



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería
Informática**

**Aplicación de Gestión de
Quirófanos mediante
Inteligencia Computacional
Documentación Técnica**



Presentado por Jesús García Armario
en Universidad de Burgos — 2 de julio de 2023
Tutores: Bruno Baruque Zanon, Daniel Urda
Muñoz y Raúl Marticorena Sánchez

Índice general

Índice general	i
Índice de figuras	iii
Índice de tablas	v
Apéndice A Plan de Proyecto Software	1
A.1. Introducción	1
A.2. Planificación temporal	2
A.3. Estudio de viabilidad	6
Apéndice B Especificación de Requisitos	11
B.1. Introducción	11
B.2. Objetivos generales	12
B.3. Catalogo de requisitos	12
B.4. Especificación de requisitos	15
Apéndice C Especificación de diseño	29
C.1. Introducción	29
C.2. Diseño de datos	29
C.3. Diseño procedimental	31
C.4. Diseño arquitectónico	34
Apéndice D Documentación técnica de programación	45
D.1. Introducción	45
D.2. Estructura de directorios	45
D.3. Manual del programador	48

D.4. Compilación, instalación y ejecución del proyecto	52
D.5. Pruebas del sistema	66
Apéndice E Documentación de usuario	69
E.1. Introducción	69
E.2. Requisitos de usuarios	69
E.3. Instalación	70
E.4. Manual del usuario	70
Bibliografía	79

Índice de figuras

B.1. Diagrama de Casos de uso	15
C.1. Diagrama Relacional	30
C.2. Diagrama Relacional	30
C.3. Diag Int-Seq: Modificar Perfil de Usuario	31
C.4. Diag Int-Seq: Gestión de Usuarios	32
C.5. Diag Int-Seq: Gestión de Predicciones	33
C.6. Diag Int-Seq: Gestión de Planificaciones	34
C.7. Esquema Cliente-Servidor de APP-WEB y API	35
C.8. Esquema Cliente-Servidor de Usuario y APP-WEB	36
C.9. Modelo-Vista-Controlador	37
C.10. Diagrama de paquetes API	38
C.11. Diagrama de paquetes APP Web	39
C.12. Diagrama de clases del subsistema API - 1	41
C.13. Diagrama de clases del subsistema API - 2	42
C.14. Diagrama de clases del subsistema APP-WEB - 1	43
C.15. Diagrama de clases del subsistema APP-WEB - 2	44
D.1. Vista del repositorio y función de fork	49
D.2. Clonar Repositorio con Github Desktop	50
D.3. Agregar repositorio local al entorno de trabajo de Visual Studio Code	51
D.4. Activación y desactivación de entorno virtual	52
D.5. Localización de enlace al despliegue de la aplicación en GitHub	53
D.6. Configuración de usuario maestro en base de datos RDS	54
D.7. Panel de configuración y características de BD en Amazon RDS	55
D.8. Ejemplo de conexión a BD RDS usando MySQL Workbench	55
D.9. Vista de script y resultado de su ejecución en conexión a BD RDS	56

D.10.Código fuente con las rutas de conexión con el esquema <i>gestor_quirófanos</i> en la base de datos	57
D.11.Creación de contenedor para la API desde línea de comandos	58
D.12.Formulario de creación de un repositorio Amazon ECR	59
D.13.Ejemplo de claves de envío Amazon ECR	60
D.14.Ejecución satisfactoria de la subida de un contenedor a repositorio Amazon ECR	60
D.15.Visualización de panel de de configuración de nuestro cluster ubutfgcluster	61
D.16.Definición de familia de tareas API-TFG, añadiendo enlace a la imagen del contenedor en ECR	62
D.17.Definición de servicio <i>gestorquirófanosapi</i> en el cluster	62
D.18.Localización de URL del despliegue de la API	63
D.19.Modificación de las rutas	64
D.20.Comandos de envío del contenedor <i>gestorquirófanosapp</i> al nuevo repositorio ECR	64
D.21.Obtención de la dirección pública del despliegue de la APP	65
D.22.Comprobación del correcto despliegue del sistema en AWS	65
D.23.Ejemplo de gráfico de rendimiento (<i>KNN</i>) disponible en los Jupyter Notebooks	67
D.24.Ejemplo de <i>request</i> y <i>response</i> durante las pruebas de sistema en la API	68
E.1. Ejemplo de mensaje de error al no localizar al usuario	71
E.2. Login satisfactorio y redirección al panel de usuario	71
E.3. Localización de perfil de usuario	72
E.4. Panel de usuario y función de edición	72
E.5. Vista del Panel de Predicciones	73
E.6. Ventana flotante para solicitar una nueva predicción a la API	74
E.7. Acceso al panel de planificaciones	74
E.8. Ventana flotante para añadir una nueva planificación	75
E.9. Se muestra ejemplo de visualización de la programación	75
E.10.Detalle de la planificación	75
E.11.Vista del panel de administrador, incluyendo función de gestión de usuarios	76
E.12.Panel de gestión de usuarios	76
E.13.Formulario de creación de usuario	77
E.14.Ventana de modificación de usuario	77

Índice de tablas

A.1. Costes de Personal	6
A.2. Costes de Equipamiento	6
A.3. Costes Extra	7
A.4. Costes Totales	7
A.5. Librerías y Licencias	9
B.1. CU-1 Gestión de Usuarios.	16
B.2. CU-2 Creación de Usuarios.	17
B.3. CU-3 Eliminación de Usuarios.	18
B.4. CU-4 Modificación de Usuarios.	19
B.5. CU-5 Login de Usuarios.	20
B.6. CU-6 Asignación de Privilegios.	21
B.7. CU-7 Gestión de Predicciones.	22
B.8. CU-8 Visualización de Predicciones.	23
B.9. CU-9 Obtención de Predicciones.	24
B.10. CU-10 Validación de Datos.	25
B.11. CU-11 Gestión de Planificaciones.	26
B.12. CU-12 Obtención de Planificaciones.	27
B.13. CU-13 Visualización de Planificaciones.	28
E.1. Restricciones en variables de datos	73

Apéndice A

Plan de Proyecto Software

A.1. Introducción

Dentro de las etapas que configuran un proyecto, la **planificación** constituye un elemento esencial. Durante esta fase, realizaremos una estimación de los costes relativos a su ejecución: *económicos directos e indirectos, temporales*, etc.

Dividiremos este plan en dos apartados:

1. **Planificación Temporal:** trataremos de conformar un cronograma donde esbochemos las restricciones temporales, tanto para cada uno de los apartados del proyecto, como asignando fechas *inicio* y *finalización* estimadas.
2. **Estudio de Viabilidad:** estimación y previsión de *restricciones* y *costes* a los que puede enfrentarse el desarrollo de nuestro trabajo. A su vez, distinguimos:
 - a) *Viabilidad Económica:* encargada de la previsión de costes y beneficios del desarrollo.
 - b) *Viabilidad Legal:* realiza un análisis del contexto legal del desarrollo del trabajo, generalmente las licencias o la LOPD¹.

¹Ley Orgánica de Protección de Datos

A.2. Planificación temporal

Tal y como indicamos en la memoria, la metodología de gestión de proyectos empleada para el desarrollo ha sido **ágil**, *inspirada en Scrum* [7].

Existen, sin embargo, algunas consideraciones que deben ser tenidas en cuenta:

- Desarrollo **incremental** con *sprints*, reuniones de revisión y lanzamiento de una *versión*.
- Empleo de repositorio *git* como apoyo al desarrollo.
- Duración prevista de cada *sprint* limitada a un mes.
- Planificación y revisión del *sprint* anterior durante la reunión (alumno desarrollador y tutores).
- Confección de un *product backlog* de tareas pendientes, en colaboración con los tutores, y asignación al *sprint backlog*.
- Priorización de tareas al estilo *kanban*.

Cabe destacar que los *sprints* fueron definidos en el apartado *Milestones* de *GitHub* y las tareas como *issues* del mismo. Por otro lado, en base a los *Story Points*, se clasificó cada una de las tareas en función de su **complejidad y duración estimada**.

Sprints

Procedemos a desglosar el contenido de cada una de las **iteraciones**:

Sprint 0: 15 de Diciembre de 2022 a 15 de Enero de 2023

Comenzó con la reunión inicial donde se plantearon las bases del proyecto.

En esta fase se plantearon las siguientes metas:

- Redacción de la propuesta de proyecto, firma y entrega en repositorio Moodle para validación.
- Estimación de variables necesarias para los modelos.
- Reunión de documentación previa relacionada.
- Esbozo de planificación temporal.

Sprint 1: 15 de Enero de 2023 a 15 de Febrero de 2023

Tras la reunión que marcó el final de la iteración anterior, conseguimos formalizar nuestra asignación del trabajo de fin de grado según la propuesta y se alcanzaron los propósitos marcados.

Allí definimos las bases del siguiente Sprint:

- Creación del repositorio y asignación de colaboradores.
- Importación de datos, selección y preprocesado.
- Análisis estadístico inicial.

Sprint 2: 15 de Febrero de 2023 a 15 de Marzo de 2023

Tras validar los hitos del sprint previo, se promulgaron los objetivos del siguiente:

- Inicio de redacción de memoria del proyecto.
- Exploración de modelos predictivos.
- Análisis y documentación de resultados.
- Selección del modelo a explotar.

Sprint 3: 15 de Marzo de 2023 a 15 de Abril de 2023

En la reunión, comprobamos los resultados de los diferentes modelos de aprendizaje supervisado y escogimos el definitivo. Se validaron, por tanto, los pasos realizados en la iteración previa, planteando los siguientes:

- Continuar redacción de memoria del proyecto.
- Exploración de modelos de optimización.
- Implementación de modelos y análisis.
- Selección del algoritmo a explotar.

Sprint 4: 15 de Abril de 2023 a 15 de Mayo de 2023

Tras finalizar esta iteración, valoramos en la reunión los resultados y la manera de integrarlos en un producto para el cliente. Se decidió plantear como entregable final una API:

- Refactorizar clases y modelos.
- Familiarizarse con el diseño de una API con un *framework* de Python.
- Implementar una API siguiendo los objetivos y requerimientos.
- Pruebas de integración y sistema.
- Finalizar memoria.
- Comenzar anexos.

Sprint 5: 15 de Mayo de 2023 a 10 de Junio de 2023

En la última reunión se comprobaron y detallaron especificaciones del último entregable. Decidimos prolongar la entrega y añadir al entregable la implementación de la interfaz.

- Añadir funcionalidades. Refactorizaciones de código.
- Diseño de interfaz como aplicación web.
- Contenerización y migración a *cloud* de la API.
- Diseño de sistema de gestión de usuarios.
- Diseño e implementación de base de datos relacional.

Sprint 6: 10 de Junio de 2023 a 24 de Junio de 2023

El interfaz contaba ya con un sistema de gestión de usuarios y las pruebas de sistema con la API desarrollada y publicada en un servidor remoto fueron satisfactorias.

Planteamos el desarrollo durante esta iteración de:

- Implementación del sistema de gestión de predicciones.
- Implementación del sistema de gestión de planificaciones.

- Pruebas de integración con la API en servidor local y remoto.
- Pruebas de integración con la Base de Datos en servidor local y remoto.
- Despliegue de versión inicial de aplicación web en servidor remoto (AWS).
- Pruebas de sistema sobre el despliegue en AWS.
- Reestructurar memoria y anexos, añadiendo cambios.

Sprint 7: 24 de Junio de 2023 a 30 de Junio de 2023

El despliegue ha sido satisfactorio y muestra la funcionalidad requerida, cumpliendo los requerimientos funcionales y los casos de uso de mayor prioridad.

En esta iteración se propone:

- Completar anexos de la memoria, últimos apartados de diseño y manuales técnicos y de usuario.
- Pruebas de usuario en base a los requerimientos.
- Ampliar funcionalidad y refactorizar en base al resultado de las pruebas.
- Corrección de errores en documentos de memoria y anexos.

Sprint 8: 30 de Junio de 2023 a 4 de Julio de 2023

Documentación finalizada y revisada.

Proponemos:

- Confección de presentación de diapositivas para la defensa.
- Grabación de vídeo demostrativo.
- Grabación de vídeo de presentación.
- Última revisión y corrección de errores de última hora en los documentos entregables.
- Depósito de trabajo de fin de grado.

A.3. Estudio de viabilidad

Viabilidad económica

Repasaremos una aproximación a los **costes** y **beneficios** esperados si el proyecto hubiese sido desarrollado por un ente empresarial.

Estudio de Costes

Podemos considerar tres apartados principales en este estudio: **Personal**, **Equipamiento** y **Otros**.

En el primer apartado, consideraremos los costes de una empresa para contratar a un desarrollador a tiempo completo, tomando como referencia el salario anual, prorrateado a 5 meses, tal y como vemos en la tabla [A.1](#)

Concepto	Coste Anual	Prorrateo (5 meses)
<i>Salario Bruto</i>	24.000,00€	10.000,00€
<i>Retención IRPF</i>	3.252,00€	1.355,00€
<i>Seguridad Social</i>	1.524,00€	635,00€
<i>Salario Neto</i>	19.224,00€	8.010,00€

Tabla A.1: Costes de Personal

Hemos aplicado un *13,55 %* de retención sobre la nómina IRPF, considerando ausencia de deducciones tributarias.

Siguiendo la estela del análisis de costes, pasamos a analizar los empleados en el equipamiento (*hardware*, *licencias software*) para el desarrollo de esta herramienta, tal y como comprobamos en la tabla [A.2](#)

Concepto	Coste Total	Coste Amortizado(5 Meses)
<i>Ordenador Portátil</i>	1.330,00€	110,83€
<i>Licencia MS Windows 10</i>	279,00€	58,12€
<i>Suscripción Overleaf</i>	50,00€	50,00€
<i>Suscripción Internet</i>	278,00€	215,00€

Tabla A.2: Costes de Equipamiento

Por último, pasamos a realizar una estimación de aquellos **costes** que no encajan en ninguna de las otras categorías anteriores, en la table [A.3](#)

Concepto	Coste Total	Coste Amortizado(5 Meses)
<i>Consumo Eléctrico</i>	300,00€	99,99€
<i>Espacio de Trabajo</i>	4.000,00€	1333,33€
<i>Material de Oficina</i>	10,00€	8,00€

Tabla A.3: Costes Extra

Por último, se calculan los costes totales en base a los amortizados, en la tabla [A.4](#)

Concepto	Coste
<i>Costes de Personal</i>	10.000,00€
<i>Coste de Equipamiento</i>	433,95€
<i>Coste Extra</i>	1.441,32€
TOTAL	11.875,27€

Tabla A.4: Costes Totales

Análisis de Beneficios

El presente proyecto no ha sido concebido para su monetización, sino que se ofrece como un servicio *gratuito* a los gestores hospitalarios y a la comunidad científica y universitaria para explorarlo, analizarlo y adaptarlo a sus requerimientos.

Viabilidad legal

Podemos considerar dos enfoques principales en esta materia:

1. Manejo de bases de datos sanitarias.
2. Licencias

La Ley General de Sanidad establece, en su artículo 18, que una de las actuaciones del sistema de salud es la promoción de la investigación científica en el campo de la salud [4].

No obstante, todos los pacientes tienen derecho a que se preserve sus datos personales y obliga a la confidencialidad. La LOPD establece que sólo los facultativos que tienen acceso directo al tratamiento o diagnóstico del paciente pueden tener acceso a sus datos sanitarios.

Esto podría dificultar el acceso a datos sanitarios para investigación, no obstante, si estos datos están **anonimizados**, el tratamiento de la información estaría fuera de los requerimientos establecidos por la LOPD y, en este supuesto, se establece que los datos *anónimos* y los registros *anonimizados* pueden ser utilizados y cedidos SIN el consentimiento informado de los sujetos [4].

Este hecho se haya refrendado, aún en nuestros días, en la Ley Orgánica 3/2018 de Protección de Datos y Garantía de Derechos Digitales, donde vemos en su artículo 16.1: *«el acceso a la historia clínica con estos fines (de investigación o docencia) obliga a preservar los datos de identificación personal del paciente, separados de los de carácter clínico-asistencial, de manera que como regla general quede asegurado el anonimato, salvo que el propio paciente haya dado su consentimiento para no separarlos»*.

Creando dos paradigmas en cuanto al tratamiento de la información: uso de datos **disociados** o **no disociados**. En caso de que pertenezcan al *primer grupo* (los datos no permiten identificar de forma unívoca al individuo), el tratamiento de estos datos quedará *excluido* de la aplicación de la normativa en materia de Protección de Datos.

En nuestro proyecto, hemos hecho uso de datos **anonimizados** y **codificados**, tanto en su extracción como su posterior procesamiento y análisis posterior, por lo que podemos afirmar que nos encontramos dentro del marco *ético y legal* acerca del tratamiento de datos personales.

