GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

APLICACIÓN PARA EL MAPEO INTELIGENTE, ANÁLISIS Y OPTIMIZACIÓN PREDICTIVA DE COBERTURA WI-FI EN ENTORNOS INTERIORES.

Autor: Ignacio López López

Director: Atilano Ramiro Fernández-Pacheco Sánchez-Migallón

Madrid

Declaración de originalidad

Declaro bajo mi responsabilidad que el Proyecto presentado con el título **Aplicación para el mapeo inteligente, análisis y optimización predictiva de cobertura wi-fi en entornos interiores** de la ETS de Ingeniería – ICAI de la Universidad Pontificia Comillas en el curso académico **4º de Grado en Ingeniería en Tecnologías de Telecomunicación** es de mi autoría y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Uso de Inteligencia Artificial[1]

Declaro bajo mi responsabilidad que (indicar la opción correcta):

☐ No he utilizado Inteligencia Artificial en la elaboración del presente documento.

X He utilizado Inteligencia Artificial en la elaboración del presente documento y/o del Anexo B siempre en las condiciones permitidas por la Universidad Pontificia Comillas, es decir, aplicando el Nivel 2 de la Escala de Evaluación de Perkins et al. (2024): "*La IA puede utilizarse para actividades previas a la tarea, como la lluvia de ideas, la descripción y la investigación inicial. Este nivel se centra en el uso de la IA para la planificación, las síntesis y la generación de ideas, pero las evaluaciones deben hacer hincapié en la capacidad de desarrollar y refinar estas ideas de forma independiente*". En concreto, las Inteligencia Artificial ha sido empleada para:

La Inteligencia Artificial ha sido empleada para lluvia de ideas, organización y estructura del presente documento, a modo de buscador y visualización.

Firmado (alumno): Ignacio López López
Fecha: 30/06/2026

[1] Esta declaración se refiere al uso de la Inteligencia Artificial generativa para realizar los documentos del Proyecto (Anexo B y Memoria). No aplica a Proyectos donde, por su naturaleza, deban emplear inteligencia artificial como parte de los mismos (aplicación de técnicas de aprendizaje automático, redes neuronales, análisis de datos…)

Autorización para la entrega del Proyecto

El Director del Proyecto	El co-Director del Proyecto (si aplica)
71216314 B ATILANO RAMIRO Signed by: 71216314B ATILANO RAMIRO FERNÁNDEZ-PACHECO Date: 2026-06-30 22:55:13 CEST	
Fdo:	Fdo:
Fecha:	Fecha:

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

APLICACIÓN PARA EL MAPEO INTELIGENTE, ANÁLISIS Y OPTIMIZACIÓN PREDICTIVA DE COBERTURA WI-FI EN ENTORNOS INTERIORES.

Autor: Ignacio López López

Director: Atilano Ramiro Fernández-Pacheco Sánchez-Migallón

Madrid

Agradecimientos

Terminar este proyecto cierra una de las etapas más bonitas e importantes de mi vida, no termina mi formación ya que espero que eso nunca acabe, pero significa un antes y un después en ella. Durante el Grado en Ingeniería en Tecnologías de Telecomunicación, he madurado, crecido, cambiado la forma de estudiar, de pensar y en definitiva, de ver el mundo, y eso es gracias a estudiar esta ingeniería. He conocido muchas personas de diferentes edades y cada una de ellas me ha aportado diferentes visiones. He coincidido con grandes profesores y personas, en especial mi tutor Atilano, que he coincidido dos años con él. Quiero agradecerle por enseñarme de una manera entretenida y ser el tutor de este de este proyecto.

También quiero dar las gracias a mi familia, que me ha apoyado en los buenos y no tan buenos momentos por los que te hace pasar este grado. Me han entendido siempre y han hecho que sean más llevaderos aquellos momentos de agobio y de trabajo.

APLICACIÓN PARA EL MAPEO INTELIGENTE, ANÁLISIS Y OPTIMIZACIÓN PREDICTIVA DE COBERTURA WI-FI EN ENTORNOS INTERIORES.

Autor: Ignacio López
Director: Atilano Ramiro Fernández-Pacheco Sánchez-Migallón
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

Este proyecto consiste en el desarrollo de una aplicación basada en microservicios para la estimación y el análisis predictivo de la cobertura Wi-Fi en entornos interiores. A partir únicamente del plano del entorno y de la posición de los puntos de acceso, un modelo de aprendizaje profundo de tipo U-Net genera un mapa de cobertura estimado, obtenido con un error medio inferior al decibelio y de forma prácticamente instantánea, lo que acerca al gran público una capacidad reservada hasta ahora a herramientas profesionales.

Palabras clave: WiFi, cobertura en interiores, aprendizaje profundo, U-Net, microservicios, *frontend, backend*.

1. Introducción

La tecnología WiFi se ha convertido en la principal forma de acceso a Internet en entornos interiores, donde el número de dispositivos conectados no deja de crecer. Sin embargo, la calidad percibida por el usuario no depende solo de la velocidad contratada, sino de cómo se distribuye realmente la señal dentro del espacio, condicionada por las paredes, los obstáculos y la propia ubicación de los puntos de acceso. Disponer de una representación visual de la cobertura ayuda a decidir dónde colocar un repetidor o desde qué habitación conectarse y contribuye a un uso más responsable del espectro, al evitar aumentar la potencia o instalar más equipos de los necesarios. Por este motivo, el presente Trabajo de Fin de Grado propone una solución orientada al gran público que permita visualizar de forma intuitiva la cobertura WiFi en interiores mediante mapas de calor, acercando a cualquier usuario una capacidad reservada hasta ahora a herramientas profesionales.

2. Definición del proyecto

Los modelos físicos clásicos, desde la fórmula de Friis hasta las recomendaciones ITU-R P.1238 para interiores, permiten calcular las pérdidas de propagación, pero resultan demasiado complejos para un usuario sin formación técnica. La simulación física por ray tracing ofrece una gran precisión a costa de un elevado coste computacional, y las herramientas comerciales de planificación WiFi (NetSpot, Ekahau) exigen planos detallados, mediciones in situ y, a menudo, licencias caras. Frente a estos enfoques, la estimación de mapas de radio mediante aprendizaje profundo plantea la predicción de cobertura como un problema de generación de imágenes, con la arquitectura U-Net como referencia. El objetivo del proyecto es desarrollar una aplicación que estime mapas de cobertura WiFi en interiores a partir únicamente del plano del entorno y de la posición de los puntos de acceso, empleando

un modelo de aprendizaje profundo integrado en una arquitectura software completa y desplegada en la nube.

3. Descripción del sistema y del modelo

El sistema se ha construido siguiendo una arquitectura de microservicios, en la que cada módulo emplea la tecnología más adecuada y se despliega de forma independiente, comunicándose mediante APIs REST sobre HTTP.

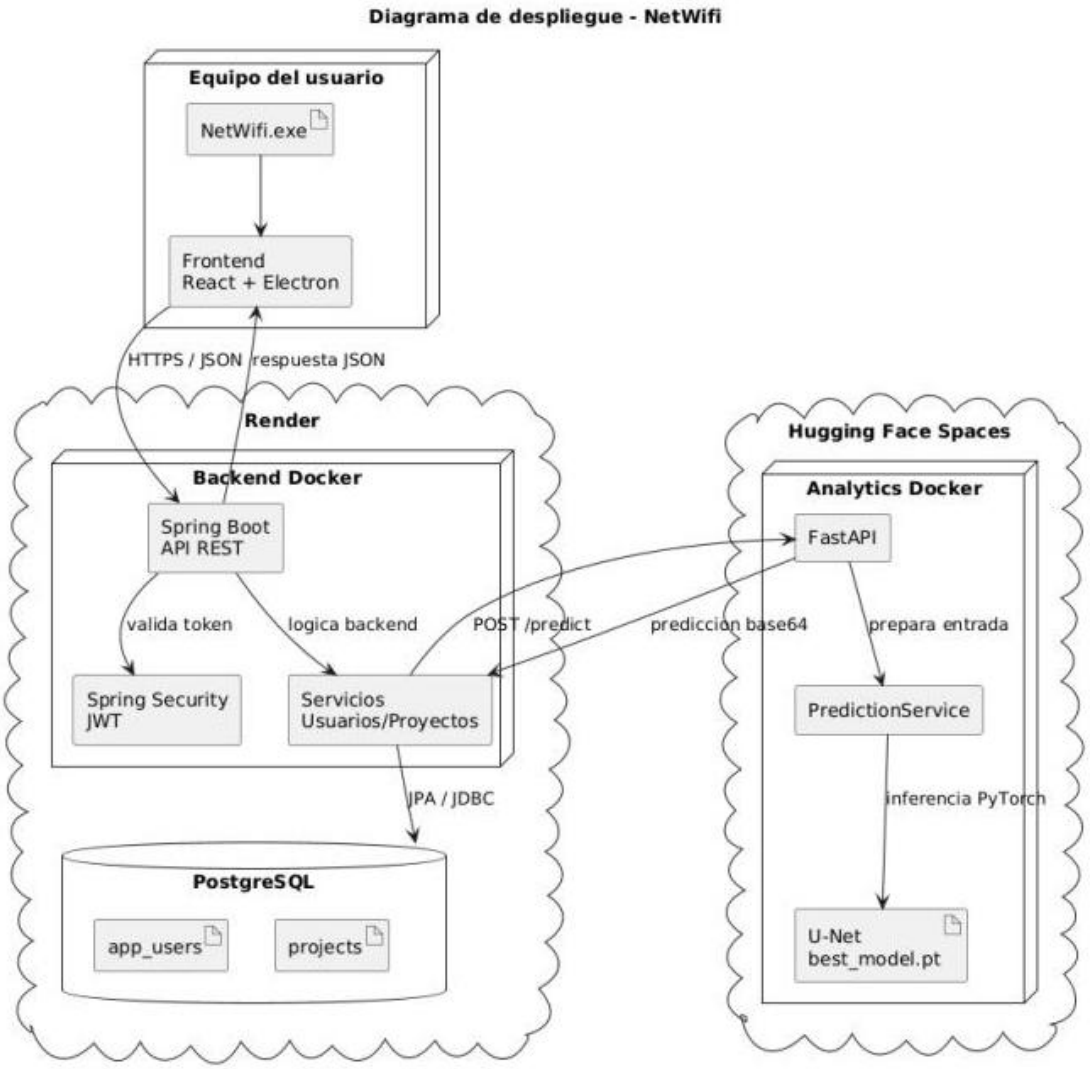

Figura 1. Diagrama de despliegue de la arquitectura de microservicios.

El sistema se compone de tres servicios y un conjunto de datos. La aplicación de escritorio (frontend), desarrollada con React, Vite y Electron y distribuida como ejecutable .exe, concentra toda la interacción con el usuario, desde el dibujo del plano y la colocación de los puntos de acceso hasta un test de red WiFi en tiempo real. El backend (Spring Boot), desplegado con Docker en Render, actúa como orquestador y gestiona la autenticación mediante JWT y la persistencia. El motor analítico (FastAPI + PyTorch), alojado en Hugging Face Spaces, se especializa en la inferencia, y una base de datos PostgreSQL persiste

usuarios y proyectos. Así, el modelo queda aislado tras el backend, donde se concentra la seguridad.

El modelo se ha entrenado con un conjunto de datos público de mapas de cobertura simulados en la banda de 5 GHz siguiendo el modelo de canal IEEE 802.11ax, representados como imágenes de 256×256 píxeles, con configuraciones de uno a tres puntos de acceso. La entrada del modelo apila dos canales, el plano y la máscara de puntos de acceso, de modo que predecir la cobertura es un problema imagen a imagen. Para resolverlo se ha implementado en PyTorch una red U-Net, una arquitectura convolucional codificador-decodificador con conexiones de salto que recupera el detalle posicional y produce un mapa nítido y alineado con el plano de entrada. Con 48 canales base y unos 43,4 millones de parámetros, se ha entrenado de forma supervisada minimizando el error cuadrático medio (MSE) con el optimizador AdamW durante 33 épocas.

4. Resultados

El modelo seleccionado se ha evaluado sobre el conjunto de test, comparando píxel a píxel cada predicción con su mapa objetivo. Teniendo en cuenta que cada mapa abarca un rango dinámico de unos 121 dB, el error medio se sitúa por debajo del decibelio, con un MAE ≈ 0,26 dB y un RMSE ≈ 0,70 dB, y disminuye al aumentar el número de puntos de acceso, ya que la cobertura se reparte de forma más homogénea.

Figura 2. Ejemplo cualitativo con el plano y la máscara de entrada, el mapa simulado, la predicción del modelo y el error.

La comparación visual confirma estas cifras, ya que la predicción reproduce con fidelidad la forma del mapa simulado, el degradado de potencia alrededor de cada emisor y la atenuación que introducen las paredes, concentrándose el error residual únicamente en los bordes. Una de las principales ventajas frente a la simulación física es la rapidez, ya que el modelo genera un mapa completo de 256×256 en torno a 9 ms sobre la GPU empleada (NVIDIA RTX 3060 Ti), y más rápido aún con el modelo desplegado, una predicción prácticamente instantánea frente al coste muy superior del *ray tracing*.

5. Conclusiones

El objetivo general se ha alcanzado de forma satisfactoria, con una aplicación funcional y desplegada en la nube, articulada sobre una arquitectura de microservicios, capaz de estimar la cobertura WiFi de un entorno interior a partir únicamente de su plano y de la posición de los puntos de acceso. Los resultados confirman que la U-Net reproduce con fidelidad el comportamiento del simulador, con un error medio inferior al decibelio y una predicción

casi instantánea. El principal logro del trabajo es haber integrado un modelo de aprendizaje profundo dentro de un sistema software completo y accesible. Como líneas de trabajo futuro se plantean la validación frente a medidas reales, un optimizador automático de la ubicación de los puntos de acceso y la ampliación a más configuraciones y bandas de frecuencia.

6. Referencias

[1] Ronneberger, O., Fischer, P., Brox, T. «U-Net: Convolutional Networks for Biomedical Image Segmentation». Springer International Publishing, pp. 234-241, 2015.

[2] Pyo, C., Sawada, H., Matsumura, T. «A Deep Learning-Based Indoor Radio Estimation Method Driven by 2.4 GHz Ray-Tracing Data». IEEE Access, 2023.

[3] Fernández-Gómez, A. J., Valenzuela-Marín, C. A. «Fast Indoor Radio Propagation Prediction Using Deep Learning». Zenodo, 2023.

[4] Unión Internacional de Telecomunicaciones (ITU-R). «Recomendación UIT-R P.1238».

[5] Kingma, D. P., Ba, J. «Adam: A Method for Stochastic Optimization». ICLR, 2015.

APPLICATION FOR THE INTELLIGENT MAPPING, ANALYSIS AND PREDICTIVE OPTIMISATION OF WI-FI COVERAGE IN INDOOR ENVIRONMENTS

Author: Ignacio López
Supervisor: Atilano Ramiro Fernández-Pacheco Sánchez-Migallón
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

This project consists of the development of a microservices-based application for the estimation and predictive analysis of Wi-Fi coverage in indoor environments. From nothing more than the floor plan of the environment and the position of the access points, a U-Net deep learning model generates an estimated coverage map, obtained with an average error below one decibel and almost instantaneously, bringing to the general public a capability so far reserved for professional tools.

Keywords: WiFi, indoor coverage, deep learning, U-Net, microservices, frontend, backend.

1. Introduction

WiFi has become the main way of accessing the Internet in indoor environments, where the number of connected devices keeps growing. However, the quality perceived by the user does not depend solely on the contracted speed, but on how the signal is actually distributed throughout the space, conditioned by walls, obstacles and the location of the access points. Having a visual representation of coverage helps to decide where to place a repeater or from which room to connect, and contributes to a more responsible use of the spectrum by avoiding unnecessarily increasing transmission power or installing more equipment than required. For this reason, this Bachelor's Thesis proposes a solution aimed at the general public that allows WiFi coverage in indoor environments to be visualised intuitively through heat maps, bringing to any user a capability so far reserved for professional tools.

2. Project definition

Classical physical models, from the Friis equation to the ITU-R P.1238 recommendations for indoor environments, allow propagation losses to be computed, but they are too complex for a user without technical training. Physical simulation through ray tracing offers high accuracy at the cost of a high computational load, and commercial WiFi planning tools (NetSpot, Ekahau) require detailed floor plans, on-site measurements and, frequently, expensive licences. In contrast, Radio Map Estimation through deep learning frames coverage prediction as an image-generation problem, with the U-Net architecture as a reference. The objective of the project is to develop an application that estimates indoor WiFi coverage maps from nothing more than the floor plan of the environment and the position of

the access points, using a deep learning model integrated within a complete software architecture deployed in the cloud.

3. System and model description

The system has been built following a microservices architecture, in which each module uses the most suitable technology and is deployed independently, communicating through REST APIs over HTTP.

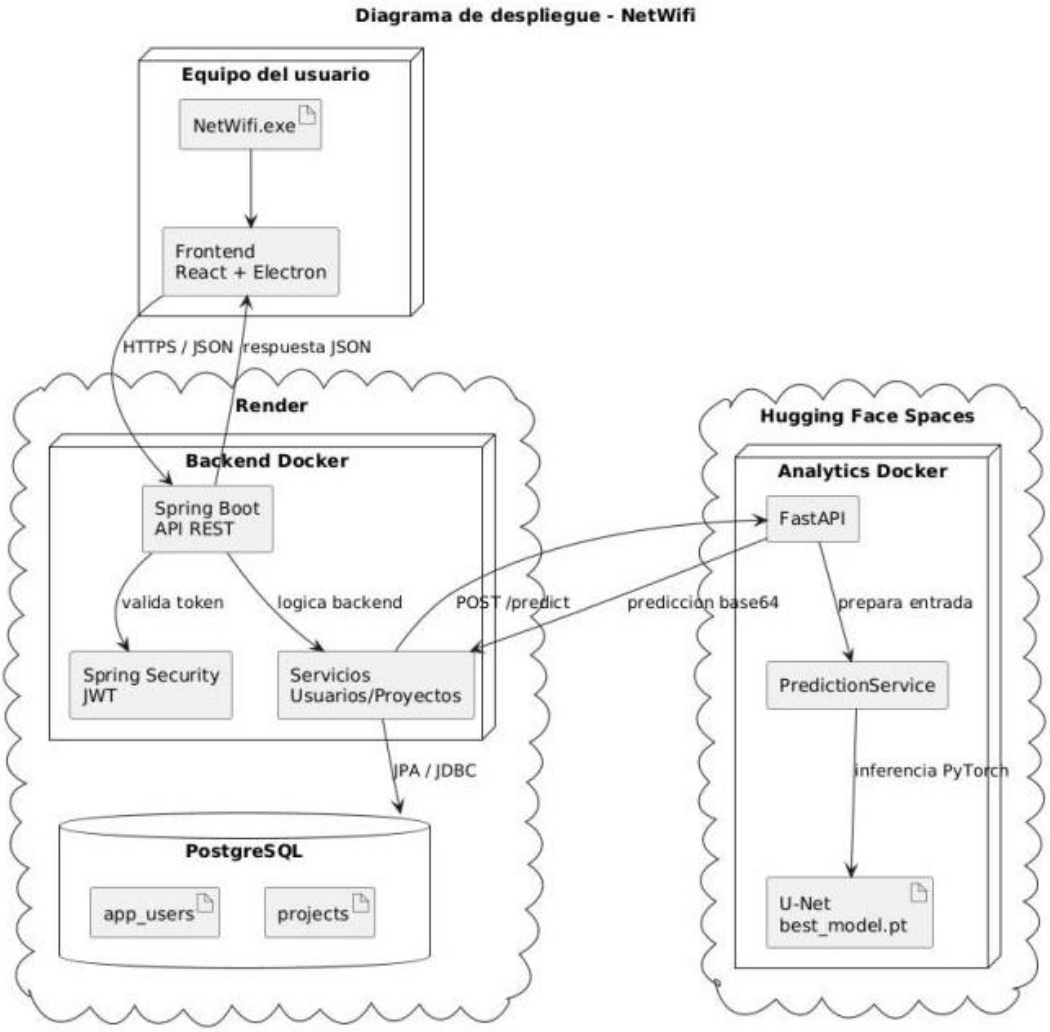

Figure 1. Deployment diagram of the microservices architecture.

The system is composed of three services and a dataset. The desktop application (frontend), developed with React, Vite and Electron and distributed as an .exe executable, concentrates all user interaction, from drawing the floor plan and placing the access points to a real-time WiFi network test. The backend (Spring Boot), deployed with Docker on Render, acts as an orchestrator and handles JWT-based authentication and persistence. The analytics engine (FastAPI + PyTorch), hosted on Hugging Face Spaces, is dedicated to inference, and a PostgreSQL database persists users and projects. In this way the model remains isolated behind the backend, where security is concentrated.

The model has been trained with a public dataset of coverage maps simulated in the 5 GHz band following the IEEE 802.11ax channel model, represented as 256×256 images, with configurations of one to three access points. The model input stacks two channels, the floor plan and the access-point mask, so that predicting coverage is an image-to-image problem. To solve it, a U-Net has been implemented in PyTorch, a convolutional encoder-decoder architecture with skip connections that recovers the positional detail and produces a sharp map aligned with the input floor plan. With 48 base channels and around 43.4 million parameters, it has been trained in a supervised manner by minimising the mean squared error (MSE) with the AdamW optimiser over 33 epochs.

4. Results

The selected model has been evaluated on the test set, comparing each prediction pixel by pixel with its target map. Bearing in mind that each map spans a dynamic range of about 121 dB, the average error stays below one decibel, with an MAE ≈ 0.26 dB and an RMSE ≈ 0.70 dB, and it decreases as the number of access points increases, since coverage is distributed more homogeneously.

Figure 2. Qualitative example with the input floor plan and mask, the simulated map, the model prediction and the error.

The visual comparison confirms these figures, since the prediction faithfully reproduces the shape of the simulated map, the power gradient around each emitter and the attenuation introduced by the walls, with the residual error concentrated only at the edges. One of the main advantages over physical simulation is speed, since the model generates a complete 256×256 map in around 9 ms on the GPU used (NVIDIA RTX 3060 Ti), even better with the deployed model, practically instantaneous prediction compared with the much higher cost of ray tracing.

5. Conclusions

The general objective has been achieved successfully, with a functional, cloud-deployed application, built on microservice architecture, capable of estimating the WiFi coverage of an indoor environment from nothing more than its floor plan and the position of the access points. The results confirm that the U-Net faithfully reproduces the behavior of the simulator, with an average error below one decibel and a nearly instantaneous prediction. The main achievement of the work is having integrated a deep learning model within a complete and accessible software system. Future lines of work include validation against real measurements, an automatic optimizer of access-point placement and the extension to more configurations and frequency bands

6. References

[1] Ronneberger, O., Fischer, P., Brox, T. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. Springer International Publishing, pp. 234-241, 2015.

[2] Pyo, C., Sawada, H., Matsumura, T. “A Deep Learning-Based Indoor Radio Estimation Method Driven by 2.4 GHz Ray-Tracing Data”. IEEE Access, 2023.

[3] Fernández-Gómez, A. J., Valenzuela-Marín, C. A. “Fast Indoor Radio Propagation Prediction Using Deep Learning”. Zenodo, 2023.

[4] International Telecommunication Union (ITU-R). “Recommendation ITU-R P.1238”.

[5] Kingma, D. P., Ba, J. “Adam: A Method for Stochastic Optimization”. ICLR, 2015.

Índice de la memoria

Índice de figuras

Capítulo 1. INTRODUCCIÓN

Desde la aparición de las redes inalámbricas de área local o WLAN (*Wireless Local Area Network*), basadas en tecnología de acceso inalámbrico a redes de comunicaciones digitales, se ha experimentado una evolución constante en la forma en que accedemos a Internet. Estas redes han logrado sustituir en muchos casos a las conexiones de cableado tradicionales permitiendo más flexibilidad y comodidad a los usuarios de la red.

Dentro de este tipo de redes nace el estándar técnico IEEE 802.11 que define los diferentes parámetros de las comunicaciones inalámbricas (normalmente viviendas u oficinas) como el uso exacto de la banda de frecuencia, velocidad y seguridad de la red. Sobre este estándar nace el término WiFi que surge como una marca de certificación de conformidad con dicho estándar impulsado por WIFI Alliance. Es conocido mundialmente y de ahora en adelante se utilizará para hacer referencia a la mencionada tecnología inalámbrica. [1]

En los últimos años, la tecnología WiFi ha ganado importancia sobre todo en entornos interiores debido al crecimiento de las necesidades de los usuarios. Ha aumentado el número de dispositivos con necesidad de conexión, así como la necesidad de una conexión estable, rápida y segura. Piense en la cantidad de dispositivos que puede tener una vivienda, ordenadores, teléfonos móviles, televisores, cámaras, electrodomésticos, dispositivos de domótica, etc.

Por este motivo, disponer de una representación visual de la cobertura WiFi ayuda al usuario a tomar decisiones fundamentales como la ubicación de dispositivos fijos tales como repetidores, o, por otra parte, el lugar de la vivienda donde asistir a una reunión online. Gracias a ello, el usuario no depende solo de la intuición. Por otra parte, esto permite hacer un uso adecuado del espectro. Una ubicación incorrecta de los amplificadores puede provocar la necesidad de aumentar la potencia de transmisión de forma innecesaria o llevar a instalar un número inadecuado de dispositivos, generando interferencias con otros usuarios de la red. Este proyecto nace también para cubrir esa necesidad, permitiendo a los usuarios hacer un uso responsable del espectro WiFi.

Es por ello que, en el presente Trabajo de Fin de Grado, se propone el desarrollo de una solución orientada al gran público que permita visualizar de forma intuitiva la cobertura WiFi en entornos interiores mediante la generación de mapas de calor. El objetivo principal es acercar herramientas tradicionalmente reservadas para entornos profesionales a cualquier usuario, facilitando la comprensión del comportamiento de su red inalámbrica y ayudándole a optimizar la ubicación de *routers*, repetidores o puntos de acceso.

Para alcanzar este objetivo, se ha diseñado e implementado una aplicación basada en una arquitectura de microservicios, lo que permite una mayor modularidad, escalabilidad y mantenibilidad del sistema. Esta arquitectura facilita la separación de responsabilidades entre los distintos componentes de la aplicación que se comunican entre si mediante el uso de APIs.

Asimismo, el proyecto integra tecnologías utilizadas en la industria del software actual. Por un lado, se emplea JavaScript para el desarrollo de la interfaz de usuario y de diversos servicios de la aplicación, permitiendo ofrecer una experiencia interactiva y accesible. Por otro lado, se utiliza Python para el desarrollo de un modelo de aprendizaje profundo encargado de estimar y predecir la cobertura WiFi en función de las características del entorno. Este modelo será una U-Net que tomará una imagen como entrada y generará otra imagen espacialmente alineada como salida. Por tanto, se dispondrá de una base de datos de imágenes para entrenar el modelo.

La combinación de estas tecnologías permite desarrollar una herramienta moderna, visual e intuitiva, capaz de ayudar al usuario a comprender y mejorar la cobertura WiFi de su entorno interior.

En cuanto a la estructura del documento, este se organiza en varios capítulos. En primer lugar, se describen las tecnologías, herramientas que se van a utilizar a lo largo del proyecto para ayudar a una mejor comprensión del resto del trabajo por parte del lector. A continuación, se desarrolla el estado del arte y posteriormente se define el trabajo. Se hará estudio de las soluciones actuales en el ámbito del proyecto y argumentará la importancia de este trabajo. A ello le seguirá la explicación del modelo que se ha desarrollado. Se explicará en detalle características tales como el análisis y diseño del sistema y su implementación. Finalmente, se llevará a cabo el análisis de los resultados obtenidos y finalizará el proyecto con las conclusiones, posibles mejoras, bibliografía y anexos.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

En este capítulo se describen las principales tecnologías utilizadas durante el desarrollo del proyecto. El objetivo es proporcionar al lector una base técnica suficiente para comprender las decisiones tomadas en los capítulos posteriores, tanto en la parte correspondiente al desarrollo de la aplicación como en la parte asociada al modelo de predicción de cobertura WiFi.

El proyecto combina tecnologías propias del desarrollo software con herramientas utilizadas en el ámbito del aprendizaje automático. Por una parte, se emplean tecnologías orientadas a la creación de una aplicación interactiva, formada por una interfaz de usuario (*frontend*), un *backend* y una base de datos. Por otra parte, se desarrolla un módulo específico de aprendizaje profundo encargado de procesar la información del entorno y generar una estimación de la cobertura inalámbrica.

2.1 TECNOLOGÍAS DE FRONTEND

En primer lugar, se presentan las tecnologías utilizadas en el frontend de la aplicación. Esta parte del sistema es la encargada de permitir la interacción directa con el usuario, por lo que resulta fundamental que sea sencilla, clara y suficientemente flexible para representar los elementos necesarios del proyecto. Para ello se han utilizado React, Vite y Electron.

2.1.1 REACT

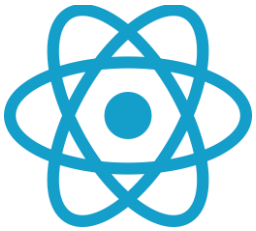

Figura 1 Logotipo React

React es una librería de JavaScript orientada al desarrollo de interfaces de usuario. Su funcionamiento se basa en la creación de componentes, que son bloques reutilizables encargados de representar una parte concreta de la interfaz.

Una de las características más habituales de React es el uso de JSX, una extensión de sintaxis de JavaScript que permite escribir estructuras similares a HTML dentro del propio código JavaScript. De esta forma, cada componente puede definir tanto su lógica como la estructura visual que se renderiza en pantalla. Aunque JSX se parece a HTML, no es exactamente lo mismo, ya que se integra directamente con JavaScript y permite incluir variables, condiciones o funciones dentro de la interfaz. [2]

Para sustituir el uso de hojas de estilo CSS que en ocasiones provocan errores y se convierten en ficheros interminables, se ha optado por el uso de MUI, que es una librería de componentes de React. Esta proporciona componentes con estilos ya preparados, tales como botones o *cards*, en los cuales se permiten pequeñas modificaciones para que se muestren como el desarrollador desea, evitando así CSS's demasiado complejos.

Otra característica importante de React es la gestión del estado de la aplicación. El estado representa la información que puede cambiar durante la ejecución, como los datos introducidos por el usuario, las opciones seleccionadas o en mi caso, el dibujo de los entornos interiores de cada usuario. Cuando este estado cambia, React actualiza las partes de la interfaz que dependen de él, evitando tener que modificar manualmente cada elemento de la página. Esto convierte muchas líneas de código en una solución elegante, en el que se detecta el cambio en una variable y salta la acción correspondiente.

2.1.2 VITE

Figura 2 Logotipo de Vite

Vite es una herramienta de desarrollo para aplicaciones frontend. Su función principal es proporcionar un entorno rápido para programar, ejecutar y construir aplicaciones basadas en tecnologías web modernas.

Durante el desarrollo de una aplicación web, es habitual realizar muchos cambios sobre el código y comprobar su efecto en la interfaz. Vite permite que este proceso sea

más ágil, ya que actualiza la aplicación de forma rápida cada vez que se modifica un archivo. Esto mejora la experiencia de desarrollo y reduce el tiempo necesario para probar cambios en la interfaz.

Además, Vite también se encarga del proceso de construcción de la aplicación. Una vez finalizado el desarrollo, genera los archivos optimizados que posteriormente pueden ser ejecutados o integrados dentro de otro entorno. En este proyecto, esta funcionalidad resulta útil porque el frontend no se plantea únicamente como una página web tradicional, sino como una parte de una aplicación de escritorio construida con Electron.

2.1.3 ELECTRON

Figura 3 Logotipo Electron

Electron es una tecnología que permite desarrollar aplicaciones de escritorio utilizando tecnologías web como JavaScript, HTML y CSS. Su principal ventaja es que permite reutilizar una interfaz desarrollada con herramientas web y ejecutarla como una aplicación local en el ordenador del usuario.

En una aplicación web, el usuario accede a la interfaz mediante un navegador. Sin embargo, con Electron la aplicación puede abrirse como un programa independiente, ofreciendo una experiencia más cercana a la de una aplicación de escritorio.

Electron se encarga de crear una ventana de escritorio donde se carga la interfaz desarrollada con React. De esta forma, la parte visual de la aplicación mantiene las ventajas del desarrollo web, pero se presenta al usuario como una aplicación instalada y ejecutable desde su propio equipo.

Esto resulta adecuado en este proyecto ya que se acceden a datos de señal WiFi sensibles que la mayoría de los navegadores bloquea por protección de datos frente a

vulnerabilidades. Estos datos son el estado de la conexión WiFi, Rssi en dbm , porcentaje de señal, banda de frecuencia, velocidades RX/TX, latencia, y la calidad de señal.

2.1.4 NODE.JS Y NPM

Figura 4 Logotipos Node.js y npm

Node.js es un entorno de ejecución de JavaScript multiplataforma y de código abierto que permite ejecutar código JavaScript fuera del navegador [3]. Aunque tradicionalmente JavaScript se asociaba principalmente al desarrollo de páginas web ejecutadas en el navegador, Node.js permite ejecutar JavaScript fuera de el.

En este proyecto, Node.js se utiliza como base del entorno de desarrollo del frontend. Herramientas como Vite, React y Electron se apoyan en el ecosistema de JavaScript moderno, por lo que requieren disponer de Node.js instalado para poder ejecutar los comandos de desarrollo, construir la aplicación y gestionar sus dependencias

Junto con Node.js se utiliza npm, que es el gestor de paquetes asociado al ecosistema JavaScript. Este permite instalar, actualizar y gestionar las librerías necesarias para el proyecto. Estas dependencias quedan definidas en el archivo package.json, donde se indican los paquetes utilizados y las versiones correspondientes. Gracias a npm, puedes instalar librerías como React, MUI, Vite o Electron sin tener que descargar y configurar manualmente cada una de ellas. [4]

Los comandos utilizados han sido:

```
npm install #Instala dependencias (React, Vite, Electron, Material UI...)

npm run electron:dev  #Arranca la app en desarrollo (Levanta Vite en
localhost:5173)

npm ru  n electron:build #Compila React y usa electron-builder que crea el
instalador (.exe)
```

2.2 TECNOLOGIAS DE BACKEND

El *backend* es la parte de la aplicación encargada de gestionar la lógica de negocio, conexión con la base de datos, y de hacer una segunda verificación de los datos que van a ser persistidos. Para ello, coordina la comunicación con otros módulos de la aplicación mediante el procesamiento de peticiones, y es ahí cuando valida, verifica, gestiona usuarios, funcionalidades, permisos y persistencia.

2.2.1 SPRINGBOOT

Figura 5 Logotipo Spring Boot

Spring Boot es un framework basado en Java que facilita la creación de aplicaciones independientes y servicios backend listos para ser ejecutados [5]. Tiene dos funciones principales. Por un lado, reducir la configuración inicial necesaria en proyectos desarrollados con el ecosistema Spring, permitiendo al desarrollador centrarse en la lógica. Por otro lado, su organización en capas es una gran ventaja. Habitualmente, y no distinto en este proyecto, estas capas son: el controlador, el servicio, el modelo y el repositorio. Este último realiza la conexión con la base de datos. Para identificar cada capa, se utilizan anotaciones como `@Service` que sirven para definir el propósito de la clase y permitir que Spring la gestione automáticamente.

2.2.2 MAVEN

Figura 6 Logotipo de Maven

Apache Maven es una herramienta de compilación para proyectos Java. Mediante un modelo de objetos de proyecto (POM), Maven gestiona la compilación, las pruebas y la documentación de un proyecto [6].

Una de las funciones más importantes de Maven es la gestión de dependencias. En un proyecto *backend* es habitual utilizar librerías externas, como las relacionadas con Spring Boot, seguridad, acceso a base de datos o validación de datos. Maven permite declarar estas dependencias en el archivo `pom.xml` y se encarga de descargarlas y hacerlas disponibles para el proyecto, evitando tener que gestionarlas manualmente. Además, se encarga de todo el ciclo de vida del proyecto, ya que también se encarga de verificar los *tests* antes del lanzamiento del *backend*.

2.2.3 RENDER Y DOCKERFILE

Render es una plataforma en la nube que permite desplegar aplicaciones, servicios web y bases de datos gestionadas. En este proyecto, se ha utilizado para alojar tanto el *backend* de la aplicación como la base de datos PostgreSQL utilizada por el sistema.

Por un lado, el *backend* se despliega en Render, que gestiona las claves necesarias como por ejemplo las de la base de datos, ejecuta el servicio en la nube permitiendo que pueda recibir peticiones HTTP desde el *frontend*. Para logar este despliegue, se ha utilizado Docker como mecanismo de empaquetado. Para ello, se creó un archivo `Dockerfile`, que se encarga de ejecutar y construir la aplicación Spring Boot dentro de un contenedor. Con ese archivo, Docker puede construir la imagen personalizada y estandarizada. Puedes seleccionar exactamente que versión de Java se ha de utilizar para ejecutar el proyecto así como los comandos necesarios para lograrlo, evitando así problemas de versionado, consiguiendo ejecutar el proyecto fuera del entorno local.

El `Dockerfile` utilizado en ese proyecto es el siguiente:

```
FROM eclipse-temurin:21-jdk AS build
WORKDIR /app

COPY . .
RUN chmod +x mvnw && ./mvnw clean package -DskipTests

FROM eclipse-temurin:21-jre
WORKDIR /app

COPY --from=build /app/target/*.jar app.jar

EXPOSE 8080
CMD ["java", "-jar", "app.jar"]
```

Código 1 DockerFile

Primero, se indica la imagen del contenedor que se utilizará, en mi caso Java 21 JDK por su robustez. A continuación, mediante el comando `RUN`, ejecuta el comando escrito para darle permisos de ejecución al archivo `mvnw`, limpiar las compilaciones anteriores que pudiera tener, compilar de nuevo y empaquetar el archivo. Una vez compilado, se quiere ejecutar la aplicación (por eso se utiliza `jre`). Por último, se cambia el nombre de la aplicación, se indica el puerto interno que utilizará y finalmente, mediante la instrucción `CMD`, se indica el comando que se ejecutará por defecto al iniciar el contenedor.

Esto último es necesario porque la opción escogida en Render para desplegar el *backend*, es gratuita, con el inconveniente de que el servicio entra reposo tras un periodo de inactividad, y, por tanto, la primera petición que recibe arranca otra vez el servicio.

Por último, también Render se utiliza para alojar la base de datos PostgreSQL. Esto permite que la información de la aplicación se almacene en un entorno remoto y no dependa únicamente de una instalación local.

2.2.4 API REST Y PROTOCOLO HTTP

Una API REST es una interfaz de programación de aplicaciones (API) que se ajusta a los principios de diseño del estilo arquitectónico de transferencia de estado representacional (REST), un estilo utilizado para conectar sistemas de hipermedia distribuidos.

Definido en 2000 por el informático DRr. Roy Fielding en su tesis doctoral, REST ofrece a los desarrolladores un nivel relativamente alto de flexibilidad, coherencia, escalabilidad y eficiencia. Las API REST proporcionan una forma ligera de construir API web y se utilizan habitualmente para facilitar el intercambio de datos entre aplicaciones, servicios web y bases de datos, y para conectar componentes en arquitecturas de microservicios. [7]

Este tipo de comunicación entre servicios se basa en el protocolo HTTP (*Hypertext Transfer Protocol*), que es lo utilizado en el proyecto. Este protocolo de comunicaciones basado en el principio cliente-servidor permite transferir información (texto, imagen y video) entre diferentes servicios.

En este proyecto, tanto el *frontend* como el motor analítico de Python con *FastApi* hacen uso de esta tecnología para comunicarse con el *backend*. De esta forma hay un intermediario que añade una capa de seguridad y robustez a la aplicación, ya que por ejemplo, el *frontend* no se conecta directamente con la base de datos, si no que se utiliza el servidor de Spring Boot a modo de orquestador. Quiero destacar, que por la naturaleza de esta interfaz de comunicaciones, todos los datos viajan en formato JSON. Los métodos utilizados en este proyecto han sido:

- GET solicita o recupera información del servidor
- POST envía nuevos datos para crear un recurso
- PUT actualiza o reemplaza un recurso existente
- DELETE elimina un recurso determinado.

2.2.5 POSTGRESQL

Figura 7 Logotipo PostgreSQL

Esta tecnología pertenece a la capa de persistencia de las tecnologías de *backend.*

PostgreSQL es un sistema de gestión de bases de datos objeto-relacionales (ORDBMS) basado en POSTGRES , desarrollado en el Departamento de Informática de la Universidad de California en Berkeley. POSTGRES fue pionero en muchos conceptos que posteriormente se incorporaron a algunos sistemas de bases de datos comerciales. [8]

La ventaja es que admite gran parte del estándar SQL para realizar operaciones sobre la base de datos. Mediante esta tecnología es posible crear tablas, insertar registros, modificar datos o eliminarlos.

La utilidad que se la ha dado a esta tecnología ha sido la de persistir la información de los usuarios registrados así como la información sus proyectos de predicción de cobertura WiFi en entornos interiores. De esta forma, un usuario podrá disponer de la información que desee en cualquier momento.

2.3 TECNOLOGÍAS DEL MOTOR ANALÍTICO

El motor analítico constituye la parte del sistema encargada de procesar los datos de entrada y generar la predicción correspondiente. A diferencia del frontend y del backend, esta parte no está orientada directamente a la interacción con el usuario ni a la gestión de la aplicación, sino al desarrollo, entrenamiento y ejecución del modelo de aprendizaje profundo

2.3.1 REDES NEURONALES

Antes de explicar el modelo utilizado en este proyecto, es necesario introducir primero el concepto de red neuronal artificial. Una red neuronal es un modelo de aprendizaje automático formado por neuronas organizadas en capas, cuyo objetivo es aprender una relación entre unos datos de entrada y una salida esperada [1].

De forma simplificada, cada neurona recibe una serie de valores, les aplica unos pesos, añade un término de sesgo y utiliza una función de activación para generar su salida. Esta salida se transmite a las siguientes capas de la red, donde la información se va transformando progresivamente hasta obtener la predicción final.

El aprendizaje de la red se produce durante el entrenamiento. En esta fase, el modelo compara la salida que ha generado con el resultado esperado y ajusta sus pesos y sesgos para reducir el error cometido. Por tanto, la red no se programa mediante una regla fija, sino que aprende a partir de ejemplos.

Cuando una red neuronal incorpora varias capas intermedias, se habla de aprendizaje profundo o *Deep Learning*. Estas capas permiten que el modelo aprenda representaciones cada vez más complejas de los datos. En las primeras capas se extraen características más simples, mientras que en las capas posteriores se combinan para obtener información de mayor nivel. [9]

Dentro del aprendizaje profundo, en este proyecto se emplea una red neuronal convolucional (CNN). Estas se distinguen de otras redes neuronales por su rendimiento superior con entradas de señal de imagen [10]. Este enfoque resulta adecuado para el proyecto porque tanto la entrada como la salida del modelo se representan mediante imágenes. La red debe conservar la relación espacial entre los elementos de entrada y el mapa generado, por lo que una arquitectura convolucional permite procesar la información de forma más natural que una red neuronal completamente conectada.

2.3.2 PYTORCH

Figura 8 Logotipo Python, PyTorch

PyTorch es una librería de Python de código abierto utilizada para el desarrollo de modelos de aprendizaje profundo. Permite trabajar con tensores, definir redes neuronales, calcular funciones de pérdida y entrenar modelos mediante algoritmos de optimización [11].

En este proyecto, para poder predecir mapas de cobertura en entornos interiores se ha desarrollado un modelo de *Deep Learning* utilizando esta librería. Me ha permitido definir de una forma simple la arquitectura del modelo, cargar los datos de entrenamiento ejecutar el proceso de aprendizaje para poder generar predicciones.

Uno de los elementos principales de PyTorch es el tensor. Los tensores son matrices n-dimensionales, con la suposición implícita de que pueden ejecutarse en una GPU [12]. Esto permite que se puedan procesar grandes volúmenes de datos simultáneamente, aprovechando la arquitectura paralela de este tipo de procesador gráfico.

En el caso de este proyecto, las imágenes de entrada y salida se transforman en tensores para poder ser procesadas por la red neuronal. PyTorch también proporciona módulos específicos para definir redes neuronales. Estos módulos permiten crear capas convolucionales, normalización, operaciones de reducción de resolución y otros bloques necesarios para construir la U-NET.

2.3.3 HUGGING FACE SPACES

Figura 9 Logotipo Hugging Face

Hugging Face es una plataforma orientada al alojamiento, publicación y despliegue de modelos y aplicaciones relacionadas con aprendizaje automático. Dentro de esta plataforma, Hugging Face Spaces permite desplegar aplicaciones de inteligencia artificial de forma accesible.

En este proyecto, Hugging Face Spaces se utiliza para alojar el servicio de inferencia del modelo de predicción de cobertura WiFi. No se despliega en esta plataforma la aplicación completa, sino únicamente la parte correspondiente al motor de aprendizaje profundo. El servicio desplegado contiene una API desarrollada con FastAPI, junto con el modelo y el código necesario para cargar la arquitectura U-Net y ejecutar la predicción. Esta API expone distintos endpoints, entre ellos GET /health, utilizado para comprobar el estado del servicio, y POST /predict, encargado de recibir los datos de entrada y devolver el mapa de cobertura estimado.

Cuando el usuario solicita una simulación, el *backend* envía al servicio alojado en Hugging Face la imagen del plano del entorno y la máscara con la posición de los puntos de acceso, codificadas como datos de imagen. El servicio de inferencia procesa esta información, ejecuta el modelo U-Net y devuelve como respuesta la imagen del mapa de cobertura predicho.

De esta forma, Hugging Face Spaces permite separar la inferencia del modelo del resto de la aplicación. El *frontend* y el *backend* mantienen sus propias responsabilidades, mientras que el modelo de aprendizaje profundo se ejecuta en un servicio independiente especializado en la predicción.

Capítulo 3. ESTADO DEL ARTE

En este capítulo se analiza la evolución de las técnicas utilizadas para estudiar y predecir la cobertura radioeléctrica. El objetivo es situar el presente Trabajo de Fin de Grado dentro del contexto técnico actual, partiendo de los modelos clásicos de propagación hasta llegar a los métodos basados en aprendizaje profundo.

El problema de estimar la cobertura de una red inalámbrica no es nuevo. Desde los primeros modelos de transmisión radio hasta las herramientas actuales de planificación WiFi, el objetivo ha sido siempre el mismo: conocer cómo se comporta una señal en un entorno determinado. Sin embargo, la forma de abordar este problema ha cambiado mucho con el paso del tiempo, especialmente con la aparición de técnicas de simulación y, más recientemente, de modelos basados en datos.

3.1 EVOLUCIÓN HISTÓRICA DE LA PREDICCIÓN DE COBERTURA RADIOELÉCTRICA

Todo comienza cuando en 1946, Harald T. Friis publicó una de las fórmulas más conocidas para estimar la potencia recibida en un enlace radio en espacio libre en la que tenía en cuenta las pérdidas del espacio libre.

$$\frac{P_r}{P_t} = \left(\frac{\lambda}{4\pi R}\right)^2 * G_t(\theta, \vartheta) * G_r(\theta, \vartheta) \text{ [13]}$$

Mediante esta ecuación se permitió relacionar la potencia transmitida, la distancia entre antenas, la frecuencia y las ganancias de transmisión y recepción. Aunque estas primeras aproximaciones resultaban útiles para comprender enlaces sencillos, estaban pensadas para entornos mucho más controlados que una vivienda u oficina real.

Décadas después, con el crecimiento de las comunicaciones móviles y el auge de los entornos urbanos aparecieron otros modelos capaces de caracterizar la señal en escenarios reales urbanísticos. Uno de los más conocidos fue el modelo de Okumura-Hata, publicado en 1980, que proponía una fórmula empírica para estimar pérdidas de propagación en servicios móviles terrestres. Este modelo motivó la aparición de otros modelos como el desarrollado por la Unión Europea COS 231 [13].

Con este modelo parecía estar todo solucionado, pero en el año 1997 aparece el estándar IEEE 802.11 que, como se mencionó en la introducción, define los diferentes parámetros de las comunicaciones inalámbricas y es sobre este estándar donde nace el término WiFi. En este momento, y motivado por el crecimiento de internet, las redes inalámbricas de área local comenzaron a extenderse progresivamente hasta convertirse en una tecnología habitual en viviendas, oficinas y espacios públicos. Sin embargo, los ingenieros descubrieron rápidamente que los modelos clásicos de propagación en exteriores (como el modelo Okumura-Hata) no servían para interiores. En una ciudad el terreno es más predecible, en cambio, dentro de un edificio una señal puede toparse con una pared de madera o roca, una puerta de metal o un piso de hormigón armado en cuestión de centímetros. Hacía falta un marco matemático unificado para entornos cerrados.

Por esta razón y fruto de esta complejidad se crean recomendaciones específicas y modelos de predicción como la ITU-R P.1238. Nacido en 1997, es el estándar global de matemáticas y física de radiofrecuencia que los ingenieros de telecomunicaciones utilizan para calcular cómo viajan las ondas de radio dentro de un mismo edificio [14].

Figura 10 Logotipo ITU

La ITU (*International Telecommunication Union*) diseñó la norma para que las empresas pudieran calcular cómo se comportaría la señal dentro de un edificio antes de desplegar físicamente toda la infraestructura. Para ello se proporcionaron métodos de predicción, de pérdidas de propagación, pérdidas de transmisión por paredes o incluso el movimiento de personas. Todo ello para hacer un uso adecuado del espectro y evitar así interferencias entre diferentes plantas. Esta recomendación se ha ido variando hasta día de hoy, la última actualización de la recomendación fue la ITU-R P.1238-13 aprobada en 2025.

Sin embargo, la aplicación de estas recomendaciones es demasiado complejo para usuarios con pocos o nulos conocimientos en telecomunicaciones. Por esta razón, otras aproximaciones empezaron a ganar fuerza.

3.2 SIMULACIÓN FÍSICA Y RAY TRACING

Aunque recomendaciones como la ITU-R P.1238 permitieron mejorar la planificación de sistemas inalámbricos en interiores, con el tiempo surgió la necesidad de métodos capaces de representar con mayor detalle el entorno real. A medida que las redes inalámbricas comenzaron a desplegarse en edificios cada vez más complejos, se tenían que empezar a tener en cuenta la geometría del espacio para calcular las posibles trayectorias de la señal, considerando fenómenos como reflexiones y difracciones. Esto impulsó el desarrollo de técnicas de simulación física que intentaban reproducir de forma más fiel el comportamiento de la señal.

Dentro de este tipo de métodos destaca el *ray tracing*. Esta técnica tiene su origen en el ámbito de los gráficos por ordenador, donde se utilizaba para calcular el recorrido de la luz en escenas tridimensionales. Se desarrolló para simular el comportamiento físico de la luz para crear reflejos, refracciones y sombras altamente realistas en gráficos por computadora, orientado en muchas ocasiones para construir videojuegos más realistas. En esta técnica se traza la trayectoria de la luz desde la cámara, a través de cada pixel, para mapear una escena fotorrealista [15].

Aplicado a las comunicaciones inalámbricas, el *ray tracing* representa la señal como un conjunto de rayos que se propagan desde el transmisor y que pueden interactuar con el entorno. A partir de la geometría del escenario, el modelo calcula las trayectorias que puede seguir la señal y tiene en cuenta fenómenos como la reflexión, la difracción, dispersión difusa, la transmisión a través de materiales y la propagación en diferentes direcciones (multicamino). Desde entonces, este enfoque comenzó a utilizarse para la planificación y optimización de redes, especialmente en tecnologías avanzadas como 5G y 6G.

Sin embargo, estos modelos son computacionalmente costosos para entornos geográficos complejos y se desarrollaron técnicas para reducir esa complejidad computacional como indican Chen, Delis y Bertoni al proponer una arquitectura distribuida sobre una red de estaciones de trabajo para acelerar este tipo de predicciones [16].

Por ello, el *ray tracing* es una herramienta muy potente que ha favorecido la aparición de enfoques más prácticos para el análisis de redes inalámbricas. En este contexto aparecen las herramientas actuales de planificación y análisis WiFi, cuyo objetivo principal es facilitar la interpretación de la cobertura mediante representaciones visuales como los mapas de calor.

3.3 HERRAMIENTAS ACTUALES DE PLANIFICACIÓN Y ANÁLISIS WIFI

Este tipo de herramientas se utilizan principalmente en estudios de sitio, también conocidos como *site surveys*. En estos estudios, el usuario parte normalmente de un plano del entorno y recorre físicamente el espacio tomando medidas de señal en distintos puntos. A partir de esas medidas, la herramienta genera un mapa de cobertura que permite detectar zonas con señal débil, interferencias o una ubicación poco adecuada de los puntos de acceso. La generación de estos mapas se suele hacer siguiendo las recomendaciones mencionadas anteriormente o la técnica del *ray tracing*, intentando descifrar el modelo físico que determina la señal en aquellas zonas que el usuario no ha medido.

Soluciones como NetSpot o Acrylic WiFi Heatmaps siguen este enfoque y permiten realizar análisis visuales de redes WiFi mediante mapas de calor y estudios de cobertura. Estas herramientas destacan por su facilidad para representar gráficamente la intensidad de señal y ayudar a identificar zonas problemáticas dentro de un edificio. Sin embargo, la calidad de los resultados depende en gran medida de la cantidad y distribución de las mediciones realizadas. Si el usuario no recorre suficientes puntos o no toma las muestras de forma adecuada, el mapa generado puede no reflejar con precisión el comportamiento real de la red.

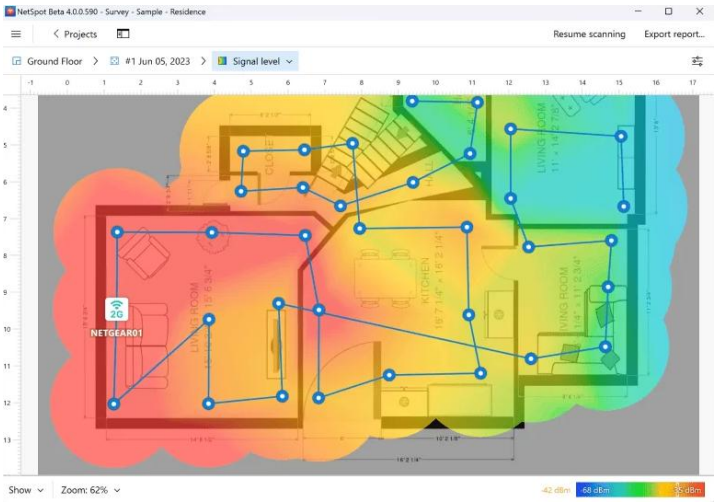

Figura 11 NetSpot Beta Simulation

En entornos más profesionales existen herramientas como Ekahau, que incorporan funciones avanzadas de planificación, validación y generación de informes técnicos. Estas soluciones permiten realizar análisis mucho más completos, incluyendo estudios predictivos y simulaciones de cobertura. Gracias a ello son ampliamente utilizadas por empresas especializadas en el diseño y despliegue de redes inalámbricas.

No obstante, incluso estas herramientas presentan ciertas limitaciones. Por un lado, suelen requerir información detallada del entorno, como planos precisos, características de los

materiales de construcción o parámetros específicos de los dispositivos de red. Por otro, muchas de sus funcionalidades están pensadas para usuarios con conocimientos técnicos avanzados, capaces de interpretar correctamente los resultados obtenidos y tomar decisiones a partir de ellos. Además, algunas de estas soluciones implican costes de licencia elevados, lo que dificulta su adopción en contextos domésticos o por parte de usuarios ocasionales. Todas estas características limitan su uso a un público concreto de ingenieros, y a personas que pueden permitirse el coste de contratar a uno y pagar la licencia de dichas herramientas.

Suponiendo entonces que las aplicaciones mencionadas logran estimar de manera correcta el mapa de señal resultante en una vivienda, nos topamos con la desventaja de que su utilización suele requerir tiempo, experiencia y conocimientos técnicos.

3.4 ESTIMACIÓN DE MAPAS DE RADIO MEDIANTE APRENDIZAJE AUTOMÁTICO

Frente a los enfoques anteriores basados en formulaciones físicas, en los últimos años ha cobrado fuerza la estimación de mapas de radio o Radio Map Estimation (RME). Este planteamiento consiste en obtener el mapa de cobertura de un entorno a partir de datos, ya sean medidas parciales tomadas en algunos puntos, resultados de simulaciones o ejemplos previos, en lugar de resolver de forma explícita las ecuaciones de propagación en cada posición.

En su forma más sencilla, la estimación de mapas de radio puede abordarse mediante técnicas de interpolación espacial, que reconstruyen el valor de la señal en zonas no medidas a partir de las muestras disponibles. Sin embargo, estos métodos no capturan adecuadamente la influencia de la estructura del entorno, ya que tratan el espacio de forma continua y no tienen en cuenta de manera natural elementos como las paredes o los obstáculos que condicionan la propagación.

Para superar esta limitación ha ganado protagonismo el uso de técnicas de aprendizaje profundo (*Deep Learning*) y, en particular, de las redes neuronales convolucionales (CNN). Estas redes son capaces de aprender, a partir de un conjunto suficientemente amplio de ejemplos, la relación entre la estructura de un entorno y la forma en que se distribuye la señal en él. De este modo, la predicción de cobertura puede plantearse como un problema de generación de imágenes, a partir de una imagen que representa el entorno se obtiene otra imagen que representa el mapa de cobertura.

Entre las arquitecturas más utilizadas para este tipo de tareas destaca la U-Net, una red convolucional con estructura de codificador-decodificador y conexiones directas (*skip connections*) entre las etapas de bajada y de subida. Fue concebida originalmente para la segmentación de imágenes biomédicas [17], pero su capacidad para transformar una imagen de entrada en otra imagen espacialmente alineada la ha convertido en una opción muy adecuada para la estimación de cobertura.

En este caso, la red recibe una representación del plano del entorno junto con la posición de los puntos de acceso y genera como salida el mapa de potencia recibida.

Un estudio especialmente relevante para este proyecto y basado en esta idea es el de Pyo, Sawada y Matsumura, que entrenan un modelo de aprendizaje profundo con mapas de radio generados mediante *ray tracing* en la banda de 2,4 GHz sobre entornos interiores imitados. Para ello proponen una arquitectura propia, denominada RadioResUNet (*Radio Residual U-Net*), que parte de la U-Net e incorpora conexiones residuales para facilitar el entrenamiento de la red [18].

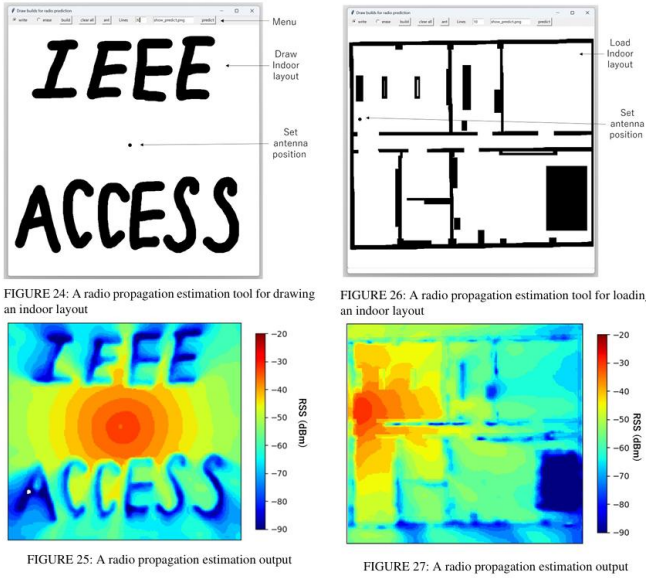

FIGURE 24: A radio propagation estimation tool for drawing an indoor layout

FIGURE 26: A radio propagation estimation tool for loadin an indoor layout

FIGURE 25: A radio propagation estimation output

FIGURE 27: A radio propagation estimation output

Figura 12 Resultado de estudio RME con RadioResUNet basado en Ray Tracing

De este modo, el modelo aprende a reproducir el detalle físico de la simulación, pero ofreciendo predicciones casi instantáneas durante la inferencia, que validan frente a medidas reales. Este estudio confirma que el aprendizaje profundo permite aprovechar la precisión de los métodos deterministas sin asumir su elevado coste en cada predicción.

Este es precisamente el enfoque que adopta el presente Trabajo de Fin de Grado. Para entrenar el modelo se parte de un conjunto de datos público de mapas de cobertura simulados en la banda de 5 GHz para entornos interiores, generados a partir de planos de oficinas y aulas con distintas configuraciones de puntos de acceso. A partir de estos datos se entrena una U-Net que recibe como entrada el plano del entorno y la ubicación de los puntos de acceso, y devuelve el mapa de cobertura estimado

3.5 MEDICIÓN DE SEÑAL

Además de las herramientas de planificación WiFi, también existen pruebas de red más sencillas orientadas al usuario general, como los test de velocidad disponibles en Internet. Este tipo de pruebas, accesibles desde buscadores o plataformas como Google, permiten estimar parámetros como la velocidad de descarga, la velocidad de subida o la latencia de la conexión. Sin embargo, estos resultados no siempre explican por qué una red WiFi funciona peor en una zona concreta de la vivienda, ya que la velocidad final depende de muchos factores y no únicamente de la cobertura inalámbrica.

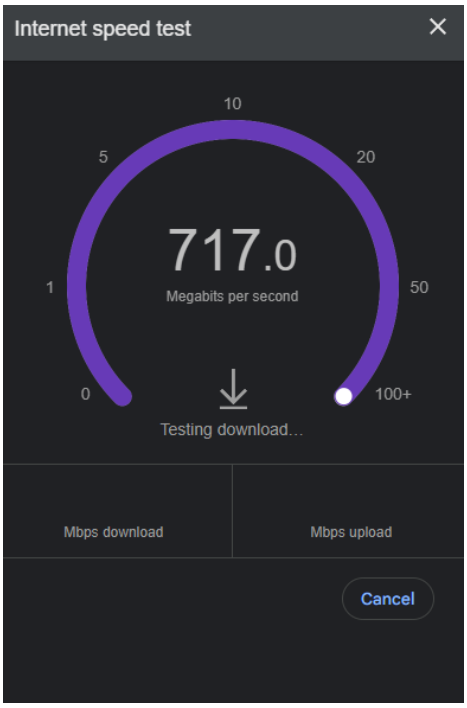

Figura 13 Test de velocidad de Internet Google

Por este motivo, en la aplicación desarrollada se incorpora un *test* de red con un enfoque distinto. En lugar de medir únicamente la velocidad de Internet, el *test* utilizaría comandos del propio sistema operativo para obtener información relacionada con la conexión WiFi local, como la intensidad de señal, el punto de acceso al que está conectado el equipo o el canal utilizado. Esta información no sustituye a una medición profesional, pero puede ayudar al usuario a interpretar mejor la calidad de la señal y a tener en cuenta posibles problemas derivados de interferencias, saturación del canal o una ubicación poco adecuada del router.

Capítulo 4. DEFINICIÓN DEL TRABAJO

Una vez revisadas las distintas formas de abordar la predicción de cobertura y las soluciones existentes, en este capítulo se concreta el trabajo desarrollado. Se presentan la justificación que motiva el proyecto, los objetivos que persigue, su alcance, la metodología seguida durante el desarrollo y la planificación temporal con la que se ha organizado.

El proyecto se centra en el desarrollo de una aplicación destinada al análisis y la optimización predictiva de la cobertura Wi-Fi en entornos interiores. Para ello se combina una aplicación de escritorio basada en tecnologías web, una API de backend para la gestión de usuarios y proyectos, y un módulo analítico basado en aprendizaje automático para estimar mapas de cobertura a partir de planos interiores y posiciones de puntos de acceso

4.1 JUSTIFICACIÓN

La conectividad inalámbrica se ha convertido en un requisito básico en viviendas, oficinas, centros educativos y espacios de trabajo compartido. En muchos de estos entornos, la calidad percibida por el usuario no depende únicamente de la velocidad contratada o de la capacidad nominal del router, sino de la distribución real de la señal dentro del espacio. La presencia de paredes, pasillos, zonas alejadas del punto de acceso, mobiliario y otros obstáculos puede generar pérdidas de señal, zonas de baja cobertura y variaciones significativas en la experiencia de uso

Como se ha visto en el capítulo anterior, las soluciones disponibles para conocer la cobertura WiFi de un entorno presentan limitaciones desde el punto de vista de un usuario no especializado. Los métodos de simulación física, como el ray tracing, ofrecen una gran precisión, pero requieren una descripción detallada del entorno y un coste computacional elevado. Por otra parte las aplicaciones comerciales existentes, se apoyan en medidas reales, suelen requerir licencias caras y exigen cierta interpretación técnica de los resultados.

Existe, por tanto, un espacio para una herramienta más accesible, capaz de ofrecer una estimación inicial de la cobertura antes incluso de realizar ninguna medida y partiendo únicamente de un plano sencillo del entorno. Para muchos usuarios domésticos, una representación aproximada pero inmediata de cómo se distribuirá la señal en su vivienda resulta más útil en la práctica que un estudio exhaustivo que requiere medios profesionales e incluso conocimientos de arquitectura.

Disponer de esta información ayuda al usuario a tomar decisiones cotidianas, como elegir la ubicación de un repetidor o de un punto de acceso, o identificar de antemano las zonas donde la conexión será más débil. Además, una colocación más informada de los dispositivos contribuye a un uso más responsable del espectro, ya que evita aumentar innecesariamente la potencia de transmisión o instalar más equipos de los necesarios, lo que a su vez reduce las interferencias con otras redes cercanas.

Por tanto, el propósito de este proyecto es construir una aplicación que, apoyándose en las tecnologías web descritas en el Capítulo 2, dé un uso práctico y accesible al modelo de *Deep Learning* desarrollado. La aplicación se concibe, además, con una arquitectura basada en microservicios, que facilita su despliegue en la nube (*cloud*) y permite desarrollar y mantener cada componente de forma independiente.

La viabilidad de este enfoque se ve reforzada por dos factores recientes. Por un lado, la disponibilidad de conjuntos de datos de mapas de cobertura simulados siguiendo el estándar IEEE 802.11ax channel model, que permiten entrenar modelos sin necesidad de realizar un *dataset* de medidas propias. Por otro, el auge y el contexto del aprendizaje automático de las técnicas de aprendizaje profundo aplicadas al tratamiento de imágenes, hacen factible abordar la estimación de cobertura como un problema de generación de imágenes con un coste de cálculo asumible.

Por otro lado, el uso de un Test Wifi en tiempo real permite al usuario saber la calidad de su señal WiFi en tiempo real independiente de la velocidad externa contratada. El modelo predice la cobertura sobre el plano, y el test la mide en el sitio donde estás. Además, también se incorpora el test de velocidad tradicional.

4.2 OBJETIVO

El objetivo general del Trabajo de Fin de Grado es desarrollar una aplicación capaz de estimar mapas de cobertura WiFi en entornos interiores a partir del plano del entorno y de la posición de uno o varios puntos de acceso, utilizando un modelo de aprendizaje profundo integrado dentro de una arquitectura software completa y desplegada en un entorno *cloud* para que sea accesible a todos los usuarios.

A partir del objetivo general se definen los siguientes objetivos específicos:

- Desarrollar el sistema software que da soporte a la herramienta, formado por una aplicación de escritorio que permita a cualquier usuario, con independencia de sus conocimientos técnicos, describir su entorno de forma sencilla e intuitiva dibujando el plano y situando los puntos de acceso, y por un backend que centralice la seguridad, la lógica de negocio y la coordinación del sistema, garantizando la gestión segura y persistente de los usuarios y de sus proyectos sobre una base de datos relacional.

- Diseñar la extracción de datos y entrenar el modelo de estimación de mapas de radio basado en una arquitectura U-Net, que constituye el núcleo del sistema, evaluando además la calidad de las predicciones que genera.

- Desplegar el sistema completo y empaquetar la aplicación para su distribución

4.3 METODOLOGÍA

Para el desarrollo del proyecto se ha seguido una metodología ágil, habitual en proyectos de desarrollo de software. Este enfoque permite organizar el trabajo en ciclos breves e iterativos, en los que se van definiendo, implementando y revisando distintas funcionalidades de la aplicación.

La principal ventaja de esta metodología es que permite avanzar de forma progresiva, obteniendo versiones parciales pero funcionales del sistema. Esto es porque en cada iteración, toda la parte del sistema que hay que modificar para conseguir una funcionalidad, crece es paralelo. De esta manera, el proyecto no se desarrolla como un bloque único al final del proceso, sino que evoluciona mediante mejoras sucesivas hasta alcanzar una versión completa y validable.

Para favorecer esta forma de trabajo se ha adoptado como decisión de diseño una arquitectura de microservicios, que separa la aplicación en componentes independientes y permite desarrollarlos y probarlos por separado.

4.4 PLANIFICACIÓN Y ESTIMACIÓN ECONÓMICA

El desarrollo se ha organizado en varias fases, con cierto solapamiento entre ellas propio de un trabajo con metodología ágil. Se muestra de forma visual mediante el diagrama de Gantt asociado al proyecto.

	Diciembre	Enero	Febrero	Marzo	Abril	Mayo	Junio
Análisis del Sistema y Búsqueda de Datos							
Preparación Y Configuración del entorno							
Desarrollo del *Backend*							
Desarrollo del *Frontend*							
Motor Analítico							
Integración de módulos							
Correcciones							
Documentación							

Figura 14 Diagrama de Gantt de la planificación del proyecto

Conviene señalar que los tiempos asignados a cada tarea tienen son orientativos. Durante el desarrollo del proyecto, algunas actividades comenzaron antes de lo previsto, mientras que otras se prolongaron más allá de la fecha inicialmente estimada. O por otro lado, la

necesidad de ajustar en pequeña medida los diferentes módulos de la aplicación. Lo que se quiere informar en este diagrama son las tareas más importantes que se llevaron a cabo.

Por lo que, el diagrama de Gantt permite representar de forma general las fases seguidas durante la realización del trabajo y mostrar la importancia que se le dio a cada módulo de manera orientativa.

Para la estimación económica solo se tiene en cuenta el precio de los principales componentes de hardware del ordenador de mesa utilizado para desarrollar el proyecto. Este ordenador se ha utilizado sobre todo para el desarrollo y entrenamiento de la U-Net utilizada, y todas las tareas desarrolladas en mayo y junio. Sin embargo, para el desarrollo del resto de la aplicación se ha utilizado el portátil MSI GF63 con procesador Intel Core i7, tarjeta gráfica dedicada NVIDIA GTX 1660 Ti y 16 GB de RAM. Sin embargo, como todo el proyecto se puede desarrollar con el ordenador más potente pero no con el portátil, solo contará el que mejores prestaciones tiene para la estimación económica.

Por otra parte, todo el software es gratuito e incluso *open-source.* Tanto las herramientas utilizadas para el desarrollo de la aplicación como para el despliegue en la nube se han utilizado tecnologías gratuitas o en su defecto, pruebas gratuitas con menos prestaciones como es el caso del despliegue en Render o Hugging Face.

Componente	Características principales	Coste (€)
Procesador	AMD Ryzen 7 3800X, 8 núcleos y 16 hilos, 3,9 GHz	296,99
Memoria RAM	32 GB RAM	275,99
Tarjeta gráfica	NVIDIA GeForce RTX 3060 Ti, 8 GB de memoria dedicada	628,99
Placa base	MSI MS-7C37, plataforma AMD AM4	214,9
Sistema operativo	Windows 11 Home 64 bits	98,99
Ventilador Externo	Mars Gaming MNBC2	15.00
Total Estimado		**1530,86**

Tabla 1 Estimación Económica

Para poder utilizar la aplicación, el usuario únicamente necesita disponer de un equipo compatible con Windows y conexión a Internet. Al distribuirse la aplicación como un archivo ejecutable .exe, no es necesario que el usuario instale manualmente las tecnologías utilizadas durante el desarrollo, como React, Java, Python, PyTorch o PostgreSQL. Estas tecnologías forman parte del entorno de desarrollo o se ejecutan en

servicios externos. Sin embargo, si un usuario deseara replicar el proyecto, los componentes de la tabla serían los requisitos mínimos para hacerlo.

Aunque el proyecto se ha desarrollado utilizando tecnologías gratuitas y servicios en sus capas gratuitas, una explotación real de la aplicación tendría costes asociados. A medida que aumentase el número de usuarios, sería necesario asumir gastos de infraestructura, como servidores para ejecutar la inferencia del modelo, base de datos, almacenamiento y mantenimiento del sistema.

Concepto (coste mensual)	**Pesimista**	**Realista**	**Optimista**
Plan de workspace (Render)	Gratis	23 €	23 €
Backend – servicio web	6 €	23 €	155 €
Inferencia (Hugging Face)	Gratis	45 €	340 €
Base de datos PostgreSQL	6 €	20 €	80 €
Almacenamiento	Incluido	5 €	18 €
Dominio y certificados	1 €	2 €	4 €
Mantenimiento y operación (estimado)	—	55 €	200 €
Total mensual	14 €	173 €	820 €
Total anual	170 €	2.080 €	9.840 €

Tabla 2 Presupuestos de operación

Por ello, resulta útil plantear posibles formas de sostenibilidad económica.

Una primera opción sería ofrecer la aplicación con un modelo gratuito y funciones de pago. La versión básica podría permitir crear algunos proyectos y simular configuraciones sencillas, mientras que una versión avanzada incluiría más puntos de acceso, proyectos ilimitados o informes en PDF . Este enfoque encaja bien con usuarios domésticos, ya que permite probar la herramienta antes de pagar por funciones adicionales.

Otra posibilidad sería orientar la aplicación a un uso profesional, por ejemplo, para pequeñas empresas o gestores de edificios. En este caso, el pago podría realizarse mediante una suscripción mensual o anual, ofreciendo funciones más completas como gestión de varios planos o acceso a resultados más detallados. Esta opción requeriría que la herramienta ofreciera una precisión suficiente para justificar su uso en entornos profesionales.

Por último, también podría plantearse licenciar el motor de predicción a terceros. En este caso, la aplicación no se vendería directamente al usuario final, sino que el servicio de inferencia podría integrarse en plataformas de fabricantes de *routers*, operadores de telecomunicaciones o empresas del sector inmobiliario. Esta alternativa tendría un mayor potencial económico, aunque también exigiría una validación más rigurosa del modelo con medidas reales y una integración técnica más robusta.

Capítulo 5. SISTEMA/MODELO DESARROLLADO

En este capítulo se explica en detalle el sistema desarrollado en este proyecto y cómo se ha construido.

En primer lugar, se presenta una visión general de la arquitectura de microservicios. Después, se describe la aplicación de escritorio con la que trabaja el usuario y el *backend* que lo coordina todo. A continuación, se explica el conjunto de datos con el que se ha entrenado el modelo, las bases del aprendizaje profundo y la arquitectura de la red U-Net, junto con su entrenamiento. Y, por último, se describe el servicio que ejecuta el modelo y cómo se ha desplegado todo el sistema.

5.1 ARQUITECTURA DE MICROSERVICIOS

El motivo por el que se ha escogido esta arquitectura es la necesidad de un despliegue independiente. Cada módulo de la aplicación se ha desarrollado utilizando tecnologías completamente distintas. El modelo vive en el ecosistema de Python y PyTorch, mientras que la lógica de negocio encaja mejor en Java y la interfaz en JavaScript (React). Este requisito motivó una arquitectura de microservicios, que permite que cada parte emplee su tecnología idónea y se despliegue por separado. De esta forma, cada módulo tiene una responsabilidad delimitada con su propio ciclo de vida.

Estos servicios se comunican mediante APIs REST sobre HTTP, intercambiando los datos en formato JSON.

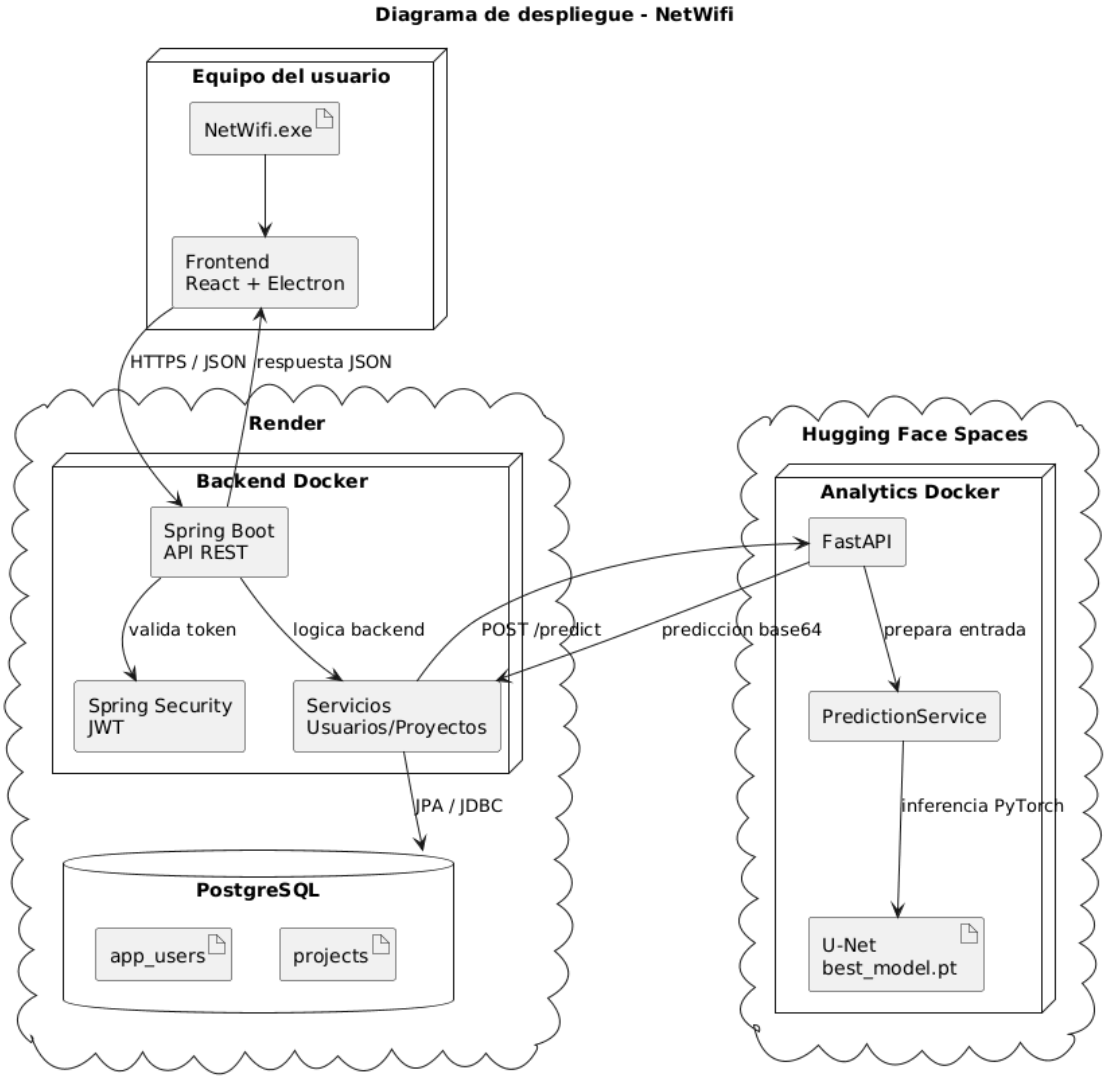

Figura 15 Diagrama de Despliegue

El sistema se compone de tres servicios y un conjunto de datos:

- Aplicación de escritorio (*frontend*). Interfaz en React, Vite y Electron. En ella se produce la interacción con el usuario, registro/sesión, test de wifi, carga del plano, colocación de puntos de acceso y visualización del resultado. Se presenta al usuario como un ejecutable .exe

- Backend (Spring Boot). API REST en Java que actúa como orquestador: gestiona la autenticación, la persistencia de los proyectos y la coordinación con el motor analítico. Este se empaqueta con un Dockerfile y se despliega en Render.

- Motor analítico (FastAPI + PyTorch). Servicio en Python especializado exclusivamente en la inferencia: carga la red U-Net entrenada y, dada una entrada,

devuelve el mapa de cobertura. Se empaqueta con un Dockerfile y se sube junto con el mejor modelo a Huggin Face.

- Base de datos (PostgreSQL). Capa de persistencia de usuarios y proyectos. También se aloja en Render utilizando la versión 16.

Mediante este diseño, también se consigue un grado más de seguridad en la aplicación. Como se puede observar en la figura 15, tanto el *frontend* como el modelo de *Deep learning* necesitan hacer la petición o enviar datos al *backend* antes de comunicarse con la base de datos. De esta forma, aparte de los primeros filtros de seguridad que existen por ejemplo en el *frontend*, todo se necesita verificar en el *backend* en el cual se concentra la lógica de seguridad, de negocio y de orquestación de la aplicación.

Se puede observar que, de esta forma, el motor analítico queda completamente aislado, solo se comunica con el *backend*, módulo por el que tiene que pasar el usuario antes de comunicarse con el servicio de inferencia. Por lo tanto, queda aislado y solo se dedica a predecir.

5.2 APLICACIÓN DE ESCRITORIO (FRONTEND)

El *frontend* se ha desarrollado con React y se empaqueta como aplicación de escritorio mediante Electron. Esta decisión permite distribuir la aplicación como un archivo ejecutable .exe para Windows, de forma que el usuario pueda utilizarla como una aplicación local sin necesidad de abrir un navegador ni instalar manualmente el entorno de desarrollo. Como se mencionó en el apartado 3.5, la aplicación implementa un *test* para obtener toda la información de la señal WiFi a la que esta conectada el usuario. Esta funcionalidad se consigue ejecutando comandos del sistema para leer el estado de la interfaz WiFi de Windows. Esto, está bloqueado en la mayoría de los navegadores por lo que se optó por desarrollar la aplicación en formato de escritorio en lugar de web.

Puesto que el objetivo de este módulo es concentrar en él toda la interacción con el usuario, es conveniente describir todas las funcionalidades que tiene el sistema mediante diagramas UML de casos de uso.

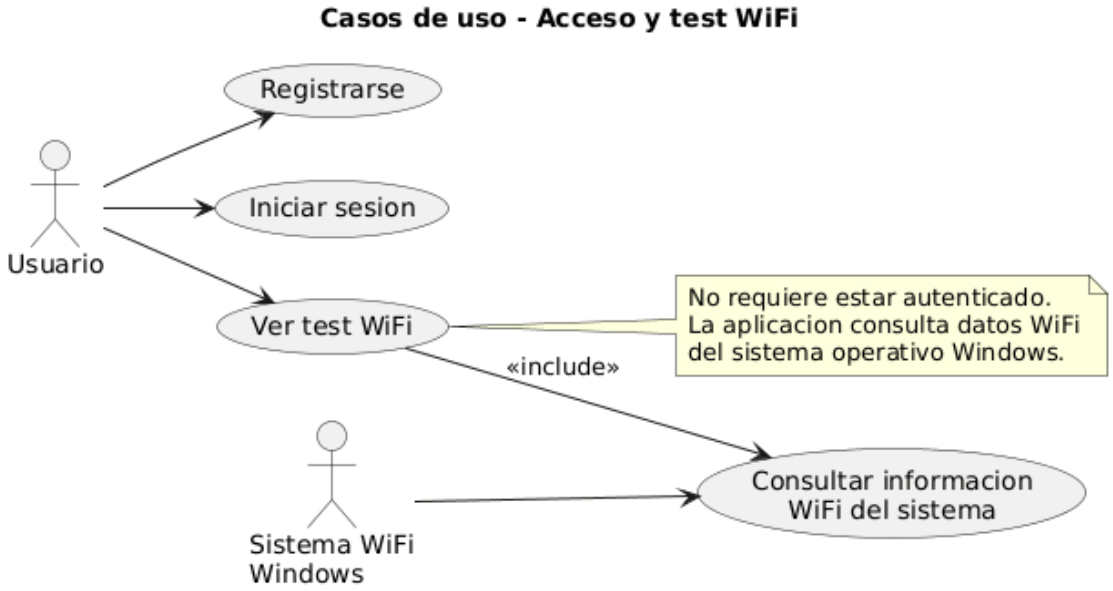

Figura 16 Caso de Uso Usuario No Autenticado

Figura 17 Caso de Uso Usuario Autenticado

Como se observa en los diagramas anteriores, las funcionalidades se dividen en dos grupos según el estado del usuario. Las acciones de registro e inicio junto con la consulta de la red WiFi, están disponibles sin necesidad de autenticación, mientras que cerrar sesión, borrar cuenta, la gestión de proyectos y la simulación solo son accesibles una vez que el usuario ha iniciado sesión.

5.2.1 SERVICIO DE USUARIOS

Para gestionar el registro y la autenticación de los usuarios se ha utilizado el estándar JSON Web Token (JWT) [19].Es un método muy habitual en las APIs REST y en las arquitecturas de microservicios, porque permite que el *backend* sea *stateless*, es decir, que no guarde en memoria información sobre la sesión de cada usuario. Esto encaja bien con un sistema en el que el *frontend* está separado del *backend* y este responde únicamente con datos en formato JSON, y no con páginas ya construidas.

Este token es compuesto por tres partes: el primero corresponde al encabezado o *header* el cual contiene metadatos sobre el tipo de token empleado y el algoritmo asociado a la firma o cifrado. Posterior al encabezado, encontramos el *payload*, esta sección contiene la información que se desea trasmitir al utilizar el token, en nuestro caso el usuario. Luego, se encuentra la firma, la cual valida el origen del token y permite verificar si ha sido modificado; los JWT son tokens autónomos, contando en sí con toda la información necesaria para verificar su validez [20].

En este último componente es donde reside la seguridad, la firma esta generada combinando la cabecera y el *payload* con una clave secreta y calcula un hash.

Como la función de hash es unidireccional y muy sensible a cualquier cambio, basta con modificar un solo carácter del token para que la firma deje de ser válida. Y, aunque el algoritmo de firma es conocido, nadie puede generar una firma correcta sin la clave secreta, que solo conoce el *backend*.

Cuando el usuario inicia sesión correctamente, el *backend* genera un token y lo devuelve al *frontend*, que lo almacena en el equipo. A partir de ese momento, cada petición que requiere autenticación incluye el token en la cabecera de autorización, de modo que el *backend* puede comprobar en cada llamada que el usuario está autenticado antes de responder. El *backend* confirma que el *token* es válido y sustrae la información del usuario para verificarlo en base de datos y poder llevar a cabo la acción necesaria.

En esta aplicación, el *token* no se guarda en base de datos, si no que, incorpora una marca de expiración entre sus datos, y la aplicación controla su validez mediante un temporizador. De esta forma, cuando el *token* caduca, deja de aceptarse y el usuario debe iniciar sesión de nuevo para obtener uno nuevo.

5.2.2 ARQUITECTURA DEL *FRONTEND*

La estructura se ha organizado siguiendo una separación por responsabilidades. Con ello, evitas que toda la lógica se concentre en un único archivo y facilita la creación del proyecto con una metodología ágil.

```
FrontEnd/
   electron/          → app de escritorio
     src/
     app/           → estado global, rutas y navegación
     pages/         → las pantallas (Home, Login, Registro, Editor, Test)
     features/      → lógica por funcionalidad: auth, forms, projects, wifi
                      (cada una con sus llamadas a la API y sus hooks)
     components/    → componentes de estructura reutilizables
     shared/        → lo común (la URL del backend centralizada)
```

La carpeta `electron/` contiene la configuración necesaria para ejecutar la aplicación como programa de escritorio. Esta parte se encarga de crear la ventana principal, cargar la interfaz generada por React y permitir determinadas interacciones con el sistema operativo. Gracias a esta capa, la aplicación puede distribuirse posteriormente como un ejecutable para Windows.

El componente principal, App.jsx dentro de `app/`, guarda el estado general (la pantalla activa, el usuario, el token en localStorage , el proyecto elegido) y controla el paso entre pantallas: la página de inicio con la lista de proyectos, el inicio de sesión y el registro, el editor de proyectos y el test de red. De esta forma, solo existe una página que cambia lo que muestra sin recargar toda la página. Las demás páginas llaman a sus funciones cuando sea necesario y esta cambia la ruta que se muestra.

La carpeta `pages/` agrupa las pantallas principales de la aplicación. Entre ellas se encuentran la pantalla de inicio, el formulario de inicio de sesión, el formulario de registro, la vista de proyectos, el editor del entorno y la pantalla correspondiente al test de red. Esta separación permite que cada pantalla tenga una responsabilidad clara y evita mezclar la lógica de distintas partes de la interfaz.

La carpeta `features/` contiene las funcionalidades de la aplicación (autenticación, proyectos y test de red) junto con sus llamadas al *backend* a través de las APIs. También es donde se definen los *hooks*. Esto es una funcionalidad de React que da estado y comportamiento a un componente. El más básico es `useState`, que crea una variable cuyo cambio hace que React vuelva a dibujar automáticamente la parte de la interfaz que depende de ella. El otro utilizado es `useEffect`, sirve para acciones externas al dibujo como temporizadores o llamadas al *backend*.

Por último, la carpeta `components/` contiene componentes reutilizables de la interfaz, como barras de navegación, botones, formularios, tarjetas, diálogos o elementos visuales compartidos. Estos componentes permiten mantener una apariencia coherente en toda la aplicación y reducen la duplicación de código.

5.2.3 REGISTRO/INICIO DE SESIÓN

A continuación, se muestra cual sería el flujo de comunicación para el caso de uso de Registrarse mediante el diagrama de secuencia:

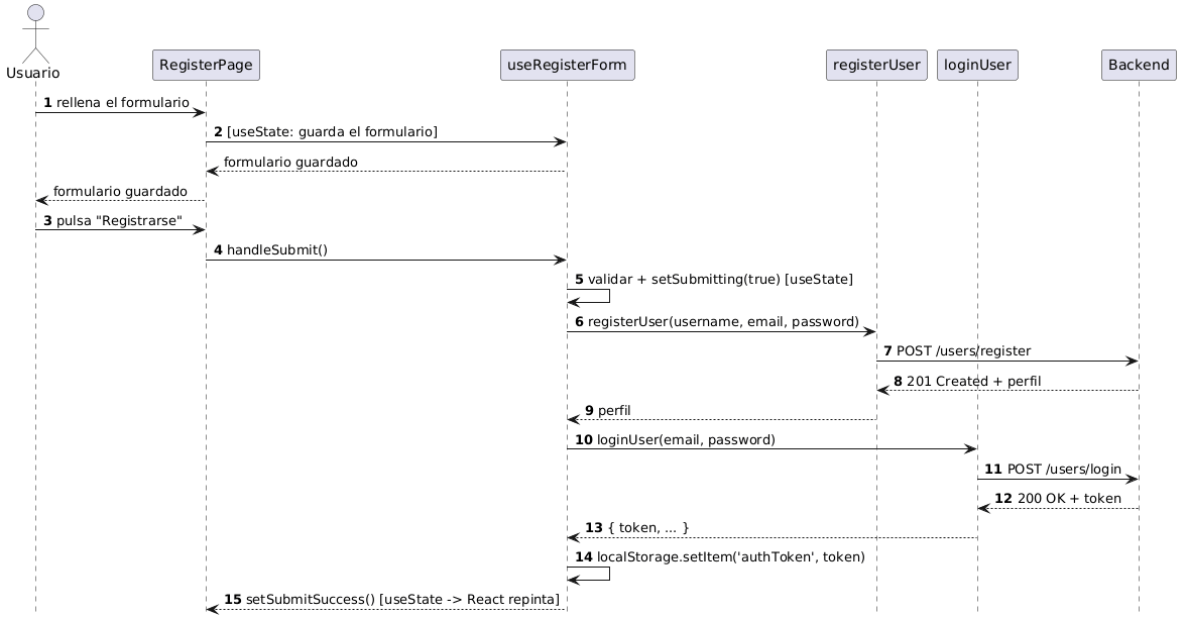

Figura 18 Diagrama de Secuencia Caso de Uso: Registrarse

El usuario rellena el formulario y, mientras escribe, el *hook* `useRegisterForm` va guardando los datos con `useState` (por eso la pantalla se actualiza sola). Al pulsar "*Create Account*", el *hook* valida los campos y hace dos llamadas seguidas al *backend*: primero `registerUser (POST /users/register)`, que crea el usuario con la contraseña cifrada, y justo después `loginUser (POST /users/login)`, que devuelve el token. El *hook* guarda ese token en `localStorage` y actualiza el estado para mostrar el mensaje de éxito y pasar a la pantalla de inicio.

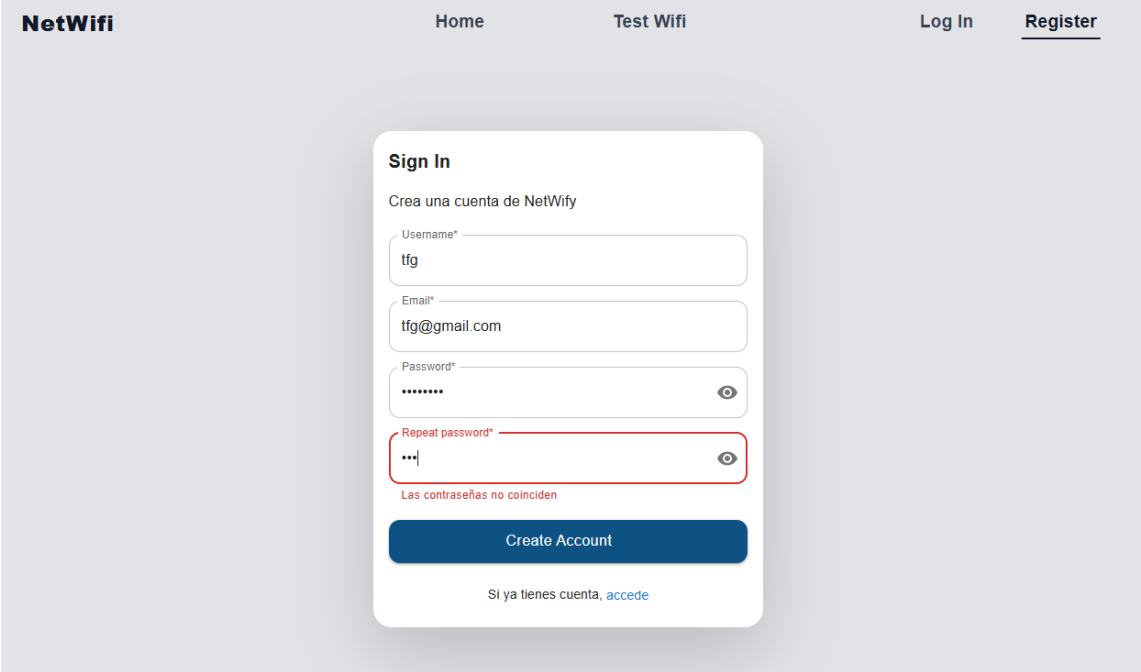

Figura 19 Página de Registro

5.2.4 PÁGINA PRINCIPAL

Cuando el usuario se registra o inicia sesión de forma exitosa, `useEffect` detecta que hay un usuario y lanza una llamada al *backend* para que le devuelva los proyectos que dicho usuario tiene guardados.

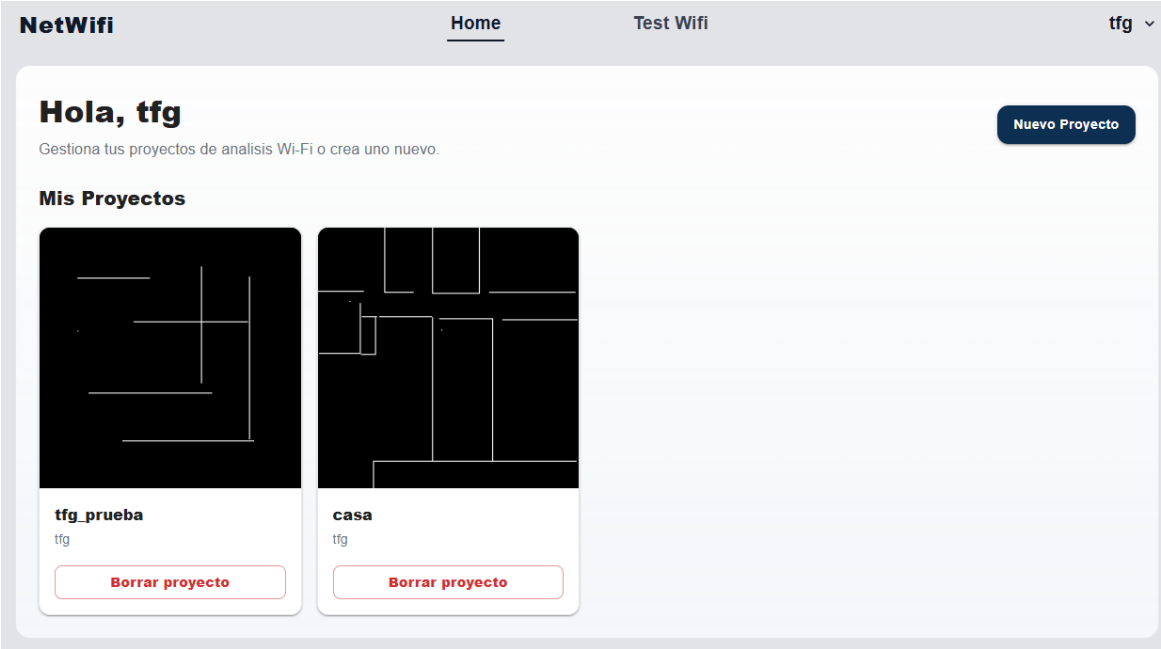

Figura 20 Página Principal

Por otra parte, se muestra también cual es el flujo de comunicación para la otra acción mas representativa de la aplicación, simular un proyecto guardado.

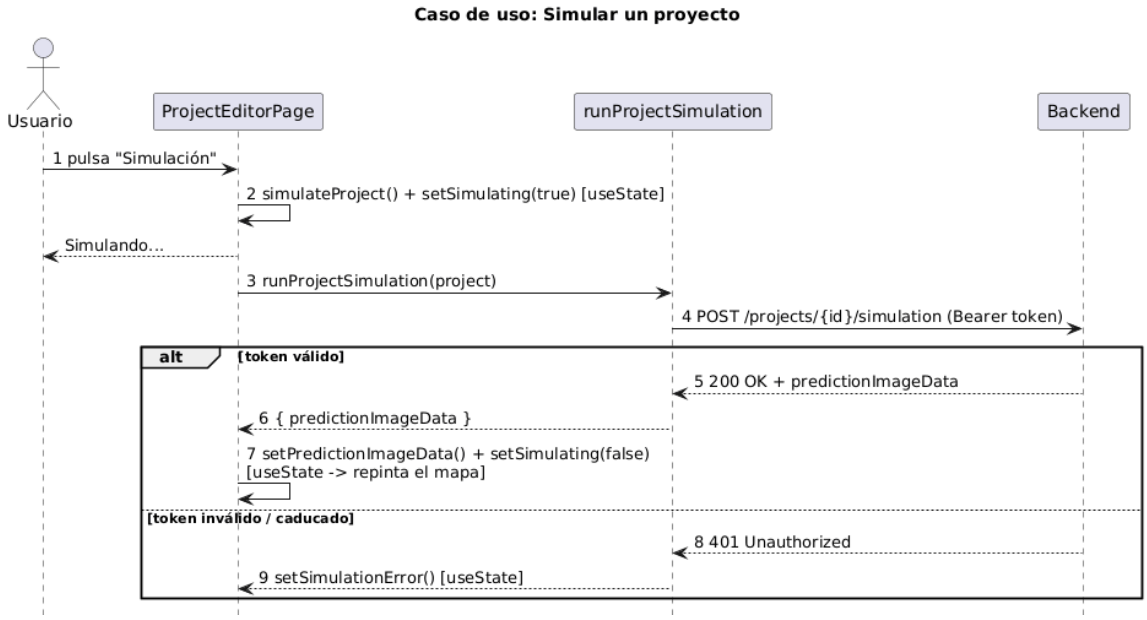

Figura 21 Diagrama de Secuencia Caso de Uso: Simular un proyecto

Al pulsar el botón de simulación, la pantalla ProjectEditorPage marca el estado *simulating* con `useState` (para deshabilitar el botón y mostrar "Simulando...") y llama a runProjectSimulation, que envía un `POST /projects/{id}/simulation` con el token en la cabecera. Si el token es válido, el backend devuelve el mapa de cobertura (predictionImageData), la pantalla lo guarda en su estado y React lo repinta sobre el plano

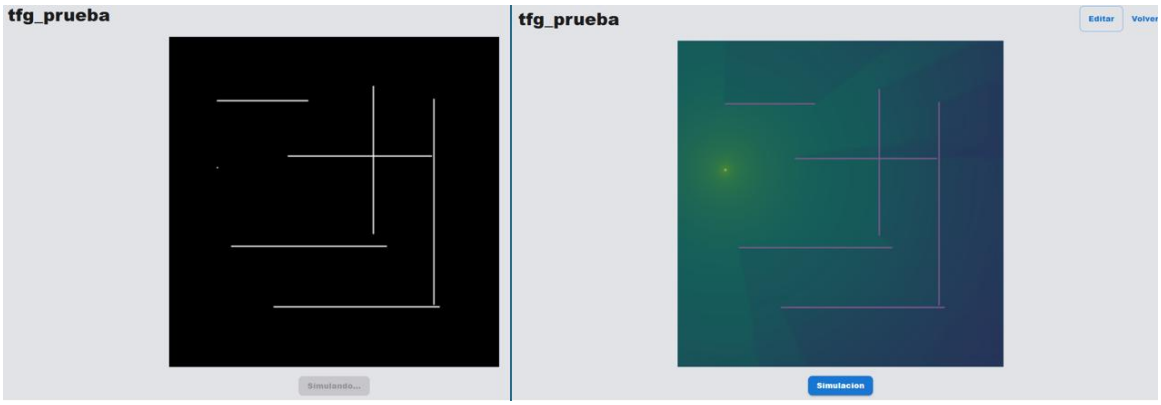

Figura 22 Página de simulación

5.2.5 EDITOR DE MAPAS

La interfaz del editor está construida con componentes de Material UI (MUI) (como se mencionó en el capítulo 2), y la zona de dibujo es un elemento SVG con un sistema de coordenadas fijo de 256 × 256. Este tamaño coincide con el que espera la U-Net por lo que trabajar desde el principio con el mismo tamaño que usa la red evita cambiar de escala, y por tanto nunca se descuadran el plano ni el mapa predicho (consultar apartado 5.5.2).

Para que los planos se parezcan a los del conjunto de entrenamiento, las paredes se obligan a ser rectas, horizontales o verticales: mientras se arrastra el ratón, se compara la diferencia entre x e y dejando solo la dirección que más cambia. Cada *router* es un punto blanco, y el número máximo es tres, igual que en el conjunto de datos que se ha utilizado para entrenar el modelo.

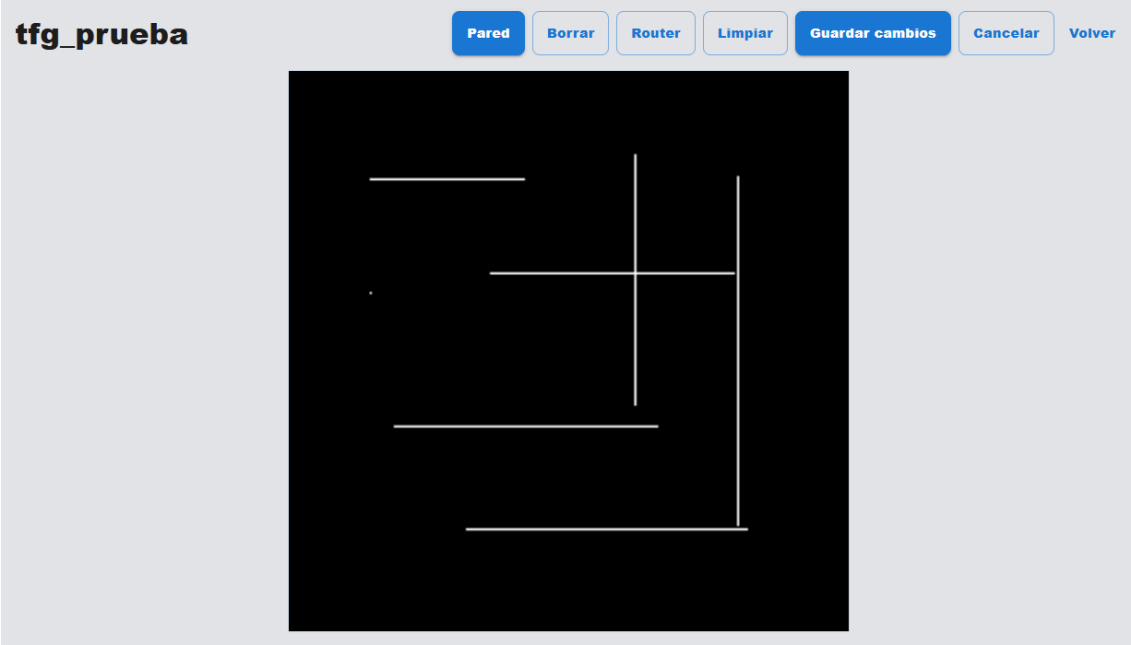

Figura 23 Página Editor de planos

Al guardar un proyecto, el editor no manda un solo dibujo, sino tres imágenes PNG en base64 :

- Imagen visual: Paredes y routers juntos. Es la imagen para que el usuario visualice su proyecto que se usa como miniatura del proyecto y como vista en el editor.
- Imagen de *layout*. Fondo negro con las paredes en blanco, sin *routers*. Es solo la estructura del entorno.
- Máscara de APs. Fondo negro con los *routers* en blanco, sin paredes. Es solo la posición de los emisores. Tanto esta imagen como la del layout son los dos canales de entrada para el modelo, como se verá más adelante.

5.2.6 TEST DE RED

Por último, como se mencionó en los capítulos anteriores, se ha desarrollado un test de red para comprobar el estado de la señal WiFi que recibe el equipo en tiempo real sin simulaciones. Esto permite al usuario comprobar la veracidad de la simulación así como tomar decisiones importantes en tiempo real como moverse de habitación para tener una reunión. Para ello la aplicación ejecuta un *script* que ejecuta el comando de Windows: `netsh wlan show interfaces.`

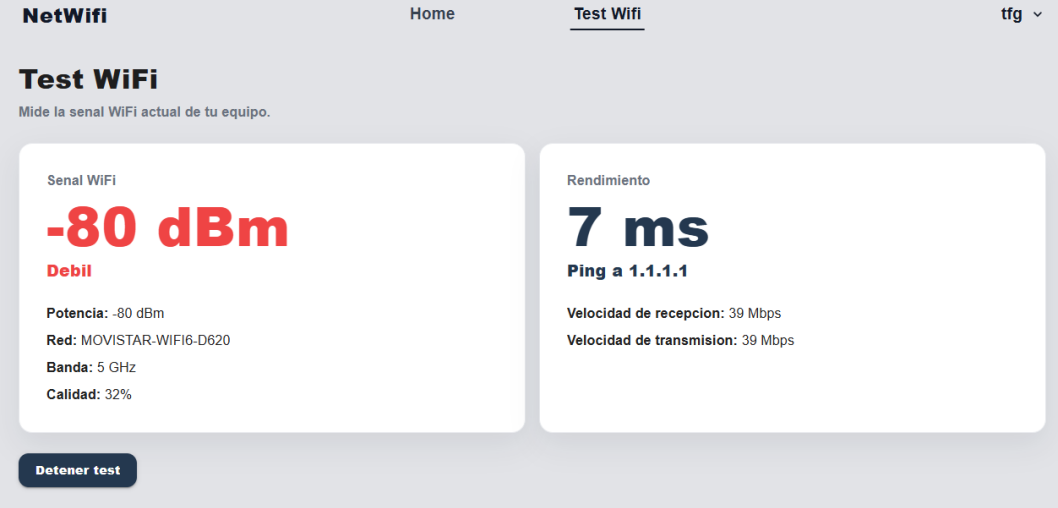

Figura 24 Test de WiFi

El primer bloque de información corresponde a la señal WiFi. El valor principal se expresa en dBm, una unidad utilizada para representar la potencia de la señal. En redes WiFi, estos valores suelen ser negativos, cuanto más cercano a cero sea el valor, mayor será la potencia recibida.

Además de la potencia, se muestra el nombre de la red WiFi a la que está conectado el equipo. También se incluye la banda de frecuencia a la que está conectado. Esta aplicación sirve para 5 GHz, muy común en la actualidad. Esto se debe a que el modelo de *Deep learning* ha sido entrenado con simulaciones de comportamiento de la señal inalámbrica operando en esta banda de frecuencia. Por último, se incluye un valor de calidad expresado en porcentaje, que facilita la interpretación de la señal para el usuario. En el ejemplo, la calidad es del 32 %, lo que refuerza la clasificación de la conexión como débil.

El segundo bloque de información está relacionado con el rendimiento de la conexión. Se muestra el resultado de una prueba de latencia mediante un ping a la dirección 1.1.1.1. Esta dirección corresponde a un servidor público de Cloudflare utilizado habitualmente para pruebas de conectividad, por lo que el valor obtenido indica el tiempo que tarda un paquete en ir y volver a un punto externo en Internet, expresado en milisegundos. Esto es un *test* de velocidad tradicional

5.3 BACKEND

El backend está hecho con Java 21 y Spring Boot. Se encarga de la recepción de las peticiones, la lógica de negocio, el modelo de datos y el acceso a la base de datos.

Su función es proporcionar una API REST que pueda ser consumida por la aplicación de escritorio. A través de esta API se gestionan las operaciones principales del sistema, como la autenticación del usuario, la gestión de proyectos y la solicitud de simulaciones.

Por un lado, gestiona a los usuarios (registro, inicio de sesión y borrado de cuenta). Por otro, gestiona los proyectos de cada usuario (crear, listar, editar, guardar y borrar). Y, por último, orquesta la simulación. Cuando el usuario la solicita, es el *backend* quien recupera las imágenes del proyecto y se las envía al servicio de inferencia, devolviendo después el resultado al *frontend*. De esta manera, el *frontend* nunca habla directamente con el modelo, lo que mantiene la seguridad centralizada en un único punto.

5.3.1 ARQUITECTURA DEL BACKEND

El código se ha organizado siguiendo una separación por responsabilidades, de forma que cada parte tenga una función clara y el proyecto sea fácil de mantener y ampliar. La estructura sigue la organización en capas típica de Spring: controlador, servicio, modelo y repositorio:

```
backend/                          → servidor Spring Boot
   Dockerfile                      → despliegue Docker en Render
   pom.xml                         → dependencias y build Maven
   mvnw                            → ejecutable Maven del proyecto
   src/main/java/backend/
      config/                  → seguridad, JWT y CORS
      controller/              → endpoints REST que utiliza el frontend
      exception/               → generalizar errores de la API
      model/                   → modelos de datos
         dto/                  → datos de entrada/salida
         entity/               → tablas JPA que representan la bd
      repository/              → acceso a PostgreSQL
      service/                 → lógica de negocio
      BackendApplication.java  → arranque de Spring Boot
   src/main/resources/         → configuración
     application.properties    → BD, JWT, CORS y ML
   src/test/                   → tests del backend
```

El diagrama de clases asociado a esta arquitectura es el siguiente:

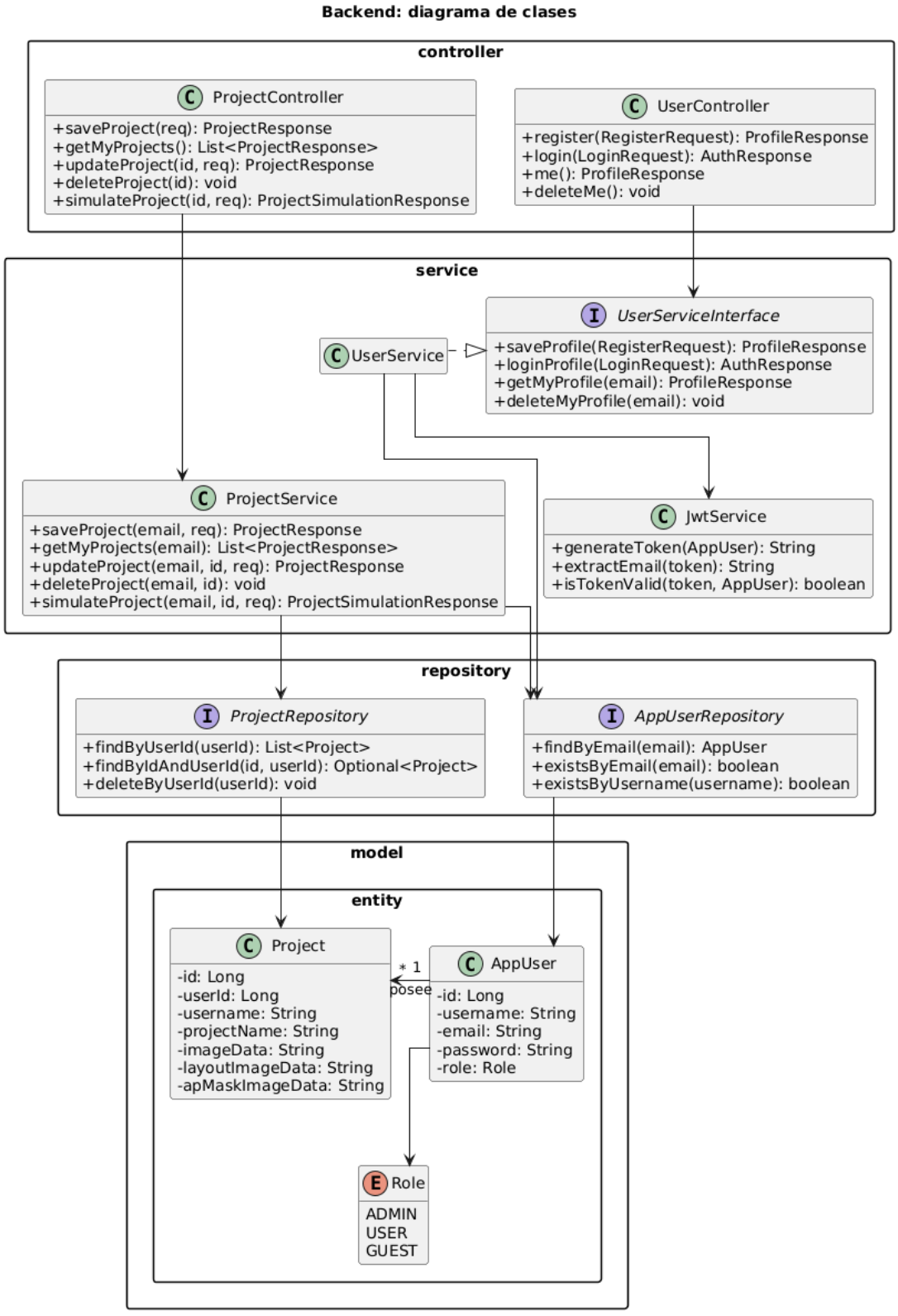

Figura 25 Diagrama de Clases del Backend

En la capa de *controller* se encuentran las clases `ProjectController` y `UserController`, que reciben las peticiones HTTP. La primera agrupa las operaciones sobre proyectos (guardar, listar, actualizar, borrar y simular) y la segunda las de usuario (registro, inicio de sesión, consulta y borrado de la cuenta). Estas clases no contienen lógica de negocio, se limitan a recibir la petición, delegar en la capa de servicio y devolver una respuesta en forma de objeto de transferencia de datos (DTO), como `ProjectResponse` o `AuthResponse.` Esto sirve para después mostrar el nombre del usuario o el proyecto para listarlo. No se muestran en el diagrama debido a que ensucian la imagen por la gran cantidad de DTOs que existen en el proyecto.

La capa de *service* contiene la lógica de negocio. `ProjectService` implementa las operaciones sobre proyectos y `UserService` las de usuario. Este último se define a través de la interfaz `UserServiceInterface`, lo que desacopla al controlador de la implementación concreta. Junto a ellos aparece `JwtService,` responsable de todo lo relacionado con los tokens: generarlos al iniciar, extraer de ellos el usuario y comprobar su validez. Conviene fijarse en que muchos métodos de los servicios reciben el email del usuario como parámetro, es así como se garantiza que cada usuario solo opera sobre sus propios datos.

La capa de *repository* se encarga del acceso a la base de datos. `ProjectRepository` y `AppUserRepository` son interfaces gestionadas por Spring Data JPA, que proporciona automáticamente la implementación de métodos como `findByEmail`, `findByUserId` o `findByIdAndUserId` sin necesidad de escribir las consultas a mano.

Por último, la capa de *model* reúne las entidades, que son la representación de los datos en el código. La entidad `AppUser` guarda los datos de la cuenta (nombre, email, contraseña y rol) y la entidad Project guarda el proyecto, incluyendo las tres imágenes en base64 (*imageData, layoutImageData y apMaskImageData*).

Ahora bien, este diagrama de clases describe la estructura estática del backend, qué clases existen y cómo se relacionan, pero no el orden en que se invocan durante una operación concreta. Para mostrar esa visión dinámica se emplean los diagramas de secuencia, que reflejan paso a paso la interacción entre estas clases a lo largo del tiempo y que se presentan a continuación.

5.3.2 DIAGRAMAS DE SECUENCIA

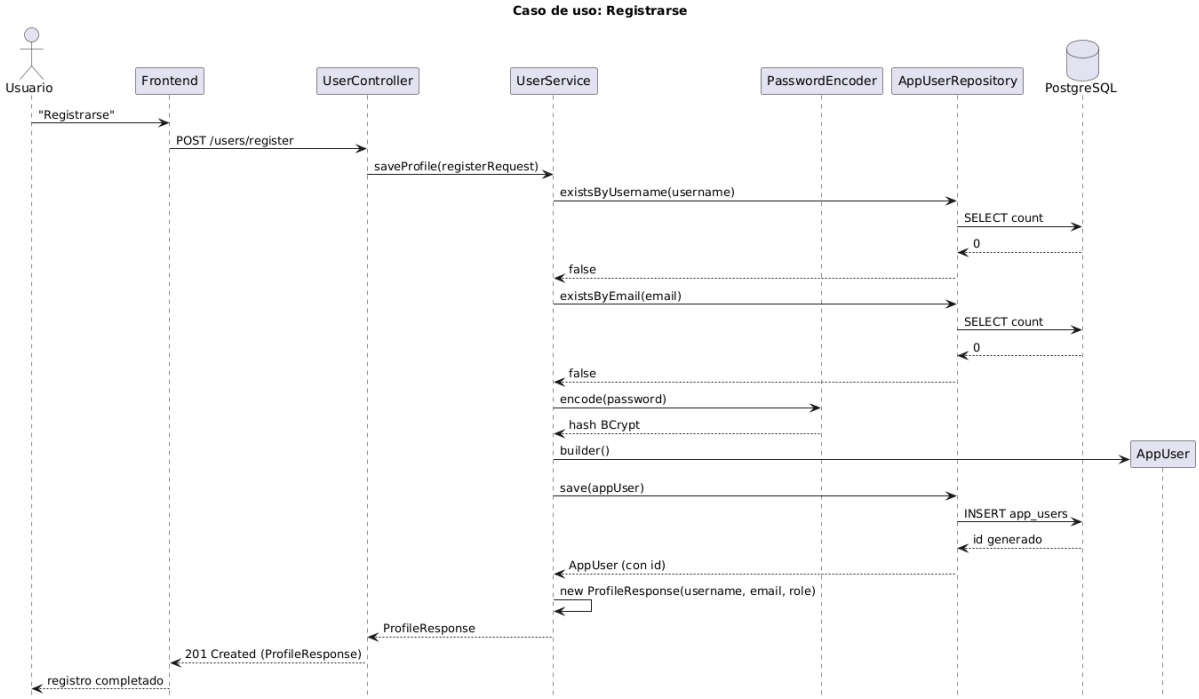

Figura 26 Diagrama de Secuencia de Registro

En el registro, el backend valida que el nombre de usuario y el correo no estén ya en uso antes de crear la cuenta. Un detalle relevante es que la contraseña nunca se almacena en claro: se cifra con un hash BCrypt, de manera que la base de datos guarda solo su versión cifrada. La respuesta que se devuelve al usuario no incluye la contraseña.

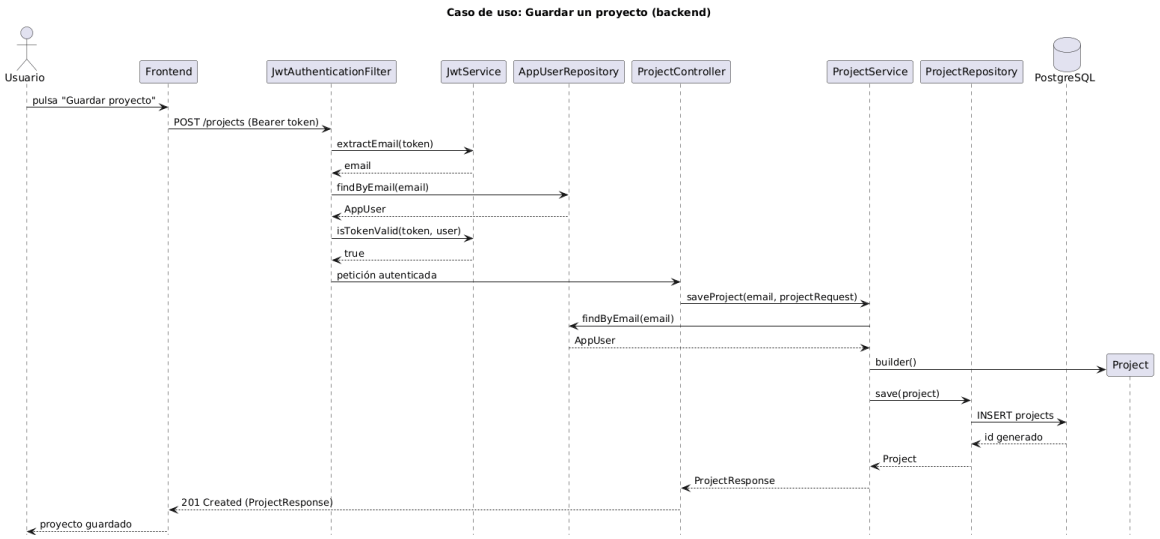

Figura 27 Diagrama de Secuencia de Guardar un Proyecto

En este caso, se observa como el token se valida antes de que llegue al controlador, por lo que cuando las demás clases actúan es seguro que el usuario esté autenticado.

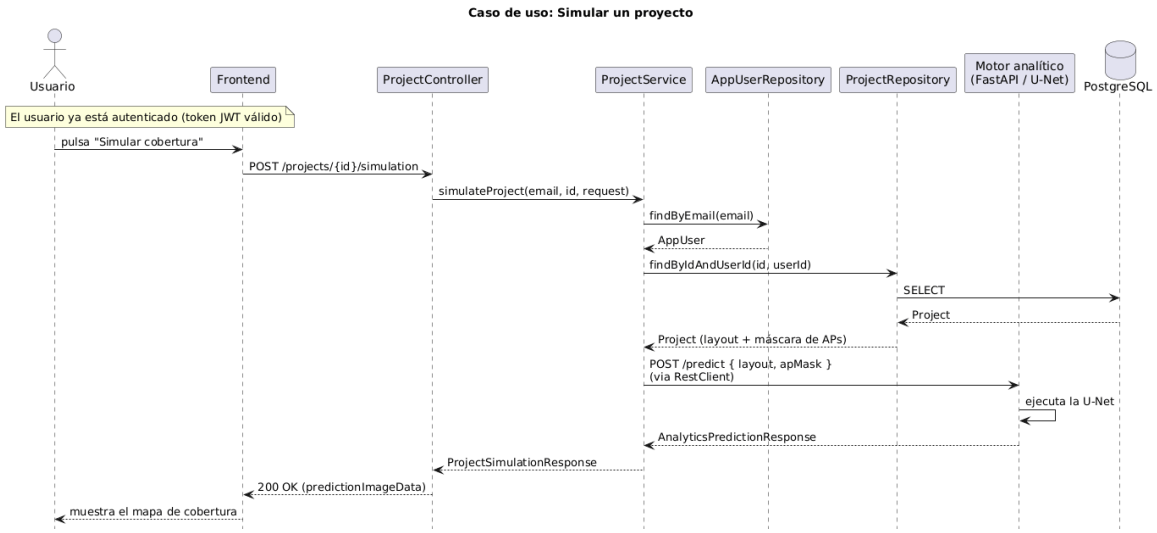

Figura 28 Diagrama de Secuencia de Simular un proyecto

En el caso de simular proyecto, se observa como primero se busca el proyecto concreto en la base de datos y luego se llama al servicio de FastAPI de Python perteneciente al módulo de *Deep learning,* que devuelve la imagen de predicción y se superpone sobre la del proyecto.

5.3.3 API REST Y ENDPOINTS

Como se ha mencionado en varias ocasiones en este proyecto, este módulo actúa como un orquestador entre los demás módulos, es por ello que la comunicación entre los diferentes mediante una API REST con el método HTTP para cada ruta merece una explicación aparte. Las rutas que se exponen de parte de este módulo son las siguientes:

RUTA	**Servicio**
POST /users/register	Registro de un nuevo usuario
POST /users/login	Inicio de sesión, devuelve el *token*
GET /users/me	Datos del usuario autenticado
DELETE /users/me	Borrado de la cuenta
POST /projects	Creación de un proyecto
GET /projects/me	Lista proyectos del usuario
PUT /projects/{id}	Actualización de un proyecto

DELETE /projects/{id}	Borrado de un proyecto
POST /projects/{id}/simulation	Ejecución de la simulación

Tabla 3 APIs del Backend

Para acceder a todas estas APIs, el *frontend* dispone de la URL del *backend* desplegado en RENDER, a la cual se le añade la ruta especificada en la tabla al final. Para el caso del servicio de la ejecución de la simulación, el controlador hace una llamada a la API de FastAPI, añadiendo esa ruta a la URL de Hugging Face donde esta el modelo desplegado.

5.4 BASE DE DATOS

Para almacenar la información de forma permanente se ha utilizado la base de datos relacional PostgreSQL, gestionada mediante Spring Data JPA. JPA se encarga de traducir automáticamente las entidades de Java en tablas de la base de datos, de forma que no es necesario escribir las consultas a mano para las operaciones habitual.

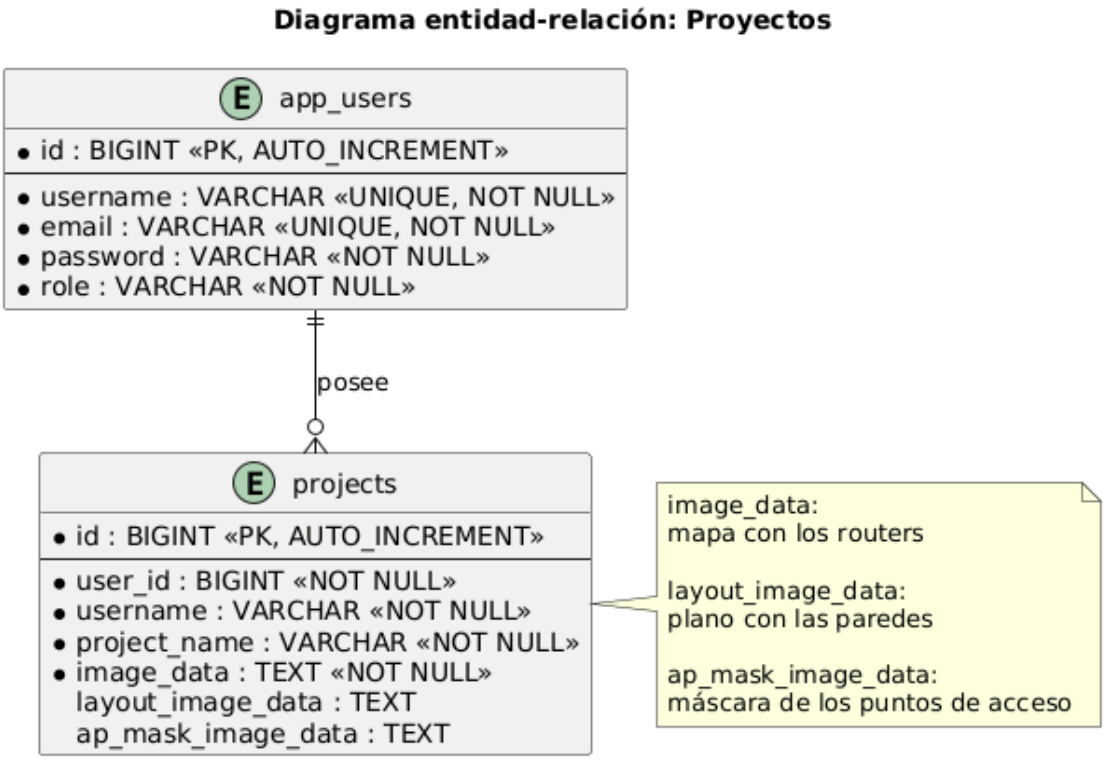

Figura 29 Diagrama Usuario - Proyectos

Este es un ejemplo del usuario que se guarda en base de datos. La de proyectos guarda, además del nombre y el propietario, las tres imágenes que genera el editor (la imagen visual, el layout y la máscara de APs), almacenadas como texto en formato base64. Ambas entidades están relacionadas. Un usuario puede tener varios proyectos, y cada proyecto pertenece a un único usuario.

A continuación, se muestran las tablas en las bases de datos utilizando la aplicación de pgAdmin 4:

```
SELECT * FROM app_users;
```

Data Output | Messages | Notifications

Showing rows: 1 to 3 | Page No:

	id [PK] bigint	email character varying (255)	password character varying (255)	role character varying (255)	username character varying (255)
1	1	tfg@gmail.com	$2a$10$AHKY/vyR8rClGjMJw2OQ0upiMsoaSy/wT/pahtMaShcb4ZTJexLXe	USER	tfg
2	2	nacho@gmail.com	$2a$10$HmrqvuRXEVi00rZ/zUS.GO6QvXkUDCpCTBkRy2tOjAKIwTuWUcXJ...	USER	nacho
3	3	1234@gmail.com	$2a$10$JAmnAz3MY7tu1DtTfddIHOlv4WLzQaAS4DQC83i8mYivPywQ5W5...	USER	test

Figura 30 Base de datos de Usuarios

Query | Query History

```
SELECT * FROM projects where id=3;
```

Data Output | Messages | Notifications

id [PK] bigint	ap_mask_image_data text	image_data text	layout_image_data text	project_name character varying (255)	user_id bigint	username character varying (255)

Figura 31 Base de datos de Proyectos Vacía

Query | Query History

```
SELECT * FROM projects;
```

Data Output | Messages | Notifications

	id [PK] bigint	ap_mask_image_data text
1	2	...
2	6	

Figura 32 Base de datos de Proyectos asociados

5.5 DATASET

El modelo se entrena con un conjunto de datos público de acceso abierto, disponible a través del repositorio Zenodo [21].

5.5.1 ORIGEN Y CARACTERÍSTICAS

Este *dataset* contiene mapas de cobertura simulados en la banda de 5 GHz para entornos interiores, generados a partir de planos de oficinas y aulas. Toda la información se representa como imágenes en escala de grises de tamaño fijo.

Para este proyecto se han utilizado los datos relacionados con *Radio Maps Estimation* (RME). Está formado por 60 planos de construcciones interiores con 1000 distribuciones de puntos de acceso para el entrenamiento, y 20 planos adicionales con 50 distribuciones para la prueba; las distribuciones son aleatorias y contemplan estructuras WLAN de entre uno y cinco puntos de acceso, sin embargo, en ese proyecto se van a utilizar de uno a 3 puntos de acceso. Esto es por dos razones: disminuye el tiempo de entrenamiento, realmente el usuario objetivo por el que se desarrolla esta plataforma no es común que utilice hasta 5 puntos de acceso. En total reúne unos 61 000 mapas, generados sin considerar interferencias entre canales, es decir, sin contar interferencias o redes vecinas.

Por tanto, el *dataset* utilizado contiene, para cada número de puntos de acceso AP[2]:

- 1 AP: 61.000
- 2 AP: 61.000
- 3 AP: 61.000

Por otra parte, todas las simulaciones se han llevado a cabo siguiendo el modelo `IEEE 802.11ax channel model`, mediante scripts de Matlab. Este modelo define como calcular las pérdidas en interiores basándose en las recomendaciones ITU mencionadas anteriormente y diferentes modelos físicos de propagación de ondas que se pueden encontrar en la documentación asociada. Cada mapa de cobertura está acotado entre dos valores de potencia.

La potencia máxima corresponde a la entregada por el transmisor, Pr = Pt = 26 dBm, un valor acorde con el de los equipos comerciales actuales, mientras que la potencia mínima viene dada por el nivel de ruido, Pr = KTB, donde K es la constante de Boltzmann, T la temperatura ambiente (290 K) y B el ancho de banda (80 MHz).

En cuanto a su formato, las imágenes tienen una profundidad de 8 bits y un tamaño de 256×256 [21].

[2] AP son las siglas de *Access Point* , que traducido al castellano es Punto de Acceso, se refiere a *router* o amplificador

El uso de un conjunto de datos simulado permite disponer de una gran cantidad de ejemplos sin necesidad de costosas campañas de medidas reales. Como contrapartida, el modelo aprende a reproducir el comportamiento del simulador, por lo que sus predicciones deben interpretarse en ese marco y no como medidas exactas de la cobertura real. Sin embargo, se conseguirá el objetivo de ofrecer una respuesta rápida mediante el uso del aprendizaje profundo para detectar patrones físicos en el conjunto de datos.

5.5.2 ESTRUCTURA

En este apartado se explica la forma en el que se presentan los datos para dar explicación a la solución de editor de proyectos del *frontend* y a la forma de indexar los datos para entrenar el modelo.

Cada muestra indexada está compuesta por un layout, una o varias máscaras de puntos de acceso y un mapa de cobertura objetivo. El número de máscaras depende de la configuración: una máscara para 1AP, dos para 2AP y tres para 3AP.

Para una configuración de un punto de acceso, por ejemplo el escenario 3 con la configuración 2, se emparejan un layout, una única máscara y el mapa objetivo:

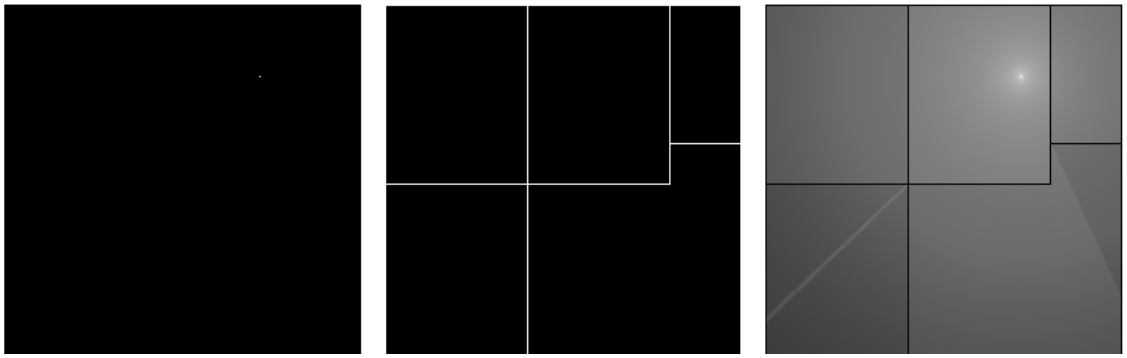

Figura 33 3_2.png *3.png* *3_2.png*

Para una configuración de dos puntos de acceso, como el escenario 5 con la configuración 3, aparecen dos máscaras, una por cada AP:

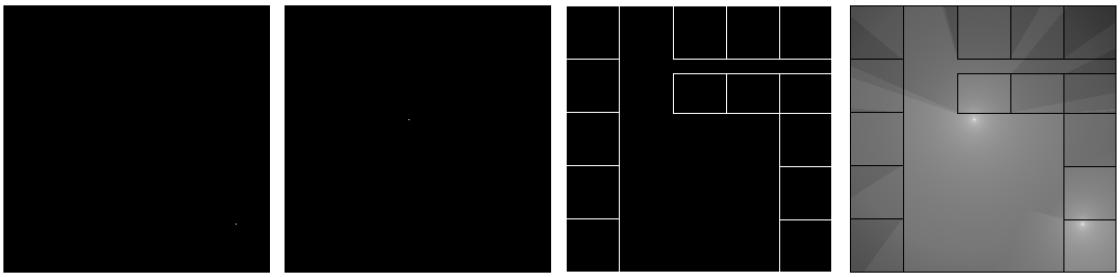

Figura 345_3_1.png *5_3_2.png* *5.png* *5_3.png*

Y para tres puntos de acceso, por ejemplo el escenario 12 con la configuración 2, se emparejan tres máscaras:

Figura 35 12_2_1.png *12_2_2.png* *12_2_3.png* *12.png*

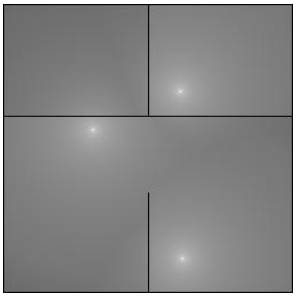

Figura 36 Mapa objetivo 12_2.png

En los tres casos, todos los archivos de una misma muestra comparten el identificador del escenario y de la configuración: `Txs/3AP/{scenario_id}_{config_id}_2.png`

Como se puede observar, existe una convención en los nombres, Por ejemplo, 25.png representa el layout correspondiente al escenario 25. Estas imágenes tienen una resolución de 256×256 píxeles y representan las paredes en blanco sobre un fondo negro. Es por esta razón que el editor de mapas del *frontend* solo permite al usuario dibujar sobre un "lienzo" con el mismo formato visual que aparece en el *dataset.*

5.5.3 INDEXADO, PARTICIÓN Y CARGA DE DATOS

La forma en la que se procesan y se presentan los datos al modelo, conocido como *data engineering* o en castellano ingeniería de datos, es una parte fundamental de cualquier trabajo que incluyan grandes cantidades de datos. Por eso, en este capítulo se explica la solución diseñada para indexar los datos y que el modelo los utilice de una forma "cómoda". Esta solución determina por tanto la forma en la que los datos para predecir imágenes por parte del usuario son enviados al modelo, pues ha de tener la misma estructura que los datos utilizados en el entrenamiento.

Para preparar los datos y que sean mas accesibles, se ha hecho un *script* de Python que recorre todo el *dataset* una sola vez y empareja todos los archivos a cada muestra, de la misma forma que se ha mostrado en el apartado anterior. Se ha creado un JSON donde se almacenan todos los datos con sus rutas correspondientes. De esta forma, se recorta tiempo y complejidad de computación al modelo.

```
{
      "sample_id": "2AP_43_686",
      "scenario_id": 43,
      "config_id": 686,
      "ap_count": 2,
      "split": "train",
      "layout_path": "Scennarios init/Scennarios B/43.png",
      "tx_paths": [
        "Txs/2AP/43_686_1.png",
        "Txs/2AP/43_686_2.png"
      ],
      "target_path": "Maps and cells/2AP/Maps/43_686.png"
    }
```

Código 2 Ejemplo de imágenes del Dataset procesadas

Este código es un ejemplo de una imagen indexada en el JSON con sus respectivas configuraciones. Cabe destacar que se han añadido todas las imágenes, ya que se ha añadido un campo que distingue para que propósito es esa parte del *dataset.* Tal y como mencionan en el repositorio de datos, la mejor forma para entrenar al modelo es la siguiente:

```
def get_split(scenario_id: int) -> str:
    # Decide si un scenario va a ir a train validation o test
    if 1 <= scenario_id <= 50:
        return "train"
    if 51 <= scenario_id <= 60:
        return "val"
    if 61 <= scenario_id <= 80:
        return "test"
    return "unassigned"
```

Código 3 Diferenciación entre train val y test

A partir del índice, una clase de conjunto de datos transforma cada muestra en el formato que necesita la red. Todas las imágenes se convierten a escala de grises, se redimensionan a un tamaño fijo de 256×256 píxeles y se normalizan a valores entre 0 y 1. Las máscaras de puntos de acceso se binarizan, de modo que solo distinguen entre presencia (valor 1) y ausencia (valor 0) de un AP.

Cuando hay varios APs, la solución que se ha propuesto es combinar en una sola imagen los 2 o 3 APs. Esto es porque el modelo va a tener 2 entradas fijas de imágenes, una con el *layout* y otra con las máscaras. De lo contrario, para cada distribución de APs tendría diferentes entradas y complicaría en gran medida la creación de la U-Net.

Con el plano y la máscara combinada ya preparados, se construye la entrada del modelo apilando ambos como dos canales: el primero contiene el plano y el segundo, la máscara de los puntos de acceso. Se obtiene así un tensor de dimensiones (2, 256, 256). La salida esperada es el mapa de cobertura, un único canal de dimensiones (1, 256, 256). Por tanto, predecir la cobertura es un problema de imagen a imagen: la red tiene que dar un valor (la señal) para cada uno de los 256 × 256 = 65.536 píxeles de la salida. No es clasificar la imagen ni detectar objetos, sino generar una imagen a partir de otra. Esto es lo que lleva de forma natural a usar una red convolucional con forma de codificador-decodificador, y en concreto una U-Net, que se explica en los apartados siguientes

5.6 REDES NEURONALES ARTIFICIALES

Las redes neuronales artificiales (RNA) son sistemas de procesamiento computacional fuertemente inspirados en el funcionamiento de los sistemas nerviosos biológicos (como el cerebro humano). Las RNA se componen principalmente de un gran número de nodos computacionales interconectados (denominados neuronas), que trabajan de forma distribuida para aprender colectivamente de la entrada y optimizar su resultado final.

Para ello, cargamos la entrada, generalmente en forma de vector multidimensional, en la capa de entrada, que la distribuye a las capas ocultas. Las capas ocultas, a su vez, toman decisiones a partir de la capa anterior y evalúan cómo un cambio estocástico interno perjudica o mejora la salida final. Este proceso se conoce como aprendizaje.

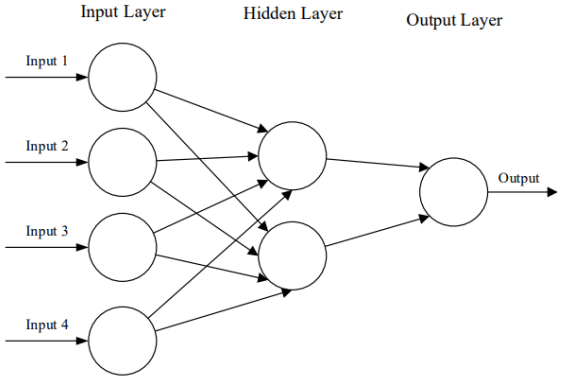

Figura 37 Representación de una Red Neuronal [22]

Cuando una red tiene muchas capas de neuronas, hablamos de aprendizaje profundo o *deep learning*. En ellas la no linealidad es imprescindible, si todas las capas fueran lineales, su composición sería de nuevo una sola transformación lineal y la profundidad no aportarían nada, en este proyecto se utilizan la ReLU y la sigmoide para evitarlo. Dentro de las redes neuronales de aprendizaje profundo existen diferentes arquitecturas, sin embargo, la más conocida es *Multi-Layer Perceptron* (MLP), en la que todas las neuronas de una capa se conectan a su vez con todas las neuronas de la siguiente capa, formando capas totalmente conectadas.

En cada capa, cada neurona aplica una función a la entrada en la que multiplica los valores de entrada por unos pesos, suma un sesgo y aplica una función de activación:

$$y = f(w1 * x1 + w2 * x2 + \cdots + wn * xn + b)$$

Los pesos y los sesgos son los parámetros del modelo, que el modelo ajusta para minimizar el error. Para ello se pueden llevar a cabo varios métodos, sin embargo, en este caso, el aprendizaje del presente sistema es de tipo supervisado, ya que el modelo se entrena con pares formados por una entrada y su salida correcta conocida (en este caso, el entorno y su mapa de cobertura)

El aprendizaje supervisado consiste en aprender mediante entradas preetiquetadas, que actúan como objetivos. Para cada ejemplo de entrenamiento, habrá un conjunto de valores de entrada (vectores) y uno o más valores de salida asociados. El objetivo de este tipo de entrenamiento es reducir el error de clasificación general del modelo, mediante el cálculo correcto del valor de salida del ejemplo de entrenamiento [22]. Esto es un problema de optimización, en el que el error es la diferencia entre lo que ha predicho el modelo y el *target* que ha de conseguir. La red busca los parámetros y su combinación que hacen mínima la función de pérdida y los ajusta mediante el descenso de gradiente. Es decir, calcula el gradiente (la dirección donde el error crece más deprisa) y se va en la dirección contraria. En el caso de una imagen, el problema es de regresión (se predice un valor continuo por píxel) y la pérdida es el error cuadrático medio (MSE), donde el signo no importa al estar elevado al cuadrado:

$$L = \left(\frac{1}{N}\right) * \sum^{n} \left(y_{pred_n} - y_n\right)^2$$

Y su gradiente:

$$\frac{dL}{dy_{pred}} = \left(\frac{2}{N}\right) * (y_{pred} - y_n)$$

Debido al cuadrado, penaliza mucho los errores grandes y es la magnitud que la red intenta reducir. La "velocidad" con la que se desciende el gradiente se configura como hiperparámetro y se llama *learning rate*. Una vez calculado cuánto se ha equivocado el modelo, se utiliza un optimizador que decide como ajustar los pesos.

Esta es la teoría básica del aprendizaje profundo, pero para este proyecto, se ha aplicado otra arquitectura diferente. Cuando la entrada de una red es una imagen, una red totalmente conectada resulta inviable.

En primer lugar, una imagen tiene una estructura donde la localización, el valor de cada píxel y sus vecinos importan, mientras que en vectores unidimensionales no. En el caso de aplicar la arquitectura de MLP, se necesitaría "aplanar" la imagen por lo que perdería esas características de estructura espacial. En segundo lugar, como se ha mencionado en el anterior apartado, la entrada en el proyecto es un tensor de 2 canales de 256x256, que resulta en 131.072 números de entrada. Como la salida ha de tener 256×256 = 65.536 píxeles, una única capa totalmente conectada que fuera de la entrada a la salida necesitaría del orden de $131.072 \times 65\,536 \approx 8{,}6 \cdot 10^9$. Esto es insostenible en términos de computación.

Por estas razones, y basado en los estudios reflejados en el estado del arte sobre las corrientes actuales de predicción de mapas de radio, se ha optado por el desarrollo de una Red Neuronal Convolucional (CNN)[3], concretamente con la arquitectura U-NET.

[3] *Convolutional Neural Network*

5.6.1 RED NEURONAL CONVOLUCIONAL

Una red neuronal convolucional es un tipo de red neuronal especialmente diseñada para trabajar con datos que presentan estructura espacial, como ocurre con las imágenes. Desde su primera concepción en 1986 [23], dedicado al reconocimiento de manuscritos, las redes neuronales convolucionales profundas han superado a las técnicas más avanzadas en muchas tareas de reconocimiento visual. Esto ocurre porque este tipo de redes explotan tres propiedades de las imágenes: la localidad (cada región se interpreta a partir de sus vecinos próximos), la invarianza por traslación (un mismo patrón debe reconocerse aparezca donde aparezca) y la compartición de parámetros (un mismo conjunto de pesos se reutiliza en toda la imagen).

La operación fundamental de una CNN es la convolución. Un *kernel* o filtro, que no es más que una pequeña matriz de pesos, se desliza sobre la imagen y, en cada posición, calcula una suma ponderada de los pixeles que cubre. Los valores del *kernel* no se fijan a mano, sino que se aprenden durante el entrenamiento, de modo que la red descubre por sí misma que patrones conviene detectar (bordes, esquinas, texturas).

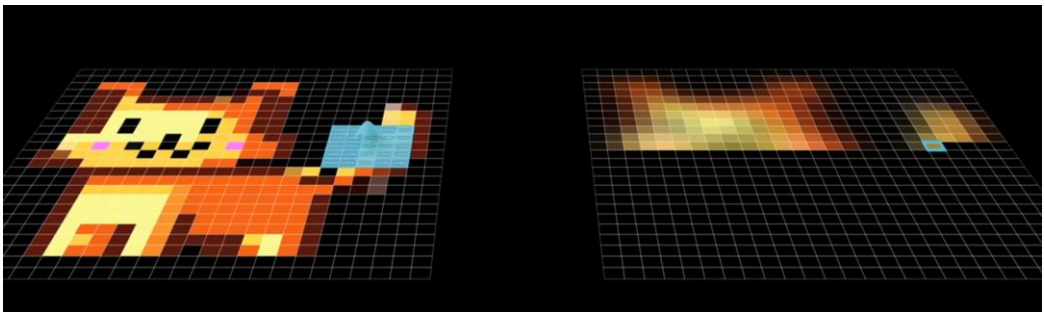

Figura 38 Representación de una Convolución en Imágenes [24]

Para que las capas profundas puedan relacionar elementos alejados entre si sin recurrir a *kernels* enormes, las CNN reducen progresivamente la resolución espacial de los mapas mediante operaciones de submuestreo (*pooling*), de manera que un *kernel* pequeño aplicado sobre un mapa reducido cubre, en la práctica, un área mayor de la imagen original.

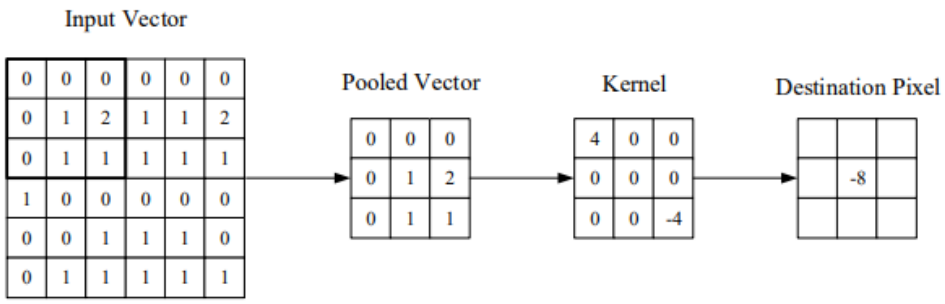

Figura 39 Representación de un Pooled Vector

Cada *kernel* produce un mapa de características, y una capa convolucional aplica varios *kernels* distintos, por ejemplo, uno que detecte líneas verticales y otro líneas horizontales, generando varios mapas.

De la misma forma que con el método de asignar pesos en la red neuronal clásica nace la arquitectura MLP, con este método convolucional surgieron otras arquitecturas basadas en este principio.

5.6.2 U-NET

La U-Net es una red convolucional con forma de codificador-decodificador y conexiones de salto (*skip connections*) entre ambos caminos. Fue propuesta originalmente por Ronneberger, Fischer y Brox para la segmentación de imágenes biomédicas [17], pero su capacidad para transformar una imagen de entrada en otra imagen espacialmente alineada la ha convertido en una arquitectura de referencia para problemas imagen-a-imagen.

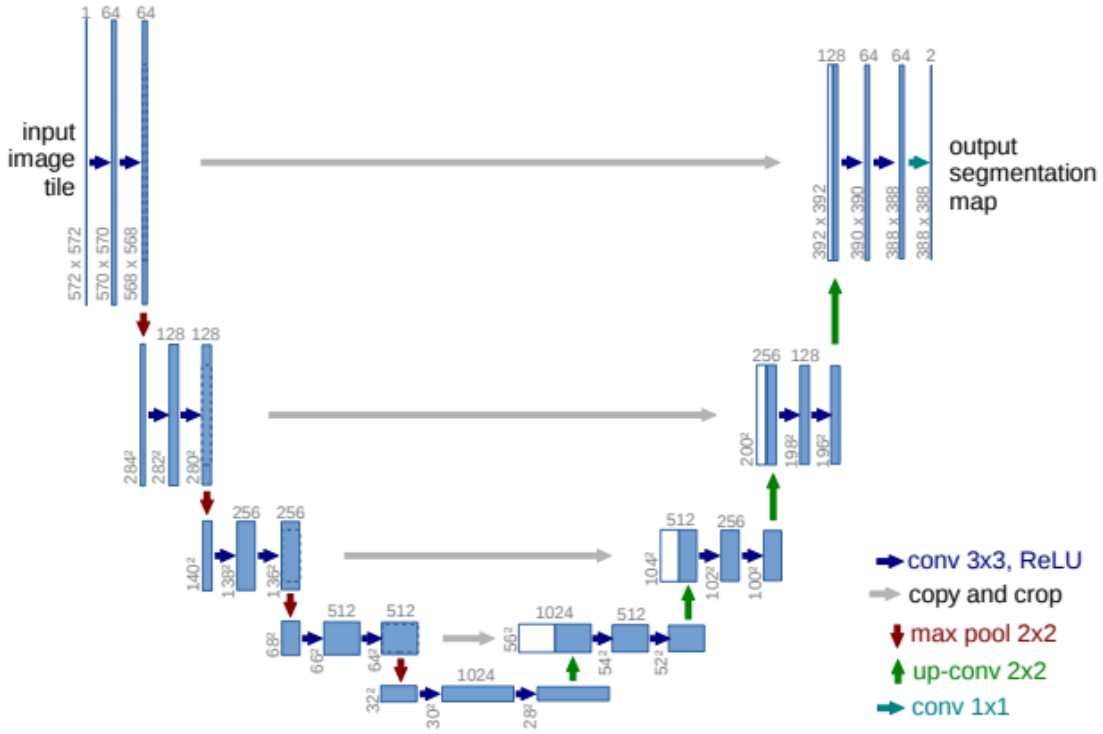

Figura 40 U-Net [17]

La rama descendente, o codificador, aplica sucesivos bloques convolucionales y reducciones de resolución mediante *max pooling*; al descender, la red pierde detalle espacial, pero gana comprensión del contexto global, a la vez que aumenta el número de canales. En el punto más bajo se sitúa el cuello de botella (*bottleneck*), la representación más comprimida del entorno. La rama ascendente, o decodificador, aumenta progresivamente la resolución y reduce los canales hasta recuperar el tamaño original y reconstruir la imagen de salida.

Las conexiones de salto (*skip connections,* en la imagen *copy and crop*) enlazan cada nivel del codificador con el bloque del decodificador de la misma resolución, concatenando ambos mapas de características por canales. Su función es recuperar el detalle espacial que se pierde al comprimir la imagen. El codificador capta el contexto global, pero diluye la posición exacta de paredes y emisores, de modo que estas conexiones reinyectan esa información de alta resolución en el decodificador. Así, cada bloque de subida combina el contexto global con el detalle posicional, produciendo un mapa de cobertura nítido y alineado con el plano de entrada en lugar de uno borroso.

5.6.2.1 U-Net implementada

La red se ha implementado en PyTorch a partir de tres bloques reutilizables, siguiendo la estructura habitual de las implementaciones de U-Net desde cero [25] y siguiendo la personalización del estudio mencionado en el estado del arte, que es referente en el sector.

El primer bloque es el doble bloque convolucional, en el que se ejecuta dos veces la secuencia de convolución.

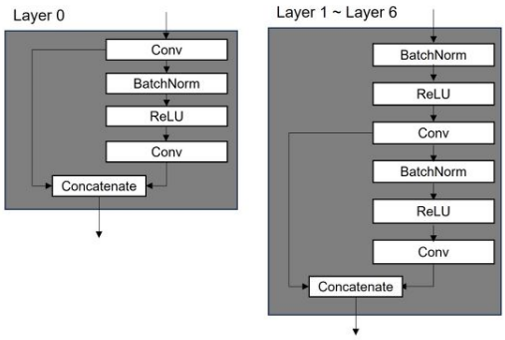

Figura 41 Diseño del Encoder-Decoder [18]

La implementación en PyTorch es la siguiente:

```
class DoubleConv(nn.Module):
    def __init__(self, in_channels: int, out_channels: int, dropout: float =
0.0) -> None:
        super().__init__()
        self.block = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1,
bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1,
bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Dropout2d(dropout) if dropout > 0 else nn.Identity(),
        )
    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return self.block(x)
```

Código 4 Módulo Doble Convolución U-Net

Primero, hay que destacar que se ha optado por una doble convolución. Esta solución permite tener menos pesos (18) mientras que la operación de convolución da el mismo resultado que un *kernel* 5x5. Esto es porque una convolución se hace sobre la anterior, que ya ha tenido en cuenta los vecinos, por lo que se puede obtener el doble de vecinos. A continuación, se aplica la normalización por lotes y ReLU. Es la unidad básica que se repite a lo largo de toda la red. La convolución se define sin sesgo (`bias=False`) porque la normalización posterior lo hace innecesario. Por último, puesto que en el entrenamiento se tuvieron procesos donde la red sobre aprendía, se ha introducido `nn.Dropout2d(dropout)` que apaga durante el entrenamiento una parte aleatoria de la red para evitar esta anomalía.

Por otra parte, el bloque de bajada:

```
class DownBlock(nn.Module):
    def __init__(self, in_channels: int, out_channels: int) -> None:
        super().__init__()
        self.block = nn.Sequential(
            nn.MaxPool2d(kernel_size=2),
            DoubleConv(in_channels, out_channels),
        )
```

Código 5 Bloque de Bajada U-Net

Un paso del codificador: primero `MaxPool2d(2)` reduce el tamaño a la mitad y luego un `DoubleConv` que procesa y duplica los canales. Bajar resolución y subir canales es el principio que sigue este módulo. Cuantos más canales existan, más patrones aprende el modelo, pero más poder computacional requiere. Empieza con 48 y acaba con 768.

Por último, el bloque de subida

```
class UpBlock(nn.Module):
    def __init__(self, in_channels: int, skip_channels: int, out_channels:
int) -> None:
        super().__init__()
        self.up = nn.Sequential(
            nn.Upsample(scale_factor=2, mode="bilinear",
align_corners=False),
            nn.Conv2d(in_channels, out_channels, kernel_size=1, bias=False),
        )
        self.conv = DoubleConv(out_channels + skip_channels, out_channels)

    def forward(self, x: torch.Tensor, skip: torch.Tensor) -> torch.Tensor:
        x = self.up(x)
        x = torch.cat([skip, x], dim=1)
        return self.conv(x)
```

Código 6 Bloque de Subida U-Net

En este módulo se definen las conexiones de salto que son de vital importancia en la arquitectura, y dobla el tamaño de la resolución.

Por último, se define la configuración se construye la U-Net, el proceso sería el siguiente

Etapa	Operación principal	Tamaño espacial	Canales
Entrada	Imagen de entrada formada por plano y máscara de puntos de acceso	256 × 256	2
input_block	DoubleConv	256 × 256	48
down1	MaxPool2d(2) + DoubleConv	128 × 128	96
down2	MaxPool2d(2) + DoubleConv	64 × 64	192
down3	MaxPool2d(2) + DoubleConv	32 × 32	384
down4	MaxPool2d(2) + DoubleConv	16 × 16	768
bottleneck	MaxPool2d(2) + DoubleConv	8 × 8	768
up1 .. up5	Upsample + skip + DoubleConv	16 × 16 → 256 × 256	768 → 48
Salida	Convolución 1×1 + sigmoide	256 × 256	1

Tabla 4 Tabla U-Net Completa

Pese a que la U-Net original utiliza 68 canales, en este trabajo, debido a las limitaciones computacionales se ha optado por disminuir a 48. Con esta decisión, el número de parámetros que tiene la red es de 43.423.249 parámetros. Esto es porque Una convolución tiene nº kernels × pesos por kernel. Como hay un kernel por canal de salida y cada kernel tiene canales_entrada × alto × ancho.

5.6.3 ENTRENAMIENTO DEL MODELO

Tras definir la arquitectura, los parámetros de la red parten de valores esencialmente aleatorios. El entrenamiento es el proceso que los ajusta hasta que la red reproduce la relación entre la estructura del entorno y la distribución de la señal.

5.6.3.1 Aprendizaje supervisado y retropropagación

Se trata de un aprendizaje supervisado, porque para cada ejemplo se conoce la respuesta correcta: el mapa de cobertura de referencia. El procedimiento, para cada muestra, consta de cuatro pasos. Primero, la red calcula su predicción a partir de la entrada (paso hacia delante). Después, una función de pérdida mide el error entre la predicción y la referencia. A continuación, mediante retro propagación (*backpropagation*) se calcula el gradiente de la pérdida respecto a cada parámetro de la red; este algoritmo aplica la regla de la cadena hacia atrás, capa por capa, para obtener de forma eficiente cuánto contribuye cada peso al error. Por último, el optimizador modifica los parámetros en la dirección que reduce la pérdida. Repitiendo el ciclo sobre miles de ejemplos, la red aprende a generalizar.

En PyTorch, este ciclo se expresa de forma directa: el cálculo de gradientes se realiza con `loss.backward()` y la actualización de los pesos con `optimizer.step()`, precedidos de `optimizer.zero_grad()` para no acumular los gradientes de iteraciones anteriores.

5.6.3.2 Función de pérdida

Para medir el error se emplea el error cuadrático medio (MSE, *Mean Squared Error*), que compara el mapa predicho y el de referencia píxel a píxel. Calcula la diferencia entre ambos valores en cada píxel, la eleva al cuadrado y promedia el resultado sobre todos los píxeles. Como se ha explicado anteriormente.

Su elección responde a varias razones. El problema es de regresión sobre valores continuos, para la que el MSE es la medida estándar. Al elevar al cuadrado, penaliza con más fuerza los errores grandes que los pequeños, lo que evita fallos groseros en zonas concretas del mapa.

5.6.3.3 Optimizador: AdamW

Conocer el error no basta, hay que decidir cómo corregir los parámetros. De ello se ocupa el optimizador, que se apoya en el descenso de gradiente. El gradiente indica, para cada parámetro, en qué dirección modificarlo para que el error disminuya, y el optimizador da un pequeño paso en esa dirección. El optimizador elegido es AdamW[4] [26]. combina dos ideas: por un lado, una forma de inercia o *momentum* que promedia los gradientes recientes y suaviza la trayectoria; por otro, una tasa de aprendizaje adaptada de forma individual a cada parámetro, en función de la magnitud típica de su gradiente.

```
optimizer = torch.optim.AdamW(
        model.parameters(),
        lr=float(training_config["learning_rate"]), # 0,0001
        weight_decay=float(training_config.get("weight_decay", 0.0)),
# 0,00001
```

Código 7 Optimizer AdamW

La tasa de aprendizaje (*learning rate*) controla el tamaño de los pasos. Un valor demasiado alto vuelve el entrenamiento inestable y uno demasiado bajo lo hace muy lento. El valor empleado, 0,0001, es conservador y favorece la estabilidad. El término de regularización o *weight decay* penaliza ligeramente que los pesos crezcan en exceso, lo que ayuda a reducir el sobreajuste.

5.6.3.4 Bucle de entrenamiento, validación e hiperparámetros

El entrenamiento del modelo se organiza en épocas. Una época corresponde a una pasada completa por el conjunto de entrenamiento, es decir, el modelo llega a ver todas las muestras disponibles para aprender. Además para cada época se remueve el conjunto de entrenamiento. Sin embargo, estas muestras no se introducen una a una ni todas al mismo tiempo, sino agrupadas en lotes, también llamados *batches*. Para ello se utiliza un `DataLoader`, que se encarga de cargar las muestras, agruparlas en tensores y, durante el entrenamiento, barajarlas al comienzo de cada época.

En cada época se diferencian dos fases. La primera es la fase de entrenamiento, en la que la red procesa el conjunto de entrenamiento lote a lote. Para cada lote, el modelo genera una predicción, calcula la pérdida respecto al mapa objetivo, obtiene los gradientes mediante retropropagación y actualiza sus parámetros mediante el optimizador. La segunda es la fase de validación, en la que el modelo se evalúa sobre un conjunto distinto de datos. En esta fase no se calculan gradientes ni se modifican los pesos de la red,

[4] Adam (*Adaptive Moment Estimation)*

únicamente se mide el error para comprobar cómo se comporta el modelo con muestras que no se han utilizado directamente para ajustar sus parámetros.

El núcleo del bucle de entrenamiento es común a ambas fases, aunque solo en la fase de entrenamiento se realiza la actualización de pesos.

Todas las decisiones anteriormente mencionadas se concentran en esta tabla de hiperparámetros y especificaciones.

Hiperparámetro	**Valor**
Canales base (base_channels)	48
Dropout (cuello de botella)	0,05
Función de pérdida	Error cuadrático medio (MSE)
Optimizador	AdamW
Tasa de aprendizaje	0,0001
Regularización (weight decay)	0,00001
Tamaño de lote (batch)	6
Número de épocas	33
Dispositivo	GPU (CUDA) Para utilizar la gráfica
Tamaño de imagen	256 × 256
Semilla aleatoria	Fija, para reproducibilidad

Tabla 5 Hiperparámetros y Especificaciones del entrenamiento

5.7 SERVICIO DE INFERENCIA Y DESPLIEGUE DEL MODELO

Una vez entrenada la U-Net hace falta un componente que la ejecute y la ponga a disposición del resto del sistema. De ello se encarga el servicio de inferencia, que aloja el modelo y atiende las peticiones de predicción. En este apartado se describe su diseño, la interfaz que expone y cómo se ha empaquetado y desplegado en la nube.

5.7.1 DISEÑO Y API DEL SERVICIO

El servicio se ha desarrollado en Python con FastAPI y constituye un microservicio independiente, separado del *backend*. Su única responsabilidad es recibir las imágenes separadas, ejecutar la red y devolver la predicción. Aislar el modelo, que arrastra dependencias pesadas como PyTorch, permite construirlo, actualizarlo o sustituirlo sin

afectar al resto de la aplicación y desplegarlo en una plataforma especializada. Como el backend ya valida al usuario, el servicio queda detrás de él y se limita a predecir, sin lógica de negocio ni acceso a la base de datos.

La interfaz consta de dos endpoints, y tanto la entrada como la salida viajan en JSON, con las imágenes codificadas en base64:

RUTA	Servicio
GET /health	Comprobación de estado del servicio. Confirma que el servicio de inferencia está activo y resulta útil para detectar si se encuentra disponible o para despertarlo cuando permanece en reposo.
POST /predict	Servicio de predicción. Recibe el *layout* del entorno y la máscara de puntos de acceso, ejecuta el modelo entrenado y devuelve el mapa de cobertura estimado.

Tabla 6 Endpoints FastApi

5.7.2 EL PROCESO DE INFERENCIA

Al arrancar, el servicio carga una sola vez el checkpoint best_model.pt, el seleccionado durante el entrenamiento por su menor error de validación, y pone el modelo en modo de evaluación, en el que se desactivan mecanismos propios del entrenamiento como el dropout. A partir de ahí, cada petición sigue los mismos pasos, las dos imágenes recibidas en base64 se decodifican y se les aplica el mismo preprocesado que en el entrenamiento. Se construye el tensor de entrada de dos canales donde el primer canal es el *layout* y el segundo la máscara de los APs. Reutilizar el preprocesado del entrenamiento es lo que garantiza que el modelo recibe la información en la representación que aprendió, por esta razón se ha escogido el editor de mapas mencionado.

La red se ejecuta desactivando el cálculo de gradientes, innecesario en inferencia, lo que ahorra memoria y tiempo. La salida es un único canal de 256×256 con los valores de cobertura estimados, que se transforma en una imagen de mapa de calor (heatmap) y se codifica de nuevo en base64 para devolverla. Es esa imagen la que el frontend acaba superponiendo sobre el plano del usuario.

5.7.3 CONTENEDORIZACIÓN Y DESPLIEGUE EN HUGGING FACE

Para desplegar el servicio se han subido a Hugging Face Spaces todos los archivos que lo componen: el módulo de FastApi y de la inferencia, la definición de la U-Net, la configuración, las dependencias (`requirements.txt`), el `Dockerfile` y el modelo entrenado `best_model.pt`. A partir del `Dockerfile`la plataforma construye la imagen y levanta el servicio, que queda alojado en la nube y accesible en una URL del tipo `https://ignaciolop-wifi-coverage-analytics.hf.space`. Como el modelo se sube junto al resto, el contenedor dispone de los pesos directamente, sin descargarlos de ningún servicio externo.

Por su parte, el *backend*, alojado en Render, necesita conocer esa dirección para poder llamar al servicio de inferencia. En lugar de escribirla en el código, la lee de una variable de entorno, lo que permite cambiarla entre los entornos de desarrollo y de producción sin modificar la aplicación.

A lo largo de este capítulo se ha descrito la construcción completa del sistema siguiendo el camino del usuario, desde la aplicación de escritorio con la que dibuja su plano y consulta la red, pasando por el *backend* que centraliza la seguridad, los proyectos y la base de datos, hasta el conjunto de datos, el modelo de aprendizaje profundo que constituye el núcleo del trabajo, su entrenamiento y el servicio que lo ejecuta en la nube. El resultado es una herramienta funcional en la que cada componente cumple una responsabilidad clara y en la que la U-Net traduce el plano y la posición de los puntos de acceso en un mapa de cobertura estimado.

Capítulo 6. ANÁLISIS DE RESULTADOS

En este capítulo se evalúa el modelo U-Net entrenado en el capítulo anterior. Primero se describe la metodología de evaluación y se justifica por qué se emplean unas métricas para entrenar y otras para medir los resultados. A continuación, se analiza la convergencia del entrenamiento, los resultados cuantitativos (globales y por número de puntos de acceso), varios ejemplos cualitativos y el coste de inferencia. Por último, se discuten las limitaciones y el alcance real de las cifras obtenidas.

6.1 METODOLOGÍA

El modelo evaluado es el punto de control `best_model.pt` , correspondiente a la época 28 (la de menor pérdida de validación), con sus 43 423 249 parámetros ya fijados. La evaluación se realiza sobre el conjunto de *test*, formado por los escenarios 61–80, que el modelo no ha visto durante el entrenamiento. Este conjunto reúne 3000 muestras (1000 por cada número de puntos de acceso: 1, 2 y 3 APs), y la predicción de cada una se compara píxel a píxel con su mapa de cobertura objetivo.

Para que las cifras tengan sentido físico conviene recordar, que cada mapa está acotado entre la potencia del transmisor (Pt = 26 dBm) y el nivel de ruido (KTB $\approx -95\ dBm$ para B = 80 MHz y T = 290 K), lo que supone un rango dinámico de ≈ 121 dB. Como los mapas se normalizan al intervalo [0, 1], un error en esa escala normalizada puede traducirse de forma aproximada a decibelios multiplicándolo por 121.

El modelo se entrena minimizando el error cuadrático medio (MSE) pero se evalúa con el error absoluto medio (MAE) y la raíz del error cuadrático medio (RMSE). La razón es que cada métrica cumple un papel distinto. El MSE es la función de pérdida (entrenamiento). Se usa para aprender porque es suave y derivable en todo su dominio y tiene un gradiente sencillo, justo lo que necesitan el descenso de gradiente y la retropropagación. Además, al elevar el error al cuadrado, penaliza con fuerza los fallos grandes, lo que orienta bien el entrenamiento.

El MAE y el RMSE son métricas de evaluación. No intervienen en el aprendizaje, solo se calculan al final para comunicar la calidad del modelo de forma interpretable. El problema del MSE para reportar es que está en unidades "al cuadrado" y no dice nada a simple vista. Por eso:

- $RMSE = \sqrt{MSE}$. Es el mismo MSE, pero devuelto a las unidades de la magnitud predicha (intensidad normalizada, convertible a dB). Indica el error "típico" en la escala de la señal.

- MAE = error absoluto medio. Es el error medio en las mismas unidades, pero sin elevar al cuadrado, por lo que es más robusto a valores atípicos y representa mejor el error habitual

- Comparar el MAE y el RMSE sirve para entender mejor cómo se equivoca el modelo. El MAE indica el error medio de forma directa, mientras que el RMSE da más importancia a los errores grandes, porque eleva las diferencias al cuadrado. Por eso, si ambos valores son parecidos, significa que los errores están bastante repartidos y no hay fallos muy grandes. En cambio, si el RMSE se separa mucho del MAE, quiere decir que hay algunos puntos donde el modelo se equivoca bastante más, aunque el error medio general no sea tan alto.

