



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería
Informática**

**Visión artificial y aprendizaje
profundo para la estimación
de medidas en radiografías
dentales**



Presentado por Álgvar San Martín Liendo
en Universidad de Burgos
el 13 de octubre de 2023

Tutores: Dr. César Ignacio García Osorio
José Miguel Ramírez Sanz

Índice general

Índice general	i
Índice de figuras	iii
Índice de tablas	iv
Apéndice A Plan de Proyecto Software	1
A.1. Introducción	1
A.2. Planificación temporal	1
A.3. Estudio de viabilidad	8
Apéndice B Especificación de Requisitos	11
B.1. Introducción	11
B.2. Objetivos generales	11
B.3. Catálogo de requisitos	11
B.4. Usuarios	13
B.5. Especificación de requisitos	13
Apéndice C Especificación de diseño	25
C.1. Introducción	25
C.2. Diseño de datos	25
C.3. Diseño procedimental	27
C.4. Diseño arquitectónico	28
Apéndice D Documentación técnica de programación	37
D.1. Introducción	37
D.2. Estructura de directorios	37

D.3. Manual del programador	44
D.4. Compilación, instalación y ejecución del proyecto	45
D.5. Metodología del back-end	46
D.6. Metodología del front-end	49
D.7. Pruebas del sistema	51
Apéndice E Documentación de usuario	55
E.1. Introducción	55
E.2. Requisitos de usuarios	55
E.3. Instalación	56
E.4. Manual del usuario	56
Bibliografía	65

Índice de figuras

B.1. Diagrama de casos de uso	14
C.1. Archivos del conjunto de datos segmentación	26
C.2. Estructura de un archivo JSON con estructura <i>COCO</i>	30
C.3. Diagrama entidad-relación	31
C.4. Diagrama relacional	32
C.5. Secuencias en la segmentación y medición de la radiografía	33
C.6. Secuencias de la creación de un modelo	34
C.7. Secuencias de la creación de un modelo	34
C.8. Diagrama del proceso de segmentación y medición.	35
C.9. Arquitectura de 3 capas de la API REST.	36
D.1. Estructura de directorios del proyecto	37
D.2. Estructura de directorios de la API REST	41
D.3. Estructura de directorios del front-end Vue	43
E.1. Ventana de login	57
E.2. Opción de logout	58
E.3. Opción de logout en el menú	59
E.4. Menú superior izquierdo	60
E.5. Función de radiografías en el menú	60
E.6. Ventana de radiografía	61
E.7. Opciones para la gestión de modelos	62
E.8. Listas de modelos	63
E.9. Formulario de creación de modelo	64

Índice de tablas

A.1. Costes totales de funcionamiento	9
B.1. CU-1 Medir dientes de una radiografía.	15
B.2. CU-1.1 Segmentar los dientes.	16
B.3. CU-1.2 Medir cada diente.	17
B.4. CU-2 Subir modelo.	18
B.5. CU-2.1 Subir modelo de segmentación.	19
B.6. CU-2.2 Subir modelo de medición.	20
B.7. CU-3 Seleccionar modelo.	21
B.8. CU-3.1 Seleccionar modelo de segmentación.	22
B.9. CU-3.2 Seleccionar modelo de medición.	23
D.1. Test-1 Requerimiento de autenticación	52
D.2. Test-2 Permisos necesarios	53
D.3. Test-3 Inanición de modelos	54

Apéndice A

Plan de Proyecto Software

A.1. Introducción

Esta sección está dedicada a documentar el progreso del proyecto a lo largo del tiempo y a analizar su viabilidad en términos económicos y legales.

Modelo de cascada

Se ha elegido la metodología en cascada para planificar el trabajo, ya que es un enfoque más sencillo y directo. Los métodos ágiles, aunque son más flexibles, pueden ser más complejos y requieren un equipo más grande. En este caso, el equipo está formado por un solo desarrollador, por lo que la opción más adecuada es la metodología en cascada.

A.2. Planificación temporal

Se pueden diferenciar dos etapas en el desarrollo del software: la primera, desde octubre hasta julio, engloba el desarrollo de los modelos de *machine learning* y desde julio a septiembre, el desarrollo de la aplicación web.

Reunión 1: 21/10/22

Se realiza la primera reunión con los tutores, se comienza a buscar información a cerca del problema y se comprueba el trabajo anterior (X-Teeth).

Reunión 2: 26/10/22

Se repasan las partes del trabajo anterior que se consideraban confusas. Se discuten los problemas de instalación de Detectron 2 en los equipos Gamma y Beta debido a las versiones de CUDA.

Reunión 3: 4/11/22

Se produce la primera reunión con el Odontólogo donde se le exponen las potenciales problemáticas y se buscan obtener indicadores que nos ayuden a la hora de estimar la medida de los dientes. Finaliza con la resolución de las dudas planteadas. Hasta la siguiente reunión se continúa buscando información sobre las endodoncias y reproduciendo los pasos del trabajo anterior.

Reunión 4: 11/11/22

Se transmiten los problemas para la instalación de las librerías requeridas por Detectron, especialmente con CUDA, que causan la imposibilidad de instalarlo mediante PIP y la compatibilidad con la versión estándar de Python. Se intentan solventar los problemas usando un contenedor docker; sin embargo, el desarrollo sobre este es complicado.

Reunión 5: 18/11/22

Se han encontrado inconsistencias entre la documentación y la generación de los conjuntos de datos, se tienen que rehacer los *notebook* de creación de los conjuntos de datos, además de las herramientas de entrenamiento de Detectron.

Reunión 6: 25/11/22

Se muestran los avances en la instalación de Detectron usando Conda y las dificultades a la hora de replicar los resultados anteriores con los scripts nuevos, una exploración de los archivos antiguos del servidor revela un error del trabajo anterior que provoca que existan las mismas imágenes en los conjuntos de entrenamiento y test.

Reunión 7: 2/12/22

Se plantan soluciones a las problemáticas anteriores, puesto que los resultados de los entrenamientos no son nada satisfactorios. Se busca información adicional para resolver el problema.

Reunión 8: 9/12/22

Se muestran los escasos avances en segmentación y se plantea la posibilidad de que el etiquetado de un único diente por radiografía sea insuficiente.

Reunión 9: 16/12/22

No se han realizado avances importantes para el trabajo debido al comienzo de los exámenes de 1^a convocatoria. Se corrigen los notebooks previos.

Reunión 10: 13/1/23

Durante el transcurso de las vacaciones de Navidad se han etiquetado a mano todas las imágenes del conjunto nuevo y se han realizado entrenamientos, esta vez con un resultado satisfactorio. Se discute sobre el método para obtener automáticamente el diente correcto en radiografías con múltiples en los que no está claro cuál es el central. Se sopesa la posibilidad de utilizar otras herramientas de segmentación de imágenes como YOLO o U-nets dependiendo del tiempo disponible.

Reunión 11: 10/2/23

No ha sido posible la implementación de las herramientas planteadas en la reunión anterior, se da por finalizada la etapa de segmentación de imágenes al haberse conseguido un modelo satisfactorio con Detectron, se ha comprobado sobre imágenes de otra máquina de radiografías y el modelo generaliza con éxito. Se comienza a preparar la etapa de medición de los dientes segmentados.

Reunión 12: 24/2/23

Se muestran los avances en la generación del conjunto de datos orientado a la regresión, las imágenes se recortan por su caja para obtener una nueva imagen únicamente con el diente deseado. Para la próxima reunión se deberá

buscar un modelo de *deep learning* apto para el problema y usar este conjunto para crear un modelo de regresión.

Reunión 13: 14/3/23

Se muestran los resultados de los primeros experimentos con una CNN sencilla, usado una única neurona de salida, son mejores que el resultado de un clasificador de media, pero están muy por encima del >1 mm deseado. Para la siguiente etapa se deben buscar arquitecturas que nos ayuden a mejorar este resultado, además de mejorar el entrenamiento mediante *image augmentation*.

Reunión 14: 28/3/23

Se mejoran los resultados respecto a la reunión anterior mediante el uso de arquitecturas predefinidas de CNN como ResNet. Pero los resultados siguen sin estar por debajo de 1 mm, por lo que para la siguiente reunión se experimenta con distintas arquitecturas y tratamientos de imagen.

Reunión 15: 14/4/23

Los resultados siguen estancados a pesar de la experimentación con *image augmentation* y el uso de otras arquitecturas. Se sigue intentando mejorar el proceso hasta la siguiente reunión.

Reunión 16: 21/4/23

Se consiguen resultados mucho mejores modificando la técnica de recorte a la hora de generar el conjunto de datos, en vez de recortar la caja de la detección, se recorta por el contorno del diente. El error medio desciende a menos de 1mm, pero hay dudas sobre la validez de los resultados debido al uso del conjunto de test en la parada temprana.

Reunión 17: 5/5/23

Al revisar el proceso de entrenamiento se confirma que ha sido erróneo, produce *overfit*, y debe modificarse. Para la siguiente reunión se debe distribuir el conjunto de datos de tal forma que los datos de validación no se encuentren en test.

Reunión 18: 19/5/23

Se implementa el testeo correcto de los resultados sobre una sola partición del conjunto, el resultado es peor de lo que se esperaba, con errores máximos de 4 mm, y hay que volver a buscar métodos para reducirlos. Para la próxima reunión también se deben almacenar los modelos previos en caso de que se quieran comparar y se tiene que elaborar un *noteboook* que permita cargarlos y comprobar su rendimiento.

Reunión 19: 26/5/23

Se presenta el *notebook* con diagramas del entrenamiento de los modelos, estos ayudan a la visualización de los resultados, pero se consideran insuficientes, deben ser ampliados en este periodo. En cuanto a los resultados, estos siguen sin ser satisfactorios porque se encuentran de media por encima de 1 mm.

Reunión 20: 12/6/23

En este periodo se amplía la visualización de los resultados de los entrenamientos para hacer comparaciones con distintas particiones de datos, lo que muestra la necesidad de implementar validación cruzada para obtener resultados realmente precisos.

Reunión 21: 19/6/23

Se ha aplicado la validación cruzada y se ha actualizado la visualización de los entrenamientos para adaptarse a los nuevos resultados, además, se ha implementado el proceso completo desde la segmentación a la medición de los dientes individuales.

Reunión 22: 3/7/23

Desde la reunión anterior se han obtenido unos resultados mejores, usando la máscara binaria del diente, pero los resultados siguen sin cumplir el objetivo del proyecto. También se ha completado la documentación de la memoria. Al considerarse los resultados insatisfactorios, se decide ampliar la entrega para mejorar la documentación y crear una aplicación que aplique los modelos elaborados.

Independiente 1: 10/7/23

Antes de comenzar la aplicación se han repasado el apartado de segmentación:

- Después de volver a investigar en busca de radiografías, se elabora un conjunto de datos de segmentación más amplio con radiografías similares.
- Se ha generado un modelo de segmentación nuevo usando YOLO, que no había dado tiempo durante el primer cuatrimestre.

Una vez terminado se ponen en claro los requisitos de la aplicación, debe ser mínima, para ajustarse a las horas de trabajo planeadas. Se fijan 2 características principales: la medición de los dientes y la gestión de los modelos con los que se hacen las mediciones.

Independiente 2: 16/7/23

Ya se tiene el primer borrador de la estructura, se procede a buscar un entorno para desarrollarlo, que se ajuste al modelo de API REST y cliente. Inicialmente, se pretendía usar Flask para la API y Angular para el cliente, pero se encuentra la alternativa de FastAPI, que es una capa sobre Flask que incluye documentación automática, y Vue, que es similar a Angular, pero como se ha encontrado una [plantilla](https://github.com/rafsaf/minimal-fastapi-postgres-template)¹ que implementa la base de datos, autenticación y los contenedores Docker se opta por esta opción. Se comienza a trabajar en la API.

Independiente 3: 25/7/23

Aparece la necesidad de encontrar una manera de almacenar distintos modelos para varias tareas en una plataforma, junto con las librerías que usan. Para evitar tener que instalar Pytorch, Detectron y YOLO se busca un marco común y se encuentra el estándar [ONNX](https://onnx.ai/)² como método de estandarizar todo el almacenaje.

¹<https://github.com/rafsaf/minimal-fastapi-postgres-template>

²<https://onnx.ai/>

Independiente 4: 30/7/23

Se ha completado la definición de los datos en el ORM y se ha creado un servicio con el proceso de medición de los dientes, cargando los modelos ONNX sin necesidad de instalar librerías extra.

Independiente 5: 13/8/23

Las funcionalidades mínimas de la API se han completado y funcionan sin el *front-end*. Se comienzan a desarrollar las partes análogas en el cliente. Las versiones de los paquetes de NPM ya no están soportadas y es necesario actualizar varios de los componentes de la plantilla.

Independiente 6: 28/8/23

Se completan las funciones de la API y se refactoriza para que se ajuste a la estructura de capas. Se mejora la visualización de las medidas de los dientes.

Independiente 7: 1/9/23

Es necesario modificar la base de datos para implementar un gestor de modelos global, se producen varias migraciones y se decide actualizar a SQLAlchemy 2.0 para simplificar las relaciones, implica portar las clases anteriores.

Independiente 8: 4/9/23

Se ha decidido portar todo el *front-end* a *Vue3*, esta decisión viene motivada por el uso de un modelo de componentes usado en la plantilla **basado en clases**³ no mantenido y desaconsejado. Esta decisión retrasa 2 días el desarrollo porque implica rehacer parcialmente los componentes ya existentes.

Independiente 9: 6/9/23

Se completa una versión mínima del cliente que ya interactúa con las funciones del servidor, pero es tosca de utilizar. Se crea una nueva rama de desarrollo, *main*, que se reserva para las versiones usables.

³<https://class-component.vuejs.org/>

Independiente 10: 9/9/23

Los cambios introducidos hasta ahora se pasan a la documentación, pero se continúa el desarrollo, esta vez volviendo a la API para mejorar la documentación y la gestión de los errores.

Independiente 11: 16/9/23

Se realizan mejoras generales en el cliente para mejorar la experiencia de usuario y se resuelven fallos en el servidor en cuanto a la eliminación de datos relacionados en la BDD.