Por otro lado, el empleo de software implica el acceso y uso de herramientas registradas, especialmente librerías. Algunas de las usadas y las licencias que asocian pueden verse en la tabla A.5

Producto	Licencia
<i>Python</i>	OSI-Open Source
Pandas	BSD 3-Clause
Numpy	BSD
Scikit-Learn	BSD 3-Clause
Flask	BSD 3-Clause

Tabla A.5: Librerías y Licencias

Como podemos comprobar, nos encontramos ante una licencia BSD 3-clause, la cual es una *permisiva*, permitiendo a los desarrolladores modificar el software con toda la libertad que dispongan siempre que se incluya en él el copyright y la nota de licencia.

Por tanto, podemos afirmar que nuestro proyecto no dispone de **conflictos legales** que puedan interferir en la viabilidad del trabajo.

Por último, destacar que este proyecto ha sido realizado bajo una licencia **open source GPL-3.0**, cuyo archivo principal se encuentra en la raíz del repositorio, así como todas las características y disposiciones adicionales que se derivan de su uso.

Apéndice *B*

Especificación de Requisitos

B.1. Introducción

La redacción del catálogo de requisitos ha sido elaborada siguiendo las recomendaciones propuestas en el estándar *IEEE 830-1998* [1], y su propósito es **doble**: servir de contrato entre clientes y desarrolladores, así como documento base para el posterior análisis del sistema en desarrollo.

Según este estándar, toda especificación de requisitos debe cumplir con las siguientes **características**:

- **Correcta**: una especificación de requisitos es correcta si, y sólo si, todos y cada uno de los requerimientos que contiene *deben* estar presentes en nuestro proyecto de software.
- **Sin ambigüedades**: si sólo existe una única interpretación para cada uno de los requisitos descritos. Si un término puede tener más de un significado, debería incluirse un glosario que especifique la acepción a la que hacemos referencia en nuestro catálogo.
- **Completa**: contiene todos los requerimientos, junto a la definición de sus referencias.
- **Consistente**: se refiere a la *consistencia interna*, de tal modo que si un requerimiento no cumple con otro de mayor nivel de especificación (*requerimientos del sistema, por ejemplo*), entonces no será correcto.
- **Verificable**: si existe un proceso finito y costo-efectivo capaz de chequear que todos los requerimientos se cumplen.

- **Modificable:** será modificable si su estructura y estilo admiten cambios en su contenido de forma fácil, completa y consistente sin afectar a la estructura y el estilo subyacentes.
- **Trazable:** si el origen de cada requisito especificado es *claro* y facilita que sea referenciado en desarrollos futuros.

B.2. Objetivos generales

Este trabajo se ha realizado persiguiendo el cumplimiento de los siguientes **objetivos**:

- Se desea desarrollar un *sistema software* que permita a los usuarios la planificación de intervenciones quirúrgicas.
- Se desea que el sistema sea capaz de *predecir* la duración de una intervención quirúrgica.
- Se desea que el sistema *optimice* la planificación en base a la prioridad, la duración y las restricciones temporales descritas por los usuarios.
- Se desea *encapsular* el sistema en una API para dar libertad a los clientes en el desarrollo de su interfaz.
- Se desea *implementar* una aplicación web que ejemplifique el funcionamiento del sistema e integre: API, GUI y gestión de usuarios.

B.3. Catalogo de requisitos

En este apartado, desglosaremos los **requerimientos funcionales y no funcionales** en base al estándar [1] y los **objetivos** descritos en este capítulo.

Requisitos Funcionales

- **RF-1 Predicción de tiempo quirúrgico:** El sistema debe ser capaz de predecir la duración de una intervención quirúrgica.
 - **RF-1.1 Introducir datos:** La aplicación debe ser capaz de recibir un listado de datos de pacientes.

- **RF-1.1.1 Validar datos:** El sistema debe proporcionar *feedback* de validez de los datos introducidos al usuario.
 - **RF-1.2 Predecir duración:** Si los datos son válidos, el sistema debe predecir la duración del listado de intervenciones entregado.
 - **RF-1.3 Devolver resultados:** El sistema debe permitir al usuario obtener el listado introducido junto a la duración.
 - **RF-1.4 Exportar resultados:** El sistema debe permitir al usuario obtener el listado en un formato *estandarizable*.
- **RF-2 Planificación de intervenciones quirúrgicas:** El sistema debe ser capaz de planificar de forma *óptima o subóptima* un conjunto de intervenciones de entre un *set de posibles*, en una colección de quirófanos y en un número de días determinados.
 - **RF-2.1 Recoger datos:** La aplicación debe ser capaz de recoger un listado de datos relativos a pacientes.
 - **RF-2.1.1 Validar datos:** El sistema proporcionará *feedback* relativo a los datos recogidos.
 - **RF-2.1.2 Clasificar datos:** El sistema diferenciará las categorías en función de los datos que recogen la duración y los que no.
 - **RF-2.2 Obtener duración:** El sistema deberá obtener o predecir (*RF1*) la duración de cada una de los casos propuestos.
 - **RF-2.3 Obtener días y quirófanos:** El sistema debe permitir al usuario introducir un número de salas quirúrgicas y de días para establecer el horizonte temporal de la planificación.
 - **RF-2.4 Devolver resultados:** El sistema ofrecerá una planificación propuesta al usuario.
 - **RF-2.5 Exportar resultados:** El sistema debe permitir la exportación de resultados en un formato *estándar*.
- **RF-3 Gestión de Usuarios:** La aplicación debe ser capaz de gestionar usuarios.
 - **RF-3.1 Establecer roles de usuario:** El sistema debe diferenciar entre dos roles, administrador y usuario.
 - **RF-3.1.1 Gestionar permisos de usuario:** El sistema debe *limitar* el acceso a las funcionalidades en función del *rol* de usuario.

- **RF-3.2 Gestionar cuentas de usuario:** El sistema debe permitir que el administrador realice funciones de *creación, modificación y eliminación* de cuentas de usuario.
- **RF-4 Gestionar planificaciones:** El sistema debe permitir que los usuarios realicen labores de gestión de sus planificaciones quirúrgicas.
 - **RF-4.1 Listar planificaciones:** El usuario visualiza un listado con todas las planificaciones que ha realizado.
 - **RF-4.2 Visualizar planificaciones:** El usuario accede a una vista de la planificación realizada, tras su selección.
 - **RF-4.3 Eliminar planificaciones:** El usuario puede eliminar del listado las planificaciones que desee del sistema de almacenamiento persistente.

Requisitos No Funcionales

En este apartado, nos centraremos en aquellos apartados que reflejan las características *no funcionales*, tales como la *usabilidad, flexibilidad o rendimiento* que debemos esperar de este proyecto.[3]

- **RNF-1 Rendimiento:** el sistema debe poseer unos tiempos de cálculo y carga aceptables para un navegador web y un servidor sin capacidad de cómputo extra contratada.
- **RNF-2 Escalabilidad:** la aplicación debe permitir la adición de nuevas funciones de forma fácil y transparente para el desarrollador.
- **RNF-3 Seguridad:** los datos privados y sensibles, como *contraseñas* deben ser gestionadas de la forma adecuada.
- **RNF-4 Disponibilidad:** el sistema debe estar disponible para su uso para todo navegador compatible con *HTML5* y conexión a *internet*.
- **RNF-5 Usabilidad:** el interfaz será *user friendly*, intuitivo para los usuarios y dotado de un modelo de aprendizaje sencillo acerca de las funcionalidades ofrecidas.
- **RNF-6 Mantenibilidad:** el patrón de desarrollo debe permitir su fácil mantenimiento posterior.

B.4. Especificación de requisitos

Presentamos en este apartado el **diagrama de casos de uso** en la figura B.1

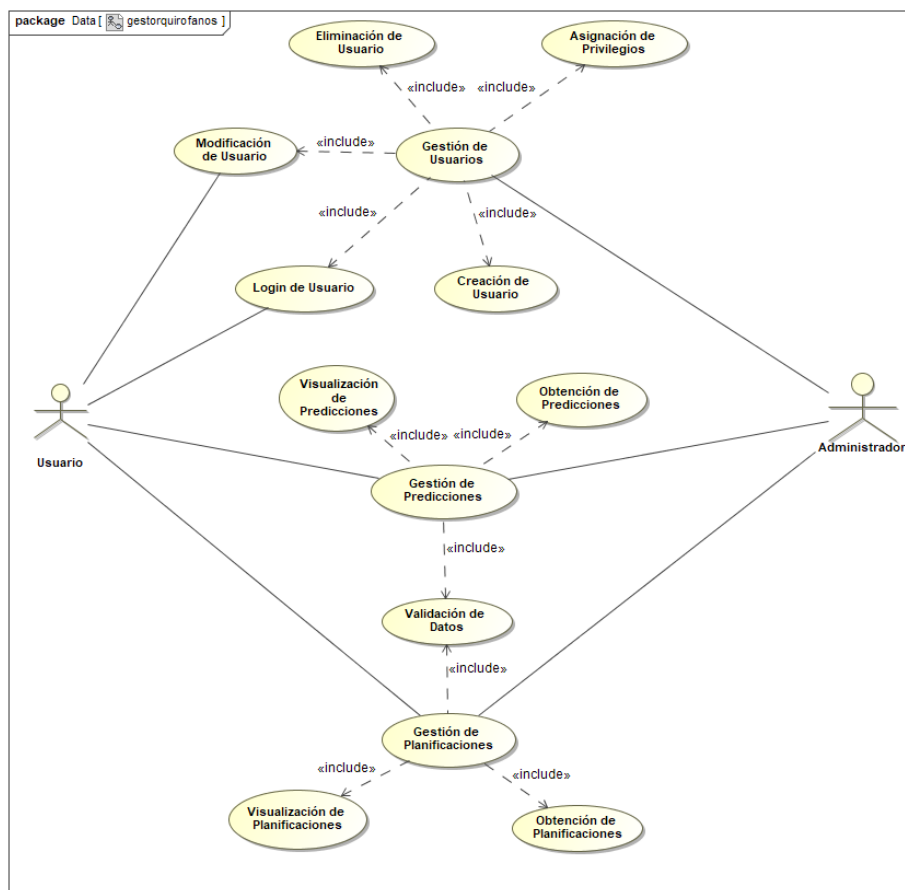


Figura B.1: Diagrama de Casos de uso

Y comenzamos a especificar cada uno de ellos:

CU-1	Gestión de Usuarios
Versión	1.0
Autor	Jesús García Armario
Requisitos asociados	RF-3
Descripción	Engloba las acciones de creación, modificación, identificación y eliminación de usuarios.
Precondición	Existe una base de datos disponible. Existe un usuario con rol de administrador.
Acciones	<ol style="list-style-type: none"> 1. Identificación del usuario. 2. Visualización de perfil. 3. Opción a editar perfil. 4. Panel de gestión de usuarios (administrador) <ol style="list-style-type: none"> a) Añadir usuarios. b) Modificar usuarios. c) Eliminar usuarios. 5. Opción a cerrar sesión.
Postcondición	Mensaje de bienvenida al usuario. Mensajes de <i>feedback</i> ante las acciones.
Excepciones	Usuario inexistente (mensaje de error). Contraseña incorrecta (mensaje de error). Privilegios insuficientes (redirección).
Importancia	Alta

Tabla B.1: CU-1 Gestión de Usuarios.

CU-2	Creación de Usuarios
Versión	1.0
Autor	Jesús García Armario
Requisitos asociados	RF-3.1; RF-3.2
Descripción	Se encarga de la acción de crear un usuario.
Precondición	Existe una base de datos disponible. Existe un usuario con rol de administrador.
Acciones	<ol style="list-style-type: none"> 1. Identificación del usuario. 2. Panel de gestión de usuarios (administrador) 3. Botón de creación de usuario. 4. Se rellenan datos de usuario. 5. Confirmación y creación de usuario.
Postcondición	Mensaje de creación satisfactoria. Actualización de la base de datos.
Excepciones	Campos de usuario no introducidos(mensaje de error). Privilegios insuficientes (redirección).
Importancia	Media

Tabla B.2: CU-2 Creación de Usuarios.

CU-3	Eliminación de Usuarios
Versión	1.0
Autor	Jesús García Armario
Requisitos asociados	RF-3.2
Descripción	Se encarga de la acción de eliminar un usuario.
Precondición	Existe una base de datos disponible. Existe un usuario con rol de administrador.
Acciones	<ol style="list-style-type: none"> 1. Identificación del usuario. 2. Panel de gestión de usuarios (administrador) 3. Se muestra listado de usuarios activos. 4. Botón de eliminar usuario.
Postcondición	Mensaje de eliminación satisfactoria. Actualización de la base de datos.
Excepciones	Base de datos no accesible (mensaje de error)
Importancia	Baja

Tabla B.3: CU-3 Eliminación de Usuarios.

CU-4	Modificación de Usuarios
Versión	1.0
Autor	Jesús García Armario
Requisitos asociados	RF-3.2
Descripción	Se encarga de la acción de modificar un usuario.
Precondición	Existe una base de datos disponible.
Acciones	<ol style="list-style-type: none"> 1. Identificación del usuario. 2. Perfil de usuario. 3. Se muestran las características del perfil. 4. Botón de modificar usuario. 5. Formulario de cambio de características.
Postcondición	Mensaje de modificación satisfactoria. Actualización de la base de datos.
Excepciones	Base de datos no accesible (mensaje de error) Campos incorrectos (mensaje de error)
Importancia	Baja

Tabla B.4: CU-4 Modificación de Usuarios.

CU-5	Login de Usuarios
Versión	1.0
Autor	Jesús García Armario
Requisitos asociados	RF-3.1
Descripción	Se encarga de iniciar sesión en el sistema
Precondición	Existe una base de datos disponible.
Acciones	<ol style="list-style-type: none"> 1. Solicitud de email y contraseña. 2. Encriptación de contraseña introducida. 3. Acceso a base de datos. 4. Comparación de email y contraseña introducidos con los almacenados. 5. Comprobación de rol de usuario. 6. Redirección a sesión en función del rol almacenado.
Postcondición	Redirección.
Excepciones	Email inexistente en base de datos (mensaje de error: Usuario inexistente) Contraseña introducida no coincide (mensaje de error: Contraseña incorrecta)
Importancia	Alta

Tabla B.5: CU-5 Login de Usuarios.

CU-6	Asignación de Privilegios
Versión	1.0
Autor	Jesús García Armario
Requisitos asociados	RF-3.1.1
Descripción	Se encarga de administrar los roles de usuario.
Precondición	Existe una base de datos disponible. Existe un usuario con rol de administrador.
Acciones	<ol style="list-style-type: none"> 1. Identificación de usuario administrador 2. Acceso a función de agregar/modificar usuario. 3. Selección de opción de administrador. 4. Inserción/Actualización en base de datos
Postcondición	Mensaje de modificación/creación satisfactoria. Actualización de la base de datos.
Excepciones	Base de datos no accesible (mensaje de error) Campos incorrectos (mensaje de error)
Importancia	Media

Tabla B.6: CU-6 Asignación de Privilegios.

CU-7	Gestión de Predicciones
Versión	1.0
Autor	Jesús García Armario
Requisitos asociados	RF-1
Descripción	Se encarga de predecir la duración de una intervención quirúrgica.
Precondición	Existe una base de datos disponible. Existe un usuario activo. Existe conexión con un servicio de predicción basado en ML.
Acciones	<ol style="list-style-type: none"> 1. Identificación de usuario. 2. Acceso a función de gestión de predicciones. 3. Visualización de listado de predicciones previas del usuario. 4. Posibilidad de visualizar, eliminar o agregar predicción. 5. Recepción de mensajes de feedback en función de la opción seleccionada. 6. Actualización y persistencia en base de datos.
Postcondición	Mensaje de modificación/creación/eliminación satisfactoria. Actualización de la base de datos.
Excepciones	Base de datos no accesible (mensaje de error) Fichero incorrecto (mensaje de error)
Importancia	Alta

Tabla B.7: CU-7 Gestión de Predicciones.

CU-8	Visualización de Predicciones
Versión	1.0
Autor	Jesús García Armario
Requisitos asociados	RF-1.3; RF-1.4; RF-2.2
Descripción	Se encarga de visualizar las predicciones almacenadas en la base de datos.
Precondición	Existe una base de datos disponible. Existe un usuario activo. Existen predicciones almacenadas en la base de datos.
Acciones	<ol style="list-style-type: none">1. Identificación de usuario.2. Acceso a función de gestión de predicciones.3. Visualización de listado de predicciones previas del usuario.4. Selección de opción de visualizar.5. Se muestra un listado formateado por pantalla.
Postcondición	Redirección a listado formateado.
Excepciones	Base de datos no accesible (mensaje de error)
Importancia	Media

Tabla B.8: CU-8 Visualización de Predicciones.