6.2 CONVERGENCIA DEL ENTRENAMIENTO

La figura 41 se observa la evolución de la pérdida (MSE) de entrenamiento y validación a lo largo de las épocas, en escala logarítmica.

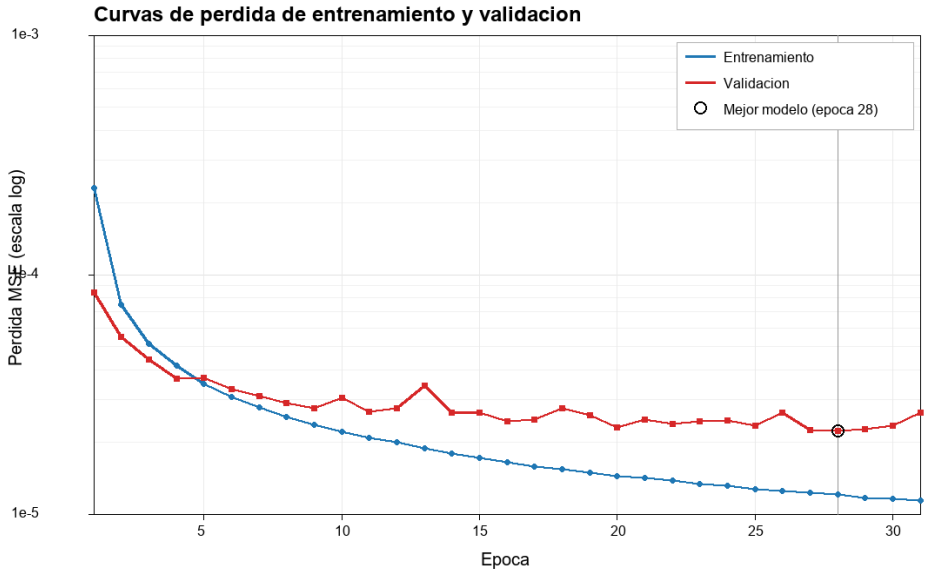

Figura 42 Curvas de pérdida de entrenamiento y validación

Ambas curvas descienden con rapidez en las primeras épocas y después se estabilizan. La menor pérdida de validación se alcanza en la época 28 (MSE de validación ≈ $2{,}23 \cdot 10^{-5}$, frente a un MSE de entrenamiento ≈ $1{,}20 \cdot 10^{-5}$), que es el modelo seleccionado. A partir de ese punto, la pérdida de entrenamiento sigue bajando lentamente mientras la de validación deja de mejorar, lo que indica que continuar entrenando solo aportaría memorización, desembocando en sobre aprendizaje.

6.3 RESULTADOS CUANTITATIVOS

La Tabla 6.2 desglosa el error según el número de puntos de acceso presentes en la muestra.

Configuración	MAE	RMSE	MAE (dB)	RMSE (dB)
1 AP	0,00306	0,00828	≈ 0,37	≈ 1,00
2 AP	0,00182	0,00437	≈ 0,22	≈ 0,53
3 AP	0,00156	0,00344	≈ 0,19	≈ 0,42

Tabla 7 Error desglosado por número de puntos de acceso

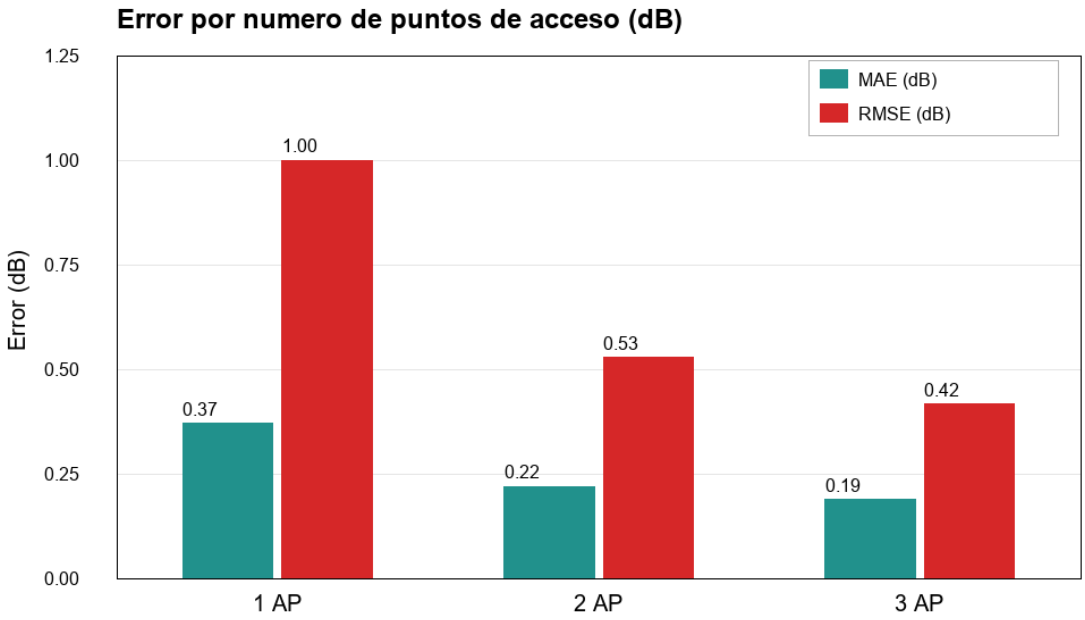

Tabla 8 MAE y RMSE (en dB) en función del número de puntos de acceso

Se observa que el error es mayor en los escenarios con un único punto de acceso y disminuye progresivamente en las configuraciones con dos y tres puntos de acceso. Esto puede deberse a que, cuando hay más puntos de acceso, la cobertura tiende a estar más repartida por el entorno y el mapa resultante presenta menos zonas extremas de baja señal. En cambio, con un único punto de acceso aparecen zonas más alejadas o peor cubiertas, donde el modelo puede cometer errores algo mayores.

6.4 RESULTADOS CUALITATIVOS

Más allá de las cifras, conviene comprobar visualmente la calidad de las predicciones. El mapa simulado se encuentra a la izquierda, el generado por el modelo a la derecha, y a la derecha, se encuentra la diferencia.

Figura 43 Ejemplo 1 Punto de acceso escenario 71

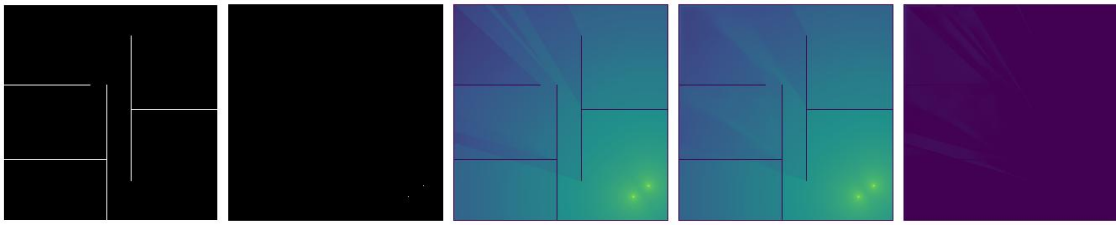

Figura 44 Ejemplo 2 Puntos de acceso escenario 77

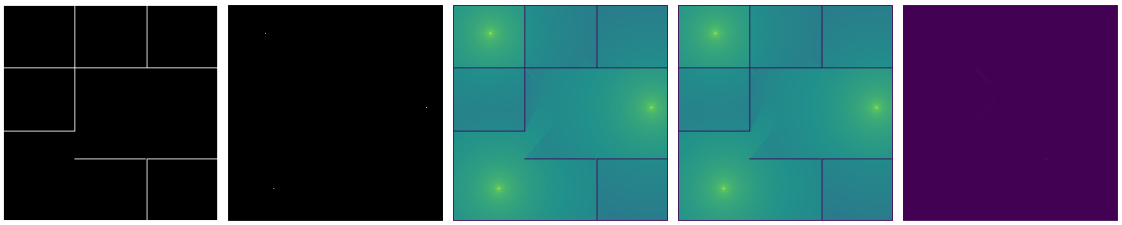

Figura 45 Ejemplo 3 Puntos de Acceso escenario 66

En los tres casos, la predicción reproduce con fidelidad la forma del mapa objetivo, el degradado de potencia alrededor de cada emisor y la atenuación que introducen las paredes. Además, al pasar de 1 a 3 puntos de acceso se observa cómo la cobertura se vuelve más homogénea y brillante, al sumarse las contribuciones de los distintos emisores. El panel de error absoluto es prácticamente uniforme y oscuro (error casi nulo), con trazas tenues localizadas en los bordes de las paredes, que es donde la señal cambia de forma más brusca y la red tiene más dificultad para ajustar la transición exacta.

Para acotar el comportamiento del modelo, se muestra a continuación el mejor y el peor caso de todo el conjunto de *test*.

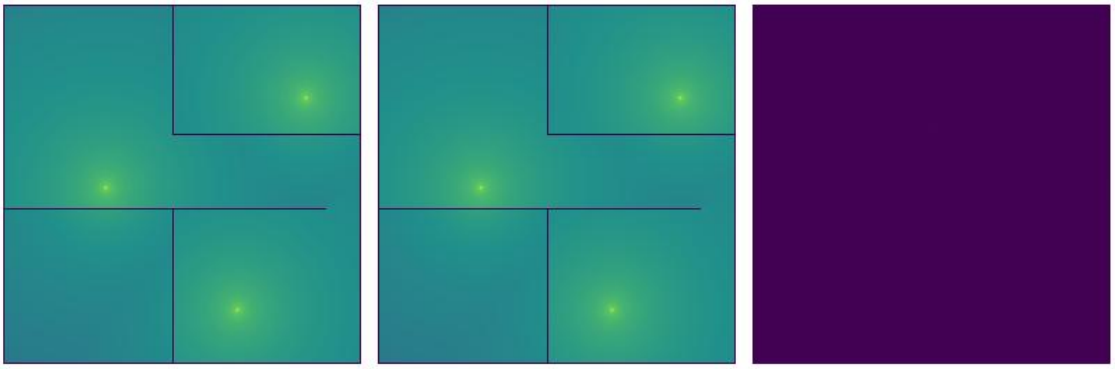

Figura 46 Mejor Predicción

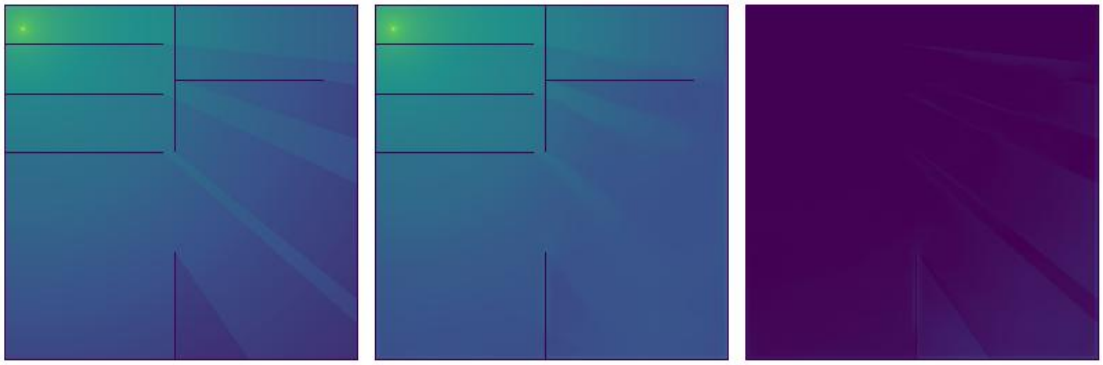

Figura 47 Peor Predicción

Se observan una predicción acorde a lo esperado. Es puntos donde hay muy poca señal o cambios bruscos, con un AP el modelo sufre más, sin embargo, con 3 APs, al cubrir todo el mapa de una buena calidad de señal, el modelo es capaz de replicarlo a la perfección.

Una de las principales ventajas del enfoque basado en aprendizaje profundo frente a la simulación física es la rapidez en la predicción. Medido sobre la GPU empleada (NVIDIA RTX 3060 Ti), el modelo tarda de media ≈ 9 ms en generar un mapa de cobertura completo de 256×256, es decir, casi instantánea, por lo que al estar desplegada en el modelo con mejores servidores el usuario tendrá una mejor experiencia.

Por tanto, los resultados confirman que la U-Net reproduce con fidelidad el comportamiento del simulador a partir únicamente del plano y de la posición de los emisores, con un error medio inferior al decibelio y una predicción casi instantánea.

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

El presente Trabajo de Fin de Grado ha abordado el desarrollo de una herramienta accesible para la estimación y el análisis predictivo de la cobertura Wi-Fi en entornos interiores. A lo largo de los capítulos anteriores se ha descrito tanto la construcción del sistema completo como el diseño, el entrenamiento y la evaluación del modelo de aprendizaje profundo que constituye su núcleo. En este último capítulo se recogen las conclusiones extraídas del trabajo realizado y se proponen las líneas de mejora y ampliación que podrían abordarse en el futuro.

7.1 CONCLUSIONES

El objetivo general del proyecto, ofrecer al gran público una herramienta, que utilizando tecnologías innovadoras, sea capaz de estimar la cobertura WiFi de un entorno interior a partir únicamente de su plano y de la posición de los puntos de acceso, se ha alcanzado de forma satisfactoria. El resultado es una aplicación funcional y desplegada en la nube, articulada sobre una arquitectura de microservicios en la que cada componente cumple una responsabilidad delimitada: una aplicación de escritorio con la que el usuario describe su entorno, un *backend* que centraliza la seguridad y la persistencia, y un motor analítico que ejecuta el modelo y devuelve el mapa de cobertura. Cada uno de los objetivos específicos planteados en el Capítulo 4, desde permitir que cualquier usuario describa su entorno de forma intuitiva hasta desplegar y empaquetar el sistema completo, ha quedado cubierto.

La decisión de diseño más determinante ha sido la adopción de una arquitectura de microservicios. Esta elección ha permitido que cada módulo se desarrolle en la tecnología más adecuada para su cometido, el ecosistema de Python y PyTorch para el modelo, Java y Spring Boot para la lógica de negocio y JavaScript con React para la interfaz, y que cada uno se despliegue y mantenga de forma independiente. Además de favorecer la metodología ágil seguida durante el desarrollo. Por otra parte, la arquitectura del modelo de aprendizaje profundo escogida para el presente trabajo ha sido la adecuada. Motivado por el estado del arte de este ámbito y acorde a los resultados se respalda dicha afirmación.

Desde el punto de vista del modelo, los resultados confirman que la red U-Net reproduce con fidelidad el comportamiento del simulador a partir de una entrada mínima. Sobre el conjunto de prueba, el error medio se sitúa por debajo del decibelio (MAE ≈ 0,26 dB y RMSE ≈ 0,70 dB sobre un rango dinámico de unos 121 dB) y la predicción es prácticamente instantánea, en torno a 9 ms por mapa sobre la GPU empleada, frente al

coste muy superior de una simulación física por ray tracing. El error obtenido, del orden de la fracción de decibelio sobre un rango de más de 120 dB, indica que la red reproduce el simulador con una fidelidad muy alta y confirma la idoneidad de la arquitectura elegida para esta tarea.

En conjunto, y dentro de su marco (reproducción rápida de mapas simulados), el modelo cumple el objetivo planteado: ofrecer una estimación visual e inmediata de la cobertura WiFi en interiores a partir de una entrada mínima, con una precisión más que suficiente para orientar al usuario en la colocación de sus puntos de acceso.

Conviene, no obstante, matizar el alcance de estas cifras. El modelo se ha entrenado con un conjunto de datos simulado en la banda de 5 GHz siguiendo el modelo de canal IEEE 802.11ax y generado sin considerar interferencias entre canales ni redes vecinas. Por ello, la red aprende a reproducir el comportamiento del simulador y sus predicciones deben interpretarse dentro de ese marco, como una estimación rápida y orientativa. No como una medida exacta de la cobertura real si no como información de cómo se distribuye una señal WiFi con la configuración del entorno interior que se escoja.

Más allá de las cifras, el principal logro del trabajo es haber integrado un modelo de aprendizaje profundo dentro de un sistema software completo, funcional y accesible. Abordando el ciclo completo de un sistema software real, desde el análisis y el diseño hasta el despliegue en la nube y el empaquetado para su distribución.

Se ha conseguido acercar a un usuario sin conocimientos técnicos una capacidad tradicionalmente reservada a herramientas profesionales y costosas, ofreciéndole una estimación visual e inmediata que le ayuda a tomar decisiones informadas sobre la ubicación de sus puntos de acceso y a hacer un uso más responsable del espectro.

7.2 TRABAJOS FUTUROS

A partir de lo desarrollado se identifican varias líneas de continuación, ordenadas según su relevancia:

- Validación frente a medidas reales. La línea más importante es contrastar las predicciones del modelo con mediciones tomadas en entornos físicos reales, de forma análoga a como se procede en el estado del arte. Ello permitiría cuantificar la desviación respecto a la realidad y, en su caso, recalibrar o reentrenar el modelo

- Optimizador automático de la ubicación de puntos de acceso. El sistema actual predice la cobertura de una configuración dada, el siguiente paso natural es aprovechar la rapidez de la inferencia para que, a partir de un plano, se busque automáticamente la posición o las posiciones de los puntos de acceso que maximizan la cobertura.

- Evaluar las interferencias que existen en tu entorno actual y ajustar con *finetunning* la U-Net para lograr mejor precisión. De esta forma podría tener en cuenta si en una habitación en un momento dado existen muchas interferencias. Aunque esa conclusión se puede sacar en el *test* de red empleado, se podría llegar a implementar de forma predictiva.

- Ampliación del rango de configuraciones y bandas. Extender el modelo a un mayor número de puntos de acceso (cuatro o cinco) y a otras bandas de frecuencia, como la de 2,4 GHz, e incluso a escenarios de varias plantas, ampliaría su aplicabilidad.

- Distribución multiplataforma y sostenibilidad económica. Por último, generar versiones para macOS y Linux, o incluso una versión móvil, ampliaría el público potencial de la aplicación. De cara a una explotación real, podrían plantearse modelos de sostenibilidad como una versión gratuita con funciones de pago

Capítulo 8. BIBLIOGRAFÍA

[1] F. F. Álvarez-Rementería y G. Á. de la Cuesta Padilla, «Libro blanco de buenas prácticas para el despliegue de redes inalámbricas de banda ancha en municipios de andalucía,» 2007. [En línea]. Available: https://www.juntadeandalucia.es/export/drupaljda/1337161061despliege_de_redes_inalambricas.pdf.

[2] «Writing Markup with JSX,» React, [En línea]. Available: https://react.dev/learn/writing-markup-with-jsx.

[3] Node.js, «https://nodejs.org/en,» [En línea].

[4] Node.js, «An introduction to the npm package manager,» [En línea]. Available: https://nodejs.org/learn/getting-started/an-introduction-to-the-npm-package-manager#:~:text=npm%20package%20manager-,Introduction%20to%20npm,everything..

[5] «Spring,» VMware Tanzu, [En línea]. Available: https://spring.io/projects/spring-boot#overview.

[6] «Apache Maven Project,» Fundación de Software Apache, [En línea]. Available: http://maven.apache.org/.

[7] D. R. Fielding, «¿Qué es la API REST?,» IBM, [En línea]. Available: https://www.ibm.com/es-es/think/topics/rest-apis.

[8] «1. ¿Qué es PostgreSQL ?,» PostgreSQL, [En línea]. Available: https://www.postgresql.org/docs/current/intro-whatis.html.

[9] «¿Qué son las redes neuronales?,» IBM, [En línea]. Available: https://www.ibm.com/es-es/think/topics/neural-networks#741977106.

[10] «¿Cómo funcionan las redes neuronales convolucionales?,» IBM, [En línea]. Available: https://www.ibm.com/es-es/think/topics/convolutional-neural-networks.

[11] PyTorch, «PyTorch documentation,» [En línea]. Available: https://docs.pytorch.org/docs/2.12/index.html.

[12] M. Labonne, «¿Qué es un tensor en el aprendizaje automático?,» 29 Marzo 2022. [En línea]. Available: https://towardsdatascience.com/what-is-a-tensor-in-deep-learning-6dedd95d6507/.

[13] P. O. González, «Tema 4 Ondas Esféricas,» Universidad Pontificia Comillas.

[14] U. I. d. T. (ITU), «Recomendación UIT-R P.1238-9,» 06 2017. [En línea]. Available: https://www.itu.int/dms_pubrec/itu-r/rec/p/R-REC-P.1238-9-201706-I!!PDF-S.pdf.

[15] «Nvidia Developer,» NVIDIA Corporation, 2026. [En línea]. Available: https://developer.nvidia.com/discover/ray-tracing.

[16] Z. Chen, A. Delis y H. L. Bertoni, «Radio-wave propagation prediction using ray-tracing techniques on a network of workstations (NOW),» *Journal of Parallel and Distributed Computing,* vol. 64, pp. 1127-1156, 2004.

[17] O. F. P. B. T. Ronneberger, «U-Net: Convolutional Networks for Biomedical Image Segmentation,» *Springer International Publishing,* pp. 234--241, 2015.

[18] H. S. y. T. M. C. Pyo, «A Deep Learning-Based Indoor Radio Estimation Method Driven by 2.4 GHz Ray-Tracing Data,» *IEEE Access,* 2023.

[19] J. B. y. N. S. M. Jones, «JSON Web Token (JWT),» Mayo 2015. [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc7519.

[20] M. P. Vizcaíno, «Guía de Implementación: JWT para la Autenticación en Java,» 15 Feb 2024. [En línea]. Available: https://medium.com/somos-pragma/gu%C3%ADa-de-implementaci%C3%B3n-jwt-para-la-autenticaci%C3%B3n-en-java-db47b04eda54.

[21] F.-G. A. J. a. V.-M. C. A, «Fast Indoor Radio Propagation Prediction Using Deep-Learning,» Zenodo, 28 May 2023. [En línea].

[22] R. N. Keiron O'Shea, «An Introduction to Convolutional Neural Networks,» 2015.

[23] Y. L. L. B. Y. B. a. P. Haner, «GradientBased Learning Applied to Document,» 1989.

[24] *But What is a Convolution?*. [Película]. 3Blue1Brown, 2026.

[25] S. R. R. R. Rath, «Implementing UNet from Scratch Using PyTorch,» 3 April 2023. [En línea]. Available: https://debuggercafe.com/unet-from-scratch-using-pytorch/.

[26] D. P. K. y. J. Ba, «Adam: A Method for Stochastic Optimization,» ICLR, 2015.

ANEXO I: ALINEACIÓN DEL PROYECTO CON LOS ODS

En relación con los Objetivos de Desarrollo Sostenible, este proyecto se alinea principalmente con el ODS 9, Industria, innovación e infraestructura, ya que busca aplicar soluciones tecnológicas actuales para mejorar el aprovechamiento de las infraestructuras de comunicación ya existentes. Además, también guarda relación con el ODS 12, Producción y consumo responsables, al fomentar un uso más eficiente de los recursos disponibles y evitar decisiones poco justificadas, como la instalación innecesaria de nuevos dispositivos o el aumento de potencia sin un criterio técnico previo. De esta manera, el trabajo no solo tiene una finalidad técnica, sino que también contribuye a un uso más consciente, eficiente y sostenible de la tecnología en entornos cotidianos.

Figura 48 Objetivos de desarrollo sostenible

ANEXO II MANUALES

Manual de Instalación

En primer lugar accede a este *link* y sigue los pasos de las imágenes:

https://drive.google.com/file/d/1hwm3GKV1pZI2Wu8w_nQEYZD04YS7g9TG/view

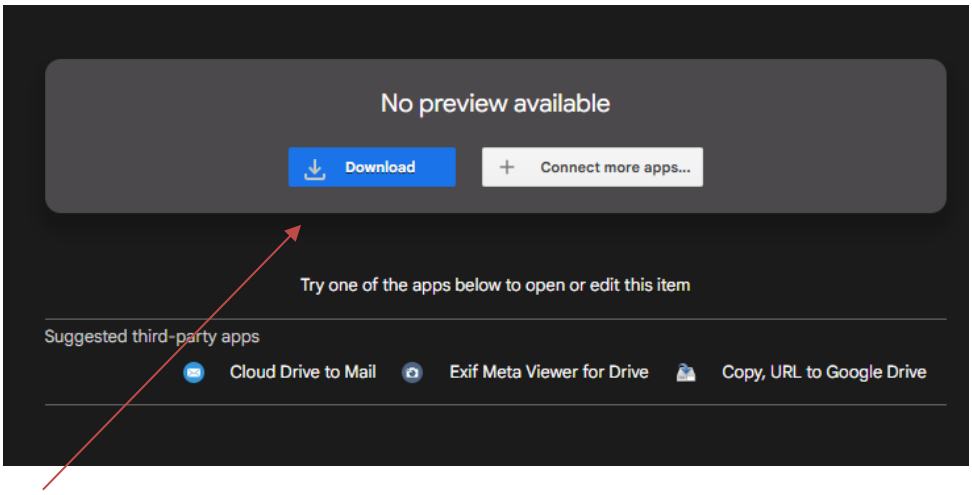

Figura 49 Página de Descarga

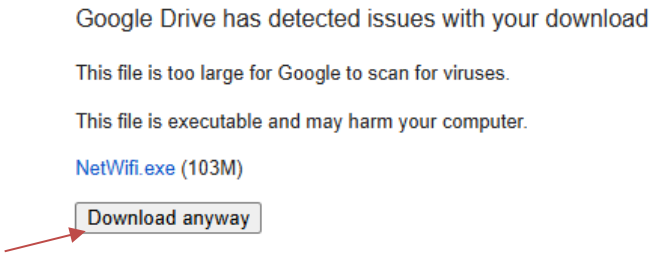

Figura 50 Descarga

Una vez descargado, aparece arriba a la derecha en el buscador y hace doble *click* en la aplicación. En caso de que no funcione, la encontrará en descargas de su ordenador y hará ahí doble *click*.

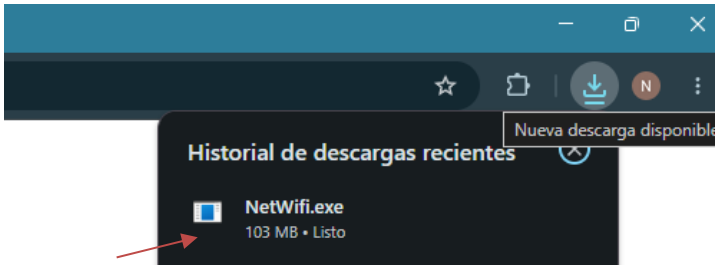

Figura 51 Aplicación Instalada

Una vez instalada, puede encontrarla buscando NetWifi en el buscador de Microsoft. Si pulsa el botón de "Abrir", puede disfrutar de la aplicación.