Reunión 23: 18/9/23

Se ha completado la aplicación y se presenta al tutor, junto al trabajo desarrollado desde julio. Se valora positivamente el funcionamiento de esta. Se hacen correcciones en la documentación para su entrega.

A.3. Estudio de viabilidad

Este trabajo cuenta con 2 puntos relevantes desde el punto de vista económico y legal, el primero es el coste del *hardware* necesario para el desarrollo de modelos de aprendizaje profundo y el segundo son las implicaciones legales de usar información médica cuando tratamos las radiografías.

Viabilidad económica

A la hora de calcular la viabilidad económica del proyecto debemos tener en cuenta los costes tanto humanos como materiales que se requieren para mantener la aplicación funcionando y para mejorar sus funcionalidades con el tiempo. En este caso se parte del supuesto de que en un entorno real futuro harían falta 2 empleados, uno gestionaría el despliegue y la integración, mientras que el otro continuaría el desarrollo de las nuevas funciones.

Costes humanos:

- *Ingeniero de software*: Sería el encargado de continuar el desarrollo de la aplicación, mejorar la precisión de los modelos y adaptarse a tecnologías nuevas. Su salario estimado sería de 40.000 € brutos al año.

- *DevOps*: Es el encargado de integrar de forma continua las actualizaciones que se lancen en la aplicación, además de buscar posibles fallos en el despliegue. Su salario estimado sería de 30.000 € brutos al año.

Costes materiales:

- *Equipos informáticos*: Suponiendo que contamos con 2 trabajadores en el desarrollo y cada uno necesita por lo menos un equipo, estimamos que el ingeniero necesitaría unos 2000 € cada 3 años para la adquisición de hardware de alto rendimiento, mientras que el encargado de DevOps requeriría de un hardware más modesto, estimándose en 1000 € cada 3 años.
- *Hosting*: Para facilitar el acceso a la aplicación en un hipotético lanzamiento, esta estaría desplegada en un servicio en la nube que costaría entre 50 € y 100 € al mes mientras la carga no sea demasiado alta, que podría ascender a más de 500 € al mes⁴.
- *Internet*: Suponiendo que ambos trabajadores trabajan en remoto y nos tenemos que hacer cargo del coste de su conexión a internet, en España bastaría con 30 €/persona por mes para tener una conexión de alta velocidad.

Por lo tanto, si queremos conocer los costes anuales de funcionamiento, tenemos que sumar todos los gastos tal como se muestra en la Tabla A.1:

	Anual	Mensual
Costes humanos	70.000 €	5.833 €
Hardware	1.000 €	84 €
Hosting	1.200 €	100 €
Internet	720 €	60 €
Total	72.920 €	6.077 €

Tabla A.1: Costes totales de funcionamiento

Viabilidad legal

El problema que nos encontramos es que vamos a tratar con una serie de radiografías de usuarios que no podemos almacenar más tiempo que

⁴Estimado sobre Amazon AWS, los precios son extremadamente flexibles según el uso

el requerido para procesarlas y devolverles el resultado. Además, debemos añadir un aviso informando de que, de acuerdo a la ley de protección de datos, los datos que sean provistos por los usuarios no serán almacenados más del tiempo necesario. Sin embargo, el modelo entrenado sí que es apto para su distribución, ya que no contiene las imágenes como tal.

Licencias de las librerías usadas:

- **Pytorch:** Es una librería desarrollada por Facebook, por lo que utiliza una **versión modificada de la licencia BSD**⁵. En resumen se permite el uso, modificación y distribución del código de Pytorch, incluyendo los modelos preentrenados que incluye, con algunas excepciones [9].
- **ResNet:** Es la arquitectura usada en los modelos de regresión y se le aplica la licencia Apache 2.0, que permite su distribución comercial [11].
- **Detectron2:** La librería encargada de la segmentación usa una licencia Apache 2.0, permite la comercialización.
- **YOLOv8:** Es una alternativa a Detectron2, también permite su distribución comercial, pero bajo la licencia AGPL-3.0 [4], existe la posibilidad de usar una licencia más restrictiva, que no obligue al liberar el código pagando una licencia comercial. El uso de esta librería nos obliga a que nuestro trabajo use la misma licencia.
- **FastAPI:** La librería principal utilizada en el *back-end* usa una licencia MIT, es muy permisiva y permite la distribución comercial sin el código fuente [10].
- **Vue.js:** El framework del cliente también se distribuye bajo la licencia MIT [12].

⁵<https://github.com/pytorch/examples/blob/main/LICENSE>

Apéndice *B*

Especificación de Requisitos

B.1. Introducción

En la especificación de requisitos se plasman los objetivos del proyecto, los requisitos que debe cumplir y sus casos de uso.

B.2. Objetivos generales

Los objetivos generales del proyecto han sido:

1. Investigar la aplicación de técnicas de *machine learning* convencional y de *deep learning* tanto para la segmentación de las radiografías como para la estimación de la medida de la raíz.
2. Elaborar un resumen con los resultados de cada modelo de tal forma que estos pueden ser evaluados de la forma más objetiva posible.
3. Minimizar el error obtenido en la medición a menos de 1 mm, que es lo que se consideraría menor al error humano.
4. Crear una visualización de los resultados cómoda para el usuario.

B.3. Catálogo de requisitos

El catálogo de requisitos contiene las condiciones que se deben cumplir para considerar que se cumplen los objetivos, se dividen en requisitos funcionales o RF cuando estos se refieren a las funciones que debe realizar

el programa final y no funcionales o RNF cuando se refieren a aspectos relacionados con la experiencia del usuario y otras funcionalidades complementarias.

Requisitos Funcionales (RF)

- RF-1 **Segmentación de dientes:** La aplicación debe ser capaz de realizar una segmentación de los todos los dientes y las raíces que se muestren en una radiografía de una forma precisa.
- RF-2 **Medición de dientes:** La aplicación debe ser capas de hacer estimaciones precisas sobre la longitud del canal reticular de cada diente identificado en la radiografía.
- RF-3 **Visualización de las mediciones:** El resultado de todo el proceso de medición y segmentación debe ser mostrado en forma de una sola imagen que muestre el contorno de cada diente y su medida, de tal forma que pueda ser fácilmente interpretado por el odontólogo.
- RF-4 **Gestión de modelos:** La aplicación debe permitir actualizar los modelos usados por el usuario de la forma más sencilla posible, sin intervención de este.

Requisitos No Funcionales (RNF)

- RNF-1 **Facilidad de uso:** Se debe poder usar la aplicación de forma intuitiva, de tal forma que el usuario haga el mínimo de pasos posibles para obtener el resultado.
- RNF-2 **Velocidad:** El proceso de medición automática debe ser más rápido que el proceso manual realizado por un odontólogo.
- RNF-3 **Privacidad:** La aplicación no debe almacenar datos personales de los pacientes, ni las radiografías más tiempo que el estrictamente necesario para su procesado.
- RNF-4 **Versatilidad:** La aplicación debe poder ser usada con el mayor número de máquinas de rayos-X posible.
- RNF-5 **Escalabilidad:** El componente que contiene los modelos va a ser el centro de la aplicación y debe ser escalable en un futuro si se desarrollan modelos más precisos.

B.4. Usuarios

A continuación se introducen los tipos de usuarios que van a hacer uso de la aplicación para entender mejor sus roles en el diagrama de uso [B.1](#). Debido al sector al que se enfoca se ha decidido no incluir un usuario anónimo o no autenticado.

U-1 **Usuario registrado:** Es el odontólogo, se le va a permitir el acceso únicamente a las funciones de su campo, por lo tanto, no va a gestionar usuarios, ni modelos, ni ninguna otra cuestión que no sea necesaria para su trabajo. Esta restricción se considera vital para mantener una experiencia rápida y sencilla.

U-2 **Administrador:** Es un desarrollador de la aplicación o una persona con conocimientos técnicos a la que se asigna la responsabilidad de llevar a cabo todas las tareas no relacionadas con el trabajo diario del odontólogo, como la gestión de usuarios o la actualización de los modelos.

B.5. Especificación de requisitos

En esta sección se detallan los requisitos planteados en el catálogo de requisitos [B.3](#)

Casos de uso

Los casos de uso detallan las interacciones del usuario con la aplicación.

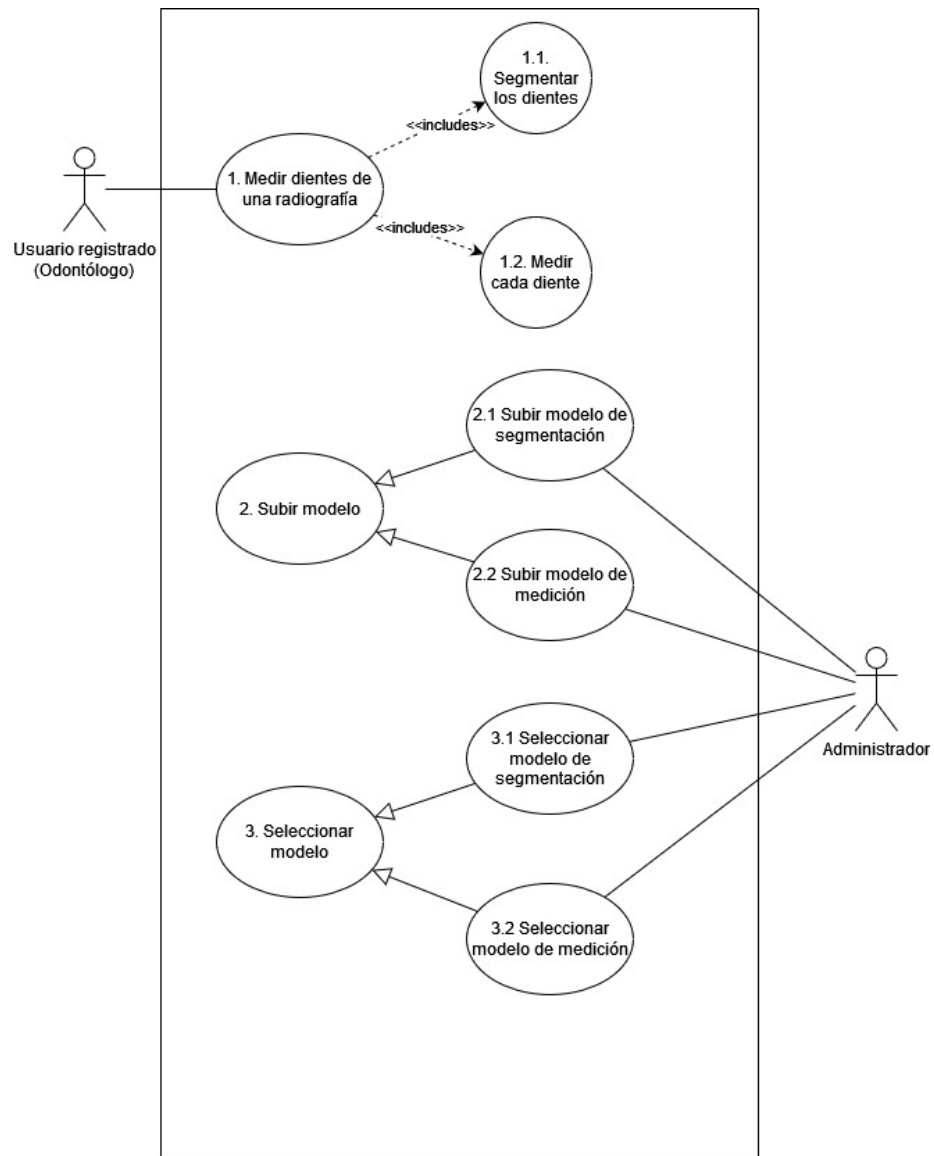


Figura B.1: Diagrama de casos de uso

CU-1	Medir dientes de una radiografía
Versión	1.0
Autor	Álvar San Martín
Requisitos asociados	RF-1, RF-2, RF-3, RNF-2
Descripción	Representa la funcionalidad completa, opaca al usuario, que solo introduce una imagen y recibe las mediciones.
Precondición	Haber obtenido una radiografía en formato digital.
Acciones	<ol style="list-style-type: none"> 1. El usuario carga una imagen en la aplicación. 2. El usuario recibe una imagen completamente anotada.
Postcondición	El usuario ha recibido la imagen anotada y con todas las mediciones.
Excepciones	<p>En caso de que el usuario se equivoque y no seleccione una imagen compatible (.png o .jpg) se le notificará que se está intentando introducir una imagen incorrecta.</p> <p>En caso de que las dimensiones no sean las correctas, se le notificará que es posible que la máquina que usa no sea compatible.</p>
Importancia	Alta

Tabla B.1: CU-1 Medir dientes de una radiografía.

CU-1.1	Segmentar los dientes
Versión	1.0
Autor	Álvar San Martín
Requisitos asociados	RF-1, RF-3, RNF-2
Descripción	Engloba el proceso de segmentación de la imagen, conlleva desde la identificación de elementos para su recorte hasta el dibujado de contornos para la visualización.
Precondición	El usuario ha subido una imagen a la aplicación.
Acciones	<ol style="list-style-type: none"> 1. El programa segmenta la imagen introducida y recorta dientes individuales.
Postcondición	El modelo de segmentación ha detectado por lo menos un diente y ha creado las máscaras de cada elemento.
Excepciones	En caso de que no se haya detectado ningún diente, se notificará al usuario que la imagen introducida no es clara o no contiene dientes.
Importancia	Alta

Tabla B.2: CU-1.1 Segmentar los dientes.

CU-1.2	Medir cada diente
Versión	1.0
Autor	Álvar San Martín
Requisitos asociados	RF-2, RF-3, RNF-2
Descripción	Representa la funcionalidad completa, opaca al usuario, que solo introduce una imagen y recibe las mediciones.
Precondición	El modelo de segmentación ha detectado por lo menos un diente y ha creado las máscaras de cada elemento.
Acciones	<ol style="list-style-type: none"> 1. Recibimos un conjunto de recortes. 2. Se mide cada recorte y se devuelven las medidas.
Postcondición	Se han hecho tantas mediciones como imágenes de entrada.
Excepciones	-
Importancia	Alta

Tabla B.3: CU-1.2 Medir cada diente.

