Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

# Aplicación web para la gestión de citas de una cadena de clínicas médicas

**Estudante:** Jorge Díaz Seoane
**Dirección:** Carlos Vázquez Regueiro

A Coruña, xuño de 2021.

*Me gustaría dedicar este trabajo de fin grado a mi familia por apoyarme siempre y a todos los profesores que me han enseñado en mi paso por la universidad, dándome los conocimientos que me han permitido su realización.*

**Agradecimientos**

Agradezco a todas las personas que me han ayudado a lo largo de mis años universitarios, tanto profesores como compañeros y amigos y a mi tutor por aceptar llevarme este trabajo y ayudarme en su realización.

## Resumen

El proyecto que se ha realizado en este Trabajo Fin de Grado (TFG) trata sobre la gestión de citas de una red de clínicas médicas. La finalidad es tener un sistema que gestione las citas de una manera lo más automáticamente posible, facilitando el trabajo tanto al administrativo como al médico como al paciente.

Se hace uso de *Telegram* para la comunicación con el paciente y Quick Response code (QR) para la confirmación de las citas. También se hace uso del calendario de *Google* para crear eventos con las citas. Una parte importante del proyecto es la reprogramación automática de las mismas.

A mayores de la gestión de citas, se implementa también una parte de gestión administrativa, más destinada a clínicas que están empezando y no tanto a clínicas con una estructura ya desarrollada. En éstas últimas se hace más hincapié en la integración de sus datos, permitiendo la importación y exportación de los mismos.

Para su realización se ha hecho uso del framework *Django* y diversos lenguajes de programación (*Python*, *JavaScript* y *HTML*).



## Abstract

The project that has been carried out in this Final Degree Project deals with the management of appointments of a network of medical clinics. The aim is to have a system that manages appointments as automatically as possible, avoiding work for the administrative staff, the doctor and the patient.

Telegram is used for communication with the patient and Quick Response code (QR) for appointment confirmation. Google calendar is also used to create events with appointments. An important part of the project is the automatic rescheduling of appointments.

In addition to the appointment management, an administrative management part is also implemented, more intended for clinics that are just starting and not so much for clinics with an already developed structure. For the latter, more emphasis is placed on the integration of their data, allowing the import and export of data.

For its realization, the framework Django and several programming languages (Python, JavaScript and HTML) have been used.

**Palabras clave:**

- Trabajo fin de grado
- Clínicas médicas
- Gestión citas médicas
- Django
- Final Degree Project
- Medical clinics
- Medical appointment management
- Django

# Índice general


**1 Introducción** **1**

1.1 Motivación . . . 1

1.1.1 Trabajos relacionados . . . 2

1.1.2 Reglamento General de Protección de Datos . . . 2

1.2 Objetivos . . . 3

1.3 Organización del proyecto . . . 4

**2 Herramientas y tecnologías** **5**

2.1 Lenguajes de programación . . . 5

2.1.1 Python . . . 5

2.1.2 JavaScript . . . 5

2.1.3 HTML . . . 6

2.2 Frameworks . . . 6

2.2.1 Django 3.1.4 . . . 6

2.2.2 Bootstrap 4 . . . 6

2.3 Bases de datos . . . 7

2.3.1 SQLite 3 . . . 7

2.4 Herramientas de comunicación y de gestión . . . 7

2.4.1 Ngrok . . . 7

2.4.2 Bot Telegram . . . 7

2.4.3 Git . . . 8

2.4.4 Docker . . . 8

2.5 Herramientas de edición . . . 9

2.5.1 Visual Studio Code . . . 9

2.5.2 LaTeX . . . 9

2.5.3 Draw.io . . . 9

**3 Metodología de desarrollo** **11**
3.1 Elección . . . 11
3.2 Scrum . . . 11
3.2.1 Explicación . . . 11
3.2.2 Aplicación en el proyecto . . . 12

**4 Análisis de requisitos** **13**
4.1 Requisitos funcionales . . . 13
4.1.1 Definición de los actores . . . 13
4.1.2 Casos de uso . . . 13
4.2 Requisitos no funcionales . . . 15

**5 Planificación y evaluación de costes** **17**
5.1 Planificación . . . 17
5.2 Seguimiento . . . 20
5.3 Evaluación de costes . . . 23
5.3.1 Costes *hardware* y *software* . . . 23
5.3.2 Costes personal . . . 23
5.3.3 Costes totales . . . 23

**6 Diseño** **25**
6.1 Arquitectura del sistema . . . 25
6.1.1 Patrón de diseño Django . . . 25
6.1.2 Arquitectura del sistema . . . 26
6.2 Diagrama de clases . . . 30
6.3 Modelo de datos . . . 32
6.3.1 Modelo entidad-relación . . . 32
6.3.2 Modelo relacional . . . 33
6.4 Seguridad y protección de datos . . . 34
6.4.1 Seguridad Django . . . 34
6.4.2 Aplicación del Reglamento General de Protección de Datos . . . 35
6.5 Módulos del sistema . . . 36
6.5.1 Horas citas . . . 36
6.5.2 Confirmación de citas . . . 36
6.5.3 Reprogramación de citas . . . 37
6.6 Canales de comunicación del sistema . . . 38
6.7 Generalización . . . 39
6.7.1 Inspección Técnica de Vehículos . . . 39

6.7.2 Peluquería . . . 39
6.7.3 Clínica veterinaria . . . 40

**7 Implementación 41**
7.1 Funcionalidades del sistema . . . 41
7.1.1 Generación lista intervalo horas citas . . . 41
7.1.2 Calendario de Google . . . 43
7.1.3 Importación datos . . . 43
7.1.4 Copia de seguridad . . . 44
7.1.5 Registro jornada laboral . . . 44
7.2 Implementación Django . . . 45
7.2.1 URLs . . . 45
7.2.2 Estructura de carpetas . . . 46
7.3 Configuraciones . . . 48
7.3.1 Ngrok . . . 48
7.3.2 Bot Telegram . . . 49
7.4 Docker . . . 50

**8 Pruebas 53**
8.1 Pruebas de caja negra . . . 53
8.2 Pruebas de caja blanca . . . 56
8.3 Pruebas de aceptación . . . 62
8.3.1 Fase 1: pruebas iniciales . . . 62
8.3.2 Fase 2 o Rectificación . . . 63

**9 Conclusiones y Líneas futuras 65**
9.1 Conclusiones . . . 65
9.2 Líneas futuras . . . 66

**A Casos de uso 69**

**B Manual de usuario 87**
B.1 Procedimientos de instalación, arranque y finalización de la Aplicación . . . 87
B.2 Descripción de Menús, Pantallas y Procesos . . . 87
B.2.1 Elementos comunes a toda la Aplicación . . . 87
B.2.2 Menús, pantallas y procesos de la Aplicación . . . 88
B.2.3 Menús y pantallas de ayuda . . . 102
B.3 Procedimientos Especiales . . . 102

**Lista de acrónimos** **103**

**Glosario** **105**

**Bibliografía** **107**

# Índice de figuras


5.1 Diagrama de Gantt. . . . . 19
5.2 Diagrama de Gantt final. . . . . 22

6.1 Ciclo solicitud-respuesta Django. . . . . 27
6.2 Configuración Django. . . . . 28
6.3 Estructura básica funcionamiento Django. . . . . 28
6.4 Diagrama de clases de la aplicación *ClínicasMédicas*. . . . . 31
6.5 Modelo Entidad-Relación de la aplicación *ClínicasMédicas*. . . . . 32
6.6 Modelo relacional de la aplicación *ClínicasMédicas*. . . . . 33

7.1 Estructura carpetas de la aplicación *ClínicasMédicas*. . . . . 47
7.2 Ngrok en funcionamiento. . . . . 48
7.3 Archivo settings.py. Allowed_hosts. . . . . 48
7.4 Archivo *requirements.txt*. . . . . 50
7.5 Archivos *Docker* de la aplicación *ClínicasMédicas*: *docker-compose.yml* y *Dockerfile*. . . . . 50

8.1 Resultado test *Django*. . . . . 56
8.2 Test *models.py* (Gestión Citas) de la aplicación *ClínicasMédicas*. . . . . 57
8.3 Test *views.py* (Gestion Administrativa) de la aplicación *ClínicasMédicas*. . . . . 59
8.4 Test *views.py* (Gestion Citas) de la aplicación *ClínicasMédicas*. . . . . 60
8.5 Cobertura test. . . . . 61
8.6 Funciones probadas y no probadas con un test. . . . . 61

B.1 Página inicial. . . . . 88
B.2 Página Inicio de sesión. . . . . 89
B.3 Página Registro. . . . . 90
B.4 Página inicial con sesión iniciada. . . . . 90

B.5 Página área personal. . . . . 91
B.6 Página pedir cita. . . . . 91
B.7 Página elegir hora cita. . . . . 91
B.8 Mensaje *Telegram* con la cita. . . . . 92
B.9 Correo electrónico con la cita. . . . . 92
B.10 Página modificar datos paciente. . . . . 92
B.11 Página preferencias. . . . . 93
B.12 Página citas paciente (rol paciente). . . . . 93
B.13 Cuadro diálogo eliminar cita. . . . . 93
B.14 Página inicial del médico. . . . . 94
B.15 Página agenda médico. . . . . 94
B.16 Página ver pacientes. . . . . 95
B.17 Página citas paciente (rol médico). . . . . 95
B.18 Página inicial administrativo. . . . . 96
B.19 Página formulario añadir médico. . . . . 97
B.20 Página tabla médicos. . . . . 98
B.21 Página agenda médico (vista desde el rol de administrativo). . . . . 98
B.22 Agenda médico inicial. . . . . 98
B.23 Agenda médico después del cambio. . . . . 99
B.24 Correo electrónico enviado al paciente. . . . . 99
B.25 Página listar clínicas. . . . . 99
B.26 Página importar/exportar. . . . . 100
B.27 Página importar datos. . . . . 100
B.28 Ejemplo archivo csv. . . . . 100
B.29 Correo electrónico con recordatorio de cita. . . . . 101
B.30 Página filtro sala de espera. . . . . 101
B.31 Página sala de espera. . . . . 101

# Índice de cuadros


4.1 Actores. . . . 13
4.2 Requisito no funcional: Seguridad. . . . 15
4.3 Requisito no funcional: Usabilidad. . . . 15
4.4 Requisito no funcional: Concurrencia. . . . 15
4.5 Requisito no funcional: Tiempo. . . . 16

5.1 Costes *hardware* y *software*. . . . 23
5.2 Costes trabajador. . . . 24
5.3 Costes totales. . . . 24

8.1 Caso de prueba: Añadir Paciente. . . . 53
8.2 Caso de prueba: Editar y eliminar Paciente. . . . 54
8.3 Caso de prueba: Pedir Cita. . . . 54
8.4 Caso de prueba: Editar y eliminar Cita. . . . 54
8.5 Caso de prueba: Confirmar Cita. . . . 54
8.6 Caso de prueba: Reprogramación Citas. . . . 54
8.7 Caso de prueba: Restaurar BBDD. . . . 55
8.8 Caso de prueba: Importar datos de archivo csv. . . . 55

A.1 Caso de uso: Iniciar Sesión. . . . 70
A.2 Caso de uso: Registrarse. . . . 70
A.3 Caso de uso: Contacto. . . . 70
A.4 Caso de uso: Ver cuadro médico. . . . 71
A.5 Caso de uso: Pedir cita. . . . 71
A.6 Caso de uso: Eliminar cita. . . . 72
A.7 Caso de uso: Editar cita. . . . 72
A.8 Caso de uso: Recuperar contraseña. . . . 73
A.9 Caso de uso: Modificar datos. . . . 73

A.10 Caso de uso: Ver agenda médica. . . . . 74
A.11 Caso de uso: Fichar entrada/salida jornada laboral. . . . . 74
A.12 Caso de uso: Llamar paciente. . . . . 74
A.13 Caso de uso: Paciente visto. . . . . 75
A.14 Caso de uso: Sala de espera. . . . . 75
A.15 Caso de uso: Añadir médico. . . . . 75
A.16 Caso de uso: Añadir administrativo. . . . . 76
A.17 Caso de uso: Añadir paciente. . . . . 76
A.18 Caso de uso: Añadir clínica. . . . . 77
A.19 Caso de uso: Editar médico. . . . . 77
A.20 Caso de uso: Editar clínica. . . . . 78
A.21 Caso de uso: Editar administrativo. . . . . 78
A.22 Caso de uso: Editar paciente. . . . . 79
A.23 Caso de uso: Eliminar médico. . . . . 79
A.24 Caso de uso: Eliminar paciente. . . . . 80
A.25 Caso de uso: Eliminar administrativo. . . . . 80
A.26 Caso de uso: Eliminar clínica. . . . . 81
A.27 Caso de uso: Confirmar asistencia cita paciente código QR. . . . . 81
A.28 Caso de uso: Enviar recordatorio citas. . . . . 82
A.29 Caso de uso: Restaurar Base de datos. . . . . 82
A.30 Caso de uso: Importar/exportar datos. . . . . 82
A.31 Caso de uso: Reprogramar citas. . . . . 83
A.32 Caso de uso: Correlacionar cita. . . . . 84
A.33 Caso de uso: Llamar paciente automático. . . . . 84
A.34 Caso de uso: Eliminar último paciente llamado. . . . . 85
A.35 Caso de uso: Preferencias. . . . . 85
A.36 Casos de uso disponibles para cada actor. . . . . 86

Capítulo 1

# Introducción

En este primer capítulo se verá la motivación, objetivos y organización del proyecto. Se pretende ver una idea inicial del trabajo, sin entrar en detalles que ya se verán más adelante.

## 1.1 Motivación

La motivación de este trabajo es crear una aplicación web que permita tanto su uso en clínicas que están empezando y precisan de una herramienta de gestión sencilla e intuitiva, como aquellas que quieren mejorar parte de su sistema, permitiendo la integración. La parte en la que más se enfoca esta aplicación es en la gestión de citas, permitiendo interacción entre los pacientes y administrativos de una manera rápida y que no resulte demasiado molesta a ninguna de las partes. Pero también permite la gestión administrativa para aquellas clínicas que no dispongan de ninguna herramienta. Se tuvo en cuenta la comodidad al hacer uso de la misma, minimizando el número de interacciones del usuario con la aplicación.

Permite el uso por parte de los pacientes para solicitar cita y la gestión de la misma, facilitando su uso, dado que solo es necesario darse de alta y no se requiere ningún mecanismo físico como una tarjeta o similares. En las clínicas más tradicionales, la cita solo es posible solicitarla presencialmente o telefónicamente pero no por el uso de una aplicación web, lo que facilita la solicitud de la misma y ahorra tiempo y molestias tanto al paciente como al administrativo.

Como usuarios de servicios donde se requieren citas, como el médico o la Inspección Técnica de Vehículos (ITV), se vio que era posible mejorar ciertos aspectos de las mismas e incorporar ciertos elementos, tales como el uso de Quick Response code (QR) para confirmar la asistencia o la incorporación de *Telegram* para enviar mensajes al usuario.

Si bien este proyecto se centra en la gestión de citas médicas, sería posible adaptarlo, con algunos cambios, a otros servicios como el de la ITV, veterinarios, etc. En el caso de la ITV, los

médicos serían los trabajadores de las diferentes líneas de las que disponen (la especialidad del médico pasaría a ser el tipo de vehículo que cada línea revisa) y los pacientes serían los usuarios que, con sus coches, acuden a la línea.

Debido al alcance del proyecto, y al equipo de desarrollo del mismo (compuesto únicamente por una persona), no se piensa en abarcar todo lo que influye en la gestión de una clínica médica pero sí dar soluciones a ciertas partes de la misma, modernizándolas.

### 1.1.1 Trabajos relacionados

Debido a que el software de gestión de citas del que hacen uso las clínicas es privativo o es hecho en exclusiva para ellos, no se ha podido comparar este proyecto con otros trabajos relacionados. Por este motivo, como usuarios del *Sergas (Servicio Galego de Saúde)*, se ha comparado con lo que ellos utilizan.

Por una parte, el *Sergas* no dispone de pantallas para llamar a los pacientes en todos los centros de salud. En algunos el médico sigue llamando a los pacientes de manera oral. Tampoco incluye el uso de avisos por correo electrónico ni incorpora un *bot* de *Telegram* o similares. Sin embargo, sí hace uso de mensajería móvil *SMS* (sistema que este proyecto no utiliza). A la hora de guardar citas en el calendario permite el uso de cualquier calendario (mientras que este proyecto solo hace uso del de *Google*).

El *Sergas* diferencia dos tipos de citas (consulta y receta) para el médico de cabecera, con intervalos de duración distintos. Mientras que este proyecto solo tiene un tipo de cita con su correspondiente intervalo.

### 1.1.2 Reglamento General de Protección de Datos

En este apartado se verá lo relativo a la protección de datos siguiendo el Reglamento General de Protección de Datos (RGPD). [1]

Lo primero a tener en cuenta es la diferencia que el RGPD hace entre datos personales y datos sensibles [2]. Los primeros (Artículo 4, punto 1) se refieren a datos que permiten la identificación directa o indirecta de una persona, tales como el nombre, los apellidos o la dirección. Los segundos (Artículo 9) hacen referencia a datos que por su naturaleza o relación con las libertades y derechos fundamentales de las personas requieren de una especial protección, tales como ideas políticas, afiliación sindical o datos sanitarios. En este proyecto no se tratan datos sensibles, solo datos personales.

## 1.2 Objetivos

La finalidad de este proyecto es, como se ha visto, tener una aplicación para gestionar las citas, para ello definiremos una serie de objetivos:

- Añadir, editar, listar y borrar clínicas, médicos, pacientes y administrativos.
- Solicitar, listar, editar y borrar una cita. Cuando se solicita una cita, se envía un correo electrónico al paciente. También se le envía un enlace personalizado para añadir la cita, mediante la creación de un evento, en el calendario de *Google*.
- Cuando el paciente acuda a la clínica, tener la posibilidad de confirmar la cita mediante un administrativo o mediante el móvil y un código *QR*.
- Si un día un médico no puede atender a sus pacientes, realizar la reprogramación de citas de manera automática.
- Posibilidad de interactuar, automáticamente, con el paciente mediante la aplicación de mensajería *Telegram*.
- Añadir datos desde un archivo Comma-Separated Values (CSV), evitando tener que introducirlos de uno en uno.
- Pantalla en las salas de espera para avisar al paciente.
- Cumplir con el RGPD y la seguridad de la aplicación.

## 1.3 Organización del proyecto

Este proyecto se organiza en 9 capítulos, 2 anexos, 1 lista de acrónimos, 1 glosario y la bibliografía.

- Introducción: capítulo actual donde se ven, por encima, los aspectos principales del proyecto, sin profundizar en ellos, para que el lector comprenda con un vistazo general la finalidad del proyecto.
- Herramientas y tecnologías: capítulo que trata sobre las herramientas y tecnologías empleadas para realizar la aplicación sobre la que se construye después esta memoria.
- Metodología: capítulo donde se ve la metodología empleada en el desarrollo de la aplicación, comentando sus ventajas y desventajas.
- Análisis de requisitos: capítulo dedicado a los requisitos funcionales y no funcionales que la aplicación debe seguir.
- Planificación y evaluación de costes: capítulo dedicado a ver la planificación del proyecto y los costes asociados al mismo.
- Diseño: capítulo con los detalles de la aplicación, permite la realización de la misma. Se hace uso de los requisitos obtenidos en el capítulo anterior.
- Implementación: capítulo para mostrar como está implementada la aplicación. Se hace uso del diseño creado en el capítulo anterior.
- Pruebas: capítulo dedicado a las pruebas realizadas a la aplicación, para ver su correcto funcionamiento.
- Conclusiones y líneas futuras: capítulo final del proyecto, donde se comentan las conclusiones sacadas y se ven las posibles líneas de trabajo futuras que pueden surgir usando este proyecto de base.

Capítulo 2

# Herramientas y tecnologías

En este capítulo se verán las herramientas y tecnologías empleadas en el proyecto *ClínicasMédicas*.

Se han agrupado en cuatro grandes bloques: los lenguajes de programación, los frameworks empleados, las bases de datos, las herramientas de comunicaciones y gestión y los editores.

## 2.1 Lenguajes de programación

### 2.1.1 Python

*Python* es un lenguaje de programación multiplataforma, interpretado, de tipado dinámico y orientado a objetos. Fácil de aprender, potente y permite un desarrollo rápido de aplicaciones. [3]

*Guido van Rossum* lo publicó por primera vez el 20 de febrero del año 1991, en *alt.sources*, siendo la versión publicada la 0.9.0. El nombre hace referencia al grupo *Monty Python*, del que el autor era aficionado. La versión 1.0 llegaría en enero de 1994. [4]

En este proyecto se hace uso de *Python* porque se eligió usar *Django* y es el lenguaje de programación que éste usa. Concretamente se hace uso de la versión 3.8.5. No fue necesaria su instalación porque ya viene por defecto en el Sistema Operativo (S.O.) (*Ubuntu 20.04*) utilizado para el desarrollo.

### 2.1.2 JavaScript

*JavaScript* es un lenguaje de programación ligero que es muy usado para hacer script en programación web. Fue creado por *Brendan Eich*. [5]

Se suele usar *JavaScript* para realizar ciertas acciones que requieren un nivel de complejidad superior al habitual. Permite que estas acciones las realice la parte cliente desde su

navegador, aligerando la carga del servidor. [6]

En este proyecto se hace un uso mínimo de este lenguaje para modelar ciertos aspectos de algunas vistas. Más concretamente, se usa para insertar un mapa de *Google Maps* y para recargar de manera automática la vista de la sala de espera (cada 10 segundos).

### 2.1.3 HTML

*HTML* son las siglas de *Hyper Text Markup Language*, es decir **Lenguaje de marcado de hipertexto**, que es el lenguaje de marcado estándar para la creación de páginas web. Describe la estructura de la página web a través de una serie de elementos. Fue creado por *Tim Berners-Lee* en el Conseil Européen pour la Recherche Nucléaire (CERN). [7]

En este proyecto se usa *HTML* para crear las diferentes vistas con las que interactúa el usuario.

## 2.2 Frameworks

### 2.2.1 Django 3.1.4

*Django* en un framework web de alto nivel desarrollado en *Python* que permite un rápido desarrollo de aplicaciones web. Es gratuito y de código abierto [8]. Sus creadores son: *Adrian Holovaty*, *Simon Willison*, *Jacob Kaplan-Moss* y *Wilson Miner*. El nombre hace alusión al guitarrista *Django Reinhardt*. [9]

*Django* hace uso del patrón MTV (*Model-Template-View*), que es una variante del MVC (*Model-View-Controller*). Esto se verá en detalle en el apartado de patrones 6.1.1.

Se ha elegido tanto por la facilidad a la hora de desarrollar las distintas partes de la aplicación, como porque permite al desarrollador abstenerse de aplicar las secuencias Structured Query Language (SQL) para generar las tablas de la Bases de Datos (BBDD) y trabajar con los distintos datos. También se encarga del apartado de la seguridad y la concurrencia. Finalmente porque hace uso de *Python*, que es un lenguaje que simplifica mucho la programación.

### 2.2.2 Bootstrap 4

*Boostrap* es un framework de código abierto para la personalización de los sitios web mediante el uso de *Cascading Style Sheets (CSS)*, *HTML* y *JavaScript*. Fue creado por *Mark Otto* y *Jacob Thornton*, trabajadores de *Twitter*, en el año 2011. Nació como solución interna para su empresa, con el nombre de *Blueprint*, pero con el tiempo fue liberado como proyecto de código abierto y pasó a llamarse *Bootstrap*. [10] [11]

En este proyecto se usa para darle diseño a las vistas creadas mediante *HTML*. Se decidió hacer uso de él y no aplicar CSS directamente porque es más sencillo de aplicar y no requiere

un esfuerzo extra por parte del desarrollador.

## 2.3 Bases de datos

### 2.3.1 SQLite 3

*SQLite* es un sistema gestor de bases de datos, escrito como una biblioteca en lenguaje C. El código es de dominio público y es gratuito. Fue creado por *Richard Hipp* y lanzado en agosto del año 2000. [12] [13]

Para implementar la BBDD en este proyecto se hace uso de un archivo denominado *db.sqlite3* que crea *Django*. Además, *Django* permite crear las tablas mediante clases *Python* (en el archivo *models.py*), evitando las secuencias SQL. También el añadir, modificar y eliminar datos se hace desde código *Python*, lo que simplifica el trabajo al desarrollador.

En este proyecto se usa porque es el que viene por defecto en *Django* y no se vio la necesidad de utilizar uno distinto. Como *Django* se encarga de realizar las tareas sobre esta BBDD, no se volverá a ver nada más de ella en esta memoria.

## 2.4 Herramientas de comunicación y de gestión

En este apartado se comentan las herramientas empleadas para hacer accesible desde internet nuestra aplicación que se ejecuta en local. También se comenta la aplicación de mensajería, el gestor de versiones y el de contenedores.

### 2.4.1 Ngrok

*Ngrok* [14] es una aplicación que permite crear túneles para poder acceder desde fuera a servicios que se tienen en local. En este caso se hace uso de ella para la parte de confirmar la cita mediante QR desde el móvil o desde el *bot* de *Telegram*. En un proyecto real se tendría una dirección IP única que permitiría realizar esta operación sin necesidad de crear un túnel pero como no se dispone de dicha IP, se usará esta aplicación para probar su funcionamiento. La configuración se verá en el capítulo 7.

### 2.4.2 Bot Telegram

*Telegram* es un servicio de mensajería gratuito que permite la creación de *bots*. Los *bots* son programas automáticos que permiten realizar ciertas tareas mediante el envío y recepción de mensajes.

En este proyecto se usa para la comunicación con el paciente. En el capítulo 7 se verá la configuración del mismo.

### 2.4.3 Git

*Git* es un software de control de versiones diseñado por *Linus Torvalds*. Control de versiones es la gestión de los diferentes cambios que se realizan sobre algún producto (en este caso, sobre la aplicación desarrollada).

Inicialmente se utilizó el repositorio de *git* que proporciona la facultad pero, posteriormente, una vez finalizado el proyecto, se subió a un repositorio público de la plataforma *Gitlab* para permitir el acceso al código fuente a cualquier interesado.

