



Universidad  
Rey Juan Carlos

GRADO EN INGENIERÍA EN SISTEMAS AUDIOVISUALES  
Y MULTIMEDIA

Curso Académico 2020/2021

Trabajo Fin de Grado

TÍTULO DEL TRABAJO EN MAYÚSCULAS

Autor : Jorge De Pablo Martínez

Tutor : Dr. Gregorio Robles Martínez



# Trabajo Fin de Grado

Título del Trabajo con Letras Capitales para Sustantivos y Adjetivos

**Autor :** Jorge De Pablo Martínez

**Tutor :** Dr. Gregorio Robles Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el día                      de  
de 202X, siendo calificada por el siguiente tribunal:

**Presidente:**

**Secretario:**

**Vocal:**

y habiendo obtenido la siguiente calificación:

**Calificación:**

Fuenlabrada, a                      de                      de 202X



*Dedicado a  
mi familia / mi abuelo / mi abuela*



# Agradecimientos

Aquí vienen los agradecimientos... Aunque está bien acordarse de la pareja, no hay que olvidarse de dar las gracias a tu madre, que aunque a veces no lo parezca disfrutará tanto de tus logros como tú... Además, la pareja quizás no sea para siempre, pero tu madre sí.





# Resumen

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

La aplicación web Ghymkhana App

- ¿De qué va este proyecto? ¿Cuál es su objetivo principal?
- ¿Cómo se ha realizado? ¿Qué tecnologías están involucradas?
- ¿En qué contexto se ha realizado el proyecto? ¿Es un proyecto dentro de un marco general?

Lo mejor es escribir el resumen al final.



# Summary

Here comes a translation of the “Resumen” into English. Please, double check it for correct grammar and spelling. As it is the translation of the “Resumen”, which is supposed to be written at the end, this as well should be filled out just before submitting.



# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Contexto . . . . .	1
1.2. Estructura de la memoria . . . . .	2
<b>2. Objetivos</b>	<b>3</b>
2.1. Objetivo general . . . . .	3
2.2. Objetivos específicos . . . . .	3
2.3. Planificación temporal . . . . .	3
<b>3. Estado del arte</b>	<b>5</b>
3.1. UML: Unified Modeling Language . . . . .	5
3.1.1. Diagramas Estructurales . . . . .	6
3.1.2. Diagramas de Comportamiento . . . . .	8
3.2. Python . . . . .	8
3.3. Django . . . . .	10
3.3.1. Modelo Vista Controlador (MVC) . . . . .	10
3.4. Git . . . . .	11
3.5. Bootstrap . . . . .	12
3.6. Heroku . . . . .	12
<b>4. Diseño e implementación</b>	<b>15</b>
4.1. Arquitectura general . . . . .	15
4.2. Módulo Vistas-Controlador . . . . .	15
4.3. Módulo Templates-Vistas . . . . .	18
4.3.1. Internacionalización . . . . .	21

4.4. Módulo Models-Modelo . . . . .	21
<b>5. Experimentos y validación</b>	<b>25</b>
<b>6. Resultados</b>	<b>27</b>
<b>7. Conclusiones</b>	<b>29</b>
7.1. Consecución de objetivos . . . . .	29
7.2. Aplicación de lo aprendido . . . . .	29
7.3. Lecciones aprendidas . . . . .	30
7.4. Trabajos futuros . . . . .	30
<b>A. Manual de usuario</b>	<b>31</b>
<b>B. Anexo I: Ejemplos de Diagramas UML</b>	<b>33</b>
<b>Bibliografía</b>	<b>35</b>

# Índice de figuras

3.1. Jerarquía de diagramas en UML 2.2 (como diagrama de clases). . . . .	6
3.2. Top uso de lenguajes de programación alojados en GitHub . . . . .	9
4.1. Estructura de la relación modelo-vista-controlador. . . . .	16
4.2. Diagrama entidad-relación de la aplicación Ghymkhana App. . . . .	24





# Capítulo 1

## Introducción

La siguiente 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 de lenguaje unificado modelado.

