Discord Bot manager

TRABAJO FINAL DE CICLO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Pelayo Palacio Suárez

curso académico 2024/2025

Oviedo, 22/05/2025

Contenido

[**1. Descripción general del proyecto** 4](#_Toc177584882)

[ Introducción: Resumen de las características del proyecto 4](#_Toc177584883)

[ Presentación de las características 4](#_Toc177584884)

[ Alcance del proyecto: Detalle del problema planteado indicando, punto por punto, todos los detalles que se van a gestionar. 5](#_Toc177584885)

[ Justificación y análisis de la realidad 7](#_Toc177584886)

[ Marco legal 9](#_Toc177584887)

[ Marco teórico 9](#_Toc177584888)

[ Temporalidad 14](#_Toc177584889)

[**2. estudio de la viabilidad del sistema** 15](#_Toc177584890)

[ Análisis económico (DAFO) 15](#_Toc177584891)

[ Plan de marketing 15](#_Toc177584892)

[**3. Descripción del entorno tecnológico** 16](#_Toc177584893)

[ Descripción de todos los perfiles de usuario que habrá en la aplicación 16](#_Toc177584894)

[ Descripción de las tecnologías para cada perfil 17](#_Toc177584895)

[**4. Especificación de requisitos** 18](#_Toc177584896)

[ Requisitos funcionales 18](#_Toc177584897)

[ Requisitos no funcionales 19](#_Toc177584898)

[**5. Modelo entidad / relación** 21](#_Toc177584899)

[ Explicar por qué ha salido ese E/R, de dónde salen las Entidades y de dónde salen las Relaciones. 21](#_Toc177584900)

[ Plasmar el modelo E/R 21](#_Toc177584901)

[ Explicar el origen de los atributos 21](#_Toc177584902)

[ Explicar por qué se han elegido esas claves primarias. 21](#_Toc177584903)

[ Modelo Relacional 21](#_Toc177584904)

[**6. Diagrama de procesos** 21](#_Toc177584905)

[ Casos de Uso 21](#_Toc177584906)

[ Diagrama de clases 21](#_Toc177584907)

[**7. Diseño Interfaz** 21](#_Toc177584908)

[ Diagramación y prototipado del diseño gráfico de la interfaz 21](#_Toc177584909)

[ Guías de estilo 23](#_Toc177584910)

[ Mapa de navegación 24](#_Toc177584911)

[**8 Pruebas** 24](#_Toc177584912)

[ Pruebas unitarias 24](#_Toc177584913)

[ Pruebas de integridad 24](#_Toc177584914)

[**9. Manual de usuario** 24](#_Toc177584915)

[ Manual del cliente 24](#_Toc177584916)

[ Manual de instalación y despliegue 24](#_Toc177584917)

[**10. CONCLUSIONES** 24](#_Toc177584918)

[**11. Anexo** 24](#_Toc177584919)

[ Bibliografía 24](#_Toc177584920)

[ Referencias a texto y/o web de ayuda 24](#_Toc177584921)

[ Referencia de imágenes 24](#_Toc177584922)

# **1. Descripción general del proyecto**

* Introducción: Resumen de las características del proyecto

El proyecto Discord Bot Manager ha evolucionado para aprovechar tecnologías modernas que permiten una experiencia multiplataforma real y una interfaz de usuario avanzada. Ahora, la solución se basa en una arquitectura compuesta por Electron (para aplicaciones de escritorio y móvil), un backend desarrollado en Spring Boot y una interfaz web integrada (SPA/PWA). Esta combinación permite ofrecer una aplicación robusta, flexible y fácilmente mantenible, superando las limitaciones de tecnologías previas como JavaFX.

La elección de Electron responde a la necesidad de interfaces modernas, personalizables y con soporte multiplataforma (Windows, Linux, macOS e incluso Android mediante wrappers). Spring Boot proporciona un backend sólido, seguro y escalable, encargado de la lógica de negocio, la persistencia y la integración con la API de Discord. La interfaz web, desarrollada como SPA/PWA, se integra perfectamente en Electron y permite una experiencia de usuario fluida y coherente en todos los dispositivos.

Dentro de las funciones disponibles se encuentran los comandos básicos de moderación, log de dichas acciones, manipulación de servidores, gestión modular de funcionalidades y visualización avanzada de datos, todo ello accesible tanto desde escritorio como desde móvil.

* Presentación de las características

Aplicación de escritorio y móvil (Electron + Web SPA/PWA):

* + Permite ejecutar el Bot y gestionar sus módulos desde una interfaz moderna y personalizable.
  + Configuración avanzada de comandos, módulos y parámetros del Bot.
  + Visualización de información de servidores y usuarios, así como logs de acciones, todo sincronizado con la base de datos.
  + Experiencia de usuario coherente y multiplataforma.

Backend (Spring Boot):

* + Lógica de negocio centralizada, gestión de usuarios, experiencia, penalizaciones y módulos.
  + API RESTful segura para la comunicación con el frontend (Electron/web).
  + Integración con la API de Discord y la base de datos SQL.
  + Escalabilidad y facilidad de mantenimiento.

Justificación del cambio tecnológico

La migración desde JavaFX a Electron y tecnologías web modernas se debe a la necesidad de interfaces más flexibles, atractivas y con mejor soporte multiplataforma. Electron permite reutilizar tecnologías web estándar (HTML, CSS, JS/TS) y acceder a una comunidad y recursos mucho más amplios, facilitando la integración de librerías, temas y componentes visuales avanzados. Spring Boot, por su parte, garantiza un backend robusto y escalable, desacoplado de la interfaz, lo que facilita el mantenimiento y la evolución futura del proyecto.

* Alcance del proyecto: Detalle del problema planteado indicando, punto por punto, todos los detalles que se van a gestionar.

Alcance del proyecto:

El proyecto tiene como objetivo principal proporcionar una solución accesible y personalizable para la gestión de bots de Discord a través de una aplicación multiplataforma. Esta solución aborda varios problemas que enfrentan los usuarios al administrar y personalizar bots, como la dependencia de servicios de hosting remotos, la complejidad de configurar funcionalidades avanzadas, y la falta de control directo sobre los datos y el rendimiento del Bot.

Problemas planteados:

* + Dependencia de un hosting remoto: Los bots de Discord suelen necesitar servidores externos para ejecutarse, lo que implica costos adicionales, falta de control sobre el entorno y riesgos de seguridad.
  + Falta de accesibilidad a funciones avanzadas: Muchas requieren de servicios premium o gran conocimiento técnico para implementarlos.
  + -Carencia de aplicaciones locales para el manejo de los bots: La mayoría de bots de Discord tienen una dashboard online a través de la cual se controlan.
  + Seguridad y privacidad: El hosting local permite evitar problemas de seguridad al no tener control sobre los protocolos con los que se tratan los datos.
  + Facilidad de uso: La mayoría de bots son de fácil uso, aunque no disponen de una estructura estándar de desarrollo.

Detalles que gestionar:

* + Creación de una aplicación de escritorio
  + Desarrollo de una interfaz que permita al usuario:
    - Configurar módulos y comandos del Bot.
    - Ejecutar el Bot localmente y actuar como host para otras aplicaciones.
    - Visualizar información relevante de servidores y usuarios (ID, acciones, registros, etc.).
  + Implementación de una base de datos para registrar actividades y datos del Bot.
  + Desarrollo de una aplicación móvil
    - Función principal como host para bots en dispositivos móviles.
    - Configuración del bot al igual que la de escritorio, pero con menos acceso a datos.
    - Soporte para dispositivos antiguos o en desuso, maximizando el aprovechamiento del hardware existente.
  + Diseño y funcionalidad del Bot de Discord
    - Comandos básicos para moderación (gestión de usuarios, permisos, logs).
    - Herramientas de administración de servidores, como configuraciones automáticas y reportes.
    - Opciones de personalización para adaptarse a necesidades específicas de servidores de Discord.
  + Gestión de seguridad y datos
    - Hay que asegurar que toda la información manejada esté protegida contra accesos no autorizados.
    - Cumplir con las normativas de privacidad de datos aplicables a Discord y a los usuarios finales.
  + Soporte multiplataforma y escalabilidad
    - Garantizar una experiencia fluida y consistente entre las versiones de escritorio y móvil.
    - Facilitar actualizaciones y mejoras para mantener la compatibilidad con la API de Discord.
  + Facilidad de uso y documentación
    - Desarrollo de manuales detallados para instalación, configuración y uso.
    - Diseño de una interfaz intuitiva que reduzca la curva de aprendizaje, permitiendo a usuarios sin experiencia técnica aprovechar el producto.
* Justificación y análisis de la realidad

Justificación del proyecto:

* + Demanda creciente de bots personalizados:
    - A medida que las comunidades de Discord crecen en tamaño y complejidad, existe una necesidad cada vez mayor de bots que puedan adaptarse a las necesidades específicas de cada servidor.
  + Control total del entorno y los datos:
    - La mayoría de los bots de Discord dependen de servicios de hosting remoto, lo que implica riesgos relacionados con la privacidad, la seguridad y la dependencia de terceros.
  + Accesibilidad y facilidad de uso:
    - La posibilidad de usar el Bot tanto en aplicaciones de escritorio como móviles abre la puerta a una mayor flexibilidad. Además, el diseño multiplataforma facilita el acceso y uso de esta herramienta, incluso para aquellos con recursos limitados, como dispositivos antiguos o en desuso.
  + Aplicación práctica de conocimientos técnicos:
    - Este proyecto no solo resuelve un problema práctico, sino que también sirve como ejercicio para aplicar los conocimientos adquiridos en el ciclo formativo de Desarrollo de Aplicaciones Multiplataforma, integrando múltiples tecnologías y principios de diseño en un caso real de uso.

Análisis de la realidad:

