Trabajo de Fin de Grado

Aplicación móvil de iOS, para fans de la Formula 1

TITULACIÓN: Grado en Ingeniería Informática

AUTOR: Javier Ocón Barreiro

TUTORIZADO POR:
María Dolores Afonso Suárez

Fecha Mayo 2024

Agradecimientos

En primer lugar, me gustaría expresar mi más sincero agradecimiento a mi tutora, María Dolores Afonso Suárez, por su invaluable guía y apoyo durante la realización de este Trabajo Fin de Título. Su dedicación y conocimiento experto han sido fundamentales para el éxito de este proyecto.

A continuación, quiero agradecer a mi familia y amigos por su apoyo incondicional durante todo el proceso. Su aliento y comprensión me han motivado a seguir adelante en los momentos difíciles.

También quiero agradecer a mis compañeros de clase por su colaboración y compañerismo. Las discusiones y el intercambio de ideas han sido enriquecedores y han contribuido a mejorar este trabajo.

Por último, quiero agradecer a todas las personas que han participado de alguna manera en este proyecto. Su colaboración ha sido esencial para lograr los objetivos establecidos.

Gracias a todos por su apoyo y por haber hecho posible este logro.

Resumen

Esta página ofrece un resumen del proyecto de fin de grado. El proyecto se resumirá tanto en Español como en Inglés para garantizar una mayor accesibilidad y comprensión por parte de una audiencia más amplia. A continuación, se presentan los resúmenes en ambos idiomas.

Resumen en Español

'F1 Fan App' es una propuesta de aplicación móvil que ofrece una experiencia interactiva para los aficionados de la Fórmula 1. Diseñada para usuarios de iOS, busca proporcionar información en tiempo real, análisis de datos, contenido exclusivo y funciones de interacción con la comunidad. Este proyecto de ingeniería informática se centra en desarrollar una interfaz intuitiva, integrar datos en tiempo real y ofrecer contenido personalizado. Además, prioriza la interactividad y la escalabilidad, con el objetivo de crear una plataforma que mejore la forma en que los aficionados interactúan con la Fórmula 1. Se aplicarán habilidades de ingeniería de software y se desarrollará una aplicación iOS que contará con tecnologías para la implementación de chats y bases de datos, entre otras.

Resumen en Inglés

'F1 Fan App' is a mobile application concept that offers an interactive experience for Formula 1 fans. Designed for iOS users, it aims to provide real-time information, data analysis, exclusive content, and community interaction features. This computer engineering project focuses on developing an intuitive interface, integrating real-time data, and delivering personalized content. It also prioritizes interactivity and scalability to create a platform which enhances how fans engage with Formula 1. Software engineering skills will be applied, and a comprehensive iOS application will be developed, featuring technologies for implementing chats and databases, among others.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

En la era digital actual, la tecnología juega un papel crucial en la vida diaria de millones de personas, transformando diversos ámbitos, incluido el deporte, y ofreciendo nuevas experiencias a los aficionados. La Fórmula 1, reconocida como una de las categorías más prestigiosas y tecnológicamente avanzadas del automovilismo, cuenta con una base de seguidores global y apasionada. Con esto en mente, surge el proyecto "F1 Fan App", una aplicación móvil diseñada para brindar una experiencia interactiva y enriquecedora a los aficionados de la Fórmula 1.

Este proyecto se propone crear una plataforma todo-en-uno que combine información en tiempo real, interactividad y contenido exclusivo sobre la Fórmula 1. La "F1 Fan App.^está concebida para mejorar la experiencia del usuario antes, durante y después de cada Gran Premio, ofreciendo características como actualizaciones en vivo, análisis de datos, interacción con la comunidad y contenido exclusivo, incluyendo entrevistas y análisis técnicos.

Esta propuesta de TFG se enfoca en el desarrollo de una aplicación móvil para el Sistema Operativo iOS, en el marco de la Ingeniería Informática. Más allá de ser una herramienta informativa, la aplicación busca enriquecer la manera en que los aficionados interactúan con el mundo de la Fórmula 1. Con funcionalidades diseñadas para gestionar tanto la interfaz de usuario como el contenido en tiempo real, este proyecto representa una oportunidad para aplicar habilidades de ingeniería del software en un contexto práctico.

Capítulo 2

Estado actual y objetivos

2.1. Estado Actual

Aplicaciones Oficiales de la Fórmula 1

La organización de la Fórmula 1 ha desarrollado varias aplicaciones oficiales que proporcionan una amplia gama de datos e información relacionada con la competición. Entre las funcionalidades que ofrecen se incluyen:

- ✔ Resultados y estadísticas de las carreras.
- ✔ Información sobre equipos y pilotos.
- ✔ Noticias y actualizaciones.
- ✔ Datos en tiempo real.

Estas aplicaciones están disponibles tanto en versiones web como móviles. A continuación, se describen algunas de las aplicaciones oficiales:

- F1 Play

F1 Play es una aplicación que permite a los usuarios realizar predicciones sobre las carreras y ganar premios y experiencias únicas, como asistir a una carrera con todos los gastos pagados y acceder a zonas exclusivas si se aciertan todas las predicciones propuestas [8]. La ilustración 2.1 muestra la pantalla principal de la aplicación **F1 Play**.

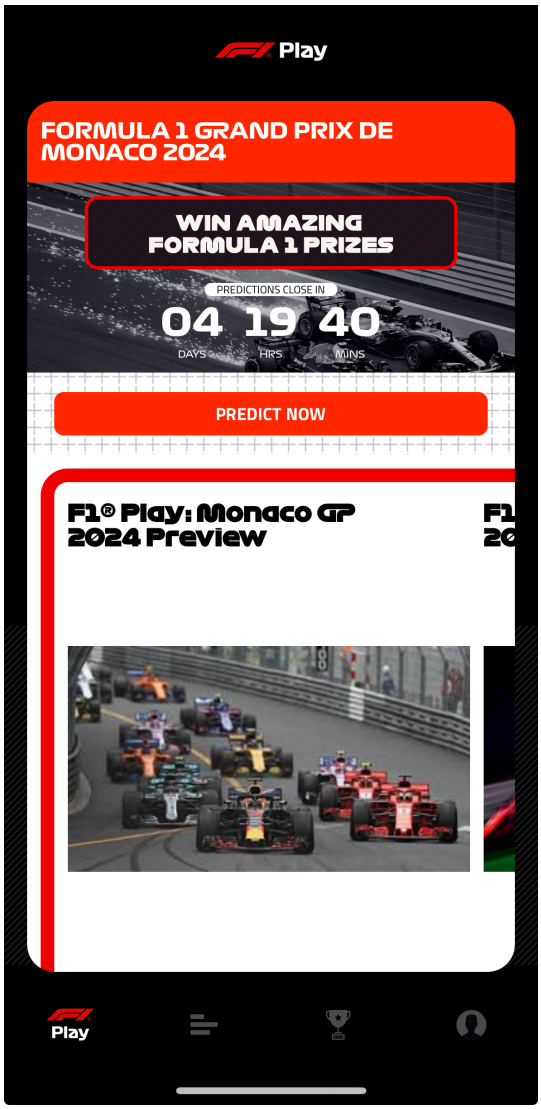

Ilustración 2.1: Pantalla principal de la aplicación ”F1 Play”.

- Formula 1

La aplicación principal de la **Fórmula** 1 ofrece noticias, información sobre pilotos, equipos, circuitos, el estado del campeonato de equipos y pilotos, el calendario, datos históricos de las carreras, y acceso a datos en directo. Esta aplicación está

disponible en versiones móvil y web. [2] A continuación, se muestran capturas de ambas versiones:

- Formula 1 App Móvil

  En las ilustraciones 2.2 & 2.3, se muestran diferentes pantallas de la aplicación móvil de la Formula 1, denominada "Formula 1".

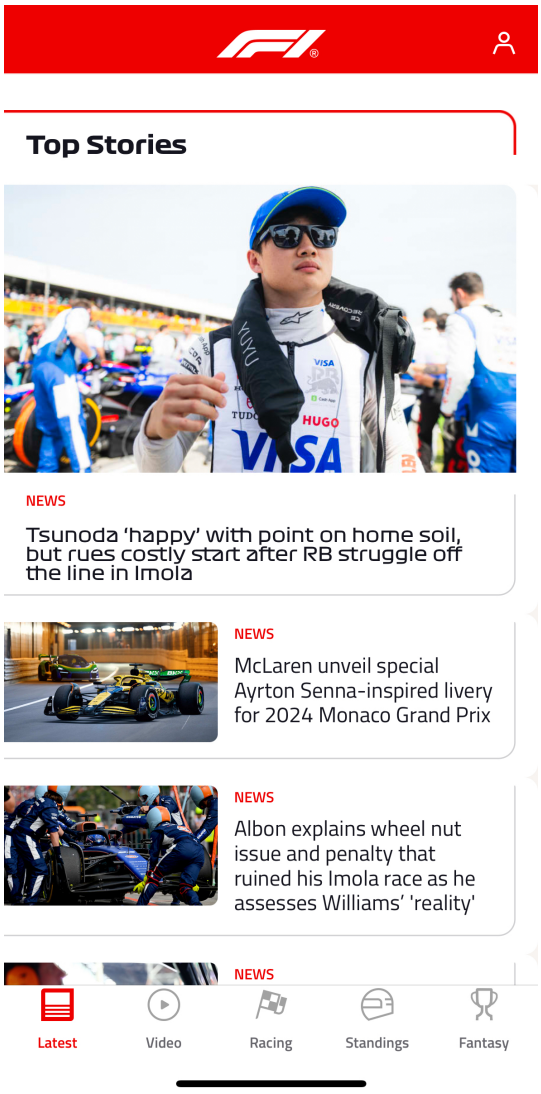

Ilustración 2.2: Pantalla principal de la aplicación móvil "Formula 1".

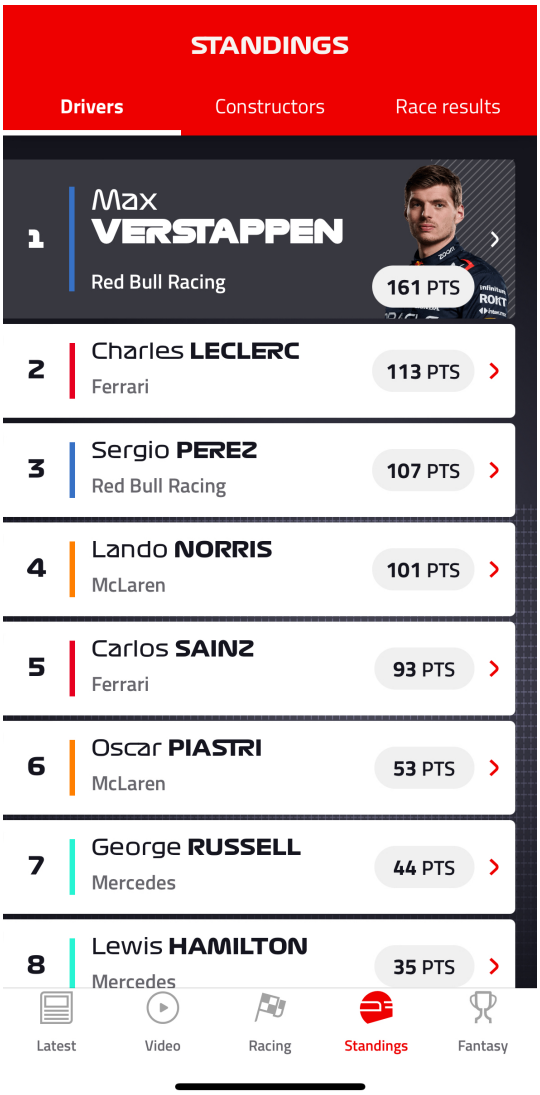

Ilustración 2.3: Pantalla de estado de campeonato de la aplicación móvil ”Formula 1”.

- Formula 1 Web - Live Timing

  En las ilustraciones 2.4 & 2.5, se muestran diferentes pantallas de la sección de ”Live Timing”de la aplicación web de la Formula 1.

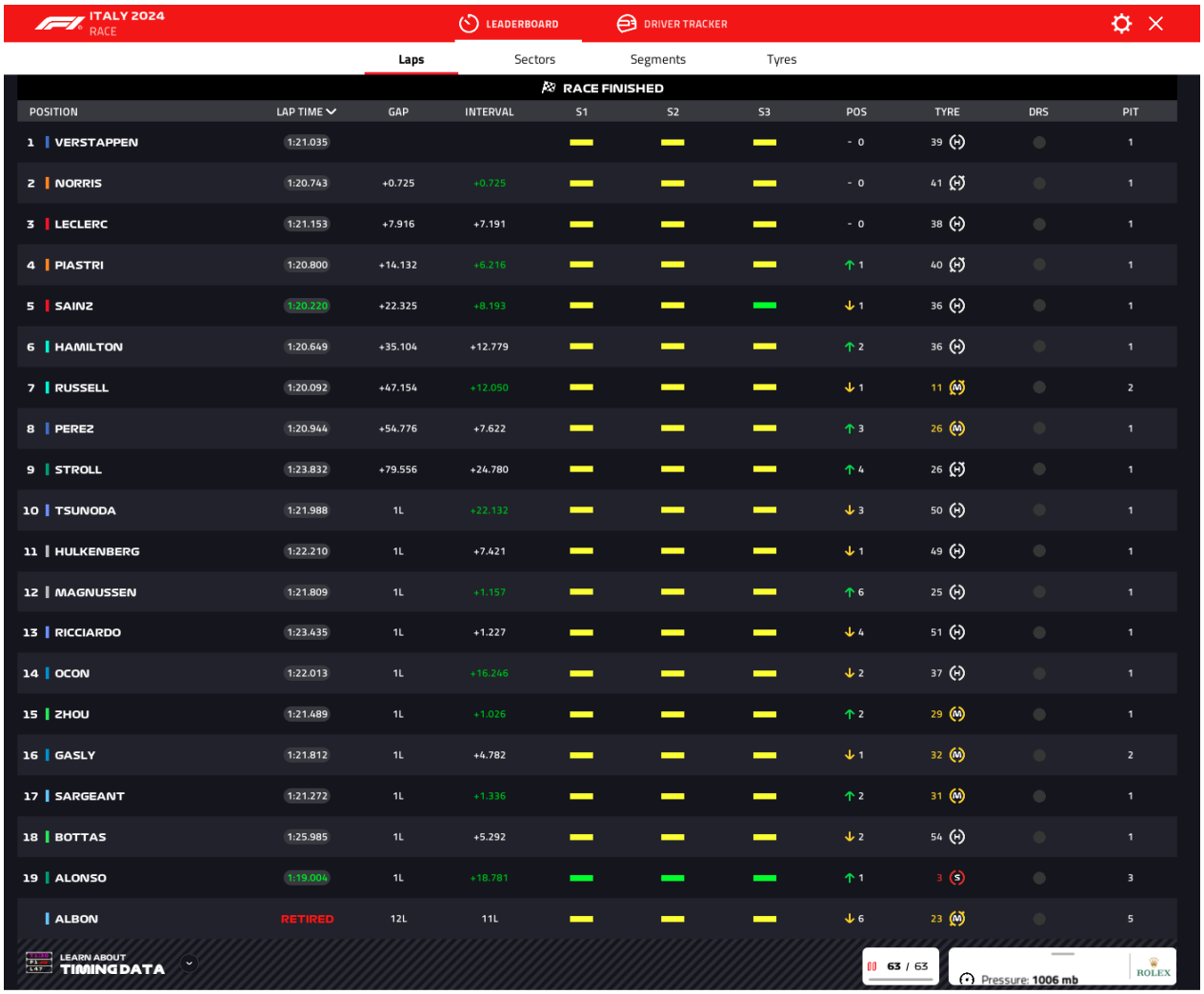

Ilustración 2.4: Captura de pantalla de la sección 'Live Timing' según posiciones de la aplicación web "Formula 1".

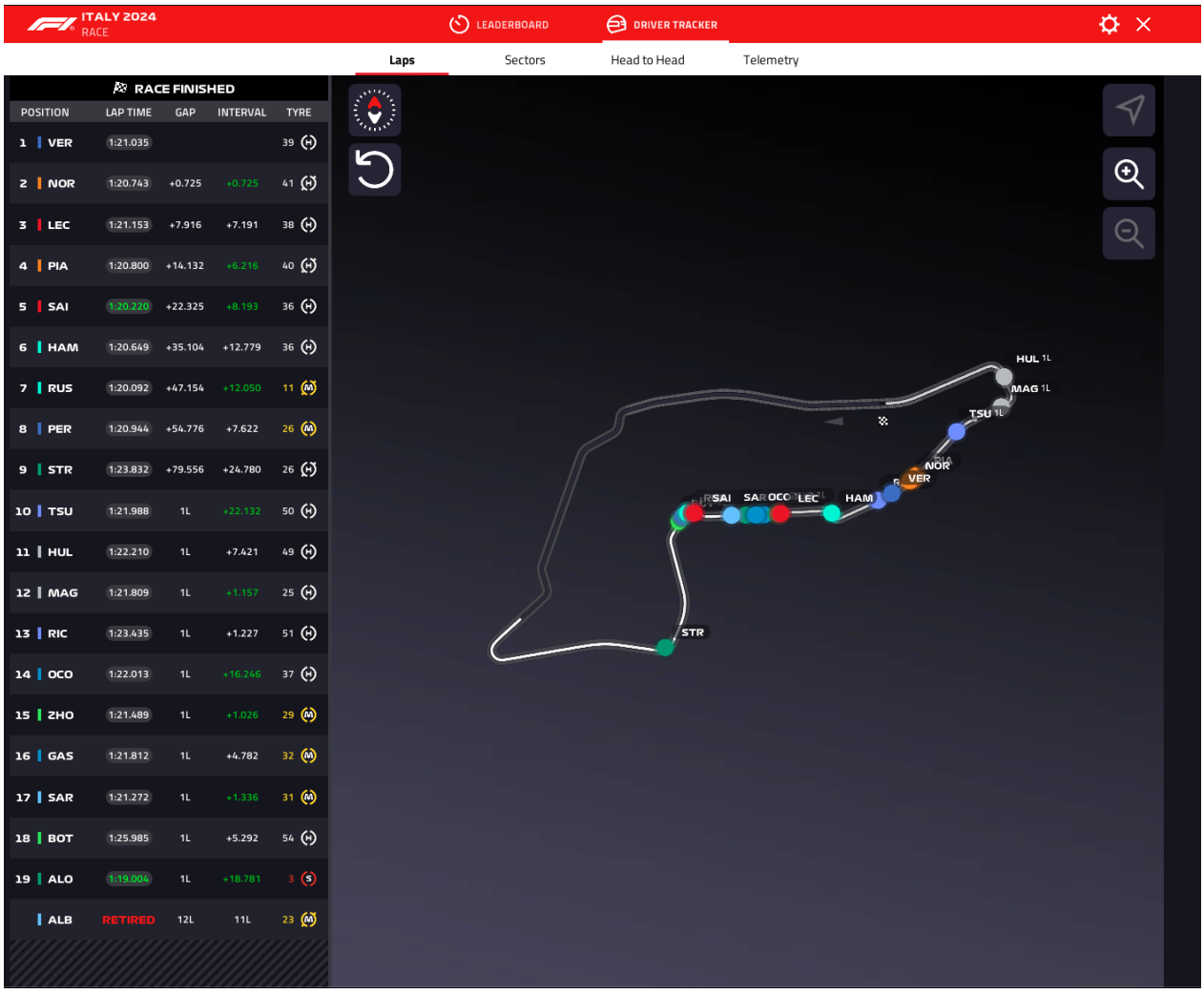

Ilustración 2.5: Captura de pantalla de la sección 'Live Timing' según posición en la pista de la aplicación web "Formula 1".

- F1 Clash

  F1 Clash es un juego de gestión de carreras de Fórmula 1 para dispositivos móviles, licenciado oficialmente por la Fórmula 1. Permite a los jugadores asumir el rol de jefe de equipo, gestionando pilotos, coches y estrategias para competir en emocionantes carreras 1v1 contra otros jugadores de todo el mundo [12]. En la ilustración 2.6, se muestra una captura de pantalla de la página de descarga de la aplicación **F1 Clash** en la Apple Store.

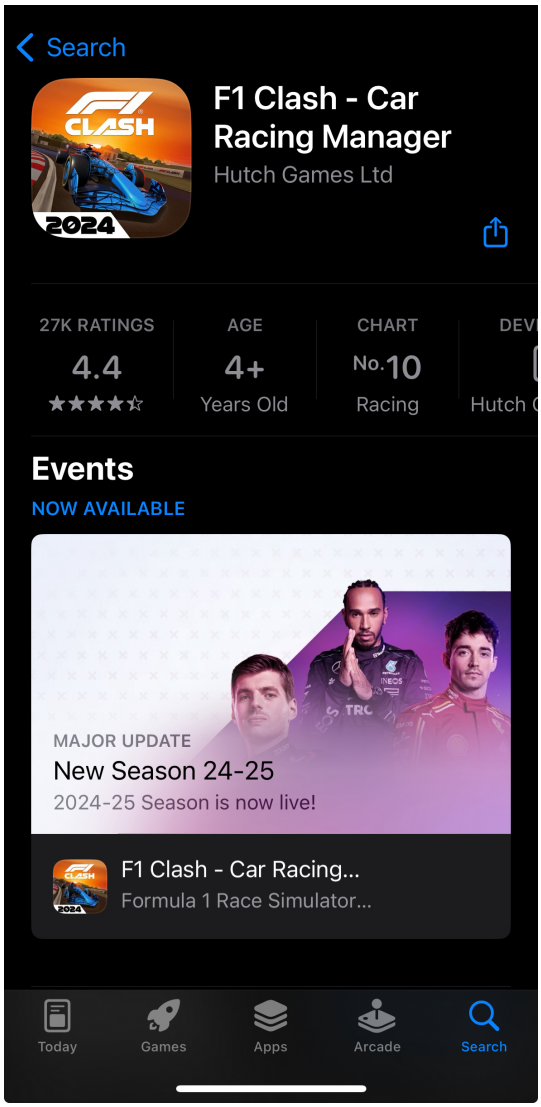

Ilustración 2.6: Captura de pantalla de la aplicación 'F1 Clash' en la AppStore.

Otras Aplicaciones Web

Además de las aplicaciones oficiales, hay diversas aplicaciones web que ofrecen servicios específicos como la visualización de datos en directo, chats de discusión y análisis de carreras. Sin embargo, estos servicios suelen estar separados en diferentes aplicaciones, lo que puede ser inconveniente para los usuarios que desean acceder a toda la información en un solo lugar.

A continuación se mostrarán algunas de las aplicaciones no afiliadas a la **Formula 1**:

- F1-Dash

F1-Dash ofrece tiempos en vivo y resultados de carreras para las carreras de Fórmula 1. También incluye información sobre las condiciones climáticas en la pista, así como zonas DRS y el estado del pit lane. **F1-Dash** no está afiliado a la Fórmula 1 de ninguna manera [14]. La ilustración 2.7 muestra una captura de pantalla de la sección ”Dashboard”de **F1-Dash**.

