está basado el proyecto, para esto, es necesario crear en la carpeta raíz del proyecto un archivo llamado *runtime.txt*, en el caso particular de este proyecto este archivo está compuesto de una sola línea.

```
python-3.7.13
```

También será necesario añadir en el archivo *requirements.txt* todas las dependencias del proyecto para que Heroku pueda instalarlas. Es importante durante el proceso de desarrollo haber tenido un entorno aislado y tener todas las dependencias del proyecto correctamente instaladas.

```
$ pip freeze > requirements.txt
```

Para que las máquinas de Heroku tengan la información necesaria para ejecutar las piezas de la aplicación es necesario un archivo *Procfile*. En el caso de este proyecto se compone como se indica a continuación:

```
web: gunicorn ProjectUML.wsgi
```

Para que en el servidor tenga archivos estáticos es necesario editar el fichero de *settings.py* y añadir el siguiente *middleware*, su función es básicamente para mejorar la eficiencia de la aplicación, comprimiendo y almacenando en caché información de los ficheros estáticos del proyecto.

```
STATICFILES_STORAGE =  
'whitenoise.storage.CompressedManifestStaticFilesStorage'
```

Una vez hecho todo esto, suponiendo que el repositorio del proyecto ya tenga una rama Git iniciada para el control de versiones y que el usuario se haya registrado dándose de alta en Heroku, se enlazará el directorio Git con el proyecto existente en Heroku de manera local.

```
$ heroku create -a project-uml
```



```
# si el proyecto ya existe se debe usar este otro comando:  
$ heroku git:remote -a project-uml  
$ git add -A  
$ git commit -m "cambios para despliegue en Heroku"  
$ git push heroku main
```

En este momento, el proyecto ya estaría desplegado y listo para usarse, sin embargo, no tendría nada de información contenida en la base de datos, aunque sí tendrá su estructura formada. Es necesario migrar la base de datos local usada en desarrollo hacia la base de datos PostgreSQL.

```
$ python3 manage.py dumpdata --exclude contenttypes > data.json  
$ git add data.json  
$ git commit -m "se añade archivo para migración de base de datos."  
$ git push heroku main  
$ heroku run python3 manage.py loaddata data.json
```


Capítulo 5

Resultados

5.1. Acceso al proyecto

Se puede acceder a *Gymkhana App* a través del siguiente enlace:

- <https://project-uml.herokuapp.com/>

Y el código fuente utilizado se encuentra en un repositorio público de GitHub:

- <https://github.com/jorgedepablo/ProjectUML>

5.2. Página principal

La página principal, también comúnmente conocida como *home*, tiene la única función de presentar la aplicación de manera clara y limpia, con no demasiada información para no bombardear ni agobiar a los potenciales usuarios que puedan visitar la página. En este punto también se exponen dos botones principales que llevan a las dos principales funcionalidades de la aplicación: empezar a jugar y poder registrarse para acceder a las funcionalidades que permiten crear nuevos retos y juegos.

5.3. Menú de juegos

El menú de juego es la página a la que accedes inmediatamente después de pulsar el botón *START!* de la página principal. Aquí se presentan en manera de lista descendiente todos los

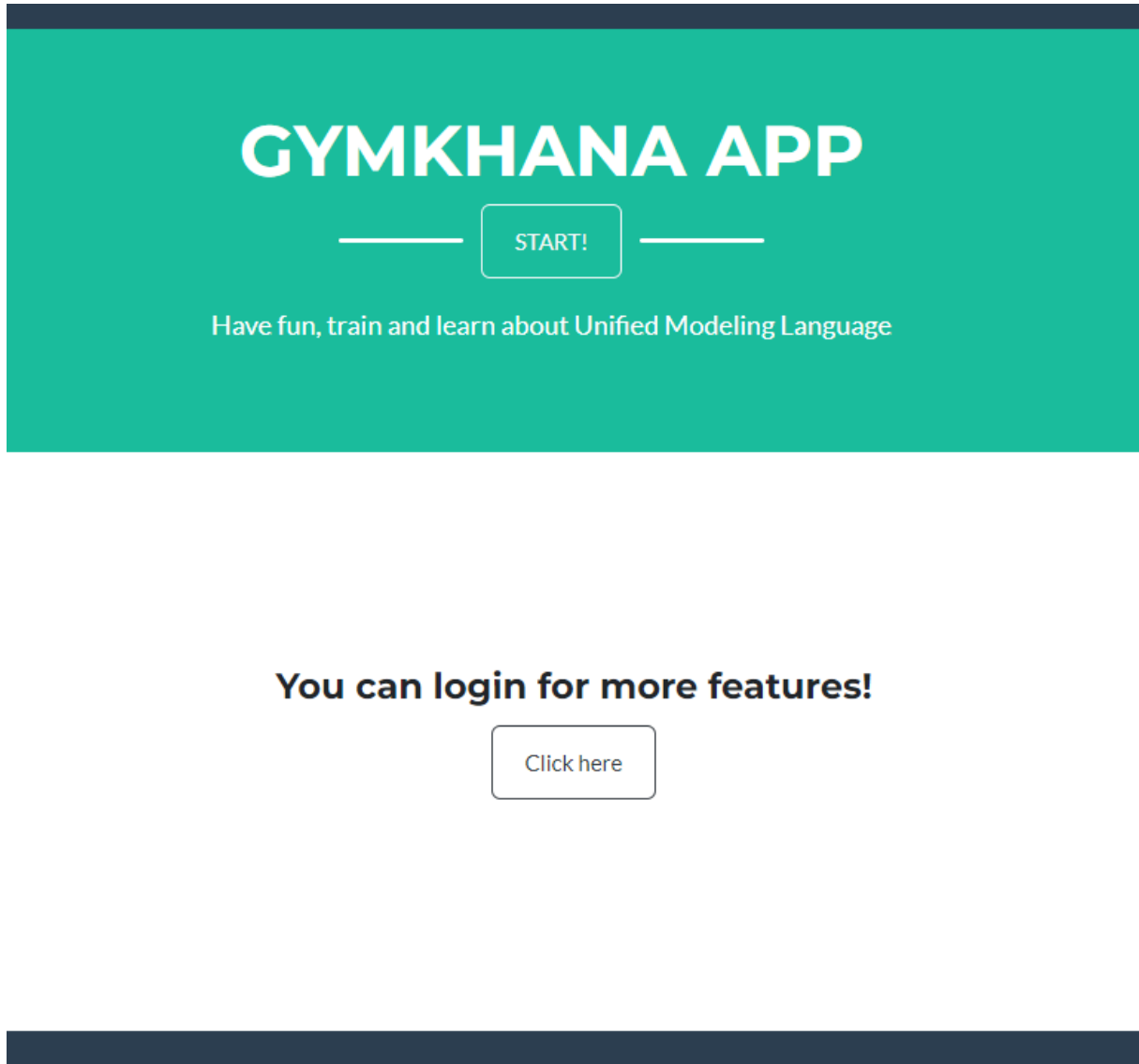


Figura 5.1: Página principal de la aplicación.

juegos subidos a la aplicación. Pulsando sobre la imagen de cada uno de los juegos la aplicación debería llevar al usuario al primer reto de cada una de las pruebas.

5.4. Página de los retos

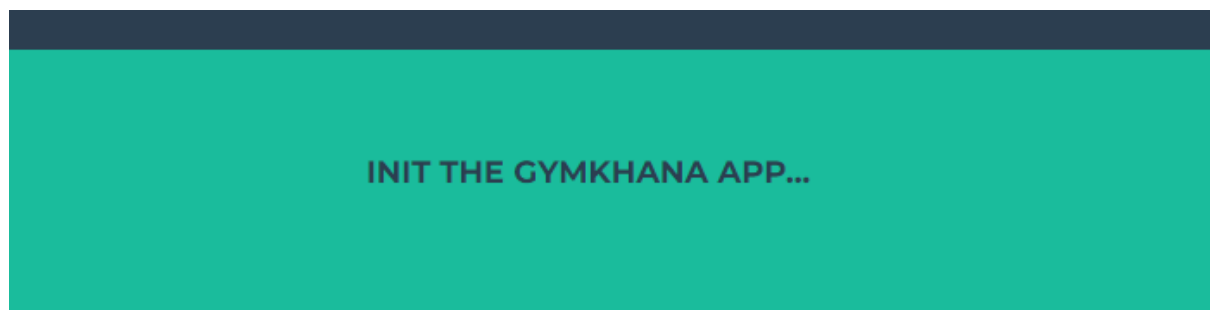
Una vez pulsado sobre uno de los juegos redirigirá al usuario a la página con el primer reto del juego seleccionado. Todos los retos siguen el mismo esquema:

1. En la cabecera del reto se ve el nombre del tipo de diagrama UML con el que está relacionado el reto.
2. Para abrir el reto tenemos un pequeño resumen explicando el tipo de diagrama que es y para qué sirve.
3. Luego se plantea la pregunta o tarea a la que debe responder el usuario para poder completar el reto.
4. Debajo está la imagen con el diagrama UML sobre el cual el usuario debe extraer la información para poder contestar el reto.
5. Por último, un cuadro de texto y un botón de enviar la respuesta.