CU-9	Obtención de Predicciones
Versión	1.0
Autor	Jesús García Armario
Requisitos asociados	RF-1.2; RF-1.3; RF-2.1 ; RF-2.2
Descripción	Se encarga de obtener la predicción de duración.
Precondición	Existe una base de datos disponible. Existe un usuario activo. Existe conexión con un servicio de predicción basado en ML.
Acciones	<ol style="list-style-type: none"> 1. Identificación de usuario. 2. Acceso a función de gestión de predicciones. 3. Visualización de listado de predicciones previas del usuario. 4. Funcionalidad de agregar predicción. 5. Selección de archivo de nuestro sistema. 6. Conexión con API de predicción. 7. Validación de contenido del fichero y cálculo de predicción en la API. 8. Recepción de respuesta.
Postcondición	Mensaje de creación satisfactoria. Actualización de la base de datos.
Excepciones	Base de datos no accesible (mensaje de error) Fichero incorrecto (mensaje de error) Campos del fichero incorrectos (mensaje de error)
Importancia	Alta

Tabla B.9: CU-9 Obtención de Predicciones.

CU-10	Validación de Datos
Versión	1.0
Autor	Jesús García Armario
Requisitos asociados	RF-2.1.1; RF-2.1.2
Descripción	Se encarga de validar los ficheros enviados a la API.
Precondición	Existe una base de datos disponible. Existe un usuario activo. Existe conexión con una API gestora de los servicios de predicción y planificación.
Acciones	<ol style="list-style-type: none"> 1. Identificación de usuario. 2. Acceso a función de gestión de predicciones y/o planificaciones. 3. Envío de fichero a API. 4. Recepción de mensajes de feedback en función de las características del fichero introducido.
Postcondición	Mensaje de envío satisfactorio. Actualización de la base de datos.
Excepciones	Base de datos no accesible (mensaje de error) Fichero incorrecto (mensaje de error) Extensión de fichero incorrecta (mensaje de error).
Importancia	Baja

Tabla B.10: CU-10 Validación de Datos.

CU-11	Gestión de Planificaciones
Versión	1.0
Autor	Jesús García Armario
Requisitos asociados	RF-2; RF-4
Descripción	Se encarga de obtener y visualizar las planificaciones propuestas por el sistema .
Precondición	Existe una base de datos disponible. Existe un usuario activo. Existe conexión con una API gestora de los servicios de predicción y planificación.
Acciones	<ol style="list-style-type: none"> 1. Identificación de usuario. 2. Acceso a función de gestión de planificaciones. 3. Visualización de un listado de planificaciones. 4. Opciones de añadir, visualizar o eliminar una planificación. 5. Recepción de mensajes de feedback en función de la opción seleccionada.
Postcondición	Mensaje/s de solicitud/es satisfactorias.
Excepciones	Base de datos no accesible (mensaje de error) Parámetros incorrectos (mensaje de error)
Importancia	Alta

Tabla B.11: CU-11 Gestión de Planificaciones.

CU-12	Obtención de Planificaciones
Versión	1.0
Autor	Jesús García Armario
Requisitos asociados	RF-2.2; RF-2.3; RF-2.4; RF-4.1
Descripción	Se encarga de obtener las planificaciones propuestas por el sistema .
Precondición	Existe una base de datos disponible. Existe un usuario activo. Existe conexión con una API gestora de los servicios de predicción y planificación.
Acciones	<ol style="list-style-type: none"> 1. Identificación de usuario. 2. Acceso a función de gestión de planificaciones. 3. Visualización de un listado de planificaciones. 4. Opción de añadir una nueva planificación. 5. Envío de archivo a API. 6. Recepción de respuesta.
Postcondición	Mensaje de solicitud satisfactoria. Actualización de la base de datos.
Excepciones	Base de datos no accesible (mensaje de error) Parámetros incorrectos (mensaje de error)
Importancia	Alta

Tabla B.12: CU-12 Obtención de Planificaciones.

CU-13	Visualización de Planificaciones
Versión	1.0
Autor	Jesús García Armario
Requisitos asociados	RF-2.5; RF-4.2
Descripción	Se encarga de visualizar las planificaciones propuestas por el sistema .
Precondición	Existe una base de datos disponible. Existe un usuario activo. Existe conexión con una API gestora de los servicios de predicción y planificación.
Acciones	<ol style="list-style-type: none"> 1. Identificación de usuario. 2. Acceso a función de gestión de planificaciones. 3. Visualización de un listado de planificaciones. 4. Opción de visualizar una planificación. 5. Se visualiza propuesta formateada por pantalla.
Postcondición	Visualización de propuesta en formato de agenda temporal por pantalla.
Excepciones	Base de datos no accesible (mensaje de error)
Importancia	Media

Tabla B.13: CU-13 Visualización de Planificaciones.

Apéndice C

Especificación de diseño

C.1. Introducción

En este apartado desglosaremos las estrategias de diseño tomadas en consideración: *conjuntos de datos, clases, procedimientos..*, para el cumplimiento de los requerimientos funcionales y no funcionales reseñados en el primer apartado.

C.2. Diseño de datos

Entidades

- **Usuario (*User*):** Consta de un identificador auto-incrementado como clave primaria, así como atributos para el *nombre*, su *correo electrónico*, contraseña *encriptada* y fechas de *creación* y *modificación*.
- **Planificación:** Consta de un identificador como clave primaria, el identificador del usuario que la creó (*FK*) y la fecha de creación.
- **Predicción:** Sigue la misma estructura de atributos que la entidad *Planificación*.

Diagrama Relacional

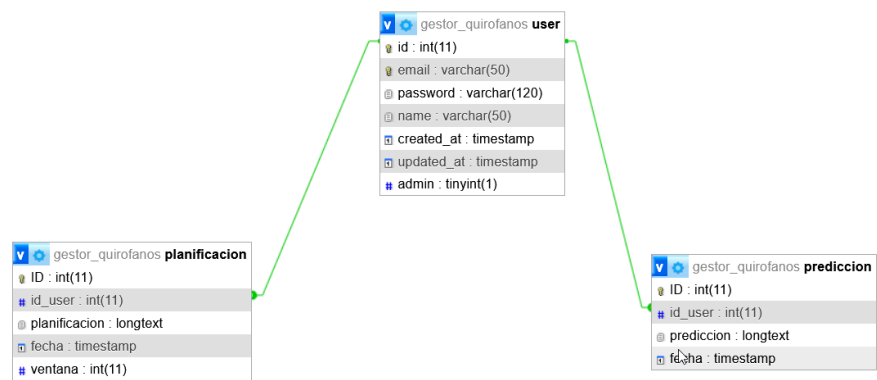


Figura C.1: Diagrama Relacional

Diagrama E-R

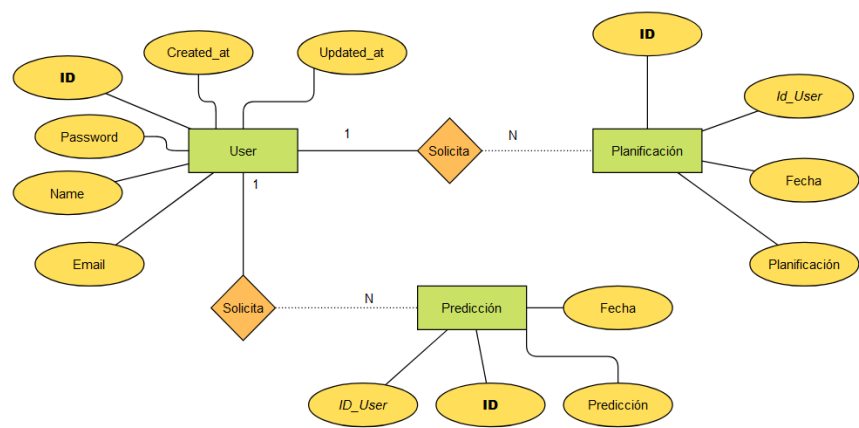


Figura C.2: Diagrama Relacional

C.3. Diseño procedimental

En este apartado, se desgranar los procedimientos que permiten especificar el funcionamiento interno de las aplicaciones.

Existen muchos tipos de diagramas que pueden representar estas funcionalidades, aunque para esta ocasión hemos elegido los *diagramas de interacción* y, dentro de éstos, los de *secuencia* [2]. En él, registraremos el comportamiento de nuestro sistema mediante una secuencia de eventos **ordenados por tiempo**.

En primer lugar, reflejaremos la acción de *modificar el perfil* de un usuario en nuestro interfaz, con la secuencia de acciones representadas en el diagrama **C.3**

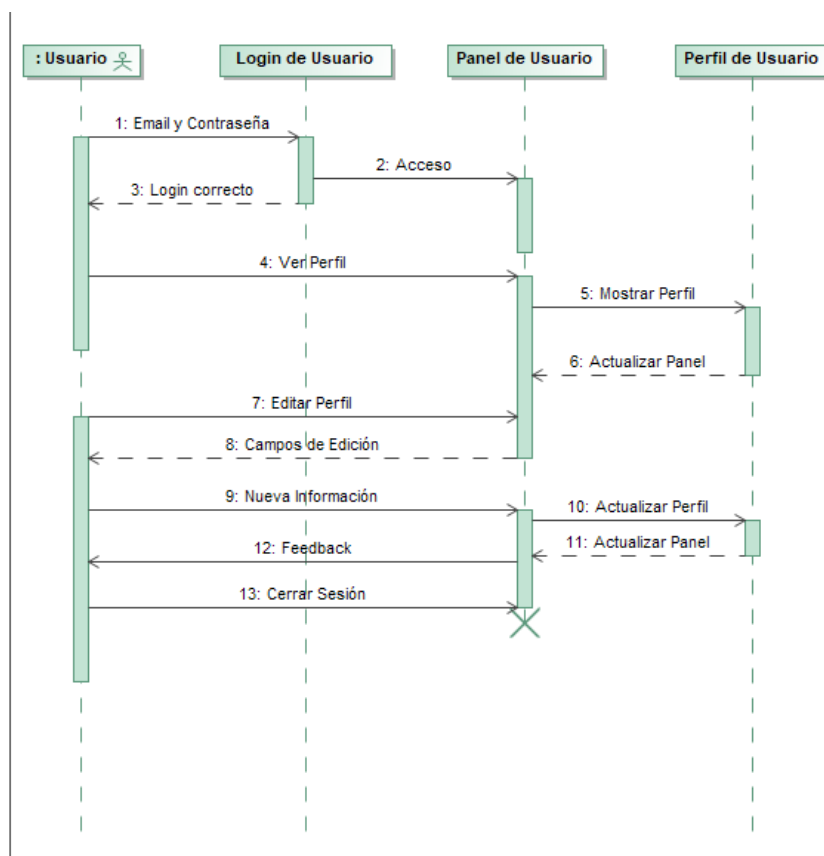


Figura C.3: Diag Int-Seg: Modificar Perfil de Usuario

Mostramos ahora las acciones que puede realizar el usuario **administrador** sobre las acciones de creación, modificación y eliminación de usuario, como vemos en C.4

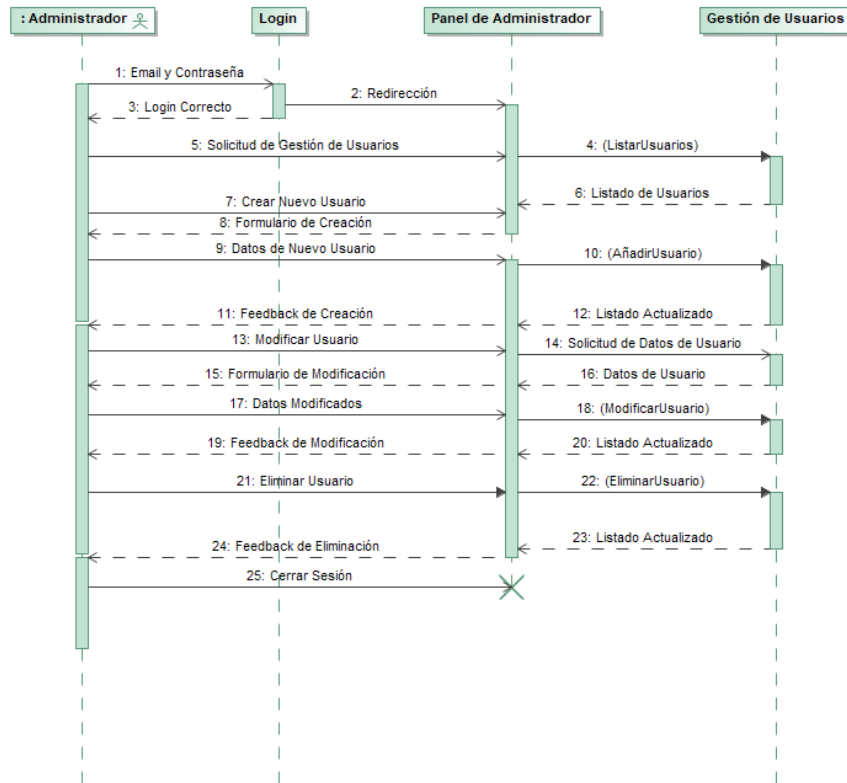


Figura C.4: Diag Int-Seg: Gestión de Usuarios

Dentro de este proyecto, podemos diferenciar dos **funcionalidades**: Planificación de Intervenciones y Predicción de duración, tanto de forma *independiente* como *interrelacionada* (llamada de una funcionalidad a otra en caso de ser requerida).

El usuario puede *crear, visualizar o eliminar* una determinada **predicción**, siguiendo el siguiente esquema (partiendo del supuesto de un inicio de sesión exitoso), en el diagrama C.5

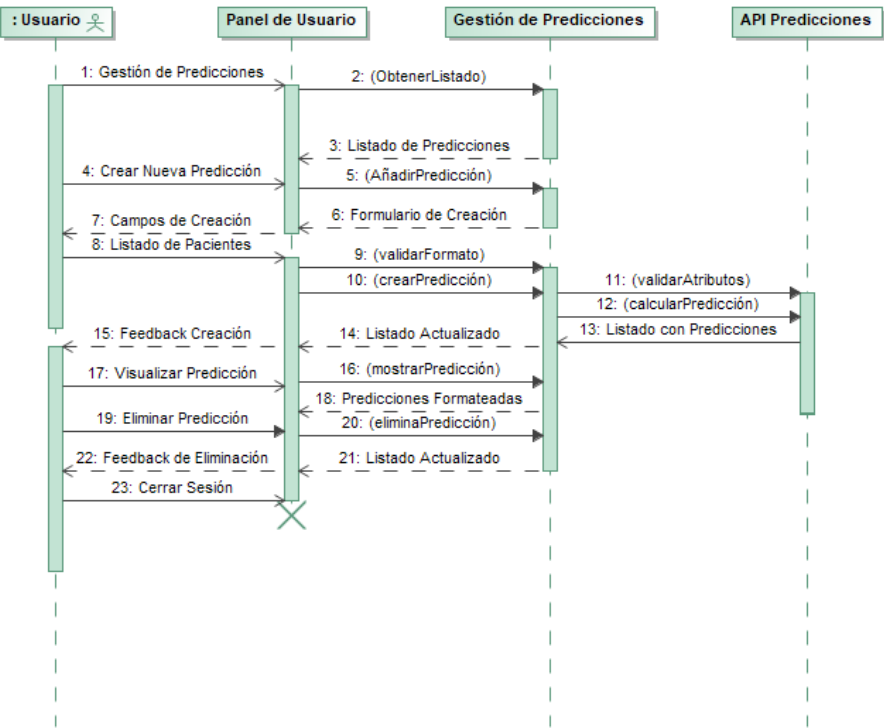


Figura C.5: Diag Int-Seg: Gestión de Predicciones

De modo similar, gestionaremos las **planificaciones**, con las acciones previstas en el diagrama [C.6](#)

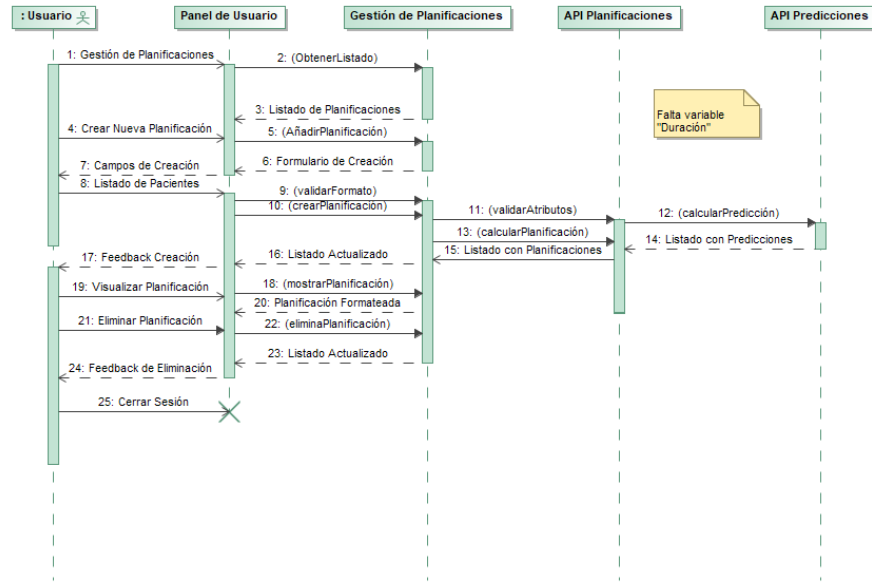


Figura C.6: Diag Int-Seg: Gestión de Planificaciones

C.4. Diseño arquitectónico

Al haber optado para el despliegue por el desarrollo tanto de una API como proveedora de servicios como por una *aplicación web* que sirva de GUI para los usuarios clientes, se han seguido los patrones **cliente-servidor** y **MVC** (*Modelo Vista Controlador*) en el diseño de la arquitectura de este proyecto software.