* + Popularidad de Discord:
    - En los últimos años, Discord se ha posicionado como una de las plataformas de comunicación más utilizadas, con más de 300 millones de usuarios registrados y comunidades activas en todo el mundo. Esta popularidad genera un ecosistema ideal para el desarrollo de herramientas avanzadas como este proyecto.
  + Competencia y mercado actual:
    - Actualmente, existen varios bots con funcionalidades avanzadas (como MEE6, Dyno, o Carl-Bot), pero muchos de ellos operan bajo modelos de negocio freemium que limitan las funciones más atractivas a usuarios pagos. Este proyecto busca posicionarse como una alternativa flexible, de bajo costo y con posibilidades de personalización.
  + Avances tecnológicos y accesibilidad:
    - Con el auge de dispositivos móviles más potentes y el acceso a tecnologías multiplataforma, los usuarios esperan soluciones que puedan ser gestionadas desde cualquier lugar. Este proyecto se alinea con esa expectativa al ofrecer una integración completa entre escritorio y móvil.
  + Retos legales y éticos:
    - La gestión de datos personales y las actividades de moderación requieren un enfoque cuidadoso para cumplir con las normativas internacionales de privacidad, como el Reglamento General de Protección de Datos (GDPR) en Europa. El proyecto incluye mecanismos para garantizar que los datos se gestionen de manera segura y ética.
  + Crecimiento y evolución de la API de Discord:
    - Discord actualiza continuamente su API para incluir nuevas funciones y mejorar la experiencia del usuario. Este proyecto está diseñado con un enfoque escalable que facilita su mantenimiento y actualización frente a estos cambios.
* Marco legal
  + Reglamento General de Protección de Datos (GDPR):
  + Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo, de 27 de abril de 2016, relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos. Este reglamento establece las bases para la recopilación, almacenamiento y procesamiento de datos personales en la Unión Europea.
  + Ley Orgánica de Protección de Datos y Garantía de Derechos Digitales (LOPDGDD):
  + Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales, que adapta el GDPR al marco legal español y regula el tratamiento de datos personales en España.
  + Normas de uso de la API de Discord:
  + Discord API Terms of Service (Discord, Inc.). Este documento establece las condiciones legales para el uso de la API de Discord, incluyendo las limitaciones sobre el uso de datos, el respeto a los derechos de los usuarios y las políticas de privacidad.
  + Ley de Servicios de la Sociedad de la Información y de Comercio Electrónico (LSSI-CE):
  + Ley 34/2002, de 11 de julio, que regula las actividades relacionadas con los servicios de la sociedad de la información en España, incluyendo la provisión de servicios digitales y el comercio electrónico.
* Marco teórico
  + 1. Introducción a los Bots de Discord
    - Discord se ha consolidado como una plataforma versátil para la comunicación y gestión de comunidades en diversos ámbitos, incluyendo la educación y el sector profesional. Gracias a su arquitectura abierta y la posibilidad de integrar bots, se ha convertido en una herramienta clave para la automatización de tareas, la gestión de usuarios y la facilitación de interacciones dentro de servidores especializados (López et al., 2021).
  + 2. Uso Educativo de los Bots de Discord
    - El empleo de bots en entornos educativos ha ganado popularidad debido a su capacidad para mejorar la participación estudiantil y automatizar procesos administrativos. Según estudios recientes, los bots en Discord permiten la creación de entornos de aprendizaje más interactivos mediante la integración de recordatorios, evaluaciones automáticas y foros de discusión virtuales (Smith & Johnson, 2022). Además, pueden actuar como tutores virtuales, proporcionando retroalimentación inmediata y facilitando el acceso a materiales de estudio.
    - Un análisis detallado de su aplicación en la educación revela que estos bots pueden aumentar la motivación de los estudiantes al incorporar elementos de gamificación y recompensas basadas en la participación activa(Kumar & Singh, 2021). Así, los bots no solo complementan la enseñanza tradicional, sino que también fomentan el aprendizaje colaborativo y la interacción entre estudiantes.
  + 3. Aplicaciones Profesionales de los Bots de Discord
    - En el sector profesional, los bots de Discord han demostrado ser herramientas eficaces para la gestión de equipos, la automatización de procesos y la organización de eventos virtuales. Según un estudio publicado por García et al. (2023), las empresas han comenzado a implementar bots para la asignación de tareas, el seguimiento de proyectos y la notificación de eventos importantes dentro de sus servidores corporativos.
    - Un caso de estudio señala que la automatización de respuestas y la integración con otras plataformas como Trello o Slack mejoran significativamente la productividad en entornos de trabajo colaborativo (Miller & Thompson, 2020). Además, el uso de inteligencia artificial en los bots permite ofrecer soporte técnico automatizado, reduciendo la carga de trabajo en departamentos de atención al cliente y optimizando la experiencia de usuario.
  + 4. Beneficios y Desafíos
    - Beneficios:
      * **Accesibilidad y disponibilidad**: Los bots operan 24/7, garantizando asistencia constante tanto en el ámbito educativo como en el profesional.
      * **Automatización**: Reducción de carga administrativa mediante la automatización de respuestas, recordatorios y evaluaciones.
      * **Interactividad y engagement**: Mejora de la participación a través de gamificación y respuestas personalizadas.
    - Desafíos:
      * **Seguridad y privacidad**: La recopilación de datos puede representar un riesgo si no se implementan medidas adecuadas de protección.
      * **Mantenimiento y actualización**: Los bots requieren revisiones periódicas para garantizar su funcionamiento y adaptación a nuevas necesidades.
      * **Curva de aprendizaje**: Los usuarios pueden enfrentar dificultades iniciales en la implementación y configuración de los bots.
  + 5. Conclusión
    - Los bots de Discord representan una solución innovadora tanto en el ámbito educativo como profesional. Su capacidad para automatizar tareas, mejorar la interacción y facilitar la gestión de comunidades los convierte en herramientas valiosas en la era digital. A pesar de los desafíos asociados, su implementación adecuada puede generar beneficios significativos, contribuyendo a una mayor eficiencia y participación en distintos entornos.
  + Referencias
    - García, R., López, M., & Pérez, J. (2023). *Implementation of AI-powered bots in professional environments*. En *Proceedings of ACM Digital Conference*. https://dl.acm.org/doi/abs/10.1145/3653707
    - Kumar, A., & Singh, R. (2021). *A Comprehensive Review on Bot Discord Bots for Interactive Learning*. En *International Conference on Computer Science and Engineering*. https://d1wqtxts1xzle7.cloudfront.net/89967831/CSEIT2172100-libre.pdf
    - López, M., Rodríguez, D., & Torres, L. (2021). *Aplicaciones de bots en entornos digitales*. En *LACCEI 2021 Virtual Edition*. https://laccei.org/LACCEI2021-VirtualEdition/full\_papers/FP9.pdf
    - Miller, C., & Thompson, B. (2020). *Automated task management and collaboration through Discord bots*. En *Proceedings of the ACM Conference on Human Factors in Computing Systems*. https://dl.acm.org/doi/abs/10.1145/3334480.3382960
    - Smith, J., & Johnson, K. (2022). *Uso de Discord en la educación y sus beneficios*. En *Proceedings of the LearnTechLib Conference*. https://www.learntechlib.org/p/220539/
* Temporalidad
  + Planteamiento de ideas iniciales: Septiembre
  + Inicio de la documentación: Septiembre-Octubre
  + Desarrollo de la lógica de la aplicación
    - Bot: Octubre-Abril
    - Base de datos: Abril-Mayo
    - Persistencia: Abril-Mayo
    - Eventos: Abril-Mayo
  + Github del proyecto: Octubre-Junio
  + Desarrollo de la interfaz:
    - Escritorio: Abril-Mayo
    - Móvil: Mayo
  + Testing con usuarios reales: Mayo
  + Revisiones de los resultados de las pruebas: Mayo

Tabla

El contenido generado por IA puede ser incorrecto.

# **estudio de la viabilidad del sistema**

* Ventajas de Electron frente a JavaFX
  + Mayor soporte multiplataforma real (Windows, Linux, macOS e incluso Android mediante wrappers).
  + Acceso a una comunidad global y a una gran cantidad de recursos, librerías y componentes visuales modernos.
  + Facilidad para integrar tecnologías web estándar (HTML, CSS, JS/TS), permitiendo interfaces más atractivas y personalizables.
  + Actualizaciones y mantenimiento más sencillos gracias a la separación entre frontend (Electron/web) y backend (Spring Boot).
  + Mejor integración con herramientas de desarrollo modernas y flujos DevOps.
* Reducción de la curva de aprendizaje y acceso a librerías.
  + El uso de tecnologías web modernas reduce la curva de aprendizaje para nuevos desarrolladores, ya que HTML, CSS y JavaScript/TypeScript son ampliamente conocidos y documentados. Además, la integración de librerías de UI, frameworks y utilidades es mucho más sencilla y rápida, lo que acelera el desarrollo y mejora la calidad del producto final.
* Análisis económico (DAFO)
* Debilidades

El éxito de la aplicación depende de la estabilidad y evolución de la plataforma Discord y de Electron.

* + -El desarrollo multiplataforma puede requerir pruebas adicionales para asegurar la compatibilidad en todos los sistemas.
  + La integración entre Electron y Spring Boot puede requerir conocimientos en ambos entornos.
* Amenazas
  + Competencia: Existen otros bots y dashboards con stacks tecnológicos modernos que pueden captar usuarios.
  + Seguridad: La gestión de datos personales y la comunicación entre frontend y backend debe ser especialmente cuidada.
  + Evolución tecnológica: Cambios en la API de Discord, Electron o frameworks web pueden requerir actualizaciones periódicas.
* Fortalezas
  + Soporte multiplataforma real y experiencia de usuario coherente en todos los dispositivos.
  + Acceso a una comunidad y recursos muy amplios gracias al uso de tecnologías web.
  + Facilidad para añadir nuevas funcionalidades y módulos mediante la arquitectura modular.
  + Interfaces modernas y personalizables, alineadas con las expectativas actuales de los usuarios.
* Oportunidades
  + Crecimiento de la plataforma Discord y demanda de soluciones avanzadas de gestión.
  + Posibilidad de integrar fácilmente nuevas librerías, módulos y servicios externos.
  + Mayor facilidad para atraer colaboradores y desarrolladores gracias al stack tecnológico moderno.
  + Potencial para extender la aplicación a nuevos dispositivos (tablets, móviles, etc.) con mínimos cambios.

A diagram of a company

AI-generated content may be incorrect.

* Plan de marketing
  + Análisis de mercado:
    - Público objetivo:
      * Administradores de servidores.
      * Streamers y creadores de contenido con comunidades en Discord.
      * Empresas y grupos de trabajo que utilicen Discord como medio de soporte.
      * Desarrolladores de bots que quieran aprender o entender ciertas funciones.
    - Análisis de la competencia:
      * Existen bots populares que ofrecen todas las funciones, pero de forma individualizada, ninguno proporciona el conjunto completo.
      * Existen frameworks que permiten el desarrollo de bots de forma personal.
      * Existen aplicaciones para el manejo de bots, pero son todas o en su gran mayoría aplicaciones webs.
    - Diferenciación propuesta:
      * Gestión modular.
      * Interfaz simple e intuitiva.
      * Automatización de tareas y control avanzado de usuarios.
      * Adaptable a multitud de sistemas.
  + Estrategias de lanzamiento:
    - Desarrollo y pruebas (beta cerrada para el equipo de desarrollo o en comunidades pequeñas para tener control de todas las variables)
    - Lanzamiento en GitHub como proyecto personal y educativo, sin dar licencia comercial.
    - Promoción y expansión en comunidades ya existentes intentando entrar en contacto con creadores de contenido emergentes que puedan requerir estos servicios.
  + Posibles canales de marketing y promoción:
    - Comunidades existentes de desarrolladores de bots.
    - Redes sociales mediante playlists del desarrollo de esta o promocionando directamente las funciones de la aplicación.
    - Crear un servidor de soporte para agrupar a todos los posibles clientes que vengan desde otros canales.
    - Colaboraciones con otros proyectos para dar el nombre a conocer.
  + Monetización y modelo de negocio:
    - Plan Freemium:
      * Versión gratuita con funcionalidades básicas.
      * Versión premium con acceso a mayor funcionalidad y soporte.
    - Donaciones.
    - Venta de módulos específicos.