Después de enviar la respuesta el usuario es redirigido a otra página donde se le dice al usuario si la respuesta es correcta o no. En caso de que sea correcta el usuario podrá continuar con el siguiente reto, si es que el juego está formado por más, y en caso de que sea incorrecta, se le da la opción de volver al menú de juegos o volver a intentar el reto fallado.

5.5. Inicio de sesión

La sección de inicio de sesión se reduce a un simple *login*. Por motivos de seguridad no se ha implementado las funcionalidades de registro o dar de alta a un nuevo usuario. El uso es simple; el usuario debe introducir las credenciales dadas por el administrador de la aplicación y pulsar el botón de iniciar sesión.



Juego 1

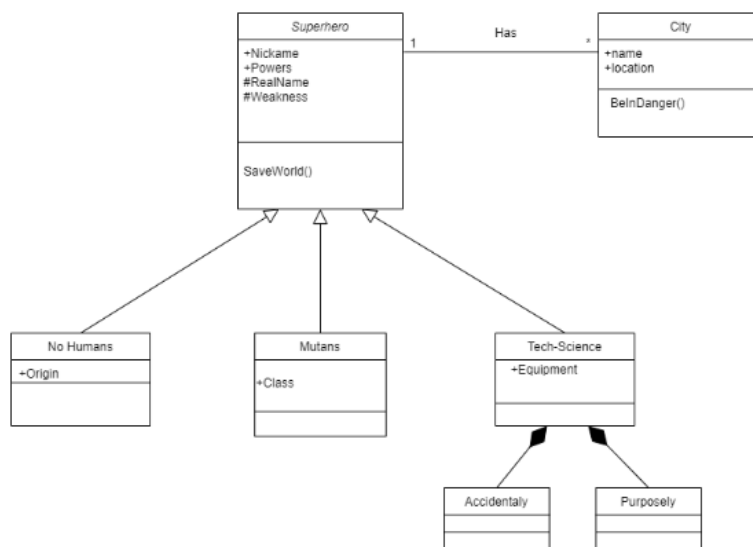


Figura 5.2: Menú de juegos de la aplicación.

UML: CLASS DIAGRAM

A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

In which of the following classes on the diagram would superman belong?



send

Figura 5.3: Ejemplo de un reto.

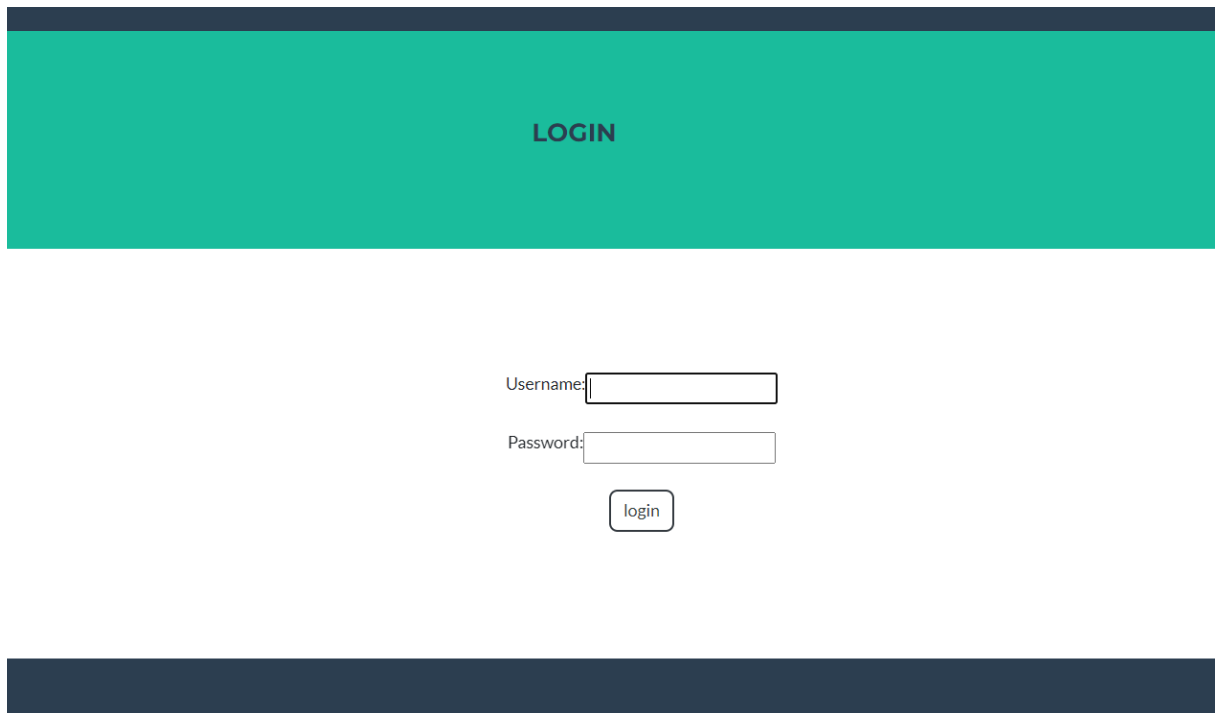


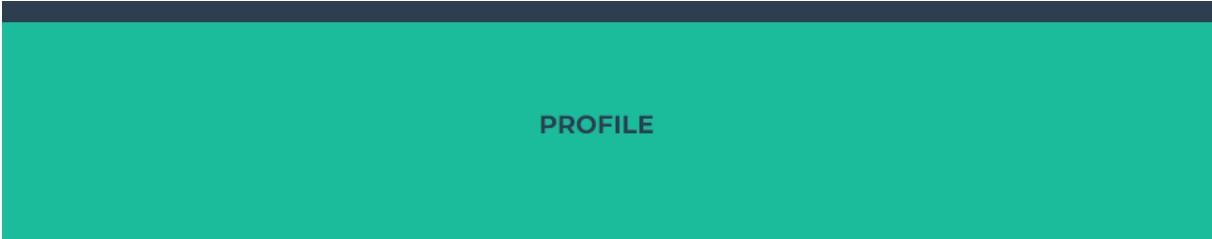
Figura 5.4: Página de inicio de sesión.

5.6. Perfil del usuario

Una vez se ha iniciado sesión y el usuario está registrado, se accede al perfil. Dentro de este se encuentran los puntos acumulados por haber superado retos y acceso a varias funciones. Desde este punto, es en el único punto desde el que es posible acceder a poder subir retos y poder crear juegos. En la parte inferior está el botón para poder cerrar sesión y volver a la aplicación como usuario anónimo.

5.7. Subida de retos

Si el usuario ha accedido hasta la página de subida de retos, para poder continuar, deberá rellenar un formulario. Se supone que esta funcionalidad solo se le concede a usuarios de confianza, por tanto, aunque el formulario sí que sea resistente a intentos de subida con campos vacíos, no está diseñado para comprobar que los campos rellenados sean coherentes con lo que se espera recibir. Se asume que el usuario al que se le han concedido estos permisos subirá retos coherentes y de manera responsable. También hay que tener en cuenta que, en este punto, se pueden subir imágenes que se guardarán directamente en la máquina que hospeda la web en He-



Hello, Jorge!
Points: 45

- START!
- UPLOAD CHALLENGE
- CREATE A GAME!
- Logout

Figura 5.5: Perfil del usuario.