El enlace de *Gitlab* es el siguiente: pulsar aquí

### 2.4.4 Docker

*Docker* es un software que automatiza el despliegue de aplicaciones mediante el uso de contenedores, de esta manera la aplicación se puede utilizar independientemente del software que tenga el equipo. Por debajo utiliza funcionalidades proporcionadas por el *kernel* de *Linux*. Inicialmente empezó como un desarrollo por parte de *Solomon Hykes* en un proyecto interno de *dotCloud*. Posteriormente tendría ayuda de otros ingenieros. En marzo de 2013 sería liberado como código abierto. [15]

La ventaja de usa un contenedor para el servidor respecto al uso de un máquina virtual es que requiere de menos recursos. En una máquina virtual, ésta tiene en exclusiva una parte de los recursos del sistema físico. Sin embargo, un contenedor corre como una aplicación más del sistema, compartiendo los recursos con el resto. Un ejemplo sería con el uso de la memoria *RAM*, una máquina virtual usa toda la *RAM* que se le asigne al crearla (2GB por ejemplo), mientras que un contenedor usaría la que necesite, del total del sistema físico, en cada momento.

En este proyecto se usa *Docker* para el despliegue de la aplicación, de esta manera cualquier interesado podrá ver y probar la aplicación sin necesidad de replicar el equipo usado para el desarrollo. En el apartado 7.4 se verá la configuración concreta para este proyecto.

## 2.5 Herramientas de edición

En este apartado se han agrupado los editores de ficheros, la herramienta de composición de texto y la creación de diagramas.

### 2.5.1 Visual Studio Code

*Visual Studio Code* es un editor de código fuente desarrollado por *Microsoft*. Fue anunciado en abril de 2015 y lanzado en abril de 2016. [16]

Es el editor de código que se ha utilizado para el desarrollo del proyecto. No hay ningún motivo en especial para usar éste editor frente a otras alternativas como *Geany*, *Sublime Text*, etc.

### 2.5.2 LaTeX

LaTeX es un sistema para la composición de textos creado por *Leslie Lamport* como un gran conjunto de macros de TeX. TeX es un sistema de tipografía creado por *Donald E. Knuth*. [17] [18] [19]

En este proyecto se emplea LaTeX siguiendo el modelo de TFG da Facultade de Informática da Universidade da Coruña. [20]

### 2.5.3 Draw.io

draw.io es una página web que permite la creación de diagramas. En este proyecto se hace uso de ella para crear los diagramas de clase, el modelo entidad-relación y el modelo relacional.

Es propiedad de *Seibert/Media*, una consultora alemana especializada en *Atlassian* y *Google Cloud*. [21]

Capítulo 3

# Metodología de desarrollo

En este capítulo se verá la metodología empleada para el desarrollo de la aplicación, explicando cómo se ha seleccionado y cómo se ha aplicado en este proyecto.

## 3.1 Elección

Debido a la naturaleza individual de un TFG y el poco tiempo del que se dispone para la realización del mismo, se ha de hacer uso de una metodología que permita completarlo en el tiempo requerido. Por ello, se ha elegido hacer uso de una metodología ágil, que, como su nombre indica, permite desarrollar una aplicación de manera ágil y rápida, mediante procesos incrementales. Se descarta el uso del desarrollo en cascada por la poca flexibilidad que ofrece, al no permitir volver hacia atrás en las fases de desarrollo [22]. También se descarta el uso del desarrollo en espiral, más pensado para desarrollos a medio plazo, porque no se precisa realizar análisis de riesgos [23]. Tampoco es necesario el uso de prototipos, así que se descarta el uso de metodologías basadas en prototipos.

Existen múltiples metodologías ágiles, tales como: *Scrum*, *Kanban*, *eXtreme Programming* (o XP), Desarrollo Ligero (o *Lean*), *Scrumban* (que combina *Scrum* y *Kanban*), etc. [24]

Para este proyecto se ha elegido utilizar *Scrum*.

## 3.2 Scrum

En este apartado se verá la explicación de *Scrum* y como se aplica en este proyecto.

### 3.2.1 Explicación

*Scrum* [25] es una de las metodologías ágiles más conocidas y utilizadas, consiste en una serie de roles y prácticas para crear entregas incrementales de un producto. Los roles que se tienen son los siguientes:

- Dueño del producto: persona encargada de transmitir los requerimientos de la aplicación. En este caso, este rol es compartido entre el autor y el director del proyecto.
- Persona al mando: líder del proyecto, también se encarga de cumplir con los plazos de entrega. Este rol es desempeñado por el autor del proyecto.
- Miembros del equipo de desarrollo: personas encargadas de desarrollar la aplicación. En este caso, solo hay una persona, que es el autor de este proyecto, que desempeña varios roles: analista, programador y probador de *software*.

Proceso *Scrum*:

- Análisis de requisitos: en esta fase se eligen los requisitos que tendrá el sistema. En este caso, los requisitos salieron de las diversas reuniones entre el autor y el director del proyecto. 4
- Planificación: fase para planificar las tareas y tiempo de entrega. En este caso, el que planifica es el autor. 5
- Revisión con cliente: fase para revisar si se van cumpliendo los objetivos acordados. Esto también se ve en las reuniones entre autor y director.

En *Scrum* hay un elemento denominado *Sprint*, que es el proceso de desarrollo a realizar en cada entrega incremental (va entre planificación y revisión con el cliente). Con los requisitos del cliente, se divide el trabajo en una serie de *sprints*, al final de cada *sprint* hay un producto utilizable, que sirve de base para el siguiente *sprint*. Esto se realiza hasta que se termina el producto (la aplicación de este proyecto en este caso).

### 3.2.2 Aplicación en el proyecto

La idea detrás de *Scrum* es tener siempre productos utilizables, partir de algo que realice pocas funciones e ir incrementando sus funcionalidades. En este proyecto se empezó haciendo la parte de la gestión de usuarios. Se hizo así para poder probar luego el resto de funcionalidades. Una vez terminada esta parte, se procedió a realizar la gestión administrativa. En esta parte se hizo todo lo que no implicase la gestión de citas ni la interacción con las mismas. Finalmente, se hizo la gestión de citas.

Con esto finalizado, se procedieron a tener reuniones entre autor y director. A raíz de estas reuniones fueron saliendo más requisitos y casos de uso, por lo que el proyecto fue creciendo incrementalmente, siguiendo la filosofía de esta metodología. En el capítulo 5 se verá que tareas se realizaron en cada *sprint*.

Capítulo 4

# Análisis de requisitos

En este capítulo se verá el análisis de requisitos del sistema. Es una parte importante del proyecto debido a que influye directamente en el diseño e implementación. Por una parte se hablará de los requisitos funcionales y por otra de los no funcionales.

## 4.1 Requisitos funcionales

Los requisitos funcionales son las funciones que va a cumplir el sistema. Primero se verá la definición de los actores del sistema y después los casos de uso del mismo.

### 4.1.1 Definición de los actores

En esta aplicación se decidieron crear tres actores distintos, cada uno con unas funcionalidades concretas. Por un lado están los médicos, que son las personas que ofrecen los servicios de las clínicas médicas. Por otro están los pacientes, que son los consumidores de estos servicios. Por último están los administrativos, que conectan a médicos y pacientes. En la tabla 4.1 se puede ver una descripción de cada actor.

### 4.1.2 Casos de uso

Se pueden ver todos los casos de uso en el apéndice A.

| Actor | Nombre | Descripción |
|---|---|---|
| ACT-01 | Administrativo | Persona encargada de la gestión de la clínica. |
| ACT-02 | Médico | Persona encargada de atender a los pacientes. |
| ACT-03 | Paciente | Persona que acude a la clínica para ser atendida. |

Cuadro 4.1: Actores.

Se puede dividir este proyecto en varios bloques relacionados entre sí. Los casos de uso se asocian a uno de estos bloques. Estos bloques no tienen relación con las aplicaciones creadas en *Django* (las cuales se verán en detalle en otros capítulos). Los bloques pueden ser los siguientes:

- **Gestión de usuarios**

  En este bloque va todo lo relacionado con los usuarios, es decir, el registro A.2, el inicio de sesión A.1, cerrar sesión y recuperar la contraseña A.8. También la parte de modificar datos de usuario A.9 y las preferencias en los canales de comunicación A.35.

- **Gestión administrativa**

  En este bloque va lo relacionado con la parte administrativa de la clínica, esto es, dar de alta, modificar o eliminar médicos, pacientes, administrativos y clínicas (casos de uso del A.15 al A.26). También se incluye aquí fichar la entrada y salida de la jornada laboral A.11, importar y exportar datos A.30, las copias de seguridad A.29, el formulario de contacto A.3 y el cuadro médico A.4.

- **Gestión de citas**

  En este último bloque va lo relacionado a las citas, esto es, dar de alta A.5, modificar A.7, listar A.10 o eliminar A.6 citas, forzar cita (se explica en la misma tabla que pedir cita: A.5).

  A mayores va también lo relacionado con la cita una vez que el paciente va a acudir a la misma: confirmar asistencia A.27 (se puede hacer a través de un administrativo, por QR o por *Telegram*), llamar al paciente A.12, confirmar paciente visto A.13, llamar paciente automático A.33, eliminar último paciente llamado A.34, reprogramar citas A.31, correlacionar citas A.32 y enviar recordatorio de cita A.28.

  En esta parte también es donde el administrativo habilita el QR (no tiene un caso de uso asociado porque de normal iría impreso en unas hojas que estarían repartidas por la clínica) y las pantallas de la sala de espera A.14.

| RNF-01 | Seguridad |
|---|---|
| Descripción | El sistema deberá ser seguro, diferenciando a usuarios registrados de los que no lo estén. Además, cada actor solo podrá acceder a los datos que esté autorizado a ver/tratar. |

Cuadro 4.2: Requisito no funcional: Seguridad.

| RNF-02 | Usabilidad |
|---|---|
| Descripción | El sistema deberá presentar una interfaz de usuario lo más intuitiva posible, disminuyendo el número de interacciones entre el usuario y el propio sistema. También deberá permitir su uso en diferentes dispositivos, adaptando su diseño a los mismos. |

Cuadro 4.3: Requisito no funcional: Usabilidad.

## 4.2 Requisitos no funcionales

Un requisito no funcional es un parámetro para la operativa del sistema, en lugar de una función del mismo. Es decir, indican características de funcionamiento del sistema.

En este proyecto se han definido cuatro requisitos no funcionales, que se entiende que son los más importantes. Uno de ellos es la seguridad (ver 4.2), por un lado para proteger los datos de cada usuario y que un actor no pueda acceder a las funcionalidades de otro y por otro lado para cumplir con el RGPD.

Otro es la usabilidad (ver 4.3), es importante que la aplicación sea fácil de usar y que se adapte a cada dispositivo, dado que no es lo mismo ver una página web desde un ordenador que hacerlo desde el móvil, por eso es importante que cada dispositivo tenga los diseños adaptados a su tamaño de pantalla.

La concurrencia (ver 4.4) es también importante por tratarse de una aplicación web a la que van a acceder múltiples usuarios. No se puede permitir que un usuario pueda llegar a interferir en las transacciones de otro usuario.

Por último, también hay que tener presente el tiempo (ver 4.5). Todos los proyectos vienen marcados por un tiempo de entrega, que se negocia con el cliente. En este caso, la entrega viene marcada por la fecha límite para entregar el TFG. Por ello es necesario tener una buena planificación y adaptarse a ella, para cumplir con el plazo.

| RNF-03 | Concurrencia |
|---|---|
| Descripción | El sistema deberá permitir el acceso y modificación simultáneo a los datos, permitiendo que varios usuarios puedan trabajar sobre los mismos datos sin que ocurran errores ni pérdidas. |

Cuadro 4.4: Requisito no funcional: Concurrencia.

| RNF-04 | Tiempo |
|---|---|
| Descripción | El sistema deberá ser desarrollado en el tiempo disponible, el cual viene marcado por la fecha límite de entrega del TFG. |

Cuadro 4.5: Requisito no funcional: Tiempo.

Capítulo 5

# Planificación y evaluación de costes

En este capítulo se verá la planificación, el seguimiento y los costes del proyecto.

## 5.1 Planificación

Para la planificación hay que tener en cuenta que solo se dispone de un trabajador (el autor de este proyecto en este caso), así que se distribuyen las 300 horas de duración de un TFG entre las diferentes fases con un solo trabajador, que, como se ha visto en el capítulo de metodología 3, desempeña tres roles: analista, programador y *tester* (persona que realiza las pruebas de la aplicación). A mayores se ha puesto otro recurso (al que se ha denominado Desarrollador) que realiza las tareas del proyecto que no tienen que ver directamente con la aplicación. En concreto, el aprendizaje y lectura de documentación, la redacción de la memoria y la realización de la presentación.

El proyecto empezó de manera autónoma por parte del alumno, el cual realizó una serie de *sprints* por su cuenta antes de tener la primera reunión con el director. En la planificación inicial todavía no se sabía el número de reuniones ni los cambios surgidos de las mismas, por ese motivo se sobreestimó el tiempo para adaptarlo a esas 300 horas, siendo conscientes de que no es el tiempo real.

El inicio del proyecto fue el 7 de diciembre de 2020 y la finalización se marcó para el día 22 de junio de 2021, siendo éste el día anterior a la fecha límite de entrega.

Los *sprints* realizados en esta planificación inicial fueron los siguientes:

- Gestión de usuarios. En este *sprint* se realizó todo lo que está relacionado con los usuarios: el registro, el inicio y fin de sesión, la recuperación de la contraseña y la modificación de los datos.

- Gestión Administrativa. En este *sprint* se hizo la parte de añadir, listar, borrar y editar pacientes, médicos, administrativos y clínicas. También la copia de seguridad, el registro

de la jornada laboral, el cuadro médico y el formulario de contacto.

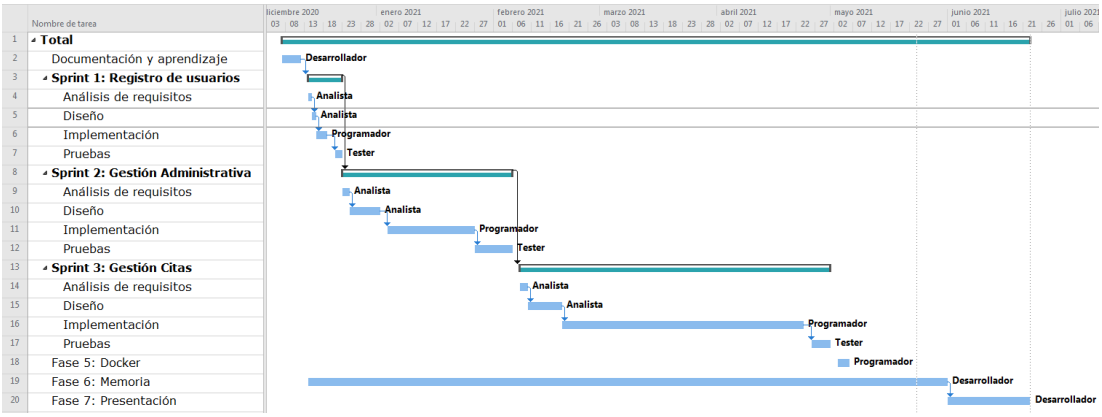
- Gestión de Citas. En este *sprint* se llevó a cabo la parte relacionada con las citas. En concreto: añadir, editar, listar y borrar citas y enviar el recordatorio de las citas del día siguiente.

A mayores de los *sprints* está la parte de *Docker*, la memoria y la presentación. La memoria se empezó a realizar a la par que el *Sprint 1: Gestión de usuarios*, aunque la mayor carga de trabajo se planificó realizar una vez terminada la aplicación. Para la presentación se decidió esperar a que se terminase la elaboración de la memoria. Y para *Docker* es necesario tener la aplicación finalizada, por ese motivo va después del último *sprint*.

En la figura 5.1 se puede ver el diagrama de *Gantt* de este proyecto.

| | Nombre de tarea | Duración | Comienzo | Fin | Pre | Nombres de los recursos |
|---|---|---|---|---|---|---|
| 1 | **Total** | **142 días** | **lun 07/12/20** | **mar 22/06/21** | | |
| 2 | Documentación y aprendizaje | 5 días? | lun 07/12/20 | vie 11/12/20 | | Desarrollador |
| 3 | **Sprint 1: Registro de usuarios** | **7 días?** | **lun 14/12/20** | **mar 22/12/20** | **2** | |
| 4 | Análisis de requisitos | 1 día? | lun 14/12/20 | lun 14/12/20 | | Analista |
| 5 | Diseño | 1 día? | mar 15/12/20 | mar 15/12/20 | 4 | Analista |
| 6 | Implementación | 3 días? | mié 16/12/20 | vie 18/12/20 | 5 | Programador |
| 7 | Pruebas | 2 días? | lun 21/12/20 | mar 22/12/20 | 6 | Tester |
| 8 | **Sprint 2: Gestión Administrativa** | **33 días?** | **mié 23/12/20** | **vie 05/02/21** | **3** | |
| 9 | Análisis de requisitos | 2 días? | mié 23/12/20 | jue 24/12/20 | | Analista |
| 10 | Diseño | 6 días? | vie 25/12/20 | vie 01/01/21 | 9 | Analista |
| 11 | Implementación | 17 días? | lun 04/01/21 | mar 26/01/21 | 10 | Programador |
| 12 | Pruebas | 8 días? | mié 27/01/21 | vie 05/02/21 | 11 | Tester |
| 13 | **Sprint 3: Gestión Citas** | **60 días?** | **lun 08/02/21** | **vie 30/04/21** | **8** | |
| 14 | Análisis de requisitos | 2 días? | lun 08/02/21 | mar 09/02/21 | | Analista |
| 15 | Diseño | 7 días? | mié 10/02/21 | jue 18/02/21 | 14 | Analista |
| 16 | Implementación | 46 días? | vie 19/02/21 | vie 23/04/21 | 15 | Programador |
| 17 | Pruebas | 5 días? | lun 26/04/21 | vie 30/04/21 | 16 | Tester |
| 18 | Fase 5: Docker | 3 días? | lun 03/05/21 | mié 05/05/21 | | Programador |
| 19 | Fase 6: Memoria | 121 días? | lun 14/12/20 | lun 31/05/21 | | Desarrollador |
| 20 | Fase 7: Presentación | 16 días? | mar 01/06/21 | mar 22/06/21 | 19 | Desarrollador |

(a) Parte 1.




(b) Parte 2.

Figura 5.1: Diagrama de Gantt.

## 5.2 Seguimiento

En este apartado se verá como evolucionó el proyecto respecto a lo inicialmente planificado y, al final, se verá la planificación real, es decir, el tiempo que finalmente llevó su realización.

Durante el transcurso de la realización del proyecto, hubo varias reuniones entre el autor del proyecto y el tutor que lo dirige. En dichas reuniones se fueron consensuando cambios y añadidos a la aplicación. De esta manera, después de algunas de esas reuniones, la planificación inicial fue variando, aunque no los plazos, los cuales se consiguieron cumplir. Esto se debe a que la planificación inicial tuvo en cuenta que podría haber cambios y por eso se usaron plazos más alargados de lo necesario. En caso de que no hubiese cambios, la planificación inicial no hubiese sido correcta, dado que el proyecto habría finalizado en un tiempo menor.

Para la realización de la memoria se decidió empezar al inicio del proyecto (como se vio en el apartado anterior, se empezó a realizar a la par que el *sprint* de gestión de usuarios) e ir haciéndola hasta el final del mismo. Si bien es cierto que la mayor carga de trabajo fue una vez finalizada la aplicación.

El diagrama de Gantt con la planificación real se puede ver en la figura 5.2. Este diagrama se realiza una vez el proyecto está finalizado, es decir, es el tiempo que realmente llevó la realización del mismo. Como se puede apreciar, y se comentó anteriormente, los plazos son iguales pero se puede ver como ahora se tienen en cuenta las reuniones y los cambios realizados después de las mismas.

Los *sprints* surgidos de las reuniones son los siguientes:

- Primera reunión. En esta reunión se añadieron las siguientes funcionalidades: pantallas en sala de espera, confirmación por QR, importación/exportación de datos en hojas de cálculo, reprogramación de citas (ambos métodos: anular y reubicar), llamada paciente y confirmación paciente visto.

- Segunda reunión. En esta reunión se añadieron las siguientes funcionalidades: *bot* de *Telegram*, forzar cita, cita correlacionada y automatizar llamada siguiente paciente.

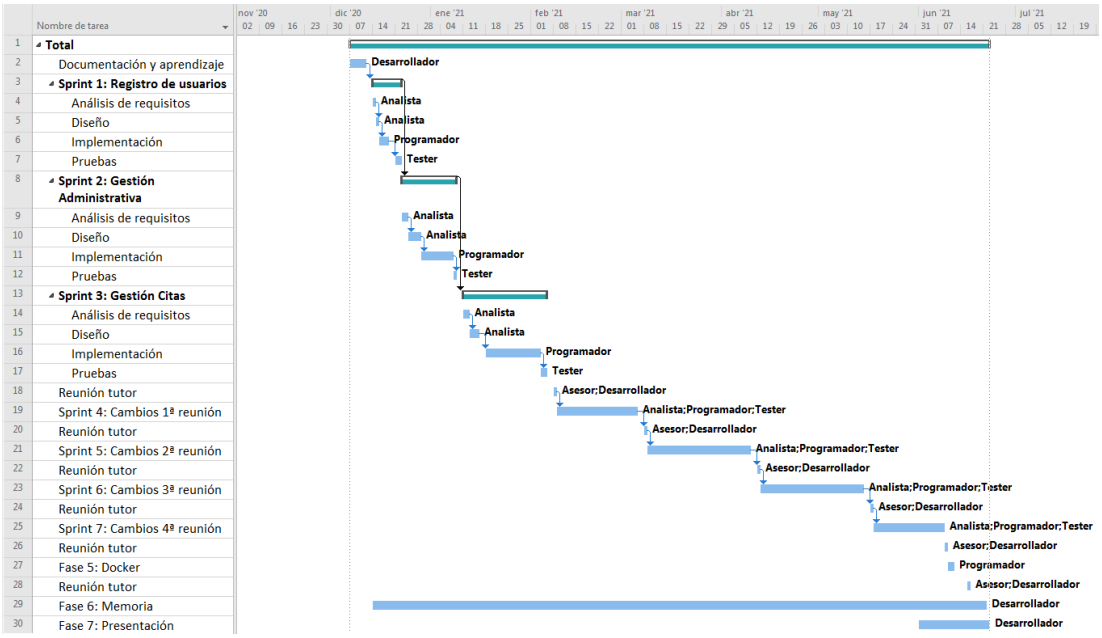
- Tercera reunión. En esta reunión se añadieron las siguientes funcionalidades: ley de protección de datos, preferencias canales de comunicación, calendario de *Google* y opción de eliminar cuenta para los pacientes.

- Cuarta reunión. En esta reunión se añadieron las siguientes funcionalidades: la importación/exportación se pasa a hacer desde CSV (en lugar de hojas de cálculo como se hacía después de la primera reunión), mejoras visuales para dispositivos móviles, se avisa al paciente si en una fecha y para un médico ya tiene una cita (aunque se permite

pedir otra, se le avisa para que sea consciente), automatizar siguiente llamada también confirma al último paciente visto y opción de eliminar la última llamada realizada (por si el paciente es llamado pero no acude a la sala).

| | Nombre de tarea | Duración | Comienzo | Fin | Pr | Nombres de los recursos |
|---|---|---|---|---|---|---|
| 1 | **Total** | **142 días?** | **lun 07/12/2** | **mar 22/06/21** | | |
| 2 | Documentación y aprendizaje | 5 días? | lun 07/12/2 | vie 11/12/20 | | Desarrollador |
| 3 | **Sprint 1: Registro de usuarios** | **7 días?** | **lun 14/12/2** | **mar 22/12/20** | **2** | |
| 4 | Análisis de requisitos | 1 día? | lun 14/12/2 | lun 14/12/20 | | Analista |
| 5 | Diseño | 1 día? | mar 15/12/2 | mar 15/12/20 | 4 | Analista |
| 6 | Implementación | 3 días? | mié 16/12/2 | vie 18/12/20 | 5 | Programador |
| 7 | Pruebas | 2 días? | lun 21/12/2 | mar 22/12/20 | 6 | Tester |
| 8 | **Sprint 2: Gestión Administrativa** | **13 días?** | **mié 23/12/20** | **vie 08/01/21** | **3** | |
| 9 | Análisis de requisitos | 2 días? | mié 23/12/2 | jue 24/12/20 | | Analista |
| 10 | Diseño | 2 días? | vie 25/12/2 | lun 28/12/20 | 9 | Analista |
| 11 | Implementación | 8 días? | mar 29/12/2 | jue 07/01/21 | 10 | Programador |
| 12 | Pruebas | 1 día? | vie 08/01/21 | vie 08/01/21 | 11 | Tester |
| 13 | **Sprint 3: Gestión Citas** | **20 días?** | **lun 11/01/2** | **vie 05/02/21** | **8** | |
| 14 | Análisis de requisitos | 2 días? | lun 11/01/2 | mar 12/01/21 | | Analista |
| 15 | Diseño | 3 días? | mié 13/01/2 | vie 15/01/21 | 14 | Analista |
| 16 | Implementación | 13 días? | lun 18/01/2 | mié 03/02/21 | 15 | Programador |
| 17 | Pruebas | 2 días? | jue 04/02/2 | vie 05/02/21 | 16 | Tester |
| 18 | Reunión tutor | 1 día? | lun 08/02/2 | lun 08/02/21 | | Asesor;Desarrollador |
| 19 | Sprint 4: Cambios 1ª reunión | 19 días? | mar 09/02/2 | vie 05/03/21 | 18 | Analista;Programador;Tester |
| 20 | Reunión tutor | 1 día? | lun 08/03/2 | lun 08/03/21 | 19 | Asesor;Desarrollador |
| 21 | Sprint 5: Cambios 2ª reunión | 24 días? | mar 09/03/2 | vie 09/04/21 | 20 | Analista;Programador;Tester |
| 22 | Reunión tutor | 1 día? | lun 12/04/2 | lun 12/04/21 | 21 | Asesor;Desarrollador |
| 23 | Sprint 6: Cambios 3ª reunión | 24 días | mar 13/04/2 | vie 14/05/21 | 22 | Analista;Programador;Tester |
| 24 | Reunión tutor | 1 día? | lun 17/05/2 | lun 17/05/21 | 23 | Asesor;Desarrollador |
| 25 | Sprint 7: Cambios 4ª reunión | 16 días? | mar 18/05/2 | mar 08/06/21 | 24 | Analista;Programador;Tester |
| 26 | Reunión tutor | 1 día? | mié 09/06/2 | mié 09/06/21 | | Asesor;Desarrollador |
| 27 | Fase 5: Docker | 2 días? | jue 10/06/2 | vie 11/06/21 | | Programador |
| 28 | Reunión tutor | 1 día? | mié 16/06/2 | mié 16/06/21 | | Asesor;Desarrollador |
| 29 | Fase 6: Memoria | 136 días? | lun 14/12/2 | lun 21/06/21 | | Desarrollador |
| 30 | Fase 7: Presentación | 16 días? | mar 01/06/2 | mar 22/06/21 | | Desarrollador |

(a) Parte 1.