# **3. Descripción del entorno tecnológico**

* Descripción de todos los perfiles de usuario que habrá en la aplicación
  + La aplicación contará con los siguientes perfiles de usuario, cada uno con roles y funciones específicas:
    - Administrador del sistema
      * Perfil destinado a los propietarios del servidor de Discord o usuarios avanzados.
      * Funciones principales:
        + Configuración avanzada del Bot (módulos, comandos personalizados, permisos).
        + Gestión de los datos almacenados en la base de datos.
        + Monitorización y auditoría de los registros de acciones del Bot.
    - Moderador del servidor
      * Usuarios encargados de la moderación dentro de los servidores de Discord.
      * Funciones principales:
        + Uso de comandos de moderación (ban, kick, mute, etc.).
        + Acceso limitado a los registros de moderación.
        + Generación de reportes básicos sobre actividades en el servidor.
    - Usuario final
      * Miembros regulares del servidor de Discord que interactúan con el Bot.
      * Funciones principales:
        + Uso de comandos básicos definidos por los administradores (consultas, juegos, encuestas).
        + Interacción con funciones no administrativas del Bot.
* Descripción de las tecnologías para cada perfil
  + Administrador del sistema
    - Aplicación de escritorio y móvil (Electron + SPA/PWA):
      * Herramienta principal para configurar y personalizar el Bot.
      * Interfaz gráfica moderna y multiplataforma, accesible desde Windows, Linux, macOS y Android (mediante wrappers).
      * Acceso a funciones avanzadas: gestión de módulos, comandos, visualización de registros y control del estado del Bot en tiempo real.
    - - Backend (Spring Boot):
      * Servicio centralizado que gestiona la lógica de negocio, la persistencia y la integración con la API de Discord.
      * Base de datos SQL (MySQL, PostgreSQL, etc.) para almacenamiento seguro de configuraciones, registros y datos de usuarios.
  + Moderador del servidor
    - Aplicación móvil (Electron empaquetado para Android o acceso vía navegador como PWA):
    - Acceso rápido y remoto a funcionalidades clave del Bot.
    - Consultar registros básicos de moderación y ejecutar comandos desde dispositivos móviles.
    - Comunicación segura con el backend mediante API RESTful.
  + Usuario final
    - Bot de Discord (integración mediante Spring Boot y librerías Java para Discord):
      * Interacción directa con los usuarios dentro de Discord.
      * Comandos personalizados accesibles desde la interfaz de Discord (por ejemplo, /ayuda, /stats).
      * Respuestas inmediatas mediante WebSockets y eventos de Discord.
    - Servidor local o en la nube:
      * Ejecuta el Bot y procesa las solicitudes de los usuarios finales.
      * Escalable para soportar múltiples usuarios y peticiones concurrentes.
* Arquitectura general
  + Frontend: SPA/PWA desarrollada con tecnologías web modernas (React, Vue o similar), integrada en Electron para escritorio y móvil.
  + Backend: Spring Boot, expone una API RESTful segura y gestiona la lógica de negocio y la persistencia.
  + Comunicación: API RESTful y WebSockets para interacción en tiempo real entre frontend, backend y Discord.
  + Base de datos: SQL (MySQL, PostgreSQL, etc.) para almacenamiento de configuraciones, logs y datos de usuario.

# **4. Especificación de requisitos**

* Requisitos funcionales
  + Frontend (Electron/SPA/PWA)
    - Permitir el inicio de sesión seguro de administradores, moderadores y usuarios finales.
    - Visualizar y gestionar la lista de servidores de Discord asociados al bot.
    - Mostrar información detallada de cada servidor: usuarios, roles, canales, logs y estadísticas.
    - Permitir la configuración y personalización de módulos y comandos del bot desde la interfaz gráfica.
    - Activar/desactivar módulos y comandos en tiempo real.
    - Visualizar logs de acciones de moderación y actividad de usuarios.
    - Gestionar penalizaciones: aplicar, revocar y consultar sanciones desde la interfaz.
    - Mostrar y editar la configuración general del bot (tokens, endpoints, parámetros de seguridad).
    - Permitir la gestión de roles y permisos de usuarios y moderadores.
    - Visualizar y filtrar registros de experiencia y niveles de usuario.
    - Acceso a panel de administración para la gestión avanzada de la base de datos y logs.
    - Notificaciones en tiempo real de eventos relevantes (acciones de moderación, cambios de configuración, etc.).
    - Acceso multiplataforma: escritorio (Windows, Linux, macOS), móvil (Android mediante wrapper o PWA), navegador web (PWA).
  + Aplicación móvil (Electron wrapper/PWA)
    - Permitir acceso rápido a funciones de moderación y consulta de logs desde dispositivos móviles.
    - Visualizar información básica de servidores y usuarios.
    - Ejecutar comandos de moderación (ban, kick, mute, warn) desde el móvil.
    - Recibir notificaciones push de eventos importantes.
    - Consultar y gestionar penalizaciones activas.
    - Acceso seguro mediante autenticación.
  + Backend (Spring Boot)
    - Gestionar la autenticación y autorización de usuarios (admin, moderador, usuario final).
    - Exponer una API RESTful segura para todas las operaciones de frontend y móvil.
    - Gestionar la lógica de negocio para módulos, comandos, penalizaciones, experiencia y roles.
    - Integrar con la API de Discord para recibir eventos y ejecutar acciones en servidores.
    - Registrar todas las acciones relevantes en la base de datos (logs de moderación, cambios de configuración, etc.).
    - Gestionar la persistencia de usuarios, experiencia, penalizaciones, módulos y configuraciones.
    - Proporcionar endpoints para la consulta y actualización de configuraciones del bot.
    - Gestionar la comunicación en tiempo real mediante WebSockets para eventos y notificaciones.
    - Permitir la integración y gestión dinámica de nuevos módulos y comandos.
    - Validar y proteger el acceso a datos sensibles (tokens, credenciales, etc.).
    - Soportar operaciones CRUD completas sobre usuarios, experiencia, penalizaciones y módulos.
    - Permitir la exportación e importación de datos para backup y migraciones.
  + Bot de Discord (integración backend)
    - Procesar comandos recibidos desde Discord y delegar la lógica al backend.
    - Registrar automáticamente la actividad de usuarios (mensajes, tiempo en voz, interacciones).
    - Aplicar penalizaciones y gestionar roles en Discord según las acciones del backend.
    - Enviar mensajes y notificaciones a canales de Discord según eventos definidos.
    - Sincronizar el estado del bot y los módulos activos con la configuración almacenada en el backend.
  + Base de datos (SQL)
    - Almacenar de forma estructurada usuarios, experiencia, penalizaciones, módulos, comandos, logs y configuraciones.
    - Garantizar la integridad referencial y la consistencia de los datos.
    - Permitir consultas optimizadas para informes y visualización en la interfaz.
  + Requisitos transversales
    - Todas las operaciones críticas deben estar protegidas mediante autenticación y autorización.
    - El sistema debe registrar cualquier cambio relevante para auditoría y trazabilidad.
    - La interfaz debe ser accesible y usable para usuarios sin conocimientos técnicos avanzados.
    - El sistema debe ser fácilmente extensible para añadir nuevos módulos, comandos y funcionalidades.
* Requisitos no funcionales

Los requisitos no funcionales especifican las características del sistema que no están relacionadas con funciones específicas, sino con el rendimiento, la seguridad y la usabilidad. Estos requisitos son fundamentales para garantizar la calidad y la eficiencia del sistema.

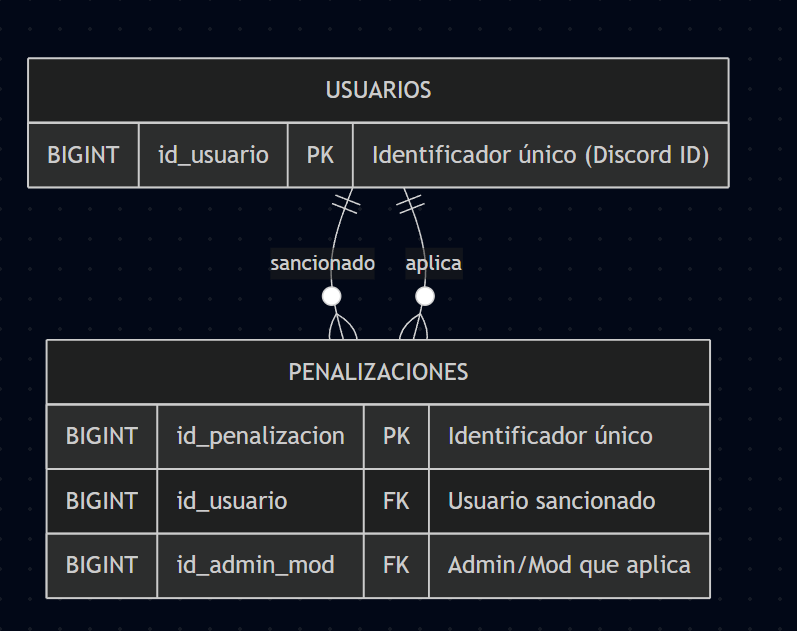
* + El sistema debe ser compatible con Windows, macOS, Linux y Android (mediante Electron o PWA).
  + El frontend debe estar desarrollado como SPA/PWA con tecnologías web modernas (React, Vue, etc.) e integrado en Electron.
  + El backend debe estar desarrollado en Spring Boot y ser fácilmente desplegable en local o en la nube.
  + El sistema debe ser capaz de manejar múltiples solicitudes simultáneas sin afectar el rendimiento.
  + La base de datos SQL debe ser capaz de manejar grandes volúmenes de datos y mantener la integridad de los mismos.
  + El sistema debe garantizar la seguridad de los datos almacenados y transmitidos mediante cifrado SSL/TLS.

Los datos sensibles, como contraseñas y tokens de Discord, deben ser almacenados de forma segura mediante cifrado.

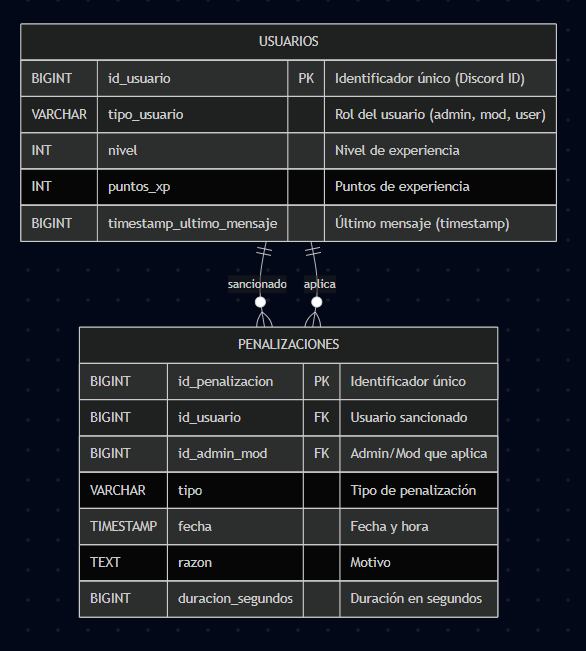
* + El sistema debe seguir las mejores prácticas de seguridad para proteger los datos personales de los usuarios conforme a las normativas de privacidad (GDPR, LOPDGDD).
  + La aplicación debe ser fácil de usar, con interfaces intuitivas y coherentes, proporcionando retroalimentación clara al usuario.
  + El sistema debe ser fácilmente mantenible, con documentación detallada para la instalación, configuración y actualización.
  + Se deben ofrecer actualizaciones periódicas para mantener la compatibilidad con las últimas versiones de la API de Discord y mejorar las funcionalidades del Bot.
  + El sistema debe permitir la integración de nuevos módulos y funcionalidades sin comprometer la estabilidad.
* Requisitos de integración y compatibilidad multiplataforma
  + - La comunicación entre Electron (frontend) y Spring Boot (backend) debe realizarse mediante API RESTful y WebSockets.
  + - El sistema debe ser capaz de detectar la plataforma de ejecución (Windows, Linux, macOS, Android) y adaptar la interfaz según corresponda.
  + - La aplicación debe poder empaquetarse y distribuirse fácilmente para cada plataforma objetivo.
  + - El frontend debe funcionar tanto integrado en Electron como en modo PWA desde navegador moderno.
  + - El backend debe permitir la configuración de endpoints y credenciales de Discord de forma segura y centralizada.
* Comandos implementados(actualmente o a futuro)
  + Comandos de moderació
    - `/ban [usuario] [motivo] [duración]` — Banea a un usuario del servidor.
    - `/kick [usuario] [motivo]` — Expulsa a un usuario del servidor.
    - `/mute [usuario] [duración] [motivo]` — Silencia a un usuario durante un tiempo determinado.
    - `/unmute [usuario]` — Quita el silencio a un usuario.
    - `/warn [usuario] [motivo]` — Advierte a un usuario y registra la advertencia.
    - `/unban [usuario]` — Revoca el baneo de un usuario.
    - `/purge [cantidad]` — Elimina un número de mensajes recientes en un canal.
  + Comandos de consulta y utilidad
    - `/perfil` — Muestra el perfil del usuario, incluyendo experiencia, nivel y penalizaciones.
    - `/nivel` — Muestra el nivel y la experiencia actual del usuario.
    - `/exp [usuario]` — Consulta la experiencia de un usuario específico.
    - `/stats` — Muestra estadísticas generales del servidor.
    - `/logs [usuario]` — Muestra el historial de acciones/moderación de un usuario.
    - `/ayuda` — Muestra la lista de comandos disponibles y su descripción.
  + Comandos administrativos
    - `/config [parámetro] [valor]` — Permite modificar parámetros de configuración del bot.
    - `/modulo [nombre] [activar|desactivar]` — Activa o desactiva módulos funcionales del bot.
    - `/roles [usuario] [añadir|quitar] [rol]` — Gestiona los roles de los usuarios.
    - `/modperms [usuario] [permisos]` — Asigna o revoca permisos de moderador.
    - `/revocar [penalización\_id]` — Revoca una penalización activa.
  + Comandos de interacción y comunidad
    - `/encuesta [pregunta] [opciones]` — Crea una encuesta interactiva en el canal.
    - `/juego [nombre]` — Inicia un minijuego disponible en el bot.
    - `/gif [palabra\_clave]` — Inserta un gif relacionado en el canal.

