UNIVERSIDAD DE SANTIAGO DE COMPOSTELA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA

Sistema para la reconstrucción de tomografías de impedancia eléctrica en tiempo real, basado en técnicas de Machine Learning

Autor:
Martín Aller Domínguez

Directores:
José Manuel Cotos Yáñez
David Mera Pérez

Grado en Ingeniería Informática

Noviembre 2020

Trabajo de Fin de Grado presentado en la Escuela Técnica Superior de Ingeniería de la Universidad de Santiago de Compostela para la obtención del Grado en Ingeniería Informática

D. José Manuel Cotos Yáñez, Profesor del Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela, y **D. David Mera Pérez**, Investigador posdoctoral del Centro de Investigación en Tecnologías Inteligentes de la Universidad de Santiago de Compostela,

INFORMAN:

Que la presente memoria, titulada *Sistema para la reconstrucción de tomografías de impedancia eléctrica en tiempo real, basado en técnicas de Machine Learning*, presentada por **D. Martín Aller Domínguez** para superar los créditos correspondientes al Trabajo de Fin de Grado de la titulación del Grado en Ingeniería Informática, se realizó bajo nuestra dirección en el Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela.

Y para que así conste a los efectos oportunos, expiden el presente informe en Santiago de Compostela, a 9 de noviembre de 2020:

El director,	El codirector,	El alumno,
COTOS YAÑEZ JOSE MANUEL - 32649200X Firmado digitalmente por COTOS YAÑEZ JOSE MANUEL - 32649200X Fecha: 2020.11.09 12:56:19 +01'00'	Firmado por MERA PÉREZ, DAVID (FIRMA) el día 09/11/2020 con un certificado emitido por AC DNIE 002	ALLER DOMINGUEZ MARTIN - 46095056C Firmado digitalmente por ALLER DOMINGUEZ MARTIN - 46095056C Fecha: 2020.11.09 12:12:12 +01'00'
José M. Cotos Yáñez	David Mera Pérez	Martín Aller Domínguez

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Tomografía de Impedancia Eléctrica

En ciertos entornos industriales y médicos es necesario obtener información sobre el interior de un cuerpo a través de métodos no invasivos. Para lograrlo, se pueden utilizar técnicas que analicen cualitativamente alguna propiedad interna del cuerpo para generar una imagen representativa de dicha propiedad, de modo que la imagen pueda ser examinada por un experto en el dominio del problema. Sin embargo, en algunos problemas es necesario realizar un análisis más profundo del interior del cuerpo desde un punto de vista cuantitativo, obteniendo un valor específico de la propiedad deseada para cada punto interno del cuerpo analizado.

Las técnicas para obtener información sobre el interior de un cuerpo en dos o tres dimensiones se conocen como tomografías. La más conocida es la **Tomografía Axial Computarizada** (TAC), mediante la cual se aplican rayos X sobre un cuerpo, con el objetivo de generar imágenes transversales del interior de éste. El uso de esta técnica conlleva la emisión de radiación ionizante, por lo que para el empleo del TAC se necesita personal cualificado y entornos controlados con niveles de seguridad elevados. Estos requerimientos suponen que la utilización del TAC sea inviable en ciertos ámbitos industriales. Por esta razón, es interesante analizar el empleo de técnicas alternativas que no presenten este problema, pudiendo ser utilizadas con mayor facilidad. Una de estas técnicas se conoce como **Tomografía de Impedancia Eléctrica** (EIT, por sus siglas en inglés), la cual permite medir la conductividad eléctrica de cada uno de los puntos dentro del cuerpo examinado, pudiendo inferir características y particularidades del interior del cuerpo a través de las medidas obtenidas. Para lograr esto, se colocan electrodos alrededor del cuerpo que se desea analizar, de forma que se suministran iterativamente corrientes eléctricas, siguiendo un cierto patrón de estimulación. Las corrientes eléctricas atraviesan el cuerpo y son captadas por algunos de los otros electrodos colocados sobre dicho cuerpo. El proceso de obtener las medidas de voltaje se conoce como *forward problem*, mientras que el proceso de estimar

la conductividad eléctrica de cada punto del interior del cuerpo a partir de los voltajes medidos se conoce como el *inverse problem*. Este último es un problema altamente no lineal con una solución no trivial.

Un ejemplo representativo en el que el empleo de EIT puede ser de gran utilidad es la **industria maderera**. En este tipo de industria, es necesario analizar en tiempo real la distribución de humedad en productos con base en madera. Las aproximaciones seguidas en la actualidad consisten en el análisis en laboratorio de muestras aleatorias, cuyos resultados afectan a un lote completo. De esta forma, si la muestra aleatoria se corresponde con un producto defectuoso (aquél que contiene distribuciones heterogéneas de humedad), se descarta todo el lote, a pesar de que la mayor parte de las piezas del lote se encuentren en buen estado. Mediante la utilización de EIT, podría llevarse a cabo un análisis individual de cada pieza, descartando piezas concretas defectuosas. De esta forma, se evitaría tener que desechar lotes completos, evitando el coste que ello supone.

En la actualidad, para resolver el problema inverso, se utilizan dos tipos de algoritmos: aquellos que asumen una cierta linealidad en la respuesta del cuerpo a los voltajes suministrados (sacrificando la precisión para ganar eficiencia) o, alternativamente, algoritmos iterativos, que son más precisos que los anteriores, pero requieren una gran cantidad de esfuerzo computacional y temporal. Ninguno de estos algoritmos es adecuado para la detección de piezas defectuosas en la industria maderera, puesto que dicha detección debe realizarse en tiempo real y con alta precisión.

1.2. Solución propuesta

Para resolver los problemas anteriores, en este trabajo se propone el empleo de técnicas de *Machine Learning*, para resolver el problema inverso en el menor tiempo posible, obteniendo resultados con alta precisión.

Los cuerpos con los que se trabajará serán bidimensionales y constituirán la sección transversal de un cilindro, representando el tronco de un árbol. A lo largo de este trabajo se utilizará el término **malla** para hacer referencia a cada uno de estos cuerpos. En la figura 1.1 se muestra la representación gráfica de una malla.

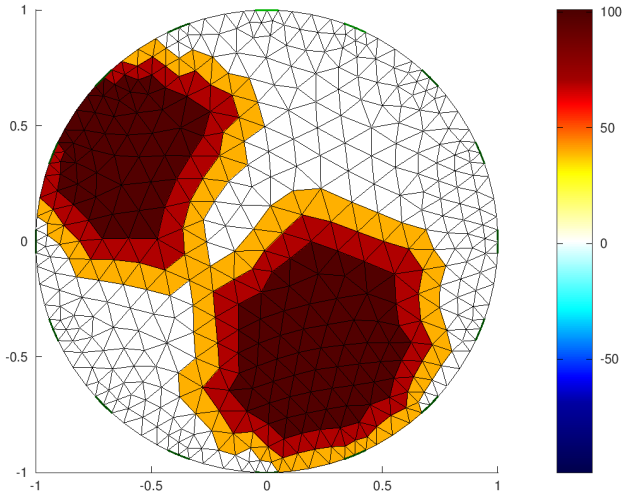

Figura 1.1: Ejemplo de malla con dos artefactos y 16 electrodos.

Las mallas que se utilizarán estarán divididas en 844 elementos triangulares, cada uno de los cuales tendrá asignado un valor de conductividad eléctrica, medida en S/m (siemens/metro). En la figura anterior se observan dos regiones con valores elevados de conductividad eléctrica. Este tipo de regiones recibirán el nombre de **artefactos**. Por otra parte, en el perímetro de la malla de la figura pueden observarse 16 pequeñas líneas de color verde. Cada una de estas líneas representa un **electrodo** colocado sobre la malla.

Con el propósito de entrenar los modelos de *Machine Learning*, se emplearán ***datasets***, los cuales estarán integrados por miles de mallas de hasta tres artefactos. Para cada malla, se dispondrá de 208 voltajes generados por los electrodos, así como de las 844 conductividades asociadas a cada uno de elementos triangulares de la malla. Empleando este tipo de *datasets*, se entrenarán tres clases de modelos de *Machine Learning*: Redes Neuronales (DNN), *Random Forest* (RF) y Máquinas de Soporte Vectorial (SVM).

1.3. Objetivos del trabajo

Partiendo de los puntos previos, con este trabajo se propone el desarrollo de un software que permita lo siguiente:

- **Reconstrucción de imágenes EIT**. Esta reconstrucción consistirá en la predicción de los valores de conductividad eléctrica de un cuerpo a partir de los voltajes captados por los electrodos (solución al problema inverso) y en la generación de la imagen con la representación de dichos valores de conductividad en cada uno de los puntos del interior del cuerpo.

- **Entrenamiento y almacenamiento de modelos de *Machine Learning***. Las predicciones de los valores de conductividad se realizarán mediante el empleo de modelos de *Machine Learning*, los cuales podrán ser de los siguientes tipos: redes neuronales (DNN), *Random Forest* (RF) y máquinas de soporte vectorial (SVM). El software permitirá entrenar modelos de estos tipos, configurando determinados parámetros para cada modelo.

- **Generación y almacenamiento de *datasets***. En este software, cada *dataset* consistirá en un conjunto de mallas. A su vez, cada malla estará compuesta por un conjunto de voltajes (los voltajes captados por los electrodos), así como por un conjunto de conductividades eléctricas, asociadas a los diferentes puntos de la malla. La representación gráfica de los valores de conductividad eléctrica en los diferentes puntos de una malla permitirá visualizar la composición de esa malla y, por tanto, determinar cuántos artefactos contiene, siendo un artefacto un grupo de puntos contiguos de la malla con valores elevados de conductividad eléctrica. El software permitirá generar *datasets* especificando determinadas características deseadas y almacenar dichos *datasets* en el sistema.

- **Gestión de las tareas lanzadas por los usuarios**. El entrenamiento de modelos de *Machine Learning*, así como la generación de *datasets* son tareas costosas que requieren tiempos de computación elevados. El software deberá permitir que el usuario lance estas tareas en una cola de tareas, mientras continúa haciendo uso de la aplicación.

- **Gestión de los usuarios**. El software permitirá a los usuarios crear una cuenta personal. Los usuarios podrán entrenar modelos y generar *datasets*, los cuales podrán tener asignada una visibilidad pública o privada. Por tanto, no sólo será preciso gestionar la autenticación de los usuarios, sino también los permisos de los que disponen.

Capítulo 2

Planificación y presupuestos

En este capítulo se presentará la planificación propuesta para la realización del TFG, la cual constará de tres bloques: la planificación temporal, la gestión de costes y la gestión de riesgos. Para ejecutar con éxito las tareas asociadas a estos tres bloques, se tomará como referencia la guía del PMBOK (*Project Management Body of Knowledge*) [2]. La guía del PMBOK contiene las prácticas recomendadas para gestionar de forma eficaz un proyecto.

2.1. Planificación temporal

La planificación temporal del TFG consistirá en el desarrollo de su cronograma. Se llevarán a cabo los siguientes procesos recomendados por el PMBOK:

- Definición de las actividades.
- Secuenciación de las actividades.
- Estimación de la duración de las actividades.

2.1.1. Ciclo de vida

Antes de realizar las tareas asociadas a los tres procesos anteriores, es preciso seleccionar un ciclo de vida que se adecúe a las características del proyecto. A pesar de que los requisitos del proyecto son claros, existe un alto nivel de incertidumbre en lo relativo a la integración de las tecnologías que se emplearán en la fase de implementación, desconociéndose previamente si esta integración se podrá llevar a cabo sin dificultades.

Considerando el escenario anterior, se ha optado por emplear un ciclo de vida **incremental**. El ciclo de vida incremental consta de una fase de análisis y de una fase de diseño genéricas y de fases de diseño y codificación específicas en cada uno de los incrementos que se definan. Un enfoque habitual para ejecutar el ciclo

de vida incremental consiste en asociar una serie de requisitos a cada incremento, de forma que tras la finalización del incremento, se incorporen al software nuevas funcionalidades completamente operativas. Sin embargo, el enfoque que se adoptará en este proyecto será ligeramente diferente, puesto que cada incremento estará asociado a un módulo del sistema, el cual deberá ser integrado con los módulos desarrollados hasta ese momento.

2.1.2. EDT

La EDT (Estructura de Descomposición del Trabajo) permite subdividir los entregables del proyecto en componentes más pequeños, los cuales son más fáciles de gestionar. Mediante la EDT, se ejecutará el proceso de definición de las actividades, propuesto por el PMBOK.

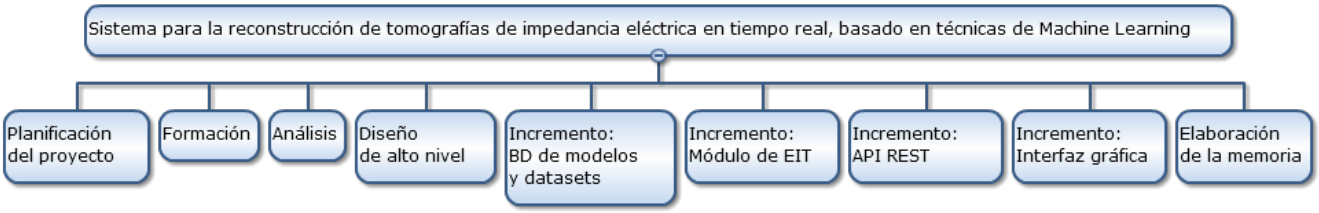

Figura 2.1: EDT de primer nivel.

En la figura 2.1 se muestra la EDT de primer nivel para el TFG. Como se puede observar, se han definido cuatro incrementos para el desarrollo del proyecto. Las actividades del diagrama anterior se han explotado en EDTs de segundo y tercer nivel (exceptuando las actividades de *Diseño de alto nivel* y *Elaboración de la memoria*), tal y como se puede ver en las figuras 2.2, 2.3, 2.4, 2.5, 2.6, 2.7 y 2.8.

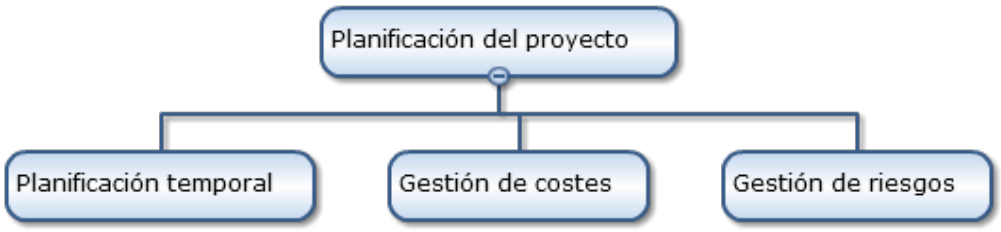

Figura 2.2: EDT de segundo nivel. Planificación del proyecto.

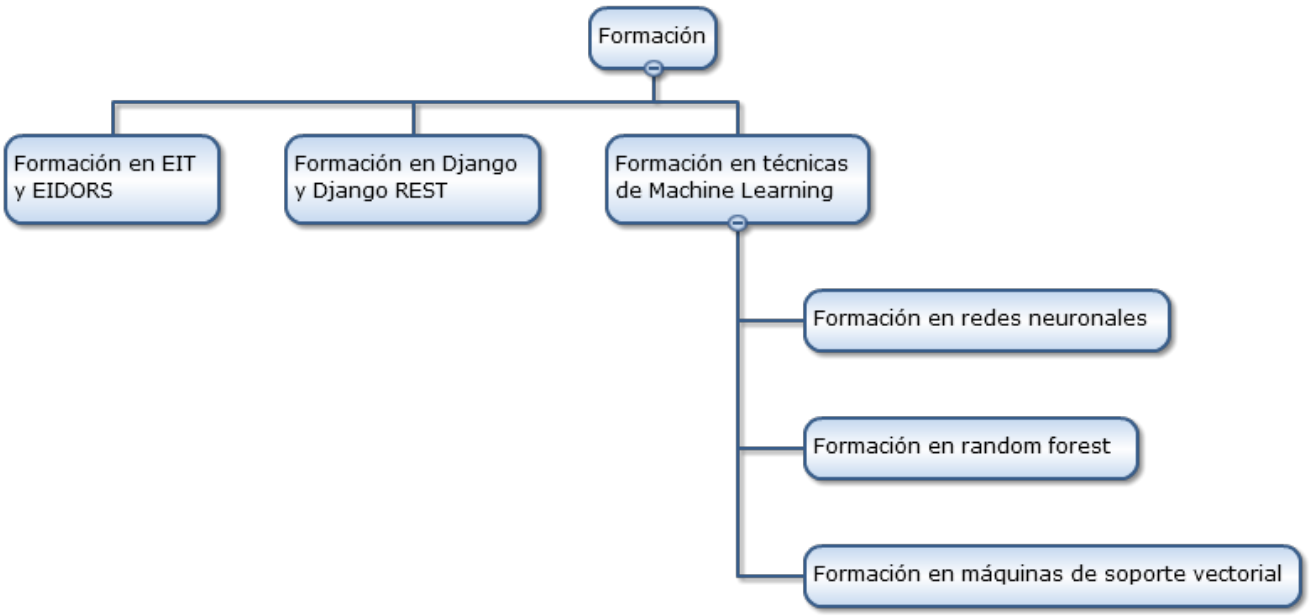

Figura 2.3: EDT de segundo nivel. Formación.

Figura 2.4: EDT de segundo nivel. Análisis.

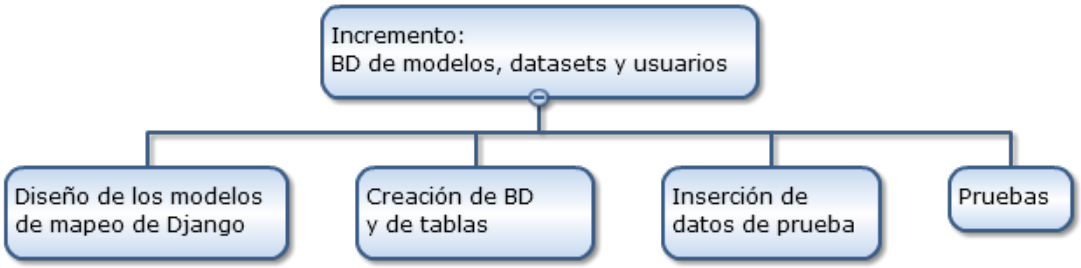

Figura 2.5: EDT de segundo nivel. Primer sprint: BD de modelos, *datasets* y usuarios

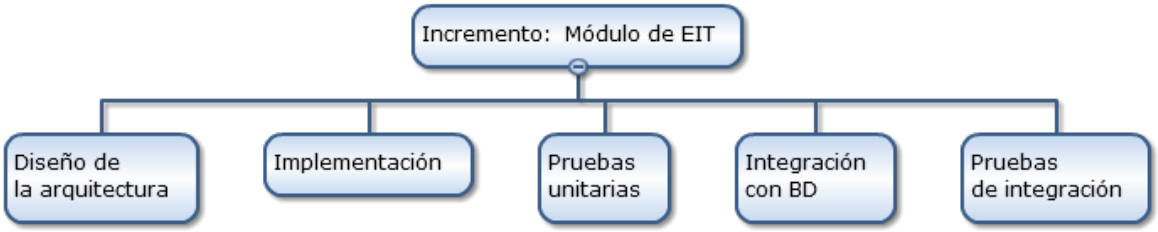

Figura 2.6: EDT de segundo nivel. Segundo sprint: módulo de EIT

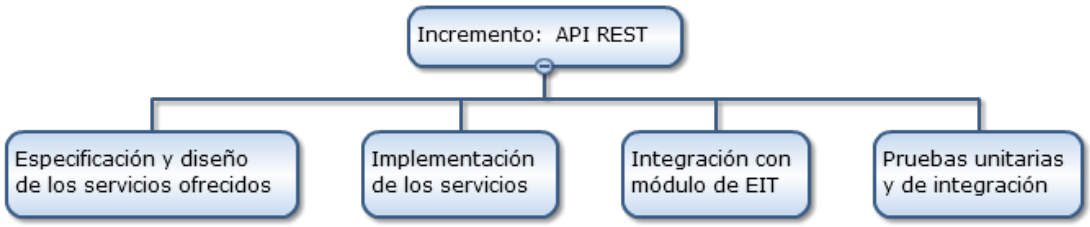

Figura 2.7: EDT de segundo nivel. Tercer sprint: API REST

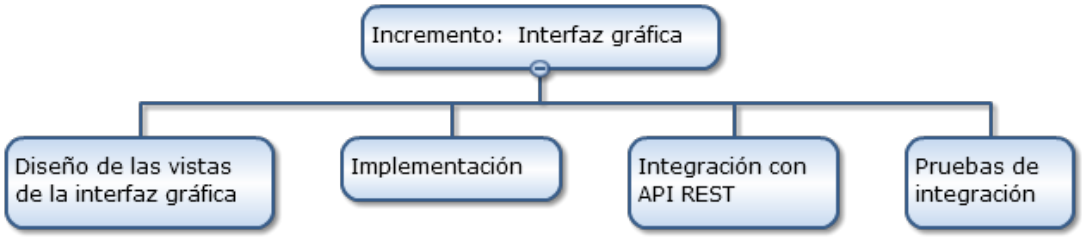

Figura 2.8: EDT de segundo nivel. Cuarto sprint: interfaz gráfica

2.1.3. Diagrama de Gantt

Una vez que se han definido las actividades de las que constará el proyecto, es necesario secuenciarlas y definir su duración. Para ello, se ha elaborado un diagrama de Gantt.

Cabe señalar que el anteproyecto de este TFG fue aprobado a principios del mes de marzo de 2020. No obstante, debido a la situación familiar del alumno derivada de la crisis del coronavirus, el proyecto no pudo iniciarse en ese mismo mes. El 5 de junio de 2020, la Secretaría Xeral de la USC publicó la Instrucción 4/2020 [3], en la cual se establecía que el alumnado de la USC podía solicitar el aplazamiento de la entrega del TFG por causas derivadas del coronavirus y, en caso de que dicha solicitud fuese aprobada, el TFG podría ser entregado fuera de plazo ordinario. El alumno realizó la solicitud y comenzó el TFG en el mes de junio de 2020. La planificación temporal propuesta en este apartado se realizó suponiendo que la solicitud del alumno sería aceptada y que, por tanto, podría entregar el TFG fuera del plazo ordinario, tal y como finalmente aconteció. En el apartado de gestión de riesgos (apartado 2.3) se incluyó un riesgo que contempla la posibilidad de que la solicitud de aplazamiento del alumno no fuese aceptada.

En el diagrama de Gantt que se muestra a continuación, no sólo se refleja el orden y la duración de las tareas a realizar, sino también la fecha de realización y el rol que desempeña el alumno en cada una de las tareas. Con el propósito de facilitar su visualización, se ha dividido el diagrama de Gantt en diferentes bloques, los cuales se muestran en las figuras 2.9, 2.10, 2.11, 2.12, 2.13, 2.14, 2.15 y 2.16.

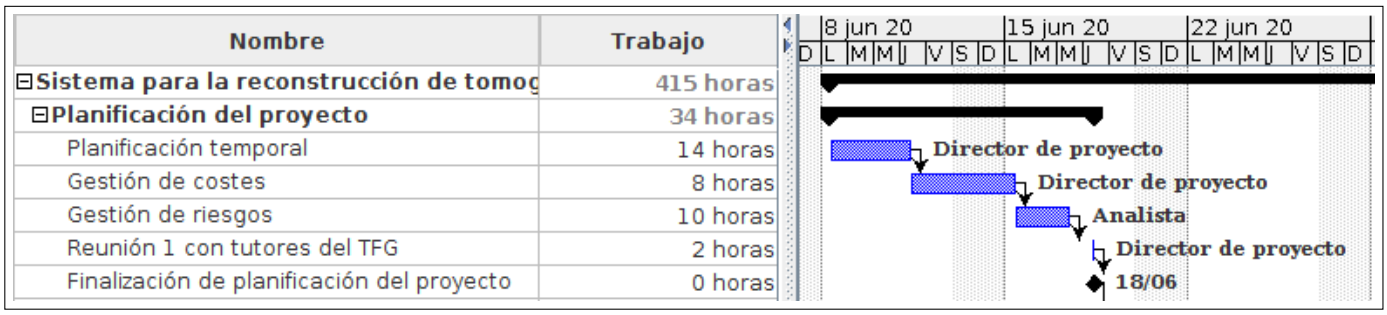

Figura 2.9: Diagrama de gantt. Planificación.

Figura 2.10: Diagrama de gantt. Análisis y diseño de alto nivel.

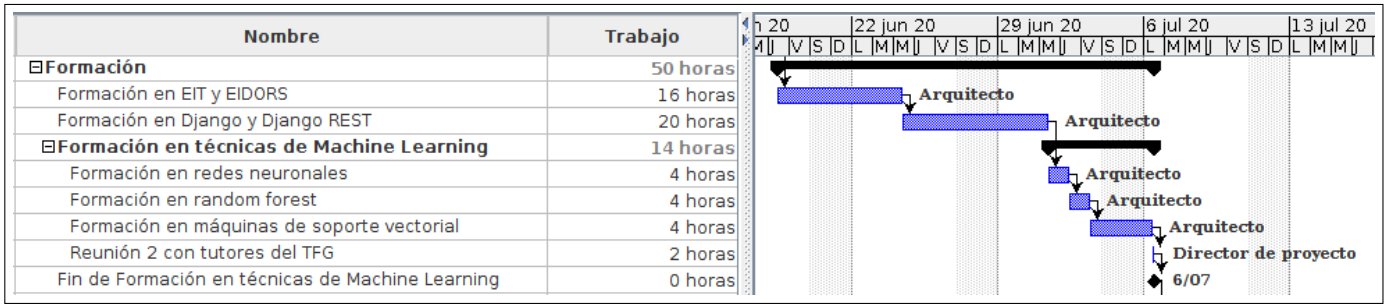

Figura 2.11: Diagrama de gantt. Formación.

Nombre	Trabajo
Incremento 1: BD de modelos, datasets y usuarios	9 horas
Diseño de los modelos de mapeo de Django	2 horas
Creación de BD y de tablas	0,5 horas
Inserción de datos de prueba	2 horas
Pruebas	3 horas
Reunión 4 con tutores del TFG	1,5 horas
FIn de Incremento 1	0 horas

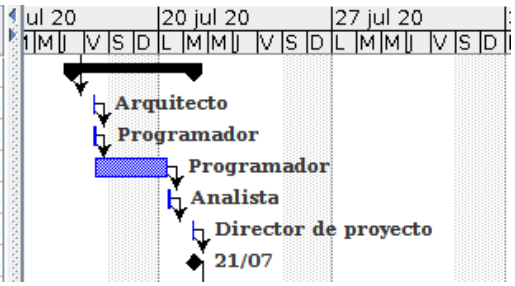

Figura 2.12: Diagrama de gantt. Incremento 1.

Nombre	Trabajo
Incremento 2: Módulo de EIT	104,5 horas
Diseño de la arquitectura	6 horas
Implementación	80 horas
Pruebas unitarias	10 horas
Integración con BD de modelos, datasets y usuarios	4 horas
Pruebas de integración	3 horas
Reunión 5 con tutores del TFG	1,5 horas
Fin de Incremento 2	0 horas

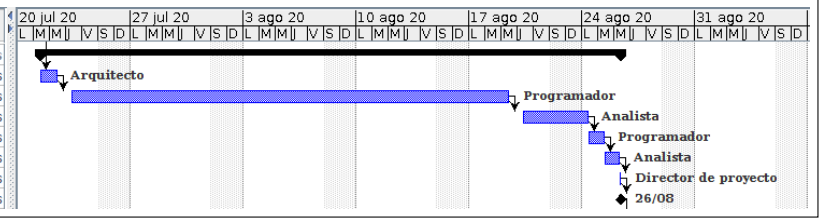

Figura 2.13: Diagrama de gantt. Incremento 2.

Nombre	Trabajo
Incremento 3: API REST	64,5 horas
Especificación y diseño de los servicios ofrecidos	8 horas
Implementación de los servicios	45 horas
Integración con módulo de EIT	4 horas
Pruebas unitarias y de integración	6 horas
Reunión 6 con tutores del TFG	1,5 horas
Fin de Incremento 3	0 horas

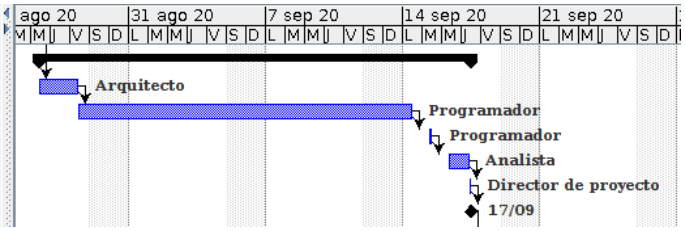

Figura 2.14: Diagrama de gantt. Incremento 3.

Nombre	Trabajo
Incremento 4: Interfaz gráfica	54,5 horas
Diseño de las vistas de la interfaz gráfica	8 horas
Implementación	25 horas
Integración con API REST	12 horas
Pruebas de integración	4 horas
Pruebas de usabilidad	4 horas
Reunión 7 con tutores del TFG	1,5 horas
Fin de Incremento 4	0 horas

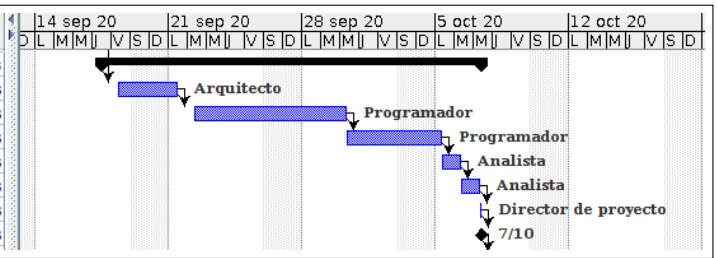

Figura 2.15: Diagrama de gantt. Incremento 4.

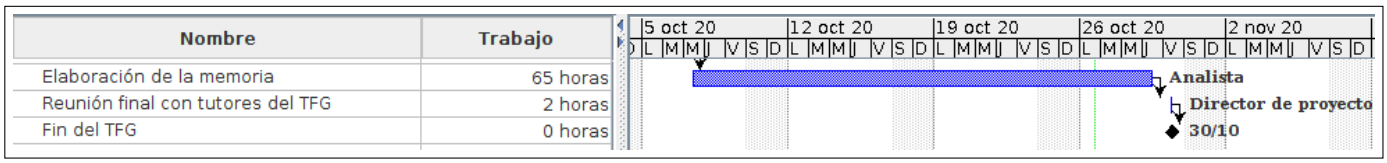

Figura 2.16: Diagrama de gantt. Elaboración de memoria y fin de TFG.

2.2. Gestión de costes

2.2.1. Costes de los recursos humanos

En el desarrollo del TFG participan tres personas: el alumno y los dos tutores. Los tutores serán considerados clientes, de manera que no tienen ningún coste asociado. Para calcular el coste relativo a las horas trabajadas por el alumno, se tendrán en consideración los roles que desempeñó en las diferentes tareas de las que consta el proyecto: director de proyecto, arquitecto de software, analista y programador. Se ha decidido tener en cuenta estos diferentes roles con el propósito de realizar una estimación lo más realista posible de un proyecto de Ingeniería de Software. En el diagrama de Gantt (figuras 2.9, 2.10, 2.11, 2.12, 2.13, 2.14, 2.15 y 2.16) se muestra qué rol desempeña el alumno para cada tarea.

Para estimar el coste de cada rol, se utilizará el estudio salarial sobre el sector TIC en Galicia elaborado por la consultora Vitae [4]. Siguiendo este informe, se supondrá que los sueldos brutos anuales del director de proyecto, del arquitecto de software, del analista y del programador son 36000 €, 29000 €, 18000 €y 16000 €, respectivamente. Por otra parte, se supondrá que los costes asociados a la Seguridad Social son del 33 % para todos los roles. Por tanto, para obtener el coste total anual de cada rol, se multiplicará su sueldo bruto anual por un factor de 1.33.

Una vez que se hayan calculado los costes totales anuales para los diferentes roles, es preciso calcular cuál es el coste por hora de cada rol. Para ello, se debe dividir el coste total anual entre el número de horas que cada rol trabajaría en un año a jornada completa. Para estimar cuál es el número de horas anuales, se empleará el *XIX Convenio colectivo del sector de empresas de ingeniería y oficinas de estudios técnicos*, disponible en el BOE [5]. Según el anterior convenio, el número máximo de horas anuales en jornada ordinaria para cada uno de los cuatro roles mencionados es de 1792 horas.

En la la tabla 2.1 se muestran los resultados obtenidos realizando los cálculos explicados a lo largo de este apartado.

Rol	Sueldo bruto anual	Coste total anual	Coste / hora
Director de proyecto	36000	47880	26.72
Arquitecto de software	29000	38570	21.52
Analista	18000	23940	13.36
Programador	16000	21280	11.88

Cuadro 2.1: Costes de los diferentes roles (€).

El coste total asociado a los recursos humanos se obtiene multiplicando el coste/hora de cada rol por el número de horas que el alumno trabaja desempeñando ese rol. El número de horas trabajadas por el alumno para cada rol puede consultarse en el diagrama de Gantt. En la tabla 2.2 se muestra el coste total para el proyecto que supone cada uno de los roles.

Rol	Nº de horas trabajadas	Coste / hora	Coste total
Director de proyecto	35.5	26.72	948.52
Arquitecto de software	76	21.52	1635.78
Analista	131	13.36	1750.08
Programador	172.5	11.88	2048.44

Cuadro 2.2: Costes totales de los diferentes roles (€).

Por tanto, el coste total asociado al trabajo del alumno es:

$$948.52 + 1635.78 + 1750.08 + 2048.44 = \mathbf{6382.81\ €}$$

2.2.2. Costes de los recursos materiales

Tal y como se explicará en el capítulo 4, las tecnologías y herramientas software empleadas para el proyecto son libres, de forma que su uso no supone ningún coste. Por tanto, el único recurso material cuya utilización implica un coste para el proyecto es el ordenador portátil con el que se ha realizado todo el trabajo.

No obstante, para imputar al proyecto el coste correcto correspondiente al uso del equipo informático, es necesario amortizarlo previamente. La **amortización** es la depreciación y pérdida de valor que sufre un activo con el paso del tiempo. El coste para el proyecto asociado al empleo del ordenador portátil será la pérdida de valor experimentada por éste a lo largo del proyecto.

El método más empleado para amortizar un activo es el empleo de las **tablas de amortización**. En la web de la Agencia Tributaria se puede encontrar la tabla de coeficientes de amortización lineal [6], que incluye un apartado para equipos informáticos, el cual se muestra a continuación.

Tipo de elemento	**Coeficiente lineal máximo**	**Período de años máximo**
Equipos electrónicos	20 %	10

Cuadro 2.3: Amortización de equipos electrónicos.