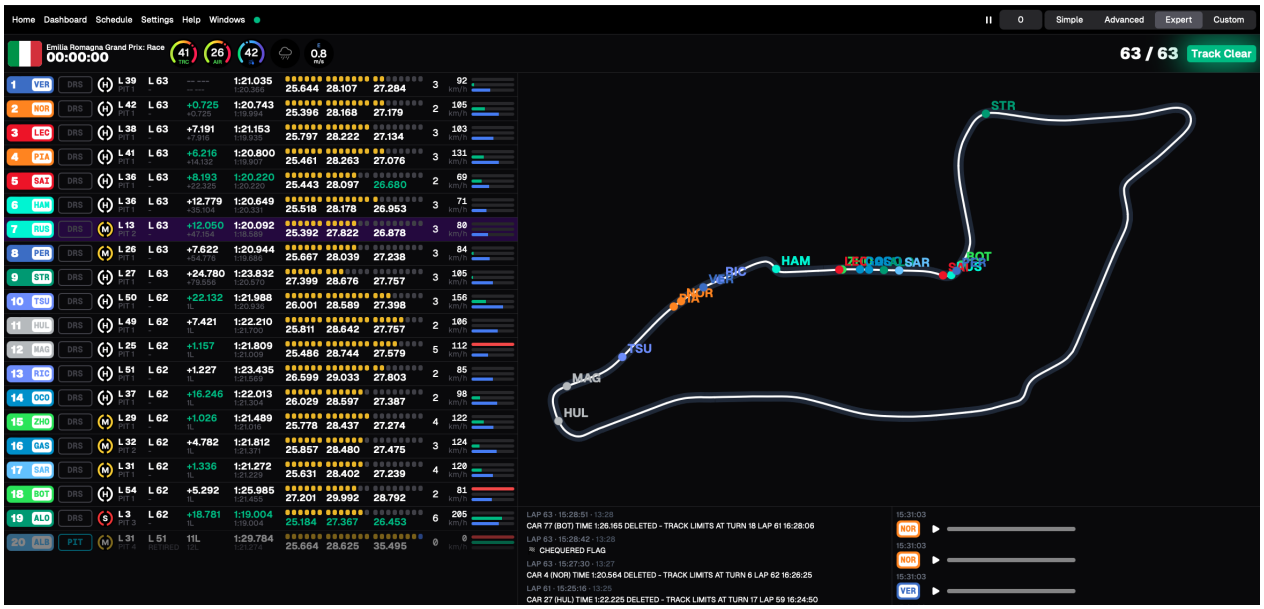

Ilustración 2.7: Pantalla de Dashboard de la aplicación web ”F1-Dash”.

- F1 Tfeed

F1 Tfeed ofrece tiempos en vivo y resultados de carreras para las carreras de Fórmula 1. En este caso, este si incluye un chat en directo, lo cual hace más interesante y divertida la estancia en esta página. La ilustración 2.8 muestra una captura de pantalla de la pantalla principal de **F1 Tfeed**.

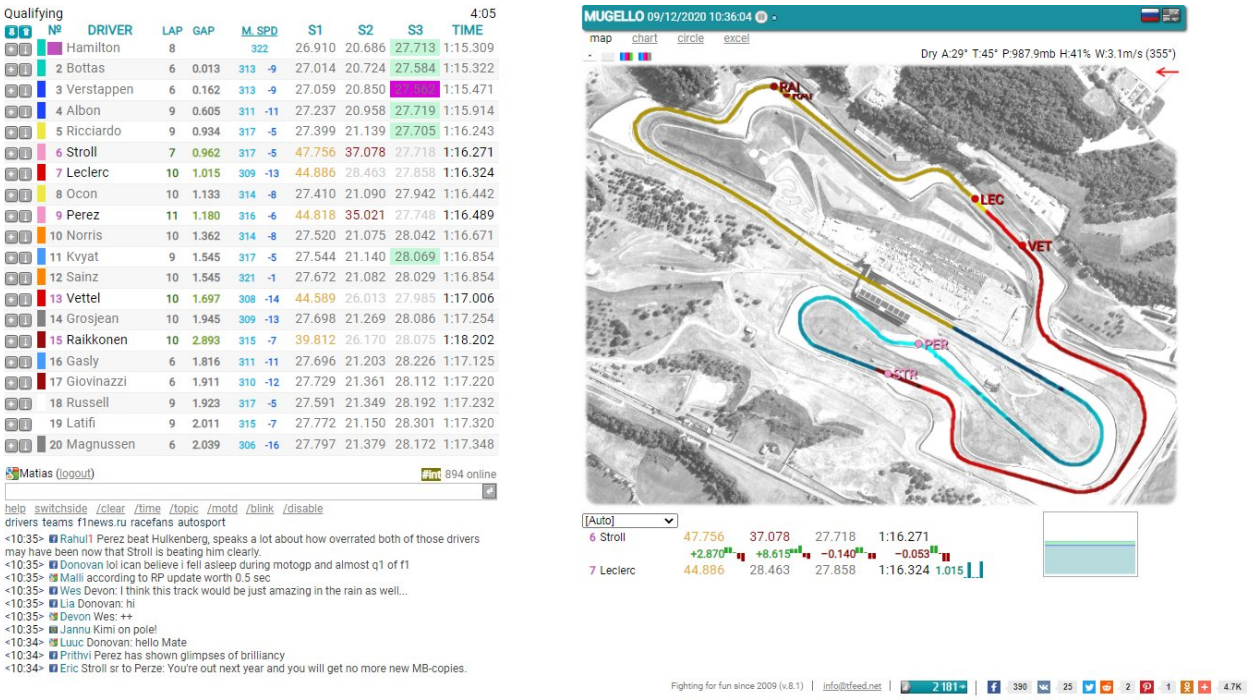

Ilustración 2.8: Pantalla de Dashboard de la página web ”F1 Tfeed”.

- GP Fans

GP Fans es una página web, parecida a la oficial de la Fórmula 1, que ofrece noticias, datos sobre pilotos, equipos y carreras entre otros [10]. Sin embargo, esta no incluye chats ni datos en directo. La ilustración 2.9 muestra una captura de pantalla de la pantalla principal de **GP Fans**.

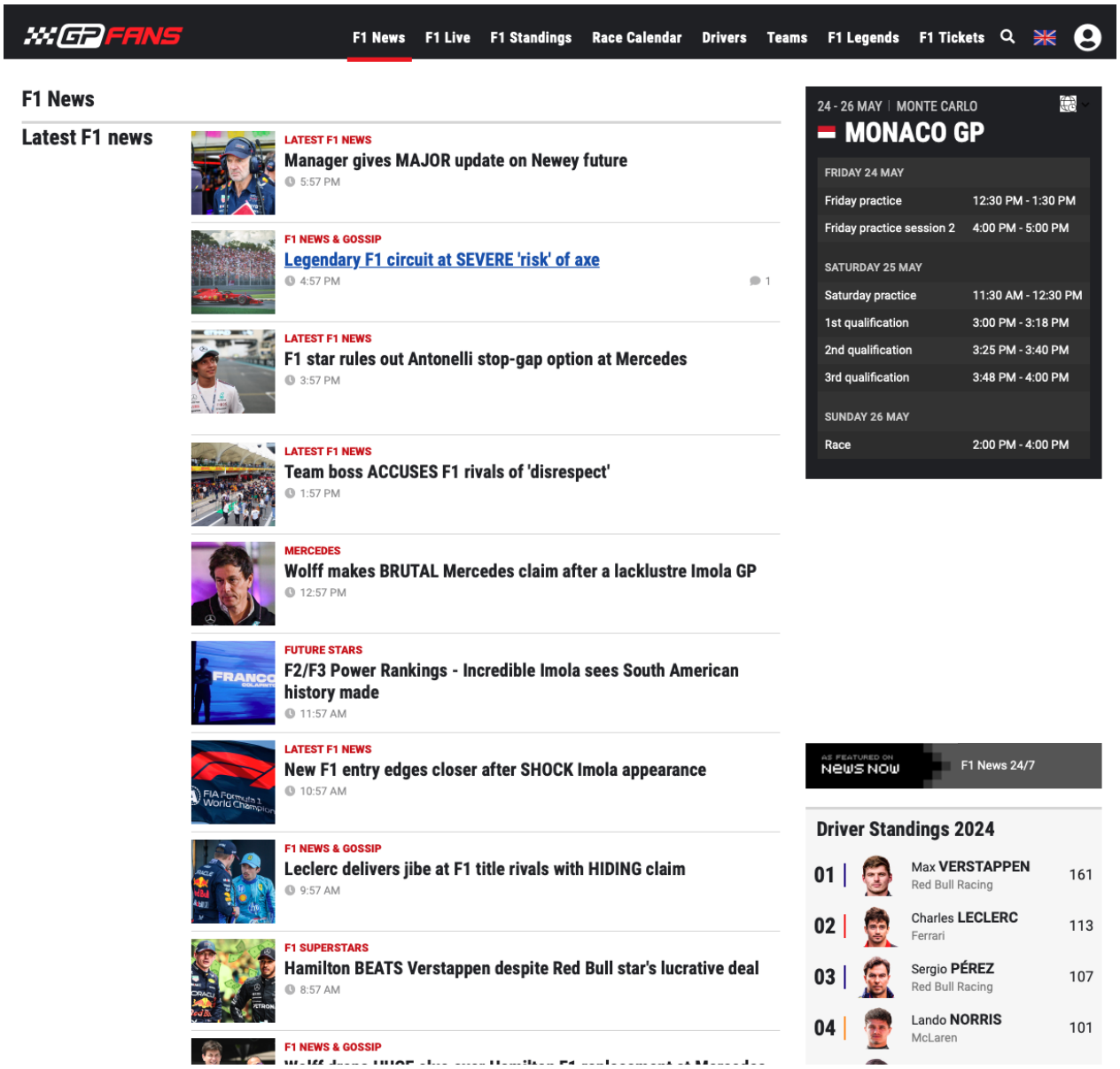

Ilustración 2.9: Pantalla de Noticias de la página web "GP Fans".

Aplicaciones Móviles

En el ámbito de las aplicaciones móviles, actualmente no existe una aplicación que integre de manera completa todas las funcionalidades deseadas por los aficionados a la Fórmula 1. En lugar de eso, las funciones están fragmentadas entre diversas aplicaciones, por ejemplo:

- ✔ Aplicaciones para recibir información y noticias.

- ✔ Aplicaciones para ver datos en directo durante las carreras.
- ✔ Aplicaciones para participar en chats en directo
- ✔ Aplicaciones de juegos de Fórmula 1.

Esta fragmentación implica que los usuarios deben utilizar múltiples aplicaciones para obtener una experiencia completa, lo cual puede ser incómodo y poco eficiente. La falta de una solución integral representa una oportunidad para el desarrollo de una aplicación que combine todas estas características en una sola plataforma.

2.2. Objetivos

El objetivo general se centra en el desarrollo de una App que aborde desafíos como el procesamiento de datos en tiempo real, la implementación de una interfaz intuitiva y la integración de diversas fuentes de contenido. Con una visión técnica se plantea desde el punto de vista de la robustez y en el ámbito del desarrollo haciendo uso de una metodología que facilite el desarrollo ágil. Para ello se establece una serie de objetivos específicos:

1. Desarrollar una Interfaz de Usuario Atractiva y Fácil de Navegar: Es fundamental centrarse en la experiencia del usuario al diseñar la interfaz. Esto implica utilizar principios de diseño intuitivos que permitan a los usuarios navegar fácilmente por la aplicación. Elementos como una navegación clara, iconografía familiar, y disposición organizada del contenido ayudan a mejorar la usabilidad. Además, se deben considerar colores y tipografías que no solo sean atractivos visualmente, sino también accesibles para diferentes usuarios, incluidos aquellos con discapacidades.

2. Integración de Datos en Tiempo Real: La aplicación debe mostrar información actualizada de manera constante. Esto incluye resultados de carreras en tiempo real, estadísticas y clasificaciones actualizadas conforme ocurren los eventos. Utilizar fuentes de datos confiables y asegurar una actualización constante de la información son clave para mantener a los usuarios informados y comprometidos.

3. Contenido Exclusivo y Personalizado: Además de los datos básicos, la aplicación puede ofrecer valor añadido a través de análisis profundos, entrevistas exclusivas con pilotos y personalidades del paddock, y contenido detrás de cámaras que no está disponible en otras plataformas.

4. Interactividad y Comunidad:Fomentar la interacción entre los usuarios es crucial para construir una comunidad sólida en torno a la aplicación. Esto se puede lograr mediante la implementación de funciones como foros de discusión donde los fans pueden debatir sobre carreras y eventos y juegos relacionados con la Fórmula 1 que aumenten la participación activa de los usuarios.

5. Adaptabilidad y Escalabilidad: La aplicación debe estar diseñada para crecer y adaptarse fácilmente a medida que se introducen nuevas características y se expande el contenido. Utilizar arquitecturas escalables en la nube y adoptar prácticas de desarrollo ágiles facilita la implementación rápida de actualizaciones y nuevas funcionalidades sin comprometer la estabilidad del sistema. Esto garantiza que la aplicación pueda mantenerse relevante y competitiva en un entorno digital dinámico y en constante evolución.

Aprender a desarrollar aplicaciones completas en iOS, aprendiendo como usar APIs en Xcode. También se aprenderá la tecnología para realizar el chat. Uso de base de datos y su implementación de uso en la API/Xcode.

Capítulo 3

Competencias específicas y aportaciones del trabajo

3.1. Competencias Específicas - Comunes a la rama de informática

En el contexto del proyecto desarrollado, se han identificado y demostrado diversas competencias específicas para el campo de la informática. A continuación se presentan las competencias adquiridas y validadas a través de este trabajo:

Código de Competencia	**Descripción de Competencia**
CI7	Conocimiento, diseño y utilización de forma eficiente de los tipos y estructuras de datos más adecuados a la resolución de un problema.
CI8	Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma robusta, segura y eficiente, eligiendo el paradigma y los lenguajes de programación más adecuados.
CI12	Conocimiento y aplicación de las características, funcionalidades y estructura de las bases de datos, que permitan su adecuado uso, y el diseño y el análisis e implementación de aplicaciones basadas en ellos.
CI14	Conocimiento y aplicación de los principios fundamentales y técnicas básicas de la programación paralela, concurrente, distribuida y de tiempo real.
CI16	Conocimiento y aplicación de los principios, metodologías y ciclos de vida de la ingeniería del software.

Cuadro 3.1: Tabla de competencias específicas comunes

Justificación de CI7

En la implementación del proyecto, se utilizaron diversos tipos de datos en MySQL para optimizar el almacenamiento y manejo de la información. A continuación se detalla el uso de estos tipos de datos:

- ✔ **DATETIME**: Este tipo de dato se utilizó para almacenar fechas y horas, permitiendo el registro preciso de eventos temporales. Su empleo es crucial para tareas que requieren una cronología exacta, como el seguimiento de cambios, auditorías y la programación de eventos.

- ✔ **INT**: Los valores enteros son fundamentales para identificar registros de manera única. En este proyecto, el tipo de dato INT se empleó para identificadores primarios y contadores, asegurando así una referencia rápida y eficiente a los registros de la base de datos.

- ✔ **TEXT**: Para almacenar grandes bloques de texto, como descripciones detalladas, comentarios de los usuarios y otros datos extensos, se utilizó el tipo de dato TEXT. Este tipo permite manejar cadenas de texto de gran longitud sin preocuparse por un límite específico, facilitando la inclusión de información extensa y detallada.

- ✔ **VARCHAR**: Este tipo de dato es adecuado para cadenas de texto de longitud variable, como nombres, títulos y otros datos textuales que no exceden un límite predefinido. En este proyecto, VARCHAR se usó para campos donde la longitud del texto es moderada y se conoce de antemano, optimizando así el uso de espacio en la base de datos.

Además del uso eficiente de los tipos de datos en MySQL, se implementó el formato **JSON** para el intercambio de información a través de la API. JSON (JavaScript Object Notation) es un formato ligero y de fácil lectura para estructurar datos. Las ventajas de utilizar JSON incluyen:

- ✔ **Flexibilidad**: JSON permite estructurar datos complejos de manera sencilla, soportando arrays y objetos anidados, lo que facilita la representación de relaciones complejas entre los datos.

- ✔ **Compatibilidad**: Es compatible con la mayoría de los lenguajes de programación, lo que facilita la integración entre diferentes sistemas y tecnologías.

- ✔ **Eficiencia en la transmisión**: El formato compacto de JSON reduce la cantidad de datos transmitidos entre el servidor y los clientes, mejorando así el rendimiento de la comunicación y disminuyendo la latencia.

- ✔ **Facilidad de uso**: JSON es fácil de leer y escribir tanto para humanos como para máquinas, lo que simplifica el desarrollo y el mantenimiento de la API.

La utilización de estos tipos de datos y tecnologías no solo optimizó el rendimiento y la eficiencia de la base de datos y la API, sino que también aseguró la integridad y accesibilidad de la información a lo largo de todo el proyecto.

Justificación de CI8

En este proyecto, se ha demostrado la capacidad para analizar, diseñar, construir y mantener aplicaciones de manera robusta, segura y eficiente seleccionando adecuadamente paradigmas y lenguajes de programación:

- ✔ **Análisis y Diseño**: Se realizó un análisis detallado de los requisitos para diseñar una aplicación móvil intuitiva y una interfaz web para la gestión de la base da datos. También se diseñó las interfaces y arquitecturas de ambas aplicaciones.

- ✔ **Construcción Robusta**: La aplicación móvil se implementó utilizando Swift, un len-

guaje nativo de iOS conocido por su robustez y rendimiento en plataformas iOS. Para la interfaz web, se eligieron tecnologías compatibles que garantizaran la escalabilidad y la seguridad de la base de datos.

- ✔ **Seguridad**: Se implementaron medidas de seguridad sólidas tanto en la aplicación móvil como en la interfaz web para proteger los datos sensibles, privacidad de los usuarios & accesos no autorizados.

- ✔ **Eficiencia**: Se seleccionaron lenguajes y tecnologías que optimizan el rendimiento y el manejo de recursos, con fin de que la aplicación funcione de manera eficiente incluso bajo cargas elevadas de datos y usuarios.

Estas decisiones reflejan la habilidad para evaluar y seleccionar paradigmas y lenguajes de programación adecuados para cada componente del proyecto, asegurando que la aplicación sea robusta, segura y eficiente desde su diseño hasta su implementación y mantenimiento continuo.

Justificación de CI12

Para gestionar y almacenar los datos, se eligió una base de datos relacional MySQL debido a sus características robustas y ampliamente reconocidas en la gestión de datos estructurados. MySQL proporciona una serie de ventajas clave como la integridad referencial, la capacidad de realizar consultas complejas, y un soporte robusto para transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), lo cual es esencial para mantener la integridad y coherencia de los datos en aplicaciones críticas.

- ✔ **MySQL**: Se utilizó MySQL como la base de datos relacional principal para almacenar datos estructurados relacionados con las aplicaciones. Se eligieron los tipos de datos más adecuados para cada tipo de información, como `DATETIME` para fechas y horas, `INT` para valores numéricos enteros, `TEXT` para grandes bloques de texto, y `VARCHAR` para cadenas de caracteres de longitud variable. Esto aseguró una eficiente organización y acceso a los datos, facilitando tanto su gestión como su consulta.

- ✔ **Firebase Storage**: Para el almacenamiento de imágenes y otros archivos multimedia, se utilizó Firebase Storage. Esta plataforma ofrece una solución eficiente y escalable para el manejo de archivos, permitiendo su acceso y almacenamiento de forma segura. La integración de Firebase Storage con otras herramientas de Firebase también facilita la gestión de autenticación y reglas de acceso, mejorando la seguridad y la facilidad de uso.

El uso de MySQL para la gestión de datos estructurados y Firebase Storage para el manejo de archivos multimedia, demuestra una comprensión avanzada y la aplicación efectiva de las características, funcionalidades y estructura de las bases de datos. Estas decisiones no solo aseguran un rendimiento óptimo y una gestión eficiente de los datos, sino que también facilitan el desarrollo y mantenimiento de aplicaciones robustas y escalables.

Justificación de CI14

Para asegurar una carga de datos eficiente y en tiempo real, se han utilizado funciones asíncronas y estrategias de almacenamiento en caché tanto en la aplicación iOS desarrollada en Swift como en la aplicación web. Estas técnicas permiten optimizar el rendimiento y la experiencia del usuario al acceder y mostrar datos de manera rápida y eficiente, especialmente en entornos donde la conectividad de red puede ser variable o limitada.

- ✔ **Aplicación iOS**: En la aplicación iOS, desarrollada en Swift, se implementó un sistema de caché local utilizando las capacidades de almacenamiento en disco del dispositivo. Esto permite almacenar temporalmente datos frecuentemente accedidos, reduciendo la necesidad de realizar múltiples solicitudes al servidor y mejorando así la respuesta y la carga de datos incluso en condiciones de red adversas.

- ✔ **Aplicación web**: En la aplicación web, se utilizó almacenamiento en caché del lado del cliente mediante tecnologías como el almacenamiento local del navegador o el uso de servicios de almacenamiento en memoria como Redis. Esto permite que los datos sean recuperados rápidamente sin depender completamente de la conectividad con el servidor, mejorando la velocidad de carga de la aplicación y la experiencia del usuario.

La combinación de funciones asíncronas y estrategias de caché demuestra una aplicación efectiva de los principios de programación paralela, concurrente y distribuida. Estas técnicas no solo optimizan el rendimiento de las aplicaciones, sino que también aseguran una experiencia de usuario consistente y fluida, independientemente de las condiciones de red. Además, la capacidad de gestionar datos en tiempo real y de manera eficiente subraya la habilidad para implementar soluciones robustas en entornos complejos y dinámicos.

Justificación de CI16

A continuación, se describen las prácticas y metodologías implementadas que aseguran un desarrollo de software estructurado, eficiente y de alta calidad:

- ✔ **Principios de Ingeniería de Software**: Se han seguido principios fundamentales como la modularidad, reutilización de código, y mantenimiento. La modularidad se

logró mediante el diseño de componentes independientes y cohesivos, lo cual facilita la gestión del proyecto y mejora la mantenibilidad del código. La reutilización de código se implementó mediante la creación de librerías y funciones reutilizables, reduciendo así el esfuerzo y tiempo de desarrollo.

- ✔ **Metodologías Ágiles**: Se utilizó la metodología ágil Scrum para gestionar el ciclo de vida del desarrollo del software. Scrum facilita la entrega de incrementos de producto de manera iterativa y con una retroalimentación continua, lo cual permite ajustar el desarrollo según las necesidades cambiantes del proyecto. Se realizaron reuniones para el seguimiento del progreso, y se planificaron Sprints para mantener una constante entrega.