# **5. Modelo entidad / relación (pendiente de revisión de la versión más reciente de la API)**

* Explicar por qué ha salido ese E/R, de dónde salen las Entidades y de dónde salen las Relaciones.
  + El modelo E/R surge de la necesidad de almacenar los usuarios, su experiencia(la cantidad de interacciones que tienen en el servidor) y las penalizaciones aplicadas a los mismos.
  + Las entidades del modelo son:
    - usuarios
      * `id\_usuario` (BIGINT, PK): Identificador único del usuario (Discord ID).
      * `tipo\_usuario` (VARCHAR): Rol del usuario (admin, mod, user).
      * `nivel` (INT): Nivel de experiencia del usuario.
      * `puntos\_xp` (INT): Puntos de experiencia acumululados.
      * `timestamp\_ultimo\_mensaje` (BIGINT): Marca temporal del último mensaje enviado (útil para control de spam o experiencia).
    - penalizaciones
      * `id\_penalizacion` (BIGINT, PK, AUTO\_INCREMENT): Identificador único de la penalización.
      * `id\_usuario` (BIGINT, FK): Usuario sancionado (referencia a `usuarios.id\_usuario`).
      * `id\_admin\_mod` (BIGINT, FK): Usuario que aplica la penalización (referencia a `usuarios.id\_usuario`).
      * `tipo` (VARCHAR): Tipo de penalización (ban, mute, warn, etc.).
      * `fecha` (TIMESTAMP): Fecha y hora de la penalización.
      * `razon` (TEXT): Motivo de la penalización.
      * `duracion\_segundos` (BIGINT): Duración de la penalización en segundos.
* Plasmar el modelo E/R



* Explicar el origen de los atributos
  + Los atributos de `usuarios` permiten identificar el usuario, su rol y gestionar el sistema de experiencia y niveles.
  + Los atributos de `penalizaciones` permiten registrar sanciones, su duración, motivo y trazabilidad de quién las aplica.
* Explicar por qué se han elegido esas claves primarias
  + `id\_usuario` es el identificador natural y único de cada usuario (Discord ID).
  + `id\_penalizacion` es un identificador autoincremental que garantiza unicidad y facilita la gestión de sanciones.
* Modelo Relacional



# **6. Diagrama de procesos (pendiente de implementación)**

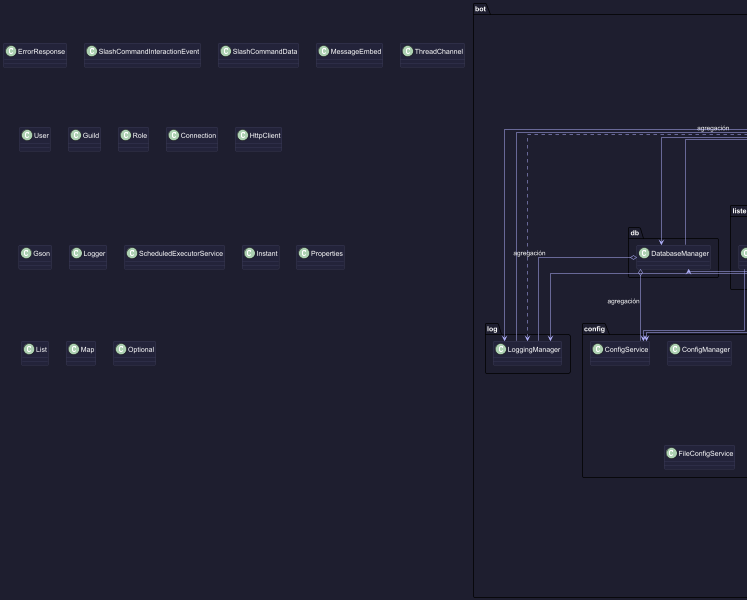
* Casos de Uso

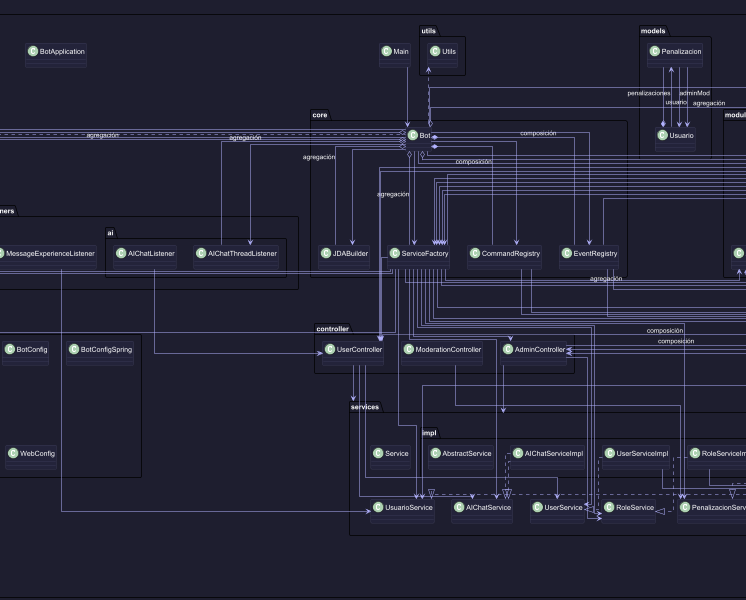
1. UC1: Consultar su perfil
   1. Actores: Usuario, Moderador, Administrador
   2. Descripción: El usuario puede ver información completa de su perfil dentro del servidor
   3. Precondiciones: Usuario registrado en el sistema
   4. Flujo básico:
      1. Usuario ejecuta comando de perfil
      2. Sistema consulta información del usuario
      3. Sistema muestra datos del perfil
   5. Resultado esperado: Usuario visualiza su información personal, nivel, experiencia y estadísticas
2. UC2: Consultar nivel y experiencia
   1. Actores: Usuario, Moderador, Administrador
   2. Descripción: El usuario puede verificar su nivel actual y puntos de experiencia
   3. Precondiciones: Usuario registrado en el sistema
   4. Flujo básico:
      1. Usuario ejecuta comando de nivel
      2. Sistema consulta progreso de experiencia
      3. Sistema calcula y muestra nivel
   5. Resultado esperado: Usuario visualiza su nivel actual, experiencia acumulada y puntos restantes para el siguiente nivel
3. UC3: Interactuar con comandos básicos
   1. Actores: Usuario
   2. Descripción: El usuario utiliza comandos generales del bot para diferentes funcionalidades
   3. Precondiciones: Usuario con permisos básicos
   4. Flujo básico:
      1. Usuario escribe comando con prefijo establecido
      2. Bot procesa el comando
      3. Bot ejecuta la acción solicitada
   5. Resultado esperado: Bot responde apropiadamente al comando ejecutado
4. UC4: Recibir experiencia por actividad
   1. Actores: Usuario
   2. Descripción: El usuario gana puntos de experiencia automáticamente al participar en el servidor
   3. Precondiciones: Usuario registrado en el sistema
   4. Flujo básico:
      1. Usuario realiza actividad (mensaje, tiempo en voz)
      2. Sistema detecta actividad y calcula EXP
      3. Sistema actualiza experiencia del usuario
   5. Resultado esperado: La experiencia del usuario aumenta y puede subir de nivel si alcanza el umbral requerido
5. UC5: Aplicar penalizaciones
   1. Actores: Moderador, Administrador
   2. Descripción: Aplicar sanciones a usuarios que infringen las normas del servidor
   3. Precondiciones: Usuario con rol de moderador o administrador
   4. Flujo básico:
      1. Moderador selecciona usuario a sancionar
      2. Moderador elige tipo de penalización y motivo
      3. Sistema aplica la sanción y la registra
   5. Resultado esperado: Usuario sancionado recibe la penalización correspondiente y se guarda en el historial
6. UC6: Consultar registros de usuarios
   1. Actores: Moderador, Administrador
   2. Descripción: Revisar el historial de actividad y penalizaciones de cualquier usuario
   3. Precondiciones: Usuario con rol de moderador o administrador
   4. Flujo básico:
      1. Moderador solicita historial de un usuario
      2. Sistema consulta registros en base de datos
      3. Sistema muestra información completa
   5. Resultado esperado: Moderador visualiza el historial completo de actividad y sanciones del usuario solicitado
7. UC7: Gestionar roles básicos
   1. Actores: Moderador, Administrador
   2. Descripción: Asignar o remover roles a usuarios del servidor
   3. Precondiciones: Usuario con permisos para gestionar roles
   4. Flujo básico:
      1. Moderador selecciona usuario
      2. Moderador indica rol a asignar/remover
      3. Sistema actualiza roles del usuario
   5. Resultado esperado: Los roles del usuario son modificados según lo solicitado
8. UC8: Revocar penalizaciones
   1. Actores: Moderador, Administrador
   2. Descripción: Eliminar sanciones activas aplicadas a usuarios
   3. Precondiciones: Usuario con permisos para gestionar penalizaciones
   4. Flujo básico:
      1. Moderador consulta penalizaciones activas
      2. Moderador selecciona penalización a revocar
      3. Sistema cancela la penalización
   5. Resultado esperado: La penalización queda registrada como revocada y sus efectos son eliminados
9. UC9: Configurar parámetros del bot
   1. Actores: Administrador
   2. Descripción: Modificar la configuración general y comportamiento del bot
   3. Precondiciones: Usuario con rol de administrador
   4. Flujo básico:
      1. Administrador accede a configuración
      2. Administrador modifica parámetros
      3. Sistema aplica y guarda cambios
   5. Resultado esperado: La configuración del bot se actualiza según los nuevos parámetros establecidos
10. UC10: Gestionar módulos y comandos
    1. Actores: Administrador
    2. Descripción: Activar, desactivar o configurar módulos y comandos específicos
    3. Precondiciones: Usuario con rol de administrador
    4. Flujo básico:
       1. Administrador selecciona módulo/comando
       2. Administrador cambia estado o configuración
       3. Sistema actualiza disponibilidad
    5. Resultado esperado: Los módulos y comandos se comportan según la nueva configuración establecida
11. UC11: Gestionar permisos de moderadores
    1. Actores: Administrador
    2. Descripción: Asignar o revocar permisos específicos a usuarios moderadores
    3. Precondiciones: Usuario con rol de administrador
    4. Flujo básico:
       1. Administrador selecciona moderador
       2. Administrador modifica permisos
       3. Sistema actualiza permisos del moderador
    5. Resultado esperado: El moderador tiene nuevos permisos asignados o removidos en el sistema
12. UC12: Ver estadísticas del servidor
    1. Actores: Administrador
    2. Descripción: Consultar estadísticas globales de actividad y uso del servidor
    3. Precondiciones: Usuario con rol de administrador
    4. Flujo básico:
       1. Administrador solicita estadísticas
       2. Sistema recopila y procesa datos
       3. Sistema presenta información estadística
    5. Resultado esperado: Administrador visualiza datos estadísticos completos sobre el uso del servidor

