



Universidad  
Rey Juan Carlos

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE LA  
TELECOMUNICACIÓN

Curso Académico 2022/2023

Trabajo Fin de Grado

OrganizApp: Sistema de Gestión de Agenda de  
Actividades

Autor/a : Fernando Torrijos Silva  
Tutor/a : Dr. Jesús María González Barahona



# Trabajo Fin de Grado

OrganizApp: Sistema de Gestión de Agenda de Actividades

**Autor/a :** Fernando Torrijos Silva

**Tutor/a :** Dr. Jesús María González Barahona

La defensa del presente Proyecto Fin de Grado/Máster se realizó el día            de  
de 2023, siendo calificada por el siguiente tribunal:

**Presidente:**

**Secretario:**

**Vocal:**

y habiendo obtenido la siguiente calificación:

**Calificación:**

Fuenlabrada, a            de            de 2023



*Dedicado a  
Espe, Rocis, Alfon, Mili, Adriano,  
mis padres y mis hermanas.*



# Agradecimientos

A mi tutor, Jesús María González Barahona, por todo su apoyo y por ser la parte tranquila del equipo; sé que es difícil aguantar mis nervios.

A Espe, por ser la pareja más comprensiva del mundo y por escuchar mis batallitas aunque solo sea por verme una sonrisa.

A mi amigo y hermano, Adriano, uno de los ingenieros más increíbles que conozco, por darme siempre la valentía cuando la pierdo.

A mis padres y hermanas, que me han animado a luchar siempre aún viendo mis bajones.

A Rocis y a sus padres, por tratarme como a un hijo y confiar siempre en mí aunque los resultados dijeran lo contrario. Gracias a eso he llegado hasta aquí.

Y a todos mis profesores. Sé que todos querían sacar la mejor versión de mí y solo puedo tener palabras de agradecimiento.

## *AGRADECIMIENTOS*



# Resumen

La idea principal de este proyecto, es poder tener a mano una herramienta capaz de mostrar los avances diarios en actividades cotidianas de un usuario: tiempo gastado, número de días empleados, gestión de los ingresos, de la actividad física, etc. La base de esta aplicación, está inspirada en libros como *The Bullet Journal Method: Track the Past, Order the Present, Design the Future* de Ryder Carroll y su método para gestionar mejor tu tiempo. Eso sumado a los conocimientos del autor sobre aplicaciones web, ha llevado a la realización de este proyecto llamado **organizApp**.

Esta herramienta es una aplicación web, es decir, es un programa que funciona en Internet sin necesidad de instalarse en tu ordenador; todos los datos son almacenados y administrados en la web. También es multiusuario, capaz de gestionar varios usuarios usando las mismas actividades que se presentan en la aplicación. Por lo tanto, habrá diferentes datos guardados para cada perfil de usuario.

¿Por qué aparece este proyecto? Porque el tiempo es lo más valioso que una persona puede tener. Si se gestiona bien y se tiene control del mismo, pueden hacerse muchas más cosas y ser mucho más productivo. Si no hay organización, no hay una buena gestión del tiempo y, por lo tanto, se desaprovecharía. Con este proyecto lo que se pretende es, tener una mejor gestión de lo que una persona hace, no desde un punto de vista general para cualquier persona, ya que se ha hecho basándose en el gusto del autor (se puede generalizar mucho más, por supuesto), pero sí para tener una idea de cómo de efectivo es tener una aplicación web que gestione tus actividades. Es mucho más cómodo que tenerlo escrito en papel y de esta manera se mantiene más el interés por su uso, ya que a mayor comodidad mejor experiencia de usuario.

Se ha realizado mediante diversas tecnologías. Para el backend o parte interna del proyecto: Django como framework, Python, Plotly, HTTP y MySQL. Y para el frontend o parte de usuario: HTML5, Bootstrap, CSS y JavaScript. Como herramientas de trabajo se han usado Vim y TiliX y para la gestión del código Git. Para finalizar el proyecto, se ha realizado su despliegue mediante ngrok. Todas estas tecnologías y herramientas se explicarán en la memoria: qué son, dónde se han utilizado en el proyecto y por qué.



# Summary

The main idea of this project is to be able to have at hand a tool capable of showing the daily progress in daily activities of a user: time spent, number of days employed, income management, physical activity, etc. The basis of this application is inspired by books like *The Bullet Journal Method: Track the Past, Order the Present, Design the Future* by Ryder Carroll and his method to better manage your time. This, added to the author's knowledge of web applications, has led to the realization of this project called **organizApp**.

This tool is a web application, that is, it is a program that works on the Internet without having to be installed on your computer; all data is stored and managed on the web. It is also multi-user, capable of managing several users using the same activities that are presented in the application. Therefore, there will be different data saved for each user profile.

Why does this project appear? Because time is the most valuable thing a person can have. If you manage it well and have control of it, you can do much more and be much more productive. If there is no organization, there is no good time management and, therefore, it would be wasted. What is intended with this project is to have a better management of what a person does, not from a general point of view for any person, since it has been done based on the author's taste (it can be generalized much more, for course), but to get an idea of how effective it is to have a web application that manages your activities. It is much more comfortable than having it written on paper and in this way the interest in its use is maintained, since the greater the comfort, the better the user experience.

It has been done using various technologies. For the backend or internal part of the project: Django as a framework, Python, Plotly, HTTP and MySQL. And for the frontend or user part: HTML5, Bootstrap, CSS and JavaScript. Vim and Tilix have been used as work tools and Git for code management. To finish the project, it has been deployed using ngrok. All these technologies and tools will be explained in the report: what they are, where they have been used in the project and why.

## *SUMMARY*

# Índice general

<b>1</b>	<b>Introducción</b>	<b>1</b>
1.1	Contexto General . . . . .	1
1.2	Objetivos . . . . .	2
1.2.1	Objetivo general . . . . .	2
1.2.2	Objetivos específicos . . . . .	3
1.3	Estructura de la memoria . . . . .	3
<b>2</b>	<b>Tecnologías empleadas</b>	<b>5</b>
2.1	Python . . . . .	6
2.2	Django . . . . .	7
2.2.1	El patrón de diseño MVC . . . . .	8
2.3	HTML5 . . . . .	8
2.4	Plotly . . . . .	9
2.5	HTTP . . . . .	9
2.6	Bootstrap . . . . .	11
2.7	CSS . . . . .	11
2.8	JavaScript . . . . .	11

2.9	MySQL . . . . .	12
2.10	Git . . . . .	13
2.11	Herramientas de trabajo: Vim & Tilix . . . . .	13
<b>3</b>	<b>Etapas del proyecto</b>	<b>15</b>
3.1	Etapa 1: Configuración y puesta en marcha . . . . .	15
3.2	Etapa 2: Funcionalidades a desarrollar . . . . .	16
3.3	Etapa 3: Diseño e Imagen . . . . .	16
3.4	Etapa 4: Visualización gráfica . . . . .	17
3.5	Etapa 5: Despliegue y prueba de usuarios reales . . . . .	20
<b>4</b>	<b>Funcionalidades y uso</b>	<b>21</b>
4.1	Visión general . . . . .	21
4.2	Actividades . . . . .	21
4.2.1	Sueño . . . . .	21
4.2.2	Gastos   Ingresos . . . . .	26
4.2.3	Ocio . . . . .	26
4.2.4	Deporte . . . . .	28
4.2.5	Estudio . . . . .	28
4.3	Características personales, menú y gráficas y estadísticas . . . . .	28
4.4	Registro . . . . .	33
4.5	Perfil . . . . .	33
4.6	Salir . . . . .	35
4.7	Administrador . . . . .	35

## ÍNDICE GENERAL

<b>5</b>	<b>Arquitectura interna y parte técnica</b>	<b>37</b>
5.1	Estructura del código: MVC y plantillas . . . . .	37
5.2	Actividades . . . . .	38
5.2.1	Expenses . . . . .	39
5.3	Registro . . . . .	43
5.4	Núcleo o <i>core</i> . . . . .	44
5.5	Seguimiento . . . . .	44
5.6	Errores . . . . .	45
5.7	Interacción con la Base de Datos . . . . .	45
5.8	Validadores . . . . .	47
5.9	Testing . . . . .	48
5.10	Logging . . . . .	51
5.11	Despliegue con <i>ngrok</i> . . . . .	52
<b>6</b>	<b>Conclusiones y trabajos futuros</b>	<b>55</b>
6.1	Consecución de objetivos . . . . .	55
6.2	Aplicación de lo aprendido . . . . .	55
6.3	Conocimientos adquiridos . . . . .	56
6.4	Prueba de usuarios externos . . . . .	57
6.5	Trabajos futuros . . . . .	59
6.6	Planificación . . . . .	59
<b>A</b>	<b>Manual de configuración</b>	<b>61</b>
A.1	Instalar MySQL . . . . .	61

A.2 Configuración del logging . . . . .	61
<b>Referencias</b>	<b>65</b>



# Índice de figuras

2.1	Funcionamiento de HTML. . . . .	9
2.2	Varios tipos de gráficos con la librería Plotly. . . . .	10
2.3	Logo oficial de JavaScript. . . . .	12
2.4	Ejemplo de tabla en MySQL. . . . .	13
2.5	Ejemplo de la aplicación Tilinx con varias ventanas haciendo diferentes tareas. . . . .	14
3.1	Comienzo del proyecto. . . . .	16
3.2	Versión beta de aplicación. Inicios con ajuste de ventana. . . . .	18
3.3	Imagen 2D de Gastos e Ingresos con Chart.js. . . . .	19
3.4	Imagen 2D y 3D visualizando datos de Ocio y Estudio. . . . .	19
4.1	Diagrama de la aplicación. . . . .	22
4.2	Registrar horas de sueño. . . . .	23
4.3	Índice de sueños. . . . .	24
4.4	Editar sueño. . . . .	25
4.5	Detalles, botón de edición y botón de eliminación de Sueño. . . . .	25
4.6	Índice de gastos e ingresos. . . . .	27
4.7	Detalles de Gasto o Ingreso. . . . .	27

4.8	Índice de deportes. . . . .	28
4.9	Detalles de deporte. . . . .	29
4.10	Características personales del usuario. . . . .	30
4.11	Menú de la aplicación. . . . .	30
4.12	Gráficas y Estadísticas. . . . .	31
4.13	Visualización 2D de Gastos e Ingresos. . . . .	32
4.14	Gráficas de Estudio y Ocio. . . . .	34
4.15	Visualización 3D de Deporte y Sueño. . . . .	35
4.16	Página de inicio de sesión. . . . .	36
4.17	Formulario de registro. . . . .	36
5.1	Esquema general interno de la aplicación. . . . .	38
5.2	Esquema general interno de Django: MVC. . . . .	39
5.3	Parte interna de una aplicación. . . . .	40
5.4	Trozo de ejemplo de <i>tfg.log</i> . . . . .	44
5.5	Ejemplo de correo para controlar errores. . . . .	46
5.6	Gasto guardado en BBDD. . . . .	47
5.7	Gasto editado en BBDD. . . . .	47
5.8	Gasto eliminado de BBDD. . . . .	47
5.9	Formulario de registro avisando de error en contraseña. . . . .	49
5.10	Validador para un registro de Deporte. . . . .	50
5.11	Resolución de los tests creados. . . . .	51
5.12	<i>tfg.log</i> muestra que se ha eliminado un gasto. . . . .	52

## ÍNDICE DE FIGURAS

5.13	Funcionamiento de ngrok. . . . .	53
6.1	Usuarios que han usado la aplicación. . . . .	57
6.2	Planificación temporal del TFG. . . . .	60

## ÍNDICE DE FIGURAS

# Índice de fragmentos de código

5.1	Cambiar SQLite3 por MySQL. . . . .	46
5.2	Validador para la app <i>sports</i> . . . . .	48
5.3	Parte de plantilla <i>sports_create.html</i> : ventana emergente que muestra alerta de fallo. . . . .	49
5.4	Test para comprobar si <i>ExpensesForm</i> es válido. . . . .	51
A.5	Creación de Base de Datos. . . . .	62
A.6	Configuración del logging. . . . .	63

## ÍNDICE DE FRAGMENTOS DE CÓDIGO

# Capítulo 1

## Introducción

Con el tiempo, cada vez hay más aparatos electrónicos que intentan facilitar información acerca de las personas, ya sea su actividad física, su ritmo cardíaco, horas dormidas, etc. Son temas que interesan y que facilitan estos aparatos, aunque los intereses que haya por detrás sean otros: dónde están esas personas, qué edad tienen, qué hacen, con quién están ..., datos que son muy valiosos para las empresas, ya que tienen un estudio claro de sus usuarios y por ende pueden saber de ellos. Estos datos pueden venderse o utilizarse para vender, pero esto es otro tema aparte que no tiene nada que ver con el proyecto. Si se quiere saber más de cómo las empresas obtienen datos personales y qué hacen con ellos, es interesante leer el libro *Datanomics* de Paloma Llana [9].

Este proyecto trata de tener algo parecido a lo dicho anteriormente sin finalidades comerciales de por medio. Su objetivo es tener, de la manera más clara y simple posible, todas las actividades de interés de un usuario bajo control sin tener que preocuparse nada más que de tener la *agenda* al día; de la visualización, control y gestión de datos se encarga la propia aplicación. Las actividades que se presentan son: Sueño, Gastos | Ingresos, Ocio, Deporte y Estudio. Todas ellas se encargan, para cada perfil de usuario, de tener un registro diario de cada actividad.

### 1.1 Contexto General

La organización proporciona productividad y esto es algo clave, ya sea para tener más ocio, para rendir más en el trabajo, para tener más tiempo con la familia o para tener más tiempo para ti. Dejar el tiempo al libre albedrío puede suponer un desastre en lo que a aprovechamiento del tiempo se refiere. Si una persona es responsable, debe saber qué es lo que tiene que hacer con su tiempo y cumplir sus objetivos y la mejor manera es tener su día organizado.

No es solo algo personal, sino que también las empresas tienen esto muy en cuenta y constantemente están innovando y estudiando maneras donde el operario rinda lo máximo posible, para que de esta manera se obtengan mayores ganancias. A día de hoy, que está todo digitalizado prácticamente, se tiene software que aplica este concepto [6], lo cual ayuda mucho a aprovechar mejor el tiempo y a no hacer demasiadas reuniones hablando de este tema. Lo ideal es que el propio software sea lo suficientemente versátil para que todo esté bajo los mínimos establecidos y que la gestión esté prácticamente automatizada y autogestionada.

Un ejemplo de por qué la organización es tan importante, se puede ver en el estudio que se hizo a partir de los datos para una muestra representativa del tejido productivo privado en Cataluña (1.283 empresas) [11], donde llega a la conclusión de que la organización por procesos y, por lo tanto, control de los mismos, ejercen efectos positivos en la eficiencia laboral y que es un factor determinante y positivo en el nivel de productividad aparente del trabajo en la empresa catalana.

El objetivo de este proyecto, es que cualquier persona pueda tener presente todos los avances que realiza diariamente y de esta manera no perder su tiempo y no liarse, teniendo siempre constancia de todo detalle y pudiendo visualizar cualquier dato que le interese, ya sea por mejorar o por cambiar algún concepto de su día a día.

## 1.2 Objetivos

### 1.2.1 Objetivo general

Crear una herramienta web, capaz de organizar diariamente las actividades de una persona, mostrando sus resultados en cada una de ellas. Cada una tendrá las características necesarias para poder tener un seguimiento correcto: tiempo, nivel de satisfacción, qué hago en la actividad, cuánto me gasto, etc. El usuario podrá tener registrado todo lo que hace y acceso a todos sus movimientos dentro de la aplicación.

Este proyecto se ha enfocado en actividades que interesan al autor del proyecto, desde su punto de vista. Se ha inspirado en aplicaciones como Google Fit, Samsung Health, GPS de Strava, Huawei Health, ..., aplicaciones que muestran un seguimiento de actividades físicas y salud de una persona: cuánto ha dedicado a un deporte, sueño diario, pasos realizados, etc. Pero además, se ha querido añadir más funcionalidades fuera del deporte y la salud, como por ejemplo gastos e ingresos que se realizan, tiempo de estudio, ocio y más. En resumen, el objetivo es tener un registro de lo que más emplea su tiempo un usuario y tener un cómodo seguimiento.



## 1.2.2 Objetivos específicos

Esta sección desglosa los diferentes objetivos específicos del proyecto:

- **Visualización de datos.** Mostrar los datos de manera gráfica, tanto en 2D como en 3D.
- **Frontend: HTML5, Plotly, Bootstrap, CSS y JavaScript.** Desarrollar la parte visual de usuario con estas tecnologías.
- **Backend: Python, Django y MySQL.** Desarrollar la parte lógica de la aplicación con estas tecnologías.
- **Despliegue.** Proporcionar una plataforma que permita el uso de la aplicación por usuarios externos.
- **Multiusuario.** Capacitar a la aplicación para llevar el registro de diferentes usuarios, con sus diferentes movimientos y avances entre ellos.
- **Diseño atractivo.** Implementar una aplicación que sea, además de funcional, atractiva al que la usa.
- **Diferentes funcionalidades.** Desarrollar actividades que fuesen de interés.
- **Usabilidad en dispositivos móviles.** Tener la opción de usar la aplicación, además de en una computadora, en un dispositivo móvil.
- **Manejo de errores.** Poder corregir y editar funcionalidades debido al mal uso de la aplicación, no solo por el desarrollador sino por el mismo usuario.
- **Intuitivo y fácil.** Usar la aplicación sin necesidad de tener un manual a mano.
- **Mínimo riesgo de uso.** No tener ningún tipo de temor de estropear cualquier parte de la aplicación; total libertad de uso sin preocupaciones.

## 1.3 Estructura de la memoria

A continuación, se muestra la estructura de la memoria para tener un vistazo rápido de la misma. Está todo detallado a continuación:

- **2. Tecnologías empleadas.** Muestra las tecnologías y herramientas utilizadas: el porqué de su uso en el proyecto y una breve descripción de las mismas.

- **3. Etapas del proyecto.** Etapas realizadas en el desarrollo del proyecto, desde su inicio hasta su finalización.
- **4. Funcionalidades y uso.** En este capítulo se explica detalladamente cómo se usa la aplicación web y qué características tiene.
- **5. Arquitectura interna y parte técnica.** Detalles técnicos del proyecto, mostrando su arquitectura interna.
- **6. Conclusiones y trabajos futuros.** Este capítulo es el final de la memoria, donde se muestran conclusiones de todo el trabajo realizado y algunas ideas para proyectos futuros.

# Capítulo 2

## Tecnologías empleadas

A continuación se mostrará una breve exposición de las tecnologías y herramientas utilizadas en el proyecto.