- ✔ **Gestión del Ciclo de Vida del Software**: El proyecto se dividió en fases claras siguiendo el ciclo de vida del desarrollo de software: análisis de requisitos, diseño, implementación, pruebas, despliegue y mantenimiento. Cada fase se documentó y se revisó cuidadosamente para asegurar la calidad y cumplir con los objetivos del proyecto.

- ✔ **Control de Versiones**: Se utilizó Git como sistema de control de versiones para gestionar el código fuente del proyecto (Aplicación iOS & Interfaz Web). Git permitió llevar un registro detallado de los cambios realizados, permitiendo revertir cambios si era necesario.

- ✔ **Pruebas y Calidad del Software**: Se implementaron pruebas automatizadas y manuales para asegurar la calidad del software. Las pruebas unitarias (automatizadas) verificaron que cada componente funcionara correctamente de manera aislada. También se realizaron pruebas de aceptación del usuario para validar que el producto final cumpliera con los requisitos y expectativas.

- ✔ **Documentación**: Se mantuvo una documentación a lo largo de todo el ciclo de vida del proyecto. Esto incluyó la documentación de requisitos, diseño y documentación de código. La documentación no solo facilita la comprensión y el uso del software por parte de otros desarrolladores y usuarios finales, sino que también es crucial para el mantenimiento y la evolución futura del proyecto.

La adopción de estos principios, metodologías y prácticas a lo largo del ciclo de vida del desarrollo de software ha permitido entregar un producto de alta calidad, alineado con las necesidades del cliente y fácil de mantener y escalar en el futuro. Este enfoque estructurado y metódico demuestra un profundo conocimiento y aplicación de la ingeniería del software, asegurando que el proyecto se desarrolló de manera eficiente y efectiva.

3.2. Competencias Específicas - Tecnologías Específicas

En el contexto del proyecto desarrollado, se han identificado y demostrado diversas competencias específicas para tecnologías específicas de la informática. A continuación se presentan las competencias adquiridas y validadas a través de este trabajo:

Código de Competencia	**Descripción de Competencia**
TI3	Capacidad para emplear metodologías centradas en el usuario y la organización para el desarrollo, evaluación y gestión de aplicaciones y sistemas basados en tecnologías de la información que aseguren la accesibilidad, ergonomía y usabilidad de los sistemas.
TI6	Capacidad de concebir sistemas, aplicaciones y servicios basados en tecnologías de red, incluyendo Internet, web, comercio electrónico, multimedia, servicios interactivos y computación móvil.
TI7	Capacidad para comprender, aplicar y gestionar la garantía y seguridad de los sistemas informáticos.

Cuadro 3.2: Tabla de competencias específicas de tecnologías específicas

Justificación de TI3

A continuación, se describen las prácticas y metodologías implementadas para cumplir con esta competencia:

- ✔ **Metodologías Centradas en el Usuario (UCD)**: Se aplicaron principios de diseño centrado en el usuario (UCD) durante todas las fases del desarrollo del proyecto. Esto incluyó la realización de estudios y análisis de las necesidades de los usuarios antes del inicio del diseño, garantizando que las soluciones desarrolladas estuvieran alineadas con las expectativas y requerimientos.

- ✔ **Prototipos y Pruebas de Usabilidad**: Se desarrollaron prototipos de baja y alta fidelidad como parte del proceso iterativo de diseño de la interfaz de usuario. Estos prototipos, que mantenían la misma paleta de colores debido a la coherencia visual, se modificaban continuamente según los comentarios de los usuarios sobre la usabilidad de la propia aplicación. Aunque no se realizó un test de usabilidad formal, los usuarios compartieron sus experiencias al usar la aplicación, lo que proporcionó una retroalimentación valiosa. Esta información fue fundamental para identificar y corre-

gir problemas de navegación y funcionalidad, asegurando una experiencia de usuario intuitiva y eficiente.

✔ **Accesibilidad**: Directrices de Accesibilidad para el Contenido Web (WCAG): Se siguieron las directrices de WCAG para asegurar que el sistema fuera accesible para usuarios con discapacidades. Esto implica:

- **Navegación mediante teclado**: Las interfaces se diseñaron para ser completamente navegables usando solo el teclado, lo cual es crucial para usuarios con discapacidades motrices.

- **Contraste adecuado**: Se garantizó que el contraste entre el texto y el fondo fuera suficiente para usuarios con discapacidades visuales, facilitando la lectura y la interacción con la interfaz.

Evaluación de Accesibilidad en la Interfaz Web:

- **Lighthouse**: Se utilizó Lighthouse, una herramienta de auditoría de accesibilidad, obteniendo una puntuación promedio de `89/100` en accesibilidad en todas las pantallas de la interfaz web.

- **Aria-label y tabindex**: Se implementaron atributos como aria-label y tabindex para mejorar la accesibilidad. Estos atributos ayudan a proporcionar descripciones más claras y control de navegación mediante teclado.

Pruebas y Mejoras de Accesibilidad en la App 'F1 Fan':

- **Accessibility Inspector en Xcode**: Se usó la herramienta Accessibility Inspector para probar y mejorar la accesibilidad de la aplicación 'F1 Fan'.

- **accessibilityLabel**: Se añadió accessibilityLabel a los elementos de la interfaz para proporcionar descripciones accesibles a los usuarios que utilizan VoiceOver.

- **accessibilityTraits**: Se configuraron accessibilityTraits para definir las características y comportamientos de los elementos de la interfaz, facilitando su uso por parte de tecnologías asistivas.

✔ **Ergonomía y Diseño de Interacción**: El diseño de la interfaz de usuario se realizó

teniendo en cuenta principios ergonómicos, asegurando que las interacciones fueran cómodas y eficientes. Se prestó especial atención a la disposición de los elementos en pantalla, la navegación y la carga cognitiva del usuario, creando una interfaz que minimiza el esfuerzo requerido para usar la aplicación.

✔ **Diseño Responsive**: La interfaz web fue desarrollada con un diseño responsive, asegurando que la aplicación fuera accesible y usable en una variedad de dispositivos, incluyendo computadoras de escritorio, tabletas y teléfonos móviles. Esto garantiza una experiencia de usuario consistente y positiva sin importar el dispositivo utilizado.

El enfoque en metodologías centradas en el usuario y la organización, junto con la implementación de prácticas de accesibilidad, ergonomía y usabilidad, asegura que las aplicaciones desarrolladas no solo cumplan con los estándares técnicos, sino que también proporcionen una experiencia de usuario óptima. Este enfoque integral y centrado en el usuario demuestra una comprensión profunda y aplicación efectiva de los principios de diseño y gestión de sistemas basados en tecnologías de la información.

Justificación de TI6

A continuación, se describen las soluciones desarrolladas y las tecnologías empleadas para cumplir con esta competencia:

✔ **Aplicación Móvil (iOS)**: Se desarrolló una aplicación móvil para la plataforma iOS utilizando Swift. Esta aplicación permite a los usuarios interactuar con el sistema desde sus dispositivos móviles, proporcionando acceso a funcionalidades clave en cualquier momento y lugar. La aplicación móvil incluye características interactivas y multimedia que mejoran la experiencia del usuario.

✔ **Interfaz Web para Manejo de la Base de Datos**: Se diseñó y desarrolló una interfaz web utilizando HTML, CSS y JavaScript para la gestión de la base de datos. Esta interfaz permite a los administradores y usuarios autorizados acceder, modificar y gestionar los datos de manera eficiente y segura desde cualquier navegador web. La interfaz web está diseñada para ser intuitiva y fácil de usar, facilitando la administración y monitoreo de los datos.

✔ **APIs para la Integración de Sistemas**: Para la integración de la aplicación móvil y la interfaz web con la base de datos y demás servicios, se desarrollaron dos APIs:

- **API en PHP**: Se implementó una API en PHP para manejar las solicitudes de la aplicación móvil. Esta API se encarga de procesar las solicitudes, interactuar con

la base de datos MySQL y retornar las respuestas necesarias a la aplicación móvil en formato JSON. La elección de PHP se debió a su robustez y amplia adopción en el desarrollo backend.

- **API en Node.js**: Se desarrolló una API en Node.js para gestionar las solicitudes de la interfaz web. Node.js, con su arquitectura orientada a eventos y su capacidad para manejar múltiples conexiones simultáneamente, es ideal para aplicaciones en tiempo real y de alto rendimiento. Esta API también interactúa con la base de datos MySQL y provee las respuestas en formato JSON.

El desarrollo de una aplicación móvil para iOS, una interfaz web para la gestión de la base de datos, y dos APIs para la integración de estos componentes demuestra una comprensión avanzada y una aplicación efectiva de las tecnologías de red. Estas soluciones no solo optimizan el rendimiento y la funcionalidad del sistema, sino que también aseguran una experiencia de usuario consistente y fluida a través de diferentes plataformas.

Justificación de TI7

A continuación se describen las prácticas y medidas implementadas para asegurar la seguridad y la gestión de riesgos en el sistema:

- ✔ **Seguridad en la Aplicación Móvil (iOS)**: Aunque la aplicación móvil no requiere inicio de sesión ni crea cuentas de usuario, se implementaron prácticas de seguridad para proteger los datos y garantizar la integridad de las operaciones. Esto incluye la validación de inputs en el chat para prevenir ataques de inyección de código (SQL injection) y otras vulnerabilidades comunes.

- ✔ **Interfaz Web Segura**: La interfaz web para la gestión de la base de datos MySQL y Firebase Storage incorpora un sistema de autenticación mediante usuario y contraseña, gestionado con cookies para mantener la sesión activa de manera segura. Esta autenticación controla el acceso a las funcionalidades CRUD (Crear, Leer, Actualizar, Eliminar) y al almacenamiento de archivos en Firebase Storage, asegurando que solo usuarios autorizados puedan realizar estas operaciones.

- ✔ **APIs Seguras**: Las APIs desarrolladas en PHP y Node.js utilizan prácticas de seguridad como la validación de inputs para prevenir ataques, tales como la manipulación de datos no autorizada. Además, permiten la comunicación segura a través de HTTP y HTTPS, gestionan correctamente las políticas de CORS (Cross-Origin Resource Sharing) para evitar vulnerabilidades de seguridad relacionadas con el intercambio de recursos entre dominios.

La implementación de estas medidas asegura que el proyecto 'F1 Fan App' no solo cumpla un estándard de seguridad, sino que también demuestre una comprensión profunda y una aplicación efectiva de las prácticas de garantía y seguridad de los sistemas informáticos. La combinación de seguridad en la aplicación móvil, la interfaz web, las APIs y la gestión de permisos y roles proporciona una infraestructura robusta y confiable para proteger los datos y las operaciones del sistema.

Capítulo 4

Desarrollo

4.1. Metodología

Para el desarrollo del proyecto se estableció una metodología que dividía el proceso en fases. Esta metodología facilitó la agilidad en las etapas de diseño, desarrollo, pruebas y documentación, permitiendo realizar revisiones que posibilitaron modificaciones según fuera necesario. Se utilizó Kanban en Trello para gestionar las tareas, organizándolas en diferentes estados como **IDEAS**, **ToDo**, **Doing**, **Testing** y **Done**. Esto permitió visualizar claramente el flujo de trabajo y priorizar las actividades pendientes según su estado y relevancia. En la ilustración 4.1 se muestra el estado del Trello en el SPRINT final donde el objetivo principal es hacer esta memoria final.

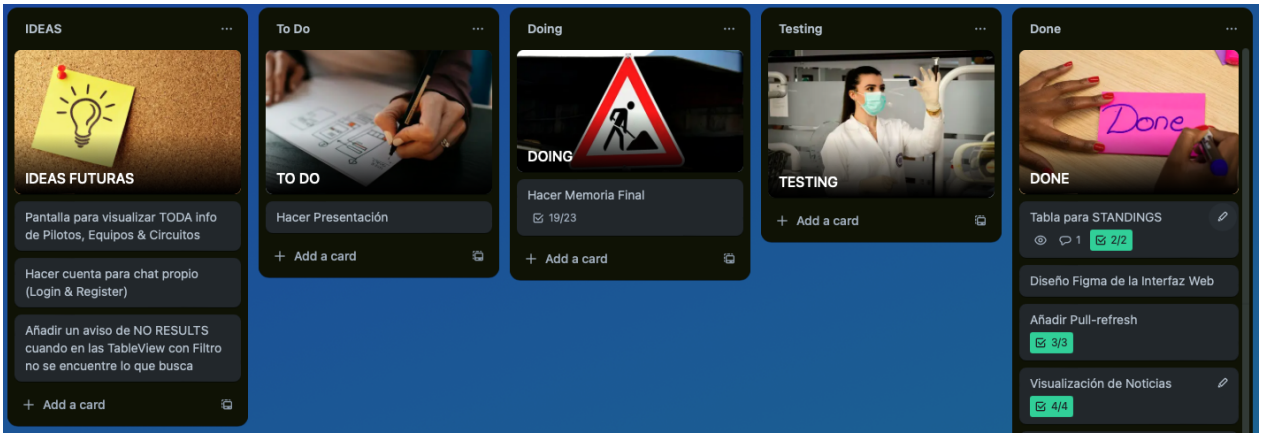

Ilustración 4.1: Captura de pantalla del uso de Kanban en Trello

Se adoptaron algunos aspectos y herramientas del marco de trabajo **Scrum** cuando se adaptaban a las necesidades específicas del proyecto. Actué como la persona a cargo del desarrollo, definiendo los requisitos como cliente del proyecto, mientras se mantenían reuniones periódicas con la tutora para presentar el producto, discutir ideas y mostrar los avances realizados.

4.2. Diseño

Todo el proceso de diseño del proyecto 'F1 Fan App' se realizó utilizando la herramienta de diseño colaborativo **Figma** [7]. A continuación, se muestran los diseños de la aplicación móvil iOS y la interfaz web para manejar la base de datos y archivos multimedia de Firebase Storage.

4.2.1. Diseño de la Aplicación Móvil (iOS)

A continuación se mostrará los diseños de las pantallas para la Aplicación iOS. También puede accederse al diseño original en **Figma** haciendo click aquí.

Ilustración 4.2: Diseño de pantalla de bienvenida de 'F1 Fan App'.

En la Ilustración 4.2, se muestra una captura del diseño de la pantalla de carga de la aplicación, donde se puede observar el logo de la propia.

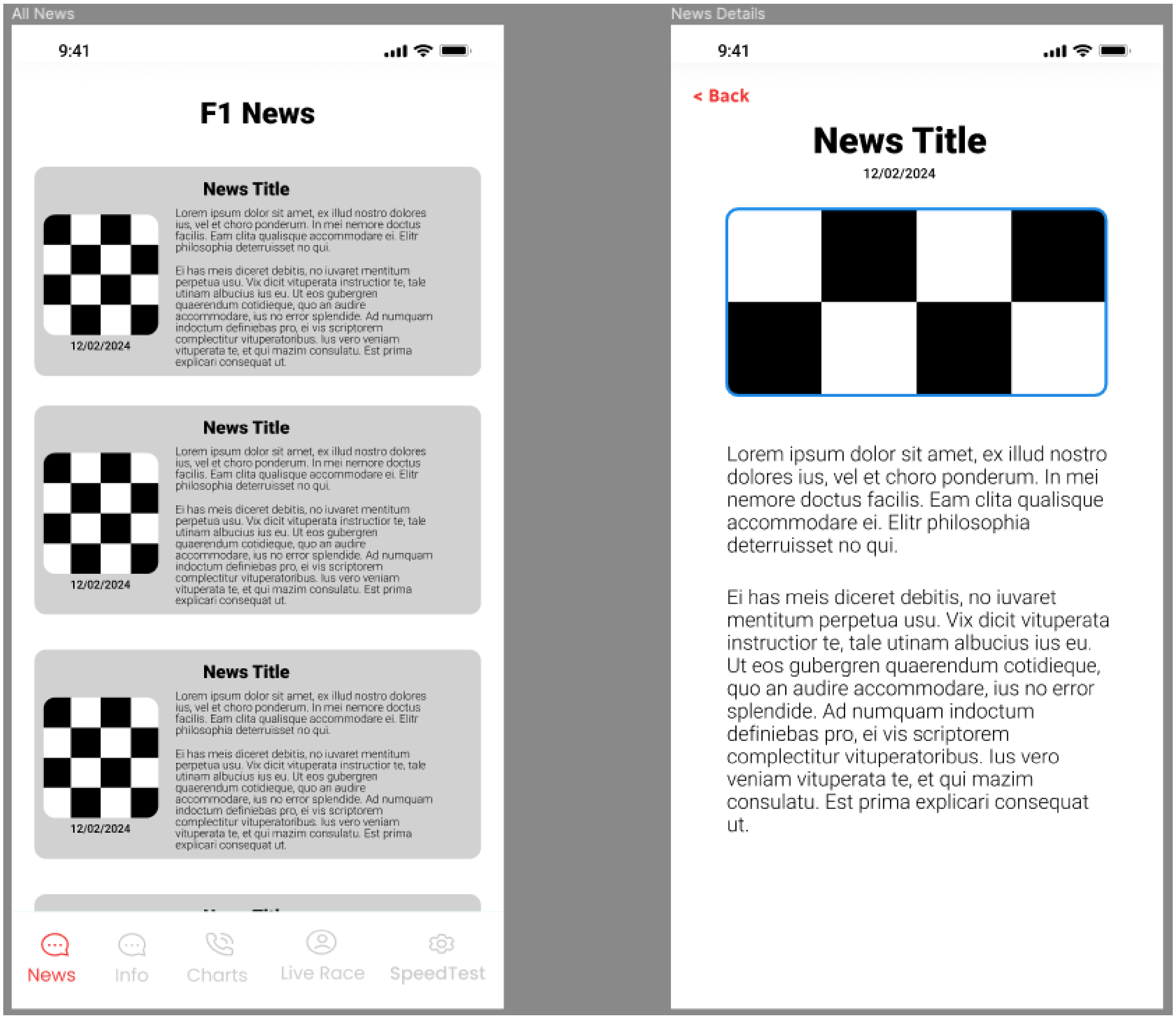

Ilustración 4.3: Diseño de pantallas de noticias de 'F1 Fan App'.

En la Ilustración 4.3, se muestra una captura del diseño de las pantallas de visualización de noticias.

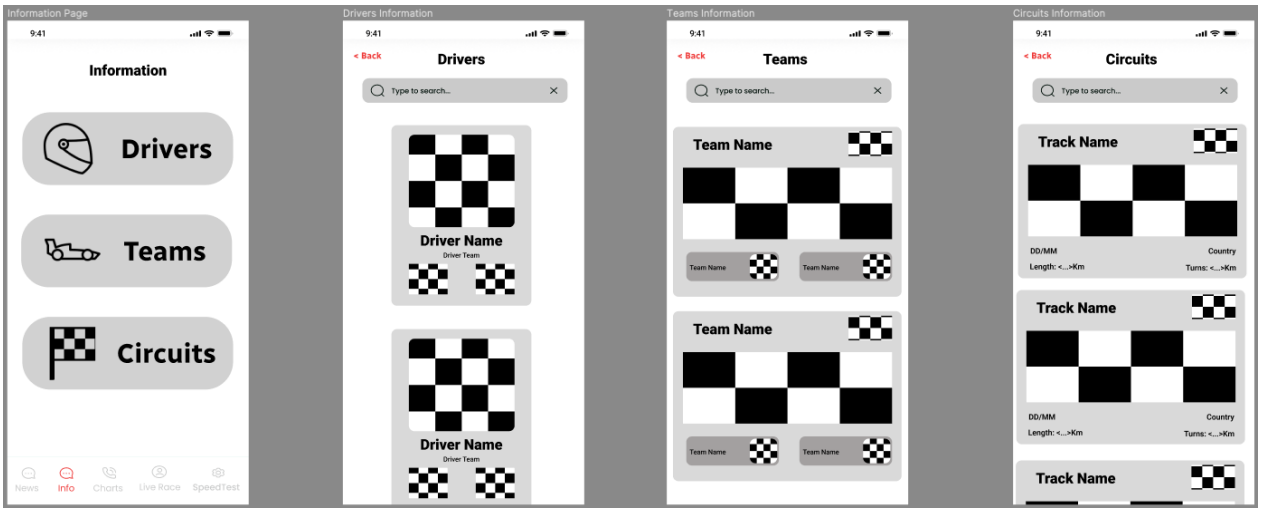

Ilustración 4.4: Diseño de pantallas de información de 'F1 Fan App'.

En la Ilustración 4.4, se muestra una captura del diseño de las pantallas de visualización de información sobre los Pilotos, Equipos y Circuitos.

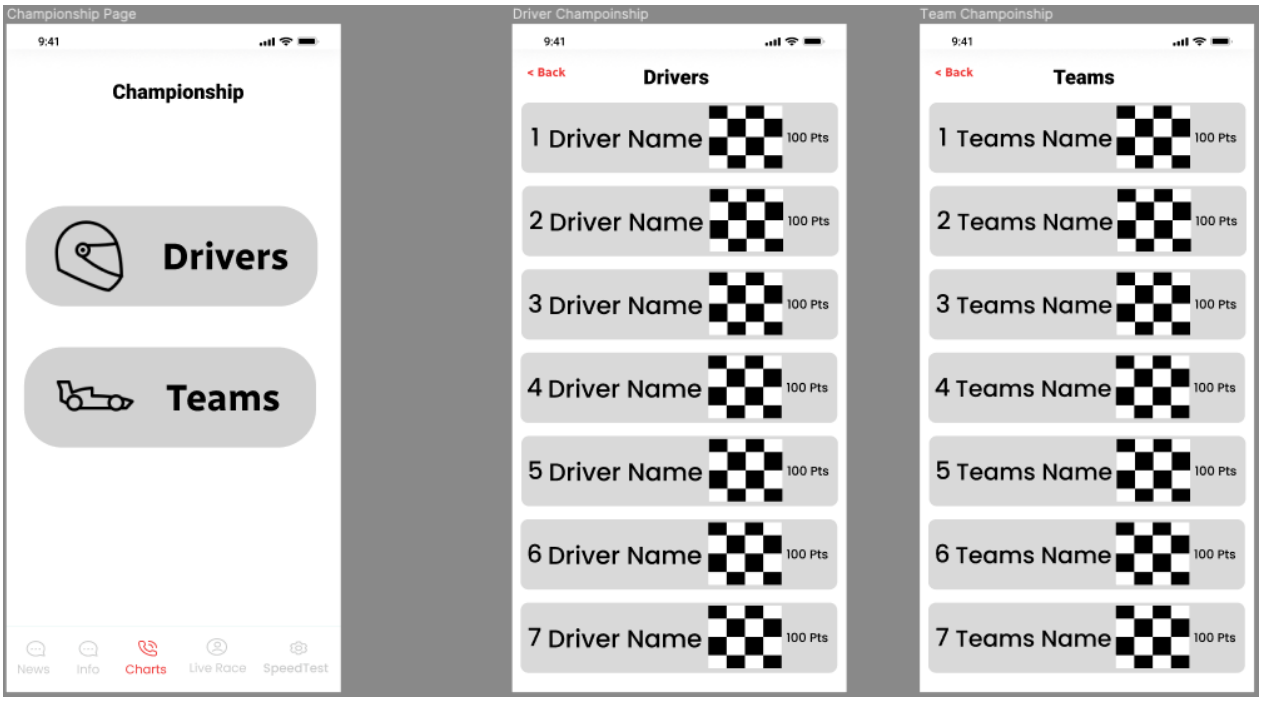

Ilustración 4.5: Diseño de pantallas de campeonatos de 'F1 Fan App'.