### 1.1. Contexto

Gymkhana App es una aplicación web con simples juegos que consisten en la resolución de diagramas UML (lenguaje unificado modelado, 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 mini-juegos consisten en 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 entre nueve y doce años.

Se trata de divulgar y enseñar conceptos de UML y que, a al vez, los más jóvenes se familiaricen con estos últimos a través los sencillos juegos y retos que contiene Gymkhana App.

## 1.2. Estructura de la memoria

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

- Capítulo 1. Introducción. En el primer capítulo se hace una introducción al proyecto y se exponen las ideas básicas del mismo además de definir su estructura.
- En el capítulo 2 (ojo, otra referencia automática) se muestran los objetivos del proyecto.
- A continuación se presenta el estado del arte en el capítulo 3.
- ...

# 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 unificado modelado y sus tipos de diagramas. El desarrollo del este proyecto se ha hecho con Django y apoyándose principalmente en otras tecnologías y servicios como Heroku, Git y Bootstrap.

### 2.2. Objetivos específicos

Los objetivos específicos se pueden entender como las tareas en las que se ha desglosado el objetivo general. Y, sí, también vienen en.

### 2.3. Planificación temporal

A mí me gusta que aquí pongáis una descripción de lo que os ha llevado realizar el trabajo. Hay gente que añade un diagrama de GANTT. Lo importante es que quede claro cuánto tiempo llevas (tiempo natural, p.ej., 6 meses) y a qué nivel de esfuerzo (p.ej., principalmente los fines de semana).



# 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 unificado modelado.

### 3.1. UML: Unified Modeling Language

El lenguaje unificado modelado, también conocido por sus siglas en inglés UML, 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 unificado modelado 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.

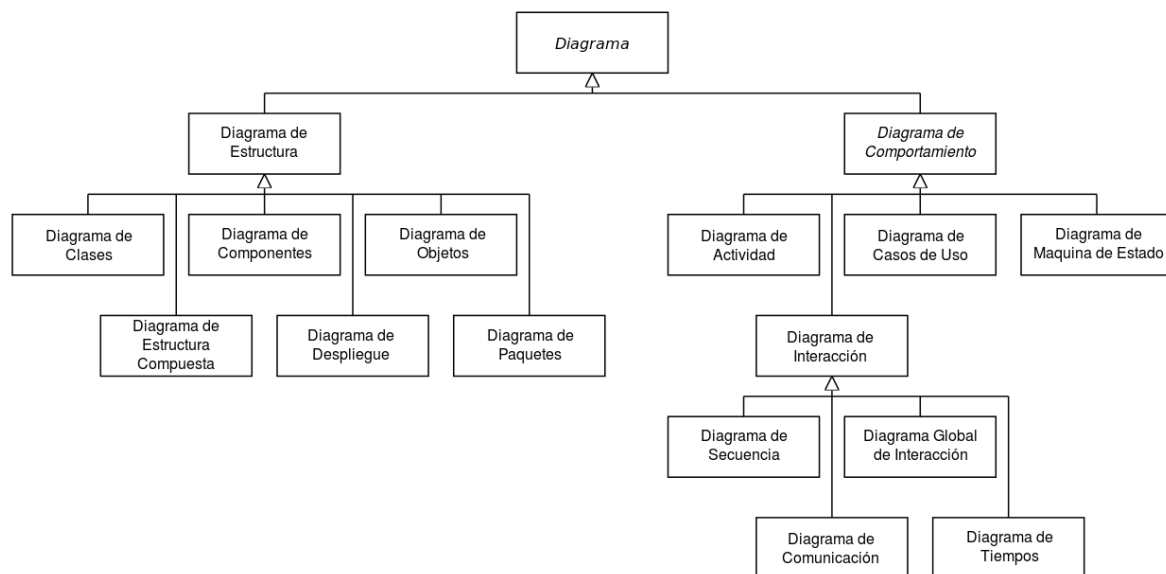


Figura 3.1: Jerarquía de diagramas en UML 2.2 (como diagrama de clases).