CU-2	Subir modelo
Versión	2.0
Autor	Álvar San Martín
Requisitos asociados	RF-4
Descripción	Representa la funcionalidad genérica que permite subir nuevos modelos a la aplicación.
Precondición	El usuario es administrador.
Acciones	<ol style="list-style-type: none">1. El administrador se sitúa en la ventana de creación de modelo.2. Introduce los datos del modelo.3. Adjunta el archivo del modelo.4. Confirma la creación del modelo.
Postcondición	El modelo se ha subido correctamente.
Excepciones	El administrador no ha introducido alguno de los datos requeridos.
Importancia	Alta

Tabla B.4: CU-2 Subir modelo.

CU-2.1	Subir modelo de segmentación
Versión	2.0
Autor	Álvar San Martín
Requisitos asociados	RF-4
Descripción	Representa la funcionalidad que permite subir nuevos modelos de segmentación a la aplicación.
Precondición	El usuario es administrador.
Acciones	<ol style="list-style-type: none"> 1. El administrador se sitúa en la ventana de creación de modelo. 2. Selecciona que el tipo de modelo sea de segmentación. 3. Introduce los datos del modelo, junto a su rendimiento de segmentación. 4. Adjunta el archivo del modelo. 5. Confirma la creación del modelo.
Postcondición	El modelo se ha subido correctamente.
Excepciones	El administrador no ha introducido alguno de los datos requeridos.
Importancia	Alta

Tabla B.5: CU-2.1 Subir modelo de segmentación.

CU-2	Subir modelo de medición
Versión	2.0
Autor	Álvar San Martín
Requisitos asociados	RF-4
Descripción	Representa la funcionalidad que permite subir nuevos modelos de medición a la aplicación.
Precondición	El usuario es administrador.
Acciones	<ol style="list-style-type: none"> 1. El administrador se sitúa en la ventana de creación de modelo. 2. Selecciona que el tipo de modelo sea de segmentación. 3. Introduce los datos del modelo, incluyendo su rendimiento de medición. 4. Adjunta el archivo del modelo. 5. Confirma la creación del modelo.
Postcondición	El modelo se ha subido correctamente.
Excepciones	El administrador no ha introducido alguno de los datos requeridos.
Importancia	Alta

Tabla B.6: CU-2.2 Subir modelo de medición.

CU-3	Seleccionar modelo
Versión	2.0
Autor	Álvar San Martín
Requisitos asociados	RF-4
Descripción	Representa la funcionalidad genérica que permite al administrador seleccionar el modelo que va a ser utilizado por el usuario.
Precondición	<ol style="list-style-type: none">1. El usuario es administrador.2. Existe más de 1 modelo del tipo en la aplicación.
Acciones	<ol style="list-style-type: none">1. El administrador accede a la lista de modelos.2. El administrador selecciona un modelo nuevo entre los que han sido previamente subidos.
Postcondición	El modelo se ha cambiado correctamente y se muestra el cambio.
Excepciones	
Importancia	Media

Tabla B.7: CU-3 Seleccionar modelo.

CU-3.1	Seleccionar modelo de segmentación
Versión	2.0
Autor	Álvar San Martín
Requisitos asociados	RF-4
Descripción	Representa la funcionalidad que permite al administrador seleccionar el modelo de segmentación que va a ser usado en el proceso de medición de los dientes de forma transparente al usuario.
Precondición	<ol style="list-style-type: none"> 1. El usuario es administrador. 2. Existe más de 1 modelo de segmentación en la aplicación.
Acciones	<ol style="list-style-type: none"> 1. El administrador accede a la lista de modelos. 2. El administrador selecciona un modelo nuevo entre los que han sido previamente subidos.
Postcondición	El modelo se ha cambiado correctamente y se muestra el cambio.
Excepciones	
Importancia	Media

Tabla B.8: CU-3.1 Seleccionar modelo de segmentación.

CU-3.2	Seleccionar modelo
Versión	2.0
Autor	Álvar San Martín
Requisitos asociados	RF-4
Descripción	Representa la funcionalidad que permite al administrador seleccionar el modelo de medición que va a ser usado en el proceso de medición de los dientes de forma transparente al usuario.
Precondición	<ol style="list-style-type: none"> 1. El usuario es administrador. 2. Existe más de 1 modelo de medición en la aplicación.
Acciones	<ol style="list-style-type: none"> 1. El administrador accede a la lista de modelos. 2. El administrador selecciona un modelo nuevo entre los que han sido previamente subidos.
Postcondición	El modelo se ha cambiado correctamente y se muestra el cambio.
Excepciones	
Importancia	Media

Tabla B.9: CU-3.2 Seleccionar modelo de medición.

Apéndice C

Especificación de diseño

C.1. Introducción

En la especificación de diseño detallamos como interactúan entre ellos los componentes de la aplicación.

C.2. Diseño de datos

El trabajo se estructura en 2 partes, primero el estudio de los datos y la creación de modelos y segundo la aplicación donde se aplican los resultados de la primera parte. Las partes del proyecto no comparten datos a no ser que sean específicamente exportados, por lo que no es necesario que se encuentren en la misma máquina.

En la primera parte, los datos más importantes son los implicados en el entrenamiento y se almacenan en forma de conjuntos de datos en el sistema de archivos de la máquina en la que se trabaja. Por motivos de privacidad no se publican estos conjuntos en el repositorio de *GitHub*.

Conjunto para segmentación

El conjunto de datos utilizado en la segmentación se genera gracias a la aplicación *label-studio* y el resultado es una estructura de datos basándose en el formato *COCO*, como se muestra en la Figura C.1 para los archivos y en la C.2 para el archivo JSON *COCO* con las etiquetas.

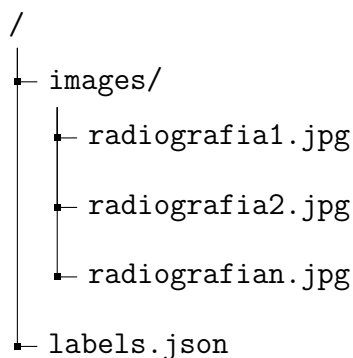


Figura C.1: Archivos del conjunto de datos segmentación

Conjuntos para la regresión

Para la regresión hemos generado 3 conjuntos siguiendo el esquema de la segmentación, con una estructura casi idéntica a [C.1](#), pero con un archivo *CSV* con los datos en vez de un *JSON*. El archivo *CSV* tiene las siguientes columnas:

- **id_diente**: identificador de la imagen de la radiografía junto con su formato.
- **pos_diente**: posición del diente dentro de la mandíbula, no se usa en la regresión por *deep learning*, pero se conserva para un futuro.
- **longitud**: longitud del diente en milímetros, es la etiqueta que vamos a usar en la regresión.

Datos de la aplicación

En la segunda parte, en la aplicación, se ha utilizado una base de datos relacional PostgreSQL, que al ser *open-source* y tener una licencia de uso muy permisiva [\[2\]](#) es ideal para el desarrollo. Además, se ha utilizado el ORM SQLAlchemy [\[3\]](#) para acelerar el desarrollo, evitando tener que acceder la base de datos directamente mediante consultas.

Diseño de la base de datos

Para definir la estructura de los datos, se ha comenzado por un diagrama entidad-relación [C.3](#), posteriormente se ha convertido en un diagrama relacional [C.4](#) que finalmente se ha traducido en clases del ORM SQLAlchemy.

Modelo entidad-relación

Destacar la existencia de una relación IS-A únicamente entre dos entidades, Usuario y Administrador, que indica que un Usuario puede serlo a secas o puede ser además Administrador.

Modelo relacional

En este caso particular C.4, qué se trabaja con un ORM y no directamente sobre la base de datos, se diseña un diagrama orientado a crear los objetos de SQLAlchemy y no un documento DDL. Por lo tanto, existen casos de generalización para la clase Modelo que no existirían en un diseño convencional. En conclusión, este diagrama es un punto medio entre uno relacional y un diagrama de clases.

C.3. Diseño procedimental

En el diseño procedimental describimos los pasos o procesos implicados en llevar a cabo una función de la aplicación. La aplicación cuenta con una función principal, la medición de dientes, y una secundaria, la gestión de los modelos que se usan en la medición.

- **Medición del diente:** La Figura C.5 describe el proceso desde que el usuario envía una imagen de una radiografía a la API hasta que esta le devuelve la imagen con las mediciones dibujadas. Se ignora la fase en la que el usuario carga la imagen en el cliente, al no considerarse relevante. En rasgos generales, este diagrama se puede simplificar en un proceso genérico, sin entrar en detalles de la arquitectura de la API, como se ve en la Figura C.8.
- **Creación de modelo:** En el diagrama de secuencia C.6, se observa el proceso por el cual un usuario carga los datos de un modelo, que puede ser de regresión o de segmentación gracias a la generalización, en la API. En este caso hay un servicio encargado de escribir el archivo del modelo en el sistema de archivos y otro que graba los datos en la BDD.
- **Cambio de estado:** En el diagrama C.7 se detalla el proceso por el que seleccionamos un nuevo modelo, de segmentación o regresión, para ser usado por el usuario. Este proceso no cambia el estado anterior, sino que crea uno nuevo más reciente.

C.4. Diseño arquitectónico

En el diseño arquitectónico se refleja la estructura interna de la aplicación y las razones que justifican esta distribución. Nos encontramos con dos componentes con estructuras diferenciadas, el *front-end* desarrollado sobre el *framework* *Vue* [5] y el *back-end* creado con el *framework* *FastAPI* [1].

La API y FastAPI

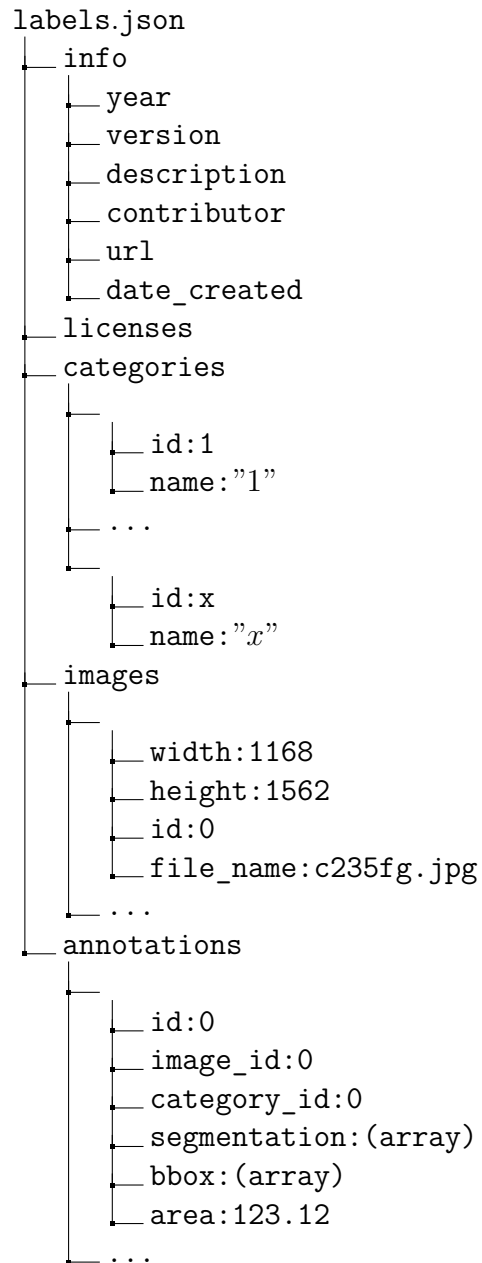
Se decide usar una API REST [7] para la aplicación porque permite flexibilizar el uso de la aplicación con independencia del dispositivo, motivado por el uso de algoritmos de *machine learning* que exigen una capacidad de cómputo considerable y solo pueden ser ejecutados con eficacia en un hardware específico. Como consecuencia, es posible usar la aplicación tanto en ordenadores como en dispositivos móviles.

FastAPI ofrece la posibilidad de personalizar la arquitectura de la API REST con las capas que se consideren necesarias. Se ha optado por la arquitectura de 3 capas (presentación, negocio y datos) para minimizar la complejidad mientras se diferencian claramente las funciones de la aplicación, como se refleja en la Figura C.9.

Descripción de las funciones de cada capa:

- **Presentación:** Contiene la interacción de la API con el exterior, gestiona las entradas y salidas de información que se producen por medio de peticiones HTTP [8]. Las funciones que se ejecutan para servir cada petición se ubican dentro del directorio `/api_vx/endpoints/`, donde la *x* representa la versión de la API a la que se accede. Para la validación de las entradas y salidas se utilizan modelos de *Pydantic* [6], que además contribuyen a la contención automática de la aplicación.
- **Negocio:** Representa las transformaciones de los datos dentro de la aplicación, contiene las clases y funciones que permiten la segmentación y medición de las radiografías, además de otras utilidades, como la escritura de archivos.
- **Datos:** Agrupa las partes encargadas de interactuar directamente con el almacenamiento de los datos. Por una parte, se encuentran los modelos del ORM, que definen la base de datos y la interacción con estos. Por otro lado, existe un apartado que encapsula las operaciones

CRUD que se ejecutan sobre las clases ORM, de este modo se atomizan las operaciones y son más fáciles de utilizar en la capa de negocio.