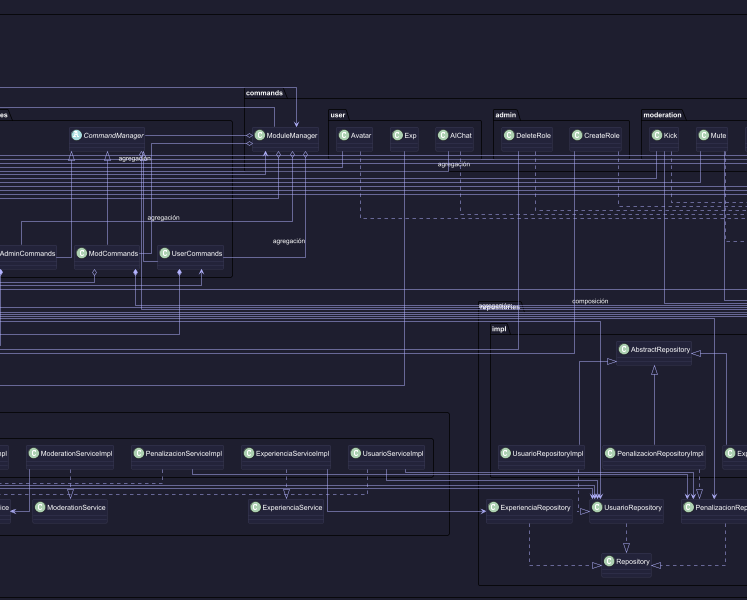
Diagrama

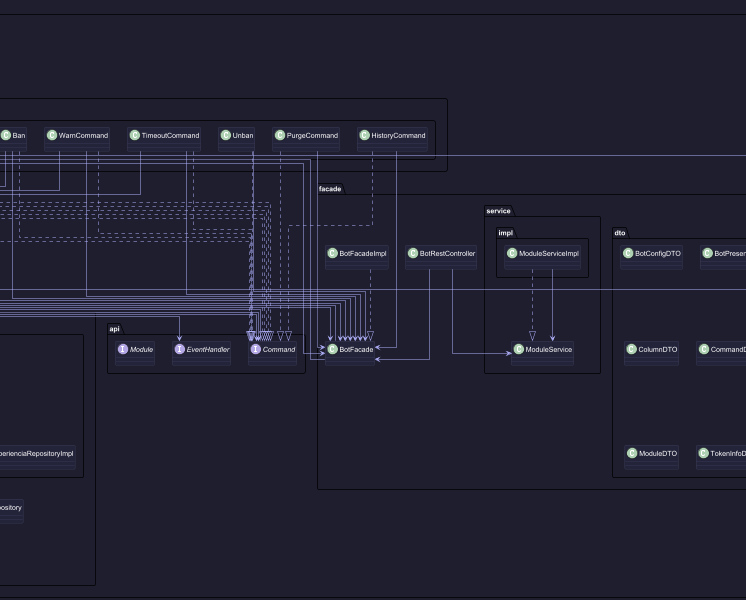
El contenido generado por IA puede ser incorrecto.

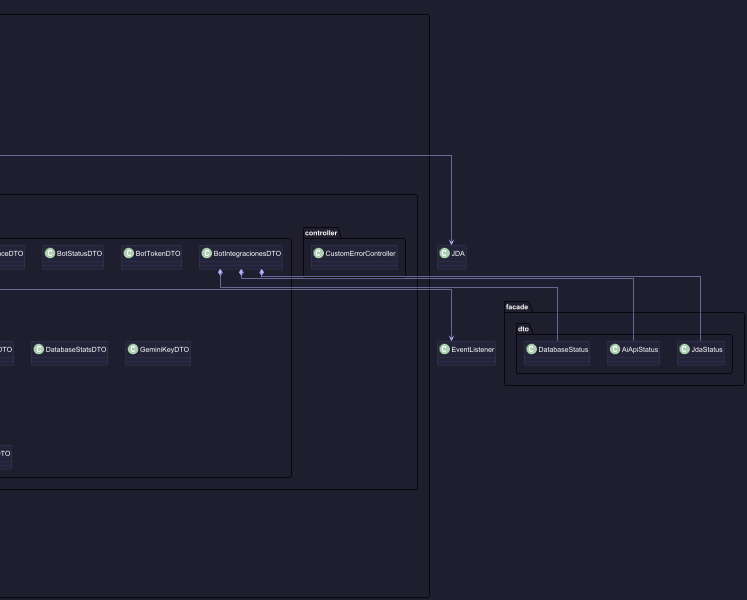
* Flujos principales
  + Inicio de sesión y autenticación:
    - El usuario accede a la aplicación (Electron/web/PWA).
    - Introduce sus credenciales o inicia sesión con Discord.
    - El frontend envía la solicitud al backend, que valida los datos y responde con los permisos y datos del usuario.
  + Ejecución de comandos de moderación:
    - El moderador/administrador ejecuta un comando (ej. /ban) en Discord o desde la interfaz.
    - El frontend o el bot envía la petición al backend.
    - El backend valida permisos, registra la acción en la base de datos y ejecuta la acción en Discord si corresponde.
    - Se actualizan los logs y se notifica al usuario afectado.
  + Gestión de penalizaciones:
    - El moderador aplica una penalización desde la interfaz o Discord.
    - El backend registra la penalización en la tabla `penalizaciones` y actualiza la información del usuario.
    - El usuario sancionado recibe notificación y la penalización queda reflejada en su historial.
  + Actualización de experiencia y nivel:
    - El usuario realiza una acción relevante (mensaje, participación, etc.).
    - El backend actualiza los campos `nivel` y `puntos\_xp` en la tabla `usuarios`.
    - Si corresponde, se notifica al usuario de subida de nivel.
  + Consulta de logs y estadísticas:
    - El usuario o moderador solicita información desde la interfaz.
    - El frontend consulta al backend, que recupera los datos de la base de datos y los devuelve para su visualización.
* Diagrama de clases











# **7. Diseño Interfaz**

* Diagramación y prototipado del diseño gráfico de la interfaz(pendiente de decisión final sobre tecnologías de visualización a usar)

Interfaz de usuario gráfica, Aplicación, Teams

El contenido generado por IA puede ser incorrecto.Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.Captura de pantalla de un celular

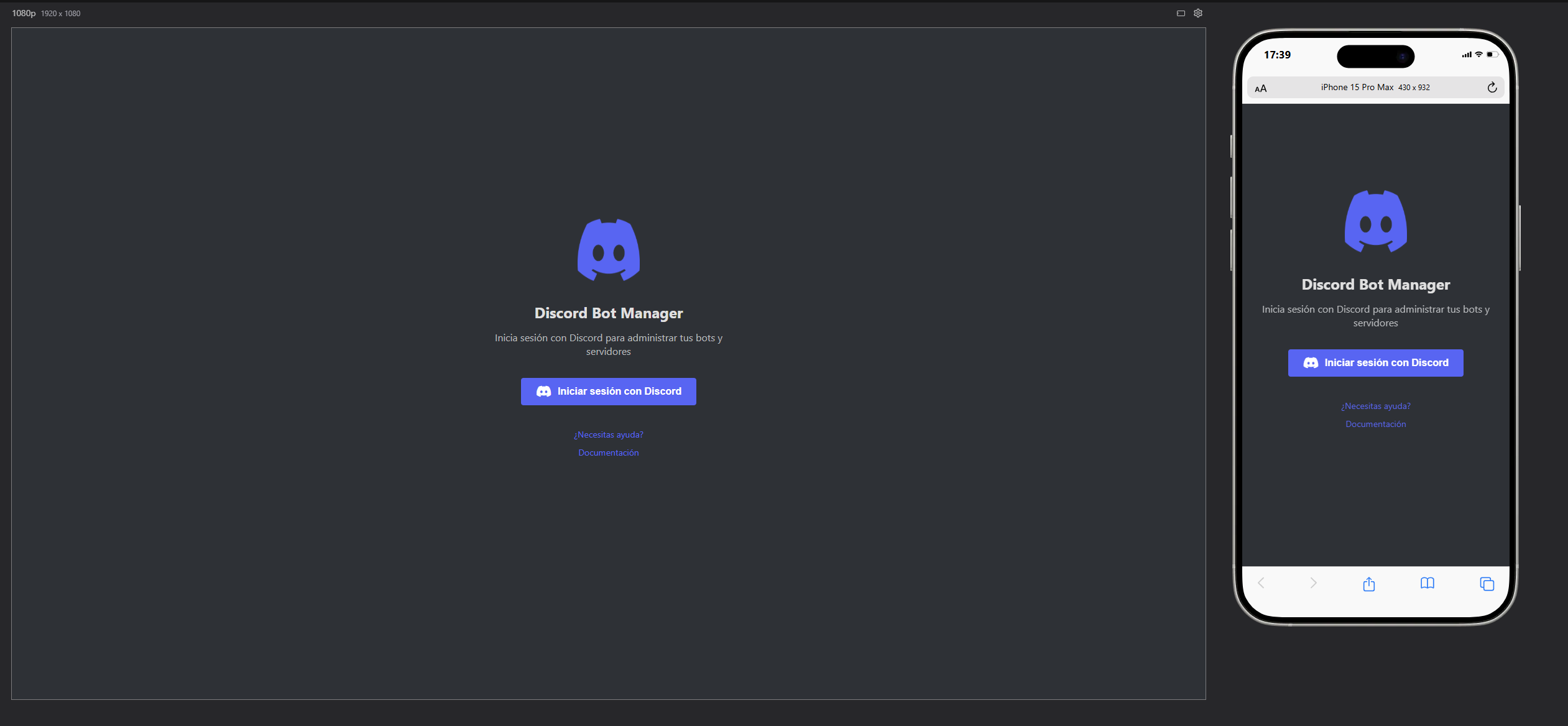
El contenido generado por IA puede ser incorrecto.Captura de pantalla de un celular

