UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

GRADUADO EN INGENIERÍA DEL SOFTWARE

PLATAFORMA WEB PARA LA DETECCIÓN Y DENUNCIA DE DISCURSOS DE ODIO EN LAS REDES SOCIALES EN EL CONTEXTO DEL FÚTBOL

WEB PLATFORM FOR THE DETECTION AND REPORTING OF HATE SPEECH ON SOCIAL MEDIA IN THE CONTEXT OF FOOTBALL

Realizado por
PABLO SENCIALES DE LA HIGUERA

Tutorizado por
EDUARDO GUZMÁN DE LOS RISCOS

Departamento
LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

MÁLAGA, JUNIO DE 2025

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA DEL SOFTWARE

PLATAFORMA WEB PARA LA DETECCIÓN Y DENUNCIA DE DISCURSOS DE ODIO EN LAS REDES SOCIALES EN EL CONTEXTO DEL FÚTBOL

WEB PLATFORM FOR THE DETECTION AND REPORTING OF HATE SPEECH ON SOCIAL MEDIA IN THE CONTEXT OF FOOTBALL

Realizado por
Pablo Senciales de la Higuera

Tutorizado por
EDUARDO GUZMAN DE LOS RISCOS

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, JUNIO DE 2025

Fecha defensa: JULIO de 2025

Resumen

Este Trabajo de Fin de Grado busca proporcionar una solución a una problemática social compleja derivada del uso y extensión de las redes sociales (RRSS), un problema especialmente en contextos o entornos emocionales y apasionados como el fútbol donde el discurso de odio y las conductas inapropiadas son recurrentes. Esto no solo afecta negativamente en los medios digitales, sino que también tiene un gran impacto en la convivencia y bienestar social.

El objetivo principal es, por tanto, crear una plataforma web donde de manera centralizada y simplificada se pueda denunciar, verificar y seguir los reportes creados relacionados con el discurso de odio. Apuntando a crear un entorno más saludable y seguro para todos los usuarios.

Es por ello por lo que se crea Fairplay360, una plataforma web usable y atractiva que permite crear reportes de manera sencilla y rápida a partir de diferentes medios como un texto, una imagen con texto o un enlace de una publicación de X (anteriormente Twitter), tras esto el seguimiento se llevará mediante notificaciones al correo electrónico. Cada reporte, además puede ser cumplimentado con el contexto que quiera aportar el usuario, mejorando así la precisión en su análisis. Las denuncias son procesadas por una Inteligencia Artificial (IA) avanzada, que dictaminará si el contenido proporcionado se considera discurso de odio o no. Finalmente, los administradores tendrán la labor de aceptar o rechazar estas denuncias.

En resumen, esta plataforma agiliza el sistema actual de denuncia vía correo electrónico cumpliendo el objetivo principal propuesto, además hace uso de una IA que aporta un análisis al contenido, siendo, sin embargo, necesario una gestión humana debido a las limitaciones de esta.

Palabras clave:

Discurso de Odio, NextJS, Flask, Deepseek, Google Cloud Platform.

Abstract

This Final Degree Project seeks to provide a solution to a complex social problem arising from the use and spread of social media (SM), a problem especially in emotional and enthusiastic contexts or environments such as football, where hate speech and inappropriate behavior are common. This not only negatively impacts digital media but also has a significant impact on social coexistence and well-being.

The main objective, therefore, is to create a web platform where reports related to hate speech can be created, verified, and tracked in a centralized and simplified manner. The aim is to create a healthier and safer environment for all users.

This is why Fairplay360 was created, a usable and engaging web platform that allows reports to be created quickly and easily from different media such as text, an image with text, or a link to a post on X (formerly Twitter). After this, reports will be followed up via email notifications. Each report can also be supplemented with the context the user is willing to provide, thus improving the accuracy of its analysis. Reports are processed by advanced Artificial Intelligence (AI), which will determine whether the content provided is considered hate speech or not. Finally, administrators will be responsible for accepting or rejecting these reports.

To sum up, this platform streamlines the current email reporting system, fulfilling its primary objective. It also uses AI to analyze the content, although human management is required due to its limitations.

Keywords:

Hate Speech, NextJS, Flask, Deepseek, Google Cloud Platform.

Índice

1

Introducción

1.1 Motivación

Impacto del discurso de odio en la sociedad

La tendencia actual en redes sociales como X, tras la compra de Elon Musk, quien promueve lo que considera libertad de expresión, ha desencadenado en un ambiente de crispación generalizada debido a la falta de moderación. Esta situación se vive sobre todo en atmósferas donde se crean “equipos” como pueden ser la política o el deporte. Dicha dinámica no solo se limita al entorno digital, sino que por lo general trasciende al mundo real, generando conflictos interpersonales, incrementando la polarización social e incluso provocando incidentes de discriminación, violencia verbal y física.

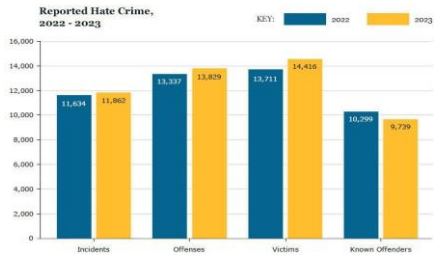


Figura 1. Estadísticas de crímenes de odio en EE. UU. entre 2022 y 2023. Fuente: *U.S. Department of Justice* (2024). Recuperado de https://www.justice.gov/hatecrimes/hate-crime-statistics

Caso específico del fútbol

Recientemente La Real Federación Española de Fútbol (RFEF) ha implementado un protocolo especifico contra el racismo y discriminación que puede suponer desde una suspensión temporal hasta la suspensión definitiva del partido en caso de detectar la proliferación de insultos desde la grada.

Entre los casos más mediáticos en nuestro país encontramos ejemplos como el del jugador del Real Madrid, Vinícius Júnior u otros incidentes con el jugador del Fútbol Club Barcelona, Lamine Yamal (ambos han sufrido insultos racistas desde las gradas). Estas situaciones ponen en manifiesto el problema y la necesidad de encontrar soluciones, no solo por la mala imagen que da al fútbol español, sino por el daño producido a los futbolistas.

Aunque estos protocolos han supuesto un gran avance, solo son aplicables en el ámbito presencial y aún existe una gran brecha con la gestión del discurso de odio en las redes sociales donde estos comportamientos se ven amplificados y sin un control adecuado. Precisamente esta es la problemática que aborda este Trabajo de Fin de Grado.

Ineficiencia de las soluciones actuales

La solución actual implementada por LaLiga se basa en una dirección de correo electrónico habilitada para que los usuarios puedan facilitar contenido que consideren denunciable. Sin embargo, este proceso resulta poco transparente, carece de mecanismos de seguimiento y no ofrece retroalimentación al usuario, lo que puede desalentar la participación generando desconfianza.

Responsabilización de las plataformas

Como se ha mencionado, plataformas como X dan cabida a estos comportamientos con apenas restricciones. Se debe poner el foco sobre estas grandes corporaciones para que se responsabilicen y se centren en crear espacios seguros. Uno de los objetivos de este TFG es intentar cumplir este papel con la combinación de tecnologías avanzadas y supervisión humana.

1.2 Objetivos

El objetivo principal de este TFG es proporcionar una solución integral a la problemática derivada de la toxicidad y discurso de odio presente en las redes sociales que denominaremos Fairplay360. Por lo tanto, se pretende crear una aplicación web siguiendo los principios y buenas prácticas de la ingeniería de software, cubriendo todas las fases del ciclo de vida del desarrollo. Además, se integrará un sistema de análisis con inteligencia artificial para clasificar automáticamente el contenido, facilitando la detección de discurso de odio y mejorando el proceso de verificación, agilizando integralmente todo el procedimiento de denuncia.

Se tienen también los siguientes objetivos:

- Desarrollar una web intuitiva y usable, tanto para usuarios como para administradores, aplicando principios de diseño centrados en la experiencia de usuario (UX).

- Implementar las siguientes funcionalidades a alto nivel:
    1. El usuario puede crear un reporte a partir de texto, imagen con texto o un enlace a una publicación de X.
    2. Los reportes serán analizados automáticamente por una inteligencia artificial que los clasificará en función de su contenido.
    3. El sistema generará el análisis en formato PDF que se enviará por correo electrónico.
    4. Los administradores podrán gestionar los reportes, aceptándolos o rechazándolos.
    5. Los administradores podrán gestionar el acceso a los usuarios, pudiendo prohibirlo o restaurarlo.
    6. El sistema registrará la actividad de los administradores para garantizar el correcto uso y transparencia.

- Implementar un sistema de autenticación de usuario con tokens, que proteja el uso de la aplicación.

1.3 Antecedentes

Recientemente han estado surgiendo distintas iniciativas y plataformas cuyo objetivo es combatir el discurso de odio en las redes sociales, sin embargo, la mayoría de estas soluciones carecen del sistema integral de reporte que justifica la necesidad de las propuestas incluidas en Fairplay360.

Soluciones en las redes sociales

- **X (Twitter)**: X tiene integrado un sistema de denuncia manual en las publicaciones que contengan contenido ofensivo, no obstante, el proceso resulta lento, poco transparente y rara vez da retroalimentación al usuario.

- **Meta (Instagram, Facebook)**: Meta por su parte sí incluye un sistema de detección basado en IA con su modelo propio RoBERTa, basado en el modelo BERT de Google. Este modelo tiende a presentar errores, tanto falsos positivos como falsos negativos, debido, en parte, a sus limitaciones como la dificultad para detectar sarcasmo o ironía, además de la complejidad que supone entender muchas veces el contexto.

Otras plataformas

- **Bodyguard.ai**: Bodyguard ofrece su API a terceros, haciendo uso de su sistema basado en inteligencia artificial es capaz de analizar en tiempo real mensajes pudiendo detectar y clasificar comentarios en categorías como homofobia, racismo, sexismo y un amplio abanico. Su principal objetivo es proteger perfiles públicos o marcas que hagan uso de sus servicios, en lugar de a usuarios generales.

- **Hatebase**: Es una gran base de datos colaborativa sobre discurso de odio en una gran variedad de idiomas, se enfoca en un análisis lingüístico y no en la gestión de los reportes.

1.4 Metodología de desarrollo

Para la gestión y planificación del desarrollo del sistema se ha seguido una metodología ágil, concretamente Scrum. Esta elección se debe a la necesidad de contar con un enfoque iterativo e incremental, que permitiera una evolución continua de las funcionalidades, así como una rápida adaptación a los cambios surgidos durante el desarrollo del proyecto.

Scrum permite dividir el trabajo en sprints (ciclos de desarrollo cortos y planificados), al final de los cuales se obtiene un entregable funcional. Cada sprint ha incluido las fases de diseño, implementación, pruebas y revisión, permitiendo evaluar el progreso y realizar mejoras progresivas.

Durante el desarrollo, se ha utilizado una herramienta Kanban como Trello para gestionar las tareas, definir los sprints y mantener el control del avance del proyecto. Las tareas fueron clasificadas por estados (pendiente, en progreso *y* completada), lo que permitió una organización clara del flujo de trabajo.

Gracias a esta metodología, se logró un desarrollo progresivo, flexible y enfocado en la calidad, adaptándose a cambios necesarios a medida que avanzaban las fases de diseño e implementación. Nótese que puede encontrar la estructura de desglose de trabajo (EDT) en el *Anexo C*.

1.5 Estructura de la memoria

En este documento se definen aspectos tan básicos como la motivación del proyecto hasta las conclusiones y una descripción detallada de todo el proceso de desarrollo. Se pueden detallar los puntos definidos en el índice:

1. Introducción

1.1. Motivación

Se explica de dónde surge la idea y qué motivos impulsaron el desarrollo del proyecto, destacando el impacto que tiene el discurso de odio actualmente en la sociedad.

1.2. Objetivos

Se desarrollan los objetivos, tanto el principal como los secundarios, que buscamos con este proyecto, definiendo como objetivo principal la creación de una plataforma que ofrezca una solución integral a la problemática del discurso de odio.

1.3. Antecedentes

Se identifican proyectos y herramientas de carácter similar, además observamos sus carencias y lo que nuestra solución puede ofrecer a cambio.

1.4. Metodología de desarrollo

Se describe la metodología de trabajo empleada durante el desarrollo del proyecto, detallando cómo se organizó de forma iterativa y adaptativa en fases.

2. Tecnologías y herramientas utilizadas

Se describen detalladamente cada herramienta y tecnología, además de los lenguajes de programación y frameworks que se han utilizado durante el desarrollo del proyecto, justificando en cada caso la elección en función de los requisitos y necesidades.

3. Especificación y análisis

Se aborda la fase de análisis de requisitos y la especificación funcional del sistema, describimos los requisitos funcionales y se definen las principales características de la aplicación.

4. Diseño del sistema

Se presentan las decisiones de diseño adoptadas, incluyendo la arquitectura del sistema, diagrama de clases y de base de datos, así como las estrategias seguidas para garantizar la escalabilidad y mantenibilidad del sistema.

5. Implementación y Pruebas

Se explica la fase de desarrollo, centrándose en cómo se han implementado las principales funcionalidades, además de las pruebas llevadas a cabo para garantizar la fiabilidad del sistema.

6. Conclusiones y Trabajos Futuros

Se exponen las conclusiones del proyecto y se destacan los objetivos alcanzados, también las dificultades encontradas durante el desarrollo y las posibles mejoras y futuros trabajos que se podrían llevar a cabo.

2

Tecnologías y herramientas utilizadas

2.1 NextJS

NextJS es un framework o marco de trabajo de React basado en JavaScript. Actualmente es uno de los frameworks de desarrollo web más populares destacando su capacidad de ofrecer soluciones completas integrando en un mismo proyecto el front-end y el back-end, es decir, aplicaciones full-stack. Entre las características más destacadas de Next.js se incluyen:

- **Renderizado híbrido**: Next.js permite elegir distintos métodos de renderizado, adaptándose a las necesidades de cada componente o página de la aplicación. Ofrece renderizado del lado del servidor (*Server-Side Rendering*, SSR), es decir, el código de React se procesa en el servidor que devuelve código HTML al cliente, esto puede mejorar los tiempos de carga inicial y disminuir la carga de trabajo del navegador. A su vez, cuenta con renderizado del lado del cliente (*Client-Side Rendering,* CSR) lo que es la manera habitual en la que trabaja React, generando el contenido dinámicamente en el navegador. Finalmente, contamos con la generación de páginas estáticas (*Static Site Generation*, SSG) que se procesa durante el tiempo de compilación, haciendo que estas páginas puedan ser cargadas en el cliente sin necesidad de trabajo adicional.

- **Rutas automáticas**: Next.js cuenta con un sistema de enrutado basado en la estructura de carpetas del proyecto, lo que elimina la necesidad de configurar las rutas de forma manual como se hace clásicamente en React. Renderiza el archivo *page.js* en la ruta con el nombre de la carpeta, obsérvese que la carpeta raíz *app* toma como ruta '/'. Además, permite anidamiento y rutas dinámicas como vemos en el ejemplo de la *Figura 2*.

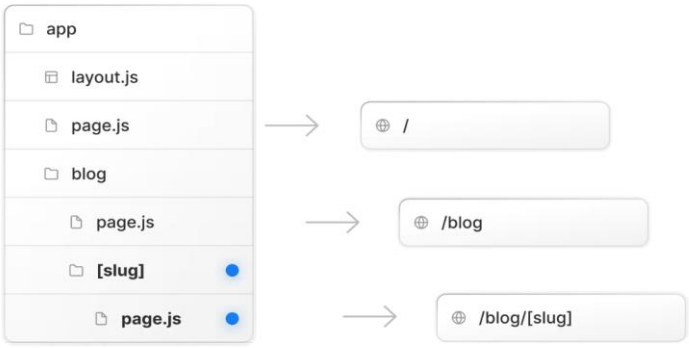


Figura 2. Ejemplo de sistema de enrutamiento en Next.js. Fuente: *Next.js Documentation* (2025). Recuperado de https://nextjs.org/docs/app/getting-started/layouts-and-pages

- **API Routes**: Permite definir funciones de back-end en el mismo proyecto haciendo uso de las API routes, con un enrutado similar al mencionado en el apartado anterior. Esto, facilita realizar funcionalidades y lógica del servidor,

como gestionar peticiones HTTP o acceso a base de datos, sin necesidad de tener un back-end independiente. Next.js cuenta, además, con sus propias librerías y utilidades para definir estas funciones de manera sencilla, siguiendo patrones comunes en frameworks de back-end.

- **Optimización automática**: Integra funcionalidades y componentes propios orientados a la optimización y mejora de la experiencia de usuario. Destacan la carga optimizada de imágenes con el componente *<Image>*, sustituto de la etiqueta HTML *<img>*, que hace uso de la carga perezosa (*lazy loading*) haciendo que la imagen cargue solo cuando va a ser visible en pantalla. De manera similar, tenemos el componente *<Link>*, sustituto de la etiqueta *<a>*, que incorpora la técnica de prefetching, cargando en segundo plano las rutas para favorecer una navegación más fluida y rápida para el usuario.

Es gracias a estas optimizaciones que Next.js ayuda a mejorar el rendimiento y reducir el consumo de recursos de nuestra aplicación.

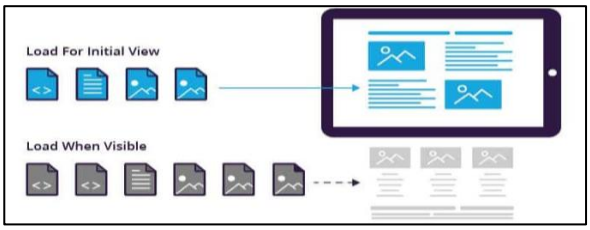


Figura 3. Ejemplo ilustrativo de lazy loading en el cliente. Fuente: *Imperva* (s.f.). Recuperado de https://www.imperva.com/learn/performance/lazy-loading/

- **Compatibilidad con TypeScript**: Aunque Next.js está basado en JavaScript, ofrece soporte nativo para TypeScript, lo que permite mejorar la robustez y mantenibilidad del código mediante tipado estático que ofrece. Además, se integra con ESLint, una herramienta que permite analizar y detectar malas prácticas y antipatrones en nuestro código, como, por ejemplo, variables sin utilizar, código redundante o una constante mal definida con la palabra clave *let* en lugar de *const*.

- **Facilidad de despliegue**: Está diseñado para integrarse de forma rápida y sencilla en servicios PaaS (Platform as a Service) como Vercel, los creadores de Next.js, u otros servicios en la nube como AWS, Google Cloud o Azure. Estas integraciones simplifican la gestión de recursos y configuración del entorno, mejorando la escalabilidad de las aplicaciones y facilitando la aplicación de buenas prácticas en el flujo de desarrollo.

Se puede concluir con que Next.js es una opción moderna, potente, escalable y eficiente, que permite desarrollar en un mismo proyecto tanto la lógica de presentación como funcionalidades del servidor lo que nos simplifica notablemente la arquitectura de la aplicación. En este proyecto se ha explotado esta funcionalidad de Next.js para implementar funcionalidades sin que interactúen directamente con la base de datos.

Otras tecnologías relacionadas:

- **NextAuth.js**: Es una biblioteca de autenticación para aplicaciones desarrolladas con Next.js, diseñada para gestionar de forma sencilla, segura y flexible el proceso de autenticación de usuarios. Permite autenticarse no solo haciendo uso de proveedores como GitHub o Google mediante OAuth, sino con credenciales propios definidos en nuestro servicio web, además permite utilizar JWT para gestionar la sesión. Destaca su fácil implementación y configuración, también la gran cantidad de personalización que ofrece.

- **Nodemailer**: Un paquete de Node.js que facilita el envío de correos electrónico desde nuestro servidor en Next.js con una configuración sencilla, además permite hacer uso de protocolos como IMAP, POP y SMTP. Permite enviar correos en formato de texto plano, con plantillas HTML y adjuntando archivos.

- **Pdf-lib**: Una biblioteca que permite crear, modificar y manipular archivos PDF directamente desde el navegador o en entornos de servidor basados en Node.js, sin necesidad de utilizar herramientas externas o dependencias nativas

- **i18n (Next Internationalization)**: Sistema de internacionalización nativo de Next.js que permite definir múltiples idiomas dentro de la aplicación de forma sencilla. Mediante el uso de archivos de traducción, rutas adaptadas y contexto lingüístico, se facilita que la interfaz pueda mostrarse en diferentes idiomas según la preferencia del usuario. En este proyecto se ha configurado el soporte para español e inglés, mejorando la accesibilidad y el alcance de la plataforma.