Figura C.2: Estructura de un archivo JSON con estructura *COCO*

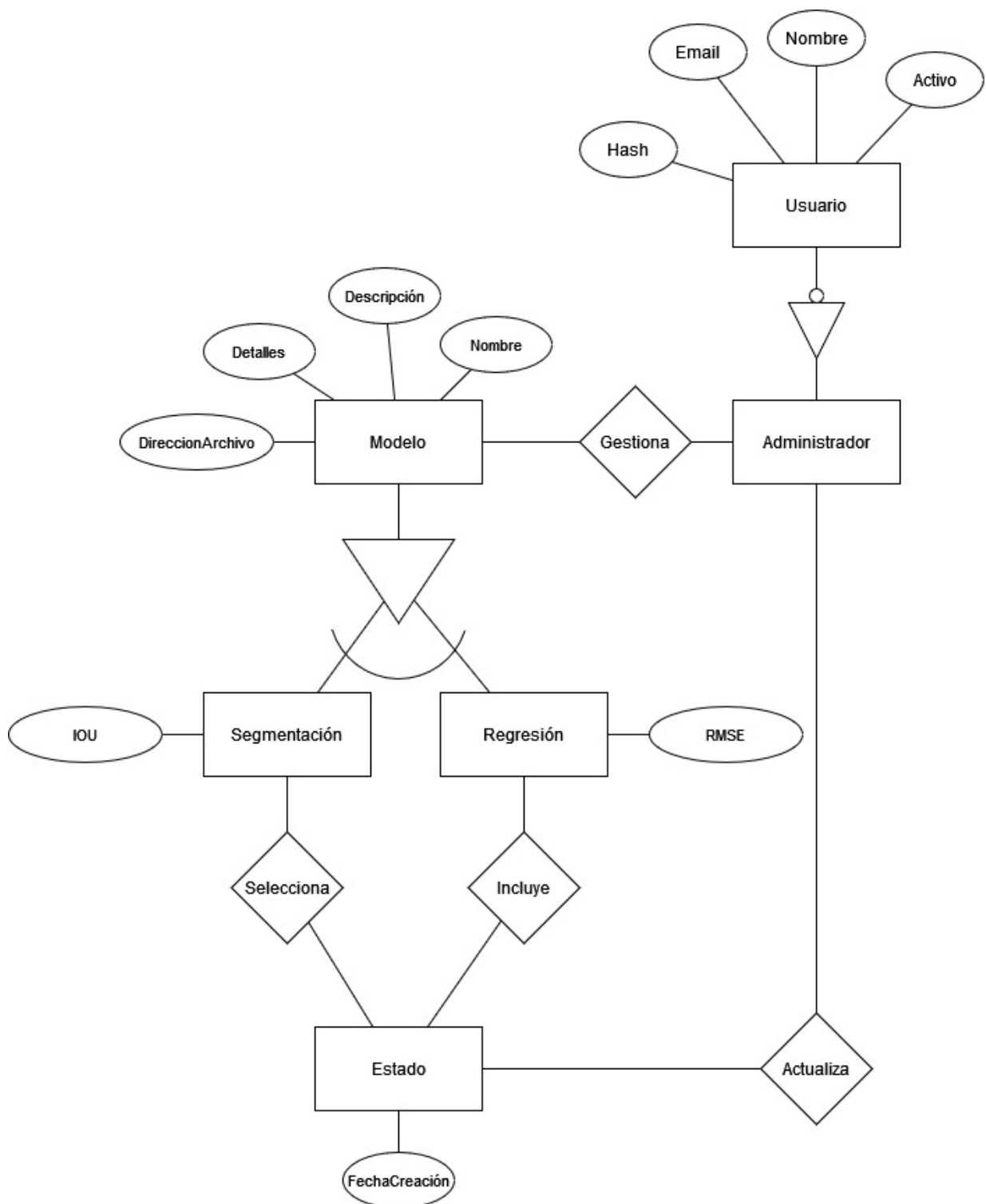


Figura C.3: Diagrama entidad-relación

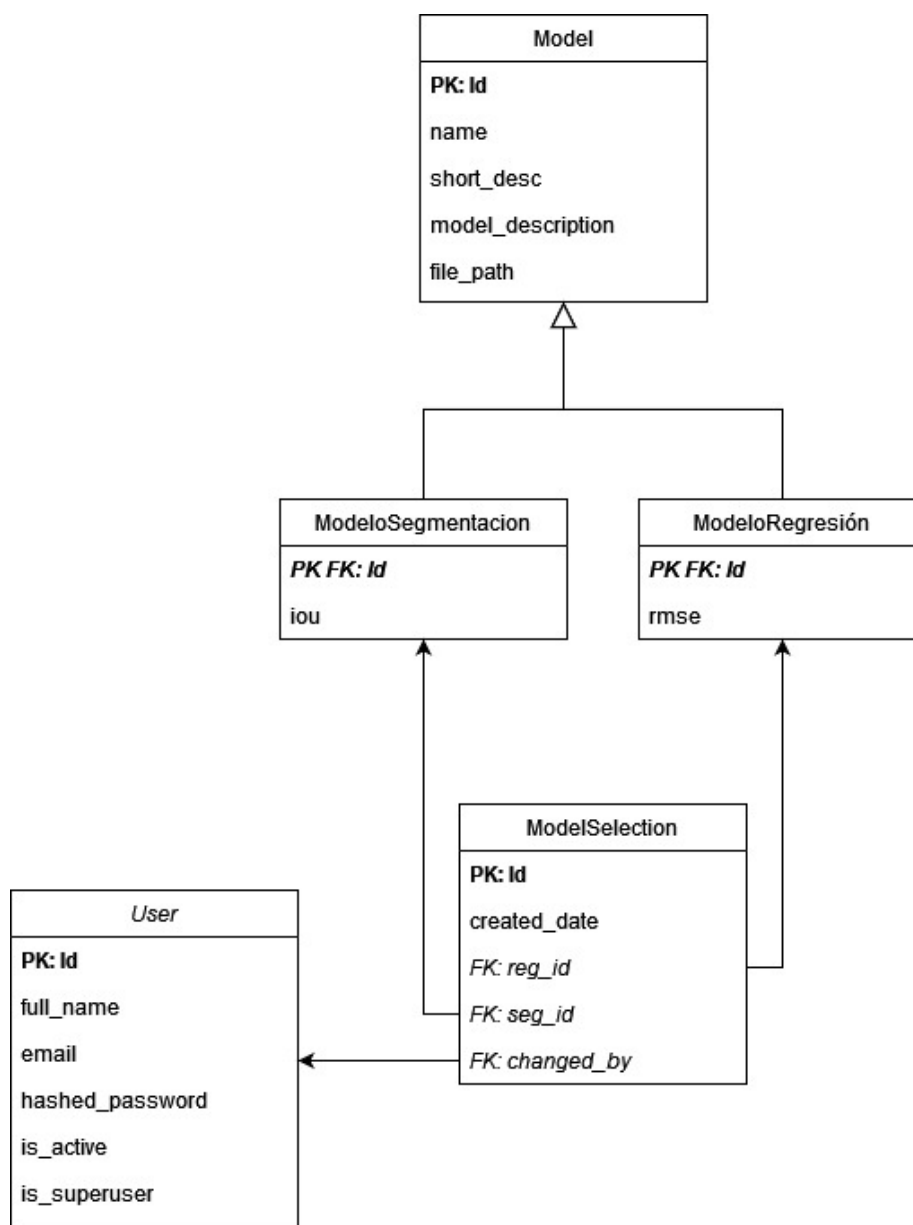


Figura C.4: Diagrama relacional

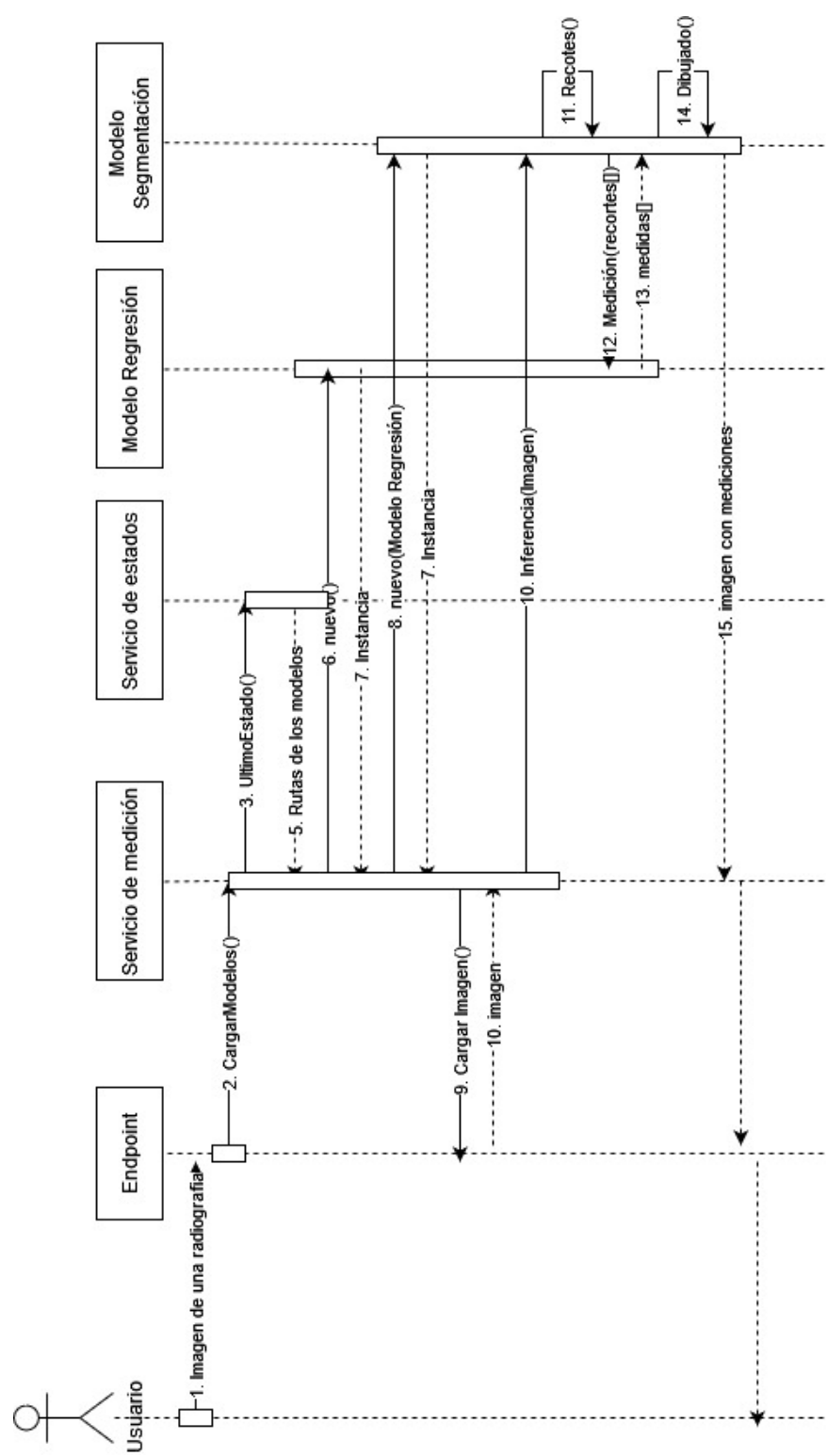


Figura C.5: Secuencias en la segmentación y medición de la radiografía

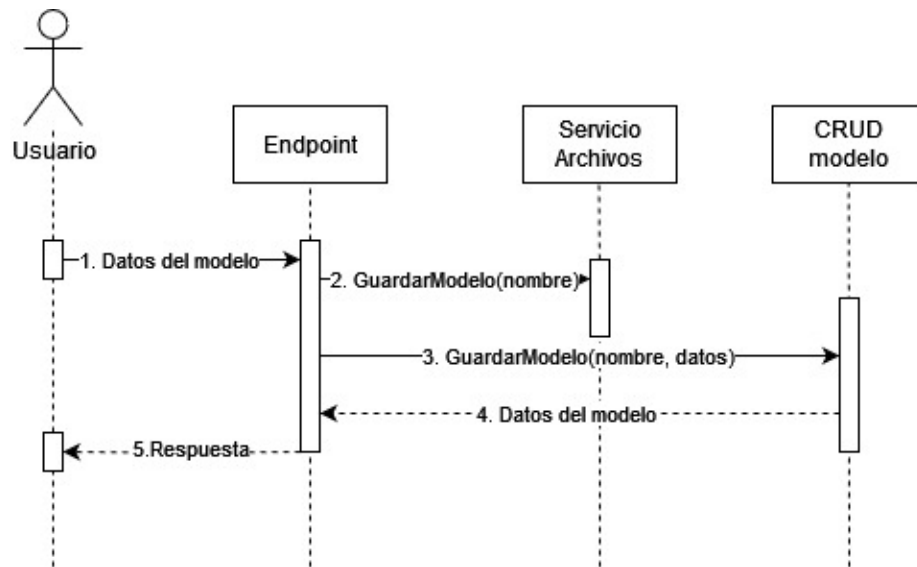


Figura C.6: Secuencias de la creación de un modelo

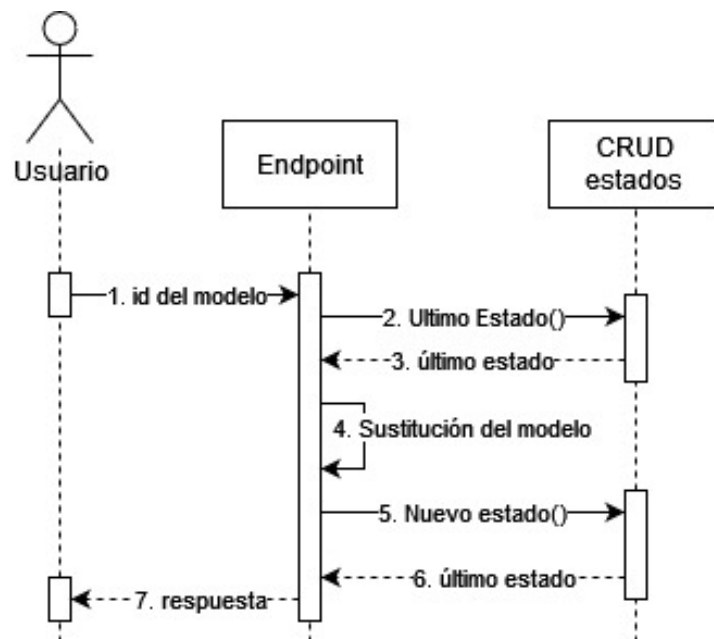


Figura C.7: Secuencias de la creación de un modelo

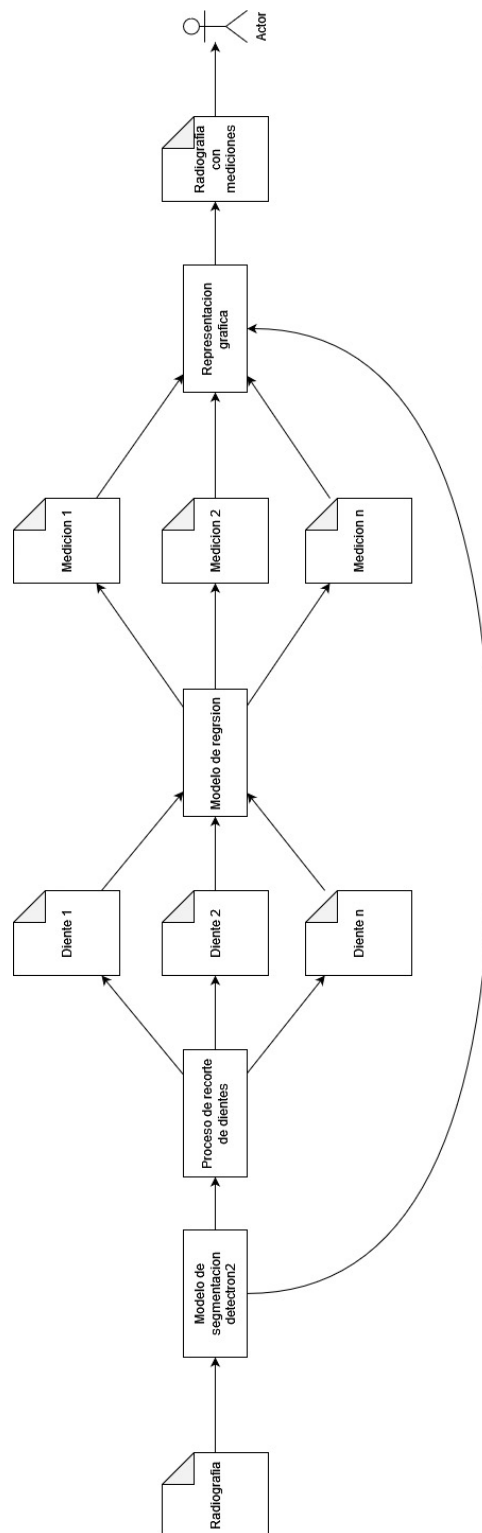


Figura C.8: Diagrama del proceso de segmentación y medición.

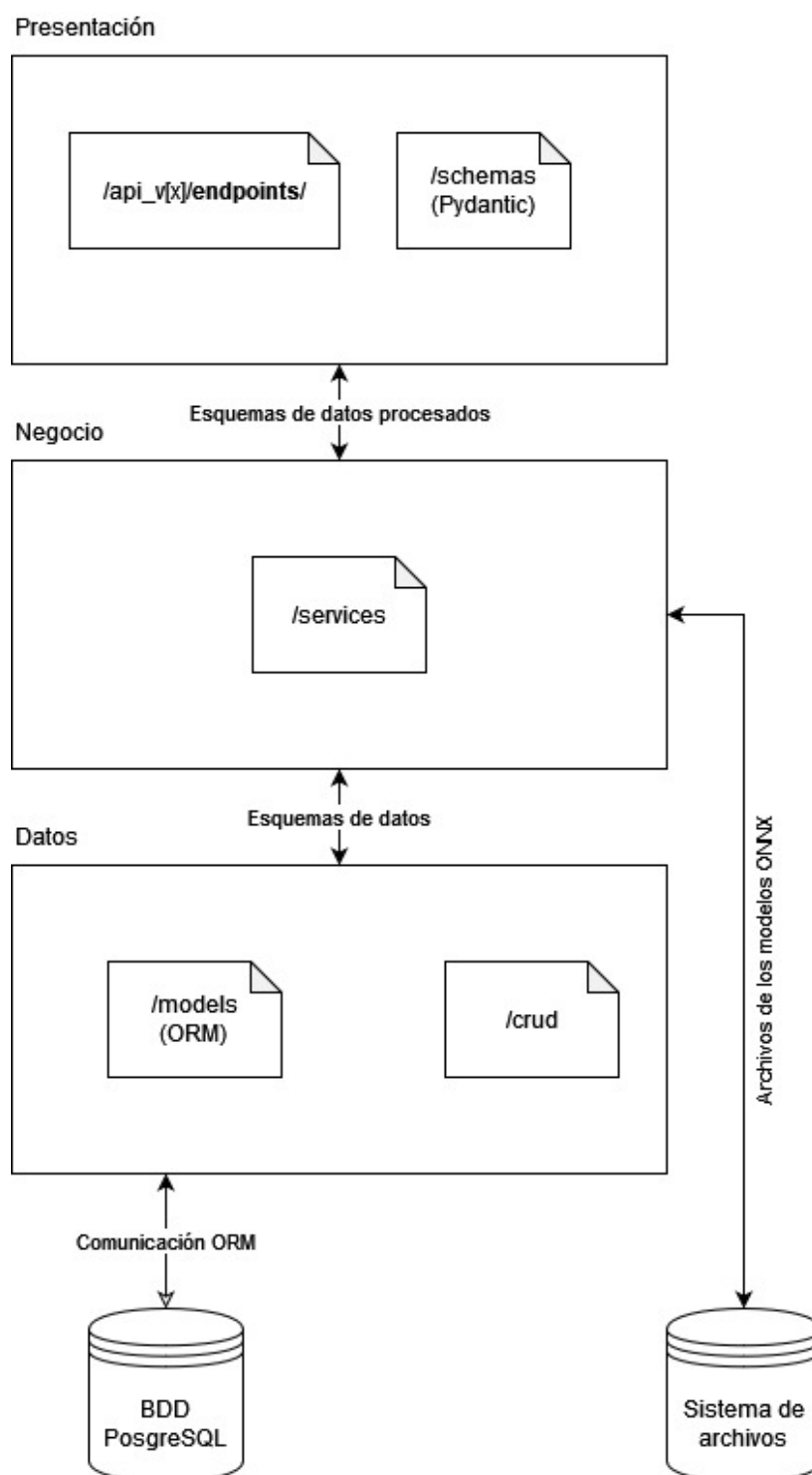


Figura C.9: Arquitectura de 3 capas de la API REST.

Apéndice D

Documentación técnica de programación

D.1. Introducción

Esta sección tiene el objetivo de dar a conocer a los futuros desarrolladores que trabajen sobre esta aplicación sus elementos principales y cómo trabajar sobre ellos. Este apéndice contiene la información tanto para la parte del estudio y entrenamiento de modelos como del desarrollo de la aplicación web.

D.2. Estructura de directorios

Entrenamiento de modelos

En esta sección detallamos la estructura de los directorios de todo el proyecto, incluyendo los dedicados al entrenamiento de los modelos, visualización de los resultados y gestión de conjuntos de datos. En el árbol del proyecto diferenciamos 3 carpetas principales, que comienzan por mayúscula:

```
/
├── Notebooks/
├── Datasets/
├── Funciones/
```

Figura D.1: Estructura de directorios del proyecto

- **Notebooks/**: Este directorio contiene todos los *notebooks* en formato *Jupyter Notebook* o *.ipynb*. Han servido de base para desarrollar las partes de la aplicación, permitiendo la visualización paso a paso de los procesos y de los datos con los que se trabaja mediante gráficos. Se recomienda trabajar sobre ellos mediante las herramientas *JupyterLab* o *Visual Studio Code*.

Los *notebooks* que contiene son:

- **Aplicacion.ipynb**: Contiene un prototipo del funcionamiento, la aplicación, se carga la imagen, se hacen los recortes y se miden. Se usa como mesa de trabajo para visualizar los pasos intermedios.
- **Documentacion.ipynb**: Contiene algunos de los gráficos generados para la documentación del proyecto.
- **Entrenador.ipynb**: Es el *notebook* utilizado para entrenar el modelo de *Detectron2*.
- **Generacion_COCO.ipynb**: Se utiliza para generar un dataset con formato *COCO* a partir del conjunto de imágenes y ficheros provistos por el odontólogo. Dependiendo de los nombres dados a las imágenes se debe modificar la expresión regular que los identifica.
- **Recorte_bbox.ipynb**: Genera los conjuntos de datos para la regresión haciendo los recortes necesarios sobre las imágenes de un conjunto de datos con formato *COCO*, requiere pasos extra para adaptar las anotaciones que se encuentran en un documento *PDF*.
- **Regresion_CNN.ipynb**: Este *notebook* se empleó para estudiar el uso de una red neuronal convolucional personalizada para la regresión. El proceso es extremadamente simple y no se emplea un planificador del entrenamiento ni validación cruzada, por lo que solo es práctico para hacernos una idea de si la técnica es adecuada.
- **Regresion_resnet.ipynb**: En este caso se desarrolla un proceso de entrenamiento completo, pensado para redes neuronales residuales, incluye preselección de los datos, *data augmentation*, planificador del entrenamiento y validación cruzada. Este *notebook* se utiliza para desarrollar el proceso, no para ejecutarlo, ya que es muy ineficiente y el consumo de memoria es excesivo, para los entrenamientos reales se ha creado un script normal que es mucho más eficiente.

- **Testeo_modelos.ipynb**: Su propósito es comprobar que el modelo de segmentación generaliza sobre imágenes que no estaban incluidas en el conjunto de datos inicial por estar tomadas con otras máquinas.
 - **Union_datasets.ipynb**: Se utiliza únicamente en caso de que queramos juntar 2 conjuntos de datos que inicialmente están separados.
 - **Visualizacion_modelos.ipynb**: Está dedicado a la visualización de los resultados de las redes neuronales y su comparación, incluye varios tipos de gráficos. No interactúa directamente sobre los modelos, sino sobre sus resultados.
 - **Visualizacion.ipynb**: Dedicado a la exploración inicial de las imágenes y a comprobar la segmentación de estas.