Desde 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 OMT<sup>1</sup>, organización que respalda UML, tiene formalmente liberada la versión 2.5.1, que es sobre la cual este proyecto se ha desarrollado. Naturalmente entre medias de han aparecido y varias versiones menores las cual 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.

### 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

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

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 y entre los objetos.

**Diagrama de despliegue** Esta clase de diagrama muestra la arquitectura de ejecución de un sistema. Incluyendo 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 como 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, 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 la misma. 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 es 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 del mismo mediante la interacción con los usuarios y/u otros sistemas.

**Diagrama de interacción** Este tipo de diagrama describen en detalle un determinado escenario de casos de uso. Ilustran la interacción entre el conjunto de objetos que cooperan en la realización de un proceso.

**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 es un lenguaje de programación interpretado, dinámico y multiplataforma con licencia de código abierto<sup>2</sup>. Su máxima es hacer un lenguaje que sea fácilmente legible y con

---

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



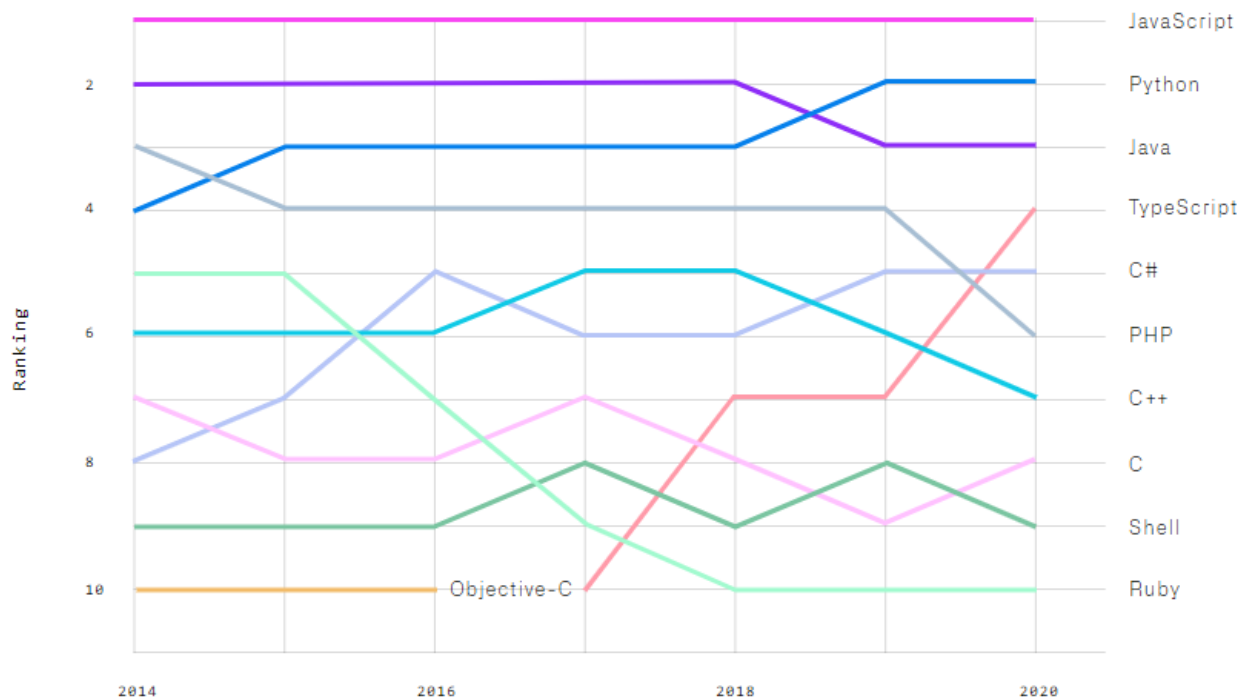


Figura 3.2: Top uso de lenguajes de programación alojados en GitHub

una empinada curva de aprendizaje. Python fue creado a principios de los 90s por Guido van Rossum en los Países Bajos, como curiosidad, saber que le debe su nombre gracias la afición que tenía su creador por el grupo de humoristas británicos Monty Python.