El equipo informático utilizado para realizar todo el trabajo es un Toshiba Satellite P50-C-13x, adquirido en enero de 2017 por 1020 euros. En primer lugar, se estimará el valor del equipo antes de iniciar el proyecto. Para ello, se aplicará el coeficiente lineal máximo de amortización para cada uno de los años de uso del equipo anteriores al desarrollo del proyecto: 2017, 2018 y 2019. De esta forma, se considera que el equipo cada año pierde un 20 % de su valor. Así, el valor del equipo al comienzo de 2020 es el siguiente:

$$1020 \cdot (1 - 0.2)^3 = \mathbf{522.24\ €}$$

Se considerará que el coste asociado al proyecto por el uso del portátil es el valor que pierde durante los 6 meses de 2020 que dura el proyecto. De esta forma, el coeficiente de desamortización que se aplicará será del 10 %, en lugar del 20 %, obteniendo el siguiente coste:

$$522.24 \cdot 0.1 = \mathbf{52.22\ €}$$

2.2.3. Coste del proyecto

El coste total se obtiene mediante la suma de los costes calculados en los apartados 2.2.1 y 2.2.2. Además, es necesario imputar también los **costes indirectos del proyecto** (luz, agua, gas e internet). Para ello, se multiplicará el coste total por un factor de 1.2, ya que se estima ya que se estima que los costes indirectos suponen el 20 % de los costes totales.

Costes de recursos humanos	6382.81
Costes de recursos materiales	52.22
Coste total	**6435.03**
Coste total con costes indirectos	**7722.04**

Cuadro 2.4: Coste total (€).

Como se puede ver en la tabla anterior, el coste total del proyecto es de **7722.04 €**.

2.3. Gestión de riesgos

En este apartado se definirá el plan de gestión de riesgos que se seguirá durante el desarrollo del TFG. Para ello, se seguirán las cuatro siguientes fases:

1. **Identificación de riesgos**. Se elabora una lista con los diferentes riesgos a los que está expuesto el TFG.

2. **Análisis de riesgos**. Se analiza la probabilidad de que cada uno de los riesgos se produzca, así como su impacto sobre el desarrollo en caso de que el riesgo se produzca.

3. **Planificación de riesgos**. Para los riesgos críticos, se establece un plan de actuación para tratar el riesgo. Existen diferentes tipos de planes:

   - **Prevención**: se reduce de la probabilidad de que el riesgo suceda.
   - **Minimización**: se reduce el impacto del riesgo en caso de que éste suceda.
   - **Contingencia**: se diseña un plan que se lleva a cabo una vez que suceda el riesgo.
   - **Transferencia**: se transfiere el riesgo a un tercero a través de la asunción de un coste.
   - **Aceptación**: se acepta el riesgo, asumiendo el coste que el riesgo implica.
   - **Evitación**: se reduce a 0 la probabilidad de que el riesgo suceda.

4. **Supervisión de riesgos**. Se define un indicador (el cual debe ser fácilmente medible) para cada riesgo y un umbral asociado a cada indicador, de manera que si en un momento dado se sobrepasa el umbral, se considera que la probabilidad de que el riesgo suceda inminentemente es elevada.

A pesar de que se ha utilizado la metodología propuesta en el PMBOK, en la fase de planificación de riesgos se han dividido los planes de mitigación mencionados en el PMBOK en dos tipos de planes específicos: los planes de prevención y los planes de minimización. Ésta es la terminología empleada por Sommerville [7].

2.3.1. Definición de las categorías de probabilidad e impacto

En este apartado se definirán las tres categorías de probabilidad y las cuatro categorías de impacto mediante las cuales se clasificarán los riesgos.

Las tres categorías de probabilidad que se contemplan son las siguientes: baja, media y alta.

Categoría	Probabilidad
Baja	Inferior o igual a 20 %.
Media	Mayor que 20 % y menor o igual que 70 %.
Alta	Mayor que 70 %.

Cuadro 2.5: Categorías de probabilidad.

Las cuatro categorías de impacto que se contemplan son las siguientes: insignificante, tolerable, serio y catastrófico.

Impacto	Descripción
Insignificante	La ocurrencia de un riesgo de impacto insignificante implicaría el incremento en hasta un 5 % de las horas totales del proyecto, lo que supondría realizar hasta **21 horas adicionales** de trabajo.
Tolerable	La ocurrencia de un riesgo de impacto tolerable implicaría el incremento en hasta un 10 % de las horas totales del proyecto, lo que supondría realizar hasta 42 horas adicionales de trabajo.
Serio	La ocurrencia de un riesgo de impacto serio implicaría el incremento en hasta un 15 % de las horas totales del proyecto, lo que supondría realizar hasta 63 horas adicionales de trabajo.
Catastrófico	La ocurrencia de un riesgo de impacto catastrófico implicaría el incremento en más de un 15 % de las horas totales del proyecto, lo que supondría que el TFG no se podría entregar dentro del plazo previsto.

Cuadro 2.6: Categorías de impacto.

2.3.2. Identificación y análisis de riesgos

En este apartado se expondrán los riesgos identificados que pueden afectar al desarrollo del TFG. A cada riesgo se le asignará una probabilidad de ocurrencia y un impacto, empleando las categorías definidas en el apartado anterior.

ID	Riesgo	Descripción	P	I

RSG1	Problemas de compatibilidad entre la biblioteca EIDORS, Matlab y Python	El sistema cuenta con módulos que utilizan diferentes lenguajes de programación, de forma que pueden surgir problemas en la integración de dichos módulos.	Alta	Serio
RSG2	Pérdida de información	El trabajo se realizará íntegramente en el equipo del alumno, de forma que la pérdida del portátil o la corrupción de la información almacenada en él supondría la pérdida total del trabajo realizado para el TFG.	Baja	Catastrófico
RSG3	Tiempo de formación superior al estimado	Una parte de las horas de dedicación estimadas se corresponde con actividades de formación en EIT, EIDORS, técnicas de *Machine Learning* y Python. La estimación de la duración de estas actividades tiene un alto grado de incertidumbre, por lo que el alumno podría necesitar más tiempo del previsto para adquirir los conocimientos necesarios. En esta situación, toda la planificación se vería afectada.	Alta	Tolerable
RSG4	Mala comunicación entre el alumno y los tutores	El tutor y el cotutor trabajan en proyectos diferentes y participan en congresos periódicamente, por lo que no siempre será posible establecer una reunión en la que puedan participar ambos. Una mala coordinación entre el alumno, el tutor y el cotutor puede retrasar el trabajo.	Alta	Serio

RSG5	Funcionamiento incorrecto de la biblioteca EIDORS	EIDORS es un software libre y muy especializado, de forma que el número de usuarios que lo utilizan es relativamente reducido. Por este motivo, no ha sido "testeadoïntensivamente, por lo que durante el desarrollo del sistema pueden surgir problemas con el uso de esta biblioteca, lo que retrasaría el trabajo e incluso podría impedir la implementación de algunas de las funcionalidades previstas en el análisis de requisitos	Baja	Serio
RSG6	Mala definición de los incrementos	Tal y como se ha definido en el apartado 2.1.1, se utilizará un ciclo de vida iterativo incremental. Una definición incorrecta de los incrementos (en la que se asigne el mismo tiempo de trabajo a cada incremento sin considerar la carga de trabajo de la que consta el incremento) puede implicar que en los incrementos finales el alumno deba hacer un número de horas semanales superior al estimado.	Baja	Tolerable
RSG7	Eliminación accidental de un archivo o directorio	Durante el desarrollo del TFG, el alumno podría eliminar accidentalmente un archivo o un directorio. En este último caso, podría perder parte o la totalidad del trabajo realizado, lo que imposibilitaría presentar el TFG en el plazo previsto.	Media	Serio

RSG8	Dificultad para replicar los experimentos del artículo de referencia	Los experimentos realizados en el artículo utilizado como punto de partida para realizar este TFG tienen un cierto grado de complejidad, por lo que el alumno podría necesitar más horas de las previstas para replicarlos y comprender su funcionamiento.	Media	Tolerable
RSG9	Rechazo de la solicitud de aplazamiento por parte de la Secretaría Xeral de la USC	Tal y como se explicó en el apartado 2.1.3, la planificación para el TFG ha sido elaborada con el propósito de presentar el proyecto fuera de plazo ordinario, como consecuencia de la situación personal del alumno (al amparo de la Instrucción 4/2020 de la Secretaría Xeral). No obstante, la solicitud enviada por el alumno debe ser aprobada por la propia Secretaría Xeral. En caso de que dicha solicitud fuese rechazada, el TFG no podría presentarse en el curso académico 2019/2020.	Baja	Catastrófico

Cuadro 2.7: Lista de riesgos.

A continuación, se muestra una matriz de probabilidad-impacto. En la región marcada en color naranja se encuentran los riesgos críticos para el TFG.

Probabilidad	**Alta**		RSG3	RSG1,RSG4	
	Media		RSG8	RSG7	
	Baja		RSG6	RSG5	RSG2, RSG9
		Insignificante	**Tolerable**	**Serio**	**Catastrófico**
		Impacto			

Cuadro 2.8: Matriz de probabilidad-impacto.

2.3.3. Planificación de riesgos

Una vez que se han identificado y analizado los riesgos que pueden afectar al TFG, se definirán las estrategias para tratar los riesgos que se encuentran en la región naranja de la matriz de probabilidad-impacto.

RSG1	**Problemas de compatibilidad entre la biblioteca EIDORS, Matlab y Python**
Tipo de estrategia	Prevención y contingencia.
Descripción	Antes de iniciar la fase de codificación, se realizará un análisis de diferentes frameworks que permitan ejecutar funciones de Matlab desde Python. Las pruebas a realizar consistirán en la ejecución de diferentes funciones de la biblioteca EIDORS desde Python, utilizando cada uno de los frameworks considerados. Se seleccionará el framework mediante el cual se obtenga un menor número de llamadas fallidas a funciones. En caso de que no se encuentre ningún framework que proporcione un grado de compatibilidad aceptable, las llamadas a Matlab desde Python se realizarán mediante la ejecución scripts.
RSG2	**Imposibilidad de acceder al trabajo realizado**
Tipo de estrategia	Evitación.
Descripción	El alumno creará la carpeta *TFG_AllerDominguez* en un directorio de su equipo sincronizado con una cuenta de Dropbox, utilizando la versión gratuita. En esta carpeta almacenará todos los ficheros correspondientes al TFG. En caso de que se corrompa el equipo del alumno, podrá acceder igualmente al trabajo realizado a través de su cuenta de Dropbox, empleando cualquier otro equipo. Esta carpeta estará sincronizada con sendas carpetas de los codirectores, con lo que se triplican los almacenamientos físicos eliminando así el riesgo.
RSG3	**Tiempo de formación superior al estimado**
Tipo de estrategia	Minimización

Descripción	La planificación se realizará de tal forma que se dejará libre la última semana de junio. En caso de que durante las actividades de formación se detecte que el tiempo necesario es superior al estimado, se podrá retrasar la planificación hasta una semana, obteniendo así el tiempo adicional necesario para la formación.
RSG4	**Poca disponibilidad temporal por parte de los tutores**
Tipo de estrategia	Minimización
Descripción	En las semanas en las que no se pueda llevar a cabo la reunión prevista con ambos tutores por los compromisos de éstos, se realizará una reunión a través de Skype en horario de tarde, atendiendo a la disponibilidad de los tutores. De esta forma, los dos tutores podrán participar simultáneamente en la reunión. Por otra parte, la carpeta de Dropbox utilizada por el alumno para almacenar la documentación y el código del TFG será compartida con ambos tutores, pudiendo acceder al trabajo del alumno cuando lo deseen, de manera que se puedan mantener informados en todo momento de cuál el progreso del alumno.
RSG7	**Eliminación accidental de un archivo o directorio**
Tipo de estrategia	Contingencia y minimización.
Descripción	Si el alumno detecta la pérdida de los archivos en un plazo igual o inferior a 30 días, podrá recuperarlos accediendo a su cuenta de Dropbox desde un navegador. En caso de que hayan transcurrido más de 30 días, esta opción no será posible, por lo que se hará uso de backups para disponer de una copia de los archivos. Se realizará un backup completo semanalmente del directorio que contiene los ficheros del TFG. El alumno almacenará dicho backup en un disco duro externo. De esta forma, si detecta que se ha eliminado un archivo hace más de 30 días, podrá recuperarlo a través del backup almacenado en el disco duro externo.

RSG9	**Rechazo de la solicitud de aplazamiento por parte de la Secretaría Xeral de la USC**
Tipo de estrategia	Minimización y contingencia.
Descripción	Por un lado, se utilizará una estrategia de minimización con el propósito de reducir la probabilidad de que la solicitud de aplazamiento del alumno sea rechazada. Para ello, en la solicitud que el alumno envíe a la Secretaría Xeral, se adjuntará en formato PDF la documentación que acredita la situación personal y familiar del alumno. En caso de que la estrategia de minimización no funcione y se rechace la solicitud, se utilizará un plan de contingencia, consistente en la reelaboración de la planificación del proyecto, con el propósito de presentarlo en el mes de febrero de 2021.

Cuadro 2.9: Planificación de los riesgos críticos.

2.3.4. Supervisión de riesgos

En este apartado se definirán **indicadores** para los riesgos que se tratarán mediante estrategias de minimización y de contingencia. Para cada indicador, se definirá un **umbral**, de manera que, si el valor del indicador alcanza o supera el umbral establecido, se considerará que la probabilidad de que acontezca el riesgo inminentemente es elevada.

RSG3	**Tiempo de formación superior al estimado**
Indicador	Cociente entre el número de horas planificadas para las actividades de formación realizadas hasta el momento y el número de horas reales necesitadas. Si se alcanza un valor igual o inferior al umbral, se considera que existe una probabilidad elevada de que se produzca el riesgo.
Umbral	Valor del cociente de 0.85
RSG4	**Poca disponibilidad temporal por parte de los tutores**
Indicador	Número de reuniones consecutivas canceladas o a las que alguno de los tutores no ha podido asistir.
Umbral	3 reuniones.

RSG5	**Funcionamiento incorrecto de la biblioteca EIDORS**
Indicador	Número de errores irresolubles en las llamadas a funciones de EIDORS.
Umbral	5 errores
RSG7	**Eliminación accidental de un archivo o directorio**
Indicador	Número de archivos eliminados de forma accidental.
Umbral	3 archivos eliminados accidentalmente.

Cuadro 2.10: Supervisión de los riesgos críticos.

Para el riesgo RSG9 (rechazo de la solicitud de aplazamiento por parte de la Secretaría Xeral de la USC), a pesar de utilizar una estrategia de minimización, no se incluyen indicadores, puesto que no existen factores externos analizables por parte del alumno que permitan predecir la decisión que tome finalmente la Secretaría Xeral.

Capítulo 3

Especificación de requisitos y casos de uso

Un requisito puede definirse como una condición o capacidad que necesita el usuario para resolver un problema o conseguir un objetivo determinado. En este apartado se describirán los **requisitos funcionales**, los **requisitos no funcionales** y los **requisitos de información** del sistema a desarrollar. Por otro lado, a partir de los requisitos funcionales, se definirán los casos de uso más relevantes de la aplicación.

Además de la descripción, para cada requisito se han incluido dos atributos adicionales:

- **Importancia**. Indica el interés que tiene un requisito para el cliente del proyecto (se considerará que tanto el tutor como el cotutor son los clientes del proyecto). Este atributo podrá adoptar los siguientes valores:
  - **Normal**. El cliente lo pide y lo necesita.
  - **Esperado**. El cliente no lo pide, pero lo necesita.
  - **Estimulante**. El cliente no lo pide, pero la inclusión del requisito en el sistema le resulta satisfactoria.
- **Estabilidad**. Indica la probabilidad de que un requisito cambie a lo largo del tiempo. Puede adoptar tres valores: baja, media, alta.

3.1. Requisitos funcionales

Los requisitos funcionales definen las funcionalidades que implementará la aplicación. Se han identificado 23 requisitos funcionales.

RF1	**Iniciar sesión**
Descripción	El sistema deberá permitir a un usuario iniciar sesión introduciendo su nombre de usuario y su contraseña.
Importancia	Esperado
Estabilidad	Alta

Cuadro 3.1: Requisito funcional RF1.

RF2	**Cerrar sesión**
Descripción	El sistema deberá permitir a un usuario cerrar una sesión abierta, regresando a la pantalla de inicio.
Importancia	Esperado
Estabilidad	Alta

Cuadro 3.2: Requisito funcional RF2.

RF3	**Registrar nuevo usuario**
Descripción	El sistema permitirá a un usuario registrarse en el sistema, introduciendo los siguientes datos: *nickname*, contraseña, correo electrónico, nombre y apellidos. Una vez que el usuario haya cubierto y enviado el formulario de registro, se enviará le un correo de forma automática para que confirme su registro.
Importancia	Esperado
Estabilidad	Alta

Cuadro 3.3: Requisito funcional RF3.

RF4	**Editar usuario**
Descripción	El sistema permitirá a un usuario editar la siguiente información de su cuenta: contraseña, correo electrónico, nombre y apellidos.
Importancia	Esperado
Estabilidad	Media

Cuadro 3.4: Requisito funcional RF4.

RF5	**Eliminar usuario**
Descripción	El sistema permitirá a un administrador eliminar a un usuario registrado, de manera que todos sus datos serán borrados del sistema.
Importancia	Esperado
Estabilidad	Alta

Cuadro 3.5: Requisito funcional RF5.

RF6	**Entrenar red neuronal**
Descripción	El sistema permitirá a un usuario entrenar una red neuronal. El usuario podrá introducir o seleccionar los siguientes parámetros del modelo: ▪ *Dataset* con el que desea entrenar el modelo. ▪ Número de capas ocultas. ▪ Número de neuronas en cada una de las capas ocultas. ▪ Función de activación de las capas ocultas. ▪ Función de activación de la capa de salida. ▪ Tamaño de los lotes. ▪ Learning rate. ▪ Momentum. ▪ Visibilidad del modelo (público o privado). ▪ Métricas que desea emplear para evaluar el modelo.
Importancia	Esperado
Estabilidad	Media

Cuadro 3.6: Requisito funcional RF6.

RF7	**Entrenar *random forest***
Descripción	El sistema permitirá a un usuario entrenar un *random forest*. El usuario podrá introducir o seleccionar los siguientes parámetros del modelo. ▪ *Dataset* con el que desea entrenar el modelo. ▪ Número de estimadores empleados. ▪ Profundidad máxima de cada rama. ▪ Número mínimo de muestras necesarias para dividir un nodo interno. ▪ Número mínimo de muestras necesarias para que un nodo pueda ser nodo hoja. ▪ Visibilidad del modelo. ▪ Métricas que desea emplear para evaluar el modelo.
Importancia	Esperado
Estabilidad	Media

Cuadro 3.7: Requisito funcional RF7.

RF8	**Entrenar máquina de soporte vectorial**
Descripción	El sistema permitirá a un usuario entrenar una máquina de soporte vectorial. El usuario podrá introducir o seleccionar los siguientes parámetros del modelo: ▪ *Dataset* con el que desea entrenar el modelo. ▪ Kernel. ▪ Grado. Esta información sólo será relevante en caso de emplear un kernel de tipo polinómico. ▪ Gamma. ▪ Coeficiente 0. ▪ Tolerancia. ▪ Constante C. ▪ Epsilon. ▪ Visibilidad del modelo. ▪ Métricas que desea emplear para evaluar el modelo.
Importancia	Esperado
Estabilidad	Media

Cuadro 3.8: Requisito funcional RF8.

RF9	**Almacenar modelo**
Descripción	El sistema permitirá a un usuario almacenar un modelo entrenado, de forma que dicho modelo pueda ser utilizado posteriormente para reconstruir la imagen de un cuerpo.
Importancia	Normal
Estabilidad	Media

Cuadro 3.9: Requisito funcional RF9.

RF10	**Consultar modelos**
Descripción	El sistema permitirá a un usuario consultar la lista de modelos almacenados en el sistema que sean de tipo público o que hayan sido entrenados por el propio usuario. El usuario podrá consultar las características particulares de cada uno de los modelos.
Importancia	Normal
Estabilidad	Media

Cuadro 3.10: Requisito funcional RF10.

RF11	**Eliminar modelo**
Descripción	El sistema permitirá a un usuario eliminar un modelo almacenado, siempre y cuando este modelo haya sido generado por el propio usuario.
Importancia	Normal
Estabilidad	Esperado

Cuadro 3.11: Requisito funcional RF11.

RF12	**Comparación de modelos**
Descripción	El sistema permitirá a un usuario comparar hasta cuatro modelos de forma simultánea. La comparación se llevará a cabo realizando predicciones sobre el *dataset* que elija el usuario y evaluando dichas predicciones mediante las métricas seleccionadas también por el usuario. El usuario podrá seleccionar las siguientes métricas: ▪ Error cuadrático medio (MSE). ▪ Error absoluto medio (MAE). ▪ Error logarítmico cuadrático medio (MSLE). ▪ Porcentaje de acierto. Además de mostrar al usuario una comparativa de los resultados obtenidos con los diferentes modelos para cada uno de las métricas, se reconstruirá con cada uno de los modelos una misma imagen elegida de forma aleatoria de entre las que forman el *dataset* seleccionado previamente por el usuario.
Importancia	Estimulante
Estabilidad	Baja

Cuadro 3.12: Requisito funcional RF12.

RF13	**Reconstruir imagen**
Descripción	El sistema permitirá a un usuario realizar la reconstrucción de la imagen de un cuerpo, seleccionando para ello uno de los modelos almacenados en el propio sistema, así como el *dataset* del cual desea obtener el cuerpo para analizar. El usuario podrá visualizar la reconstrucción de la imagen con los valores de conductividad reales, así como la reconstrucción de la imagen con los valores de conductividad predichos por el modelo.
Importancia	Esperado
Estabilidad	Media

Cuadro 3.13: Requisito funcional RF13.

RF14	**Predecir conductividad eléctrica**
Descripción	El sistema permitirá a un usuario subir un fichero que contenga los voltajes asociados a un conjunto de mallas y realizar la predicción de las conductividades de dichas mallas, seleccionando para ello uno de los modelos almacenados en el propio sistema. Una vez que se haya realizado la predicción, el usuario podrá reconstruir la imagen cualquiera de las mallas.
Importancia	Esperado
Estabilidad	Media

Cuadro 3.14: Requisito funcional RF14.

RF15	**Añadir *dataset***
Descripción	El sistema permitirá a un usuario añadir un nuevo *dataset* al sistema. El usuario podrá establecer si se trata de un *dataset* público o privado. De manera análoga al caso de los modelos, en caso de tratarse de un *dataset* privado, éste no podrá ser utilizado por el resto de usuarios.
Importancia	Estimulante
Estabilidad	Media

Cuadro 3.15: Requisito funcional RF15.

RF16	**Generar *dataset***
Descripción	El sistema permitirá a un usuario crear un *dataset*. Las mallas del *dataset* se generarán de manera pseudoaleatoria. No obstante, el usuario podrá determinar las siguientes características del *dataset*: ▪ Número de mallas de uno, dos y tres artefactos. ▪ Radio mínimo y máximo de los artefactos. ▪ Visibilidad del *dataset*. ▪ Semilla de generación de las mallas.
Importancia	Estimulante
Estabilidad	Media

Cuadro 3.16: Requisito funcional RF16.

RF17	**Almacenar *dataset***
Descripción	El sistema permitirá a un usuario almacenar un *dataset* generado, de forma que dicho *dataset* pueda ser utilizado posteriormente para entrenar o comparar modelos.
Importancia	Normal
Estabilidad	Media

Cuadro 3.17: Requisito funcional RF17.

RF18	**Consultar datasets**
Descripción	El sistema permitirá a un usuario consultar la lista de *datasets* almacenados en el sistema que sean de tipo público o que hayan sido generados o añadidos por el propio usuario. El usuario podrá consultar las características particulares de cada uno de los *datasets*.
Importancia	Estimulante
Estabilidad	Alta

Cuadro 3.18: Requisito funcional RF18.

RF19	**Descargar *dataset***
Descripción	El sistema permitirá a un usuario descargar en formato CSV los datasets públicos almacenados en el sistema o que hayan sido generados o añadidos por el propio usuario.
Importancia	Estimulante
Estabilidad	Alta

Cuadro 3.19: Requisito funcional RF19.

RF20	**Eliminar dataset**
Descripción	El sistema permitirá a un usuario eliminar los datasets públicos almacenados en el sistema que hayan sido generados o añadidos por el propio usuario.
Importancia	Estimulante
Estabilidad	Alta

Cuadro 3.20: Requisito funcional RF20.

RF21	**Consultar entrenamientos**
Descripción	El sistema permitirá a un usuario consultar qué entrenamientos de modelos están en curso en un momento dado, así como los entrenamientos que ya han finalizado.
Importancia	Estimulante
Estabilidad	Media

Cuadro 3.21: Requisito funcional RF21.

RF22	**Consultar datasets en generación**
Descripción	El sistema permitirá a un usuario consultar qué entrenamientos de modelos están en curso en un momento dado, así como los entrenamientos que ya han finalizado.
Importancia	Estimulante
Estabilidad	Media

Cuadro 3.22: Requisito funcional RF22.

RF23	**Visualización de corte transversal de una imagen**
Descripción	El sistema permitirá a un usuario generar cortes sobre el eje Y de una imagen reconstruida. El usuario podrá seleccionar el valor del eje Y para el cual desea realizar el corte. El resultado de la realización de un corte es una gráfica en la que se podrá visualizar cuáles son los valores de conductividad de un cuerpo (tanto los valores reales como los predichos) a lo largo del eje X de dicho cuerpo, para el eje Y fijado por el propio usuario.
Importancia	Estimulante
Estabilidad	Alta

Cuadro 3.23: Requisito funcional RF23.

RF24	**Ayuda contextual**
Descripción	El sistema permitirá al usuario consultar información de ayuda relativa a la pantalla en la que se encuentre en un momento determinado. De este modo, cuando el usuario solicite ayuda, se mostrará únicamente la información relevante para la pantalla desde la que haya solicitado la ayuda.
Importancia	Estimulante
Estabilidad	Alta

Cuadro 3.24: Requisito funcional RF24.

3.2. Requisitos no funcionales

Los **requisitos no funcionales** son restricciones que el sistema debe cumplir. Se han identificado tres requisitos no funcionales.

RNF-1	**Software libre**
Descripción	El sistema a desarrollar deberá diseñarse e implementarse utilizando únicamente herramientas y tecnologías de uso libre.
Importancia	Normal
Estabilidad	Alta

Cuadro 3.25: Requisito no funcional RNF1.

RNF-2	**Restricción en las eliminaciones**
Descripción	Un usuario no podrá nunca eliminar los modelos y datasets generados por otro usuario.
Importancia	Normal
Estabilidad	Alta

Cuadro 3.26: Requisito funcional RNF2.

RNF-3	**Usabilidad**
Descripción	El sistema deberá obtener una puntuación media igual o superior a 4/5 en el test de usabilidad definido en el apartado 7.2.1 , tras ser evaluado por al menos cuatro usuarios.
Importancia	Estimulante
Estabilidad	Alta

Cuadro 3.27: Requisito funcional RNF3.

3.3. Requisitos de información

Los requisitos de información indican qué datos se guardarán en almacenamiento persistente. Se han identificado nueve requisitos de información.

RI1	**Usuario**
Descripción	El sistema deberá almacenar la siguiente información sobre cada usuario: ▪ Nombre de usuario. ▪ Contraseña resumida mediante la función hash SHA256. ▪ Correo electrónico. ▪ Tipo de usuario. ▪ Nombre completo.
Importancia	Esperado
Estabilidad	Alta

Cuadro 3.28: Requisito funcional RI1.

RI2	**Modelo genérico**
Descripción	El sistema deberá incluir la siguiente información sobre cada modelo almacenado: ▪ Identificador. ▪ Tipo. ▪ Fecha y hora de inicio de entrenamiento. ▪ Fecha y hora de finalización de entrenamiento. ▪ Creador. Es el usuario que ha generado el modelo. ▪ Path. Es el directorio en el que se encuentra el binario del modelo. ▪ *Dataset*. Es el identificador del *dataset* utilizado para entrenar el modelo. ▪ Umbral de postprocesado. Es el valor calculado mediante la técnica de las curvas ROC que se utilizará para aplicar postprocesado en las predicciones realizadas con el modelo (véase apartado 6.3).
Importancia	Normal
Estabilidad	Alta

Cuadro 3.29: Requisito funcional RI2.

RI3	**Modelo de red neuronal**
Descripción	El sistema deberá incluir la siguiente información sobre cada modelo de red neuronal almacenado: ▪ Identificador del modelo genérico al cual está asociado. ▪ Número de capas ocultas. ▪ Número de neuronas en cada una de las capas ocultas. ▪ Función de activación de las capas ocultas. ▪ Función de activación de la capa de salida. ▪ Tamaño de los lotes. ▪ Learning rate. ▪ Momentum. ▪ Archivo binario con la arquitectura de la red. ▪ Archivo binario con los pesos de la red.
Importancia	Esperado
Estabilidad	Media

Cuadro 3.30: Requisito funcional RI3.

RI4	**Modelo de *random forest***
Descripción	El sistema deberá incluir la siguiente información sobre cada modelo de *random forest* almacenado: ▪ Identificador del modelo genérico al cual está asociado. ▪ Número de estimadores empleados. ▪ Profundidad máxima de cada rama. ▪ Número mínimo de muestras necesarias para dividir un nodo interno. ▪ Número mínimo de muestras necesarias para que un nodo pueda ser nodo hoja. ▪ Path del fichero binario que contiene el modelo entrenado.
Importancia	Esperado
Estabilidad	Media

Cuadro 3.31: Requisito funcional RI4.

RI5	**Modelo de máquina de soporte vectorial**
Descripción	El sistema deberá incluir la siguiente información sobre cada modelo de máquina de soporte vectorial almacenado: ▪ Identificador del modelo genérico al cual está asociado. ▪ Kernel. ▪ Grado. Esta información sólo será relevante en caso de emplear un kernel de tipo polinómico. ▪ Gamma. ▪ Coeficiente 0. ▪ Tolerancia. ▪ Constante C. ▪ Epsilon. ▪ Path del fichero binario que contiene el modelo entrenado.
Importancia	Esperado
Estabilidad	Media

Cuadro 3.32: Requisito funcional RI5.

RI6	***Dataset***
Descripción	El sistema deberá almacenar la siguiente información sobre cada *dataset*: ▪ Identificador. ▪ Semilla de generación. ▪ Radio mínimo de los artefactos. ▪ Radio máximo de los artefactos. ▪ Visibilidad.
Importancia	Normal
Estabilidad	Media

Cuadro 3.33: Requisito funcional RI6.

RI7	**Malla**
Descripción	El sistema deberá almacenar la siguiente información sobre cada malla: ▪ Identificador del *dataset* al que pertenece. ▪ Índice de la malla. ▪ Número de artefactos contenidos por la malla. ▪ Voltajes. ▪ Impedancias.
Importancia	Normal
Estabilidad	Media

Cuadro 3.34: Requisito funcional RI7.

RI8	**Métrica**
Descripción	El sistema deberá incluir la siguiente información sobre cada métrica almacenada: ▪ Identificador del modelo genérico al que está asociada la métrica. ▪ Nombre de la métrica. ▪ Valor de la métrica.
Importancia	Estimulante
Estabilidad	Media

Cuadro 3.35: Requisito funcional RI8.

RI9	**Matriz de confusión**
Descripción	El sistema deberá incluir la siguiente información sobre cada matriz de confusión almacenada: ▪ Identificador del modelo genérico al que está asociada la matriz de confusión. ▪ Número de verdaderos negativos. ▪ Número de verdaderos positivos. ▪ Número de falsos negativos. ▪ Número de falsos positivos.
Importancia	Estimulante
Estabilidad	Media

Cuadro 3.36: Requisito funcional RI9.

3.4. Casos de uso

Un **caso de uso** puede definirse como un conjunto de secuencias de acciones que ejecuta un sistema para producir un resultado observable de valor para un actor. En este apartado **se describirán los casos de uso del sistema de mayor importancia**. Para cada caso de uso se incluirá un escenario principal y, en algunos de los casos de uso, se describirán ciertos escenarios alternativos.

3.4.1. Diagrama de casos de uso

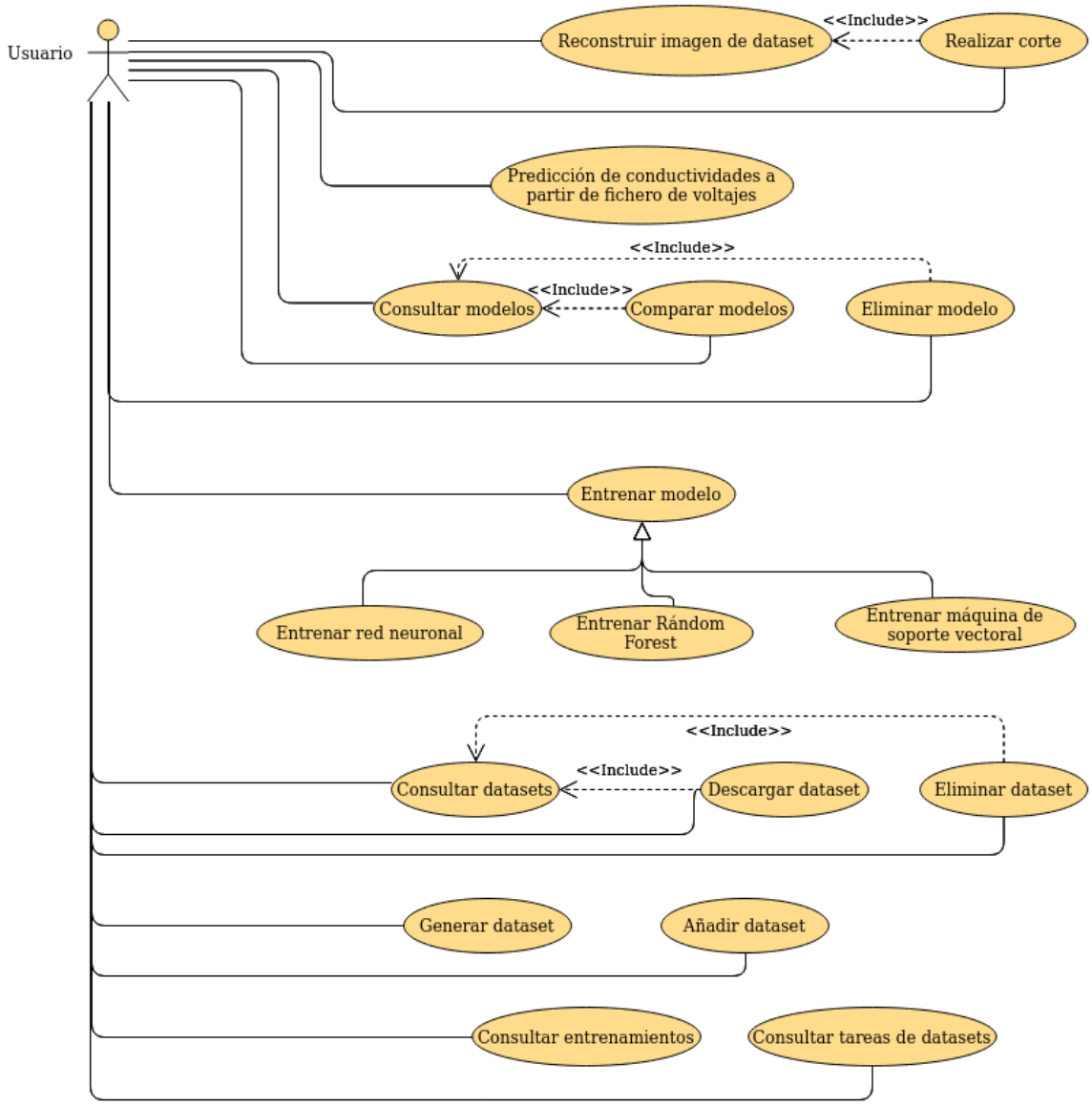

Figura 3.1: Diagrama de casos de uso.