En la Ilustración 4.5, se muestra una captura del diseño de las pantallas de visualización de estado del campeonato de Pilotos y Equipos, ordenado por puntos.

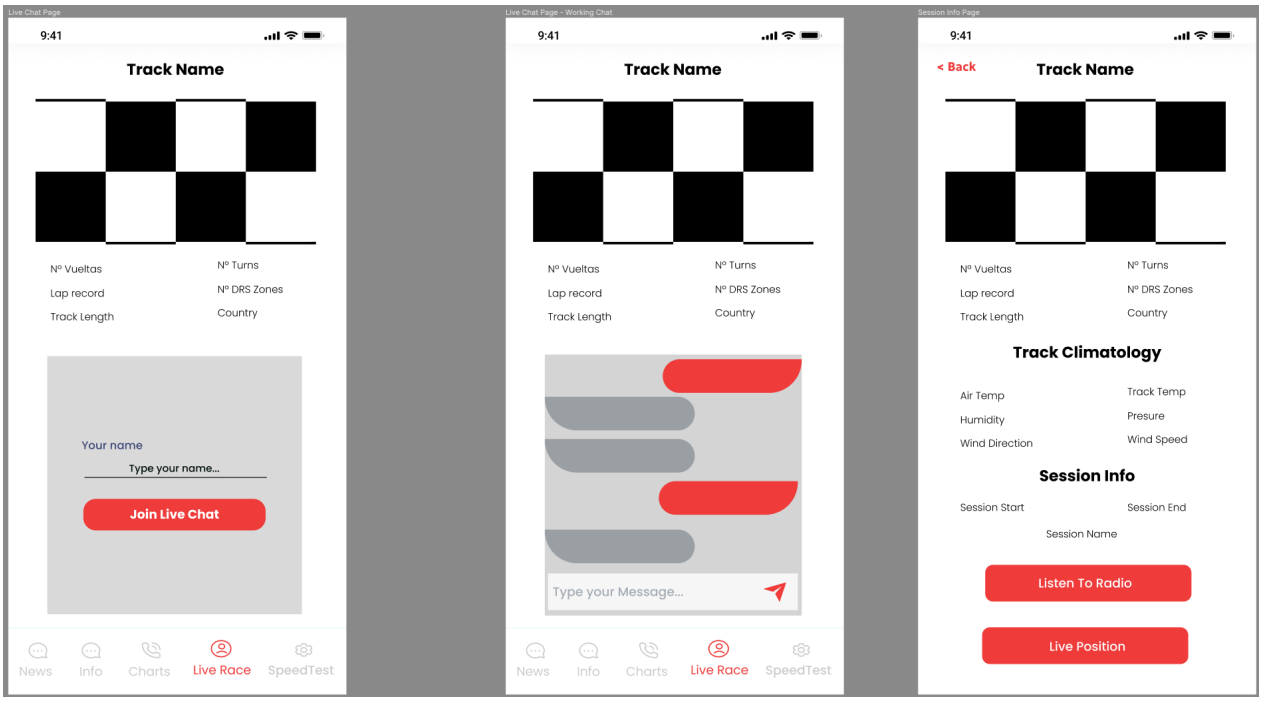

Ilustración 4.6: Diseño de 3 de las pantallas de la sección en Directo de 'F1 Fan App'.

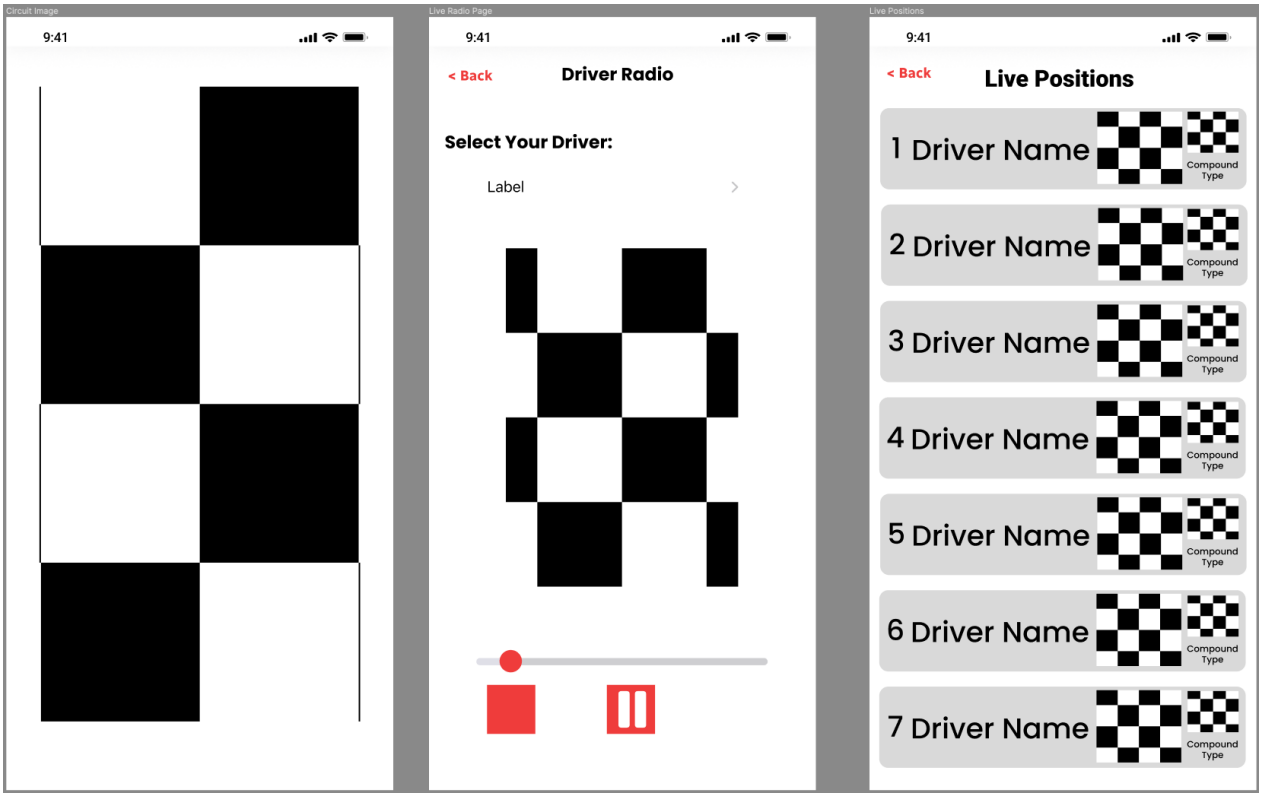

Ilustración 4.7: Diseño de las otras 3 pantallas de la sección en Directo de 'F1 Fan App'.

En las Ilustraciones 4.6 & 4.7 se muestra una captura del diseño de las pantallas de información en directo, donde se encuentra información del circuito actual o siguiente, el chat, información climatoológica del circuito, información de la sesióna actual, imágen del circuito, pantalla de escucha de última radio de los pilotos y posiciones en directo.

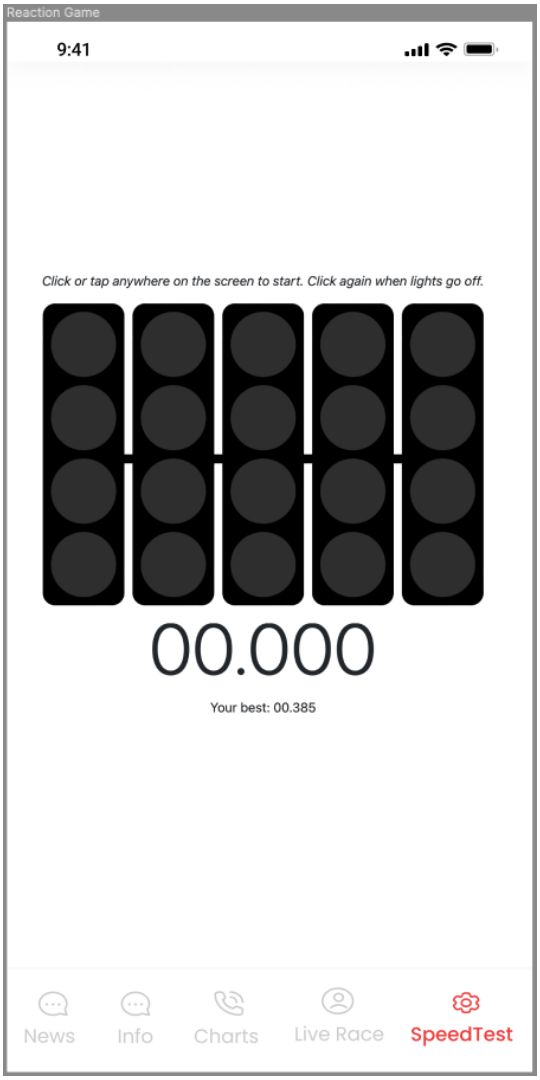

Ilustración 4.8: Diseño de la pantalla de juego de 'F1 Fan App'.

En la Ilustración 4.8, se muestra una captura del diseño de la pantalla juego de tiempo de reacción, inspirado en el semáforo oficial de la F1.

4.2.2. Diseño de la Interfaz Web para Manejo de la Base de Datos

A continuación se mostrará los diseños de las pantallas para la Interfaz Web. También puede accederse al diseño original en **Figma** haciendo click aquí.

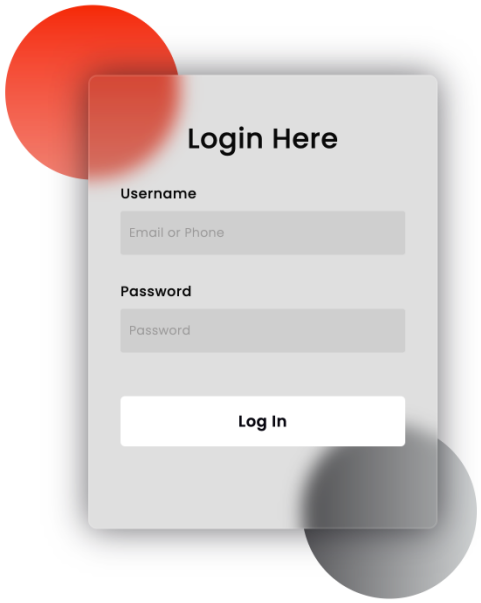

Ilustración 4.9: Diseño de pantalla de Login de Interfaz Web.

En la Ilustración 4.9, se muestra una captura del diseño de la pantalla de login para la interfaz web.

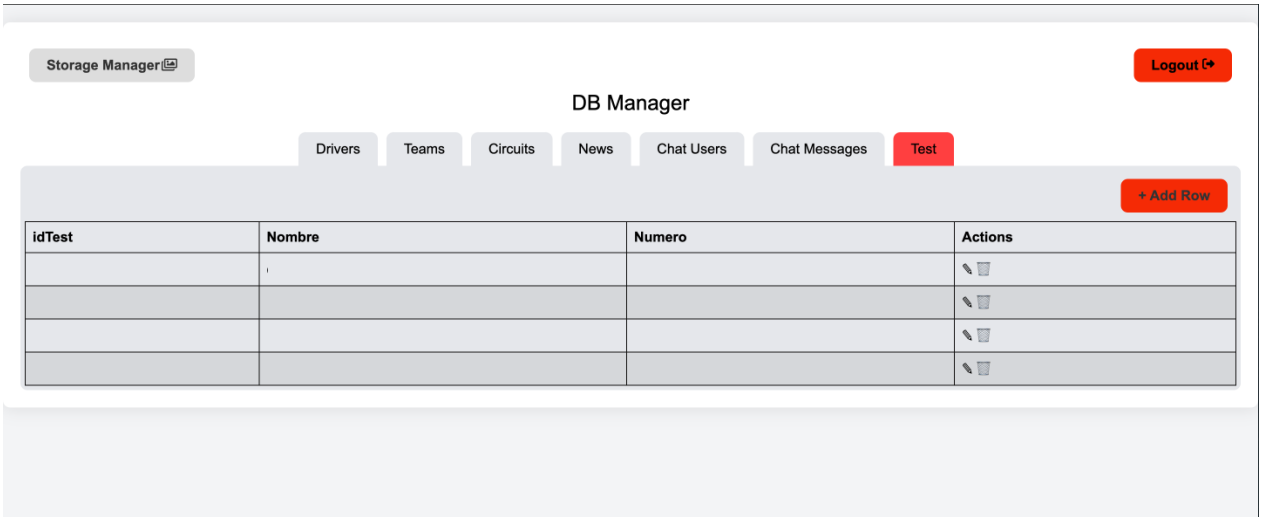

Ilustración 4.10: Diseño de pantalla de datos MYSQL de Interfaz Web.

En la Ilustración 4.10, se muestra una captura del diseño de la pantalla principal donde se visualizan los datos de la base de datos MYSQL, mostrando las tablas disponibles, los datos de la tabla, el botón para cambiar a la pantalla de Firebase Storage (Storage Manager) y las acciones de editar, borrar y añadir los datos de la tabla

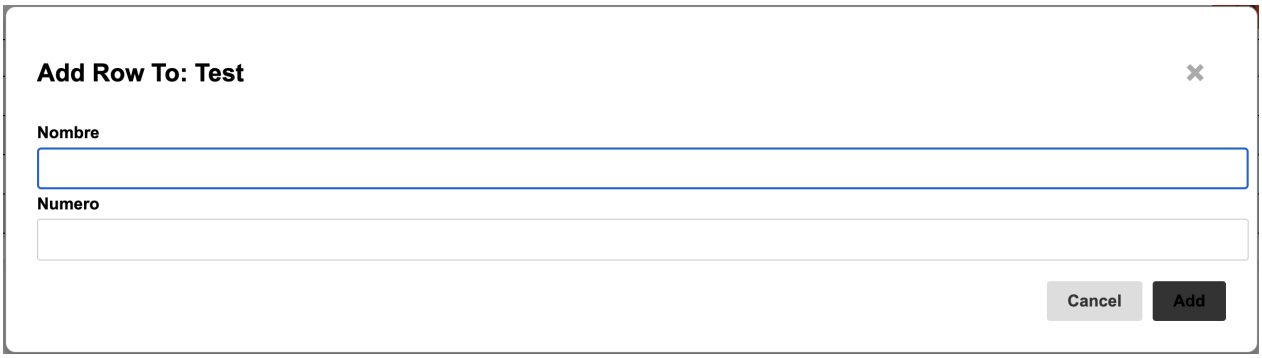

Ilustración 4.11: Diseño de modal de Añadir datos en MYSQL en la Interfaz Web.

En la Ilustración 4.11, se muestra una captura del diseño del modal para añadir datos a una tabla en la base de datos MYSQL. En el modal se cargan los campos necesarios a rellenar para cada tabla.

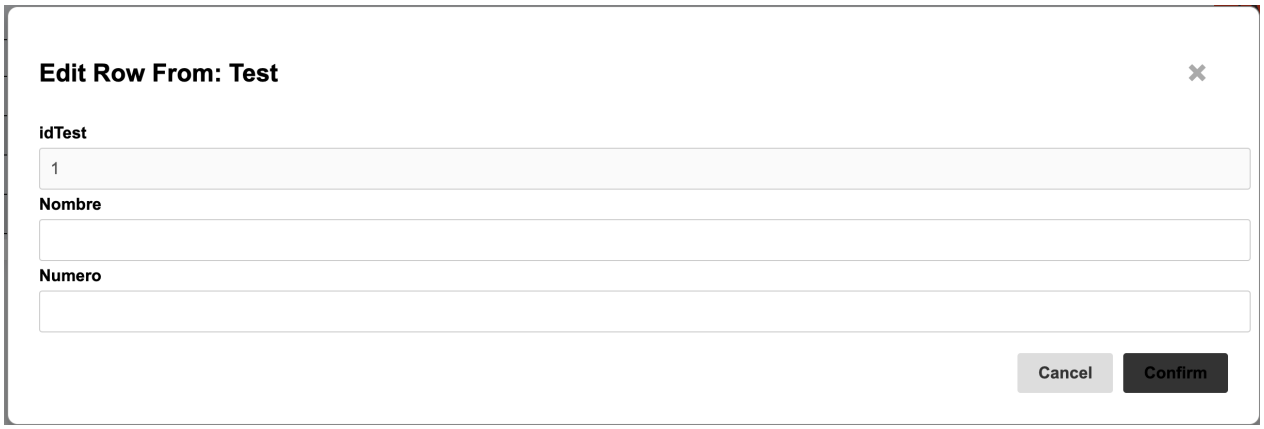

Ilustración 4.12: Diseño de modal de Editar datos en MYSQL en la Interfaz Web.

En la Ilustración 4.12, se muestra una captura del diseño del modal para editar datos a una tabla en la base de datos MYSQL. Al igual que en el modal de Añadir, en el modal se cargan los campos necesarios a rellenar para cada tabla.

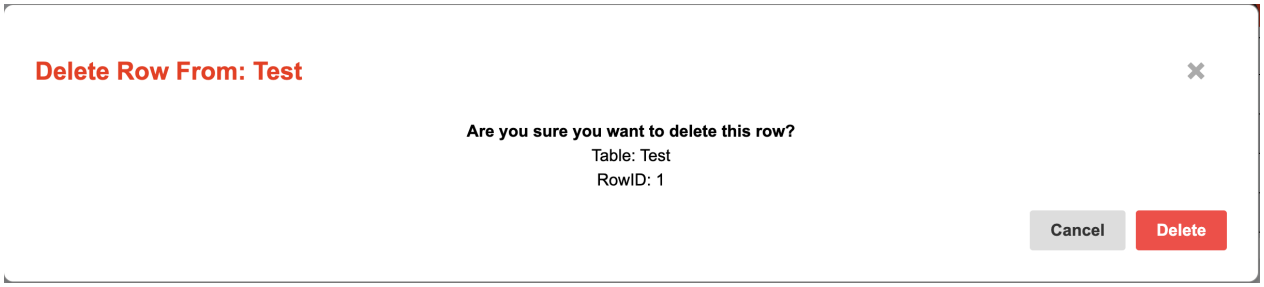

Ilustración 4.13: Diseño de modal de Eliminar datos en MYSQL en la Interfaz Web.

En la Ilustración 4.13, se muestra una captura del diseño del modal para eliminar datos a una tabla en la base de datos MYSQL. En este modal se muestran colores rojos, para generar un estado de alerta en el usuario, ya que se están borrando datos de una base de datos.

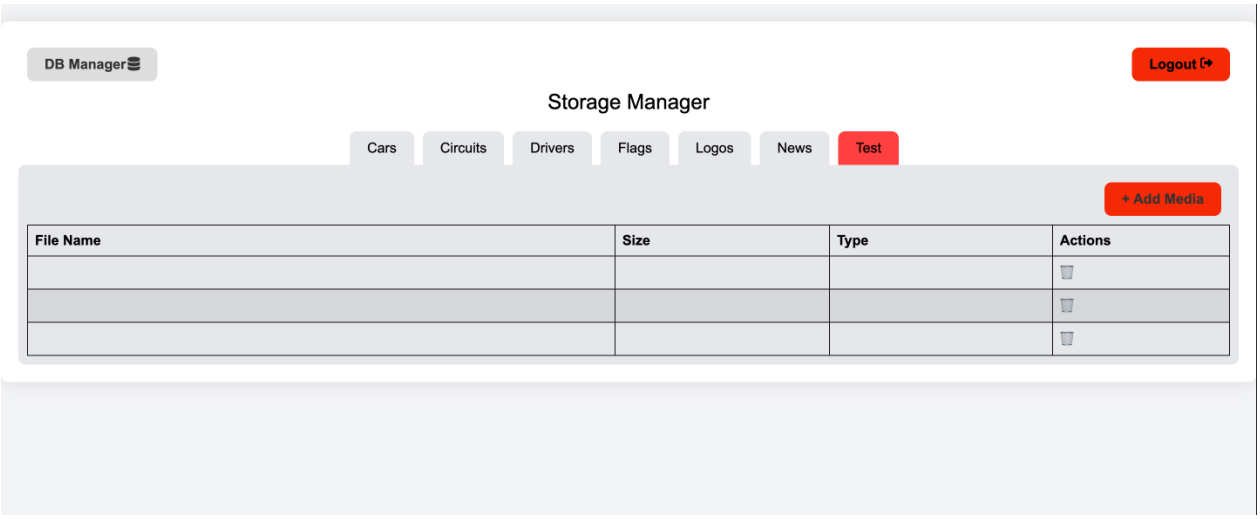

Ilustración 4.14: Diseño de pantalla de datos Firebase Storage de Interfaz Web.

En la Ilustración 4.14, se muestra una captura del diseño de la pantalla principal donde se visualizan los datos de Firebase Storage, mostrando las colecciones disponibles, los archivos multimedia en cada colección, el botón para cambiar a la pantalla de MYSQL (DB Manager) y las acciones de borrar, visualizar y añadir los archivos multimedia.

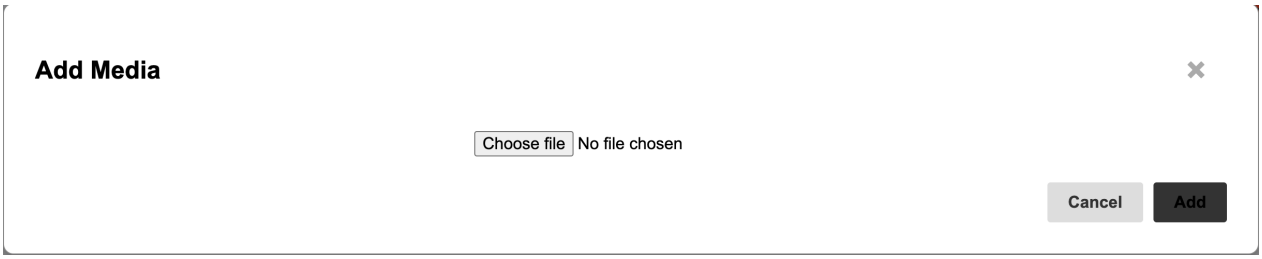

Ilustración 4.15: Diseño de modal de Añadir archivos multimedia en Firebase Storage en la Interfaz Web.