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 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.

Gracias a 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 fue considerada "*deprecated*" el 1 de enero de 2020.

### 3.3. Django

Django 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 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 este patrón en tres grandes módulos: modelo, vista y controlador.

- **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 del mismo 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 es un sistema de control de versiones gratuito y de código abierto diseñado para manejar cualquier tipo de proyecto, tanto en pequeños como los muy grandes con gran velocidad y eficiencia. Vino desarrollado por la mano de Linus Toval, 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 como, sobre todo, de trabajo en equipo. Te permite, crear varias ramas de desarrollo y provee de herramientas para poder unir y 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 AWS<sup>3</sup> 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

---

<sup>3</sup>Amazon Web Services

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 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 basado en HTML y CSS, así como extensiones de JavaScript adicionales. Fue desarrollado por Mark Otto y Hacob Thornton como marco de trabajo para fomentar las herramientas internas de Twitter, en 2011 se liberó 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 el punto 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 es una plataforma que ofrece servicios de computación en la nube que soporta 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 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.



# 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 como se ve en la figura 4.1, en esta figura las líneas sólidas muestran una relación directa entre los elementos y las rayadas una asociación indirecta. 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 módulos, se encarga de recibir las peticiones del módulo de vistas, gestionar internamente estas peticiones y decidir que 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

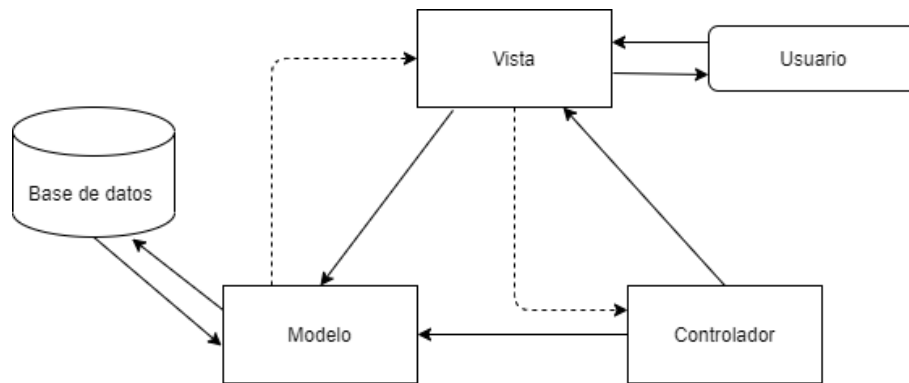


Figura 4.1: Estructura de las relación modelo-vista-controlador.

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 redirige con esta petición a la página principal o home, donde se simplemente se puede ver dos botones grandes que te permiten o bien iniciar sesión y registrarte o bien iniciar Gymkhana App.
- **/start**: Cuando inicias la Gymkhana 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 elegido, y si vienes de un reto anterior o acaba de empezar el juego. Así podemos elegir cual 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 a el reto donde el usuario debe introducir la respuesta al mismo para poder avanzar.
- **/response**: Una vez el usuario ha rellenado la repuesta al reto se envía una petición tipo GET con la información contenida del reto al que pertenece, el juego, si ha habido una reto del mismo juego anterior y la respuesta al reto. En caso de ser correcta el servidor redirigirá al usuario bien al nuevo reto, si se está jugando a un juego con varios retos, o 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 que de dicho reto, pero solo en caso de que no lo halla 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" esta 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 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 esta 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, proporcionará 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 views 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

NOTA: PODEMOS AÑADIR PANTALLAZOS CON TODA LA INTERFAZ EN LOS ANEXOS PARA RELLENAR

### 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 respuesta de los usuarios en diferentes idiomas y comprobar que coinciden con la respuesta de un reto concreto.