2.2 Flask

Flask es un framework de desarrollo web basado en Python, cuyo objetivo es el desarrollo de aplicaciones back-end y servicios de API RESTful, es decir, que están a la espera de recibir peticiones HTTP y procesarlas pudiendo definir los Endpoints. Destaca su ligereza y simplicidad, además que, al igual que Python, su curva de aprendizaje es muy reducida, es por ello por lo que resulta ideal para proyectos que requieran rapidez en el desarrollo y también flexibilidad.

La principal ventaja de Flask frente a otros frameworks como Django es que, a diferencia de este, no impone una estructura de proyecto ni incluye componentes innecesarios. Es por ello por lo que Flask con su enfoque minimalista permite seleccionar únicamente las librerías que se necesiten. Entre las características de Flask destacan:

- **Ligereza y alto rendimiento**: Al no incluir componentes adicionales por defecto, permite construir aplicaciones rápidas y eficientes.

- **Integración sencilla con bases de datos**: Compatible con múltiples motores de bases de datos, tanto relacionales (MySQL, PostgreSQL) como no relacionales (MongoDB, junto a MongoEngine).

- **Ideal para APIs REST**: Su simplicidad lo hace especialmente adecuado para desarrollar APIs ligeras, que pueden ser fácilmente consumidas por otros servicios.

En resumen, Flask ofrece un entorno ligero y adaptable ideal para desarrollar APIs, en este proyecto ha sido utilizado para crear un servicio que comunique directamente con la base de datos, haciendo uso de MongoEngine.

Otras tecnologías relacionadas:

- **MongoEngine**: Es un ODM (Object-Document Mapper) diseñado para Python, se enfoca en facilitar la interacción con MongoDB que ofrece un enfoque orientado a objetos. Permite trabajar con documentos de MongoDB a través de clases y objetos de Python, proporcionando una capa de abstracción que simplifica las transacciones con la base de datos.

- **BeautifulSoup**: Es la librería por excelencia para hacer web scraping en Python, es decir, extracción y análisis de datos de archivos HTML y XML. Permitiendo recorrer páginas web, filtrando el contenido por etiquetas de forma sencilla y eficiente.

- **Pandas**: Potente librería de Python especializada en la manipulación, análisis y procesamiento de datos estructurados. Está muy expandida en entornos de análisis de datos, y ciencia de datos debido a su facilidad para manejar grandes volúmenes de información de forma eficiente y con una sintaxis intuitiva.

2.3 MongoDB

MongoDB es una base de datos no relacional basada en documentos JSON (JavaScript Object Notation). Se ha convertido en una de las soluciones NoSQL más populares y potentes en la actualidad debido, en parte, a la flexibilidad que ofrece, su capacidad de escalabilidad y alto rendimiento en las operaciones de escritura y lectura.

Su modelo basado en documentos nos permite manejar estructuras de datos dinámicas y complejas pudiendo definir esquemas tolerantes al cambio, es por ello por lo que es una solución perfecta para proyectos donde los campos pueden ser variados y evolucionar en el tiempo. Entre sus principales características destacan:

- **Alta escalabilidad**: Destaca su escalabilidad horizontal ya que permite agregar más servidores o nodos para poder distribuir la carga de trabajo.

- **Facilidad de uso e integración**: MongoDB ha sido diseñado para ser sencillo de utilizar convirtiéndose en una elección ideal para un desarrollo rápido y evolutivo, además cuenta con drivers oficiales para una gran cantidad de lenguajes de programación, lo que facilita la integración en aplicaciones basadas en tecnologías como PHP, Python o Ruby, entre otras.

- **MongoDB Atlas**: Atlas es una solución en la nube que nos permite alojar de forma sencilla e incluso gratuita una base de datos MongoDB. Atlas gestiona automáticamente el rendimiento, copias de seguridad y funcionamiento del servidor, esto permite centrarse en el desarrollo.

- **Código abierto**: Cualquiera puede descargarse, utilizar y modificar el código en función de sus necesidades, esto fomenta una comunidad activa y a la mejora continua del proyecto.

Por todas estas razones, MongoDB resulta en una opción más que adecuada para este proyecto, permitiendo una gestión eficiente de los reportes, su posterior análisis y garantizando la escalabilidad y seguridad de los datos.

2.4 Google Cloud Platform

Google Cloud Platform (GCP) es el conjunto de servicios de computación en la nube ofrecido por Google, está diseñado para proporcionar soluciones escalables, sólidas y seguras. Su funcionamiento se basa en crear, gestionar y desplegar aplicaciones y servicios de forma eficiente, delegando la gestión de los recursos físicos. Sus principales características son:

- **Amplia gama de servicios**: Ofrece servicios de computación como Compute Engine o App Engine, también de almacenamiento donde destaca Cloud Storage, bases de datos como Cloud SQL o su solución no relacional Firestore. Incluye, además, servicios de inteligencia artificial y machine learning (Vertex AI, AutoML), entre muchos otros. Más adelante se describirán los servicios utilizados en el proyecto.

- **Escalabilidad y alta disponibilidad**: Google Cloud permite ajustar dinámicamente los recursos según la demanda y necesidad del proyecto, garantizando así la disponibilidad y el rendimiento de las aplicaciones.

- **Seguridad avanzada**: Integra servicios de protección de datos y autenticación, generación de claves en formato JSON, así como certificaciones de cumplimiento con los principales estándares de seguridad ISO.

- **Modelos de facturación flexibles**: Permite pagar únicamente por los recursos utilizados, ofreciendo un formato de pago bajo demanda.

Es por estas capacidades que ofrece GCP por lo que resulta ideal tanto para startups y pequeños proyectos que buscan un entorno de desarrollo rápido y económico, como para grandes empresas que necesitan soluciones de alta disponibilidad, procesamiento de grandes volúmenes de datos y servicios avanzados de inteligencia artificial.

Servicios de GCP utilizados:

- **Google Cloud Storage**: Es el servicio de almacenamiento en la nube de GCP, altamente escalable y seguro que permite almacenar datos no estructurados como imágenes o documentos. Ofrece alta disponibilidad, replicación de datos y control de acceso a los ficheros mediante políticas de permisos, lo que lo convierte en una solución ideal para gestionar contenido estático. Se ha utilizado para el almacenamiento de las denuncias en formato PDF.

- **Google Cloud Vision**: Servicio de inteligencia artificial que permite analizar y procesar imágenes mediante modelos de machine learning. Ofrece capacidades como etiquetado de imágenes, reconocimiento óptico de texto en imágenes (OCR), identificación de objetos y análisis de rostros. En este proyecto, se ha utilizado para extraer el texto de las imágenes reportadas por los usuarios. Ofrece hasta mil peticiones gratuitas mensuales.

- **Google Cloud Auth**: Servicio de autenticación y gestión de identidades, permite implementar de forma segura el control de acceso a las aplicaciones con las cuentas de Google. La autenticación se realiza mediante tokens. Su integración garantiza la seguridad de los usuarios y facilita el inicio de sesión a los usuarios y la gestión de sesiones. Se ha integrado con la librería NextAuth.js.

- **Google Cloud reCAPTCHA**: Herramienta de protección frente a bots y accesos automatizados maliciosos. Ayuda a garantizar que las acciones realizadas en la plataforma como la creación de reportes por usuarios no regustrados son llevadas a cabo por personas reales. La integración de reCAPTCHA contribuye a mejorar la seguridad de la aplicación, ayuda a prevenir ataques de spam, contribuye a mitigar los intentos de saturar los endpoints mediante peticiones automatizadas, lo que ayuda a reducir el riesgo de ataques de denegación de servicio (DDoS) a nivel de aplicación.

2.5 Deepseek API

Deepseek es un modelo avanzado de inteligencia artificial generativa (IAG) especializado en procesamiento de lenguaje natural (NLP), diseñado para comprender, razonar y generar contenido textual de alta calidad. A diferencia de modelos específicos centrados en la clasificación de texto, Deepseek destaca por su capacidad de realizar tareas de razonamiento lógico, resolución de problemas complejos y análisis profundo de contextos, lo que lo hace especialmente adecuado para entornos donde necesitamos

analizar un texto basado en un contexto ya sea explícito o no, pudiendo captar ironía y sarcasmo, dando un resultado más acertado. Entre sus características descantan:

- **Capacidades de Razonamiento Avanzado**: Deepseek cuenta con el modelo R1 el cual no se limita a generar texto fluido, sino que está entrenado para aplicar principios de lógica y razonamiento, permitiendo resolver problemas matemáticos, analizar casos complejos y ofrecer explicaciones fundamentadas.

- **Amplia Comprensión Multidisciplinar**: Ha sido entrenado en grandes corpus de datos que abarcan disciplinas como matemáticas, física, informática, derecho, finanzas, entre otras, lo que le permite ofrecer respuestas especializadas en diferentes ámbitos del conocimiento.

- **Oferta competitiva**: DeepSeek ha irrumpido en un mercado tradicionalmente dominado por compañías como OpenAI (ChatGPT) y Anthropic (Claude), ofreciendo modelos de rendimiento comparable, especialmente en tareas de razonamiento lógico, matemáticas avanzadas y generación de código, a un coste significativamente más bajo. Gracias a su estrategia de precios accesibles y a la alta eficiencia de sus modelos, DeepSeek se presenta como una alternativa viable para proyectos que requieren capacidades avanzadas de IA sin asumir los altos costes asociados a modelos como GPT-o1 o Claude 3.7 Sonnet.

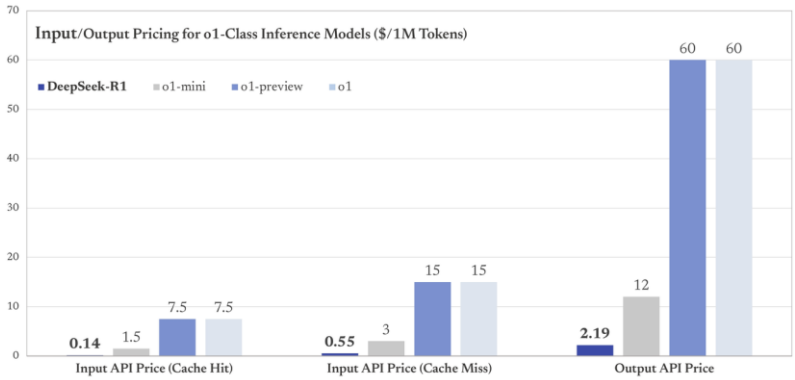


Figura 4. Comparativa de precios entre DeepSeek-R1 y modelos de la clase o1 en dólares. Fuente: *DeepSeek API Documentation* (s.f.). Recuperado de https://api docs.deepseek.com/news/news250120

	AIME 2024 pass@1	AIME 2024 cons@64	MATH-500 pass@1	GPQA Diamond pass@1	LiveCodeBench pass@1	CodeForces rating
GPT-4o-0513	9.3	13.4	74.6	49.9	32.9	759.0
Claude-3.5-Sonnet-1022	16.0	26.7	78.3	65.0	38.9	717.0
o1-mini	63.6	80.0	90.0	60.0	53.8	**1820.0**
QwQ-32B	44.0	60.0	90.6	54.5	41.9	1316.0
DeepSeek-R1-Distill-Qwen-1.5B	28.9	52.7	83.9	33.8	16.9	954.0
DeepSeek-R1-Distill-Qwen-7B	55.5	83.3	92.8	49.1	37.6	1189.0
DeepSeek-R1-Distill-Qwen-14B	69.7	80.0	93.9	59.1	53.1	1481.0
DeepSeek-R1-Distill-Qwen-32B	**72.6**	83.3	94.3	62.1	57.2	1691.0
DeepSeek-R1-Distill-Llama-8B	50.4	80.0	89.1	49.0	39.6	1205.0
DeepSeek-R1-Distill-Llama-70B	70.0	**86.7**	**94.5**	**65.2**	**57.5**	1633.0

Figura 5. Comparativa de rendimiento de distintos modelos de lenguaje en diferentes benchmarks. Fuente: *DeepSeek API Documentation* (s.f.). Recuperado de https://api-docs.deepseek.com/news/news250120

- **Control de Salida y Personalización**: Ofrece la posibilidad de ajustar la creatividad de las respuestas (a través de parámetros como la temperatura) y de orientar las salidas hacia respuestas más concisas, extensas o detalladas según la necesidad del usuario.

En este proyecto, DeepSeek ha sido el encargado de analizar el contenido de los reportes, clasificándolos en discurso de odio o no. A su vez, gracias al modelo R1, se obtiene un razonamiento de la decisión que se añade a la resolución que recibe el usuario.

2.6 Azure Computer Vision

Azure, al igual que GCP es una plataforma de servicios de computación en la nube, en este caso de Microsoft, se ha integrado el servicio de Computer Vision, que proporciona una gran cantidad de funcionalidades como reconocimiento óptico de texto, detección de caras, detección de elementos en imágenes y la funcionalidad que ha sido explotada en este caso, descripción de imágenes. Además, ofrece un plan sin costes de cinco mil peticiones mensuales.

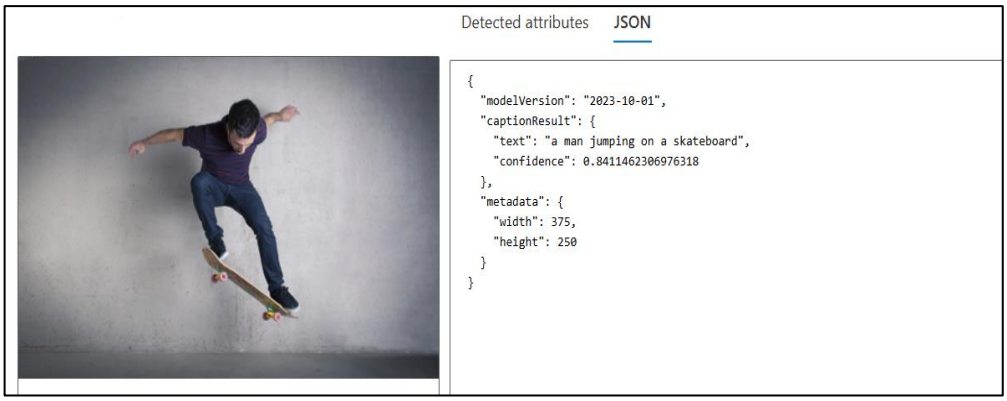


Figura 6. Ejemplo de uso de la API de Azure Cloud Vision en la generación de descripciones de imágenes. Fuente: *Azure Cognitive Services* (s.f.). Recuperado de https://portal.vision.cognitive.azure.com/demo/image-captioning

2.7 ScraperAPI

ScraperAPI es un servicio en la nube que facilita la recolección de datos de sitios web a través de técnicas de web scraping. Su principal ventaja es que permite realizar peticiones HTTP para obtener el contenido de páginas web sin necesidad de preocuparse por las restricciones habituales que aplican los sitios web para bloquear o limitar el scraping. Sus principales características son:

- **Gestión automática de proxies**: Permite realizar peticiones desde diferentes direcciones IP, lo que ayuda a evitar bloqueos y restricciones basadas en la ubicación o el volumen de peticiones.

- **Manejo de CAPTCHAs y bloqueos**: Incluye mecanismos para resolver o evitar CAPTCHAs de forma automática.

- **Soporte para contenido dinámico**: Puede procesar tanto contenido estático como páginas que utilizan JavaScript para cargar información.

- **Fácil integración**: La API se utiliza mediante peticiones HTTP estándar, lo que permite integrarla fácilmente en aplicaciones desarrolladas en cualquier lenguaje de programación.

- **Plan gratuito**: Ofrece un plan gratuito con un número limitado de peticiones mensuales, ideal para proyectos de pequeño alcance o pruebas iniciales.

Gracias a estas características, ScraperAPI es una herramienta muy útil y cómoda para automatizar la recolección de información de sitios web sin necesidad de desarrollar soluciones desde cero. En esta aplicación, su principal uso ha sido para extraer contenido de publicaciones de X, ya que es contenido generado dinámicamente.

2.8 Vercel

Vercel es una plataforma como servicio (PaaS) que permite desplegar proyectos software de forma gratuita, sencilla y eficazmente. Vercel es la empresa creadora de Next.js con el cual cuenta con una compatibilidad completa, sin embargo, también acepta proyectos basados en otros frameworks de JavaScript y con la configuración adecuada también proyectos basados en Flask, por ejemplo. Tiene soporte para funciones de servidor, permitiendo ejecutar la lógica de back-end sin necesidad de gestionar servidores.

Su principal atractivo es la automatización de despliegues a partir de repositorios Git, es decir, cada Commit que se ejecute se desplegará automáticamente en Vercel.

2.9 Github

Github es una plataforma de desarrollo colaborativo basada en la web que permite almacenar, gestionar y compartir proyectos de software utilizando el sistema de control de versiones Git. Es una herramienta clave para cualquier proyecto de desarrollo tanto individual como en equipo. Entre sus características más destacadas:

- **Control de versiones con Git**: Permite llevar un historial completo de los cambios en el código, facilitando el seguimiento y modificaciones a lo largo del tiempo.

- **Gestión de ramas**: Facilita la creación de entornos paralelos de desarrollo, permitiendo que varias personas trabajen en funcionalidades distintas dentro de un mismo proyecto, también hacer pruebas sin riesgo a afectar a la versión principal del proyecto.

- **Autenticación**: Github también permite utilizar su pasarela de autenticación de forma sencilla para gestionar el acceso de los usuarios haciendo uso del protocolo OAuth2.

Por estos motivos Github ha sido una pieza clave en este proyecto, fomentando el seguimiento y la mantenibilidad del código.

2.10 Cloudinary

Cloudinary es una plataforma como servicio (PaaS) especializada en la gestión de imágenes y videos para aplicaciones web y móviles. Proporciona una solución integral para almacenar, transformar, optimizar y entregar archivos multimedia de forma eficiente mediante una CDN (Content Delivery Network).

Además, ofrece una API potente y fácil de integrar, con SDKs disponibles para múltiples lenguajes y frameworks. Gracias a su integración con sistemas de almacenamiento en la nube y herramientas de desarrollo modernas, Cloudinary se convierte en una opción ideal para gestionar las imágenes proporcionadas como método de creación de denuncias.

2.11 Herramientas de diseño

Visual Paradigm

Visual Paradigm es una herramienta de modelado de software y diseño visual que permite crear, gestionar y documentar diagramas utilizados en la ingeniería de software, el análisis de sistemas y la gestión de proyectos.

Está orientada a facilitar la representación gráfica de arquitecturas de software, procesos de negocio y modelos de datos, ayudando en la comunicación y comprensión de sistemas complejos.

Figma

Figma, por su parte, es una herramienta enfocada al diseño de interfaces de usuario de usuario (UI/UX), contando con un diseño moderno y atractivo. Permite crear maquetas visuales y prototipos interactivos que simulan en flujo de navegación entre las distintas vistas. Su principal ventaja es que no se necesita instalación al estar basado en la web por lo que funciona directamente en el navegador.

Además, Figma cuenta con una gran comunidad que comparte sus creaciones para integrarlas en tus propios proyectos.

2.12 Entornos de desarrollos integrados (IDEs)

Visual Studio Code

VSCode es el IDE multiplataforma más popular a nivel global, está desarrollado por Microsoft, destaca por su diseño ligero, minimalista y altamente personalizable. Sus características más destacables son:

- **Multilenguaje**: Soporta múltiples lenguajes de programación gracias a su ecosistema de extensiones.

- **Extensiones potentes**: Dispone de una amplia galería de extensiones que permiten integrar linters, depuradores, autocompletado, Git, Jupyter Notebooks, etc.

- **Terminal integrada**: Permite ejecutar comandos sin salir del entorno.

- **Control de versiones integrado**: Integra Git nativamente, facilitando commits, ramas y push/pull.

- **Ligero y rápido**: Ideal para proyectos pequeños y medianos, o cuando se prioriza la rapidez.

Pycharm

Pycharm es un IDE específico de Python, desarrollado por JetBrains (creadores de IntelliJ, PhpStorm, entre otros). Ofrece un entorno robusto y completo para el desarrollo de software en Python, especialmente útil en contextos como el desarrollo web con compatibilidad nativa con marcos de trabajo como Flask o Django. Sus principales características son:

- **Soporte completo para Python**: Autocompletado avanzado, detección de errores, refactorización, y navegación entre archivos.