Una breve explicación del porqué se han usado, es:

Para el **Backend** o parte lógica de la aplicación:

- **Python**. Por su sencillez y facilidad de gestión de datos.
- **Django**. Como framework (plantilla de software), por su seguridad y amplias herramientas para la creación de software.
- **MySQL**. Como administrador de la base de datos, porque es sencillo y permite relacionar datos con puntos unívocos entre sí, permitiendo facilidad de gestión de datos.

Para el **Frontend** o parte visual de la aplicación:

- **HTML5**. Como código que estructura la aplicación web, al ser el estándar de programación web actual y, por lo tanto, el más actualizado.
- **Plotly**. Para la visualización de datos, ya que tiene un amplio abanico de gráficos de todo tipo.
- **Bootstrap**. Como biblioteca que ofrece ventajas a la hora de diseñar la aplicación, ya que ahorra tiempo para diseñar elementos básicos como: la barra de navegación, menú, estructura sin fallos de visualización, etc.
- **CSS**. Para los detalles de color y estilo, por ser compatible con anteriores tecnologías y sencillo de usar.

- **JavaScript.** Para añadir detalles avanzados dentro de la aplicación. Es sencillo de aprender y te permite hacer gran variedad de funciones, en el caso de la aplicación: cambiar texto en botones, añadir y esconder botones, calendario interactivo, por ejemplo. Aunque tiene infinidad de funciones más, que se pueden implementar en un futuro.

Fuera del backend y del frontend, como herramientas de trabajo se ha usado **Vim y Tilix**, al ser herramientas familiares al autor de este proyecto y por su rapidez de acción, **Git** para controlar el código y tener siempre un registro de todos los movimientos que se hagan en el software y **HTTP** como protocolo de intercambio de documentos hipermedia HTML entre cliente (usuario/navegador) y servidor que te proporciona Django: *python manage.py runserver*; se hablará de ello más adelante.

## 2.1 Python

Python<sup>1</sup> es un lenguaje de alto nivel (alejado del lenguaje de máquina), destinado principalmente para desarrollo web y aplicaciones. Es la lógica que se encarga de que todo funcione en el proyecto. Destaca por lo siguiente [2]:

- **RAD.** Atractivo en el campo del Desarrollo Rápido de Aplicaciones (RAD) porque ofrece tipado dinámico. ¿Qué es RAD? Es una técnica de desarrollo de software, donde la rapidez de entrega del producto a los clientes es lo principal, por lo que se está constantemente dando avances por parte del proveedor y respuesta por parte del cliente. De esta manera, el tipado dinámico (las variables pueden tomar valores de distintos tipos de datos) ofrece versatilidad y, por lo tanto, rapidez a la hora de crear software.
- **Sencillo.** Fácil de aprender, ya que tiene una sintaxis única que se centra en la legibilidad.
- **Modular.** Uso de módulos y paquetes. Una vez se ha desarrollado un módulo o paquete, se puede escalar para su uso en otros proyectos.
- **Multiplataforma.** Se puede ejecutar en Linux, Windows, Macintosh, etc.
- **Orientado a objetos.** Es decir, centrado en particularizar cada parte del código como si fuera un objeto. Cada objeto es una pieza de mayor o menor jerarquía dentro de un programa y entre todas las piezas, se genera un sistema que funciona. Nos olvidamos de pensar en funciones, clases ..., para poder tener mayor facilidad de comprensión.

---

<sup>1</sup><https://www.python.org/>

- **Open Source.** Código accesible al público y gratuito.

Un libro muy recomendado para aprender Python es *Learning Python: Crash Course Tutorial* del autor del lenguaje Guido van Rossum [12]. Este libro sirve para profundizar más en cada parte del lenguaje desde el punto de vista del autor, aunque hay muchísima información para aprender en Internet o en cursos en línea.

Debido a la sencillez de Python y a su facilidad para tratamiento de datos, muchas tecnologías utilizan este lenguaje, aspecto que ha hecho que sea de gran interés dentro del mundo laboral [13]:

- **Data analytics y Big Data.** Facilidad a la hora de analizar y gestionar gran cantidad de datos en tiempo real.
- **Data mining.** Gran limpieza y organización de datos para el uso de algoritmos de aprendizaje automático.
- **Data science.** Sencillez y potencia para trabajar con gran número de datos.
- **Inteligencia artificial.** Gran parte del avance en la inteligencia artificial (IA), es gracias a Python. Su robustez y sencillez, permite plasmar ideas complejas en pocas líneas.
- **Blockchain.** Base sobre la que se sustentan las criptomonedas, Python permite a los desarrolladores crear una *cadena de bloques*<sup>2</sup> sencilla.
- **Machine learning.** En el tratamiento de datos, Python es muy eficaz, permitiendo facilidades dentro de este campo.
- **Desarrollo web.** Desarrollar webs complejas en menos líneas de código.
- **Juegos y gráficos 3D.** Gran capacidad para manejar gráficos 3D.

## 2.2 Django

Django<sup>3</sup> es un framework de aplicaciones web gratuito y de código abierto (open source) escrito en Python. Un framework web es un conjunto de componentes que te ayudan a desarrollar sitios web más fácil y rápidamente. Desde hace bastante, los desarrolladores se dieron cuenta de que siempre se enfrentaban a los mismos problemas cuando construían sitios web, y por eso se unieron y crearon frameworks con componentes listos para usarse [4] [8].

---

<sup>2</sup>[https://es.wikipedia.org/wiki/Cadena\\_de\\_bloques](https://es.wikipedia.org/wiki/Cadena_de_bloques)

<sup>3</sup><https://www.djangoproject.com/>

### 2.2.1 El patrón de diseño MVC

El funcionamiento de Django se divide en 3 partes: Modelo - Vista - Controlador. Y para el diseño de la página web están las plantillas.

- **Modelo.** Contiene la descripción de la tabla de la base de datos (*models.py*). Se puede manipular esta tabla mediante declaraciones SQL, pero es mucho más sencillo si lo haces usando código Python.
- **Vista.** Contiene la lógica de la página (*views.py*), es decir, lo que maneja las acciones que se hagan en la aplicación.
- **Controlador.** Especifica qué vista es llamada según el patrón URL (*urls.py*).
- **Plantillas.** Describe el diseño de la página (*plantilla.html*). Se usan diferentes tecnologías que se hablarán más adelante.

Con Django, podemos manejar diferentes sistemas de gestión de bases de datos: MySQL, PostgreSQL, etc.

## 2.3 HTML5

HTML<sup>4</sup> es el código que estructura una página web, colocando sus diferentes elementos donde interesa. Su nombre son las sigas de *HyperText Markup Language*, que significa *Lenguaje de Marcado de Hipertexto*. Es el estándar con el que están programadas todas las webs. Su última versión, la que se ha utilizado, es HTML5, cuyas nuevas características ayudan a una mayor adaptabilidad en lo que a creación de contenido se refiere. Es la versión más actual y aunque muchas características de su anterior versión, HTML4, permanecen, se ha conseguido que sea más fácil escribir código con su nueva sintaxis. Como diferencia, por ejemplo, permite añadir elementos multimedia a tu aplicación web, como audio y vídeo, permitiendo desprenderse de CSS y JavaScript para estas finalidades. Al ser la versión más moderna, los navegadores modernos solo permiten esta versión, dejando atrás HTML4 y no permitiendo funcionalidades de esta.

Su funcionamiento es sencillo, tal y como muestra la Figura 2.1. Desde el navegador se realiza una petición a un servidor. Después el servidor recupera de su disco duro esa página, la devuelve al navegador y la página se muestra [7].

---

<sup>4</sup><https://www.w3schools.com/html/>

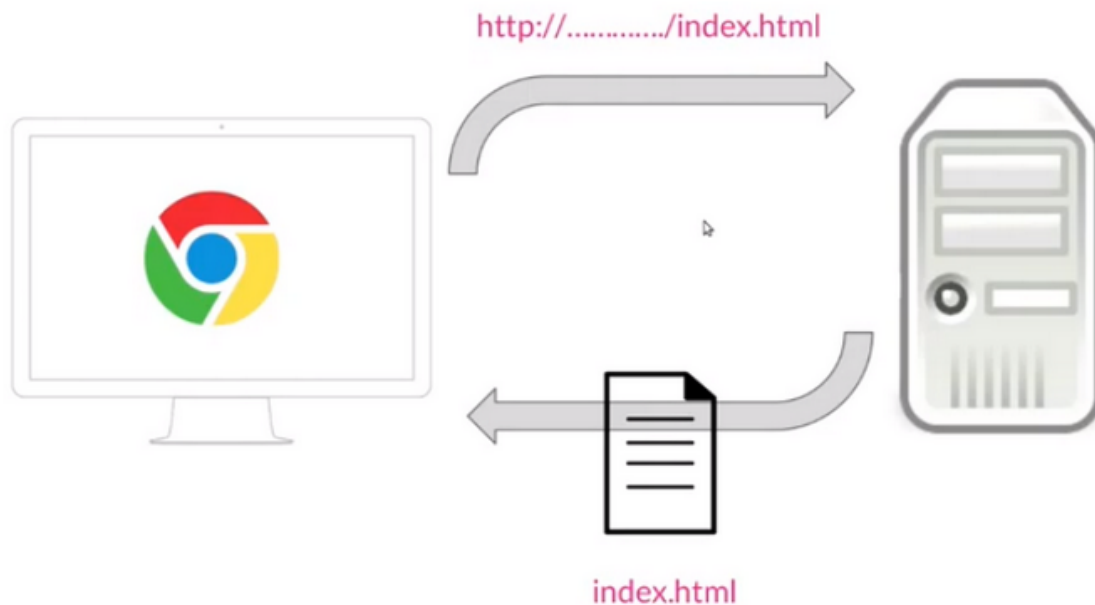


Figura 2.1: Funcionamiento de HTML.

## 2.4 Plotly

Plotly<sup>5</sup> es una amplia biblioteca de gráficos en **Python**, **Java**, **Matlab**, **R** o **JavaScript**, usada para visualización y análisis de datos online. Posee cualquier tipo de gráficas: columnas, líneas, histogramas, 3D, etc.

¿Por qué se ha usado en el proyecto? Porque es una herramienta muy útil tener en un simple click un resumen en imagen de todos los datos que uno quiere ver. En este caso se han usado dos tipo de visualizaciones: 2D y 3D. Se hace mucho más cómodo ver tus datos en formato imagen que en formato texto. Para poder tener una idea de por qué es mejor, se puede leer esta entrada: [Text vs. Images: Which Content Format is Effective?](#), que explica un poco cómo nuestro cerebro asimila de esta forma mejor las cosas.

Hay gran variedad de tipos de gráficos con Plotly. Se pueden ver varios ejemplos en la Figura 2.2.

## 2.5 HTTP

HTTP [3] es un protocolo de la capa de aplicación para la transmisión de documentos hipermedia, como **HTML**. Fue diseñado para la comunicación entre los navegadores y servidores web. Tiene las siguientes características:

---

<sup>5</sup><https://plotly.com/>

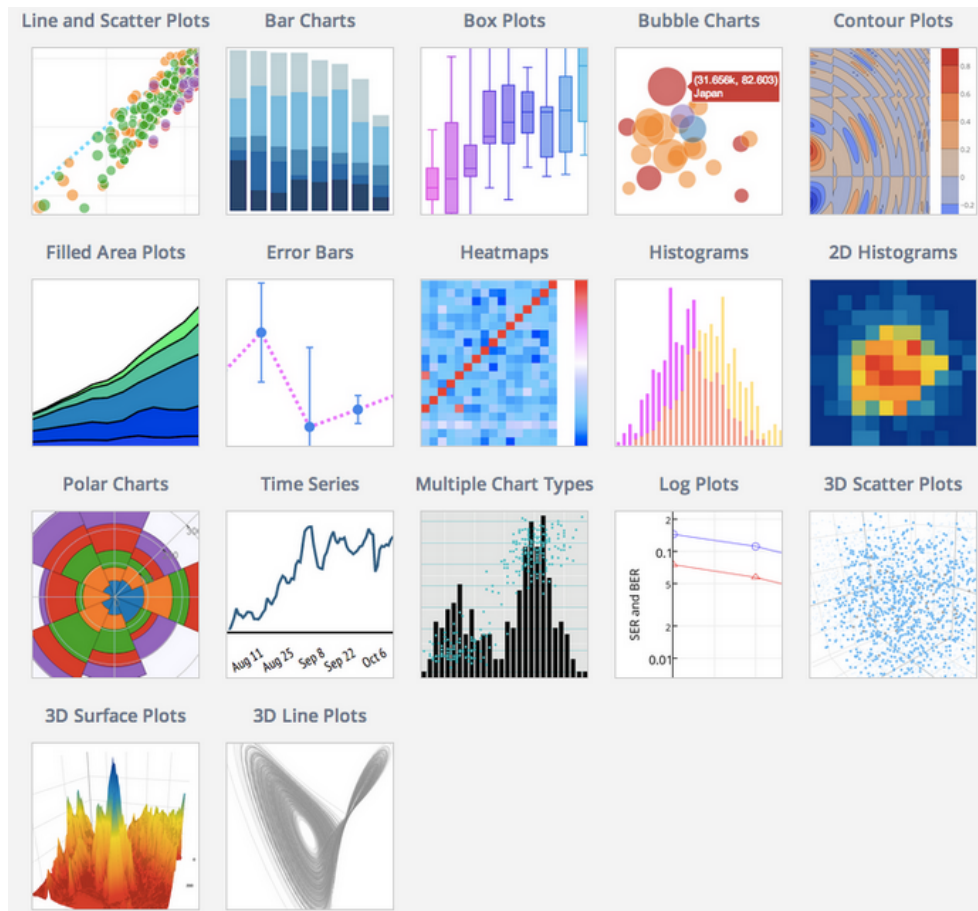


Figura 2.2: Varios tipos de gráficos con la librería Plotly.

- **Modelo cliente-servidor.** Un cliente establece una conexión, realizando una petición a un servidor y espera una respuesta del mismo.
- **Protocolo sin estado.** El servidor no guarda ningún dato (estado) entre dos peticiones, es decir, trata cada petición como una petición independiente sin tener en cuenta ninguna anterior: solicitud-respuesta.
- **Conexión segura.** Aunque en la mayoría de casos se basa en una conexión TCP/IP, puede ser usado sobre cualquier capa de transporte segura, es decir, sobre cualquier protocolo que no pierda mensajes silenciosamente (UDP).

Las *vistas* (views.py) en el proyecto, son las responsables de ejecutar las peticiones HTTP.



## 2.6 Bootstrap

Bootstrap<sup>6</sup> es una biblioteca (conjunto de funciones ya programadas para usarse en un programa) usada para diseño de sitios y aplicaciones web. Está formada por plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación, ..., cualquier elemento de diseño basado en **HTML** y **CSS**, así como extensiones de **JavaScript** adicionales.

Es open source, multiplataforma y desde la versión 2.0, el diseño gráfico de la página se ajusta dinámicamente, tomando en cuenta las características del dispositivo usado.

Gracias a Bootstrap, aparte de aprovechar todas sus ventajas en un portátil, se puede visualizar el proyecto en un dispositivo móvil (para una visualización idónea, es mejor poner el móvil en horizontal).

## 2.7 CSS

CSS<sup>7</sup> u **Hojas de Estilo en Cascada** (del inglés *Cascading Style Sheets*), es el lenguaje de estilos que se usa para describir la presentación de documentos **HTML** o **XML**. Está diseñado para marcar la separación del contenido del documento y la forma de presentación de este, como las capas, colores y fuentes. Esto proporciona flexibilidad, ya que un archivo .css puede ser usado por ejemplo en varios documentos HTML.

Su funcionamiento es simple. Cuando desde un navegador se solicita una página, esta petición va a un servidor web, que devuelve la página que se ha solicitado. Para aplicar los estilos en las páginas HTML, se utiliza un fichero aparte, que es una hoja de estilos CSS. Cuando estos documentos llegan al navegador, se lee el documento HTML, se le aplican los estilos CSS y finalmente se muestra la página al usuario.

## 2.8 JavaScript

JavaScript es un lenguaje de programación que se usa para añadir características interactivas a un sitio web. Sus principales características son [10]:

- **Lenguaje del lado del cliente.** Se ejecuta en la máquina del propio cliente.
- **Lenguaje orientado a objetos.** Utiliza clases y objetos como estructuras.

---

<sup>6</sup><https://getbootstrap.com/>

<sup>7</sup><https://developer.mozilla.org/es/docs/Web/CSS>

- **De tipado débil o no tipado.** No es necesario especificar el tipo de dato.
- **De alto nivel.** Su sintaxis se encuentra alejada del nivel máquina.
- **Lenguaje interpretado.** No necesita una compilación previa para su ejecución y permite convertir las líneas de código en el lenguaje de la máquina.
- **Muy utilizado por desarrolladores.**

Dentro de la aplicación, se usa dentro de los cambios que se van realizando mientras se va usando:

- **Cambiar texto de los botones.** Como por ejemplo en *Gastos | Ingresos* cuando se ingresa o se gasta dinero.
- **Poner o no una carpeta.** Como en el caso de los índices de las actividades: si existe que ponga la carpeta, sino no.
- **Mostrar los calendarios a la hora de elegir una fecha.** Cuyo calendario está personalizado en el fichero *custom-datepicker.js*.
- Etc.



Figura 2.3: Logo oficial de JavaScript.

## 2.9 MySQL

MySQL<sup>8</sup> es un sistema de administración de bases de datos para bases de datos relacionales, es decir, cada registro de una tabla tiene un identificador único, que puede ser utilizado en otra tabla como atributo de un registro. Utiliza tablas para almacenar y organizar la información. Destaca que es open source y multiplataforma. Un ejemplo de tabla se muestra en la Figura 2.4.

---

<sup>8</sup><https://www.mysql.com/>

```
mysql> SELECT * FROM sports_sports;
```

id	name	body_zone	duration	date_sport	loss_weight	type_spo	competition	lesion	user_id
18	0	0	0	2022-04-08	0	0	0	0	1
19	2	2	1	2022-04-08	1	0	1	0	1
20	4	0	2	2022-03-18	1	0	0	0	1
21	5	2	2	2022-03-13	1	0	0	0	1
22	1	5	1	2022-04-09	0	0	0	0	1
23	6	5	2	2022-03-16	0	0	0	0	1
24	3	5	1	2022-03-12	1	1	0	0	1
25	0	0	1	2022-03-19	0	0	0	0	1
26	1	5	2	2022-03-21	1	0	0	0	1
27	0	0	3	2022-03-01	1	0	0	0	1
28	4	4	1	2022-03-04	1	0	0	0	1
29	4	0	1	2022-03-10	1	0	0	0	1
30	3	5	1	2022-04-12	0	1	0	0	1
32	3	1	2	2022-03-23	1	0	0	0	1
34	1	1	0	2022-04-14	0	1	1	0	1

Figura 2.4: Ejemplo de tabla en MySQL.