CREATE CHALLENGE

Fill in the form to upload a challenge

Challenge Name:

Question:

Answer:

Choose the type of the UML diagram:

Class Diagram▼

Choose the image of the UML diagram:

Seleccionar archivo

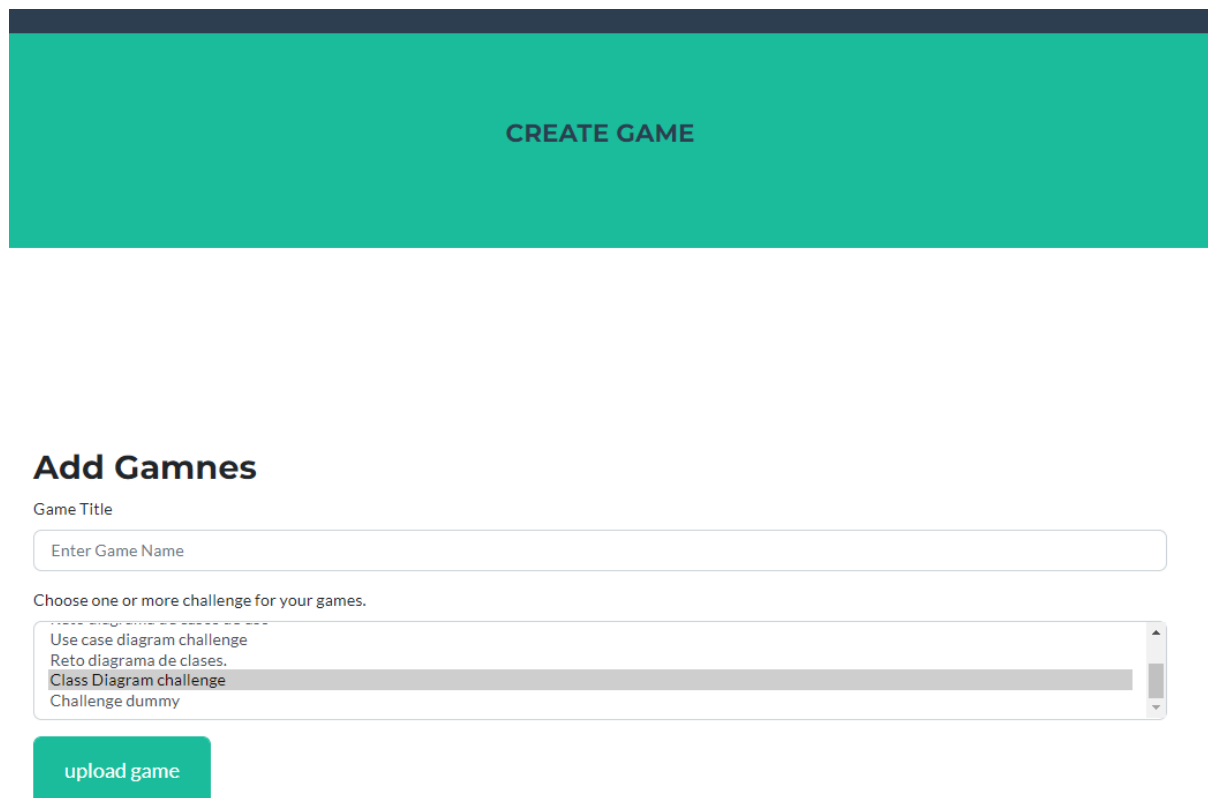
Ninguno archivo selec.

Set the points of the challenge:

upload challenge

Figura 5.6: Formulario de subida de retos.

roku, por tanto, si esto estuviera abierto al público en general, se podría generar una sobrecarga de memoria en la máquina que hospeda la web. El formulario es bastante claro con los campos a rellenar, la imagen la debemos seleccionar haciendo click en el campo que pide la misma, y subiendo la imagen deseada desde la memoria del ordenador desde el que se esté haciendo la solicitud. Para completar correctamente la subida de reto es necesario pulsar el botón situado al final de la página.



CREATE GAME

Add Gamnes

Game Title

Choose one or more challenge for your games.

- Use case diagram challenge
- Reto diagrama de clases.
- Class Diagram challenge**
- Challenge dummy

upload game

Figura 5.7: Formulario de creación de juegos.

5.8. Creación de juegos

Esta página funciona de manera muy parecida a la anterior, y solo tendrá acceso aquel usuario que se ha registrado a través del inicio de sesión. Consiste, al igual que en el módulo anterior, en rellenar un formulario sencillo. Es necesario añadir un título para el juego y seleccionar en el menú desplegable uno o varios retos que compondrán dicho juego. Para completar la subida del juego hay que pulsar el botón que hay al final de la página.

Capítulo 6

Experimentos y validación

El fin de este proyecto es ofrecer una aplicación web que pueda ser utilizada por docentes y estudiantes para realizar dos tareas específicas: Poder crear juegos compuestos por gymkhanas y poder resolver los mismos.

Los primeros experimentos y validaciones se llevaron a cabo en dos fases; la fase de validación en local, sobre WSL, y la fase de validación con el software desplegado en Heroku.

6.1. Validación sobre un entorno WSL.

Para validar de manera prácticamente inmediata el desarrollo de nuevas funcionalidades o los cambios que se han ido añadiendo, se ha utilizado, en este caso, un subsistema de Windows para Linux, del que se habla más en profundidad en la sección 3.10 de este documento. Este tipo de entorno es ideal para el desarrollo de software debido a que puedes emular un entorno con el SO y con las dependencias y librerías similares a las que tendría un servidor Linux donde se acabará alojando el código de la aplicación.

A nivel de control de versiones, las validaciones sobre WSL se han hecho en la rama *develop*, donde se añadían las modificaciones de software que requería el proyecto en cada punto.

De esta manera, durante el desarrollo se ha validado el correcto funcionamiento de cada característica, de manera individual y progresiva, en un servidor web desplegado en el *localhost*. Una vez aprobadas las nuevas funcionalidades y cambios se llevaba a cabo la segunda fase de validación.

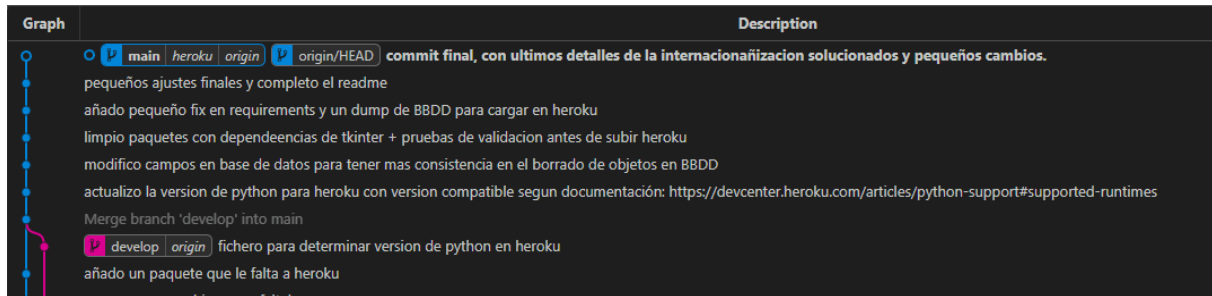


Figura 6.1: Gráfico de Git del repositorio.

6.2. Validación en la plataforma Heroku.

Después de la primera fase de desarrollo y validación se despliega el software en la plataforma Heroku, como se vio en la sección 4.5 de este proyecto.

Hay que remarcar que, antes del despliegue en Heroku, el proyecto requería ciertas modificaciones a nivel de archivos de configuración o referencias a ficheros y rutas internas. Esto es debido a que el software va a alojarse en otra máquina, por tanto, la arquitectura de ficheros no coincide con lo desplegado en el servidor WSL local. Esto es fácilmente manejable gracias al control de versiones de Git, donde la referencia estos ficheros se dejaba ajustada con los parámetros necesarios de WSL en la rama *develop* y en la rama *main* lo correspondiente a las necesidades de la máquina virtualizada de Heroku. De este modo, cada vez que se completaba una fase de validación en local, y se requería emprender una validación en Heroku, era necesario mergear y añadir los últimos cambios hechos sobre el software a la rama *main* del proyecto. Una vez hecho eso, si se encontraba algún problema durante esta fase de validación relacionado con Heroku, se añadían los *bugfix* sobre esta rama. Se muestra un ejemplo de esto en la figura 6.1.