- **Datasets/**: En este directorio se encuentran los conjuntos de datos, originales y generados, que se utilizan en los entrenamientos. Este directorio no se encuentra en el repositorio al contener información que no se puede publicar.

Conjuntos originales:

- **CasosOrigen**: Contiene 53 imágenes de alta resolución hechas por una máquina de radiografías modelo *RVG6200*. No se ha modificado de ninguna forma.
- **CasosNuevos**: Contiene las imágenes y segmentaciones de un conjunto de 125 radiografías sobre dientes aislados tomadas por un aparato de radiografías modelo *RVG6100*. No se ha modificado de ninguna forma.

Conjuntos orientados a la segmentación:

- **Dataset**: Conjunto de datos ya estructurado en formato *COCO* con las imágenes del conjunto *CasosOrigen*. Está enfocado a la segmentación.

Conjuntos orientados a la regresión:

- **Recortes**: Contiene las radiografías recortadas, según su *boundary box*, del conjunto *CasosOrigen* junto con sus medidas. Se han excluido las imágenes inválidas.
- **Recortes_Completos**: Contiene las radiografías recortadas por su contorno de los conjuntos *CasosOrigen* y *CasosNuevos*.

- **Recortes_Mascara:** Contiene las máscaras binarias de las radiografías recortadas por su contorno de los conjuntos *CasosOrigen* y *CasosNuevos*.
- **Recortes_Nuevos:** Contiene las radiografías recortadas, según su *boundary box*, del conjunto *CasosNuevos* junto con sus medidas. Se han excluido las imágenes inválidas.
- **Funciones/:** Es un directorio utilizado a modo de biblioteca de funciones para el entrenamiento de las redes neuronales, contiene archivos para separar en módulos cada etapa del entrenamiento. Los archivos más importantes son:
 - **dataloaders.py:** Contiene las funcionalidades dedicadas a la carga de los datos para introducirlos en el modelo.
 - **DientesDataset.py:** Contiene la clase que se usa para abstraer el conjunto de datos con el que se trabaja.
 - **Padding.py:** En una clase que se utiliza en el *data augmentation* para mantener las proporciones de una imagen añadiendo bordes en blanco.
 - **particiones.py:** Se encarga de almacenar las funciones que nos permiten partir el dataset utilizado en partes aleatorias o en *folds* si usamos validación cruzada.
 - **test.py:** Contiene la función de testeo de un modelo en un instante.
 - **train.py:** Contiene la función de entrenamiento de la red neuronal, sirve para cualquier modelo de regresión.

Aplicación web: API REST

La API REST tiene una estructura de directorios [D.2](#) que sigue el patrón establecido en la documentación de FastAPI [\[1\]](#), con el añadido de los directorios para almacenar los modelos.

Detalles y funciones de cada directorio:

- **alembic/:** Contiene las versiones de la BDD que genera la librería *Alembic*. Excepto en situaciones excepcionales no se deben modificar los archivos a mano, estos se generan automáticamente cuando se modifican los objetos ORM y se ejecuta una migración.

```
/app
├── alembic/
├── app/
│   ├── api/
│   │   └── api_v1/
│   │       └── endpoints/
│   ├── core/
│   ├── crud/
│   ├── db/
│   ├── models/
│   ├── schemas/
│   ├── services/
│   └── tests/
├── modelos_regresion/
├── modelos_segmentacion/
└── scripts/
```

Figura D.2: Estructura de directorios de la API REST

- **modelos_regresion/ y modelos_segmetnacion/**: Almacenan los archivos de los modelos implicados en la aplicación. Los archivos de ambos deben usar la extensión `.onnx`. Introducir modelos a mano en estos directorios no implica que estos vayan a aparecer automáticamente en la aplicación si sus datos no son introducidos en la BDD.
- **scripts/**: Contiene scripts útiles para la ejecución de los tests y el formato automático del código,
- **api/**: Guarda las versiones de la API, bajo el nombre de `api_v[versión]`. Varias versiones de la API pueden coexistir mientras se creen directorios nuevos, esta práctica es recomendable para mantener retro-compatibilidad con consumidores previos. Dentro de cada directorio de versión de la API se pueden encontrar uno o más archivos con los *endpoints* de esa versión concreta.
- **core/**: Contiene archivos que cargan la configuración general de la API, almacena todo lo implicado en el funcionamiento de la aplicación como la ubicación de la BDD, su usuario, el dominio o la configuración de la autenticación. Sin embargo, no se especifican las opciones en sí, estas se encuentran en variables de entorno que se inicializan en los scripts de docker.