Modelo Cliente-Servidor

La arquitectura cliente-servidor, nos presenta una serie de *ventajas de diseño*[6]:

- A Facilita el **mantenimiento**, dado que los roles se encuentran distribuidos a lo largo de varios servidores independientes.
- B Permite una fácil **escalabilidad** y modularidad del sistema.
- C Los datos están centralizados, de forma que varios clientes distintos pueden acceder desde diversas localizaciones. La existencia de recursos

compartidos facilita a su vez la gestión, modificación y **reutilización** de módulos software.

Siguiendo este modelo, y aplicado al sistema particular desarrollado en el proyecto, podríamos definir **dos modelos** cliente-servidor:

- La comunicación entre el **interfaz web y la API** proveedora de servicios de predicción y planificación, referenciada en la figura C.7

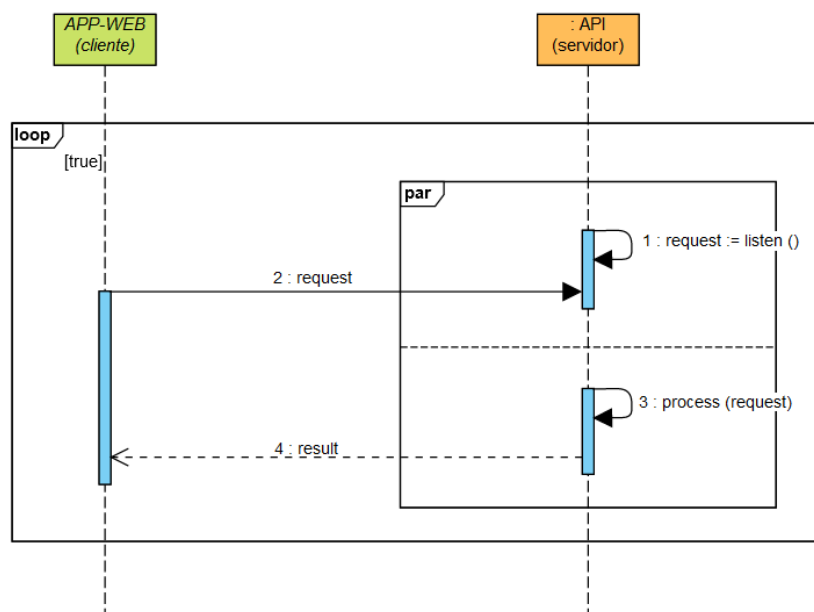


Figura C.7: Esquema Cliente-Servidor de APP-WEB y API

- Por otra parte, definiremos la comunicación entre los **usuarios clientes** y la **aplicación web distribuida**, como vemos en la imagen C.8

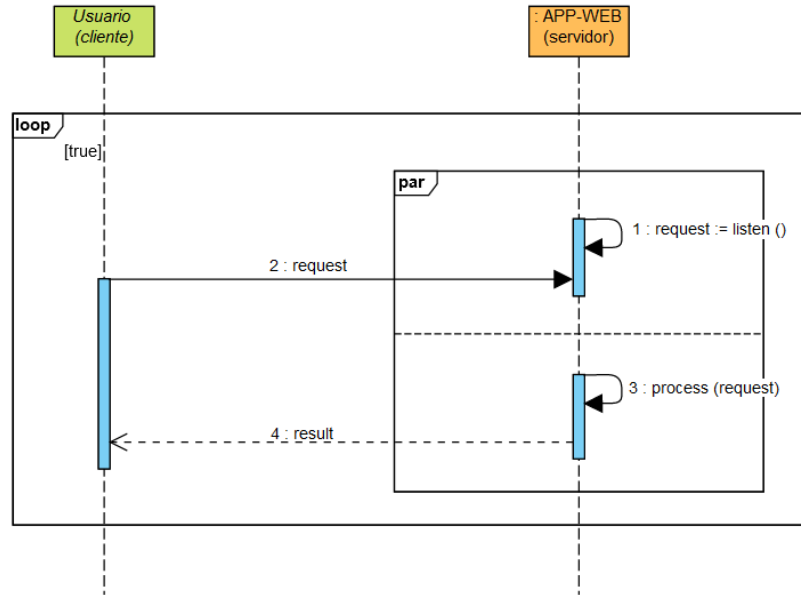


Figura C.8: Esquema Cliente-Servidor de Usuario y APP-WEB

Modelo-Vista-Controlador

En este caso, debido a la implementación de una base de datos, así como de dos servicios computacionales *externalizados*, parece ser adecuado separar la **vista** del **modelo de datos** subyacente [5].

Aquí, tanto el interfaz de usuario, como la lógica de negocio y la integración con otros sistemas y subsistemas han sido desarrolladas en Python, y desplegadas en servidores externos a partir de las funcionalidades que ofrece Amazon Web Services:

- **AWS Elastic Container Services:** para ejecutar los servidores y las tareas.
- **AWS RDS:** para almacenamiento y lógica de base de datos relacional.
- **AWS Elastic Container Registry:** para almacenamiento persistente de los *contenedores* contruidos a partir de los ficheros fuente de la aplicación.

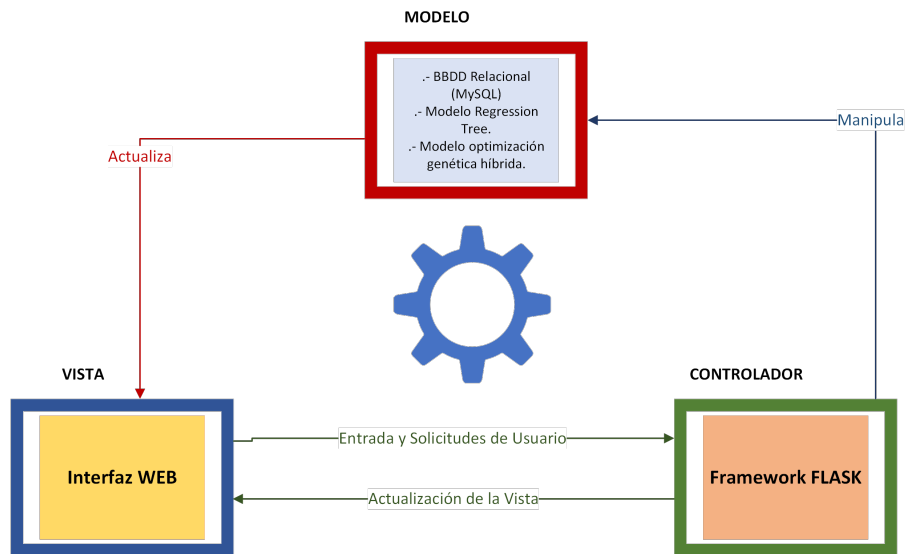


Figura C.9: Modelo-Vista-Controlador

Podemos comprobar cómo el *Framework Flask* es el encargado de aportar la solución al desarrollo del interfaz web siguiendo este modelo, comportándose como el controlador y aislando el modelo de datos de la vista.

Diseño de Paquetes

Para facilitar la *legibilidad* del código, tratamos de ofrecer una estructuración del sistema en dos *subsistemas* (API y APP-WEB), comunicados según el patrón cliente-servidor, de forma distribuida, y dividiendo los paquetes de cada uno de ellos siguiendo un *enfoque por características*.

En este enfoque, todas las clases requeridas para una misma funcionalidad se encuentran en el mismo paquete, consiguiendo una **alta cohesión** entre las clases de un mismo paquete y un **acoplamiento reducido** entre diferentes paquetes.

Así, podemos comprobar la notación de los paquetes y cómo se distribuyen según su funcionalidad de forma jerárquica y caracterizable, diferenciando el diagrama de paquetes para el subsistema API [C.10](#) y el del interfaz web [C.11](#)

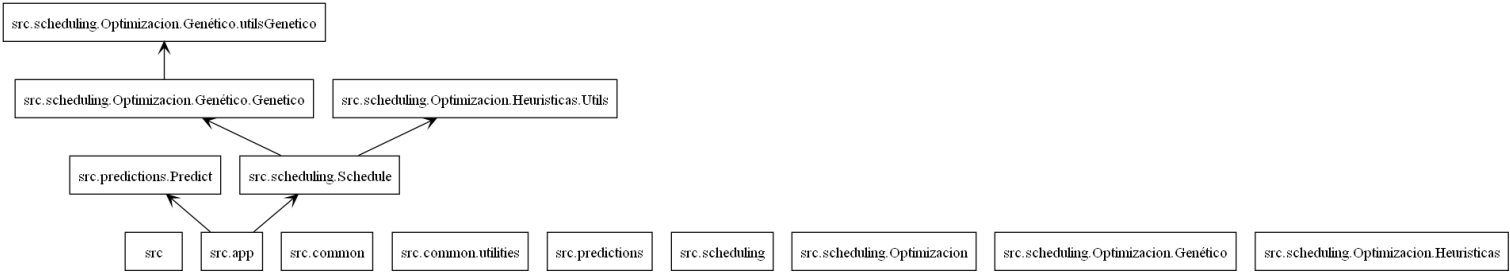


Figura C.10: Diagrama de paquetes API

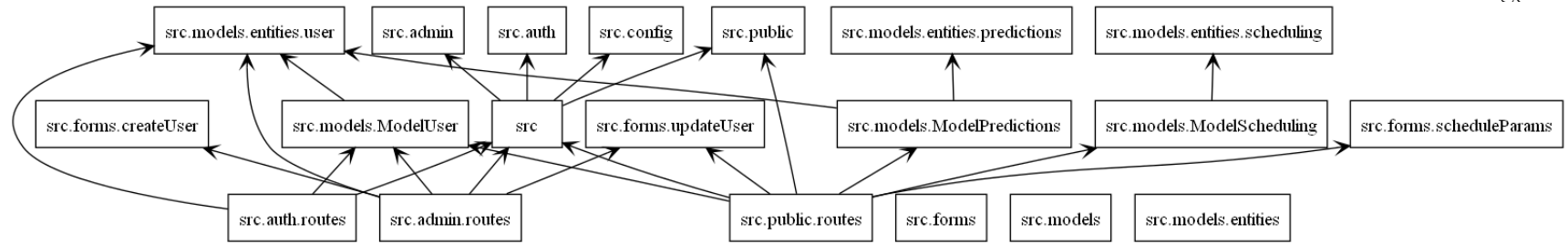


Figura C.11: Diagrama de paquetes APP Web

Por último, podemos ver el diagrama de clases desglosadas dentro de cada subsistema, y localizar su posición en el paquete a partir de los diagramas de paquetes mostrados con anterioridad, siguiendo la notación ***dot***, tanto para la API ([C.12](#), [C.13](#)) como para la aplicación web ([C.14](#), [C.15](#))