## 2.10 Git

Git<sup>9</sup> es un software de control de versiones creado por Linus Torvalds<sup>10</sup>, con el fin de tener bajo control aplicaciones con un gran número de archivos de código fuente. Registra los cambios en los archivos y gracias a él, es posible coordinar el trabajo entre varias personas que comparten el mismo repositorio de código en diferentes lugares.

Las características más relevantes son:

- **Desarrollo no lineal.** Esto permite rapidez en la gestión y mezclado de diferentes versiones.
- **Gestión distribuida.** Git da a cada programador una copia local del historial del desarrollo entero.
- **Información publicada por HTTP, FTP, rsync o SSH.**
- **Gestión eficiente de proyectos grandes.**
- **Notificación de cualquier cambio.**

## 2.11 Herramientas de trabajo: Vim & Tilix

Vim<sup>11</sup>, versión mejorada de Vi, es un editor de texto que permite mover, copiar, eliminar o insertar caracteres con mucha versatilidad.

<sup>9</sup><https://git-scm.com/>

<sup>10</sup>[https://es.wikipedia.org/wiki/Linus\\_Torvalds](https://es.wikipedia.org/wiki/Linus_Torvalds)

<sup>11</sup><https://www.vim.org/>

Está presente en todos los sistemas UNIX y es comúnmente utilizado por programadores para escribir código fuente de software. Con los plugins necesarios, puede fácilmente sustituir cualquier IDE (software para el diseño de aplicaciones, que combina herramientas comunes para desarrolladores en una sola interfaz) y aprovechar su principal característica que es la gran rapidez de acceso al código, ya que no consume casi nada de RAM.

Por otro lado está Tilix. Es una aplicación de terminal gráfica que destaca por tener opciones como la de partir, redimensionar y cerrar terminales en varias sesiones a nivel de una ventana. Es una herramienta muy flexible, que permite a las personas que hacen gran uso de la consola GNU/Linux ir con gran fluidez. Un ejemplo de esto último se muestra en la Figura 2.5.

```

1: Terminal
user: admin on radiometrica via 1618/8618 | 218MiB/7618
[ 12:30:35 ] >

2: ~/Escribiste/VR/C/porfati/TPC/organizaApp/organizaApp/dreams/views.py
views.py
1 #
2 # dreams/views.py
3 #
4
5 from django.contrib.auth.models import User
6 from django.shortcuts import render, redirect, get_object_or_404
7 from .forms import DreamsForm
8 from .models import Dreams
9 from registration.models import Profile
10 from django.contrib.auth.decorators import login_required, permission_required
11 import logging
12
13 log = logging.getLogger(__name__)
14
15 # Create your views here.
16
17 @login_required
18 def dreams_create(request):
19     template = 'dreams/dreams_create.html'
20     form = DreamsForm(request.POST or None)
21
22     if request.method == 'POST':
23         if form.is_valid():
24             # gets you a model object, then you can add your extra data and
25             # save it to the database.
26             form.save()
27             return redirect('dreams:detail', kwargs={'pk': form.instance.pk})
28     else:
29         form = DreamsForm()
30     return render(request, template, {'form': form})
31
32 if __name__ == '__main__':
33     python manage.py runserver --noinput
34
35 "views.py" 183L, 3259C

4: Terminal
1 ||| 2.7% 5 ||| 4.6%
2 ||||| 9.9% 6 ||||| 4.6%
3 ||| 2.7% 7 ||||| 6.6%
4 ||| 4.6% 8 ||| 4.6%
Mem ||||| 5.786/15.46 Tasks: 191, 1095 thr: 1 running
Sup ||||| 0%/2.006 Load average: 0.56 0.75 0.77
Uptime: 05:00:42

PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
25871 fer 20 0 152M 4128 3756 S 0 0 0 0:00.00 xdg-dbus-proxy --args=33
26080 fer 20 0 46452 31268 11608 S 0 0 0.2 0:00.20 vim views.py
25722 fer 20 0 974M 77100 58004 S 0 0 0.5 0:00.00 tilix
25723 fer 20 0 974M 77100 58004 S 0 0 0.5 0:00.00 tilix
25724 fer 20 0 974M 77100 58004 S 0 0 0.5 0:00.00 tilix
25725 fer 20 0 974M 77100 58004 S 0 0 0.5 0:00.00 tilix
25726 fer 20 0 974M 77100 58004 S 0 0 0.5 0:00.00 tilix
25727 fer 20 0 974M 77100 58004 S 0 0 0.5 0:00.00 tilix
25728 fer 20 0 974M 77100 58004 S 0 0 0.5 0:00.00 tilix
25729 fer 20 0 974M 77100 58004 S 0 0 0.5 0:00.00 tilix
25730 fer 20 0 974M 77100 58004 S 0 0 0.5 0:00.00 tilix
25731 fer 20 0 974M 77100 58004 S 0 0 0.5 0:00.00 tilix
26012 fer 20 0 974M 77100 58004 S 0 0 0.5 0:00.00 tilix
25719 fer 20 0 974M 77100 58004 S 0 0 0.5 0:01.04 tilix
959 root 20 0 12180 6844 6004 S 0 0 0 0:00.00 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startu

3: Terminal
user: ftorrijos on ncondes-pc via 744MiB/11618 | 587MiB/2618
[ 12:30:51 ] >
  
```

Figura 2.5: Ejemplo de la aplicación Tilix con varias ventanas haciendo diferentes tareas.

# Capítulo 3

## Etapas del proyecto

En este capítulo se hablará, resumidamente, de cómo ha ido creciendo la aplicación, desde sus inicios hasta el final: pruebas, errores y aprendizaje. Aproximadamente, se ha tardado nueve meses en realizar el proyecto. De media, se han empleado tres, cuatro días a la semana por las tardes, complementándolo con mi trabajo y la finalización de la carrera.

### 3.1 Etapa 1: Configuración y puesta en marcha

Para empezar el proyecto, a principios de octubre de 2022 (ver Figura 3.1), lo primero que se hizo fue crear las bases del mismo, que se detallan a continuación:

- **Configuración de Vim.** IDE a usar. Se implantaron todos los *plugins* necesarios para poder realizar fácilmente el trabajo. Inicialmente, Vim viene con pocos recursos, pero *googleando* un poco, se puede generar una herramienta rápida y eficiente para el desarrollo de software.
- **Creación de una cuenta Gitlab.** Para poder subir y gestionar el código que se implementa.
- **Configuración de Git.** Instalación y configuración de Git en el ordenador.
- **Preparación del entorno: Django.** Instalar el framework (plantilla para el software a desarrollar) Django y configurarlo para el proyecto.
- **Creación de la Base de Datos.** Primero se instaló **MySQL** (se puede ver cómo en A.1), que será nuestro gestor de la Base de Datos. Seguidamente, se creó la aplicación y se conectó a MySQL.

Con todas estas partes configuradas, ya se pudo empezar la creación del proyecto en el ordenador: creación de la app y desarrollo. Hubo ciertas partes que hubo que refrescar por parte del autor, como MySQL y Git. A lo largo del proyecto, hicieron falta ciertas librerías que se fueron instalando sobre la marcha.

```
user: fer on fer-IdeaPad-3-15ITL6 🌿 main via 🐍 pyenv via 6GiB/15GiB | 0B/2GiB un
derwent 37m15s
🕒 [ 12:06:24 ] > git log --reverse
commit e82d5bab0994ded2f6053b197253dfe90ffad93e
Author: ftorrij <ftorrij@gmail.com>
Date: Thu Oct 14 12:23:21 2021 +0000
```

Figura 3.1: Comienzo del proyecto.

## 3.2 Etapa 2: Funcionalidades a desarrollar

Se tomó la decisión de implementar cinco funcionalidades de interés para un usuario: Sueño, Gastos | Ingresos, Ocio, Deporte y Estudio.

En el transcurso de cada actividad que se iba desarrollando, aparecían errores. Por ejemplo, al principio aparecían errores de usuario, que se solventaron obligando al usuario a registrarse antes de empezar a usar la app (hay otras formas de arreglarlo). También aparecían fallos de sintaxis, pero siempre el *Traceback* (rastreador) ayudaba a averiguar qué pasaba y se arreglaba. Otros fallos que no se mostraban, pero que para el usuario sí que deben ser fallos, son los que crean conceptos sin sentido, como por ejemplo con las fechas: tú no puedes crear horas negativas, aunque el número exista, hay que hacer ver a la aplicación que en el mundo real eso no es posible. O por ejemplo crear sucesos posteriores a la fecha actual, de igual manera hay que hacer ver a la aplicación que debe ser un error. Para ello, se crearon los *validadores*, que son creados manualmente por el programador y hace ver al usuario que está cometiendo un error y que debe corregirlo si quiere crear un determinado ítem. Se explicará con detalle en la sección 5.2.1 de los formularios.

## 3.3 Etapa 3: Diseño e Imagen

Después de que todo estuviese organizado y claro, había que hacer un diseño de cómo sería la aplicación a vista de usuario (frontend). Se ha hecho un diseño acorde a una aplicación web, pero con el móvil también es una versión amigable.

Primero, era la organización de cada parte: dónde estaría el menú y el perfil del usuario. Al principio, la adaptación de cada parte a la hora de redimensionar la ventana era pésima, tal y como aparece en la Figura 3.2; hubo que ir adaptando para que a la hora de poner un

tamaño cualquiera de ventana, se viese bien y sin complicaciones. Después, yendo a cada aplicación que muestra el menú, se quería hacer lo más intuitivo posible, de manera que mientras interactuase el usuario con la aplicación, pudiera ver, sin necesidad de seguir un manual, qué sucedía en cada acción y qué se mostraba (todas las aplicaciones tienen una estructura similar). Al principio, el índice de los ítems se mostraba en una URL separada de la URL de creación, pero se pensó que era mejor tener una especie de *carpeta*, que contuviese todos los ítems de cada aplicación; de esta forma es muy cómodo entrar en la aplicación correspondiente y tener tu carpeta organizada. Todo se iba haciendo de manera incremental y de manera que si el usuario pincha en cualquier parte pueda ver lo que significa.

Lo último que se diseñó fue la edición de cada ítem. Al empezar a desarrollarlo, era una simple URL que te llevaba al ítem en cuestión y podías editarlo sin más, pero se pensó que era una buena idea hacer que a la hora de editar, saliese una ventana emergente o *popup*. Una ventana emergente es una ventana que aparece encima del contenido, en este caso encima de las características del ítem, que sobresale haciendo que la parte de detrás esté como más difuminada, destacando y llamando más la atención. Es más cómodo a la hora de interactuar, ya que, se abre la ventana, guardas o haces lo que consideres y vuelves a estar rápidamente en el ítem, quitando tiempos de espera a diferencia que si cargamos una URL aparte.

Finalmente, se jugó con una paleta de colores que fuese agradable a la vista y atrayente para quién emplee la aplicación.

## 3.4 Etapa 4: Visualización gráfica

Esta etapa muestra el núcleo de este proyecto, que es la visualización tanto 2D como 3D de los datos. Para el autor ha sido lo más novedoso de aprender y lo más divertido. En un principio no era parte ni siquiera del proyecto, pero al realizar su desarrollo, se vio la gran utilidad que conlleva su uso, hasta tal punto de convertirse en la parte más atractiva.

En el inicio de la creación de la aplicación, todos los datos se mostraban mientras el usuario iba interactuando con cada aplicación. Cuando querías saber algo de un ítem específico, se iba al ítem y veías todos sus detalles. Llegados a este punto la aplicación ya era funcional, pero faltaba algo, aunque el trabajo que había detrás era considerable. Se tuvo la idea de mostrar los datos de manera gráfica, de tal manera que con un simple vistazo se pueda ver un resumen entre dos fechas, inicial y final, de una aplicación específica. Si se quiere ver los datos de una manera más concreta, la función que había inicial sigue operativa, ya que muestra el funcionamiento interno del proyecto. Teniendo claro que se quería mostrar los datos de manera gráfica, se podía llegar aún más lejos, que es la relación que puede haber entre aplicaciones, y así es. Se empezó queriendo visualizar, en formato 2D, la

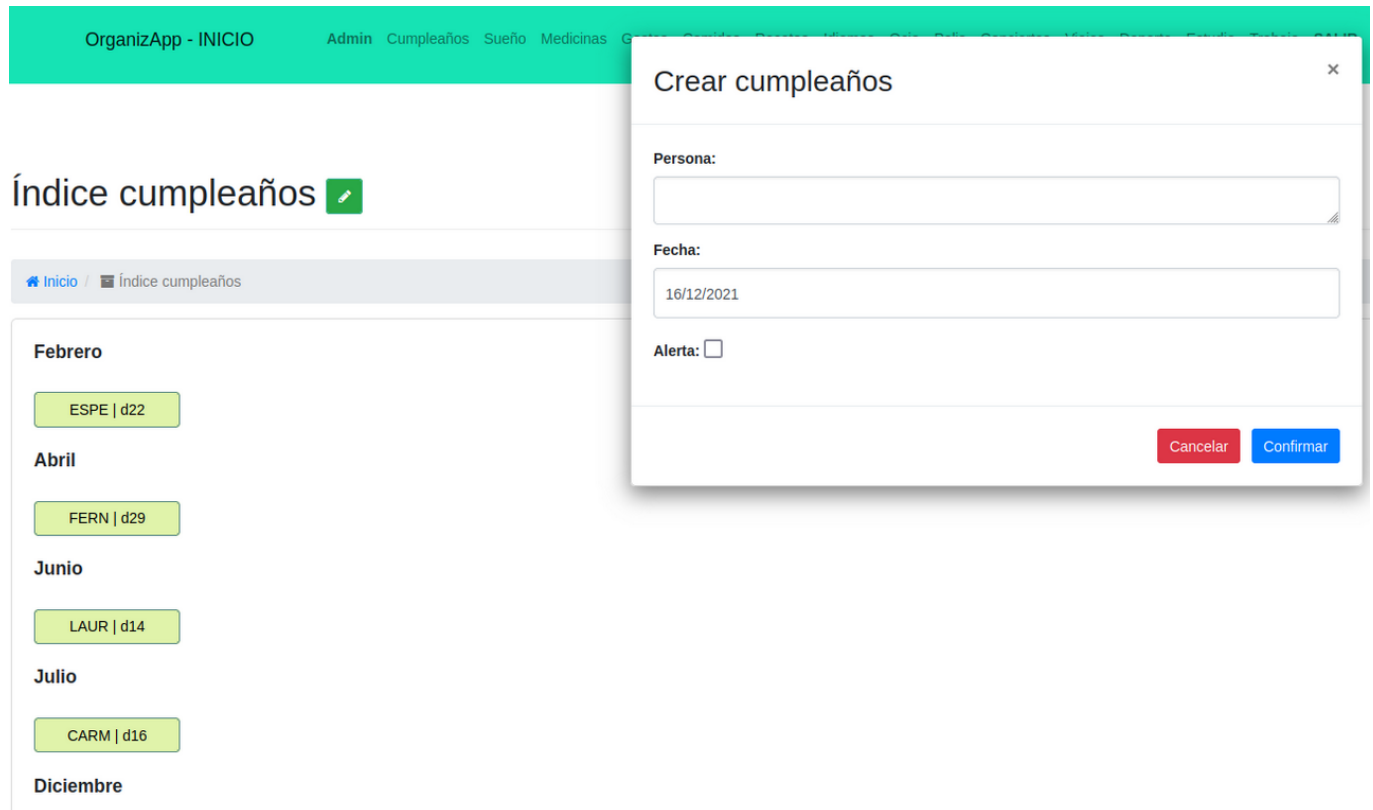
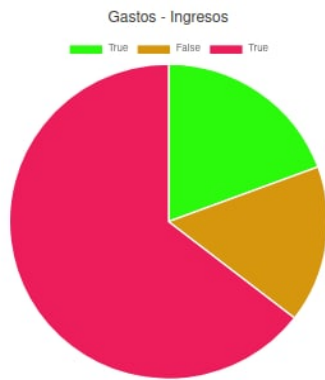
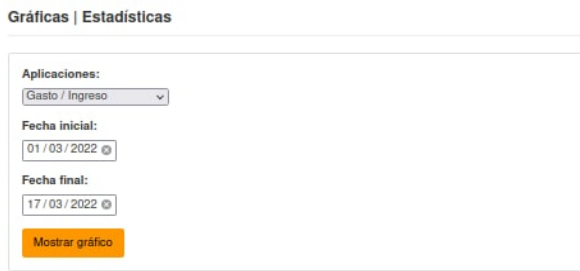


Figura 3.2: Versión beta de aplicación. Inicios con ajuste de ventana.

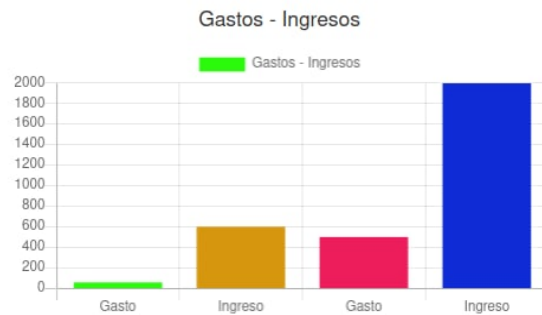
relación entre *gastos* e *ingresos*, algo que cualquier usuario que quiera llevar el control de su capital querría. Para realizar tal gráfica en 2D (como en 3D que hablaremos más adelante) se ha tenido que aprender a usar la librería de Python **Plotly**. Antes de usar esta librería, se usó *Chart.js*, pero los resultados eran mucho más simples y menos detallados que con Plotly, por lo que se optó por usar mejor esta última. Se puede ver un resultado usando Chart.js en la Figura 3.3.

Después de probar y probar la librería Plotly, se descubrió que no solo se podía mostrar gráficos en 2D, sino también en 3D (ver Figura 3.4). Cuando ya se tuvo claro qué aplicaciones tenía sentido relacionarlas entre ellas, se empezó a probar la visualización de datos en este formato, aunque en un principio se iba a hacer todo en 2D. No solo se pueden visualizar los datos de manera cómoda, sino que se puede individualizar para ver los datos de una de las dos aplicaciones que se muestran. De esta manera, se terminó de hacer esta última parte de la aplicación; todos los datos se muestran jugando con colores y formas, una manera más de practicar con Plotly.





(a) Diagrama circular.



(b) Histograma.

Figura 3.3: Imagen 2D de Gastos e Ingresos con Chart.js.



(a) Imagen 2D.



(b) Imagen 3D.

Figura 3.4: Imagen 2D y 3D visualizando datos de Ocio y Estudio.

### 3.5 Etapa 5: Despliegue y prueba de usuarios reales

Estando el proyecto ya acabado, se dispuso a ponerlo en red, para que personas ajenas al autor pudiesen probar la aplicación. En un principio se usó un hosting dedicado llamado *Pythonanywhere*. Se probó y se usó sin problema, pero a día de hoy, su versión gratuita tiene muchos inconvenientes de cuota, no pudiendo desplegar todo el contenido del código o no visualizando todas las partes de la aplicación de cara al usuario. Después de desechar la idea de usar una versión de pago, se buscó otra alternativa, mucho más fácil y mucho más rápida: **ngrok**. Esta herramienta se encarga de crear un túnel específico hacia un servidor local, que puede ser por ejemplo una Raspberry Pi que tengas en casa. Todos los problemas que había con *Pythonanywhere*, se solventaron con esta solución.

