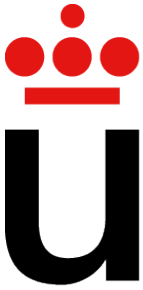

Universidad
Rey Juan Carlos

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

II

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

Índice de figuras

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.

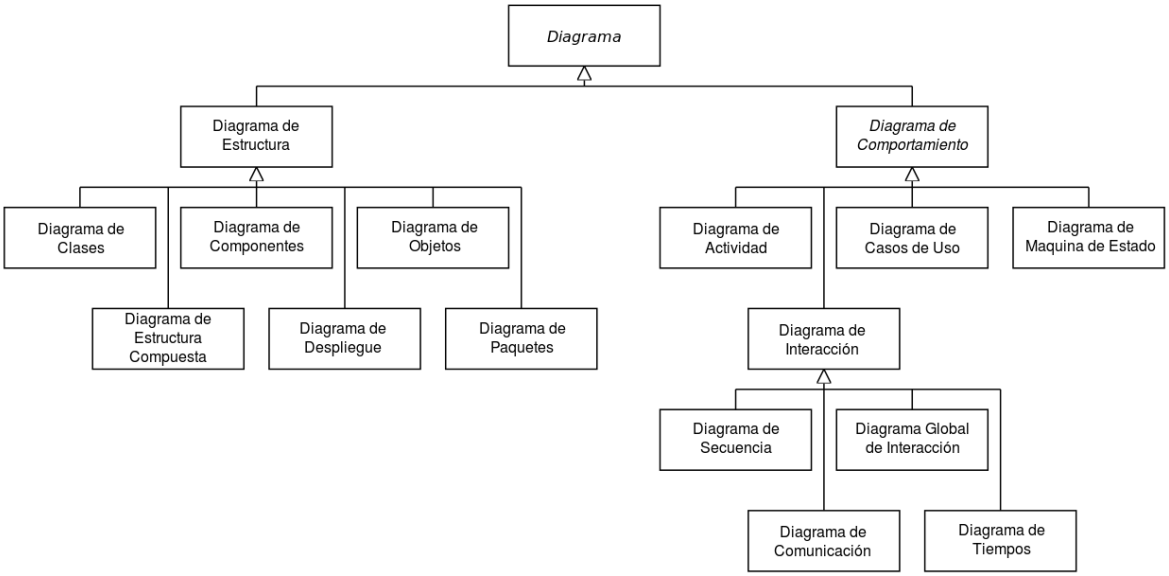

Figura 3.1: Jerarquía de diagramas en UML 2.2 (como diagrama de clases).

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[1], 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 ve 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

[1]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 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[2]. Su máxima es hacer un lenguaje que sea fácilmente legible y con