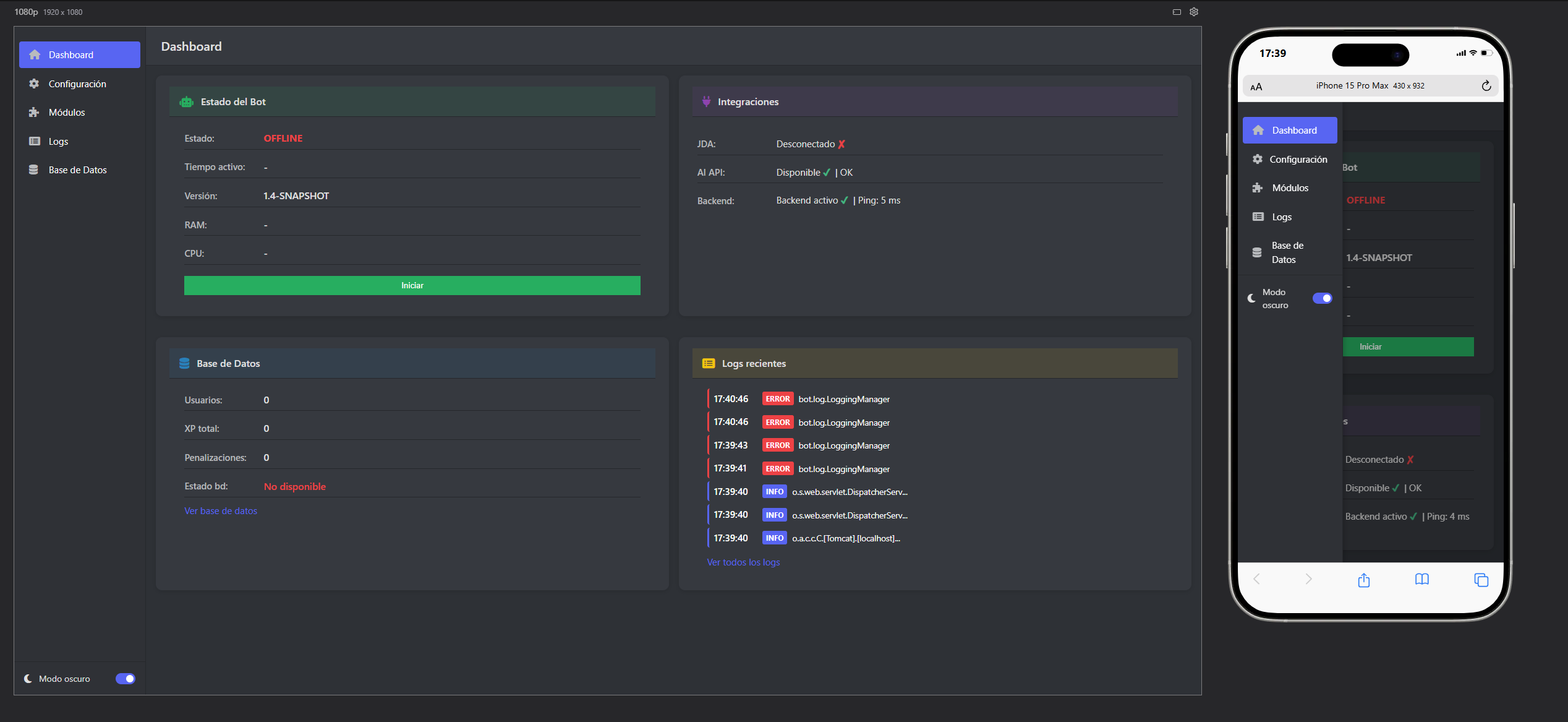
El contenido generado por IA puede ser incorrecto.

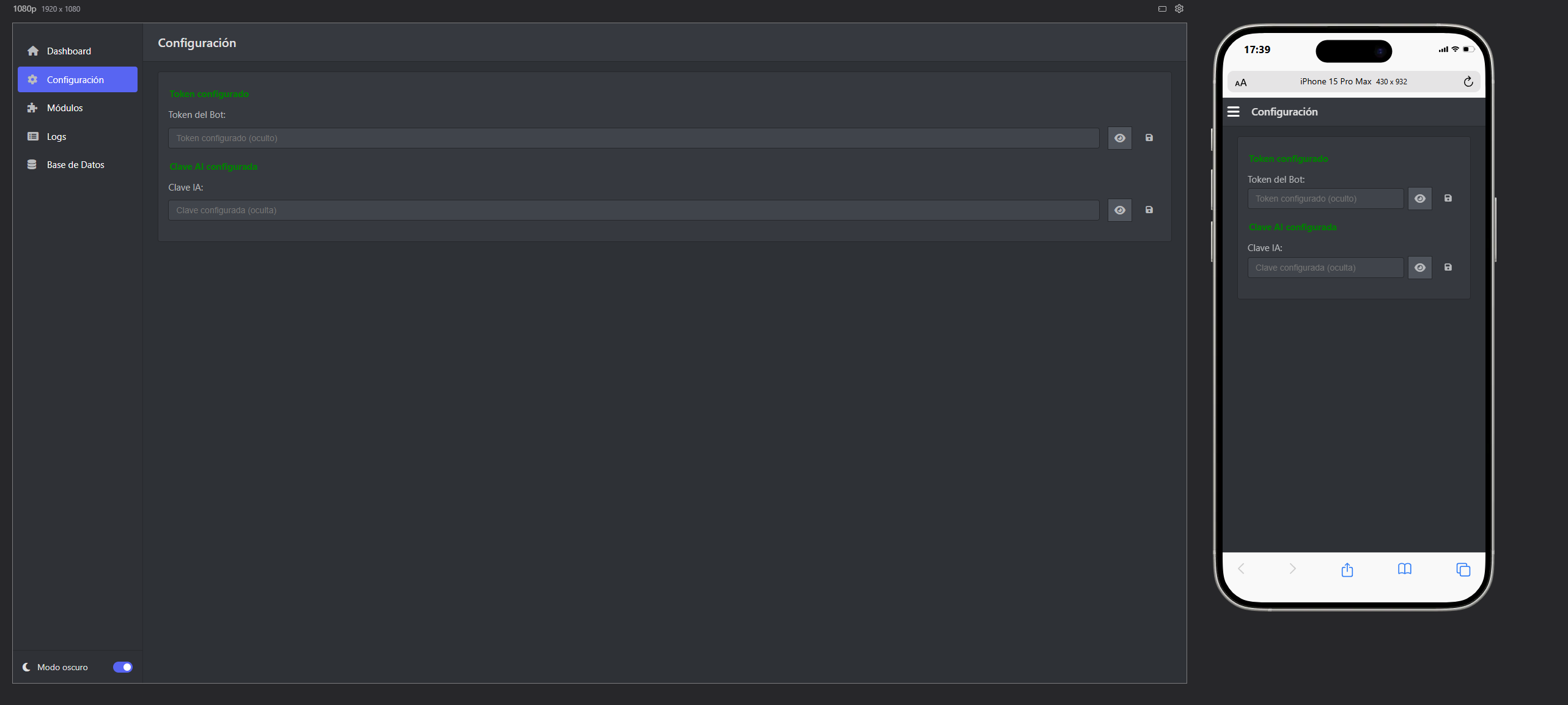
Captura de pantalla de un celular

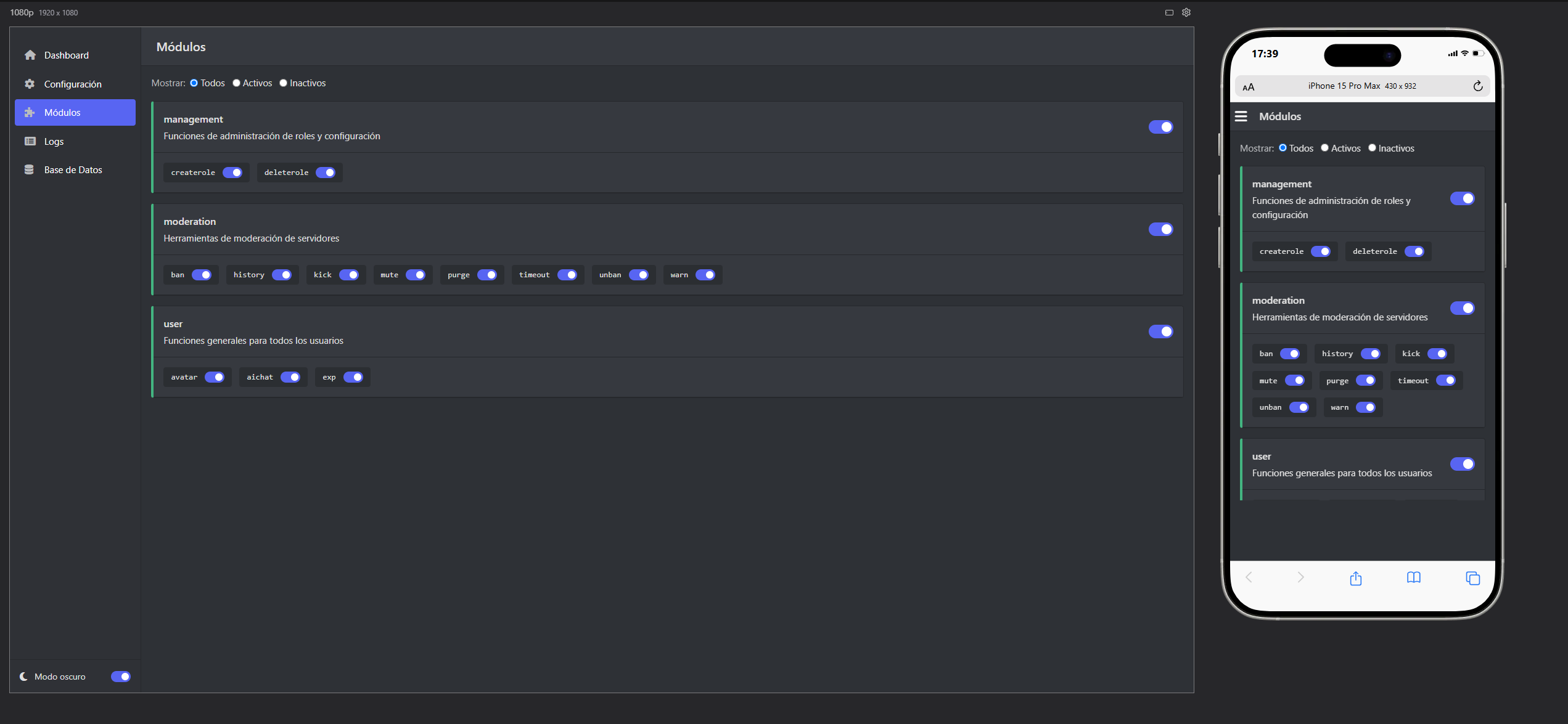
El contenido generado por IA puede ser incorrecto.