(b) Parte 2.

Figura 5.2: Diagrama de Gantt final.

| Unidades | Elemento | Coste unitario | Subtotal |
|---|---|---|---|
| 1 | HP Laptop 15-bs023la (i5-7200U, 8GB ram, 2TB disco duro) | 456'00€ | 456'00€ |
| 1 | Smartphone Xiaomi Redmi 9C | 117'00€ | 117'00€ |
| 1 | S.O. Linux (Ubuntu 20.04) | - | - |
| 1 | Django 3.1.4 | - | - |
| 1 | SQLite 3 | - | - |
| 1 | Python 3.8.5 | - | - |
| 1 | Docker 19.03.13 | - | - |
| | | **Total** | **573'00€** |

Cuadro 5.1: Costes *hardware* y *software*.

## 5.3 Evaluación de costes

Los costes asociados al desarrollo de la aplicación *ClínicasMédicas* se pueden dividir en costes de material (*hardware* y *software*) y costes de personal.

### 5.3.1 Costes *hardware* y *software*

Para la parte de *hardware* son necesarios un ordenador portátil y un teléfono móvil. Los precios y las características se puede ver en la tabla 5.1. En esta misma tabla se puede ver el S.O. y el *software* utilizados, todos tienen un coste de 0€ por tratarse de *software* libre y gratuito. La explicación de cada uno y el motivo de utilizarlo se pueden ver en el capítulo 2.

### 5.3.2 Costes personal

Dada la naturaleza individual de un TFG, sólo hay una persona que realiza el trabajo, la cual desempeña diversos roles según las etapas de desarrollo. Hay que tener en cuenta el número de horas de cada rol y su precio medio por hora. Se añadirá el tiempo del profesor, que realiza el rol de asesor.

El alumno realiza los roles de jefe de proyecto, analista, programador y *tester* (persona que realiza las pruebas de la aplicación). Como se vio en planificación, también realiza la labor de lo que se ha denominado desarrollador.

Ver la tabla 5.2.

### 5.3.3 Costes totales

En la tabla 5.3 se pueden ver los costes totales del proyecto.

Asciende el proyecto de "Aplicación web para la gestión de citas de una cadena de clínicas médicas", a la cantidad de NUEVE MIL SEISCIENTOS CUARENTA Y SIETE EUROS CON TREINTA Y TRES CÉNTIMOS (I.V.A. Incluido).

| Unidades (horas) | Trabajador | Coste unitario | Subtotal |
|---|---|---|---|
| 100 | Desarrollador | 20'00€ | 2.000'00€ |
| 20 | Jefe de proyecto | 50'00€ | 1.000'00€ |
| 60 | Analista | 30'00€ | 1.800'00€ |
| 100 | Programador | 20'00€ | 2.000'00€ |
| 20 | Tester | 15'00€ | 300'00€ |
| 10 | Asesor | 30'00€ | 300'00€ |
| | | **Total** | **7.400'00€** |

Cuadro 5.2: Costes trabajador.

| Concepto | Coste |
|---|---|
| *Software + Hardware* | 573'00€ |
| Trabajador | 7.400'00€ |
| **Total sin I.V.A.** | 7.973'00€ |
| **I.V.A. (21%)** | 1.674'33€ |
| **Total** | **9.647'33€** |

Cuadro 5.3: Costes totales.

# Capítulo 6

# Diseño

En este capítulo se verá el diseño de la aplicación *ClínicasMédicas*. Para realizar el diseño se recogen los datos obtenidos durante el análisis de requisitos. Es importante que estos requisitos sean completos y de calidad. En caso contrario, no será posible realizar un buen diseño que cumpla con las expectativas del proyecto.

## 6.1 Arquitectura del sistema

Un proyecto en *Django* se divide en aplicaciones. La idea de la aplicación es crear partes independientes y reusables para otros proyectos. En este caso, se crearon dos aplicaciones. Una encargada de la gestión administrativa de la clínica y otra encargada de la gestión de citas. A mayores se hace uso de una aplicación de *Django* que gestiona los usuarios, necesaria para la parte de recuperar la contraseña.

### 6.1.1 Patrón de diseño Django

*Django* hace uso del patrón de diseño MTV (*Model-Template-View* o Modelo-Plantilla-Vista), que es una versión, usada por *Django*, del MVC (*Model-View-Controller* o Modelo-Vista-Controlador).

El patrón MVC separa el código en las tres capas anteriormente mencionadas, cada una teniendo una responsabilidad: [26]

- Modelo: es la capa responsable de acceder a los datos de la BBDD y modificarlos.
- Vista: es la capa responsable de la interfaz de usuario, es lo que ve el usuario y lo que le permite interactuar con el sistema.
- Controlador: es la capa responsable de la lógica de negocio y la que conecta las 2 anteriores. Por tanto, esta capa ni muestra datos de salida ni modifica datos del sistema, simplemente actúa de puente entre la vista y el modelo.

Por su parte, el patrón MTV realiza alguna modificación del patrón MVC, aunque la idea general sigue siendo la misma: [27] [28]

- Modelo: al igual que en el patrón MVC es la capa que accede y modifica los datos de la BBDD. Va en el archivo *models.py*. Existe un archivo por cada aplicación del proyecto.
- Plantilla: es el equivalente a la vista del patrón MVC, es decir, es la capa responsable de la interfaz de usuario. Las plantillas van en la carpeta *templates* y son archivos *HTML*, hay una por cada aplicación.
- Vista: es el equivalente al controlador del patrón MVC, sirve de puente entre el modelo y la plantilla. Las vistas van en el archivo *views.py*, nuevamente hay uno por cada aplicación.

En la figura 6.1 se puede ver el ciclo de una solicitud y su respuesta en *Django*.

### 6.1.2 Arquitectura del sistema

El sistema utiliza la arquitectura típica de *Django*, que es una arquitectura *shared nothing*. Esta arquitectura se caracteriza por tener nodos independientes y autosuficientes, evitando puntos de contención en el sistema. *Django* maneja esto de forma más o menos transparente. [29] [30]

En este proyecto solo se tiene un equipo, que actúa de servidor (se puede ver en la figura 6.2 la arquitectura con la que empiezan la mayoría de servidores, entre ellos este proyecto), pero, gracias a esta arquitectura, se podría escalar en un futuro.

El *hardware* necesario es:

- Ordenador con acceso a internet para el desarrollo (en el capítulo 5 se pueden ver las características del ordenador empleado en este proyecto). También es necesario un sistema para acceder después a las vistas.
- Pantallas para la sala de espera.
- Móvil con acceso a internet.
- Sistema que funcione de servidor (para este proyecto se utiliza el mismo ordenador que para el desarrollo pero se podrían tener dos diferentes).

Para la parte *software* es necesario:

- Las herramientas y tecnologías vistas en el capítulo 2.

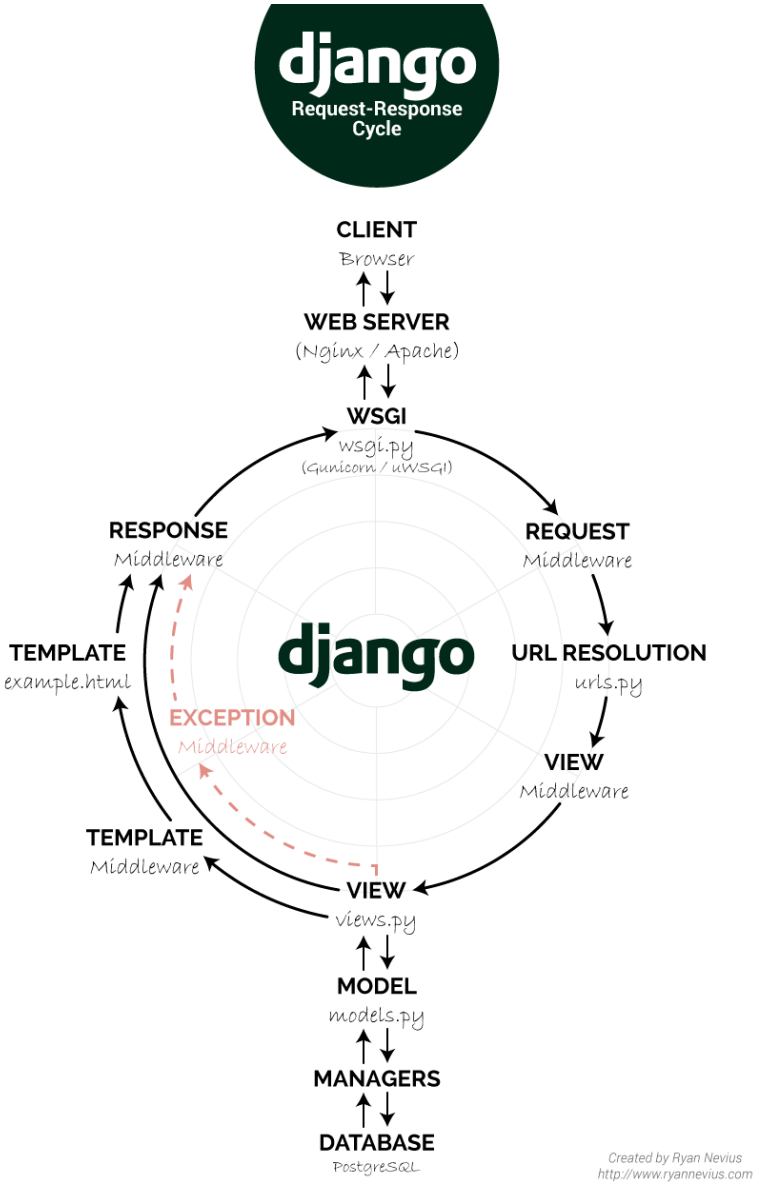



Figura 6.1: Ciclo solicitud-respuesta Django.

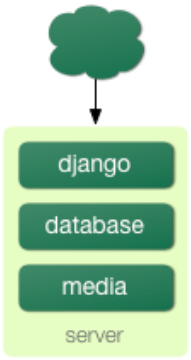



Figura 6.2: Configuración Django.

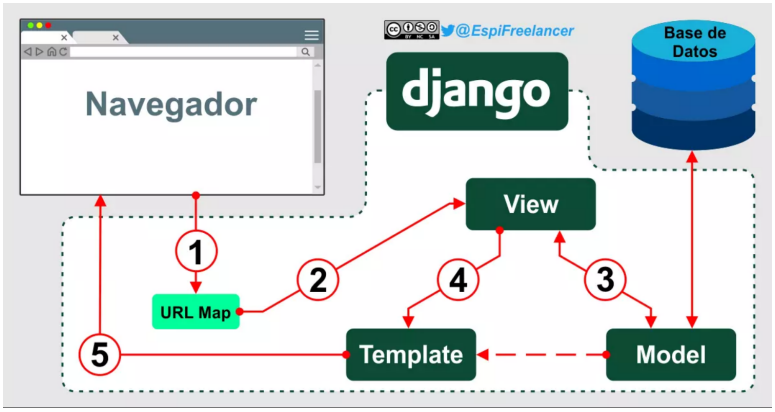



Figura 6.3: Estructura básica funcionamiento Django.

- Navegador web para acceder a las vistas de la aplicación.

En la figura 6.3 se puede la estructura básica del funcionamiento de *Django*.

Aparte de los archivos usados en el patrón de diseño (ver 6.1.1), existen archivos adicionales que son necesarios para el correcto funcionamiento. Son los siguientes:

- Formularios: los formularios sirven para recoger los datos de las vistas y pasárselos al modelo, para que los guarde en la BBDD. También se utilizan cuando se van a editar datos. Van en el archivo *forms.py*.
- URLs: las URLs sirven para acceder a las diferentes plantillas y conectarlas con su correspondiente vista. Van en el archivo *urls.py*.
- Configuración: la configuración va en el archivo *settings.py* y es donde se ponen las diferentes aplicaciones de las que se hace uso, se modifica el huso horario, se permite

el acceso al proyecto, etc. En este caso solo hay un archivo para todo el proyecto, a diferencia de los anteriores, que existe uno para cada aplicación del mismo.

*Django* también proporciona un panel de administración, al que se puede acceder desde *localhost:8000/admin*. Requiere crear un superusuario, que se puede crear con la siguiente sentencia:

```
1 python3 manage.py createsuperuser
```

El archivo *manage.py* es creado automáticamente al crear el proyecto, no requiere modificación y se usa para lanzar las sentencias (por ejemplo, para crear el superusuario o para poner a correr el servidor).

## 6.2 Diagrama de clases

En la siguiente figura (ver 6.4) se puede ver el diagrama de clases. Un diagrama de clases es una forma visual de ver las clases que componen la aplicación, sus atributos y como éstas se relacionan entre sí.

La clase *User* es una clase propia de Django, de la cual se hace uso para crear y gestionar los usuarios que tiene la aplicación. Los atributos mostrados son los que se utilizarán, dejando sin añadir aquellos de los que no se hace uso.

Como se vio en el apartado anterior, el proyecto está dividido en dos aplicaciones *Django*. Una para la parte administrativa y otra para la gestión de citas. En la parte administrativa, se tendrán las siguientes clases: Paciente, Clínica, Médico, Administrativo y Jornada. Para la gestión de citas se hará uso de clases de la parte administrativa y, a mayores, se definirá una clase propia: Cita.

De los pacientes se guarda su usuario (en el caso de paciente con usuarios, para los que no tienen usuario este campo será nulo), nombre, apellidos, email, teléfono, *chatid* (que es el id necesario para poder contactar mediante el *bot* de *Telegram*) y las dos opciones para las preferencias de los canales de comunicación: enviar_correo y enviar_telegram. Las preferencias se podrían almacenar en una clase aparte pero se decidió hacerlo de esta manera. El id, tanto en esta clase como en el resto, se genera automáticamente cuando se añade a la BBDD.

En el caso de los médicos y administrativos se guarda el nombre, apellidos y email (este email es el profesional de la clínica, no el suyo personal). También se guarda en que clínica trabajan y el usuario que tienen asociado (necesario para iniciar sesión en el sistema). De los médicos, a mayores, se guarda la especialidad, el número de colegiado y la sala en la que atienden.

En la clase Cita hay un atributo **cita_correlacionada** que relaciona una cita con otra. Esto es necesario porque hay citas, del mismo paciente, que tienen relación entre ellas. Un ejemplo de esto es cuando un paciente tiene primero una prueba y seguidamente, el mismo día, la cita con el médico para ver los resultados de dicha prueba. El atributo **orden_llamada** guarda el orden en que son llamados los pacientes, esto se hace para la parte de la llamada automática, que confirma al último visto. El resto de atributos son el paciente, el médico, la clínica, la fecha, la hora, la sala, confirmada (que es donde se guarda si una cita está o no confirmada), llamada_medico (que es donde se guarda si un paciente ha sido llamado o no), atendido (si ya fue atendido o no) y codigo (código de la cita, que se usa para llamar al paciente en la sala de espera).

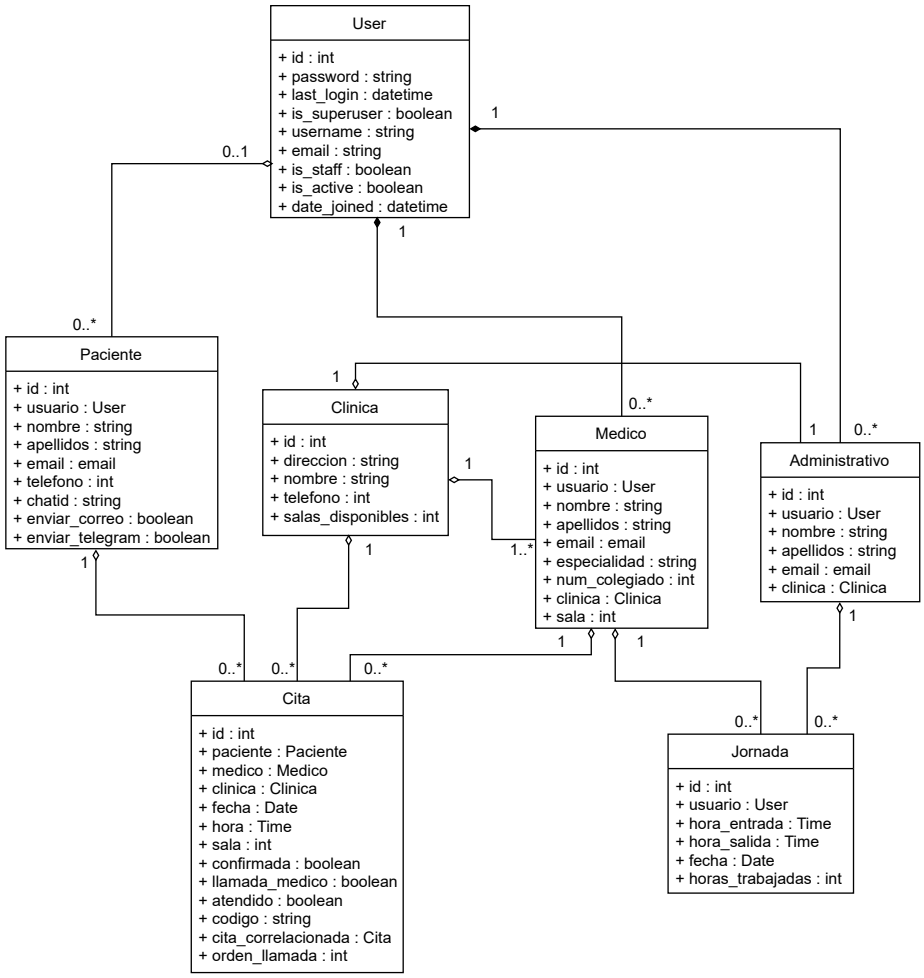



Figura 6.4: Diagrama de clases de la aplicación *ClínicasMédicas*.

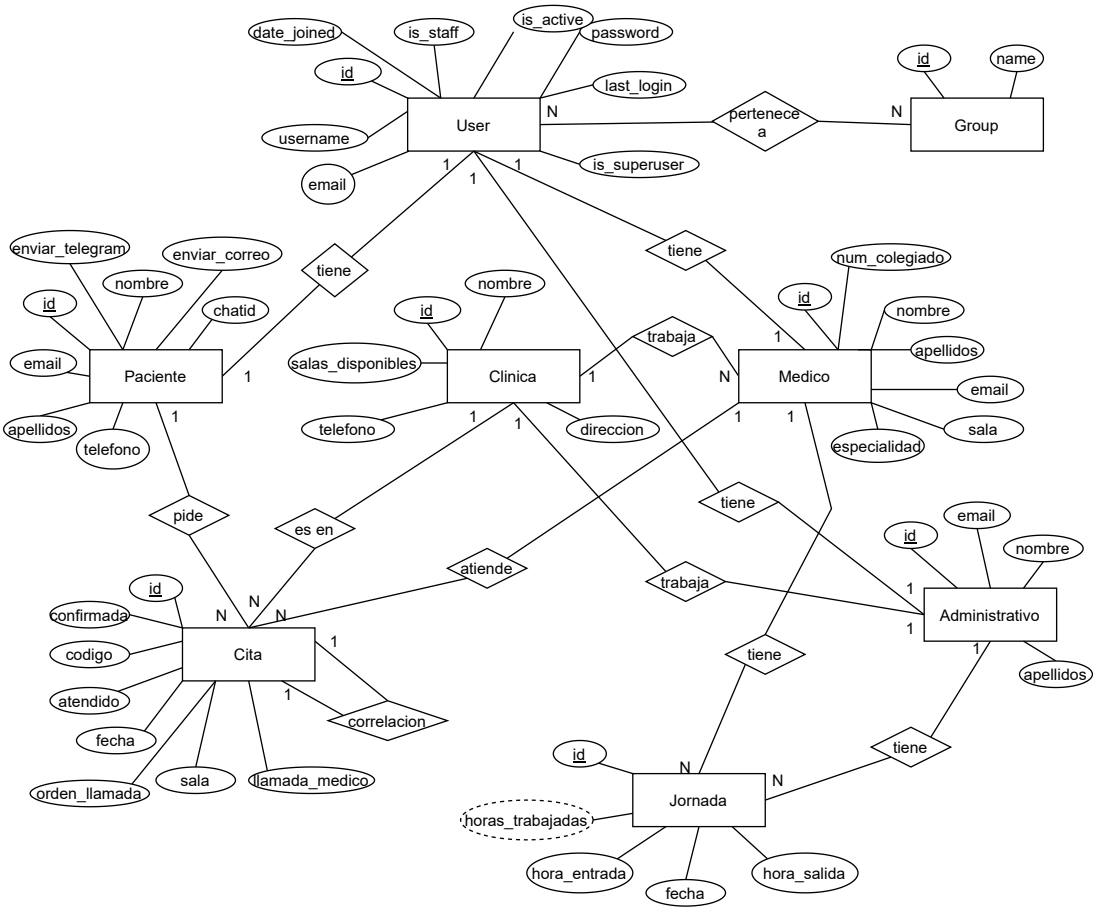



Figura 6.5: Modelo Entidad-Relación de la aplicación *ClínicasMédicas*.

## 6.3 Modelo de datos

### 6.3.1 Modelo entidad-relación

En este apartado se verá el modelo entidad-relación, que sirve para ver como serán las tablas de la BBDD. Este modelo no es directamente implementable, primero se debe hacer el paso al modelo relacional, que se verá en el siguiente apartado. Las entidades *User* y *Group* son proporcionadas por *Django*, el resto se crearán. Ver figura 6.5.

Debido a que es *Django* el que se encarga de crear las tablas de la BBDD las entidades y los atributos son iguales que las clases creadas en el apartado anterior. La diferencia es que las clases son las que usan luego en el sistema para realizar las funciones, mientras que las tablas de la BBDD son las que almacenan la información que las funciones van creando o modificando.

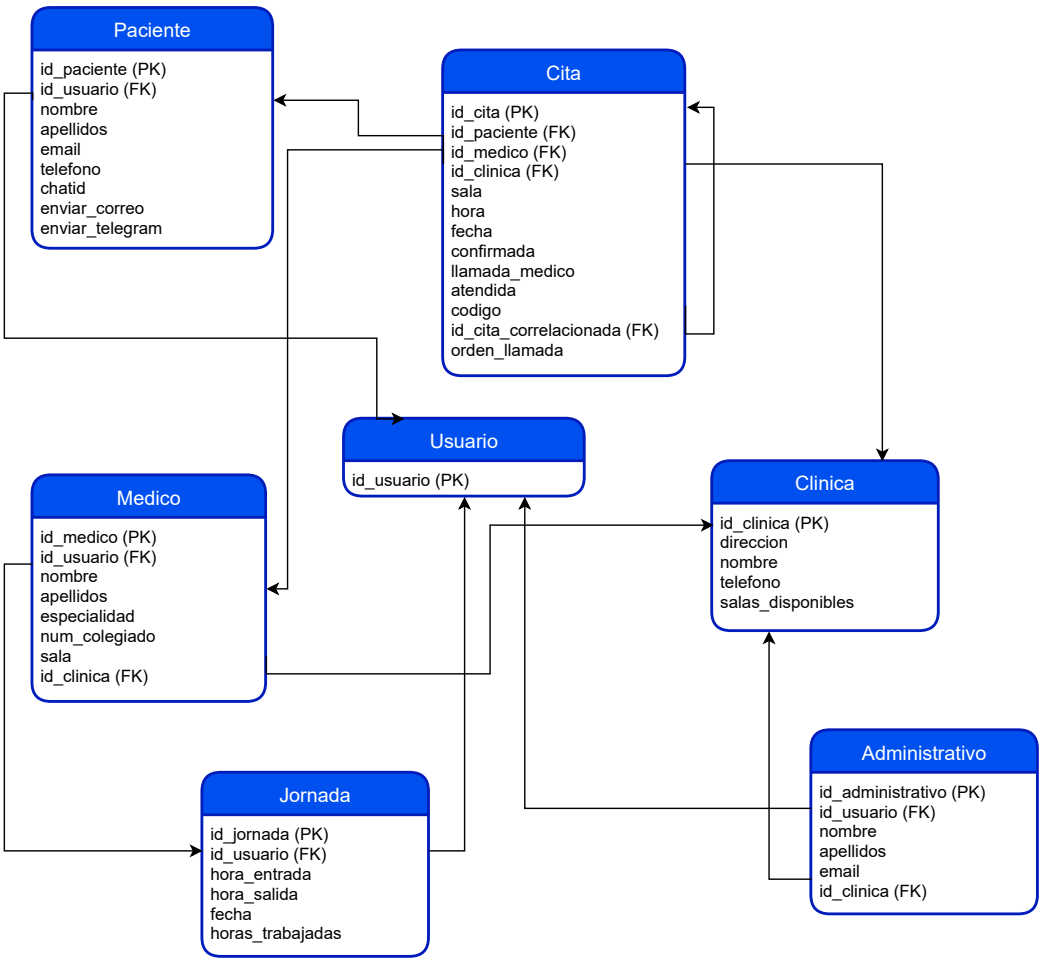



Figura 6.6: Modelo relacional de la aplicación *ClínicasMédicas*.

### 6.3.2 Modelo relacional

En este apartado se verá el paso del modelo entidad-relación al modelo relacional. Este paso es necesario para poder implementarlo luego en una BBDD, *Django* se encarga de hacer esto de manera transparente al programador. Como se mencionó anteriormente, las tablas *User* y *Group* ya son proporcionadas por *Django*, así que no se verá su transformación. Ver figura 6.6.

Como no se incorpora la transformación a *User*, dado que es una tabla que ya trae *Django*, solo se puso en el modelo relacional a modo de referencia para indicar que claves foráneas se dirigen a dicha tabla. *PK* es *Primary Key* (clave primaria) y *FK* es *Foreign Key* (clave foránea).

## 6.4 Seguridad y protección de datos

En este apartado se verán dos aspectos relativos a la seguridad. Por una parte se verá la seguridad que proporciona *Django* frente a ataques comunes. Por otra parte se verán las decisiones tomadas en cuanto al cumplimiento de la protección de datos.

### 6.4.1 Seguridad Django

Dado que este proyecto se centra en la gestión de citas principalmente, se tomó la decisión de no incluir datos relativos a informes médicos ni datos sensibles de los pacientes, médicos o administrativos. De esta manera, solo se guardan datos tales como el nombre, el email o el teléfono.