De la misma manera, para poder traducir las imágenes de los diagramas UML que se muestran, cada diagrama tiene que estar disponible en dos idiomas diferentes, es decir cada diagrama está representado por dos imágenes, una en inglés y otra en castellano, así se seleccionará la imagen correspondiente dependiendo del idioma del usuario.

## 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 decir que permite que múltiples procesos tengan el archivo de base de datos abierto y leer a la vez. En la figura 4.2 se puede ver el diagrama entidad-relación que define la base de datos de la aplicación, el proyecto de Django tiene más tablas a parte 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:** Una 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.
  - **awnsner:** Respuesta a la pregunta clave.
  - **image:** Campo tipo image 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.
  - **challenges:** Uno o varios retos de los contenidos en la tabla Challenges.
  - **creator:** Usuario que ha creado el juego.

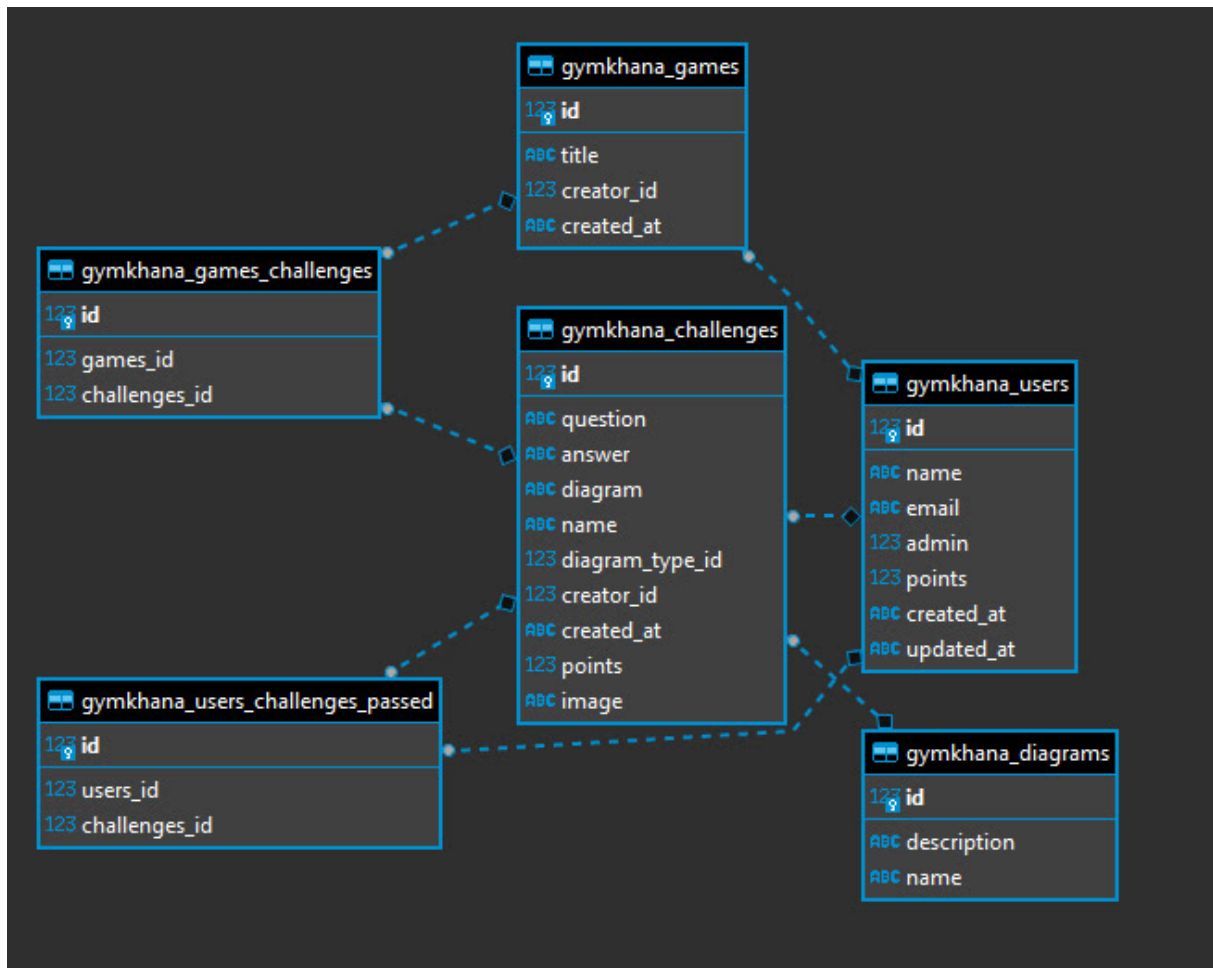


Figura 4.2: Diagrama entidad-relación de la aplicación Ghymkhana App.



## **Capítulo 5**

# **Experimentos y validación**

Este capítulo se introdujo como requisito en 2019. Describe los experimentos y casos de test que tuviste que implementar para validar tus resultados. Incluye también los resultados de validación que permiten afirmar que tus resultados son correctos.



# Capítulo 6

## Resultados

En este capítulo se incluyen los resultados de tu trabajo fin de grado.

Si es una herramienta de análisis lo que has realizado, aquí puedes poner ejemplos de haberla utilizado para que se vea su utilidad.



# Capítulo 7

## Conclusiones

### 7.1. Consecución de objetivos

Esta sección es la sección espejo de las dos primeras del capítulo de objetivos, donde se planteaba el objetivo general y se elaboraban los específicos.