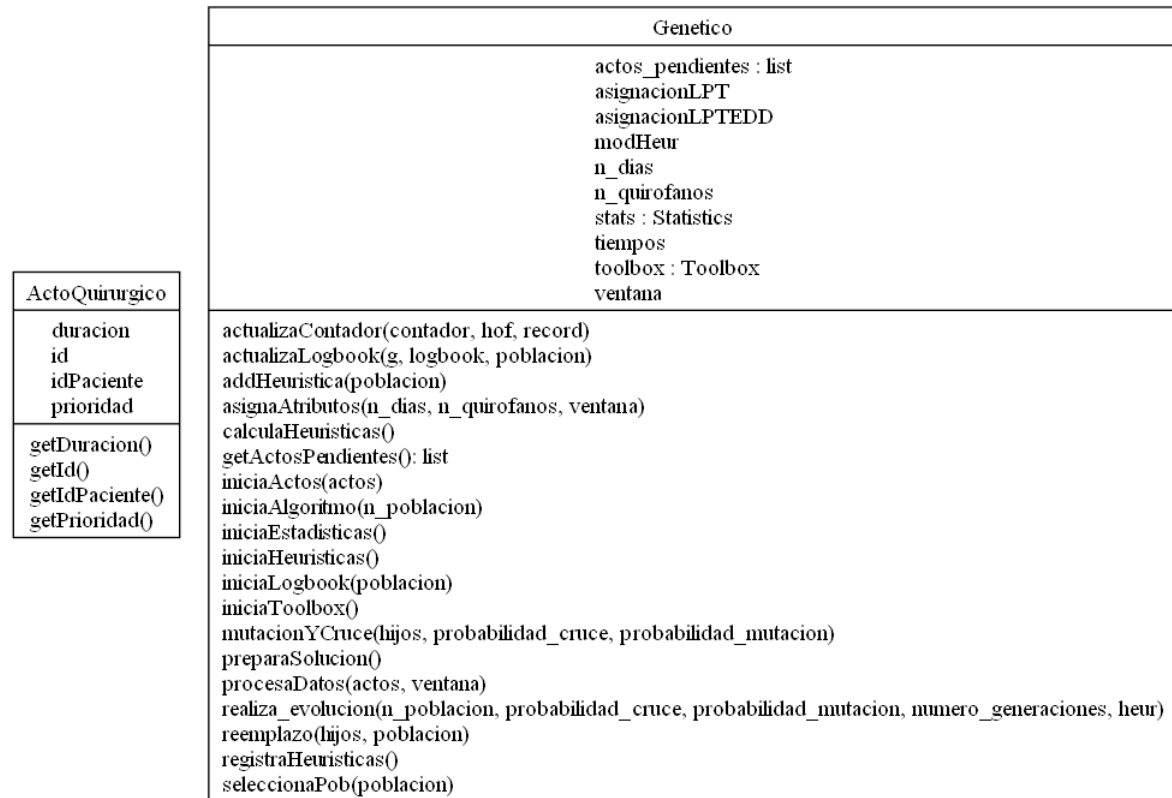


Figura C.12: Diagrama de clases del subsistema API - 1

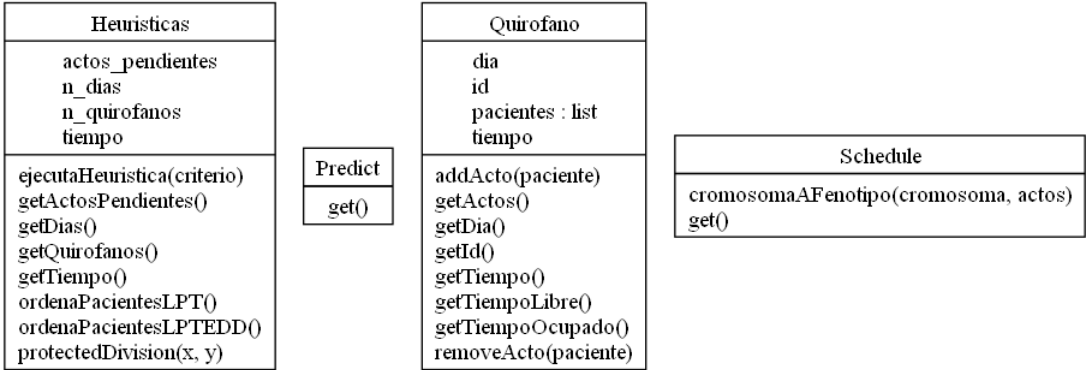


Figura C.13: Diagrama de clases del subsistema API - 2

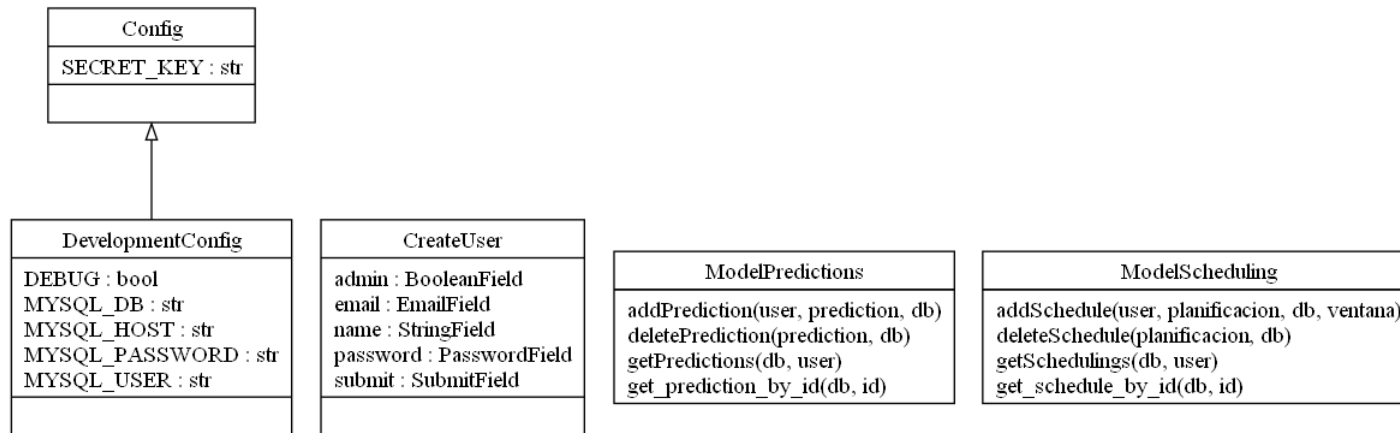


Figura C.14: Diagrama de clases del subsistema APP-WEB - 1

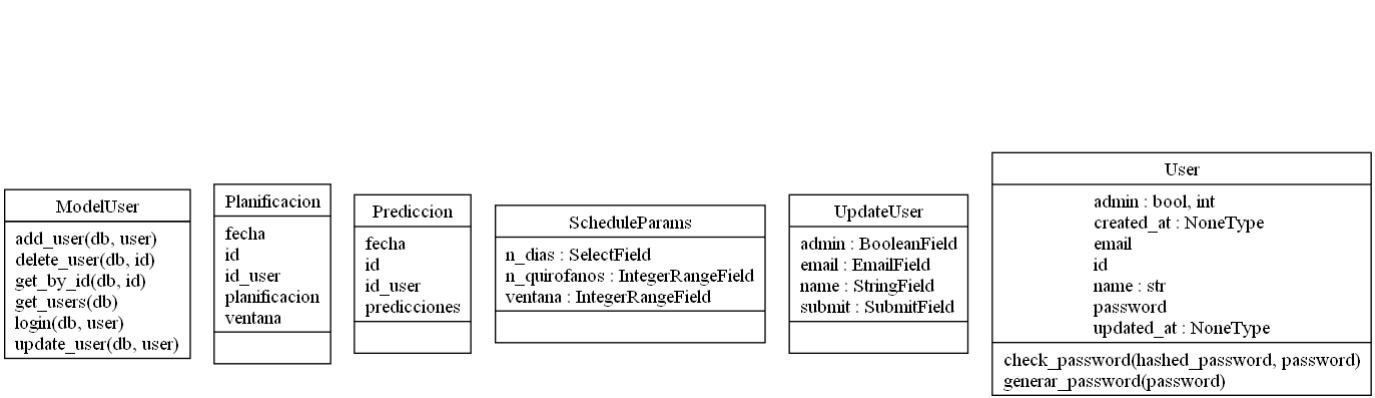


Figura C.15: Diagrama de clases del subsistema APP-WEB - 2

Apéndice *D*

Documentación técnica de programación

D.1. Introducción

Para realizar un correcto **análisis** del desarrollo de este proyecto, son necesarias algunas consideraciones respecto al entorno de desarrollo, las dependencias funcionales y las opciones de integración y despliegue.

D.2. Estructura de directorios

Dentro del repositorio, podemos encontrar:

- `./`: Directorio raíz. Contiene la licencia, el logo, el fichero *léeme* para la página principal de GitHub, la estructura de base datos para su importación y todos los paquetes y directorios que conforman el proyecto.
- `./API`: Sistema que contiene la API con los servicios de Predicción y Planificación quirúrgica.
- `./API/uploads`: Directorio de almacenamiento temporal que recoge los archivos enviados por los usuarios antes de ser procesados y eliminados.
- `./API/src`: Código fuente de la API.

- *./API/src/scheduling*: Contiene las clases y paquetes encargados de la tarea de planificación, así como el archivo Dockerfile para ser encapsulado.
- *./API/src/scheduling/Optimizacion*: Directorio con clases y paquetes con los algoritmos de optimización y planificación.
- *./API/src/scheduling/Optimizacion/Genético*: Clases para el algoritmo genético.
- *./API/src/scheduling/Optimizacion/Heurísticas*: Clases para las heurísticas de planificación.
- *./API/src/predictions*: Directorio con clases encargadas de cargar el modelo de ML predictivo.
- *./API/src/common*: Clases comunes a todos los paquetes. Encargados fundamentalmente de tareas de procesamiento de datos y ficheros.
- *./APP-WEB*: Contiene el Dockerfile y el código fuente de la interfaz web.
- *./APP-WEB/src*: Código fuente de la aplicación.
- *./APP-WEB/src/admin*: Contiene las rutas a las funcionalidades del usuario administrador.
- *./APP-WEB/src/auth*: Contiene las rutas para la funcionalidad de autenticación en sistema.
- *./APP-WEB/src/forms*: Contiene los formularios en formato Flask WTF para incluirlos en las plantillas HTML.
- *./APP-WEB/src/models*: Clases y paquetes que encapsulan el modelo de comunicación con la base de datos.
- *./APP-WEB/src/models/entities*: Contiene las clases que representan las entidades persistentes de la BBDD.
- *./APP-WEB/src/public*: Contiene las rutas para todas las funcionalidades disponibles para los usuarios.
- *./APP-WEB/src/static*: Contiene los archivos de formato de estilos, imágenes... a incluir en las plantillas estáticas.

- *./APP-WEB/src/templates*: Contiene las plantillas en lenguaje de marcado HTML, que se corresponden con la *Vista* del usuario.
- *./APP-WEB/src/templates/admin*: Plantillas dedicadas a la vista del administrador.
- *./APP-WEB/src/templates/user*: Plantillas para la vista de cualquier usuario.
- *./APP-WEB/src/templates/auth*: Plantillas para la función de identificación en el sistema.
- *./Documentación*: Se incluye la memoria, anexos y diagramas.
- *./Documentación/img*: Ruta de las imágenes que se encuentran en el entregable.
- *./Documentación/tex*: Capítulos y apartados de memoria y anexos, en formato latex.
- *./Documentación/UML*: Directorio con diagramas de casos de uso, interacción y secuencia.
- *./Experimentación*: Colección de Jupyter Notebooks que ilustran el proceso de investigación llevado a cabo hasta obtener la solución propuesta.
- *./Experimentación/Datos*: Colección de conjuntos de datos anonimizados para labores de ML.
- *./Experimentación/Modelos*: Análisis, diseño y explotación de modelos predictivos de aprendizaje supervisado.
- *./Experimentación/Optimización*: Análisis, diseño y explotación de algoritmos de planificación paralela.
- *./Experimentación/Preprocesado*: Herramientas de preprocesamiento de datos a partir de los listados originales.
- *./Preprocesado*: Clase con utilidades para estandarizar y homogeneizar las fuentes de datos en un formato compatible con los modelos propuestos.

D.3. Manual del programador

En este apartado desglosaremos la estructura del sistema de forma que sirva de referencia para futuros desarrolladores para su análisis y contribución al proyecto.

Para ello, haremos referencia al **entorno y dependencias** necesarias para el desarrollo, la obtención del **código fuente**, su **ejecución** y posterior **exportación**.

Entorno de desarrollo

Para comenzar con la *explotación* del sistema, se recomiendan las siguientes herramientas y dependencias:

Python 3.8-3.11

En nuestro caso hemos usado la última versión ofrecida por **Anaconda**, pues ya incluye gran parte de las librerías necesarias para las tareas de análisis y minería de datos, cuya guía de instalación se encuentra disponible [aquí](#).

Virtualenv

Incluido en la mayor parte de distribuciones de Python (*incluido en Anaconda*). Permite trabajar con entornos *virtuales*, de gran utilidad cuando construimos subsistemas exportables y queremos **acotar** las librerías necesarias para cada entorno.

En caso de no estar disponible, puede instalarse con el comando pip: `pip -U install virtualenv`

IDEs

Se recomienda la instalación de una interfaz de apoyo al desarrollo compatible con Python. Recomendamos la instalación de **Visual Studio Code**, la **extensión de Python** y el **plugin de plantillas Bootstrap 5**.

Por otra parte, el IDE Pycharm, en su versión **Professional**, incluye soporte y apoyo al desarrollo web con Flask, añadiendo integración con HTML, JS y SQL y está disponible de forma *gratuita* para la comunidad educativa y es accesible desde [aquí](#).

Sin embargo, pese a haber explorado ambos IDE para la confección del código fuente, la totalidad del proyecto puede ser desarrollado desde cualquiera de ellos, sin necesidad de combinar su uso.

Git

Necesario para hacer uso del **repositorio**. Nuestro repositorio es público, recomendándose realizar un *fork* del mismo en una cuenta privada de *GitHub* y trabajar desde una copia privada del mismo en nuestro entorno:

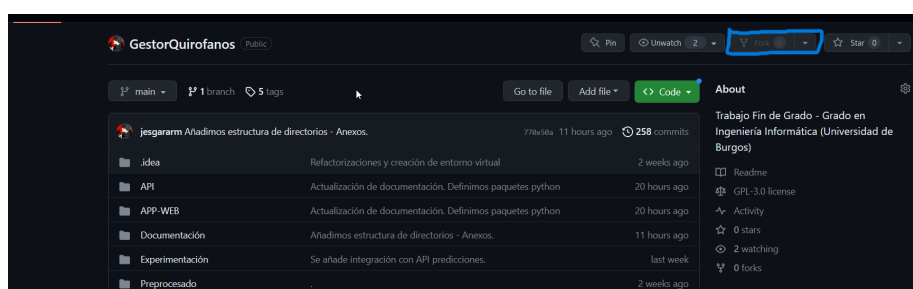


Figura D.1: Vista del repositorio y función de fork

Por otro lado, es recomendable disponer de *git* instalado en el computador principal, para poder acceder al código fuente y poder realizar modificaciones con cambios persistentes.

Recomendamos para tal índole la instalación de la suite **Github Desktop**, pues permite desde un interfaz sencillo y comprensible realizar la clonación del repositorio en un equipo local y manejar las modificaciones y las *ramas* de trabajo sin necesidad de conocer los parámetros y funcionalidades *git* desde la línea de comandos.

Sistema Gestor de Bases de Datos (SGBD)

Es recomendable, para realizar pruebas y comprender el diseño de datos, contar con un sistema gestor de base de datos **compatible con MySQL e InnoDB**.

Recomendamos la instalación de **XAMPP**, que es un entorno de desarrollo PHP que incluye el SGBD phpmyadmin. Desde ese entorno hemos diseñado y realizado las pruebas locales del sistema, previa a su exportación del esquema en fichero sql.

Docker

Dado que usaremos contenedores para encapsular la lógica del sistema y, posteriormente, desplegarlos para su funcionamiento desde cualquier computador, debemos tener el *daemon* Docker instalado en nuestro sistema.

Al igual que con *git*, recomendamos la instalación del GUI oficial de Docker, **Docker Desktop**.

Obtención del código fuente

Una vez instalado **Github Desktop** y realizado un *fork* del repositorio, podemos obtener su contenido desde el propio interfaz: **File > Clone Repository**

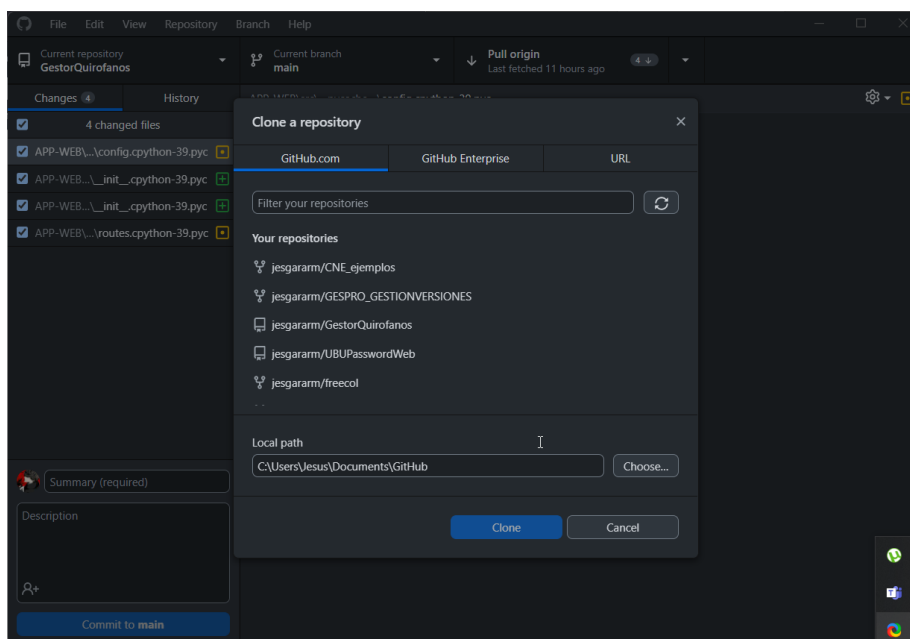


Figura D.2: Clonar Repositorio con Github Desktop

Por otro lado, desde *Git Bash*, abriendo la terminal en el directorio local donde deseamos obtener la copia del código fuente, bastará con ejecutar el siguiente comando:

```
git clone https://github.com/jesgararm/GestorQuirofanos.git
```

Importación del proyecto e instalación de dependencias

Dado que Python es un lenguaje *interpretado*, no es necesaria su compilación, por lo que los IDEs no requieren de proyectos con dependencias funcionales preestablecidas (*al contrario que otros desarrollados en lenguajes compilados, como Java*).

Para empezar a trabajar en el repositorio clonado, basta con añadir al entorno de trabajo de nuestro IDE la localización local del repositorio. Por ejemplo, desde VS Code: **File > Add Folder to Workspace**

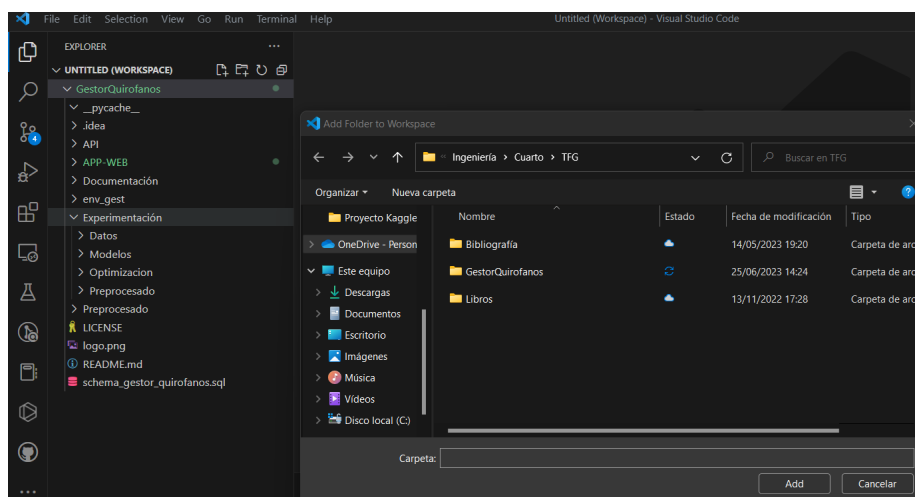


Figura D.3: Agregar repositorio local al entorno de trabajo de Visual Studio Code

Entornos virtuales e instalación de librerías y dependencias

Python y la librería *virtualenv* nos permiten trabajar con **entornos virtuales**. Esto nos permite instalar y trabajar con entornos de python **aislados**, cada uno con sus librerías y dependencias independientes del paquete que tengamos instalados en la raíz de nuestro sistema.

Cabe destacar que, una vez instalado y activado un entorno, todas las dependencias agregadas al mismo serán *exclusivas* de éste y no serán duplicadas en el entorno principal. Del mismo modo, las librerías principales no serán accesibles desde el nuevo entorno, debiendo configurarlo y definirlo desde cero tras su activación.

Recomendamos la creación de **dos entornos virtuales**, uno para cada subsistema. Para ello, nos situaremos en el directorio `./API` o `./APP-WEB` y ejecutaremos el comando:

```
virtualenv [nombreentorno]
```

Una vez realizado, pasaremos a su activación, ejecutando el script correspondiente (desde Windows):

```
./[nombreentorno]/Scripts/activate
```

Desde allí, podemos instalar las librerías y dependencias. Se incluye un fichero `requirements.txt` con las librerías en la raíz de ambos directorios (API y APP-WEB), por lo que su instalación es sencilla con el comando pip:

```
pip install -r requirements.txt
```

Para salir del entorno y pasar al principal (PATH de nuestro S.O), ejecutaremos `deactivate`.

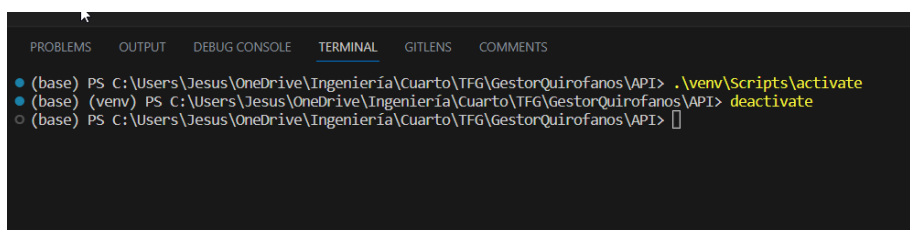
A screenshot of a terminal window with a dark background. At the top, there are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is active), 'GIT LENS', and 'COMMENTS'. The terminal shows three lines of command history: 1. A blue prompt character followed by '(base) PS C:\Users\Jesus\OneDrive\Ingeniería\Cuarto\TFG\GestorQuirofanos\API>' and the command './venv\Scripts\activate' in yellow. 2. A blue prompt character followed by '(base) (venv) PS C:\Users\Jesus\OneDrive\Ingeniería\Cuarto\TFG\GestorQuirofanos\API>' and the command 'deactivate' in yellow. 3. A grey prompt character followed by '(base) PS C:\Users\Jesus\OneDrive\Ingeniería\Cuarto\TFG\GestorQuirofanos\API>' and an empty command line.

Figura D.4: Activación y desactivación de entorno virtual

D.4. Compilación, instalación y ejecución del proyecto

El proyecto está elaborado principalmente en Python, lenguaje *interpretado*, por lo que tan sólo se requiere un intérprete de Python con las librerías necesarias para su ejecución, no siendo necesaria su **compilación**.

Por otro lado, se ofrece un *despliegue* en Amazon Web Services desde al que acceder al mismo. El enlace a la aplicación web se encuentra actualizado en la sección **README** del repositorio:

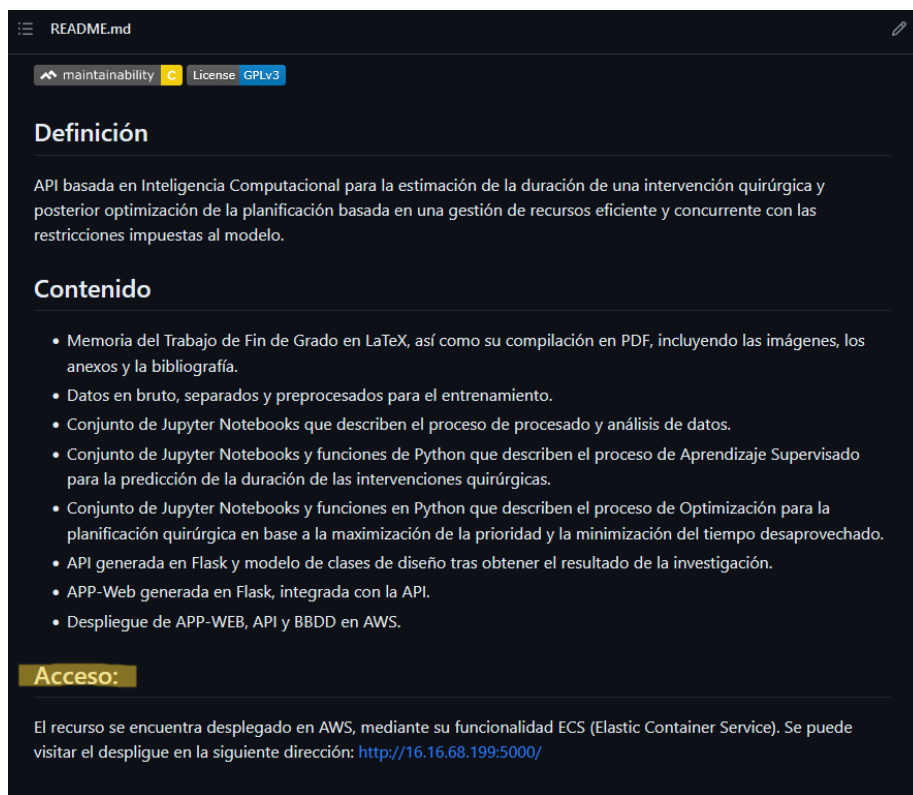


Figura D.5: Localización de enlace al despliegue de la aplicación en GitHub

Por tanto, en este apartado se describen los pasos a seguir para *desplegar* la aplicación en AWS.

Primer paso: Configuración de base de datos

Creación e inicio de Base de Datos en AWS

Accederemos, tras iniciar sesión, a **Amazon RDS**, que es el servicio de base de datos relacionales. Una vez allí, crearemos una nueva base de datos (*existe un enlace directo a la funcionalidad desde el panel principal*), seleccionando MySQL y configurando los datos de usuario maestro según el contenido del fichero `./APP-WEB/src/config.py`, que en nuestro caso se corresponden con *root* como nombre de usuario y *gestorquiroyanos* como contraseña.

Configuración

Identificador del clúster de base de datos [Información](#)
 Ingresa un nombre para el clúster de base de datos. El nombre debe ser único entre todos los clústeres de base de datos de la cuenta de AWS de la región de AWS actual.

database-1

El identificador del clúster de base de datos no distingue entre mayúsculas y minúsculas, pero se almacena todo en minúsculas (por ejemplo, "miclústerdebasedatos"). Restricciones: de 1 a 60 caracteres alfanuméricos o guiones. El primer carácter debe ser una letra. No puede contener dos guiones consecutivos. No puede terminar con un guion.

▼ **Configuración de credenciales**

Nombre de usuario maestro [Información](#)
 Escriba un ID de inicio de sesión para el usuario maestro del clúster de la base de datos.

root

De 1 a 16 caracteres alfanuméricos. El primer carácter debe ser una letra.

☐ **Administrar credenciales maestras en AWS Secrets Manager**
 Administre las credenciales de usuario maestras en Secrets Manager. RDS puede generar una contraseña por usted y administrarla durante todo su ciclo de vida.

❗ Si administra las credenciales de usuario maestro en Secrets Manager, algunas características de RDS no son compatibles. [Más información](#)

☐ **Generación automática de contraseña**
 Amazon RDS puede generar una contraseña en su nombre, o bien puede especificar su propia contraseña.

Contraseña maestra [Información](#)

●●●●●●●●●●

Restricciones: debe tener al menos 8 caracteres ASCII imprimibles. No puede contener ninguno de los siguientes caracteres: / (barra diagonal), " (comillas simples), " (dobles comillas) y @ (signo de arroba).

[Confirmar la contraseña maestra](#) [Información](#)

Figura D.6: Configuración de usuario maestro en base de datos RDS

El resto de los parámetros pueden mantenerse inalterables hasta finalizar el formulario.

Obtención de parámetros de conexión e importación del esquema

Tras la creación, desde el panel *Bases de Datos* de la barra de herramientas en RDS, obtendremos un listado de todas las BBDD relacionales creadas.

Al seleccionar la recién creada, acudiremos a un panel que muestra las características de la misma. Desde allí, en el apartado *Conectividad y Seguridad* > *Punto de enlace y puerto*, obtendremos los parámetros de conexión para la gestión de la base de datos desde un SGBD.

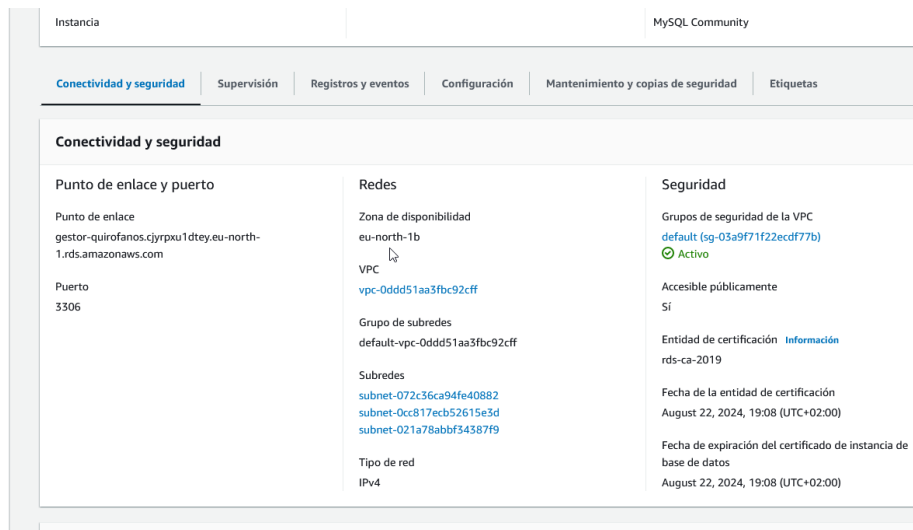


Figura D.7: Panel de configuración y características de BD en Amazon RDS

Tras su obtención, nos conectaremos a la misma, creando una nueva conexión y añadiendo los valores de enlace y puerto, así como el usuario y contraseña.

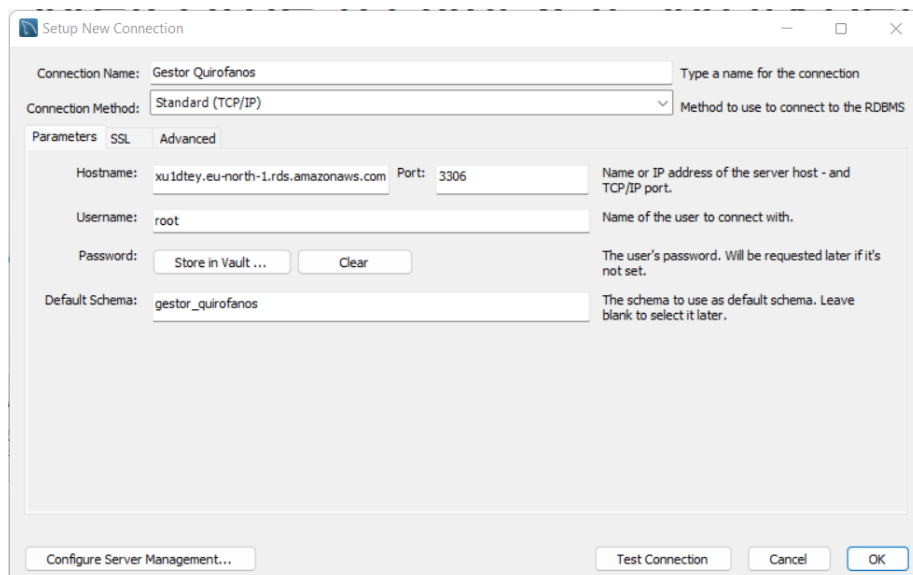


Figura D.8: Ejemplo de conexión a BD RDS usando MySQL Workbench

Tras establecerse la conexión, deberemos crear un *nuevo schema*, denominado *gestor_quirófanos* y, tras seleccionarlo, ejecutar el script localizado en: `./schema_gestor_quirofanos.sql`.

Tras la ejecución del script, se crearán las tablas y las dependencias funcionales entre ellas, así como habrá algunos usuarios, predicciones y planificaciones de prueba.

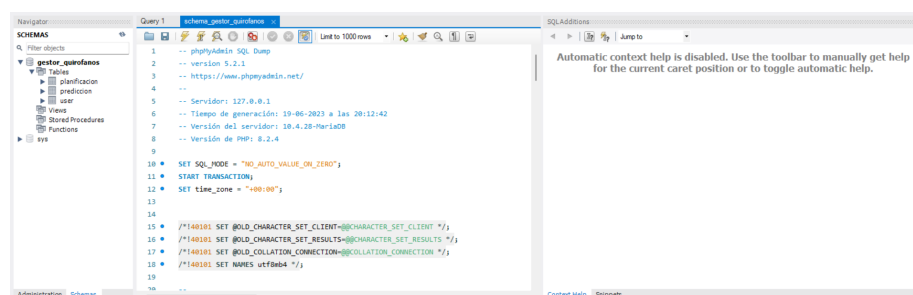


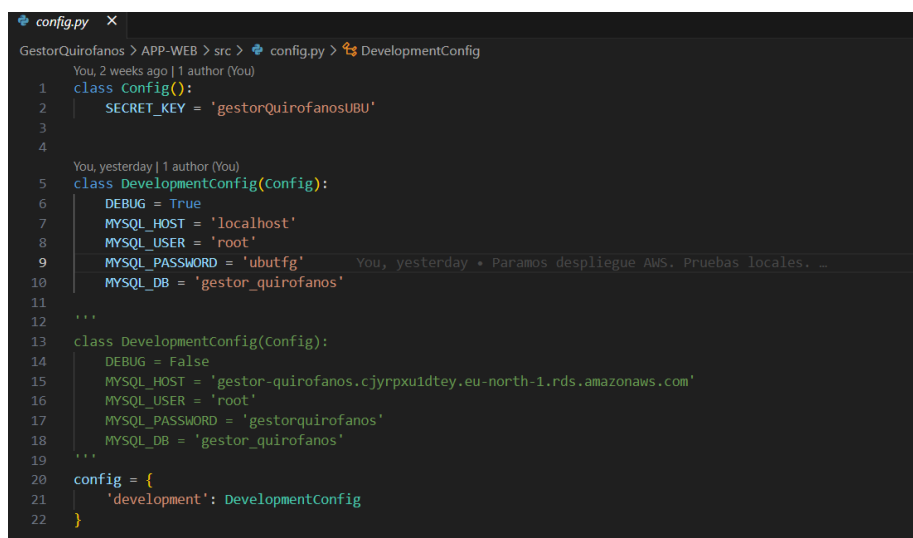
Figura D.9: Vista de script y resultado de su ejecución en conexión a BD RDS

Actualización de parámetros de conexión a BD

Es en el fichero `./APP-WEB/src/config.py` donde se especifican los parámetros de conexión de nuestro interfaz con la base de datos.

Allí encontramos dos definiciones en la clase ***DevelopmentConfig***, una para la conexión con la BD local y otra para la remota.

Para configurarlo, bastará con dejar comentada la porción de código que no nos interesa y actualizar los datos de conexión en la variable `MYSQL_HOST`.



```
1 class Config():
2     SECRET_KEY = 'gestorQuirofanosUBU'
3
4
5 class DevelopmentConfig(Config):
6     DEBUG = True
7     MYSQL_HOST = 'localhost'
8     MYSQL_USER = 'root'
9     MYSQL_PASSWORD = 'ubutfg'
10    MYSQL_DB = 'gestor_quirofanos'
11
12    ...
13    class DevelopmentConfig(Config):
14        DEBUG = False
15        MYSQL_HOST = 'gestor-quiroyfanos.cjyrpxuidtey.eu-north-1.rds.amazonaws.com'
16        MYSQL_USER = 'root'
17        MYSQL_PASSWORD = 'gestorquiroyfanos'
18        MYSQL_DB = 'gestor_quirofanos'
19    ...
20    config = {
21        'development': DevelopmentConfig
22    }
```

Figura D.10: Código fuente con las rutas de conexión con el esquema *gestor_quirofanos* en la base de datos

Segundo paso: Creación de contenedor API

Contando con la existencia del *daemon* Docker ejecutándose (basta con iniciar el interfaz Docker Desktop en segundo plano) y los ficheros *Dockerfile*, podremos crear con facilidad desde línea de comandos estos contenedores.

Debemos crear dos contenedores, uno en cada subsistema, a partir del contenido del directorio API y APP-WEB.

Dado que debemos contar con la **información de conexión de la API** de cara a ejecutar el interfaz web, debemos proceder con estos pasos de forma secuencial para cada uno de ambos subsistemas.

Por ello, nos situamos en `./API` y ejecutamos `docker build -t [nombreakapi]`

Tras la ejecución *automática* de los pasos detallados en *Dockerfile*, se creará una *imagen* del contenedor, que podremos ejecutar o desplegar en un servidor.

```
(base) PS C:\Users\Jesus\OneDrive\Ingeniería\cuarto\TFG\GestorQuirofanos\API> docker build -t gestorquirofanosapi .  
[+] Building 51.3s (10/10) FINISHED  
• => [internal] load .dockerignore  
=> => transferring context: 2B  
=> [internal] load build definition from Dockerfile  
=> => transferring dockerfile: 171B  
=> [internal] load metadata for docker.io/library/python:3.9.5-slim-buster  
=> [auth] library/python:pull token for registry-1.docker.io  
=> [1/4] FROM docker.io/library/python:3.9.5-slim-buster@sha256:9828573e6a0b02b6d0ff0bae0716b027aa21cf8e59ac18a7672  
=> [internal] load build context  
=> => transferring context: 1.36MB  
=> CACHED [2/4] WORKDIR /app  
=> [3/4] COPY . /app  
=> [4/4] RUN pip install -r requirements.txt  
=> exporting to image  
=> => exporting layers  
=> => writing image sha256:0ddcf4d497f85b53dcecc1bc85031d31eabb4e4972ea58b5570f9f1ebab918d6  
=> => naming to docker.io/library/gestorquirofanosapi
```

Figura D.11: Creación de contenedor para la API desde línea de comandos

Tercer paso: Desplegar la API

Publicación en Amazon ECR

Para desplegar la API debemos, en primer lugar, almacenar el docker en un **repositorio** Amazon, haciendo uso de la herramienta **Elastic Container Registry**.

Debemos crear un nuevo repositorio (*enlace en la ventana principal*), seleccionar la visibilidad pública y añadir un nombre que **coincida** con el del contenedor, no siendo necesarias más modificaciones.

Figura D.12: Formulario de creación de un repositorio Amazon ECR

Para el siguiente paso, es conveniente tener instalado el gestor en línea de comandos **AWS CLI**.

Una vez instalada esta utilidad, seleccionaremos nuestro repositorio y marcaremos la opción **Mostrar claves de envío**, abriéndose una ventana con los pasos a seguir para publicar la imagen del contenedor en nuestro repositorio:

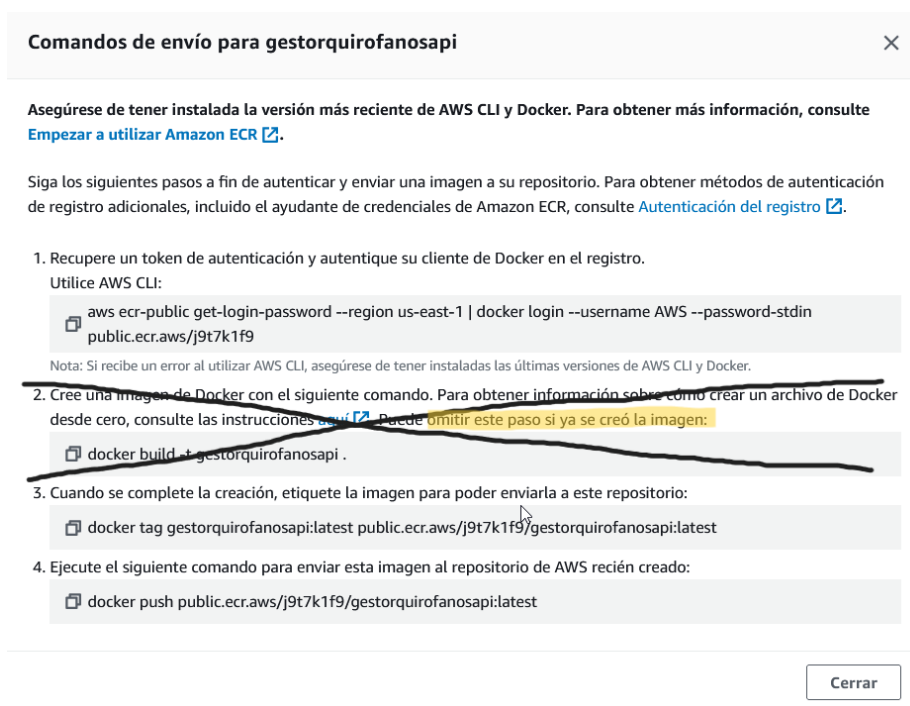


Figura D.13: Ejemplo de claves de envío Amazon ECR

Si seguimos los pasos detallados en la ventana, conseguiremos publicar la imagen del contenedor en nuestro repositorio:

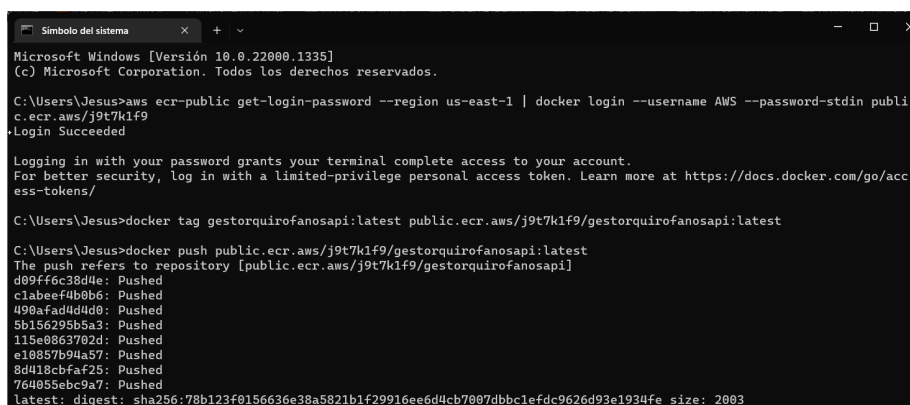


Figura D.14: Ejecución satisfactoria de la subida de un contenedor a repositorio Amazon ECR

Despliegue en Amazon ECS

Una vez almacenado, podremos ejecutar el contenedor en un servidor proporcionado por AWS, mediante la función *Elastic Container Services*.

Deberemos crear (*si no disponemos aún*) un clúster, encargado de ejecutar contenedores a modo de **servicios y tareas**:

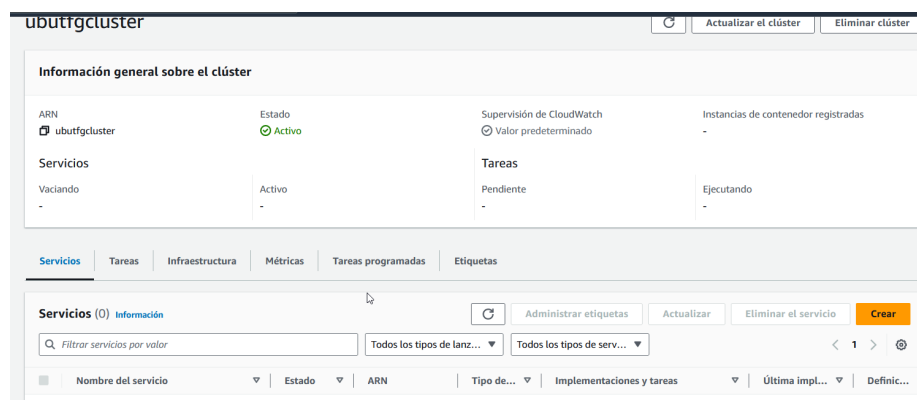


Figura D.15: Visualización de panel de de configuración de nuestro cluster **ubutfgcluster**

Crearemos un nuevo *servicio* mediante la opción habilitada para ello en nuestro panel. Podremos dejar todas las opciones marcadas **por defecto**, debiendo añadir una *definición de tarea* con la URI de la imagen en ECR y el puerto 4000, guardándola con el nombre de *API-TFG*:

Contenedor: 1 [Información](#) Contenedor esencial Eliminar

Detalles del contenedor
Especifique un nombre, una imagen de contenedor y si el contenedor debe marcarse como esencial. Cada definición de tarea debe tener al menos un contenedor esencial.

Nombre: URI de imagen: Contenedor esencial:

Registro privado [Información](#)
Almacene las credenciales en Secrets Manager y, a continuación, utilice las credenciales para hacer referencia a las imágenes en los registros privados.

☒ Autenticación de registro privado

Mapeos de puertos [Información](#)
Agregue mapeos de puertos para permitir que el contenedor obtenga acceso a los puertos del host para enviar o recibir tráfico. Cualquier cambio en la configuración de mapeos de puertos afecta a la configuración de conexión del servicio asociada.

Puerto del contenedor: Protocolo: Nombre del puerto: Protocolo de la aplicación: Eliminar

Figura D.16: Definición de familia de tareas API-TFG, añadiendo enlace a la imagen del contenedor en ECR

Tras la definición de la tarea, volveremos al formulario anterior, donde registraremos el servicio *gestorquirofanosapi*, ligándolo a la familia de tareas API-TFG:

Definición de tarea
Seleccione una definición de tarea existente. Para crear una nueva definición de tarea, vaya a [Definiciones de tareas](#).

☐ Especificar la revisión manualmente
Ingrese manualmente la revisión en lugar de elegir entre las 100 revisiones más recientes para la familia de definición de tareas seleccionada.

Familia: Revisión:

Nombre del servicio
Asigne un nombre único a este servicio.

Tipo de servicio [Información](#)
Especifique el tipo de servicio que seguirá el programador de servicio.

☒ Réplica
Coloque y mantenga un número deseado de tareas en su clúster.

☐ Daemon
Coloque y mantenga una copia de la tarea en cada instancia del contenedor.

Tareas deseadas
Especifique el número de tareas que se van a lanzar.

Figura D.17: Definición de servicio *gestorquirofanosapi* en el cluster

El servicio arrancará unos instantes después, permitiendo obtener la dirección de acceso a la API en ejecución y pasar al encapsulamiento y despliegue del interfaz.

Cuarto Paso: Configuración de rutas de acceso a la API

Tras arrancar el servicio, deberemos obtener la **dirección pública** de la tarea en ejecución. Para ello: Selección de servicio > Tareas > Tarea > Configuración > IP pública

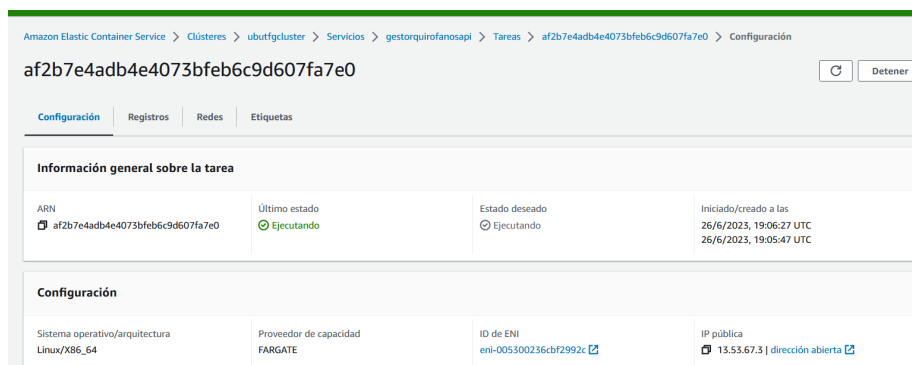


Figura D.18: Localización de URL del despliegue de la API

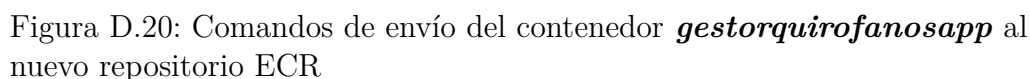
Una vez allí, en los métodos localizados en `./APP-WEB/src/public/routes.py` donde se especifique la conexión con la API (*upload* y *uploadSched*), comentaremos la línea de conexión con el servidor local y añadiremos:

- `http://[url]:4000/predict`: Para las funciones de predicción.
- `http://[url]:4000/schedule`: Para las funciones de planificación.



De forma similar al paso anterior, nos situamos en línea de comandos en el directorio `./APP-WEB` y ejecutamos:

Tras su ejecución, crearemos un nuevo repositorio, de nombre *gestorquirofanosapp*, siguiendo los comandos del **tercer paso** (aunque sustituyendo el nombre *gestorquirofanosapi* por *gestorquirofanosapp*):



Por último, deberemos definir una nueva tarea (que hemos nombrado como *APP-TFG*) y seleccionarla como base de un nuevo servicio en el clúster, que llamaremos *gestorquirofanosapp*, estableciendo el puerto 5000.

Una vez iniciado el servicio, accederemos a la tarea y a su dirección, siendo ésta la base para la construcción de la **URL del despliegue de nuestra aplicación web**:

$$URL = http://[url-tarea]:5000$$



Figura D.21: Obtención de la dirección pública del despliegue de la APP

Sexto Paso: Comprobación del despliegue

Accedemos a la URL y comprobamos su funcionamiento:

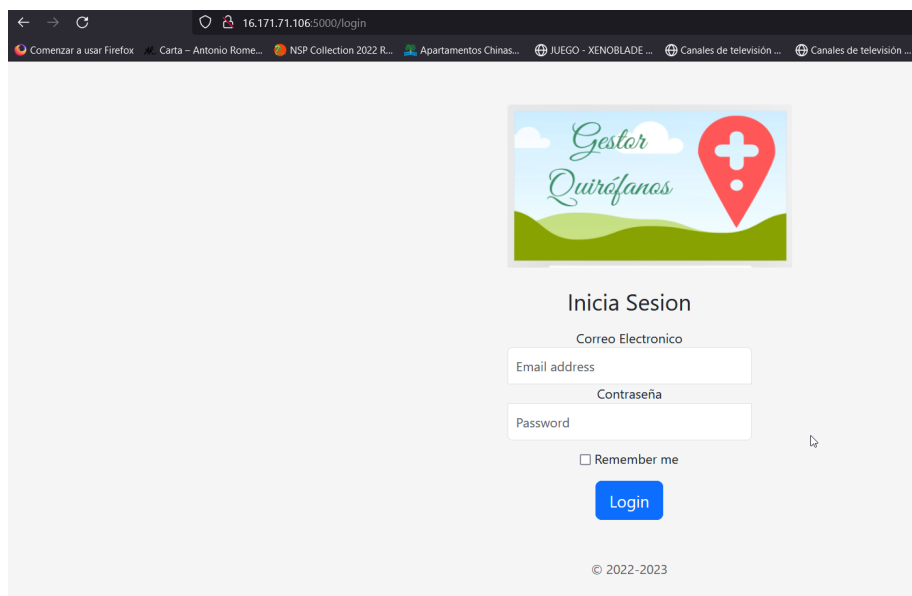


Figura D.22: Comprobación del correcto despliegue del sistema en AWS

D.5. Pruebas del sistema

Podemos distinguir varias etapas en nuestro proceso de pruebas del sistema:

- Pruebas de los modelos de aprendizaje supervisado.
- Pruebas de los algoritmos de optimización.
- Pruebas de sistema en la API.
- Pruebas de sistema e integración en aplicación web.

Pruebas de los modelos de aprendizaje supervisado y de algoritmos de optimización

En este tipo de pruebas se analizó el **rendimiento** de los distintos modelos, incluyendo representaciones gráficas de la adaptación y ajuste de los mismos respecto a los valores *reales*.

Aprovechando la **versatilidad** de python como lenguaje *interpretado* y la existencia de los cuadernos Jupyter, pudimos documentar todo el proceso de análisis y pruebas dentro de los mismos.

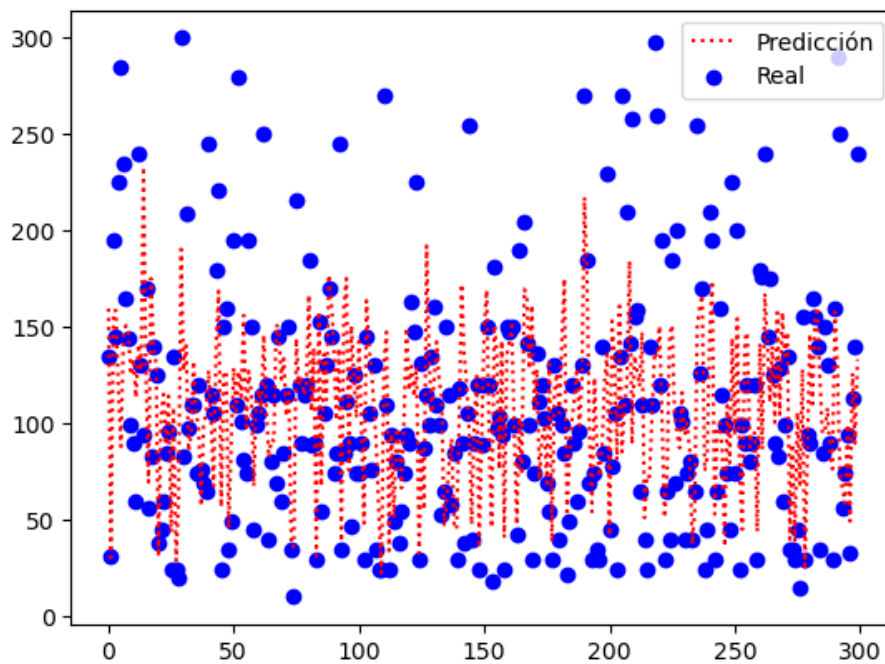


Figura D.23: Ejemplo de gráfico de rendimiento (*KNN*) disponible en los Jupyter Notebooks

Todos los cuadernos, con las celdas de código ejecutadas, pueden encontrarse en: `./Experimentación/Modelos`.

Por otro lado y, siguiendo un enfoque similar, se documentaron las pruebas y el análisis del rendimiento de los algoritmos de optimización (*genético*, *heurísticas* y *PSO*) en Jupyter. Accesibles en: `./Experimentación/Optimizacion`

Pruebas en la API

Una vez integrados los modelos en una API, procedimos a su *testeo* mediante un enfoque de **caja negra**.

Para ello, nos valimos del software **Postman**, que es una API capaz de conectarse y explotar otras APIs (*realizar peticiones GET ,POST ,UPDATE...*) y es ampliamente usada en la comunidad de desarrolladores para la ejecución de pruebas de sistema sobre otras interfaces de programación de aplicaciones.

Se siguieron las recomendaciones habituales para el diseño de los casos de prueba:

- Un caso de prueba por cada requerimiento.
- Empleo de valores límites en los parámetros del formulario.
- Prueba de valores válidos e inválidos.