---

No se ha mencionado todas las reuniones que se han tenido con el tutor, parte clave en el desarrollo tanto del proyecto como de esta memoria.

# Capítulo 4

## Funcionalidades y uso

En este capítulo, se describe detalladamente cada apartado de la aplicación, las características de cada una, su funcionalidad y cómo usarse.

### 4.1 Visión general

La aplicación tiene su base en seis tablas: *Dreams* (Sueño), *Expenses* (Gastos | Ingresos), *Leisures* (Ocio), *Sports* (Deporte), *Studies* (Estudio) y *Profile* (Usuario). Todas tienen relación con *Profile*, tal y como se muestra en la Figura 4.1, ya que cada usuario tiene su *Perfil* de Usuario con diferentes datos en cada actividad/funcionalidad.

### 4.2 Actividades

Aquí se explicarán las diferentes aplicaciones creadas en el proyecto; funcionamiento de cada una y detalles de cada apartado. Detallar que si no se está registrado, te obliga a registrarte para poder interactuar con cada aplicación. Mientras vas seleccionando la sección (creación, índice, detalles) irá apareciendo en una barra arriba, donde se podrá navegar con comodidad hacia delante o hacia detrás.

#### 4.2.1 Sueño

Sueño o como aparece en la Base de Datos *dreams*, es la actividad que gestiona las horas dormidas diarias por el usuario. Puede mostrar cada día la fecha, el nombre del usuario, las

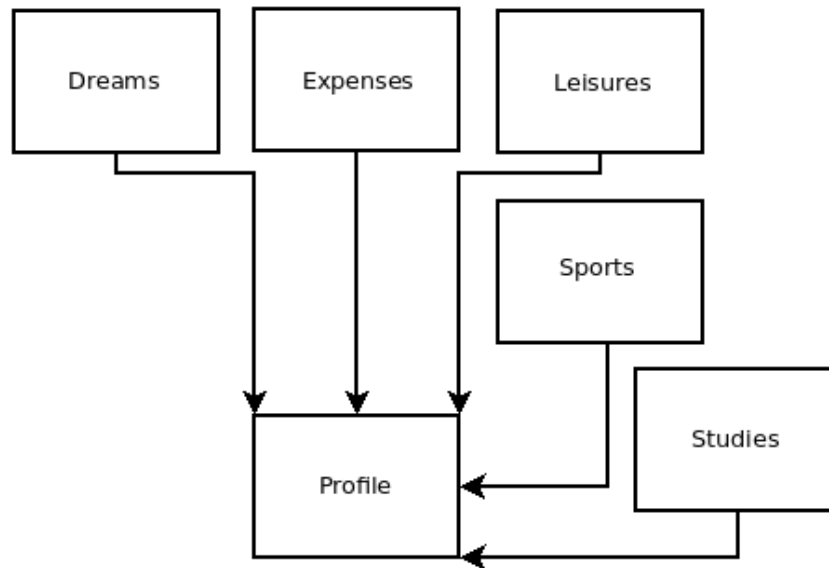


Figura 4.1: Diagrama de la aplicación.

horas dormidas y si padece alguna enfermedad relacionada con el sueño (apnea, insomnio, narcolepsia).

En esta aplicación puede realizar las siguientes acciones:

- **Crear.** En la URL `/dreams/dreams_create/`, se puede registrar las horas que se han dormido ese día y si se padece de alguna enfermedad, tal y como aparece en la Figura 4.2.
- **Visualizar índice.** Mostrará en la URL `/dreams/dreams_index/[YEAR]` o dando a la carpeta/botón con el año actual donde se crea el sueño (ver Figura 4.2), todo el registro que se ha estado haciendo, por mes y día (ver Figura 4.3).
- **Visualizar detalles.** En el índice al pinchar en algún día concreto, mostrará los detalles de ese sueño; aparecerá lo que se ha hecho al crear el sueño de ese día.
- **Eliminar.** Dentro de un sueño concreto, se puede eliminar si no se quiere dentro de la Base de Datos.
- **Editar.** Se puede editar un *sueño* concreto si el usuario ha cometido algún error al crearlo. La forma que tiene este apartado se puede ver en la Figura 4.4.

Tanto los detalles del sueño, como la eliminación y la edición, se muestra una imagen explicativa en la Figura 4.5.

## (Crear sueño)

[Inicio](#) / [Sueño](#)

2022

**Usuari@:**

**Día:**

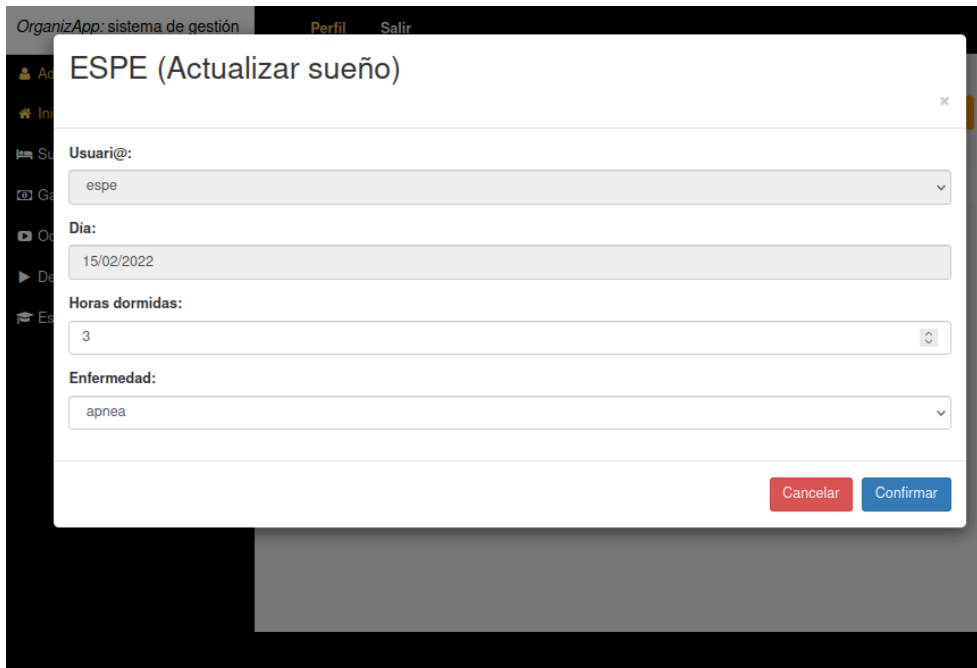
**Horas dormidas:**

**Enfermedad:**

Figura 4.2: Registrar horas de sueño.

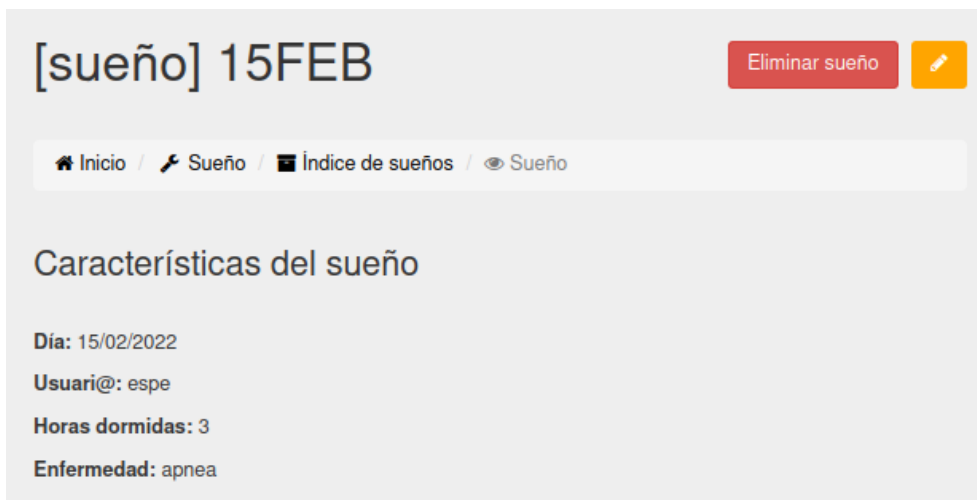


Figura 4.3: Índice de sueños.



The screenshot shows a web application interface for editing a dream record. The main window is titled "ESPE (Actualizar sueño)". It contains several input fields: "Usuari@" with a dropdown menu showing "espe"; "Dia:" with a date field showing "15/02/2022"; "Horas dormidas:" with a numeric input field showing "3"; and "Enfermedad:" with a dropdown menu showing "apnea". At the bottom right of the form, there are two buttons: "Cancelar" (red) and "Confirmar" (blue). The background shows a sidebar with navigation icons and a top navigation bar with "OrganizApp: sistema de gestión", "Perfil", and "Salir".

Figura 4.4: Editar sueño.



The screenshot shows the details page for a dream record. The title is "[sueño] 15FEB". In the top right corner, there are two buttons: "Eliminar sueño" (red) and an edit icon (yellow). Below the title, there is a breadcrumb navigation path: "Inicio / Sueño / Índice de sueños / Sueño". The main content area is titled "Características del sueño" and lists the following details: "Dia: 15/02/2022", "Usuari@: espe", "Horas dormidas: 3", and "Enfermedad: apnea".

Figura 4.5: Detalles, botón de edición y botón de eliminación de Sueño.

### 4.2.2 Gastos | Ingresos

Esta pestaña del menú sirve para controlar los gastos e ingresos por mes del usuario. Al igual que la anterior aplicación, se podrán realizar las mismas cinco acciones, pero con características diferentes. Por ahorrar espacio se omitirán algunas imágenes, ya que el concepto es parecido en todas las aplicaciones; se pondrán aquellas que sean necesarias. Las acciones a realizar son:

- **Crear.** En la URL `/expenses/expenses_create/`, se crea un gasto o ingreso. Importante detallar que al crear un gasto/ingreso, se actualizará automáticamente en el perfil del usuario su cantidad de dinero debido a un *comando* creado para tal fin. Los detalles del mismo se explicarán en el Capítulo 5.
- **Visualizar índice.** En la misma URL donde se crea el gasto/ingreso hay una carpeta negra con el año actual, que al pinchar, se pueden visualizar todos los gastos e ingresos del año tal y como se muestra en la Figura 4.6.
- **Visualizar detalles.** Dentro del índice si se pincha cualquier gasto o ingreso, aparecerán los detalles del mismo: si es un gasto o ingreso (ver Figura 4.7), día, usuario, cantidad, tipo y nota. En esta misma sección se puede eliminar o editar; se explicará a continuación.
- **Eliminar.** Al igual que en la aplicación anterior, en los detalles del gasto/ingreso hay un botón donde poder eliminarlo si se quiere.
- **Editar.** En los detalles habrá un botón donde poder editar, similar a la aplicación anterior, donde aparecerá una ventana emergente donde cancelar o confirmar esta acción.

### 4.2.3 Ocio

Ocio o *leisures* en la BBDD, es la parte de la agenda donde el usuario puede registrar lo que hace en su tiempo libre. Tiene acceso a cualquier día del mes en la sección de *índice* pinchando en el botón negro con el año actual, al igual que en las anteriores aplicaciones, donde aparecerán los detalles del mismo: día, usuario, descripción, tiempo empleado, sensación y tipo de ocio. Creación, índice, detalles, eliminación y edición, es similar a las anteriores aplicaciones; cambia el estilo y la organización.





Figura 4.6: Índice de gastos e ingresos.



(a) Gasto.

(b) Ingreso.

Figura 4.7: Detalles de Gasto o Ingreso.

#### 4.2.4 Deporte

Esta sección del proyecto (*sports*) almacena la actividad deportiva del usuario, donde puede registrar la cantidad y el tipo de deporte que realiza y después de pincharlo en el menú, se maneja de la misma forma que se ha dicho en anteriores aplicaciones. Los detalles que aparecen son: día, usuario, nombre del deporte, zona del cuerpo, tiempo empleado, peso perdido, tipo de deporte, si es o no competición y si hay alguna lesión. En la Figura 4.8 aparece un ejemplo de índice y en la Figura 4.9 aparece un ejemplo de un día concreto.



Figura 4.8: Índice de deportes.

#### 4.2.5 Estudio

Estudio o *studies* en la BBDD, almacena la actividad estudiantil del usuario con: fecha, asignatura y horas estudiadas. El procedimiento de interacción es el mismo que se ha dicho anteriormente: creación, índice, detalles, eliminación y edición.

### 4.3 Características personales, menú y gráficas y estadísticas

En la *home* (página principal), se puede visualizar el perfil del usuario con las características principales. Está dividida en tres partes:



Figura 4.9: Detalles de deporte.

- **Características personales.** Se ve la *edad* del usuario, su *dinero* ahorrado hasta el momento, *trabajo* donde ejerce su actividad profesional, *biografía* que explica qué realiza laboralmente y un *link* que enlaza a un sitio que le gusta. Se puede ver un ejemplo en la Figura 4.10.
- **Menú de aplicaciones.** Muestra las diferentes aplicaciones que se han desarrollado (ver Figura 4.11). Cada aplicación es una actividad que realiza el usuario habitualmente.
- **Gráficas y estadísticas.** En la Figura 4.12 se observan las tres diferentes vistas generales de todas las aplicaciones. Se tiene en cuenta la relación que hay entre ellas y como consecuencia, se muestra una comparativa por tamaños y número de ítems que permite ver el progreso de cada una. Esta parte se explicará con más detalle a continuación.

El tercer punto es el que ofrece, a «vista de pájaro», una visualización de todas las aplicaciones para ver su progreso en el tiempo. Hay dos tipos de visualización: 2D y 3D.

En el formato 2D se visualiza la aplicación *Expenses*, que muestra una comparativa desde una fecha inicial hasta una fecha final de gastos e ingresos. Se tiene al alcance el máximo y mínimo ingreso/gasto, la media y cada uno de los gastos e ingresos que se han ido haciendo en ese rango temporal, tal y como se muestra en la Figura 4.13.



Figura 4.10: Características personales del usuario.

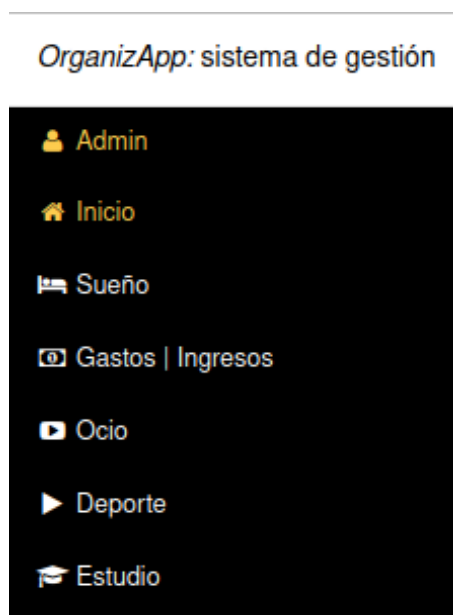
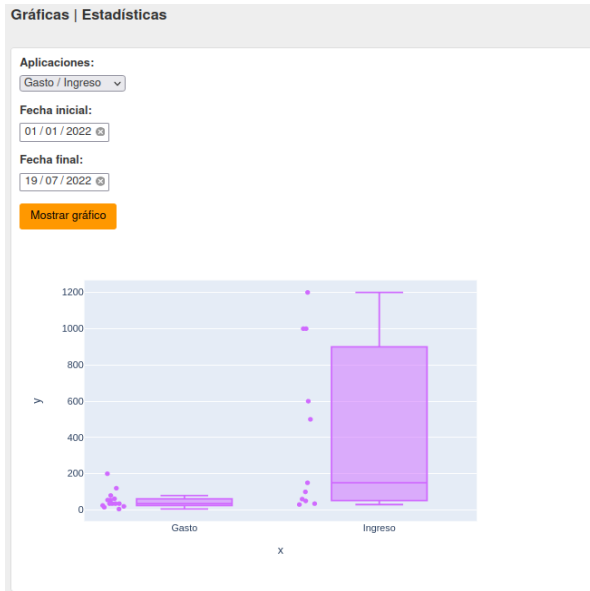


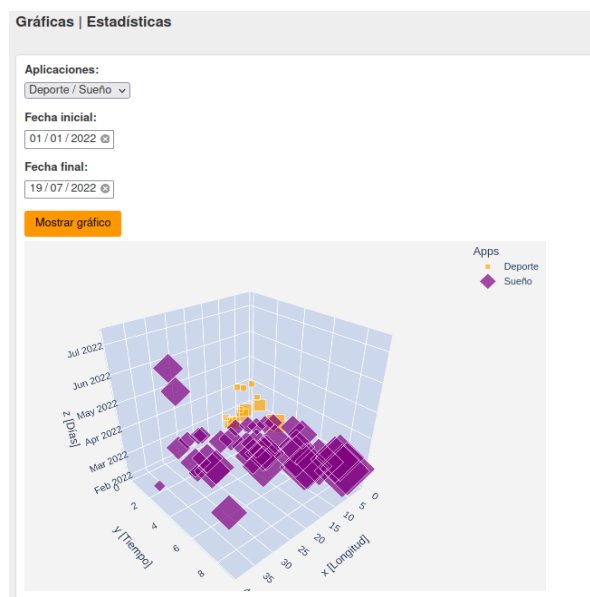
Figura 4.11: Menú de la aplicación.



(a) Gastos e Ingresos.



(b) Ocio y Estudio.



(c) Deporte y Sueño.

Figura 4.12: Gráficas y Estadísticas.