- **Depurador y herramientas de pruebas:** Integración con unittest, pytest, y depuración paso a paso.

- **Integración con bases de datos**: Acceso y consulta directa a bases de datos desde el IDE.

- **Swagger integrado**: Pycharm ofrece soporte para la edición y previsualización de documentación OpenAPI.

3

Especificación y análisis

Para el proyecto se ha desarrollado un Documento General de Requisitos (DGR) del que se describen sus contenidos a continuación.

3.1 Participantes

- **Pablo Senciales de la Higuera**

  Estudiante y desarrollador principal del proyecto. Encargado del diseño, especificación y posterior implementación.

- **Eduardo Guzmán de los Riscos**

  Director del Trabajo de Fin de Grado. Ha ejercido funciones de supervisión, asesoramiento técnico y revisión durante todas las fases del desarrollo.

3.2 Actores

- **Invitado**

  Puede analizar contenido para comprobar si constituye discurso de odio. Si desea formalizar una denuncia, deberá proporcionar un correo electrónico para ser notificado del estado de esta. No requiere registro previo.

- **Usuario registrado**

  Tiene las mismas funcionalidades del invitado, pero además puede acceder a su historial de denuncias y comprobar el estado de cada una.

- **Administrador**

  Gestiona las denuncias recibidas verificando su contenido y administra los usuarios del sistema.

- **Sistema**

  Se encarga de analizar automáticamente el contenido proporcionado por los usuarios haciendo uso de una IA, a su vez, genera los reportes en formato PDF y gestiona las notificaciones por correo electrónico, así como almacenar y actualizar el estado de las denuncias.

3.3 Requisitos funcionales

Id	Descripción	Prioridad	Actor
RF-01	Permitir la subida de texto/imagen/tuit para su análisis.	Alta	Usuario/ Invitado
RF-02	El sistema utilizará un LLM para analizar el contenido y detectar discurso de odio.	Alta	Sistema

RF-03	Se generará un informe en PDF para su denuncia.	Alta	Sistema
RF-04	Los invitados deberán aportar un correo electrónico para realizar una denuncia.	Media	Invitado
RF-05	El sistema notificará los cambios de estado de las denuncias vía email a los usuarios.	Media	Sistema
RF-06	Los usuarios podrán visualizar y consultar el estado de sus denuncias.	Alta	Usuario
RF-07	Los usuarios podrán descargar sus denuncias.	Alta	Usuario
RF-08	Los administradores podrán prohibir el acceso a un usuario basado en su correo electrónico.	Baja	Administrador
RF-09	Los administradores podrán aceptar o rechazar las denuncias.	Alta	Administrador
RF-10	El sistema permitirá autenticación con OAuth 2.0.	Alta	Sistema
RF-11	Los administradores podrán visualizar todas las denuncias.	Alta	Administrador
RF-12	El sistema registrará la actividad de los administradores en logs para auditoría y trazabilidad.	Baja	Sistema
RF-13	Se generarán estadísticas y métricas como la tasa de aceptación/rechazo, tasa de detección, etc.	Baja	Sistema

RF-14	El sistema integrará un captcha para evitar spam.	Baja	Sistema
RF-15	Los usuarios podrán eliminar su cuenta de acuerdo con la LOPD.	Alta	Usuario
RF-16	Los administradores podrán marcar una denuncia como “procesando”, “aceptada” o “rechazada”.	Media	Administrador

3.4 Requisitos no funcionales

Id	Descripción	Prioridad	Actor
RNF-01	Diseño atractivo y minimalista	Alta	Sistema
RNF-02	La plataforma tendrá soporte multilenguaje de español e inglés.	Baja	Sistema
RNF-03	La plataforma será responsiva, adaptándose a distintos tamaños de pantalla.	Alta	Sistema
RNF-04	La arquitectura del sistema deberá facilitar su futura.	Media	Sistema
RNF-05	El sistema deberá ser fácilmente mantenible y permitir actualizaciones sin afectar al servicio.	Media	Sistema

3.5 Casos de uso

- **Usuario registrado crea un reporte a partir de un enlace de X.**

  Usuario registrado a partir de su contenido crea un reporte que genera un PDF con el análisis que posteriormente le será enviado vía mail y se almacena en la base de datos.

  Precondiciones

  - El usuario tendrá una cuenta registrada y sin el acceso limitado, habiendo iniciado sesión.
  - El usuario debe estar en la vista de creación de reportes.

  Postcondiciones

  - El sistema ha analizado el contenido, generado un PDF y enviado por mail al usuario.
  - Se ha creado un reporte en la BBDD.

  Escenario principal

  1. El usuario rellena el formulario con el enlace de X y el contexto que considere y le da a analizar.
  2. El sistema envía el contenido a los servicios externos para su análisis.
  3. El sistema genera un PDF con el contenido, razonamiento del análisis y conclusión.
  4. Envía un mail al usuario con el PDF.
  5. El sistema crea una denuncia en la BBDD.

  Excepciones

  2.a. Los servicios externos no están disponibles, el sistema genera un mensaje de error.

  Diagrama de secuencia del escenario principal

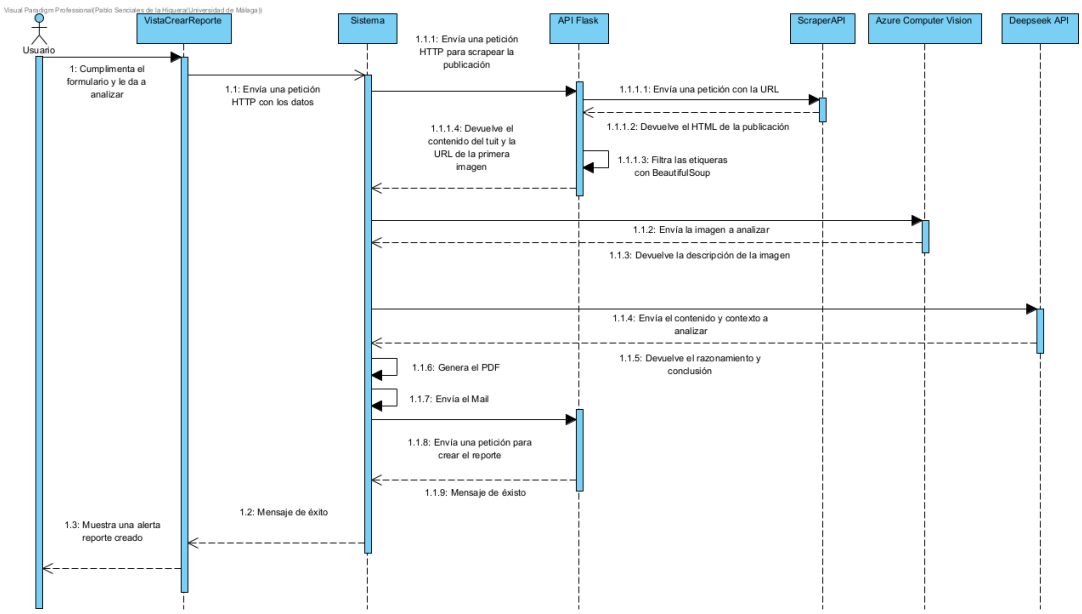


Figura 7. Diagrama de secuencia del escenario principal de la creación de una denuncia a partir de una publicación de X.

- **Administrador acepta un reporte**

  Administrador, tras revisar el contenido, toma la decisión de aceptar el reporte.

  Precondiciones

  - El administrador cuenta con una cuenta con el rol de administrador.
  - El administrador debe estar en la vista de valorar reportes de los usuarios.

  Postcondiciones

  - Se envía un mail al usuario que creó el reporte.
  - Se actualiza el estado del reporte en la BBDD.
  - Se crea una resolución en la BBDD

  Escenario principal

  1. El administrador selecciona valorar el reporte deseado y cumplimenta los motivos, seleccionando ‘aceptar’.
  2. El sistema envía un mail al usuario con el cambio de estado.
  3. El sistema actualiza el estado del reporte.
  4. El sistema crea una resolución en la BBDD.

Diagrama de secuencia del escenario principal

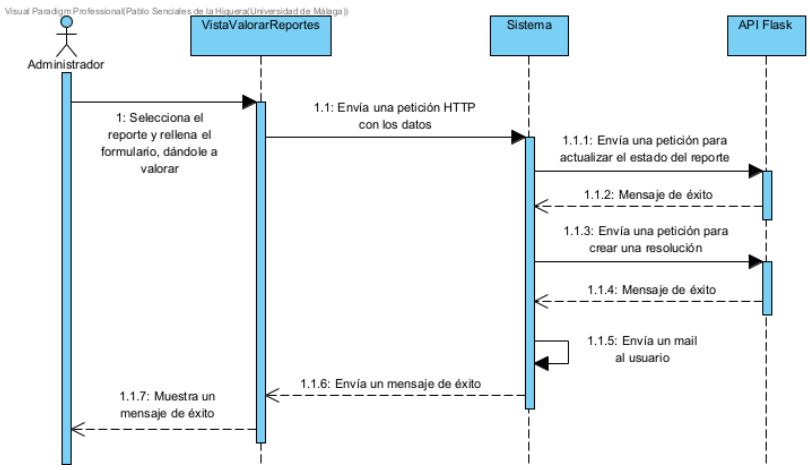


Figura 8. Diagrama de secuencia del escenario principal de la aceptación de una denuncia por un administrador

- **Administrador prohíbe el acceso a un usuario.**

  Administrador decide prohibir el acceso a un usuario basado en su correo electrónico.

  Precondiciones

  - El administrador cuenta con una cuenta con el rol de administrador.
  - El administrador debe estar en la vista de valorar reportes de los usuarios.

  Postcondiciones

  - Se envía un mail al usuario al que corresponda el correo electrónico.
  - Se crea un log con la acción del administrador en la BBDD.
  - Se crea un 'usuario prohibido' en la BBDD.

  Escenario principal

  1. El administrador selecciona prohibir el acceso del usuario de un reporte.
  2. El sistema envía un mail al correo electrónico del usuario.
  3. El sistema crea un 'usuario prohibido'.

4. El sistema crea un log con la acción de haber prohibido el acceso a un usuario.

Diagrama de secuencia del escenario principal

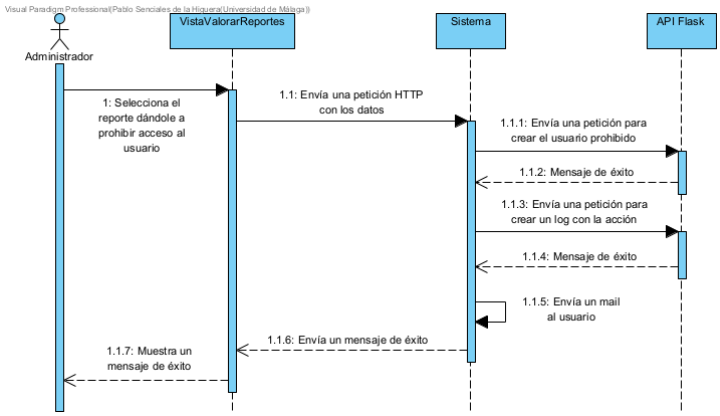


Figura 9. Diagrama de secuencia del escenario principal de la prohibición de acceso a un usuario.

3.6 Modelo del dominio

La *Figura 10* representa el modelo de clases del dominio del sistema. Se observa, que la entidad Usuario hereda de invitado, extendiendo su funcionalidad y relaciones. El usuario se relaciona mediante composición **1:1** con las entidades Historial y Sesión, lo que implica que ambas dependen completamente de su existencia, si el usuario es eliminado, tanto su historial como sesión serán eliminados.

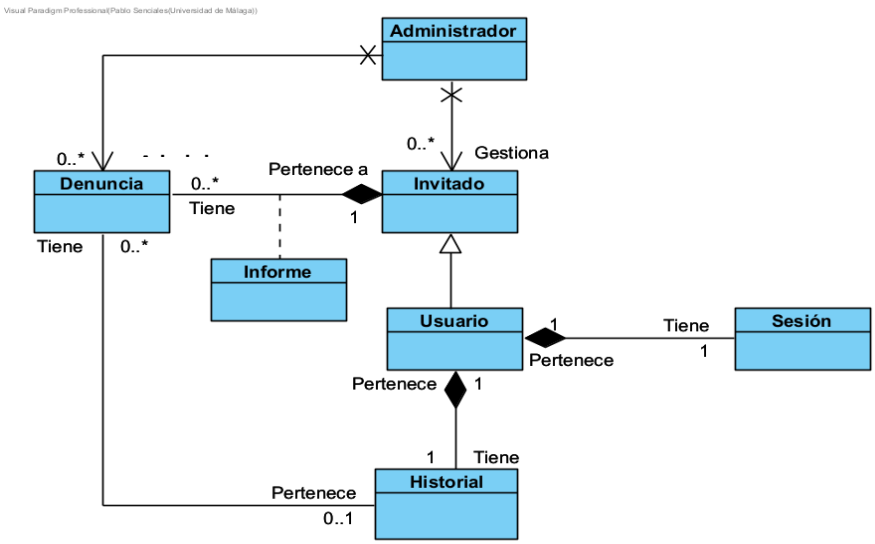


Figura 10. Diagrama del modelo del dominio del sistema.

Por otro lado, la entidad Invitado (y Usuario, por extensión), se relaciona con la entidad Denuncia mediante una relación de composición **0..*:1**, lo que indica que un invitado puede tener múltiples denuncias asociadas, pero estas no pueden existir sin un propietario. A su vez, siempre que se cree una relación entre de Denuncia-Invitado, se crea una entidad Informe que depende de esta relación para su existencia.

Finalmente, la entidad Administrador se relaciona unidireccionalmente con multiplicidad **0..*** con las entidades Invitado y Denuncia, lo que significa que los administradores gestionan múltiples usuarios y reportes sin que exista una dependencia en el sentido inverso.

4

Diseño del sistema

4.1 Arquitectura del sistema

La *Figura 11* muestra la arquitectura técnica completa del sistema Fairplay360, destacando los distintos módulos que lo componen y sus interacciones con los servicios externos. El sistema está estructurado en varios bloques funcionales y tecnológicos, que se comunican entre sí de forma segura mediante el protocolo HTTPS.

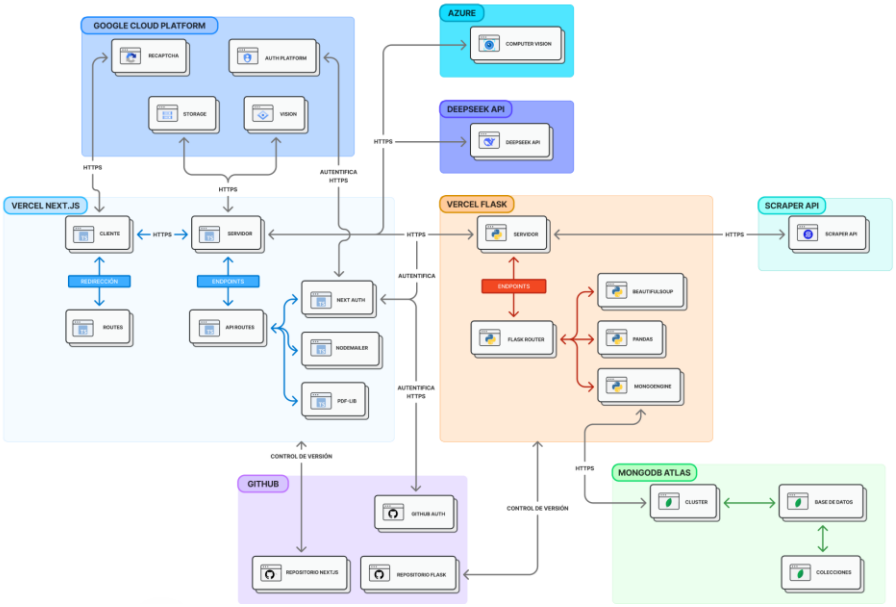


Figura 11. Esquema de la arquitectura del sistema.

- **Front-end y Back-end en Vercel (Next.js)**

  El cliente se comunica con el servidor de Next.js, que gestiona Routes y API Routes. Componentes como NextAuth, Nodemailer y pdf-lib permiten implementar la autenticación, el envío de correos y la generación de informes en PDF. Este entorno se despliega en la plataforma Vercel y se encuentra versionado a través de GitHub.

- **Back-end en Vercel (Flask)**

  Un segundo back-end desarrollado en Flask, desplegado también en Vercel, expone endpoints que interaccionan con la base de datos. Este servidor consume servicios de scraping (mediante ScraperAPI), procesamiento HTML con BeautifulSoup, análisis estadístico con Pandas y acceso a base de datos mediante MongoEngine.

- **Servicios externos**:
  - **Google Cloud Platform** proporciona servicios como:
    - reCAPTCHA para protección contra bots.
    - Auth Platform para autenticación segura.
    - Storage para almacenar las denuncias PDF.
    - Vision API para el OCR.
  - **Azure Computer Vision** se utiliza para el análisis de imágenes.
  - **Deepseek API** proporciona procesamiento de lenguaje natural para clasificar posibles discursos de odio.
  - **ScraperAPI** se usa para la extracción de información de las publicaciones de X que sirvan como denuncia.

- **MongoDB Atlas**:

  Es el sistema de almacenamiento centralizado de la plataforma, donde se alojan las colecciones de usuarios, denuncias, informes y sesiones. Está conectado con el back-end de Flask mediante una conexión segura.

- **Control de versiones (GitHub)**:

  Ambos repositorios (Next.js y Flask) se gestionan mediante GitHub que además nos permite un despliegue sencillo en Vercel. También integra su servicio de autenticación en el sistema.

Este diseño modular y escalable permite mantener separados los distintos dominios funcionales (interfaz, análisis, datos, seguridad), facilitando su evolución y mantenimiento a largo plazo.

4.2 Diseño de la base de datos

La *Figura 12* muestra el modelo de la base de datos, compuesto por varias entidades clave que representan el núcleo funcional del sistema Fairplay360. Este diseño está orientado a una base de datos NoSQL como MongoDB, pero se representa en formato relacional para una mayor claridad estructural. Además, cada entidad contiene los atributos principales necesarios para la gestión de usuarios, denuncias, archivos y resoluciones.

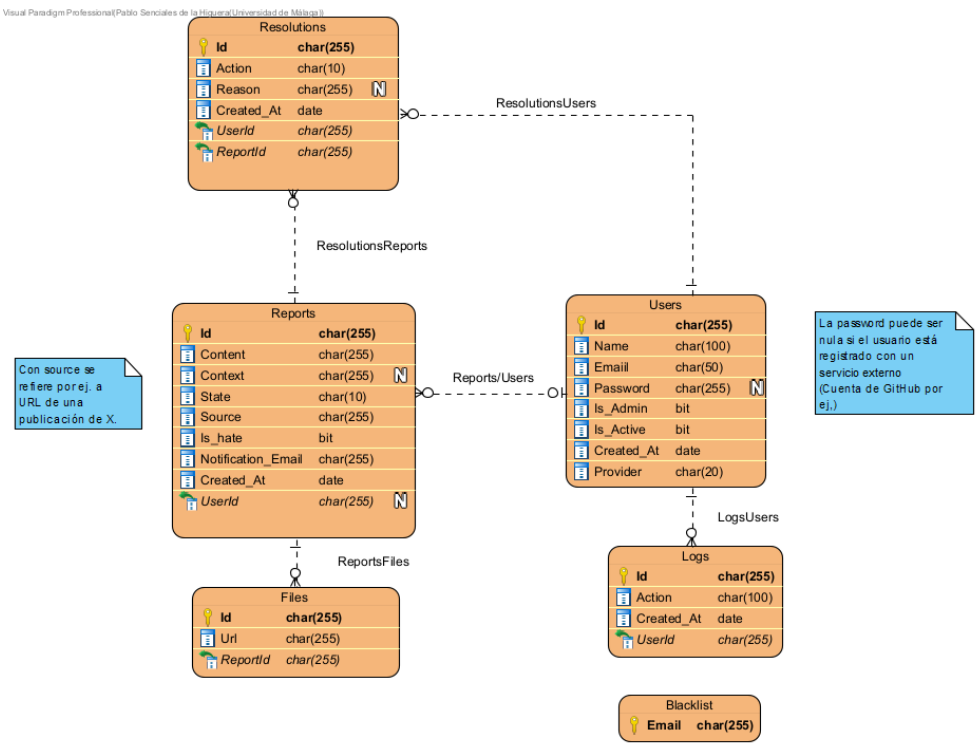


Figura 12. Diagrama del modelo relacional de la base de datos del sistema.