Es aquí donde hay que debatir qué se ha conseguido y qué no. Cuando algo no se ha conseguido, se ha de justificar, en términos de qué problemas se han encontrado y qué medidas se han tomado para mitigar esos problemas.

Y si has llegado hasta aquí, siempre es bueno pasarle el corrector ortográfico, que las erratas quedan fatal en la memoria final. Para eso, en Linux tenemos *aspell*, que se ejecuta de la siguiente manera desde la línea de *shell*:

```
aspell --lang=es_ES -c memoria.tex
```

### 7.2. Aplicación de lo aprendido

Aquí viene lo que has aprendido durante el Grado/Máster y que has aplicado en el TF-G/TFM. Una buena idea es poner las asignaturas más relacionadas y comentar en un párrafo los conocimientos y habilidades puestos en práctica.

1. a

2. b

### **7.3. Lecciones aprendidas**

Aquí viene lo que has aprendido en el Trabajo Fin de Grado/Máster.

1. Aquí viene uno.
2. Aquí viene otro.

### **7.4. Trabajos futuros**

Ningún proyecto ni software se termina, así que aquí vienen ideas y funcionalidades que estaría bien tener implementadas en el futuro.

Es un apartado que sirve para dar ideas de cara a futuros TFGs/TFM.

# **Apéndice A**

## **Manual de usuario**

Esto es un apéndice. Si has creado una aplicación, siempre viene bien tener un manual de usuario. Pues ponlo aquí.





## Apéndice B

## Anexo I: Ejemplos de Diagramas UML

Aqui vienen las fotitos de los diagramas UML

```
@inproceedings{robles2005self,  
title={Self-organized development in libre software:  
a model based on the stigmergy concept},  
author={Robles, Gregorio and Merelo, Juan Juli\'an  
and Gonz\'alez-Barahona, Jes\'us M.},  
booktitle={ProSim'05},  
year={2005}  
}
```



# Bibliografía

- [1] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, Inc., 1999.
- [2] G. Robles, J. J. Merelo, and J. M. González-Barahona. Self-organized development in libre software: a model based on the stigmergy concept. In *ProSim'05*, 2005.