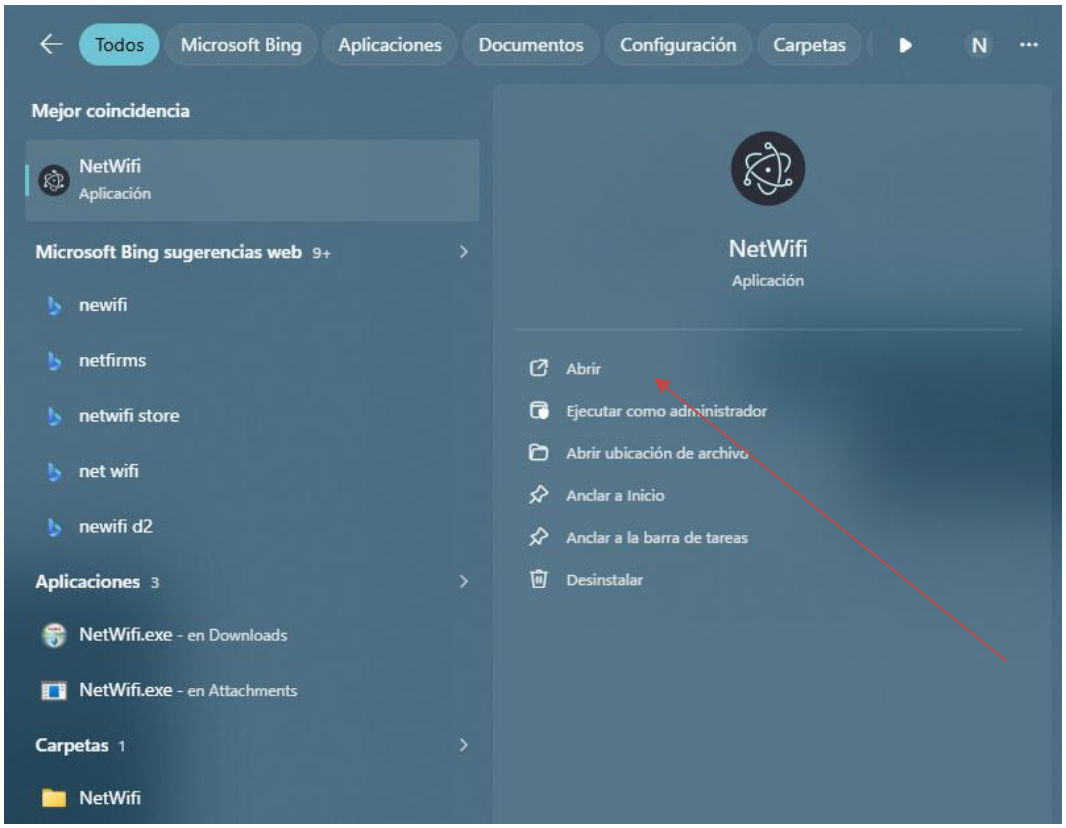

Figura 52 Buscador Aplicación Instalada

Manual de Usuario

La primera toma de contacto con la aplicación es la siguiente:

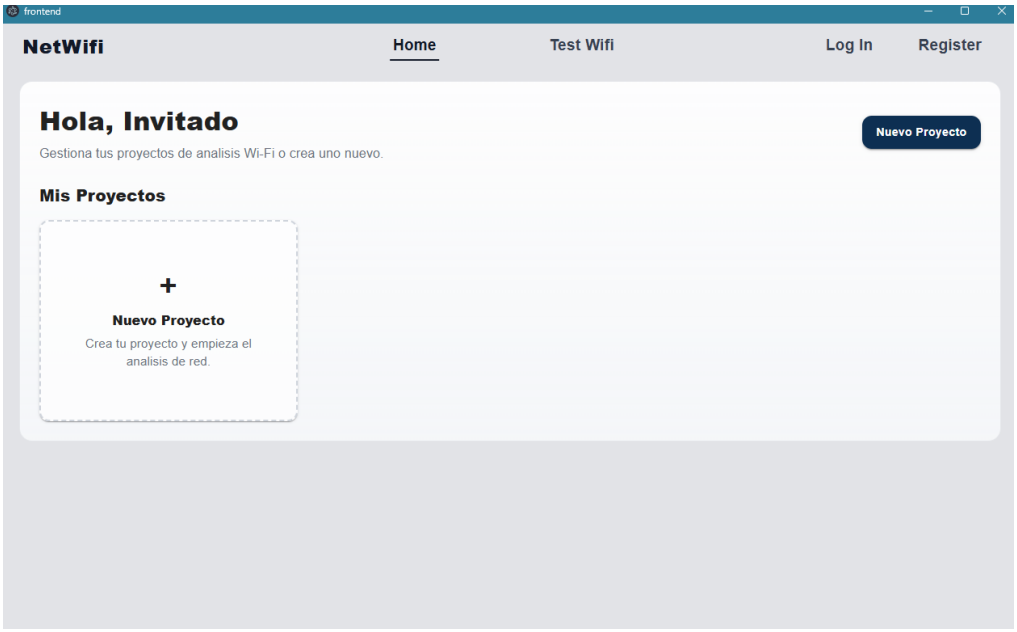

Figura 53 Página Principal

En ella puede iniciar un *Test* de Wifi:

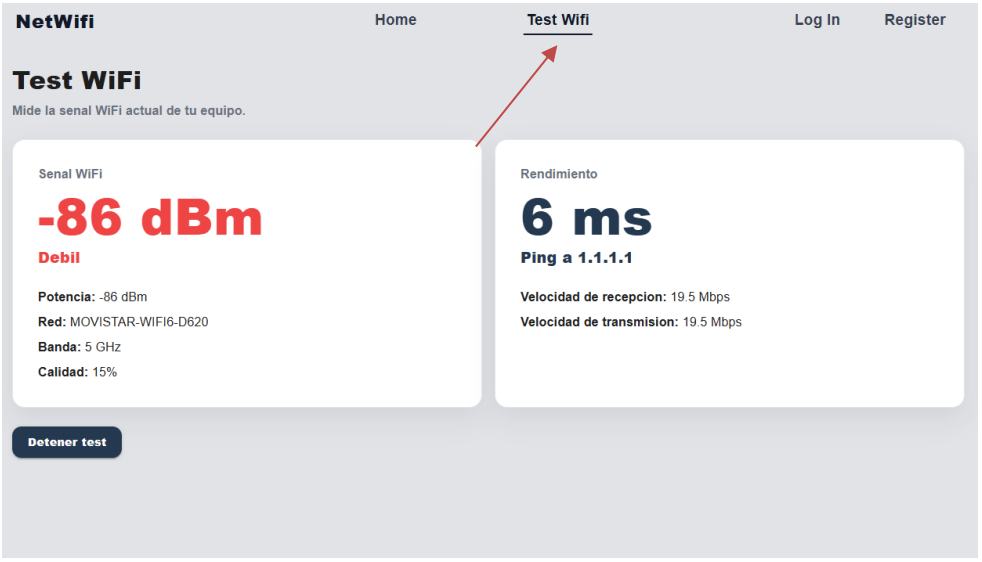

Figura 54 Test de Wifi

Para poder crear un proyecto deberá Registrarse (*Register*) o Iniciar Sesión (*Log In*):

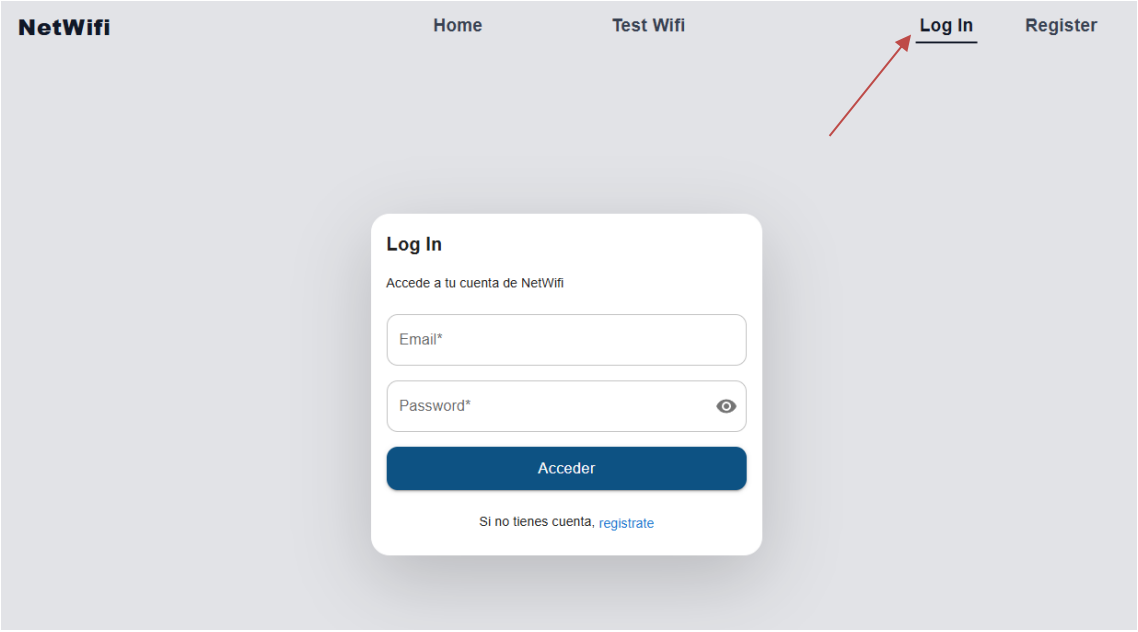

Figura 55 Página de Log In

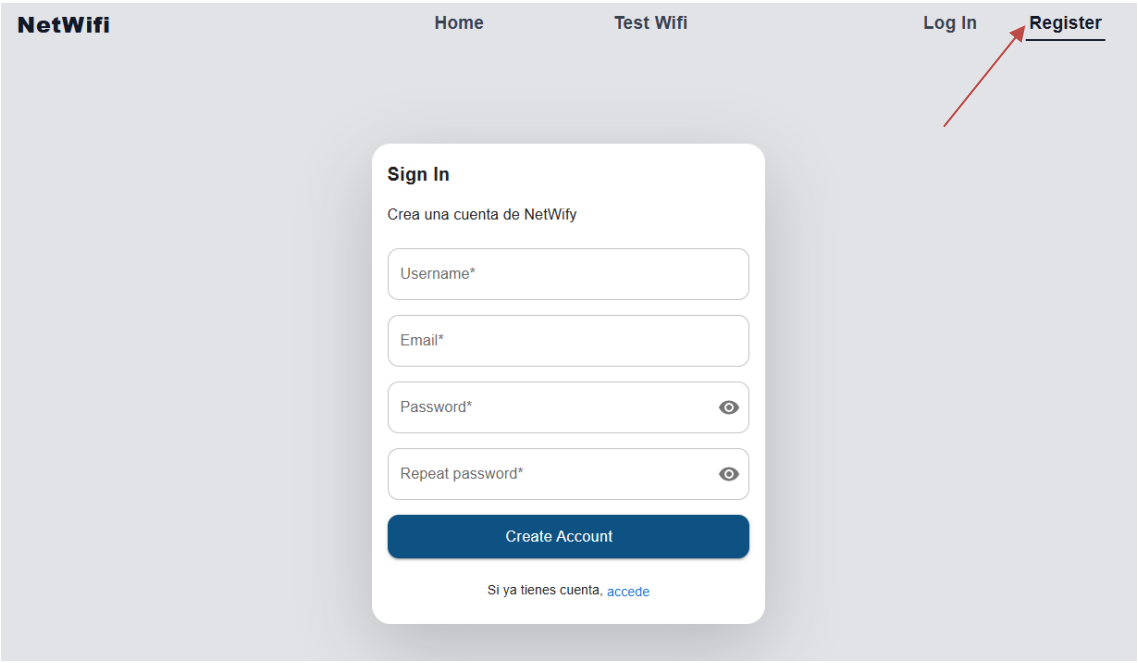

Figura 56 Página de Registro

Una vez accede puede observar el nombre de usuario arriba a la derecha y sus proyectos creados si hubiera alguno. Para crear un proyecto debe pulsar "Nuevo Proyecto":

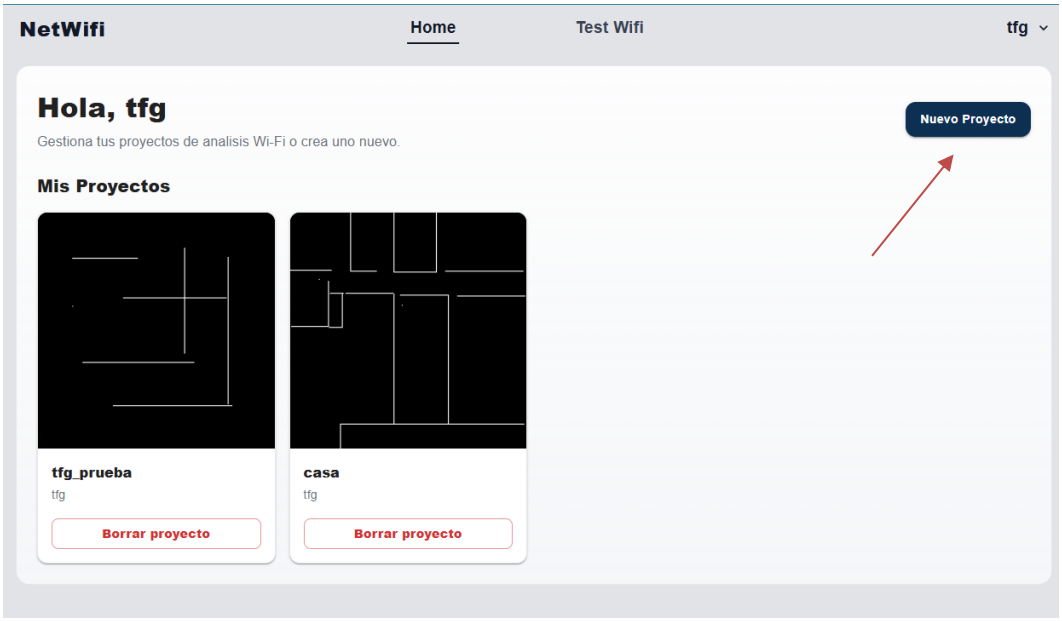

Figura 57 Crear Proyecto

Una vez pulse, aparece la pantalla del editor de proyecto, donde puede dibujar paredes rectas, borrar los elementos dibujados de forma completa, añadir *router's* , limpiar el proyecto entero o guardar:

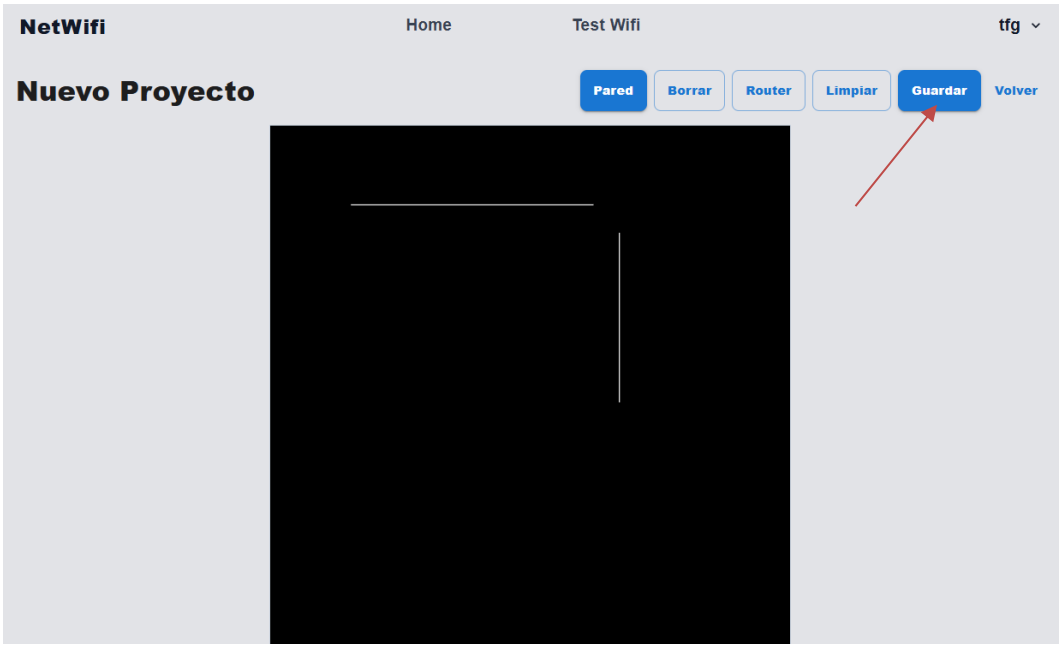

Figura 58 Editor de Proyectos

Si escoge el botón de "Guardar", deberá introducir un nombre:

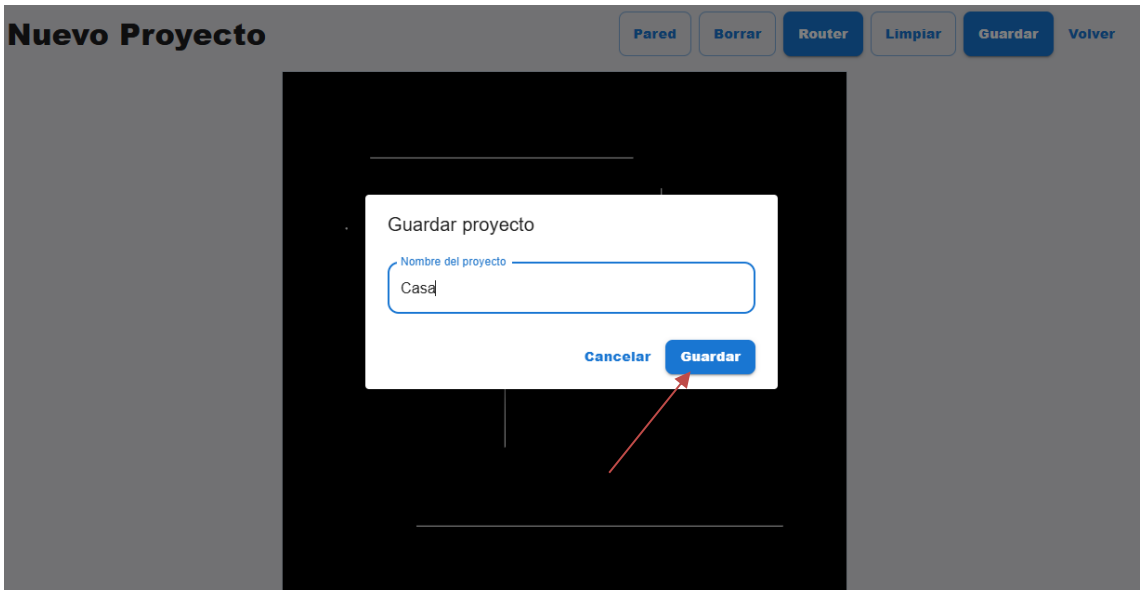

Figura 59 Guardar Proyecto

Una vez guardado, puede acceder a él en la página principal:

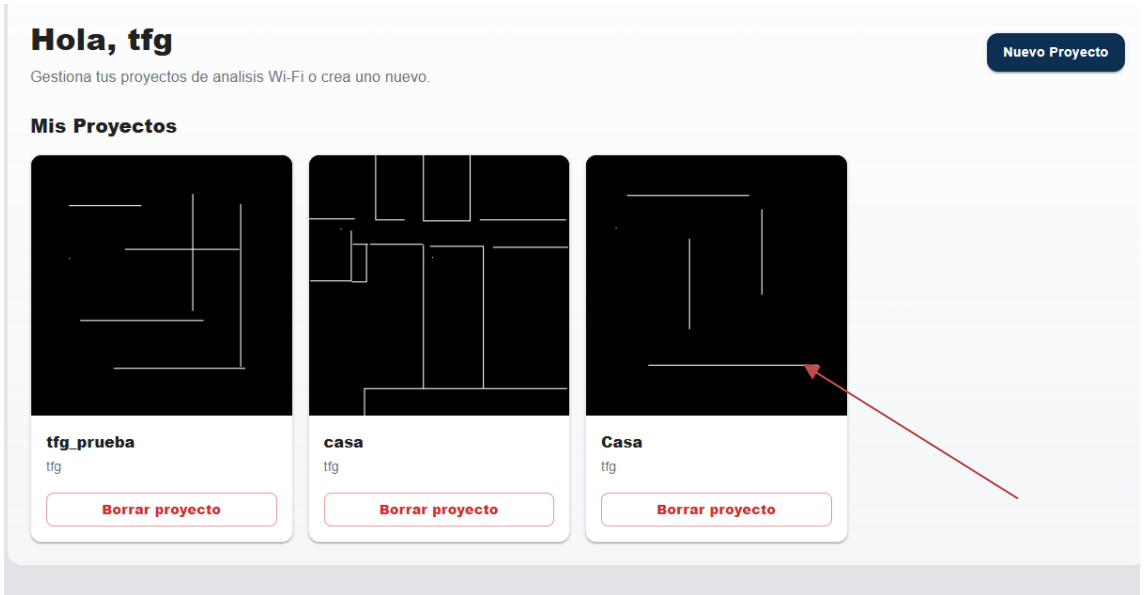

Figura 60 Abrir Proyecto Guardado

Cuando abre un proyecto guardado puede editarlo, sin embargo, el borrador es una "goma" digital. Su uso es para "abrir" puertas o ventanales interiores.

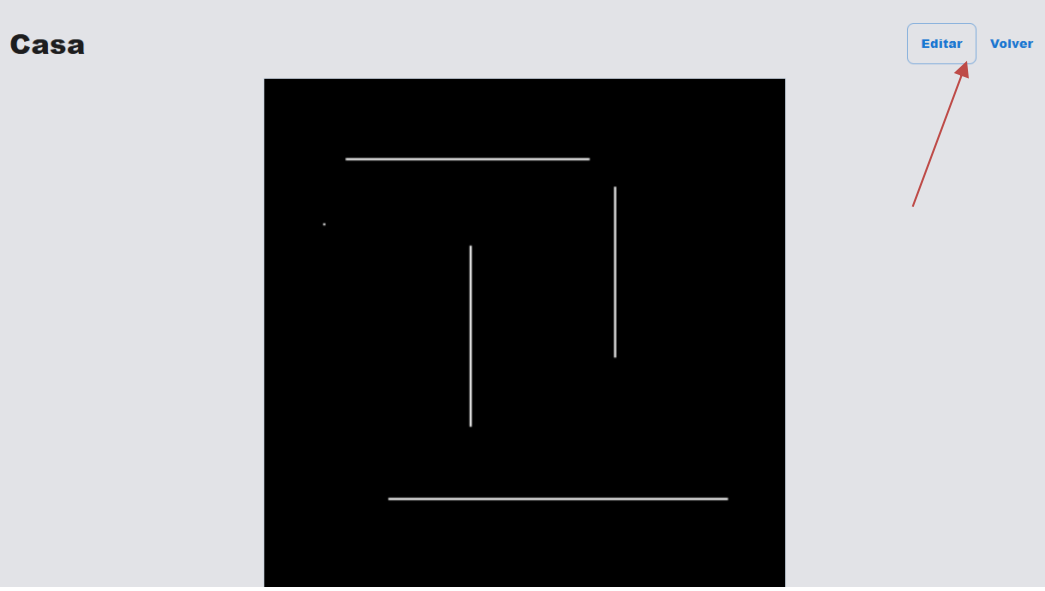

Figura 61 Editar Proyecto

Una vez termine de guardarlo, si desliza hacia abajo aparece un botón de "Simulación", con el que podrá utilizar el modelo de aprendizaje profundo para estimar la distribución de la señal WiFi en su mapa:

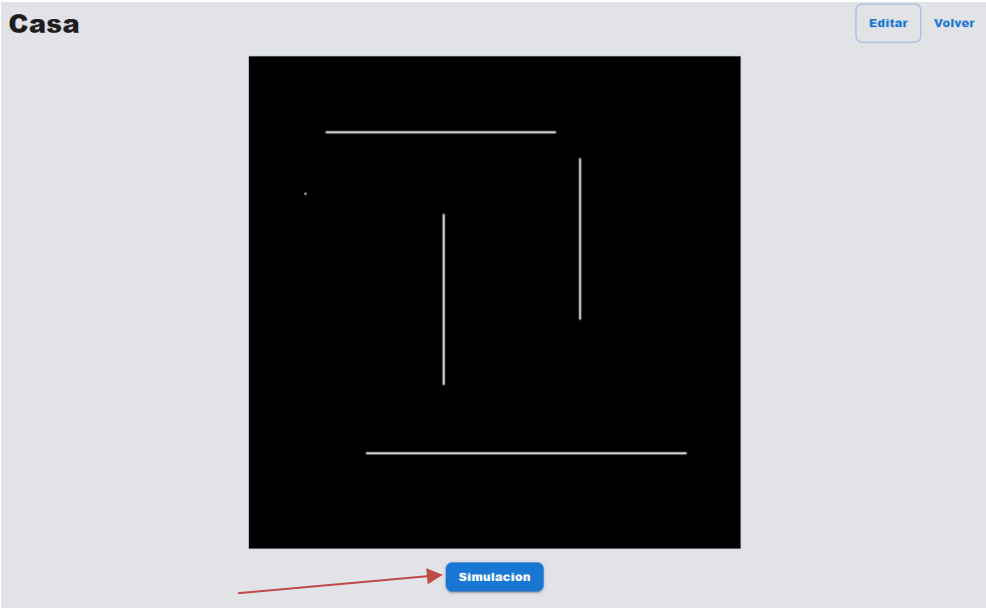

Figura 62 Simular Proyecto

Una vez simulado puede volver a la página principal:

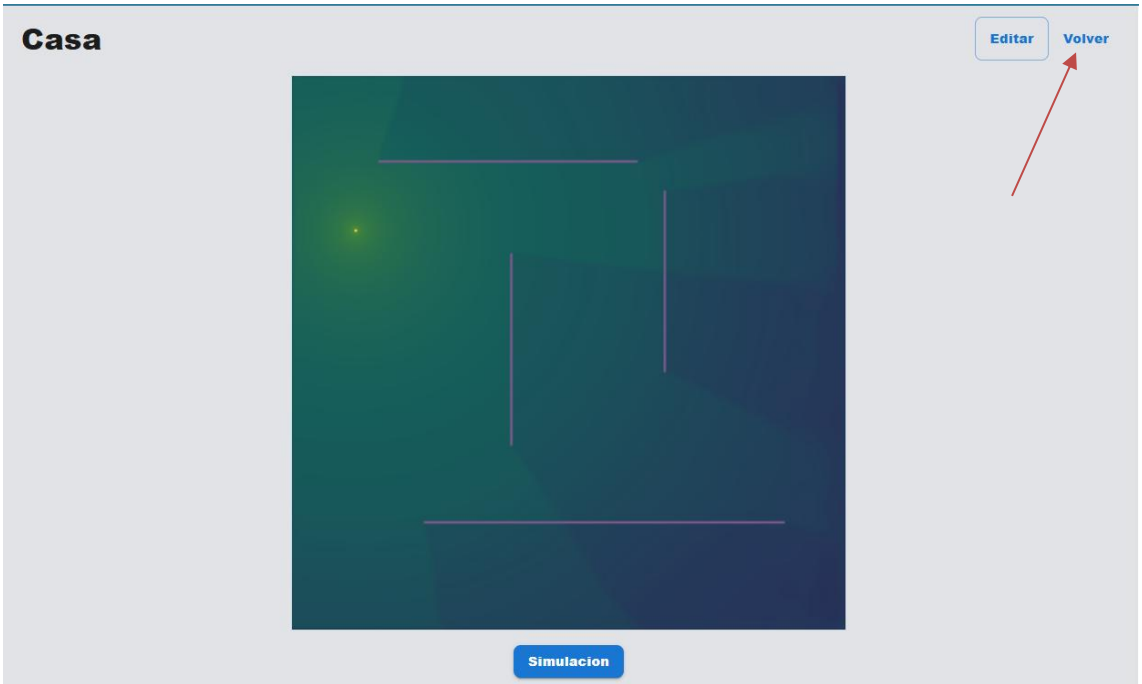

Figura 63 Simulación Hecha

Por último, puede borrar los proyectos, cerrar sesión o borrar la cuenta creada:

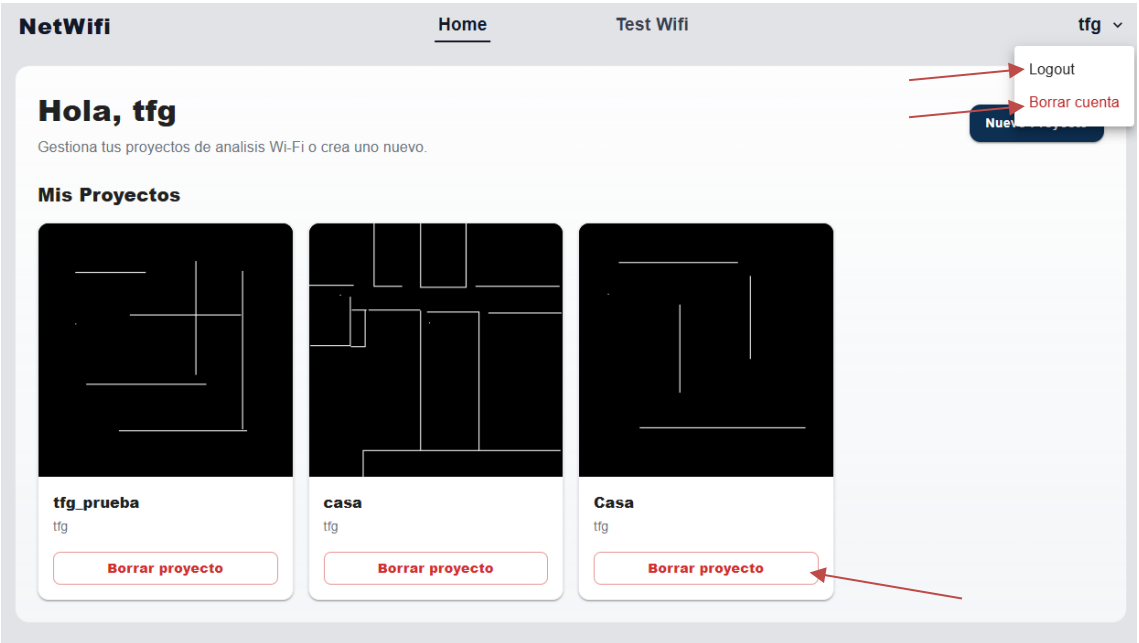

Figura 64 Acciones de Borrar