Users: Almacena los datos de los usuarios registrados. El campo Password puede ser nulo si el usuario accede a través de un proveedor externo (por ejemplo, GitHub con OAuth). Se incluye un campo Is_Admin para distinguir administradores, también otro campo Is_Active necesario para la técnica de *soft delete* (borrado lógico, en lugar de eliminar físicamente al usuario en la base de datos, solo se inhabilita).

Reports: Representa las denuncias realizadas por usuarios o invitados. Contiene el contenido denunciado, un posible contexto adicional, su estado (State), y si ha sido clasificado como discurso de odio (Is_hate). Debe estar vinculado a un correo de notificación, sin embargo, no necesariamente a un usuario debido a que puede pertenecer a un invitado (usuario no registrado).

Files: Almacena archivos como la denuncia en PDF asociada a un reporte. Cada archivo contiene su URL y se relaciona con un ReportId.

Resolutions: Contiene las decisiones tomadas por los administradores sobre cada reporte. Incluye una acción (Action), una razón justificada (Reason), y la fecha de resolución. Está vinculado al administrador que la emite (UserId) y al reporte afectado.

Logs: Registro de acciones realizadas por administradores en el sistema, lo que garantiza la trazabilidad y auditoría. Cada log indica qué acción se ha realizado, quién la ha realizado y cuándo.

Blacklist: Almacena los correos electrónicos bloqueados del sistema. Se usa para impedir que ciertos usuarios reincidan en comportamientos inapropiados.

4.3 Diseño del modelo de datos

Al tratarse de una base de datos no relacional, se puede adaptar el diagrama de la *Figura 12* para simplificar el funcionamiento y aprovechar la flexibilidad de los documentos JSON que ofrece MongoDB. A diferencia de los modelos relacionales,

donde cada entidad se representa en tablas separadas y conectadas mediante claves foráneas, MongoDB permite incrustar documentos dentro de otros, mejorando el rendimiento en operaciones de lectura.

Por ejemplo, la entidad Reports contiene una relación con Files que en un modelo relacional se representa como una tabla aparte. En MongoDB, esta relación se puede modelar como una lista de subdocumentos embebidos dentro del propio documento del reporte, lo que facilita su consulta y simplifica la estructura general. A su vez, y de manera similar, la entidad Resolutions se encuentra embebida dentro de Reports. Sin embargo, los Logs se han mantenido en una colección aparte, por la necesidad de consultar estos datos de manera independiente para mantener la trazibilidad. En la Figura *13* se encuentra una instancia de un reporte en formato JSON, donde se observan los cambios mencionados.

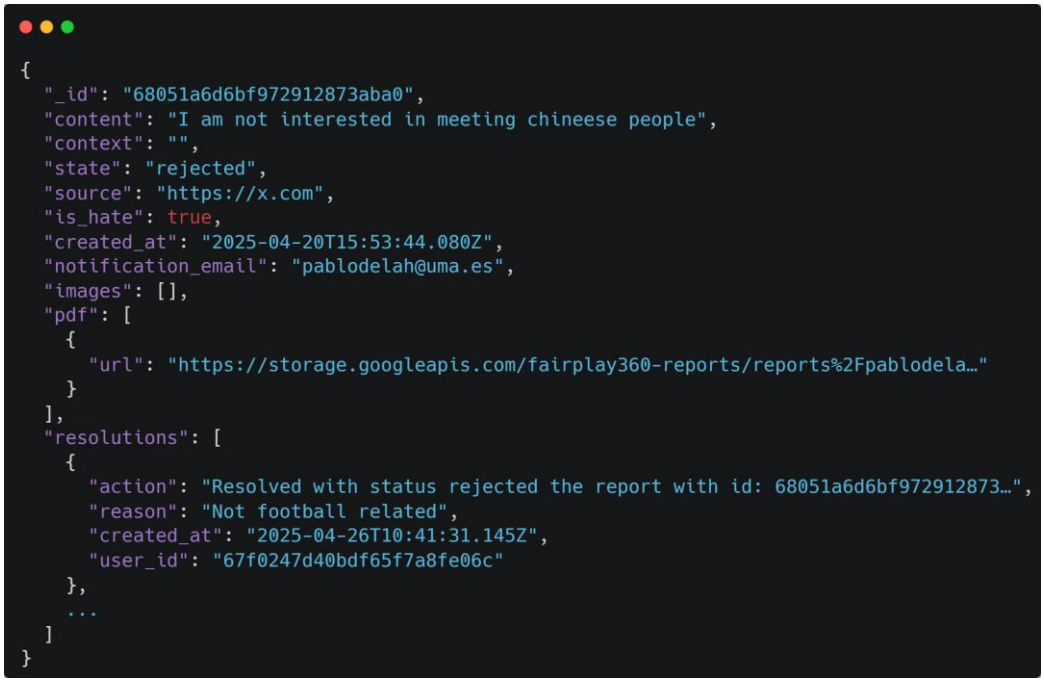


Figura 13. Estructura en formato JSON de una denuncia.

4.5 Diseño de interfaz

Para el desarrollo de la interfaz de usuario de la plataforma, se utilizó inicialmente Figma como herramienta de prototipado. Con ella se crearon los primeros bocetos (mockups) de las principales pantallas, lo que permitió validar visualmente la estructura, la disposición de los elementos y la experiencia de usuario antes de comenzar la implementación.

Esta fase fue esencial para anticipar posibles problemas de usabilidad, definir un diseño claro y minimalista. Los mockups sirvieron también como base para establecer la navegación entre vistas, los estilos visuales y la coherencia en los componentes.

Una vez definidos los prototipos en Figma, se procedió a su implementación en el entorno de desarrollo utilizando Next.js con soporte de TailwindCSS, manteniendo la fidelidad con el diseño propuesto.

Entre las vistas más importantes se encuentra la de creación de una denuncia con las distintas pestañas que indican el origen de los datos (*Figura 14*), en la implementación final se optó por eliminar el checkbox que indicaba si se añadía contexto, ya que carecía de sentido.

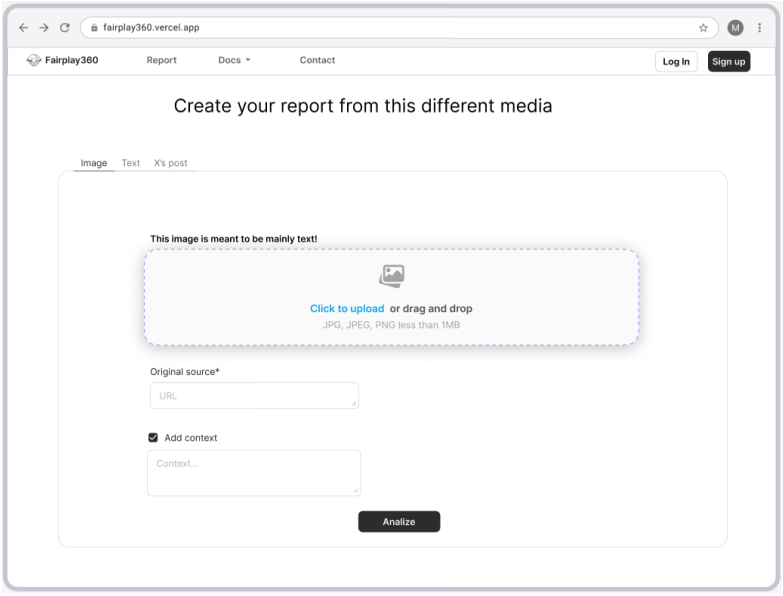


Figura 14. Prototipo de interfaz de la vista 'Formulario de denuncia'.

También, se encuentra la vista dónde se visualizan las denuncias creadas por un usuario (*Figura 15*). En el diseño final se eliminó la barra de búsqueda ya que no había un criterio claro por el cual buscar (los administradores sí tienen una búsqueda por email de usuario), también se eliminó la imagen que aparece en cada denuncia. Sin embargo, se amplío la pestaña de ordenación y filtros. Por último, se cambió la opción de descargar la denuncia por la opción de visualizar, donde se abre el PDF con su URL pública alojada en Google Storage.

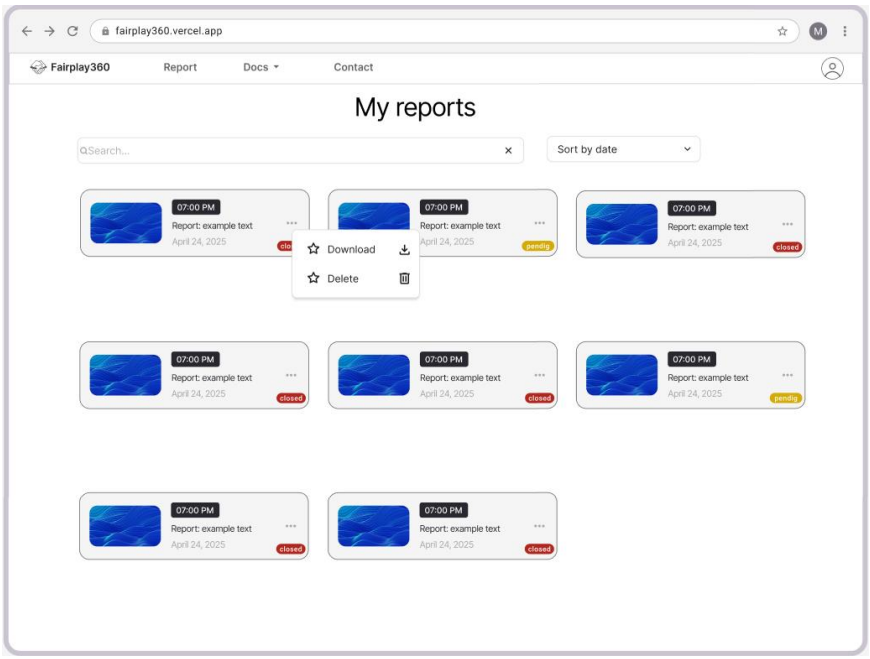


Figura 15. Prototipo de interfaz de la vista 'Mis denuncias'.

En el siguiente enlace se puede consultar el prototipo interactivo completo: Prototipos.

5

Implementación y pruebas

5.1 Implementación

A continuación, se repasan los distintos módulos del proyecto, entrando en detalles de cómo fue su desarrollo.

5.1.1 Servicio Flask

Fue el primer módulo implementado, aunque ha ido evolucionando en función de las necesidades que fueron surgiendo durante el desarrollo del front-end.

Este back-end actúa principalmente como una interfaz de acceso a la base de datos, encargándose de exponer endpoints seguros y bien organizados para interactuar con las colecciones definidas. Además, incluye una funcionalidad específica para la

extracción de publicaciones externas, utilizando ScraperAPI en combinación con BeautifulSoup para limpiar y extraer datos relevantes del HTML.

- **Estructura**

La estructura de carpetas se ha organizado siguiendo criterios de modularidad y claridad, permitiendo una fácil escalabilidad y mantenimiento. A continuación, se describe el contenido de cada archivo y carpeta:

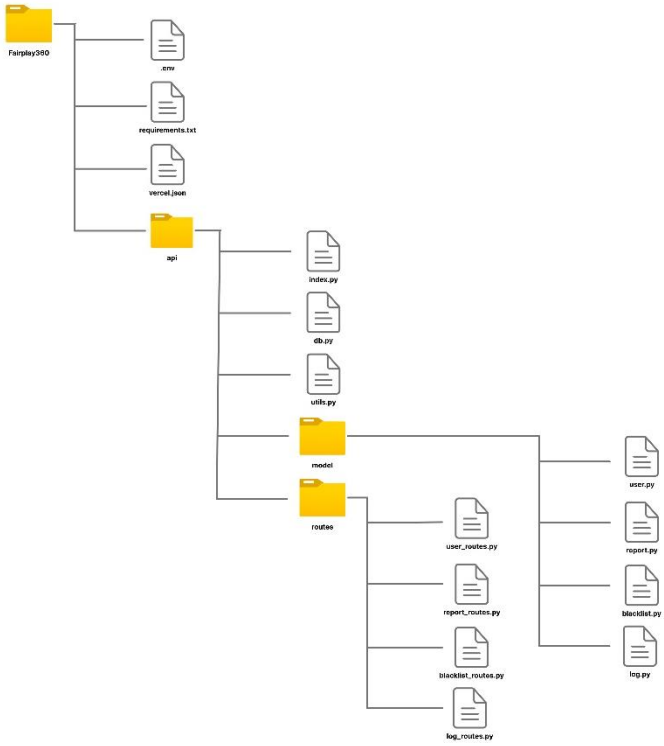


Figura 16. Estructura de carpetas en Flask.

Carpeta Fairplay360

Contiene los siguientes archivos:

.env: Contiene las variables de entorno, como la URI de conexión a Atlas o las claves de acceso a APIs externas.

requirements.txt: Lista todas las dependencias del proyecto (Flask, MongoEngine, Request, etc.) que nos permite instalar el entorno fácilmente.

vercel.json: Archivo de configuración para el despliegue en Vercel.

Carpeta api

Que se organiza de la siguiente forma:

- **index.py**: Archivo principal donde se instancia la aplicación Flask y se registran los distintos blueprints de las rutas. Es el punto de entrada de la aplicación.

- **db.py**: Contiene la configuración de la conexión a la base de datos MongoDB Atlas.

- **utils.py**: Archivo con funciones auxiliares, como validación de entradas, manejo de errores y formateo de respuestas.

- **Carpeta model:** Define los esquemas de los datos utilizando MongoEngine, un archivo por cada entidad. En la *Figura 17* se ejemplifica cómo se definen estas estructuras.

```python
class Log(Document):
    action = StringField(required=True)
    created_at = DateTimeField(default=lambda: datetime.now(UTC))
    user_id = StringField(required=True)
```

Figura 17. Fragmento de código de la definición de un esquema en MongoEngine.

- **Carpeta routes:** Contiene la definición de las distintas rutas, organizadas por entidad. Se han definido rutas de creación, lectura, modificación y borrado (CRUD), además de otras funciones específicas, utilizando Flask Blueprint para modularizar la lógica. En la *Figura 18* se ejemplifica cómo se define una ruta básica *'Get All'*.

```python
from api.model import Log
from flask import Blueprint

log_bp = Blueprint('log_routes', __name__, url_prefix='/logs')

@log_bp.route('/', methods=['GET'])
@log_bp.route('', methods=['GET'])
def get_logs():
    logs = Log.objects() # Devuelve todos los logs de la BBDD
    if not logs:
        return jsonify({}), 200
    return jsonify(logs.to_json()), 200
```

Figura 18. Fragmento de código con la definición de la función get_logs.

➢ **Otros detalles de la implementación**

Seguiridad

Flask permite definir un middleware previo al procesamiento de las peticiones, es decir, intercepta todas las peticiones antes de llegar a la ruta especificada, permitiendo ejecutar una lógica previa a su procesamiento objetivo. En esta aplicación

su propósito es validar los tokens de sesión así restringiendo el acceso a rutas protegidas.

La función *verify_session* funciona de la siguiente manera, se comprueba, en función del proveedor del usuario, si el token de acceso es válido. Para esto tanto Github como Google generan un token cada vez que se inicia la sesión y tienen habilitado un endpoint para comprobar su validez, a su vez, por parte del servidor se almacena en la base de datos un token cada vez que se inicia la sesión con credenciales de la aplicación para su posterior verificación.

```
@app.before_request
def check_access_token():
    if request.path == "/login" or request.path.startswith("/users/email") or request.method
== "OPTIONS" or request.path.startswith("/users/reset-password"):
        return
    data = {}
    if request.method != "GET" and request.method != "DELETE":
        data = request.json
    if verify_error := verify_session(data, request):
        return verify_error
    return
```

Figura 19. Fragmento de código del middleware.

Además, se observa que dentro del middleware (*Figura 19*) implementado mediante *@app.before_request*, es posible excluir ciertos endpoints públicos que no requieren autenticación, como el de login o la recuperación de contraseña. También se contempla explícitamente la exclusión de las peticiones HTTP de tipo OPTIONS, que son utilizadas por los navegadores para realizar lo que se conoce como una preflight request.

Estas solicitudes previas se envían de forma automática antes de ejecutar ciertas operaciones (POST, PUT, etc.), con el objetivo de comprobar las políticas de seguridad del servidor, como las reglas definidas por CORS (*Cross-Origin Resource Sharing*). Dado que estas peticiones no contienen datos útiles ni requieren autenticación, deben quedar fuera del flujo de verificación de sesión para evitar errores de validación o bloqueos innecesarios.

Scraping

Para el análisis de publicaciones de X, es necesario extraer la información mediante *web scraping*, como se ha mencionado, se ha utilizado ScraperAPI, la llamada a la API se ha extraído en la función *web_scrap*. Nótese que esta ruta está definida en el *index.py.*

```python
from bs4 import BeautifulSoup
from flask import request, jsonify

@app.route('/scrape_tweets', methods=['POST'])
def scrape_tweets():
    data = request.json
    if missing := missing_fields(['url'], data):
        return missing

    url = data['url']

    html_rendered = web_scrap(url)
    soup = BeautifulSoup(html_rendered, 'html.parser')
```

Figura 20. Fragmento de código de la función scrape_tweets

A continuación, se debe utilizar la herramienta de inspección del navegador para ver la estructura HTML y localizar el contenido a extraer.

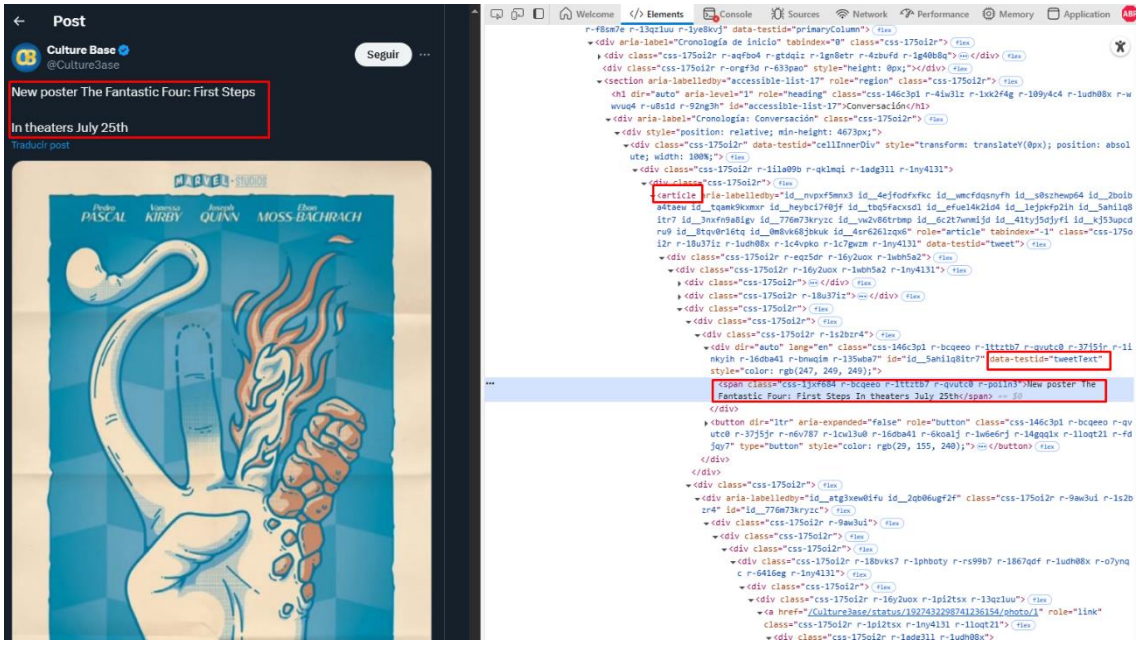


Figura 21. HTML de una publicación de X, URL original: https://x.com/Culture3ase/status/1927432298741236154

Continuando con el código de la *Figura 20* y conociendo ya la estructura (*Figura 21*), se debe filtrar el contenido utilizando BeautifulSoup (*Figura 22*).

```python
# Se busca el primer elemento <article>
article = soup.find('article')
tweet_text = None
image_src = None

if article:
    # Dentro del article, se busca el primer <div> con el atributo data-
    testid="tweetText"
    tweet_div = article.find('div', {'data-testid': 'tweetText'})
    if tweet_div:
        # Dentro de ese div, se busca el <span> que contiene el texto
        tweet_span = tweet_div.find('span')
        if tweet_span:
            tweet_text = tweet_span.get_text(strip=True)
    # De manera similar, se busca dentro del article el div que contiene <img>
    image_div = article.find('div', {'data-testid': 'tweetPhoto'})
    if image_div:
        image = image_div.find('img')
        if image:
            image_src = image.get('src')
return jsonify({'tweet': tweet_text, 'img':image_src}), 200
```

Figura 22. Fragmento de código haciendo uso de la librería BeautifulSoup.