[2]Python Software Foundation License: `https://docs.python.org/3/license.html`

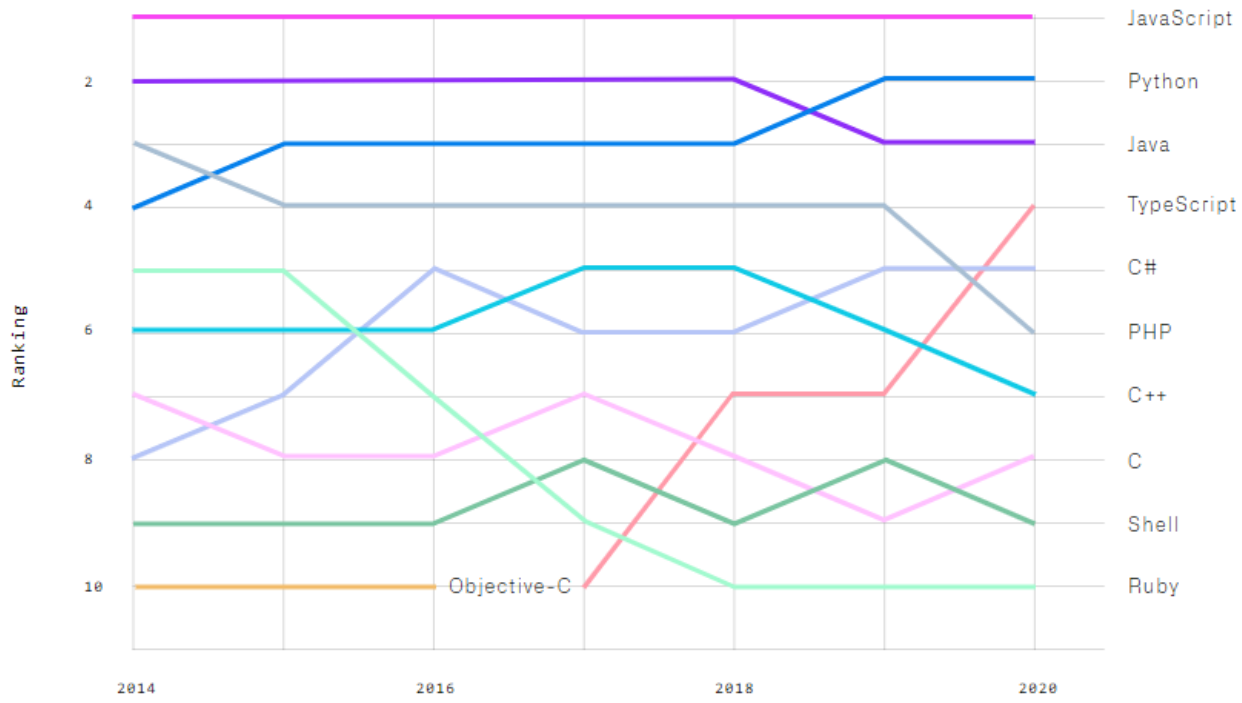

Figura 3.2: Top uso de lenguajes de progrmación alojados en GitHub

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 sú única manera de definir la estructura del código en 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 personan posibles. Django funciona bajo Python, es decir, que para poder trabajar con Django es necesario tener instalado Python.

Un framework, traduciéndolo 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 susttuir 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 siios como CodeComit que perteneze a AWS[3] 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

[3] 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. Boostrap

Boostrap 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.

Boostrat tiene soporte relativo para HTML5 y CSS3, pero es compatible con la mayoría de los navegadores web. Desde su versión 2.0 soporta diseños wen adaptables. Esto significa que eñ 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 interactua 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 Boostrap.

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 afectado 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: Unternamente 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 e 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 una arquitectura de modelo-vista controlador como se ve en la figura 4.1, en esta figura las lineas 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 interactua 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 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.

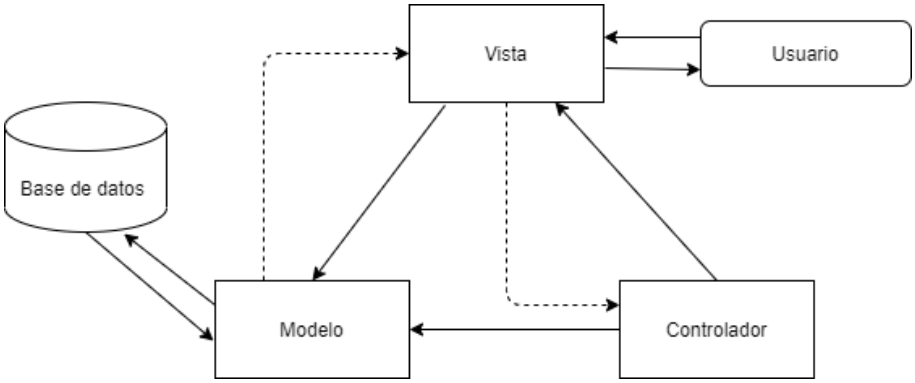

Figura 4.1: Estructura de las relación modelo-vista-controlador.

4.2. Módulo Vistas-Controlador

4.3. Módulo Templates-Vistas

4.4. Módulo Models-Modelo

Recuerda que toda figura que añadas a tu memoria debe ser explicada. Sí, aunque te parezca evidente lo que se ve en la figura 4.1, la figura en sí solamente es un apoyo a tu texto. Así que explica lo que se ve en la figura, haciendo referencia a la misma tal y como ves aquí. Por ejemplo: En la figura 4.1 se puede ver que la estructura del *parser* básico, que consta de seis componentes diferentes: los datos se obtienen de la red, y según el tipo de dato, se pasará a un *parser* específico y bla, bla, bla...

Si utilizas una base de datos, no te olvides de incluir también un diagrama de entidad-relación.

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 TFG/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/TFMs.

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 Articial 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.