En ambas fases el proceso de validación era el mismo. A través de pruebas manuales, probando las nuevas funcionalidades y comprobando que no entraban en conflicto con las funcionalidades ya validadas en fases anteriores del desarrollo. Por ejemplo, si se quería validar que funcionaba la creación de juegos o retos, se subía en cada caso un reto “*dummy*”, cuya única función era comprobar que los flujos de trabajo del software funcionaban correctamente. Esto se ha hecho con todas las funcionalidades requeridas durante el desarrollo del proyecto, hasta alcanzar el resultado actual.

Capítulo 7

Conclusiones

7.1. Consecución de objetivos

En este proyecto sí se ha conseguido desplegar una aplicación web que puede llenarse de juegos y retos que divulguen sobre el lenguaje de modelado unificado. Se ha hecho desde cero, con tecnologías de código abierto y/o servicios gratuitos. La aplicación es sencilla, limpia y fácil de entender para gente de todas las edades. Se ha conseguido también internacionalizarla usando especificaciones y estándares conocidos por toda la comunidad de desarrolladores.

En cuanto los objetivos específicos:

1. **Primera versión de la aplicación funcionando en *localhost*.** Este objetivo ha sido alcanzado con creces, ha sido el objetivo más abstracto y con más tareas a realizar. En este objetivo va incluido el diseño inicial del proyecto, elección de los estilos y la mayor parte de lo visto en el Capítulo 4. Cabe destacar que desde este punto de partida inicial se han añadido diferentes cambios críticos para el proyecto.
2. **Agrupación de juegos y retos, y navegación en menú entre ellos.** Para llevar a cabo este objetivo fue necesario reorganizar parte de la estructura de la base de datos inicial, especialmente los campos de *Challenges* y *Games*. También para poder gestionar la navegación entre retos contenidos en juegos se añadieron varias funciones extras en el archivo *utils.py*
3. **Internacionalización del proyecto.** Gracias al estándar i18n se ha podido alcanzar este objetivo con relativa facilidad. Fue necesario cambiar prácticamente todos los textos de

la plantilla e invertir tiempo en la traducción y compilación de los archivos *.po*, pero considero que es una pieza clave del proyecto y un objetivo alcanzado con éxito.

4. **Sistema de gestión de usuarios que puedan subir retos y crear juegos.** Se puede decir que este ha sido el objetivo más complicado del proyecto. Desarrollar un *login* en Django en principio es sencillo, y te da muchas facilidades la documentación, pero en este caso lo complicado era diseñar la aplicación de tal forma que solo ciertos usuarios pudiesen registrarse. El objetivo de estar registrado es poder subir retos, lo que conlleva poder subir archivos a la base de datos del proyecto, y, por tanto, es necesario un control de acceso más restrictivo en este caso. Se decidió en este punto que solo el administrador o administradores del proyecto tengan la potestad de crear usuarios.
5. **Añadir sistema de puntuación.** Para poder cumplir este objetivo fue necesario cambiar la estructura de la base de datos de nuevo y añadir ciertas funciones en *utils.py* para poder gestionar toda la parte de retorno de puntuaciones.
6. **Despliegue de la aplicación y hacerla pública.** Este último objetivo se logró de igual forma que los otros. Llegados a este punto del desarrollo este último fue relativamente sencillo, gracias a la documentación de Heroku y que está diseñado y orientado para ser utilizado en este tipo de proyectos.

Gracias a esto, se ha hecho accesible una herramienta que pueden usar docentes y estudiantes de todo el mundo. En contra partida, es esto último lo que no se ha conseguido. En términos generales no se ha logrado que ningún docente o alumno (más allá de pruebas específicas para el proyecto) usen la aplicación desarrollada en este proyecto. No obstante, tanto el código como la aplicación se quedará desplegada en la red de Internet para que se tenga acceso desde cualquier parte del mundo en cualquier momento.

7.2. Aplicación de lo aprendido

Este proyecto ha puesto en práctica muchos de los conocimientos enseñados en el Grado en Ingeniería en Sistemas Audiovisuales y Multimedia y también otros adquiridos de manera autodidacta o en el entorno laboral. Especialmente los relacionados con el desarrollo de software.

Se puede decir que las asignaturas que más han aportado en cuanto a conocimientos utilizados en este proyecto es la de Construcción de Servicios y Aplicaciones Audiovisuales en Internet y Protocolos para la Transmisión de Audio y Vídeo en Internet. En estas asignaturas aprendí mucho sobre desarrollo web (HTML, JavaScript y CSS) y programación con Python, aprendiendo a usar la programación orientada a objetos, estructuras de datos, extracción y escritura de información de ficheros y, en general, profesionalizarme con el uso de una herramienta tan potente como es Python. Además, hay que remarcar todo el conocimiento sobre Linux, programación, protocolos de comunicación, arquitectura de sistemas, tratamiento de imágenes y muchos más conocimientos adquiridos en el resto de las asignaturas que han sido indispensables para llevar a cabo este trabajo.

También se han aplicado muchos contenidos de la asignatura optativa Laboratorio de Tecnologías Audiovisuales en la Web que, en mi caso específico yo no cursé, ya que durante la movilidad internacional que realicé, convalidé esa asignatura con otra llamada Databases. No obstante, se me facilitó algunos contenidos docentes de esta asignatura para que me ayudaran en el desarrollo de este proyecto. Al mismo tiempo, la asignatura de base de datos cursada durante la movilidad también me aportó conocimientos que han sido esenciales para entender los modelos de entidad relación y los modelos de bases de datos utilizadas para la aplicación web.

7.3. Lecciones aprendidas

El desarrollo de este trabajo ha servido para reforzar y ampliar conocimientos que adquirí durante la carrera y, además, adquirir otros nuevos y aprender sobre tecnologías y conceptos anteriormente desconocidos para mí.

1. **Lenguaje unificado modelado.** El objetivo de este proyecto es tratar de divulgar y expandir conocimientos sobre este lenguaje de modelado y para ello es naturalmente necesario conocer este mismo profundamente. Aunque antes de empezar este proyecto sí que sabía acerca de UML y usado muchos de sus esquemas en varias ocasiones, sabía bastante menos de lo que se hoy en día. La investigación realizada para el desarrollo de este proyecto me ha aportado mucha nueva información, anteriormente desconocida para mí, sobre este lenguaje que considero que me será muy útil en todos los procesos de modelado de software a los que me enfrente en el resto de mi carrera profesional.

2. **Django.** La tecnología principal utilizada en el desarrollo del proyecto ha sido Python y Django. A pesar de que he trabajado anteriormente con Python y es una tecnología en la que me siento cómodo desarrollando, no había desarrollado ningún proyecto Django anterior a este. Creo que esta tecnología es idónea si estás acostumbrado a trabajar con Python y haber podido aprender a usar esta potente herramienta durante el desarrollo de este proyecto me ha aportado mucho para mi *background* profesional con Python.
3. **Diseño de datos.** Enfrentarme al diseño e implementación de una base de datos, teniendo que estudiar bien los campos y las relaciones entre estos para que una aplicación funcione correctamente ha sido un reto nuevo anteriormente desconocido para mí.
4. **Internacionalización y i18n.** Gracias al desarrollo de este proyecto he podido conocer un estándar tan potente y útil como i18n, es una de las tecnologías que más me ha sorprendido descubrir, me parece sencilla, completa y útil en muchos aspectos y considero que es fundamental en prácticamente cualquier proyecto web con ámbito internacional.
5. **Heroku.** También he aprendido bastante sobre Heroku, durante el desarrollo he tenido que leer mucha documentación para llegar a entender cómo funciona y poder hacer un despliegue de código limpio. Aprender sobre este tipo de servicios es muy útil ya que fácilmente puedes tener un pequeño proyecto web con acceso mundial y gratuitamente.

7.4. Trabajos futuros

Como todos los proyectos web, esta aplicación permite nuevas implementaciones, mejoras y cambios que pueden hacer de *Gymkhana App* una herramienta más completa, óptima, segura y útil en líneas generales. Algunas de las líneas de trabajo futuras podrían ser las siguientes:

- Hacer más segura la aplicación, de manera que así puedan subir retos y juegos cualquier usuario que se quiera registrar, asegurándose de que con un software automatizado no puedan lanzar cientos de peticiones que puede que haga peligrar la integridad del proyecto.
- Hacer compatible la interfaz de usuario con dispositivos móviles y tabletas. Esta aplicación se ha desarrollado para visualizarse en una web a través de navegadores convencio-

nales sobre ordenadores, aunque parte de ella sí es lo que se denomina *responsive*, no se ha enfatizado en hacerla totalmente compatible con dispositivos móviles.

