Instituto Puig Castellar

Ciclo formativo: DAW

Grado: CFGS

Proyecto intermodular

SPOTYDLE

Desarrollo de un juego de retos diarios musicales con lógica de búsqueda inteligente y feedback en tiempo real.

Autor: Carlos García Montes

Tutor: David Delgado Santamaría

Curso: 2025-2026

Fecha de entrega: 17/05/2026

Resumen del proyecto

El proyecto **Spotydle** consiste en el diseño y desarrollo de un juego web inspirado en la dinámica de retos diarios, centrado exclusivamente en el ámbito musical. El objetivo principal es ofrecer una experiencia lúdica y de enriquecimiento cultural donde los usuarios puedan poner a prueba sus conocimientos musicales, tratando de adivinar canciones a partir de un sistema progresivo de pistas.

El núcleo de la aplicación es el desafío diario, que propone cada jornada una nueva canción que los jugadores deben identificar en un número limitado de intentos. Spotydle busca fomentar el interés por la música de una manera divertida e interactiva, facilitando el descubrimiento de nuevos artistas y géneros mediante un hábito de juego cotidiano. El desarrollo se ha estructurado para permitir la inclusión de diferentes modos de juego según el estilo musical, adaptándose así a los gustos de un público amplio y diverso.

Para alcanzar estos objetivos, se ha seguido una metodología basada en la planificación por fases, priorizando la usabilidad y la integración fluida de contenidos musicales. El resultado es una plataforma funcional que actúa como puente entre el entretenimiento y la cultura musical, permitiendo disfrutar y descubrir nueva música de una manera accesible.

Palabras clave

Juego web · Reto diario · Descubrimiento musical · Cultura musical · Entretenimiento

Abstract

The Spotydle project focuses on the design and development of a web-based game inspired by daily challenge dynamics, centered exclusively on the musical sphere. The primary goal is to offer a playful and culturally enriching experience where users can test their musical knowledge by attempting to guess songs through a progressive hint system.

At the core of the application lies the daily challenge, which presents a new song each day for players to identify within a limited number of attempts. Spotydle aims to foster interest in music in a fun and interactive way, facilitating the discovery of new artists and genres through a daily gaming habit. The development has been structured to allow for the inclusion of different game modes based on musical styles, thus catering to the tastes of a broad and diverse audience.

To achieve these goals, a phased planning methodology was adopted, prioritizing usability and the seamless integration of musical content. The result is a functional platform that acts as a bridge between entertainment and musical culture, allowing for the enjoyment and discovery of music in an accessible manner.

Keywords

Web game · Daily challenge · Music discovery · Musical culture · Entertainment

Índice

1. Presentación del proyecto

1.1 Introducción

El presente proyecto consiste en el desarrollo de **Spotydle**, una aplicación web interactiva diseñada como un juego de retos musicales diarios. El propósito fundamental de esta plataforma es fusionar el entretenimiento digital con el enriquecimiento cultural, ofreciendo a los usuarios un espacio donde poner a prueba y ampliar sus conocimientos musicales de una manera dinámica y accesible.

La idea principal de la solución se basa en una mecánica de adivinanzas inspirada en juegos de lógica populares, pero trasladada íntegramente al ámbito de la música. El núcleo de la experiencia es el desafío diario, en el cual se presenta una canción distinta cada jornada. Los jugadores deben identificar tanto al artista como el título de la obra dentro de un número limitado de intentos, utilizando para ello un sistema de pistas progresivas que incluye fragmentos de audio, datos técnicos y elementos visuales.

Spotydle no solo busca ofrecer un reto lúdico, sino también fomentar el descubrimiento de nuevos géneros y artistas, adaptando la experiencia a diversos perfiles de usuario mediante la implementación de distintos modos de juego basados en estilos musicales. El software ha sido diseñado priorizando la usabilidad y la integración fluida de contenidos multimedia, garantizando una interacción intuitiva en diversos dispositivos.

A lo largo de esta memoria se detallan las diferentes etapas que han dado forma al proyecto, incluyendo el análisis de los requisitos funcionales, el diseño de la arquitectura técnica, las decisiones tomadas durante la fase de desarrollo y las conclusiones extraídas tras la obtención de un producto mínimo viable totalmente funcional.

1.2 Contexto

En los últimos años, el sector del entretenimiento digital ha experimentado una notable transformación impulsada por el éxito de juegos de desafíos diarios, siendo *Wordle* el referente principal de esta tendencia. Este fenómeno ha consolidado un modelo de micro-juegos web accesibles que fomentan la participación recurrente y breve del usuario, integrándose fácilmente en la rutina cotidiana.

Dentro de esta corriente, han surgido diversas adaptaciones enfocadas en el ámbito musical. No obstante, tras un análisis del panorama actual, se observa que muchas de estas propuestas presentan carencias significativas en cuanto a su profundidad y contenido. Por una parte, un gran número de aplicaciones existentes se limita a mecánicas excesivamente simplificadas, ofreciendo únicamente una selección aleatoria de canciones sin permitir al usuario elegir entre diferentes categorías o modos de juego.

Por otra parte, se detecta un vacío importante en lo que respecta a la localización del contenido para el público hispanohablante. La mayoría de las plataformas internacionales centran su catálogo en el mercado anglosajón, omitiendo géneros de gran relevancia cultural y comercial en la actualidad, como es el caso de la música urbana española. Esta falta de diversidad musical dificulta que determinados perfiles de usuarios se sientan representados o motivados por los retos propuestos.

Asimismo, el funcionamiento técnico de algunas de estas alternativas no siempre cumple con los estándares de usabilidad y rendimiento esperados en aplicaciones modernas, presentando interfaces poco intuitivas o integraciones multimedia deficientes. En este escenario, surge la oportunidad de desarrollar una plataforma que no solo robustezca la lógica del juego, sino que también atienda a las preferencias específicas de la cultura musical hispana, proporcionando una experiencia más completa y personalizada.

1.3 Justificación

La necesidad de desarrollar **Spotydle** se fundamenta, en primer lugar, en la detección de una sinergia idónea entre los hábitos de consumo de los aficionados a la música y las dinámicas de los micro-juegos de retos diarios. Se ha identificado que el público consumidor de música, independientemente de sus preferencias de género, se encuentra estrechamente vinculado al uso de redes sociales de consumo rápido como TikTok o X (anteriormente Twitter). Este perfil de usuario interactúa de forma constante con contenidos digitales dinámicos, lo que convierte al formato de desafío diario en el complemento ideal para su rutina digital. La confluencia entre la comunidad de oyentes y la naturaleza compartible de la aplicación genera un factor de viralidad muy elevado, propiciando un crecimiento orgánico de la plataforma a través de la interacción social y competitiva entre los propios usuarios.

Asimismo, el proyecto encuentra su razón de ser en el valor que aporta a la divulgación y al enriquecimiento cultural en toda su diversidad. La plataforma se justifica como una herramienta interactiva diseñada específicamente para incentivar la exploración del panorama musical hispanohablante, pero también dando lugar al recuerdo de viejos clásicos internacionales. El sistema permite a los usuarios adoptar un rol activo en el consumo multimedia, sirviendo como un canal para el descubrimiento de nuevos artistas emergentes en diversos géneros, al tiempo que ejerce una función retrospectiva que invita a recordar canciones antiguas o clásicos que marcaron la historia de la música. Este enfoque transversal transforma el juego en un puente directo hacia la preservación y difusión de la cultura musical global.

Desde una perspectiva académica y tecnológica, enmarcada en las competencias del ciclo formativo de DAW, el desarrollo de esta aplicación plantea un escenario de alta exigencia técnica que justifica el tiempo invertido. La implementación de una solución robusta y escalable ha requerido el dominio de una arquitectura Full-Stack moderna mediante el framework **Next.js**, la gestión de estados complejos en el cliente y la integración de sistemas de autenticación seguros a través de **NextAuth**. Adicionalmente, el proyecto ha demandado la resolución de retos técnicos reales del desarrollo de software actual, tales como el consumo eficiente de la API de iTunes, la sincronización precisa de flujos de reproducción y

la aplicación de parches de rendimiento multimedia para garantizar la compatibilidad en dispositivos móviles, especialmente en el ecosistema iOS.

1.4 Objetivos

Objetivo general

Desarrollar una aplicación web funcional de retos musicales diarios que, mediante un sistema progresivo de pistas y mecánicas de juego dinámicas, incentive el conocimiento y la cultura musical de los usuarios de forma accesible.

Objetivos específicos

Para alcanzar el objetivo general, se han definido las siguientes metas técnicas y funcionales:

- Implementar un sistema de reto diario automatizado que presente una composición musical única cada 24 horas.
- Desarrollar un motor de pistas dinámico que incluya fragmentos de audio, metadatos técnicos (año y género), elementos visuales procesados y pistas de texto.
- Integrar la API de iTunes para la recuperación y gestión de contenidos multimedia y datos bibliográficos en tiempo real.
- Implementar un sistema de autenticación seguro que permita a los usuarios gestionar sus sesiones y realizar un seguimiento de su progreso.
- Diseñar e implementar múltiples modos de juego basados en diversos géneros musicales para personalizar la experiencia según el perfil del usuario.
- Desarrollar un buscador inteligente con lógica de filtrado específica para equilibrar la dificultad y evitar la ambigüedad en las respuestas.
- Garantizar la compatibilidad multiplataforma y el rendimiento óptimo de la reproducción multimedia en dispositivos móviles, con especial atención al ecosistema iOS.

La consecución de estos objetivos permitirá ofrecer una herramienta robusta y atractiva que no solo cumpla una función lúdica, sino que se convierta en un referente para la difusión cultural y tecnológica en el ámbito musical.

2. Estrategia y planificación

2.1 Estrategia de desarrollo y viabilidad

El planteamiento estratégico de **Spotydle** se ha estructurado con el fin de garantizar la transición eficiente desde una fase conceptual hasta la obtención de un producto mínimo viable (MVP) completamente funcional y estable para su entrega final.

Punto de partida: El proyecto se ha desarrollado íntegramente desde cero. Aunque se toma como referencia conceptual el modelo de juego de desafíos diarios popularizado por *Wordle*, no se ha reutilizado código fuente ni plantillas preexistentes. Todo el sistema de componentes, la lógica de validación de los intentos, el motor de pistas dinámicas y la integración de servicios de terceros han sido diseñados y programados a medida para asegurar un control absoluto sobre el software y su rendimiento.

Estrategia de desarrollo: Se ha adoptado una estrategia de desarrollo progresiva e incremental, dividida en bloques funcionales independientes. Este enfoque modular ha permitido validar y testear de forma aislada cada característica del sistema antes de integrarse en el núcleo de la aplicación, minimizando la propagación de errores de estado. El flujo de trabajo se estructuró en las siguientes fases prioritarias:

1. ***Diseño y maquetación del frontend:*** Conceptualización visual de la interfaz de usuario (UI), definición de la hoja de estilos global y creación de las vistas principales adaptables a formatos móviles mediante un diseño puramente *responsive*.
2. ***Lógica del juego y motor de pistas progresivas:*** Programación del algoritmo principal de validación de intentos y el sistema de juego. En esta fase se unificó la ingestión de datos multimedia en tiempo real a través de la API de iTunes con el desarrollo del bucle dinámico encargado del desvelado secuencial de información (audios, metadatos técnicos y componentes visuales).
3. ***Sistema de cuentas, control de sesiones y clasificaciones:*** Implementación del módulo crítico de seguridad y autenticación mediante NextAuth para gestionar el flujo de accesos. Asimismo, se integró el almacenamiento en base de datos para registrar el historial de partidas, sincronizar las estadísticas de los usuarios y construir el sistema de clasificaciones (*rankings*) competitivos.
4. ***Optimización, compatibilidad y despliegue:*** Ajuste técnico de las interfaces, resolución de problemas críticos de rendimiento y asincronía multimedia en navegadores móviles (especialmente en el ecosistema iOS) y publicación definitiva del software en el entorno de producción público.