En la Ilustración 4.15, se muestra una captura del diseño del modal para añadir archivos multimedia a una colección en Firebase Storage.

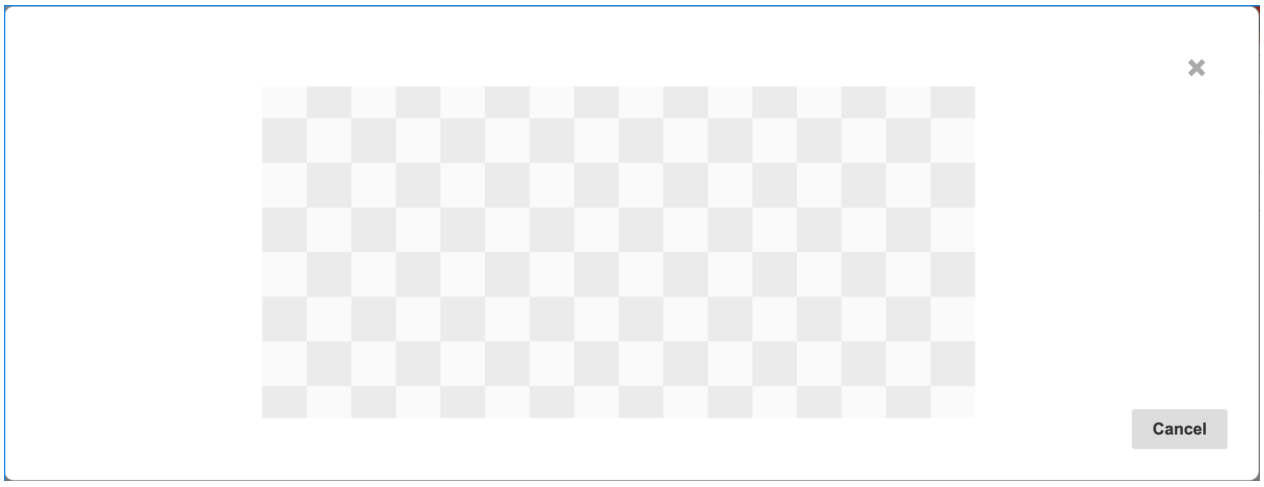

Ilustración 4.16: Diseño de modal de Visualizar archivos multimedia en Firebase Storage en la Interfaz Web.

En la Ilustración 4.16, se muestra una captura del diseño del modal para visualizar archivos multimedia de una colección en Firebase Storage.

Ilustración 4.17: Diseño de modal de eliminar archivos multimedia en Firebase Storage en la Interfaz Web.

En la Ilustración 4.17, se muestra una captura del diseño del modal para eliminar archivos multimedia de una colección en Firebase Storage.

4.2.3. Diseño de la Base de Datos - MySQL

En esta sección se presentará el diseño de la base de datos **MySQL** utilizada en el desarrollo de la aplicación iOS **"F1 Fan App"** y su correspondiente Interfaz Web. Se explicarán las decisiones de diseño, se mostrarán los esquemas de las tablas y se describirá la lógica de

las relaciones entre ellas. Este diseño garantiza la integridad de los datos y la escalabilidad necesaria para esta aplicación.

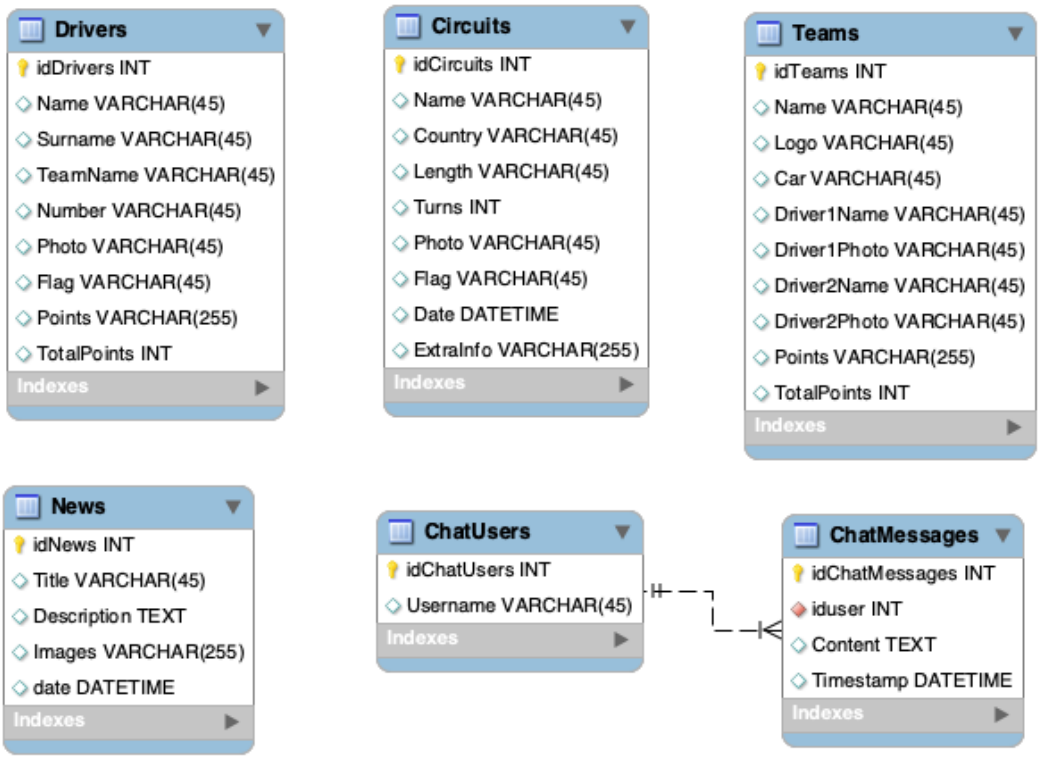

Ilustración 4.18: Diagrama EER(Enhanced Entity-Relationship) de la base de datos MySQL.

Como se observa en el diagrama EER [15] de la ilustración 4.18, la base de datos está compuesta por seis tablas principales: **Drivers**, **Circuits**, **Teams**, **ChatMessages**, **News**, y **ChatUsers**. En el ANEXO I se tratará la descripción de las tablas, sus datos y relaciones.

4.2.4. Diseño de la Arquitecura del proyecto

La arquitectura del proyecto **"F1 Fan App"** se compone de varios componentes interconectados que trabajan juntos para proporcionar una experiencia completa y funcional tanto en la Aplicación Móvil como en la Interfaz Web. En la ilustración 4.19, se observa un diagrama que muestra la arquitectura general del proyecto.

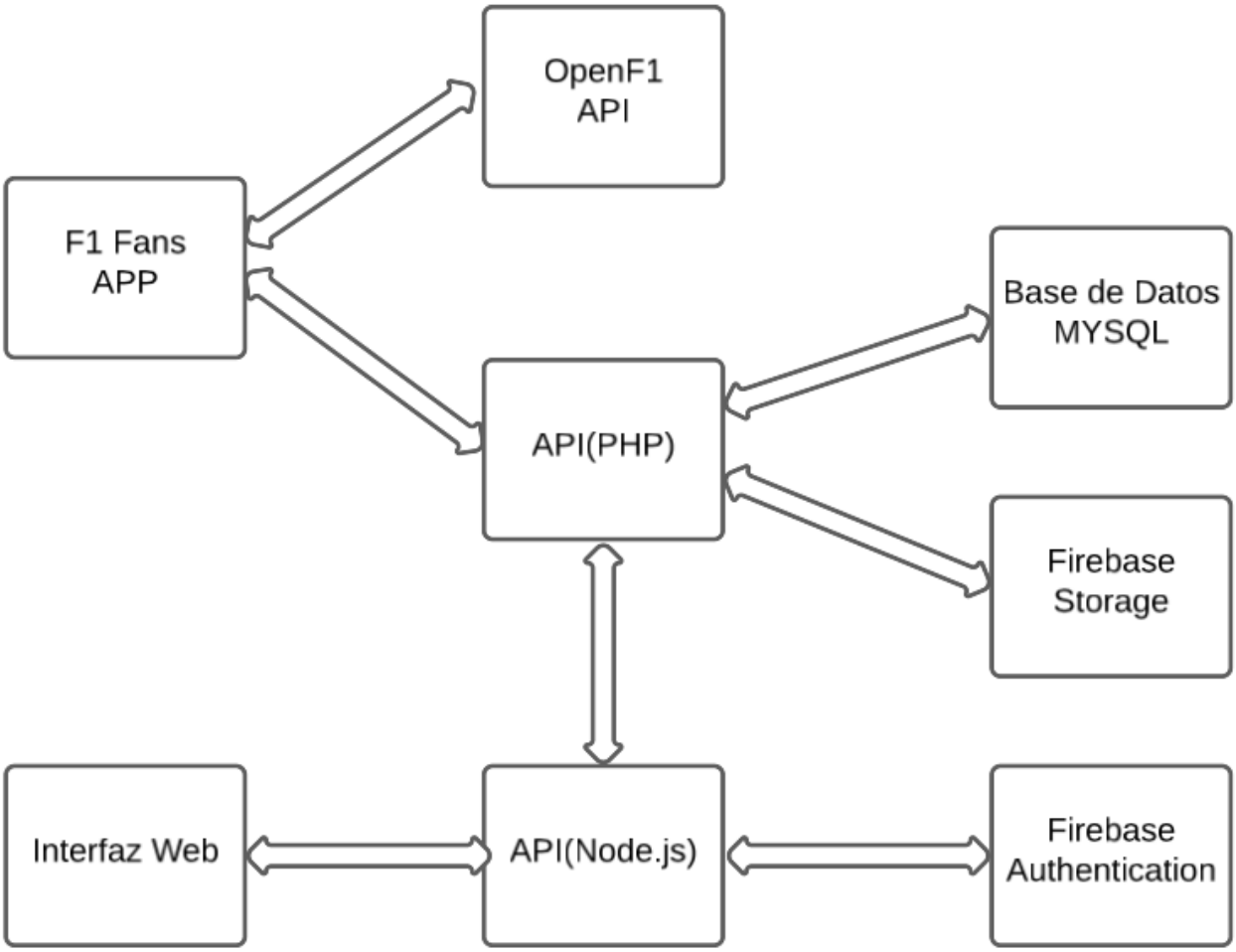

Ilustración 4.19: Diagrama del diseño de la Arquitectura del proyecto.

La arquitectura del diagrama 4.19 no sigue estrictamente un patrón de presentación como MVC, MVP o MVVM. En su lugar, se centra en la integración de servicios y la separación de responsabilidades a través de APIs, lo cual es más típico en arquitecturas basadas en microservicios o integraciones de backend.

Descripción de los Componentes

- **F1 Fan App**
  - La aplicación móvil desarrollada para usuarios de iOS. Esta aplicación permite a los usuarios acceder a información en tiempo real sobre los pilotos, equipos, y circuitos de Fórmula 1, además de interactuar con otras funcionalidades como noticias y chat en vivo entre otras.

- **Interfaz Web**
  - Una plataforma accesible desde cualquier navegador que ofrece manejar los datos de la base de datos **MySQL** y **Firebase Storage**.
- **API (PHP)**
  - El núcleo del backend, desarrollado en **PHP**, que maneja la lógica de negocio y se comunica con la base de datos **MySQL**, el almacenamiento en **Firebase Storage**. Esta API es responsable de procesar las solicitudes de la aplicación móvil y la interfaz web, asegurando que los datos sean entregados de manera precisa y eficiente.
- **API (Node.js)**
  - Otro componente del backend desarrollado en **Node.js** que trabaja en conjunto con la API en **PHP** para manejar ciertas funcionalidades específicas, proporcionando flexibilidad y escalabilidad al sistema. Esta API también se comunica con Firebase Authentication para la gestión de usuarios y seguridad de la aplicación web.
- **Base de Datos MySQL**
  - El sistema de gestión de bases de datos relacional utilizado para almacenar la información de los `pilotos, equipos, circuitos, noticias` y `mensajes de chat`. **MySQL** proporciona una estructura sólida y eficiente para la gestión de datos.
- **Firebase Storage**
  - Utilizado para almacenar recursos multimedia como fotos de `pilotos, equipos y circuitos`, así como imágenes de `noticias`. **Firebase Storage** ofrece una solución escalable y segura para el almacenamiento de estos archivos multimedia.
- **Firebase Authentication**
  - Servicio utilizado para gestionar la autenticación de los usuarios tanto en la

aplicación móvil como en la interfaz web. **Firebase Authentication** facilita el manejo de usuarios, ofreciendo opciones de inicio de sesión con correo electrónico, redes sociales y otros proveedores de identidad.

- **OpenF1 API**

  - Una API externa no oficial de la Fórmula 1 que proporciona datos actualizados, incluyendo posiciones actuales, radios, datos de neumáticos, sesiones y climatología de pista entre otros [3]. Esta API se integra con el backend para asegurar que la información presentada a los usuarios esté siempre actualizada.

Esta arquitectura modular y bien estructurada asegura que el sistema sea escalable, mantenible y capaz de manejar un gran volumen de usuarios y datos de manera eficiente.

4.2.5. Diseño Creativo

En esta sección se describen los aspectos clave del diseño creativo de la aplicación F1 Fan App, incluyendo la paleta de colores, el logotipo y el GIF utilizado como loader en la aplicación y la interfaz web.

Paleta de Colores

En la ilustración 4.20 se puede observar la paleta de colores [9] seleccionada para la aplicación. Esta está inspirada en los colores tradicionales de la Fórmula 1, los cuales son el Negro, Rojo y Blanco. A parte de estos colores se ha elegido un color grisáceo. A continuación se explicará que uso se le ha dado a cada color dentro de la aplicación iOS y Interfaz Web:

- **Negro (#0F0E0F)**: El color negro se ha usado principalmente como color para todos los textos, tanto como para títulos, subtítulos y párrafos entre otros.

- **Blanco (#FEFEFE)**: El blanco se ha usado como color base para el fondo de las aplicaciones.

- **Gris (#DCDCF0)**: El gris se ha usado como color principal de los contenedores que encapsulen una estructura de datos, como en la cartas de información de Pilotos, Equipos, Circuitos, Noticias y fondo del Chat.

- **Rojo (#F2403F)**: El rojo es el color de acentuación principal de la aplicación. Este se muestra en los botones de vuelta atrás a otra pantalla, en los iconos del tab inferior, botón de enviar en el chat y en varios sitios más.

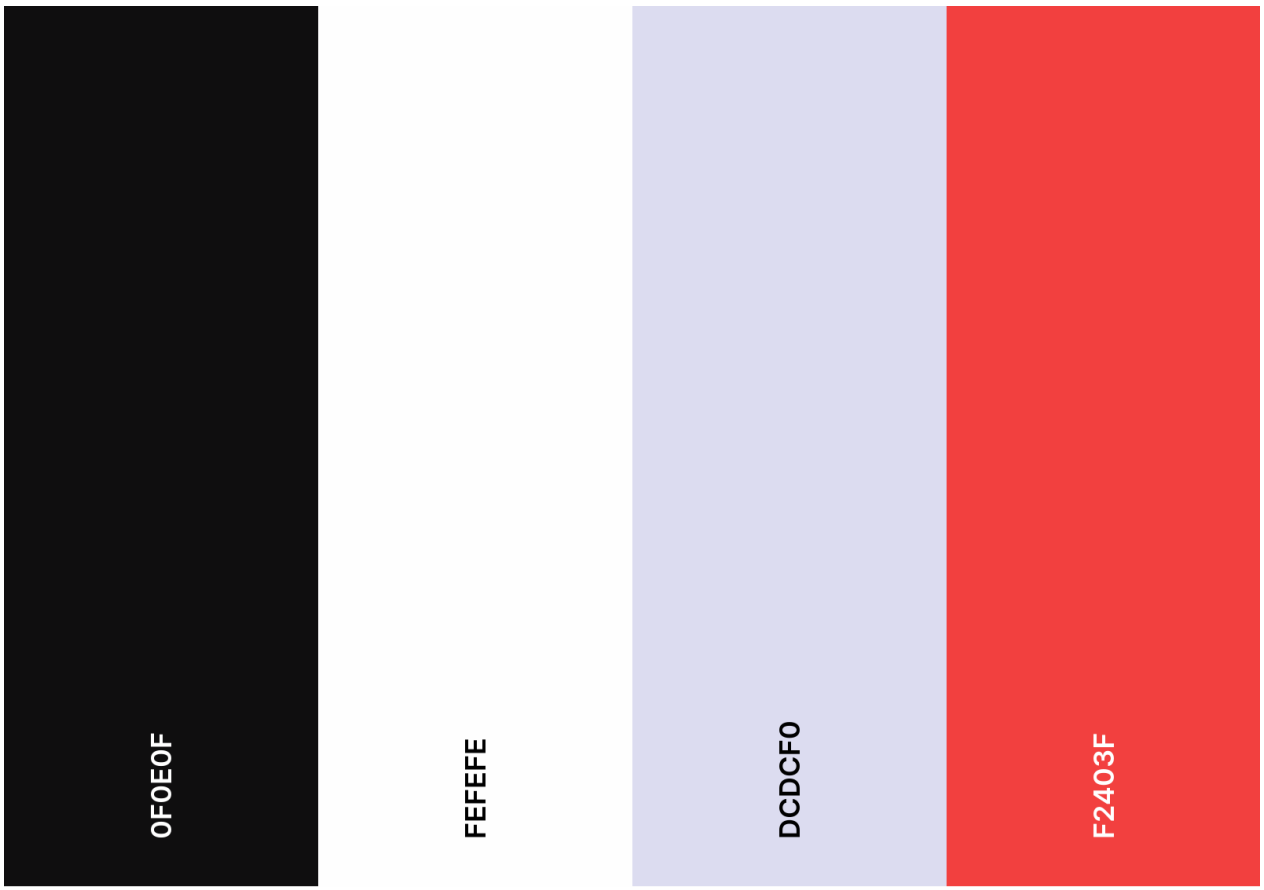

Ilustración 4.20: Paleta de colores usada en las aplicaciones

Logotipo de la Aplicación

En la ilustración 4.21 podemos observar el logotipo de la aplicación **"F1 Fan"** [11]. Este logotipo de la aplicación muestra a una persona agitando una bandera de cuadros, representando un símbolo icónico del mundo de las carreras, junto al nombre de la aplicación **"F1 Fan"**. La Fuente usada para el logotipo se llama `"Formula1 Display Regular"`, el cual da un aspecto agresivo y futurista, justo la mismo que transmiten los aspectos de los más recientes coches de Formula 1.

Ilustración 4.21: Logotipo de 'F1 Fan' App

Loader

Los tiempos de espera en una aplicación puede matar el entusiasmo y ganas de usar una aplicación, por lo cual es importante usar pantallas de carga o loaders para hacer ese tiempo de espera más entretenido y ameno. En el caso de **"F1 Fan"** App, se ha diseñado un loader [1], el cual se usa tanto en la aplicación iOS como en la Interfaz Web. En la ilustración 4.22, se muestra el loader usado en las aplicaciones. Como loader, se utiliza un GIF que incorpora dos de los colores primarios de la paleta de colores(4.20), estos son el gris(#DCDCF0) y el rojo(#F2403F).

Ilustración 4.22: GIF utilizado como loader en la aplicación e interfaz web

4.3. Desarrollo

4.3.1. Lenguajes usados

Para el desarrollo del proyecto, se seleccionaron diversos lenguajes de programación y tecnologías adecuadas a las necesidades específicas de cada componente:

- ✔ **Swift (iOS Aplicación - F1 Fan App):** Swift fue elegido para el desarrollo de la aplicación móvil debido a su integración nativa con iOS y su alto rendimiento. Además, Swift es el lenguaje preferido por Apple para el desarrollo de aplicaciones en su ecosistema, proporcionando acceso directo a las APIs y funcionalidades del sistema operativo.

- ✔ **PHP (API):** PHP se utilizó para desarrollar la API que alimenta la aplicación móvil con datos desde la base de datos **MySQL**, como información de pilotos, equipos, circuitos y noticias actualizadas entre otros. PHP es ampliamente compatible con bases de datos relacionales y ofrece robustez en la gestión de datos para aplicaciones web y móviles.

- ✔ **HTML, CSS, JS (Interfaz web para la Base de Datos):** Se optó por utilizar **HTML**, **CSS** y **JavaScript** para desarrollar la interfaz web que permite la gestión y visualización de la base de datos de la aplicación. Estos lenguajes proporcionan una base sólida y ampliamente soportada para la construcción de interfaces web. **HTML** define la estructura del contenido, **CSS** se encarga del diseño y la presentación, y **JavaScript** proporciona interactividad y dinamismo a la interfaz. Se eligieron estos lenguajes por su capacidad para crear interfaces eficientes y personalizadas, adaptándose a las necesidades específicas del proyecto sin agregar la complejidad adicional que **React** o **Next.js** podrían implicar en este contexto particular.

- ✔ **MySQL (Base de Datos): MySQL** fue elegido como el sistema de gestión de bases de datos para almacenar y administrar eficientemente los datos del proyecto. Se optó por **MySQL** debido a su capacidad para almacenar datos en tablas y establecer relaciones entre ellas, aprovechando su naturaleza como una base de datos relacional. Además, **MySQL** es reconocido por su alto rendimiento, escalabilidad y fiabilidad, lo cual lo hace ideal para entornos con elevado tráfico de datos.

- ✔ **Node.js (otra API): Node.js** se utilizó para desarrollar una API adicional que facilita la integración de datos desde la base de datos y la conexión con Firebase Authentication, empleada específicamente para la interfaz web. **Node.js** se destaca por su capacidad para facilitar una comunicación rápida y eficiente con diversos servicios y APIs ex-

ternas, lo cual lo convierte en una elección adecuada para aplicaciones que requieren integraciones complejas y dinámicas.

La elección de estos lenguajes y tecnologías se basó en sus capacidades específicas para cumplir con los requisitos funcionales y no funcionales del proyecto, asegurando un desarrollo eficiente, escalable y orientado a ofrecer una experiencia de usuario óptima tanto en la aplicación móvil como en la interfaz web. Además, se decidió explorar diferentes tecnologías para las APIs, utilizando PHP y Node.js, con el objetivo de evaluar y comparar su desempeño y adecuación a las necesidades específicas de integración y comunicación de datos del proyecto.

4.3.2. Entornos de Desarrollo

Para el desarrollo del proyecto, se utilizaron los siguientes entornos y herramientas específicas:

- ✔ **Xcode (App iOS):** Xcode es el entorno de desarrollo integrado (IDE) nativo para aplicaciones iOS, proporcionado por Apple. Es una herramienta integral que incluye todo lo necesario para diseñar, desarrollar, compilar y depurar aplicaciones para dispositivos que ejecutan iOS, macOS, watchOS y tvOS [6]. Xcode ofrece un conjunto completo de herramientas, como el Interface Builder para diseñar interfaces de usuario, y simuladores para probar aplicaciones en diferentes dispositivos y versiones de iOS. Además, su integración con Swift y Objective-C permite un desarrollo ágil y eficiente, asegurando que las aplicaciones sean optimizadas para el rendimiento y la compatibilidad con el ecosistema de Apple.