- Ampliar la información que se tiene de un usuario registrado, como por ejemplo los retos ya superados, porcentajes y estadísticas del porcentaje de juegos jugados y por jugar, y otra información que detalle más logros superados y pendientes para jugador.
- Si se logra el primer objetivo de esta lista, se podría implementar el inicio de sesión a través de Google o Facebook con la intención de hacer más sencilla la implicación del usuario con *Gymkhana App*.
- Evolucionar o mejorar el sistema de retos para que el juego consista en algo más que en dar una respuesta a un reto, algo más interactivo o completo.
- Como todos los proyectos de software, siempre se pueden mejorar la resistencia a errores. Someter la aplicación a una fase de QA¹ para asegurar el buen funcionamiento de la aplicación en cualquier caustica a la que esté sometido el programa.
- Conseguir difundir el uso de la aplicación entre docentes y alumnos y obtener de estos un buen muestreo tanto de opiniones y resultados para comprobar si realmente *Gymkhana App* cumple el propósito de familiarizar y ayudar a comprender de mejor manera conceptos del lenguaje de modelado unificado.

¹QA: De las siglas Quality Assurance, consiste en asegurar que el producto final sea de calidad, robusto y funcione correctamente

Apéndice A

Anexo I: Ejemplos de Diagramas UML

A.1. Diagrama de clases

En este diagrama de clases, hay una clase padre y dos clases hijas. En este caso la clase padre cuenta con unos atributos y métodos comunes, los cuales comparte con las clases hijas. Las clases hijas en cambio tienen cada una unos atributos y métodos que no pueden compartir entre ellas ni con la clase padre.

A.2. Diagrama de objetos

Los diagramas de objetos están fuertemente relacionados con los diagramas de clases, tanto es así que hasta comparten símbolos y figuras. Básicamente para seguir con el objeto anterior de los vehículos, en este ejemplo se puede ver los datos o “fotografía” de un objeto del sistema en un momento determinado del tiempo.

A.3. Diagrama de despliegue

Este diagrama de despliegue modela la arquitectura en tiempo de ejecución de un sistema. Se muestra la configuración de los elementos de hardware, identificados como nodos. Y muestra las trazas de elementos y artefactos del software en estos nodos.

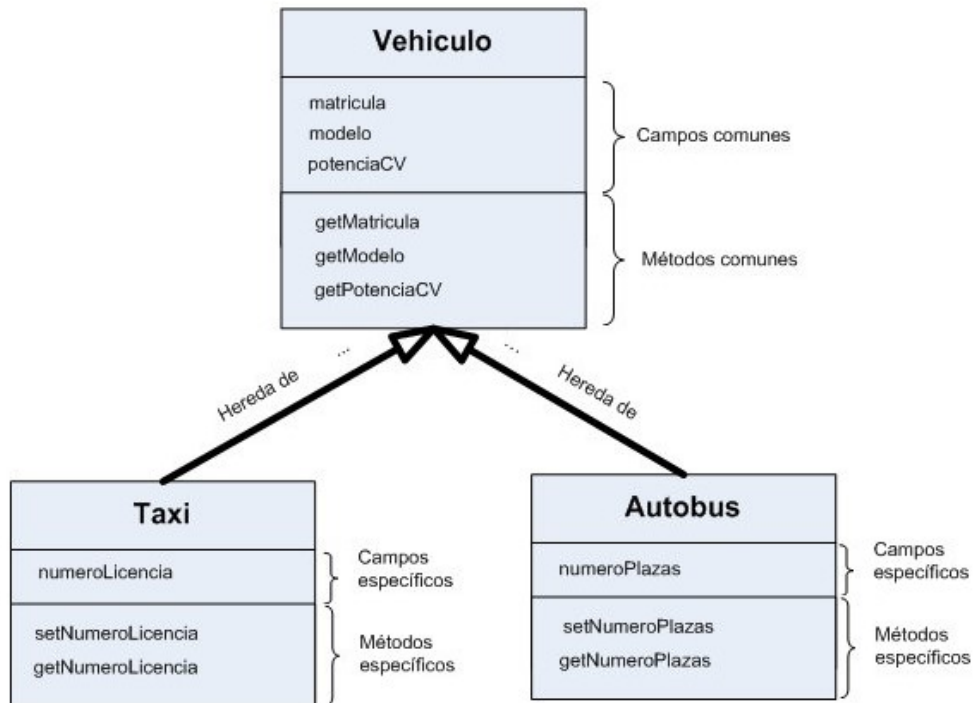


Figura A.1: Ejemplo de diagrama de clases.

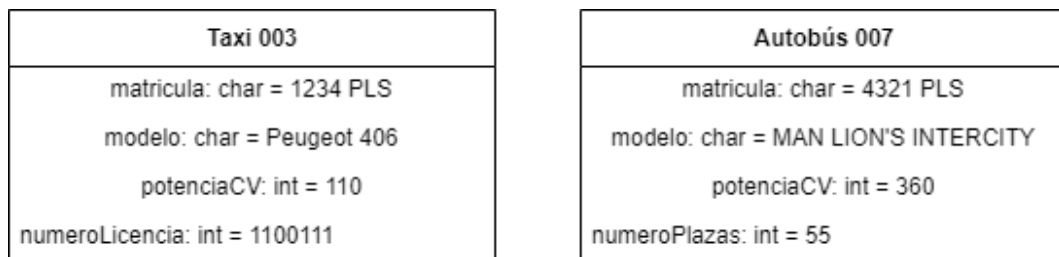


Figura A.2: Ejemplo de diagrama de objetos.

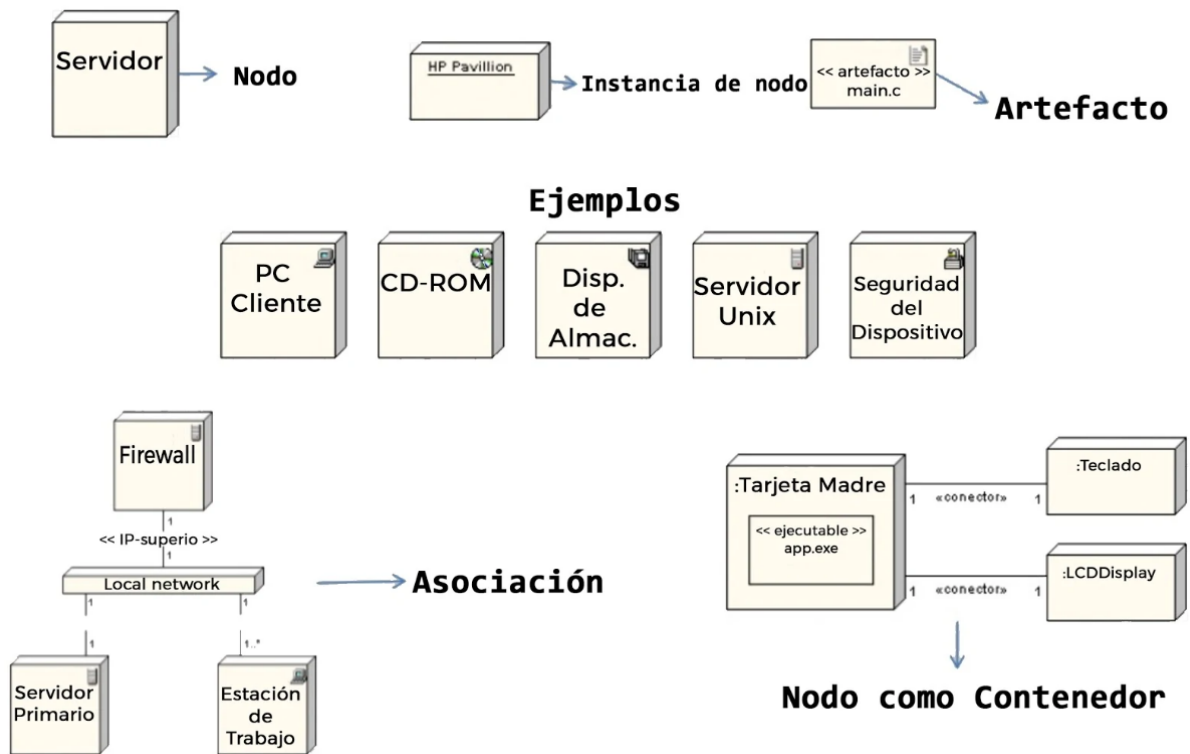


Figura A.3: Ejemplo de diagrama de despliegue.