Viabilidad del proyecto: La viabilidad del software se ha evaluado de forma rigurosa bajo tres criterios fundamentales:

- **Viabilidad Técnica:** Se ha garantizado al unificar el entorno de desarrollo bajo el framework **Next.js**. Al prescindir de una arquitectura desacoplada con un backend independiente en otro lenguaje, se ha reducido drásticamente la sobrecarga de despliegue y configuración de servidores, facilitando una comunicación directa y segura entre los *Route Handlers* de la API y la base de datos a través de un único entorno de ejecución.
- **Viabilidad Temporal:** El proyecto ha demostrado ser viable dentro del calendario disponible gracias a una estricta delimitación del alcance del MVP. Ante imprevistos de alta complejidad —como las restricciones multimedia detectadas en entornos móviles—, se priorizó la estabilidad del núcleo del juego y el comportamiento del reproductor inteligente, simplificando características secundarias que no afectaban a la experiencia principal del usuario.
- **Viabilidad de Recursos:** El requerimiento de recursos económicos ha sido nulo, haciendo el proyecto altamente viable. El desarrollo se ha apoyado exclusivamente en herramientas de código abierto, librerías de distribución libre y el consumo de una API pública y gratuita que no exige cuotas de suscripción ni pasarelas de pago para su explotación en entornos académicos.

2.2 Metodología de trabajo

El desarrollo del proyecto se ha guiado por una metodología ágil de carácter iterativo e incremental, diseñada para aportar flexibilidad y control sobre la evolución del software ante requisitos emergentes.

Enfoque y organización del trabajo

El proceso de trabajo se estructuró inicialmente mediante la elaboración de un mapa de ruta detallado que recopilaba la totalidad de los requisitos técnicos y funcionales del sistema. Esta planificación funcionó a modo de *backlog* o lista priorizada de tareas. A medida que se avanzaba en el calendario, se procedía a la ejecución, testeo y marcado progresivo de cada hito completado, asegurando que el núcleo de la aplicación fuera siempre operativo antes de añadir nuevas capas de complejidad.

Dada la naturaleza del software y tal como es habitual en los entornos de desarrollo ágiles, la planificación original no se concibió como un bloque rígido. A lo largo de las iteraciones surgieron necesidades técnicas imprevistas, descubrimientos sobre la asincronía multimedia y optimizaciones de interfaz que demandaron la incorporación de nuevas implementaciones sobre la marcha. Este enfoque adaptativo permitió integrar dichos elementos sin desestabilizar el calendario general de entrega.

Control de versiones y ubicuidad del desarrollo

El pilar fundamental sobre el que se ha sostenido toda la infraestructura del trabajo ha sido el sistema de control de versiones **Git**, gestionado de forma remota a través de un repositorio en **GitHub**. El uso de Git se ha explotado bajo criterios profesionales mediante una estrategia de ramificación (*branching*) estricta. Para cada nuevo componente, vista de la interfaz o funcionalidad del backend, se creó una rama independiente aislada de la línea principal (*main*). Una vez verificado el correcto funcionamiento del módulo mediante pruebas locales, se procedía a su fusión de forma segura, minimizando así el riesgo de regresión o pérdida de código funcional .

Adicionalmente, esta arquitectura basada en Git ha sido una herramienta crítica para resolver el reto de la deslocalización del desarrollo. Al centralizar el código en la nube, se garantizó una total ubicuidad, permitiendo alternar el trabajo diario entre el entorno de la estación local en el centro educativo y el espacio de desarrollo doméstico sin sufrir conflictos de sincronización, retrasos en la transferencia de archivos o dependencias desactualizadas. GitHub actuó de este modo tanto de plataforma de integración continua como de copia de seguridad automatizada de todo el proceso técnico.

2.3 Planificación

La estructuración temporal de **Spotydle** se dividió de manera estratégica en dos grandes bloques macroeconómicos: un primer bloque dedicado de forma exclusiva a la consultoría, estrategia y análisis conceptual, y un segundo bloque de ejecución técnica intensiva donde se desarrolló el grueso del código fuente tanto en el frontend como en el backend .

El calendario real de trabajo se distribuyó a lo largo del ciclo mediante las siguientes fases detalladas:

- **Fase 1: Conceptualización y Diseño de Estrategia (Septiembre - Octubre)**

  Esta fase inicial se centró en la definición de la idea de negocio y el alcance del producto mínimo viable (MVP) . Se estudiaron las mecánicas de juego basadas en desafíos diarios y se establecieron las bases de la lógica lúdica que definiría la plataforma, sin realizar aún implementaciones de código.

- **Fase 2: Análisis Técnico y Modelado de Requisitos (Noviembre - Diciembre)**

  Un bloque dedicado íntegramente a la ingeniería del software antes de la fase de construcción. Durante estos meses se llevó a cabo el levantamiento de los casos de uso, la especificación minuciosa de los requisitos funcionales y no funcionales, y el estudio comparativo avanzado del stack tecnológico (evaluando la viabilidad técnica de los entornos y determinando la arquitectura Full-Stack definitiva) .

- **Fase 3: Desarrollo de Software e Implementación Full-Stack (Enero - Abril)**
  Fase de ejecución técnica pura y construcción del sistema. En el mes de enero se inició formalmente el desarrollo con la maquetación del frontend, el diseño de la interfaz de usuario (*UI*) y la creación de los estilos globales *responsive* mediante Tailwind CSS. Posteriormente, de forma incremental, se integró el backend: la programación del motor del juego, el algoritmo de desvelado secuencial de pistas, el consumo asíncrono de la API de iTunes, el control de sesiones seguras mediante NextAuth y la persistencia de datos para las clasificaciones y estadísticas.

- **Fase 4: Testing, Optimización y Despliegue en Producción (Mayo)** Un *sprint* técnico final de una semana de duración enfocado en el control de calidad . Se dedicó de forma exclusiva a la ejecución de pruebas funcionales, la resolución de imprevistos críticos relacionados con las políticas de reproducción asíncrona de audio en dispositivos móviles (especialmente en el ecosistema iOS) y la publicación definitiva de la aplicación en el entorno de producción público.

Distribución temporal y carga de trabajo Para reflejar de manera precisa el peso y la duración de cada una de las fases descritas, se presenta la siguiente tabla cronológica del proyecto:

Fase del Proyecto	Período de Ejecución	Duración Estimada
Fase 1: Conceptualización y Estrategia	Septiembre - Octubre	8 semanas
Fase 2: Análisis Técnico y Requisitos	Noviembre - Diciembre	8 semanas
Fase 3: Implementación Full-Stack	Enero - Abril	15 semanas
Fase 4: Testing y Despliegue Final	Mayo	1-2 semanas

3. Análisis

3.1 Casos de uso

El sistema de **Spotydle** se define bajo la interacción de un actor principal: el usuario autenticado (o jugador), quien interactúa con la interfaz web tanto para consumir los retos multimedia como para gestionar sus flujos de navegación. A continuación, se documentan los casos de uso más representativos y críticos que modelan el comportamiento de la aplicación.

CU-01: Resolver un desafío musical

- **Actor:** Usuario registrado / Jugador.
- **Descripción:** El usuario participa en el reto de adivinanza interactiva de la jornada, reproduciendo las pistas multimedia disponibles de manera progresiva e introduciendo intentos hasta identificar la canción o agotar sus oportunidades.
- **Flujo principal:**
    - El usuario accede a la vista de juego del reto diario.
    - El sistema inicializa la interfaz en modo activo y bloquea las pistas avanzadas, activando únicamente la primera de ellas ("Audio Invertido" de 5 segundos).
    - El usuario interactúa con el reproductor de audio inteligente para escuchar el fragmento habilitado.
    - El usuario introduce el nombre del artista o de la canción en la barra de búsqueda inteligente.
    - El sistema procesa la cadena de texto, filtra posibles ambigüedades y valida el intento contra la solución almacenada.
    - Al ser un intento incorrecto, el sistema registra el fallo, actualiza el estado visual del intento y desbloquea de forma secuencial la siguiente pista disponible (Ficha técnica con año y género).
    - El usuario repite el proceso consumiendo las nuevas pistas habilitadas (audios normales de mayor duración y la portada con distorsión visual progresiva).
    - El usuario introduce la respuesta correcta antes de agotar los seis intentos disponibles.
    - El sistema detiene el juego, cambia el estado de la pista a completado, elimina el efecto de distorsión de la portada y actualiza de forma persistente las estadísticas del perfil del jugador.

- **Flujo alternativo:**
    - Si en el sexto intento el usuario introduce una respuesta errónea o vacía, el sistema cambia el estado a fallido, detiene el motor de juego, desvela la carátula e información de la canción correcta y finaliza la sesión de juego sin bonificar al usuario en la clasificación.
    - Si el usuario intenta acceder a la ruta de juego de forma directa sin haber iniciado sesión previamente, el interceptor del servidor detiene la carga multimedia y redirige automáticamente al usuario hacia la vista de inicio de sesión.
- **Resultado:** El progreso de la partida diaria queda firmemente registrado en el servidor y los marcadores globales se actualizan para el cómputo del ranking.

CU-02: Selección de modo de juego desde el menú principal

Campo	**Descripción**
Identificador	CU-02
Nombre	Selección de modo de juego
Actor	Usuario registrado / Jugador
Precondiciones	El usuario ha completado el inicio de sesión mediante NextAuth y se encuentra en la vista raíz del menú de juego (/play).

Flujo principal	1. El usuario visualiza la interfaz de bienvenida personalizada con su identificador en mayúsculas. 2. El sistema realiza una petición asíncrona interna para recuperar el catálogo de categorías musicales disponibles. 3. El usuario examina las tarjetas visuales de los niveles y selecciona un estilo musical específico para jugar. 4. El usuario pulsa sobre la tarjeta del modo seleccionado. 5. El sistema intercepta la acción, valida la disponibilidad de la ruta y redirige dinámicamente al jugador hacia la URL del submodo elegido para inicializar el nuevo tablero de juego.
Flujo alternativo	Si la tarjeta del modo seleccionado cuenta con la propiedad de bloqueo activa, el sistema anula la acción de pulsación, omite la redirección y mantiene al usuario en la vista del menú principal. Si la API interna experimenta una falla de conexión o devuelve datos corruptos, el sistema detiene la animación de las tarjetas de carga (*skeletons*) y renderiza un mensaje explícito de error técnico en la interfaz.

Postcondiciones	El sistema inicializa un entorno de juego aislado cargando una cola de canciones adaptada de forma exclusiva a los filtros correspondientes del género musical seleccionado.

3.2 Requisitos funcionales

Los requisitos funcionales detallan las acciones específicas, servicios y restricciones que **Spotydle** ofrece para satisfacer las necesidades de juego, seguridad y consulta de los usuarios . Estas especificaciones definen el comportamiento automático del software y las interacciones permitidas en la interfaz .

Para aportar una visión clara sobre el alcance de la aplicación, los requisitos se han organizado en la siguiente tabla y se les ha asignado una prioridad:

- **Prioridad Alta:** Constituyen el núcleo imprescindible del Producto Mínimo Viable (MVP) para que el juego sea completamente operativo.
- **Prioridad Media:** Funcionalidades de valor añadido que enriquecen la experiencia competitiva y de personalización.
- **Prioridad Baja:** Elementos complementarios o de refinamiento estético.

ID	Requisito Funcional	Prioridad
RF01	El sistema permitirá a los usuarios registrar una cuenta única introduciendo nombre, correo electrónico y contraseña.	Alta
RF02	El sistema permitirá iniciar y cerrar sesión manteniendo el control seguro del estado de la sesión.	Alta
RF03	El sistema generará de forma automatizada un nuevo desafío musical oculto único cada 24 horas.	Alta

RF04	El sistema habilitará un límite estricto de seis (6) intentos por jornada para resolver el reto diario.	Alta
RF05	El sistema validará en tiempo real el intento del usuario contra la solución almacenada, distinguiendo aciertos, fallos y aciertos parciales.	Alta
RF06	El sistema desvelará de forma secuencial y progresiva una nueva pista de información multimedia tras cada intento fallido.	Alta
RF07	El sistema permitirá reproducir un fragmento inicial de audio en modo invertido como primera pista del juego.	Media
RF08	El sistema mostrará los metadatos técnicos de la canción (año de lanzamiento y género musical) como segunda pista.	Media
RF09	El sistema mostrará la carátula oficial del álbum con un efecto de desenfoque.	Media
RF10	El sistema desvelará las iniciales del nombre del artista y del título de la canción únicamente en el último intento disponible.	Media
RF11	El sistema ofrecerá un buscador asíncrono con sugerencias predictivas alimentado por un catálogo musical externo en tiempo real.	Alta
RF12	El sistema normalizará las entradas del buscador para filtrar ambigüedades en las respuestas del usuario y equilibrar la dificultad.	Alta

RF13	El sistema permitirá la selección de múltiples modos de juego basados en estilos y géneros musicales específicos desde el menú principal.	Media
RF14	El sistema registrará de forma persistente el historial de partidas del usuario, incluyendo rachas de victorias, porcentaje de aciertos y estadísticas globales.	Media
RF15	El sistema mostrará un ranking global competitivo (*leaderboard*) accesible para todos los usuarios autenticados.	Media
RF16	El sistema tendrá un diseño minimalista y sencillo, otorgando al usuario una experiencia agradable y cómoda.	Baja

Los requisitos marcados con prioridad alta garantizan una plataforma completamente jugable, estable y segura. El cumplimiento integrado de toda la lista permite obtener un ecosistema de software robusto que trasciende la simple mecánica lúdica para convertirse en una aplicación web moderna y competitiva.