- **crud/**: Facilita los scripts encargados de las operaciones CRUD (Crate, Read, Update, Delete) de los objetos ORM. Para una programación ágil todos los archivos representan una clase que contiene todas las operaciones, esta clase hereda de una clase base llamada *CRUDBase* que ya implementa las operaciones elementales y se le inyecta el objeto ORM y los esquemas de validación.
- **db/**: En este directorio se encuentra la configuración e inicialización del ORM, pero al igual que en **core/** no se guardan los datos en sí. No es un directorio que se vaya a editar, salvo en ocasiones excepcionales.
- **models/**: Contiene la definición de la base de datos en forma de objetos ORM, todos ellos deben heredar, directa o indirectamente, de la clase base instanciada en *db/base_class.py* de lo contrario las clases no se mapearán en la BDD real.
- **schema/**: Los esquemas de validación de datos se encuentran aquí, todos están relacionados con la librería *Pydantic*. Las clases contenidas en este directorio definen cómo se reciben y envían los datos en los *endpoints* e incluso como se comunican las funciones si deseamos un tipado fuerte.
- **services/**: Almacena los scripts de la capa de negocio, es decir, todo el tratamiento de los datos que es transparente al consumidor. Aquí se produce el tratamiento de las imágenes y la gestión de los archivos.
- **tests/**: Comprende los test de la aplicación, tanto de los endpoints, comprobando la salida de una petición concreta, como de las funciones CRUD.

Aplicación web: Front-end

El *front-end* tiene una estructura de directorios **D.3** que sigue el patrón establecido en la **documentación de Vue**¹, ligeramente modificado para acomodar una *store* de *Vuex*.

Roles de cada directorio:

- **dist/**: Es el directorio en el que se vuelca la aplicación una vez compilada y minimizada, no contiene archivos que se deban modificar a mano.

¹<https://vuex.vuejs.org/guide/structure.html>


```
/frontend/  
├── dist/  
├── public/  
├── src/  
│   ├── assets/  
│   ├── components/  
│   ├── interfaces/  
│   ├── plugins/  
│   ├── router/  
│   ├── store/  
│   ├── types/  
│   ├── views/  
│   ├── api.ts  
│   └── main.ts
```

Figura D.3: Estructura de directorios del front-end Vue

- **public/**: En este directorio se encuentra la base HTML que se encarga de cargar los scripts y el icono de la aplicación. No se recomienda modificar el HTML desde este archivo.
- **src/**: Directorio base del que cuelgan todos los componentes de la aplicación, todas las funcionalidades, a excepción de la configuración, se encuentran aquí.
- **assets/**: Almacena los recursos estáticos de la aplicación, como pueden ser imágenes o fuentes.
- **components/**: En este directorio encontramos todos los componentes reutilizables, se recomienda no agruparlos en sub-carpetas para facilitar su localización.
- **interfaces/**: Tiene el propósito de almacenar las interfaces que se utilizarán para manejar tipos de datos complejos. Al contrario que en otros lenguajes, especialmente en Java, su propósito principal no es definir la estructura de clases compatibles.
- **plugins/**: Aquí Vue almacena sus complementos propios, que se instalan mediante *vue-cli*.
- **router/**: Se encarga de almacenar la lógica de enrutamiento del cliente, Vue usa una arquitectura de una sola página, aquí se especifica que vista se carga por cada ruta del navegador.

- **store/**: Es el directorio creado para gestionar la lógica de *Vuex*, a su vez contiene un directorio para cada estado que se quiera definir por separado, junto con sus operaciones, que se explican más adelante en [D.5](#).
- **types/**: Agrupa archivos utilizados para definir tipos de datos complejos, contiene un conjunto de interfaces útiles para gestionar las llamadas a la API.
- **views/**: El propósito de este directorio es agrupar las vistas, que no son otra cosa que componentes que no se reutilizan, sino que sirven como marco para insertar componentes e información propias de una página. Las vistas únicamente son referenciadas desde la función de enrutado en *router/*.
- **api.ts**: Es un archivo donde se definen todas las funciones que reciben o envían información al servidor, a la API REST.
- **main.ts**: Es el archivo donde se instancia la aplicación y se le agregan las utilidades principales: *Vuex*, *Vuetify* y *Vue-Router*

D.3. Manual del programador

En los puntos anteriores ya se ha explicado la estructura de nuestro árbol de archivos y todas las partes relevantes, este punto se va a centrar en recomendaciones para el funcionamiento de estos archivos.

Problema en la gestión de memoria

La mayoría del trabajo se centra en el trabajo sobre *notebooks* .ipynb y lo normal es ejecutarlos mediante *jupyter notebook*, *jupyter lab* o *google colab*. Sin embargo, cuando la ejecución implica trabajar sobre la GPU, estas 3 aplicaciones tienen el mismo problema por el que no se borra la memoria gráfica y, por lo tanto, se acaba consumiendo por completo, a pesar de que modelos como *resnet18* o *resnet30* con un tamaño de *batch* de 10 imágenes no consumen más de 3 GB de memoria. Por lo tanto, se recomienda crear scripts *Python* tradicionales para los entrenamientos, o bien, utilizar *Visual Studio Code* con un kernel *Python* y no *Jupyter*, este método funciona bien en cuanto uso de memoria, pero depende de que la terminal sobre la que funciona permanezca abierta y ha dado problemas al usarse mediante *SSH*.

D.4. Compilación, instalación y ejecución del proyecto

En esta sección se explica cómo instalar y ejecutar cada uno de los apartados del proyecto para poder continuar con su desarrollo. Las 2 partes se encuentran en repositorios distintos:

1. Entrenamiento de modelos²
2. Aplicación³

Instalación: Entrenamiento de modelos

Para la ejecución del proyecto es necesario instalar una serie de bibliotecas de *Python*, pero gracias a *Conda* se ha generado un fichero *requirements.yml* que automatiza la instalación. Si fuese necesario usar otras versiones de algún paquete, lo principal es instalar *Detectron2*, *PyTorch* y *TorchVision*, primero mediante *Conda* y si no es posible, como se explica en sus respectivos manuales:

- *Pytorch*, *TorchVision*⁴
- *Detectron2*⁵

Se recomienda aprovechar los entornos virtuales de *Conda* para separar la instalación del resto del sistema.

Los conjuntos de datos no se pueden subir a GitHub, se deben solicitar para ser incluidos en el proyecto.

Instalación: Aplicación

Antes de comenzar con la instalación, y en especial si la intención es continuar con el desarrollo, es necesario instalar los siguientes paquetes:

- Docker

²https://github.com/AlvarSML/TFG_2023_VisionArtificial

³https://github.com/AlvarSML/tfg23_aplicacion

⁴<https://pytorch.org/get-started/locally/>

⁵<https://detectron2.readthedocs.io/en/latest/tutorials/install.html>

- Docker Compose
- Node.js > 19.0
- Poetry

Si se está trabajando sobre WSL en Windows, la instalación de Docker se hace en Windows mediante Docker Desktop y la de los demás paquetes dentro de WSL.

La aplicación completa, *front-end*, *back-end*, *proxy* y demás utilidades se encuentran publicadas en el repositorio de GitHub, con excepción de los modelos, que por su elevado tamaño, no han sido incluidos, pero se pueden generar propios con los *notebooks* que se proveen.

Una vez se ha clonado el repositorio en la máquina, bastará con ejecutar el script de *docker-compose*, situado en el directorio raíz, para que se monten las imágenes y se lancen los contenedores con el siguiente comando:

```
$ docker-compose up -d
```

Cuando todos los contenedores se estén ejecutando, se podrá acceder a la aplicación mediante un navegador dirigiéndose a *localhost* y a la documentación de la API en *localhost/docs* (*Swagger*) y en *localhost/redoc* (*OpenAPI*)

D.5. Metodología del back-end

Si se quiere programar sobre la API se debe respetar la metodología con la que se ha desarrollado previamente, para que el proceso sea ordenado y rápido.

Contenedor Docker

La API se encuentra en el contenedor *backend* dentro de docker y se desarrolla desde este, no es necesario correrlo en un entorno independiente ya que mientras se trabaje en el entorno de desarrollo, *dev* en la configuración de *docker-compose*, se lanza un proceso que recarga el servidor cada vez que se guarda un archivo. Cuando queramos lanzar la aplicación únicamente hará falta modificar el entorno a producción y se eliminara la recarga y el modo depuración.

Dependencias

A diferencia que en un desarrollo más simple, donde se utiliza el gestor de paquetes *pip*, en este caso se utiliza el gestor de dependencias *poetry*, que funciona de forma similar a *pip*, pero utiliza dos archivos para almacenar más detalles de los paquetes que usamos y sus dependencias: *pyproject.toml* para definir las dependencias de nuestro proyecto y *poetry.lock* donde *Poetry* vuelca todos los detalles de estas dependencias. Se está usando *poetry* en vez de *pip* o *conda* porque es el gestor que se incluye en la plantilla, y no merece la pena invertir horas de trabajo en hacer el cambio a uno de los anteriores.

Si queremos añadir un nuevo paquete podemos añadirlo directamente en el archivo *.toml* o mediante el siguiente comando:

```
$ poetry add [paquete]
```

Pero el comando anterior únicamente añade el paquete y no lo instala, para confirmar los cambios tenemos 2 opciones, la primera instala los paquetes en el entorno que tengamos activo y el segundo además de instalarlo añade sus detalles al archivo *.lock*, se recomienda usar la segunda:

1. `$ poetry install`
2. `$ poetry lock`

ORM

Para el manejo de los datos se ha optado por el ORM *SQLAlchemy*, lo que implica que la definición de la BDD, solo se debe modificar usando las herramientas propias del ORM. De este modo se permiten las migraciones cuando creamos una nueva versión y es fácil seguir los cambios. Es importante saber que se usa la versión 2.0 de *SQLAlchemy*, y muchas de sus funciones no están disponibles en versiones anteriores, lo que implica que la documentación a consultar es esta: [Documentación de SQLAlchemy 2.0 declarative](https://docs.sqlalchemy.org/en/20/orm/extensions/declarative/api.html)⁶ Todas las clases que queramos incluir en la BDD deben cumplir 2 condiciones:

1. Deben heredar de la base declarativa, definida en */app/db/base_class.py*
2. Se deben importar en */app/app/db/base.py*

⁶<https://docs.sqlalchemy.org/en/20/orm/extensions/declarative/api.html>

Migraciones

Cuando se modifica una clase del ORM, esta no se modifica en la BDD en tiempo real, sino que debemos comenzar una migración. Para implementar las migraciones, dependemos del paquete *alembic* y cuando se quiera empezar una se deben seguir una serie de pasos:

1. Acceder al contenedor de docker:

```
$ docker-compose exec backend bash
```

2. Generar una revisión, *alembic* genera un script con los cambios a aplicar en la BDD actual:

```
$ alembic revision --autogenerate \
-m "resumen_de_los_cambios"
```

3. Confirmar los cambios en el control de versiones Git y aplicar los cambios:

```
$ git commit -m "cambios"
$ alembic upgrade head
```

Implementación de nuevas funcionalidades

A continuación se enumeran una serie de reglas con el objeto de mantener la estructura del código de la API.

1. Cuando se crean un *endpoint* nuevo, este se debe incluir en el router de *FastApi*, ubicado en el archivo *api.py* que se encuentra en cada directorio de versión de la API */app/app/api_v[x]/*
2. En cada *endpoint* solo se debe incluir la mínima cantidad de código para que funcione, únicamente llamadas a servicios y control de errores. Cualquier fragmento de código que produzca algún tratamiento de datos se considera lógica de negocio y, por tanto, deberá ser considerado como servicio.
3. En los servicios el acceso a los datos de la BDD se hace por medio de llamadas a funciones CRUD, definidas en su directorio, por lo tanto, no se deberán usar los objeto ORM para hacer consultas desde servicios o aplicar cambios sobre los objetos.

4. Las operaciones CRUD que se definan dentro de este directorio deben ser atómicas, si la operación implica la creación, o modificación de más de una entidad de la BDD a través de varios pasos se considerará un servicio.
5. Al sistema de archivos, para el almacenaje y gestión de modelos, se accede a través del servicio *archivos* y no desde operaciones CRUD.

D.6. Metodología del front-end

Para el desarrollo del *front-end*, que se encuentra desacoplado del *back-end*, se ha decidido usar el *framework* *Vue* junto con *Typescript* para mantener una documentación *automática* de las funciones, añadiendo el tipado del que carece JavaScript. El desacople de este componente nos permite usar más de un cliente que se comuniquen con la API REST, de tal forma que, si la aplicación requiere de un cambio de plataforma, es posible desarrollar una aplicación de escritorio o móvil con el mismo *back-end*.

Dependencias

En este caso se utiliza el gestor de paquetes de *Node.js*, *NPM*. Este gestor utiliza los archivos *package.json*, para especificar los paquetes deseados y sus versiones mínimas o máximas, y *package.lock*, que de forma similar a *Poetry*, especifica la versión actual de cada paquete instalado junto a sus detalles. Para instalar un nuevo paquete y actualizar el archivo *.lock*, usamos un único comando:

```
$ npm install [paquete]@[version]
```

Instalación y ejecución

Si se consulta el archivo *Dockerfile* se puede comprobar que la versión que se sirve desde el contenedor no es una versión de desarrollo, sino que al compilar la imagen, se compila también el proyecto y el resultado es una versión que no podemos editar. Para desarrollar el *front-end* debemos parar su contenedor y en su lugar lanzar el proyecto desde un entorno local mediante npm:

```
$ # Asumiendo que nos encontramos en /frontend/  
$ npm run serve
```

Este comando lanza el servicio *vue-cli-service* que crea una versión de desarrollo que se recargará automáticamente cada vez que detecte un cambio en los archivos.

Cuando queramos pasar los cambios al contenedor debemos compilar el proyecto mediante el script de NPM:

```
$ npm run build
```

Este comando minimiza los archivos, los convierte en *.html* y *.js* convencional y los guarda en el directorio */frontend/dev/*.

Vue

El *framework* utilizado en el *front-end* ha sido *Vue*. *Vue* permite desarrollar aplicaciones de una sola página reactivas, es decir, que los componentes se cargan y descargan en la misma página, lo que consigue una experiencia fluida para el usuario y permite al programador actualizar los datos que aparecen en pantalla de una forma muy sencilla.