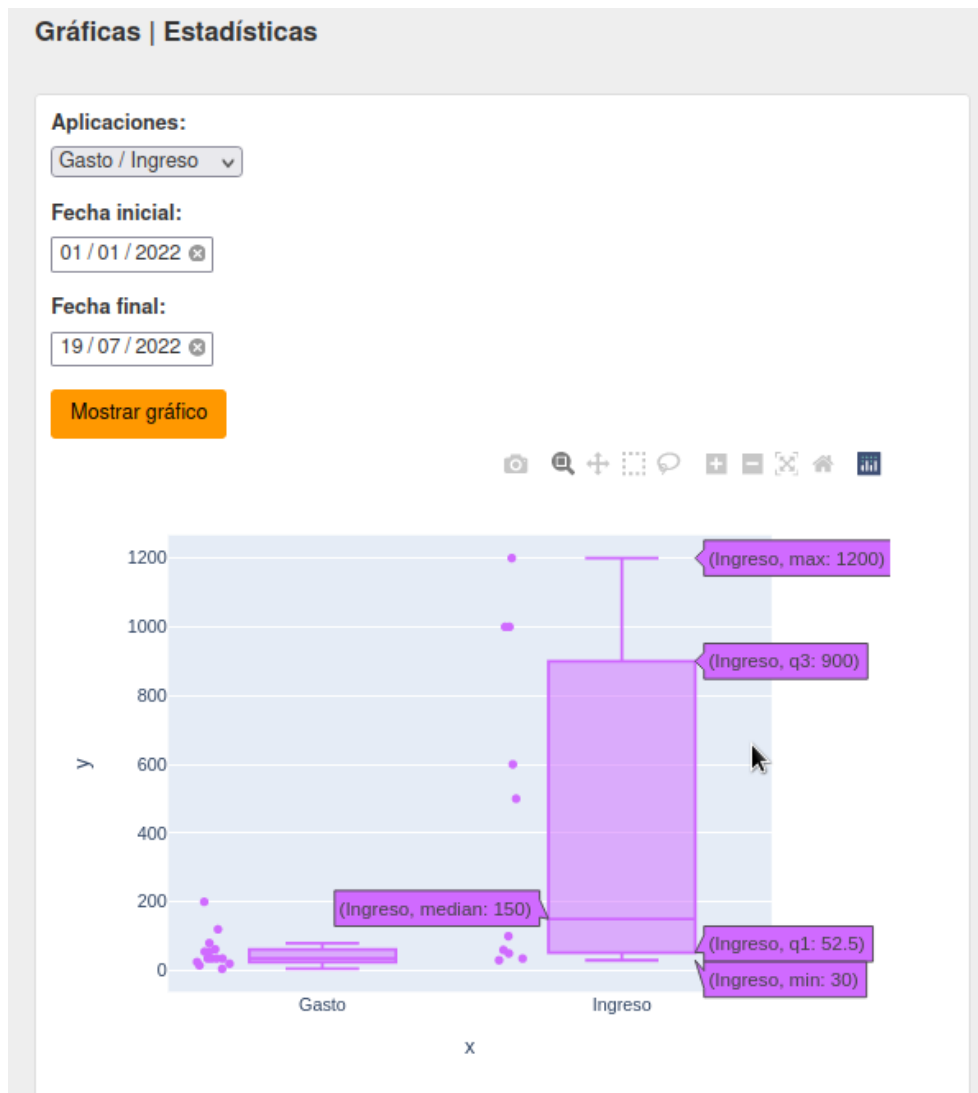


Figura 4.13: Visualización 2D de Gastos e Ingresos.

En el formato 3D, se visualiza el resto de aplicaciones: *Leisures*, *Sports*, *Dreams* y *Studies*. Está la comparativa entre ocio y estudio como muestra la Figura 4.14 y entre deporte y sueño como muestra la Figura 4.15. En cada una, se pueden ver los campos más importantes de cada actividad; por ejemplo en el caso de Deporte: tipo de deporte, día que se realizó ese deporte y tiempo empleado. Es una forma cómoda de saber cómo va tu *agenda*. La gráfica se puede ampliar, mover, descargar, rotar e individualizar a una determinada aplicación. La biblioteca utilizada ha sido **Plotly**<sup>1</sup>.

Puntualizar, que si no hay datos dentro del rango temporal que se elija, aparecerá el mensaje *No hay datos*.

## 4.4 Registro

Es la aplicación donde el nuevo usuario puede registrarse dentro de la aplicación web y permite interactuar con las **Actividades**. Si no se está registrado o no se ha iniciado sesión, aparecerá una pantalla como la que aparece en la Figura 4.16 para registrarse o acceder a la aplicación. En el caso de *iniciar sesión* solo hay que poner las credenciales y si es la primera vez que se accede, hay que registrarse dando a *Registro* en la pantalla principal. Una vez ahí, hay que seguir los pasos que aparecen en la Figura 4.17.

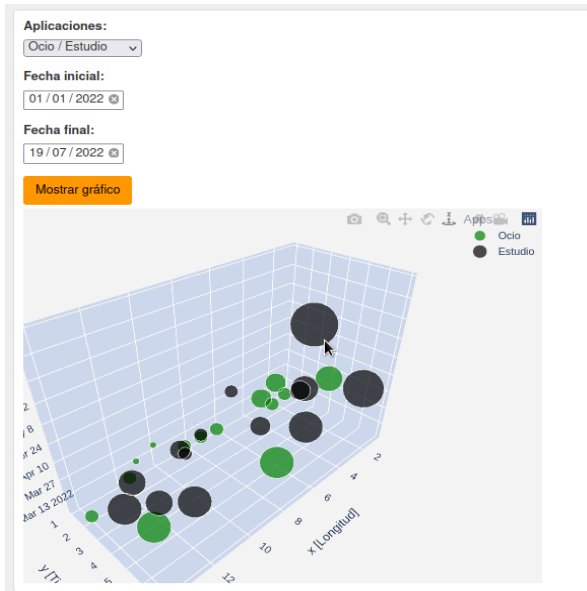
Para registrarse en la aplicación, hay que irse al menú de la izquierda abajo del todo donde pone Registro. Ahí se pincha y se siguen las indicaciones que aparecen en la pantalla. Finalmente, se da a Confirmar y ya estaría el usuario registrado correctamente. Cualquier error o fallo que surgiese, la aplicación ya se encarga de notificarlo por pantalla para corregirlo.

## 4.5 Perfil

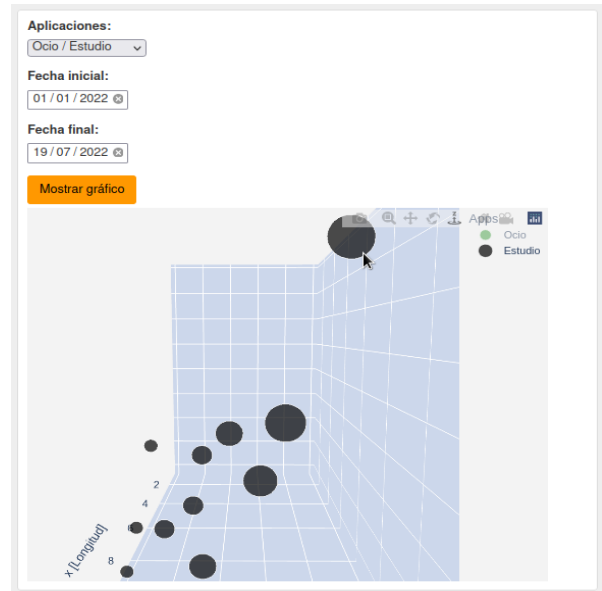
En el menú de la izquierda, tras registrarse, se pueden ver y editar los detalles del perfil. Tras dar a Actualizar si se han hecho cambios, actualizará con los nuevos atributos del perfil. En la página principal aparecen todos los detalles del usuario.

---

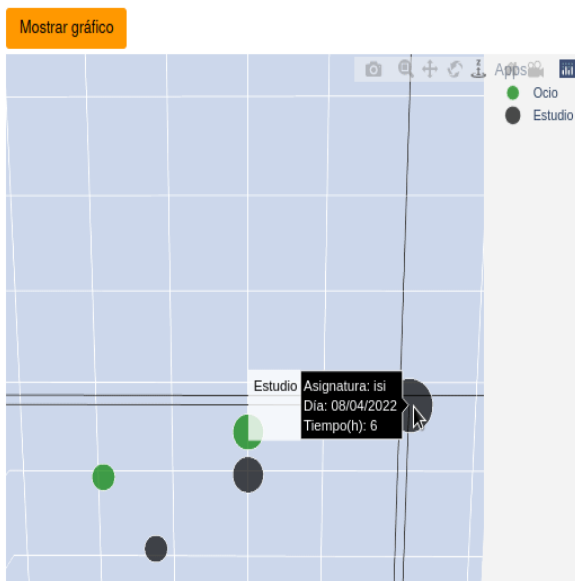
<sup>1</sup><https://plotly.com/python/>



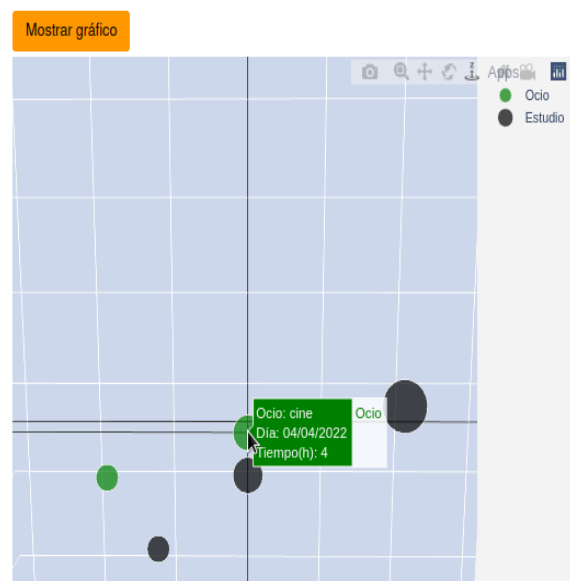
(a) Vista general 3D de Ocio y Estudio.



(b) Vista individualizada de la actividad Estudio.



(c) Detalles y características de Estudio.



(d) Detalles y características de Ocio.

Figura 4.14: Gráficas de Estudio y Ocio.



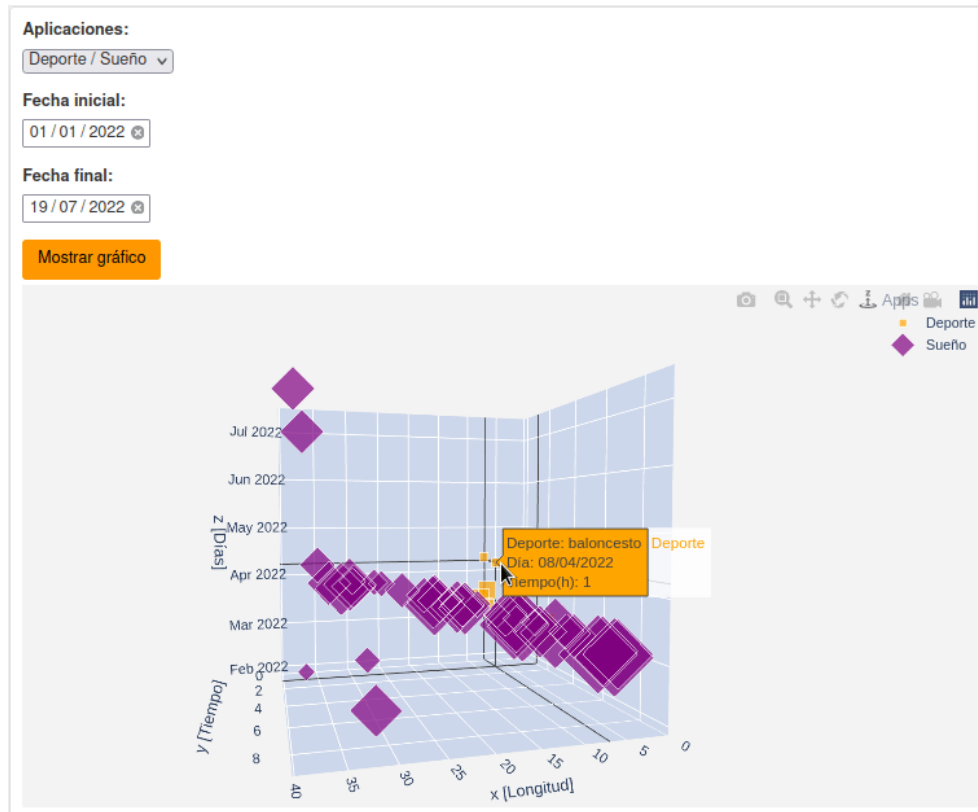


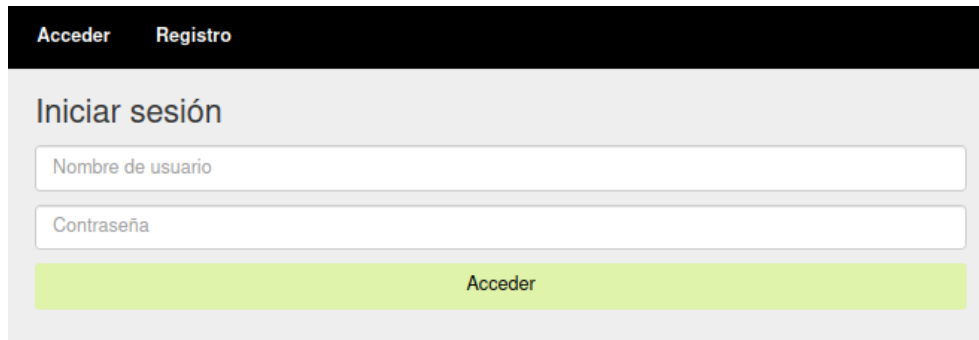
Figura 4.15: Visualización 3D de Deporte y Sueño.

## 4.6 Salir

Si se está dentro de un perfil de usuario, se puede salir para que pueda loguearse con otro usuario dando a Salir en el menú. Aparecerán las opciones de Acceder o Registro automáticamente.

## 4.7 Administrador

Solo el Administrador podrá ver la opción de Admin en el menú de la izquierda, para poder manejar gráficamente la Base de Datos del proyecto. El administrador habrá sido creado previamente en el directorio donde aparece el programa `manage.py` de la siguiente manera: `python3 manage.py createsuperuser`.



Acceder Registro

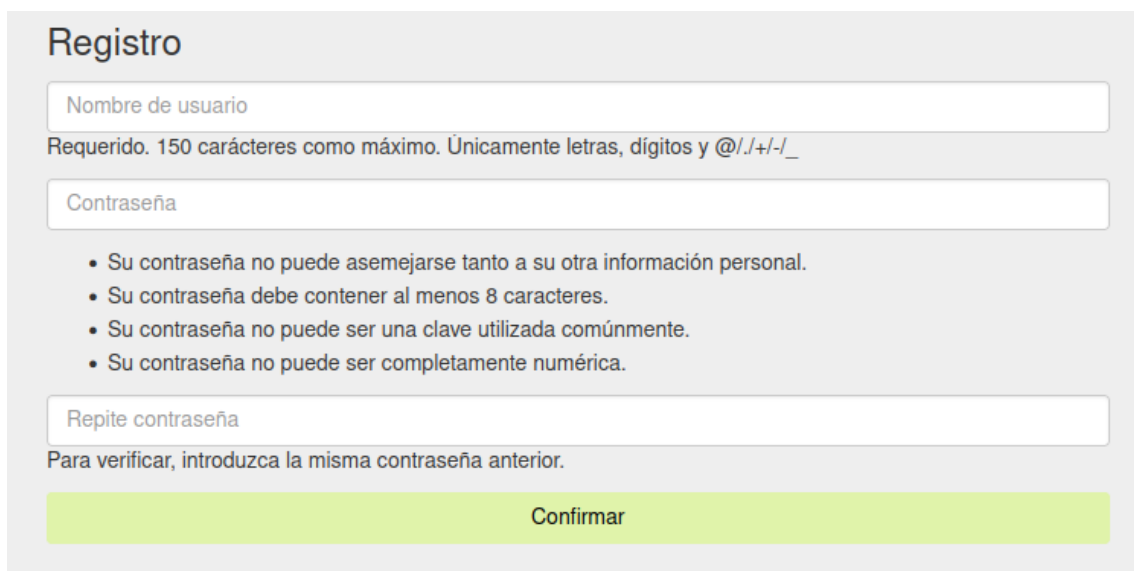
### Iniciar sesión

Nombre de usuario

Contraseña

Acceder

Figura 4.16: Página de inicio de sesión.



### Registro

Nombre de usuario

Requerido. 150 caracteres como máximo. Únicamente letras, dígitos y @/./+/\_

Contraseña

- Su contraseña no puede asemejarse tanto a su otra información personal.
- Su contraseña debe contener al menos 8 caracteres.
- Su contraseña no puede ser una clave utilizada comúnmente.
- Su contraseña no puede ser completamente numérica.

Repita contraseña

Para verificar, introduzca la misma contraseña anterior.

Confirmar

Figura 4.17: Formulario de registro.

# Capítulo 5

## Arquitectura interna y parte técnica

### 5.1 Estructura del código: MVC y plantillas

Como ya se dijo, **Django** se organiza en tres partes (Modelo | `models.py` - Vista | `views.py` - Controlador | `urls.py`) y luego las plantillas renderizan las vistas para mostrar al usuario el *Front End* o lado del cliente de la aplicación web.

Para que todo funcione, se ha tenido que estructurar el proyecto en siete aplicaciones: *core*, *dreams*, *expenses*, *leisures*, *registration*, *sports* y *studies*. Estas siete aplicaciones a su vez se agrupan en 3 grupos: actividades, registro y núcleo, que a continuación se explicarán, y todas ellas comparten las tres partes de Django y las plantillas; he aquí la magia y la potencia de Django. Luego aparte, hay un directorio llamado *media* donde guardar imágenes a usar en la app, el programa `manage.py` que se encarga de ejecutar los comandos de Django, de los cuales está *runserver* que arranca la aplicación en un servidor y el programa `settings.py`, que son los ajustes principales de la aplicación (lenguaje, carpeta donde guardar las imágenes, dirección de correo, quién es el administrador, etc). Un esquema simplificado se puede ver en la Figura 5.1 cuyos colores diferencian cada parte y una visión más técnica, pero quizás más difícil de entender, es la que se muestra en la Figura 5.2. Aquí se puede ver, qué relación y qué función proporciona cada parte (MVC) <sup>1</sup>.

Si se quiere ver el código fuente del proyecto: <https://gitlab.com/ftorrij/organizapp>.

---

<sup>1</sup><https://git-scm.com/>

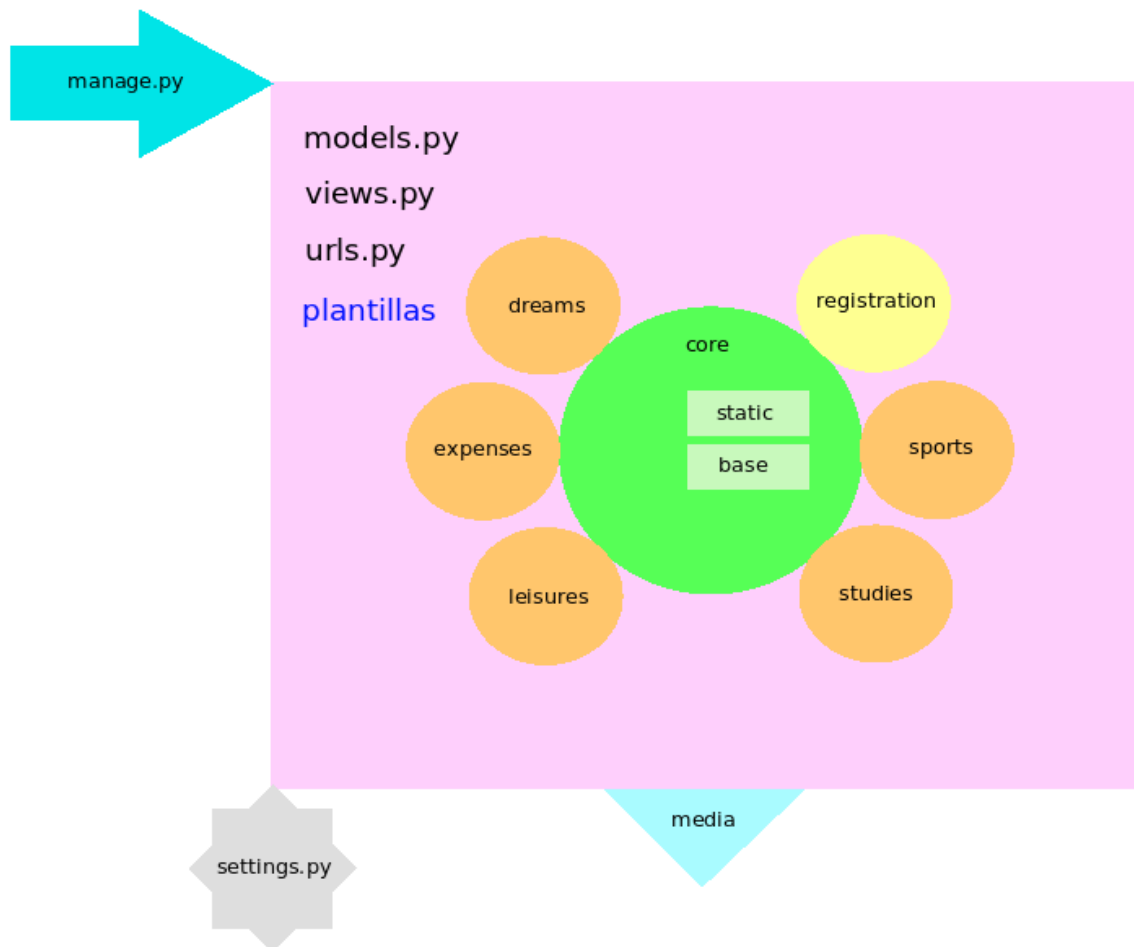


Figura 5.1: Esquema general interno de la aplicación.

## 5.2 Actividades

Son las aplicaciones que el usuario utiliza dentro de la *agenda*, cuya apariencia ya se ha visto anteriormente. En total son cinco: *dreams*, *expenses*, *leisures*, *sports* y *studies*. Para no repetir información, se va a detallar la estructura de código de una de ellas, por ejemplo, la aplicación *expenses*, ya que todas siguen el mismo patrón. El siguiente capítulo abordará cómo se ha hecho la parte de validación y errores con cada una.

Dentro de la aplicación que se va a abordar ahora, destacar que utiliza, a diferencia de las otras cuatro, un *comando* hecho por el programador en *Django*, que actualiza la cantidad de dinero del usuario. Se puede hacer de una manera más sencilla, pero se ha querido mostrar el potencial que se puede conseguir con los comandos de Django, aparte de los que vienen ya por omisión (*migrate*, *makemigrations*, ...). Es algo muy útil si se quisiera lanzar tareas automáticas en el *crontab*<sup>2</sup> de un sistema GNU/Linux, por ejemplo; en este caso se ha realizado directamente dentro del código, pero se podría lanzar aparte si se

<sup>2</sup><https://blog.desdelinux.net/cron-crontab-explicados/>

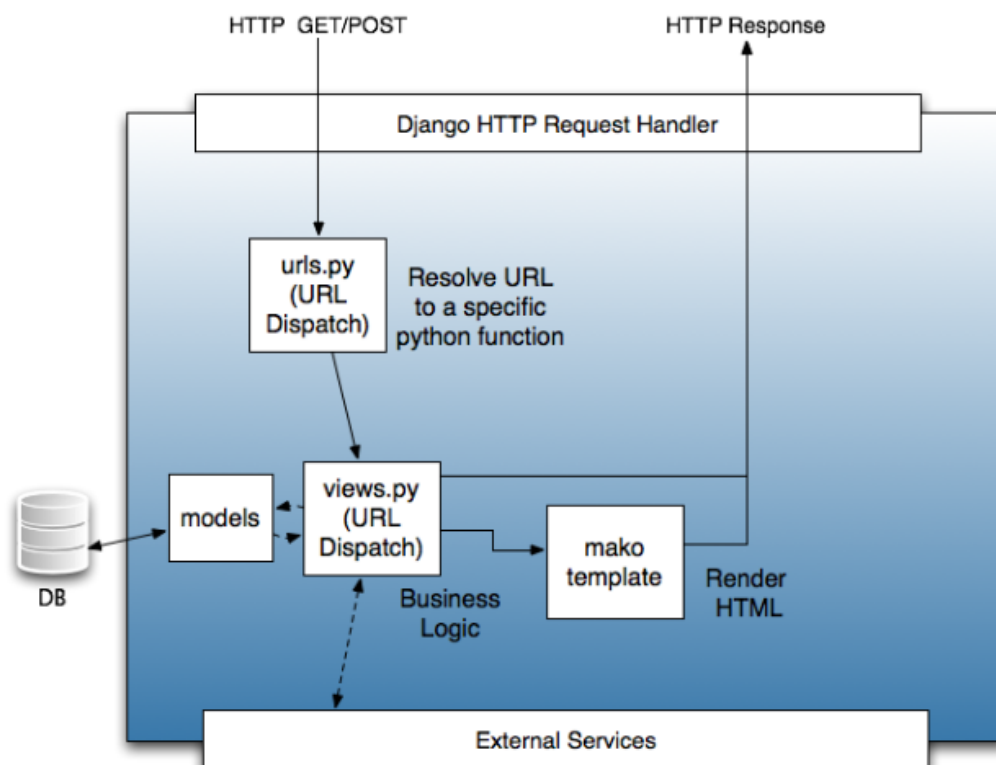


Figura 5.2: Esquema general interno de Django: MVC.

quisiera. No se va a entrar en detalle sobre esto (puede consultarse en el enlace que hay al pie de página lo comentado sobre el crontab y su uso).

### 5.2.1 Expenses

Un ejemplo del interior de una aplicación se puede ver en la Figura 5.3. Se va a explicar un poco las partes más generales a continuación, y después las más relevantes con más detalle:

- **admin.py.** Se registra el modelo (models.py), poco más.
- **apps.py.** Configura cómo se quiere que la aplicación aparezca en el *Panel de Administrador*, sitio el cual se accede en el menú de la izquierda de la app si eres Administrador o *Staff*; hay que tener los permisos necesarios para entrar y es donde poder manejar de una manera visual la Base de Datos. No se van a abordar muchos detalles ya que no es relevante y tampoco se ha usado en exceso en el transcurso del proyecto. Es bastante sencillo e intuitivo.
- **migrations.** Es el directorio donde se guarda el registro de los cambios que se van

```
user: fer on fer-IdeaPad-3-15ITL6 main via pyenv via 6GiB/15GiB | 0B/2GiB
[ 12:33:54 ] > ls | column -t | sed '/_.*d/'
admin.py
apps.py
forms.py
migrations
models.py
templates
templatetags
tests
urls.py
views.py
```

Figura 5.3: Parte interna de una aplicación.

a realizan en la Base de Datos usando el comando de Django *makemigrations*. Luego con *migrate* se aplican esos cambios.

Ahora se explicarán los ficheros y carpetas más relevantes, en lo que a desarrollo del proyecto se refiere, por separado. Hay que recordar que es un patrón idéntico para todas las aplicaciones.

### forms.py

Es la parte de la aplicación donde se customizan los formularios que guardan los datos introducidos en la Base de Datos y es usada en las vistas (*views.py*). Hay dos tipos: *forms.Form*, donde el formulario es tal cual se construye y *forms.ModelForm*, donde se aprovecha el modelo para el formulario. Dependiendo de lo que se quiera, interesará más uno que otro. Para más información se puede ir a la página oficial de Django: <https://www.djangoproject.com/>.

Están las siguientes funciones:

- **def \_\_init\_\_**. Inicializa los atributos de los objetos.
- **Validadores: def clean\_[ATRIBUTO]**. Hace que los atributos de los objetos cumplan unas determinadas características. Si no se cumplen esas características, saldrá un mensaje de error, también personalizado, que indica qué se ha hecho mal. Resumiendo mucho, esto se gestiona en *forms.py* pero se personaliza en las *plantillas* donde se usa el formulario.
- **class Meta**. Hace que los atributos sean de una determinada manera/tipo en el formulario.

## models.py

Crea mediante una clase, la tabla en la Base de Datos para *Expenses* (de igual manera para otra aplicación); define los campos/atributos del objeto.

Como se ha mencionado antes, *expenses* es especial al resto de aplicaciones ya que contiene la creación de un comando en Django, que actualiza la cantidad de dinero del usuario si se hace algún movimiento de gasto o ingreso. Internamente dentro del código, es usado en *views.py* en las funciones que realizan esta acción, ya sea editar, crear o eliminar. La clase *ExpensesManager* dentro de *models.py*, es la que se encarga de la lógica del comando. Está puesto un *log* que va informando de los cambios que van sucediendo y que pueden verse en el fichero de log creado *tfg.log*.

## templates

Las plantillas o *templates*, son las encargadas de renderizar las vistas para mostrar al usuario final el resultado de ejecutar el código de manera accesible y poder interactuar con la aplicación web.

Todas las actividades/aplicaciones siguen el mismo patrón; difieren en los estilos, en la manera de visualización y en el nombre de la plantilla, pero todas tienen cuatro plantillas con las mismas características:

- **[APP]\_create.html**. Renderiza la vista *expenses\_create* y es la encargada de mostrar el formulario de creación de un gasto/ingreso. La vista proporciona a esta plantilla el formulario *ExpensesForm* de *forms.py*.
- **[APP]\_detail.html**. Renderiza la vista *expenses\_detail* y es la encargada de mostrar los detalles de un gasto/ingreso.
- **[APP]\_index.html**. Renderiza la vista *expenses\_index* y es la encargada de mostrar el índice de gastos e ingresos.
- **[APP]\_edit.html**. Renderiza la vista *expenses\_edit* y es la encargada de mostrar el formulario de edición de un gasto/ingreso. La vista proporciona a esta plantilla el formulario *ExpensesForm* de *forms.py*. Tiene la peculiaridad de que es un *modal-dialog*<sup>3</sup>, es decir, aparece como un cuadro de diálogo, que es lo que normalmente se espera uno al editar evitando tiempos de espera de una pestaña a otra.

---

<sup>3</sup><https://www.tutorialesprogramacionya.com/bootstrap4ya/detalleconcepto.php?punto=50&codigo=50&inicio=40>

### templatetags

Es la parte de la aplicación donde los *templates* tienen su reserva de código para utilizar si les hace falta. Se evita de esta manera sobrecargar de código a las vistas, ya que las vistas tienen que ser lo que Django considera como vista, que es coger la Base de Datos y emplearla para un determinado fin. Así se tendrá una estructura de código limpia y ordenada. Se puede también crear un programa aparte que coja todo lo externo a las vistas, pero queda mucho más ordenado si cada aplicación tiene su *templatetags*.

### tests

Aquí se puede testear lo que se quiera de la aplicación. Es código que se usa para probar funcionalidades sin ensuciar la Base de Datos, ya sea referente a las vistas, modelo, formulario o urls.

Las buenas prácticas recomiendan que primero se hagan tests y luego se programe lo definitivo dentro de la aplicación. En este proyecto se ha hecho un poco mezcla de estos dos pasos, primero por desconocimiento del uso de los tests y segundo porque había que avanzar; pero lo correcto es lo que se ha dicho. De hecho, así se evitan posibles grandes errores en el despliegue de mejoras dentro de una aplicación grande donde colaboran muchas personas. En este caso, el riesgo era mínimo.

### urls.py

Es el controlador de la aplicación. Especifica qué vista es llamada según el patrón URL. Hay cinco vistas diferenciadas: *create*, *detail*, *delete*, *edit* e *index*; se explicarán con detalle a continuación.

Aquí hay dos tipos de *urls.py* que hay que saber diferenciar: las de las aplicaciones, que es el controlador **particular** de cada una y controlan las vistas que se han dicho antes y las de la aplicación web *organizApp*, que es el controlador **general** que maneja las urls particulares. Hay que imaginarse las generales como una caja grande y dentro de esa caja están cajas más pequeñas, que son las concretas de las aplicaciones (dicho de una manera coloquial para que se entienda bien).

### views.py

Por último está el motor que hace que la aplicación funcione: las *vistas*. Aquí está la lógica que manipula la Base de Datos para unos determinados objetivos; ya se han mencionado



por encima antes: create, detail, delete, edit e index y se resumen de la siguiente manera:

- **def [APP]\_create.** Crea mediante un formulario un objeto. Se utiliza en este caso el comando que se ha mencionado antes en la Sección 5.2.
- **def [APP]\_detail.** Detalla el objeto. Su plantilla se encarga de customizar la visualización.
- **def [APP]\_index.** Muestra la lista de objetos. Su plantilla se encarga de customizar la visualización.
- **def [APP]\_edit.** Edita el objeto mediante un formulario y se utiliza el comando que se ha mencionado.
- **def [APP]\_delete.** Elimina el objeto de la Base de Datos y se utiliza el comando que se ha mencionado.

## 5.3 Registro

Esta aplicación a diferencia del resto, está basada en CBV o *Class Based Views*, una modalidad de Django que permite modelar las vistas como clases en vez de funciones o FBV (cuyas siglas en inglés son *Function Based Views*). Se ha querido hacer así por probar esta modalidad, ya que tiene ciertas ventajas con respecto a las FBV. La más importante es que tiene un nivel más alto de reutilización de código, ya que una clase es una estructura más completa que una función y puede implementarse en ella funciones que se usan de manera común; así se puede usar esta clase para realizar un conjunto de funciones. Django ya proporciona vistas basadas en clases como plantillas a usar y extender como se quiera<sup>4</sup>. Particularizando a nuestra app, se han implementado dos clases:

- **class SignUpView(CreateView).** Crea un usuario nuevo. A su vez, contiene las funciones *def get\_success\_url* para la redirección y *def get\_form* para el formulario. Se usa la clase de Django `UserCreationForm`.
- **class ProfileUpdate(UpdateView).** Actualiza el perfil del usuario. A diferencia de la anterior clase, usa la función *def get\_object* para recuperar el objeto a editar.

---

<sup>4</sup><https://ccbv.co.uk/>

```

188 File "/home/fer/.local/lib/python3.8/site-packages/django/contrib/admin/templatetags/base.py", line 33, in render
189     return super().render(context)
190 File "/home/fer/.local/lib/python3.8/site-packages/django/template/library.py", line 214, in render
191     _dict = self.func(*resolved_args, **resolved_kwargs)
192 File "/home/fer/.local/lib/python3.8/site-packages/django/contrib/admin/templatetags/admin_list.py", line 308, in result_list
193     'results': list(results(cl)),
194 File "/home/fer/.local/lib/python3.8/site-packages/django/contrib/admin/templatetags/admin_list.py", line 284, in results
195     yield ResultList(None, items_for_result(cl, res, None))
196 File "/home/fer/.local/lib/python3.8/site-packages/django/contrib/admin/templatetags/admin_list.py", line 275, in __init__
197     super().__init__(*items)
198 File "/home/fer/.local/lib/python3.8/site-packages/django/contrib/admin/templatetags/admin_list.py", line 200, in items_for_result
199     f, attr, value = lookup_field(field_name, result, cl.model_admin)
200 File "/home/fer/.local/lib/python3.8/site-packages/django/contrib/admin/utils.py", line 278, in lookup_field
201     value = attr()
202 File "/home/fer/Escritorio/URJC-portatil/TFG/organizApp/organizapp/organizApp/leisures/models.py", line 44, in __str__
203     return self.name
204 AttributeError: 'Leisures' object has no attribute 'name'
205 [21/Apr/2022 15:48:42] "GET /admin/leisures/leisures/ HTTP/1.1" 500 351384
206 [03/May/2022 11:34:16] "GET /favicon.ico HTTP/1.1" 404 3111
207 [03/May/2022 11:34:42] "GET /static/core/css/bootstrap.min.css.map HTTP/1.1" 404 1849
208 [03/May/2022 11:35:08] "GET /static/core/css/bootstrap.min.css.map HTTP/1.1" 404 1849
209 [03/May/2022 11:35:11] "GET /static/core/css/bootstrap.min.css.map HTTP/1.1" 404 1849
210 [03/May/2022 11:35:13] "GET /static/core/css/bootstrap.min.css.map HTTP/1.1" 404 1849
211 [03/May/2022 11:35:21] "GET /static/core/css/bootstrap.min.css.map HTTP/1.1" 404 1849
212 [03/May/2022 11:35:25] "GET /static/core/css/bootstrap.min.css.map HTTP/1.1" 404 1849
213 [03/May/2022 11:36:40] "GET /static/core/css/bootstrap.min.css.map HTTP/1.1" 404 1849
214 [03/May/2022 11:36:55] "GET /static/core/css/bootstrap.min.css.map HTTP/1.1" 404 1849
215 [20/Jun/2022 07:06:26] "GET /favicon.ico HTTP/1.1" 404 3111
216 [20/Jun/2022 07:07:32] [DREAM] user [espe] | curr day [2022-06-20] | sleep_hours [5] | illness []
217 [20/Jun/2022 07:08:07] [EXPENSE] day_expense [2022-06-20] | quant [200] | category [1] | user [espe] | plus [False]
218 [20/Jun/2022 07:08:07] -- Changing money for espe, date: 2022/06/20, user: espe, quant: 200, plus: False
219 [20/Jun/2022 07:08:07] -- Old user found: espe
220 [20/Jun/2022 07:08:07] -- Expenses.update called with: quant=200, user=espe, plus > False
221 [11/Jul/2022 06:14:59] "GET /favicon.ico HTTP/1.1" 404 3111
222 [11/Jul/2022 06:15:10] [DREAM] user [espe] | curr day [2022-07-11] | sleep_hours [5] | illness []
223 [11/Jul/2022 06:15:39] [EXPENSE] day_expense [2022-07-11] | quant [150] | category [1] | user [espe] | plus [False]
224 [11/Jul/2022 06:15:39] -- Changing money for espe, date: 2022/07/11, user: espe, quant: 150, plus: False
225 [11/Jul/2022 06:15:39] -- Old user found: espe
226 [11/Jul/2022 06:15:39] -- Expenses.update called with: quant=150, user=espe, plus > False
227 [11/Jul/2022 06:19:48] -- Changing money for espe, date: 2022/07/11, user: espe, quant: 150, plus: True
228 [11/Jul/2022 06:19:48] -- Old user found: espe
229 [11/Jul/2022 06:19:48] -- Expenses.update called with: quant=150, user=espe, plus > True
230 [11/Jul/2022 06:19:48] [EXPENSE] - '2022-07-11' has been _updated_ | quant = 150 | plus = True
231 [11/Jul/2022 06:32:52] Internal Server Error: /dreams/dreams_create/

```

Figura 5.4: Trozo de ejemplo de *tfg.log*.

## 5.4 Núcleo o *core*

La parte donde empieza a crecer la aplicación web es el núcleo o, como se define en la app, *core*. Aquí están los principales estilos en el directorio *static* para CSS, JavaScript, Bootstrap, imágenes, fuentes, etc. También está la plantilla base para el resto de aplicaciones y las plantillas para la página principal o *home*.

## 5.5 Seguimiento

Aparte de ver los movimientos en la propia aplicación, se ha creado un fichero de *log*[1] nombrado como *tfg.log* ubicado en la carpeta del proyecto llamada *logs*. Un ejemplo del fichero *tfg.log* aparece en la Figura 5.4. Es una visión mucho más específica de la aplicación, ya que no solo se registran los movimientos que el programador ha visto necesarios, sino que también se registran los errores que surgen. En la siguiente sección se explicará cómo encontrar errores en el proyecto (si se usan editores como *Vim* o *Vi* es muy sencillo localizar lo que uno quiere buscar).

## 5.6 Errores

Los errores pueden verse de dos maneras en este proyecto:

- **Log.** Se ha mencionado antes. Mediante cualquier editor de texto abriendo el fichero *tfg.log*, se pueden encontrar los errores que han ido surgiendo mientras la aplicación ha estado en marcha. Por ejemplo, abriendo el fichero con *Vim* y poniendo */error*, aparecerán todas las coincidencias en el texto que tengan ese patrón. Después al dar a *Enter*, pulsando *n* se puede ir hacia delante o *N* para ir hacia atrás y va seleccionando cada *match*.
- **Correo.** Se ha creado en la aplicación un correo que te avisa de cualquier posible error, detallando el mismo, al igual que aparecería en *tfg.log*. Es una manera muy cómoda de detectar posibles *bugs* o debilidades del código, que el programador no ha tenido en cuenta. Una vez lanzada la aplicación, se puede ir resolviendo sobre la marcha posibles fallos que surjan; la Figura 5.5 muestra esto mismo (más adelante se explicará cómo se ha programado). Se puede elegir a qué direcciones de correo quieres que se envíe el error.

## 5.7 Interacción con la Base de Datos

Primero, antes de empezar a desarrollar la aplicación web, se cambió la Base de Datos que Django te pone por defecto, *SQLite3*, por *MySQL*; se explica cómo instalar *MySQL* en el ordenador en A.1. Cuando ya está *MySQL* instalado, hay que configurar el *setting.py* para cambiar *SQLite3* por *MySQL*, tal y como aparece en 5.2.

Ya teniendo todo configurado, se va a ver a continuación cómo la BBDD de la aplicación se modifica con las diferentes vistas creadas *create*, *edit* y *delete*. Como las cinco actividades siguen un mismo patrón de diseño, cogeremos de ejemplo *Expenses*.

Al crear un Gasto, se ve que queda guardado en la BBDD (ver Figura 5.6). Si se edita el campo de *note*, se comprueba que efectivamente cambia como puede verse en la Figura 5.7. Al eliminar en la aplicación el último gasto, ya no aparece en la BBDD (ver Figura 5.8 que la *id=134* desaparece).



Figura 5.5: Ejemplo de correo para controlar errores.

```

# Database
# https://docs.djangoproject.com/en/3.2/ref/settings/#databases

DATABASES = {
    'default': {
        # 'ENGINE': 'django.db.backends.sqlite3',
        'ENGINE': 'django.db.backends.mysql',
        # 'NAME': BASE_DIR / 'db.sqlite3',
        'NAME': 'webdb',
        'USER': 'webdb',
        'PASSWORD': 'passwordguay',
        'HOST': 'localhost',
        'PORT': '',
    }
}

```

Código 5.1: Cambiar SQLite3 por MySQL.

```
mysql> SELECT * FROM expenses_expenses ORDER BY id DESC LIMIT 1;
+----+-----+-----+-----+-----+-----+-----+-----+
| id | day_expense | quant | concept | category | note | plus | user_id |
+----+-----+-----+-----+-----+-----+-----+-----+
| 134 | 2022-07-26 | 150 | gasolina fin de semana a salamanca | 1 | prueba crear | 0 | 1 |
+----+-----+-----+-----+-----+-----+-----+-----+
```

Figura 5.6: Gasto guardado en BBDD.

```
mysql> SELECT * FROM expenses_expenses ORDER BY id DESC LIMIT 1;
+----+-----+-----+-----+-----+-----+-----+-----+
| id | day_expense | quant | concept | category | note | plus | user_id |
+----+-----+-----+-----+-----+-----+-----+-----+
| 134 | 2022-07-26 | 150 | | 1 | editar prueba | 0 | 1 |
+----+-----+-----+-----+-----+-----+-----+-----+
```

Figura 5.7: Gasto editado en BBDD.

## 5.8 Validadores

Sin tener en cuenta los tests y trasteando en la aplicación, ya teniendo todas las vistas creadas, hay que probar que los campos de los formularios son todos coherentes: fechas, cantidades, horas, etc. Para ello se usan los *validadores* de los formularios y dentro de las *plantillas* se plasman los formularios con los validadores ya configurados. En esta aplicación se han hecho varios validadores y algunos se verán más adelante. Un ejemplo sencillo que a todo el mundo le viene a la mente, es cuando nos registramos en cualquier página web con una dirección de correo y una contraseña; si la contraseña no sigue los requisitos preestablecidos, te muestra un mensaje avisando de que hay algo que no está bien. En la aplicación de Registro de esta app web esto también aparece (ver Figura 5.9).

Por ejemplo en la aplicación *sports*, el programador no quiere que en un mismo día haya varios deportes. Para ello ha tenido que crear una función/validador que controle esta acción. Tampoco se quiere que se guarde en la BBDD un registro de deporte de un día posterior al actual. Estas dos cosas se pueden agrupar en una misma función que trate la *fecha* de creación de un registro de deporte, tal y como aparece en 5.2. Luego en las plantillas, se tiene que plasmar el formulario con el control de errores que muestre al usuario que se ha equivocado. Esto último se pudo ver en la Figura 5.10 cuyo código es 5.3. Si no se han creado validadores, Django te muestra errores que suceden por defecto.

Este procedimiento se ha hecho con todas las aplicaciones, cada una con sus respectivos validadores.

```
mysql> SELECT * FROM expenses_expenses ORDER BY id DESC LIMIT 1;
+----+-----+-----+-----+-----+-----+-----+-----+
| id | day_expense | quant | concept | category | note | plus | user_id |
+----+-----+-----+-----+-----+-----+-----+-----+
| 133 | 2022-07-11 | 150 | | 1 | | 1 | 1 |
+----+-----+-----+-----+-----+-----+-----+-----+
```

Figura 5.8: Gasto eliminado de BBDD.

```

# ~~ validators

def clean_date_sport(self):
    date_sport = self.cleaned_data.get('date_sport')
    userS = self.cleaned_data.get('user')

    sports = Sports.objects.filter(
        date_sport=date_sport,
        user=userS)

    # use id
    if self.id:
        sports = sports.exclude(pk=self.id)

    # -- don't repeat days of the month
    if sports.count() > 0:
        raise forms.ValidationError(
            f'{date_sport} ya está creado. Modifique o borre el mismo.')

    if date_sport > datetime.date.today():
        raise forms.ValidationError(
            f'Fecha mayor que la actual: {datetime.date.today()}')

    return date_sport

```

Código 5.2: Validador para la app *sports*.

## 5.9 Testing

El *testing* es una de las herramientas que se pueden implementar en el desarrollo de los sistemas para mejorar programas y aplicaciones. Es importante, ya que permitirá, a través de la verificación en etapas tempranas del desarrollo de la arquitectura y el diseño de las aplicaciones, controlar, mejorar y avanzar sobre un proceso de construcción de un código más estable [15]. Django tiene formas de realizar testing<sup>5</sup> muy efectivas para su estructura de framework, ya sea para testear urls, vistas, modelos o formularios. Mediante el uso de *assert*, Django te permite hacer comprobaciones de igualdad, diferencia, comprobar si se usa un formulario de manera incorrecta y después programar el respectivo validador (en vez de hacerlo directamente estropeando la Base de Datos), jugar con el modelo, etc. Es muy útil si se quiere estar tranquilo de que no ocurra nada en las tablas. De otra manera, hay que estar pendiente y modificar las tablas, perdiendo tiempo. Hay dos tipos de programadores: los que realizan tests y los que no. Si eres de los segundos, al menos es recomendable tener conocimiento de **GIT**.

Un ejemplo de algún test que se ha hecho, puede verse en 5.4 donde se comprueba si *ExpensesForm* es un formulario válido. Luego para comprobar los test, hay que ejecutar en tu carpeta principal del proyecto: `python3 manage.py test` y te correrá todos los tests creados, cuyo resultado puede verse en la Figura 5.11.

<sup>5</sup><https://docs.djangoproject.com/en/4.0/topics/testing/overview/>

```

<div class="form-group" id="date_sport">
  {{ form.date_sport.label_tag }}
  {% if form.date_sport.errors %}
    <div class="alert alert-danger">
      <button type="button"
        class="close"
        data-dismiss="alert"
        aria-hidden="true">
        &times;
      </button>
      {{ form.date_sport.errors }}
    </div>
  {% endif %}
  {{ form.date_sport }}
</div>

```

Código 5.3: Parte de plantilla *sports\_create.html*: ventana emergente que muestra alerta de fallo.

The screenshot shows a registration form with the following elements:

- Header:** "Registro"
- Username Field:** Input field containing "pruebaValidador". Below it, a requirement: "Requerido. 150 caracteres como máximo. Únicamente letras, dígitos y @/./+/\_".
- Password Field:** Input field labeled "Contraseña".
- Validation Messages:**
  - Su contraseña no puede asemejarse tanto a su otra información personal.
  - Su contraseña debe contener al menos 8 caracteres.
  - Su contraseña no puede ser una clave utilizada comúnmente.
  - Su contraseña no puede ser completamente numérica.
  - Esta contraseña es demasiado corta. Debe contener al menos 8 caracteres.
  - Esta contraseña es demasiado común.
  - Esta contraseña es completamente numérica.
- Repeat Password Field:** Input field labeled "Repite contraseña". Below it, a requirement: "Para verificar, introduzca la misma contraseña anterior."
- Submit Button:** A green button labeled "Confirmar".

Figura 5.9: Formulario de registro avisando de error en contraseña.



(a) Deporte creado con fecha 01/03/2022.

**Fecha:**

- 2022-03-01 ya está creado. Modifique o borre el mismo. ×

01/03/2022

(b) Deporte creado con esa fecha.

**Fecha:**

- Fecha mayor que la actual: 2022-07-27. ×

13/08/2022

(c) Deporte con una fecha mayor a la actual.

Figura 5.10: Validador para un registro de Deporte.



```

+ [ 13:21:47 ] > python3 manage.py test
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
[TEST] DreamsForm valid
.[TEST] ExpensesForm valid
.[TEST] creado usuario y su perfil conectado
[TEST] Profile exists
.[TEST] dreams_create
.[TEST] dreams_detail
.[TEST] dreams_edit
.[TEST] signup
.
-----
Ran 7 tests in 0.073s

OK
Destroying test database for alias 'default'...

```

Figura 5.11: Resolución de los tests creados.

No se ha profundizado mucho en este tema, pero sería un tema para desarrollar únicamente aparte. Es algo muy extenso e importante dentro del desarrollo software.

```

class TestForms(TestCase):

    def test_expenses_form_no_valid_data(self):
        form = ExpensesForm(data={
            'day_expense': '2022-01-01',
            'quant': 7,
            'concept': 'test',
            'category': '0',
            'note': 'test',
            'plus': 'False'
        })

        self.assertFalse(form.is_valid())
        log.info("[TEST] ExpensesForm valid")

```

Código 5.4: Test para comprobar si *ExpensesForm* es válido.

## 5.10 Logging

El *logging* es una de las partes más importante del proyecto, desde el punto de vista del autor de esta app web. ¿Qué es y qué hace? Es un módulo que proporciona **Django** y permite tener registros o *logs* de todo lo que ocurre en cada acción que se realiza en la aplicación web. Es totalmente personalizable, es decir, se tendrán registros de aquello que el programador quiera tener. Tener un buen logging y seguimiento significa estar tranquilo mientras la aplicación está corriendo. Si se introducen más actividades un poco más elaboradas que

necesiten de la mano de tareas automáticas que completen la agenda y se introducen hilos para esas tareas para que la eficiencia sea mayor, se necesitaría de un logging bien hecho que permita ver qué está ocurriendo; esto, en el caso de ponernos en algo más serio como son los hilos, cuyo seguimiento es muy complicado si no se hace de otra forma. En esta aplicación, se ha realizado un logging para todas las acciones que se realicen en ella, profundizando en el comando realizado para los gastos/ingresos. Se puede ver un ejemplo de esto mismo en la Figura 5.12, donde se elimina un gasto de la lista. Es totalmente personalizable. Recordar también que vale para informar de errores que ocurran (para más información ir a la Sección 5.6).

Si se quiere saber cómo configurar el logging ir a A.6.

```
-- Changing money for espe, date: 2022/06/20, user: espe, quant: 200, plus: True
-- Old user found: espe
-- Expenses.update called with: quant=200, user=espe, plus > True
[EXPENSE] '2022-06-20' has been _deleted_ | quant = 200 | plus = False |
```

Figura 5.12: *tfg.log* muestra que se ha eliminado un gasto.

## 5.11 Despliegue con *ngrok*

Casi finalizando y rematando el proyecto, se tuvo la iniciativa de también desplegar la aplicación de una manera sencilla con *ngrok* <sup>6</sup>. A diferencia de los alojamientos web, donde subes el código y ejecutas la aplicación, te ahorras todo eso y simplemente con tener corriendo tu aplicación, *ngrok* se encarga de todo lo demás.

*ngrok*, es una herramienta que permite crear túneles que son seguros hacia un servidor local, es decir, a tu ordenador o donde tengas desplegada la aplicación. En el caso del autor del proyecto, la aplicación está desplegada y corriendo en una Raspberry Pi, que funciona las veinticuatro horas del día, ya que consume muy poco y puede hacer pruebas cuando quiera. Además, está creada una VPN <sup>7</sup> para poder hacer cambios si fueran necesarios, desde cualquier parte con conexión a Internet. El funcionamiento de *ngrok* se explica gráficamente en la Figura 5.13 [14]. En ella se puede ver cómo podría verse *in situ*, cambios en un despliegue mientras un cliente o colaborador ve los resultados que el propio desarrollador está viendo.

Para ver los pasos a seguir en la instalación y ejecución de *ngrok*, se puede visitar este enlace: <https://www.sdos.es/blog/ngrok-una-herramienta-con-la-que-hacer-publico-tu-localhost-de-forma-facil-y-rapida>.

---

<sup>6</sup><https://ngrok.com/>

<sup>7</sup>[https://es.wikipedia.org/wiki/Red\\_privada\\_virtual](https://es.wikipedia.org/wiki/Red_privada_virtual)

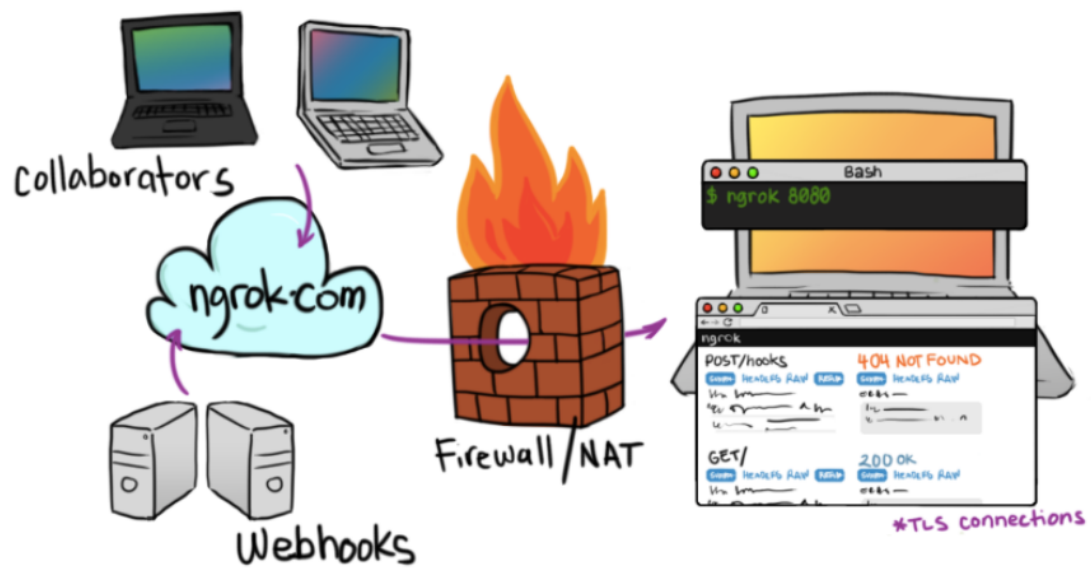


Figura 5.13: Funcionamiento de ngrok.



# Capítulo 6

## Conclusiones y trabajos futuros

### 6.1 Consecución de objetivos

Este proyecto ha llegado hasta el final con todos los objetivos planteados cumplidos. Permite ser una agenda de actividades diarias y se puede ver el seguimiento de un usuario, ya que todo está registrado y plasmado en la aplicación web. Gracias a las gráficas e índices, se puede ver el progreso y la cantidad de ítems en cada actividad y en cualquier momento se puede visualizar cualquier dato que se quiera. Como errores en la aplicación, han surgido muchos a lo largo de su creación, pero gracias a que desde el principio se tenía registro de fallos en los logs y aviso de todo lo que ocurría en el correo, se han ido solventando sin problema sobre la marcha. Sobre todo, ha ayudado mucho el uso de otros usuarios diferentes al desarrollador a usarla sin decirles cómo se usa; de esta manera no estaban guiados y encontraban fallos en su uso.

### 6.2 Aplicación de lo aprendido

Durante mi trayectoria académica, he aprendido muchos conocimientos, que por supuesto he aplicado a este proyecto. Detallo a continuación:

1. **Programación de Sistemas de Telecomunicación.** Me asentó las bases de la programación y el significado de cliente-servidor en HTTP.
2. **Sistemas Operativos.** Me ayudó a saber cómo de eficiente y rápida era mi aplicación dentro del sistema: qué rápida es, cuánto consume.

3. **Servicios y Aplicaciones Telemáticas.** Me ha enseñado la base del proyecto que es Django y su uso en todas sus partes: backend y frontend.
4. **Desarrollo y Aplicaciones Telemáticas.** Aportó parte de los conocimientos necesarios para el frontend y buenas prácticas para su desarrollo.
5. **Ingeniería de Sistemas de Información.** Me ha enseñado la importancia de tener una buena estructura en la Base de Datos y el poder manipularla a placer. También el uso de git, ya que tenía unos conocimientos básicos.
6. **Ingeniería en Sistemas Telemáticos.** Gracias a ella, he podido saber que un buen *logging* evita de muchos problemas a la hora de depurar mi código y efectivamente, así es.

### 6.3 Conocimientos adquiridos

A lo largo de todo este trabajo he podido aprender, debido a todos los fallos que han ido surgiendo y herramientas que he ido necesitando, varias cosas:

1. **Mejor manejo de JavaScript.** Aunque en la aplicación no se aprecie mucho, dentro del código en las *plantillas*, le he dado mucho mejor formato a mi frontend.
2. **ngrok.** Algo que para mí ha sido un descubrimiento muy útil y valioso, ya que con los alojamientos web como *pythonanywhere* o *heroku*, debido a que la cuota que elegía era gratuita, no me dejaba desplegar todo el contenido de mi código, ya que tenía un límite de capacidad o simplemente tenía problemas de configuración. Gracias a *ngrok*, solo con tener mi aplicación corriendo y ejecutando esta herramienta, varias personas han podido ver mi trabajo fuera de mi red.
3. **Perfeccionar uso de comandos en Django.** Es algo que veo muy útil y aunque ya lo sabía utilizar, he podido ampliar conocimientos de todo su potencial. Cualquier tarea automática se puede hacer con comandos.
4. **Mejor uso de Bootstrap.** He podido ver que se puede personalizar mucho más de lo que creía con ello, sobre todo para ver mi aplicación en un formato mucho más legible en un dispositivo móvil.
5. **Creación de VPN en Raspberry Pi.** Para poder monitorizar mi aplicación, con el log creado *tfg.log*, desde cualquier parte del mundo. Es muy sencillo siguiendo cualquier tutorial que se encuentre en Internet. En mi caso he usado *PiVPN* para la configuración y *Wireguard* para ejecutar el fichero de configuración en mi móvil. También se puede poner *Wireguard* en un portátil.

6. **Uso de Plotly.** Gracias a esta herramienta, se ha podido realizar la visualización en 2D y 3D de los datos; es lo más vistoso dentro del proyecto y a la vez lo más útil.

## 6.4 Prueba de usuarios externos

Al finalizar el proyecto, el autor ha conseguido que varias personas usen su aplicación. Esto se ha hecho con el objetivo de conocer hasta qué punto es ahora mismo provechoso su uso y para saber qué experiencias y pensamientos se han tenido.

Como puede verse en la Figura 6.1, en total han usado la aplicación nueve usuarios; todos ellos han sido avisados de poder usarla libremente sin explicarles cómo funciona.

The screenshot shows the Django Admin interface for user management. The sidebar on the left contains a menu with categories and sub-items, each with a '+ Añadir' button. The 'Usuarios' category is highlighted. The main content area is titled 'Seleccione usuario a modificar' and features a search bar with a magnifying glass icon and a 'Buscar' button. Below the search bar is an 'Acción:' dropdown menu and a 'Ir' button. The user list is displayed in a table with columns for 'NOMBRE DE USUARIO' and 'DIRECCIÓN DE CORREO ELECTRÓNICO'. The users listed are Bigsion, C, Pedro, Ruben90s, danikorko, esme, espe1, fer (with email fer@fer.com), and jgbarah. At the bottom of the list, it indicates '9 usuarios'.

Figura 6.1: Usuarios que han usado la aplicación.

De todos ellos, la persona que más ha usado la aplicación ha sido la usuaria espe1. Su

interpretación de uso y conclusión final, después de varios días de uso, ha sido la siguiente:

*"organizApp me ha parecido una aplicación de manejo sencillo. A nivel de uso es bastante intuitiva y no es necesario utilizar manuales, ni realizar procesos complicados. Es bastante fluída y rápida, tanto en el ordenador como en el móvil. Me resulta bastante útil sobre todo la parte de Gastos/Ingresos, para poder tener control de mis cuentas. En lo referente al control de Sueño, aparece una opción de registro de las horas dormidas diariamente y puede observarse si se tiene un descanso adecuado. La sección deportiva marca las rutinas realizadas en este área y con ello podemos visualizar y llevar el control de dichas prácticas observando el tiempo empleado, el peso perdido y la zona ejercitada. El apartado de gestión de ocio, posibilita registrar el tiempo que dedicamos a actividades como ir al cine, leer, salir con amigos u otras aplicaciones que sean de nuestro interés. Por último, esta app nos permite registrar el tiempo que dedicamos al estudio. Este apartado me resulta de gran utilidad para estudiantes, para que puedan llevar el control del tiempo empleado en cada asignatura. La parte gráfica es muy interesante; puedo ver de un solo vistazo todo lo que he hecho en la aplicación. Sobre todo lo que más me gusta son las gráficas 3D, porque se puede interactuar con ellas. Como punto negativo, he echado en falta más campos como opción en cada área, por ejemplo me hubiera gustado tener un listado más extenso de deportes o de actividades de ocio. Como mejora pondría el registro de actividades planteadas a realizar en el futuro cercano, por ejemplo, si dentro de las rutinas de una persona se encuentra ir al gimnasio los martes y jueves de 17:00 a 19:00 p.m., que la aplicación permita registrarlo con anterioridad y que aparezca en la aplicación que toca esa tarea. Pienso que sería mucho más eficiente la aplicación. En resumen, me ha resultado práctica y útil para ciertas cosas de mi día a día, otras actividades no tanto."*

Otros usuarios como C y Pedro, también han dado bastante uso a la aplicación. Por parte de Pedro, destacar que no sabía muy bien dónde encontrar lo que estaba realizando en la aplicación; tampoco se le dijo cómo usarla. Dijo lo mismo que espe1 en el tema de tener un listado de tareas antes de realizarlas y que la aplicación te avise de que tienes que hacerlas. C fue de todos el más crítico. Al desarrollador le dijo que necesitaba tener más información para usarla, los gráficos prefería tenerlos de diferentes colores para cada ítem y si el móvil se pone en vertical, los índices no se ven del todo claros. En cuanto al uso, admitió que era bastante práctico. Fue el usuario que encontró más fallos; ponía cosas mal adrede para ver hasta qué punto era consistente. En cuanto a los otros usuarios, no han tenido mucho tiempo de usar la aplicación, pero ya van introduciendo varios ítems en ella. Hay que recalcar que lleva una semana y media desplegada y no se ha informado a muchos más usuarios de su existencia.

La aplicación sigue desplegada y seguirá desplegada bastante tiempo. La url para su acceso es: <https://7b0d-87-218-20-34.eu.ngrok.io/>.

Resumiendo, gracias a su colaboración se han arreglado ciertos bugs que no se han tenido en cuenta y debido a que se tiene el correo que avisa cuando hay fallos, se han podido solventar sobre la marcha. Todas las opiniones obtenidas se tendrán en cuenta para la mejora y actualización que se haga de la aplicación.



## 6.5 Trabajos futuros

A continuación enumero varias cosas que se podrían hacer de cara a un futuro; ya se han comentado varias en la opiniones de los usuarios que han usado la aplicación:

1. **Lista de tareas para cumplir como objetivo.** Hacer un listado de tareas que se quieran cumplir y que la aplicación te avise que se tienen que hacer.
2. **Premiar si se cumplen las tareas.** Si se han cumplido los objetivos, se premiará al usuario con puntos, por ejemplo.
3. **Alertas si no se cumplen los objetivos.** Avisar al usuario de que no está cumpliendo su objetivo diario, para así motivarle a cumplirlo.
4. **Incidencias dentro de la aplicación.** Que aparezca un registro de incidencias que vayan surgiendo y que el administrador lo pueda visualizar en la aplicación en vez de por correo o mediante la consulta del log.
5. **Descarga de datos en CSV.** Tener la posibilidad de descargar datos que el usuario quiera tener en un fichero.
6. **Indicativo de porcentaje superado.** Tener un apartado que muestre de un simple vistazo, el porcentaje conseguido de la lista de tareas que se ha supuesto para hacer.
7. **Poner muchas más actividades.** Todas las actividades presentes en el proyecto, se han hecho basándose en el gusto del autor; se puede generalizar la aplicación mucho más y, por lo tanto, llegar al interés de muchas más personas. Por ejemplo, actividades como: medicinas que tomas, comidas, recetas, clases de universidad, trabajo, ocio, películas vistas, viajes realizados, conciertos, cumpleaños, idiomas aprendidos, etc.

## 6.6 Planificación

Para terminar, se mostrará la planificación temporal de este proyecto, que se ha tenido que implementar con la finalización de la carrera. Aparte de lo que se muestra, comentar que por las mañanas/tardes tengo un horario de ocho horas de trabajo desde hace muchos años antes.

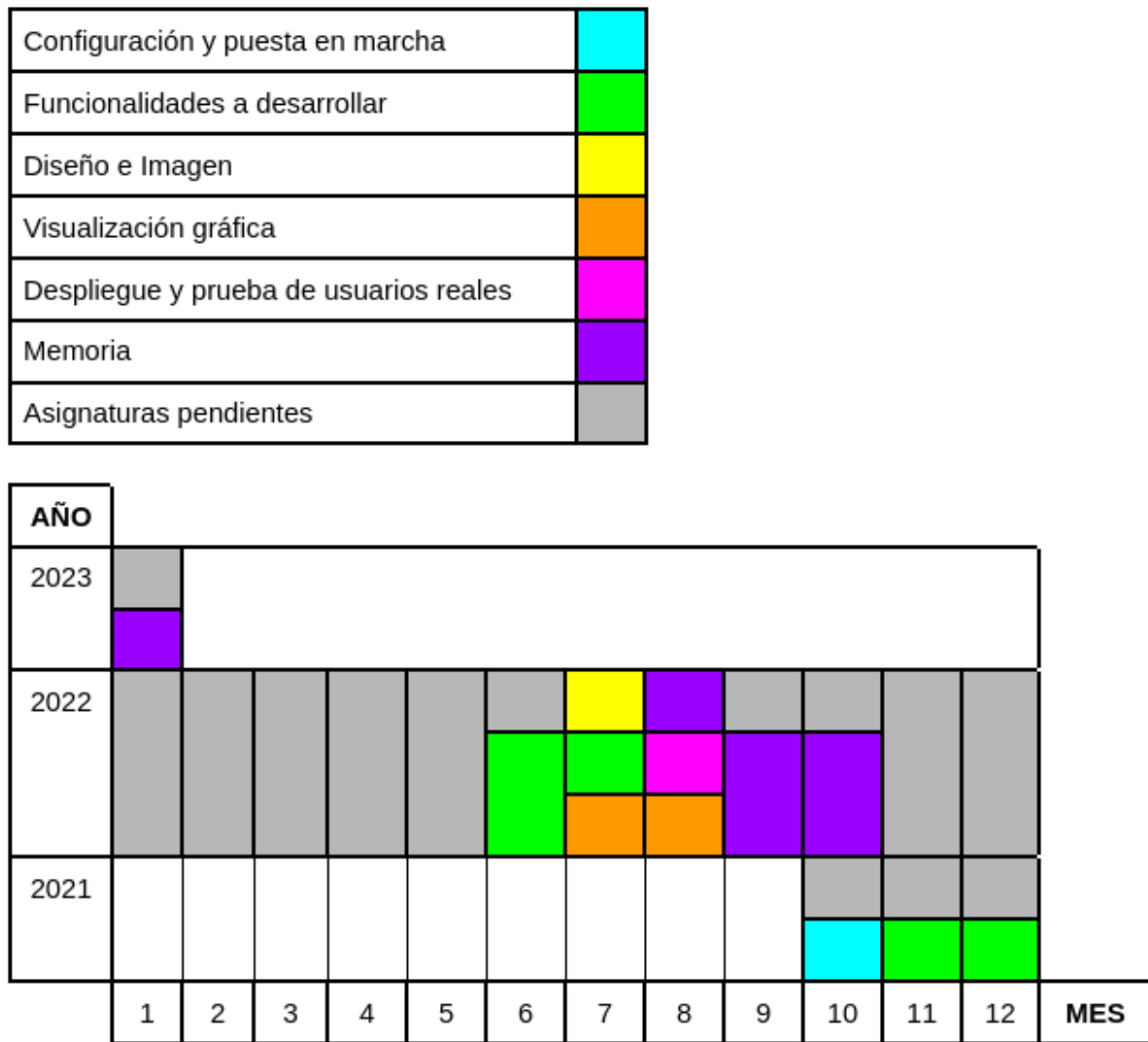


Figura 6.2: Planificación temporal del TFG.

# Apéndice A

## Manual de configuración

### A.1 Instalar MySQL

En tu ordenador, tienes que abrir cualquier terminal y seguir los pasos de [A.5](#). Esto es opcional y puede instalarse cualquier tipo de Base de Datos, como por ejemplo PostgreSQL.

### A.2 Configuración del logging

Para configurar el logging [\[5\]](#) con el cual podemos tener un seguimiento de nuestra app, hay que seguir los siguientes pasos:

1. Abrir settings.py.
2. Escribir lo que aparece en [A.6](#).

```

sudo apt update
sudo apt install mysql-server
configuramos mysql:

>> sudo mysql_secure_installation

|--- seguridad password: 0
|--- password: passwordguay
|--- a todo lo demás: y|Y

>> probar MariaDB

|--- vemos estado: sudo service mysql status > active (running) > OK
|--- sudo mysqladmin -p -u root version: si sale 'Server version, Copyright, UNIX
→ sockt ...' todo OK

>> ajustar autenticación y los privilegios del usuario

|--- sudo mysql
|--- SELECT user,authentication_string,plugin,host FROM mysql.user;
|--- ALTER USER 'root'@'localhost' IDENTIFIED WITH caching_sha2_password BY
→ 'passwordguay';
|--- FLUSH PRIVILEGES;
|--- SELECT user,authentication_string,plugin,host FROM mysql.user;
|--- exit

>> creamos usuario

|--- mysql -u root -p
|--- contraseña: passwordguay
|--- CREATE USER 'webdb'@'localhost' IDENTIFIED BY 'passwordguay';
|--- GRANT ALL PRIVILEGES ON *.* TO 'webdb'@'localhost' WITH GRANT OPTION;
|--- exit

comprobación:

|--- systemctl status mysql.service
|--- sudo mysqladmin -p -u root version

creamos BBDD:

|--- mysql -u webdb -p
|--- CREATE DATABASE webdb;
|--- SHOW DATABASES;

```

Código A.5: Creación de Base de Datos.

```

LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'verbose': {
            'format': "%(asctime)s %(levelname)s %(name)s:%(lineno)s" +
                "(message)s",
            'datefmt': "%d/%b/%Y %H:%M:%S"
        },
        'simple': {
            'format': "%(asctime)s %(message)s",
            'datefmt': "%d/%b/%Y %H:%M:%S"
        },
    },
    'handlers': {
        # include the default Django email handler for errors
        # this is what you'd get without configuring logging at all.
        'mail_admins': {
            'class': 'django.utils.log.AdminEmailHandler',
            'level': 'ERROR',
            # but the emails are plain text by default - HTML is nicer
            'include_html': True,
        },
        # log to a text file that can be rotated by logrotate
        'logfile': {
            'level': 'INFO',
            'class': 'logging.handlers.RotatingFileHandler',
            'filename': os.path.join(BASE_DIR, '../logs/tfg.log'),
            'formatter': 'simple',
        },
        'console': {
            'level': 'INFO',
            'class': 'logging.StreamHandler',
        },
    },
    'loggers': {
        # again, default Django configuration to email unhandled exceptions
        'django.request': {
            'handlers': ['mail_admins'],
            'level': 'ERROR',
            'propagate': True,
        },
        # might as well log any errors anywhere else in Django
        'django': {
            'handlers': ['console'],
            'level': 'WARN',
            'propagate': True,
        },
        # your own app - this assumes all your logger names start
        # with "myapp."
        'info_data': {
            'handlers': ['console', 'logfile'],
            'level': 'INFO',
            'propagate': False,
        },
        # your own app - this assumes all your logger names start
        # with "myapp."
        '': {
            'handlers': ['console', 'logfile', 'mail_admins'],
            'level': 'INFO', # or maybe INFO or DEBUG
            'propagate': True
        },
    },
}

```

Código A.6: Configuración del logging.



# Referencias

- [1] José Manuel Cabrera Alejandro Díaz. *¿Qué son los Logs y por qué deben interesarte?* DBi. URL: <https://dbibyhavas.io/es/blog/que-son-los-logs/>.
- [2] AULA21. *¿Qué es Python?* Centro de Formación Técnica para la Industria. URL: <https://www.cursosaula21.com/que-es-python/>.
- [3] MDN contributors. *HTTP*. Mdn Web Docs. URL: <https://developer.mozilla.org/es/docs/Web/HTTP>.
- [4] DjangoGirls. *¿Qué es Django?* DjangoGirls. URL: <https://tutorial.djangogirls.org/es/django/>.
- [5] Djangoproject. *Logging | Django*. Djangoproject. URL: <https://docs.djangoproject.com/en/4.0/topics/logging/>.
- [6] ECD. *Software de empresa para mejora de la productividad*. ECD. URL: <https://www.elconfidencialdigital.com/articulo/innovacion/software-empresa-mejora-productividad/20220624095217417205.html>.
- [7] Juan Diego Pérez Giménez. *Funcionamiento HTML5*. OpenWebinars. URL: <https://openwebinars.net/blog/que-es-html5/>.
- [8] Adrian Holovaty. *The Definitive Guide to Django: Web Development Done Right*. Springer, 2009.
- [9] Paloma Llanez. *Datanomics: Todos los datos personales que das sin darte cuenta y todo lo que las empresas hacen con ellos*. Deusto, 2019.
- [10] Miteris. *¿Qué es Javascript? Características y Librerías*. Miteris. URL: <https://www.miteris.com/blog/que-es-javascript-caracteristicas-librerias/>.
- [11] Joan Torrent-Sellens y Pilar Ficapal-Cusí. «TIC, cualificación, organización y productividad del trabajo: un análisis empírico sobre las nuevas fuentes de la eficiencia empresarial en Cataluña». En: *Investigaciones Regionales* 20 (2010), págs. 93-115.
- [12] Guido van Rossum. *Learning Python: Crash Course Tutorial*. Medina Univ Pr Intl, 2020.
- [13] Santander. *¿Dónde se utiliza Python?* Santander. URL: <https://www.becas-santander.com/es/blog/python-que-es.html>.

- [14] Alan Shreve. *ngrok - Introspected tunnels to localhost*.  
URL: <https://github.com/inconshreveable/ngrok>.
- [15] Nicolas Soto. *¿Qué es el testing de software?* Craft Code.  
URL: <https://craft-code.com/que-es-el-testing-de-software/>.