- ✔ **Visual Studio Code (Ambas APIs - PHP & Node.js):** Visual Studio Code (VSC) se empleó para el desarrollo de ambas APIs, tanto la API en PHP como la API en Node.js. VSC es conocido por su versatilidad, soporte extensivo para diversos lenguajes de programación y una amplia gama de extensiones que facilitan el desarrollo y la depuración de aplicaciones web y backend.

- ✔ **WebStorm (Interfaz Web):** WebStorm se utilizó como entorno de desarrollo integrado (IDE) para la creación de la interfaz web del proyecto. Está optimizado para el desarrollo web, proporcionando herramientas avanzadas para la edición de código, depuración y administración de proyectos web complejos [13].

- ✔ **Figma (Diseño de App iOS & Interfaz Web):** Figma fue la herramienta principal utilizada para el diseño tanto de la aplicación móvil de iOS como de la interfaz web. Permite la colaboración en tiempo real y facilita la creación de diseños interactivos

y prototipos que ayudan a visualizar y validar la experiencia de usuario antes de la implementación.

Cada uno de estos entornos y herramientas fue seleccionado por sus capacidades específicas para apoyar eficazmente el desarrollo, diseño y gestión del proyecto, asegurando coherencia y eficiencia en todas las etapas del proceso de desarrollo de software.

4.3.3. Control de Versiones

Para el control de versiones del proyecto, se utilizaron repositorios independientes para la Aplicación iOS(F1 Fan App) y la Interfaz Web. GitHub fue la plataforma seleccionada para alojar estos repositorios y gestionar el control de versiones del código fuente.

GitHub es una plataforma de desarrollo colaborativo que utiliza Git para el control de versiones. Permite a los desarrolladores trabajar juntos en proyectos desde cualquier lugar, ofreciendo herramientas para la gestión de código, seguimiento de problemas y colaboración en equipo. GitHub facilita la revisión de código, la integración continua y la implementación de prácticas de desarrollo ágil.

Para este proyecto, se utilizó una sola rama **master** ya que no se planifica un lanzamiento a producción. La rama master contenía la versión más actual y estable del código en desarrollo. Además, se optó por mantener una sola rama porque el desarrollo fue realizado por una sola persona, eliminando la necesidad de combinar ramas creadas por diferentes desarrolladores. Tampoco se requerían ramas de hotfixes dado que no se esperaba manejar correcciones rápidas de errores críticos.

En caso de que se considerara un lanzamiento a producción, sería necesario implementar una estrategia de ramas más estructurada.

4.4. Pruebas

4.4.1. Pruebas de Usabilidad

Para evaluar la efectividad y facilidad de uso de la Interfaz Web y la aplicación móvil 'F1 Fan', se llevaron a cabo pruebas de usabilidad con usuarios reales. En estas pruebas, se invitó a los usuarios a interactuar libremente con ambas plataformas, permitiéndoles explorar las funcionalidades y realizar tareas típicas sin guía previa.

Durante la interacción, se observó a los usuarios para identificar posibles puntos de fricción, confusión o áreas de mejora. Al finalizar la sesión, se les pidió que compartieran su feedback espontáneamente sobre su experiencia general, destacando aspectos positivos y negativos.

Las observaciones y comentarios recogidos de estas pruebas se analizaron y utilizaron para realizar mejoras iterativas en el diseño y la funcionalidad de la aplicación y la interfaz web. Este enfoque permitió adaptar el proyecto a las necesidades y expectativas reales de los usuarios, incrementando así su satisfacción y la usabilidad del producto final.

Es importante destacar que, en estas pruebas, no se obligó a los participantes a completar ningún cuestionario formal. En cambio, se fomentó una retroalimentación natural y sin restricciones, lo cual proporcionó una visión más genuina y detallada sobre la experiencia del usuario.

4.4.2. Pruebas de Accesibilidad

Para asegurar que la aplicación móvil 'F1 Fan' y la Interfaz Web sean accesibles para todos los usuarios, incluyendo aquellos con discapacidades, se realizaron exhaustivas pruebas de accesibilidad.

En la aplicación 'F1 Fan', se utilizó una herramienta de Xcode denominada `'Accessibility Inspector'`. Esta herramienta permite a los desarrolladores identificar y corregir problemas de accesibilidad en sus aplicaciones. El `'Accessibility Inspector'` proporciona una serie de pruebas automáticas y manuales que verifican aspectos como la usabilidad con lectores de pantalla, la adecuación del contraste de colores, la navegabilidad mediante gestos y la adecuación de los tamaños de texto [5]. Con esta herramienta, se pudo mejorar significativa-

mente la experiencia de usuarios con discapacidades visuales, auditivas, motoras o cognitivas, asegurando que la aplicación sea más inclusiva.

En la ilustración 4.23 se puede observar un ejemplo del uso del `'Accesibility Inspector'`.

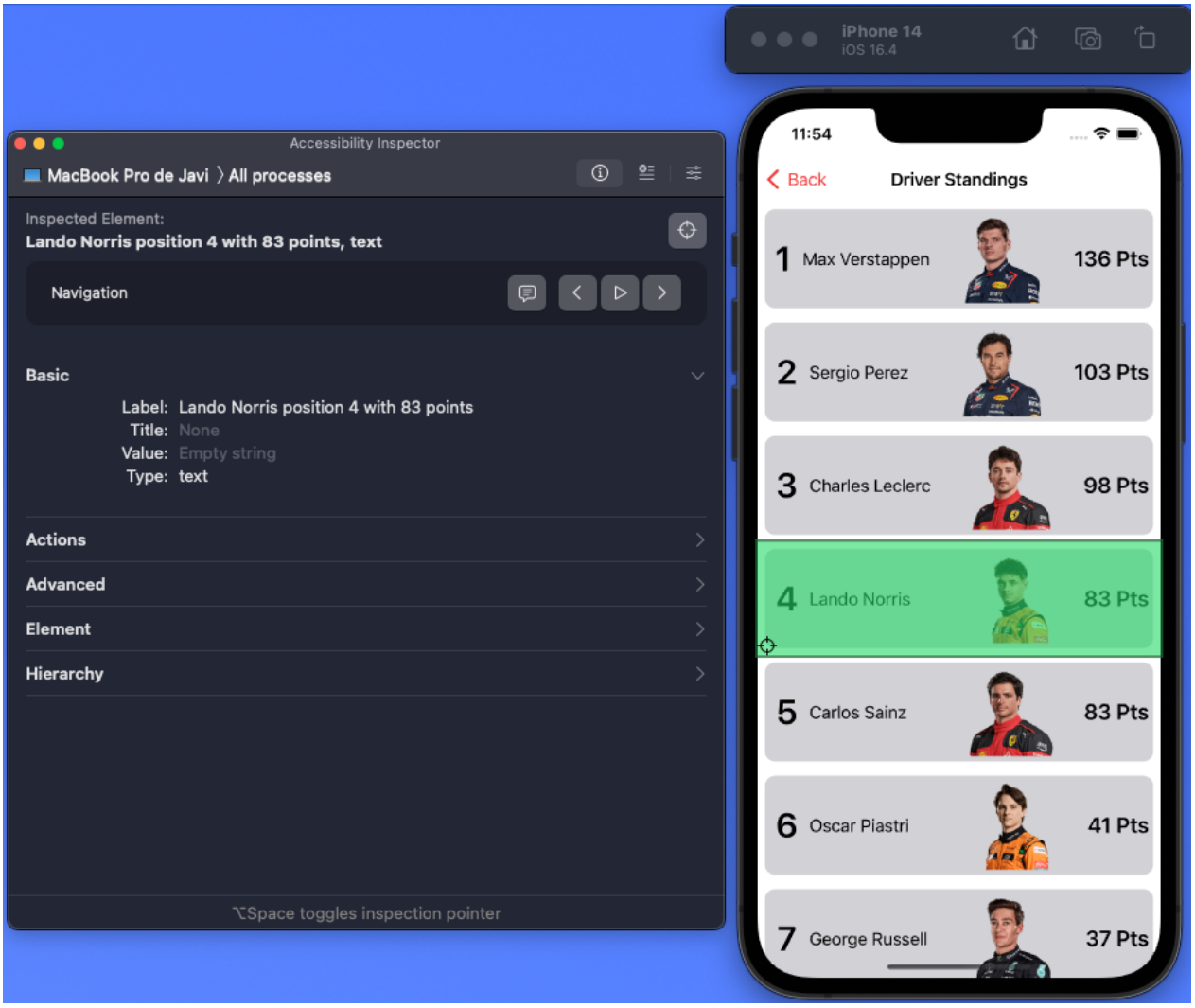

Ilustración 4.23: Ejemplo de uso de la herramienta 'Accesibility Inspector'

Para la Interfaz Web, se utilizaron las pruebas de accesibilidad de `Lighthouse`. `Lighthouse` es una herramienta automatizada de código abierto desarrollada por Google, que se ejecuta en Google Chrome. Esta herramienta evalúa las páginas web en varias categorías, incluyendo accesibilidad, y proporciona una puntuación detallada junto con recomendaciones específicas para mejoras. Las pruebas de accesibilidad de `Lighthouse` abarcan verificaciones como el correcto etiquetado de los elementos interactivos, la navegabilidad con teclado, el soporte para lectores de pantalla y la claridad visual de los contenidos. En nuestras pruebas, la Interfaz Web obtuvo una puntuación promedio de `89/100`, indicando un buen nivel de accesibilidad

con espacio para mejoras adicionales.

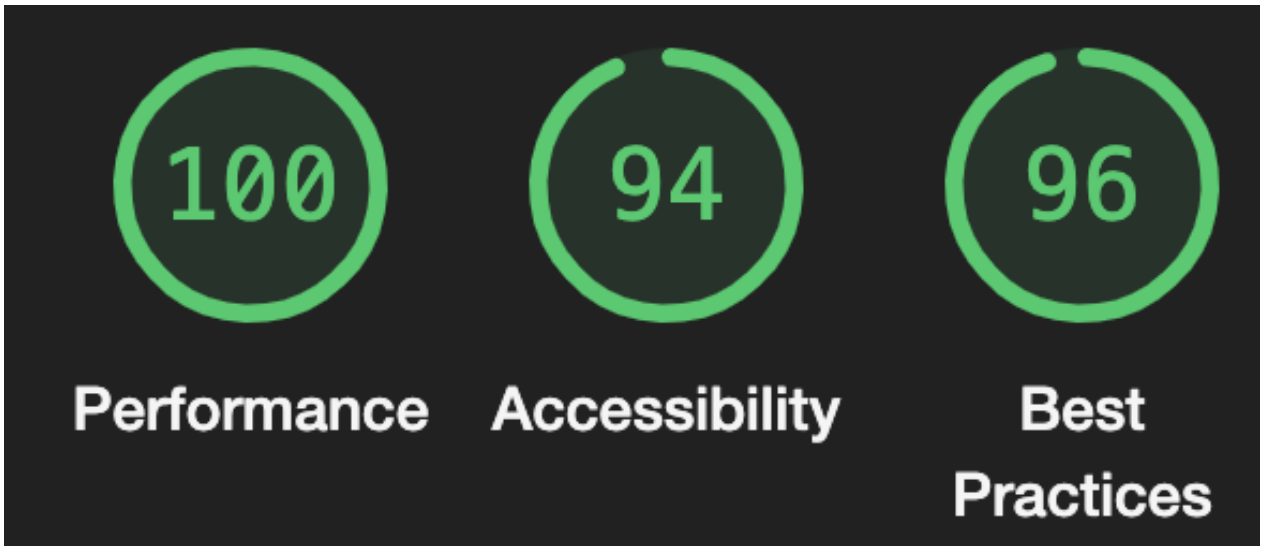

Ilustración 4.24: Resultado de la prueba de 'Lighthouse' en la pantalla de `Login` de la Interfaz Web

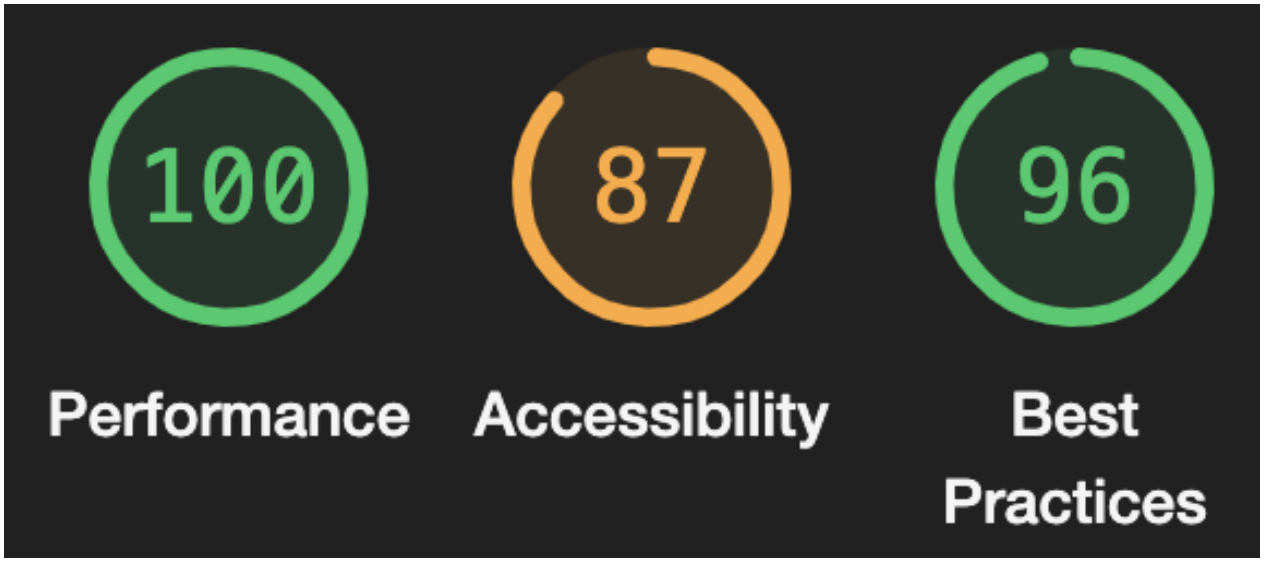

Ilustración 4.25: Resultado de la prueba de 'Lighthouse' en la pantalla de `DB Manager` de la Interfaz Web

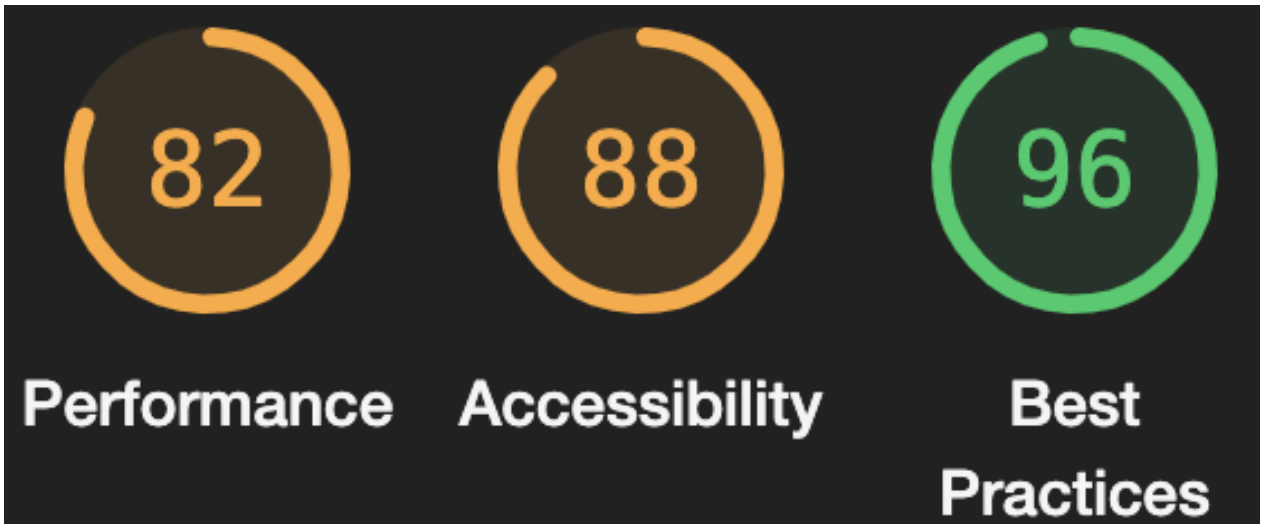

Ilustración 4.26: Resultado de la prueba de 'Lighthouse' en la pantalla de `Storage Manager` de la Interfaz Web

En las ilustraciones 4.24, 4.25 & 4.26, se muestran los resultados detallados de las pruebas realizadas con Lighthouse, destacando tanto los aspectos positivos como las áreas que requieren atención. Estas pruebas y sus resultados han sido fundamentales para guiar las mejoras en la accesibilidad del proyecto, asegurando que todos los usuarios puedan tener una experiencia óptima.

4.4.3. Pruebas de Interfaz Web con Cypress

En este apartado se presenta el uso de Cypress para realizar pruebas automatizadas en la interfaz web desarrollada para manejar las bases de datos. Cypress es una herramienta moderna y potente diseñada para pruebas de extremo a extremo (End-to-End, E2E) en aplicaciones web [4].

Estructura del Proyecto Cypress

El proyecto Cypress sigue una estructura básica que incluye directorios y archivos clave:

- **cypress/** - Directorio principal que contiene todos los archivos de configuración y pruebas.
    - **fixtures/** - Directorio para almacenar datos estáticos utilizados en las prue-

bas.

- **integration/** - Directorio donde se encuentran los archivos de prueba (‘*.spec.js‘ o ‘*.spec.ts‘).

- **plugins/** - Directorio para plugins de Cypress.

- **support/** - Directorio para archivos de soporte como comandos personalizados y configuración.

- **cypress.json** - Archivo de configuración principal de Cypress.

- **package.json** - Archivo de configuración de npm que contiene las dependencias del proyecto.

Casos de Prueba Utilizando Cypress

A continuación, se detallan tres casos de prueba específicos que se han implementado utilizando Cypress para verificar la funcionalidad de la interfaz web destinada al manejo de la base de datos.

Pruebas de Login

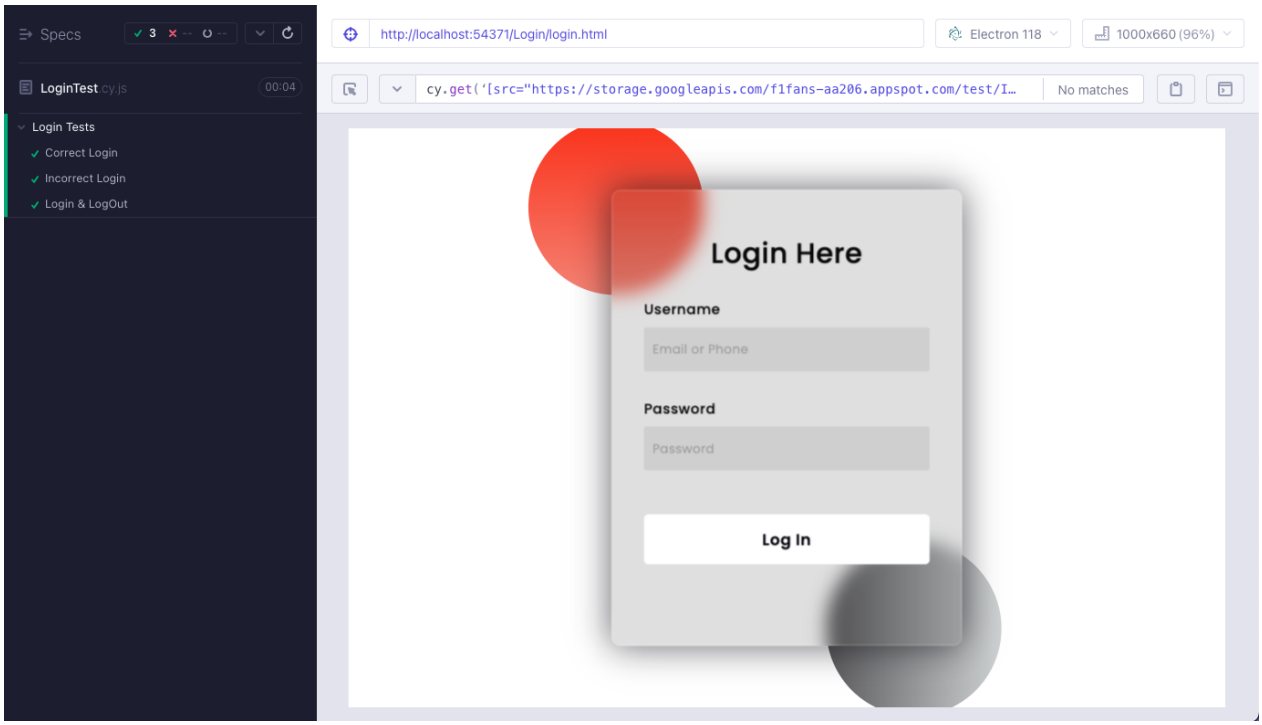

Ilustración 4.27: Resultado de la ejecución de los tests de Login.

Este test verifica el proceso de inicio de sesión en la aplicación web a través del uso correcto e incorrecto del usuario y contraseña, probando casos de inicio de sesión correctos, incorrectos y de logout una vez realizado correctamente el login. En la ilustración 4.27 podemos observar el resultado de correr los tests para las funcionalidades de LogIn.

Código del Test

En la ilustración 4.28, se muestra el contenido del archivo `'LoginTest.cy.js'` para la realización de los tests de la sección de 'Login' en la Interfaz Web.

```
describe('Login Tests', () => {

  it('Correct Login', () => {
    cy.login();
    cy.contains('DB Manager').should('be.visible');
    cy.contains('Invalid Credentials').should('not.exist');

    cy.get('#Drivers').should('be.visible');
    cy.get('#Teams').should('be.visible');
    cy.get('#Circuits').should('be.visible');
    cy.get('#News').should('be.visible');
    cy.get('#ChatUsers').should('be.visible');
    cy.get('#ChatMessages').should('be.visible');
    cy.get('#Test').should('be.visible');
  })

  it('Incorrect Login', () => {
    cy.loginIncorrect();
    cy.contains('DB Manager').should('not.exist');
    cy.contains('Invalid Credentials').should('be.visible');

    cy.get('#Drivers').should('not.exist');
    cy.get('#Teams').should('not.exist');
    cy.get('#Circuits').should('not.exist');
    cy.get('#News').should('not.exist');
    cy.get('#ChatUsers').should('not.exist');
    cy.get('#ChatMessages').should('not.exist');
    cy.get('#Test').should('not.exist');
  })

  it('Login & LogOut', () => {
    cy.login();
    cy.contains('DB Manager').should('be.visible');
    cy.contains('Invalid Credentials').should('not.exist');

    cy.get('#logoutBTN').click({force:true});
    cy.contains('DB Manager').should('not.exist');
    cy.contains('Login Here').should('be.visible');
  })
})
```

Ilustración 4.28: Contenido del archivo 'LoginTest.cy.js' para los tests de Login.