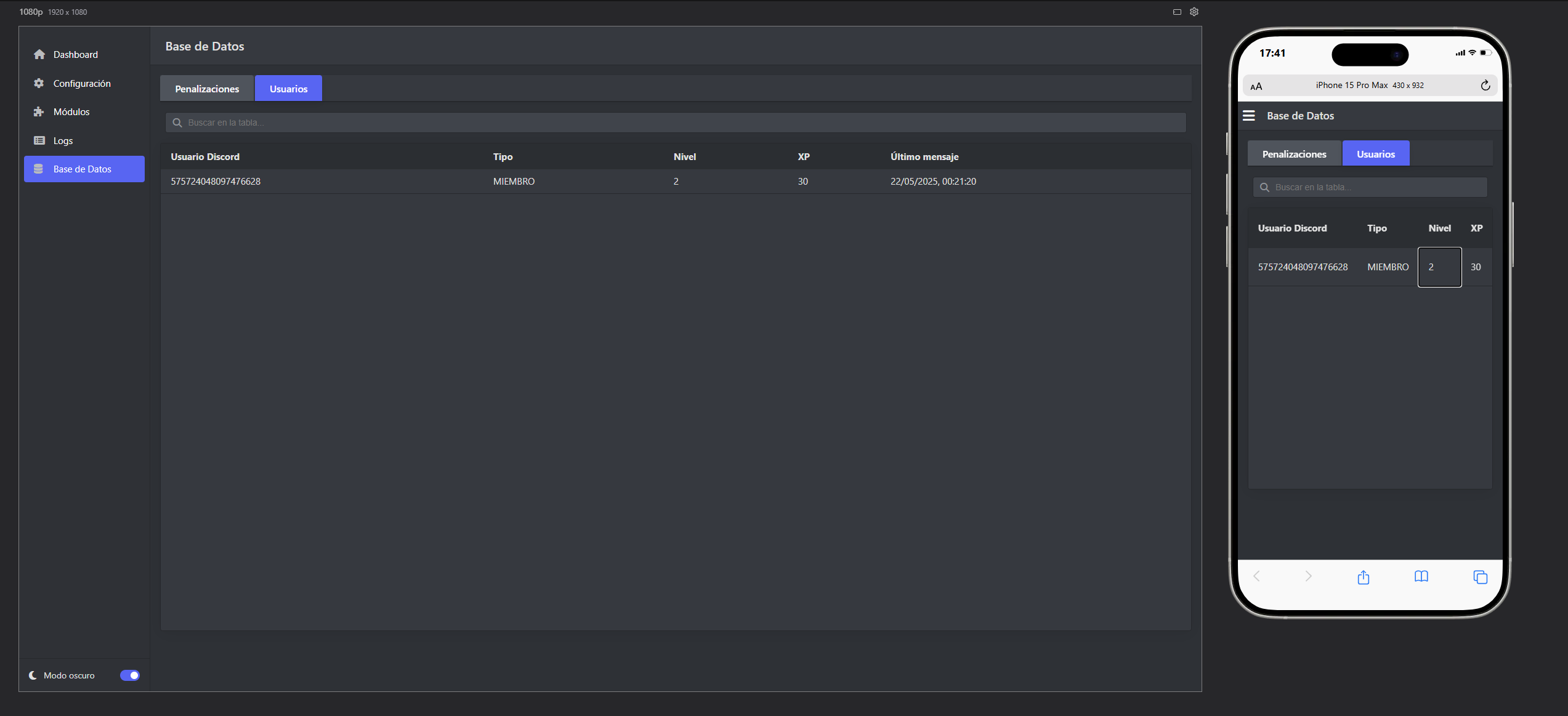
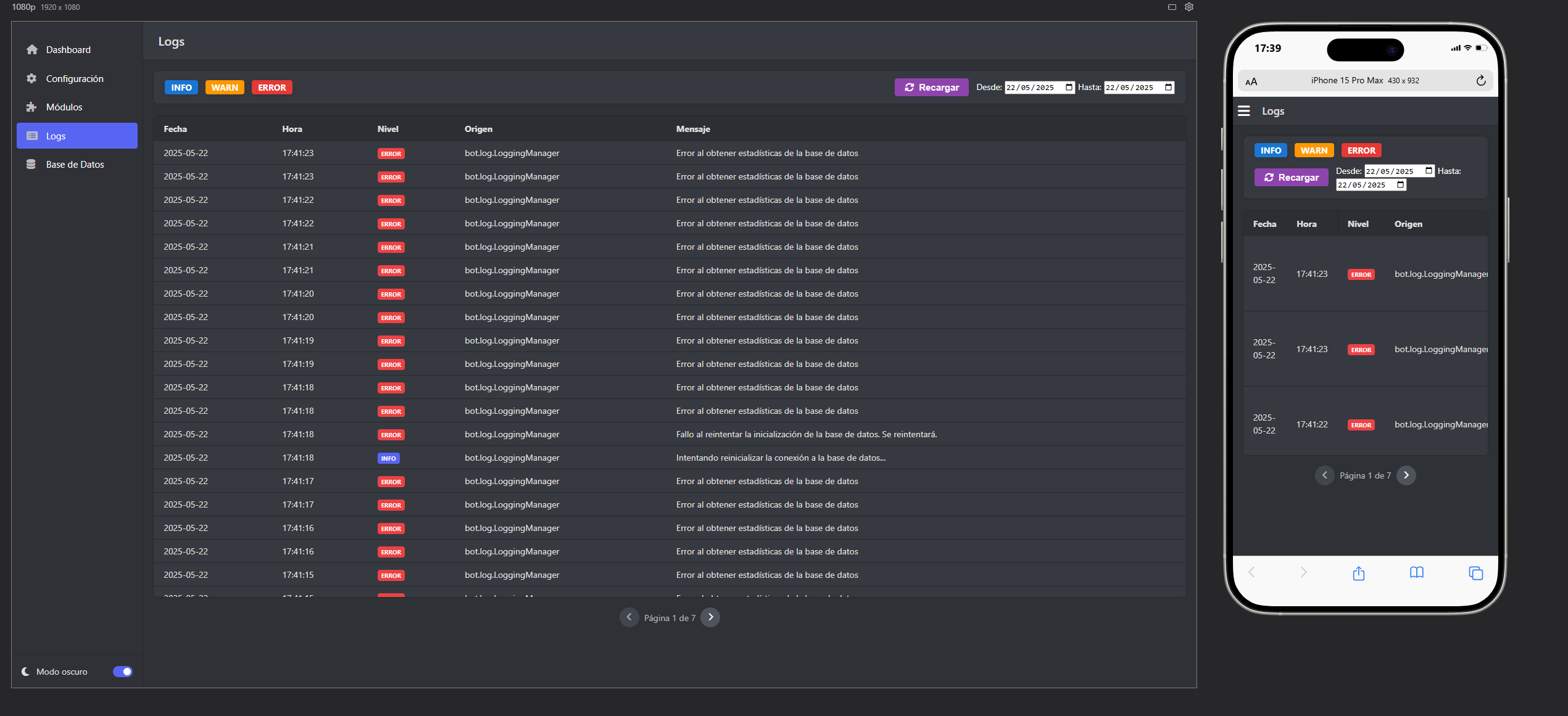
* Diseño final:











* Guías de estilo (pendiente de agregar códigos de colores, etc.)

Teniendo en cuenta que es una aplicación dedicada al manejo de bots utilizados en la red social Discord se intentará en la medida de lo posible imitar los colores de su propia interfaz, es decir, una escala de grises en su modo oscuro con colores de los botones destacados en rojo para las opciones importantes como eliminar o borrar y azul para las que se puedan revertir o sean de movimiento entre ventanas.

Captura de pantalla de computadora

Descripción generada automáticamente

* Colores principales
  + Fondo principal: #36393f
  + Fondo secundario/paneles: #2f3136
  + Fondo de campos de entrada: #40444b
  + Fondo de hover/selección: #4f545c
  + Bordes y separadores: #202225
* Texto
  + Texto principal: #ffffff
  + Texto secundario: #b9bbbe
  + Texto de ayuda o deshabilitado: #72767d
* Botones y elementos interactivos
  + Botón primario/azul: #5865f2
  + Botón de acción/destructivo (rojo): #ed4245
  + Botón de confirmación (verde): #57f287
  + Botón de advertencia (amarillo): #fee75c
  + Elementos destacados (iconos, enlaces): #7289da
  + Botón de alternancia (switch) activo: #5865f2
  + Botón de alternancia (switch) inactivo: #4f545c
* Componentes y estructura
  + Panel lateral (sidebar):
    - Fondo: #2f3136
    - Iconos y texto seleccionados: #7289da
    - Iconos y texto no seleccionados: #b9bbbe
  + Tarjetas y paneles:
    - Fondo: #2f3136
    - Bordes: #202225
  + Tablas:
    - Fondo de cabecera: #2f3136
    - Fondo de filas: #36393f
    - Texto: #ffffff (principal), #b9bbbe (secundario)
  + Etiquetas de estado (INFO, WARN, ERROR):
    - INFO: #5865f2
    - WARN: #fee75c
    - ERROR: #ed4245

# **8. Pruebas**

* Pruebas unitarias

|  |  |  |
| --- | --- | --- |
| Componente | Prueba | Descripción |
| Usuario.java | Creación de usuario | Verificar la correcta creación de un usuario con parámetros válidos |
| Usuario.java | Gestión de experiencia | Comprobar la asignación y obtención de experiencia |
| Usuario.java | Gestión de penalizaciones | Validar la adición de penalizaciones a un usuario |
| Experiencia.java | Inicialización | Verificar la inicialización correcta de experiencia |
| Experiencia.java | Subida de nivel | Comprobar el cálculo correcto al subir de nivel |
| Experiencia.java | Relación con Usuario | Validar la relación bidireccional con Usuario |
| Penalizacion.java | Creación de penalización | Verificar la creación correcta de penalizaciones |
| Penalizacion.java | Conversión de duraciones | Comprobar la conversión correcta de duraciones |
| Penalizacion.java | Relaciones | Validar la relación con Usuario y AdminMod |
| Repositorios | Operaciones CRUD | Verificar operaciones básicas (create, read, update, delete) |
| Repositorios | Consultas personalizadas | Comprobar consultas como findByIdUsuario() o findByTipoUsuario() |
| Repositorios | Manejo de errores | Validar el manejo correcto de errores de SQL |
| UsuarioServiceImpl | actualizarExperiencia() | Verificar el funcionamiento con diferentes valores |
| UsuarioServiceImpl | agregarPenalizacion() | Comprobar con distintos tipos de penalización |
| UsuarioServiceImpl | Búsquedas por tipo | Validar búsquedas por tipo de usuario |
| ExperienciaServiceImpl | aumentarExperiencia() | Verificar incluyendo casos de subidas de nivel |
| ExperienciaServiceImpl | subirNivel() | Comprobar con diferentes niveles iniciales y finales |
| ExperienciaServiceImpl | Búsquedas por nivel | Validar métodos de búsqueda por nivel |
| PenalizacionServiceImpl | tienePenalizacionActiva() | Verificar con penalizaciones válidas e inválidas |
| PenalizacionServiceImpl | revocarPenalizacion() | Comprobar funcionamiento correcto de revocaciones |
| PenalizacionServiceImpl | Búsquedas por fecha | Validar búsquedas por rango de fechas |
| Controladores | Delegación a servicios | Verificar la delegación correcta a los servicios |
| Controladores | Manejo de respuestas | Comprobar el manejo de respuestas y errores |
| CommandManager | Registro de comandos | Verificar el registro correcto de comandos |
| CommandManager | Ejecución de comandos | Comprobar la ejecución con diferentes permisos |
| CommandManager | Activación/desactivación | Validar la activación/desactivación de comandos |
| Utils | Validación de IDs | Verificar validación de IDs de usuario |
| Utils | Formateo de mensajes | Comprobar formateo de mensajes de penalización |
| Utils | Conversión de duraciones | Validar conversión entre diferentes formatos de tiempo |

* Pruebas de integridad

|  |  |  |
| --- | --- | --- |
| Componente | Prueba | Descripción |
| DatabaseManager | Inicialización | Verificar la inicialización correcta de la base de datos |
| DatabaseManager | Carga de esquema | Comprobar la carga correcta del esquema SQL |
| DatabaseManager | Conexiones | Validar conexión y cierre de conexiones |
| Servicios/Repos | Integración básica | Verificar que los servicios utilizan correctamente sus repositorios |
| Servicios/Repos | Transacciones complejas | Comprobar transacciones que involucran múltiples repositorios |
| Servicios/Repos | Operaciones en cascada | Validar el manejo de cascadas (eliminación de usuarios, etc.) |
| Controladores/Servicios | Integración básica | Verificar que los controladores utilizan correctamente sus servicios |
| Controladores/Servicios | Flujos completos | Comprobar flujos como registro, actualización de exp y penalización |
| ModuleManager/Bot | Registro de módulos | Verificar el registro y activación correcta de módulos |
| ModuleManager/Bot | Interacción entre módulos | Comprobar la interacción entre diferentes módulos |
| ModuleManager/Bot | Ciclo de vida | Validar el ciclo de vida completo de comandos |
| Integración JDA | Comandos slash | Verificar respuesta a comandos slash |
| Integración JDA | Eventos del servidor | Comprobar manejo de eventos del servidor |
| Integración JDA | Interacciones | Validar interacciones de usuarios |