3.3 Requisitos no funcionales

Usabilidad

- **Interfaz intuitiva:** El diseño de la interfaz de usuario (UI) debe ser limpio y directo, permitiendo que cualquier jugador comprenda las mecánicas del juego de forma inmediata y realice sus intentos con el mínimo número de pasos, eliminando la necesidad de instrucciones complementarias o formación previa.
- **Diseño adaptativo (Responsive):** Toda la interfaz gráfica debe estar optimizada mediante estilos fluidos para garantizar una correcta visualización y adaptabilidad en pantallas táctiles de dispositivos móviles, tabletas y ordenadores de escritorio, priorizando la accesibilidad móvil.

Rendimiento

- **Optimización de carga:** La aplicación debe aprovechar las capacidades de renderizado del framework para ofrecer un tiempo de respuesta ágil en la carga inicial de la página y en las transiciones de estado del tablero de juego.
- **Fluidez multimedia:** La comunicación asíncrona con el servicio externo de música debe garantizar que los fragmentos de audio de las pistas se carguen y comiencen su reproducción en un intervalo mínimo de tiempo tras ser solicitados, evitando interrupciones en el hilo principal del cliente.
- **Latencia del buscador:** El motor de búsqueda predictiva de canciones debe procesar las entradas de texto del usuario y retornar las sugerencias del catálogo musical en tiempo real, garantizando una interacción fluida en la barra de entrada de datos.

Seguridad

- **Autenticación e integridad de sesiones:** El control de accesos e inicios de sesión debe implementarse bajo protocolos criptográficos seguros que protejan los tokens de sesión frente a interceptaciones de terceros y aseguren la identidad del usuario en cada petición.
- **Protección de datos y persistencia:** Las credenciales de acceso de los usuarios se almacenarán de forma cifrada en la base de datos mediante algoritmos hash de un solo sentido irreversibles.

Compatibilidad

- **Soporte multiplataforma y de navegadores:** La plataforma web debe asegurar su correcto funcionamiento, renderizado estético y comportamiento lógico en los navegadores web modernos más utilizados del mercado actual.
- **Compatibilidad multimedia móvil:** El sistema de reproducción multimedia y captura de eventos de audio debe incluir parches de código e implementaciones asíncronas específicas que eludan de forma segura las restrictivas políticas de reproducción automática (*autoplay*) y desbloqueo de flujos multimedia impuestas por sistemas operativos móviles, garantizando una experiencia idéntica en entornos de escritorio y dispositivos móviles.

Mantenibilidad

- **Arquitectura desacoplada y código limpio:** El código fuente de la aplicación debe estructurarse de manera modular, separando estrictamente los componentes de la interfaz de usuario de la lógica interna que gobierna el motor del juego y las llamadas a los servicios backend. Esta organización debe facilitar la depuración de errores y permitir la incorporación ágil de nuevos modos de juego, categorías o variaciones lúdicas en futuras iteraciones sin necesidad de reestructurar la lógica base del sistema.

3.4 Análisis de alternativas tecnológicas

Las decisiones tecnológicas que conforman el ecosistema de **Spotydle** no se han tomado por hábito o de forma arbitraria. Para cada componente crítico de la aplicación, se han evaluado alternativas reales del mercado bajo criterios de rendimiento, seguridad, mantenibilidad y adecuación al paradigma de desarrollo web actual .

A continuación, se detalla el análisis comparativo y la justificación técnica de las soluciones adoptadas:

Consumo de Catálogo Musical y Contenido Multimedia

- **Alternativas consideradas:** Spotify Web API frente a **iTunes Search API**.
    - **Justificación de la elección:** Inicialmente se consideró la API oficial de Spotify por su popularidad. Sin embargo, se descartó debido al severo endurecimiento de sus políticas de desarrollo y restricciones de acceso. Los flujos de autenticación OAuth de Spotify exigen que cada jugador tenga una cuenta activa o esté en listas blancas de desarrollo, introduciendo fallas críticas donde el sistema denegaba las peticiones o directamente no devolvía los datos de las canciones ni los fragmentos de audio de prueba. Se seleccionó la **API de iTunes** por ser un endpoint público y abierto que no requiere el consumo de cuotas restrictivas ni autenticación previa para consultas de catálogo. Garantiza una disponibilidad inmediata de metadatos precisos y ofrece enlaces de descarga directa para fragmentos de audio de 30 segundos, agilizando la lógica del reproductor asíncrono en cualquier dispositivo.

Framework y Arquitectura de la Aplicación

- **Alternativas consideradas:** Arquitectura desacoplada MERN (*Single Page Application* con React y un servidor independiente en Node.js con Express) frente a Arquitectura *Full-Stack* unificada mediante **Next.js**.
    - **Justificación de la elección:** Se descartó la separación en dos servidores para evitar la sobrecarga en la latencia de red, la complejidad en la configuración de políticas CORS y la duplicidad en el despliegue de infraestructuras. Se seleccionó **Next.js** debido a su capacidad para unificar el cliente y el backend dentro de un único entorno de ejecución mediante los *Route Handlers* de la API. Esto permite una comunicación directa y optimizada con la base de datos, además de aprovechar las ventajas de rendimiento que ofrece el renderizado en el servidor (*Server Components*).

Lenguaje de Programación y Tipado

- **Alternativas consideradas:** JavaScript vanilla frente a **TypeScript**.
    - **Justificación de la elección:** Aunque JavaScript ofrece una velocidad inicial de desarrollo flexible, en un proyecto con un manejo intensivo de estados (como el bucle de intentos y pistas del juego) puede derivar en errores de tipado difíciles de depurar en tiempo de ejecución. Se eligió **TypeScript** porque aporta tipado estático y detección de errores en tiempo de compilación. Esta decisión fue crucial para modelar con exactitud las estructuras de las respuestas asíncronas de la API de iTunes, garantizando la robustez del software.

Diseño y Estilos de la Interfaz

- **Alternativas consideradas:** Bootstrap frente a **Tailwind CSS**.
    - **Justificación de la elección:** Bootstrap fue evaluado por su rapidez para maquetar mediante componentes predefinidos. Sin embargo, se descartó porque genera hojas de estilo pesadas con código que no se utiliza y limita la personalización visual. Se seleccionó **Tailwind CSS** por su enfoque *utility-first*, que permite construir interfaces completamente a medida y altamente dinámicas directamente desde las propiedades de los componentes. Al compilar únicamente las clases utilizadas, optimiza la velocidad de carga en dispositivos móviles.

Infraestructura de Persistencia y Base de Datos

- **Alternativas consideradas:** Instancia tradicional de PostgreSQL / MongoDB Atlas frente a **Neon (Serverless PostgreSQL)**.
    - **Justificación de la elección:** El uso de una base de datos local tradicional se descartó debido a las complicaciones de sincronización entre el entorno del centro educativo y el espacio doméstico. Por su parte, los entornos NoSQL como MongoDB no aportaban ventajas reales para una estructura de datos rígida y relacional (usuarios, partidas, estadísticas y clasificaciones). Se eligió **Neon** por ser una solución PostgreSQL en la nube nativa para entornos *serverless*, perfectamente integrada con Next.js. Destaca por su capacidad de escalar a cero (reduciendo el consumo de recursos cuando no hay tráfico) y por permitir la creación de ramas de base de datos (*branching*) aisladas para pruebas de desarrollo.

Mapeo de Datos y Acceso a la Base de Datos

- **Alternativas consideradas:** SQL Nativo (mediante el controlador pg) frente a **Prisma ORM**.
    - **Justificación de la elección:** Escribir consultas SQL nativas ofrece un control absoluto, pero incrementa el tiempo de desarrollo y carece de tipado nativo. Se optó por **Prisma ORM** porque actúa como un cliente de base de datos totalmente securizado y autogenerado a partir de un esquema centralizado. Su integración nativa con TypeScript garantiza que cualquier consulta a la base de datos retorne objetos con tipos estrictos, acelerando la implementación de la lógica de los *rankings* y el historial de partidas.

Seguridad y Autenticación

- **Alternativas consideradas:** Sistema de autenticación manual mediante JWT autónomos frente a **NextAuth.js** integrado con **Google Cloud Console**.
- **Justificación de la elección:** Desarrollar un sistema de autenticación propio desde cero introduce riesgos críticos de seguridad en la gestión de las *cookies* de sesión. Se seleccionó la combinación de **NextAuth.js** y **Google Cloud** (OAuth 2.0). NextAuth gestiona de forma transparente el cifrado de las sesiones en el servidor, mientras que la integración con la API de Google proporciona un método de acceso seguro y rápido para el usuario, eliminando la necesidad de almacenar credenciales sensibles y reduciendo la fricción en el registro.

Micro-interacciones y Animaciones de Interfaz

- **Alternativas consideradas:** Transiciones nativas de CSS frente a la combinación de **Framer Motion** y **Canvas Confetti**.
    - **Justificación de la elección:** Las transiciones de CSS puro resultaban insuficientes para gestionar animaciones complejas basadas en estados de React (como el desvelado progresivo de pistas y efectos de desenfoque). Se seleccionó **Framer Motion** por su control declarativo sobre el ciclo de vida de los componentes, garantizando animaciones fluidas y aceleradas por hardware en dispositivos móviles. Asimismo, se integró **Canvas Confetti** para proporcionar una respuesta visual explícita e inmediata tras la resolución del reto diario, reforzando la experiencia de gamificación del producto.

Framework de Testing y Automatización de Pruebas

- **Alternativas consideradas:** Jest frente a **Vitest** en combinación con **React Testing Library**.
    - **Justificación de la elección:** Jest es el entorno de pruebas tradicional más extendido, pero su configuración con TypeScript y arquitecturas basadas en ESM modernes suele requerir compiladores externos pesados que ralentizan la ejecución. Se eligió **Vitest** debido a su integración nativa con las herramientas de construcción modernas, reutilizando la misma configuración del bundler para ejecutar los tests de forma extremadamente rápida en memoria. Junto con **React Testing Library** y un entorno **jsdom**, se ha dispuesto de una infraestructura robusta para simular el comportamiento del cliente y verificar las validaciones del motor del juego de forma automatizada.

4. Diseño

4.1 Arquitectura del sistema

La arquitectura estructural de **Spotydle** se ha diseñado bajo el paradigma de un monolito moderno unificado proporcionado por el framework **Next.js**. A diferencia de los modelos desacoplados tradicionales que dividen físicamente el cliente del servidor, esta solución centraliza ambos entornos de ejecución, optimizando los tiempos de despliegue y eliminando la latencia en la comunicación de red interna.