En la figura 3.1 se muestra un diagrama de casos de uso, el cual incluye los **principales casos de uso del sistema**. Se ha incluido una relación de generalización entre los casos de uso Entrenar red neuronal, Entrenar árbol de decisión y Entrenar máquina de soporte vectorial con el caso de uso Entrenar modelo, puesto que los tres primeros son casos específicos del último. Entre algunos de los casos de uso identificados existen relaciones de inclusión. Esto sucede cuando un caso de uso incorpora el comportamiento de otro. Por ejemplo, el caso de uso Eliminar modelo incluye al caso de uso Consultar modelos ya que para eliminar un modelo es preciso realizar previamente una consulta de los modelos almacenados en el sistema, de manera que ejecutar el caso de uso Eliminar modelo implica necesariamente ejecutar el caso de uso Consultar modelos.

3.4.2. Descripción de los casos de uso

En este apartado se describirán con detalle los casos de uso más importantes del diagrama de casos de uso del apartado anterior.

Caso de uso CU1: Registrarse
Propósito: crear una cuenta para un nuevo usuario en el sistema y almacenar sus datos.
Actores: usuario
Precondiciones:
Poscondiciones: se crea una cuenta en el sistema para el usuario que se ha registrado.
Curso normal de los eventos:

1. El usuario selecciona la opción Registrarse en la ventana de inicio del sistema.

2. El sistema carga la ventana de registro, mostrando al usuario un formulario de registro.

3. El usuario cubre el formulario, introduciendo los datos requeridos: nombre de usuario, contraseña, confirmación de la contraseña, nombre real, apellidos, correo electrónico e idioma preferido. El usuario selecciona Registrarse.

4. El sistema envía un correo electrónico con un código de confirmación al usuario a la cuenta de correo electrónico especificada durante el registro. A continuación, el sistema carga la pantalla de confirmación de registro, indicando al usuario que introduzca el código de confirmación recibido.

5. El usuario introduce el código de confirmación recibido.

6. El sistema carga una nueva pantalla indicando al usuario que el registro se ha completado con éxito.

Curso alternativo 1:

4. Se produce un error de registro debido a que el usuario cubrió de forma incorrecta o dejó sin cubrir alguno de los campos del formulario de registro.

    a) El sistema indica al usuario los campos que ha cubierto de forma incorrecta (y la razón por la que la información introducida no es adecuada) y aquellos campos que ha dejado sin cubrir. Se vuelve al paso 3.

Curso alternativo 2:

6. El código de confirmación de registro introducido por el usuario no es correcto.

    a) El sistema muestra un mensaje de error indicando al usuario que ha introducido un código incorrecto. Se vuelve al paso 5.

Caso de uso CU2: Reconstruir imagen
Propósito: reconstruir la imagen de una malla a partir de un conjunto de voltajes.
Actores: usuario.
Precondiciones: debe existir algún modelo público almacenado en el sistema y algún *dataset*.
Poscondiciones: se reconstruye la imagen de la malla seleccionada por el usuario.
Curso normal de los eventos:

1. El usuario selecciona la opción Reconstrucción de Imágenes del menú principal.

2. El sistema muestra al usuario la lista de modelos disponibles.

3. El usuario selecciona el modelo con el que desea realizar la reconstrucción y pulsa en Seleccionar modelo.

4. El sistema muestra al usuario una nueva ventana con la lista de *datasets* disponibles, así como la lista de las diez primeras mallas del *dataset* por defecto.

5. El usuario filtra las mallas según los parámetros de su interés. A continuación, selecciona una malla y pulsa en Reconstruir imagen.

6. El sistema reconstruye la imagen de la malla seleccionada, empleando el modelo seleccionado y muestra al usuario la imagen real, así como la imagen reconstruida mediante el modelo seleccionado por el usuario.

Curso alternativo 1:

5. El usuario pulsa en Reconstruir imagen sin haber seleccionado ninguna malla.

   a) El sistema muestra un mensaje de error al usuario, indicándole que debe seleccionar una malla.

Curso alternativo 2:

5. No hay ninguna malla que cumpla con los requisitos establecidos por el usuario en el filtrado.

   a) El sistema indica al usuario que no hay ninguna malla que cumpla con los requisitos establecidos. Se repite el paso 5.

Caso de uso CU3: Realizar corte
Propósito: realizar el corte en el eje Y=a de una imagen, siendo a un valor seleccionado por el usuario.
Actores: usuario.
Precondiciones: el usuario debe haber reconstruido alguna imagen.
Poscondiciones: se genera una gráfica en la que se muestran los valores de conductividad de la malla a lo largo de su eje X, para el valor del eje Y y=a, establecido previamente por el usuario.

Curso normal de los eventos:

1. El sistema muestra al usuario un rango para el eje Y y un marcador que el usuario puede desplazar sobre dicho rango para seleccionar el valor del eje.

2. El usuario selecciona el valor para el eje Y sobre el cual desea realizar el corte.

3. El sistema genera y muestra al usuario una gráfica cuyo eje Y contiene valores de conductividad (medidos en S/m) y cuyo eje X se corresponde con el eje X de la malla (medido en metros). En la gráfica aparecen representados tanto los valores de conductividad de la malla real como los valores conductividad obtenidos mediante la reconstrucción realizada con el modelo.

Caso de uso CU4: Consultar modelos
Propósito: mostrar a un usuario los modelos públicos o entrenados por el propio usuario que se encuentran almacenados en el sistema.
Actores: usuario.
Precondiciones: debe existir en el sistema algún modelo público o entrenado por el usuario.
Poscondiciones: se muestran todos los modelos públicos o entrenados por el usuario que cumplan con los requisitos establecidos por éste.
Curso normal de los eventos:

1. En el apartado de modelos de la ventana principal, el usuario pulsa en la opción Acceder.

2. El sistema carga todos los modelos públicos o entrenados por el propio usuario almacenados en el sistema y muestra al usuario las opciones de filtrado.

3. El usuario selecciona las opciones de filtrado que le interesan y pulsa en Filtrar.

4. El sistema carga los modelos que cumplen con las opciones de filtrado indicadas por el usuario.

Curso alternativo 1:

4. No existen modelos en el sistema que cumplan con las opciones de filtrado indicadas por el usuario.

    a) El sistema muestra un mensaje indicando que no hay modelos que cumplan con las condiciones especificadas. Se vuelve al paso 3.

Caso de uso CU5: Comparar modelos
Propósito: realizar una comparación de diferentes modelos mediante el empleo de una serie de métricas.
Actores: usuario.
Precondiciones: deben existir en el sistema al menos dos modelos visibles para el usuario. Debe existir en el sistema al menos un *dataset*.
Poscondiciones: se genera una comparación de los modelos seleccionados por el usuario, evaluando mediante determinadas métricas los resultados de las predicciones de los modelos sobre el *dataset* indicado.

Curso normal de los eventos:

1. El usuario ejecuta el caso de uso CU4 y obtiene una lista de los modelos públicos o entrenados por él que se encuentran almacenados en el sistema.
2. El usuario debe seleccionar al menos dos modelos para compararlos y pulsa en Comparar modelos.
3. El sistema carga una ventana para definir la configuración de la comparación.
4. El usuario selecciona el *dataset* mediante el que se evaluarán los modelos, indica si desea o no postprocesar las predicciones de los modelos y elige las métricas que se utilizarán para comparar los modelos. Las métricas que puede elegir el usuario son las siguientes:
   - Error cuadrático medio (MSE).
   - Error absoluto medio (MAE).
   - Error logarítmico cuadrático medio (MLSE).
   - Porcentaje de acierto.
5. Una vez que el usuario ha seleccionado todos los parámetros de la comparación, pulsa en Comparar modelos.
6. El sistema muestra un icono de carga y un mensaje indicando al usuario que se está realizando la comparación. Una vez que el sistema ha realizado la comparación, muestra una ventana al usuario con los resultados de las métricas para cada uno de los modelos. Además, el sistema muestra la reconstrucción por cada uno de los modelos de una imagen seleccionada de forma aleatoria, además de la imagen real.

Curso alternativo 1:

2. El usuario selecciona uno o ningún modelo.
   a) El sistema muestra un mensaje de error indicando que es necesario seleccionar al menos dos modelos para poder realizar la comparación. Se vuelve al paso 2.

Curso alternativo 2:

4. El usuario no selecciona ninguna métrica.
   a) El sistema muestra un mensaje de error indicando que es necesario seleccionar al menos una métrica para poder realizar la comparación. Se vuelve al paso 4.

Caso de uso CU6: Eliminar modelo
Propósito: permite a un usuario eliminar uno de los modelos entrenados por él.
Actores: usuario.
Precondiciones: debe existir en el sistema algún modelo entrenado por el usuario.
Poscondiciones: se muestran todos los modelos públicos o entrenados por el usuario que cumplan con los requisitos establecidos por éste.
Curso normal de los eventos:

1. El usuario ejecuta el caso de uso CU4 y obtiene una lista de los modelos públicos o entrenados por él que se encuentran almacenados en el sistema. Una de las opciones que se presentan para cada uno de los modelos es la opción Eliminar modelo.

2. El usuario pulsa en la opción Eliminar modelo para el modelo que desea eliminar.

3. El sistema carga una ventana de confirmación, preguntando al usuario si realmente desea eliminar el modelo seleccionado.

4. El usuario pulsa en la opción Sí.

5. El sistema elimina el modelo de su almacenamiento.

Curso alternativo 1:

4. El usuario pulsa en la opción No. Se vuelve al paso 1.

Caso de uso CU7: Entrenar red neuronal
Propósito: generar un modelo mediante el entrenamiento de una red neuronal.
Actores: usuario.
Precondiciones: debe existir algún *dataset* en el sistema.
Poscondiciones: se genera un modelo de tipo red neuronal capaz de predecir valores de conductividad a partir de voltajes.
Curso normal de los eventos:

1. En la sección de Modelos del sistema, el usuario selecciona la opción Entrenar nuevo modelo.

2. El sistema carga la ventana de selección de tipo de modelo.

3. El usuario selecciona la opción Red neuronal.

4. El sistema carga la ventana de selección de parámetros para el modelo a entrenar.

5. El usuario introduce o selecciona los siguientes parámetros sobre el modelo:
   - *dataset* de entrenamiento.
   - Número de capas ocultas.
   - Número de neuronas en cada capa oculta.
   - Función de activación para las capas ocultas y para la capa de salida.
   - Función de error.
   - Número de épocas.
   - Indicación de si el entrenamiento se realiza por lotes y tamaño de los lotes en caso afirmativo.
   - Lista de métricas.
   - Learning rate.
   - Momentum.
   - Visibilidad.
   - Comentarios adicionales.

   A continuación, el usuario selecciona la opción Iniciar entrenamiento.

6. El sistema carga la ventana de entrenamientos. En el apartado de Entrenamientos en curso aparece el registro del nuevo entrenamiento iniciado por el usuario.

7. Cuando el entrenamiento finaliza, el sistema envía un correo notificando al usuario que el entrenamiento ha finalizado.

Curso alternativo 1:

4. El usuario introduce algún parámetro de forma incorrecta o no selecciona algún parámetro obligatorio.

   a) El sistema muestra un mensaje de error al usuario y le indica qué parámetros ha introducido de forma incorrecta (y por qué la información introducida no es correcta) o cuáles ha olvidado introducir o seleccionar. Se vuelve al paso 4.

Caso de uso CU8: Entrenar *random forest*
Propósito: generar un modelo mediante el entrenamiento de un *random forest*.
Actores: usuario.
Precondiciones: debe existir algún *dataset* en el sistema.
Poscondiciones: se genera un modelo de tipo *random forest* capaz de predecir valores de conductividad a partir de voltajes.
Curso normal de los eventos:

1. En la sección de Modelos del sistema, el usuario selecciona la opción Entrenar nuevo modelo.
2. El sistema carga la ventana de selección de tipo de modelo.
3. El usuario selecciona la opción *Random forest*.
4. El usuario introduce o selecciona los siguientes parámetros sobre el modelo:
   - *dataset* de entrenamiento.
   - Número de estimadores.
   - Profundidad máxima de las ramas.
   - Número mínimo de muestras para división.
   - Número mínimo de muestras para generar un nodo hoja.
   - Lista de métricas.
   - Visibilidad.
   - Comentarios adicionales.

   A continuación, el usuario selecciona la opción Iniciar entrenamiento.
5. El sistema carga la ventana de entrenamientos. En el apartado de Entrenamientos en curso aparece el registro del nuevo entrenamiento de tipo *random forest* iniciado por el usuario.
6. Cuando el entrenamiento finaliza, el sistema envía un correo notificando al usuario que el entrenamiento del modelo de *random forest* ha finalizado.

Curso alternativo 1:

4. El usuario introduce algún parámetro de forma incorrecta o no selecciona algún parámetro obligatorio.
   a) El sistema muestra un mensaje de error al usuario y le indica qué parámetros ha introducido de forma incorrecta (y por qué la información introducida no es correcta) o cuáles ha olvidado introducir o seleccionar. Se vuelve al paso 4.

Caso de uso CU9: Entrenar máquina de soporte vectorial
Propósito: generar un modelo mediante el entrenamiento de una máquina de soporte vectorial.
Actores: usuario.
Precondiciones: debe existir algún *dataset* en el sistema.
Poscondiciones: se genera un modelo de tipo máquina de soporte vectorial capaz de predecir valores de conductividad a partir de voltajes.

Curso normal de los eventos:

1. En la sección de Modelos del sistema, el usuario selecciona la opción Entrenar nuevo modelo.

2. El sistema carga la ventana de selección de tipo de modelo.

3. El usuario selecciona la opción Máquina de soporte vectorial.

4. El usuario introduce o selecciona los siguientes parámetros sobre el modelo:

   - *dataset* de entrenamiento.
   - Kernel.
   - Grado (el usuario sólo puede especificar este parámetro en caso de que el kernel sea de tipo polinómico).
   - Gamma.
   - Coeficiente 0.
   - Tolerancia.
   - Constante C.
   - Épsilon.
   - Lista de métricas.
   - Visibilidad.
   - Comentarios adicionales.

   A continuación, el usuario selecciona la opción Iniciar entrenamiento.

5. El sistema carga la ventana de entrenamientos. En el apartado de Entrenamientos en curso aparece el registro del nuevo entrenamiento de tipo máquina de soporte vectorial iniciado por el usuario.

6. Cuando el entrenamiento finaliza, el sistema envía un correo notificando al usuario que el entrenamiento del modelo de máquina de soporte vectorial ha finalizado.

Curso alternativo 1:

4. El usuario introduce algún parámetro de forma incorrecta o no selecciona algún parámetro obligatorio.

    a) El sistema muestra un mensaje de error al usuario y le indica qué parámetros ha introducido de forma incorrecta (y por qué la información introducida no es correcta) o cuáles ha olvidado introducir o seleccionar. Se vuelve al paso 4.

Caso de uso CU10: Eliminar modelo
Propósito: eliminar un modelo almacenado en el sistema.
Actores: usuario.
Precondiciones: debe existir en el sistema algún modelo creado por el usuario.
Poscondiciones: se elimina del sistema el modelo seleccionado por el usuario.
Curso normal de los eventos:

1. En el apartado de entrenamientos de la ventana principal, el usuario pulsa en la opción Acceder.

2. El sistema carga la ventana con la lista de modelos públicos o entrenados por el propio usuario. En los modelos entrenados por el propio usuario, el sistema muestra la opción Eliminar.

3. El usuario pulsa en la opción Eliminar en el modelo que desea eliminar del sistema.

4. El sistema carga una ventana de confirmación, preguntando al usuario si desea realmente eliminar el modelo.

5. El usuario selecciona la opción Sí.

6. El sistema elimina el modelo de la base de datos. A continuación, el sistema carga una ventana informando al usuario de que el modelo se ha eliminado con éxito.

Curso alternativo 1:

5. El usuario selecciona la opción No.

    a) El sistema carga de nuevo la ventana de modelos. Se vuelve al paso 2.

Caso de uso CU11: Consultar *datasets*
Propósito: mostrar a un usuario los *datasets* públicos o añadidos/generados por el propio usuario que se encuentran almacenados en el sistema.
Actores: usuario.
Precondiciones: debe existir en el sistema algún *dataset* público o añadido/generado por el usuario.
Poscondiciones: se muestran todos los *datasets* públicos o añadidos/generados por el usuario que cumplan con los requisitos establecidos por éste.

Curso normal de los eventos:

1. En el apartado de *datasets* de la ventana principal, el usuario pulsa en la opción Acceder.

2. El sistema carga todos los *datasets* públicos o añadidos/generados por el propio usuario almacenados en el sistema y muestra al usuario las opciones de filtrado.

3. El usuario selecciona las opciones de filtrado que le interesan y pulsa en Filtrar.

4. El sistema carga los *datasets* que cumplen con las opciones de filtrado indicadas por el usuario.

Curso alternativo 1:

4. No existen *datasets* en el sistema que cumplan con las opciones de filtrado indicadas por el usuario.

    a) El sistema muestra un mensaje indicando que no hay *datasets* que cumplan con las condiciones especificadas. Se vuelve al paso 3.

Caso de uso CU12: Descargar *dataset*.
Propósito: permitir a un usuario descargar en su equipo un *dataset* en formato CSV.
Actores: usuario.
Precondiciones: debe existir en el sistema algún *dataset* público o añadido/generado por el usuario.
Poscondiciones: el *dataset* seleccionado por el usuario es descargado en su equipo en el directorio indicado por el propio usuario.

Curso normal de los eventos:

1. Se ejecuta el caso de uso CU11. Para cada uno de los *datasets* cargados, se muestra al usuario la opción de Descargar.

2. El usuario selecciona la opción Descargar para el *dataset* que desea guardar en su equipo.

3. El usuario selecciona las opciones de filtrado que le interesan y pulsa en Filtrar.

4. El sistema carga los *datasets* que cumplen con las opciones de filtrado indicadas por el usuario.

5. El sistema prepara el *dataset* para ser descargado. Durante esta preparación del *dataset*, muestra al usuario un símbolo de carga. Una vez que el *dataset* está preparado, el sistema muestra al usuario una nueva ventana de confirmación de la descarga.

6. El usuario pulsa en la opción Descargar de la nueva ventana.

7. El sistema muestra una ventana de selección del directorio de descarga.

8. El usuario selecciona el directorio en el que desea descargar el *dataset*.

9. El sistema guarda el *dataset* en formato CSV en el directorio indicado por el usuario

Curso alternativo 1:

6. El usuario decide finalmente no descargar el *dataset* y regresa a la página anterior. Se repite el paso 1.

Caso de uso CU13: Eliminar *dataset*
Propósito: eliminar un *dataset* almacenado en el sistema.
Actores: usuario.
Precondiciones: debe existir en el sistema algún *dataset* añadido/generado por el usuario.
Poscondiciones: se elimina del sistema el *dataset* seleccionado por el usuario.
Curso normal de los eventos:

1. El usuario ejecuta el caso de uso CU11 y obtiene una lista de los *datasets* públicos o añadidos/generados por él que se encuentran almacenados en el sistema. Una de las opciones que se presentan para cada uno de los *datasets* es la opción Eliminar.

2. El usuario pulsa en la opción Eliminar en el *dataset* que desea eliminar del sistema.

3. El sistema carga una ventana de confirmación, preguntando al usuario si desea realmente eliminar el *dataset*.

4. El usuario selecciona la opción Sí.

5. El sistema elimina el *dataset* de la base de datos . A continuación, el sistema carga una ventana informando al usuario de que el *dataset* se ha eliminado con éxito.

Caso de uso CU14: Añadir *dataset*
Propósito: subir un *dataset* al sistema desde el equipo del usuario
Actores: usuario.
Precondiciones:
Poscondiciones: el *dataset* seleccionado por el usuario es subido al sistema.
Curso normal de los eventos:

1. En la ventana de *datasets*, el usuario pulsa en la opción Subir *dataset*.

2. El sistema carga la ventana de subida de un *dataset*.

3. El usuario introduce el tamaño mínimo y máximo del radio de los artefactos de las mallas que integran el *dataset*, la semilla con la que el *dataset* ha sido generado e indica si desea que el *dataset* sea visible. Además, el usuario selecciona el fichero de su equipo que contiene el *dataset* y pulsa en la opción Subir *dataset*.

4. El sistema valida el fichero subido por el usuario y carga la ventana de tareas de *datasets*. El *dataset* subido por el usuario aparece en la lista de *Datasets* en curso de ser subidos o generados.

Curso alternativo 1:

4. El sistema determina que el fichero subido no tiene un formato o una estructura válida y muestra al usuario un error informándolo de la situación. Se repite el paso 3.

Caso de uso CU16: Generar *dataset*
Propósito: generar un nuevo *dataset* en el sistema con las características especificadas por el usuario.
Actores: usuario.
Precondiciones:
Poscondiciones: se genera un nuevo *dataset*, el cual es almacenado en el sistema.
Curso normal de los eventos:

1. En la ventana de *datasets*, el usuario pulsa en la opción Generar *dataset*.

2. El sistema carga la ventana de generación de un *dataset*.

3. El usuario introduce el número de mallas con uno, dos y tres artefactos, el tamaño mínimo y máximo del radio de los artefactos de las mallas que integran el *dataset*, la semilla con la que el *dataset* ha sido generado e indica si desea que el *dataset* sea visible. El usuario pulsa en la opción Generar *dataset*.

4. El sistema carga la ventana de tareas de *datasets*. El *dataset* cuya generación ha iniciado el usuario aparece en la lista de *Datasets* en curso de ser subidos o generados.

Capítulo 4

Tecnologías y herramientas

En este apartado se incluye la descripción de las principales tecnologías y herramientas utilizadas para realizar el proyecto. Cabe señalar que tanto las tecnologías como las herramientas que se presentarán a continuación son **software libre**, por lo que no han supuesto coste alguno para el proyecto.

4.1. Tecnologías

4.1.1. Python

El lenguaje de propósito general utilizado para implementar el *backend* del sistema fue Python. Aunque Python no suele ser un lenguaje habitual para implementar servicios web, a diferencia de otros lenguajes como Java, se consideró el lenguaje apropiado para este proyecto. Python es el lenguaje más empleado en el ámbito de la Inteligencia Artificial, por lo que dispone de numerosos módulos relacionados con este campo y, por tanto, de gran utilidad para la consecución de los objetivos de este proyecto. Considerando que el propósito de este proyecto es el desarrollo de una aplicación capaz de simular tomografías de impedancia eléctrica mediante técnicas de *Machine Learning*, se optó por emplear Python como lenguaje de implementación del *backend*. En particular, en este proyecto fueron fundamentales los módulos numpy y sklearn de Python, para el manejo de *datasets* y el entrenamiento de modelos de *Machine Learning*, respectivamente.

4.1.2. Django REST

Django REST [8] es un *framework* de Python para el desarrollo de APIs REST (las características de una API REST se explicarán en el apartado 5.1 del trabajo). Django REST enmascara la complejidad y los riesgos de seguridad asociados desarrollo de APIs REST. Por esta razón, se logra una notable aceleración en la fase de codificación. Cabe destacar que mediante el empleo de Django REST, no

fue necesario escribir consultas de SQL puro para realizar accesos a la base de datos.

4.1.3. React

Para desarrollar el *frontend* del sistema, se empleó React, [9] un *framework* de JavaScript para el desarrollo de interfaces gráficas. React se basa en el empleo de componentes, los cuales son clases de JavaScript. Cada una de las vistas de la aplicación puede estar conformada por uno o varios componentes. Por otra parte, la utilización de este *framework* facilitó la implementación de comportamientos dinámicos en las vistas, como la aparición de símbolos de carga al pulsar un botón o la creación de ciertos nodos en la estructura DOM del documento HTML como respuesta a una interacción del usuario.

Por otra parte, para realizar las llamadas a los servicios ofrecidos por el *backend* desde el *frontend* implementado en React, se utilizó la biblioteca Axios [10], un cliente HTTP para JavaScript.

4.1.4. Keras

La biblioteca Keras [11] de Python se utilizó para incorporar la funcionalidad de entrenamiento de redes neuronales, así como la funcionalidad de realizar predicciones utilizando dichos modelos. Cabe señalar que como *backend* de Keras se empleó, a su vez, la biblioteca TensorFlow [12].

4.1.5. EIDORS

La biblioteca EIDORS [13] se utilizó para simular la realización de tomografías de impedancia eléctrica, haciendo uso de los algoritmos que incluye la biblioteca para resolver tanto el *forward problem* como el inverse problem. Aunque EIDORS es una biblioteca de MATLAB, muchas de sus funciones pueden utilizarse a través de Octave. Para este trabajo, se hizo uso del módulo *oct2py* de Python [14], que permite ejecutar funciones de Octave desde Python. Por otra parte, para la generación de las representaciones gráficas de las mallas se utilizó la biblioteca NetGen [15].

4.1.6. Celery

Celery [16] es una cola de tareas implementada en Python. El entrenamiento de modelos y la generación de *datasets* son tareas costosas con un tiempo de ejecución muy elevado, por lo que conviene ejecutarlas de forma asíncrona. Celery se ha utilizado para gestionar estas tareas de manera eficiente.

4.1.7. C++

Para construir los *datasets*, se hizo uso de un algoritmo de generación de mallas, diseñado por el grupo de investigación COGRADE del CiTIUS. Dicho algoritmo se utiliza a través de un programa escrito en C++, el cual se ha integrado con la aplicación de Django REST (véase apartado 5.1.3). C++ es un lenguaje de programación que extiende al lenguaje C, incorporando el paradigma de orientación a objetos.

4.1.8. HTML5

HTML5 es la quinta versión de HTML (HyperText Markup Language), el lenguaje de marcas estándar para la elaboración de páginas web. Es un lenguaje de marcas porque que utiliza etiquetas (marcas) para definir cierta información sobre el contenido y estructura de las páginas del sitio web que se muestran al usuario. De las nuevas características incorporadas en la quinta versión, en este proyecto se ha hecho uso especialmente de aquéllas relacionadas con los formularios, como los atributos required, min o max, evitando así tener que implementar ciertas funciones en JavaScript, logrando acelerar el proceso de codificación.

4.1.9. CSS3

CSS3 es la última versión de CSS (Cascading Style Sheets). CSS es un lenguaje de diseño que permite definir cuál será la apariencia y cómo se mostrará gráficamente un documento HTML o cualquier tipo de documento XML. En este proyecto, para definir la apariencia gráfica de las vistas, se utilizó principalmente la biblioteca Bootstrap. No obstante, se empleó CSS3 directamente para definir ciertas características de la visualización que la biblioteca Bootstrap no permite definir.

4.1.10. PostgreSQL

PostgresSQL es un SGBD (Sistema de Gestión de Bases de Datos) relacional y de código abierto.

4.1.11. Bootstrap

Tal y como se mencionó anteriormente, la apariencia de las vistas se definió principalmente mediante el *framework* Bootstrap [17]. Una de las características fundamentales de este *framework* es que la colocación de los elementos en pantalla se realiza a través de la división de ésta en filas de hasta 12 columnas. De esta manera, permite construir con facilidad vistas responsivas que se adapten al tamaño de la pantalla del dispositivo que emplee el usuario.

4.2. Herramientas

4.2.1. Visual Studio Code

La única herramienta empleada para la codificación fue Visual Studio Code [18], que es un editor de código fuente, con ciertas características de un IDE. Dispone de numerosos paquetes que permiten extender sus funcionalidades. Para el desarrollo del código de la aplicación, se emplearon todas las extensiones de resaltado de sintaxis asociadas a los diferentes lenguajes empleados: Python, HTML, CSS y JavaScript.

4.2.2. Project Libre

Project Libre es un software para la gestión y administración de proyectos [19]. En este proyecto se utilizó para realizar el diagrama de Gantt de la planificación temporal, así como para calcular los costes del proyecto.

4.2.3. StarUML

Se utilizó la herramienta StarUML [20] para la elaboración de todos los diagramas del estándar UML incluidos en la memoria: diagrama de paquetes, diagrama de clases, diagrama del modelo entidad-relación, diagramas de actividad y diagramas de secuencia.

4.2.4. Diagrams.net

Se utilizó la herramienta de dibujo online Diagrams.net para la elaboración del diagrama de arquitectura [21].

4.2.5. NinjaMock

Se utilizó la versión gratuita de la herramienta online NinjaMock para diseñar los prototipos de la interfaz gráfica de la aplicación [22].

4.2.6. LaTeX

Se utilizó LaTeX para redactar este documento, empleando como editor la herramienta online Overleaf [23].

Capítulo 5

Diseño e implementación

5.1. Selección de la arquitectura

Tal y como se ha mencionado previamente, se desea desarrollar una aplicación web que cumpla con los requisitos definidos en el capítulo 3. No obstante, existen aproximaciones muy diferentes para el diseño de una aplicación web. En este apartado se compararán dos de las principales aproximaciones: el uso de una arquitectura web tradicional (monolítica) y el empleo de servicios web. Para ambas alternativas, se describirán las ventajas y las desventajas que presentan. Finalmente, se seleccionará la aproximación que se estime más conveniente para el proyecto y se justificará la decisión tomada.

5.1.1. Arquitectura web monolítica

La arquitectura web tradicional o monolítica se caracteriza por la integración de la lógica de negocio, la capa de acceso a datos y las vistas en un único programa. Estos tres componentes se encuentran interconectados y son interdependientes. En la tabla 5.1 se muestran las ventajas y desventajas de esta arquitectura.

Ventajas	Desventajas
El diseño y el desarrollo son sencillos, de forma que se ahorra tiempo tanto en la fase de diseño como en la de codificación. Por otra parte, no es necesario realizar los cambios y actualizaciones por separado.	La principal desventaja de esta arquitectura es su difícil mantenimiento. A medida que aumenta el tiempo de vida de la aplicación web, la complejidad del código fuente crece considerablemente, dificultando la realización de cambios y actualizaciones.
Al no ser preciso realizar llamadas a máquinas remotas, se obtiene un mejor rendimiento, siempre y cuando la aplicación se haya diseñado e implementado de forma adecuada.	La aplicación sólo puede ser utilizada por personas que hagan uso de la interfaz gráfica, de manera que no existe la posibilidad de que dicha aplicación sea utilizada también por otras aplicaciones.
Garantizar la seguridad de la aplicación es más simple.	Dificultad para adoptar e incorporar nuevas tecnologías.

Cuadro 5.1: Ventajas y desventajas del empleo de una arquitectura web monolítica.

5.1.2. Servicios web

Según WC3, un servicio web es un "sistema de software diseñado para soportar interacciones máquina-a-máquina de forma interoperable" [26]. Los servicios web se caracterizan por el uso de protocolos estandarizados para realizar las comunicaciones. En una arquitectura de este tipo, un proveedor de servicios ofrece un conjunto de servicios, los cuales son consumidos por diferentes clientes (consumidores). En la tabla 5.2 se muestran las ventajas y desventajas del empleo de servicios web.

Ventajas	Desventajas
Los servicios pueden ser consumidos por aplicaciones, sin la intervención de un usuario humano.	El diseño y la codificación son más complejos que en el caso de una arquitectura monolítica.
Los servicios pueden ser consumidos por terceros autorizados, de manera que su ámbito de uso no se limita únicamente a una aplicación propia.	Garantizar la seguridad presenta más dificultades que en el caso de una arquitectura monolítica.
Al emplear componentes independientes, el mantenimiento es más sencillo. Los servicios pueden ser testeados con facilidad.	

Cuadro 5.2: Ventajas y desventajas del empleo de servicios web.

5.1.3. Arquitectura seleccionada

Considerando las ventajas y desventajas expuestas para cada una de las arquitecturas anteriores, se decidió que la utilización de **servicios web** era la opción preferible para el proyecto. Dado que muchas de las funcionalidades descritas en el apartado 3.1 tienen aplicación industrial y científica, resulta interesante ofrecer un conjunto de servicios que incorporen estas funcionalidades, para que dichos servicios puedan ser consumidos por terceros. En el caso de que se emplease una arquitectura monolítica, su utilidad sería menor, puesto que las funcionalidades del sistema sólo podrían ser explotadas por usuarios a través de la interacción con la interfaz gráfica.

Los servicios web serán ofrecidos mediante una **API REST.** La arquitectura REST (*Representational State Transfer*) permite construir APIs en la web, empleando el protocolo HTTP. Presenta tres características fundamentales:

- No tiene estado, puesto que se utiliza el protocolo HTTP, el cual tampoco tiene estado.
- Soporta los verbos HTTP comunes (GET, POST, DELETE, PUT, etc).
- La información se devuelve en formato JSON o XML.

En una API REST, los servicios son ofrecidos a través de URLs, las cuales reciben el nombre de *endpoints*. Cada *endpoint* acepta una serie de verbos HTTP.

La aplicación web que se desarrollará constará de dos componentes principales y desacoplados: el *backend* y el *frontend*. El *backend* es el componente que incorpora la API REST, mientras que el *frontend* es el componente que incorpora la interfaz gráfica. El *frontend* realizará llamadas al *backend* y utilizará los servicios ofrecidos por éste, actualizando la vista de la aplicación. El diseño y funcionamiento del sistema se detallará en los siguientes apartados.

5.2. Diseño del *backend*

5.2.1. Diagrama de arquitectura

En primer lugar, se muestra un diagrama de alto nivel de la arquitectura del *backend*.

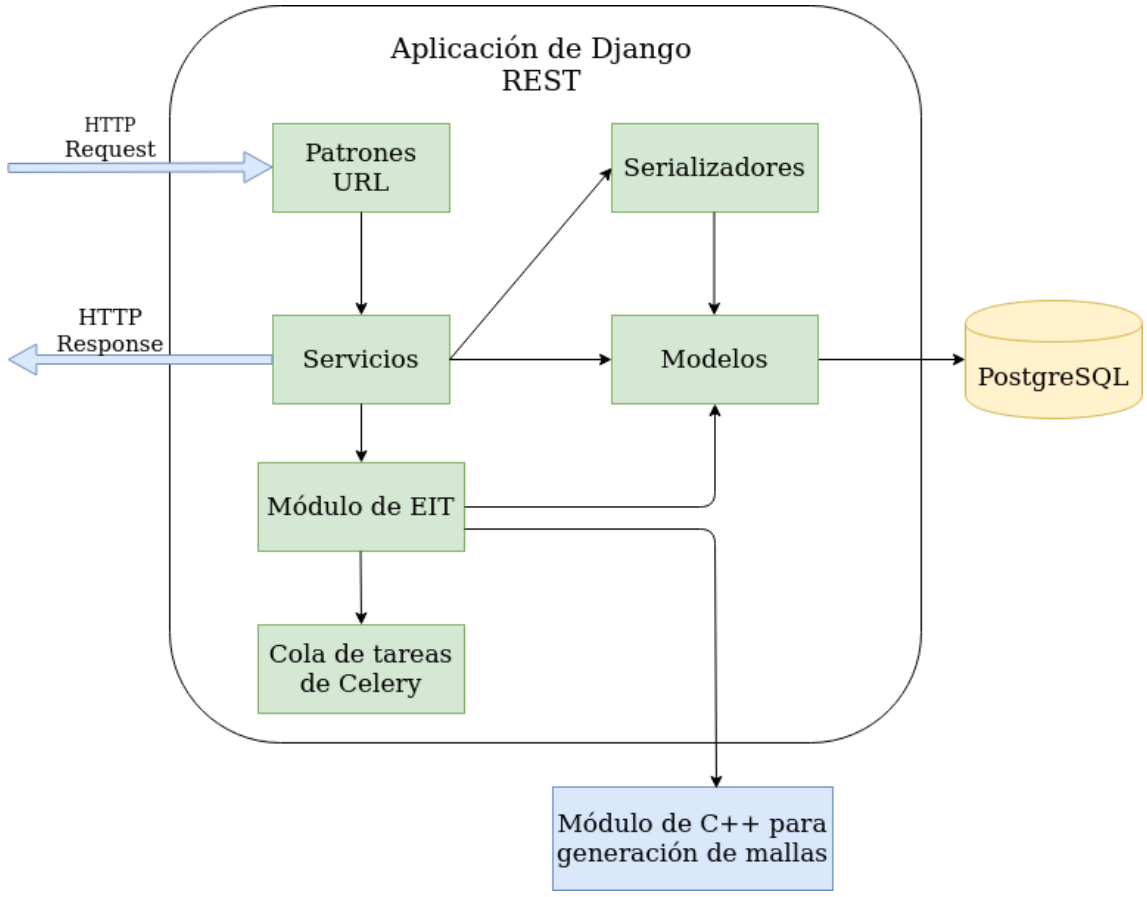

Figura 5.1: Diagrama de arquitectura.