Principales ataques sobre los que *Django* proporciona seguridad [31]:

- **Protección contra XSS (*Cross Site Scripting*)**

  Un ataque *XSS* permite a un atacante ejecutar un script en la parte cliente de otro usuario.

  El uso de plantillas en *Django* protege contra la mayoría de ataques *XSS* pero también tiene sus limitaciones. Por ejemplo, *Django* escapa caracteres específicos:

  ```
  1 <style class={{ var }}>...</style>
  ```

  Aprovechando esa variable **var** un atacante podría ejecutar un código *JavaScript* no autorizado.

- **Protección contra CSRF (*Cross Site Request Forgery*)**

  Los ataques *CSRF* permiten a un atacante entrar con las credenciales de otro usuario sin que éste sea consciente y realizar ataques con esas credenciales.

  *Django* proporciona protección contra este tipo de ataques mediante la inclusión de **{% csrf-token %}** en los formularios de las vistas. Con esto, el atacante tendría que conocer el *token* del usuario (que es secreto y específico del usuario) para poder llevar a cabo este tipo de ataque.

- **Protección contra *SQL Injection***

  *SQL Injection* es un ataque que permite ejecutar código *SQL* sobre la BBDD para eliminar registros o filtrar datos.

*Django* protege de este tipo de ataques mediante la parametrización de las consultas, es decir, el código *SQL* de una consulta se construye independiente de los parámetros de la consulta.

- **Protección contra *Clickjacking***

  *Clickjacking* es un ataque donde el atacante le muestra al usuario una vista similar a la de la aplicación para que cuando pulse sobre algún enlace, le redirija a otro sitio distinto, le robe sus credenciales o le haga realizar alguna acción sin que se entere.

  *Django* proporciona protección contra estos ataques. Utilizando un navegador compatible se evita que la vista pueda ser modificada o enmarcada por otra *web*.

- ***SSL/HTTPS***

  *Django* permite hacer uso de *HTTPS*, dando esa seguridad extra respecto a *HTTP*. Debido a que puede ser peligroso configurarlo mal, se ha decidido no habilitarlo en esta aplicación.

### 6.4.2 Aplicación del Reglamento General de Protección de Datos

En el apartado 1.1.2 ya se vieron los fundamentos teóricos del RGPD, en este se verá la aplicación del mismo a la aplicación *ClínicasMédicas*.

Para cumplir con este reglamento, en la aplicación se han tomado una serie de medidas:

- Confirmación del usuario para algunos procedimientos. Se requiere la aceptación explícita por parte del usuario para poder realizar algunos procedimientos, como registrarse en la aplicación o enviar un formulario de contacto.

- Correo electrónico para poder ejercer los llamados derechos ARCO (Acceso, Rectificación, Cancelación y Oposición). El usuario tiene a su disposición un correo electrónico para comunicar que quiere ejercer sus derechos.

- Aviso de *cookies* al entrar en la aplicación. La primera vez que un usuario entra en la aplicación le sale un aviso de que se van a almacenar ciertas *cookies* y el usuario debe aceptar para continuar navegando por la aplicación.

## 6.5 Módulos del sistema

En este apartado se verán los módulos del sistema y las decisiones tomadas respecto a los mismos.

### 6.5.1 Horas citas

En el momento de elegir la hora para una cita se tomó la decisión de implementarlo como un sistema cerrado con unas horas preestablecidas. Las horas van en intervalos de 15 minutos, empezando desde las 08:00 y finalizando a las 15:00. En el apartado 7.1.1 se verá un script para poder modificar este intervalo y las horas de inicio y fin. Los pacientes solo pueden seleccionar un intervalo horario que esté libre (es decir, si el intervalo de las 08:00 a las 08:15 está ocupado, éste no le aparecerá para seleccionar). Los médicos y administrativos pueden forzar una cita, esto es, pueden hacer que 2 citas de pacientes distintos estén en el mismo intervalo. A la hora de seleccionar un intervalo horario por este procedimiento (forzar cita), le aparecerán todos los intervalos estén o no libres. La única limitación es que un paciente no puede tener dos citas a la misma hora el mismo día con el mismo médico.

Más adelante se vio que esta decisión no fue del todo correcta, debido a que no todas las citas deberían tener la misma duración pero el coste de rehacer todo el sistema era demasiado elevado y, debido al tiempo disponible para el desarrollo de un TFG, se decidió dejarlo de esta manera, siendo conscientes de esta limitación.

### 6.5.2 Confirmación de citas

En este apartado se verán las decisiones tomadas a la hora de confirmar la asistencia a una cita.

Para confirmar una cita se puede hacer de 3 maneras distintas:

- A través de un administrativo. El paciente llega a la clínica, habla con el administrativo y éste confirma la cita.
- A través de un QR. El paciente escanea con el móvil un código QR que está a la entrada de la clínica, éste le redirige a una URL donde le salen las citas de ese día y le permite marcar de una en una aquellas que quiera confirmar.
- A través de *Telegram*. Cuando se pide una cita, si el paciente tiene habilitada la comunicación con *Telegram*, se le envía una URL, ésta le redirige a la misma dirección que el método anterior, siendo igual el resto del procedimiento.

En la vista donde están las citas del día (que es a donde redirige la URL tanto del QR como del *Telegram*) se decidió permitir al paciente elegir que citas confirmar, en lugar de confirmar

todas sin darle opción al paciente. Lo habitual será que el paciente solo tenga una cita pero se puede dar el caso de que tenga que hacer una prueba y después tenga cita con el médico o que tenga citas con médicos distintos. Por estos motivos, se eligió hacer de esta manera.

### 6.5.3 Reprogramación de citas

En este apartado se verá una aspecto de la aplicación que, por su naturaleza, se quiere destacar y explicar la idea detrás del mismo. Concretamente, se verá la reprogramación de citas médicas.

La reprogramación de citas médicas se lleva a cabo cuando un médico no está disponible en fechas donde ya han sido dadas citas. La aplicación está diseñada teniendo en cuenta dos tipos de pacientes, por un lado pacientes con correo electrónico y, por otro, pacientes sin él. La reprogramación de citas de los primeros se realizará de manera automática, mientras que la de los segundos será realizada manualmente por medio de un administrativo, que tendrá que ponerse en contacto por teléfono con los pacientes. En lugar de correo electrónico también es posible utilizar el *bot* de *Telegram*, siendo igual el procedimiento. En cualquier caso, el paciente debe haber dado el visto bueno (en Preferencias) para permitir la comunicación por estos medios.

Para reprogramar citas se tendrá 2 métodos. El primero consiste en anular todas las citas de las fechas introducidas y avisar al paciente, por correo electrónico, que solicite una nueva cita. La segunda consiste en mover todas las citas a huecos disponibles en fechas en las que el médico esté, avisando igualmente por correo electrónico de la nueva cita. Esto aplica para aquellos que se hace de manera automática, para el resto, como se dijo anteriormente, deberá ser hecho a mano por un administrativo.

## 6.6 Canales de comunicación del sistema

En este proyecto se tienen los siguientes canales para la comunicación con el paciente:

- Correo electrónico.
- *Bot* de *Telegram*.
- Teléfono.

A través de estos canales existe una comunicación con los usuarios. Los pacientes tienen una vista a su disposición donde decidir si quieren recibir o no notificaciones a través del correo electrónico y de *Telegram*. Se decidió no dar opción de elegir si quieren o no comunicación por teléfono debido a que hay ciertas cosas que requieren su uso. De todas maneras, existe la opción de que el paciente solicite a un administrativo que elimine su teléfono de la BBDD. De esta manera, en caso de que una cita se anule o se modifique, si no tiene más canales permitidos, el paciente no se enteraría hasta que no vaya a la clínica. Se eligió hacer de esta manera porque se intenta que el sistema funcione de la manera más automáticamente posible, dejando a decisión del paciente el salir de los flujos de funcionamiento habituales.

El *bot* de *Telegram* está configurado para enviar los siguientes mensajes (para esta explicación se supondrá que el paciente tiene permitido el envío de mensajes por este medio):

- Cuando un paciente solicita una cita. El *bot* envía un mensaje indicando la fecha y hora de la cita. También envía un enlace personalizado del calendario de *Google* por si el paciente desea añadirlo. Y envía un enlace para que el paciente pueda confirmar la cita cuando acuda a la misma. Para evitar que confirme citas con anterioridad, el sistema comprueba solo las del mismo día en las que el paciente pulsa el enlace.
- Cuando se confirma una cita. El *bot* envía un mensaje indicando que la cita ha sido confirmada correctamente.
- Cuando un médico llama a un paciente. El *bot* envía un mensaje diciéndole a que sala debe acudir.
- Cuando se reprograma una cita. Tanto si la cita es anulada como si es pospuesta, el *bot* envía un mensaje al paciente avisándolo de estos cambios.

## 6.7 Generalización

En este apartado se verán los cambios a realizar para adaptar la aplicación a otros dominios que también requieren citas. Concretamente se verá el caso de la ITV, el caso de una peluquería y el caso de una clínica veterinaria.

Con estos 3 ejemplos se puede observar como el proyecto realizado se puede reutilizar para cualquier servicio donde se requieran citas. Entre otros motivos, este es uno de la conveniencia de usar *Django*, dado que permite implementar estos cambios de una forma relativamente sencilla gracias a dividir el proyecto en aplicaciones reutilizables.

### 6.7.1 Inspección Técnica de Vehículos

En la ITV en lugar de médicos, clínicas y pacientes hay operarios, estaciones y clientes del servicio. Las estaciones suelen disponer de varias líneas para la revisión del vehículo, algunas de estas líneas revisan tipos de vehículos distintos (tales como camiones o tractores). Por este motivo, en lugar de la especialidad del médico, el operario tiene el tipo de vehículo que inspecciona. A diferencia de los médicos, que solo tienen una especialidad, los operarios pueden inspeccionar distintos tipos de vehículos.

- **User**: esta clase se mantiene.
- Paciente –> **Cliente**: los pacientes pasarían a ser los clientes que van a realizar la inspección de su vehículo.
- Clínica –> **Estación de ITV**: las clínicas serían las distintas estaciones disponibles.
- Médico –> **Operario**: los médicos pasarían a ser los operarios que realizan la inspección, la sala pasaría a ser la línea que cada operario atiende y la especialidad el tipo de vehículos que son atendidos por cada operario.
- **Administrativo**: esta clase se mantiene.
- **Jornada**: también se mantiene.
- **Cita**: se mantiene.

### 6.7.2 Peluquería

Una peluquería es un caso particular ya que normalmente solo hay un local, aunque también hay casos de cadenas. Los clientes de la peluquería se podría prescindir de ellos, dado que lo normal es no registrar los datos personales, pero se mantiene esa clase para la parte de solicitar cita a través de la aplicación. Los peluqueros también hacen la parte administrativa, aunque si es una peluquería grande se podría añadir un administrativo para ciertas tareas.

- **User**: esta clase se mantiene.
- Paciente –> **Cliente**: los pacientes pasarían a ser los clientes de la peluquería.
- Clínica –> **Peluquería**: las clínicas serían las distintas peluquerías. En caso de solo tener una, se podría prescindir de esta clase
- Médico –> **Peluquero**: los médicos pasarían a ser los peluqueros, que también realizan la función del administrativo. Si la peluquería es muy grande, se podría considerar la inclusión de un administrativo que atienda el teléfono, dé las citas y cobre a los clientes.
- **Administrativo**: esta clase no es necesaria.
- **Jornada**: también se mantiene.
- **Cita**: se mantiene.

### 6.7.3 Clínica veterinaria

En este caso los cambios son mínimos por tratarse, con sus diferencias, de clínicas similares. Los médicos pasarían a ser los veterinarios y los pacientes los animales que son atendidos.

- **User**: esta clase se mantiene.
- Paciente –> **Animal**: los pacientes pasarían a ser los animales que la clínica atiende. También es necesario guardar datos de la persona que lleva el animal a la clínica. No se ve necesario el crear una clase nueva, dado que la persona siempre va asociada al animal en cuestión.
- **Clínica**: se mantiene.
- Médico –> **Veterinario**: los médicos pasarían a ser los veterinarios.
- **Administrativo**: esta clase se mantiene.
- **Jornada**: también se mantiene.
- **Cita**: se mantiene.

Capítulo 7

# Implementación

En este capítulo se verá la implementación de la aplicación. Como se mencionó en apartados anteriores, para la implementación se hace uso del framework *Django*.

## 7.1 Funcionalidades del sistema

En este apartado se comentarán aspectos relativos a la implementación de ciertas funcionalidades del sistema.

### 7.1.1 Generación lista intervalo horas citas

Para generar las horas que se utilizan para los intervalos que se ofrecen luego a los pacientes, se utiliza un script que genera una lista de *Python*. Esta lista después se pega en el archivo *vars.py*. Los parámetros de entrada de este script son la hora inicial, el intervalo de duración de las citas y la hora final. Tanto la hora de entrada como la hora de salida tienen que ser horas exactas. Un ejemplo de entrada sería:

```
1 ./generar_horas.sh 8 15 15
```

Esto genera la lista que se usa en esta aplicación, empieza a las 08:00 (primer parámetro), con intervalos de 15 minutos (segundo parámetro) y hasta las 15:00 (tercer parámetro).

El script es el siguiente:

```
n=0
m=$1
m2=$2
m3=0

while [ $m -lt $(( $3 )) ]
do
  if [ $m3 -lt 10 ]
  then
    if [ $m2 -lt 10 ]
    then
      echo "($n, '"$m:0$m3" - "$m:0$m2"'),"
    else
      echo "($n, '"$m:0$m3" - "$m:$m2"'),"
    fi
  else
    if [ $m2 == 60 ]
    then
      echo "($n, '"$m:$m3" - "$(($m+1)):00"'),"
    else
      echo "($n, '"$m:$m3" - "$m:$m2"'),"
    fi
  fi
  n=$(( n+1 ))
  if [ $m2 -gt 60 ]
  then
    m=$(( m+1 ))
    m2=$2
    m3=0
  else
    m2=$(( m2+$2 ))
    if [ $m2 -gt 60 ]
    then
      m=$(( m+1 ))
      m2=$2
      m3=0
    else
      m3=$(( m3+$2 ))
    fi
  fi
done
```

### 7.1.2 Calendario de Google

Cuando un paciente solicita una cita, el sistema envía un mensaje a través de *Telegram* o correo electrónico. En dicho mensaje, va un enlace personalizado con la cita, el día y la hora para añadir un evento en el calendario de *Google*.

Para realizar esta operación se utiliza una petición URL al calendario de *Google* pasándole los parámetros adecuados. Dicha URL se genera de la siguiente manera:

Lo primero es una parte común *https://www.google.com/calendar/render?action=TEMPLATE*. A partir de aquí se pone el caso particular de cada paciente. Usando la etiqueta *text* se pone el título del evento y usando *dates* se pone la fecha y la hora.

Por ejemplo, una cita el día 25 de mayo de 12:00 a 12:15 quedaría de la siguiente manera:

```
https://www.google.com/calendar/render?action=TEMPLATE&text=Cita
&dates=20210525T100000Z%2F20210525T101500Z
```

Hay que tener en cuenta que la hora va en *GMT+0*, por lo que habrá que restar o sumar horas en función de la hora del sistema. En este caso, el proyecto es realizado desde *GMT+2*, así que se tienen que restar 2 horas.

### 7.1.3 Importación datos

La aplicación permite la importación de datos para facilitar el trabajo al administrativo y para permitir la integración con sistemas ya montados. Originalmente la importación se iba a hacer desde hojas de cálculo pero al final se eligió que la extensión de los archivos de importación sea **.csv** (CSV). El sistema también permite la exportación (en formato *.csv* nuevamente) pero no se verá en este apartado, dado que no tiene nada especial.

Como los usuarios se almacenan en una tabla diferente a los pacientes, administrativos y médicos, esto requiere un procedimiento especial. Para ello es necesario importar los usuarios primeramente y, después, mirar sus id [1] para añadirlos en los archivos *.csv* de pacientes, médicos y administrativos. Al final, éstos últimos ya se pueden importar. En el caso de los pacientes, hay la opción de que no tengan usuario, así que se podrían importar directamente, pero tanto los médicos como los administrativos requieren de uno.

[1] Para hacer esto se deberán exportar los datos de usuarios o que un administrador del sistema lo mire desde el panel de administración de *Django*. Es más rápido el primer método, dado que el segundo requiere de ir mirando de uno en uno mientras que con el primero salen todos en un mismo archivo

### 7.1.4 Copia de seguridad

Las copias de seguridad se realizan automáticamente todos los días a las 08:00. Para ello hay una tarea programada que la realiza cuando llega la hora indicada. Se utiliza el sistema *cron* propio de *Linux*, el cual es un proceso en segundo plano que se inicia al arrancar el sistema, comprueba si existen tareas programadas, las horas para las que lo están y se encarga de ejecutarlas.

Para restaurar una BBDD desde una copia de seguridad lo deberá hacer un administrativo mediante una función habilitada para ello. La función es automática, el administrativo simplemente pulsa sobre el botón habilitado y el sistema se encarga de seleccionar la última copia disponible y restaurar la BBDD.

### 7.1.5 Registro jornada laboral

La aplicación registra de manera automática la jornada laboral de los administrativos y médicos. Para ello, el inicio de la jornada se registra cuando el usuario inicia sesión y el fin de la misma lo hace cuando el usuario cierra sesión.

En la BBDD, como se vio, hay una tabla (llamada Jornada) donde se registran estos datos. Hay un campo donde se almacenan las horas trabajadas, que sale de restar la hora de entrada a la hora de salida. Dicho campo no se muestra a los usuarios, solo el administrador del sistema informático tiene acceso al mismo.

Inicialmente este fichaje se hacía de manera manual pero se decidió realizar de manera automática para facilitar el trabajo a los usuarios y para evitar que los mismos se olviden de registrarlo.

## 7.2 Implementación Django

En este apartado se verán las URLs de la aplicación y la distribución de carpetas de *Django*.

### 7.2.1 URLs

Cuando se accede a una vista en la aplicación, *Django* coge la URL del archivo *urls.py* y mira que función tiene asociada del archivo *views.py*, que es la que se muestra después al usuario (haciendo uso de las plantillas, archivos .html, que se encuentran en la carpeta *templates*).

Hay que tener en cuenta que se dispone de una URL inicial, de la cual salen todas las demás. En este caso, dicha URL es **http://localhost:8000/**. En esta aplicación hay muchas URLs, debido a las diversas funcionalidades que tiene y a las vistas intermedias.

A continuación se pueden ver las URLs principales de las que salen las demás. Las de citas salen directamente de *http://localhost:8000/* y las de administración salen de *http://localhost:8000/administracion*.

```
urlpatterns = [
    path('accounts/', include('django.contrib.auth.urls')),
    path('admin/', admin.site.urls),
    path('', include('GestionCitas.urls')),
    path('administracion/', include('GestionAdministrativa.urls')),
]
```

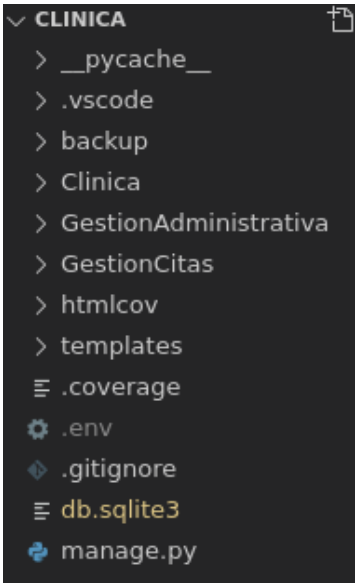
### 7.2.2 Estructura de carpetas

En este apartado se verá como se distribuyen los distintos archivos en *Django*. Como se mencionó anteriormente, *Django* se divide en aplicaciones. La idea detrás de esto es poder reutilizar estas aplicaciones para otros proyectos. En este proyecto hay dos aplicaciones, una para la parte administrativa y otra para la parte de citas. Además, se hace uso de un aplicación que trae *Django* para recuperar las contraseñas. Ésta última, al ser de *Django*, no tiene una carpeta ni archivos propios, solo se incluye en el archivo *urls.py* general. El resto sí tienen una carpeta y archivo propios.

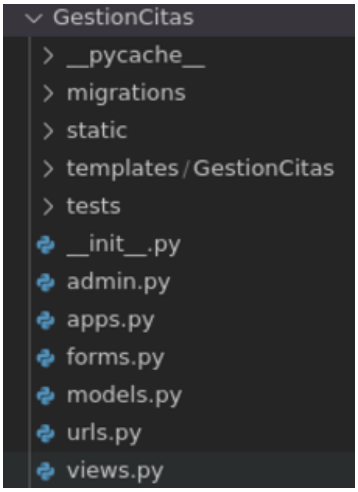
En la figura 7.1a se puede ver el directorio principal del proyecto. En el directorio *backup* es donde se almacenan las copias de seguridad creadas diariamente. Los tres siguientes directorios (cuya estructura se puede ver en las figuras 7.1b, 7.1c y 7.1d) son las aplicaciones del proyecto.

La de *Clinica* es la aplicación principal y es creada al crear el proyecto. Las otras dos (*GestionAdministrativa* y *GestionCitas*) se crean, mediante un comando de *Django*, y es donde van las funciones, las vistas y las URLs del proyecto. Los principales archivos de estos directorios son: *models.py* (archivo donde van los modelos de la aplicación), *views.py* (archivo donde van las funciones de la aplicación), *forms.py* (archivo donde van los formularios de la aplicación) y *urls.py* (donde van las URLs de la aplicación). Como se puede observar, cada aplicación tiene sus propios archivos independientes. Esto facilita la reutilización en otro proyectos.

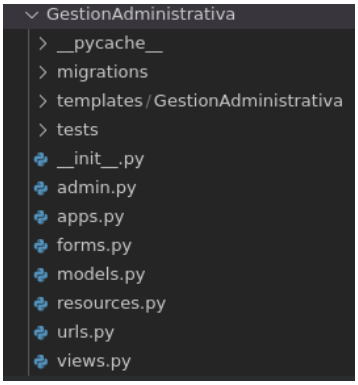
En cada aplicación hay dos directorios (*templates* y *tests*). El primero es donde van los archivos *.html*, que es lo que se le muestra al usuario por pantalla. El segundo es donde van los test para probar la aplicación. En el directorio de *GestionCitas* (ver figura 7.1d) hay otro directorio a mayores, llamada *static*, aquí es donde se almacenan las imágenes, los archivos *.css* y los archivos descargados para utilizar *bootstrap*.



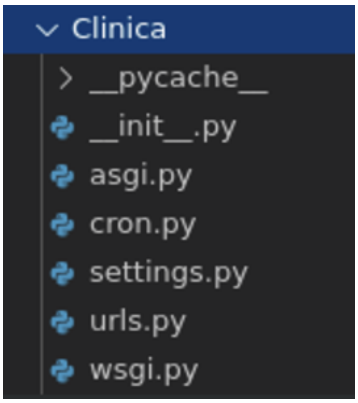

(a) Directorio principal.




(b) Directorio Gestión Citas.




(c) Directorio Gestión Administrativa.




(d) Directorio Clínica.

Figura 7.1: Estructura carpetas de la aplicación *ClínicasMédicas*.

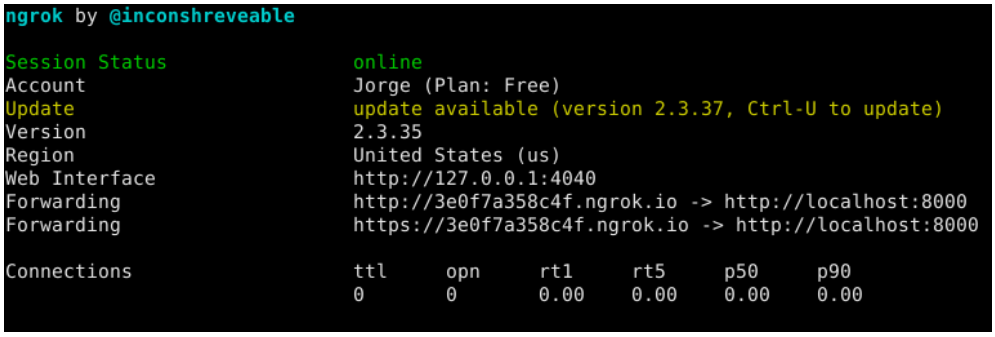



Figura 7.2: Ngrok en funcionamiento.

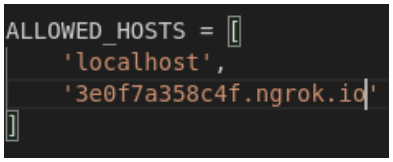



Figura 7.3: Archivo settings.py. Allowed_hosts.

## 7.3 Configuraciones

En este apartado se verán las configuraciones de *ngrok* y del *bot* de *Telegram*.

### 7.3.1 Ngrok

En primer lugar es necesario crear una cuenta en su página web. Una vez creada se procede a descargar los archivos. Desde terminal se debe configurar el token.

```
./ngrok authtoken <your_auth_token>
```

Con esto configurado ya se puede hacer uso de ella. Se lanza el siguiente comando. Nótese que se hace uso del mismo puerto que usa *Django* (el puerto 8000). La parte del *auth* es opcional pero permite poner credenciales de acceso para evitar que puedan entrar al túnel y acceder a la aplicación.

```
./ngrok http [-auth:"user:password"] 8000
```

Se debe permitir acceder a la Uniform Resource Locator (URL) que *Ngrok* nos proporciona (ver 7.2), para ello se irá al archivo *settings.py* y en *ALLOWED_HOSTS* se añadirá la dirección URL correspondiente (ver 7.3). Como no se dispone de cuenta *premium*, esta dirección no es fija, irá variando cada vez que se lance la aplicación, así que se debe modificar el archivo anteriormente citado de cada vuelta.

Por último, se genera el QR asociado a dicha URL y se añade a los archivos de la aplicación. Por la limitación vista anteriormente, hay que generar el QR cada vez que se lanza la

aplicación. En un entorno real, este QR sería fijo y asociado a la dirección URL de la clínica.

### 7.3.2 Bot Telegram

Para poder tener un *bot* personalizado, el primer paso es entrar desde *Telegram* al *bot BotFather*, el cual permite crear un *bot*. [32]

```
/start
/newbot
```

Escribiendo el primer comando, el *bot* muestra los comandos disponibles. El segundo comando crea un nuevo *bot*. Al lanzarlo pedirá introducir un nombre para el *bot* y un nombre de usuario (el cual debe acabar obligatoriamente en "bot"). Si el nombre de usuario está libre, el *bot* se creará correctamente. Este nombre de usuario es como se identifica el *bot* en los servidores de *Telegram*.