Conceptualmente, el sistema se organiza en cuatro capas de responsabilidad independientes, lo que permite aislar los componentes lógicos y asegurar que cada elemento cumpla una función única y específica dentro del ciclo de vida de la aplicación:

Conceptualmente, el sistema se organiza en cuatro capas de responsabilidad independientes, lo que permite aislar los componentes lógicos y asegurar que cada elemento cumpla una función única y específica dentro del ciclo de vida de la aplicación:

1. **Capa de Presentación (Frontend / Cliente):** Es el entorno con el que interactúa directamente el usuario y se ejecuta de forma íntegra en el navegador web del dispositivo cliente. Está construida mediante componentes dinámicos de React y estilizada de forma fluida con Tailwind CSS. Sus responsabilidades exclusivas son la captura de eventos del jugador (teclado, clics y búsquedas), la gestión del estado local inmediato del tablero (conteo de intentos de la partida en curso) y la ejecución de las micro-interacciones visuales y las animaciones multimedia (Framer Motion y Canvas Confetti).
2. **Capa de Lógica de Negocio y Servidor (Backend / API):** Constituida por los *Route Handlers* internos de Next.js, esta capa se ejecuta exclusivamente en el entorno del servidor. Actúa como un muro de seguridad crítico: es la encargada de procesar las validaciones de los intentos de los jugadores comparándolos con la solución de la canción diaria sin exponer jamás dicha solución al cliente (evitando así filtraciones en el código del navegador). Asimismo, gestiona los interceptores de seguridad para el control de accesos mediante NextAuth y centraliza las peticiones asíncronas de mayor peso lógico.
3. **Capa de Persistencia y Acceso a Datos:** Representa la base de almacenamiento del sistema, sustentada por el motor relacional de **Neon (Serverless PostgreSQL)** en la nube. El servidor no ataca directamente la base de datos mediante consultas textuales, sino que se apoya en **Prisma ORM** como capa de abstracción de datos. Esta subcapa se responsabiliza de garantizar la integridad y la consistencia de los datos relacionales de la aplicación, controlando de forma segura el registro de

perfiles, la actualización del historial estadístico de las partidas y la tabulación de las clasificaciones competitivas globales (*rankings*).

4. **Capa de Servicios Externos (Infraestructura de Terceros):** Integrada por la **API de iTunes**, funciona de manera asíncrona para nutrir de contenido multimedia a la plataforma. Cuando el servidor o el cliente solicitan una búsqueda o inicializan el desafío diario, este servicio externo procesa la petición y retorna un objeto de datos estructurado con los metadatos bibliográficos, la carátula oficial y el flujo de streaming del fragmento de audio de 30 segundos, evitando que Spotydle deba almacenar archivos multimedia pesados en sus propios servidores.

Flujo de la Información y Justificación del Modelo La comunicación entre componentes sigue un flujo estrictamente controlado y unidireccional. El cliente realiza peticiones asíncronas (HTTP) hacia los endpoints de la API de la aplicación; el servidor recibe la solicitud, valida la sesión del usuario a través del módulo de seguridad y decide si interactúa con la capa de datos (Prisma/Neon) para registrar estadísticas o si consulta al catálogo externo (API de iTunes) para recuperar pistas musicales. El cliente nunca posee acceso directo a la base de datos ni a los algoritmos de resolución del juego.

Esta separación de responsabilidades y la centralización Full-Stack justifican la elección arquitectónica, ya que proveen un entorno de alta seguridad para las dinámicas de juego (gamificación), facilitan el aislamiento de errores durante la fase de pruebas y aseguran que la plataforma sea escalable para la introducción de futuros modos de juego sin poner en riesgo la estabilidad del núcleo del sistema.

4.2 Modelo de datos

La persistencia de la información de **Spotydle** se sustenta sobre un modelo de datos relacional normalizado e implementado en el motor de base de datos **PostgreSQL** a través del ORM **Prisma** . La estructura ha sido diseñada con el propósito de dar soporte simultáneo tanto a los flujos automatizados de control de accesos y sesiones de seguridad corporativas (módulo de NextAuth), como a la lógica lúdica avanzada del juego, la cual incluye el almacenamiento de estadísticas complejas y un sistema de persistencia de partidas multidispositivo en tiempo real.

El ecosistema de datos está integrado por seis entidades principales estructuradas para mitigar la redundancia de datos y asegurar un rendimiento óptimo en las consultas asíncronas.

Descripción de las Entidades y Tablas

1. **User (Usuario):** Centraliza los perfiles de los jugadores. Almacena tanto los registros nativos por credenciales como las cuentas vinculadas por identificación federada (Google). Incluye propiedades de personalización visual de la interfaz como el identificador de avatar elegido.
2. **Account (Cuenta):** Administra los enlaces técnicos con los proveedores de identidad OAuth externos. Almacena de forma segura los parámetros técnicos de comunicación devueltos por servidores externos como los tokens de acceso y tiempos de expiración.
3. **Session (Sesión):** Controla el tiempo de vida y la validez de los inicios de sesión activos en los distintos dispositivos de los usuarios mediante tokens de sesión únicos y fechas de caducidad fijas.
4. **VerificationToken (Token de Verificación):** Tabla auxiliar destinada al almacenamiento temporal de cadenas firmadas con el fin de securizar flujos especiales de verificación de identidad o autenticación.
5. **GameMode (Modo de Juego):** Define el catálogo estático de listas, categorías o variantes lúdicas accesibles en el frontend (desafío diario, estilos musicales particulares, etc.). Almacena las consultas internas utilizadas para filtrar las canciones y el estado de bloqueo del nivel.
6. **ModeStat (Estadísticas del Modo):** Constituye el núcleo analítico de la aplicación. Registra de forma persistente e histórica el progreso competitivo del usuario por cada modo de juego por separado, gestionando rachas de victorias y la distribución exacta de aciertos. Además, incluye estructuras de objetos semiestructurados para recuperar el estado de una partida inacabada desde cualquier dispositivo.

Diccionario de Estructuras (Esquema Lógico)

Para comprobar la composición exacta del diseño de datos, se especifican a continuación los atributos de las tres tablas centrales del negocio lúdico mediante tipos de datos lógicos:

Tabla: User

Campo	Tipo Lógico	Restricciones	Descripción
id	Cadena (Texto)	Clave Primaria, Único	Identificador único del perfil de usuario.
name	Cadena (Texto)	Opcional	Nombre o seudónimo público para las clasificaciones.
email	Cadena (Texto)	Único, Indexado	Correo electrónico de acceso.
emailVerified	Fecha y Hora	Opcional	Fecha de validación de la cuenta de usuario.
image	Cadena (Texto)	Opcional	Enlace a la imagen de perfil de la cuenta de Google.
password	Cadena (Texto)	Opcional	Contraseña protegida mediante hash.
avatarId	Cadena (Texto)	Por defecto: "1"	Identificador del avatar personalizado del usuario.
verifyToken	Cadena (Texto)	Único, Opcional	Ficha única de verificación del perfil de usuario.

Tabla: GameMode

Campo	Tipo Lógico	Restricciones	Descripción
id	Cadena (Texto)	Clave Primaria	Identificador único del nivel o categoría musical.
title	Cadena (Texto)	-	Nombre público del modo (ej. "Urbano", "Pop").
query	Cadena (Texto)	-	Parámetro utilizado para filtrar las llamadas a la API musical.
imageUrl	Cadena (Texto)	-	Ruta de la imagen ilustrativa de la tarjeta del nivel.
isLocked	Booleano	Por defecto: false	Determina si el nivel está restringido para el público.
slug	Cadena (Texto)	Único	Cadena amigable para la construcción de rutas dinámicas.

Tabla: ModeStat

Campo	Tipo Lógico	Restricciones	Descripción
id	Cadena (Texto)	Clave Primaria	Identificador de la estadística de juego.
userId	Cadena (Texto)	Clave Foránea	Vinculación directa con el ID de la tabla User.
modeSlug	Cadena (Texto)	-	Referencia de texto directa al slug de GameMode.
gamesPlayed	Entero	Por defecto: 0	Total de desafíos iniciados en este modo de juego.
gamesWon	Entero	Por defecto: 0	Total de desafíos completados con éxito.
currentStreak	Entero	Por defecto: 0	Racha de días seguidos resolviendo el reto con éxito.
maxStreak	Entero	Por defecto: 0	Racha máxima de victorias registrada históricamente.
distribution	Matriz (Enteros)	Vector de 6 posiciones	Registro de en qué número de intento se acierta la canción.

lastCompletedDate	Cadena (Texto)	Opcional	Última fecha en la que se resolvió con éxito el reto.
lastGameState	Cadena (Texto)	Por defecto: "playing"	Estado actual de la partida activa (playing, won, lost).
lastTargetSong	Estructura JSON	Opcional	Datos completos de la canción objetivo asignada en el día.
lastGuesses	Estructura JSON	Opcional	Lista con los nombres de las canciones introducidas por el usuario.
lastGuessDetails	Estructura JSON	Opcional	Metadatos detallados de evaluación de cada intento realizado.
lastPlayedDate	Cadena (Texto)	Opcional	Última fecha en la que se realizó un intento en la aplicación.
createdAt	Fecha y Hora	Por defecto: Ahora	Registro de creación de la fila de estadísticas.
updatedAt	Fecha y Hora	Actualización automática	Fecha de la última modificación en la partida.

Relaciones del Sistema

La cohesión técnica y la consistencia del modelo de datos se consolidan a través de relaciones de integridad de tipo **Uno a Muchos (1:N)** acopladas de manera estricta mediante reglas de borrado en cascada, garantizando que si se elimina un perfil, se limpien de forma automática sus dependencias en la infraestructura de la base de datos:

- **Relación User - Account (1:N):** Un jugador dispone de una o varias credenciales de autenticación registradas en el sistema (por ejemplo, accesos con diferentes plataformas federadas), pero cada registro técnico de cuenta pertenece en exclusiva a un único usuario de la base de datos.
- **Relación User - Session (1:N):** Permite el control de la concurrencia simultánea de accesos en el software. Un usuario puede mantener abiertas múltiples sesiones válidas de forma simultánea en distintos navegadores, pero cada token de sesión está asociado indisolublemente a un único ID de usuario.
- **Relación User - ModeStat (1:N):** Un jugador posee una colección de filas de estadísticas según los diferentes modos a los que decida jugar. La clave foránea userId en la tabla ModeStat mapea de forma directa esta interacción. Cada registro analítico se une a un único usuario del sistema.
- **Relación Lógica GameMode - ModeStat (1:1):** Aunque en el motor físico no se define una clave foránea estricta para aligerar las consultas, existe una integridad referencial lógica basada en reglas de negocio entre la propiedad slug de la tabla GameMode y la propiedad modeSlug de la tabla ModeStat. Esto asegura que las estadísticas se calculen de forma aislada e independiente por cada categoría de juego disponible.

Justificación del Diseño Técnico

La selección de esta estructura de datos responde a un criterio avanzado de optimización de bases de datos para aplicaciones en entornos móviles y web. Al unificar los campos de rendimiento dentro de la misma tabla de estadísticas (ModeStat) y explotar el uso de tipos de datos complejos como vectores (Int[] para la propiedad distribution) y objetos semiestructurados (Json para propiedades como lastGuesses o lastTargetSong), se evita la necesidad de crear tablas adicionales complejas de relación secundaria (como un histórico independiente de intentos línea por línea).

Este enfoque de ingeniería permite recuperar o salvar el estado completo de una partida interrumpida con una única consulta de lectura o escritura en el servidor. Asimismo, facilita que las clasificaciones globales (*leaderboards*) competitivas se actualicen mediante sencillas funciones de ordenación sobre los campos agregados, disminuyendo la sobrecarga computacional del sistema y garantizando una escalabilidad fluida con un consumo de recursos mínimo en entornos integrados en la nube.

4.3 Diseño de interfaz

El diseño de la interfaz de usuario (*UI*) y la experiencia de uso (*UX*) de **Spotydle** se han conceptualizado bajo el principio del minimalismo funcional y el enfoque *mobile-first*. Dado que el público objetivo interactúa de forma masiva a través de dispositivos móviles y redes sociales, la interfaz prioriza la eliminación de elementos superfluos, garantizando que las dinámicas de juego sean directas, intuitivas y no requieran un proceso de aprendizaje previo por parte del jugador.

Para lograr una experiencia de uso homogénea y fluida, se ha explotado el sistema de diseño utilitario de **Tailwind CSS**, asegurando un renderizado elástico y *responsive* en cualquier resolución de pantalla. Asimismo, las transiciones de estado y el desvelado de componentes multimedia están gestionados por **Framer Motion**, aportando micro-interacciones orgánicas que mejoran la inmersión lúdica.

Descripción de las Vistas Principales

El ecosistema de pantallas de la aplicación se estructura en cuatro componentes visuales de interacción independientes:

1. Vista de Autenticación y Acceso (Landing/Login)
   - **Descripción y Acciones:** Funciona como la puerta de entrada al software. Presenta un diseño limpio con la identidad corporativa del juego y un módulo centralizado de acceso. Permite la autenticación clásica mediante credenciales de correo y contraseña, o el acceso directo y seguro en un solo clic a través del botón nativo federado de Google (OAuth 2.0).
   - **Decisiones de Diseño:** Se optó por una interfaz de contraste alto con fondos oscuros (*dark mode*) para evocar la estética habitual de las grandes plataformas de *streaming* musical, reduciendo la fatiga visual del usuario.