La arquitectura de la aplicación se corresponde mayoritariamente con la arquitectura típica de una aplicación de Django REST, con la inclusión de dos

módulos adicionales: el módulo de EIT y la cola de tareas de Celery. A continuación, se describirán brevemente los componentes del *backend* presentes en la figura 5.1:

- **Patrones URL**: cuando la aplicación recibe una petición HTTP, el módulo de patrones URL examina los patrones definidos y, cuando encuentra un primer patrón que coincida con la URL solicitada, llama al servicio asociado a dicha URL. Cada uno de los patrones definidos es un *endpoint* de la API REST.

- **Servicios**: en el módulo de servicios se encuentran definidos los servicios ofrecidos por la API REST. Cada servicio devuelve una respuesta HTTP con los recursos solicitados por la petición HTTP recibida o, en caso de error, con el código de error correspondiente.

- **Módulo de EIT**: se trata de un módulo tradicional de Python, el cual contiene clases con los métodos que implementan las funcionalidades del sistema tomográfico: reconstrucción de imágenes, entrenamiento de modelos, generación de *datasets*, etc.

- **Modelos Django**: en este módulo se definen las clases que se mapearán en la base de datos. Los modelos permiten encapsular la información que maneja la aplicación. En esta aplicación, se han definido clases para los *datasets*, las mallas o los modelos de *Machine Learning*, entre otros. En este punto, cabe señalar que, a pesar de que se utiliza el mismo término para hacer referencia a ellos, no se deben confundir los modelos de Django con los modelos de *Machine Learning*. Para almacenar modelos de *Machine Learning* en la BD, se han definido los correspondientes modelos de Django, los cuales no son más que clases de Python con ciertos atributos especiales para realizar satisfactoriamente el mapeo en la BD.

- **Serializadores**: dado que los recursos devueltos por los servicios se envían en formato JSON, es necesario disponer de clases que serialicen (conviertan a formato JSON) los modelos de Django. En este módulo se definen las clases encargadas de realizar la serialización.

- **PostgreSQL**: los modelos se almacenan de forma persistente en una BD de datos. Tal y como se ha explicado en el capítulo 4, se ha utilizado PostgreSQL como SGBD.

- **Cola de tareas de Celery**: algunos de los métodos del módulo de EIT, generan y envían tareas, las cuales son enviadas a una cola de tareas de Celery.

- **Módulo de C++ para la generación de mallas**: se trata de un módulo externo al que el sistema llama cuando un usuario inicia la generación de un nuevo *dataset*.

5.2.2. Diagrama de clases del módulo de acceso a BD

A continuación, se muestra el diagrama de clases del módulo de acceso a BD.

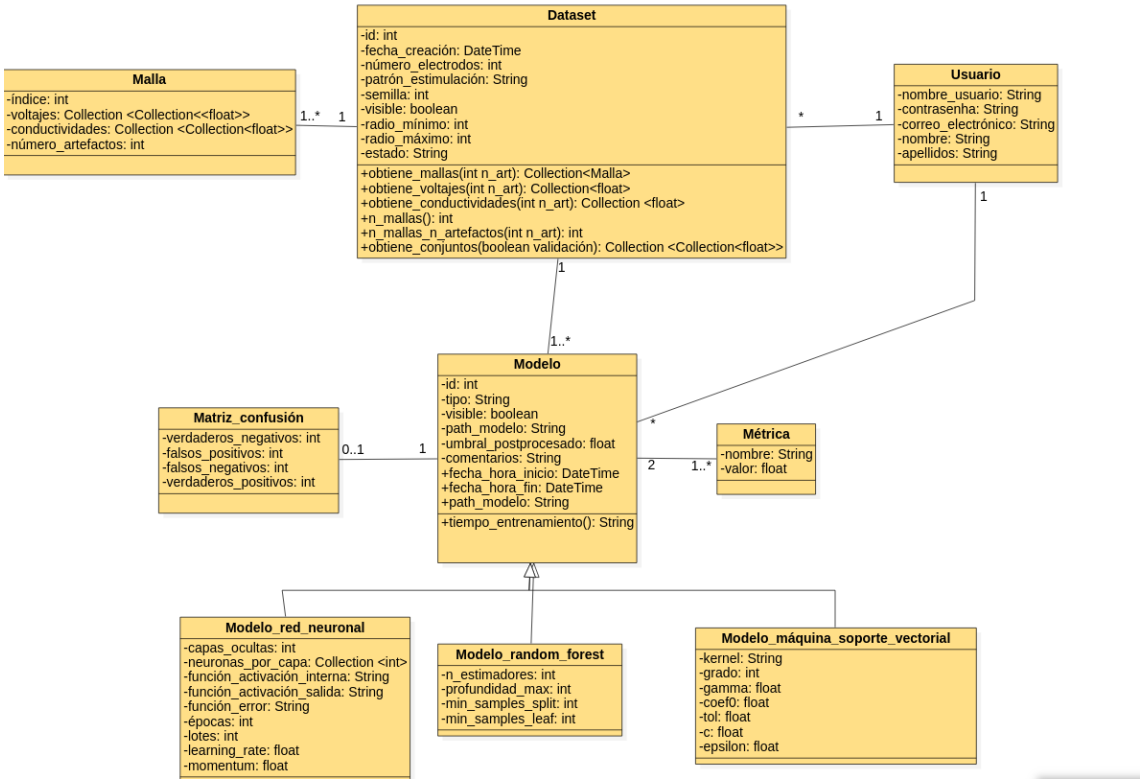

Figura 5.2: Diagrama de clases del módulo de acceso a BD.

Las clases presentes en el diagrama 5.2 son mapeadas por Django en tablas de la base de datos. Como se puede observar en el diagrama, los *datasets* constan de un conjunto de mallas. Cada *dataset* es creado por un usuario y cada usuario puede generar un número ilimitado de *datasets*. Por otro lado, un mismo *dataset* puede ser utilizado para entrenar diferentes modelos de *Machine Learning*. Cada modelo de *Machine Learning* tiene asociadas varias métricas, con las que se evalúa la precisión del modelo y, adicionalmente, algunos de los modelos también tienen asociada una matriz de confusión (véase el capítulo 6). Dado que los modelos pueden ser de tres tipos (redes neuronales, *random forest* y máquinas de soporte vectorial), existe una relación de herencia entre la clase *Modelo* y las clases *Modelo_red_neuronal*, *Modelo_random_forest* y *Modelo_máquina_soporte_vectorial*, de

manera que estas tres últimas clases heredan todos los atributos y métodos de la clase *Modelo*. Todas estas clases se encuentran definidas en el fichero *models.py*, siguiendo la convención de Django.

5.2.3. Diagrama de clases del módulo de EIT

En la siguiente figura, se muestra el diagrama de clases del módulo de EIT.

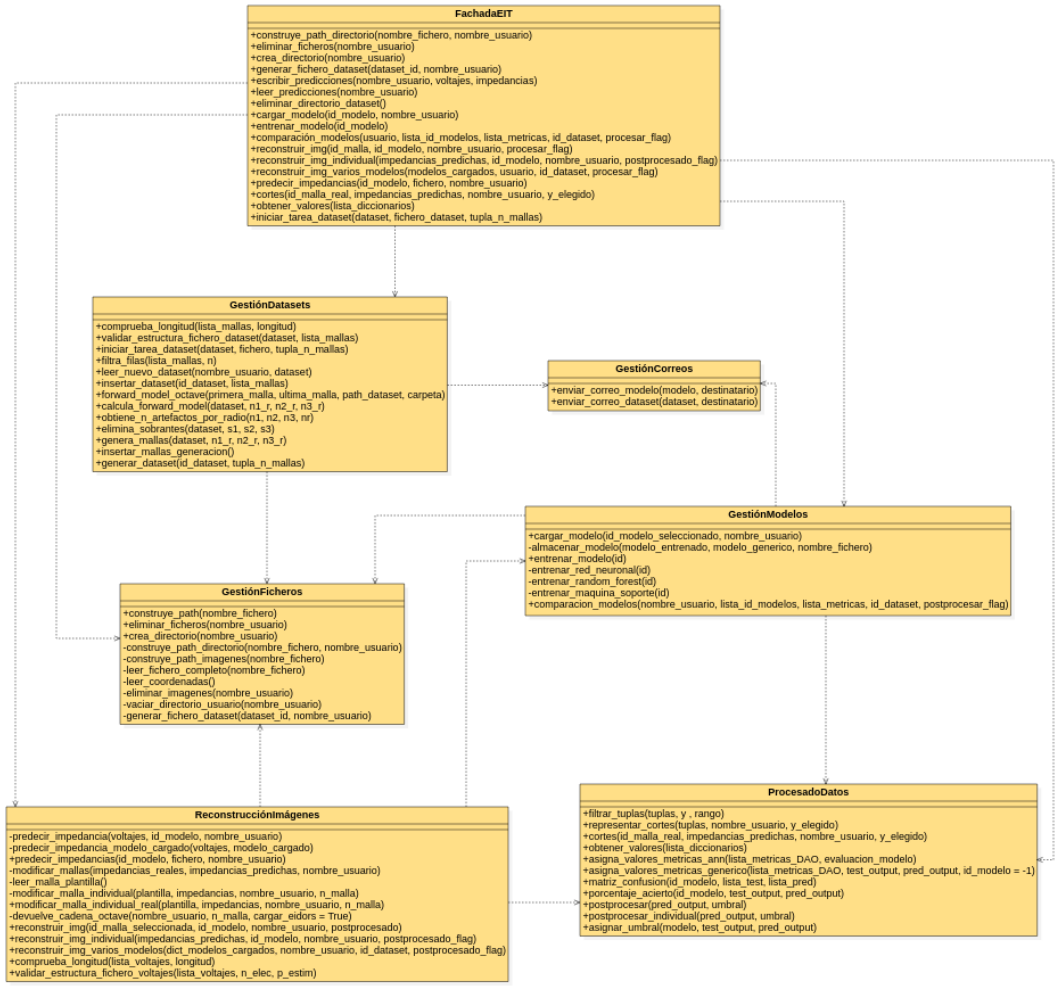

Figura 5.3: Diagrama de clases del módulo de EIT.

Todas las clases del módulo de EIT son clases estáticas, puesto que todos sus métodos han sido definidos como estáticos. Esto implica que los métodos de dichas clases pueden ser empleados sin necesidad de instanciar las clases. Las clases de este módulo son las que implementan las principales funcionalidades del sistema. Las clases del módulo son la siguientes:

- **GestiónModelos**. Incluye los métodos necesarios para entrenar, cargar y comparar los diferentes tipos de modelos del sistema.
- **GestiónDatasets**. Incluye los métodos necesarios para generar o subir *datasets* al sistema.
- **ReconstrucciónImágenes**. Incluye los métodos necesarios para realizar las predicciones de impedancias de mallas, así como las reconstrucciones de las imágenes asociadas a las mallas.
- **ProcesadoDatos**. Incluye los métodos necesarios para realizar los cortes en las mallas, calcular métricas para la comparación de modelos, así como para realizar el postprocesado de los datos predichos por un modelo (véase capítulo 6).
- **GestiónFicheros**. Algunas de las clases del módulo necesitan generar o hacer uso de ficheros. Esta clase ofrece todos los métodos necesarios para gestionar los diferentes ficheros que necesiten las demás clases.
- **GestiónCorreos**. Incluye los métodos para enviar correos electrónicos a los usuarios que hayan enviado tareas a la cola de tareas, con el propósito de informales de la finalización de dichas tareas.
- **FachadaEIT**. Siguiendo el patrón fachada (véase apartado 5.2.4), esta clase se emplea con el propósito de que no existan múltiples dependencias entre el módulo de servicios y las clases del módulo de EIT. En particular, el módulo de servicios tendrá una única dependencia con la clase *FachadaEIT*.

5.2.4. Patrones de diseño empleados en el *backend*

Patrón Fachada

El **patrón Fachada** es un patrón de tipo estructural. Los patrones de tipo estructural son aquéllos que permiten conformar estructuras de una cierta complejidad mediante la combinación de clases. El patrón fachada permite proporcionar una interfaz simple para un subsistema complejo. De esta forma, se logra descomponer un sistema en un conjunto de subsistemas, los cuales se comunican a través de sus respectivas fachadas. Así, se reducen considerablemente las dependencias entre las clases de la aplicación. La fachada conoce las clases de las que consta el subsistema, de forma que delega las peticiones que recibe en dichas clases. En el caso particular de esta aplicación, la clase FachadaIA desempeña el papel de fachada del módulo de EIT y tiene dependencias con las clases GestiónFicheros, GestiónModelos, GestiónDatasets, ProcesadoDatos y ReconstrucciónImágenes. Estas cinco clases son las que implementan la lógica de negocio.

Patrones DAO y DTA

El patrón DAO (Data Access Object) permite separar la lógica de negocio de la lógica de acceso a los datos. De esta manera, se emplea una interfaz que recibe peticiones de clientes que desean acceder a la fuente de datos, sin necesidad de que los clientes conozcan los detalles de implementación de la fuente de datos. Por otra parte, mediante el patrón DTA (Data Transfer Object) la información obtenida de la fuente de datos se empaqueta en un objeto Python.

El framework Django incorpora ambos patrones. La incorporación del patrón DAO en dicho framework es evidente, puesto que sin necesidad de modificar el código que implementa la lógica de negocio, puede sustituirse la base de datos utilizada por la aplicación, cambiando el valor de la variable ENGINE en el fichero settings.py.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
                  .
                  .
                  .
    }
}
```

La utilización del patrón DTA en Django también es clara, puesto que en el fichero *models.py* se definen las clases que se utilizarán para empaquetar los datos obtenidos al realizar las consultas en la BD.

5.2.5. Diagramas de secuencia

En este apartado se mostrarán los diagramas de secuencia de los casos de uso más relevantes del sistema. Los diagramas de secuencia permiten realizar el modelado de comportamiento del sistema, representando interacciones entre objetos y reflejando el orden en el que estas interacciones se producen.

Reconstrucción de la imagen de la malla de un *dataset*

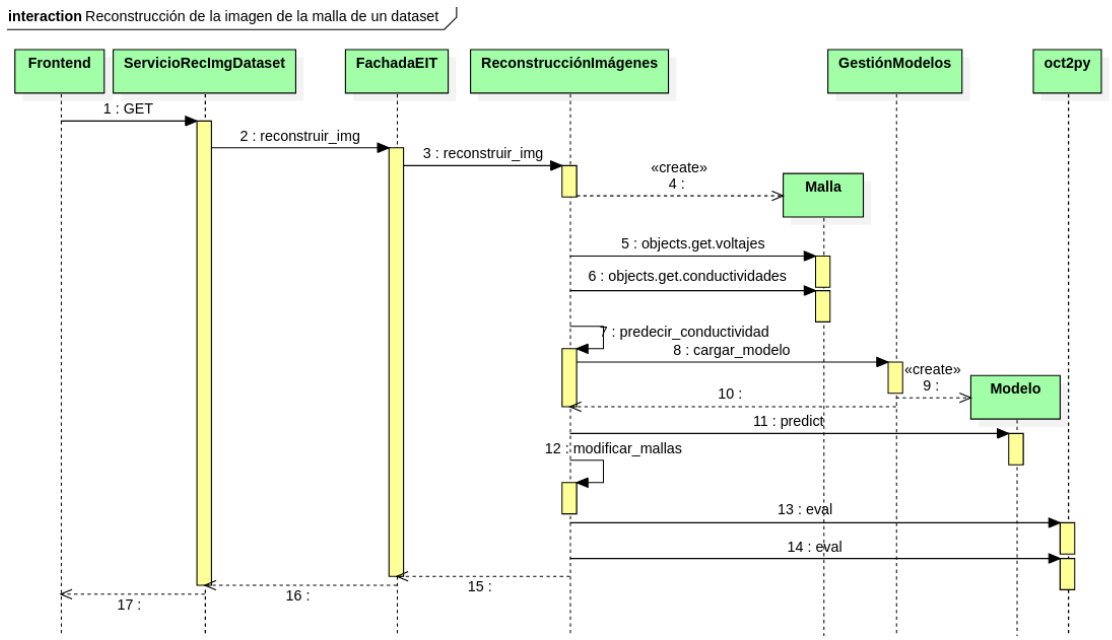

Figura 5.4: Diagrama de secuencia: reconstrucción de la de la imagen de la malla de un *dataset*.

Los pasos reflejados en la 5.4 son los siguientes:

1. Desde el *frontend* se envía una petición HTTP al servicio de reconstrucción de imágenes de *datasets*, utilizando GET como verbo de la petición. En los parámetros de la petición, se indica el *dataset*, el índice de la malla cuya imagen se desea reconstruir y el modelo que se quiere utilizar para la reconstrucción.

2. Desde el servicio, se llama al método *reconstruir_img* de la fachada del módulo de EIT.

3. El método de la fachada llama al método *reconstruir_img* de la clase *ReconstrucciónImágenes*.

4. Se realiza una consulta en la base de datos, para obtener la información de la malla cuya imagen se reconstruirá. Se instancia un objeto de la clase *Malla*.

5, 6. Se obtienen los voltajes y las impedancias de la malla

7, 8, 9, 10. Se llama al método *predecir_impedancias*, el cual, a su vez, llama al método *cargar_modelo* de la clases *GestiónModelos*. Se instancia un objeto de la clase *Modelo*.

11. Se suministran los voltajes de la malla como entrada del modelo cargado y se realizan las predicciones de las impedancias.

12. A partir del fichero base que contiene la información para representar una malla vacía, se generan dos ficheros en el formato adecuado para EIDORS. Uno de ellos contiene las impedancias reales de la malla, mientras que el otro contiene las impedancias predichas.

13, 14. Para generar la imagen real y la imagen reconstruida con el modelo, se ejecuta el correspondiente código de Octave (a través del módulo *oct2py* de Python), el cual construye ambas imágenes a partir de los dos ficheros generados en el paso 12.

15, 16, 17. Se devuelve al *frontend* las urls de las imágenes generadas, así como la lista de impedancias predichas para la imagen reconstruida.

Realización de predicciones de un conjunto de voltajes

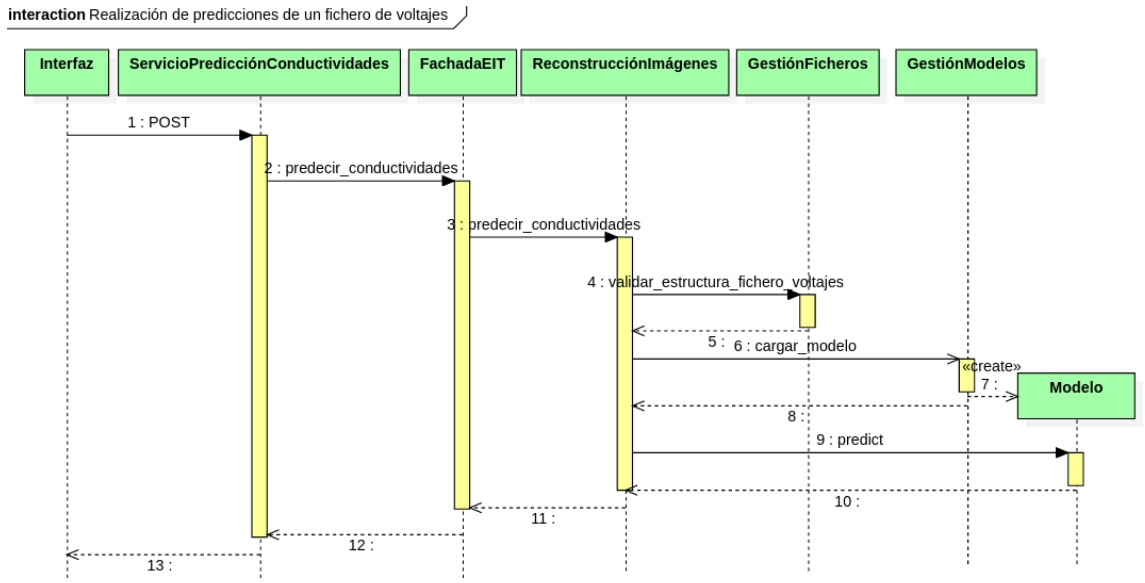

Figura 5.5: Diagrama de secuencia: realización de predicciones de un conjunto de voltajes.

Los pasos reflejados en la 5.5 son los siguientes:

1. Desde el *frontend* se envía una petición HTTP al servicio de predicción de impedancias, utilizando POST como verbo de la petición. En el cuerpo de la petición, se envía un fichero con un conjunto de voltajes, así como el identificador del modelo con el que se desea realizar la predicción.

2. Desde el servicio, se llama al método *predecir_impedancias* de la fachada del módulo de EIT.

3. El método de la fachada llama al método *predecir_impedancias* de la clase *ReconstrucciónImágenes*.

4, 5. Se llama al método *validar_estructura_fichero_voltajes* de la clase *GestiónFicheros* para determinar si el fichero recibido por el *backend* tiene el formato adecuado.

6, 7, 8. Una vez se ha determinado que el fichero de voltajes tiene el formato correcto, se carga el modelo correspondiente, mediante el método *cargar_modelo*, de la clase *GestióModelos*.

9. Mediante el modelo cargado y los voltajes suministrados, se realizan las predicciones de las impedancias.

10, 11, 12, 13. Se devuelven al *frontend* las impedancias predichas por el modelo.

Entrenamiento de una red neuronal

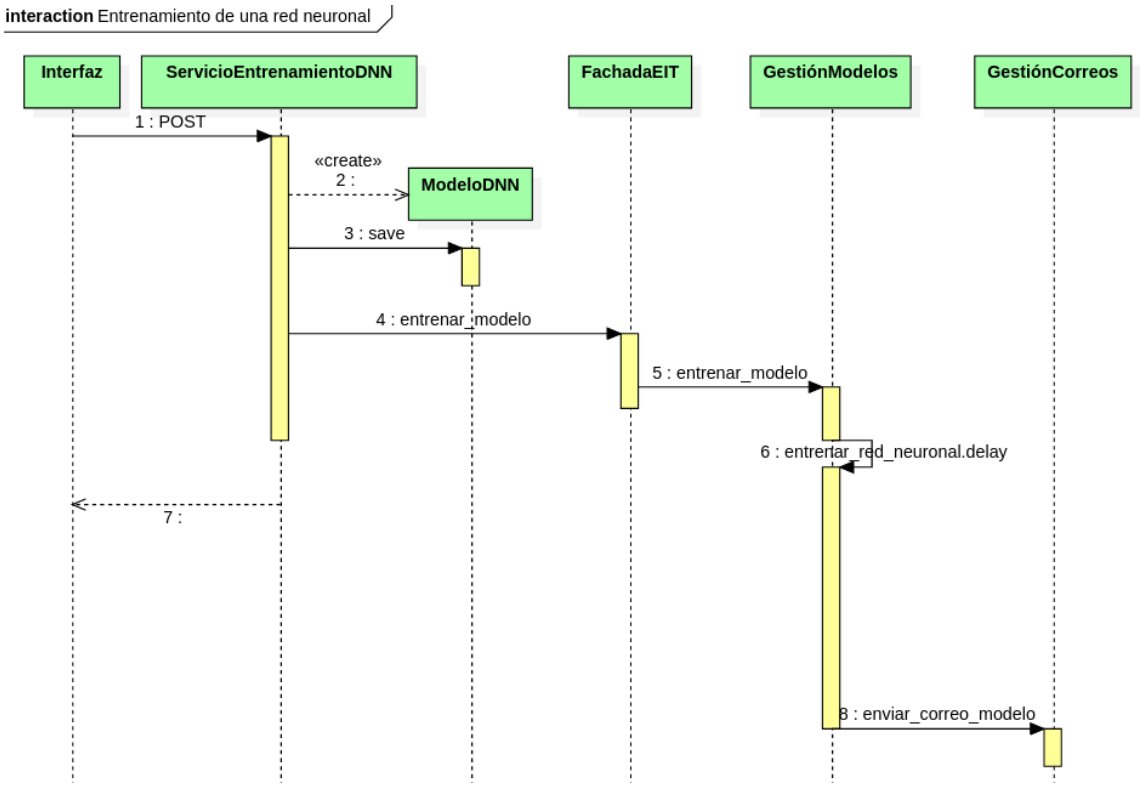

Figura 5.6: Diagrama de secuencia: entrenamiento de una red neuronal.

Los pasos reflejados en la 5.6 son los siguientes:

1. Desde el *frontend* se envía una petición HTTP al servicio de entrenamiento de redes neuronales, utilizando POST como verbo de la petición. En el cuerpo de la petición, se envían los atributos que se desea que tenga el modelo a entrenar.

2. Desde el servicio, se instancia un objeto de la clase *Modelo_DNN* asignándole los atributos indicados en la petición HTTP.

3. Mediante el método *save*, se guarda la información del nuevo modelo en la base de datos.

4. Desde el servicio, se llama al método *entrenar_modelo* de la fachada del módulo de EIT.

5. Se llama al método *entrenar_modelo* de la clase *GestiónModelos*.

6. Dado que el tipo de modelo es una red neuronal, se llama al método *entrenar_red_neuronal*, para iniciar el entrenamiento de la red neuronal. Mediante la palabra clave *delay*, el método es tratado como una tarea, la cual es enviada a la cola de tareas de Celery.

7. Se envía una respuesta al *frontend* con el código 200 de HTTP, indicando que el entrenamiento se ha enviado con éxito a la cola de tareas.

8. Una vez que ha finalizado el entrenamiento, se llama al método *enviar_correo_modelo*, mediante el cual se envía un correo electrónico a la dirección asociada al usuario que realizó el entrenamiento, informándolo de la finalización de éste.

Comparación de varios modelos

En este ejemplo, se supondrá que la comparación entre modelos se realiza utilizando como métrica el porcentaje de acierto. Además, se supondrá también que se ha seleccionado la opción de postprocesar los resultados.

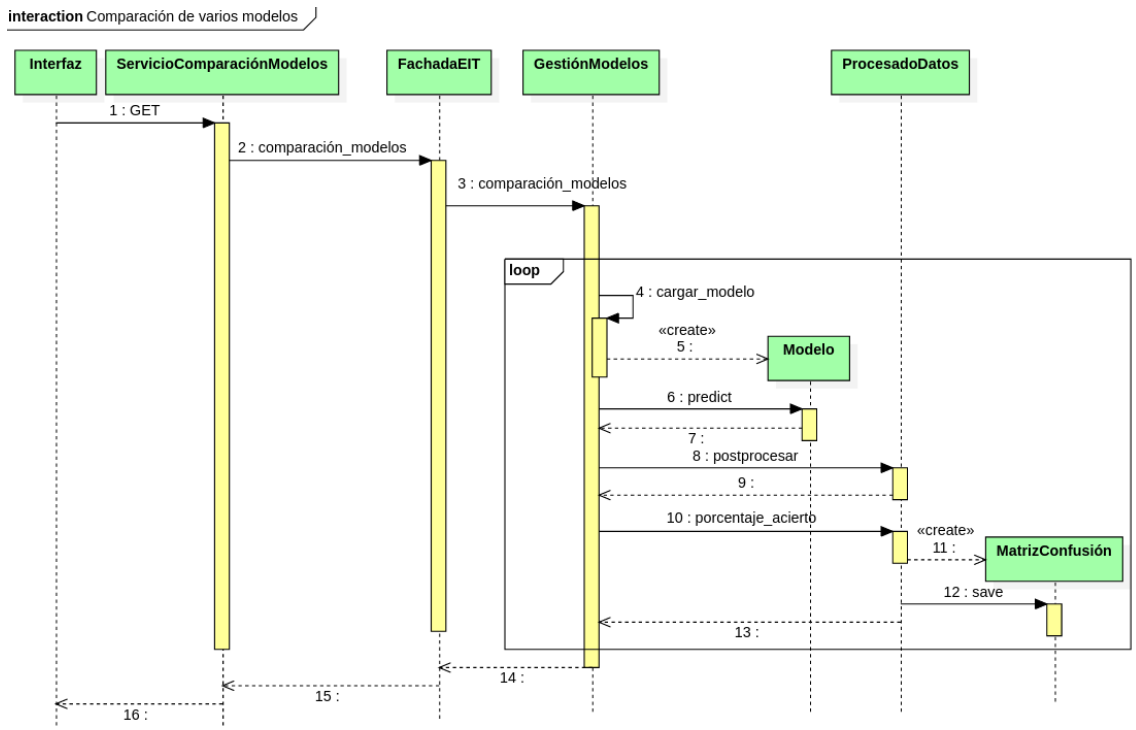

Figura 5.7: Diagrama de secuencia: comparación de varios modelos.

Los pasos reflejados en la 5.7 son los siguientes:

1. Desde el *frontend* se envía una petición HTTP al servicio de comparación de modelos, utilizando GET como verbo de la petición. En los parámetros de la petición, se indica IDs de los modelos que se desea comparar, la lista de métricas que se desea utilizar para la comparación, el *dataset* sobre el que se realizarán las predicciones y, finalmente, se indica si se desea postprocesar o no los resultados.

2. Desde el servicio, se llama al método *comparación_modelos* de la fachada del módulo de EIT.

3. Desde el método anterior, se llama al método *comparación_modelos* de la clase *GestiónModelos*.

A continuación, para cada uno de los modelos a comparar, se realizan los siguientes pasos:

4, 5. Mediante el método *cargar_modelo*, se instancia el correspondiente modelo.

6, 7. Se realizan las predicciones de todo el conjunto de mallas del *dataset*.

8, 9. Se postprocesan las predicciones realizadas, mediante el método *postprocesar* de la clase *ProcesadoDatos*.

10. Se llama al método *porcentaje_acierto* de la clase *ProcesadoDatos* y se calcula el porcentaje de acierto a partir de las predicciones postprocesadas.

11. Se instancia un objeto de la clase *MatrizConfusión* con los resultados del paso 10.

12. Empleando el método *save*, se guarda la matriz de confusión generada en la base de datos.

13. El método *porcentaje_acierto* devuelve el porcentaje de acierto calculado y la matriz de confusión generada.

Cuando se ha iterado sobre todos los modelos, se realizan los últimos pasos:

14, 15, 16. Como respuesta a la petición del *frontend*, se devuelve un diccionario con los valores de las métricas obtenidos por cada modelo (en este ejemplo, la única métrica es el porcentaje de acierto) y un diccionario con las matrices de confusión asociadas a los modelos.

Generación de un *dataset*

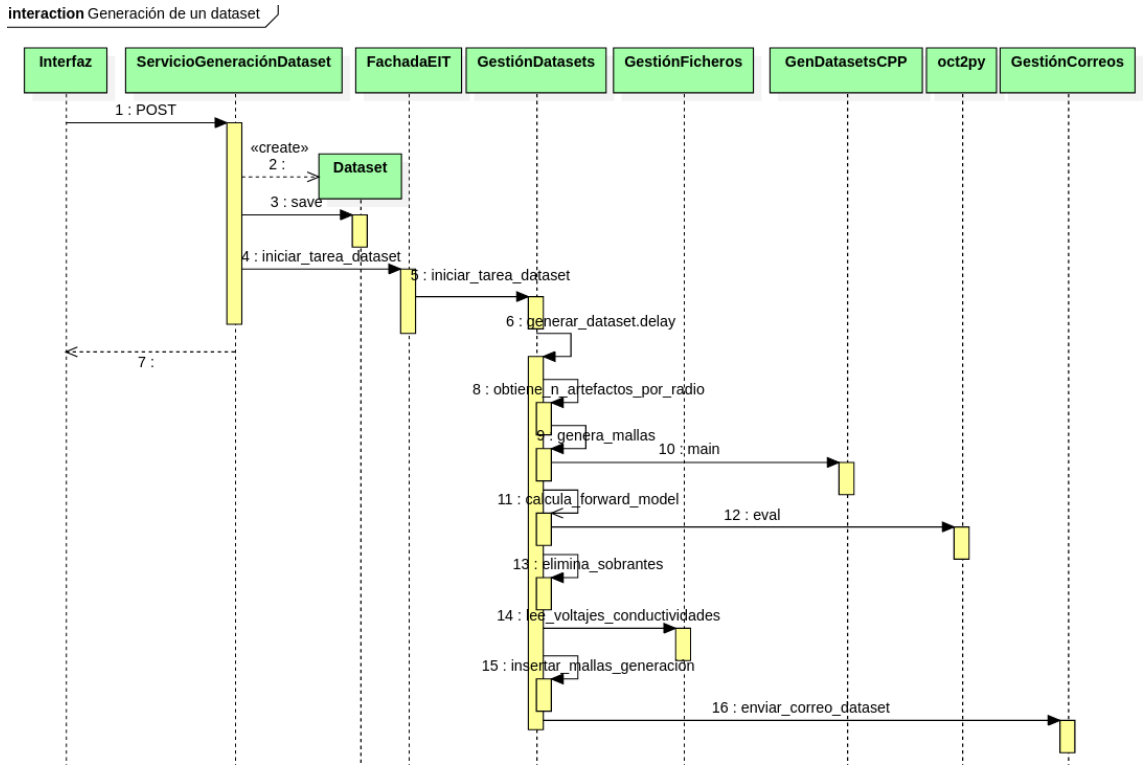

Figura 5.8: Diagrama de secuencia: generación de un *dataset*.

Los pasos reflejados en la 5.8 son los siguientes:

1. Desde el *frontend* se envía una petición HTTP al servicio de generación de *datasets*, utilizando POST como verbo de la petición. En el cuerpo de la petición, se envían los atributos que se desea que tenga el *dataset* a generar.

2. Desde el servicio, se instancia un objeto de la clase *Dataset* asignándole los atributos indicados en la petición HTTP.

3. Mediante el método *save*, se guarda la información del nuevo *dataset* en la base de datos.

4. Desde el servicio, se llama al método *iniciar_tarea_dataset* de la fachada del módulo de EIT.

5. Desde la fachada del módulo de EIT, se llama al método *iniciar_tarea_dataset* de la clase *GestiónDatasets*.

6. Se llama al método *generar_dataset*, para iniciar la generación del *dataset*. Mediante la palabra clave *delay*, el método es tratado como una tarea, la cual es enviada a la cola de tareas de Celery.

7. Se envía una respuesta al *frontend* con el código 200 de HTTP, indicando que la tarea de generación del *dataset* se ha enviado con éxito a la cola de tareas.

8. Se llama al método *obtiene_n_artefactos_por_radio*. A partir del número de mallas de uno, dos y tres artefactos y de los radios indicados en la petición, este método determina cuántas mallas de cada tipo deben generarse para cada uno de los radios.

9, 10. Se llama al método *genera_mallas*, el cual, a su vez, ejecuta el módulo de C++ para la generación de mallas. Este módulo genera un directorio con un conjunto de ficheros en un formato legible para EIDORS. Cada fichero contiene los valores de conductividad de una malla.

11, 12. Para cada una de las mallas es necesario calcular sus valores de voltaje. Para ello, se llama al método *calcular_forward_model*, quien ejecuta el código de Octave correspondiente, a través del módulo *oct2py* de Python. Para cada malla, se genera un nuevo fichero de voltajes en un formato propio de EIDORS.

13. Se llama al método *elimina_sobrantes*, el cual elimina algunos ficheros sin utilidad que se generan en los pasos anteriores.

14. Se llama al método *lee_voltajes_impedancias* para leer los valores de voltaje e conductividad de los ficheros generados.

15. Mediante el método *insertar_mallas_generación*, se instancian los objetos de la clase *Malla* que conformarán el *dataset*. Cuando se han instanciado todas las mallas, se hace una única inserción en la base de datos.

16. Cuando ha finalizado la generación del *dataset*, mediante el método *enviar_correo_dataset* de la clase *GestiónCorreos*, se envía un correo al usuario que realizó la petición de generación del *dataset*, informándolo de que la generación ha finalizado.

5.2.6. Integración entre Python y EIDORS

Uno de los aspectos fundamentales del proyecto es la integración entre Python y EIDORS. Tal y como se explicó en el apartado 4.1.5, EIDORS es una biblioteca que puede ser utilizada en MATLAB y Octave. Permite simular la realización de tomografías de impedancia eléctrica. Sin embargo, el *backend* del sistema está implementado en Python, por lo que fue necesario buscar una manera adecuada de llamar a las funciones de EIDORS desde Python. La fórmula finalmente elegida para ejecutar las funciones de EIDORS consistió en hacer uso de la biblioteca *oct2py* de Python, la cual permite hacer llamadas a Octave desde Python, de la siguiente manera:

```
from oct2py import octave
octave.eval("CONJUNTO DE SENTENCIAS DE OCTAVE")
```

Como se puede ver en el código anterior, mediante el método *eval* es posible ejecutar desde Python cualquier código de Octave y, por tanto, es posible ejecutar las funciones de la biblioteca EIDORS.

5.2.7. Autenticación y permisos

Los servicios ofrecidos por el *backend* pueden ser consumidos por cualquier software y no únicamente por personas a través de la interfaz gráfica. Por tanto, es necesario gestionar la autenticación y los permisos de forma adecuada. En el sistema SageTomo se ha optado por una autenticación basada en *tokens*. Cuando un usuario o un software se identifican con éxito mediante el servicio de *login*, el sistema genera una cadena de caracteres aleatoria (*token*), la cual es incluida en la respuesta HTTP que se devuelve al cliente. Todas las peticiones HTTP para hacer uso de los servicios que reciba el *backend* deben incluir un *token*. En caso de que no se incluya el *token* o de que el *token* incluido no se corresponda con ningún usuario autenticado, la petición será rechazada.

En lo relativo a los permisos, cualquier usuario autenticado puede hacer uso de todos los servicios ofrecidos por el *backend*. Por tanto, en el fichero *settings.py*, la configuración de permisos y autenticación incluida es la siguiente:

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES' :
    ['rest_framework.authentication.TokenAuthentication',],

    'DEFAULT_PERMISSION_CLASSES' :
    ['rest_framework.permissions.IsAuthenticated',],

}
```

Se puede observar que en la opción DEFAULT_PERMISSION_CLASSES se utiliza la clase de Django REST *IsAuthenticated*, mediante la cual se indica que, por defecto, los usuarios autenticados tendrán permiso para hacer uso de todos los servicios.