Se está utilizando la versión 3 de *Vue*, que es compatible con la versión 5 de *Typescript*, la 4 de *Vuex* y la 4 de *Vue-Router*.

Vue tiene 2 modos que se pueden utilizar para crear componentes nuevos, *Options* y *Composition*. A la hora de crear componentes nuevos se recomienda usar *Options* por coherencia con el resto, pero ambos modos pueden coexistir en componentes distintos.

Vuex

Vuex es el gestor de estados de *Vue*, permite acceder a una serie de datos independientemente del componente. *Vuex* organiza los datos en una *store* que contiene un estado, métodos de acceso, *getters*, mutaciones y acciones. Los métodos de acceso permiten consultar los datos del estado, las mutaciones modifican los datos y las acciones realizan operaciones complejas de lectura y escritura. Dentro de los componentes también existe un estado local, pero este se reserva para datos sencillos que solo van a ser usados en el contexto del componente.

Vue-Router

Vue-Router no sustituye en ningún caso la seguridad de la API y no debe ser la única barrera para acceder a un conjunto de datos en el servidor, es una herramienta para mejorar la experiencia de usuario y no de seguridad.

Vue-Router es el elemento que gestiona las rutas de la aplicación para que cada vez que se modifique la URL en el navegador, la página no se recargue, sino que se cargue la vista pertinente. El árbol de vistas se encuentra en `/router/index.ts` con una estructura *JSON* con varios niveles que representan las funcionalidades, desde aquí se puede elegir que partes son visibles a cada tipo de usuario y que redirecciones se deben llevar a cabo en cada caso.

D.7. Pruebas del sistema

En la primera parte del proyecto, las pruebas de los modelos de *deep learning* se realizan obteniendo métricas estadísticas acerca de su rendimiento y ya han sido plasmadas en la memoria. En el caso de la aplicación, se han elaborado pruebas para comprobar la robustez de la API, e interceptar peticiones incorrectas del cliente, emitiendo después el código HTTP correcto.

Test-1	Requerimiento de autenticación			
Versión	1.0			
Autor	Álvar San Martín			
Descripción	Se comprueban todas las entradas de la API que requieren de autenticación enviando una petición correcta pero sin un token de sesión asociado			
Precondición	Ninguna			
Postcondición	Se devuelve el error <i>401 Not authenticated</i>			
Entrada API	Solicitud	Salida	Resultado	
get /models	-	401	Correcto	
post /models/nuevo_modelo_reg	Modelo de regresión correcto	401	Correcto	
post /models/nuevo_modelo_seg	Modelo de segmentación correcto	401	Correcto	
get /models/get_regression	-	401	Correcto	
get /models/get_segmentation	-	401	Correcto	
del /models/delete/{id}	Identificador de un modelo existente	401	Correcto	
get /states	-	401	Correcto	
post /states/change_reg	Identificador de modelo de regresión	401	Correcto	
post /states/change_seg	Identificador de modelo de segmentación	401	Correcto	
post /radiografias/subir	Archivo de imagen	401	Correcto	

Tabla D.1: Test-1 Requerimiento de autenticación

Test-2	Permisos necesarios		
Versión	1.0		
Autor	Álvar San Martín		
Descripción	Se comprueban todas las entradas de la API que requieren de permisos de administrador con peticiones correctas que deben estar acompañadas por un token de usuario regular		
Precondición	Se tiene un token de usuario regular, no administrador		
Postcondición	Se devuelve el error <i>403 Forbidden</i>		
Entrada API	Solicitud	Salida	Resultado
get /models	-	403	Correcto
post /models/nuevo_modelo_reg	Modelo de regresión correcto	403	Correcto
post /models/nuevo_modelo_seg	Modelo de segmentación correcto	403	Correcto
get /models/get_regression	-	403	Correcto
get /models/get_segmentation	-	403	Correcto
del /models/delete/{id}	Identificador de un modelo existente	403	Correcto
get /states	-	403	Correcto
post /states/change_reg	Identificador de modelo de regresión	403	Correcto
post /states/change_seg	Identificador de modelo de segmentación	403	Correcto

Tabla D.2: Test-2 Permisos necesarios

Test-3	Inanición de modelos
Versión	1.0
Autor	Álvar San Martín
Descripción	En ningún momento puede haber menos de un modelo de cada tipo
Precondiciones	<div><div>1. Se tiene un token de administrador</div><div>2. Se tiene un solo modelo de cada tipo en la BDD</div></div>
Postcondición	Se devuelve el error <i>403 Forbidden</i>

Entrada API	Solicitud	Salida	Resultado
del /models/delete/{id}	Identificador del último modelo de regresión	403	Correcto
del /models/delete/{id}	Identificador del último modelo de segmentación	403	Correcto

Tabla D.3: Test-3 Inanición de modelos

Apéndice *E*

Documentación de usuario

E.1. Introducción

Este apéndice se dedica a guiar al usuario en el uso de nuestra aplicación, asumiendo que el desarrollador ha desplegado la aplicación en la red local o en un *hosting*.

E.2. Requisitos de usuarios

La aplicación está basada en un entorno web, por lo que solo es necesario contar con conexión a internet y usar un navegador actualizado, como pueden ser:

- Google Chrome
- Mozilla Firefox
- Microsoft Edge
- Safari

Como apunte, esta lista incluye a sus equivalentes móviles y no hay soporte para funcionar en ninguna versión *Internet Explorer*.

Para acceder por primera vez a la aplicación se proporcionará una cuenta de administrador, en caso de que haya una persona con conocimientos técnicos que vaya a mantener actualizados los modelos, si no es el caso, únicamente se proporcionarán cuentas de usuario regular.

E.3. Instalación

La instalación es competencia del encargado de *dev-ops* y no del usuario, al tratarse de una aplicación web y no de escritorio. En caso de instalarse en un equipo como aplicación local (no recomendado), se deberá tener instalado Docker y Docker-compose, en cuanto a requisitos de *hardware* se recomienda tener 25 GB de espacio libre en el disco, 8 GB de memoria RAM y se recomienda usar una *GPU* compatible con *CUDA* o *ROCm*.

E.4. Manual del usuario

En esta sección se ilustra una breve guía de las funciones principales de la aplicación, a pesar de que no es necesario debido a que la interfaz es simple y se han procurado usar iconos y menús estándar con los que ya está familiarizado el usuario medio.

Inicio y cierre de sesión

El inicio de sesión (login) es obligatorio para cualquier usuario, la aplicación redirige automáticamente a la ventana de inicio [E.1](#) cuando el usuario no está logueado o su sesión ha expirado. Para cerrar sesión (logout) encontramos un botón [E.4](#) en el menú superior derecho [E.2](#) o en el menú lateral izquierdo [E.3](#).

Menú principal

Las funcionalidades de la aplicación se encuentran en el menú izquierdo, desplegable desde el botón superior izquierdo [E.4](#).

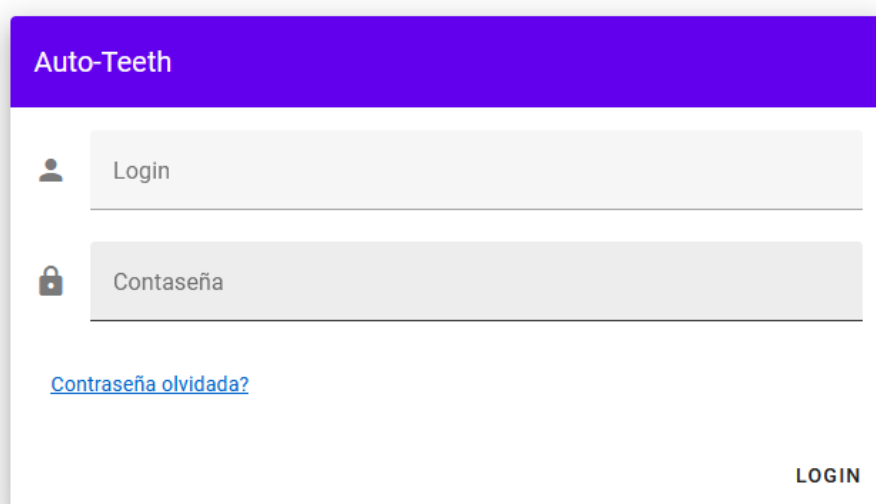
Herramienta de radiografías

Para acceder a la función de medido de radiografías se despliega al menú lateral izquierdo y se selecciona la opción *radiografías* marcada con el icono de una radiografía [E.5](#).

Para procesar una radiografía se usa el selector de archivos o se arrastra la imagen desde la carpeta a este, una vez se muestra la preselección, se presiona el botón *medir* y al cabo de unos segundos la imagen procesada será visible [E.6](#).

Gestión de modelos

Función de administrador. Los administradores, en el menú desplegable izquierdo, seleccionando la opción *modelos* E.7, pueden gestionar los modelos que usa la herramienta *radiografías*. Para seleccionar un modelo y activarlo basta con buscarlo en su lista correspondiente E.8 y seleccionar *activar*, el modelo activo aparece con un botón *activo* en rojo. Para subir un nuevo modelo se puede seleccionar la opción *nuevo modelo* en el menú izquierdo E.7 o en el botón a la derecha de cada lista E.8, desde ambas opciones se llega al mismo formulario de creación de modelo E.9.



The image shows a login window titled "Auto-Teeth". It features a purple header bar with the title. Below the header, there are two input fields: "Login" with a user icon and "Contraseña" with a lock icon. A blue link "Contraseña olvidada?" is positioned below the password field. In the bottom right corner, there is a "LOGIN" button.