* Otras pruebas(pueden incluir pruebas no realizadas si son de funcionalidades a futuro):
  + Pruebas de backend (Spring Boot)
  + Prueba: Crear usuario válido
    - Input: POST `/api/usuarios` con body `{id\_usuario: 123456789, tipo\_usuario: 'user', nivel: 1, puntos\_xp: 0}`
    - Output: Usuario insertado en la tabla `usuarios` (HTTP 201)
    - Resultado: OK
  + - Prueba: Crear usuario con ID duplicado
    - Input: POST `/api/usuarios` con body `{id\_usuario: 123456789, ...}` (ya existente)
    - Output: Error de clave primaria duplicada (HTTP 400 o 500)
    - Resultado: OK
  + Prueba: Crear usuario con campos vacíos o nulos
    - Input: POST `/api/usuarios` con body `{}`
    - Output: Error de validación/SQL (HTTP 400 o 500)
    - Resultado: OK
  + Prueba: Actualizar experiencia de usuario
    - Input: PUT `/api/db/experiencia/{id}` con body `{puntos\_xp: 150}`
    - Output: Campo `puntos\_xp` actualizado (HTTP 200)
    - Resultado: OK
  + Prueba: Aplicar penalización tipo 'mute'
    - Input: POST `/api/penalizaciones` con body `{id\_usuario: 123456789, id\_admin\_mod: 987654321, tipo: 'mute', fecha: ..., razon: 'spam', duracion\_segundos: 600}`
    - Output: Nueva fila en `penalizaciones` (HTTP 201)
    - Resultado: OK
  + Prueba: Insertar penalización con usuario inexistente
    - Input: POST `/api/penalizaciones` con body `{id\_usuario: 999999999, ...}`
    - Output: Error de integridad referencial (HTTP 400 o 500)
    - Resultado: OK
  + Prueba: Obtener información de usuario existente
    - Input: GET `/api/user/123456789`
    - Output: Objeto usuario (HTTP 200)
    - Resultado: OK
  + Prueba: Obtener información de usuario inexistente
    - Input: GET `/api/user/999999999`
    - Output: HTTP 404
    - Resultado: OK
  + Prueba: Obtener historial de penalizaciones de usuario
    - Input: GET `/api/user/123456789/history`
    - Output: Lista de penalizaciones (HTTP 200)
    - Resultado: OK
  + Prueba: Obtener logs
    - Input: GET `/api/logs?limit=10`
    - Output: Lista de logs (HTTP 200)
    - Resultado: OK
  + Prueba: Baneo de usuario
    - Input: POST `/api/ban?guildId=1&userId=123456789&reason=spam`
    - Output: Usuario baneado, penalización registrada (HTTP 200)
    - Resultado: OK
  + Prueba: Obtener configuración del bot
    - Input: GET `/api/config`
    - Output: Configuración actual (HTTP 200)
    - Resultado: OK
  + Prueba: Guardar token del bot
    - Input: POST `/api/config/token` con body `{token: 'xxxx'}`
    - Output: HTTP 200
    - Resultado: OK
  + Prueba: Obtener estado de integraciones
    - Input: GET `/api/bot/integraciones`
    - Output: DTO de integraciones (HTTP 200)
    - Resultado: OK
  + Prueba: Obtener nombres de tablas
    - Input: GET `/api/db/tables`
    - Output: Lista de nombres de tablas (HTTP 200)
    - Resultado: OK
  + Prueba: Obtener columnas de una tabla
    - Input: GET `/api/db/tables/usuarios/columns`
    - Output: Lista de columnas (HTTP 200)
    - Resultado: OK
  + Prueba: Obtener datos de una tabla
    - Input: GET `/api/db/tables/usuarios/data`
    - Output: Lista de filas (HTTP 200)
    - Resultado: OK
  + Prueba: Obtener estadísticas de la base de datos
    - Input: GET `/api/db/stats`
    - Output: DTO de estadísticas (HTTP 200)
    - Resultado: OK
  + Prueba: Ping backend
    - Input: GET `/api/botfacade/ping`
    - Output: "pong" (HTTP 200)
    - Resultado: OK
  + Pruebas de frontend (Electron/web)
    - Prueba: Login y acceso a la interfaz principal (si aplica autenticación)
    - Prueba: Visualización de módulos y comandos (`/api/modules`)
    - Prueba: Activar/desactivar módulos y comandos (`/api/modules/{nombre}/enable|disable`)
    - Prueba: Visualización de usuarios y penalizaciones (usando los endpoints `/user/{id}` y `/user/{id}/history`)
    - Prueba: Visualización y edición de configuración (`/api/config`)
    - Prueba: Visualización de logs (`/api/logs`)
    - Prueba: Acceso a paneles restringidos según permisos (si aplica)
  + Pruebas de integración (backend + frontend + Discord)
    - Prueba: Ejecutar comando `/ban` desde Discord y comprobar registro en la base de datos (POST `/api/ban`)
    - Prueba: Aplicar penalización desde la interfaz y comprobar reflejo en Discord y base de datos (POST `/api/penalizaciones`)
    - Prueba: Cambiar configuración desde la interfaz y comprobar efecto en el bot (POST `/api/config/token`, etc.)
  + Pruebas de compatibilidad multiplataforma
    - Prueba: Iniciar aplicación en Windows, Linux, macOS y Android (Electron wrapper o PWA)
    - Prueba: Visualización y funcionamiento correcto en diferentes resoluciones y dispositivos
  + Pruebas de usuario y usabilidad
    - Prueba: Navegación por la interfaz y ejecución de acciones básicas (usuarios reales)
    - Prueba: Introducción de datos inválidos en formularios y comprobación de mensajes de error
    - Prueba: Acceso a funcionalidades restringidas sin permisos (si aplica)
  + Pruebas de concurrencia y seguridad
    - Prueba: Acceso concurrente a endpoints críticos (alta carga de peticiones)
    - Prueba: Intentos de acceso a endpoints sin autenticación (si aplica seguridad)
    - Prueba: Intento de inyección SQL en campos de entrada (por ejemplo, motivo de penalización)

# **9. Manual de usuario (Pendiente de creación de instalador y aplicación final)**

* Manual del cliente
  + Acceso a la aplicación
    - El sistema se compone de dos partes independientes: el backend (Spring Boot) y el frontend (Electron, web o wrapper para Android).
    - El backend debe estar en ejecución para que las aplicaciones cliente puedan funcionar correctamente.
  + Acceso desde escritorio (Electron)
    - Asegúrate de que el backend está ejecutándose (ver instrucciones de instalación).
    - Ejecuta la aplicación de escritorio (archivo ejecutable generado por Electron).
    - Inicia sesión o accede con tu cuenta de Discord si la autenticación está habilitada.
    - Navega por las diferentes secciones: servidores, módulos, logs, configuración, etc.
  + Acceso desde navegador web (SPA/PWA)
    - Asegúrate de que el backend está ejecutándose.
    - Abre el navegador y accede a la URL donde se encuentra desplegada la web (por ejemplo, `https://localhost:8443/`).
    - Inicia sesión o accede con tu cuenta de Discord si la autenticación está habilitada.
    - 4. Utiliza la interfaz web para gestionar el bot y consultar información.
  + Acceso desde móvil (wrapper Android o PWA)
    - Asegúrate de que el backend está ejecutándose y accesible desde la red local o internet.
    - Abre la app móvil instalada (Electron wrapper) o accede a la PWA desde el navegador móvil.
    - Inicia sesión y navega por las funciones disponibles.
* Manual de instalación y despliegue
  + Backend (Spring Boot)
    - Descarga el proyecto o el archivo JAR generado.
    - Abre una terminal y navega hasta la carpeta del backend.
    - Ejecuta el backend con el comando:
    - ```
    - java -jar nombre-del-backend.jar
    - ```
    - El backend estará disponible en el puerto configurado (por ejemplo, `https://localhost:8443/api`).
    - \*\*Nota:\*\* La versión web SPA/PWA del frontend se sirve automáticamente desde el mismo backend en la ruta raíz (por ejemplo, `http://localhost:8080/`). No es necesario ejecutar un servidor web adicional para la interfaz web.
  + Frontend (Electron)
    - Descarga la aplicación de escritorio para tu sistema operativo.
    - Ejecuta el archivo correspondiente (`.exe`, `.AppImage`, `.dmg`, etc.).
    - La aplicación detectará automáticamente el backend si está en la misma máquina o red (puedes configurar la URL del backend en la sección de configuración si es necesario[característica premium]).
  + Frontend (Web SPA/PWA)
    - No es necesario instalar ni ejecutar un servidor web adicional.
    - Accede a la interfaz web directamente desde el navegador en la URL del backend (por ejemplo, `https://localhost:8443/`).
    - Si usas una versión desplegada en otro servidor, accede directamente a la URL pública proporcionada.
  + Frontend (Android wrapper)
    - Instala la app en tu dispositivo Android (APK proporcionado).
    - Abre la app y configura la URL del backend si es necesario.
    - Utiliza la app como en escritorio o web.

# **10. CONCLUSIONES**

El desarrollo de Discord Bot Manager ha supuesto un proyecto integral y multidisciplinar que ha permitido aplicar y consolidar los conocimientos adquiridos en el ciclo de Desarrollo de Aplicaciones Multiplataforma. La elección de una arquitectura moderna basada en Electron para el frontend (escritorio, web y móvil) y Spring Boot para el backend ha facilitado la creación de una solución robusta, escalable y fácilmente mantenible.

Se ha conseguido implementar con éxito un sistema completo de gestión de usuarios, experiencia, penalizaciones y módulos, permitiendo la administración avanzada de bots de Discord desde diferentes plataformas. La integración con la API de Discord y la gestión de eventos en tiempo real han supuesto retos técnicos que se han resuelto mediante una arquitectura modular y el uso de patrones de diseño adecuados.

La separación clara entre frontend y backend ha demostrado ser clave para la mantenibilidad y la escalabilidad del sistema, permitiendo añadir nuevas funcionalidades y adaptar la interfaz a las necesidades de los usuarios sin afectar la lógica de negocio. El uso de tecnologías web modernas ha reducido la curva de aprendizaje y ha facilitado la integración de recursos y librerías externas.

El proyecto ha reforzado la importancia de una buena planificación, el diseño previo de la arquitectura y la documentación exhaustiva. La gestión de dependencias y la automatización de pruebas han contribuido a la calidad y estabilidad del producto final.

Entre las posibles mejoras futuras destacan la creación de nuevos módulos (audio, IA, integración con servicios externos), la ampliación de la compatibilidad móvil y la optimización de la experiencia de usuario. El sistema está preparado para evolucionar y adaptarse a las necesidades cambiantes de la comunidad de Discord y de los administradores de servidores.

En resumen, Discord Bot Manager es una solución flexible, potente y preparada para el crecimiento, que pone en valor el aprendizaje práctico y la aplicación real de los conocimientos adquiridos en el ámbito del desarrollo multiplataforma.

# **11. Anexo**

* Bibliografía
* Discord API. (s.f.). \*Discord Developer Portal\*. <https://discord.com/developers/docs/intro>
* Spring Boot. (s.f.). \*Spring Boot Reference Documentation\*. <https://docs.spring.io/spring-boot/>
* Electron. (s.f.). \*Electron Documentation\*. <https://www.electronjs.org/docs/latest>
* Discord.js. (s.f.). \*Guide\*. <https://discordjs.guide/#before-you-begin>
* Discord jda(s.f.). \*Documentation\*, <https://jda.wiki/introduction/jda/>
* - Java Platform, Standard Edition. (s.f.). \*Java SE Documentation\*. <https://docs.oracle.com/en/java/javase/>
* - PlantUML. (s.f.). \*PlantUML Documentation\*. <https://plantuml.com/es/>
* - DeepWiki. (2024). \*DiscordBot\_app - Documentación y recursos del proyecto\*. <https://deepwiki.com/PelayoPS/DiscordBot_app>
* - Pelayo Palacio Suárez. (2024). \*Discord Bot Manager: Trabajo final de ciclo en desarrollo de aplicaciones multiplataforma\* [Manuscrito inédito]. Oviedo.
* Referencias a texto y/o web de ayuda
  + PelayoPS. (s.f.-a). JAVA\_discord\_bot [Repositorio GitHub]. GitHub. <https://github.com/PelayoPS/JAVA_discord_bot>
  + PelayoPS. (s.f.-b). DiscordBotDev [Repositorio GitHub]. GitHub. <https://github.com/PelayoPS/DiscordBotDev>
  + PelayoPS. (s.f.-c). DiscordBot [Repositorio GitHub]. GitHub. <https://github.com/PelayoPS/DiscordBot>
  + PelayoPS. (s.f.-d). UML-Generator [Repositorio GitHub]. GitHub. <https://github.com/PelayoPS/UML-Generator>