2. Menú Principal y Selector de Categorías (/play)
   - **Descripción y Acciones:** Una vez verificada la sesión mediante NextAuth, esta pantalla da la bienvenida personalizada al usuario (renderizando su alias técnico) y despliega el catálogo dinámico de modos de juego almacenados en la base de datos PostgreSQL. El usuario puede visualizar las opciones en forma de tarjetas interactivas (*cards*).
   - **Decisiones de Diseño:** Durante la carga asíncrona de los datos desde el servidor, se implementan componentes de sustitución (*skeletons*) con animaciones de pulsación atenuada. Esto mejora sustancialmente la percepción de la velocidad del sitio web,

evitando la sensación de congelamiento de la pantalla mientras se recuperan las imágenes de Neon.

3. Tablero del Desafío Lúdico (Vista del Juego)

- **Descripción y Acciones:** Es el núcleo funcional del proyecto. Su arquitectura visual se organiza verticalmente para facilitar el juego a una sola mano en pantallas móviles. Contiene los siguientes elementos interactivos ordenados de manera estricta:
    - *Indicador de Estado:* Un marcador superior que monitoriza de forma visual el número de intentos consumidos sobre el límite de seis oportunidades.
    - *Contenedor de la Portada:* Muestra la carátula oficial de la canción provista por la API de iTunes, sujeta a un filtro de desenfoque (*blur*) criptográfico controlado por el estado del juego. El desenfoque disminuye de forma matemática tras cada fallo.
    - *Reproductor Inteligente:* Un panel de control multimedia simplificado (botones de reproducción, pausa y barra de progreso temporal) que ejecuta los fragmentos de streaming de audio.
    - *Buscador Asíncrono Predictivo:* Una barra de entrada de texto inteligente. A medida que el usuario escribe, la interfaz despliega una persiana flotante con sugerencias predictivas filtradas en tiempo real (mostrando carátula, título y artista), lo que evita errores de ortografía al introducir el intento.
- **Decisiones de Diseño:** Para mantener la tensión y la dinámica lúdica, las pistas avanzadas (como las iniciales del nombre de la canción) permanecen completamente ocultas, emergiendo de forma animada solo en la parte inferior del tablero cuando el jugador se encuentra en su última oportunidad.

4. Panel de Clasificaciones y Estadísticas Históricas (Leaderboard)

- **Descripción y Acciones:** Accesible desde la barra de navegación superior. Presenta un diseño tabular sobrio enfocado exclusivamente en la legibilidad de los datos de rendimiento analítico. Muestra la posición del usuario en el ranking global competitivo respecto a la comunidad, detallando el número total de partidas disputadas, victorias acumuladas y la racha de días consecutivos en activo (*current streak*). Asimismo, dibuja un gráfico de barras horizontal que representa la distribución exacta de los intentos en los que se ha resuelto el misterio.
- **Decisiones de Diseño:** Los datos se estructuran mediante cuadrículas atenuadas, destacando de forma visual con colores vivos o bordes iluminados la fila correspondiente al usuario autenticado para facilitar una autoevaluación competitiva inmediata.

Flujo de Navegación del Sistema

La navegación dentro de **Spotydle** se ha estructurado siguiendo un mapa híbrido que combina recorridos lineales y bucles cíclicos cerrados, eliminando callejones sin salida en la interfaz para garantizar la retención del jugador:

1. **Flujo de Acceso:** El usuario inicia de forma obligatoria en la pantalla de autenticación. Si el interceptor de NextAuth valida el token, se produce una redirección lineal automática hacia el menú principal.
2. **Flujo Lúdico:** Desde el menú de juego, el usuario selecciona libremente una categoría o consulta sus marcadores globales. Al pulsar una tarjeta funcional, la URL cambia de forma dinámica incorporando el parámetro slug del nivel (ej. /play/urbano), inicializando el tablero de juego de forma aislada.
3. **Flujo de Cierre y Retorno:** El jugador interactúa de manera cíclica dentro del tablero de juego introduciendo sus intentos. Una vez que el sistema dictamina la victoria o la derrota, el reproductor y el buscador se bloquean, y emerge una ventana modal flotante animada con los resultados globales y el disparador de partículas de felicitación (*Canvas Confetti*) si corresponde. Esta vista final proporciona botones claros de acción directa para regresar al menú de selección de modos o compartir el resultado de forma textual en redes sociales, cerrando el ciclo de interacción diaria de manera limpia y fluida.

5. Desarrollo

5.1 Estructura del proyecto

Este apartado describe de manera detallada la organización interna de **Spotydle** a nivel de archivos y carpetas. El objetivo de esta disposición es permitir que cualquier desarrollador externo que acceda al repositorio o al código fuente de la aplicación pueda orientarse con rapidez y comprender con precisión la lógica de responsabilidades de cada módulo.

De acuerdo con los estándares de diseño de software modernos, la aplicación se ha estructurado bajo la arquitectura nativa del framework **Next.js** empleando el sistema de enrutamiento **App Router**, lo que permite centralizar el cliente y el servidor en un mismo espacio de trabajo manteniendo una estricta separación de responsabilidades funcionales.

A continuación, se presenta la representación visual del árbol de directorios del proyecto:

```
/spotydle-oficial
├── /prisma
│   ├── /migrations
│   └── schema.prisma
├── /public
└── /src
    ├── /app
    │   ├── /api (auth, modes, ranking, register, search, stats, user)
    │   ├── /play
    │   │   └── /[slug]
    │   ├── /profile
    │   └── /ranking
    ├── /components (auth, game, layout, ui)
    ├── /context
    ├── /hooks
    ├── /lib
└── /services
```

Descripción de los Módulos y Carpetas Principales

- **/prisma**: Aloja la infraestructura de persistencia relacional configurada para el entorno de producción en Neon. Contiene el histórico de versiones de la base de datos (/migrations) y el archivo de definición del modelo relacional schema.prisma.
- **/src/app**: Directorio centralizador del enrutamiento del software. Next.js mapea las carpetas como URLs del navegador. Dentro de este se distinguen tres grandes bloques de trabajo:
    - **/api (Route Handlers)**: Constituye el backend puro de la aplicación, encargado de procesar peticiones HTTP asíncronas de manera segura en el servidor. Se subdivide en controladores especializados para la gestión de sesiones con NextAuth (/auth), consulta de niveles (/modes), tabulación de la clasificación (/ranking), alta de usuarios nativos (/register), pasarela de búsquedas hacia el catálogo externo (/search), salvaguarda del estado de partida de los jugadores (/stats/update) y actualización de perfiles privados (/user).
    - **/play/[slug]**: Ruta dinámica basada en parámetros variables. Permite que la interfaz cargue un único tablero de juego común pero inyecte componentes, estilos y filtros de canciones específicos dependiendo del *slug* del modo seleccionado por el usuario en el menú principal (ej. Pop, Urbano, Rock).
    - **Vistas de Usuario e Interceptores**: Carpetas como /profile, /ranking, /verify y /pending-verification controlan las interfaces de interacción secundaria del jugador y el flujo seguro de validación de cuentas por correo electrónico. El archivo sandbox actúa como entorno de pruebas aislado durante el desarrollo técnico.
- **/src/components**: Almacena los elementos de interfaz de usuario desacoplados por su naturaleza funcional y lógica:
    - **/auth**: Formularios y botones específicos para el inicio de sesión y registro.
    - **/game**: Componentes críticos del bucle lúdico (reproductor inteligente, visualizador de pistas, animaciones de intentos).
    - **/layout**: Elementos estructurales compartidos como el menú de navegación global (*navbar*) y el pie de página (*footer*).
    - **/ui**: Biblioteca de diseño atómico reutilizable (botones genéricos, cuadros de entrada de texto personalizados, contenedores base).
- **Carpetas del Núcleo Lógico e Integración**:
    - **/context**: Contiene los proveedores de estado de la React Context API, permitiendo compartir datos dinámicos globales (como el estado de la sesión del usuario o variables del juego) entre componentes distantes sin sufrir acoplamiento.
    - **/hooks**: Centraliza los *custom hooks* personalizados, aislando la lógica de efectos y estados de React de la capa puramente visual.

- **/lib**: Aloja configuraciones e inicializaciones estáticas del lado del servidor, como la instancia del cliente de base de datos Prisma Client.
- **/services**: Capa de abstracción encargada de encapsular las conexiones y el mapeo de datos con la API de iTunes, evitando que los controladores dependan de código multimedia directo.
- **/test**: Almacena los scripts y configuraciones dedicados a la automatización de pruebas de software bajo el entorno de Vitest y React Testing Library.

Esta sólida organización arquitectónica garantiza que el software cumpla con los principios de alta cohesión y bajo acoplamiento, permitiendo que la lógica de negocio del servidor, los estados de sesión y los componentes interactivos del cliente puedan escalar o depurarse de forma totalmente independiente.

5.2 Implementación de funcionalidades

Este apartado constituye el núcleo técnico de la memoria . En él se describe, bloque a bloque, la ingeniería de software aplicada para construir las características fundamentales de **Spotydle**, detallando qué hace cada módulo, cómo se ha resuelto conceptualmente su arquitectura y cuáles han sido las decisiones tecnológicas críticas adoptadas para garantizar un producto funcional y robusto .

Todas las funcionalidades descritas a continuación han quedado completamente implementadas en la versión definitiva del sistema .

5.2.1 Gestión de usuarios, seguridad y control de accesos

- **Descripción funcional:** Este módulo gestiona el ciclo completo de identidad de los jugadores . Permite la creación de cuentas nativas, la autenticación segura y persistente en el sitio web y la integración del inicio de sesión rápido mediante cuentas de terceros, protegiendo las rutas de juego frente a accesos no autorizados .
- **Implementación conceptual:** La funcionalidad se ha desarrollado integrando la librería **NextAuth.js** acoplada directamente a la base de datos PostgreSQL mediante el paquete @auth/prisma-adapter. El flujo técnico se divide en dos vías de acceso autónomas:
    - *Acceso Federado (OAuth 2.0):* Delegado en los servidores de Google a través de una aplicación configurada en *Google Cloud Console*. Al autenticarse, NextAuth captura el perfil, descarga la imagen y da de alta automáticamente las filas correspondientes en las tablas User y Account.
    - *Acceso Nativo:* Gestionado por un controlador de API personalizado ubicado en src/app/api/register. Cuando un usuario introduce sus datos, el servidor

intercepta la petición, verifica mediante Prisma que el correo electrónico no esté duplicado y cifra la contraseña utilizando la librería bcryptjs con un factor de salado seguro (*salt*) antes de guardarla en la base de datos .

- **Decisiones técnicas y justificación:** Se decidió implementar un adaptador de base de datos automatizado (@auth/prisma-adapter) junto con NextAuth en lugar de desarrollar un sistema manual basado en *JSON Web Tokens* (JWT) locales para mitigar vulnerabilidades críticas de seguridad . Esta arquitectura gestiona de forma transparente el ciclo de vida de las *cookies* de sesión cifradas en el servidor y delega el almacenamiento de credenciales sensibles en grandes proveedores (Google), reduciendo el riesgo de brechas de datos dentro del sistema.