A.4. Diagrama de componentes

En este diagrama de componentes se puede visualizar la estructura y la funcionalidad de un software de correo electrónico. Muestra la interacción a través de sus tres interfaces. La línea discontinua etiquetada con “use” indica que el usuario es dependiente de esa interfaz (management port) para supervisar y administrar el correcto funcionamiento del sistema.

A.5. Diagrama de paquetes

En este ejemplo de diagrama de paquetes tiene la intención de mostrar la organización y disposición de los diferentes elementos que componen una aplicación web para compras. Las carpetas representan la anidación de elementos del software, y estas a la vez se organizan jerárquicamente dentro del diagrama.

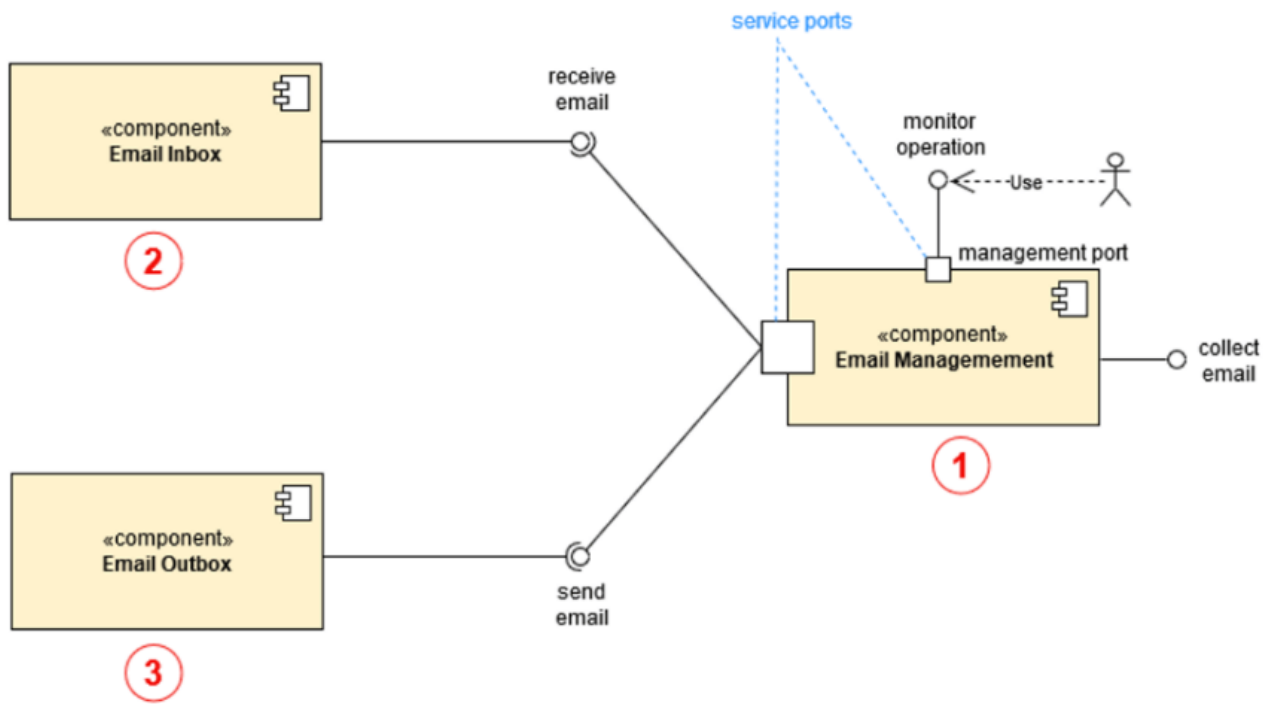


Figura A.4: Ejemplo diagrama de componentes.

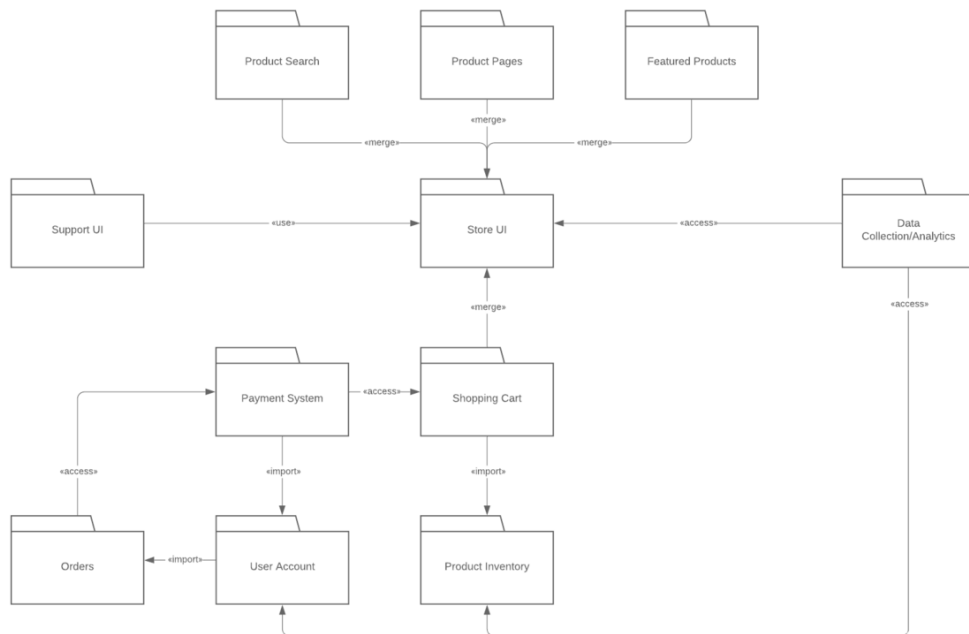


Figura A.5: Ejemplo diagrama de paquetes.

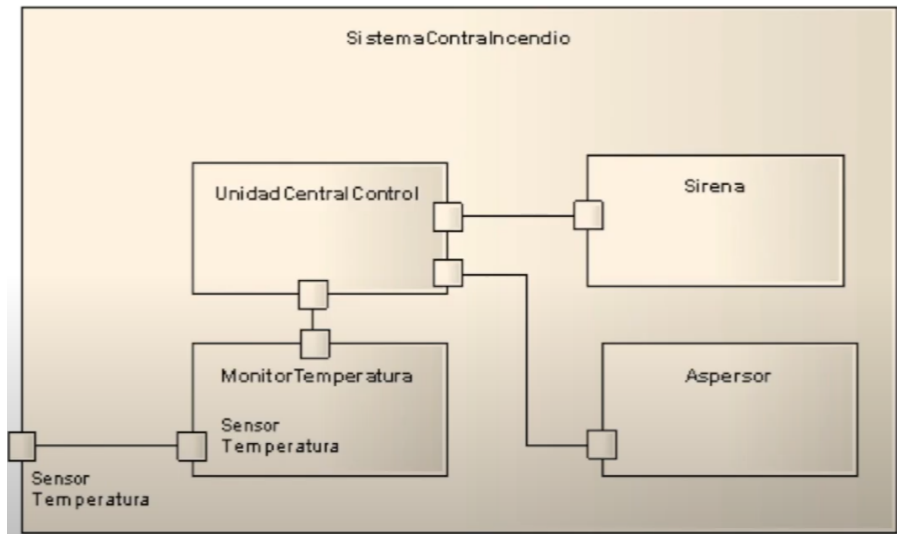


Figura A.6: Ejemplo diagrama de estructura compuesta.

A.6. Diagrama de estructura compuesta

En este ejemplo podemos apreciar el diagrama de estructura compuesta de un sistema contra incendios. Se puede ver que la unidad central de control está compuesta por componentes más pequeños (sirena, aspersores, sensores, monitores...), que a su vez pertenecen al sistema general.