5.3. Diseño del *frontend*

5.3.1. Diagrama de clases del *frontend*

Tal y como se mencionó previamente en el apartado 4.1.3, el *framework* elegido para la implementación del *frontend* es *React*. Mediante React, las vistas de la interfaz gráfica se construyen utilizando uno o varios componentes. Los componentes son clases de JavaScript que constan de un estado y de un conjunto de métodos. En la figura 5.9, se muestra un diagrama de clases de muy alto nivel del *frontend*:

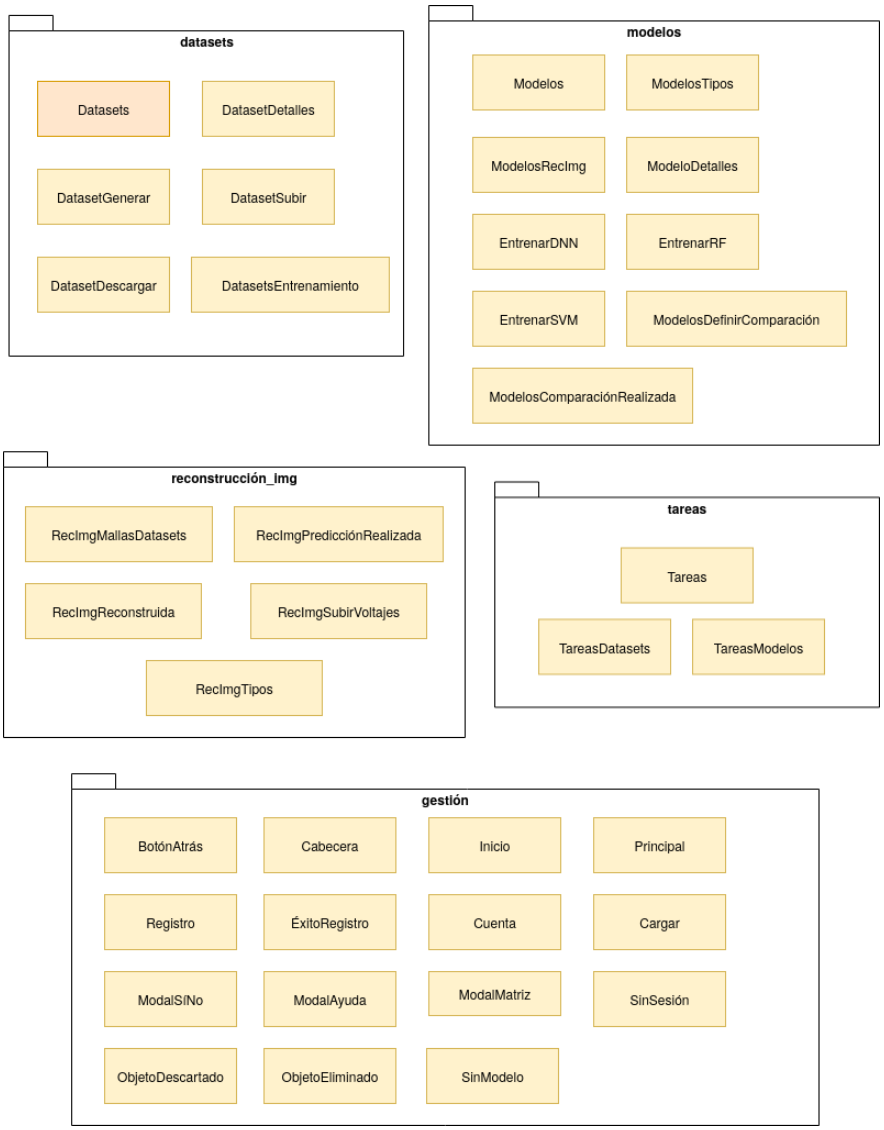

Figura 5.9: Diagrama de clases del *frontend*.

En términos generales, cada una de las clases del diagrama anterior se corres-

ponde con una de las vistas de la aplicación, con la excepción de ls siguientes clases: *BotónAtrás*, *Cabecera*, *Cargar*, *ModalSíNo*, *ModalAyuda* y *ModalMatriz*. Estas clases son componentes que se integran en las vistas de la aplicación.

Patrón Vista-Compuesta

En el *frontend* se ha utilizado el **patrón Vista-Compuesta**, el cual es un patrón de diseño tipo estructural. Empleando el patrón Vista-Compuesta, se construye una determinada vista mediante la agregación de múltiples subvistas. En el caso particular de las vistas de esta aplicación, todas las páginas que componen la vista de la aplicación contienen una cabecera. En lugar de incluir el mismo fragmento de código para la cabecera de cada una de las páginas, se utiliza un componente de React para dicha cabecera, el cual se incluye en todos los demás componentes.

5.3.2. Diseño de la interfaz gráfica

En este apartado se presentará la interfaz gráfica de la aplicación. La interfaz se ha diseñado buscando alcanzar el mayor nivel de usabilidad posible. La usabilidad se puede definir como la facilidad de uso de una aplicación. En el caso de la aplicación SmartTomo, ésta será empleada por usuarios expertos en el dominio del problema. Por tanto, los usuarios estarán familiarizados tanto con las tomografías de impedancia eléctrica como con las diferentes técnicas de *Machine learning*.

Para lograr que la aplicación sea usable, se ha diseñado la interfaz gráfica de manera que cumpla en todo momento con los diez principios heurísticos de Jacob Nielsen [27], relativos a la usabilidad de aplicaciones informáticas. En el cuadro 5.3, se muestran cuáles son los 10 principios de Nielsen.

ID	Nombre
N1	Visibilidad del estado del sistema.
N2	Utilizar el lenguaje de los usuarios.
N3	Control y libertad para el usuario.
N4	Consistencia y estándares.
N5	Prevención de errores.
N6	Minimizar la carga de la memoria del usuario.
N7	Flexibilidad y eficiencia de uso.
N8	Los diálogos estéticos y diseño minimalista.
N9	Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de los errores.
N10	Ayuda y documentación.

Cuadro 5.3: Principios heurísticos de Nielsen.

La aplicación cuenta con **más de 25 vistas diferentes**. En los siguientes apartados, se describirán únicamente las vistas más relevantes que componen la interfaz gráfica y, para aquellas vistas de mayor complejidad, se justificarán las decisiones de diseño, tomando como referencia los diez principios heurísticos de Nielsen.

5.3.3. Ventana principal

Una vez que el usuario inicia sesión (o se registra por primera vez), accede a la ventana principal de la aplicación.

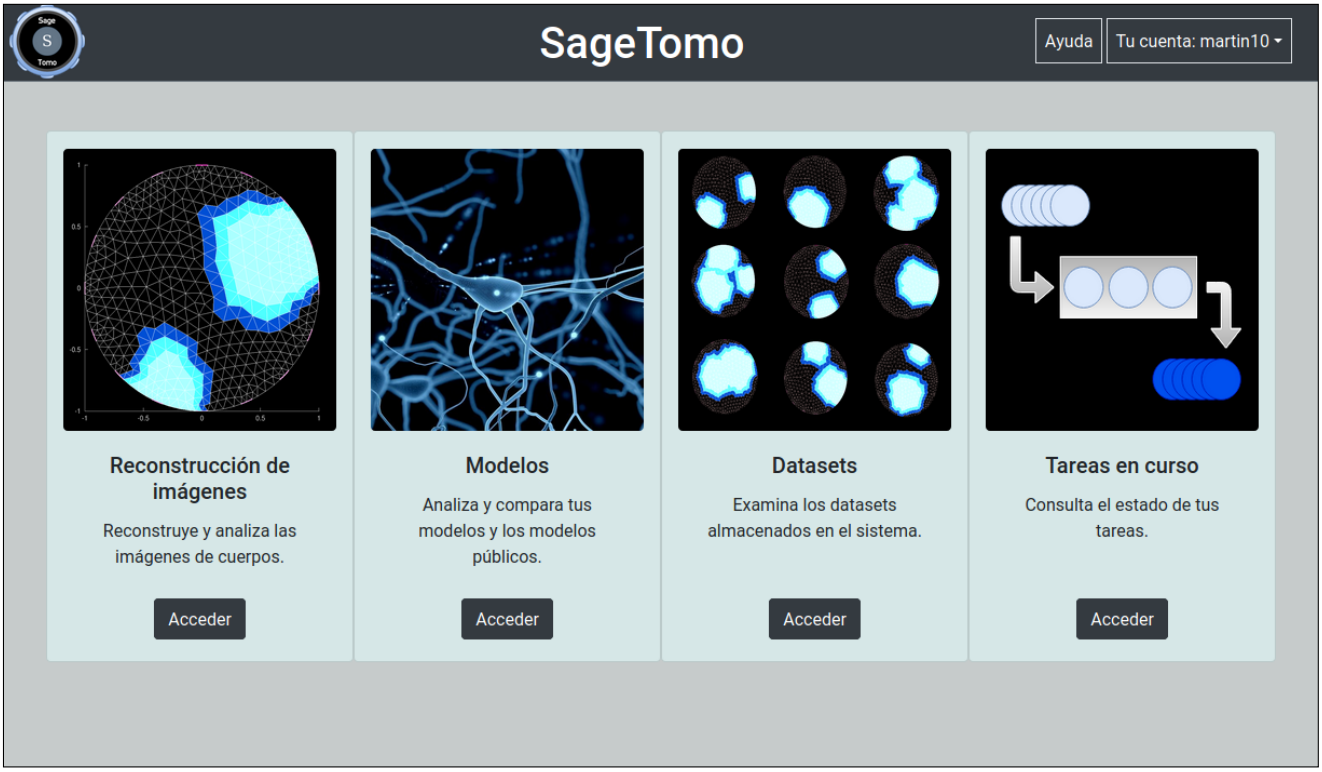

Figura 5.10: Vista principal de la aplicación.

En la ventana principal, se muestran al usuario las cuatro principales secciones de la aplicación:

- Reconstrucción de imágenes.
- Modelos.
- Datasets.
- Tareas.

Para cada una de las secciones, se ofrece una muy breve descripción de lo que puede encontrar el usuario al acceder a esa sección. En esta vista y en todas las vistas sucesivas, el usuario dispondrá en la parte superior de la ventana de un menú. En el caso de la ventana principal, el menú dispone únicamente de las opciones de *Ayuda* y *Tu cuenta*. En el caso de esta última, se incluye el símbolo de una flecha apuntando hacia abajo, para cumplir con el principio N2, puesto que cualquier usuario asocia esa flecha a un desplegable.

En las demás ventanas de la aplicación, en el menú superior aparece una nueva opción: la opción *Inicio*. Pulsando en ella, el usuario regresará a la página principal. Esta opción aparecerá en todas las vistas sucesivas que se presenten en los siguientes apartados. De esta forma, se cumple con el principio N3, puesto

que los usuarios tienen la posibilidad de regresar al menú principal en cualquier momento. Cabe señalar que el menú superior es estático y se encontrará siempre en la parte superior de la página, independientemente de que el usuario haga *scroll* hacia abajo. Se ha optado por este diseño para que el menú sea visible para el usuario en todo momento, de forma que pueda acudir con facilidad a la ventana principal.

Por otra parte, también en la esquina superior izquierda de las vistas, se encuentra el botón *Atrás*, que incluye una flecha apuntando hacia la izquierda, que es interpretada por los usuarios como una señal de regreso (se cumple con N2). Este botón también se incluirá en la mayor parte de las vistas sucesivas y permitirá al usuario regresar a la página anterior, cumpliendo con N3.

5.3.4. Tipo de reconstrucción

En la sección de *Reconstrucción de imágenes*, una vez que el usuario elija un modelo para la reconstrucción de una imagen, deberá indicar qué tipo de operación desea realizar. Puede seleccionar entre la reconstrucción de la imagen de una malla de un dataset o la predicción de las conductividades a partir de un fichero de voltajes (el cual debe subir al sistema).

Figura 5.11: Selección de la operación a realizar en la sección de *Reconstrucción de imágenes*.

5.3.5. Reconstrucción de imágenes

Selección de *dataset* y malla

Figura 5.12: Vista de selección de *dataset* y malla (I).

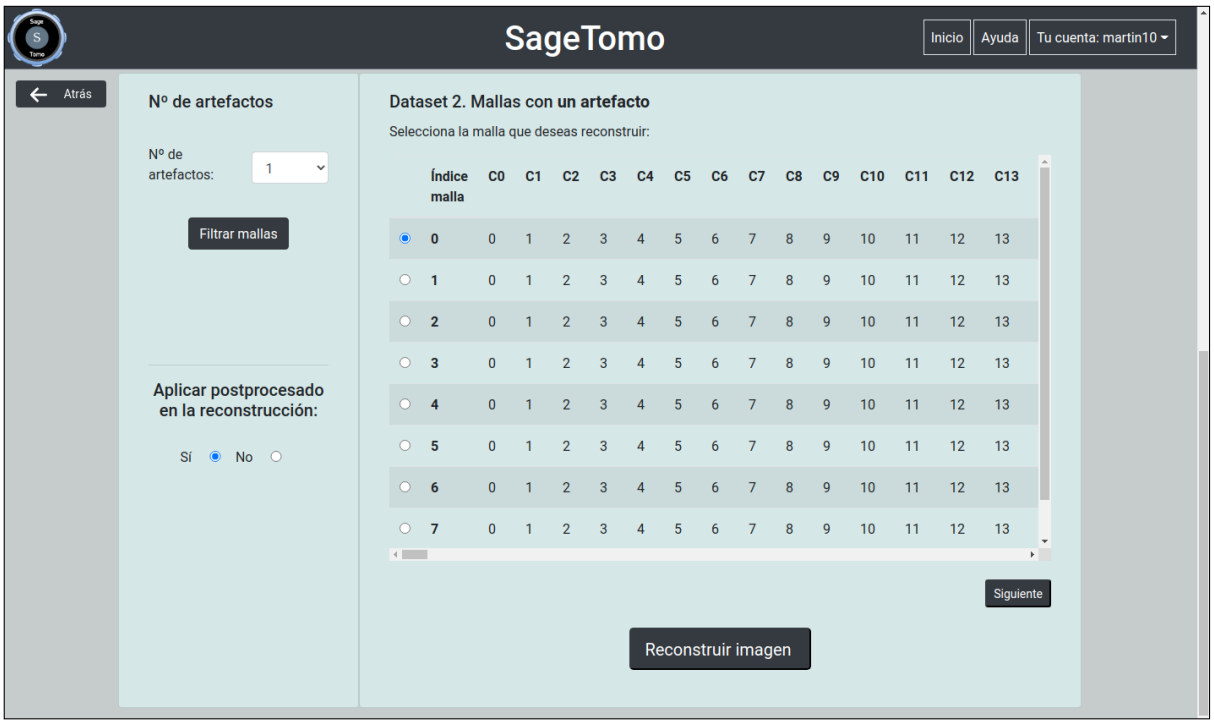

Figura 5.13: Vista de selección de *dataset* y malla (II).

Si en la vista de selección de operación, el usuario selecciona la opción *Análisis de mallas de datasets*, accederá a la vista de selección de malla. En ella podrá escoger un *dataset* y seleccionar la malla del *dataset* que desea reconstruir con el modelo. En lo relativo a la usabilidad de esta vista, cabe destacar que se busca cumplir con el principio N6 mostrando al usuario en pantalla la siguiente información: qué modelo ha seleccionado, qué *dataset* ha seleccionado y qué número de artefactos por malla ha seleccionado. De esta manera, el usuario no se ve obligado a recordar toda esta información.

Cuando el usuario haya seleccionado la malla que desea reconstruir, pulsando en el botón *Reconstruir imagen* se iniciará la reconstrucción de la imagen. Como este proceso tarda unos segundos, se muestra al usuario un símbolo habitual de carga, cumpliendo con el principio N1. Cuando la reconstrucción haya finalizado, se cargará una nueva ventana con la imagen reconstruida.

Imagen reconstruida

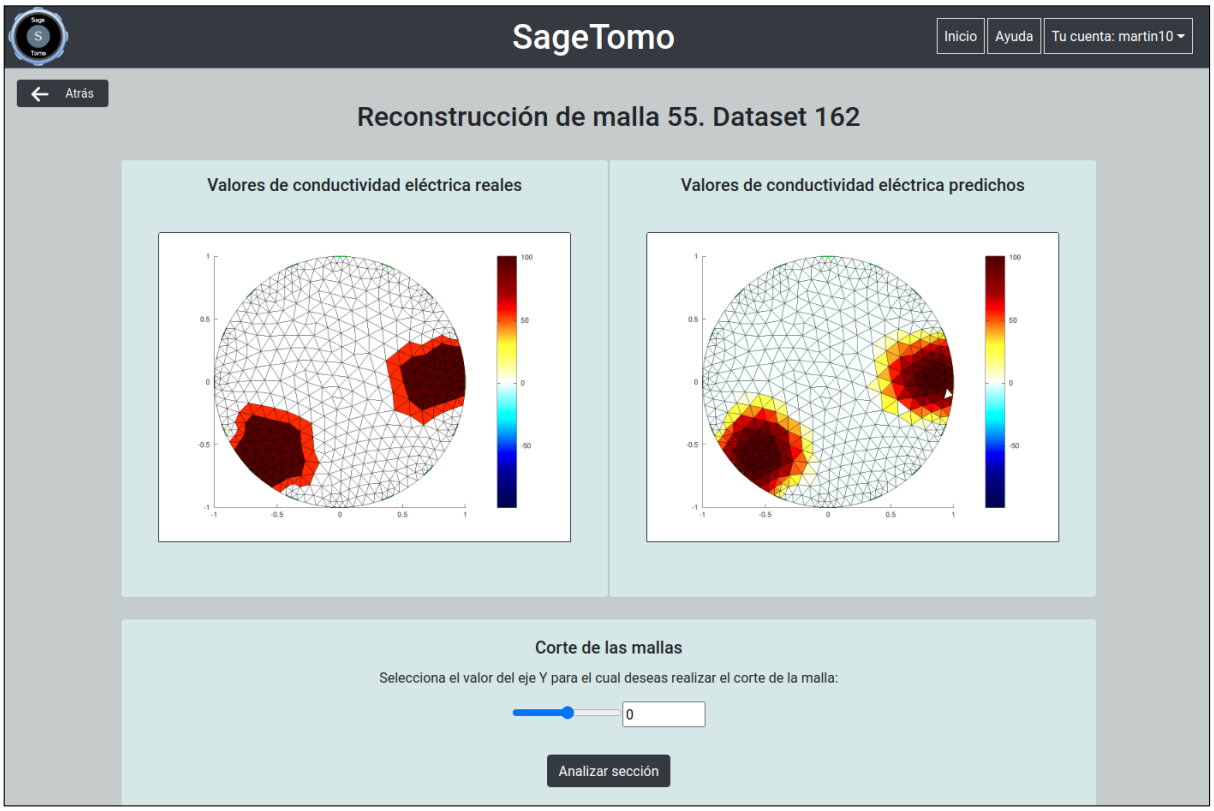

Figura 5.14: Vista de imagen reconstruida.

En esta vista, se mostrarán al usuario dos imágenes: la primera de ellas es la imagen real asociada a la malla seleccionada. La segunda imagen es la imagen reconstruida mediante el modelo elegido por el usuario. De nuevo, se indica al usuario el *dataset*, la malla y el modelos utilizados, cumpliendo con N6. Debajo de las dos imágenes se encuentra la sección para realizar cortes sobre las mallas. El usuario puede seleccionar el valor del eje Y para el que desea realizar el corte. Pulsando en *Analizar sección*, se generará en la misma ventana una gráfica en la que se muestran los valores de conductividad de la malla en el eje Y indicado por el usuario, como se puede ver en la figura 5.15.

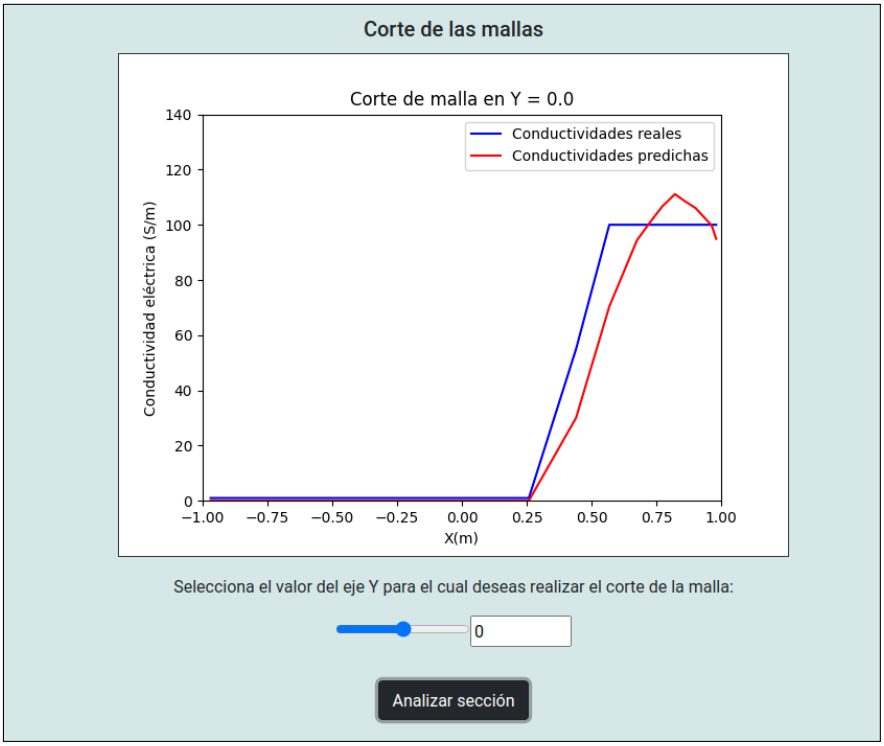

Figura 5.15: Vista de corte de una malla.

Predicciones realizadas

Si en la ventana de selección de operación, el usuario escoge la opción *Predicción de conductividades a partir de un conjunto de voltajes*, accederá a una vista en la que deberá subir un fichero CSV, el cual debe contener los voltajes asociados a mallas. Una vez que lo suba y pulse en *Realizar predicciones*, podrá visualizar los resultados de las predicciones y reconstruir las imágenes de las mallas que desee.

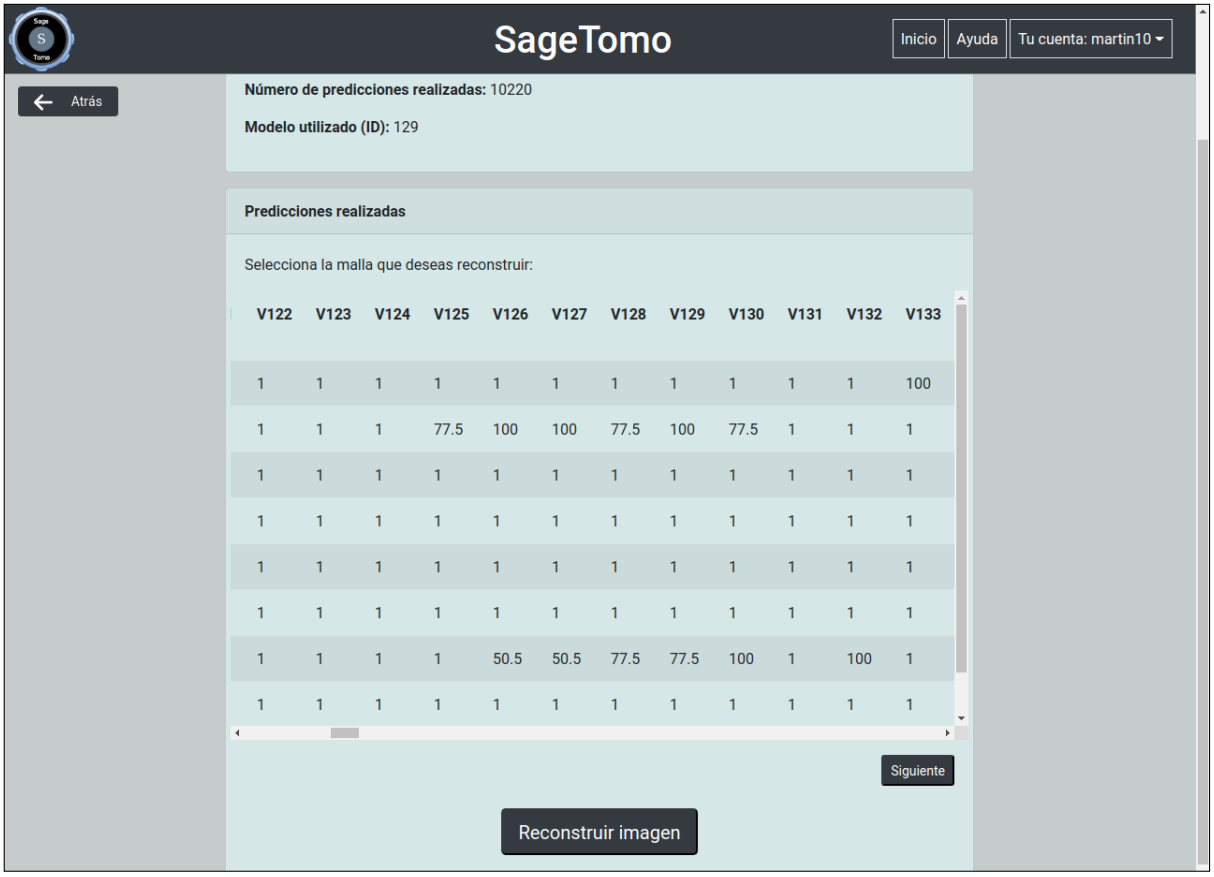

Figura 5.16: Vista de las predicciones de conductividades a partir de un conjunto de voltajes.

5.3.6. Modelos

Lista de modelos

Figura 5.17: Vista del listado de modelos disponibles.

Accediendo a la sección de *Modelos* desde la ventana principal de la aplicación, se muestra una vista que contiene el listado de modelos públicos o entrenados por el propio usuario, con la información general de cada modelo. Mediante la opción *Filtrar modelos*, el usuario tiene la posibilidad de filtrar los modelos de acuerdo a características de su interés como, por ejemplo, el tipo de modelo. Tal y como se puede ver en la figura 5.17, en la parte derecha de la fila correspondiente a cada modelo, se encuentra la opción *Ver detalles*. Pulsando en esa opción, el usuario podrá consultar información específica sobre cada modelo. El tipo de información que se mostrará variará en función del tipo de modelo que se consulte. Por ejemplo, para una red neuronal, al clicar en *Ver detalles* se accederá a una ventana como la mostrada en la figura 5.18.

Figura 5.18: Vista de los detalles de una red neuronal.

Como se puede ver en la figura 5.18, entre la información asociada a un modelo, se encuentra el *dataset* con el que se entrenó. El usuario tiene la posibilidad de consultar, a su vez, la información detallada del *dataset* utilizado para entrenar el modelo que está examinando, pulsando en el botón *Ver información* de la fila en la que se indica el ID del *dataset*. Al pulsar en ese botón, accederá a una nueva ventana de detalles del *dataset*, la cual se analizará posteriormente.

Por otra parte, cabe destacar que si una de las métricas almacenadas para el modelo es el porcentaje de acierto, el usuario tiene la posibilidad de visualizar la matriz de confusión, pulsando en *Ver matriz de confusión*. Al pulsar en ese botón, se despliega en la misma ventana un cuadro que contiene la matriz de confusión y que el usuario puede cerrar clicando en el botón *Cerrar que contiene dicho cuadro* o en el símbolo X situado en la esquina superior derecha del cuadro. Un ejemplo de la visualización de una matriz de confusión se puede ver en la figura 5.19.

Figura 5.19: Vista de una matriz de confusión.

Comparar modelos

Como se puede observar en la figura 5.17 en la ventana de *Modelos*, bajo el listado de modelos disponibles, se encuentra la opción *Comparar modelos*. El usuario tiene la posibilidad de realizar la comparación de dos a cuatro modelos, seleccionando de la lista los modelos que desea comparar. Si pulsa en *Comparar modelos* habiendo seleccionado menos de dos modelos, se le mostrará un mensaje de error (principio N9). Si ya ha seleccionado cuatro e intenta seleccionar uno más, el sistema no se lo permitirá, cumpliendo con el principio N5.

Habiendo seleccionado un número adecuado de modelos y pulsando en *Comparar modelos*, el usuario accederá a la ventana de definición de los parámetros de la comparación (figura 5.20).

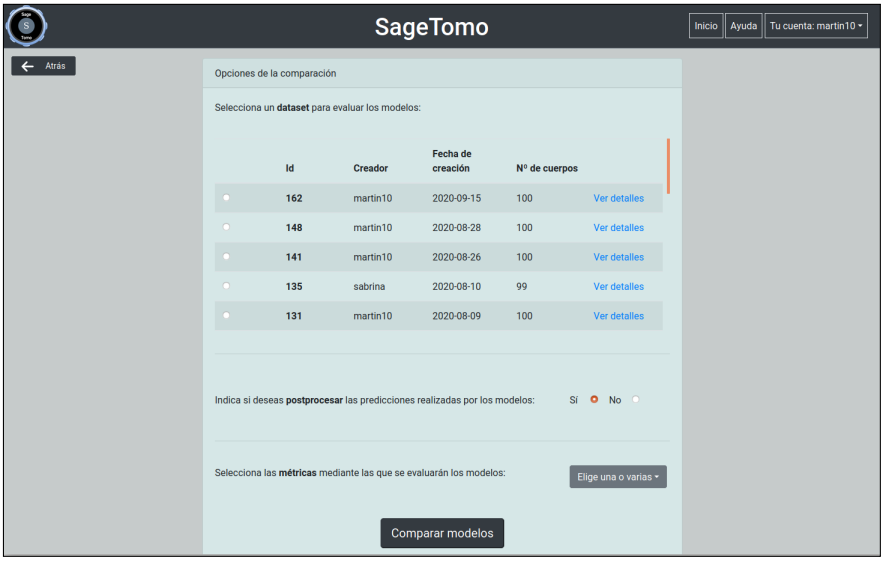

Figura 5.20: Vista de la definición de los parámetros de la comparación de modelos.

El usuario tiene la posibilidad de seleccionar el *dataset* mediante el cual se desean comparar los modelos, puede indicar si desea postprocesar o no las predicciones realizadas y puede seleccionar las métricas con las que desea evaluar los modelos. Una vez haya realizado la selección de su interés, pulsando en *Comparar modelos*, se iniciará la comparación de los modelos. Ésta puede tardar varios segundos, por lo que, de la misma forma que en el caso de reconstrucción de imágenes, se muestra al usuario un símbolo de carga, cumpliendo con el principio N1. Cuando la comparación haya finalizado, se cargará una nueva ventana con los resultados de la comparación.

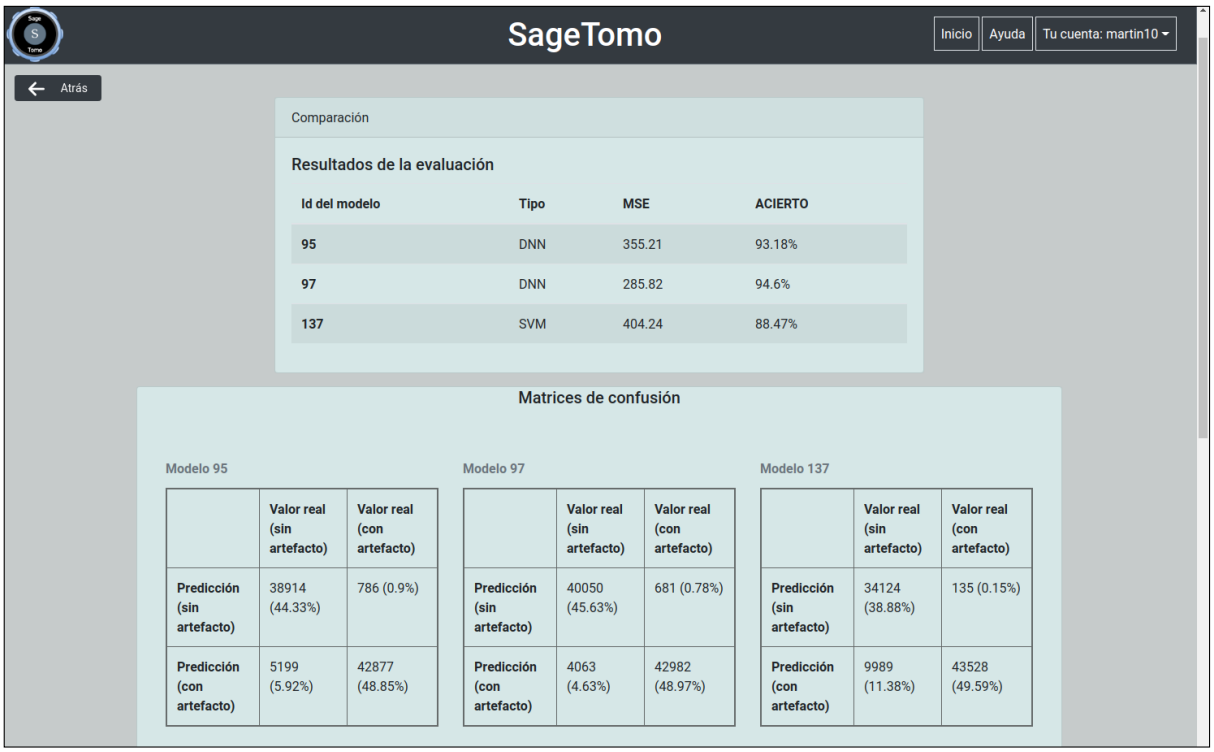

Figura 5.21: Vista de los resultados de la comparación de modelos (1).

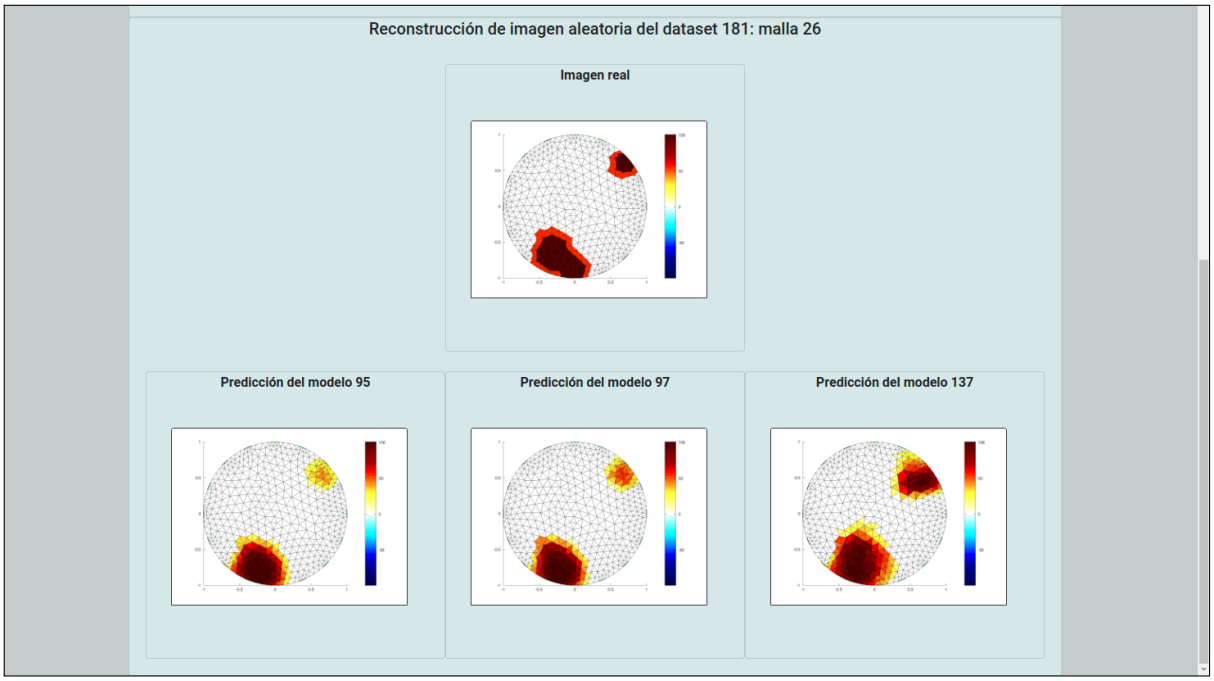

Figura 5.22: Vista de los resultados de la comparación de modelos (2).

En las figuras 5.21 y 5.22, se muestra la ventana de los resultados de una comparación entre cuatro modelos. En la parte superior de la vista, se incluye una tabla con los resultados obtenidos por cada modelo para cada métrica. En el caso de que una de las métricas sea el porcentaje de acierto, debajo de la tabla, se incluyen las matrices de confusión asociadas a cada modelo. Finalmente, se muestra un conjunto de imágenes, asociadas a una malla seleccionada de forma aleatoria de entre las mallas del *dataset* elegido por el usuario. La imagen superior es la imagen real de la malla, mientras que el resto de imágenes se corresponden con las reconstrucciones realizadas por cada uno de los modelos.

Entrenar modelos

En la ventana de *Modelos* (figura 5.17), bajo el listado de modelos, se encuentra la opción *Entrenar nuevo modelo*. Pulsando en ella, se cargará una ventana en la que el usuario podrá elegir el tipo de modelo que desea entrenar(figura 5.23).

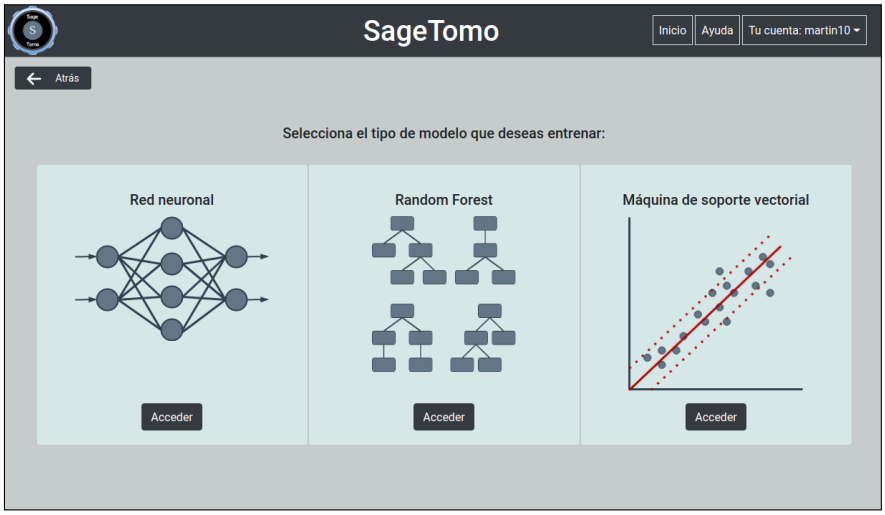

Figura 5.23: Vista de selección de tipo de modelo para entrenamiento.

Se ha optado por este diseño sencillo para cumplir con el principio N8. Clicando en un tipo de modelo, el usuario accederá a la ventana de definición de los parámetros de entrenamiento para el tipo de modelo elegido.

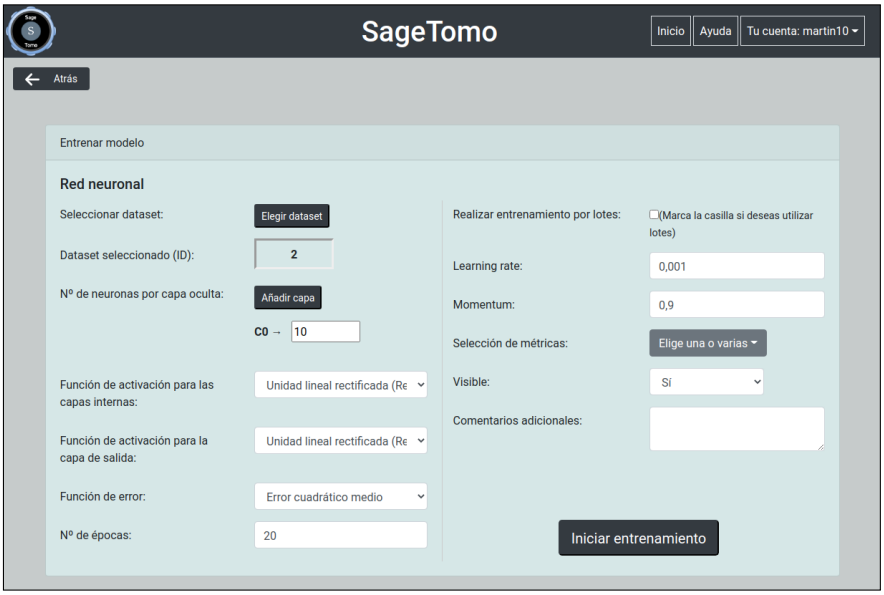

Figura 5.24: Vista de la definición de parámetros para el entrenamiento de una red neuronal.

En la figura 5.24 se puede visualizar la ventana de definición de parámetros para el entrenamiento de un modelo de tipo *Red Neuronal*. Este es el tipo de modelo cuya definición de parámetros es más compleja, debido a que se ofrece al usuario la posibilidad de seleccionar el número de capas ocultas y el número de neuronas por cada capa oculta. Se descartó la posibilidad de que el usuario

tuviese que introducir un vector, puesto que existiría una elevada probabilidad de que el usuario introdujese información en un formato incorrecto. Por tanto, se optó por un diseño dinámico, implementado con JavaScript, mediante el cual se genera un cuadro adicional de inserción de número de neuronas cada vez que el usuario incrementa en una unidad el número de capas ocultas. Análogamente, cada vez que el usuario disminuye en una unidad el número de capas ocultas, se elimina el último cuadro de inserción de número de neuronas. Se cumple con el principio N5, puesto que se evita que el usuario introduzca información en un formato incorrecto. Además, se cumple con el principio N9, dado que en el caso de que el usuario introduzca un número de neuronas no permitido (superior al límite máximo establecido), se le indica específicamente la capa o capas en las que ha introducido valores inválidos. Cuando el usuario termine de establecer o definir los parámetros del modelo que desea entrenar, si pulsa en *Iniciar entrenamiento*, comenzará el entrenamiento del modelo y el usuario será redirigido a la ventana de *Entrenamientos* (ver apartado 5.3.8), donde verá que el que el entrenamiento que acaba de iniciar se encuentra en la lista de entrenamientos en curso, cumpliendo así con el principio N1.

5.3.7. Datasets

Accediendo a la sección de *Datasets* desde la ventana principal de la aplicación, se muestra una vista que contiene el listado de datasets públicos o generados por el propio usuario, con la información general de cada *dataset*. Para cada uno de los datasets de la lista, se muestran las opciones *Ver detalles* y *Descargar*. Para aquellos datasets subidos o generados por el usuario, se incluirá también la opción *Eliminar*.

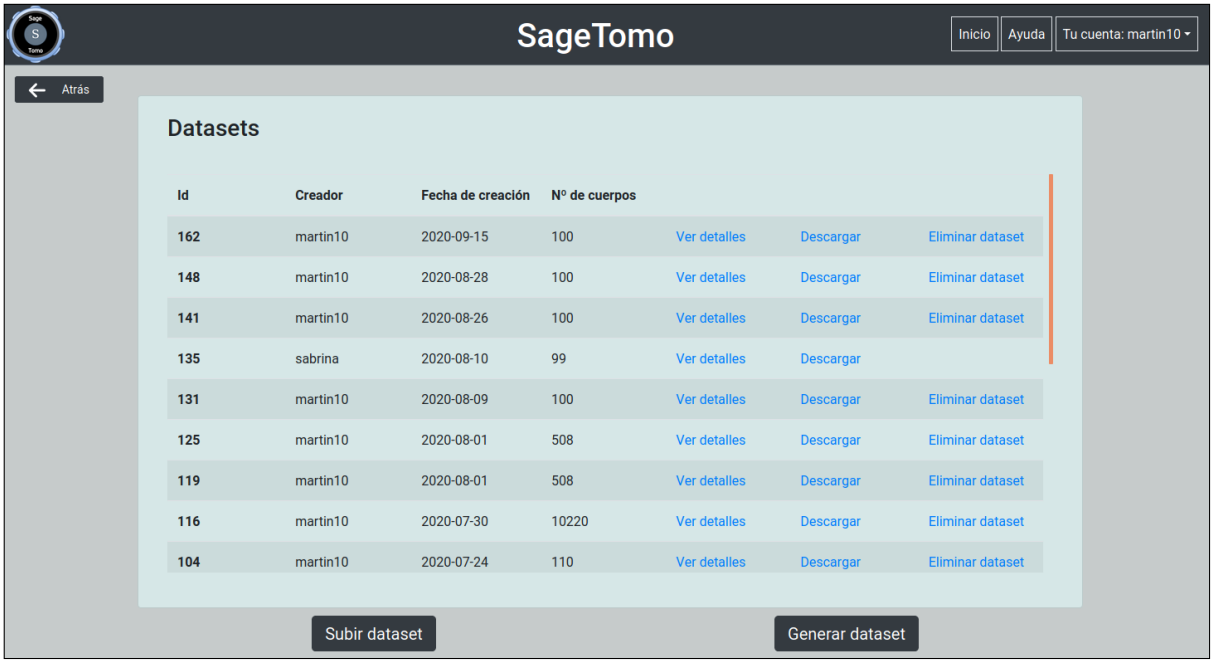

Figura 5.25: Vista de la información detallada de un *dataset*.

En caso de que el usuario pulse en *Ver detalles*, accederá a una nueva ventana en la que se le mostrará información detallada sobre el *dataset* seleccionado, como se puede ver en la figura 5.26.

Figura 5.26: Vista de la información detallada de un *dataset*.

La información general ya mostrada se incluye de nuevo en la ventana de detalles, para cumplir con el principio N6.

Generar *dataset*

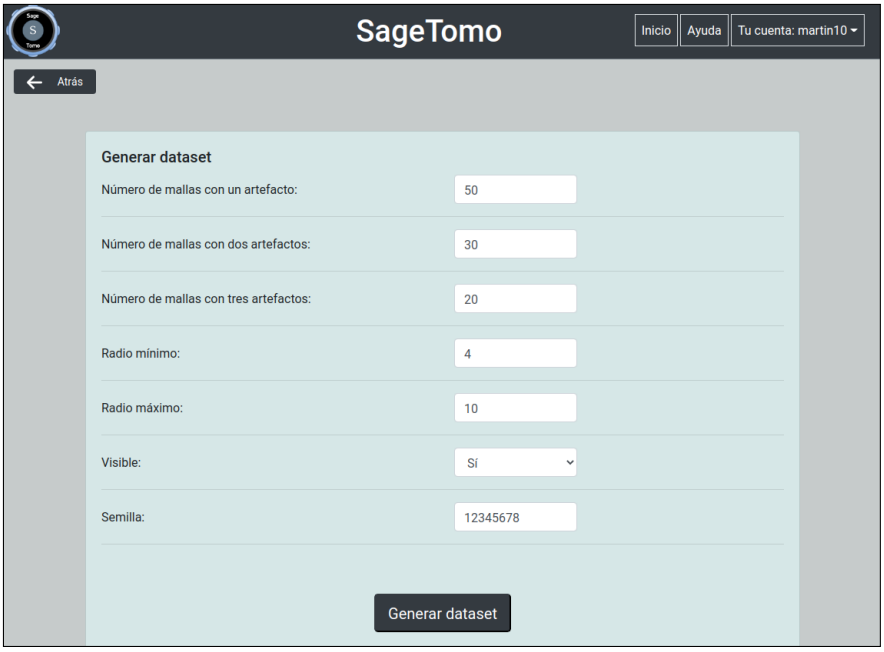

Figura 5.27: Vista de generación de *dataset*.

Si en la vista de *Datasets*, el usuario pulsa en *Generar dataset* accederá a una nueva ventana (figura 5.27) en la que podrá definir las características del *dataset* que desea generar, como el número de mallas. Al pulsar en el botón *Generar dataset* de esta ventana, se iniciará la generación del *dataset* y el usuario será redirigido a la ventana de *Datasets en generación*, donde podrá ver que el *dataset* que cuya generación acaba de iniciar se encuentra en la lista de datasets en curso de ser generados. De forma análoga a los modelos en entrenamiento, este diseño cumple con el principio *N1*.

5.3.8. Tareas

Figura 5.28: Vista de selección del tipo de tarea.

En la sección de *Tareas*, los usuarios podrán consultar los entrenamientos en curso y los datasets en generación. Cuando el usuario accede a esta sección desde el menú principal, se le mostrará una ventana sencilla (se cumple el principio N8) en la que podrá indicar si desea acceder a las tareas asociadas a los modelos (entrenamientos) o a las tareas asociadas a los datasets (generación).

Entrenamientos

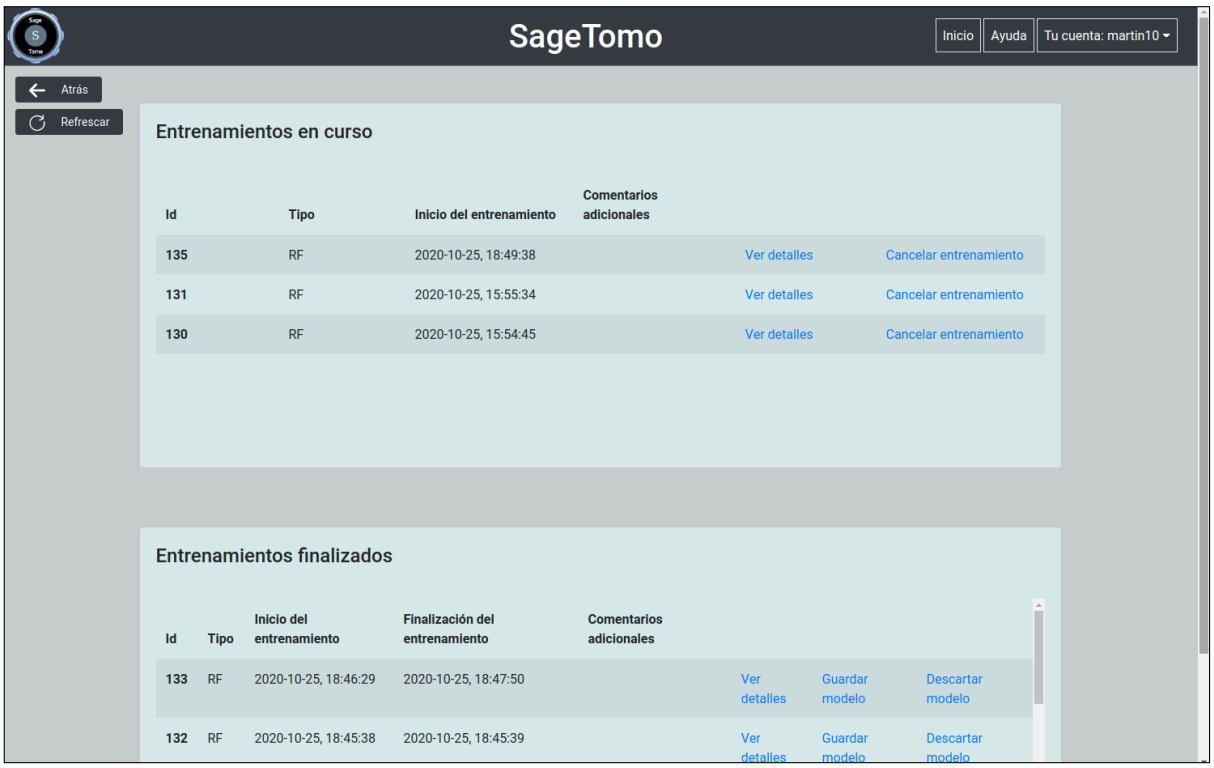

Figura 5.29: Vista de selección del tipo de tarea.

En la vista de entrenamientos, se muestran al usuario dos listas diferentes. La lista superior incluye los entrenamientos que se encuentran en curso en ese momento, mientras que la lista inferior contiene los modelos cuyo entrenamiento ya ha finalizado. En la esquina superior derecha de la ventana, el usuario tiene la posibilidad de refrescar la pantalla, pulsando en el botón *Refrescar*. Se cumple así con con el principio N3. Por otra parte, el botón incluye el símbolo típicamente asociado a la acción de recargar, por lo que se cumple también con el principio N2.

Capítulo 6

Machine Learning

En este capítulo se analizarán los aspectos más relevantes del sistema asociados al ámbito del *Machine Learning*. En primer lugar, se explicarán las características de los *datasets* utilizados y cómo éstos son empleados para entrenar los modelos. En segundo lugar, se presentarán los diferentes tipos de modelos de *Machine Learning* y se realizará una comparación entre ellos. Finalmente, se describirá la técnica de postprocesado utilizada a la hora de reconstruir las imágenes.

6.1. *Datasets*

De manera simplificada, un *dataset* se puede definir como un conjunto de datos utilizado para entrenar modelos de *Machine Learning*. Cada uno de los elementos de un *dataset* está constituido por una serie de variables, las cuales pueden ser **dependientes** (*inputs*) o **independientes** (*outputs*). Los valores de las variables dependientes están condicionados por los valores de las variables independientes. En el problema que nos ocupa, los datos que integran los *datasets* son mallas, las cuales constan de 1052 variables: 208 variables de voltaje y 844 variables de conductividad eléctrica. Las variables de conductividad eléctrica son las variables dependientes, puesto que deseamos entrenar modelos capaces de predecir su valor a partir de las variables de voltaje, que son las variables independientes.

El sistema desarrollado concede libertad al usuario a la hora de generar los *datasets*, permitiéndole seleccionar el número de mallas con uno, dos y tres artefactos, así como el radio mínimo y máximo de los artefactos. Sin embargo, si se desea generar un *dataset* que sea adecuado para entrenar modelos, éste debe contar con un número de mallas suficiente y éstas deben ser representativas. Si el número de mallas del *dataset* es demasiado reducido, los modelos entrenados con dicho *dataset* no habrán "aprendido" a generalizar el conocimiento adquirido y las predicciones realizadas con esos modelos sobre datos no pertenecientes al *dataset* serán poco precisas (*overfitting*). Por otra parte, se producirá una situación

similar si las mallas del *dataset* no son lo suficientemente representativas. Por ejemplo, si todas las mallas de un cierto *dataset* están integradas por un único artefacto con un determinado radio, un modelo entrenado con ese *dataset* tendrá dificultades para realizar las predicciones de mallas con tres artefactos de radios diversos.

6.1.1. Metodología de entrenamiento

Cuando un usuario selecciona un *dataset* para entrenar un nuevo modelo, se desordenan sus mallas de forma aleatoria. A continuación, el *dataset* se dividide en dos conjuntos: un **conjunto de entrenamiento** y un **conjunto de test**. El conjunto de entrenamiento está constituido por el 70 % de las mallas del *dataset*, mientras que el conjunto de test lo conforman el 30 % de las mallas restantes. El conjunto de entrenamiento se utiliza para entrenar el modelo, mientras que el conjunto de test se emplea únicamente para evaluar la precisión del modelo, una vez que éste ya ha sido entrenado. La razón por la que se desordenan las mallas antes de dividir el *dataset* es garantizar que tanto en el conjunto de entrenamiento como en el conjunto de test existan mallas de todos los tipos (con diferente número de artefactos y con artefactos de diferente radio), de forma que no se produzca *underfitting*. Por otro lado, la división de un *dataset* en los dos conjuntos mencionados es necesaria para poder evaluar el modelo de forma satisfactoria. Si el modelo se evaluase con el mismo conjunto de datos con el que se entrenó, no se podría determinar si el conocimiento del modelo es generalizable. Todas estas tareas se llevan a cabo empleando el método *obtiene_conjuntos* de la clase *Dataset*.

6.2. Modelos

La predicción de las 844 conductividades de una malla a partir de los 208 voltajes suministrados por sus electrodos es realizada por modelos de *Machine Learning*. Los problemas de este tipo, consistentes en predecir variables continuas dependientes a partir de variables continuas independientes reciben el nombre de problemas de **regresión**. Para solucionar este problema de regresión, los modelos del sistema SageTomo son entrenados siguiendo la metodología expuesta en el apartado anterior. Sin embargo, cada uno de los tres tipos de modelos utilizados por el sistema tiene características particulares. A continuación, se describirán los rasgos principales de los tres tipos de modelos: redes neuronales, *random forest* y máquinas de soporte vectorial.

6.2.1. Redes neuronales

Las redes neuronales se encuentran ligeramente inspiradas su homólogo biológico, de forma que una red neuronal está constituida por un conjunto de **neuronas** interconectadas. En la figura 6.1 se puede observar una arquitectura genérica para una red neuronal.

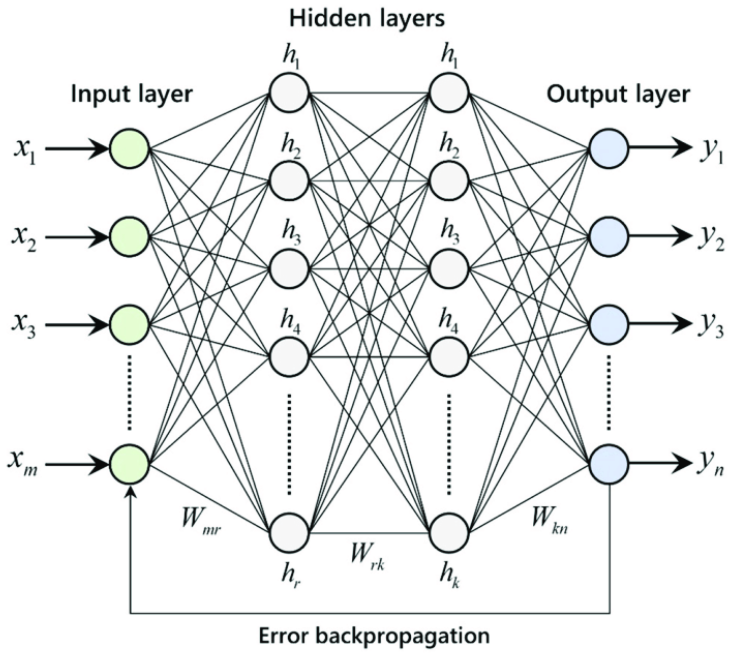

Figura 6.1: Arquitectura de una red neuronal [28].

Las neuronas de una red neuronal se distribuyen en **capas**, existiendo una capa de entrada, un conjunto de capas ocultas y una capa de salida. Las neuronas de una capa se conectan con las neuronas de la siguiente capa mediante **enlaces**, los cuales tienen asignado un **peso**. En la figura 6.2 se muestra cómo se calcula la salida de una neurona.

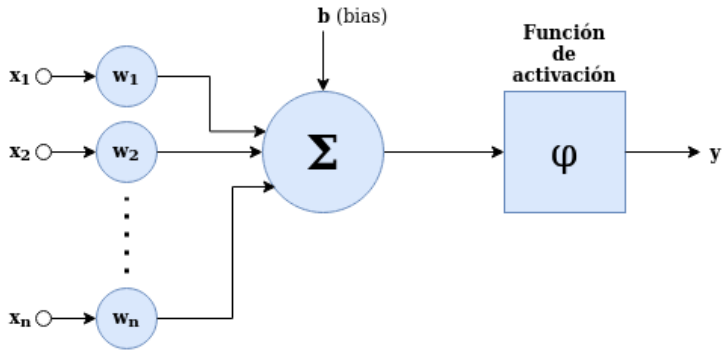

Figura 6.2: Cálculo de la salida de una neurona.

Como se puede ver en la figura anterior, además de los valores de entrada y de los pesos de los enlaces, existen dos elementos adicionales: el valor de *bias* y la función de activación. El valor de la salida y de una única neurona con n entradas x y con n enlaces (cada uno de los cuales tiene asignado un peso w), empleando un valor de *bias* b y utilizando una función de activación φ se obtiene de la siguiente manera:

$$y = \varphi(b + \sum_{i=1}^{n} x_i w_i)$$

La salida de una neurona se utiliza como entrada para las neuronas de las siguientes capas. El entrenamiento de una red neuronal consiste en ajustar los pesos de los enlaces progresivamente, hasta lograr que el modelo sea capaz de predecir con una cierta precisión la salida Y para una entrada X. Existen diferentes métodos para ajustar los pesos de una red. Las redes neuronales entrenadas por el sistema SageTomo emplean el método del **descenso del gradiente**, que es un algoritmo iterativo consistente en la minimización de una **función de coste**.

En el caso particular de los modelos del sistema a desarrollar, la capa de entrada tiene necesariamente 208 neuronas, puesto que existen 208 valores de entrada: los 208 voltajes generados por los electrodos. Por otra parte, la capa de salida tiene 844 neuronas, cada una de las cuales devuelve el valor de una de las 844 conductividades de una malla.

Existen varios factores relevantes a la hora de entrenar una red neuronal. A continuación, se describirán aquellos factores que el usuario del sistema SageTomo puede definir para entrenar este tipo de modelo:

- **Número de capas ocultas y número de neuronas por capa oculta**. Las capas ocultas son las capas de neuronas que se encuentran entre la capa de entrada y la capa de salida. El usuario puede elegir el número de capas ocultas que desea, así como el número de neuronas de cada una de las capas ocultas.

- **Función de activación para las capas internas y las capas de salida**. Tal y como se ha visto, en la salida de una neurona se aplica una función que modifica el valor de dicha salida (**función de activación**), logrando que el valor de salida de la neurona pertenezca a un intervalo concreto. Aunque podría utilizarse una función de activación particular para cada neurona, lo habitual es emplear una misma función de activación para todas las neuronas de las capas internas y otra función de activación para las neuronas de la capa de salida. Ésta es la opción que se le presenta al usuario de SageTomo, quien puede seleccionar la función de activación para ambos tipos de neuronas.

- **Función de error**. Es la función de coste que se debe minimizar durante la ejecución del método del descenso de gradiente.

- **Número de épocas**. En una época, todos los datos del conjunto de entrenamiento son suministrados a la red neuronal.

- **Entrenamiento por lotes y tamaño de los lotes**. Al utilizar un entrenamiento por lotes, el conjunto de entrenamiento se divide en subconjuntos denominados lotes. Cada vez que un lote es suministrado a la red neuronal, se actualizan los pesos de los enlaces.

- ***Learning rate***. Es el parámetro empleado en el método de descenso de gradiente para determinar cuánto se modifican los pesos en cada iteración. Cuanto mayor sea el valor del *learning rate*, mayor será la modificación de los pesos realizada en cada iteración.

- ***Momentum***. Es el parámetro empleado en el método de descenso de gradiente para acelerar o frenar la modificación de los pesos. Este factor perjudica o ayuda al *learning rate* dependiendo de lo sucedido en las iteraciones previas. De esta forma, permite evitar que el algoritmo se quede "atrapado" en un mínimo local, siendo incapaz de alcanzar el mínimo global. Cuanto menor sea el valor asignado al *momentum*, más difícil resultará que el algoritmo sea capaz de evitar los mínimos locales. Normalmente, el valor del *momentum* suele ser inversamente proporcional al *learning rate*.

En la siguiente figura se muestra como ejemplo la reconstrucción de la imagen de una malla con dos artefactos empleando una DNN.

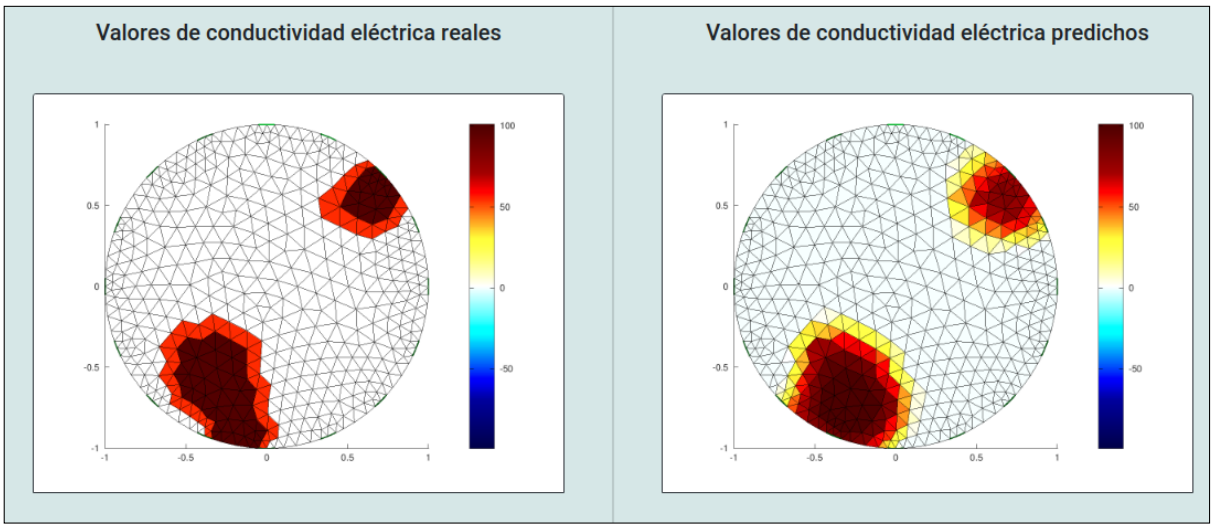

Figura 6.3: Reconstrucción mediante una DNN.

6.2.2. *Random Forest*

El *random forest* es un tipo de modelo basado en **árboles de decisión**. Los árboles de decisión empleados para resolver problemas de regresión reciben el nombre de **árboles de regresión**. En la figura 6.4 se muestra la representación de un árbol de regresión.

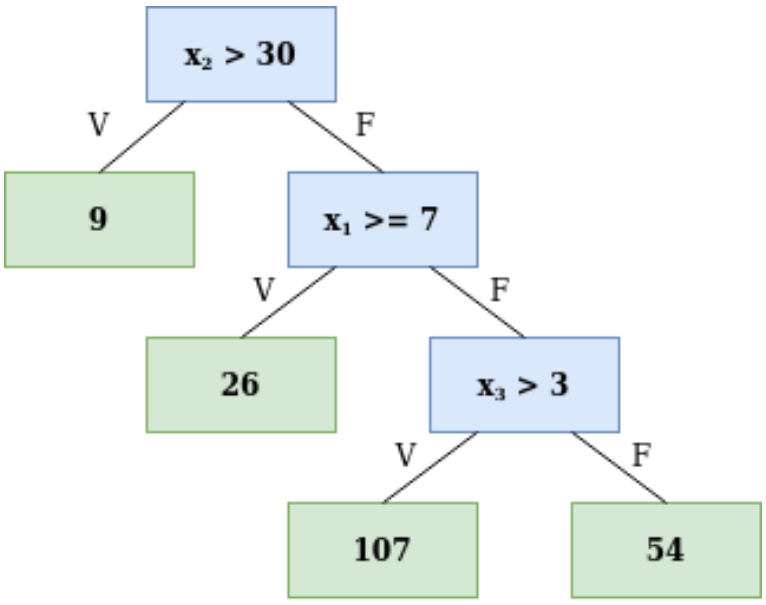

Figura 6.4: Árbol de regresión.

Como se puede ver, cada nodo del árbol constituye la evaluación de una condición. De cada nodo nacen dos ramas: una rama se corresponde con la situación

en la que la evaluación de la condición del nodo es verdadera y la otra rama se corresponde con la situación en la que la evaluación es falsa. Dado un determinado dato de entrada, se comienza evaluando la condición del nodo raíz del árbol (el nodo superior) y se sigue el camino hasta alcanzar un nodo hoja (los nodos verdes de la figura anterior). Los nodos hoja no contienen una condición, sino un valor, que es la **predicción para el dato de entrada**. Por ejemplo, para el dato tridimensional $x_1 = 9$, $x_2 = 72$, $x_3 = 8$, la predicción obtenida por el árbol de regresión de la figura es 26.

Supongamos que se dispone de un *dataset* con n datos, los cuales constan de las variables de entrada x_1, ... , x_k y de la variable de salida y. Para construir un árbol de regresión a partir de dicho *dataset*, se divide el espacio de las variables predictoras (x_i) en regiones diferentes. Para cada dato que caiga en una de esas regiones, se realiza una predicción, consistente en calcular la media de los datos de esa región. Se sigue una aproximación arriba-abajo, comenzando en el nodo raíz del árbol y dividiendo el espacio de variables predictoras en dos nuevas ramas. En cada nodo, se elije la mejor división posible del espacio de esa región. Siendo k el número de variables predictoras y c el punto de corte para una región, en un determinado nodo, el espacio de variables predictoras se divide en los dos siguientes conjuntos:

$$\begin{aligned} R_1(k,c) &= \{x | x_k \leq c\} \\ R_2(k,c) &= \{x | x_k > c\} \end{aligned}$$

Existen diferentes criterios para medir la bondad de una partición. Uno de los métodos para seleccionar las mejores particiones consiste en minimizar la suma de los cuadrados de los residuos. Siendo y_i la variable de salida para el dato x_i y $\hat{y}_{R_t}$ la predicción en la región R_t para el dato x_i, el objetivo es buscar en cada nodo los valores de k y c que minimicen la siguiente expresión:

$$\sum_{i:x_i \in R_1(k,c)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(k,c)} (y_i - \hat{y}_{R_2})^2$$

Por tanto, el objetivo es minimizar la expresión anterior (la suma de los cuadrados de los residuos).

Regresando al problema de predicción de conductividades de las mallas, se observa que existen 844 variables de salida (las 844 conductividades). Sin embargo, un árbol de regresión sólo permite predecir una variable de salida. La solución consiste en **entrenar 844 árboles de regresión**, de forma que cada uno de ellos se encargue de predecir una de las 844 impedancias.

Sin embargo, el gran problema que presentan los árboles de regresión es su dificultad para generalizar el conocimiento adquirido a través del *dataset* de entrenamiento. Es decir, estos modelos presentan una precisión reducida al realizar predicciones sobre datos a los que no se habían enfrentado anteriormente

(fenómeno que se conoce como *overfitting*). Por ello, en el sistema SageTomo, en lugar de árboles de decisión puros, se emplea el tipo de modelo conocido como *random forest*. Los *random forest* se construyen empleando la técnica de *bagging*, consistente en seleccionar aleatoriamente con repetición subconjuntos de datos del *dataset* y entrenar un árbol de regresión con cada uno de esos subconjuntos. Ahora, la predicción para un determinado dato será la media de las predicciones de todos los árboles del *random forest*. De nuevo, para el problema que nos atañe, el modelo constará de 844 *random forest*, cada uno de los cuales se encargará de predecir una de las 844 conductividades eléctricas.

A continuación, se describirán los parámetros que el usuario del sistema SageTomo puede definir para entrenar un *random forest*:

- **Número de estimadores**. Es el número de árboles de regresión de los que constará el *random forest* a entrenar.
- **Profundidad máxima de los árboles**. Es el número máximo de nodos permitidos que puede tener una rama de cada árbol de regresión.
- **Número mínimo de muestras para división**. Durante el proceso de construcción del árbol, éste es el número mínimo de datos del *dataset* que deben llegar hasta un nodo para permitir realizar una nueva división.
- **Número mínimo de muestras para un nodo hoja**. Durante el proceso de construcción del árbol, éste es el número mínimo de datos del *dataset* que deben llegar hasta un nodo para permitir que dicho nodo se convierta en nodo hoja.

En la siguiente figura se muestra como ejemplo la reconstrucción de la imagen de una malla con dos artefactos empleando un *Random Forest*.

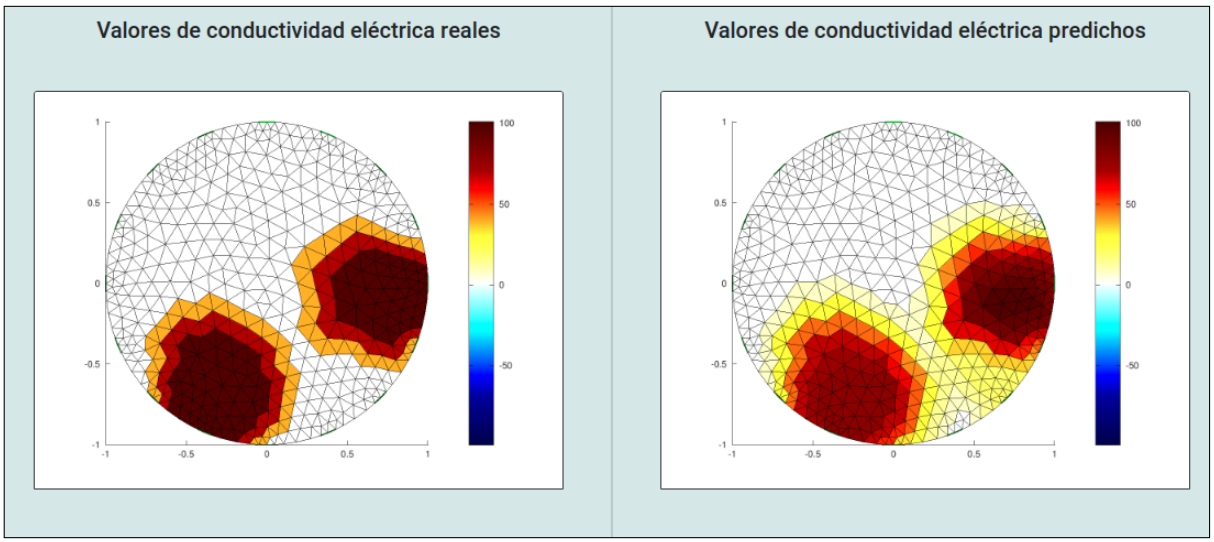

Figura 6.5: Reconstrucción mediante un *Random Forest*.

6.2.3. Máquina de soporte vectorial

El tercer tipo de modelo que el sistema SageTomo permite entrenar es el conocido como máquina de soporte vectorial (SVM, por sus siglas en inglés). Dado un *dataset* de entrenamiento, cuyos datos tienen N variables predictoras, el objetivo es encontrar un hiperplano de dimensión N que se ajuste a dichos datos. Para ello, deben encontrarse los valores adecuados para los coeficientes que caracterizan al hiperplano.

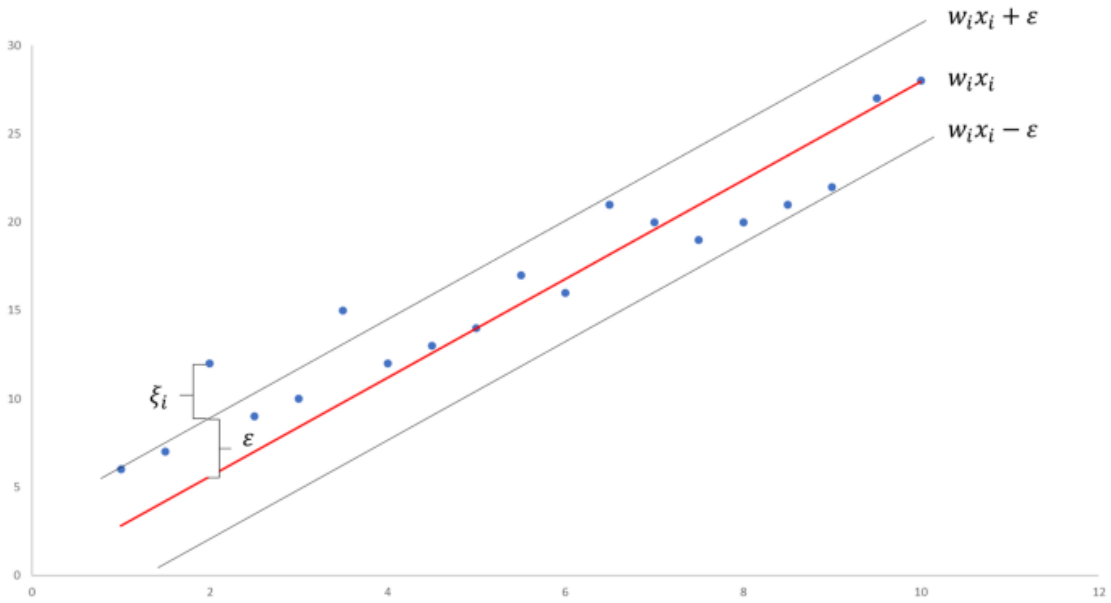

Figura 6.6: Hiperplano en un espacio bidimensional [29]

.

En la figura 6.6 se muestra un espacio bidimensional, de forma que el hiperplano construido es una recta. En la figura se observan ciertos parámetros:

- w es el vector de coeficientes del hiperplano.
- ϵ es el error máximo permitido. Este parámetro permite establecer la precisión deseada para el modelo.
- ξ es la desviación del margen definido.

Para construir el hiperplano adecuado, se debe minimizar la siguiente expresión:

$$\frac{1}{2}||w||^2 + C\sum_{i}^{n}|\xi_i|$$

El parámetro C introducido en la expresión anterior es un parámetro de regularización. Al aumentar el valor de este parámetro, se incrementa la tolerancia a puntos situados fuera del intervalo definido por ϵ. El plano construido debe cumplir la siguiente restricción:

$$|y_i - w_i x_i| \leq \epsilon + |\xi_i|$$

Para que la máquina de soporte vectorial se pueda emplear también con conjuntos de datos que no separables linealmente, se utilizan funciones de *kernel*. Estas funciones proyectan los datos de entrada en un nuevo espacio de dimensión superior, con el objetivo de encontrar un hiperplano que los separe correctamente.

El usuario del sistema SageTomo puede definir los valores para ϵ , ξ y C, así como seleccionar una función de *kernel*. Además, también puede seleccionar o establecer los valores de los siguientes parámetros:

- **Grado**. En el caso de que el usuario seleccione una función de *kernel* polinómica, puede establecer el grado de dicha función.
- ***Gamma***. Es un coeficiente empleado en las siguientes funciones de *kernel*: polinómica, de base radial y sigmoide.
- **Coeficiente 0**. Es el término independiente en la función de *kernel*. Sólo es relevante en las funciones polinómica y sigmoide.

Cabe señalar que, de forma análoga a lo que ocurre con los *random forest*, las máquinas de soporte vectorial sólo permiten realizar predicciones sobre una única variable dependiente. Por tanto, para predecir las 844 conductividades de cada malla, se entrenan 844 modelos, cada uno de los cuales predice una de las conductividades.

En la siguiente figura se muestra como ejemplo la reconstrucción de la imagen de una malla con un único artefacto empleando una SVM.

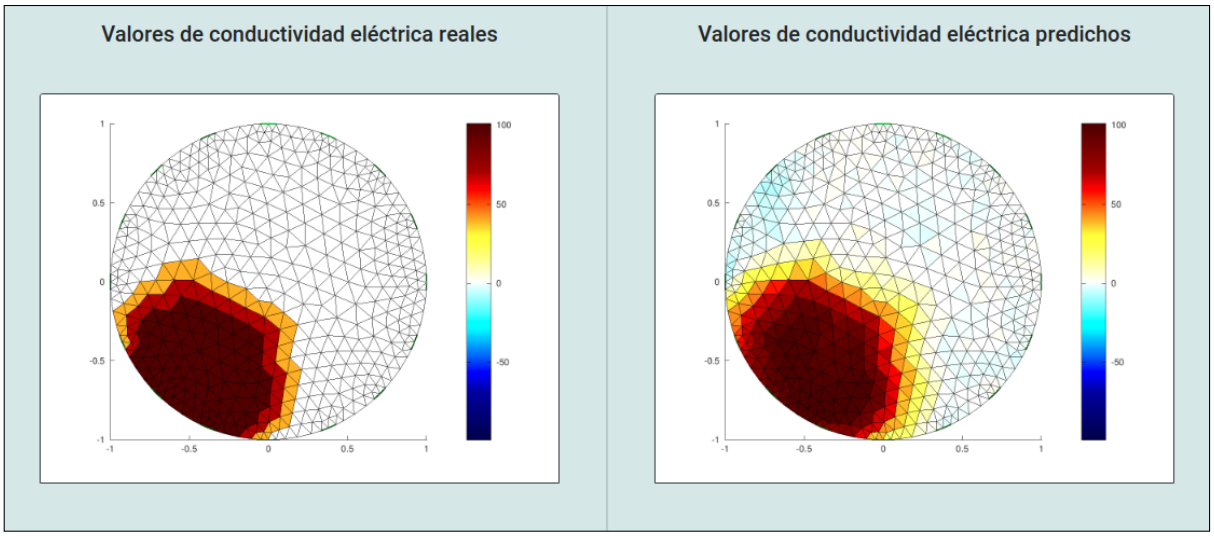

Figura 6.7: Reconstrucción mediante una SVM.

Finalmente, se muestra la comparación de la reconstrucción de una misma malla utilizando los tres tipos de modelos:

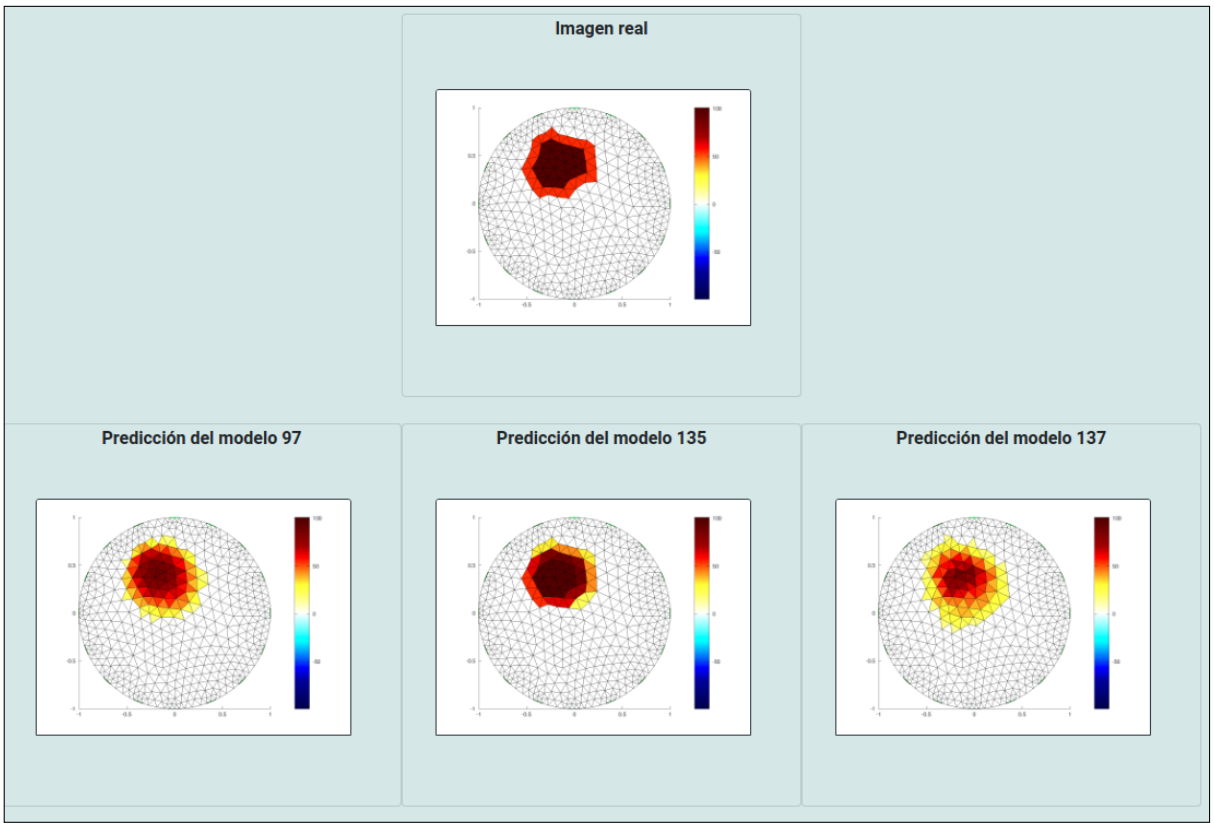

Figura 6.8: Reconstrucción de la imagen de una malla mediante una DNN (modelo 97), un *Random Forest* (modelo 135) y una SVM (modelo 137)

6.3. Postprocesado

Las predicciones realizadas por los modelos pueden tener un cierto ruido. Por tanto, el sistema SageTomo permite aplicar un algoritmo de postprocesado capaz de limpiar dichas predicciones. El algoritmo consiste en asignar un valor de conductividad de fondo (1 S/m) a todos los elementos triangulares de una malla cuyo valor de conductividad sea inferior a un cierto **umbral**. Para seleccionar el umbral óptimo, se utiliza el método de las **curvas ROC** (Receiver Operating Characteristic). Mediante este método, se busca el punto (el umbral) en el que la probabilidad de verdaderos positivos se ve maximizada, mientras que la probabilidad de falsas alarmas (ya sea por falsos positivos o por falsos negativos) se ve minimizada. Se considera que se produce un **verdadero positivo** si, dado un elemento triangular de la malla, la salida del modelo predice un valor de conductividad superior a 1 S/m y el valor real de conductividad del elemento triangular también era superior a 1 S/m. De forma análoga, un **verdadero negativo** se produce cuando el modelo predice un valor igual o inferior a 1 S/m de conductividad para un cierto elemento triangular de una malla y el valor real de

la conductividad del elemento era precisamente 1 S/m.

Capítulo 7

Pruebas y validación

En este apartado se definirán las pruebas a las que se someterá la aplicación SmartTomo. En primer lugar, se realizarán **pruebas de validación de requisitos**. Las pruebas que se definirán constarán de diferentes casos de prueba asociados a los distintos requisitos incluidos en el capítulo *3. Especificación de requisitos y casos de uso*, con el propósito de determinar el cumplimiento de dichos requisitos. En segundo lugar, se analizará la **usabilidad** de la aplicación. Para ello, la aplicación será validada por usuarios reales.

7.1. Pruebas de validación de requisitos

7.1.1. Prueba de gestión de usuarios (P1)

ID	**P1.1**
Entrada	En la ventana de inicio de sesión, se introduce "user1010" como nombre de usuario y "abcdefg10" como contraseña.
Salida	El sistema muestra un error indicando que el usuario o la contraseña no son correctos.
Requisitos asociados	RF1
Necesidades del entorno	No debe existir en el sistema el usuario "user1010"
Resultado	Éxito

Cuadro 7.1: Caso de prueba unitario P1.1.

ID	**P1.2**
Entrada	En la ventana de inicio de sesión, se introduce "user1010" como nombre de usuario y "abcdefg10" como contraseña.
Salida	Se inicia sesión con éxito y el sistema carga la ventana principal de la aplicación.
Requisitos asociados	RF1
Necesidades del entorno	Debe existir en el sistema el usuario "user1010" y su contraseña debe ser "abcdefg10".
Resultado	Éxito

Cuadro 7.2: Caso de prueba unitario P1.2.

ID	**P1.3**
Entrada	Desde la ventana principal de la aplicación, debe pulsarse en el botón *Tu cuenta* y, a continuación, debe pulsarse en la opción *Cerrar sesión*.
Salida	El sistema cierra la sesión y regresa a la pantalla de inicio de sesión.
Requisitos asociados	RF2
Necesidades del entorno	Ninguna.
Resultado	Éxito

Cuadro 7.3: Caso de prueba unitario P1.3.

ID	**P1.4**
Entrada	Desde la ventana de inicio de sesión, se pulsa en la opción *Registrarse*. En el formulario de registro se introducen los siguientes datos: "robert_plant10" como nombre de usuario, "Robert" como nombre real del usuario, "Plant Roosevelt" como apellidos y "contrasenha10" como contraseña.
Salida	El sistema almacena al usuario "robert_plant10" en la base de datos.
Requisitos asociados	RF3
Necesidades del entorno	No debe existir el usuario "robert_plant10" en el sistema.
Resultado	Éxito

Cuadro 7.4: Caso de prueba unitario P1.4.

ID	**P1.5**
Entrada	Desde la ventana de inicio de sesión, se pulsa en la opción *Registrarse*. En el formulario de registro se introducen los siguientes datos: "robert_plant10" como nombre de usuario, "Robert" como nombre real del usuario, "Plant Roosevelt" como apellidos y "contrasenha10" como contraseña.
Salida	El sistema muestra un mensaje de error al usuario, indicándole que ya existe otro usuario con ese *nickname*.
Requisitos asociados	RF3
Necesidades del entorno	Debe existir el usuario "robert_plant10" en el sistema.
Resultado	Éxito

Cuadro 7.5: Caso de prueba unitario P1.5.

ID	**P1.6**
Entrada	Desde la ventana de edición de cuenta, habiendo accedido con la cuenta "robert_plant10", se sustituye el apellido actual por "Plant Aller"
Salida	El sistema realiza la modificación del apellido del usuario "robert_plant10" en la base de datos, estableciendo "Plant Aller" como el nuevo apellido.
Requisitos asociados	RF4
Necesidades del entorno	Debe existir el usuario "robert_plant10" en el sistema y su apellido debe ser "Plant Roosevelt".
Resultado	Éxito

Cuadro 7.6: Caso de prueba unitario P1.6.

ID	**P1.7**
Entrada	Desde la ventana del administrador "admin1", se elimina al usuario "brian_johnson".
Salida	El usuario "brian_johnson" y todos sus modelos y *datasets* privados son eliminados del sistema. El atributo *creador* de los modelos y *datasets* públicos creados o generados por el usuario "brian_johnson" adquiere el valor *Desconocido*.
Requisitos asociados	RF5
Necesidades del entorno	Debe existir el administrador "admin1". Debe existir el usuario "brian_johnson".
Resultado	Éxito

Cuadro 7.7: Caso de prueba unitario P1.7.

7.1.2. Prueba de modelos (P2)

ID	**P2.1**
Entrada	Desde la cuenta del usuario "robert_plant" se inicia el entrenamiento de un modelo de tipo *Red neuronal*. Los parámetros utilizados deben ser los siguientes: ▪ *Dataset*: *dataset* por defecto del sistema con ID igual a 1. ▪ Número de capas ocultas: 1. ▪ Número de neuronas en la capa oculta: 369. ▪ Función de activación para las capas internas: ReLu. ▪ Función de activación para la capa de salida: ReLu. ▪ Función de error: SME. ▪ Nº de épocas: 20. ▪ Entrenamiento por lotes: sí. ▪ Tamaño de los lotes: 64. ▪ *Learning rate*: 0.001. ▪ *Momentum*:0.9 ▪ Métricas: SME y porcentaje de acierto. ▪ Visibilidad: público. ▪ Comentarios adicionales: "Caso de prueba unitario de red neuronal".
Salida	El sistema inicia el entrenamiento de un modelo de tipo *Red neuronal*.
Requisitos asociados	RF6
Necesidades del entorno	Debe existir el usuario "robert_plant".
Resultado	Éxito

Cuadro 7.8: Caso de prueba unitario P2.1.

ID	**P2.2**
Entrada	Desde la cuenta del usuario "robert_plant" se inicia el entrenamiento de un modelo de tipo *Red neuronal*. Los parámetros utilizados deben ser los siguientes: ▪ *Dataset*: *dataset* por defecto del sistema con ID igual a 1. ▪ Número de estimadores: 1. ▪ Profundidad máxima: 10000. ▪ Número mínimo de muestras para división: 0. ▪ Número mínimo de muestras para nodo hoja: 0. ▪ Métricas: SME y porcentaje de acierto. ▪ Visibilidad: público. ▪ Comentarios adicionales: "Caso de prueba unitario de *random forest*".
Salida	El sistema inicia el entrenamiento de un modelo de tipo *Random forest*.
Requisitos asociados	RF7
Necesidades del entorno	Debe existir el usuario "robert_plant".
Resultado	Éxito

Cuadro 7.9: Caso de prueba unitario P2.2.

ID	**P2.3**
Entrada	Desde la cuenta del usuario "robert_plant" se inicia el entrenamiento de un modelo de tipo *Red neuronal*. Los parámetros utilizados deben ser los siguientes: ▪ *Dataset*: *dataset* por defecto del sistema con ID igual a 1. ▪ Kernel: RBF ▪ Grado: 3 ▪ Gamma: Auto ▪ Coeficiente 0: 100 ▪ Tolerancia: 7 ▪ C: 1200000 ▪ Épsilon: 0.1 ▪ Métricas: SME y porcentaje de acierto. ▪ Visibilidad: público. ▪ Comentarios adicionales: "Caso de prueba unitario de *SVM*".
Salida	El sistema inicia el entrenamiento de un modelo de tipo *Máquina de soporte vectorial*.
Requisitos asociados	RF8
Necesidades del entorno	Debe existir el usuario "robert_plant".
Resultado	Éxito

Cuadro 7.10: Caso de prueba unitario P2.3.

ID	**P2.4**
Entrada	Desde la ventana de *Entrenamientos* de la cuenta del usuario "robert_plant", se selecciona la opción de *Guardar modelo* para un modelo cuyo entrenamiento ha finalizado.
Salida	El modelo cuyo entrenamiento ha finalizado se guarda en el sistema.
Requisitos asociados	RF9
Necesidades del entorno	Debe existir el usuario "robert_plant". Debe existir algún entrenamiento iniciado por el usuario "robert_plant" y dicho entrenamiento debe haber finalizado.
Resultado	Éxito

Cuadro 7.11: Caso de prueba unitario P2.4.

ID	**P2.5**
Entrada	Se accede a sección de *Modelos* la cuenta del usuario "robert_plant".
Salida	Se muestra al usuario un modelo de tipo *Red neuronal* creado por él y un modelo de tipo *Random forest* creado por el usuario "brian_johnson".
Requisitos asociados	RF10
Necesidades del entorno	En el sistema deben existir únicamente tres modelos: un modelo de tipo *Red neuronal* cuyo creador debe ser "robert_plant", un modelo de tipo *Random forest* cuyo creador debe ser "brian_johnson" (el modelo debe ser público) y un modelo de tipo *Máquina de soporte vectorial* cuyo creador debe ser "brian_johnson" (el modelo debe ser privado).
Resultado	Éxito

Cuadro 7.12: Caso de prueba unitario P2.5.

ID	**P2.6**
Entrada	En la sección de *Modelos*, utilizando la cuenta del usuario "robert_plant", se selecciona la opción *Eliminar modelo* de un modelo creado por "robert_plant".
Salida	El modelo seleccionado es eliminado del sistema.
Requisitos asociados	RF11
Necesidades del entorno	Debe existir el usuario "robert_plant". Debe existir en el sistema un modelo creado por el usuario "robert_plant".
Resultado	Éxito

Cuadro 7.13: Caso de prueba unitario P2.6.

ID	**P2.7**
Entrada	En la sección de *Modelos*, utilizando la cuenta del usuario "robert_plant", se seleccionan tres modelos: uno de tipo *Red neuronal*, otro de tipo *Random forest* y otro de tipo *Máquina de soporte vectorial*. A continuación, se pulsa en la opción *Comparar modelos*. Se escoge el *dataset* por defecto (el cual tiene ID 1), se elige la opción de postprocesar los resultados y se seleccionan el MSE y el porcentaje de acierto como métricas.
Salida	Se muestran los valores obtenidos de MSE y porcentaje de acierto para cada uno de los tres modelos. Se muestran las matrices de confusión para cada uno de los tres modelos. Se muestran cuatro imágenes: la imagen real de la malla y las reconstrucciones realizadas con cada uno de los tres modelos.
Requisitos asociados	RF12
Necesidades del entorno	Debe existir el usuario "robert_plant". Debe existir en el sistema un modelo público de tipo *Red neuronal*, un modelo público de tipo *Random Forest* y un modelo público de tipo *Máquina de soporte vectorial*.
Resultado	Éxito

Cuadro 7.14: Caso de prueba unitario P2.7.

7.1.3. Prueba de predicción de conductividades y reconstrucción de imágenes (P3)

ID	**P3.1**
Entrada	En la sección de *Reconstrucción de imágenes*, utilizando la cuenta del usuario "robert_plant", se seleccionan un modelo de tipo *Red neuronal*. A continuación, se selecciona el *dataset* por defecto con ID 1, se realiza un filtrado de mallas, estableciendo en 3 el número de artefactos que deben contener las mallas y se selecciona la primera malla de la lista.
Salida	Se muestran dos imágenes: la imagen real de la malla y la imagen reconstruida con el modelo de tipo *Red neuronal*.
Requisitos asociados	RF13
Necesidades del entorno	Debe existir el usuario "robert_plant". Debe existir en el sistema un modelo público de tipo *Red neuronal* y el *dataset* por defecto con identificador 1.
Resultado	Éxito

Cuadro 7.15: Caso de prueba unitario P3.1.

ID	**P3.2**
Entrada	En la sección de *Reconstrucción de imágenes*, utilizando la cuenta del usuario "robert_plant", se selecciona un modelo de tipo *Red neuronal*. A continuación, se selecciona el *dataset* por defecto con ID 1, se realiza un filtrado de mallas, estableciendo en 3 el número de artefactos que deben contener las mallas y se selecciona la primera malla de la lista.
Salida	Se muestran dos imágenes: la imagen real de la malla y la imagen reconstruida con el modelo de tipo *Red neuronal*.
Requisitos asociados	RF13
Necesidades del entorno	Debe existir el usuario "robert_plant". Debe existir en el sistema un modelo público de tipo *Red neuronal* y el *dataset* por defecto con identificador 1.
Resultado	Éxito

Cuadro 7.16: Caso de prueba unitario P3.2.

ID	**P3.3**
Entrada	En la sección de *Reconstrucción de imágenes*, utilizando la cuenta del usuario "robert_plant", se selecciona un modelo de tipo *Random forest*. A continuación, se selecciona el *dataset* por defecto con ID 1, se realiza un filtrado de mallas, estableciendo en 3 el número de artefactos que deben contener las mallas y se selecciona la primera malla de la lista.
Salida	Se muestran dos imágenes: la imagen real de la malla y la imagen reconstruida con el modelo de tipo *Random forest*.
Requisitos asociados	RF13
Necesidades del entorno	Debe existir el usuario "robert_plant". Debe existir en el sistema un modelo público de tipo *Random forest* y el *dataset* por defecto con identificador 1.
Resultado	Éxito

Cuadro 7.17: Caso de prueba unitario P3.3.

ID	**P3.4**
Entrada	En la sección de *Reconstrucción de imágenes*, utilizando la cuenta del usuario "robert_plant", se selecciona un modelo de tipo *Máquina de soporte vectorial*. A continuación, se selecciona el *dataset* por defecto con ID 1, se realiza un filtrado de mallas, estableciendo en 3 el número de artefactos que deben contener las mallas y se selecciona la primera malla de la lista. Una vez hecho esto, se pulsa en *Reconstruir imagen*
Salida	Se muestran dos imágenes: la imagen real de la malla y la imagen reconstruida con el modelo de tipo *Máquina de soporte vectorial*.
Requisitos asociados	RF13
Necesidades del entorno	Debe existir el usuario "robert_plant". Debe existir en el sistema un modelo público de tipo *Máquina de soporte vectorial* y el *dataset* por defecto con identificador 1.
Resultado	Éxito

Cuadro 7.18: Caso de prueba unitario P3.4.

ID	**P3.5**
Entrada	En la sección de *Reconstrucción de imágenes*, utilizando la cuenta del usuario "robert_plant", se selecciona un modelo de tipo *Red neuronal*. A continuación, se selecciona el *dataset* por defecto con ID 1, se realiza un filtrado de mallas, estableciendo en 3 el número de artefactos que deben contener las mallas y se selecciona la primera malla de la lista. Una vez hecho esto, se pulsa en *Reconstruir imagen*. Cuando la imagen haya sido reconstruida, se pulsa en la opción *Exportar reconstrucción*.
Salida	El sistema genera un archivo PDF con la reconstrucción realizada e información sobre ésta.
Requisitos asociados	RF14
Necesidades del entorno	Debe existir el usuario "robert_plant". Debe existir en el sistema un modelo público de tipo *Red neuronal* y el *dataset* por defecto con identificador 1.
Resultado	Éxito

Cuadro 7.19: Caso de prueba unitario P3.5.

ID	**P3.6**
Entrada	En la sección de *Reconstrucción de imágenes*, utilizando la cuenta del usuario "robert_plant", se selecciona un modelo de tipo *Red neuronal*. A continuación, se selecciona el *dataset* por defecto con ID 1, se realiza un filtrado de mallas, estableciendo en 3 el número de artefactos que deben contener las mallas y se selecciona la primera malla de la lista. Una vez hecho esto, se pulsa en *Reconstruir imagen*. Cuando la imagen haya sido reconstruida, se pulsa en la opción *Analizar sección*, manteniendo el valor por defecto para el eje Y (Y = 0).
Salida	El sistema genera el corte de la malla en Y = 0 y muestra la gráfica que representa dicho corte.
Requisitos asociados	RF22
Necesidades del entorno	Debe existir el usuario "robert_plant". Debe existir en el sistema un modelo público de tipo *Red neuronal* y el *dataset* por defecto con identificador 1.
Resultado	Éxito

Cuadro 7.20: Caso de prueba unitario P3.6.

7.1.4. Prueba de *datasets* (P4)

ID	**P4.1**
Entrada	En la sección de *Datasets*, utilizando la cuenta del usuario "robert_plant", se selecciona la opción *Subir dataset*. En el formulario de subida de *dataset*, se seleccionan las siguientes opciones: ▪ Nº de electrodos: 16 electrodos. ▪ Patrón de estimulación: Adyacente. ▪ Tamaño mínimo del radio de los artefactos: 4. ▪ Tamaño máximo del radio de los artefactos: 10. ▪ Normalizado: no. ▪ Visibles: sí. ▪ Semilla: 12345. ▪ *Dataset* seleccionado: se selecciona el fichero "dataset_prueba1.csv".
Salida	El sistema almacena en la base de datos el *dataset*.
Requisitos asociados	RF15
Necesidades del entorno	Debe existir el usuario "robert_plant". Debe existir en el entorno de prueba el fichero "dataset_prueba1.csv", el cual debe contener 600 líneas, cada una de las cuales con 208 + 844 + 1 valores numéricos separados por punto y coma, de forma que el último valor numérico de cada una de las líneas sea un entero perteneciente al intervalo [1,3].
Resultado	Éxito

Cuadro 7.21: Caso de prueba unitario P4.1.

ID	**P4.2**
Entrada	En la sección de *Datasets*, utilizando la cuenta del usuario "robert_plant", se selecciona la opción *Subir dataset*. En el formulario de subida de *dataset*, se seleccionan las siguientes opciones: ▪ Nº de electrodos: 16 electrodos. ▪ Patrón de estimulación: Adyacente. ▪ Tamaño mínimo del radio de los artefactos: 4. ▪ Tamaño máximo del radio de los artefactos: 10. ▪ Normalizado: no. ▪ Visibles: sí. ▪ Semilla: 12345. ▪ *Dataset* seleccionado: se selecciona el fichero "dataset_prueba1.csv".
Salida	El sistema muestra un error al usuario, indicando que el fichero subido no tiene la estructura adecuada.
Requisitos asociados	RF15
Necesidades del entorno	Debe existir el usuario "robert_plant". Debe existir en el entorno de prueba el fichero "dataset_prueba2.csv", el cual debe contener 600 líneas, cada una de las cuales con 208 + 844 + 1 valores numéricos separados por punto y coma, de forma que el último valor numérico de al menos una de las líneas sea el carácter "a".
Resultado	Éxito

Cuadro 7.22: Caso de prueba unitario P4.2.

ID	**P4.3**
Entrada	En la sección de *Datasets*, utilizando la cuenta del usuario "robert_plant", se selecciona la opción *Subir dataset*. En el formulario de subida de *dataset*, se seleccionan las siguientes opciones: ▪ Nº de electrodos: 16 electrodos. ▪ Patrón de estimulación: Adyacente. ▪ Tamaño mínimo del radio de los artefactos: 4. ▪ Tamaño máximo del radio de los artefactos: 10. ▪ Normalizado: no. ▪ Visibles: sí. ▪ Semilla: 12345. ▪ *Dataset* seleccionado: se selecciona el fichero "dataset_prueba1.csv".
Salida	El sistema muestra un error al usuario, indicando que el fichero subido no tiene la estructura adecuada.
Requisitos asociados	RF15
Necesidades del entorno	Debe existir el usuario "robert_plant". Debe existir en el entorno de prueba el fichero "dataset_prueba1.csv", el cual debe contener 600 líneas, cada una de las cuales con 208 + 844 + 1 valores numéricos separados por punto y coma, excepto la última línea, que tendrá 208 + 844 + 2 valores. El último valor numérico de cada una de las líneas será un entero perteneciente al intervalo [1,3].
Resultado	Éxito

Cuadro 7.23: Caso de prueba unitario P4.3.

ID	**P4.4**
Entrada	En la sección de *Datasets*, utilizando la cuenta del usuario "robert_plant", se selecciona la opción *Generar dataset*. En el formulario de generación de *dataset*, se seleccionan las siguientes opciones: ▪ Nº de electrodos: 16 electrodos. ▪ Patrón de estimulación: Adyacente. ▪ Número de cuerpos con un artefacto: 500. ▪ Número de cuerpos con dos artefactos: 200. ▪ Número de cuerpos con tres artefactos: 100. ▪ Tamaño mínimo del radio de los artefactos: 4. ▪ Tamaño máximo del radio de los artefactos: 10. ▪ Normalizado: no. ▪ Visibles: sí. ▪ Semilla: 12345678. ▪ *Dataset* seleccionado: se selecciona el fichero "dataset_prueba1.csv".
Salida	El sistema genera un *dataset* con 500 cuerpos de un artefactos, 200 cuerpos de dos artefactos y 100 cuerpos de tres artefactos.
Requisitos asociados	RF16
Necesidades del entorno	Debe existir el usuario "robert_plant".
Resultado	Éxito

Cuadro 7.24: Caso de prueba unitario P4.4.

ID	**P4.5**
Entrada	Se accede a la sección de *Datasets*, utilizando la cuenta del usuario "robert_plant".
Salida	El sistema muestra los *datasets* con ID 1 e ID 2.
Requisitos asociados	RF17
Necesidades del entorno	Debe existir el usuario "robert_plant". Debe existir el *dataset* con ID 1 y su creador debe ser el usuario el usuario "robert_plant". Debe existir el *dataset* con ID 2, su creador debe ser el usuario el usuario "brian_johnson" y debe tener visibilidad pública. Debe existir el *dataset* con ID 3, su creador debe ser el usuario el usuario "brian_johnson" y debe tener visibilidad privada.
Resultado	Éxito

Cuadro 7.25: Caso de prueba unitario P4.5.

ID	**P4.6**
Entrada	Se accede a la sección de *Datasets*, utilizando la cuenta del usuario "robert_plant". Para el *dataset* con ID 1, se selecciona la opción *Descargar dataset*. A contiuación, se selecciona el directorio en el que se guardará el archivo descargado
Salida	Se descarga el *dataset* con ID 1 en formato CSV en el directorio del equipo del usuario indicado por éste.
Requisitos asociados	RF18
Necesidades del entorno	Debe existir el usuario "robert_plant". Debe existir el *dataset* con ID 1.
Resultado	Éxito

Cuadro 7.26: Caso de prueba unitario P4.6.

ID	**P4.7**
Entrada	Se accede a la sección de *Datasets*, utilizando la cuenta del usuario "robert_plant". Para el *dataset* con ID 5, se selecciona la opción *Eliminar dataset*. A continuación, se confirma la eliminación.
Salida	Se descarga el *dataset* con ID 5 en formato CSV en el directorio del equipo del usuario indicado por éste.
Requisitos asociados	RF19
Necesidades del entorno	Debe existir el usuario "robert_plant". Debe existir el *dataset* con ID 5, su creador debe ser el usuario "robert_plant" y el *dataset* no puede haber sido usado para entrenar ningún modelo del sistema ni puede estar utilizándose para entrenar ningún modelo del sistema.
Resultado	Éxito

Cuadro 7.27: Caso de prueba unitario P4.7.

ID	**P4.8**
Entrada	Se accede a la sección de *Datasets*, utilizando la cuenta del usuario "robert_plant". Para el *dataset* con ID 5, se selecciona la opción *Eliminar dataset*. A continuación, se confirma la eliminación.
Salida	El sistema muestra un error al usuario indicándole que no puede eliminar un *dataset* asociado a un modelo del sistema.
Requisitos asociados	RF19
Necesidades del entorno	Debe existir el usuario "robert_plant". Debe existir el *dataset* con ID 5, su creador debe ser el usuario "robert_plant" y el *dataset* debe estar asociado a un modelo del sistema.
Resultado	Éxito

Cuadro 7.28: Caso de prueba unitario P4.8.

ID	**P4.9**
Entrada	Se accede a la sección de *Datasets*, utilizando la cuenta del usuario "robert_plant". Para el *dataset* con ID 5, se selecciona la opción *Eliminar dataset*. A continuación, se confirma la eliminación.
Salida	El sistema muestra un error al usuario indicándole que no puede eliminar un *dataset* asociado a un modelo del sistema.
Requisitos asociados	RF19
Necesidades del entorno	Debe existir el usuario "robert_plant". Debe existir el *dataset* con ID 6, su creador debe ser el usuario "robert_plant" y el *dataset* no debe estar asociado a ningún modelo del sistema.
Resultado	Éxito

Cuadro 7.29: Caso de prueba unitario P4.9.

7.1.5. Prueba de tareas (P5)

ID	**P5.1**
Entrada	Se accede a la sección de *Entrenamientos*, utilizando la cuenta del usuario "robert_plant".
Salida	El sistema muestra al usuario la lista de sus entrenamientos en curso, integrada únicamente por el modelo con id 7 y las lista de entrenamientos finalizados, integrada únicamente por el modelo con id 8.
Requisitos asociados	RF20
Necesidades del entorno	Debe existir el usuario "robert_plant". Debe existir el usuario "brian_johnson". Debe existir un entrenamiento en curso de un modelo de tipo *Red neuronal* con id 7 iniciado por el usuario "robert_plant". Debe existir un entrenamiento finalizado de un modelo de tipo *Ranfom forest* con id 8, iniciado por el usuario "robert_plant". Debe existir un entrenamiento en curso de un modelo de tipo *Ranfom forest* con id 9, iniciado por el usuario "brian_johnson".
Resultado	Éxito

Cuadro 7.30: Caso de prueba unitario P5.1.

ID	P5.2
Entrada	Se accede a la sección de *Datasets en generación*, utilizando la cuenta del usuario "robert_plant".
Salida	El sistema muestra al usuario la lista de sus *datasets* en generación, integrada únicamente por el *dataset* con id 7 y las lista de *datasets* generados, integrada únicamente por el *dataset* con id 8.
Requisitos asociados	RF21
Necesidades del entorno	Debe existir el usuario "robert_plant". Debe existir el usuario "brian_johnson". Debe existir un *dataset* en generación con id 7 creado por el usuario "robert_plant". Debe existir un *dataset* ya generado con id 8, creado por el usuario "robert_plant". Debe existir un *dataset* en generación con id 9, creado por el usuario "brian_johnson".
Resultado	Éxito

Cuadro 7.31: Caso de prueba unitario P5.2.

7.2. Validación por parte de los usuarios

7.2.1. Cuestionario de usabilidad

Las pruebas de validación por parte de los usuarios consistirán en la evaluación de la usabilidad de la aplicación. Para ello, se ha elaborado un cuestionario, empleando la herramienta de formularios de Google. Cada una de las preguntas del cuestionario pretende analizar aspectos de la aplicación relacionados con su grado de usabilidad. El modelo de respuesta que se le presenta al usuario en ocho de las diez preguntas del cuestionario es el siguiente:

- Muy de acuerdo
- De acuerdo
- Neutral
- En desacuerdo
- Muy en desacuerdo

En nueve de las diez preguntas, el usuario tiene cinco opciones para expresar su grado de acuerdo con la afirmación que se le propone. Sin embargo, en la pregunta 10, el usuario dispone de varias líneas para escribir un breve comentario.

Las preguntas de las que consta el cuestionario son las siguientes:

1. He podido reconstruir fácilmente la imagen de una malla perteneciente a un *dataset*.
2. He logrado subir con facilidad un fichero de mallas al sistema y he podido reconstruir fácilmente la imagen de una de esas mallas.
3. He podido entrenar un modelo de tipo *red neuronal* con facilidad.
4. He conseguido generar un *dataset* sin dificultad.
5. He logrado eliminar un modelo o un *dataset* sin problemas.
6. He podido consultar fácilmente qué tareas tenía en curso.
7. He podido cancelar una tarea con facilidad.
8. En todo momento sabía en qué parte de la aplicación me encontraba.
9. La sección de Ayuda me ha resultado útil.
10. ¿Incluirías algún cambio o mejora en la aplicación?

7.2.2. Resultados obtenidos

La aplicación fue probada por cuatro usuarios, los cuales completaron el cuestionario presentado en el apartado anterior. En la tabla 7.32 se muestran los resultados obtenidos, así como la puntuación media para cada pregunta del cuestionario.

	Usuario 1	**Usuario 2**	**Usuario 3**	**Usuario 4**	**Media por pregunta**
P1	4	5	5	5	4.75
P2	4	5	4	5	4.5
P3	5	5	5	5	5
P4	4	5	5	5	4.75
P5	5	5	5	5	5
P6	5	5	5	5	5
P7	5	5	5	5	5
P8	4	5	5	5	4.75
P9	3	4	5	4	4

Cuadro 7.32: Resultados de las pruebas de validación por parte de los usuarios.

La calificación media obtenida es de 4.75, por lo que se cumple con el requisito no funcional RNF3 (tabla 3.27), dado que la puntuación de usabilidad es superior a 4 puntos.

Respecto a la cuestión 10, en la que se preguntaba al usuario si realizaría algún cambio o mejora, ninguno de los usuarios propuso modificaciones. No obstante, examinando la puntuación media obtenida en cada una de las preguntas, se observa que la pregunta 9, relativa a la utilidad de la sección de *Ayuda*, es la que obtuvo una calificación menor. Por esta razón, se decidió ampliar los textos de ayuda de la aplicación, ofreciendo explicaciones más detalladas sobre el uso del sistema.

Capítulo 8

Conclusiones y posibles ampliaciones

El proyecto propuesto para el TFG pudo ser completado con éxito, a pesar de haber tenido que afrontar los inconvenientes que la crisis del coronavirus ha supuesto. El haber desarrollado el proyecto durante una situación de crisis ha puesto de manifiesto la extrema importancia de las fases de planificación y gestión de riesgos. La situación en la que se desarrolla cualquier proyecto de Ingeniería puede verse afectada por factores externos con los que no se cuenta inicialmente, pero dichos factores deben gestionarse de manera que puedan alcanzarse igualmente los objetivos del proyecto.

Un aspecto característico del TFG realizado es la formación

Durante el transcurso del proyecto se han combinado dos ramas del conocimiento diferentes: la Ingeniería del Software y la Ciencia de Datos. El sistema SageTomo es el resultado de la unión de estas dos disciplinas. Sin embargo, la ejecución de dicha unión de manera satisfactoria presentó ciertas dificultades.

El principal problema afrontado a lo largo del transcurso del proyecto fue la necesidad de trabajar con tecnologías muy diferentes. Tal y como se ha mencionado, se consideró que Python era el lenguaje más apropiado para implementar las funcionalidades asociadas al ámbito del *Machine Learning*. Sin embargo, la herramienta EIDORS para simular las tomografías de impedancia eléctrica es una biblioteca de MATLAB. Por esta razón, fue necesario realizar un análisis para determinar el mejor modo de integrar ambas tecnologías. Por otra parte, también se decidió que la forma de maximizar la utilidad de un sistema de este tipo era el empleo de servicios web. Partiendo de esta decisión, fue necesario encontrar la tecnología adecuada para ofrecer dichos servicios, concluyendo que el *framework* Django REST se adaptaba con creces a las necesidades del proyecto. Por tanto, la principal conclusión extraída de la realización de este trabajo es que el empleo de

tecnologías de características diferentes no impide alcanzar resultados óptimos, siempre y cuando las tecnologías se integren de forma correcta.

Finalmente, cabe destacar que las técnicas de IA utilizadas para la realización de las tomografías de impedancia eléctrica permiten sustituir con éxito a las técnicas analíticas tradicionales, muy costosas en recursos temporales y computacionales. Empleando este tipo de técnicas en un ámbito industrial puede mejorarse la eficiencia de los procesos de análisis del interior de cuerpos. En la introducción de este trabajo se planteó el caso de la industria maderera, en la cual es necesario analizar la distribución de humedad de productos de madera, de forma que si se detecta un producto defectuoso, se descarta todo el lote al que pertenece el producto. Mediante el empleo de las técnicas de IA propuestas en este trabajo, sería posible analizar todos los productos de un lote y descartar únicamente las piezas defectuosas. Éste es un ejemplo más de cómo las técnicas de Inteligencia Artificial tienen el potencial de penetrar en prácticamente todos los ámbitos industriales y económicos, permitiendo que se esté produciendo la denominada Cuarta Revolución Industrial.

8.1. Posibles ampliaciones

En este proyecto se han utilizado mallas de 844 elementos triangulares. El número de elementos triangulares es un factor que depende del número de electrodos utilizados para generar los voltajes. En el sistema SageTomo se utilizan 16 electrodos a la hora de generar un *dataset*. Una ampliación interesante del sistema consistiría en incluir la posibilidad de generar *datasets* eligiendo el número de electrodos que se desea utilizar. Utilizar un mayor número de electrodos implicaría que las mallas estarían divididas en un número mayor de elementos triangulares. En este escenario, se debería estudiar si esta mayor granularidad en las mallas implica un incremento en la precisión de las predicciones de los modelos.

Apéndice A

Manuales técnicos

En este apartado se incluyen las intrucciones necesarias para realizar la instalación y despliegue del sistema SageTomo, así como para el mantenimiento e incorporación de posibles ampliaciones.

A.1. Requerimientos

Para realizar la instalación del sistema SageTomo se requiere un sistema operativo Linux. Se recomienda el empleo de Ubuntu, en su versión 18.04 (o superiores).

A.2. Instalación y despliegue

La instalación del sistema SageTomo requiere la instalación de los dos componentes que integran dicho sistema: el *backend* y el *frontend*. En los dos siguientes apartados se explican cuáles son los pasos que se deben seguir.

A.2.1. Instalación del *backend*

Los pasos a realizar son los siguientes:

1. **Instalar Octave**.

   ```
   $ sudo apt-get install flatpak;

   $ flatpak remote-add --if-not-exists flathub
     https://flathub.org/repo/flathub.flatpakrepo;

   $ flatpak install flathub org.octave.Octave;
   ```

2. **Instalar Python, crear un entorno virtual y activarlo**.

```
$ sudo apt-get install python3-pip;

$ sudo pip3 install virtualenv;

$ source env_sage_tomo/bin/activate
```

3. **Instalar paquetes de Python para SageTomo**. Accede al interior del directorio *sage_tomo* y ejecuta desde un terminal el siguiente comando:

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

4. **Instalar NetGen**. Ejecuta los siguientes comandos desde un terminal:

```
$ sudo apt-add-repository universe
$ sudo add-apt-repository ppa:ngsolve/ngsolve
$ sudo apt-get update
$ sudo apt-get install ngsolve
```

A.2.2. Instalación del *frontend*

Para la instalación del *frontend*, ejecuta los siguientes comandos:

```
$ sudo apt update

$ sudo apt install nodejs

$ sudo apt install npm

$ npm install react-bootstrap bootstrap

$ npm install --save bootstrap@latest

$ npm install --save react-router-dom

$ npm install axios

$ npm install formik

$ npm install yup --save
```

A.2.3. Despliegue

Despliegue del *backend*

Desde la carpeta *sage_tomo*, ejecuta el script de despliegue del *backend*, mediante el siguiente comando:

```
$ ./ejecutar_backend.sh
```

Despliegue del *frontend*

Desde la carpeta *sage_tomo*, ejecuta el script de despliegue del *frontend*, mediante el siguiente comando:

```
$ ./ejecutar_frontend.sh
```

Una vez que el *backend* y el *frontend* se encuentren desplegados, se podrá acceder a la aplicación mediante un navegador web, a través de la siguiente URL: http://localhost:3000. Se utiliza el puerto 3000 puesto que es el puerto por defecto empleado por React (*framework* empleado para el desarrollo del *frontend*).

Detención del sistema

Desde la carpeta *sage_tomo*, ejecuta el script de detención de todo el sistema, mediante el siguiente comando:

```
$ ./detener_sistema.sh
```

A.3. Configuración

En este apartado se describirá cómo configurar algunos aspectos relevantes del sistema. Dado que el *frontend* se limita consumir los servicios ofrecidos por el *backend*, la configuración de los aspectos importantes del sistema se realiza desde el *backend*.

En la carpeta *backend/tomo_backend* se encuentra el fichero *settings.py*. Desde este fichero se pueden configurar las siguientes variables relevantes para el sistema:

- **ALLOWED_HOSTS**: esta variable permite especificar una lista de nombres de dominio a los que responderá el servidor.
- **DEBUG**: admite los valores True y False. En caso de que se le asigne el valor True, cuando se produzca un error, se mostrará información detallada en el navegador. Se recomienda asignar el valor True únicamente durante el desarrollo del sistema, evitando emplear dicho valor en la fase de producción.

- **INSTALLED_APPS**: incluye el listado de aplicaciones que utiliza el *backend*. Por ejemplo, una de las aplicaciones incluidas es *dj_rest_auth*, empleada para la autenticación de los usuarios. Siempre que se instale una nueva aplicación que vaya a ser usada en el *backend*, debe incluirse en esta lista.
- **DATABASES**: es un diccionario en el que se especifican las bases de datos a las que puede conectarse el *backend*, así como la información necesaria para poder efectuar estas conexiones.
- **TIME_ZONE**: permite especificar la zona horaria que se desea utilizar.
- **REST_FRAMEWORK**: permite especificar algunas características de la API REST. Se ha empleado esta variable para definir los permisos por defecto y el modo de autenticación.

Apéndice B

Manual de usuario

El manual de usuario contiene todas las explicaciones e instrucciones necesarias para poder hacer uso de la aplicación SmartTomo. La aplicación será usada por expertos en el dominio del problema. Por tanto, en el manual de usuario se explicará cómo utilizar la aplicación, sin ofrecer explicaciones sobre los aspectos teóricos relacionados con las tomografías de impedancia eléctrica ni con las técnicas de *Machine Learning*. Por otro lado, cabe señalar que las explicaciones que se ofrecen en el siguiente manual han sido incorporadas en la propia aplicación, mostrándose al usuario diferentes secciones del manual en función de la ventana de la aplicación en la que se encuentre en el momento en el que acuda a la ayuda.

B.1. Inicio de sesión y registro

Para poder hacer uso de la aplicación SmartTomo, el usuario debe **iniciar sesión** previamente con una cuenta de usuario, a través de la ventana de inicio de la aplicación. Para ello, debe introducir sus credenciales. En caso de no disponer de una cuenta, puede crear una nueva clicando en la opción *Regístrate*. Clicando en dicha opción, accederá a la ventana de registro, dónde deberá introducir la información de la cuenta que desea crear. Una vez que el usuario haya introducido su información y pulse en el botón *Registrarse* se le enviará un correo de confirmación a la cuenta de correo electrónico introducida.

B.2. Página principal

Una vez que el usuario acceda a la aplicación, se le mostrará la **ventana principal**. En esta ventana, podrá ver cuatro grandes opciones: *Reconstrucción de imágenes*, *Modelos*, *Datasets*, *Tareas*. Cada una de estas cuatro opciones dispone de un botón *Acceder*, de manera que si pulsa en uno de estos botones será redirigido a la sección correspondiente.

En la parte superior de ésta y de todas las ventanas de la aplicación el usuario dispondrá de un **menú**, con las opciones *Inicio*, *Ayuda* y *Tu cuenta*. Pulsando en *Inicio* volverá a la página principal que se describe en esta sección. Pulsando en *Ayuda* se abrirá un desplegable con la información más relevante sobre la ventana en la que se encuentra. Pulsando en *Tu cuenta* se le mostrarán dos opciones: *Editar cuenta* y *Cerrar sesión*. Pulsando en la primera opción podrá modificar algunos de los datos de su cuenta, mientras que pulsando en *Cerrar sesión* podrá salir de la sesión actual y regresar a la pantalla de inicio de sesión.

B.3. Reconstrucción de imágenes

B.3.1. Selección de modelo

Al acceder a la sección de *Reconstrucción de imágenes*, lo primero que se mostrará al usuario es una ventana de **selección de modelo**. Los modelos que se le mostrarán serán los públicos o aquéllos que hayan sido creados por el propio usuario. Una vez que haya seleccionado un modelo, si pulsa en el botón *Seleccionar modelo*, accederá a la ventana de selección del tipo de operación que desea realizar. Si el usuario no selecciona ningún modelo, el sistema no le permitirá acceder a la siguiente ventana y le informará de la situación. En la ventana de selección del tipo de operación se le muestran al usuario dos posibilidades. El usuario tiene la opción de analizar mallas de los *datasets* disponibles o de realizar una predicción de conductividades a partir de un fichero de voltajes que debe subir al sistema.

B.3.2. Reconstruir la imagen de una malla

En la ventana de **reconstrucción de imágenes**, el usuario verá una **lista de *datasets*** y una **lista de mallas** correspondientes al *dataset* seleccionado en ese momento. Por defecto, el *dataset* seleccionado es el de menor identificador. El usuario podrá cambiar el *dataset* seleccionado escogiendo otro de la lista y pulsando en *Cambiar dataset*. Por otra parte, el usuario tiene la posibilidad de filtrar las mallas de un determinado *dataset*, en función del número de artefactos que éstas contengan. Para ello, en el lado izquierdo de la ventana, dispone de un desplegable en el que puede elegir el número de artefactos de las mallas que desea ver, de forma que pulsando en *Filtrar mallas* se le mostrarán únicamente las mallas del *dataset* seleccionado que contengan ese número de artefactos. Por defecto, el número seleccionado es 1.

Una vez que el usuario haya seleccionado una malla, debe pulsar en *Reconstruir imagen* para generar la imagen de la malla elegida. La imagen tardará varios segundos en generarse. Cuando la generación termine, se le mostrarán dos imágenes. La imagen de la izquierda se corresponde con la **imagen real** de la malla seleccionada, mientras que la imagen de la derecha es la **imagen reconstruida**

mediante el modelo de *Machine Learning* seleccionado por el propio usuario. Los colores de cada uno de los triángulos que componen una malla varían en función del valor de impedancia eléctrica que tengan asociado.

B.3.3. Predicción de conductividades a partir de un fichero de voltajes

Si el usuario accede a la sección de *Predicción de conductividades a partir de un conjunto de voltajes* se le redirigirá a una ventana desde la que podrá subir un fichero de voltajes. El fichero debe estar en formato CSV y en cada línea debe contener los voltajes de una malla, separados por punto y coma. En el subdirectorio *ficheros_prueba* del directorio *sage_tomo* se incluye el fichero *conjunto_voltajes.csv*, el cual puede ser utilizado para probar esta funcionalidad.

Una vez que suba el fichero podrá realizar las predicciones de las conductividades. Estas predicciones se le mostrarán en una nueva ventana. El usuario podrá reconstruir la imagen de cada una de las mallas cuyas conductividades se han predicho.

B.4. Modelos

Al acceder a la ventana de *Modelos* desde la ventana principal, se mostrará un listado de los modelos públicos o que hayan sido creados por el usuario. Podrá filtrar los modelos en función del tipo de modelo (DNN, *Random Forest* o SVM), de la fecha de creación y en función de si han sido creados o no por él. Para cada uno de los modelos de la lista, el usuario tendrá la opción de consultar información detallada de los modelos, pulsando en *Ver detalles*. Por otra parte, también podrá eliminar aquellos modelos que hayan sido entrenados por él y que sean privados, pulsando en la opción *Eliminar* del modelo correspondiente. Debajo del listado de modelos se muestran dos opciones: *Comparar modelos* y *Entrenar nuevo modelo*.

B.4.1. Entrenar modelo

Al pulsar en la ventana anterior en la opción de *Entrenar nuevo modelo*, se cargará una nueva ventana en la que se puede **seleccionar el tipo de modelo** que se desea entrenar. Se puede seleccionar uno de los siguientes tipos: *Red neuronal*, *Random Forest* o *Máquina de soporte vectorial*.

Una vez que se seleccione el tipo de modelo, se accederá a una ventana en la que el usuario deberá **seleccionar o establecer ciertas características para el modelo elegido**. Todas ellas son obligatorias con la excepción del campo de

Comentarios adicionales, el cual puede dejar en blanco. Por otra parte, se debe tener en cuenta que en la lista de métricas a seleccionar, el error cuadrático medio y el porcentaje de acierto son métricas obligatorias que aparecerán marcadas por defecto y no podrán ser desmarcadas. Cuando se haya seleccionado o establecido todos los parámetros del modelo, el usuario debe pulsar en *Iniciar entrenamiento* para comenzar a entrenar el modelo. Al pulsar en este botón, será redirigido a la ventana de *Entrenamientos*, donde verá una lista con los entrenamientos en curso (incluido el entrenamiento del modelo que acaba de iniciar) y los entrenamientos finalizados. Consúltese el apartado *B.6. Tareas* para más información sobre esta ventana.

B.4.2. Comparar modelos

En la ventana de *Modelos*, el usuario tendrá la posibilidad de seleccionar hasta cuatro modelos para compararlos entre sí. El sistema no le permitirá seleccionar más de cuatro modelos. Una vez que los haya seleccionado, pulsando en *Comparar modelos*, accederá a una ventana en la que podrá definir cómo se realizará la comparación. En esta ventana, el usuario podrá determinar lo siguiente:

- El *dataset* mediante el cual desea evaluar los modelos.
- Si desea postprocesar o no las predicciones realizar por los modelos.
- Las métricas con las que se evaluarán los modelos.

La comparación de los modelos consistirá en realizar la predicción de los valores de impedancia de todas las mallas del *dataset* seleccionado y evaluar los resultados de las predicciones mediante las métricas indicadas por el usuario. Si el usuario indica que desea postprocesar el resultado, se asignará el valor de *background* (1 S/M) a todas las celdas de cada una de las mallas con un valor de impedancia igual o inferior al umbral calculado mediante curvas ROC cuando se entrenó el modelo. Respecto a las métricas a seleccionar, el usuario tiene la posibilidad de elegir cualquier métrica, independientemente de que ésta se haya utilizado o no para entrenar el modelo. Cuando el usuario seleccionado los parámetros anteriores, debe pulsar en el botón *Comparar modelos* para iniciar la comparación.

Una vez que la comparación termine, el usuario podrá visualizar los resultados obtenidos por cada uno de los modelos para cada métrica. Si entre las métricas elegidas se encuentra el porcentaje de acierto, también se mostrará la matriz de confusión obtenida por cada modelo. Por otra parte, se elegirá aleatoriamente una malla del *dataset* seleccionado y se reconstruirá la imagen real de esa malla, así como las imágenes de las predicciones realizadas por cada uno de los modelos.

B.5. *Datasets*

Al acceder a la ventana de *Datasets* desde la ventana principal, se mostrará un **listado de los *datasets*** públicos o de los que hayan sido creados por el usuario. Para cada uno de los *datasets*, el usuario tendrá la posibilidad de **consultar sus datalles** o de **descargarlo**, pulsando en *Ver detalles* y en *Descargar*, respectivamente. En el caso de la descarga, el *dataset* se descargará en formato CSV.

Debajo del listado de *dataset*, hay dos botones: *Subir dataset* y *Generar dataset*, que permiten al usuario acceder a las secciones de subida y generación de *datasets*, respectivamente.

B.5.1. Subida de dataset

En esta sección, el usuario tiene la posibilidad de **subir al sistema un *dataset* de su equipo**, especificando ciertas características del *dataset*. El *dataset* subido debe ser un archivo CSV. Cada una de las de las líneas del *dataset* debe contener los siguientes valores asociados a una malla y separados por punto y coma:

- Voltajes.
- Impedancias.
- Número de artefactos que contiene la malla.

Si el *dataset* subido por el usuario es un fichero CSV o no tiene la estructura adecuada, el sistema mostrará al usuario un mensaje de error y el *dataset* no será subido. En el subdirectorio *ficheros_prueba* del directorio *sage_tomo* se incluye el fichero *dataset_prueba.csv*, el cual puede ser utilizado para probar esta funcionalidad.

B.5.2. Generación de *dataset*

En esta sección, el usuario puede **crear un nuevo *dataset***, generando las mallas de forma aleatoria. El usuario puede determinar diferentes parámetros del *dataset*, como el número de mallas que contengan uno, dos y tres artefactos. Cabe destacar que el parámetro *Semilla* se utilizará para determinar cómo se desordena un *dataset* para el entrenamiento, de forma que los experimentos puedan ser repetibles.

Cuando el usuario haya establecido los parámetros que desea, debe pulsar en *Generar dataset* para iniciar la generación del *dataset*. Una vez pulsado, el usuario será redirigido a la ventana de tareas relacionadas con la generación de *datasets*. Consúltese el apartado *B.6. Tareas* para más información sobre esta ventana.

B.6. Tareas

Cuando el usuario accede a esta sección desde la ventana principal, se le muestra una ventana con dos opciones: *Entrenamiento de modelos* y *Subida y generación de datasets*. Pulsando en el botón *Acceder* de la primera opción, el usuario será redirigido a la ventana de entrenamientos en curso y, análogamente, pulsando en el botón *Acceder* de la segunda opción, el usuario entrará en la ventana de *datasets* en generación.

Cada una de estas ventanas contiene dos listados. El listado superior incluye los **modelos que se encuentran en entrenamiento** y los ***datasets* que se encuentran en generación**, respectivamente. El listado inferior de cada una de estas ventanas incluye los **modelos y *datasets* que ya han sido entrenados o generados**. El usuario recibirá un correo electrónico cada vez que finalice el entrenamiento de un modelo o la generación de un *dataset*. En el listado inferior, el usuario tiene la posibilidad de guardar o de descartar los modelos entrenados y los *datasets* generados. Para ello, debe pulsar en las opciones *Guardar modelo*/Guardar dataset o *Descartar modelo*/Descartar dataset.

Bibliografía

[1] Fernández-Fuentes, X.; Mera, D.; Gómez, A.; Vidal-Franco, I. Towards a Fast and Accurate EIT Inverse Problem Solver: A Machine Learning Approach', *Electronics* 2018, 7, 422.

[2] Project Management Institute, *Guía de los fundamentos para la dirección de proyectos. Guía del PMBOK*, 6ª edición, Newtown Square, 2017.

[3] Instrucción 4/2020 de la Secretaría Xeral de la USC. (*https://www.usc.gal/es/goberno/secxeral/instrucions.html*). Consultado el 11 de junio de 2020.

[4] Guía salarial del sector TIC en Galicia. 2015-2016. (*https://vdocuments.site/guia-salarial-sector-ti-galicia-2015-2016.html*). Consultado el 15 de junio de 2020.

[5] XIX Convenio colectivo del sector de empresas de ingeniería y oficinas de estudios técnicos. (*https://www.boe.es/eli/es/res/2019/10/07/(8)*). Consultado el 15 de junio de 2020.

[6] Tabla de amortización lineal. (*https://cutt.ly/MgJ2uvV*). Consultado el 15 de junio de 2020.

[7] Ian Sommerville, *Ingeniería de Software*, 9ª edición, Ed. Addison Wesley, Madrid. 2005.

[8] Django REST. (*https://www.django-rest-framework.org/*). Consultado el 1 de julio de 2020.

[9] React. (*https://es.reactjs.org/*). Consultado el 5 de octubre de 2020.

[10] Axios. (*https://github.com/axios/axios*). Consultado el 5 de octubre de 2020.

[11] Keras. (*https://keras.io/*). Consultado el 2 de octubre de 2020.

[12] TensorFlow. (*https://www.tensorflow.org/*). Consultado el 2 de octubre de 2020.

[13] EIDORS. (*http://eidors3d.sourceforge.net/*). Consultado el 10 de septiembre de 2020.

[14] Oct2py. (*https://pypi.org/project/oct2py/*). Consultado el 10 de septiembre de 2020.

[15] NetGen. (*https://ngsolve.org/*). Consultado el 10 de septiembre de 2020.

[16] Celery. (*https://docs.celeryproject.org/en/stable/getting-started/introduction.html*). Consultado el 2 de octubre de 2020.

[17] Bootstrap. (*https://getbootstrap.com/*). Consultado el 2 de octubre de 2020.

[18] Visual Studio Code. (*https://code.visualstudio.com/*). Consultado el 5 de octubre de 2020.

[19] Project Libre. (*https://www.projectlibre.com/*). Consultado el 10 de junio de 2020.

[20] Star UML. (*http://staruml.io/*). Consultado el 15 de julio de 2020.

[21] Diagrams.net. (*https://www.diagrams.net/*). Consultado el 15 de julio de 2020.

[22] NinjaMock. (*https://ninjamock.com/*). Consultado el 15 de julio de 2020.

[23] Overleaf. (*https://es.overleaf.com/*). Consultado el 20 de octubre de 2020.

[24] Arun Ravindran, *Django Design Patterns and Bes Practices*, 2ª edición, Packt, Birmingham-Mumbai, 2018.

[25] Gastón C. Hillar, *Django RESTful Web Services*, 1ª edición, Packt, Birmingham-Mumbai, 2018.

[26] Servicios web según WC3. (*https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice*). Consultado el 27 de agosto de 2020.

[27] Principios heurísticos de Jacob Nielsen. (*https://www.nngroup.com/articles/ten-usability-heuristics/*). Consultado el 27 de agosto de 2020.

[28] Arquitectura de una red neuronal. (*https://www.researchgate.net/figure/Architecture-ofmultilayerartificialneuralnetworkwitherrorbackpropagation_fig3_329216193*). Consultado el 15 de octubre de 2020.

[29] Hiperplano en un espacio bidimensional. (*https://towardsdatascience.com/anintroductiontosupport-vectorregressionsvra3ebc1672c2*). Consultado el 15 de octubre de 2020.