5.2.2 Motor de juego y consumo de catálogo multimedia asíncrono

- **Descripción funcional:** Gobierna la mecánica lúdica principal de la plataforma . Es el encargado de inicializar el reto de la jornada, realizar llamadas al catálogo de música externo y controlar el bucle interactivo de seis intentos, liberando pistas dinámicas tras cada fallo del jugador .
- **Implementación conceptual:** El motor de juego se ejecuta dentro de la ruta dinámica parametrizada src/app/play/[slug] . Al cargar la interfaz, un servicio interno (src/services) realiza una llamada asíncrona hacia la **API de iTunes** utilizando la consulta de búsqueda (query) almacenada en la tabla GameMode del nivel seleccionado. El backend procesa la respuesta, selecciona la canción objetivo y envía al cliente un objeto filtrado. El frontend captura esta información y gestiona las pistas en base al estado de intentos fallidos del usuario:
    1. *Pista 1:* Activa el reproductor de audio inteligente para escuchar un fragmento de 5 segundos con un efecto de inversión de onda.
    2. *Pistas 2 y 3:* Libera metadatos técnicos (año de lanzamiento y género) y amplía la duración del audio normal de forma secuencial.
    3. *Pistas 4 y 5:* Reduce de forma matemática la propiedad de desenfoque (blur) aplicada mediante Tailwind CSS sobre el componente de la carátula oficial.
    4. *Pista 6:* Desvela las iniciales del artista y el título mediante un filtrado de caracteres por expresión regular.
- El cuadro de entrada cuenta con un buscador predictivo asíncrono implementado en /api/search que consulta a iTunes en tiempo real según lo que escribe el usuario, mostrando opciones válidas para evitar errores de ortografía.
- **Decisiones técnicas y justificación:** La sustitución de la API de Spotify por la de **iTunes** fue la decisión de diseño más determinante del motor de juego. Las severas restricciones de acceso impuestas por Spotify (que exigen tokens individuales autenticados o listas blancas de desarrollo) provocaban fallas constantes donde la API denegaba el servicio o no retornaba los flujos de audio de prueba. iTunes Search API proveyó un endpoint público, abierto y libre de cuotas que garantiza la entrega

de metadatos precisos y enlaces de *streaming* directo de 30 segundos, asegurando la viabilidad total de la mecánica del juego.

5.2.3 Persistencia de estado de partida multidispositivo

- **Descripción funcional:** Esta funcionalidad garantiza la continuidad de la experiencia lúdica del usuario. Permite que un jugador inicie su desafío diario desde una estación local en el centro educativo y lo retome o finalice en su dispositivo móvil doméstico exactamente en el mismo intento, sin perder su progreso ni alterar sus estadísticas históricas.
- **Implementación conceptual:** A diferencia de las aplicaciones web convencionales que guardan el estado del juego en el almacenamiento local del navegador (*LocalStorage*), Spotydle sincroniza cada intento directamente en la nube. La tabla ModeStat se diseñó específicamente incorporando campos de estructura semiestructurada nativos de PostgreSQL (Json). Cada vez que el jugador introduce una respuesta en la interfaz, el sistema ejecuta una petición HTTP POST hacia el controlador interno src/app/api/stats/update. Este endpoint recibe la información y realiza una operación atómica de actualización o inserción (*upsert*) mediante Prisma, guardando el estado actual (lastGameState), la canción objetivo (lastTargetSong) y la lista de nombres de canciones introducidas (lastGuesses). Cuando el usuario inicia sesión desde cualquier otro dispositivo y entra en el nivel, el servidor lee estos campos JSON y recompone el tablero de juego de manera idéntica.
- **Decisiones técnicas y justificación:** Se descartó el almacenamiento relacional clásico (crear una tabla secundaria de intentos vinculada 1:N) debido al alto coste computacional y de rendimiento que implicaba realizar operaciones de unión de tablas (*joins*) repetitivas en cada renderizado. La explotación de los tipos de datos Json de PostgreSQL y Prisma ORM permitió empaquetar todo el estado dinámico del tablero en un único registro plano, facilitando una velocidad de lectura y escritura instantánea en entornos *serverless* en la nube.

5.2.4 Módulo de estadísticas, rachas y clasificaciones competitivas

- **Descripción funcional:** Se responsabiliza de computar el rendimiento analítico del usuario y de construir la mecánica de gamificación competitiva de la plataforma . Registra el volumen de victorias, calcula las rachas de días consecutivos jugando con éxito y genera la clasificación global (*leaderboard*).
- **Implementación conceptual:** La lógica se procesa de forma automatizada en el servidor a través del Route Handler /api/ranking. Cuando el juego detecta un acierto final, el backend incrementa las propiedades gamesPlayed y gamesWon dentro de la tabla ModeStat. El algoritmo de control de rachas compara la fecha actual con la propiedad string lastCompletedDate; si el desafío se resuelve al día consecutivo siguiente, el campo currentStreak se incrementa en una unidad y evalúa si ha superado a maxStreak.

La métrica de la gráfica de rendimiento se gestiona mediante una matriz de enteros (Int[]) en el campo distribution, donde cada posición del vector (de 0 a 5) funciona como un acumulador que se incrementa según el intento exacto en el que se adivinó la canción. Para construir el ranking global, el endpoint realiza una consulta agregada ordenando los perfiles de los usuarios de mayor a menor según su racha máxima y volumen de victorias.

- **Decisiones técnicas y justificación:** Almacenar la distribución de aciertos mediante un vector nativo de enteros (Int[]) en lugar de registros individuales evitó la sobrecarga en la base de datos Neon. Esta solución técnica permite al frontend dibujar los gráficos de barras horizontales de manera inmediata al leer un array directo de seis dígitos, optimizando los tiempos de respuesta del servidor y disminuyendo drásticamente el consumo de recursos de computación.

5.3 Pruebas

Las pruebas constituyen el pilar fundamental para certificar el correcto funcionamiento de **Spotydle**, garantizando que cada una de las características implementadas se comporte según las especificaciones de diseño establecidas en la fase de análisis. Para validar el sistema de forma exhaustiva, el plan de pruebas se dividió en dos grandes fases: una fase de verificación técnica automatizada y una fase de control de calidad (*QA*) con usuarios reales en entornos de producción.

5.3.1 Pruebas Unitarias y de Integración Automatizadas

Aprovechando el entorno de ejecución provisto por **Vitest** y **React Testing Library**, se diseñó una batería de pruebas automatizadas destinadas a blindar los bloques lógicos independientes del software antes de su integración masiva :

- **Validación del motor lúdico:** Se escribieron tests unitarios para verificar el comportamiento determinista del contador de intentos (asegurando que se detenga estrictamente al llegar al sexto fallo) y para evaluar el algoritmo de desvelado progresivo de pistas.
- **Normalización de respuestas:** Se verificó que las funciones de filtrado de texto procesen correctamente las entradas de los usuarios, comprobando casos normales y límites (como la eliminación automática de tildes, mayúsculas, espacios huérfanos o caracteres especiales en los nombres de los artistas).
- **Endpoints de la API:** Mediante simulaciones de entorno de red (*mocking*), se testearon los *Route Handlers* de /api/stats/update y /api/ranking para asegurar que las peticiones HTTP malformadas o sin token de sesión válido de NextAuth fueran rechazadas por el servidor con los códigos de estado correspondientes.

5.3.2 Pruebas Funcionales y de Aceptación (Beta Testing)

Para evaluar la experiencia de uso (*UX*), la jugabilidad y la persistencia de datos en condiciones reales, se distribuyó una versión preliminar del producto mínimo viable (MVP) a un grupo controlado de usuarios externos que actuaron como *beta testers*. El seguimiento de este proceso se estructuró bajo un protocolo formal de control de calidad, evaluando el comportamiento del juego en diferentes escenarios:

- **Pruebas de concurrencia y persistencia:** Se coordinó a varios usuarios para que ejecutaran desafíos de forma simultánea. Se verificó que la base de datos distribuida en Neon registrara correctamente las estadísticas individuales sin generar bloqueos en el servidor y que el flujo de sincronización multidispositivo restaurara los tableros de manera idéntica al alternar entre ordenadores locales y terminales móviles.
- **Flujos alternativos y de error:** Los evaluadores forzaron de manera voluntaria casos límite en la interfaz, tales como intentar enviar el formulario con la barra de búsqueda vacía, pulsar repetidamente el botón de reproducción del audio inteligente o interrumpir la conexión a internet a mitad de una partida. El sistema demostró ser altamente tolerante a fallos, bloqueando acciones inválidas y mostrando mensajes explícitos en pantalla sin romper el estado interno de React.

5.3.3 Pruebas de Interfaz y Compatibilidad (Resultados Obtenidos)

La aplicación fue testeada de forma multiplataforma en los motores de renderizado más extendidos del mercado: Blink (Google Chrome, Microsoft Edge), Gecko (Mozilla Firefox) y WebKit (Safari de Apple).

Estas pruebas multiplataforma arrojaron el resultado técnico más crítico del proyecto: mientras que en entornos de escritorio la reproducción de audio asíncrono funcionaba de manera impecable, **en dispositivos móviles basados en iOS (Safari y Chrome para móvil) las pistas musicales se bloqueaban sistemáticamente**, impidiendo jugar el primer intento.

El análisis del error determinó que las restrictivas políticas de seguridad de Apple impiden la ejecución automática (*autoplay*) de flujos multimedia que provienen de llamadas asíncronas externas (como la API de iTunes) si no nacen de una interacción humana directa. El problema se resolvió con éxito rediseñando el ciclo de vida del reproductor: se programó un disparador (*user gesture trigger*) que fuerza al usuario a pulsar un botón explícito de inicio en la interfaz móvil, desbloqueando el contexto de audio del navegador nativo antes de realizar la petición asíncrona. Tras este parche técnico, el software demostró una compatibilidad del 100% en todos los sistemas operativos móviles testeados.

6. Conclusiones

6.1 Conclusiones generales

El resultado final de este proyecto va mucho más allá del desarrollo de líneas de código; se traduce en haber sido capaz de construir, de manera completamente autónoma, una aplicación *Full-Stack* totalmente funcional, sólida y con un alto valor lúdico. **Spotydle** se ha consolidado como una plataforma que destaca por su sencillez de uso y por ofrecer una experiencia sumamente entretenida y atractiva para el público general, capturando la esencia de la gamificación moderna en un entorno web pulido . El producto obtenido no se percibe como un simple ejercicio académico, sino como un software real con una firme proyección de futuro, diseñado con una arquitectura lo suficientemente limpia como para ser lanzado al mercado y continuar evolucionando con nuevas mejoras en el corto plazo .

A nivel personal y profesional, el verdadero valor de este camino no ha residido en el dominio memorístico de la sintaxis, sino en la capacidad de adoptar y asimilar un ecosistema tecnológico completamente nuevo en un espacio de tiempo limitado. Dar el salto a entornos avanzados como Next.js, TypeScript, Prisma y bases de datos en la nube con Neon ha supuesto un reto de autoaprendizaje extraordinario. Este proceso me ha demostrado que cuento con las competencias necesarias para adaptarme con flexibilidad a las demandas y stacks cambiantes de la industria del desarrollo web actual.

Sin duda, el aprendizaje más enriquecedor ha sido desarrollar la resiliencia técnica necesaria para enfrentarme a los imprevistos de forma analítica. En un proyecto de software real, las dificultades y las carencias son inevitables; la clave del éxito ha estado en la destreza para detectarlas a tiempo, aislar su origen y diseñar soluciones viables . Un claro ejemplo de esto fue diagnosticar los bloqueos multimedia en dispositivos móviles durante la fase de pruebas y resolverlos mediante un rediseño del ciclo de vida del reproductor asíncrono. Del mismo modo, asumir con honestidad que ciertas características secundarias debían simplificarse en favor de blindar la estabilidad del motor de juego diario, refleja una capacidad de gestión de proyectos muy madura.

En definitiva, las conclusiones que extraigo son plenamente satisfactorias. He comprobado que soy capaz de idear un producto conceptual, planificar su desarrollo, superar las barreras técnicas del despliegue y materializar una aplicación atractiva que funciona de manera impecable. Cierro este ciclo con la seguridad y la confianza indispensables para incorporarme con garantías al sector profesional del desarrollo web.

6.2 Consecución de objetivos

Este apartado contrasta los resultados reales obtenidos al finalizar el desarrollo con las metas iniciales fijadas en la fase de planificación. Evaluar de forma crítica el grado de cumplimiento de estos objetivos permite medir el éxito del proyecto, justificar las soluciones de ingeniería adoptadas ante los imprevistos técnicos y certificar la madurez del producto mínimo viable (MVP) obtenido.

A continuación, se presenta la matriz analítica que evalúa la consecución del objetivo general y de cada una de las metas específicas definidas para **Spotydle**:

Evaluación de la Consecución de Metas

Objetivo Inicial Planteado	**Estado**	**Análisis y Justificación Técnica**
Objetivo General: Desarrollar una aplicación web funcional de retos musicales diarios con un sistema progresivo de pistas y mecánicas dinámicas que incentive la cultura musical de forma accesible.	**Cumplido**	Se ha materializado un software interactivo *Full-Stack* robusto bajo Next.js. El bucle de juego es ágil, no requiere fricción de aprendizaje y ofrece un producto altamente atractivo y escalable para la difusión lúdica y cultural.
OE 1: Implementar un sistema de reto diario automatizado que presente una composición musical única cada 24 horas.	**Cumplido**	Resuelto con éxito mediante lógica en el servidor. El sistema utiliza marcas de tiempo acopladas a la base de datos para refrescar la canción de la jornada de manera automatizada, bloqueando o liberando los intentos según el día.
OE 2: Desarrollar un motor de pistas dinámico que incluya fragmentos de audio, metadatos técnicos (año y género),	**Cumplido**	El motor lúdico en /play/[slug] libera las pistas secuencialmente tras cada fallo: fragmento de audio comprimido de iTunes, año de lanzamiento, género analítico, reducción de la propiedad blur

elementos visuales procesados y pistas de texto.		en la carátula y desvelado de iniciales mediante expresiones regulares.
OE 3: Integrar la API de iTunes para la recuperación y gestión de contenidos multimedia y datos bibliográficos en tiempo real.	**Cumplido**	Tras el descarte estratégico de Spotify por sus cuotas restrictivas, el módulo de servicios conecta de forma transparente con iTunes API, recuperando metadatos limpios y transmisiones de audio streaming de 30 segundos sin coste de latencia.
OE 4: Implementar un sistema de autenticación seguro que permita a los usuarios gestionar sus sesiones y realizar un seguimiento de su progreso.	**Cumplido**	Integrado de forma sólida mediante NextAuth.js (Google OAuth y registro cifrado con bcryptjs). El progreso histórico y las rachas competitivas se salvan en la nube a través de objetos planos Json en la tabla ModeStat.
OE 5: Diseñar e implementar múltiples modos de juego basados en diversos géneros musicales para personalizar la experiencia según el perfil del usuario.	**Cumplido**	La arquitectura basada en la ruta dinámica parametrizada src/app/play/[slug] lee dinámicamente los registros de la tabla GameMode (Pop, Urbano, Rock), aislando la experiencia lúdica según las preferencias del jugador.
OE 6: Desarrollar un buscador inteligente con lógica de filtrado específica para equilibrar la dificultad y evitar la ambigüedad en las respuestas.	**Cumplido**	Implementado en el controlador /api/search. Realiza consultas predictivas asíncronas hacia el catálogo y aplica algoritmos de normalización de cadenas (eliminando tildes, mayúsculas y caracteres huérfanos) para evitar fallos ortográficos injustos.

OE 7: Garantizar la compatibilidad multiplataforma y el rendimiento óptimo de la reproducción multimedia en dispositivos móviles, con especial atención al ecosistema iOS.	**Cumplido**	Se logró el funcionamiento al 100% en todos los navegadores móviles del mercado. El bloqueo nativo de reproducción automática (*autoplay*) impuesto por Apple en iOS se solucionó mediante un disparador de interacción humana (*user gesture trigger*).

Balance General del Proyecto

El análisis comparativo determina que **el proyecto ha alcanzado un grado de cumplimiento del 100% respecto a todas las metas técnicas y funcionales propuestas inicialmente**.

No se han registrado desviaciones ni recortes en el alcance del producto mínimo viable. Aquellos desafíos metodológicos que pusieron en riesgo el éxito de los objetivos —como las limitaciones de acceso en la API de Spotify o las estrictas directivas de seguridad multimedia de los navegadores en el ecosistema iOS— no provocaron la eliminación de ninguna característica, sino que impulsaron el diseño de soluciones de ingeniería alternativas y eficaces (la adopción del catálogo público de iTunes y la reestructuración del ciclo de vida del reproductor mediante eventos reactivos).

El software final no solo constituye una herramienta interactiva robusta, estable y de alto rendimiento que cumple con los estándares exigidos para un perfil profesional en Desarrollo de Aplicaciones Web, sino que ha blindado su núcleo arquitectónico para consolidarse como un producto comercializable y atractivo, preparado para absorber nuevas ampliaciones funcionales en el futuro.

6.3 Valoración de la metodología y planificación

Afrontar el desarrollo integral de una aplicación *Full-Stack* de forma completamente individual ha supuesto un desafío organizativo de primer nivel. Al no contar con un equipo de trabajo en el que delegar responsabilidades, el éxito del proyecto dependía por completo de mi capacidad para autogestionarme, actuar como diseñador, programador, analista de datos y técnico de control de calidad de manera simultánea.

Haciendo un balance crítico de cómo ha funcionado la forma de trabajo, puedo afirmar que la metodología adoptada y la planificación temporal han sido eficaces y determinantes para la entrega exitosa de **Spotydle**.

Eficacia de la organización y gestión ágil

Para suplir la falta de un equipo físico, decidí aplicar los principios de las metodologías ágiles adaptados al desarrollo en solitario para fragmentar el proyecto en tareas muy pequeñas y acotadas. Esta organización me permitió mantener un foco absoluto en el Producto Mínimo Viable (MVP).

La estrategia de priorizar el núcleo lúdico (la conexión con la API de iTunes y el motor de intentos) antes de construir las interfaces secundarias demostró ser un acierto absoluto. Gracias a este enfoque, cada bloque técnico se asentaba sobre una base ya probada, minimizando la aparición de errores colaterales y optimizando el tiempo invertido en picar código.

Evaluación del calendario y el "esprint" final

La planificación temporal fijada en el cronograma inicial sirvió como una excelente brújula. El calendario estaba bien dimensionado para las fases de análisis, modelado de datos y desarrollo base. Sin embargo, como suele ser habitual en los proyectos de ingeniería de software, la realidad del despliegue y el testeo introdujo variables imprevistas.

El descubrimiento del bug de reproducción automática en dispositivos iOS y la necesidad de implementar parches de seguridad para NextAuth consumieron más jornadas de las

previstas en el calendario teórico. Esto provocó que tuviera que intensificar el ritmo de trabajo y "apretar" notablemente en la fase final del proyecto para completar la suite de pruebas automatizadas y la redacción de la documentación técnica.

A pesar de este incremento en la intensidad del trabajo durante las últimas semanas, en ningún momento me vi superado por los calendarios ni sufrí desajustes caóticos en las fechas de entrega. Haber dejado un margen de maniobra prudencial en la planificación inicial me permitió absorber estos picos de esfuerzo sin poner en riesgo la calidad final del software.

Conclusión de la experiencia

La valoración global de la metodología de trabajo es extraordinariamente positiva y gratificante. Trabajar solo me ha obligado a salir constantemente de mi zona de confort y a asumir la responsabilidad total de cada fallo y de cada acierto. Ver que la planificación diseñada sobre el papel ha sido capaz de guiarme con éxito a través de problemas técnicos complejos, hasta materializar un juego interactivo que funciona a la perfección y que gusta al público, genera una profunda satisfacción personal. Esta experiencia no solo ha consolidado mis competencias técnicas, sino que ha reforzado de manera definitiva mi confianza para gestionar proyectos de software autónomos en el sector profesional.

6.4 Visión de futuro

Aunque **Spotydle** se entrega como una aplicación completamente funcional, madura y dotada de una gran calidad estética y lúdica —permitiendo desde el primer momento que los usuarios jueguen, compitan y disfruten de una experiencia web altamente atractiva—, el estado actual del proyecto debe entenderse como un **MVP (Producto Mínimo Viable)** robusto. La plataforma ha cumplido con rigor todos los objetivos planteados, pero su potencial de crecimiento apenas ha comenzado a explotarse.

La visión de futuro para Spotydle es clara y ambiciosa: mantener una evolución constante del software para transformarlo en una plataforma de entretenimiento masiva. Gracias a que el desarrollo se ha guiado bajo estrictos estándares de **código limpio, modularidad y separación de responsabilidades por componentes**, la aplicación goza de una deuda técnica nula. Esto significa que el sistema es altamente escalable, permitiendo inyectar nuevas características funcionales sin riesgo de corromper el núcleo del motor lúdico ya consolidado.

Las líneas de mejora y expansión planificadas para el corto y medio plazo se estructuran en los siguientes ejes de desarrollo:

- **Sistema de puntuación ponderada (Ranking por puntos):** Actualmente, la clasificación global computa de manera lineal el volumen de victorias y las rachas de días seguidos. La mejora más inmediata e innovadora será evolucionar el *leaderboard* hacia un sistema de puntuación dinámica que premie la agudeza musical dependiendo del intento exacto en el que se acierte la canción.

  Nota de escalabilidad: Implementar este cambio es sumamente sencillo debido al excelente diseño del modelo de datos actual. Como la tabla ModeStat ya almacena de forma nativa la distribución exacta de los aciertos de cada jugador mediante un vector de enteros (Int[]) y guarda el histórico de intentos en formato Json, no se requiere reestructurar la base de datos física en Neon. Solo es necesario definir el baremo de puntuación en el servidor (por ejemplo: 100 puntos si se acierta en el primer intento, 10 puntos en el sexto) y actualizar la lógica de agregación del controlador /api/ranking.

- **Expansión del catálogo y personalización:** Se proyecta la apertura de nuevos modos de juego dinámicos basados en décadas específicas (80s, 90s, 2000s), subgéneros de nicho y desafíos monográficos dedicados en exclusiva a la trayectoria de grandes artistas internacionales. Asimismo, se profundizará en la personalización estética del perfil del usuario, ampliando el sistema de avatarId hacia una tienda de cosméticos visuales desbloqueables mediante las puntuaciones obtenidas en el juego.
- **Optimización avanzada y despliegue multiplataforma:** Se realizarán tareas de optimización de carga y compresión en la pasarela de servicios de iTunes para reducir la latencia multimedia en redes móviles de baja velocidad. De igual forma, aprovechando la naturaleza reactiva de los componentes actuales, se plantea la viabilidad de empaquetar el código mediante tecnologías como *Capacitor* o *React Native* para lanzar Spotydle como una aplicación móvil nativa en las tiendas oficiales de Android (Google Play) e iOS (App Store).

Lanzamiento comercial y monetización

Soy plenamente consciente del **impacto viral** que este tipo de dinámicas de juego diario posee en las redes sociales modernas (fenómeno demostrado por referentes de la industria como *Wordle* o *Heardle*). Por ello, el destino definitivo de Spotydle no es quedarse en un repositorio local como un mero trámite académico, sino ser lanzado al gran público.

La intención estratégica es posicionar la web en el mercado digital y, una vez consolidada una comunidad activa de usuarios diarios recurrentes, introducir un modelo de negocio sostenible. Al tratarse de un juego que se ejecuta de forma óptima en dispositivos móviles, se integrarán redes de anuncios publicitarios (*ad-networks*) no intrusivos en las transiciones de las pantallas de resultados. Este enfoque comercial permitirá rentabilizar el tráfico masivo generado por la viralidad de los retos compartidos, transformando este proyecto final de ciclo en un producto digital lucrativo, competitivo y con un largo recorrido empresarial por delante.

7. Glosario

- **API (Application Programming Interface):** Interfaz de programación de aplicaciones. Conjunto de definiciones y protocolos que permite que dos aplicaciones informáticas se comuniquen e intercambien datos entre sí de forma estandarizada.
- **CORS (Cross-Origin Resource Sharing):** Intercambio de recursos de origen cruzado. Mecanismo de seguridad implementado por los navegadores web que restringe las peticiones HTTP que una aplicación realiza hacia un dominio o servidor diferente del que originó la página.
- **Framer Motion:** Librería de animación para React enfocada en la creación de transiciones y micro-interacciones visuales fluidas y declarativas, optimizadas mediante la aceleración por hardware del dispositivo cliente.
- **Full-Stack:** Enfoque de desarrollo de software que abarca tanto la capa del cliente o interfaz de usuario (*frontend*) como la capa lógica del servidor y gestión de bases de datos (*backend*).
- **Hash Criptográfico:** Algoritmo matemático de un solo sentido que transforma cualquier bloque de datos de entrada en una cadena de caracteres única de longitud fija. Se utiliza en seguridad para almacenar contraseñas de forma irreversible, impidiendo conocer el texto original.
- **iTunes Search API:** Servicio web público y abierto provisto por Apple que permite realizar consultas en tiempo real dentro del catálogo de iTunes para recuperar metadatos bibliográficos, carátulas y fragmentos de audio streaming de 30 segundos de duración.
- **JSON (JavaScript Object Notation):** Formato de intercambio de datos ligero, estructurado en formato de texto plano mediante pares de clave-valor, ampliamente utilizado en la comunicación entre servidores web y clientes debido a su facilidad de lectura y escritura.
- **JWT (JSON Web Token):** Estándar abierto estructurado en formato JSON que se utiliza para transmitir información de identidad firmada digitalmente entre las partes, empleado habitualmente para la autenticación manual de sesiones.