Pruebas de DB Manager

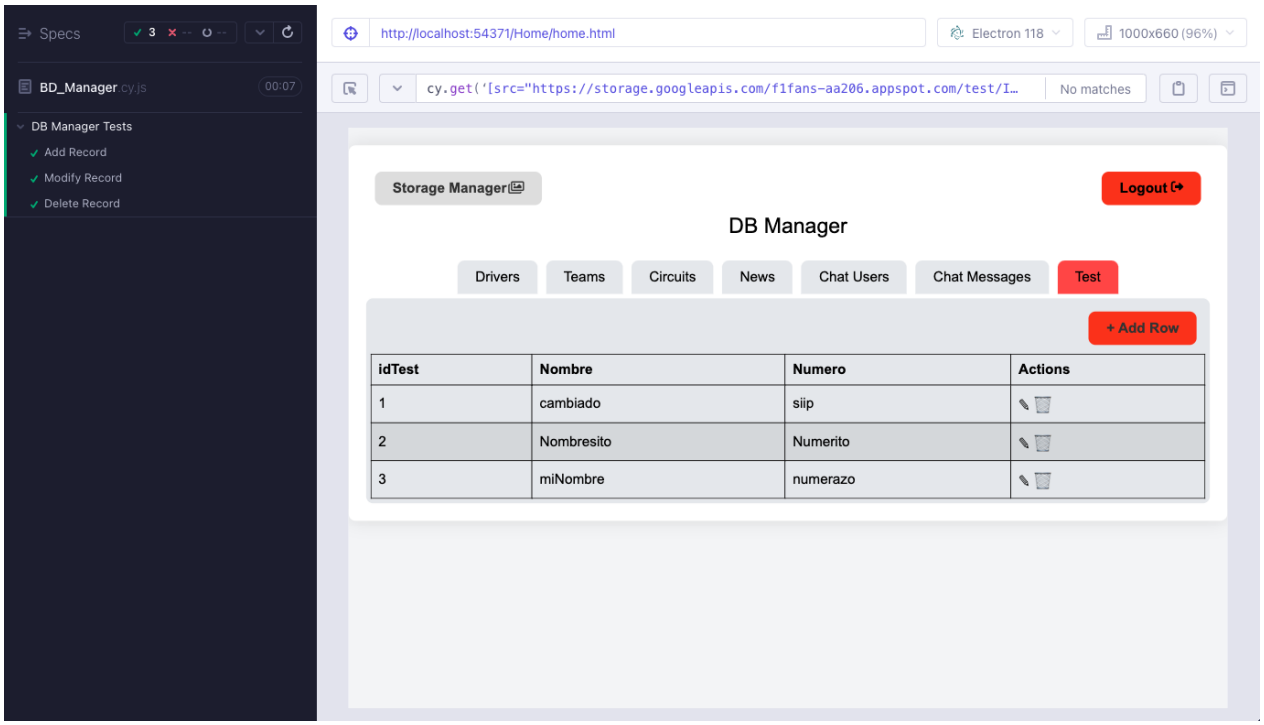

Ilustración 4.29: Resultado de la ejecución de los tests de DB Manager.

Este test verifica la funcionalidad de añadir, editar y borrar filas en las tablas de la base de datos **MySQL** desde la interfaz web. En la ilustración 4.29 podemos observar el resultado de correr los tests para las funcionalidades CRUD de el apartado de manejo de la base de datos **MySQL**.

Código del Test

En la ilustración 4.30, se muestra el contenido del archivo `'DBManager.cy.js'` para la realización de los tests de la sección de 'DB Manager' en la Interfaz Web.

```
describe('DB Manager Tests', () => {
  it('Add Record', () => {
    cy.login();

    cy.get('#Test').click({force:true});
    cy.wait(100);
    cy.get('#addBTN').click({force:true});

    cy.get('#Nombre').click({force:true}).clear({force:true}).type('TestCypress', {force:true});
    cy.get('#Numero').click({force:true}).clear({force:true}).type('1234', {force:true});

    cy.get('#AddRecordBTN').click({force:true});

    cy.contains('TestCypress').should('be.visible');
    cy.contains('1234').should('be.visible');
  });

  it('Modify Record', () => {
    cy.login();

    cy.get('#Test').click({force:true});
    cy.wait(100);
    cy.contains('TestCypress').parent().find('td').last().find('span').first().click({force:true});

    cy.get('#Nombre').click({force:true}).clear({force:true}).type('MODIFICADOTestCypress', {force:true});
    cy.get('#Numero').click({force:true}).clear({force:true}).type('5678', {force:true});

    cy.get('#EditRecordBTN').click({force:true});

    cy.contains('MODIFICADOTestCypress').should('be.visible');
    cy.contains('5678').should('be.visible');
  });

  it('Delete Record', () => {
    cy.login();

    cy.get('#Test').click({force:true});
    cy.wait(100);
    cy.contains('TestCypress').parent().find('td').last().find('span').last().click({force:true});

    cy.get('#deleteButton').click({force:true});

    cy.contains('MODIFICADOTestCypress').should('not.exist');
    cy.contains('5678').should('not.exist');
  });
})
```

Ilustración 4.30: Contenido del archivo 'DBManager.cy.js' para los tests de DB Manager.

Pruebas de Storage Manager

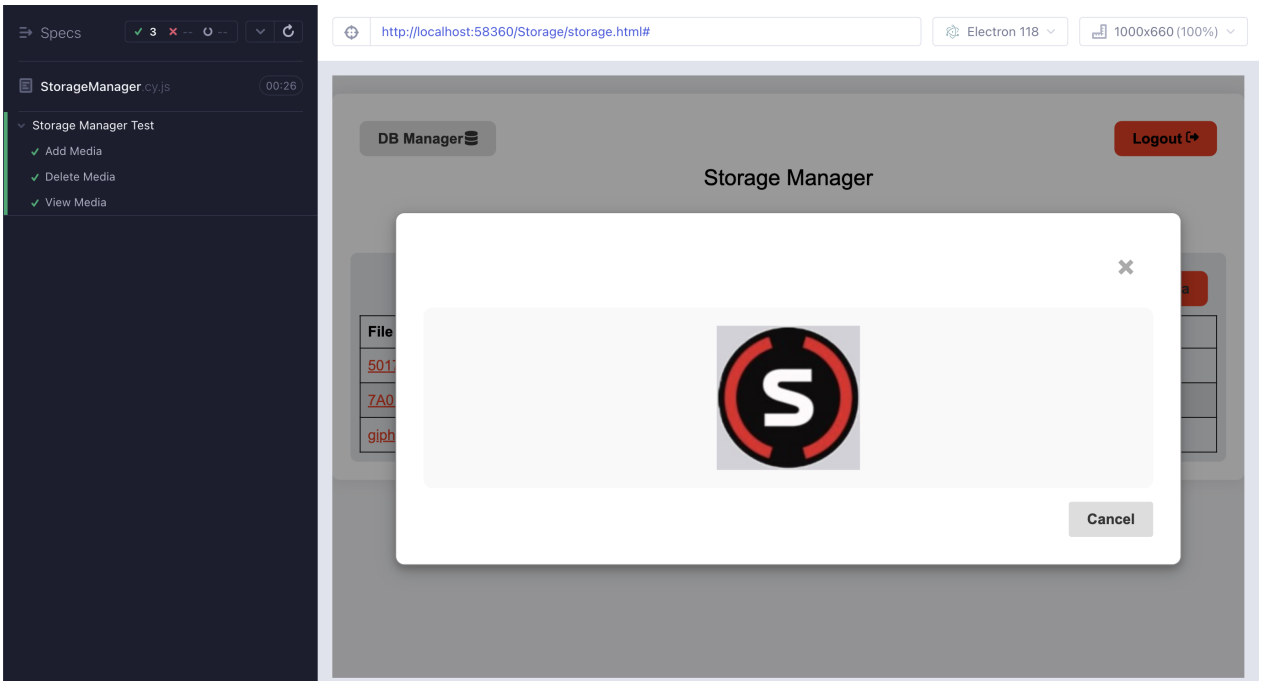

Ilustración 4.31: Resultado de la ejecución de los tests de Storage Manager.

Este test verifica la funcionalidad de añadir, visualizar y borrar contenido multimedia desde **Firebase Storage** a través de la interfaz web.En la ilustración 4.31 podemos observar el resultado de correr los tests para las funcionalidades CRD de el apartado de manejo de la base de datos **Firebase Storage**.

Código del Test

En las ilustraciones 4.32 & 4.33, se muestra el contenido del archivo `'StorageManager.cy.js'` para la realización de los tests de la sección de 'Storage Manager' en la Interfaz Web.

```
import 'cypress-file-upload';

describe('Storage Manager Test', () => {
  let fileContent;
  before(() => {
    cy.fixture('LogoF1FanApp.png').then((content) => {
      fileContent = content;
    });
  });

  it('Add Media', () => {
    cy.login();

    cy.get('#changeHomeBTN').click({force:true});

    cy.get('#test').click({force:true});
    cy.wait(5000);
    cy.get('#addBTN').click({force:true});
    cy.wait(100);

    cy.window().then((win) => {
      // Create the file input element dynamically (if needed)
      const fileInput = win.document.createElement('input');
      fileInput.type = 'file';
      fileInput.className = 'file-input'; // Set class name if required

      // Append the input to the DOM (optional, depending on your test)
      win.document.body.appendChild(fileInput);

      // Use Cypress command to attach the file
      cy.get('.file-input').attachFile({
        fileContent: fileContent,
        fileName: 'LogoF1FanApp.png',
        mimeType: 'image/png'
      });

      // Trigger change event on the element
      cy.get('.file-input').trigger('change', {force:true});


      cy.get('#AddMediaBTN').click({force:true});
      cy.wait(1000);
      cy.contains('LogoF1FanApp').should('be.visible');
    });


  });
```

Ilustración 4.32: Primera parte del contenido del archivo 'StorageManager.cy.js' para los tests de Storage Manager.

```
it('Delete Media', () => {
  cy.login();

  cy.get('#changeHomeBTN').click({force:true});

  cy.get('#test').click({force:true});
  cy.wait(5000);

  cy.contains('LogoF1FanApp').parent().parent().find('td').last().find('span').first().click({force:true});
  cy.wait(100);

  cy.get('#deleteButton').click({force:true});
  cy.contains('LogoF1FanApp').should('not.exist');
});

it('View Media', () => {
  cy.login();

  cy.get('#changeHomeBTN').click({force:true});

  cy.get('#test').click({force:true});
  cy.wait(5000);

  cy.contains('IMG_1298').click({force:true});
  cy.wait(100);

  cy.get('img').should('have.attr', 'src').and('not.be.empty');
});
```

Ilustración 4.33: Segunda parte del contenido del archivo 'StorageManager.cy.js' para los tests de Storage Manager.

Comandos de Cypress

En el desarrollo de pruebas automatizadas con Cypress, se hace uso de `commands.js` para definir y gestionar comandos personalizados que facilitan la escritura de casos de prueba más legibles y mantenibles.

El archivo `commands.js` en Cypress se encuentra en el directorio `cypress/support` y se utiliza para extender las capacidades de Cypress mediante la creación de comandos personalizados. Estos comandos pueden encapsular secuencias de comandos comunes o acciones repetitivas que se utilizan en múltiples casos de prueba.

```
Cypress.Commands.add('login', (no_visit) => {
    cy.visit('/Login/login.html');
    cy.get('#username').click({force:true}).clear({force:true}).type('admin@f1fans.com', {force:true});
    cy.get('#password').click({force:true}).clear({force:true}).type('123456', {force:true});
    cy.get('#LoginBTN').click({force:true});
})

Cypress.Commands.add('loginIncorrect', (no_visit) => {
    cy.visit('/Login/login.html');
    cy.get('#username').click({force:true}).clear({force:true}).type('admin@f1fans.com', {force:true});
    cy.get('#password').click({force:true}).clear({force:true}).type('1234567', {force:true});
    cy.get('#LoginBTN').click({force:true});
})
```

Ilustración 4.34: Contenido del archivo ‘commands.js‘ con comandos personalizados.

En la Figura 4.34 se muestra el contenido que se ha creado en `‘commands.js‘`, donde se han definido los comandos personalizados `‘login‘` y `‘loginIncorrect‘`.

Uso de Comandos Personalizados

Los comandos personalizados definidos en `‘commands.js‘` se pueden utilizar en los archivos de prueba (‘*.spec.js‘) de Cypress para simplificar y mejorar la legibilidad de los tests. Aquí se explica cómo se utilizan los comandos `‘login‘` y `‘loginIncorrect‘`:

- ✔ **cy.login():** Este comando se utiliza para realizar un inicio de sesión correcto en la aplicación durante las pruebas. Simplifica el proceso de autenticación al encapsular las acciones necesarias para iniciar sesión con credenciales válidas.

- ✔ **cy.loginIncorrect():** Este comando se utiliza para simular un intento de inicio de sesión incorrecto en la aplicación. Puede encapsular acciones como ingresar credenciales incorrectas y manejar el comportamiento esperado de la aplicación ante errores de autenticación.

El uso de comandos personalizados en Cypress proporciona una manera eficiente de reutilizar código y simplificar la escritura de casos de prueba. Facilita la creación de pruebas más mantenibles y legibles, mejorando así la eficiencia del proceso de desarrollo y pruebas automatizadas.

Resultados de las Pruebas

Se realizaron pruebas utilizando Cypress y se obtuvieron los siguientes resultados:

- **Login:** Todos los casos de prueba fueron exitosos.

- **DB Manager:** La funcionalidad de añadir, editar y borrar filas en las tablas de la base de datos MySQL funcionó correctamente.

- **Storage Manager:** La funcionalidad de añadir, visualizar y borrar contenido multimedia desde Firebase Storage funcionó correctamente.

4.5. Documentación iOS App

En este apartado se tratán dos funcionalidades de la aplicación que merecen la pena comentar y explicar:

✔ Live Chat

✔ Live Positions

4.5.1. Live Chat

Esta funcionalidad de la aplicación trata de un chat en el que los usuario pueden comentar las carreras y hablar entre ellos. Este chat está siempre disponible. Para entrar en el chat, solo hay que elegir el nombre de usuario y acceder.

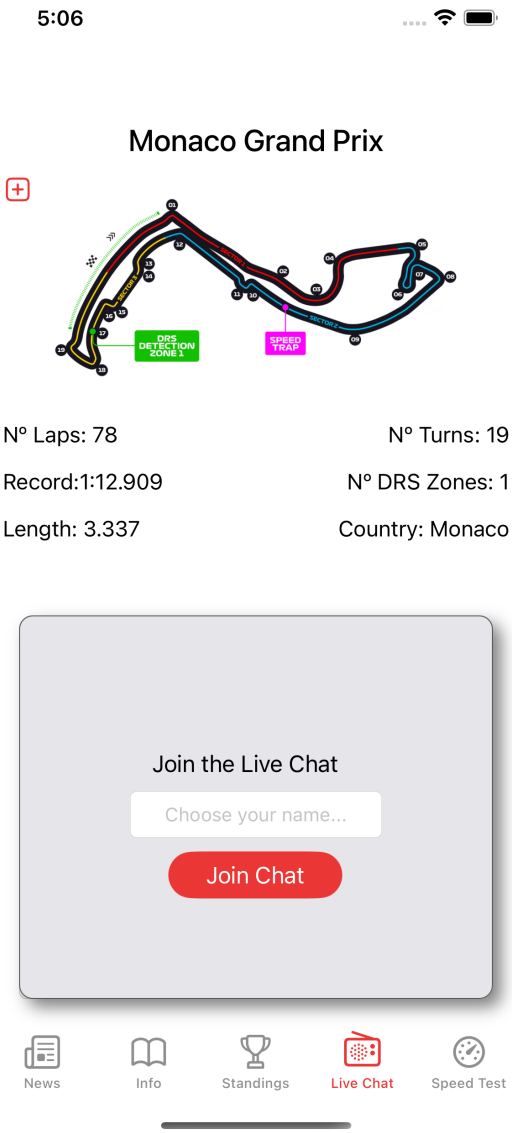

Ilustración 4.35: Pantalla de 'F1 Fan App' con el Chat cerrado

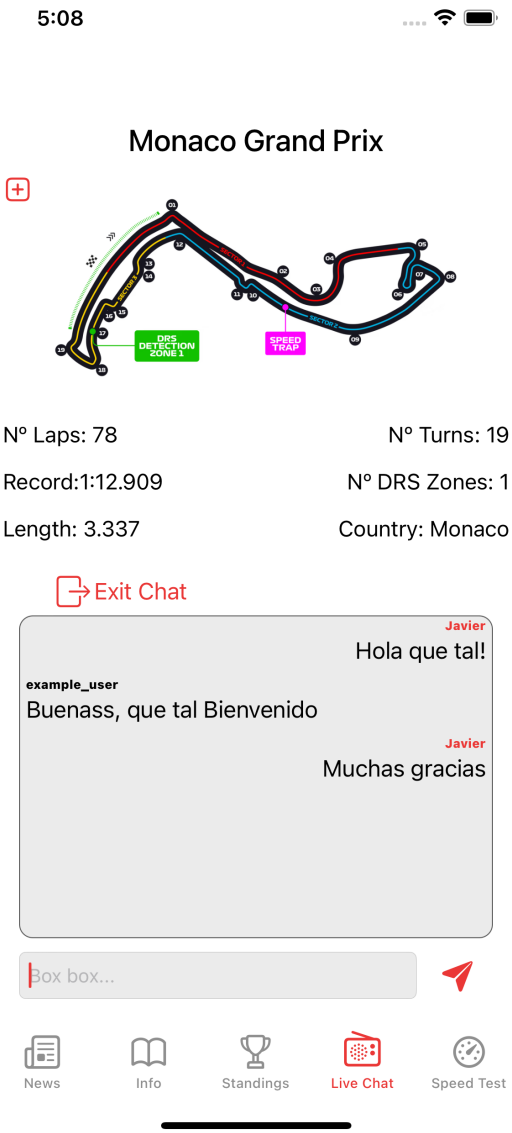

Ilustración 4.36: Pantalla de 'F1 Fan App' con el Chat abierto

En la ilustración 4.35 se observa como se ve el chat cuando está inactivo frente a la ilustración 4.36 que muestra como se ve el chat una vez se accede a el.

A continuación se explicará como se ha creado el chat:

- **Arranque y parada del chat**

Cuando un usuario introduce su nombre de usuario y accede al chat, se llama a la

función `'startChat()'`, como se observa en la ilustración 4.37, esta empieza un temporizador, el cual cada segundo(1s) llama a la función `'loadChat()'` la cual carga los mensajes en el chat.

```
func startChat() {
    // Invalidate any existing timer before starting a new one
    timer?.invalidate()

    // Start a new timer on a background queue that repeats every 1 second
    timer = Timer.scheduledTimer(withTimeInterval: 1.0, repeats: true) { _ in
        self.loadChat(self.messageTableView)
    }

    // Ensure the timer runs on a background queue
    if let timer = timer {
        RunLoop.current.add(timer, forMode: .common)
    }
}

func stopChat() {
    // Stop the timer when needed
    timer?.invalidate()
}
```

Ilustración 4.37: Código usado para la abertura y cierre del Chat

Cuando el usuario sale del chat se cambia la interfaz para hacer desaparecer el chat y se llama a la función `'stopChat()'` la cual detiene el el temoprizador que previamente `'startChat()'` había comenzado para cargar los mensajes en el chat.

- **Obtención de mensajes**

Para la obtención de mensajes, se usa la función `'loadChat()'`, la cual es llamada cada segundo(1s) por el temporizador iniciado al entrar al chat. Como se muestra en la ilustración 4.38, esta función verifica si el usuario actual tiene un identificador de usuario (`currentUserId`). Si no lo tiene, la función intenta obtenerlo a través de una llamada a la API.

```
func loadChat(_ tableView: UITableView) {

    if (self.currentUserId == "") {  // Si no tenemos nuestro User_Id, lo obtenemos
        APIUtil.getUserId(forUsername: self.currentUsername) { result in
            switch result {
            case .success(let userId):
                if let userId = userId {
                    print("User ID: \(userId) for USERNAME: \(self.currentUsername)")
                    self.currentUserId = userId  // Obtenemos el ID del remitente

                    self.messageGet()  // Llamamos a messageGet() para obtener los mensajes

                } else {
                    print("User not found.")
                    self.currentUserId = "99999"  // Establecemos un ID por defecto si el usuario no es encontrado
                }
            case .failure(let error):
                print("Error: \(error)")  // Imprimimos el error si falla la llamada a la API
            }
        }
    } else {  // Si ya tenemos el User_Id
        messageGet()  // Llamamos a messageGet() para obtener los mensajes
    }
}
```

Ilustración 4.38: Código usado para cargar el chat

```
func messageGet() {
    self.newMessages = []

    APIUtil.getAPI(from: "ChatMessages")
    if let chatMessages = UserDefaults.standard.string(forKey: "ChatMessages") {
        if let jsonData = chatMessages.data(using: .utf8) {
            do {
                let messages = try JSONDecoder().decode([ChatMessages].self, from: jsonData)

                for message in messages {
                    //Check for message timestamp (MessageTimestamp >= ChatJoinTimestamp)
                    if (compareDates(stringToDate(timestamp: message.Timestamp), self.chatJoinTimestamp)) {      //Message is a new message
                        //Gets Users Username


                        let messageUser = APIUtil.getUsername(forUserId: message.iduser)

                        let chatMessage = ChatMessage(message: message.Content, isUser: self.currentUserId == message.iduser, username: messageUser!)    //Crea mensaje ordenado
                        self.newMessages.append(chatMessage)     //Añade el mensaje a la lista de ordenados
                        print("NewMessages    ADDED: \(self.newMessages)")
                        print("AÑADIENDO: \(chatMessage.message), USER: \(chatMessage.username)")

                    } else {    //It is an old message
                        //Ignorar el mensaje
                    }
                }

                print("NewMessages: \(self.newMessages)")
                self.orderedMessages = self.newMessages
                print("LISTA DE MENSAJES: \(self.orderedMessages)")

                messageTableView.reloadData()
                scrollToBottom()


            } catch {
                print("Error decoding JSON: \(error)")
            }
        }
    } else {
        print("Chat is Disconnected")
    }
}
```

Ilustración 4.39: Código usado para cargar los mensajes en el chat

Una vez obtenido el identificador, se llama a la función `'messageGet()'`, el cual como se observa en la ilustración 4.39, se encarga de obtener todos los mensajes a través de una llamada a la API, seleccionar los mensajes posteriores a la fecha de

acceso al chat y los muestra en el chat haciendo que este vaya bajando y siempre siguiendo el último mensaje.