Una vez creado el *bot*, *BotFather* devuelve un mensaje que contiene dos partes importantes. La primera es un enlace para comunicarse con el *bot*. La segunda es un *token*, el cual es necesario para controlar el *bot*. Este *token* se debe mantener seguro, dado que, en caso de robo o pérdida, se perdería el acceso a controlar el *bot*.

Hasta ahora se tiene un *bot* creado que no realiza ninguna acción. Desde este punto, se trata de crear (desde código *Python*) la lógica que seguirá el *bot*. Con la lógica creada, lo primero que se deber hacer es que cada paciente guarde su *chatid*. Este proceso se explicará en el manual de usuario B. Una vez que el usuario tiene registrado su *chatid*, la aplicación ya procederá a enviarle mensajes mediante el *bot*.

```
Django==3.1.4
django_extensions==3.1.0
django-bootstrap-datepicker-plus==3.0.5
django-cookie-law==2.0.5
django-crispy-forms==1.10.0
django-crontab==0.7.1
django-dbbackup==3.3.0
django-import-export==2.5.0
django-mobi==0.1.7
mobi==0.3.1
tablib==3.0.0
pyTelegramBotAPI==3.7.6
python-decouple==3.3
requests==2.22.0
coverage==5.5
```

Figura 7.4: Archivo *requirements.txt*.

```
version: '3.9'

services:
  web:
    build: .
    command: python3 manage.py runserver
0:8000
    volumes:
      - .:/code
    ports:
      - "8000:8000"
```

(a) Archivo *docker-compose.yml*.

```
FROM python:3
ENV PYTHONUNBUFFERED 1

# App setup
ADD . /code
WORKDIR /code

# Requirements installation
RUN pip install -r requirements.txt
```

(b) Archivo *Dockerfile*.

Figura 7.5: Archivos *Docker* de la aplicación *ClínicasMédicas*: *docker-compose.yml* y *Dockerfile*.

## 7.4 Docker

En el capítulo 2 se vio que es *Docker*. En este apartado se verá como se configuró. [33] [34] [35]

Lo primero que se tiene que hacer es instalarlo, dado que no viene por defecto en *Linux*. Una vez instalado ya se puede proceder a configurar.

*Docker* funciona mediante archivos de configuración. Más concretamente se tienen dos archivos, llamados *Dockerfile* y *docker-compose.yml*. A mayores hay otro archivo que se usará, que no es algo de *Docker* pero se hará uso de él a la hora de configurarlo, denominado *requirements.txt*. En éste último es donde se guardan los paquetes utilizados para hacer funcionar el proyecto *Django*, tales como la versión de *Django*, la versión de *Python* y los diversos paquetes que se han utilizado (como el del *bot* de *Telegram* o el que permite la importación de datos, entre otros). Se puede ver en la figura 7.4.

En la figura 7.5a se ve lo que va en el archivo *docker-compose.yml*. Y en la figura 7.5b se puede ver el archivo *Dockerfile*.

Con estos archivos creados y configurados, se puede proceder a ejecutar los comandos para crear la imagen de *Docker* y subirlo a un repositorio para que esté accesible por cualquier interesado.

Lo primero es crear la imagen a partir de los archivos anteriores. Para ello se usa el siguiente comando:

```
docker build -t nombre:tag .
```

El nombre es el que se desea poner, en este caso se le ha llamado **tfg_gestion_citas**. El *tag* (o etiqueta) es por si se desean tener distintas versiones, en este caso no es necesario así que se podría prescindir de ponerlo pero se ha decidido ponerle uno igualmente, llamado **version_final**. El punto final es para decirle a *Docker* que los archivos están el directorio actual.

Una vez creada la imagen, se mira el id de la misma mediante el primer comando. El segundo comando sirve para marcar la imagen anteriormente creada. El *image-id* es el id obtenido mediante el primer comando. El *docker-id* es el nombre de usuario creado en la web de *Docker* para crear los repositorios. Lo siguiente vuelve a ser el nombre y la etiqueta de la imagen.

```
docker images
docker tag [Image-ID][Docker-ID]/[Image-Name]:[TAG]
```

Ya solo falta subir la imagen al repositorio (el repositorio se crea desde la web), para ello se utilizan los siguientes comandos:

```
docker login
docker push tfgclinicas2021/tfg_gestion_citas:version_final
```

El primero es para iniciar sesión con la cuenta de *Docker* y el segundo sube la imagen al repositorio.

De esta manera, la imagen ya está accesible para cualquier interesado. Para descargarla y poner el servidor a correr se utilizan los siguientes comandos:

```
docker pull tfgclinicas2021/tfg_gestion_citas:version_final
docker run -p 8000:8000
    tfgclinicas2021/tfg_gestion_citas:version_final python3
    manage.py runserver 0:8000
```

# Capítulo 8

# Pruebas

En este capítulo se verán las pruebas realizadas en el proyecto. Concretamente las pruebas de caja negra, las pruebas de caja blanca y las pruebas de aceptación.

## 8.1 Pruebas de caja negra

Las pruebas de caja negra son una serie de pruebas que se realizan sin tener en cuenta la estructura interna de la aplicación, es decir, sin tener en cuenta el código, la implementación o distintos escenarios de ejecución. Solo se mira la entrada y salida del sistema. [36]

Las pruebas de caja negra realizadas han de ser representativas de la aplicación y cubrir lo máximo posible pero no es necesario, ni apropiado, el tener pruebas de todo los casos de uso. En esta aplicación hay 4 casos donde se añaden, editan, listan y eliminan datos de la BBDD. Son los correspondientes a Clínica, Administrativo, Paciente y Médico. Para probar solo se hará uso de uno de ellos, en este caso se ha elegido Paciente por tener 2 tipos de paciente (con usuario y sin usuario). El resto de pruebas realizadas se corresponden con: pedir, eliminar y editar una cita, la reprogramación de citas, confirmar asistencia cita, restaurar la BBDD e importar datos desde un archivo CSV. Del resto de casos de uso no se realizarán pruebas de caja negra.

| ID | Descripción | Entrada | Resultado esperado |
|---|---|---|---|
| CP-01 | Añadir Paciente con todos los campos. | Formulario datos paciente. | Paciente añadido correctamente al sistema. |
| CP-02 | Añadir Paciente con campo requerido no introducido. | Formulario datos paciente con un campo requerido no introducido. | El sistema avisa de que el campo es requerido. |
| CP-03 | Añadir Paciente solo con campos requeridos. | Formulario datos paciente con campos requeridos introducidos. | Paciente añadido correctamente al sistema. |

Cuadro 8.1: Caso de prueba: Añadir Paciente.

| ID | Descripción | Entrada | Resultado esperado |
|---|---|---|---|
| CP-04 | Editar Paciente con todos los campos. | Formulario datos paciente e id paciente. | Paciente modificado correctamente del sistema. |
| CP-05 | Editar Paciente con campo requerido no introducido. | Formulario datos paciente con un campo requerido no introducido e id paciente. | El sistema avisa de que el campo es requerido. |
| CP-06 | Eliminar Paciente. | Id paciente. | El sistema elimina al paciente del sistema. |

Cuadro 8.2: Caso de prueba: Editar y eliminar Paciente.

| ID | Descripción | Entrada | Resultado esperado |
|---|---|---|---|
| CP-07 | Pedir Cita normal. | Formulario datos cita. | Cita añadida correctamente al sistema. |
| CP-08 | Pedir cita urgente. | Formulario cita urgente. | Cita urgente añadida correctamente al sistema. |
| CP-09 | Pedir cita con campo requerido no introducido. | Formulario cita con dato requerido no introducido. | El sistema avisa de que el campo es requerido. |

Cuadro 8.3: Caso de prueba: Pedir Cita.

| ID | Descripción | Entrada | Resultado esperado |
|---|---|---|---|
| CP-10 | Editar Cita con todos los campos. | Formulario datos cita e id cita. | Cita modificada correctamente del sistema. |
| CP-11 | Editar Cita con campo requerido no introducido. | Formulario datos cita con un campo requerido no introducido e id cita. | El sistema avisa de que el campo es requerido. |
| CP-12 | Eliminar Cita. | Id cita. | El sistema elimina al paciente del sistema. |

Cuadro 8.4: Caso de prueba: Editar y eliminar Cita.

| ID | Descripción | Entrada | Resultado esperado |
|---|---|---|---|
| CP-13 | Confirmar Cita (administrativo). | Id paciente e id cita. | Cita confirmada. |
| CP-14 | Confirmar Cita (QR). | Id paciente. | Cita confirmada. |
| CP-15 | Confirmar Cita (*Telegram*). | Id cita e id paciente. | Cita confirmada. |

Cuadro 8.5: Caso de prueba: Confirmar Cita.

| ID | Descripción | Entrada | Resultado esperado |
|---|---|---|---|
| CP-16 | Reprogramar Citas a fecha nueva. | Fecha inicio, fecha fin e id citas. | Citas reubicadas a nuevas fechas y pacientes avisados. |
| CP-17 | Anular Citas. | Fecha inicio, fecha fin e id citas. | Citas anuladas y pacientes avisados. |

Cuadro 8.6: Caso de prueba: Reprogramación Citas.

| ID | Descripción | Entrada | Resultado esperado |
|---|---|---|---|
| CP-18 | Restaurar BBDD (datos correctos). | Copia de seguridad de la BBDD. | BBDD restaurada. |
| CP-19 | Restaurar BBDD (datos incorrectos). | Copia de seguridad de la BBDD con datos incorrectos. | El sistema no realiza ninguna acción. |

Cuadro 8.7: Caso de prueba: Restaurar BBDD.

| ID | Descripción | Entrada | Resultado esperado |
|---|---|---|---|
| CP-20 | Importar datos de archivo csv (datos correctos). | Archivo csv con los datos a importar. | Datos añadidos a la BBDD. |
| CP-21 | Importar datos de archivo csv (datos incorrectos). | Archivo csv con formato inadecuado. | El sistema no realiza ninguna acción. |

Cuadro 8.8: Caso de prueba: Importar datos de archivo csv.

```
jorge@jorge:~/TFG/Clinica$ python3 manage.py test
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.................
----------------------------------------------------------------------
Ran 17 tests in 1.317s

OK
Destroying test database for alias 'default'...
jorge@jorge:~/TFG/Clinica$
```

Figura 8.1: Resultado test *Django*.

## 8.2 Pruebas de caja blanca

Al contrario que las anteriores, las pruebas de caja blanca sí tienen en cuenta el código de la aplicación. Existen varios tipos de estas pruebas, entre ellas, las más destacadas son las pruebas unitarias y las pruebas de integración. Las primeras prueban un método concreto de una clase, de forma aislada al resto. Mientras que las segundas prueban el flujo del programa y si éste actúa en conjunto. [37]

Para este proyecto se han realizado una serie de pruebas intentando cubrir una buena parte del código. Para probar los test en *Django* se ejecuta el siguiente comando:

```
1 python3 manage.py test
```

Una vez ejecutado se puede ver el resultado del mismo en la figura 8.1.

En la figura 8.2 se puede ver las pruebas realizadas a uno de los modelos. Los modelos se crean con palabras reservadas de *Django*, por ese motivo no es necesario comprobar que los campos sean correctos. Normalmente se prueban los métodos creados pero los modelos de este proyecto no tienen métodos. Así que se decidió probar un par de detalles simplemente. El primero es ver si la etiqueta de un modelo es correcta, en este caso se mira si la etiqueta del campo paciente del modelo *Cita* es correcta o no. El segundo es para ver si el tamaño máximo del campo código es realmente 10 o no.

Las demás figuras muestran las distintas pruebas creadas para algunas de las vistas del proyecto. Como se puede observar (ver figura 8.3) se prueban los métodos de crear, editar y eliminar para médicos, pacientes, administrativos y clínicas. Después se prueba si se inicia y cierra sesión de manera correcta. Por último se mira si el fichaje de la jornada se hace correctamente. Para probar si una función va se mira si el código devuelto por la operación coincide con *HTTPStatus.FOUND* (que es el código 302). En caso de que la operación no haga lo que tiene que hacer, pero tampoco falle, devolvería *HTTPStatus.OK* (código 200).

En la figura 8.4 se ven las otras pruebas realizadas, en la aplicación de gestión de citas, para el proyecto. Las dos primeras pruebas sirven para ver si la conexión se realiza correctamente

```python
from django.test import TestCase
from django.contrib.auth.models import User
from GestionCitas.models import Cita
from GestionAdministrativa.models import Clinica, Paciente, Medico
from datetime import datetime, date, time, timedelta

class CitaModelTest(TestCase):

    @classmethod
    def setUpTestData(cls):
        clinica = Clinica.objects.create(nombre="Clínica Prueba", telefono=981205060,
            direccion="Avda Prueba", salas_disponibles=10)
        paciente = Paciente.objects.create(nombre="Pedro", apellidos="Prueba", email="a@a.com",
            telefono=981471052)
        usuario = User.objects.create(username="jds", password="asdfg1234")
        medico = Medico.objects.create(usuario=usuario, nombre="Alba", apellidos="López", email="a@a.com",
            especialidad="Medicina General", num_colegiado=151501496, clinica=clinica, sala=1)
        fecha = date.today().isoformat()
        Cita.objects.create(paciente=paciente, medico=medico, clinica=clinica, fecha=fecha,
            hora=1, confirmada=False, llamada_medico=False, atendido=False, sala=1, codigo="PeAl1")

    def test_paciente_label(self):
        cita=Cita.objects.get(id=1)
        field_label = cita._meta.get_field('paciente').verbose_name
        self.assertEquals(field_label,'paciente')

    def test_codigo_max_length(self):
        cita=Cita.objects.get(id=1)
        max_length = cita._meta.get_field('codigo').max_length
        self.assertEquals(max_length, 10)
```

Figura 8.2: Test *models.py* (Gestión Citas) de la aplicación *ClínicasMédicas*.

y para probar si la plantilla cargada en una función es la adecuada. A partir de ahí se prueban distintas funciones como en el caso de las de gestión administrativa.

Al finalizar la ejecución de las pruebas se ha hecho uso de un paquete de *Python* para ver la cobertura, es decir, para ver que porcentaje del código se ha cubierto con las pruebas realizadas.

```python
from django.test import TestCase
from django.urls import reverse
from http import HTTPStatus
from django.contrib.auth.models import User, Group
from GestionAdministrativa.models import Clinica

class ViewsTestCase(TestCase):
    @classmethod
    def setUpTestData(cls):
        cls.test_clinica = Clinica.objects.create(nombre="Clínica Prueba", telefono=981205060,
            direccion="Avda Prueba", salas_disponibles=10)
        cls.test_clinica.save()

        cls.test_group = Group.objects.create(name='Médico')
        cls.test_group.save()

        cls.test_group2 = Group.objects.create(name='Administrativo')
        cls.test_group2.save()

        cls.test_group3 = Group.objects.create(name='Paciente')
        cls.test_group3.save()


    def test_añadir_editar_y_borrar_medico(self):
        response = self.client.post(reverse('anhadir_medico'), data={'nombre': 'Pedro',
            'apellidos': 'Gonzalez', 'username': 'pg', 'email': 'jorge.diaz.seoane@udc.es',
            'especialidad': 'Medicica General', 'num_colegiado': 151501245, 'clinica': self.test_clinica.id,
            'sala': 1, 'password1': 'asdfg1234', 'password2': 'asdfg1234'})

        self.assertEqual(response.status_code, HTTPStatus.FOUND)

        response = self.client.post(reverse('editar_medico', args=[1]), data={'nombre': 'Paco',
            'apellidos': 'Alvarez', 'email': 'jorge.diaz.seoane@udc.es',
            'especialidad': 'Medicica General', 'num_colegiado': 151501245, 'clinica': self.test_clinica.id,
```

(a) Parte 1.

```python
        response = self.client.post(reverse('editar_medico', args=[1]), data={'nombre': 'Paco',
            'apellidos': 'Alvarez', 'email': 'jorge.diaz.seoane@udc.es',
            'especialidad': 'Medicica General', 'num_colegiado': 151501245, 'clinica': self.test_clinica.id,
            'sala': 1})

        self.assertEqual(response.status_code, HTTPStatus.FOUND)

        response = self.client.post(reverse('borrar_medico', args=[1]))

        self.assertEqual(response.status_code, HTTPStatus.FOUND)

    def test_añadir_editar_y_borrar_administrativo(self):
        response = self.client.post(reverse('anhadir_administrativo'), data={'nombre': 'Alba',
            'apellidos': 'Campos', 'username': 'ac', 'email': 'jorge.diaz.seoane@udc.es',
            'clinica': self.test_clinica.id, 'password1': 'asdfg1234', 'password2': 'asdfg1234'})

        self.assertEqual(response.status_code, HTTPStatus.FOUND)

        response = self.client.post(reverse('editar_administrativo', args=[1]), data={'nombre': 'Laura',
            'apellidos':'Campos', 'email': 'jorge.diaz.seoane@udc.es', 'clinica': self.test_clinica.id})

        self.assertEqual(response.status_code, HTTPStatus.FOUND)

        response = self.client.post(reverse('borrar_administrativo', args=[1]))

        self.assertEqual(response.status_code, HTTPStatus.FOUND)

    def test_añadir_editar_y_borrar_clinica(self):
        response = self.client.post(reverse('anhadir_clinica'), data={'direccion': 'C/ Cuba 1',
            'telefono': 981257410, 'nombre': 'Clinica Margarita', 'salas_disponibles': 10})

        self.assertEqual(response.status_code, HTTPStatus.FOUND)
```

(b) Parte 2.

```
        response = self.client.post(reverse('editar_clinica', args=[1]), data={'direccion': 'C/ Cuba 2',
            'telefono': 981257412, 'nombre': 'Clinica Girasol', 'salas_disponibles': 10})

        self.assertEqual(response.status_code, HTTPStatus.FOUND)

        response = self.client.post(reverse('borrar_clinica', args=[1]))

        self.assertEqual(response.status_code, HTTPStatus.FOUND)

    def test_añadir_editar_y_borrar_paciente(self):
        response = self.client.post(reverse('anhadir_paciente'), data={'nombre': 'Juan',
            'apellidos': 'Fernandez', 'username': 'jf', 'email': 'jorge.diaz.seoane@udc.es',
            'password1': 'asdfg1234', 'password2': 'asdfg1234'})

        self.assertEqual(response.status_code, HTTPStatus.FOUND)

        response = self.client.post(reverse('editar_paciente', args=[1]), data={'nombre': 'Pedro',
        'apellidos': 'Diaz', 'email': 'jorge.diaz.seoane@udc.es', 'telefono': 981247145})

        self.assertEqual(response.status_code, HTTPStatus.FOUND)

        response = self.client.post(reverse('borrar_paciente', args=[1]))

        self.assertEqual(response.status_code, HTTPStatus.FOUND)

    def test_inicio_sesion(self):
        self.user = User.objects.create_user(username='jds', password='asdfg1234')
        response = self.client.post(reverse('inicio_sesion'), data={'username': 'jds',
            'password': 'asdfg1234', 'remember_me': True})
        self.assertEqual(response.status_code, HTTPStatus.FOUND)
```

(c) Parte 3.

```
    def test_fin_sesion(self):
        self.user = User.objects.create_user(username='testuser', password='12345')
        self.client.login(username='testuser', password='12345')
        response = self.client.post(reverse('fin_sesion'))
        self.assertEqual(response.status_code, HTTPStatus.FOUND)

    def test_fichaje_jornada(self):
        self.user = User.objects.create_user(username='testuser', password='12345')
        self.client.login(username='testuser', password='12345')

        response = self.client.post(reverse('fichar_entrada'))
        self.assertEqual(response.status_code, HTTPStatus.FOUND)

        response = self.client.post(reverse('fichar_salida'))
        self.assertEqual(response.status_code, HTTPStatus.FOUND)
```

(d) Parte 4.

Figura 8.3: Test *views.py* (Gestion Administrativa) de la aplicación *ClínicasMédicas*.

```python
from django.test import TestCase
from django.urls import reverse
from http import HTTPStatus
from django.contrib.auth.models import User
from GestionCitas.models import Cita
from GestionAdministrativa.models import Clinica, Paciente, Medico
from datetime import datetime, date, time, timedelta


class ViewsTestCase(TestCase):

    @classmethod
    def setUpTestData(cls):
        cls.test_clinica = Clinica.objects.create(nombre="Clínica Prueba", telefono=981205060,
            direccion="Avda Prueba", salas_disponibles=10)
        cls.test_clinica.save()

        cls.test_paciente = Paciente.objects.create(nombre="Pedro", apellidos="Prueba", email="a@a.com",
            telefono=981471052)
        cls.test_paciente.save()

        usuario = User.objects.create(username="jds", password="asdfg1234")

        cls.test_medico = Medico.objects.create(usuario=usuario, nombre="Alba", apellidos="López",
            email="jorge.diaz.seoane@udc.es", especialidad="Medicina General", num_colegiado=151501496,
            clinica=cls.test_clinica, sala=1)
        cls.test_medico.save()

        fecha = date.today().isoformat()

        cls.test_cita = Cita.objects.create(paciente=cls.test_paciente, medico=cls.test_medico, clinica=cls.test_clinica
            fecha=fecha, hora=1, confirmada=False, llamada_medico=False, atendido=False, sala=1,
            codigo="PeAl1")
        cls.test_cita.save()
```

(a) Parte 1.

```python
        cls.test_cita2 = Cita.objects.create(paciente=cls.test_paciente, medico=cls.test_medico, clinica=cls.test_clinica,
            fecha=fecha, hora=2, confirmada=False, llamada_medico=False, atendido=False, sala=1,
            codigo="PeAl2")
        cls.test_cita2.save()

    def test_probar_conexion(self):
        response = self.client.get('/')
        self.assertEqual(response.status_code, 200)

    def test_plantilla_correcta(self):
        response = self.client.get(reverse('listar_citas', args=[self.test_paciente.id]))
        self.assertTemplateUsed(response, 'GestionCitas/listar_citas.html')

    def test_confirmar_cita(self):
        response = self.client.post(reverse('confirmar_cita', args=[self.test_cita.id]))
        self.test_cita.refresh_from_db()
        self.assertEqual(self.test_cita.confirmada, True)

    def test_confirmar_llamada_medico(self):
        response = self.client.post(reverse('llamada_medico', args=[self.test_cita.id]))
        self.test_cita.refresh_from_db()
        self.assertEqual(self.test_cita.llamada_medico, True)

    def test_atendido(self):
        response = self.client.post(reverse('atendido', args=[self.test_cita.id]))
        self.test_cita.refresh_from_db()
        self.assertEqual(self.test_cita.atendido, True)

    def test_añadir_editar_cita(self):
        fecha = date.today().isoformat()
        response = self.client.post(reverse('anhadir_cita', args=[self.test_paciente.id]),
            data={'medico': self.test_medico.id, 'fecha': fecha})

        self.assertEqual(response.status_code, HTTPStatus.OK)
```

(b) Parte 2.

```python
        response = self.client.post(reverse('editar_cita', args=[1]), data={'medico': self.test_medico.id,
            'fecha': fecha})

        self.assertEqual(response.status_code, HTTPStatus.OK)


    def test_borrar_cita(self):
        response = self.client.post(reverse('borrar_cita' , args=[self.test_cita.id]))
        self.assertEqual(response.status_code, HTTPStatus.FOUND)

    def test_cita_correlacionada(self):
        response = self.client.post(reverse('cita_correlacionada', args=[self.test_cita2.id]),
            data={'cita_correlacionada': self.test_cita.id})

        self.assertEqual(response.status_code, HTTPStatus.FOUND)
```

(c) Parte 3.



Figura 8.4: Test *views.py* (Gestion Citas) de la aplicación *ClínicasMédicas*.


(a) Terminal.

| Total | 1296 | 389 | 0 | 70% |
|---|---|---|---|---|

(b) HTML.

Figura 8.5: Cobertura test.

```python
def añadir_paciente(request):
    form = PacienteForm()

    if request.method == "POST":
        form = PacienteForm(data=request.POST)

        if form.is_valid():
            nombre = form.cleaned_data['nombre']
            apellidos = form.cleaned_data['apellidos']
            email = form.cleaned_data['email']
            telefono = form.cleaned_data['telefono']

            paciente = Paciente(nombre=nombre, apellidos=apellidos, email=email, telefono=telefono)
            paciente.save()

            return redirect('/administracion/gestion_administrativa')

    return render (request, 'GestionAdministrativa/anhadir_paciente.html', {'form': form})
```

(a) Función probada.

```python
def listar_clinica(request):
    clinicas = Clinica.objects.all()

    if request.POST.get("nombre"):
        clinicas = clinicas.filter(nombre__icontains=request.POST.get("nombre"))

    return render(request, 'GestionAdministrativa/listar_clinica.html', {'clinicas': clinicas})
```

(b) Función no probada.

Figura 8.6: Funciones probadas y no probadas con un test.

```
coverage run --source='.' manage.py test

coverage report
coverage html
```

El primer comando es el que realiza la acción de mirar la cobertura de los test realizados. Los dos siguientes sirven para ver dicha cobertura. El primero lo muestra por terminal (ver figura 8.5a) y el segundo genera un archivo .html para ver desde un navegador (ver figura 8.5b). Éste último tiene la ventaja de que permite ir viendo en el código si una función ha sido probado por los test o no (ver figuras 8.6a y 8.6b).

## 8.3 Pruebas de aceptación

Las pruebas de aceptación sirven para ver si el cliente está conforme con el trabajo realizado, para ello éste lo prueba y da sus impresiones. Como en un TFG no hay cliente, se ha seleccionado a un conjunto de personas para probar la aplicación haciendo uso del manual de usuario y sin que el desarrollador (el autor de este proyecto) intervenga.

Las pruebas se han realizado en dos fases.

### 8.3.1 Fase 1: pruebas iniciales

Para esta fase inicial se han seleccionado dos personas. A continuación indicamos las impresiones de cada una

- **Persona 1**

  En líneas generales le pareció una aplicación fácil e intuitiva de usar pero refirió una serie de mejoras que se podían hacer en el aspecto visual y en algunas funciones. Más concretamente:

  - En la parte de Preferencias (rol paciente) la aplicación debería volver automáticamente para la vista anterior (Área Personal).
  - A la hora de pedir cita se debería mostrar la especialidad y la clínica en la que atiende el médico.
  - Cuando se muestra la tabla con las citas debería aparecer la especialidad del médico.
  - Mejoras visuales en las vistas de Área Personal, Gestión Médica y Gestión Administrativa.
  - En la vista de Cuadro Médico poner un desplegable con las especialidades en el filtro, en lugar de un cuadro para escribir. En esta misma vista, en la tabla, poner la clínica donde atiende el médico.
  - Que las acciones a realizar con el rol de médico sea más sencillo. Comenta que es algo tedioso a diferencia de los otros dos roles, donde es más sencillo e intuitivo.