5.1.2 Aplicación Next.js

Tras la primera implementación del servicio Flask, el siguiente paso fue el desarrollo de la aplicación Next.js. Esta aplicación representa la capa de presentación en el cliente y a su vez implementa la lógica de negocio encargada de la recogida de datos para los reportes y la gestión de usuarios y administradores.

- **Estructura**

La estructura de carpetas está definida según la convención establecida por el propio framework, utilizando la carpeta *app* introducida en las versiones más recientes para definir la navegación entre vistas. A continuación, se describe en detalle la organización de carpetas.

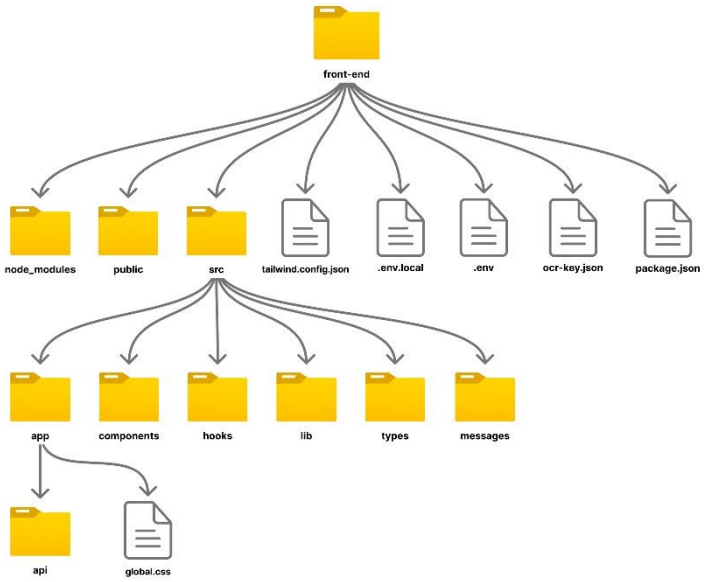


Figura 23. Estructura de carpetas en Next.js.

tailwind.config.json: Es el archivo de configuración de tailwind donde se pueden personalizar y extender los estilos por defecto como colores o tamaños.

.env.local: En este archivo se definen las variables de entorno que vayan a ser expuestas al cliente(públicas).

.env: En este archivo, sin embargo, se definen las variables de entorno que se utilizarán exclusivamente desde el servidor, y que, por tanto, no serán públicas, como las API keys o la firma de JWT.

ocr-key.json: Este archivo es la clave de los servicios de Google Cloud Platform.

package.json: Se encuentran definidas las dependencias de Node y otras definiciones del proyecto como los comandos de ejecución, el nombre y la versión.

Carpeta public: En esta carpeta se almacenan archivos estáticos que se sirven directamente al cliente sin ningún tipo de procesamiento adicional. Es el lugar ideal para recursos como el logo de la aplicación, iconos, imágenes, fuentes u otros archivos que deben ser accesibles públicamente.

Carpeta src: En esta carpeta se encuentra todo el código fuente de la aplicación. Su contenido, a su vez, se estructura de la siguiente forma:

- **Carpeta app:** Esta carpeta organiza las vistas y rutas de la aplicación, siguiendo el sistema de enrutamiento automático descrito previamente en el *apartado 2.1.*

Cada subcarpeta representa una ruta, y el archivo page.js o page.tsx define el componente que se renderiza en esa URL. En este proyecto, la carpeta app actúa como punto de entrada de la interfaz.

En la *Figura 24* se muestra la estructura principal de la vista correspondiente a la ruta /report, donde los usuarios pueden iniciar el proceso de creación de un reporte. Su código ha sido refactorizado siguiendo buenas prácticas, extrayendo la lógica de negocio en hooks personalizados (useReport) y componentes reutilizables.

El componente utiliza una animación de entrada (FadeIn) y un cargador a pantalla completa (FullScreenLoader) mientras se procesan datos. A continuación, se presenta un título descriptivo y una interfaz basada en pestañas (Tabs), que permite al usuario elegir entre diferentes tipos de medios para generar un reporte (texto, imagen, o enlace)

```jsx
export default function Report() {

/**
Extracción de datos como las variables loading
y funciones setView del hook useReport
**/

  return (
    <FadeIn duration={0.5}>
      {loading && <FullScreenLoader />}

      <div className="w-full grid place-items-center">
        <h1 className="text-3xl font-bold mt-20 text-center">
          Create a report from this different media
        </h1>
        <div className="w-full max-w-7xl mt-10">
          <Tabs
            tabs={tabs}
            loading={loading}
            setView={setView}
            setContext={setContext}
            setContent={setContent}
            setUrl={setUrl}
            setSource={setSource}
          />
        </div>
      </div>
    </FadeIn>
  );
}
```

Figura 24. Fragmento de código de la vista principal.

Esta separación entre lógica y presentación facilita el mantenimiento, la reutilización de componentes y la escalabilidad de la aplicación.

A su vez, en esta misma carpeta se definen las rutas del servidor de Next.js dentro de la subcarpeta *api*, que permite implementar lógica de servidor directamente en el mismo proyecto, sin necesidad de un back-end separado. Estas rutas actúan como endpoints API que pueden recibir y procesar peticiones HTTP GET, POST, etc.

En el contexto de este proyecto, se utilizaron para tareas como el análisis de discurso mediante un modelo LLM, la generación de documentos PDF o la gestión de usuarios y reportes. Esta funcionalidad, conocida como API Routes, facilita la integración completa de front-end y back-end en un único entorno de desarrollo, simplificando el despliegue y mantenimiento.

Un ejemplo representativo de esta funcionalidad es el endpoint, ubicado dentro de la carpeta *app/api/resolve/route.ts,* mostrado a continuación:

Esta ruta responde a peticiones HTTP de tipo *POST* y permite actualizar el estado de una denuncia en el sistema, así como registrar la resolución correspondiente y notificar por correo electrónico al usuario implicado.
Este endpoint realiza las siguientes tareas:

- Extrae los datos necesarios de la petición (*resolution, reportId, state, email*).
Valida la sesión y extrae el token de autenticación *con getServerSession* y *getToken*, necesarios para mantener la seguridad de la operación.

- Realiza una petición *PUT* al back-end de Flask, actualizando la denuncia correspondiente con los datos proporcionados. Si la respuesta es exitosa, se procede al envío de un correo electrónico notificando el resultado al usuario.

```
export async function POST(req: NextRequest) {
    try {

        const { resolution, reportId, state, email } = await req.json();
        const session = await getServerSession(authOptions);
        const token = await getToken({ req })

        /**
        *
        Comprobaciones de sesión y de datos de entrada
        *
        **/

        const headers = { Authorization: `Bearer ${token?.accessToken}`, "X-Provider":
        session?.provider, "Content-Type": "application/json" }
        const response = await
        axios.put(`${process.env.NEXT_PUBLIC_FLASK_API_URL}/reports/${reportId}`, {
        resolution: resolution, state: state }, { headers: headers });

        if (response instanceof Error) {
            return NextResponse.json({ error: response.message }, { status: 400 });
        } else {
            if (response.status === 200) {

        /**
        *
        Envío del mail
        *
        **/

            return NextResponse.json(response.data, { status: 200 });
        }

    } catch (error: unknown) {
        if (error instanceof Error) {
            console.error(`Error: ${error.message}`);
        } else {
            console.error('Unexpected error', error);
        }
        return NextResponse.json({ error: error }, { status: 500 })
    }
}
```

Figura 25. Fragmento de código de una API Route de Next.js.

Por último, en esta carpeta se almacena el archivo *global.css* que define los estilos globales de la aplicación.

- **Carpeta components**: En esta carpeta se encuentran definidos todos los componentes reutilizables de la aplicación. Estos componentes incluyen tanto los desarrollados específicamente para el proyecto (extraídos durante la refactorización del código), como aquellos provenientes de librerías externas de UI. Su organización permite mantener un código limpio, modular y escalable, facilitando la reutilización y consistencia visual en toda la plataforma.

Debido al uso de React como base del framework, se ha aprovechado el ecosistema existente para incorporar librerías de componentes que ofrecen soluciones estéticas y funcionales ya construidas. Entre las principales utilizadas destacan:

- Shadcn/ui: Librería moderna y minimalista que ofrece una amplia gama de componentes estilizados, fácilmente personalizables y coherentes visualmente entre ellos. Se pueden encontrar en Build your Component Library - shadcn/ui.

- React Bits: Proporciona componentes visuales con animaciones y efectos interactivos, como textos animados o transiciones dinámicas, que ofrecen una experiencia de usuario atractiva. Su web oficial, React Bits - Animated UI Components For React.

- Aceternity UI: Incluye una gran variedad de componentes preconstruidos y personalizables, útiles para incorporar rápidamente funcionalidades complejas con un diseño atractivo. Extraídos de Components - Aceternity UI.

Estas bibliotecas se integran fácilmente con TailwindCSS, lo que ha permitido mantener una coherencia en el estilo y acelerar el desarrollo visual de la interfaz. A su vez, se encuentran disponibles nativamente tanto en JavaScript como en TypeScript.

- **Carpeta hooks**: Esta carpeta contiene los hooks personalizados desarrollados para el proyecto. Los hooks permiten extraer y reutilizar la lógica de los componentes, manteniendo así el principio de separación de responsabilidades y mejorando la organización del código.

  Mediante esta estrategia, funcionalidades como la gestión de estado, llamadas a la API, validación de datos o control de formularios se encapsulan en funciones reutilizables, que pueden ser fácilmente invocadas desde distintos componentes sin repetir código.

La *Figura 26* representa una versión simplificada del hook personalizado useReport, encargado de gestionar el estado y la lógica del formulario de denuncias. Incluye funcionalidades clave como el manejo de sesión, la integración con *Google reCAPTCHA* para prevenir envíos automatizados, y la función *handleAnalyzePost*, que prepara y envía los datos del formulario a un endpoint del servidor. En caso de éxito, se notifica al usuario mediante una alerta. El hook devuelve tanto variables de estado como funciones, lo que permite mantener el componente principal limpio y modular.

```
export function useReport() {

  // Sesión
  const { data: session } = useSession();

  // Campos del formulario
  const [url, setUrl] = useState("");

  /*
   * Otros campos como el content, context, etc.
  */

  // Recaptcha
  const recaptchaRef = useRef<ReCAPTCHA>(null);
  const [captchaToken, setCaptchaToken] = useState("");

  // Función que gestiona el análisis de post de X
  const handleAnalyzePost = async (e: React.FormEvent) => {

      const formData = new FormData();
      formData.append("captchaToken", captchaToken)
      formData.append("url", url);
      /*
      * Otros campos como el content, context, etc.
      */
      const res = await axios.post("/api/analyze", formData);
      if (res.status === 200) {

          // Alerta de éxito
  Swal.fire({
          title: 'Email sent!',
          text: 'The analisis will be sent to your email',
          icon: 'success'
        })
      }
  };

  return {
    url,
    recaptchaRef,
    captchaToken,
    handleAnalyzePost,
    session
    /*
     * otras variables y funciones que se necesiten
    */
  };
}
```

Figura 26. Fragmento de código del hook useReport.

- **Carpeta lib**: Esta carpeta agrupa funcionalidades auxiliares que han sido extraídas del código principal y organizadas de forma modular según su

propósito específico. Entre las funciones se encuentran, por ejemplo, el envío de correos electrónicos, la generación de PDFs y análisis de imágenes, entre otras.

- **Carpeta type**: Esta carpeta contiene las definiciones de tipos personalizados utilizados a lo largo de la aplicación, fundamentales al estar desarrollada en TypeScript. Permite declarar estructuras de datos estrictas que aportan seguridad en tiempo de compilación, mejoran la legibilidad del código y evitan errores comunes.

  Entre los tipos definidos se encuentran, por ejemplo, las interfaces del modelo de reporte, los tipos relacionados con la gestión de sesión de usuario (autenticación con NextAuth), o estructuras utilizadas en la comunicación con las APIs (como respuestas del análisis o resoluciones). En la *Figura 27* se observa la definición del tipo Report mapeando a la colección definida en el back-end.

Figura 27. Definición de tipos en TypeScript.

- **Carpeta messages**: Esta carpeta contiene los archivos de traducción en formato JSON, organizados por idioma. Cada archivo define un conjunto de claves y sus correspondientes textos localizados, los cuales serán accedidos dinámicamente desde los componentes mediante la librería i18n de Next.js. Esta estructura permite mantener separada la lógica del contenido textual, facilitando la escalabilidad y el mantenimiento de los idiomas.

- **Otros detalles de la implementación**

Autenticación

Tal como se ha explicado previamente en este documento, para la autenticación se ha utilizado la librería NextAuth.js. A continuación, se describe su integración y configuración dentro del proyecto.

Lo primero es describir la configuración de la librería, en el siguiente código se definen los servicios externos de autenticación y se especifica, a su vez, la autenticación por credenciales propios de la plataforma.

- **Proveedores externos**: Se integran Google y GitHub como métodos de autenticación mediante OAuth 2.0. Los identificadores y secretos de cliente se obtienen desde variables de entorno para garantizar la seguridad.

Figura 28. Definición de proveedores externos en NextAuth.

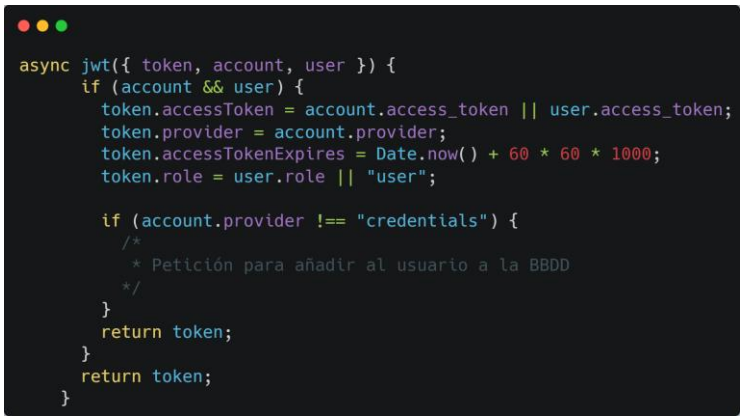
- **Autenticación por credenciales**: Se configura el proveedor CredentialsProvider, que permite a los usuarios iniciar sesión con su correo y contraseña mediante una petición a la API de Flask. En caso de credenciales incorrectas, se lanza una excepción.

```
CredentialsProvider({
      name: "Credentials",
      credentials: {
        email: {
          label: "Email",
          type: "email",
          placeholder: "user@example.com",
        },
        password: {
          label: "Password",
          type: "password",
        },
      },
      async authorize(credentials) {
        if (!credentials?.email || !credentials?.password) {
          throw new Error("Faltan credenciales");
        }

        try {
          const { data } = await axios.post(`${FLASK_API_URL}/login`, {
            email: credentials.email,
            password: credentials.password,
          });
          return data;
        } catch (error: unknown) {
          if (error instanceof Error) {
            throw new Error(error.message);
          } else {
            throw new Error("Error en la autenticación");
          }
        }
      },
    }),
```

Figura 29. Definición de proveedor con credenciales en NextAuth.

- **Callbacks personalizados:**
    - **jwt**: Se utiliza para añadir al token información adicional como el proveedor, el rol o el token de acceso. También, se registra al usuario en la base de datos si ha accedido con un proveedor externo. Cabe destacar que la duración de la sesión se encuentra limitada a una hora, ya que corresponde al tiempo de validez del token proporcionado por Google.



```
async jwt({ token, account, user }) {
      if (account && user) {
        token.accessToken = account.access_token || user.access_token;
        token.provider = account.provider;
        token.accessTokenExpires = Date.now() + 60 * 60 * 1000;
        token.role = user.role || "user";

        if (account.provider !== "credentials") {
          /*
           * Petición para añadir al usuario a la BBDD
          */
        }
        return token;
      }
      return token;
    }
```

Figura 30. Definición del callback jwt.

- **session**: Permite extender el objeto de sesión disponible en el cliente, incluyendo campos como el ID del usuario, su rol y el proveedor utilizado.

```
async session({ session, token }) {
      session.role = token.role as string;
      session.user.id = token.sub as string;
      session.provider = token.provider as string;
      return session;
    },
```

Figura 31. Definición del callback session.

- **signIn**: Verifica si el usuario tiene permitido iniciar sesión (por ejemplo, si ha sido bloqueado desde la administración).

```
async signIn({user}) {
      const email = user.email as string;
      console.log(`EMAIL: ${email}`)
      const response = await
      axios.get(`${process.env.NEXT_PUBLIC_FLASK_API_URL}/users/email/${email}`);
      const { success } = response.data;
      const isAllowedToSignIn = success !== "User banned";

      return isAllowedToSignIn;
}
```

Figura 32. Definición del callback signIn.

Con esta configuración ya se puede hacer uso de los métodos *SignIn* y *SignOut*, así como de otras funciones para recuperar la sesión y token (este se recuperará exclusivamente en el servidor para no exponer el *accessToken*).

Como se hizo con el servicio Flask, se deben proteger las API Routes y de manera similar se implementa la siguiente función de la *Figura 33* que se comporta como middleware, comprobando, además, la caducidad del token.

```
export default async function verifySession(session: Session | null, token: (JWTType & {
exp: number }) | null) {

  if (!session || !token || (token.accessTokenExpires as number) < Date.now()) {
        console.error("Session expired or invalid token");
        return { verification: false, error: "Session expired or invalid token" };
    }
    const verification = await verifyAcessToken(session.provider, token.accessToken as
string);
    if (verification !== "success") {
        console.error("Invalid access token");
        return { verification: false, error: "Invalid access token" };
    }
    return { verification: true, error: "" };
}
```

Figura 33. Definición de la función verifySession.

La función *verifyAccessToken*, comprueba en función del proveedor la validez del token de acceso.

Envío de correos electrónicos

Para las notificaciones vía mail, se ha utilizado la librería NodeMailer. Para su integración ha sido necesaria un correo electrónico, en este caso de Gmail y una clave de aplicación generada específicamente para habilitar el acceso desde servicios externos.

El funcionamiento es sencillo, primero se crea un transportador con *initTransporter*, configurado con las credenciales del remitente. Luego, la función *sendMail* permite enviar correos electrónicos personalizados, recibiendo como parámetros el destinatario, el asunto y el contenido en HTML.

```
function initTransporter() {
    return nodemailer.createTransport({
        service: "gmail",
        auth: {
            user: process.env.EMAIL_USER,
            pass: process.env.EMAIL_PASS,
        },
    });
}

export async function sendMail(to: string, subject: string, html: string) {
    const transporter = initTransporter();

    return await transporter.sendMail({
        from: `Fairplay360 <${process.env.EMAIL_USER}>`,
        to,
        subject,
        html,
    });
}
```

Figura 34. Fragmento de código para el envío de emails.

Generación de PDF

Para formalizar los reportes generados por la aplicación, se implementó una función de generación dinámica de documentos PDF utilizando la librería pdf-lib. Esta función toma como entrada los datos del reporte y genera un documento estructurado que será enviado por email.