- **Mandar mensajes**

  A la hora de escribir un mensaje y darle a enviar, tal y como se muestra en la ilustración 4.40, el hacer click en el botón de enviar, acciona una función llamada `'sendBTN()'`, la cual a través de una llamada a la API, manda el mensaje a la base de datos y posteriormente limpia el contenido de la caja de texto.

```
@IBAction func sendBTN(_ sender: Any) {
    APIUtil.postToChatMessages(username: currentUsername, message: messageTXB.text!)
    print("-------- Sending Message --------")
    print("USER: \(currentUsername)");
    print("Message: \(messageTXB.text!)");
    messageTXB.text = "";
}
```

Ilustración 4.40: Código usado para enviar mensajes al chat

- **Accesibilidad con teclado**

  A parte de la tecnología para activar y desactivar el chat, mandar mensajes y recibirlos entre otras, como es una aplicación móvil y no se tiene un teclado externo, se ha añadido la funcionalidad para que al aparecer el teclado, todo el contenido suba para así poder ver sin problema lo que se escribe en las cajas de texto donde se escriben los mensajes y nombre de usuario. Un ejemplo de esto se puede ver en las ilustraciones 4.41 & 4.42.

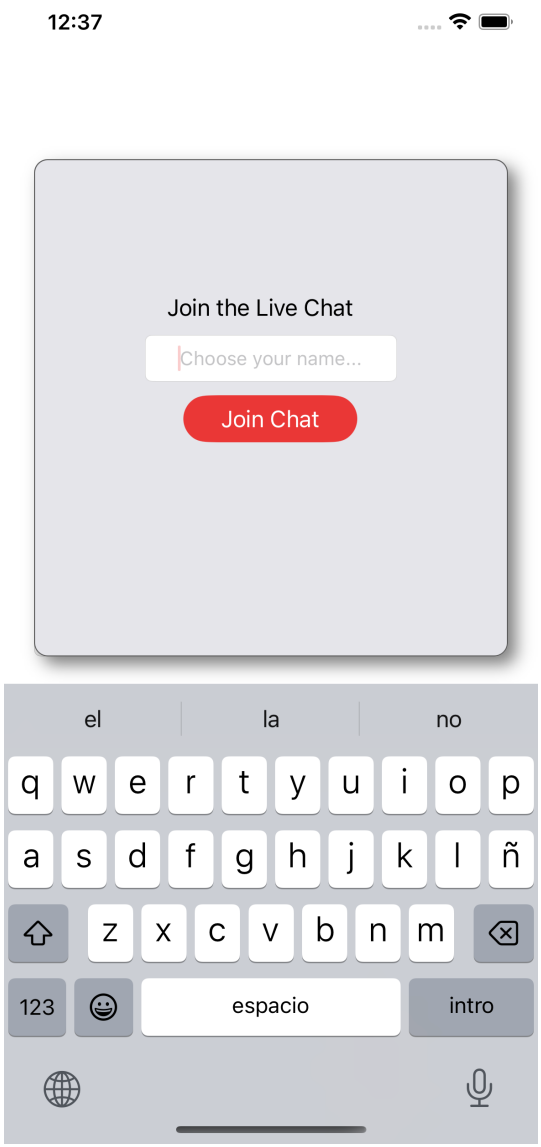

Ilustración 4.41: Ejemplo de funcionalidad de teclado en campo de nombre de usuario en el chat

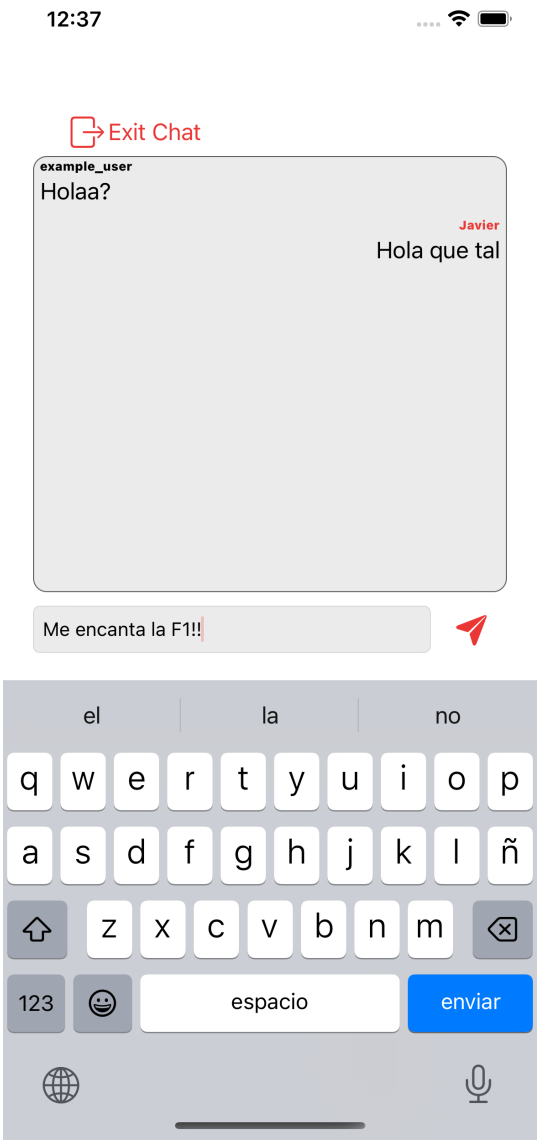

Ilustración 4.42: Ejemplo de funcionalidad de teclado en campo de mensaje en el chat

A continuación en la ilustración 4.43 se muestra el código utilizado para realizar esto, donde se maneja la aparición y desaparición del teclado, ajustando la posición de la vista principal. La función `keyboardWillShow` se llama cuando el teclado está a punto de mostrarse obtiene el tamaño del teclado desde la notificación y anima la vista moviéndola hacia arriba por la altura del teclado. La función `keyboardWillHide` se llama cuando el teclado está a punto de ocultarse y anima la vista devolviéndola a su posición original. La función `deinit` se llama cuando el objeto es desalojado de la memoria, eliminando el objeto como observador de las notificaciones del teclado para evitar problemas de memoria.

```
@objc func keyboardWillShow(_ notification: Notification) {
    if let keyboardSize = (notification.userInfo?[UIResponder.keyboardFrameEndUserInfoKey] as? NSValue)?.cgRectValue {
        UIView.animate(withDuration: 0.3) {
            self.vcView.transform = CGAffineTransform(translationX: 0, y: -keyboardSize.height)
            self.lapsLBL.isHidden = true
            self.turnsLBL.isHidden = true
            self.lengthLBL.isHidden = true
            self.countryLBL.isHidden = true
            self.recordLBL.isHidden = true
            self.drsLBL.isHidden = true
        }
    }
}

@objc func keyboardWillHide(_ notification: Notification) {
    UIView.animate(withDuration: 0.3) {
        self.vcView.transform = .identity
        self.lapsLBL.isHidden = false
        self.turnsLBL.isHidden = false
        self.lengthLBL.isHidden = false
        self.countryLBL.isHidden = false
        self.recordLBL.isHidden = false
        self.drsLBL.isHidden = false
    }
}

deinit {
    NotificationCenter.default.removeObserver(self)
}
```

Ilustración 4.43: Código usado para manejar vista en relación al teclado

4.5.2. Live Positions

Esta funcionalidad de la aplicación trata de una pantalla en el que los usuario pueden ver en tiempo real la posición de los pilotos. Para esta funcionalidad se obtienen diferentes datos para obtener la visualización mostrada en la ilustración 4.44. Para obtener la información de pilotos como el nombres o imágenes, se usa la base de datos y para obtener la información de las posiciones y neumáticos actuales, se usa la API externa de OpenF1.

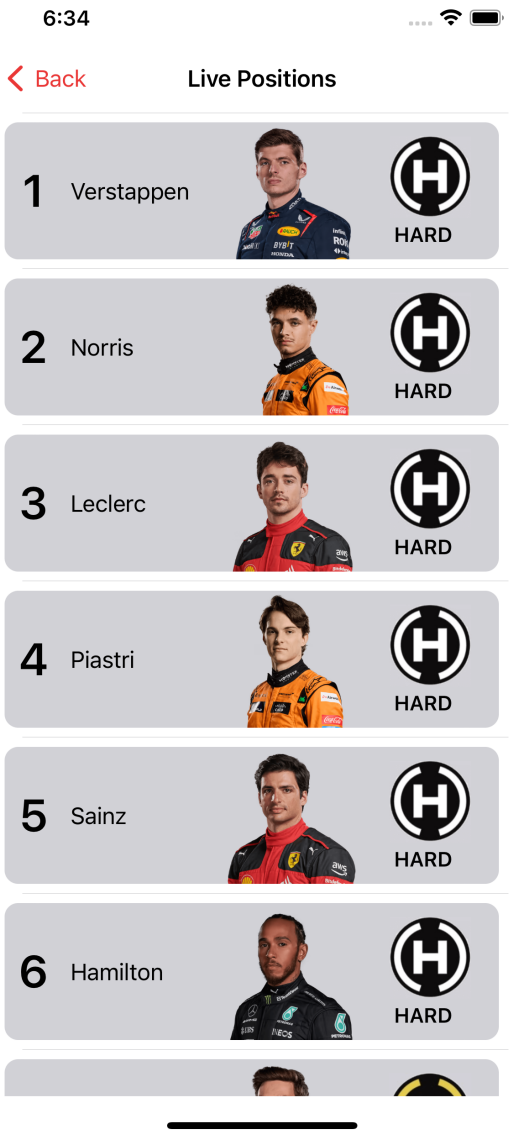

Ilustración 4.44: Captura de la pantalla de la sección 'Live Positions' de 'F1 Fan App'

A continuación se explicará como se ha conseguido esta funcionalidad:

Como se observa en la ilustración 4.45, al iniciar la pantalla, se inicia un temorizador, el cual cada 3 segundos llama a la función `'refreshData()'`, la cual como vemos en la ilustración 4.46, se encarga de iniciar todo el proceso de refresco de datos. Esta función llama a `'obtainMeetingPosition()'`(ilustración 4.47), la cual obtiene la información más reciente sobre posiciones de la API de OpenF1 y asigna a cada piloto su posición. Una vez hecho esto, esta función llama a `'obtainMeetingStint()'`, la cual como vemos en la ilustración 4.48, mediante el uso de la API externa OpenF1, se encarga de obtener los datos más actualizados sobre los neumáticos que están usando los pilotos y también los asigna a los pilotos. Una vez

hecho esto, finalmente se llama a `'getOrderedDrivers()'`(ilustración 4.49), la cual ordena la lista de pilotos ordenados por posición que se muestra en la pantalla. Una vez acabado, volvemos a la función `'refreshData()'` donde se acaba de ejecutar, imprimiendo los valores en la tabla.

```
timer?.invalidate()

// Start a new timer on a background queue that repeats every 3 seconds
timer = Timer.scheduledTimer(withTimeInterval: 3.0, repeats: true) { _ in
    self.refresData()
    //Hacer aparecer Loader
    self.loadingContainer.isHidden = true
    self.loadingIMG.isHidden = true
    self.loadingLBL.isHidden = true
}

// Ensure the timer runs on a background queue
if let timer = timer {
    RunLoop.current.add(timer, forMode: .common)
}
```

Ilustración 4.45: Código usado para iniciar el temporizador de 'Live Positions'

```
func refresData() {
    taskDone = false

    obtainMeetingPositon()  //Se van llamando una detrás de otra al final de cada función

    while (!taskDone) {
        //Wait
    }

    positionTableView.reloadData()
}
```

Ilustración 4.46: Código usado para iniciar el temporizador de 'Live Positions'

```
func obtainMeetingPositon() {
    APIUtil.getPositionData(meetingKey: currentMeetingKey) { result in
        switch result {
        case .success(let data):
            do {
                let positions = try JSONDecoder().decode([Position].self, from: data)

                var driverPositon: [Position] = []

                //Get latest Stint for each driver
                for driver in self.allDrivers {
                    driverPositon = []
                    for position in positions {
                        if (String(position.driver_number) == driver.Number) {
                            driverPositon.append(position)
                        }
                    }
                self.allPositions.append(driverPositon[driverPositon.count-1])
                }

                print("--------- Lista Positions ---------")
                for i in self.allPositions{
                    print(i)
                }

                self.obtainMeetingStint()

                print("Lista de Positions ha sido actualizada")
            } catch {
                print("Error decoding JSON: \(error)")
            }
        case .failure(let error):
            print("Error fetching STINT: \(error)")
        }

    }
}
```

Ilustración 4.47: Código usado para obtener las posiciones de los pilotos

```
func obtainMeetingStint() {
    APIUtil.getStintData(meetingKey: currentMeetingKey) { result in
        switch result {
        case .success(let data):
            do {
                let stints = try JSONDecoder().decode([Stint].self, from: data)

                var driverStint: [Stint] = []

                //Get latest Stint for each driver
                for driver in self.allDrivers {
                    driverStint = []
                    for stint in stints {
                        if (String(stint.driver_number) == driver.Number) {
                            driverStint.append(stint)
                        }
                    }
                self.allStints.append(driverStint[driverStint.count-1])
                }

                print("--------- Lista Stints ---------")
                for i in self.allStints{
                    print(i)
                }

                print("Lista de Stints ha sido actualizada")

                self.getOrderedDrivers()
            } catch {
                print("Error decoding JSON: \(error)")
            }
        case .failure(let error):
            print("Error fetching STINT: \(error)")
        }

    }
}
```

Ilustración 4.48: Código usado para obtener los neumáticos de los pilotos

```
func getOrderedDrivers() {
    var resultList: [OrderedDriver] = []

    for driver in allDrivers {
        let driverStint = allStints.first(where: { String($0.driver_number) == driver.Number })
        let driverPosition = allPositions.first(where: { String($0.driver_number) == driver.Number })

        resultList.append(OrderedDriver(position: driverPosition!.position, Photo: driver.Photo,
                                        Name: driver.Name, Surname: driver.Surname,
                                        Number: driver.Number, compound: driverStint!.compound))
    }

    resultList.sort { $0.position < $1.position }
    orderedDriverList = resultList
    print("--------- Lista Ordered Drivers ---------")
    for i in orderedDriverList{
        print(i)
    }

    //print("Refrescando Table View")
    taskDone = true
    //positionTableView.reloadData()
}
```

Ilustración 4.49: Código usado para ordenar la lista de pilotos

Capítulo 5

Conclusiones y trabajo futuro

En este trabajo, se ha desarrollado un proyecto completo y funcional que cumple con las necesidades de los fans de Formula 1. La **'F1 Fan'** App, la Interfaz Web y las APIs tienen un gran potencial para seguir creciendo y mejorando en el futuro. Este proyecto ha sido una experiencia de aprendizaje muy enriquecedora que me ha permitido aplicar mis conocimientos tecnológicos y desarrollar mis habilidades profesionales. Además, he conseguido obtener un conocimiento más amplio en varias herramientas y entornos de desarrollo, especialmente en Xcode, lo cual ha sido fundamental para el éxito de la aplicación.

Estoy seguro de que este proyecto será de gran utilidad para los fans de la Formula 1. La experiencia adquirida no solo ha fortalecido mis competencias en desarrollo de software, sino que también me ha brindado una comprensión más profunda de las plataformas y tecnologías que impulsan aplicaciones robustas y atractivas. Con esta base sólida, el proyecto tiene un brillante futuro y capacidad de evolución para satisfacer las crecientes demandas de los aficionados.

De cara al futuro, se tiene pensado realizar las siguientes acciones para mejorar y expandir el proyecto:

- Hostear y desplegar el proyecto utilizando un sistema CI/CD inteligente, lo que permitirá que el público pueda disfrutar de la aplicación, y cuando se realicen, se permitan actualizaciones más rápidas y eficientes del mismo.

- Se plantea desarrollar una versión Web de la aplicación **'F1 Fan'** para así poder tener un mayor alcance a otros usuarios y no encerrar el proyecto solo en el ecosistema de iOS.

- Añadir funcionalidades como cuentas de usuarios para permitir una mayor personalización de la aplicación.

- Expandir los apartados de visualización para incluir más datos sobre pilotos, equipos y circuitos.

Capítulo 6

Bibliografía

Bibliografía

[1] Loading gif generator. https://loading.io, 2022.

[2] Formula 1. Never miss a moment with the official f1® app. https://www.formula1.com/en/subscribe/download-the-official-F1--app.html, 2018.

[3] brgodefroy. Openf1 - an api for real-time f1 data. https://www.reddit.com/r/F1DataAnalysis/comments/16w84uz/openf1_an_api_for_realtime_f1_data/, 2023.

[4] Nicolás Cordero. Cypress, un framework de pruebas todo en uno. https://www.paradigmadigital.com/dev/cypress-un-framework-de-pruebas-todo-en-uno/, 2018.

[5] Apple Developer. Accessibility inspector - developer tool. https://developer.apple.com/documentation/accessibility/accessibility-inspector, 2017.

[6] Esat Kemal Ekren. What is xcode and how to use it? https://www.netguru.com/blog/what-is-xcode-and-how-to-use-it, 2024.

[7] Hanna F. How to use figma like a pro: A comprehensive guide for designers. https://www.temok.com/blog/how-to-use-figma/, 2023.

[8] F1Play.game. F1 play. https://play.google.com/store/apps/details?id=com.incrowdsports.isg.predictor&hl=en_US, 2019.

[9] Fabrizio. Coolors - the super fast color palettes generator. https://coolors.co, 2022.

[10] GP Fans. Gp fans - footer. https://www.gpfans.com/es/, 2017.

[11] Looka. Free logo maker. https://looka.com/logo-maker/?gad_source=1&gclid=Cj0KCQjw0_WyBhDMARIsAL1Vz8tJPs9pqFsJh-2SRv-aoWSX05VxhLM9Ner67yoTwdjrG363KaKyt
wcB, 2022.

[12] Hutch Games Ltd. F1® clash. https://www.hutch.io/our-games/f1-clash/, 2019.

[13] Alexander Obregon. Webstorm — an introduction. https://medium.com/@AlexanderObregon/dive-into-web-development-with-webstorm-a-comprehensive-introduction-74265
2023.

[14] Slowly. F1 dash. https://f1-dash.com, 2023.

[15] Tushar Soam. Create er diagram of a database in mysql workbench. https://medium.com/@tushar0618/how-to-create-er-diagram-of-a-database-in-mysql-workbench-209fbf63fd03, 2018.

Capítulo 7

Anexo I - Descripción de Base de Datos MySQL

En este ANEXO, se detallan las estructuras de las tablas(**Drivers**, **Circuits**, **Teams**, **ChatMessages**, **News**, y **ChatUsers**), el uso de cada tipo de dato y las relaciones entre ellas.

Drivers

- **idDrivers (int)**: Identificador único del piloto.
- **Name (varchar(45))**: Nombre del piloto.
- **Surname (varchar(45))**: Primer apellido del piloto.
- **TeamName (varchar(45))**: Nombre del equipo al que pertenece el piloto.
- **Number (varchar(45))**: Número del piloto.
- **Photo (varchar(45))**: Ruta de la foto del piloto.
- **Flag (varchar(45))**: Ruta de la bandera del país del piloto.

- **Points (varchar(255))**: Puntos obtenidos en la temporada actual.
- **TotalPoints (int)**: Puntos totales acumulados a lo largo de la temporada actual.

Circuits

- **idCircuits (int)**: Identificador único del circuito.
- **Name (varchar(45))**: Nombre del Grand Prix.
- **Country (varchar(45))**: País donde se encuentra el circuito.
- **Length (varchar(45))**: Longitud del circuito.
- **Turns (int)**: Número de curvas del circuito.
- **Photo (varchar(45))**: Ruta de la foto del circuito.
- **Flag (varchar(45))**: Ruta de la bandera del país del circuito.
- **Date (datetime)**: Fecha de comienzo del Grand Prix.
- **ExtraInfo (varchar(255))**: Información adicional sobre el circuito (Número de vueltas, Mejor tiempo, Número de zonas de DRS).

Teams

- **idTeams (int)**: Identificador único del equipo.
- **Name (varchar(45))**: Nombre del equipo.
- **Logo (varchar(45))**: Ruta del logo del equipo.
- **Car (varchar(45))**: Ruta del coche del equipo.

- **Driver1Name (varchar(45))**: Nombre completo del primer piloto.
- **Driver1Photo (varchar(45))**: Ruta de la foto del primer piloto.
- **Driver2Name (varchar(45))**: Nombre completo del segundo piloto.
- **Driver2Photo (varchar(45))**: Ruta de la foto del segundo piloto.
- **Points (varchar(255))**: Puntos obtenidos por el equipo en la temporada actual.
- **TotalPoints (int)**: Puntos totales acumulados por el equipo a lo largo de la temporada actual.

ChatUsers

- **idChatUsers (int)**: Identificador único del usuario del chat.
- **Username (varchar(45))**: Nombre de usuario.

ChatMessages

- **idChatMessages (int)**: Identificador único del mensaje de chat.
- **iduser (int)**: Identificador del usuario que envió el mensaje, referencia a `idChatUsers`.
- **Content (text)**: Contenido del mensaje de chat.
- **Timestamp (datetime)**: Fecha y hora en que se envió el mensaje.

News

- **idNews (int)**: Identificador único de la noticia.
- **Title (varchar(45))**: Título de la noticia.

- **Description (text)**: Descripción de la noticia.
- **Images (varchar(255))**: Ruta de las imágenes asociadas a la noticia.
- **Date (datetime)**: Fecha de publicación de la noticia.

Tipos de Datos Utilizados

✔ **int**: Se utiliza para campos que requieren un valor entero, como identificadores únicos (`idDrivers`, `idCircuits`, etc.) y contadores (`Turns`, `TotalPoints`).

✔ **varchar(45)**: Se utiliza para cadenas de texto cortas, como nombres, Rutas de Imágenes y descripciones breves (`Name`, `TeamName`, `Logo`, etc.).

✔ **varchar(255)**: Se utiliza para cadenas de texto más largas, como Rutas de imágenes y listas de información más extendida (`Points`, `ExtraInfo`, `Images`).

✔ **text**: Se utiliza para campos que pueden contener un gran volumen de texto, como el contenido de mensajes de chat y descripciones de noticias (`Content`, `Description`).

✔ **datetime**: Se utiliza para almacenar fechas y horas precisas, como las fechas de eventos y tiempos de envío de mensajes (`Date`, `Timestamp`).

Relación entre `ChatMessages` y `ChatUsers`

La tabla `ChatMessages` contiene un campo `iduser` que es una referencia a la tabla `ChatUsers`. Esta relación indica que cada mensaje de chat es enviado por un usuario específico. El campo `iduser` en `ChatMessages` es una FOREIGN KEY que apunta al campo `idChatUsers` en `ChatUsers`. Esto asegura que cada mensaje esté asociado con un usuario válido.