  Dado que estos cambios son relativamente sencillos de implementar y que se dispone del tiempo para ello, se ha procedido a realizarlos.

- **Persona 2**

  Notó los mismos problemas y mejoras que la primera persona. Lo que da a entender que son fallos por parte del desarrollador y no apreciaciones subjetivas de las personas que probaron esta aplicación.

### 8.3.2 Fase 2 o Rectificación

Una vez realizadas las mejoras propuestas, se les ha vuelto a pedir que prueben la aplicación. En esta nueva prueba, ambas han estado conformes con los cambios realizados.

A posteriori, con los cambios anteriores realizados, se le pidió a 5 personas más que la probasen. No hubo ninguna apreciación que requiera un cambio en la aplicación. Por este motivo, se puede considerar que la aplicación es intuitiva y sencilla de utilizar.

Capítulo 9

# Conclusiones y Líneas futuras

En este capítulo se verán las mejoras que se pueden implementar y las conclusiones sacadas en la realización del proyecto.

## 9.1 Conclusiones

Al inicio del proyecto se plantearon una serie de objetivos que debería cumplir (ver apartado 1.2). Se puede afirmar, una vez finalizado, que dichos objetivos se han cumplido en su totalidad. El tiempo de realización del mismo fue el estimado en la planificación inicial, aunque con los cambios propuestos en las reuniones con el tutor, los cuales no implicaron un aumento de tiempo, debido a que la planificación inicial fue sobrestimada para tenerlos en cuenta.

Se ha realizado una aplicación funcional que cubre de manera correcta la gestión de citas para una red de clínicas médicas, que era el objetivo principal de este proyecto. Tiene la limitación de no incluir ni gestión económica ni gestión de informes médicos. En el caso de esto último fue por temas de seguridad y ley de protección de datos, al incluir datos sensibles de los pacientes.

Se encontraron dificultades con el tema horario de las citas. La reprogramación de citas también fue un elemento crucial en este trabajo al tener que moverlas a nuevas horas y fechas pero se logró realizar correctamente.

Al inicio del proyecto se estuvo pensando en que herramientas utilizar para su desarrollo. Finalmente se decidió usar *Django*, del cual el autor tenía unos conocimientos mínimos debido a la materia de *Programación Integrativa*. Con el proyecto finalizado, se puede afirmar que ha sido un acierto su uso, debido a que facilita varias de las tareas en el desarrollo de una aplicación web y hace uso de *Python*, el cual también facilita la programación.

La realización de este TFG permitió al autor profundizar en diversos aspectos vistos durante la carrera: gestión de proyectos, análisis de requisitos, diseño, uso del framework *Django*,

mejorar los conocimientos de *HTML*, CSS y *bootstrap*, aprender sobre *JavaScript* y la metodología *Scrum*. En definitiva, aprender y afianzar los conceptos necesarios para elaborar un proyecto informático desde cero, así como la experiencia necesaria para ello.

## 9.2 Líneas futuras

Las principales mejoras que se pueden implementar en un futuro como ampliación de este proyecto son las siguientes:

- Cuando el paciente llega a la clínica y confirma la cita, mediante el *QR*, se le podría llamar por *SMS*, en lugar de llamarlo por la pantalla de la sala de espera.
- Debido al alcance de este proyecto, no se tuvo en cuenta la integración de la aplicación con una aplicación para la generación y mantenimiento de informes médicos ni tampoco el apartado de la gestión económica. Gracias al uso de *Django*, esto se podría hacer con relativa facilidad mediante la inclusión de las aplicaciones en el proyecto.
- Cuando se llama al paciente para que acuda a la sala del médico, se le podría indicar como llegar mediante el uso de algún mapa en línea (*Google Maps* por ejemplo).
- El *bot* de *Telegram* está hecho de forma unidireccional (del sistema al paciente), sería interesante modificarlo para que sea bidireccional, es decir, para que el paciente pueda mandarle comandos y que el sistema responda a las peticiones del mismo. Por ejemplo, que el paciente solicite una lista con las citas que tiene mediante un comando y el *bot* se las envíe.

# Apéndices

## Apéndice A

# Casos de uso

En este anexo se verán los casos de uso que tiene la aplicación *ClínicasMédicas*.

| CU-01 | Iniciar Sesión | |
|---|---|---|
| Descripción | El usuario inicia sesión en el sistema | |
| Precondición | El usuario no inició sesión pero está registrado en el sistema. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El sistema solicita al usuario que introduzca sus credenciales. |
| | 2 | El usuario introduce su nombre de usuario y contraseña. |
| | 3 | El sistema comprueba que las credenciales sean correctas. Después redirige al usuario a una dirección concreta en función del rol que tenga. |
| Errores / Alternativas | Paso | Acción |
| | 3' | Si el usuario falla en uno de los campos, el sistema no permite iniciar sesión y le pide al usuario que vuelva a introducirlos. El sistema vuelve al paso 1 |
| Postcondición | El usuario inicia sesión en el sistema | |

Cuadro A.1: Caso de uso: Iniciar Sesión.

| CU-02 | Registrarse | |
|---|---|---|
| Descripción | El usuario se registra en el sistema. | |
| Precondición | El usuario no está registrado en el sistema. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El sistema solicita una serie de datos (Nombre, Apellido, Nombre de usuario, Email, Contraseña, Confirmar contraseña). |
| | 2 | El usuario introduce los datos solicitados por el sistema. |
| | 3 | El sistema registra al usuario. |
| Errores / Alternativas | Paso | Acción |
| | 3' | Si el usuario falla en uno de los campos, el sistema le muestra un error pidiéndole que vuelva a introducir el dato. El sistema vuelve al paso 1. |
| Postcondición | El usuario queda registrado en el sistema. | |

Cuadro A.2: Caso de uso: Registrarse.

| CU-03 | Contacto | |
|---|---|---|
| Descripción | El usuario se pone en contacto con la clínica. | |
| Precondición | No | |
| Secuencia Principal | Paso | Acción |
| | 1 | El sistema solicita una serie de datos (Asunto, Email y Mensaje) y una casilla para Aceptar el envío. |
| | 2 | El usuario introduce los datos solicitados por el sistema. |
| | 3 | El sistema envía un e-mail con la información proporcionada. |
| Errores / Alternativas | Paso | Acción |
| | 2' | Si el usuario no completa uno de los campos, el sistema le muestra un error pidiéndole que lo introduzca. El sistema vuelve al paso 1. |
| Postcondición | No | |

Cuadro A.3: Caso de uso: Contacto.

| CU-04 | Ver cuadro médico | |
|---|---|---|
| Descripción | El usuario mira el cuadro médico de la clínica. | |
| Precondición | No | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Cuadro Médico. |
| | 2 | El sistema muestra la información del cuadro médico mediante una tabla con los siguientes campos: Doctor, Especialidad, Número de Colegiado, Clínica. También ofrece la posibilidad de filtrar por Especialidad. |
| Errores / Alternativas | Paso | Acción |
| | 2' | Si el usuario filtra por Especialidad, el sistema actualiza la tabla mostrando solo los médicos de dicha especialidad. |
| Postcondición | No | |

Cuadro A.4: Caso de uso: Ver cuadro médico.

| CU-05 | Pedir cita | |
|---|---|---|
| Descripción | El usuario pide una cita para ser revisado por un médico. | |
| Precondición | El usuario, paciente, debe estar registrado. En caso de no estarlo, tendrá que solicitar la cita mediante un administrativo. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Pedir Cita. |
| | 2 | El sistema pide introducir unos datos (Médico y Fecha). |
| | 3 | El usuario introduce los datos solicitados. |
| | 4 | El sistema comprueba las horas disponibles para ese Médico en esa Fecha. Muestra las horas disponibles. |
| | 5 | El usuario escoge una hora de las disponibles. |
| | 6 | El sistema registra la cita, envía un email al usuario y manda un aviso por *Telegram*. |
| Errores / Alternativas | Paso | Acción |
| | 2' | Si el usuario es médico o administrativo, tiene un recuadro para forzar una cita. |
| | 6' | Si el usuario no está registrado, y es el administrativo el que registra la cita, no se envía ningún email. Si no tiene registrado un *chatid* de *Telegram*, no se envía ningún mensaje. |
| Postcondición | El sistema registra la cita del usuario. | |

Cuadro A.5: Caso de uso: Pedir cita.

| CU-06 | Eliminar cita | |
|---|---|---|
| Descripción | El usuario elimina una cita. | |
| Precondición | El usuario, paciente, debe estar registrado. En caso de no estarlo, tendrá que ponerse en contacto con un administrativo para que se la elimine. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Área Personal. Después lo hace sobre Ver Citas. |
| | 2 | El sistema muestra una tabla con los siguientes campos: Paciente, Médico, Clínica, Sala, Fecha, Hora y Acciones (con 2 acciones: Editar y Eliminar). |
| | 3 | El usuario hace clic sobre Eliminar. |
| | 4 | El sistema pregunta al usuario si está seguro de querer borrarla. |
| | 5 | El usuario pincha sobre el sí. |
| | 6 | El sistema elimina la cita y redirige al usuario a la tabla del Paso 2. |
| Errores / Alternativas | Paso | Acción |
| | 5' | Si el usuario pincha sobre el No, el sistema no elimina la cita y redirige al usuario al Paso 2. |
| Postcondición | El sistema elimina la cita del usuario. | |

Cuadro A.6: Caso de uso: Eliminar cita.

| CU-07 | Editar cita | |
|---|---|---|
| Descripción | El usuario modifica una cita. | |
| Precondición | El usuario, paciente, debe estar registrado. En caso de no estarlo, tendrá que ponerse en contacto con un administrativo para que se la modifique. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Área Personal. Después lo hace sobre Ver Citas |
| | 2 | El sistema muestra una tabla con los siguientes campos: Paciente, Médico, Clínica, Sala, Fecha, Hora y Acciones (con 2 acciones: Editar y Eliminar). |
| | 3 | El usuario hace clic sobre Editar. |
| | 4 | El sistema muestra los datos actuales de la cita (Médico y Fecha) y permite su modificación. |
| | 5 | El usuario modifica los datos. |
| | 6 | El sistema comprueba las horas disponibles para ese Médico en esa Fecha. Muestra las horas disponibles. |
| | 7 | El usuario escoge una de las horas disponibles. |
| | 8 | El sistema registra la cita y envía un email al usuario. |
| Errores / Alternativas | Paso | Acción |
| | 8' | Si el usuario no está registrado, y es el administrativo el que modifica la cita, no se envía ningún email. |
| Postcondición | El sistema modifica la cita del usuario. | |

Cuadro A.7: Caso de uso: Editar cita.

| CU-08 | Recuperar contraseña | |
|---|---|---|
| Descripción | El usuario quiere recuperar su contraseña porque no se acuerda. | |
| Precondición | El usuario debe estar registrado. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Login. Después lo hace sobre Recuperar Contraseña. |
| | 2 | El sistema solicita que introduzca un dato (Correo electrónico). |
| | 3 | El usuario introduce el dato solicitado. |
| | 4 | El sistema comprueba que el usuario esté registrado y le envía un email para restablecer la contraseña. |
| | 5 | El usuario accede, desde el email, a la dirección. |
| | 6 | El sistema solicita que se introduzca una nueva contraseña y la confirmación de la misma. |
| | 7 | El usuario introduce los datos solicitados. |
| | 8 | El sistema guarda la nueva contraseña. |
| Errores / Alternativas | Paso | Acción |
| | 4' | Si el usuario no está registrado, el sistema da un error y vuelve al paso 2. |
| Postcondición | El sistema modifica la contraseña del usuario. | |

Cuadro A.8: Caso de uso: Recuperar contraseña.

| CU-09 | Modificar datos | |
|---|---|---|
| Descripción | El usuario quiere modificar sus datos personales. | |
| Precondición | El usuario debe estar registrado. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Área personal. Después lo hace sobre Modificar Datos. |
| | 2 | El sistema muestra los siguientes datos (con sus valores actuales): Nombre, Apellidos, Email y Teléfono. |
| | 3 | El usuario modifica los datos mostrados. |
| | 4 | El sistema guarda los nuevos datos. |
| Errores / Alternativas | Paso | Acción |
| | 2' | En este paso también es posible hacer clic sobre modificar contraseña. El sistema muestra una nueva pantalla donde el usuario introduce su contraseña actual y 2 veces la nueva. El sistema guarda la contraseña nueva. |
| Postcondición | El sistema modifica los datos del usuario. | |

Cuadro A.9: Caso de uso: Modificar datos.

| CU-10 | Ver agenda médica | |
|---|---|---|
| Descripción | El usuario, médico, quiere ver la agenda. | |
| Precondición | El usuario debe estar registrado. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Ver agenda del día. |
| | 2 | El sistema solicita que introduzca una Fecha inicial. |
| | 3 | El usuario introduce el dato solicitado. |
| | 4 | El sistema solicita que introduzca una Fecha final. |
| | 5 | El usuario introduce el dato solicitado. |
| | 6 | El sistema muestra una tabla con los siguientes datos: Paciente, Hora, Fecha, Confirmada, Llamada, Atendido. |
| Errores / Alternativas | No | |
| Postcondición | No | |

Cuadro A.10: Caso de uso: Ver agenda médica.

| CU-11 | Fichar entrada/salida jornada laboral | |
|---|---|---|
| Descripción | El usuario, médico o administrativo, ficha la entrada/salida de la jornada laboral. | |
| Precondición | El usuario debe estar registrado. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Login. |
| | 2 | El sistema ficha la entrada automáticamente cuando se inicia sesión. |
| | 3 | El usuario hace clic sobre Cerrar Sesión. |
| | 4 | El sistema ficha la salida automáticamente cuando se cierra sesión. |
| Errores / Alternativas | No | |
| Postcondición | El sistema guarda la información de la entrada/salida del usuario. | |

Cuadro A.11: Caso de uso: Fichar entrada/salida jornada laboral.

| CU-12 | Llamar paciente | |
|---|---|---|
| Descripción | El usuario, médico, llama al paciente para que acuda a la sala. | |
| Precondición | El usuario, paciente, debe haber confirmado su asistencia. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Confirmar en el campo Llamada de la tabla que muestra la Agenda médica (CU-9 A.10). |
| | 2 | El sistema manda un aviso a la tabla de la Sala de Espera (CU-14 A.14) para llamar al paciente. |
| Errores / Alternativas | Paso | Acción |
| | 2' | Si el usuario tiene añadido el *bot* de *Telegram*, y en Preferencias permite el envío de mensajes por este medio, el sistema envía un aviso de la llamada y la sala a la que tiene que acudir. |
| Postcondición | Sale un aviso para el paciente en la sala de espera. | |

Cuadro A.12: Caso de uso: Llamar paciente.

| CU-13 | Paciente visto | |
|---|---|---|
| Descripción | El usuario, médico, confirma que ha visto al paciente. | |
| Precondición | El usuario, paciente, debe haber confirmado su asistencia y debe haber sido llamado por el médico. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Confirmar en el campo Atendido de la tabla que muestra la Agenda médica (CU-9 A.10). |
| | 2 | El sistema elimina la llamada de la tabla de la Sala de Espera (CU-14 A.14). |
| Errores / Alternativas | Paso | Acción |
| | 2' | Si el usuario tiene registrado el *chatid* de *Telegram*, el sistema le envía un mensaje por este medio. |
| Postcondición | Se elimina el aviso para el paciente en la sala de espera. | |

Cuadro A.13: Caso de uso: Paciente visto.

| CU-14 | Sala espera | |
|---|---|---|
| Descripción | Pantalla de la sala de espera donde los pacientes ven cuando son llamados. | |
| Precondición | El usuario, paciente, debe haber confirmado su asistencia. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El sistema muestra una tabla con el paciente y la sala a la que debe acudir. |
| Errores / Alternativas | No | |
| Postcondición | Cuando un paciente ha sido atendido por el médico, se elimina el aviso para el paciente en la sala de espera. | |

Cuadro A.14: Caso de uso: Sala de espera.

| CU-15 | Añadir médico | |
|---|---|---|
| Descripción | El usuario, administrativo, añade un médico al sistema. | |
| Precondición | El usuario debe estar registrado y el médico no debe existir. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Añadir en el recuadro Médico. |
| | 2 | El sistema solicita los siguientes datos: Nombre, Apellidos, Nombre de usuario, Email, Especialidad, Número de colegiado, Clínica, Sala, Contraseña y Confirmar Contraseña. |
| | 3 | El usuario proporciona los datos solicitados. |
| | 4 | El sistema registra al médico en el sistema. |
| Errores / Alternativas | Paso | Acción |
| | 4' | Si el usuario falla en alguno de los campos, el sistema le muestra un error pidiéndole que vuelva a introducir el dato. El sistema vuelve al paso 2. |
| Postcondición | El médico queda registrado en el sistema. | |

Cuadro A.15: Caso de uso: Añadir médico.

| CU-16 | Añadir administrativo | |
|---|---|---|
| Descripción | El usuario, administrativo, añade un administrativo al sistema. | |
| Precondición | El usuario debe estar registrado y el administrativo que añade no debe existir. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Añadir en el recuadro Administrativo. |
| | 2 | El sistema solicita los siguientes datos: Nombre, Apellidos, Nombre de usuario, Email, Clínica, Contraseña y Confirmar Contraseña. |
| | 3 | El usuario proporciona los datos solicitados. |
| | 4 | El sistema registra al administrativo en el sistema. |
| Errores / Alternativas | Paso | Acción |
| | 4' | Si el usuario falla en alguno de los campos, el sistema le muestra un error pidiéndole que vuelva a introducir el dato. El sistema vuelve al paso 2. |
| Postcondición | El administrativo queda registrado en el sistema. | |

Cuadro A.16: Caso de uso: Añadir administrativo.

| CU-17 | Añadir paciente | |
|---|---|---|
| Descripción | El usuario, administrativo, añade un paciente al sistema. Este será el método para añadir pacientes que no se registren ellos mismos y no requiere que tengan email. | |
| Precondición | El usuario debe estar registrado y el paciente no debe existir. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Añadir en el recuadro Paciente. |
| | 2 | El sistema solicita los siguientes datos: Nombre, Apellidos, Email, Teléfono. |
| | 3 | El usuario proporciona los datos solicitados. |
| | 4 | El sistema registra al paciente en el sistema. |
| Errores / Alternativas | Paso | Acción |
| | 4' | Si el usuario falla en alguno de los campos, el sistema le muestra un error pidiéndole que vuelva a introducir el dato. El sistema vuelve al paso 2. |
| Postcondición | El paciente queda registrado en el sistema. | |

Cuadro A.17: Caso de uso: Añadir paciente.

| CU-18 | Añadir clínica | |
|---|---|---|
| Descripción | El usuario, administrativo, añade una clínica al sistema. | |
| Precondición | El usuario debe estar registrado y la clínica no debe existir. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Añadir en el recuadro Clínica. |
| | 2 | El sistema solicita los siguientes datos: Dirección, Teléfono, Nombre, Salas disponibles. |
| | 3 | El usuario proporciona los datos solicitados. |
| | 4 | El sistema registra la clínica en el sistema. |
| Errores / Alternativas | Paso | Acción |
| | 4' | Si el usuario falla en alguno de los campos, el sistema le muestra un error pidiéndole que vuelva a introducir el dato. El sistema vuelve al paso 2. |
| Postcondición | La clínica queda registrado en el sistema. | |

Cuadro A.18: Caso de uso: Añadir clínica.

| CU-19 | Editar médico | |
|---|---|---|
| Descripción | El usuario, administrativo, edita los datos de un médico en el sistema. | |
| Precondición | El usuario debe estar registrado y el médico debe existir. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Buscar en el recuadro Médico. |
| | 2 | El sistema muestra un recuadro para filtrar por el nombre del médico. |
| | 3 | El usuario proporciona el dato solicitado. |
| | 4 | El sistema muestra una tabla con los siguientes datos: Usuario, Doctor, Especialidad, Nº Colegiado, Clínica, Sala y Acciones (con 3 opciones: Agenda, Editar, Eliminar). |
| | 5 | El usuario hace clic sobre Editar. |
| | 6 | El sistema muestra los datos actuales del médico y permite su modificación. |
| | 7 | El usuario modifica los datos que considere. |
| | 8 | El sistema guarda los datos modificados del médico. |
| Errores / Alternativas | Paso | Acción |
| | 4' | Si el usuario introduce un nombre de médico que no esté en el sistema, se muestra un mensaje de error "Médico no encontrado". |
| | 8' | Si el usuario falla en alguno de los campos, el sistema le muestra un error pidiéndole que vuelva a introducir el dato. El sistema vuelve al paso 6. |
| Postcondición | Los datos del médico quedan actualizados en el sistema. | |

Cuadro A.19: Caso de uso: Editar médico.

| CU-20 | Editar clínica | |
|---|---|---|
| Descripción | El usuario, administrativo, edita los datos de una clínica en el sistema. | |
| Precondición | El usuario debe estar registrado y la clínica debe existir. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Listar en el recuadro Clínica. |
| | 2 | El sistema muestra una tabla con los siguientes datos: Nombre, Dirección, Teléfono, Salas disponibles (Total) y Acciones (con 2 opciones:Editar, Eliminar). |
| | 3 | El usuario hace clic sobre Editar. |
| | 4 | El sistema muestra los datos actuales de la clínica y permite su modificación. |
| | 5 | El usuario modifica los datos que considere. |
| | 6 | El sistema guarda los datos modificados de la clínica. |
| Errores / Alternativas | Paso | Acción |
| | 2' | El sistema proporciona un recuadro para filtrar por el nombre de la clínica. |
| | 6' | Si el usuario falla en alguno de los campos, el sistema le muestra un error pidiéndole que vuelva a introducir el dato. El sistema vuelve al paso 4. |
| Postcondición | Los datos de la clínica quedan actualizados en el sistema. | |

Cuadro A.20: Caso de uso: Editar clínica.

| CU-21 | Editar administrativo | |
|---|---|---|
| Descripción | El usuario, administrativo, edita los datos de un administrativo en el sistema. | |
| Precondición | El usuario debe estar registrado y el administrativo debe existir. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Listar en el recuadro Administrativo. |
| | 2 | El sistema muestra una tabla con los siguientes datos: Nombre, Apellidos, Email, Clínica y Acciones (con 2 opciones:Editar, Eliminar). |
| | 3 | El usuario hace clic sobre Editar. |
| | 4 | El sistema muestra los datos actuales del administrativo y permite su modificación. |
| | 5 | El usuario modifica los datos que considere. |
| | 6 | El sistema guarda los datos modificados del administrativo. |
| Errores / Alternativas | Paso | Acción |
| | 2' | El sistema proporciona un recuadro para filtrar por el nombre del administrativo. |
| | 6' | Si el usuario falla en alguno de los campos, el sistema le muestra un error pidiéndole que vuelva a introducir el dato. El sistema vuelve al paso 4. |
| Postcondición | Los datos del administrativo quedan actualizados en el sistema. | |

Cuadro A.21: Caso de uso: Editar administrativo.

| CU-22 | Editar paciente | |
|---|---|---|
| Descripción | El usuario, administrativo, edita los datos de un paciente en el sistema. | |
| Precondición | El usuario debe estar registrado y el paciente debe existir. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Buscar en el recuadro Paciente. |
| | 2 | El sistema muestra un campo para filtrar por el nombre del paciente. |
| | 3 | El usuario proporciona el dato solicitado. |
| | 4 | El sistema muestra una tabla con los siguientes datos: Nombre, Apellidos, Teléfono, Correo Electrónico y Acciones (con 3 opciones: Ver citas, Editar, Eliminar). |
| | 5 | El usuario hace clic sobre Editar. |
| | 6 | El sistema muestra los datos actuales del paciente y permite su modificación. |
| | 7 | El usuario modifica los datos que considere. |
| | 8 | El sistema guarda los datos modificados del paciente. |
| Errores / Alternativas | Paso | Acción |
| | 4’ | Si el usuario introduce un nombre de paciente que no esté en el sistema, se muestra un mensaje de error ”Paciente no encontrado”. |
| | 8’ | Si el usuario falla en alguno de los campos, el sistema le muestra un error pidiéndole que vuelva a introducir el dato. El sistema vuelve al paso 6. |
| Postcondición | Los datos del paciente quedan actualizados en el sistema. | |

Cuadro A.22: Caso de uso: Editar paciente.

| CU-23 | Eliminar médico | |
|---|---|---|
| Descripción | El usuario, administrativo, elimina un médico en el sistema. | |
| Precondición | El usuario debe estar registrado y el médico debe existir. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Buscar en el recuadro Médico. |
| | 2 | El sistema muestra un recuadro para filtrar por el nombre del médico. |
| | 3 | El usuario proporciona el dato solicitado. |
| | 4 | El sistema muestra una tabla con los siguientes datos: Usuario, Doctor, Especialidad, Nº Colegiado, Clínica, Sala y Acciones (con 3 opciones: Agenda, Editar, Eliminar). |
| | 5 | El usuario hace clic sobre Eliminar. |
| | 6 | El sistema muestra un mensaje para confirmar que se desea eliminar al médico. |
| | 7 | El usuario pincha sobre el Sí. |
| | 8 | El sistema elimina al médico. |
| Errores / Alternativas | Paso | Acción |
| | 4’ | Si el usuario introduce un nombre de médico que no esté en el sistema, se muestra un mensaje de error ”Médico no encontrado”. |
| | 8’ | Si el usuario pincha sobre el No, el médico no es eliminado. El sistema vuelve al paso 4. |
| Postcondición | El médico es eliminado del sistema. | |

Cuadro A.23: Caso de uso: Eliminar médico.

| CU-24 | Eliminar paciente | |
|---|---|---|
| Descripción | El usuario, administrativo, elimina un paciente en el sistema. | |
| Precondición | El usuario debe estar registrado y el paciente debe existir. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Buscar en el recuadro Paciente. |
| | 2 | El sistema muestra un recuadro para filtrar por el nombre del paciente. |
| | 3 | El usuario proporciona el dato solicitado. |
| | 4 | El sistema muestra una tabla con los siguientes datos: Nombre, Apellidos, Teléfono, Correo Electrónico y Acciones (con 3 opciones: Ver citas, Editar, Eliminar). |
| | 5 | El usuario hace clic sobre Eliminar. |
| | 6 | El sistema muestra un mensaje para confirmar que se desea eliminar al paciente. |
| | 7 | El usuario pincha sobre el Sí. |
| | 8 | El sistema elimina al paciente. |
| Errores / Alternativas | Paso | Acción |
| | 4’ | Si el usuario introduce un nombre de paciente que no esté en el sistema, se muestra un mensaje de error ”Paciente no encontrado”. |
| | 8’ | Si el usuario pincha sobre el No, el paciente no es eliminado. El sistema vuelve al paso 4. |
| Postcondición | El paciente es eliminado del sistema. | |