- **MVP (Minimum Viable Product):** Producto Mínimo Viable. Versión inicial de un software que cuenta con las características fundamentales para ser operativo y jugable, diseñada con el propósito de validar las mecánicas de negocio con usuarios reales y guiar futuras iteraciones.
- **Neon:** Plataforma de base de datos relacional PostgreSQL en la nube basada en una arquitectura *serverless*, optimizada para entornos modernos debido a su capacidad de separar el almacenamiento del cómputo y escalar sus recursos a cero cuando no registra tráfico.
- **Next.js:** Framework de desarrollo web construido sobre React que unifica el cliente y el servidor bajo un mismo entorno de trabajo, facilitando el renderizado en el servidor (*Server Components*) y la creación de arquitecturas de enrutamiento dinámico.
- **NextAuth.js:** Librería de autenticación modular de código abierto diseñada de forma específica para Next.js, encargada de la gestión automatizada de sesiones seguras, el control de *cookies* cifradas en el servidor y la integración con proveedores de identidad externos.
- **OAuth 2.0:** Protocolo estándar de la industria para la autorización delegada que permite a una aplicación web acceder a recursos compartidos de un perfil de usuario (como las credenciales públicas de Google) de forma segura y sin necesidad de conocer la contraseña original.
- **ORM (Object-Relational Mapping):** Mapeador objeto-relacional. Herramienta de software que actúa como pasarela de abstracción entre el código de programación y una base de datos relacional, permitiendo interactuar con las tablas y ejecutar consultas utilizando la sintaxis del lenguaje nativo sin necesidad de escribir SQL puro.
- **PostgreSQL:** Sistema de gestión de bases de datos relacionales de código abierto de nivel corporativo, caracterizado por su robustez, su estricta integridad de datos y su soporte nativo para tipos avanzados de datos como arrays y objetos JSON estructurados.
- **Prisma ORM:** Mapeador objeto-relacional de última generación para entornos Node.js y TypeScript que genera de forma automatizada un cliente de base de datos con tipado estático estricto a partir de un esquema de datos centralizado.

- **QA (Quality Assurance):** Control o aseguramiento de la calidad. Conjunto de metodologías, flujos de trabajo y protocolos aplicados durante el ciclo de vida del software para verificar que el producto final cumpla con los estándares técnicos y funcionales establecidos en la fase de análisis.
- **React Testing Library:** Biblioteca de utilidades para la automatización de pruebas de interfaz de usuario diseñada para testear componentes de React de manera realista, simulando el comportamiento exacto que experimentaría un usuario final dentro del navegador web.
- **Responsive Design:** Diseño adaptativo. Filosofía de maquetación web que, mediante el empleo de estructuras fluidas y reglas de estilos elásticas, asegura que la interfaz gráfica de una aplicación se visualice y funcione correctamente en cualquier resolución y tipo de pantalla (móviles, tabletas u ordenadores).
- **Route Handlers:** Controladores de rutas internos de Next.js que permiten construir endpoints de API personalizados dentro del sistema de archivos del servidor, procesando peticiones HTTP asíncronas (GET, POST, etc.) bajo un entorno seguro.
- **Serverless Architecture:** Arquitectura sin servidor. Modelo de computación en la nube donde el desarrollador se desentiende de la gestión, mantenimiento físico y configuración de los servidores, delegando en el proveedor de infraestructura el escalado automático de los recursos bajo demanda según el tráfico del sitio web.
- **Tailwind CSS:** Framework de diseño CSS orientado a clases utilitarias (*utility-first*) que permite construir interfaces de usuario completamente a medida y dinámicas aplicando estilos directamente sobre las etiquetas de los componentes de React, optimizando el peso final del archivo de estilos.
- **TypeScript:** Superconjunto de programación desarrollado por Microsoft que añade tipado estático opcional y detección automatizada de errores en tiempo de compilación sobre el lenguaje JavaScript estándar, mejorando la robustez y la mantenibilidad de proyectos complejos.
- **UI / UX (User Interface / User Experience):** Diseño de Interfaz de Usuario y Experiencia de Usuario. Disciplinas del diseño de software enfocadas en la creación del entorno gráfico interactivo (estética, controles y tipografías) y en la optimización de la usabilidad, fluidez y nivel de satisfacción del jugador al interactuar con el sistema.

- **Vitest:** Framework o entorno de ejecución de pruebas automatizadas nativo de alta velocidad, diseñado específicamente para integrarse con las herramientas de construcción modernas de JavaScript para ejecutar colecciones de tests unitarios directamente en memoria.

8. Bibliografía

Documentación Oficial de Tecnologías y Frameworks

- Apple Developer. (2026). *iTunes Search API: Multimedia Retrieval Resources Reference*. Apple Developer Documentation Archive. https://developer.apple.com/library/archive/documentation/AudioVideo/Conceptual/iTuneSearchAPI/
- Microsoft Corporation. (2025). *TypeScript Language Specification and Documentation*. TypeScript Project. https://www.typescriptlang.org/docs/
- Neon Labs, Inc. (2026). *Neon Serverless Postgres: Architecture and Integration Guide*. Neon Docs. https://neon.tech/docs
- NextAuth.js. (2025). *Authentication and Security for Next.js Applications (v4)*. NextAuth Documentation. https://next-auth.js.org
- Prisma Data, Inc. (2026). *Prisma ORM Reference and Database Schema Guide*. Prisma Docs. https://www.prisma.io/docs
- React Integration Team. (2025). *React 19 Core Documentation and Server Components Specification*. React Dev. https://react.dev
- Tailwind Labs, Inc. (2025). *Tailwind CSS Utility-First Framework: Production Optimization and Purging*. Tailwind CSS Docs. https://tailwindcss.com/docs
- Vercel, Inc. (2026). *Next.js 16 Web Framework: App Router Architecture and Route Handlers*. Next.js Documentation. https://nextjs.org/docs

Librerías de Terceros y Control de Calidad

- Framer Motion Devs. (2025). *Framer Motion: Production-ready animations for React*. Motion Documentation. https://motion.dev/
- Novick, B. (2024). *Canvas Confetti: Performance and Canvas Animation Elements*. GitHub Repository. https://github.com/catdad/canvas-confetti
- Testing Library Tools. (2025). *React Testing Library: Dom Testing Utilities and Best Practices*. Framework Testing Documentation. https://testing-library.com/docs/react-testing-library/intro/

- Vitest Core Team. (2026). *Vitest: Blazing Fast Unit Test Framework Native to Vite and Next.js*. Vitest Dev Guide. https://vitest.dev/guide/

Seguridad e Integridad de Datos

- Auth0 & Okta. (2025). *OAuth 2.0 Authorization Framework: Core Specification and Federated Identity*. OAuth Community Standards. https://oauth.net/2/
- OWASP Foundation. (2025). *OWASP Top 10 Application Security Risks: Mitigating SQL Injection and Session Hijacking*. Open Web Application Security Project. https://owasp.org/www-project-top-ten/

Herramientas de Inteligencia Artificial Generativa

- Google Authored AI. (2026). *Gemini 3 Flash Model: Architectural assistance and technical documentation drafting for web applications*. Google AI Research. https://deepmind.google/technologies/gemini/

9. Anexos

Anexo A: Guía de despliegue técnico y variables de entorno

Dado que el proyecto se ha diseñado bajo una arquitectura de "código limpio" y modular, la replicación o despliegue del sistema en un nuevo entorno (ya sea local o de producción en la nube) no requiere de manuales gráficos complejos, sino de la correcta configuración de su entorno técnico basado en archivos de texto estructurado.

Para garantizar la puesta en marcha del Producto Mínimo Viable (MVP) sin exponer credenciales privadas ni claves criptográficas sensibles, el sistema se apoya en un archivo centralizado de variables de entorno.

1. Plantilla de configuración de entorno (.env.example)

Cualquier administrador o desarrollador que decida desplegar **Spotydle** deberá crear un archivo físico de texto plano denominado .env en la raíz del proyecto, tomando como referencia la siguiente estructura de variables técnicas esenciales:

```
# PERSISTENCIA Y BASE DE DATOS (NEON POSTGRESQL)

DATABASE_URL="postgresql://usuario:contraseña@servidor-neon.tech/spotydle?sslmode=require"

# AUTENTICACIÓN Y SEGURIDAD (NEXTAUTH)

NEXTAUTH_URL="http://localhost:3000"

NEXTAUTH_SECRET="cadena_aleatoria_de_cifrado_criptografico_jwt"

# PROVEEDOR DE IDENTIDAD (GOOGLE CLOUD CONSOLE OAUTH)

GOOGLE_CLIENT_ID="identificador_de_cliente_proveedor_google.apps.googleusercontent.com"

GOOGLE_CLIENT_SECRET="clave_secreta_del_cliente_proveedor_google"

# CONFIGURACIÓN DE SERVICIOS AUXILIARES

NEXT_PUBLIC_APP_URL="http://localhost:3000"
```

2. Secuencia de comandos para la inicialización del sistema

Una vez configurado el archivo de texto con las credenciales correspondientes, el despliegue e inicio de la aplicación en un servidor de desarrollo local o de integración continua se ejecuta de manera secuencial a través de la interfaz de comandos de la terminal, siguiendo de forma estricta los scripts nativos declarados en el archivo de configuración del proyecto:

1. **Instalación de dependencias del ecosistema:** Descarga e integra de forma automática los paquetes críticos del backend y frontend (Next.js, Tailwind CSS, NextAuth, Axios).

   npm install

2. **Generación automática del cliente de datos:** El script automatizado lee el archivo schema.prisma y autogenera el mapa de tipos estáticos de TypeScript mapeados para interactuar de forma segura con Neon.

   npx prisma generate

3. **Verificación de la compilación y tipado:** Ejecuta una comprobación completa del código fuente, asegurando que TypeScript no detecte errores de tipado y que la estructura del framework esté lista para producción.

   npm run build

4. **Lanzamiento del servidor en entorno local:** Inicializa el entorno local de Next.js para desarrollo (por defecto, en el puerto 3000), activando el refresco en caliente (*Fast Refresh*) para testear la aplicación y aplicar modificaciones en tiempo real.

   npm run dev

Nota sobre el uso de inteligencia artificial

En cumplimiento de las directrices éticas y metodológicas establecidas en la guía del proyecto final de ciclo, se declara de forma transparente que se ha utilizado la herramienta de inteligencia artificial generativa Gemini (Google) como asistente de soporte y co-redactor en la elaboración de esta memoria técnica.

El uso de esta herramienta se ha gestionado bajo un criterio de responsabilidad activa y supervisión constante, acotando su intervención a las siguientes funciones específicas:

- Estructuración y maquetación de contenidos: Optimización de la jerarquía del documento y adaptación de los bloques informáticos al índice oficial exigido por el centro.
- Refinamiento lingüístico y estilo: Traducción de las ideas abstractas, la lógica de la programación y el bucle de juego interactivo hacia una prosa técnica, formal y académica en tercera persona.
- Soporte en documentación complementaria: Elaboración del glosario técnico de términos de desarrollo web y formateo estandarizado de las referencias bibliográficas bajo la norma APA 7.ª edición.

Declaración de autoría y asunción de responsabilidad

Es fundamental recalcar que el 100% de la arquitectura del software, la configuración del entorno de Next.js, el modelado de datos en Prisma, la integración de la API de iTunes y la resolución de todos los bugs críticos del sistema (como el control del bloqueo multimedia en dispositivos móviles iOS) son fruto del trabajo técnico, manual y real del alumno. La herramienta de IA no ha generado la lógica lúdica del juego ni ha sustituido la toma de decisiones críticas de ingeniería en el código.

En consecuencia, el autor de este proyecto certifica que todo el contenido de esta memoria ha sido minuciosamente revisado, comprendido y asumido en su totalidad, garantizando que el texto refleja con absoluta honestidad el comportamiento real del software desarrollado y los aprendizajes consolidados durante el proceso.

Licencia