El proceso comienza con la creación de un nuevo documento y la incorporación del logotipo de la aplicación, así como un encabezado personalizado con un mensaje de agradecimiento. Posteriormente, se añade la información del análisis, formateada y envuelta para asegurar su correcta visualización. Una vez construido el documento, este se guarda en una ruta temporal (/tmp en despliegue) para su posterior envío por correo electrónico.

```
function generateReportPdf(
  content: string,
  context: string,
  source: string,
  result: string,
  reasoning: string,
  to: string
): Promise<string> {
  const pdfDoc = await PDFDocument.create();
  let page = pdfDoc.addPage();
  const font = await pdfDoc.embedFont(StandardFonts.Helvetica);
  const fontBold = await pdfDoc.embedFont(StandardFonts.HelveticaBold);

  // Inserción del logo y encabezado
  const logoPath = path.join(process.cwd(), "public", "logo-no-bg.png");
  const logoBytes = fs.readFileSync(logoPath);
  const logoImage = await pdfDoc.embedPng(logoBytes);
  page.drawImage(logoImage, { x: 50, y: 700, width: 50, height: 50 });

  page.drawText(`Dear ${to},`, { x: 50, y: 650, size: 12, font: fontBold });

  // Escritura de campos del reporte
  const fields = [
    { label: "Content", value: content },
    { label: "Context", value: context },
    { label: "Source", value: source },
    { label: "Result", value: result },
    { label: "Reasoning", value: reasoning },
  ];

  let y = 620;
  for (const field of fields) {
    page.drawText(`${field.label}: ${field.value}`, { x: 50, y, size: 10, font });
    y -= 20;
  }

  // Guardado en ruta temporal
  const pdfBytes = await pdfDoc.save();
  const tempFilePath = path.join("/tmp", `temp_report_${Date.now()}.pdf`);
  fs.writeFileSync(tempFilePath, pdfBytes);
  return tempFilePath;
}
```

Figura 35. Fragmento de código para la creación de PDFs.

Tras haber generado el PDF y almacenarlo localmente, el siguiente paso es almacenarlo en Google Cloud Storage, como se observa en la *Figura 34*. Tras obtener la URL local con la función *generateReportPDF* se almacena en el Bucket de Google para posteriormente liberar el espacio local.

```
async function generateAndUploadReport(
  content: string,
  context: string,
  source: string,
  result: string,
  reasoning: string,
  to: string
): Promise<string> {
  try {
    // Genera el PDF y obtiene la ruta del archivo temporal.
    const pdfPath = await generateReportPdf(content, context, source, result, reasoning,
to);

    // Selecciona el bucket.
    const bucket = clientGoogle.bucket("fairplay360-reports");
    const destination = `reports/${to}/${Date.now()}.pdf`;

    // Sube el archivo al bucket.
    const [uploadedFile] = await bucket.upload(pdfPath, {
      destination,
      gzip: true, // Comprime el archivo
      metadata: {
        cacheControl: 'public, max-age=31536000',
      },
    });

    await uploadedFile.makePublic();
    const publicUrl = uploadedFile.publicUrl();

    // Elimina el archivo temporal
    fs.unlinkSync(pdfPath);

    return publicUrl;
  } catch (error) {
    console.error("Error en generateAndUploadReport:", error);
    throw error;
  }
}
```

Figura 36. Fragmento de código para el almacenamiento de PDFs.

Finalmente, como se ha visto anteriormente, lo enviamos por correo electrónico, añadiendo la URL obtenida de la función *generateAndUploadReport* como archivo adjunto.

```
const linkToPDF = await generateAndUploadReport(content, context, source, result, reasoning,
to);
    await transporter.sendMail({
      from: `Fairplay360 <${process.env.EMAIL_USER}>`,
      to,
      subject,
      html,
      attachments: [{
        filename: 'report.pdf',
        path: linkToPDF,
        contentType: 'application/pdf'
      }]
    }
```

Figura 37. Fragmento de código para el envío de PDFs por email.

Análisis de discurso de odio

Para el análisis automatizado de discurso de odio en los textos se ha utilizado la API de Deepseek con el modelo Reasoner, modelo de lenguaje de IAG. Este modelo ofrece una gran capacidad de razonamiento y precisión en la clasificación textual, para los objetivos y necesidades de este proyecto ha encajado perfectamente.

En la siguiente función, se observa cómo se realiza la configuración y petición a la API. Primero, se define con el rol de sistema un mensaje donde le asignamos un objetivo a la IA, tras esto con el rol de usuario podemos enviar el contenido y contexto a analizar. La API, por su parte, responde con dos elementos claves, el resultado del análisis (*hate speech, not hate speech*) y el razonamiento detrás de esta respuesta.

```
async function analyzeHateSpeech(content_input: string, context: string) {
    const completion = await openai.chat.completions.create({
    messages: [
        {
        role: "system",
        content: "You are an AI hate speech analyzer, return just 'hate speech' or 'not hate
        speech'."
        },
        {
        role: "user",
        content: `Analyze this text: ${content_input}. ${context ? `Some context:
        ${context}` : ''}`
        }
    ],
    model: "deepseek-reasoner"
    });
}
```

Figura 38. Prompt y función para enviar la petición a la API de Deepseek.

Análisis de imágenes

- **OCR**

En la *Figura 39* se encuentra la función que implementa OCR haciendo uso del cliente de Google Cloud Vision previamente configurado. Toma como entrada la URL de la imagen y devuelve un JSON con el idioma detectado en el texto y texto extraído.

```
export const ocr = async (url: string): Promise<JSON> => {
  return new Promise(async (resolve, reject) => {
    try {
      const [result] = await clientGoogle.textDetection(url);
      const annotations = result.textAnnotations;

      const filteredAnnotations = annotations && annotations.length > 0
        ? {
          locale: annotations[0].locale,
          description: annotations[0].description,
        }
        : {};

      resolve(filteredAnnotations as JSON);
    } catch (error) {
      console.error("Error en OCR:", error);
      reject(error);
    }
  });
}
```

Figura 39. Función para enviar una petición a la API de Google Vision.

- **Captioning**

En la *Figura 41* se encuentra la implementación de la función encargada de obtener una breve descripción de una imagen, utilizando el servicio de Azure AI Vision. En este caso, tomando como entrada de la función una URL, se envía una petición al endpoint habilitado por el servicio, este devolverá una breve descripción de la imagen enviada.

```
export const caption = async (url: string): Promise<string> => {
  try {
    const { data } = await axios.post(
      "https://fairplay360-image-caption.cognitiveservices.azure.com/vision/v3.2/describe?
maxCandidates=1",
      { url: url },
      {
        headers: {
          "Ocp-Apim-Subscription-Key": process.env.AZURE_VISION_API_KEY,
          "Content-Type": "application/json",
        },
      }
    );
    const captions = data.description.captions;
    return captions.length > 0 ? captions[0].text : "";
  } catch (error) {
    throw error;
  }
}
```

Figura 41. Función para enviar una petición a Azure Vision.

Internacionalización

Para la integración del sistema multilingüe en la plataforma, se ha utilizado la librería nativa de i18n en Next.js, la cual permite definir y gestionar traducciones de forma dinámica mediante archivos JSON separados por idioma.

En primer lugar, se deben definir las claves y textos correspondientes en archivos de traducción específicos, uno por cada idioma soportado (por ejemplo, es.json y en.json), como se muestra en la *Figura 42*.

```
//Archivo es.json
"home": {
    "title": "Introduciendo Fairplay360",
    "subtitle": "Una plataforma para denunciar y analizar discurso de odio"
}

//Archivo en.json
"home": {
    "title":"Introducing Fairplay360",
    "subtitle":"A platform to report and analyze hate speech"
```

Figura 42. Fragmento de código de los archivos de traducción de español e inglés.

Posteriormente, en los componentes de la aplicación se utiliza el hook *useTranslations* proporcionado por la propia librería, que permite acceder a los textos definidos en los archivos de idioma y renderizarlos de forma dinámica según el idioma activo. En la *Figura 43* se ejemplifica este proceso aplicado a un componente.

```
import { useTranslations } from 'next-intl';

export default function Home(){

const t = useTranslations();

return(
<>
  <h1 className="text-lg sm:text-2xl font-bold text-black mt-4">

    {t('home.title')}

  </h1>
<>
      )
  }
```

Figura 42. Fragmento de código de un componente con texto dinámico basado en el idioma.

La implementación ha sido minuciosa y sistemática, incluyendo no solo las vistas principales, sino también componentes reutilizables, hooks, dialogs, formularios y alertas.

5.2 Pruebas

Durante el desarrollo del proyecto se llevaron a cabo diversas pruebas manuales y funcionales para validar el correcto funcionamiento de los distintos módulos del sistema. Estas pruebas se agrupan en función del entorno sobre el que se realizaron.

- **Servicio Flask**

Pruebas de endpoints REST: Se validó la respuesta de los distintos endpoints de la API, comprobando el correcto manejo de peticiones GET, POST, PUT y DELETE. Se desarrolló una documentación OpenAPI que recopila estas pruebas.

Validación de la conexión con la base de datos: Se verificó la conexión y ejecución de consultas de inserción, modificación y eliminación de datos en MongoDB Atlas.

ScraperAPI: Se comprobó el funcionamiento de la extracción del contenido deseado de las publicaciones de X.

- **Aplicación Next.js**

Pruebas de formularios: Se comprobaron la validación de campos obligatorios, el flujo de errores, los mensajes de retroalimentación y la prevención de envíos duplicados.

Test de Flujo de usuario: Se simuló el proceso completo de creación de un reporte desde la interfaz (login, análisis, generación y seguimiento por correo).

Protección de rutas y permisos: Se comprobó que las rutas protegidas no fueran accesibles sin autenticación (NextAuth) y que los roles determinaran el acceso (por ejemplo, panel de administrador).

Pruebas de integración con APIs externas:

- **Google Vision OCR y Azure AI Vision**: Se probaron imágenes reales para verificar la calidad del texto y las descripciones generadas.

- **Deepseek API**: Se testearon distintos ejemplos de texto para observar el comportamiento del modelo y la consistencia en la clasificación de discurso de odio. Esto permitió refinar el prompt utilizado.

- **Generación y envío de informes**: Se verificó la correcta creación del PDF con pdf-lib, su almacenamiento en Google Cloud Storage y el posterior envío mediante correo electrónico con NodeMailer.

➢ **Herramientas utilizadas**

Postman: Para probar manualmente los endpoints de la API Flask, validando respuestas esperadas, manejo de errores y cabeceras de autenticación.

Vercel Logs: Herramienta ofrecida por la plataforma de despliegue que permite consultar en tiempo real los registros de ejecución tanto del frontend como de las API routes.

Google Cloud Logging: Utilizado para monitorizar el comportamiento de servicios como OCR y almacenamiento en GCS, permitiendo detectar errores y medir tiempos de respuesta.

DevTools del navegador: Fundamental para probar la lógica de los formularios, inspeccionar errores de red, revisar tokens, respuestas de la API y estado de la sesión.

Herramientas propias de desarrollo de Next.js: Visualización de rutas, errores de compilación, hot reload, integración con TypeScript y ESLint.

6

Conclusiones y trabajos futuros

6.1 Objetivos cumplidos

Durante el desarrollo de este Trabajo de Fin de Grado se ha logrado cumplir satisfactoriamente el objetivo principal: diseñar e implementar una plataforma web que permita detectar, reportar y gestionar posibles casos de discurso de odio presentes en redes sociales, integrando para ello inteligencia artificial y servicios en la nube.
Entre los logros específicos destacan:

- Desarrollo de una aplicación full-stack completa con back-end (Flask y Next.js) y front-end (Next.js).
- Integración de un sistema de análisis semántico con modelos LLM (Deepseek).

- Procesamiento automático de imágenes mediante OCR (Google Vision) y image captioning (Azure).
- Generación y envío de informes en PDF mediante NodeMailer y pdf-lib.
- Almacenamiento en la nube (Google Cloud Storage) y despliegue en Vercel.
- Implementación de control de acceso con autenticación OAuth y sistema de gestión de usuarios y administradores.
- Diseño de una interfaz usable, minimalista y multilingüe, fiel a los prototipos definidos.

Estos avances permiten disponer de una solución funcional y escalable, válida como prueba de concepto y base sólida para proyectos futuros relacionados con la moderación de contenido digital.

6.2 Dificultades encontradas

A lo largo del desarrollo del proyecto se han presentado distintos desafíos técnicos que se muestran a continuación:

- **Integración de múltiples servicios externos**: la conexión simultánea con APIs como Deepseek, Azure, Google Cloud y ScraperAPI supuso una complejidad adicional, especialmente en la gestión de errores, tiempos de espera y control de credenciales (los tokens generados por Google tienen una validez de una hora, teniendo que limitar la validez de la sesión en consecuencia).

- **Límites de acceso a recursos gratuitos**: algunos servicios cloud tienen restricciones (número de peticiones gratuitas, almacenamiento, etc.) que condicionaron las pruebas y obligaron a realizar ajustes de optimización.

- **Dificultades en el scraping de X**: debido a la generación y renderización dinámica del contenido de las publicaciones fue necesario depender de Scraper API para su correcta extracción en lugar de optar por una solución local.

- **Autenticación y control de sesión**: coordinar el sistema de autenticación OAuth en NextAuth con el back-end Flask y la base de datos requirió una configuración minuciosa para mantener la coherencia entre proveedores y sesiones.

A pesar de estas dificultades, todas pudieron ser resueltas con soluciones prácticas y documentadas, lo cual ha enriquecido notablemente el aprendizaje obtenido.

6.3 Posibles ampliaciones

Existen varias líneas de trabajo que permitirían mejorar y extender esta plataforma en futuras versiones:

- **Entrenamiento de un modelo propio especializado**: actualmente, la detección de discurso de odio se apoya en el modelo de lenguaje Deepseek Reasoner, que, si bien ofrece resultados aceptables, no está adaptado a las particularidades del lenguaje futbolístico ni a los matices culturales del entorno deportivo. Una mejora clara sería la recopilación de un corpus propio multilingüe sobre el cual entrenar un modelo propio basado en arquitecturas modernas como LLaMA, BERT o RoBERTa.

- **Versión móvil nativa multiplataforma**: aunque la aplicación web es responsive y se adapta a dispositivos móviles, desarrollar una versión nativa mediante frameworks como React Native o Flutter permitiría mejorar la experiencia de usuario y ofrecer funcionalidades exclusivas como el acceso directo a la cámara, notificaciones push, almacenamiento local de reportes, acceso biométrico, etc.

- **Implementación de moderación proactiva en redes sociales**: actualmente, el sistema opera de forma reactiva, es decir, analiza contenido tras ser enviado por un usuario. Una evolución lógica sería implementar una arquitectura de detección proactiva y en tiempo real, integrando APIs oficiales de redes como X,

Instagram o Reddit. De esta forma, la plataforma podría monitorizar automáticamente publicaciones en determinados hashtags, cuentas o temas y detectar contenido ofensivo sin intervención humana.

Estas ampliaciones permitirían convertir Fairplay360 en una herramienta aún más robusta, orientada tanto a usuarios individuales como a instituciones deportivas o educativas.

Referencias

Figuras

- U.S. Department of Justice. (2024). *Hate crime statistics 2023*. https://www.justice.gov/hatecrimes/hate-crime-statistics *(Figura 1)*
- Vercel. (2025). *Layouts and pages – Next.js Documentation*. https://nextjs.org/docs/app/getting-started/layouts-and-pages *(Figura 2)*
- Imperva. (s.f.). *Lazy loading*. https://www.imperva.com/learn/performance/lazy-loading/ *(Figura 3)*
- DeepSeek. (s.f.). *DeepSeek API Documentation*. https://api-docs.deepseek.com/news/news250120 *(Figuras 4 y 5)*
- Azure Cognitive Services. (s.f.). *Image captioning demo*. https://portal.vision.cognitive.azure.com/demo/image-captioning *(Figura 6)*

Documentación técnica y bibliografía consultada

- Aceternity UI. (s.f.). *Aceternity UI Components*. https://ui.aceternity.com/
- Azure. (s.f.). *Azure AI Vision documentation*. https://learn.microsoft.com/en-us/azure/ai-services/computer-vision/
- DeepSeek. (s.f.). *DeepSeek API documentation*. https://api-docs.deepseek.com/
- Flask. (s.f.). *Flask documentation*. https://flask.palletsprojects.com/
- Google Cloud. (s.f.). *Cloud Vision API*.

https://cloud.google.com/vision/docs

- Google Cloud. (s.f.). *Cloud Storage documentation*. https://cloud.google.com/storage/docs
- MongoDB, Inc. (s.f.). *MongoDB Atlas documentation*. https://www.mongodb.com/docs/atlas/
- MongoDB, Inc. (s.f.). *MongoDB manual*. https://www.mongodb.com/docs/manual/
- MongoEngine. (s.f.). *MongoEngine documentation*. https://docs.mongoengine.org/
- NextAuth.js. (s.f.). *NextAuth.js documentation*. https://next-auth.js.org/
- Next.js. (2025). *Next.js documentation*. https://nextjs.org/docs
- Nodemailer. (s.f.). *Nodemailer*. https://nodemailer.com/about/
- OpenAI. (s.f.). *openai – npm*. https://www.npmjs.com/package/openai
- pdf-lib. (s.f.). *pdf-lib*. https://pdf-lib.js.org/
- React Bits. (s.f.). *React Bits - Animated UI Components*. https://reactbits.dev/
- Shadcn UI. (s.f.). *Build your component library*. https://ui.shadcn.com/
- Tailwind Labs. (2024). *Tailwind CSS documentation*. https://tailwindcss.com/docs

Apéndice A

Manual de Instalación

A continuación, se describen los requisitos previos y pasos para instalar y ejecutar la aplicación web Fairplay360.

Requisitos previos

Antes de comenzar, es necesario tener instalado el siguiente software:

- **Node.js** (v18 o superior): entorno de ejecución para aplicaciones basadas en JavaScript.
- **npm**: gestor de paquetes de Node.js.
- **Python** (v3.10 o superior): lenguaje de programación utilizado para el back-end.
- **pip**: gestor de paquetes de Python.
- **Git**: sistema de control de versiones para clonar el repositorio.

Clonar el repositorio

Desde la terminal se debe ejecutar el siguiente comando:

```
git clone https://github.com/pSenciales/TFG
cd <ruta-repositorio>
```

Ejecución servicio Flask

1. Acceso al directorio del proyecto

Desde la terminal, desplácese a la carpeta del back-end:

```
cd ./code/back-end
```

2. Instalación de dependencias

Instale las librerías necesarias utilizando el siguiente comando:

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

3. Configuración del entorno

Cree un archivo llamado .env en la raíz del back-end (./code/back-end/.env) con las siguientes variables de entorno:

```
MONGO_URI= <URI de conexión a MongoDB Atlas>
JWT_SECRET= <clave secreta para la firma de tokens JWT>
SCRAPFLY_API_KEY= <clave de la API de ScraperAPI>
RECAPTCHA_SECRET= <clave secreta de Google Recaptcha>
```

4. Ejecución de la aplicación

Una vez configurado todo lo anterior, ejecute el servidor Flask ejecutando:

```
python index.py
```

Por defecto, la API se expondrá en http://localhost:5000, lista para recibir peticiones desde el cliente.

Ejecución aplicación Next.js

1. Acceder al directorio del proyecto

Desde la terminal, acceda a la carpeta del front-end:

```
cd ../code/front-end/tfg
```

2. Instalación de dependencias

Ejecute el siguiente comando para instalar todas las dependencias necesarias:

```
npm install
```

3. Configuración del entorno

Se deben crear dos archivos de variables de entorno en la raíz del proyecto:

.env.local

Este archivo incluye variables que pueden ser expuestas al navegador:

```
NEXT_PUBLIC_FLASK_API_URL=http://localhost:5000
NEXT_PUBLIC_RECAPTCHA= <clave pública de Google Recaptcha>
NEXT_PUBLIC_BASE_URL=http://localhost:3000
```

.env

Este archivo contiene variables sensibles y de uso exclusivo del servidor:

```
NEXTAUTH_URL=http://localhost:3000
GOOGLE_ID= <ID de cliente OAuth de Google>
GOOGLE_SECRET= <Secreto de cliente de Google>
GITHUB_ID= <ID de cliente OAuth de GitHub>
GITHUB_SECRET= <Secreto de cliente de GitHub>
NEXTAUTH_SECRET= <Secreto para la sesión JWT>

EMAIL_USER= <correo Gmail para envío de notificaciones>
EMAIL_PASS= <clave de aplicación generada para Gmail>

JWT_SECRET= <clave secreta para verificar JWT>
RECAPTCHA_SECRET= <clave privada de reCAPTCHA>
CLOUDINARY_API_SECRET= <secreto de Cloudinary>
CLOUDINARY_API_KEY= <API key de Cloudinary>
CLOUDINARY_CLOUD_NAME= <nombre del cloud de Cloudinary>

GOOGLE_APPLICATION_CREDENTIALS=./ocr-key.json
DEEPSEEK_API_KEY= <tu API Key de Deepseek>
AZURE_VISION_API_KEY= <tu API Key de Azure Vision>
```

Nota: Se requiere tener un archivo ocr-key.json en la raíz del proyecto. Este archivo contiene las credenciales necesarias para usar el servicio Google Cloud Vision y debe ser generado desde la consola de Google Cloud.