Cuadro A.24: Caso de uso: Eliminar paciente.

| CU-25 | Eliminar administrativo | |
|---|---|---|
| Descripción | El usuario, administrativo, elimina un administrativo en el sistema. | |
| Precondición | El usuario debe estar registrado y el administrativo debe existir. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Listar en el recuadro Administrativo. |
| | 2 | El sistema muestra una tabla con los siguientes datos: Nombre, Apellidos, Email, Clínica y Acciones (con 3 opciones: Agenda, Editar, Eliminar). |
| | 3 | El usuario hace clic sobre Eliminar. |
| | 4 | El sistema muestra un mensaje para confirmar que se desea eliminar al administrativo. |
| | 5 | El usuario pincha sobre el Sí. |
| | 6 | El sistema elimina al administrativo. |
| Errores / Alternativas | Paso | Acción |
| | 2’ | El sistema proporciona un recuadro para filtrar por el nombre del administrativo. |
| | 6’ | Si el usuario pincha sobre el No, el administrativo no es eliminado. El sistema vuelve al paso 4. |
| Postcondición | El administrativo es eliminado del sistema. | |

Cuadro A.25: Caso de uso: Eliminar administrativo.

| CU-26 | Eliminar clínica | |
|---|---|---|
| Descripción | El usuario, administrativo, elimina una clínica en el sistema. | |
| Precondición | El usuario debe estar registrado y la clínica debe existir. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Listar en el recuadro Clínica. |
| | 2 | El sistema muestra una tabla con los siguientes datos: Nombre,Dirección, Teléfono, Salas disponibles (Total) y Acciones (con 2 opciones: Editar, Eliminar). |
| | 3 | El usuario hace clic sobre Eliminar. |
| | 4 | El sistema muestra un mensaje para confirmar que se desea eliminar al administrativo. |
| | 5 | El usuario pincha sobre el Sí. |
| | 6 | El sistema elimina la clínica. |
| Errores / Alternativas | Paso | Acción |
| | 2' | El sistema proporciona un recuadro para filtrar por el nombre de la clínica. |
| | 6' | Si el usuario pincha sobre el No, la clínica no es eliminada. El sistema vuelve al paso 4. |
| Postcondición | La clínica es eliminada del sistema. | |

Cuadro A.26: Caso de uso: Eliminar clínica.

| CU-27 | Confirmar asistencia cita | |
|---|---|---|
| Descripción | El usuario confirma su asistencia a la cita mediante el escaneo de un código QR, mediante el enlace del *Telegram* o mediante un administrativo. | |
| Precondición | El usuario debe estar registrado, haber iniciado sesión y poseer un móvil para leer el QR. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario escanea el código QR. |
| | 2 | El sistema busca las citas del día del usuario y se las muestra. |
| | 3 | El usuario pulsa la casilla de la/s cita/s. |
| | 4 | El sistema confirma la/s cita/s. |
| Errores / Alternativas | Paso | Acción |
| | 2' | Si el usuario no tiene citas ese día, el sistema no realiza ninguna acción. |
| Postcondición | La cita del usuario es confirmada. | |

Cuadro A.27: Caso de uso: Confirmar asistencia cita paciente código QR.

| CU-28 | Enviar recordatorio citas | |
|---|---|---|
| Descripción | El usuario, administrativo, envía un recordatorio por email o *Telegram* a los pacientes que tienen cita al día siguiente. | |
| Precondición | El usuario debe estar registrado. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario pulsa sobre Enviar recordatorio citas día siguiente. |
| | 2 | El sistema envía un email y un mensaje por el bot de Telegram a los pacientes con cita el día siguiente. |
| Errores / Alternativas | Paso | Acción |
| | 2' | Si el paciente no tiene email o *bot* de *Telegram* activado, el sistema no envía correo a dicho paciente. Tampoco se envía nada si el paciente ha indicado en Preferencias que no quiere recibir avisos. |
| Postcondición | Los email y mensajes por *Telegram* son enviados a los pacientes. | |

Cuadro A.28: Caso de uso: Enviar recordatorio citas.

| CU-29 | Restaurar Base de datos | |
|---|---|---|
| Descripción | El usuario, administrativo, hace uso de una copia de seguridad para recuperar la información de la base de datos en caso de pérdida. | |
| Precondición | El usuario debe estar registrado y debe existir una copia de seguridad. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario pulsa sobre Restaurar BD. |
| | 2 | El sistema restaura la base de datos. |
| Errores / Alternativas | Paso | Acción |
| | 2' | Si no hay copia de seguridad, el sistema no realiza ninguna acción |
| Postcondición | El sistema restaura la base de datos. | |

Cuadro A.29: Caso de uso: Restaurar Base de datos.

| CU-30 | Importar/exportar datos | |
|---|---|---|
| Descripción | El usuario, administrativo, importa datos desde un archivo CSV. | |
| Precondición | El usuario debe estar registrado y debe existir archivo .csv con los datos estructurados de forma adecuada para la BBDD. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario pulsa sobre Importar/Exportar datos. |
| | 2 | El sistema muestra una serie de opciones para importar datos. |
| | 3 | El usuario hace clic sobre el tipo de datos que quiere importar. |
| | 4 | El sistema muestra una opción para subir un archivo. |
| | 5 | El usuario sube el archivo adecuado. |
| | 6 | El sistema importa los datos a la BBDD. |
| Errores / Alternativas | Paso | Acción |
| | 2' | El sistema muestra una serie de opciones para exportar datos. |
| | 3' | El usuario pulsa sobre los datos que quiere exportar. |
| | 4' | El sistema emite una ventana emergente para guardar el archivo .csv con los datos a exportar. |
| Postcondición | El sistema añade los datos a la BBDD o el sistema exporta los datos de la BBDD. | |

Cuadro A.30: Caso de uso: Importar/exportar datos.

| CU-31 | Reprogramar citas | |
|---|---|---|
| Descripción | El usuario, administrativo, reprograma las citas de un médico en los días que éste está ausente. | |
| Precondición | El usuario debe estar registrado y el médico debe tener citas en los días en los que se va a reprogramar. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El sistema empieza en el caso de uso CU-9 A.10. Arriba de la tabla de la agenda, existen 2 botones: Anular Citas y Posponer Citas. |
| | 2 | El usuario pulsa sobre Anular Citas. |
| | 3 | El sistema solicita una fecha de entrada y una fecha de salida. |
| | 4 | El usuario proporciona los datos solicitados. |
| | 5 | El sistema anula las citas entre ambas fechas y envía un email a cada paciente para avisarle de que su cita ha sido anulada y que soliciten una nueva. |
| Errores / Alternativas | Paso | Acción |
| | 2' | El usuario pulsa sobre Posponer Citas. |
| | 5' | Si el procedimiento es el de Posponer Citas, se le asigna una nueva cita y se avisa, vía email, de la nueva fecha. |
| | 5' | Si el paciente no tiene email ni se pospone ni se anula de manera automática, el administrativo tiene que realizar este procedimiento manualmente. |
| Postcondición | El sistema anula las citas correspondientes. | |

Cuadro A.31: Caso de uso: Reprogramar citas.

| CU-32 | Correlacionar cita | |
|---|---|---|
| Descripción | El usuario, administrativo, correlaciona 2 citas. | |
| Precondición | El usuario debe estar registrado y deben existir 2 citas del mismo paciente que tengan relación entre ellas. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Buscar en el recuadro de Paciente. |
| | 2 | El sistema muestra una cuadro de búsqueda. |
| | 3 | El usuario proporciona el dato solicitado. |
| | 4 | El sistema muestra una tabla con los siguientes campos: Nombre, Apellidos, Teléfono, Correo Electrónico y Acciones (con 3 acciones: Ver Citas, Editar y Eliminar). |
| | 5 | El usuario pulsa sobre Ver Citas. |
| | 6 | El sistema muestra una tabla con los siguientes campos: Paciente, Médico, Especialidad, Clínica, Sala, Fecha, Hora, Cita correlacionada y Acciones (con 3 acciones: Correlación, Editar y Eliminar). |
| | 7 | El usuario hace clic sobre Correlación. |
| | 8 | El sistema solicita la Cita para correlacionar una con otra. |
| | 9 | El usuario introduce el dato solicitado. |
| | 9 | El sistema guarda la información. |
| Errores / Alternativas | Paso | Acción |
| | 4’ | Si el usuario introduce un nombre de paciente que no esté en el sistema, se muestra un mensaje de error ”Paciente no encontrado”. |
| | 6’ | Si la cita ya ha sido confirmada, en Acciones sale un aviso de ”Cita confirmada”. |
| Postcondición | El sistema correlaciona las citas del usuario. | |

Cuadro A.32: Caso de uso: Correlacionar cita.

| CU-33 | Llamar paciente automático | |
|---|---|---|
| Descripción | El usuario, médico, llama, de manera automática, al siguiente paciente. | |
| Precondición | El usuario debe estar registrado. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Gestión Médica. Después lo hace sobre Llamar siguiente paciente. |
| | 2 | El sistema comprueba quién es el siguiente paciente y lo llama, mandando un mensaje a la pantalla de la sala de espera (CU-14 A.14). |
| Errores / Alternativas | Paso | Acción |
| | 2’ | Si el paciente tiene registrado un *chatid* de *Telegram*, se le envía un mensaje por este medio. |
| Postcondición | Sale un aviso para el paciente en la sala de espera. El sistema confirma como atendido al anterior paciente llamado. | |

Cuadro A.33: Caso de uso: Llamar paciente automático.

| CU-34 | Eliminar último paciente llamado | |
|---|---|---|
| Descripción | El usuario, médico, elimina la llamada al último paciente. | |
| Precondición | El usuario debe estar registrado. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Gestión Médica. Después lo hace sobre Eliminar última llamada. |
| | 2 | El sistema comprueba quien es el último paciente llamado y elimina la llamada. |
| Errores / Alternativas | No | |
| Postcondición | Se elimina la llamada al último paciente llamado. | |

Cuadro A.34: Caso de uso: Eliminar último paciente llamado.

| CU-35 | Preferencias | |
|---|---|---|
| Descripción | El usuario, paciente, elige que canales de comunicación quiere tener habilitados. | |
| Precondición | El usuario debe estar registrado. | |
| Secuencia Principal | Paso | Acción |
| | 1 | El usuario hace clic sobre Área Personal. Después lo hace sobre Preferencias. |
| | 2 | El sistema muestra una lista con 2 elementos (Enviar correo, Enviar *telegram*) y sus correspondiente casilla de marcado. |
| | 3 | El usuario marca/desmarca las casillas que le interesen. Después pulsa sobre Guardar preferencias. |
| | 4 | El sistema guarda las preferencias del usuario. |
| Errores / Alternativas | No | |
| Postcondición | Se actualizan las preferencias de canales de comunicación del usuario. | |

Cuadro A.35: Caso de uso: Preferencias.

| Caso de uso | ACT-01 (Administrativo) | ACT-02 (Médico) | ACT-03 (Paciente) |
|---|---|---|---|
| CU-01 | ✓ | ✓ | ✓ |
| CU-02 | X | X | ✓ |
| CU-03 | ✓ | X | X |
| CU-04 | X | X | ✓ |
| CU-05 | ✓ | ✓ | ✓ |
| CU-06 | ✓ | ✓ | ✓ |
| CU-07 | ✓ | ✓ | ✓ |
| CU-08 | ✓ | ✓ | ✓ |
| CU-09 | X | X | ✓ |
| CU-10 | ✓ | ✓ | X |
| CU-11 | ✓ | ✓ | X |
| CU-12 | X | ✓ | X |
| CU-13 | X | ✓ | X |
| CU-14 | ✓ | X | X |
| CU-15 | ✓ | X | X |
| CU-16 | ✓ | X | X |
| CU-17 | ✓ | X | X |
| CU-18 | ✓ | X | X |
| CU-19 | ✓ | X | X |
| CU-20 | ✓ | X | X |
| CU-21 | ✓ | X | X |
| CU-22 | ✓ | X | X |
| CU-23 | ✓ | X | X |
| CU-24 | ✓ | X | X |
| CU-25 | ✓ | X | X |
| CU-26 | ✓ | X | X |
| CU-27 | ✓ | X | ✓ |
| CU-28 | ✓ | X | X |
| CU-29 | ✓ | X | X |
| CU-30 | ✓ | X | X |
| CU-31 | ✓ | X | X |
| CU-32 | ✓ | ✓ | X |
| CU-33 | X | ✓ | X |
| CU-34 | X | ✓ | X |
| CU-35 | X | X | ✓ |

Cuadro A.36: Casos de uso disponibles para cada actor.

Apéndice B

# Manual de usuario

En este anexo se verá el manual de usuario de la aplicación. Para su realización se ha seguido la guía de referencia de la wiki de la Facultad de Informática de Coruña. [38]

## B.1 Procedimientos de instalación, arranque y finalización de la Aplicación

Para arrancar la aplicación se hará uso de *Docker*, como se ha visto en el capítulo 7. Para finalizarla llega con finalizar el contenedor de *Docker*.

Los comandos necesarios son los siguientes:

```
1 docker pull tfgclinicas2021/tfg_gestion_citas:version_final
2 docker run -p 8000:8000
    tfgclinicas2021/tfg_gestion_citas:version_final python3
    manage.py runserver 0:8000
```

El primero es para descargar el contenedor del repositorio de *Docker Hub*. El segundo es para ponerlo a funcionar, le indica a *Docker* que use el mismo puerto que usa *Django* (que es el puerto 8000).

## B.2 Descripción de Menús, Pantallas y Procesos

### B.2.1 Elementos comunes a toda la Aplicación

En esta primera figura (B.1), que es la vista inicial de la aplicación, se pueden ver los elementos comunes. Por un lado se tiene, en la esquina superior izquierda, el logo de la aplicación. En la esquina superior derecha se pueden ver los botones de Inicio (que vuelve a esta vista cuando se está en otra), Cuadro Médico (para ver los médicos y especialidades), Contacto (formulario de contacto para enviárselo a la clínica) y los botones de Login y Registro (para

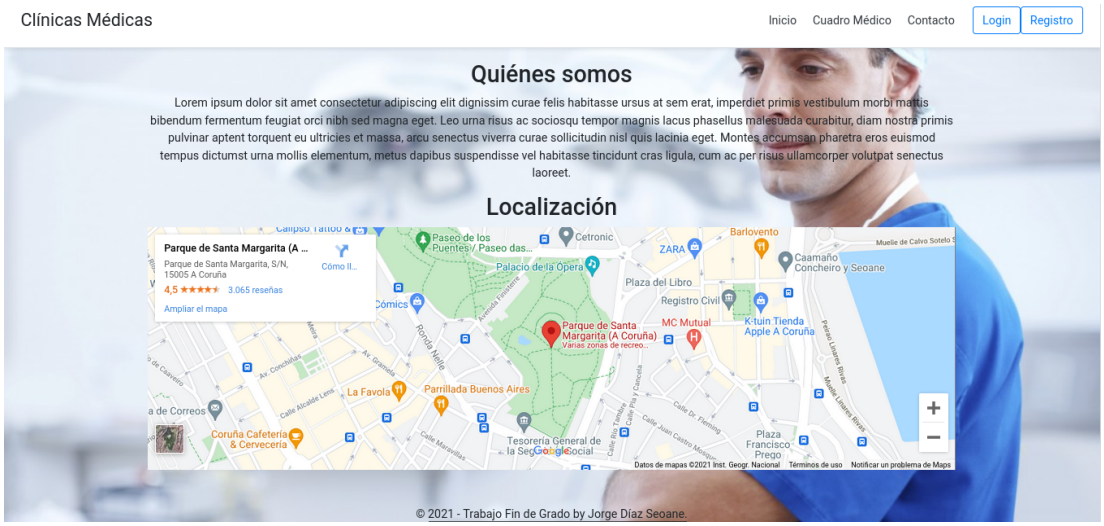



Figura B.1: Página inicial.

iniciar sesión y registrarse). Como fondo de la vista hay una imagen de un sanitario, que se mantendrá en todas las vistas de la aplicación. En la parte de abajo y centrado se tiene el pie de página, en este caso simplemente se puso, a modo de muestra, el año, que es un TFG y el autor del mismo. En un caso real aquí iría los datos de contacto y dirección de las distintas clínicas.

### B.2.2 Menús, pantallas y procesos de la Aplicación

Este apartado se va a dividir entre cuatro roles: visitante sin usuario, paciente, médico y administrativo.

#### Visitante

En la figura vista anteriormente (B.1), además de los elementos comunes ya mencionados, se pueden apreciar otros dos elementos, que solo se ven en esta vista. Por un lado hay un apartado de **Quiénes somos** y por otro la **Localización**. Como se trata de un TFG, en el primer apartado se usó el texto de prueba *lorem ipsum* y para el segundo se eligió poner una dirección genérica. En un caso real aquí irían la información de la empresa de la red de clínicas médicas y la localización de las clínicas.

En las figuras B.2 y B.3 se pueden ver las vistas usadas para iniciar sesión y registrar un usuario en el sistema. Este método de registro solo es válido para los pacientes. Más adelante se verá como se registra un administrativo y un médico, que se hará a través de un administrativo (el primer administrativo es creado por el desarrollador en el momento de desplegar la aplicación).

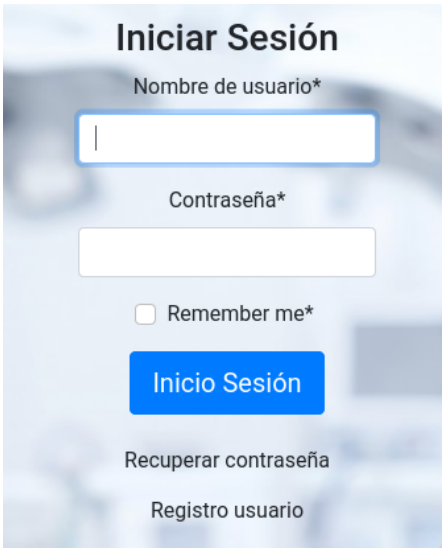



Figura B.2: Página Inicio de sesión.

### Paciente

En la figura B.4 se puede ver como varia el menú de arriba a la derecha una vez que se ha iniciado sesión. En este caso se ve desde el rol de paciente. Como se puede apreciar, aparecen dos nuevos botones (Área Personal y Pedir Cita). Pinchando en el primero, vamos a la siguiente vista (ver figura B.5) y pinchando en el segundo vamos a la siguiente (ver figura B.6).

En esta última vista, una vez elegida fecha y médico, se pincha sobre Elegir hora y se ve la vista de la siguiente figura B.7, donde están las horas disponibles del médico elegido en la fecha seleccionada. Pinchando sobre una hora, queda registrada la cita.

En las siguientes figuras (B.8 y B.9) se puede ver como llegan los mensajes por *Telegram* y por correo electrónico. En ambos casos, el mensaje va acompañado de un enlace para añadir la cita en un evento de *Google Calendar*. Más abajo se verá una opción para que el usuario decida si quiere recibir o no mensajes por estos medios.

Pulsando en modificar datos en la figura B.5, se va a la siguiente vista (ver figura B.10), donde aparecen los datos actuales del paciente y se permite su modificación. Una vez modificados se pulsa en Editar Usuario y ya se actualizan en el sistema. En esta misma vista también hay la opción de Eliminar Cuenta, la cual elimina la cuenta preguntando primero al paciente si está seguro de querer borrarla, y la opción de cambiar la contraseña.

Volviendo a la figura B.5 y pulsando sobre el segundo elemento (Preferencias) se pasa a la vista que se puede ver en la figura B.11, donde se puede configurar las preferencias del correo y el *Telegram*, para indicar si quiere recibir o no avisos por esos medios.

De vuelta a la vista del Área Personal (ver figura B.5) quedan dos opciones por ver, para

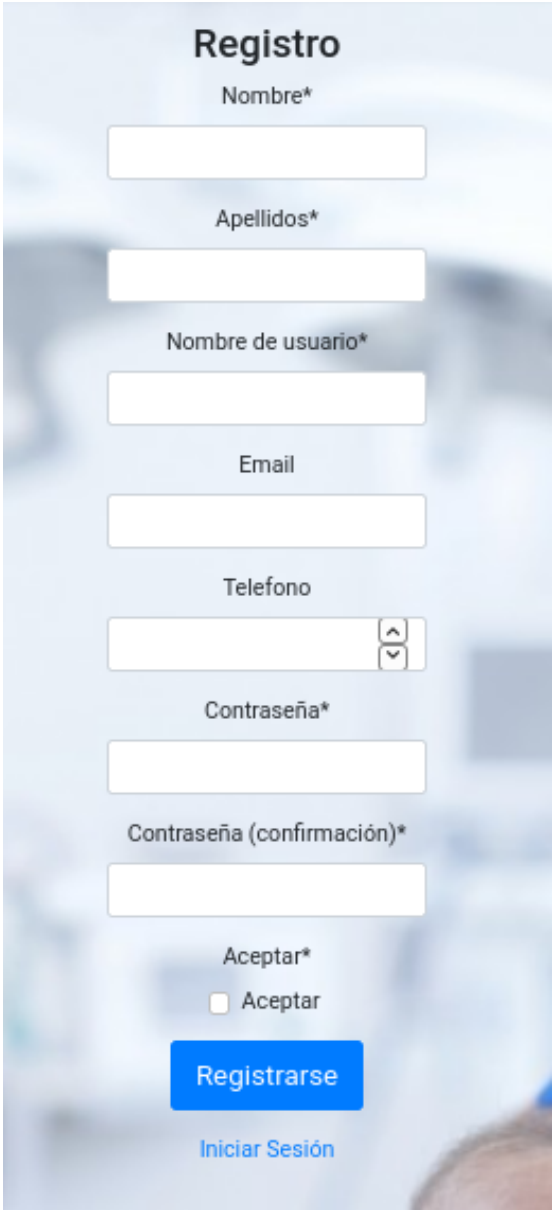



Figura B.3: Página Registro.


Figura B.4: Página inicial con sesión iniciada.


Figura B.5: Página área personal.


Figura B.6: Página pedir cita.


Figura B.7: Página elegir hora cita.

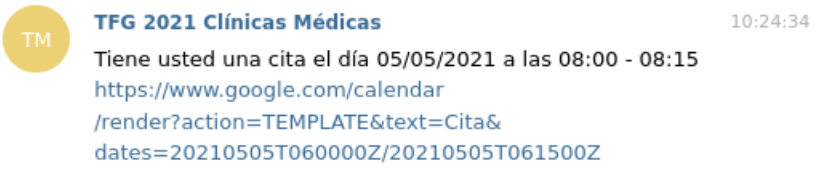



Figura B.8: Mensaje *Telegram* con la cita.

Tiene usted una cita el día 04/06/2021 a las 08:00 - 08:15
https://www.google.com/calendar/render?action=TEMPLATE&text=Cita&
dates=20210604T060000Z%2F20210604T061500Z

Figura B.9: Correo electrónico con la cita.

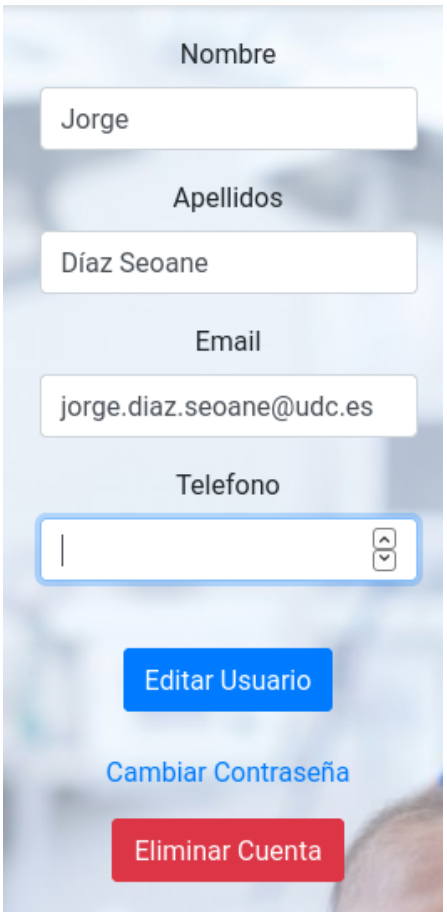



Figura B.10: Página modificar datos paciente.

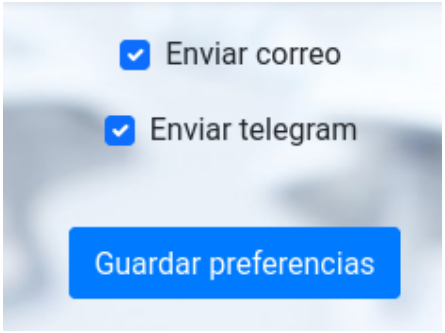



Figura B.11: Página preferencias.


Figura B.12: Página citas paciente (rol paciente).

la segunda (Añadir bot telegram) se verá en el siguiente apartado. En cuanto a la primera (Ver citas) se puede ver en la figura B.12 la vista mostrada una vez pulsado sobre ella. Como se puede observar existen tres botones, uno arriba de la tabla para crear un cita, el cual lleva a la vista mostrada en la figura B.6, y dos más a la derecha en cada fila de la tabla. El primero permite editar una cita y lleva a la misma vista anterior pero con los datos ya añadidos para poder modificarlos, mientras que el segundo muestra un cuadro de diálogo (ver figura B.13) donde pide confirmación para borrar una cita.

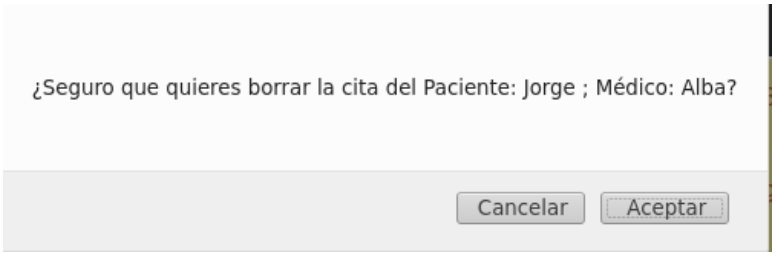



Figura B.13: Cuadro diálogo eliminar cita.

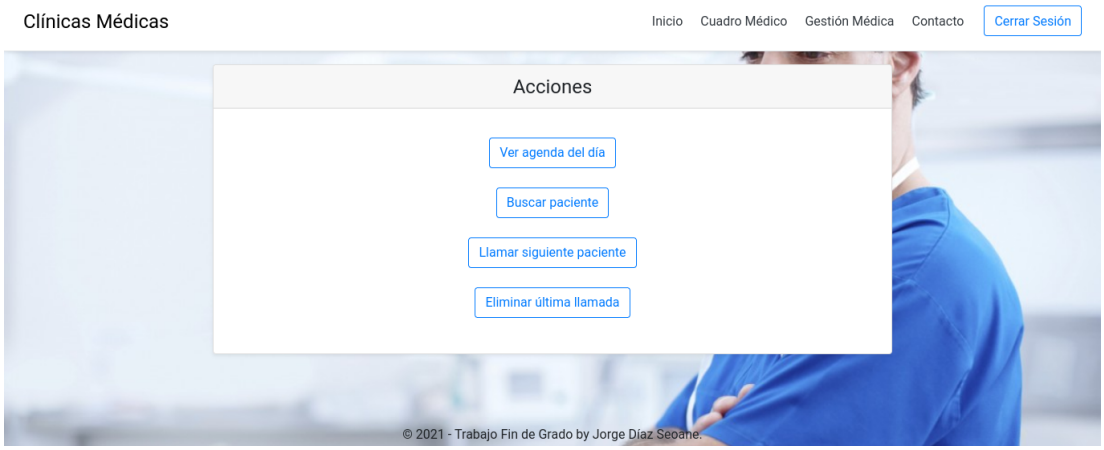



Figura B.14: Página inicial del médico.

| Paciente | Hora | Fecha | Confirmada | Llamada | Atendido |
|---|---|---|---|---|---|
| Jorge Díaz Seoane | 08:00 - 08:15 | Abr. 19, 2021 | Sí | Confirmar | Confirmar |
| Jorge Díaz Seoane | 09:00 - 09:15 | Abr. 22, 2021 | No | Confirmar | Confirmar |

Figura B.15: Página agenda médico.

### Médico

Hasta ahora se ha visto todo lo que puede hacer un paciente con una cuenta creada. A partir de aquí se verá lo que puede hacer el médico. Como se ha mencionado anteriormente, la cuenta del médico es creada por un administrativo. Cuando un médico inicia sesión, se le redirige a la vista que se puede ver en la figura B.14. En el menú que hay arriba a la derecha, a mayores de los elementos comunes ya vistos anteriormente, se puede ver que hay un botón nuevo (Gestión Médica), el cual lleva a la vista que se puede ver en esta misma figura.

Pulsando sobre Ver agenda del día, aparecen dos vistas donde se pide fecha de inicio y fecha de fin (no se incluyen capturas por ser triviales). Una vez elegidas, se ve la siguiente vista (ver figura B.16). En esta vista se puede observar si una cita ha sido confirmada (para confirmar una cita lo puede hacer el paciente o el administrativo) y el médico puede llamar a un paciente (lo que envía un aviso a la pantalla de la sala de espera) y confirmar que ha atendido a un paciente (lo que hace que se deje de mostrar el aviso en la pantalla de la sala de espera).

Volviendo a la vista anterior (ver figura B.14) y pulsando sobre la segunda opción (Buscar paciente) se salta a una vista donde simplemente hay un cuadro de texto para escribir el nombre de un paciente (si se deja en blanco, se muestran todos los pacientes) y un botón para darle a buscar. Hecho esto, se pasa a la siguiente vista (ver figura B.16). Como se puede ver, a parte

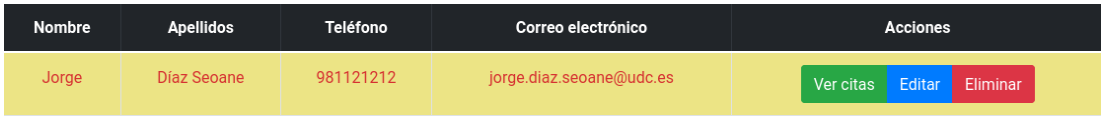



Figura B.16: Página ver pacientes.


Figura B.17: Página citas paciente (rol médico).

de los datos del paciente buscado, existen tres botones. El primero permite ver las citas del paciente en cuestión. El siguiente permite editar los datos de un paciente y el último permite eliminar un paciente. Tanto editar como eliminar siguen el mismo proceso que el visto con las citas con el rol de paciente, solo que con un paciente en lugar de una cita.

En el caso de Ver citas, pulsando sobre él se ve la siguiente vista (ver figura B.17). Arriba de la tabla se puede ver un botón para crear una cita (ya se ha visto anteriormente esta vista, ver figura B.6). En la propia tabla hay tres botones, los dos últimos son, nuevamente, los botones de editar y eliminar una cita. El primero es el que permite correlacionar dos citas. La correlación de citas sirve para unir dos citas que tienen relación entre ellas, un ejemplo de esto sería cuando un paciente tiene una cita para una prueba médica y después la cita con el médico para ver los resultados de la misma.

La tercera opción de la vista inicial del médico (ver figura B.14), Llamar siguiente paciente, es para llamar de manera automática al siguiente paciente y confirmar al último visto. No se incluye figura alguna porque no muestra ninguna vista, simplemente envía un aviso a la pantalla de la sala de espera y al *bot* de *Telegram* (éste último solo si el usuario lo tiene permitido en Preferencias).

La último opción de esta misma vista, Eliminar última llamada, elimina la última llamada que el médico ha realizado. Esto se añade por si un paciente es llamado pero no acude a la sala de espera.

### Administrativo

Visto lo que puede hacer el médico, se pasará a ver lo que puede hacer el administrativo. La vista inicial que éste ve es la que se puede observar en la figura B.18. En el menú de arriba a la derecha se puede observar que existe un botón de Gestión Administrativa, el cual redirige a la vista de la misma figura.

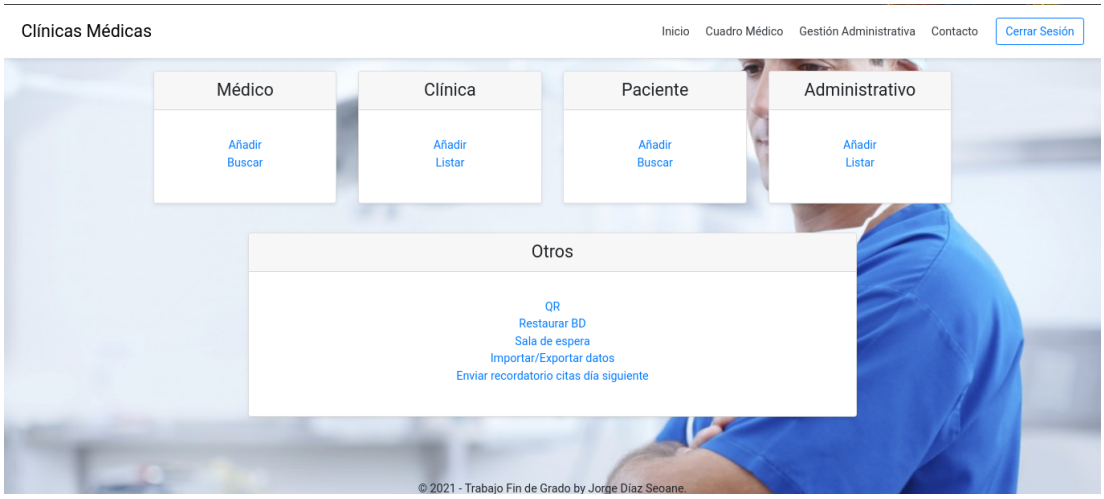



Figura B.18: Página inicial administrativo.

Como se puede observar en esta figura, hay cuatro grandes bloques y unos botones debajo de los mismos. De los cuatro bloques solo se verán dos, por ser los dos restantes similares.

Pulsando sobre añadir en el bloque del médico, se pasa a la siguiente vista (ver figura B.19), donde se puede observar un formulario para añadir un médico. Una vez complementados los campos, se pulsa sobre Añadir médico y éste se añade al sistema.

Pulsando sobre el segundo botón del bloque de Médico (Buscar), se va a una vista donde hay un cuadro para escribir para filtrar por nombre de médico, una vez filtrado (si no se escribe nada, se muestran todos los médicos) se pasa a una vista con una tabla (ver figura B.20). Los botones de editar y eliminar son similares a los vistos anteriormente para las citas del paciente pero con los médicos.

El botón de Agenda, después de pasar por dos vistas que solicitan día inicio y día fin, lleva a una vista con una tabla con la agenda del médico vista desde el rol de administrativo (ver B.21). Como se puede observar, el administrativo puede confirmar una cita de un paciente.

Los dos botones de arriba de la tabla (Anular Citas y Posponer Citas) permiten la reprogramación de citas. Ambos llevan a dos vistas donde se pide día inicio y día fin y después realizan la operación. El primero anula todas las citas de los pacientes, que tengan correo añadido, en esas fechas. El segundo las pospone, con la misma condición del correo. Para los que no disponen de correo, deberá realizarlo manualmente el administrativo. En ambos casos, se le envía un correo al paciente informándole de lo sucedido.

Se verá ahora un ejemplo de posponer citas. Se partirá de la agenda que se puede ver en la figura B.22. Las citas del día 5 pasarán a estar en el día 10, para ello se pulsa sobre el botón de Posponer Citas y en fechas se marca del día 5 (miércoles) al día 7 (viernes) [1]. Como se

[1] Los fines de semana no se dan citas, así que esas fechas no son necesarias incluirlas. Por ello la aplicación salta al lunes 10 directamente.

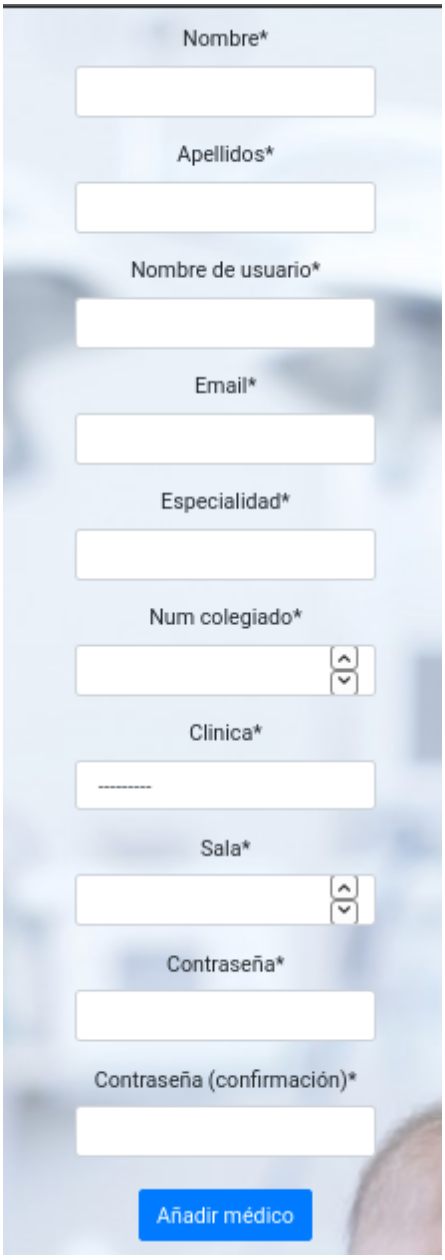



Figura B.19: Página formulario añadir médico.

| Usuario | Doctor | Especialidad | Nº Colegiado | Clínica | Sala | Acciones |
|---|---|---|---|---|---|---|
| adc | Dr. Díaz Campos, Alba | Medicina General | 151501478 | Clínica Margarita | 1 | Agenda Editar Eliminar |
| jtg | Dr. Toral García, Juan | Oftalmología | 151507845 | Clínica Girasol | 1 | Agenda Editar Eliminar |
| lfr | Dr. Fernández Rodríguez, Laura | Odontología | 151501597 | Clínica Lirio | 4 | Agenda Editar Eliminar |
| gfs | Dr. Fernández Silva, Gonzalo | Enfermería | 151501573 | Clínica Margarita | 6 | Agenda Editar Eliminar |

Figura B.20: Página tabla médicos.


| Paciente | Hora | Fecha | Confirmada | Llamada médico | Atendido |
|---|---|---|---|---|---|
| Jorge Díaz Seoane | 08:00 - 08:15 | Abr. 29, 2021 | Confirmar | No | No |
| Jorge Díaz Seoane | 09:00 - 09:15 | Abr. 29, 2021 | Confirmar | No | No |

Figura B.21: Página agenda médico (vista desde el rol de administrativo).

puede observar en la figura B.23 las citas han sido reubicadas. En la figura B.24 se puede ver un ejemplo del correo enviado al paciente con su cambio de cita.

Volviendo a la vista inicial del administrativo (ver figura B.18), el botón de añadir del resto de bloques hace lo mismo que el de médico, permitiendo añadir administrativos (el primer administrativo es creado por el desarrollador pero el resto son creados por otro administrativo), pacientes y clínicas. Por este motivo, no se verán las vistas asociadas. Pulsando sobre el botón de Listar en el bloque de Clínica, se va a la vista que se puede ver en la figura B.25. Como se puede observar, se tiene nuevamente los botones de Editar y Eliminar. En la parte derecha

| Paciente | Hora | Fecha | Confirmada | Llamada médico | Atendido |
|---|---|---|---|---|---|
| Jorge Díaz Seoane | 08:00 - 08:15 | Mayo 5, 2021 | Confirmar | No | No |
| Jorge Díaz Seoane | 09:00 - 09:15 | Mayo 5, 2021 | Confirmar | No | No |
| Jorge Díaz Seoane | 08:00 - 08:15 | Mayo 10, 2021 | Confirmar | No | No |
| Jorge Díaz Seoane | 08:15 - 08:30 | Mayo 10, 2021 | Confirmar | No | No |
| Jorge Díaz Seoane | 08:30 - 08:45 | Mayo 10, 2021 | Confirmar | No | No |

Figura B.22: Agenda médico inicial.

| Paciente | Hora | Fecha | Confirmada | Llamada médico | Atendido |
|---|---|---|---|---|---|
| Jorge Díaz Seoane | 08:00 - 08:15 | Mayo 10, 2021 | Confirmar | No | No |
| Jorge Díaz Seoane | 08:15 - 08:30 | Mayo 10, 2021 | Confirmar | No | No |
| Jorge Díaz Seoane | 08:30 - 08:45 | Mayo 10, 2021 | Confirmar | No | No |
| Jorge Díaz Seoane | 08:45 - 09:00 | Mayo 10, 2021 | Confirmar | No | No |
| Jorge Díaz Seoane | 09:00 - 09:15 | Mayo 10, 2021 | Confirmar | No | No |

Figura B.23: Agenda médico después del cambio.

> Su cita ha sido cambiada para el día 10/05/2021 a las 08:45 - 09:00

Figura B.24: Correo electrónico enviado al paciente.

de la tabla hay un cuadro para filtrar por nombre de clínica, si se deja en blanco se muestran todas las clínicas registradas.

Lo botones de Listar y Buscar de los bloques restantes (ver figura B.18), Paciente y Administrativo, son iguales que los vistos hasta ahora en Médico y Clínica. Por ese motivo, se pasa a ver los botones de la parte de abajo.

El botón de Importar/Exportar datos lleva a la página que se puede en la figura B.26, pulsando sobre alguno de los de importación se va a la página que se ve en la figura B.27 (la URL varía en cada uno para que los datos sean introducidos correctamente pero la vista es la misma). Los datos del CSV deberán ser consistentes con la BBDD. En la primera fila van los nombres de los campos de la tabla. La última columna será el campo id, el cual se deja vacío dado que será la aplicación quien lo añada en el momento de la importación. Se puede ver un

| Nombre | Dirección | Teléfono | Salas disponibles (total) | Acciones |
|---|---|---|---|---|
| Clínica Margarita | Avda Corcubión 2, 2ºA | 981487415 | 8 | Editar Eliminar |
| Clínica Tulipán | Avda Muros 25, 3º | 981024856 | 10 | Editar Eliminar |
| Clínica Girasol | Avda Noia 15, 8ºA | 981047152 | 5 | Editar Eliminar |
| Clínica Lavanda | Avda Navarra 34, 1ºA | 981164872 | 10 | Editar Eliminar |
| Clínica Crisantemo | Avda Lugo 2, 2ºA | 981204876 | 8 | Editar Eliminar |
| Clínica Lirio | Calle Donosti 12, Bajo | 981491572 | 15 | Editar Eliminar |

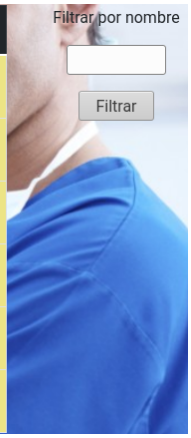



Figura B.25: Página listar clínicas.


Figura B.26: Página importar/exportar.


Figura B.27: Página importar datos.

ejemplo en la figura B.28.

Volviendo a la vista inicial del administrativo (ver figura B.18), el botón Enviar recordatorio citas día siguiente no lleva a ninguna vista, envía un recordatorio de cita para las citas del día siguiente (ver figura B.29).

El botón de Restaurar BD no lleva a ninguna vista, lo que hace es coger la última copia de seguridad de la BBDD y usarla para su restauración.

El botón de la Sala de espera lleva a la vista de la figura B.30, una vez introducidos los datos se va a la vista de la figura B.31. La idea de esta vista es tener pantallas físicas en las salas de espera y llamar por ellas a los pacientes. Por ese motivo se eligen las salas que cada pantalla va a mostrar. Un ejemplo de esto sería que, para una misma clínica, en una pantalla van las salas de la 1 a la 4 y en otra de la 5 a la 8. Esta pantalla se actualiza automáticamente cada 10 segundos.

El botón QR lleva a una vista con el QR necesario para confirmar cita mediante QR. En un caso real, este QR iría impreso en papeles distribuidos por la clínica pero aquí se dejó así para poder probar la funcionalidad.

```
direccion,telefono,nombre,salas_disponibles,id
Avda Prueba,981324567,Prueba,10,
Avda Prueba 2,981324568,Prueba 2,10,
```

Figura B.28: Ejemplo archivo csv.

> Le recordamos que tiene usted una cita el día 06/05/2021 a las 08:00 - 08:15

Figura B.29: Correo electrónico con recordatorio de cita.

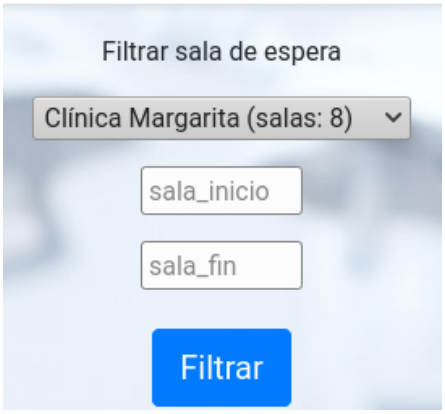



Figura B.30: Página filtro sala de espera.

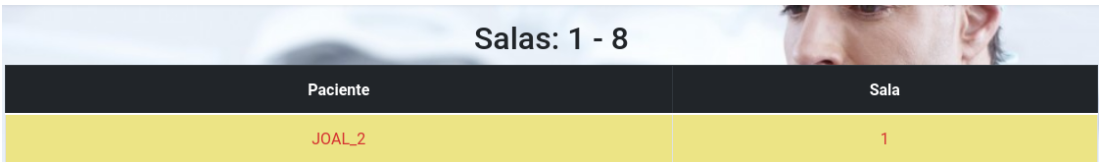



Figura B.31: Página sala de espera.

### B.2.3 Menús y pantallas de ayuda

En este apartado se verá como asociar el *bot* de *Telegram* con el paciente.

Lo primero que se debe hacer es buscar el *bot* en la aplicación de *Telegram*. En este caso, el *bot* se llama **tfg_2021_jds_bot**. Una vez iniciada la conversación con él (se puede escribir cualquier cosa pero es necesario escribir algo), habrá que ir a la aplicación, iniciar sesión y, en el área personal (ver figura B.5), pulsar sobre el botón de **Añadir bot telegram**. La aplicación ya se encarga de asociar el *chatid* con el usuario. Desde este momento, la aplicación ya puede enviarle mensajes al paciente a través de este medio.

## B.3 Procedimientos Especiales

En este apartado se verá el procedimiento de copia de seguridad y restauración de la BBDD.

La copia de seguridad se realiza automáticamente todos los días a las 08:00. Se guardan en una carpeta llamada **backup**. Para restaurar una copia de seguridad es necesario tener el rol de administrativo. Desde la vista principal de este rol (ver figura B.18) hay un botón **Restaurar BD**, pulsándolo la aplicación se encarga de cargar la última copia de seguridad disponible en la BBDD. De esta manera, la BBDD vuelve a estar en un estado donde los datos son correctos y solo se habrá perdido lo que haya desde las 08:00 hasta la hora donde la BBDD haya empezado a fallar o se hayan perdido los datos.

# Lista de acrónimos

**BBDD** Bases de Datos. vii, 6, 7, 25, 26, 28, 30, 32–34, 38, 44, 53, 55, 82, 99, 100, 102

**CERN** Conseil Européen pour la Recherche Nucléaire. 6

**CSS** Cascading Style Sheets. 6, 66

**CSV** Comma-Separated Values. 3, 20, 43, 53, 82, 99

**ITV** Inspección Técnica de Vehículos. 1, 39

**QR** Quick Response code. 1, 7, 14, 20, 36, 48, 49, 54, 100

**RGPD** Reglamento General de Protección de Datos. 2, 3, 15, 35

**S.O.** Sistema Operativo. 5, 23

**SQL** Structured Query Language. 6, 7

**TFG** Trabajo Fin de Grado. 1, 9, 11, 15–17, 23, 36, 62, 65, 88

**URL** Uniform Resource Locator. 28, 36, 43, 45, 46, 48, 49, 99

# Glosario

**framework** Conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar. [39]. 1, 5, 6, 41, 65

**script** Término informal que se usa para designar a un programa relativamente simple. [40]. 5, 34, 36, 41

# Bibliografía


[1] "Reglamento (ue) 2016/679 del parlamento europeo y del consejo de 27 de abril de 2016 relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos y por el que se deroga la directiva 95/46/ce (reglamento general de protección de datos)," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.boe.es/doue/2016/119/L00001-00088.pdf

[2] "Diferencias entre datos personales y datos sensibles a efectos del rgpd," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.pridatect.es/diferencias-entre-datos-personales-y-datos-sensibles-a-efectos-del-rgpd/

[3] "Tutorial de python," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://docs.python.org/es/3.8/tutorial/index.html

[4] "Historia de python," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Historia_de_Python

[5] "Javascript," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/JavaScript

[6] "La historia de javascript," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://hackaboss.com/blog/historia-javascript

[7] "Desarrollo de aplicaciones web. introducción general a html y css," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.um.es/docencia/barzana/DAWEB/Introduccion-a-html-y-css.html

[8] "Django project," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.djangoproject.com/

[9] "Django (framework)," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Django_(framework)

[10] "Build fast, responsive sites with bootstrap," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://getbootstrap.com/

[11] D. Suárez, "Qué es bootstrap y cómo usarlo," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://raiolanetworks.es/blog/bootstrap/#que_es_bootstrap

[12] "What is sqlite?" consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.sqlite.org/index.html

[13] "Sqlite," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://es.wikipedia.org/wiki/SQLite

[14] "Ngrok," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://ngrok.com/

[15] "Docker (software)," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Docker_(software)

[16] "Visual studio code," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Visual_Studio_Code

[17] "Latex," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://es.wikipedia.org/wiki/LaTeX

[18] "Latex – a document preparation system," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.latex-project.org/

[19] "Tex," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://es.wikipedia.org/wiki/TeX

[20] L. M. C. Souto, "Modelo de tfg para a fic udc," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.overleaf.com/project/5c3a778a4923de588891aeeb

[21] "draw.io computer-software," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://de.linkedin.com/company/draw-io

[22] P. Domínguez, "En qué consiste el modelo en cascada," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://openclassrooms.com/en/courses/4309151-gestiona-tu-proyecto-de-desarrollo/4538221-en-que-consiste-el-modelo-en-cascada

[23] "Metodología de desarrollo de software (iii) – modelo en espiral," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://aspgems.com/metodologia-de-desarrollo-de-software-iii-modelo-en-espiral/

[24] M. Maldonado, "Las mejores metodologías ágiles para la creación de software," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.digital55.com/desarrollo-tecnologia/mejores-metodologias-agiles-creacion-software/

[25] Rebeca, "La metodología Ágil más usada: Scrum," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.nextu.com/blog/que-es-scrum/

[26] M. A. Alvarez, "Qué es mvc," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://desarrolloweb.com/articulos/que-es-mvc.html

[27] "Django #1: introducción y el patrón mtv ¿qué es django?" consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.escuelapython.com/django-1-introduccion-patron-mtv/

[28] "Django appears to be a mvc framework, but you call the controller the "view", and the view the "template". how come you don't use the standard names?" consultado el 23 de junio de 2021. [En línea]. Disponible en: https://docs.djangoproject.com/en/3.1/faq/general/#faq-mtv

[29] "Arquitectura shared nothing," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Arquitectura_shared_nothing

[30] "El libro de django 1.0. 20.1. nada compartido," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://uniwebsidad.com/libros/django-1-0/capitulo-20/nada-compartido

[31] "Security in django," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://docs.djangoproject.com/es/3.1/topics/security/

[32] "Cómo programar un chatbot para telegram en python," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://robologs.net/2019/07/30/como-programar-un-chatbot-para-telegram-en-python/

[33] "Quickstart: Compose and django," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://docs.docker.com/samples/django/

[34] "Tutorial de docker: instalar y gestionar la plataforma de contenedores," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.ionos.es/digitalguide/servidores/configuracion/tutorial-docker-instalacion-y-primeros-pasos/

[35] "Web de docker," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.docker.com/

[36] "Pruebas de caja negra y un enfoque práctico," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://testingbaires.com/2017/02/26/pruebas-caja-negra-enfoque-practico/

[37] J. Garzas, "¿pruebas de integración, funcionales, de carga...? ¡qué jaleo! ¿qué diferencias hay?" consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.javiergarzas.com/2014/07/tipos-de-pruebas-10-min.html

[38] S. C. Simón, "Indice del manual del usuario," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://wiki.fic.udc.es/_media/docencia:pfc:estructura-manuales-usuario-y-referencia-pfc.pdf

[39] "Framework," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Framework

[40] "Script," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Script