A.7. Diagrama de actividad

Este ejemplo de diagrama es uno de los usados para los retos de Gymkhana App. Este tipo de diagramas es de los más conocidos, en este caso se trata de seguir el flujo de trabajo, pudiendo actuar de distinta manera dependiendo de las respuestas del sistema a diferentes preguntas (representadas con rombos).

A.8. Diagrama de casos de uso

Este ejemplo modela la funcionalidad de un “Sistema Apocalipsis Zombie” con los diferentes casos de uso representado con elipses. Se muestra también la interacción que tiene el actor (superviviente) con el sistema. Este diagrama ha sido diseñado para uno de los primeros retos de la aplicación Gymkhana App.

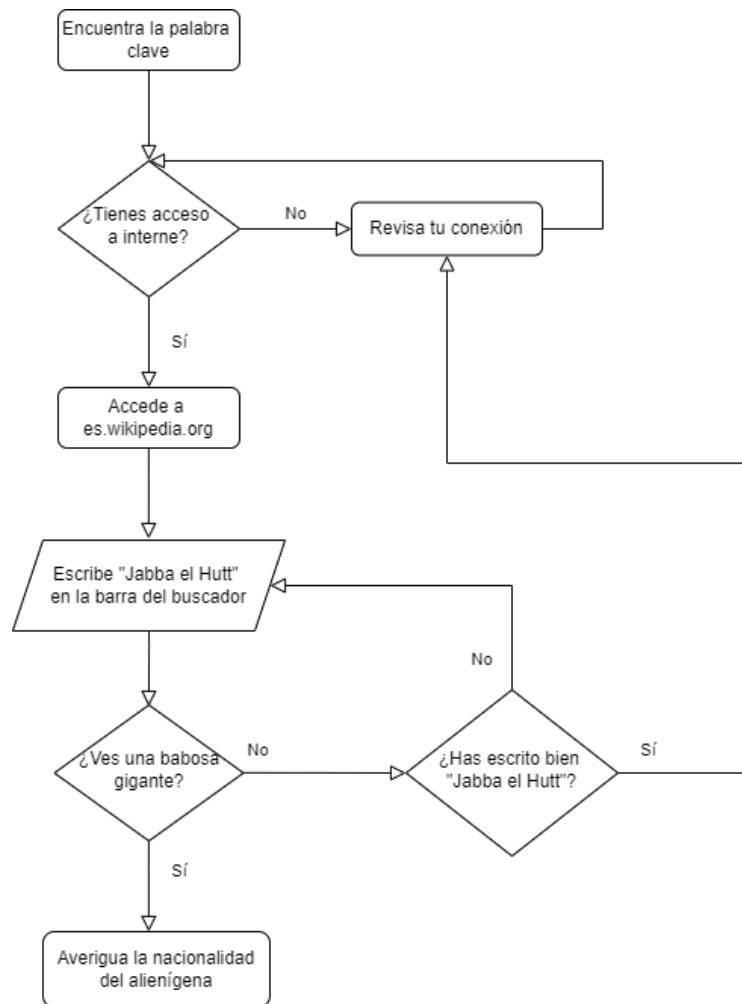


Figura A.7: Ejemplo diagrama de actividad.

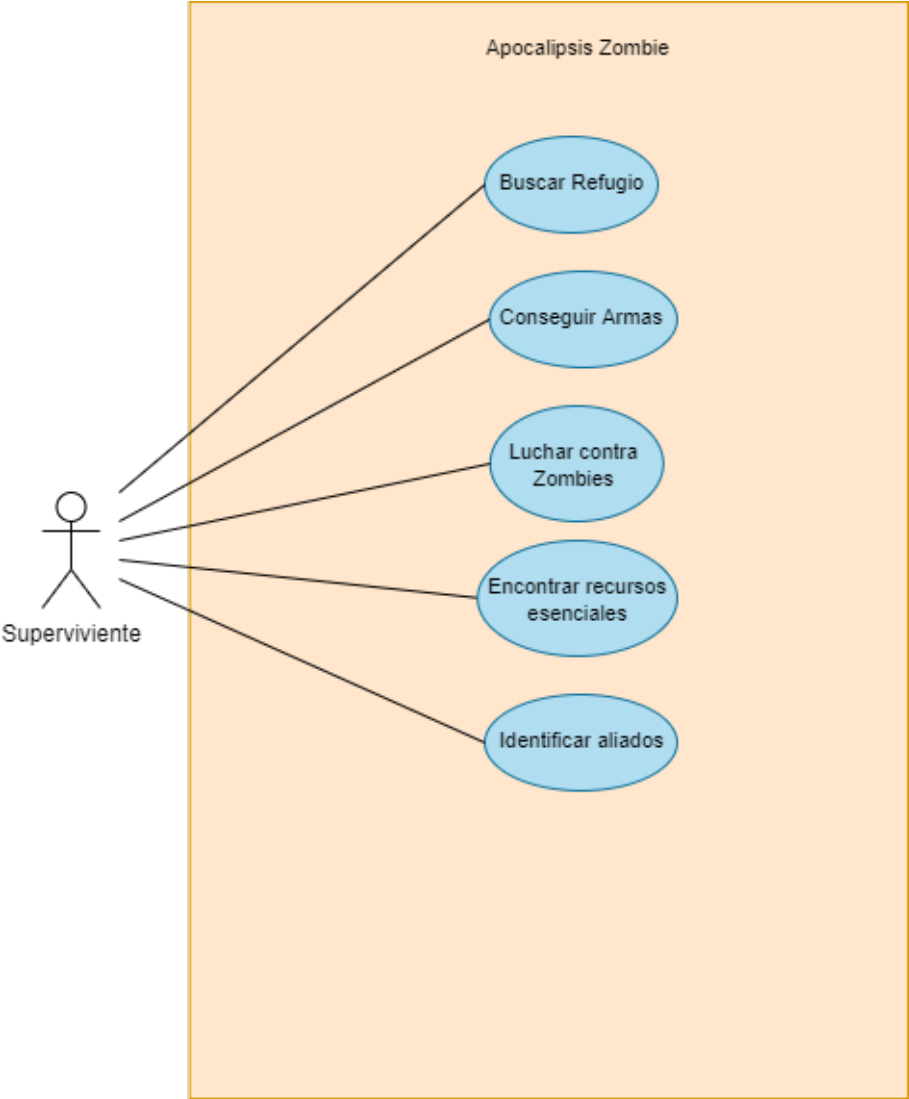


Figura A.8: Ejemplo diagrama casos de uso.

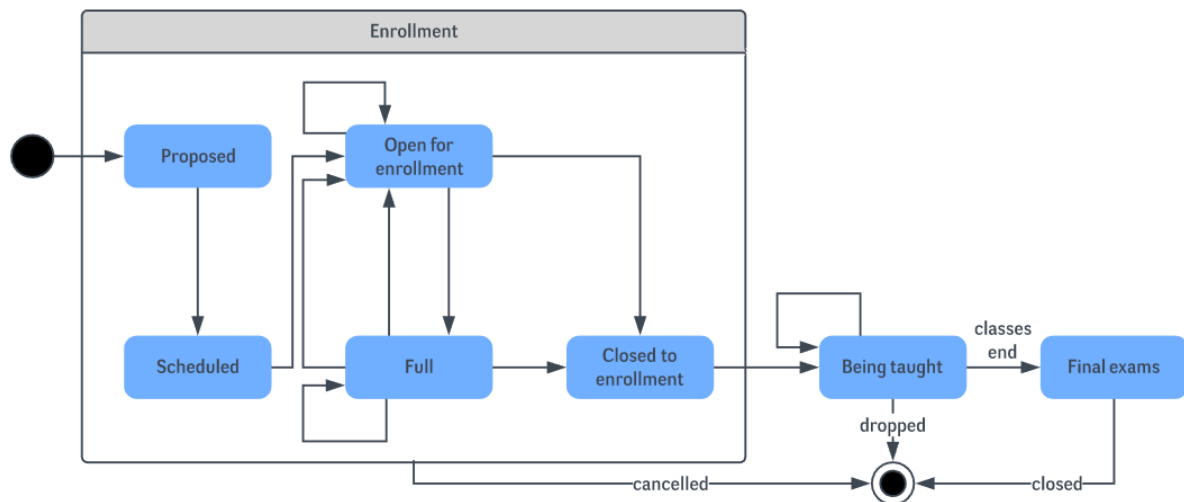


Figura A.9: Ejemplo diagrama de máquina de estados.

A.9. Diagrama de máquina de estados

En este ejemplo se muestra un diagrama del proceso de inscripción y finalización de una asignatura de la universidad. Se puede ver como el estado pasa de tratar de apuntarse a una asignatura, a poder matricularse en esta y pasar los exámenes finales.

A.10. Diagrama de interacción

Como dentro de estos tipos de diagramas se pueden clasificar cuatro más, en este apartado se aportan ejemplos sencillos de alguno de ellos.

Los diagramas de colaboración se centran en los aspectos estructurales y la arquitectura de los objetos de un software. En el ejemplo se muestra el flujo de organización de una tienda online, con el tipo de mensajes y de llamadas que se hacen entre los objetos del sistema.

Los diagramas de secuencia se centran en representar las interacciones entre los eventos de un sistema. En el ejemplo se muestra la secuencia de eventos de un software de manejo de una agenda, enlazada a una dirección de correo electrónico.

Los diagramas de tiempos se usan para representar la línea de vida de un objeto en una instancia y en el tiempo. Se representan los cambios con nivel alto y bajo con señales. En el ejemplo se puede ver un diagrama del comportamiento de los objetos que forman un sistema contraincendios.

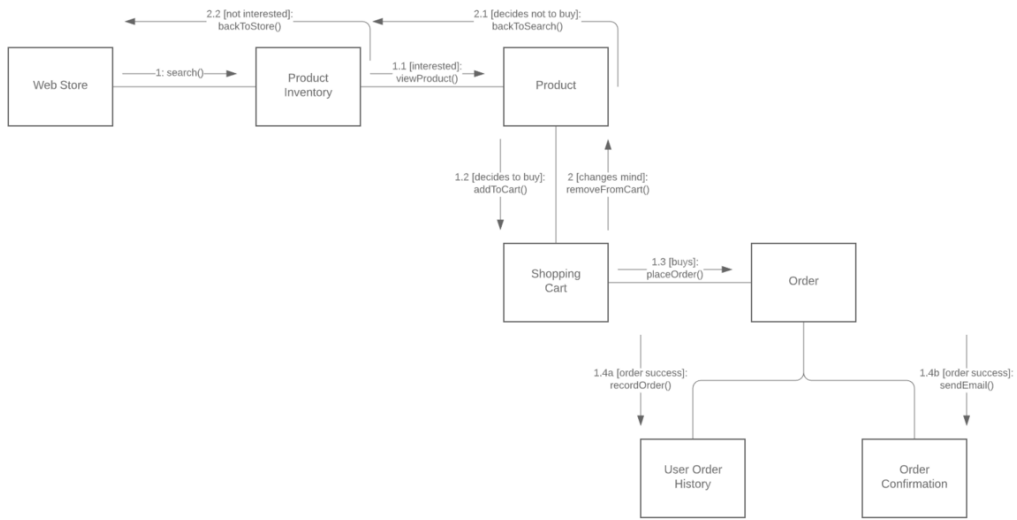


Figura A.10: Ejemplo diagrama de colaboración

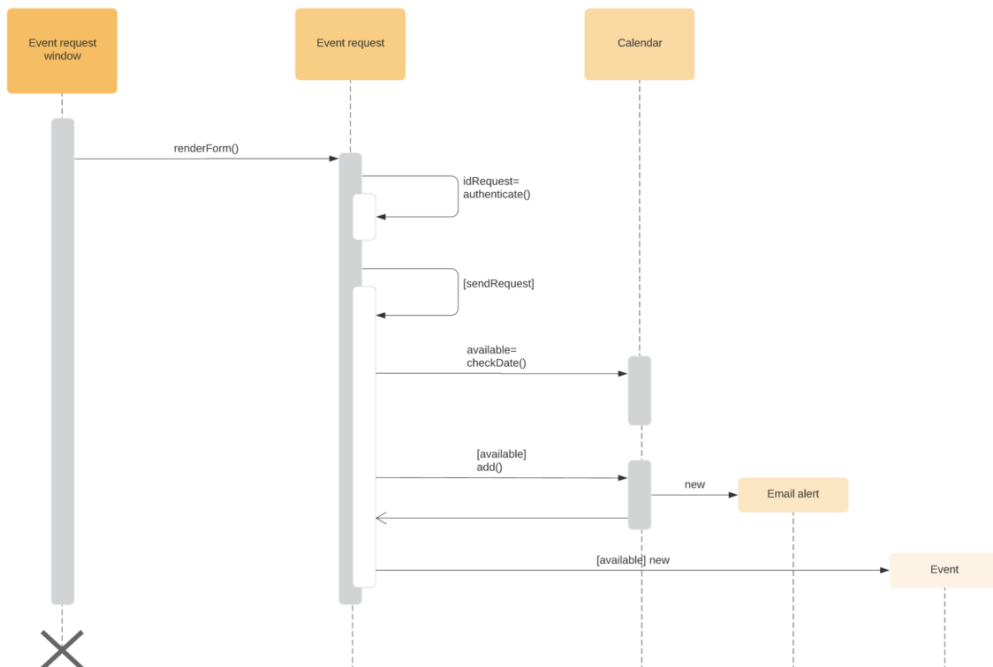


Figura A.11: Ejemplo diagrama de secuencia

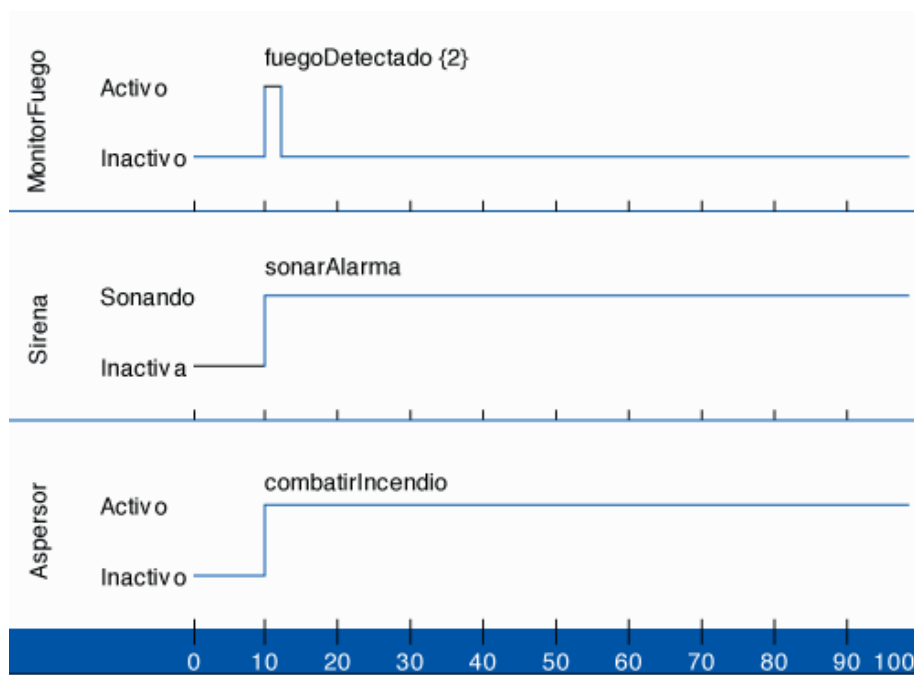


Figura A.12: Ejemplo diagrama de tiempos.

Bibliografía

- [1] Documentación django.
<https://docs.djangoproject.com/en/3.2/>.
- [2] Documentación heroku.
<https://devcenter.heroku.com/categories/reference>.
- [3] Documentación heroku postgres.
<https://devcenter.heroku.com/articles/heroku-postgresql>.
- [4] Documentación python.
<https://www.python.org/>.
- [5] Documentación wsl.
<https://docs.microsoft.com/es-es/windows/wsl/install>.
- [6] Especificación uml.
<https://www.omg.org/spec/UML/2.5.1/PDF>.
- [7] Página de bootstrap.
<https://getbootstrap.com/>.
- [8] Página de diagrams.net.
<https://www.diagrams.net/>.
- [9] Página de git.
<https://git-scm.com/>.
- [10] Página de postgresql.
<https://www.postgresql.org/about/>.

[11] Página de sqlite.

<https://www.sqlite.org/about.html>.

[12] Sitio web oficial visual studio code.

<https://code.visualstudio.com/>.