4. Ejecutar la aplicación

Una vez configurado todo correctamente, inicie la aplicación con:

```
npm run dev
```

Por defecto, la aplicación estará disponible en http://localhost:3000.

Acceso a la web desplegada

El proyecto se encuentra desplegado en Vercel para poder acceder sin necesidad de instalaciones previas, Fairplay360.

Apéndice B

Manual de Usuario

Este manual de usuario tiene como objetivo explorar todas las funcionalidades que ofrece la aplicación web Fairplay360, para que así los usuarios y administradores puedan hacer un uso adecuado y eficiente. Nótese que las capturas expuestas están hechas desde un dispositivo móvil.

Registro de usuarios

Para iniciar el registro de un usuario, se debe hacer clic en el botón de 'Sign up' (*Figura B.1), lo que llevará al proceso de registro.*

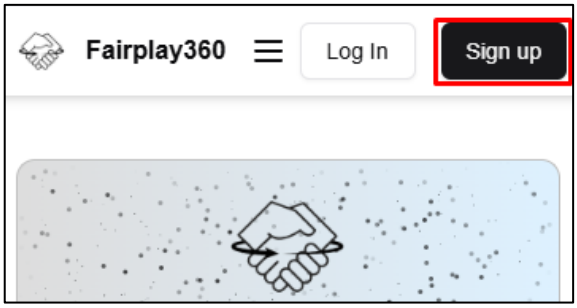


Figura B.1

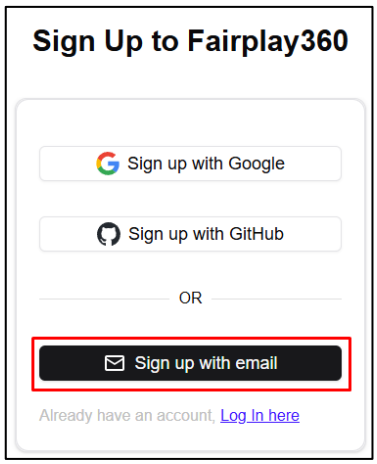


Figura B.2

Para continuar con el registro, se selecciona cualquiera de las opciones propuestas, tanto la de Google como Github no requieren explicaciones adicionales. Por tanto, se explica el proceso de registro mediante correo electrónico haciendo clic en 'Sign up with email' (*Figura B.2*).

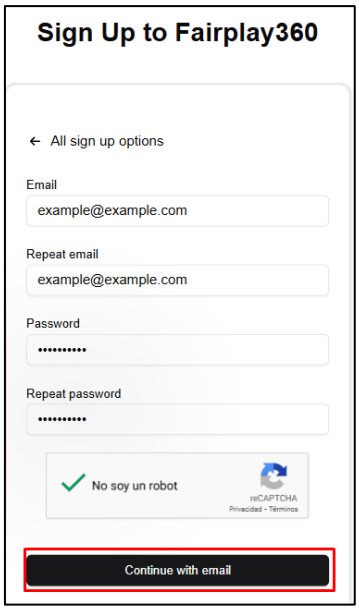


Figura B.3

Se continúa rellenando todos los campos necesarios para el registro, además de marcar el Captcha. Nótese que si algún campo no es correcto o no se ha rellenado el Captcha no se podrá continuar con el registro. Cuando todos los campos sean correctos se debe hacer clic en 'Continue with email' (*Figura B.3*).

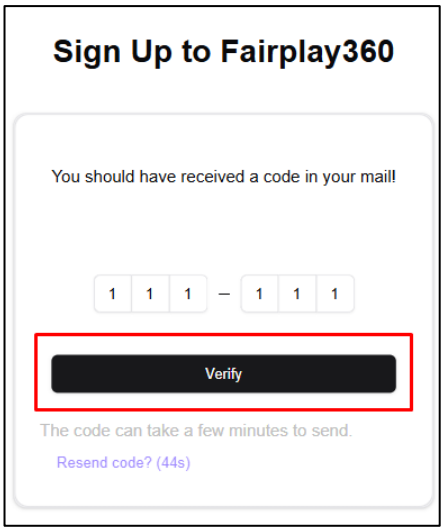


Figura B.4

Tras el paso anterior el usuario debe de haber recibido un email con un código de un solo uso (OTP) para así confirmar su correo electrónico, tras introducirlo se debe hacer clic en 'Verify' (*Figura B.4*).

Para completar el registro se debe, finalmente, rellenar los últimos campos, entre ellos una fecha de nacimiento mayor a 16 años. Se debe clicar en 'Create account' (*Figura B.5*). Automáticamente el sistema iniciará sesión, se podrá apreciar en la esquina superior derecha donde ahora aparecerá un icono con las iniciales del nombre registrado (*Figura B.6*).

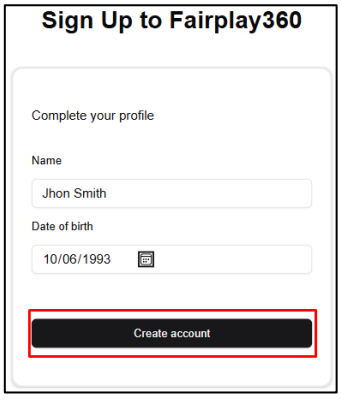


Figura B.5

Figura B.6

Inicio de sesión

Este proceso se inicia dándole clic a 'Log In' (*Figura B.7*).

Figura B.7

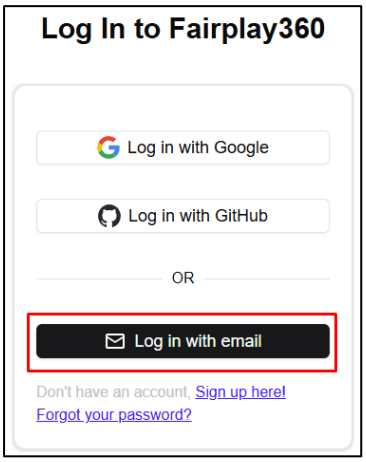


Figura B.8

Tras esto, y de manera similar al registro, se debe escoger entre las opciones ofrecidas. A continuación, se muestra el inicio de sesión a partir de correo electrónico, haciendo clic en 'Log In with email' (*Figura B.8*).

Finalmente, se deben de rellenar los campos y pulsar en 'Log In with email' (*Figura B.9)*. En caso de que las credenciales no sean correctas el sistema nos devolverá un mensaje notificándolo.

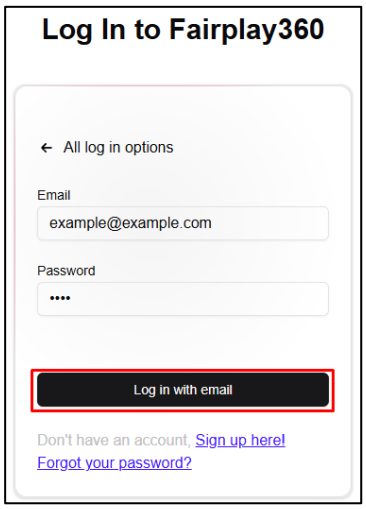


Figura B.9

Cierre de sesión

Este proceso únicamente es posible si se tiene una sesión iniciada. Se debe hacer clic en 'Log out' en el desplegable del icono del perfil (*Figura B.10*).

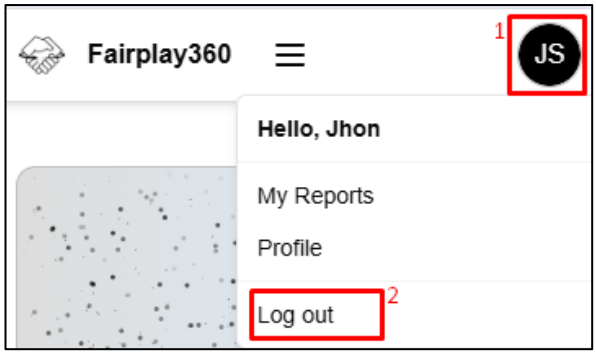


Figura B.10

Creación de una denuncia

Primero se debe hacer clic en el apartado 'Report' desde el menú hamburguesa (*Figura B.11*).

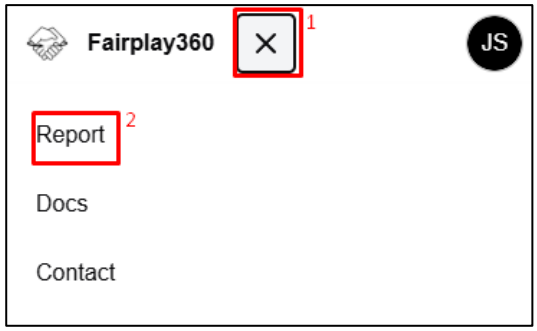


Figura B.11

Tras haber navegado a esta vista, lo siguiente es elegir el método de denuncia entre las distintas pestañas, el proceso es idéntico en todos los casos. Primero se rellenan los campos teniendo en cuenta lo siguiente, la fuente o 'source' debe tener el formato de una URL, es decir, debe tener el protocolo (http o https) y el dominio (example.com). En cuanto al archivo en el apartado 'Image', acepta todo tipo de formatos de imagen (png, jpg, jpeg, etc.).

Una vez se hayan rellenado los campos se debe hacer clic en 'Analyze' (*Figura B.12*).

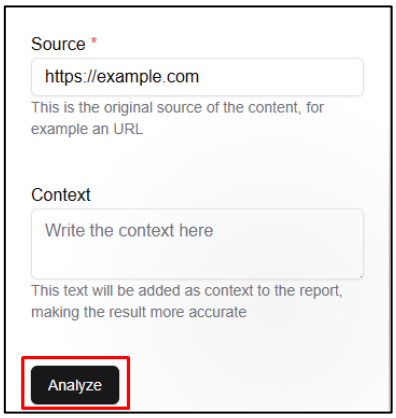


Figura B.12

En caso de no haber iniciado sesión se abrirá un dialog para añadir un correo electrónico (*Figura B.13*),

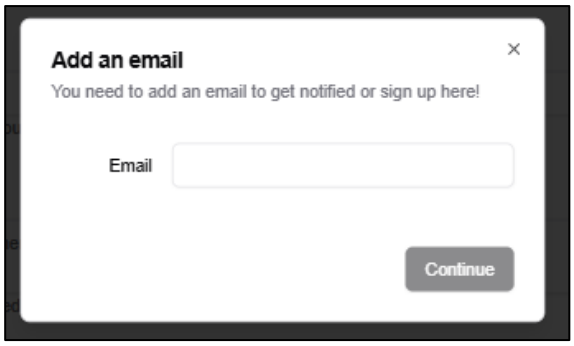


Figura B.13

Finalmente, aparecerá una pantalla de carga (*Figura B.14*) y un mensaje de éxito cuando se haya creado la denuncia (*Figura B.15*).

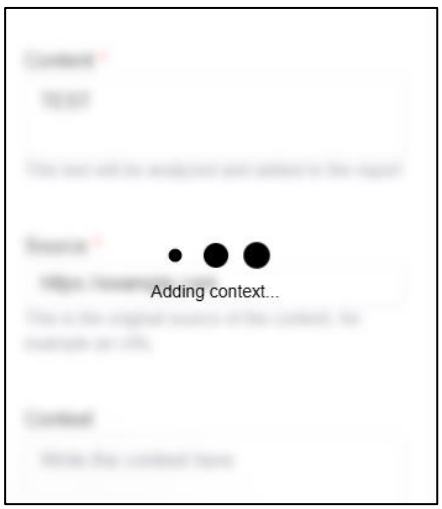


Figura B.14

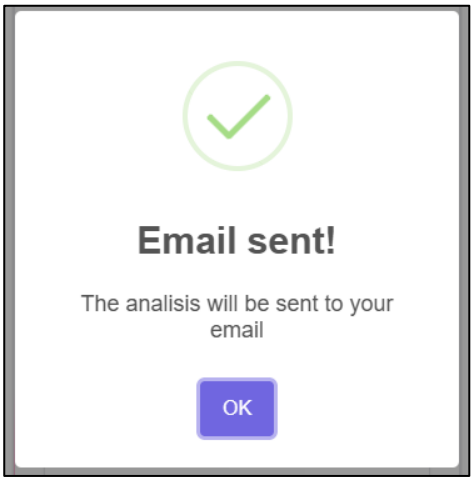
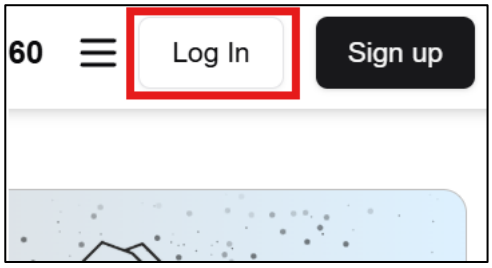


Figura B.15

Cambio de contraseña

Para poder cambiar la contraseña, se debe tener una cuenta registrada con un correo electrónico. Primero, se debe hacer clic en 'Log In' (*Figura B.16)*.



B.16

¿A continuación, se debe clicar en 'Forgot your password?' (*Figura B.17*).

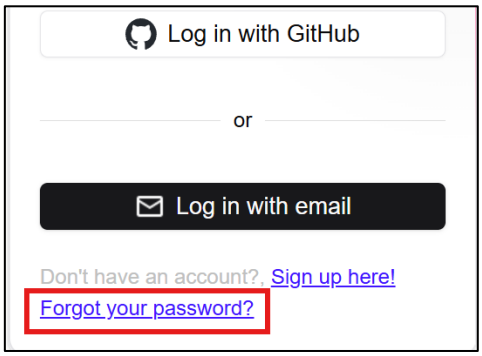


Figura B.17

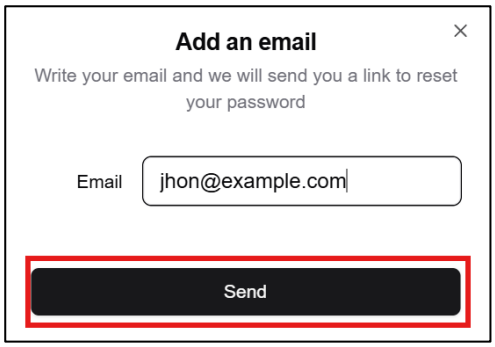


Figura B.18

Tras ellos se abrirá un dialog donde se debe añadir el email al que pertenezca la cuenta y continuar clicando el botón 'Send' (Figura B.18).

Finalmente, tras recibir el email, navegaremos hasta una página donde se deberá rellenar el formulario con la nueva contraseña, terminando el proceso al hacer clic en 'Reset password' (*Figura B.19*).

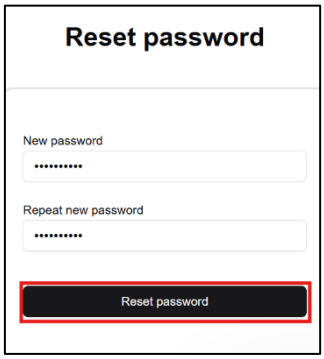


Figura B.19

Nota: Las siguientes funciones son únicamente para usuarios registrados.

Visualización de denuncias

Para acceder a las denuncias creadas se debe hacer clic en 'My reports' en el desplegable en el icono de perfil (*Figura B.20*).

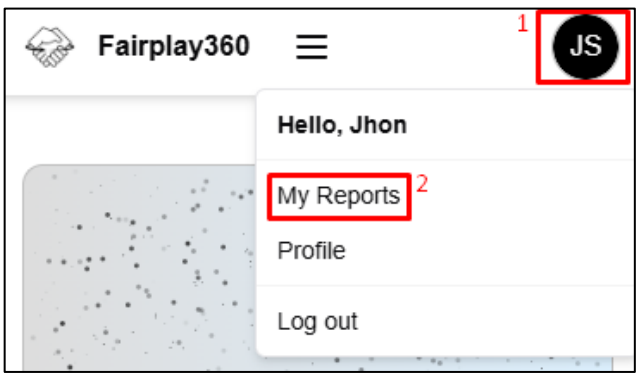


Figura B.20

Tras esto se podrán visualizar las denuncias creadas (*Figura B.21*), junto a un botón con opciones para filtrar y ordenarlas (*Figura B.22*).

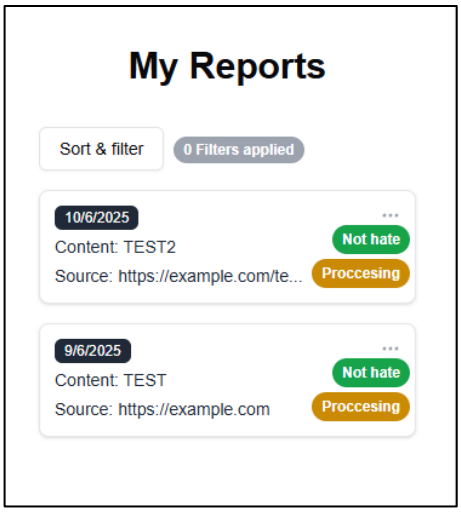


Figura B.21

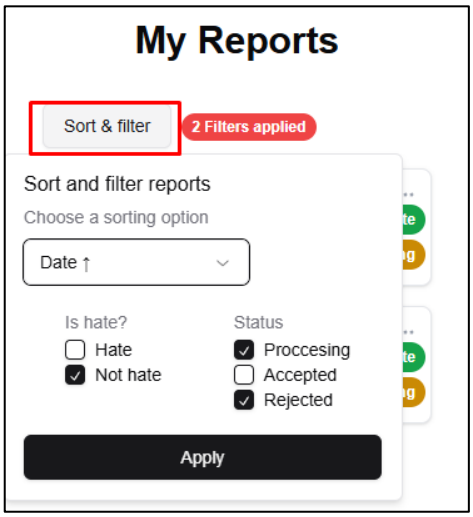


Figura B.22

También el desplegable de cada denuncia ofrece la posibilidad de abrir el PDF con el contenido o eliminar la denuncia (*Figura B.23*).

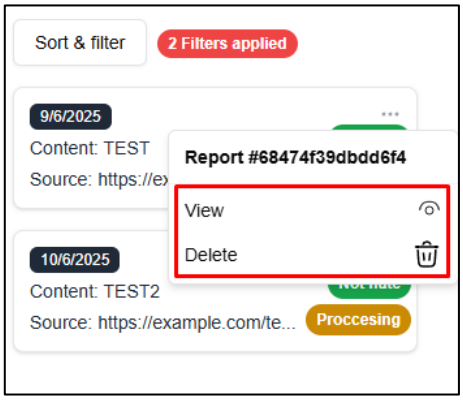


Figura B.23

Eliminar cuenta

Para eliminar una cuenta se debe hacer clic en 'Profile' en el desplegable en el icono de perfil (*Figura B.24*).

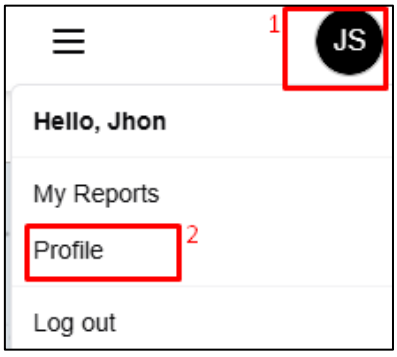


Figura B.24

En la vista del perfil se deberá hacer clic a 'DELETE ACCOUNT' (*Figura B.25*).

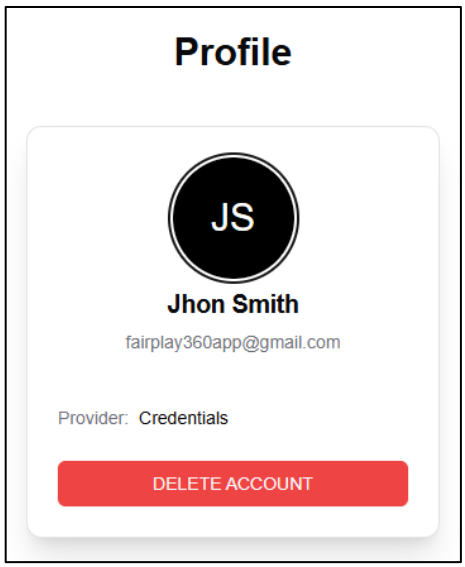


Figura B.25

Finalmente se debe escribir en el dialog la confirmación para eliminar la cuenta y hacer clic en 'Delete' (*Figura B.26*).