Figura E.1: Ventana de login

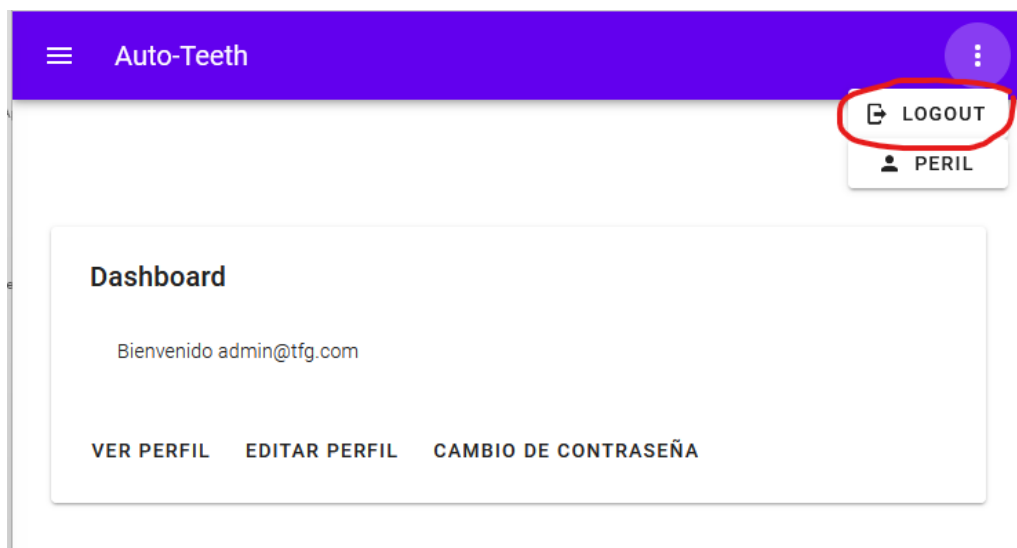


Figura E.2: Opción de logout

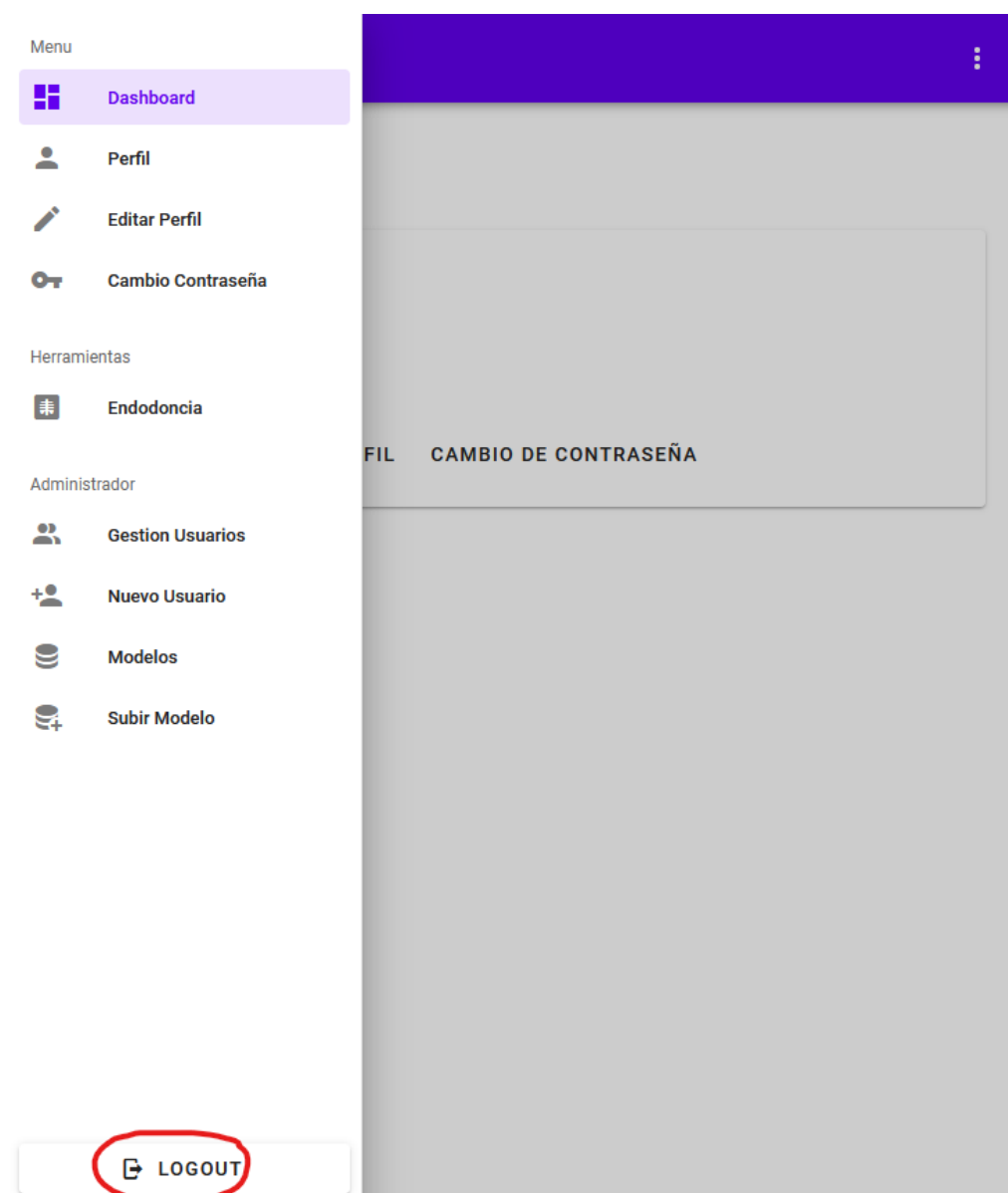


Figura E.3: Opción de logout en el menú

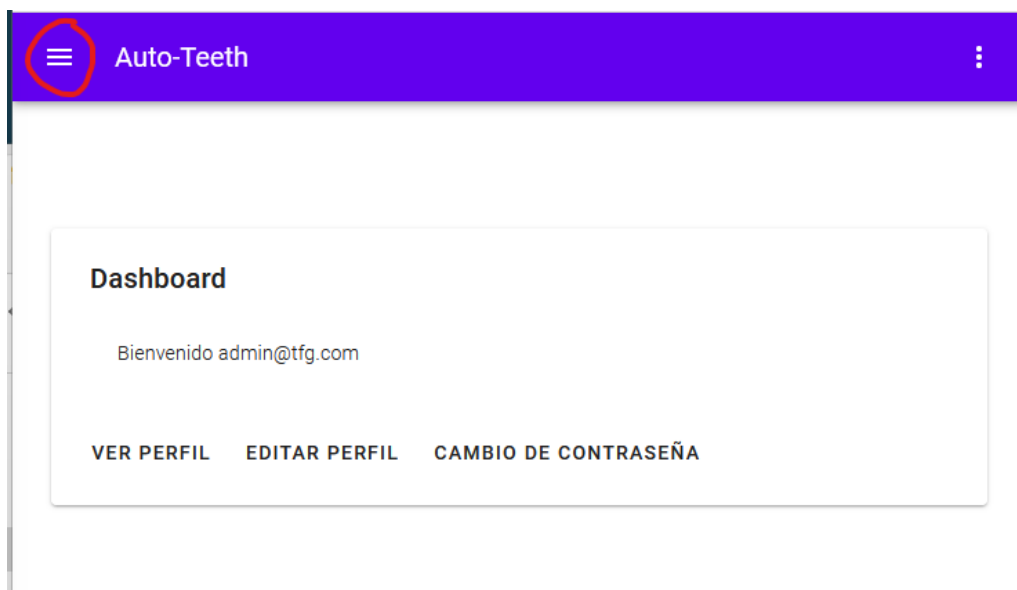


Figura E.4: Menú superior izquierdo

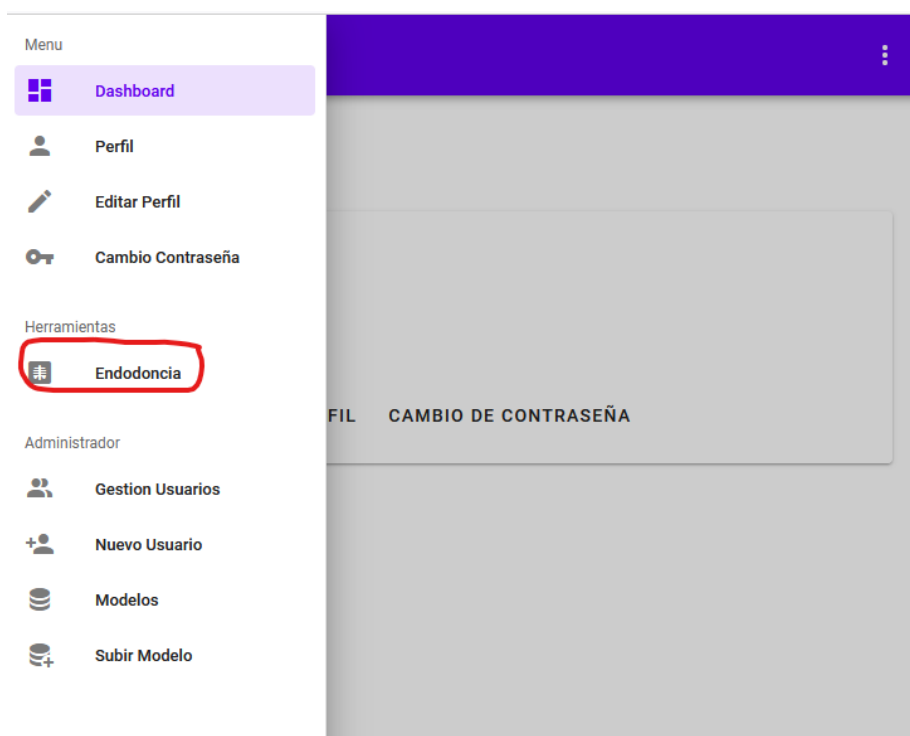


Figura E.5: Función de radiografías en el menú

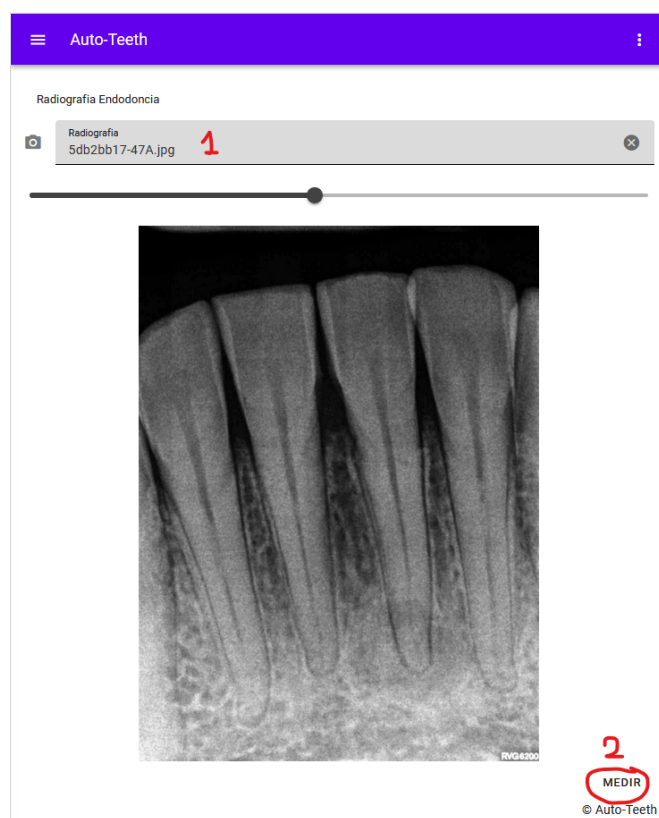


Figura E.6: Ventana de radiografía

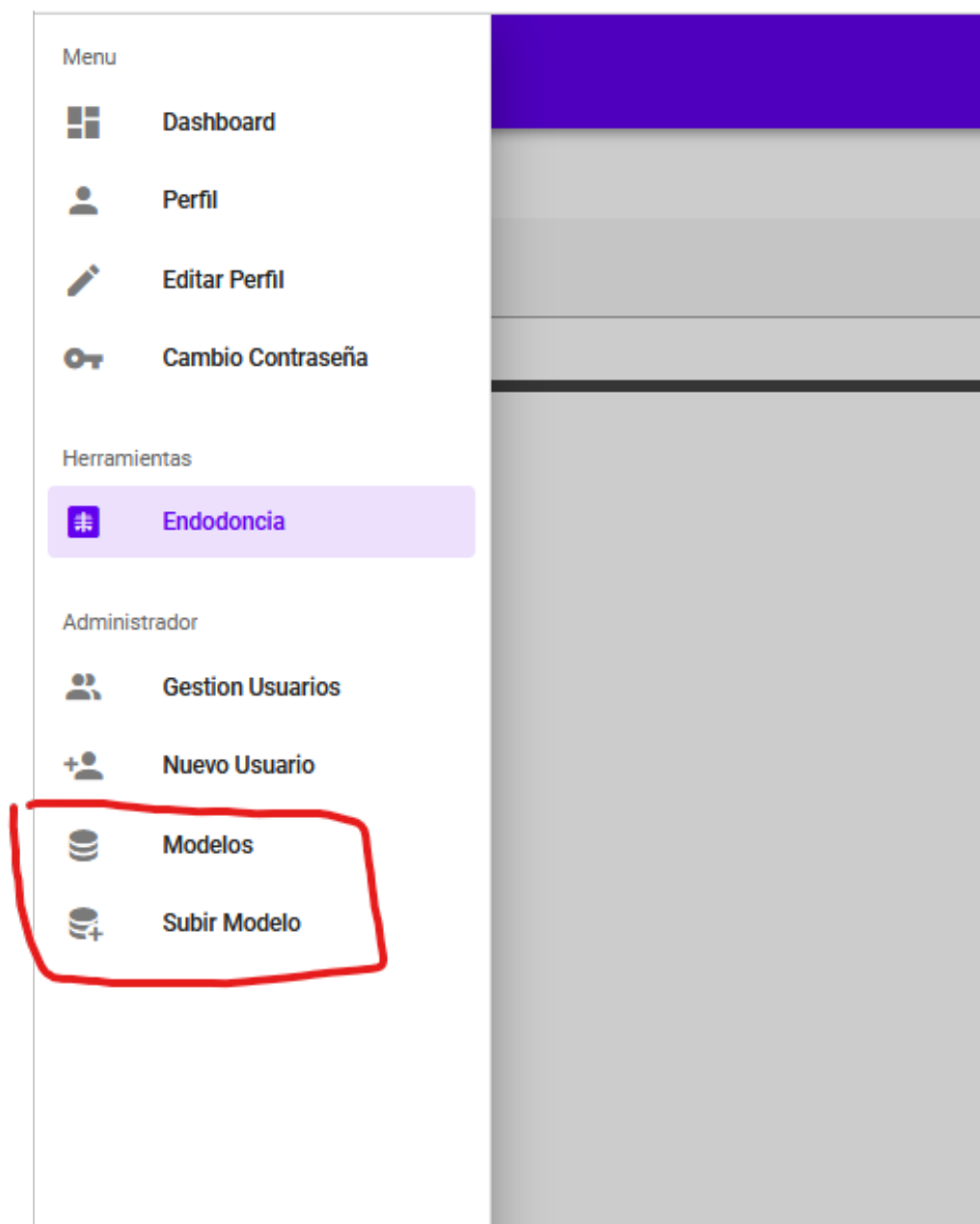


Figura E.7: Opciones para la gestión de modelos

Auto-Teeth

Modelos de Regresión

NUEVO MODELO

Nombre	Características	RMSE (mm)	Ubicacion	Activar	Eliminar
Resnet18	Modelo de regresion	12	./modelos_regresion/resnet34.onnx	ACTIVO	▼
ResNetV2	Modelo de pytorch basado en resnet 18 mejorado	1.2	./modelos_regresion/20230919_165607.onnx	ACTIVAR	<div><div></div></div> ▼

Items per page: 10 1-2 of 2

Modelos de Segmetnación

NUEVO MODELO

Nombre	Características	IOU (%)	Ubicacion	Activar	Eliminar
Yolo	Modelo de segmentación YOLO	0.8	./modelos_onnx/segmentacino_15.onnx	ACTIVO	▼

Items per page: 10 1-1 of 1

Figura E.8: Listas de modelos

Auto-Teeth

Subir modelo


REGRESION **SEGMENTACION**

Nombre

Descripcion breve

Detalles

RMSE
0 mm

 Modelo ONNX

CANCELAR **RESET** **GUARDAR**

Figura E.9: Formulario de creación de modelo

Bibliografía

- [1] FastAPI. <https://fastapi.tiangolo.com/>. Accedido el 1-9-2023.
- [2] PostgreSQL: License — postgresql.org. <https://www.postgresql.org/about/licence/>. Accedido el 7-9-2023.
- [3] SQLAlchemy — sqlalchemy.org. <https://www.sqlalchemy.org/>. Accedido el 7-9-2023.
- [4] Ultralytics | Revolutionizing the world of Vision AI. <https://ultralytics.com/license>. Accedido el 10-9-2023.
- [5] Vue.js - The Progressive JavaScript Framework | Vue.js. <https://vuejs.org/>. Accedido el 1-9-2023.
- [6] Welcome to Pydantic - Pydantic. <https://docs.pydantic.dev/latest/>. Accedido el 25-8-2023.
- [7] ¿Qué es una API REST? | IBM. <https://www.ibm.com/es-es/topics/rest-apis>. Accedido el 5-9-2023.
- [8] Métodos de petición HTTP - HTTP | MDN. <https://developer.mozilla.org/es/docs/Web/HTTP/Methods>, 7 2023. Accedido el 1-9-2023.
- [9] Pytorch. examples/LICENSE at main · pytorch/examples. <https://github.com/pytorch/examples/blob/main/LICENSE>. Accedido el 10-9-2023.
- [10] Tiangolo. fastapi/LICENSE at master · tiangolo/fastapi. <https://github.com/tiangolo/fastapi/blob/master/LICENSE>. Accedido el 10-9-2023.

- [11] Tornadomeet. ResNet/LICENSE.txt at master · tornadomeet/-ResNet. <https://github.com/tornadomeet/ResNet/blob/master/LICENSE.txt>. Accedido el 10-9-2023.
- [12] Vuejs. vue/LICENSE at main · vuejs/vue. <https://github.com/vuejs/vue/blob/main/LICENSE>. Accedido el 10-9-2023.