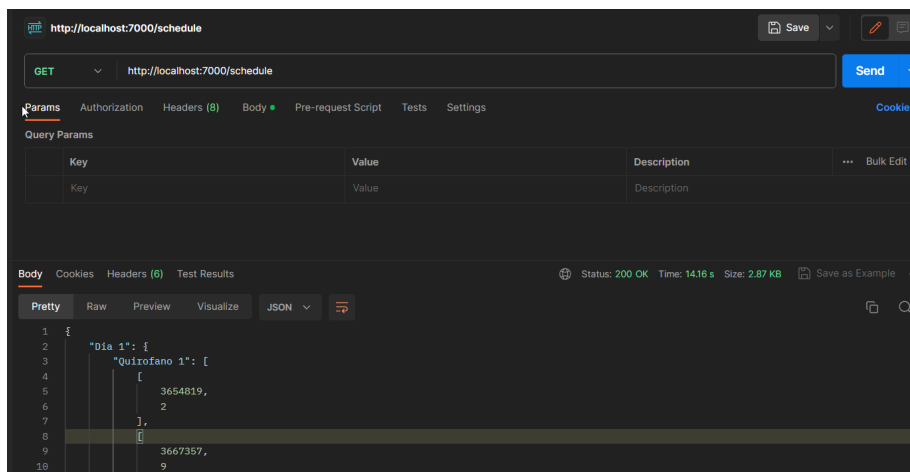


Figura D.24: Ejemplo de *request* y *response* durante las pruebas de sistema en la API

Una vez publicada y en funcionamiento dentro del servicio AWS ECS, se actualizaron las direcciones y puerto, repitiéndose la ejecución de los casos de prueba.

Pruebas en aplicación web

De nuevo, mediante un enfoque de **caja negra**, se ampliaron los casos de prueba añadiendo los requisitos funcionales de *gestión de usuario* y no funcionales de la *interfaz*.

Las pruebas se realizaron **manualmente**, dado que el número de casos de prueba y funcionalidades a testear no requirieron de automatización.

Como hecho diferenciador, se añadieron **pruebas de usuario** sobre la interfaz, solicitando y otorgando acceso a la aplicación a tres perfiles de usuario *diferentes*: cirujano, enfermero y administrativo. Sin supervisión directa, exploraron el interfaz y entregaron, en una reunión posterior, su punto de vista y recomendaciones de cara a la adecuación del software a sus requisitos.

Apéndice *E*

Documentación de usuario

E.1. Introducción

En este apartado se detallan los requisitos que deben cumplir los usuarios de cara al uso de esta aplicación, así como un recorrido que exprese las diferentes funcionalidades del interfaz y sirva como guía de uso para los mismos.

E.2. Requisitos de usuarios

En esta sección exploraremos los requerimientos que deben cumplir los clientes del producto para hacer uso con éxito de las funciones ofertadas.

Requisitos de acceso

El sistema **no incluye** una opción de registro, de modo que cualquier usuario potencial debe ser *autorizado y registrado* en el sistema por parte de un usuario con **privilegios de administrador**, pudiendo ser sancionado con la *eliminación* del registro si incumpliese las normas de uso definidas por la organización sanitaria cliente del producto.

Requisitos del sistema

Dado que la ejecución de la lógica interna se realiza en **servidores externos** provistos por AWS, no son necesarios como *requisitos mínimos* la posesión de *cualquier dispositivo* (incluye renderizado móvil) con conexión

a internet y navegador compatible con lenguaje de marcado HTML5 y protocolo HTTP v1.1.

E.3. Instalación

No es necesaria la instalación de la aplicación, dado que se encuentra desplegada en AWS ECS. Se incluye un enlace en la [página principal del repositorio](#) con acceso al despliegue desde cualquier navegador.

E.4. Manual del usuario

Incluimos en este apartado un recorrido por la interfaz, así como las diferentes funcionalidades.

Usuario sin privilegios de administrador

Inicio de sesión

Introducir el enlace en la barra de navegación y acceder a la página del login. Allí, tras introducir el email y la contraseña facilitadas por el administrador, podremos acceder al panel de usuario.

En caso de error, se mostrarán mensajes aclarativos.

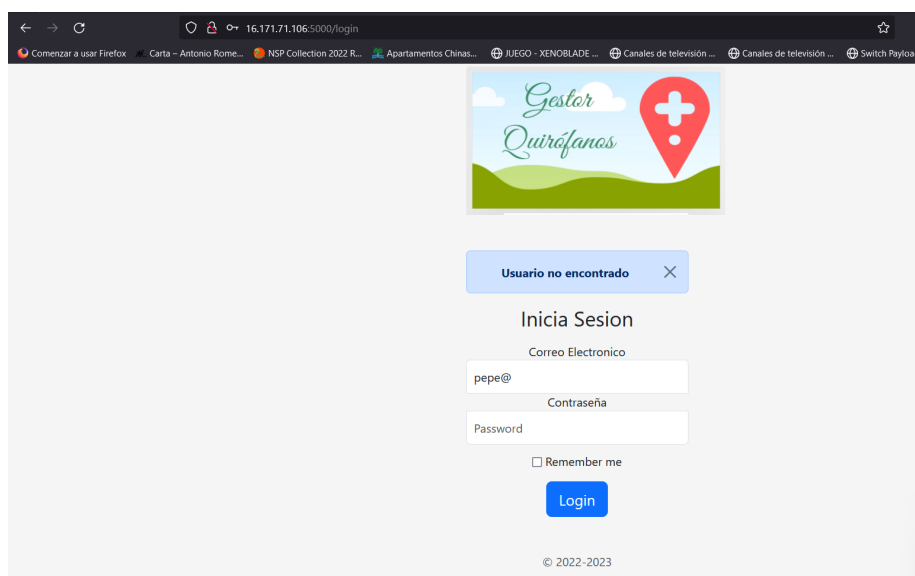


Figura E.1: Ejemplo de mensaje de error al no localizar al usuario

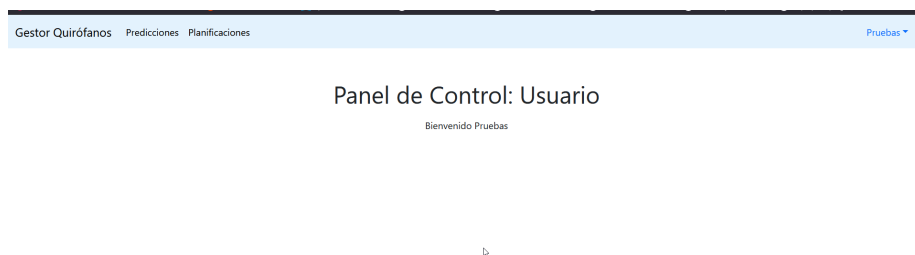


Figura E.2: Login satisfactorio y redirección al panel de usuario

Perfil de usuario

En la esquina superior derecha del panel, sobre la barra de navegación, se encuentra el **nombre de usuario**. Al seleccionarlo, se abrirá un desplegable desde el que podremos acceder a la opción *Perfil*.



Figura E.3: Localización de perfil de usuario

Se actualizará el contenido del panel, mostrando una *tarjeta* con información actual del usuario, pudiendo editar (nombre y/o email) la información referente desde el botón *Editar*.

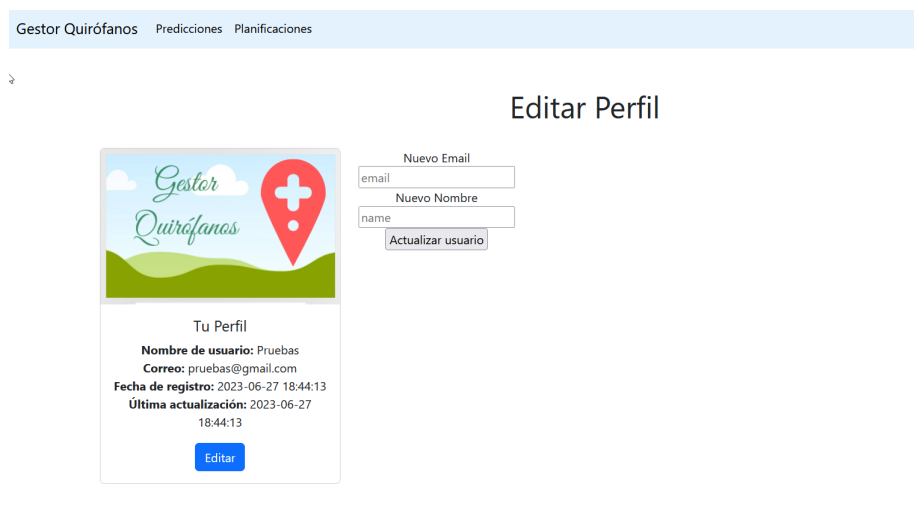


Figura E.4: Panel de usuario y función de edición

Para regresar desde **cualquier punto** del aplicativo hacia la página principal, bastará con seleccionar el nombre "Gestor Quirófanos."^{en} la esquina superior izquierda, en el interior de la barra de navegación.

Predicciones

Accedemos desde la barra de navegación a la opción *Predicciones* para visualizar el panel de predicciones.

Allí se cargará un listado de las predicciones solicitadas por el usuario, de forma que podamos visualizarlas, eliminarlas o crearlas.



Figura E.5: Vista del Panel de Predicciones

A la hora de **solicitar** una predicción, debemos subir un fichero en *formato CSV*, incluyendo las restricciones descritas en la tabla [E.1](#)

Nombre de columna	Tipo de variable	Valores Posibles
NHC	Entero	Cualquiera
INTERVENCIÓN	Real	Catálogo CIE-9
TIPO	Literal	Mayor
		Menor
TURNO	Literal	Mañana
		Tarde
CARÁCTER ECONÓMICO	Literal	Actividad Ordinaria
		Continuidad Asistencial
PONDERACIÓN	Entero	Cualquiera
ESPECIALIDAD	Literal	PLASTICA
		TORACICA
		MAXILOFACIAL
		NEUROCIRUGIA
		MAXILOFACIAL
		TRAUMATOLOGIA
		OTORRINOLARINGOLOGIA

Tabla E.1: Restricciones en variables de datos

En caso de error en el formato del fichero o en las variables, se mostrará un **mensaje de error**. Si se ha desarrollado sin incidencias, la redirección mostrará el listado actualizado junto a un **mensaje de éxito**.

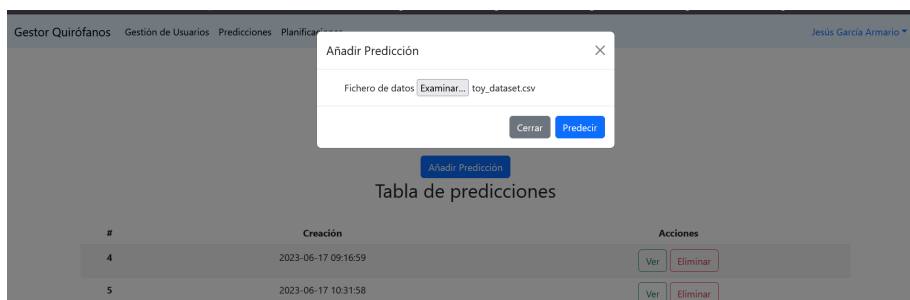


Figura E.6: Ventana flotante para solicitar una nueva predicción a la API

Planificaciones

Al igual que en el apartado anterior, accederemos al **panel de planificaciones** seleccionando la opción *Planificaciones* de la barra de navegación.



Figura E.7: Acceso al panel de planificaciones

Desde allí podremos, de forma similar al paso anterior, crear, ver o eliminar todas las planificaciones realizadas por el usuario.

Seleccionando el botón **Añadir Planificación**, accederemos a una ventana flotante donde seleccionar el archivo csv, con *las mismas restricciones* que el apartado anterior, así como añadir las especificaciones de: **número de quirófanos (1-10)**, **tiempo entre pacientes (1-60 minutos)** y **días a programar (1-4 semanas)**.

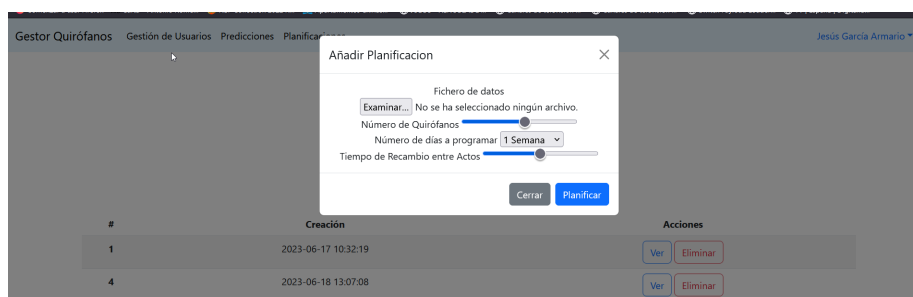


Figura E.8: Ventana flotante para añadir una nueva planificación

Por último, al seleccionar la opción **Ver**, seremos redirigidos a la planificación propuesta, mostrada como una *agenda* donde seleccionaremos el día y se mostrarán, a modo desplegable, los quirófanos numerados junto a una línea temporal:



Figura E.9: Se muestra ejemplo de visualización de la programación

Además, si seleccionamos alguno de los pacientes en la línea temporal, accederemos a una visualización detallada de su identificador y la duración propuesta por el modelo de predicción.

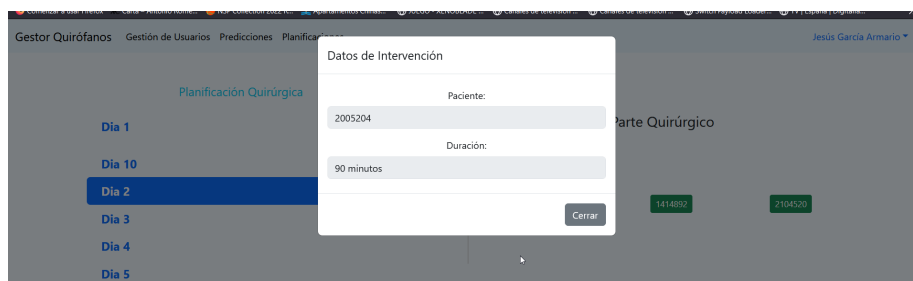


Figura E.10: Detalle de la planificación

Usuario con privilegios de administrador

A las funciones anteriormente descritas, se añade la **gestión de usuarios**.

Gestión de usuarios

Podremos acceder, tras identificarse como usuario administrador, al *panel de administrador*:

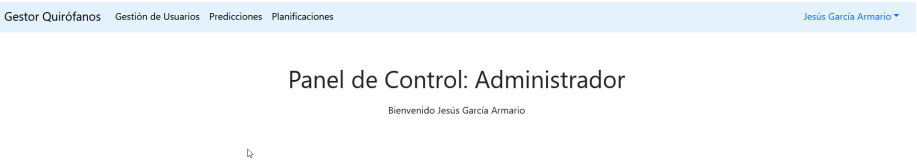


Figura E.11: Vista del panel de administrador, incluyendo función de gestión de usuarios

Desde allí, en la barra de navegación, encontraremos una función inexistente para el resto de usuarios: **Gestión de Usuarios**.



Figura E.12: Panel de gestión de usuarios

Para añadir un usuario, accederemos al formulario de creación seleccionando la opción correspondiente (*Añadir Usuario*), donde introduciremos **nombre, email, contraseña y marcaremos si es administrador o no**.

Gestión de Usuarios

Añadir Usuario

Email Contraseña Nombre admin ☐

Figura E.13: Formulario de creación de usuario

Del mismo modo y, de vuelta al panel anterior, modificaremos los datos de otros usuarios (**nombre, email y privilegios**), *sobreescribiendo* los previos o dejándolos inalterados:

#	Nombre	Email	Fecha	Admin	Acciones
83104	Jesús García Armario	jesusgarciaarmario@gmail.com	2023-06-19 18:21:38	Si	<input type="button" value="Editar"/> <input type="button" value="Eliminar"/>
93908	UBU	ubu@h...	2023-06-19 18:21:38	No	<input type="button" value="Editar"/> <input type="button" value="Eliminar"/>
13217	Pepe	pepe@...	2023-06-19 18:21:38	No	<input type="button" value="Editar"/> <input type="button" value="Eliminar"/>
	Tutores TFG	tutoresfg@ubues	2023-06-19 18:21:38	Si	<input type="button" value="Editar"/> <input type="button" value="Eliminar"/>

Figura E.14: Ventana de modificación de usuario

Bibliografía

- [1] IEEE Recommended Practice for Software Requirements Specifications. *IEEE Std 830-1998*, pages 1–40, 1998.
- [2] Carol Britton and Jill Doake. Identifying functionality: CRC cards and interaction diagrams. *A Student Guide to Object-Oriented Development*, pages 147–180, 2005.
- [3] Lawrence Chung and Julio Cesar Sampaio do Prado Leite. On Non-Functional Requirements in Software Engineering. pages 363–379. 2009.
- [4] Sofía Garrido Elustondo, Luisa Cabello Ballesteros, Inés Galende Domínguez, Rosario Riesgo Fuertes, Ricardo Rodríguez Barrientos, and Elena Polentinos Castro. Investigación y protección de datos personales en atención primaria. *Atención Primaria*, 44(3):172–177, 3 2012.
- [5] Dragos Paul Pop and Adam Altar. Designing an MVC Model for Rapid Web Application Development. *Procedia Engineering*, 69:1172–1179, 1 2014.
- [6] Muhammad Ehsan Rana and Omar S. Saleh. High assurance software architecture and design. *System Assurances: Modeling and Management*, pages 271–285, 1 2022.
- [7] Javier Sáez Hurtado. Cómo funciona la Metodología Scrum: qué es y cómo utilizarla, 12 2021.