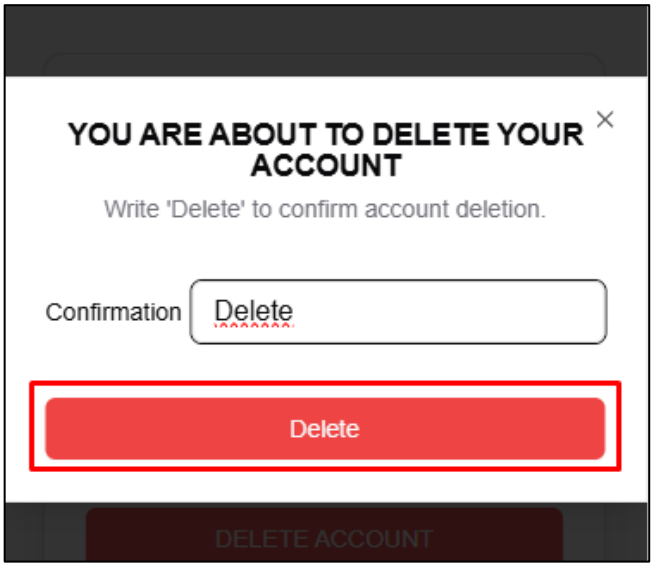


Figura B.26

Nota: Las siguientes funciones son únicamente para usuarios con el rol de administrador.

Valorar una denuncia

Para valorar una denuncia se debe hacer clic en 'Admin Portal' en el desplegable en el icono de perfil (*Figura B.27*).

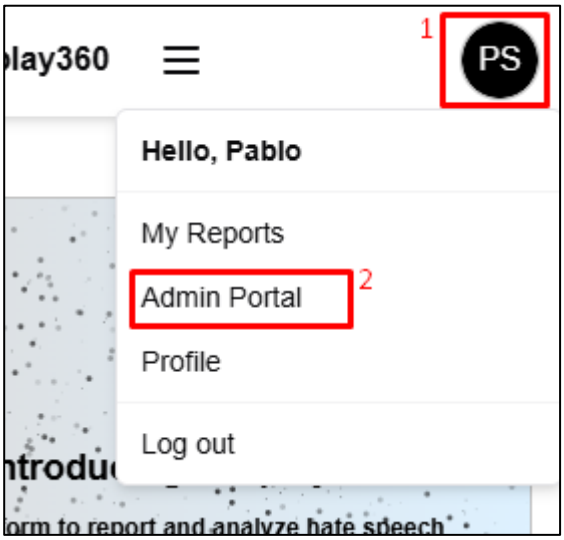


Figura B.27

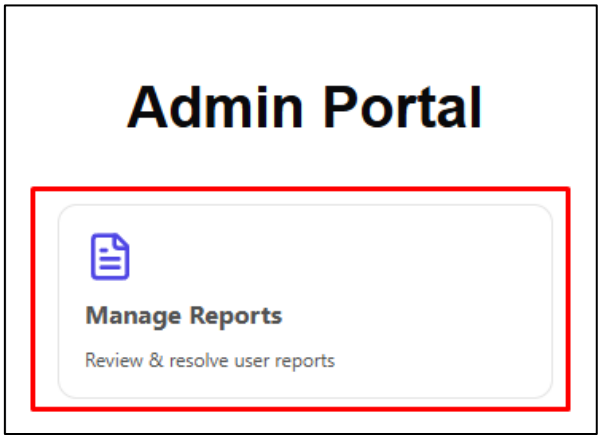


Figura B.28

A continuación, se debe hacer clic en 'Manage Reports' *(Figura B.28)* que redirigirá a la vista de las denuncias que puede gestionar el administrador.

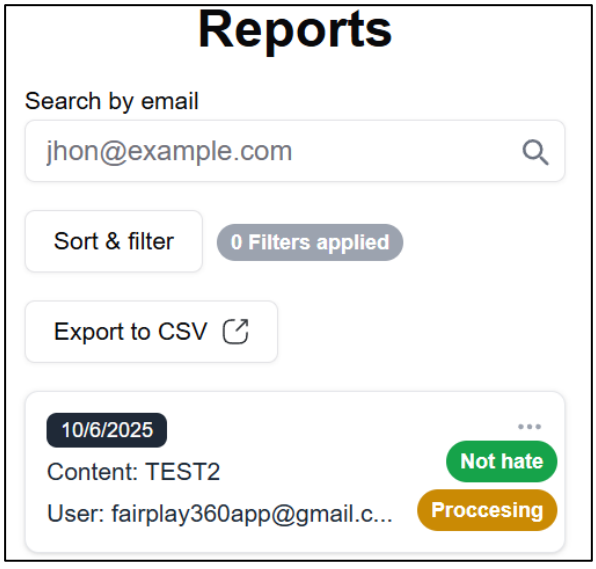


Figura B.29

La vista es similar a la de 'My Reports', incluye una barra de búsqueda por email y un botón para exportar las primeras mil denuncias con los filtros en formato CSV (*Figura B.29*).

A continuación, se debe hacer clic en el desplegable y luego en 'Resolve' (*Figura B.30*).

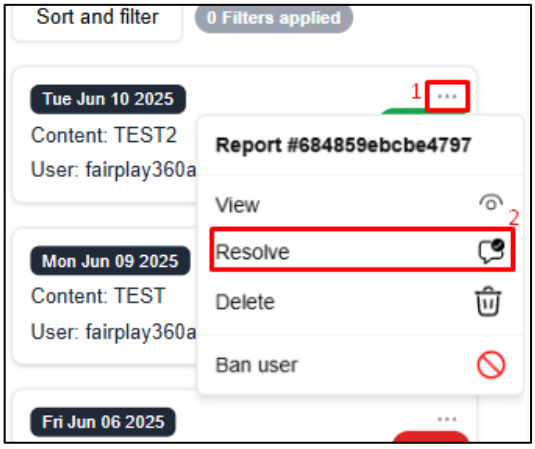


Figura B.30

Finalmente, se debe rellenar el formulario que aparece (*Figura B.31*). Automáticamente la vista se actualizará mostrando el estado nuevo de la denuncia (*Figura B.32)*, a su vez, el usuario recibirá un correo electrónico notificándole el cambio.

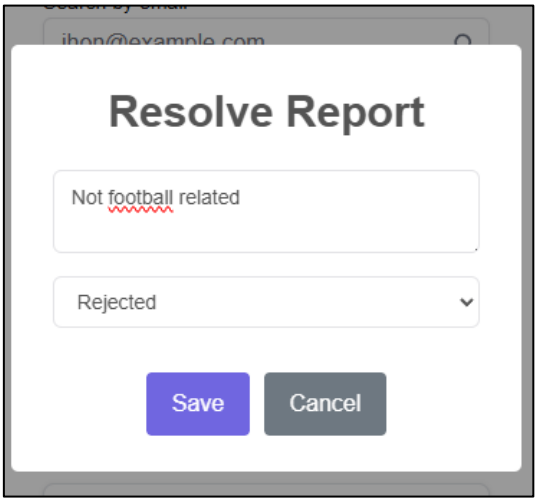


Figura B.31

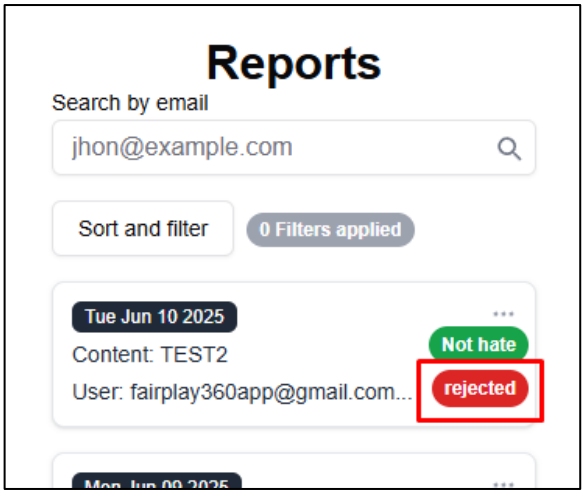


Figura B.32

Prohibir acceso a usuarios

Para valorar una denuncia se debe hacer clic en 'Admin Portal' en el desplegable en el icono de perfil (*Figura B.33*).

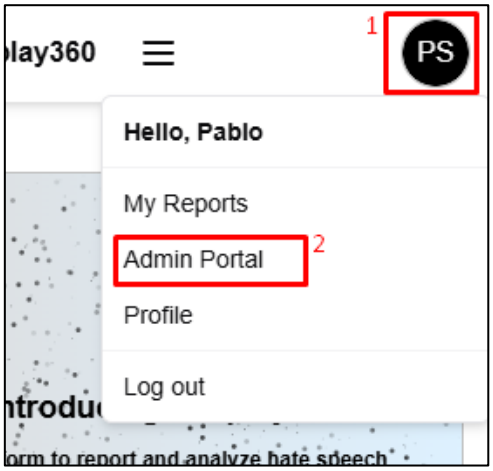


Figura B.33

A continuación, se debe hacer clic en 'Manage Reports' *(Figura B.34*) que redirigirá a la vista de las denuncias que puede gestionar el administrador.

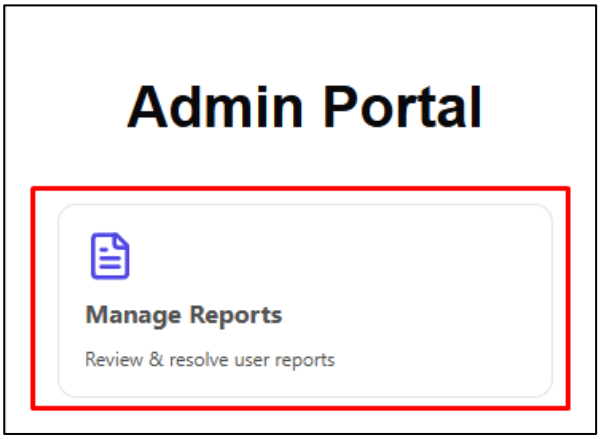


Figura B.34

A continuación, se debe hacer clic en el desplegable y luego en 'Ban user' (*Figura B.35*).

Figura B.35

Finalmente, aparecerá una alerta para confirmar la prohibición de acceso (*Figura B.36*).

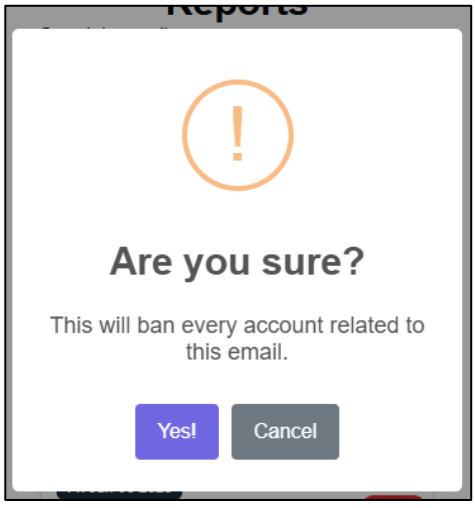


Figura B.36

Restaurar acceso a usuarios

Para valorar una denuncia se debe hacer clic en 'Admin Portal' en el desplegable en el icono de perfil (*Figura B.37*).

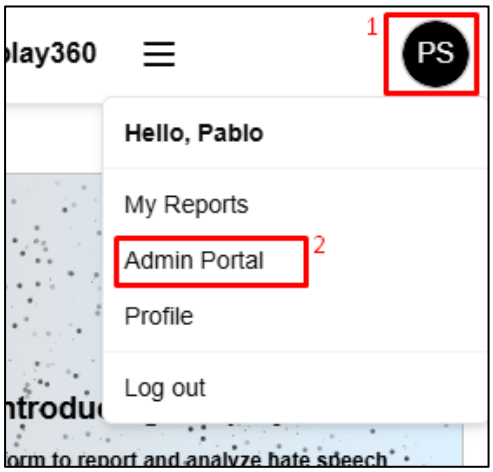


Figura B.37

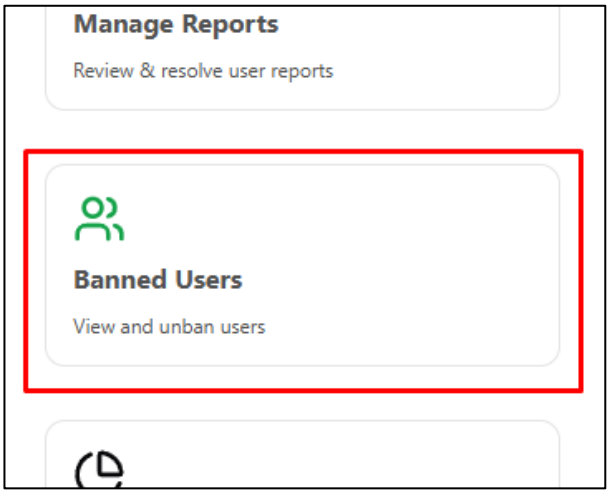


Figura B.38

A continuación, se debe hacer clic en 'Banned Users' (Figura B.38) que redirigirá a la vista de los usuarios con el acceso prohibido.

Finalmente, en el desplegable se debe hacer clic en 'Restore Access' (*Figura B.39*).

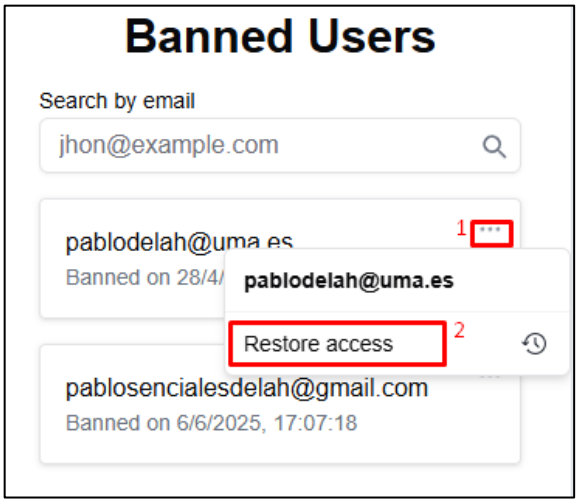


Figura B.39

Visualizar estadísticas

Para valorar una denuncia se debe hacer clic en 'Admin Portal' en el desplegable en el icono de perfil (*Figura B.40*).

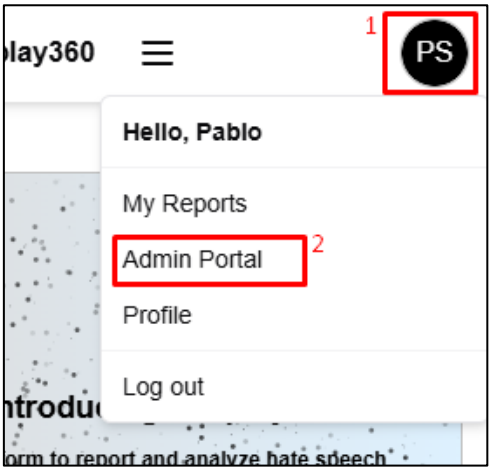


Figura B.40

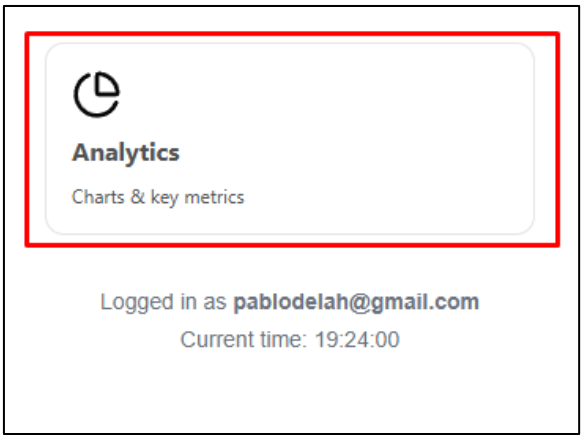


Figura B.41

A continuación, se debe hacer clic en Stats' (Figura B.41) que redirigirá a la vista de las estadísticas.

Finalmente se puede escoger el rango entre las opciones disponibles (*Figura B.42*) para la extracción de las estadísticas.

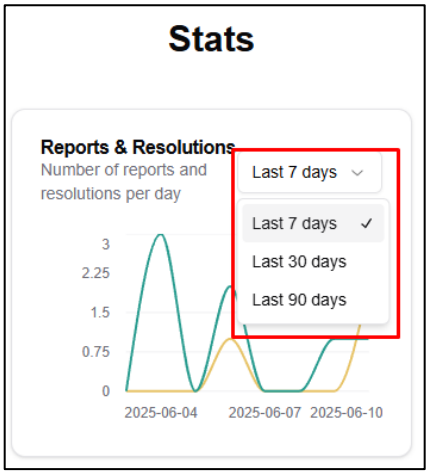


Figura B.42

Apéndice C

Estructura de Desglose de Trabajo

La Estructura de Desglose del Trabajo (EDT) es una herramienta clave en la gestión de proyectos y está recogida en el estándar del PMBOK (Project Management Body of Knowledge) del Project Management Institute (PMI). Esta estructura consiste en la descomposición jerárquica del trabajo total del proyecto en partes más pequeñas y manejables, facilitando así su planificación, ejecución, control y seguimiento.

La EDT ha sido clave para facilitar la planificación y el seguimiento del proyecto, estructurando el desarrollo en cinco fases principales: Inicio, Planificación, Ejecución, Control y Cierre. Dentro de cada una de ellas se detallan las tareas técnicas y documentales necesarias para cumplir los objetivos del proyecto, desde la investigación y análisis de requisitos hasta el despliegue, control de calidad y entrega final. A continuación, en la *Figura C.1* se presenta el EDT desarrollado para la aplicación Fairplay360.

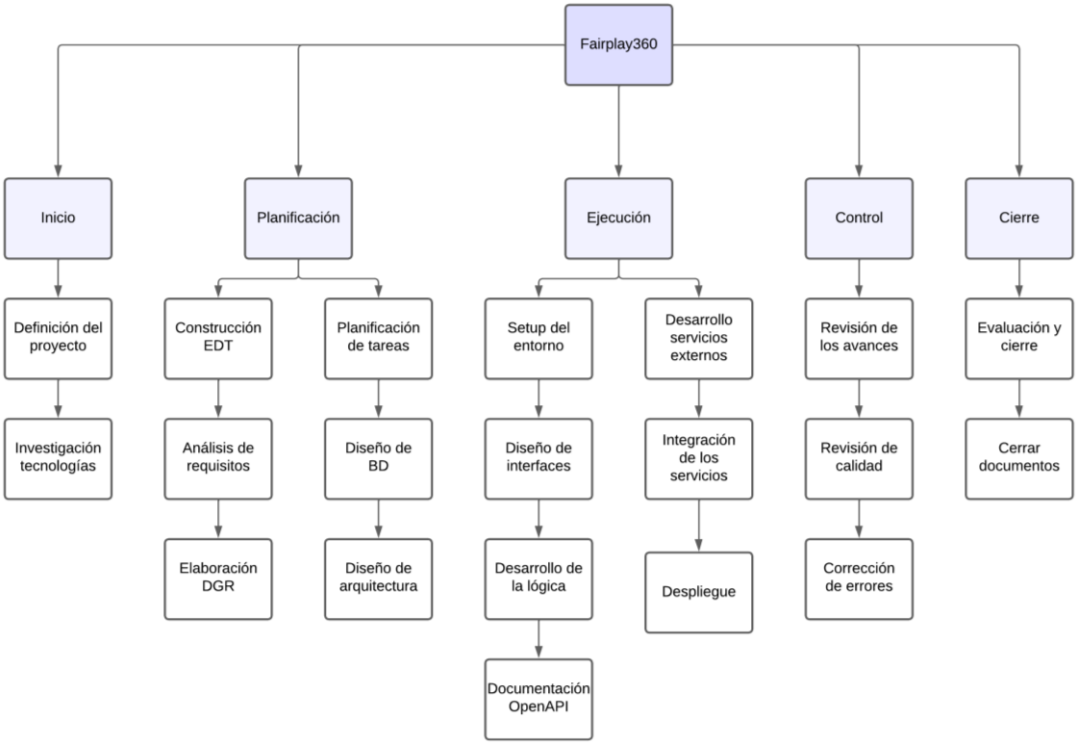


Figura C.1. Estructura de Desglose del Trabajo (EDT) del proyecto Fairplay360.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA