Ángel Arcade: minijuegos web

Proyecto final de grado superior de DAW 24/25

Icono

El contenido generado por IA puede ser incorrecto.

Autor: Ángel Barrón Martínez

Tutora proyecto: Sara Berezo Loza

Tabla de contenido

[Introducción 2](#_Toc200012557)

[Idea del proyecto 2](#_Toc200012558)

[Propuesta detallada del proyecto 3](#_Toc200012559)

[Características principales 3](#_Toc200012560)

[Justificación técnica 3](#_Toc200012561)

[Objetivos 4](#_Toc200012562)

[Objetivo general 4](#_Toc200012563)

[Objetivos específicos 4](#_Toc200012564)

[Objetivos opcionales 5](#_Toc200012565)

[Beneficios obtenidos 5](#_Toc200012566)

[Análisis de proyectos existentes y aspectos innovadores de Ángel Arcade 6](#_Toc200012567)

[Requisitos del Proyecto 7](#_Toc200012568)

[Requisitos funcionales 7](#_Toc200012569)

[Requisitos no funcionales 8](#_Toc200012570)

[Diseño técnico 10](#_Toc200012571)

[Tecnologías utilizadas 10](#_Toc200012572)

[Diseño de la base de datos 11](#_Toc200012573)

[Codificación 14](#_Toc200012574)

[Cliente (Frontend) 14](#_Toc200012575)

[App.js 14](#_Toc200012576)

[Componente LoginRegister.jsx 16](#_Toc200012577)

[Componente GameStatsMenu.jsx 17](#_Toc200012578)

[Servidor (Backend) 18](#_Toc200012579)

[Introducción 18](#_Toc200012580)

[Estructura del servidor 18](#_Toc200012581)

[Rutas y controladores 19](#_Toc200012582)

[Base de datos y modelos 20](#_Toc200012583)

[Conexión con el frontend 21](#_Toc200012584)

[Pruebas 21](#_Toc200012585)

[Objetivo de las pruebas 21](#_Toc200012586)

[Tipos de pruebas realizadas 21](#_Toc200012587)

[Pruebas funcionales de los minijuegos 21](#_Toc200012588)

[Pruebas del backend y la API REST 25](#_Toc200012589)

[Pruebas de persistencia de datos 26](#_Toc200012590)

[Pruebas de autenticación 26](#_Toc200012591)

[Pruebas de la interfaz 28](#_Toc200012592)

[Resultados 28](#_Toc200012593)

[Posibles mejoras futuras 29](#_Toc200012594)

[Mejoras en los minijuegos 29](#_Toc200012595)

[Mejoras en el sistema de estadísticas 29](#_Toc200012596)

[Mejora del sistema de logros 29](#_Toc200012597)

[Experiencia de usuario (UX/UI) 29](#_Toc200012598)

[Backend y base de datos 29](#_Toc200012599)

[Infraestructura y despliegue 29](#_Toc200012600)

[Bibliografía 30](#_Toc200012601)

# Enlaces

Repositorio en GitHub con el código, documentación y videos  
<https://github.com/manael4/AngelArcade.git>

# Introducción

El proyecto, titulado *Ángel Arcade: Minijuegos Web*, se enmarca dentro del módulo de Proyecto del Ciclo Formativo de Grado Superior en Desarrollo de Aplicaciones Web, impartido en el IES Comercio de Logroño. El objetivo principal es aplicar los conocimientos adquiridos durante el ciclo en el desarrollo de una aplicación web funcional, original y que represente un reto técnico.

En los últimos años, la creación de aplicaciones interactivas en el navegador ha cobrado gran importancia, gracias al desarrollo de tecnologías como HTML5, CSS3 y frameworks modernos como React. Esta tendencia ha permitido recuperar y modernizar ideas que en su momento dependían de tecnologías hoy obsoletas, como Adobe Flash. El proyecto surge con esta intención: recuperar el concepto de plataforma de minijuegos accesibles desde el navegador, pero con una base tecnológica actual.

## Idea del proyecto

La idea de desarrollar *Ángel Arcade: Minijuegos Web* surge de una motivación personal arraigada desde la infancia. Desde pequeño, siempre me han llamado la atención las páginas web de videojuegos, especialmente aquellas que ofrecían minijuegos online accesibles desde el navegador. Plataformas como Minijuegos.com, o Friv marcaron una etapa de entretenimiento y curiosidad que, con el tiempo, se transformó en interés por ver cómo estaban hechas.

Con la desaparición del soporte para **Adobe Flash** en 2021, muchas de estas plataformas vieron cómo una gran parte de sus juegos dejaban de funcionar. Este hecho provocó la pérdida de cientos de juegos y experiencias interactivas que marcaron a toda una generación. Ante esta situación, decidí emprender este proyecto con la idea de **recuperar el espíritu de aquellas webs**, aprovechando las tecnologías actuales como **React** para crear una plataforma moderna, funcional y fácilmente ampliable.

Mi intención es rendir homenaje a esas páginas y, al mismo tiempo, demostrar los conocimientos adquiridos durante el ciclo formativo desarrollando un producto web atractivo, interactivo y técnicamente completo.

## Propuesta detallada del proyecto

El proyecto Ángel Arcade: Minijuegos Web ha consistido en el desarrollo de una plataforma web interactiva que agrupa varios minijuegos accesibles desde cualquier navegador moderno. La idea principal ha sido crear un espacio entretenido, funcional y atractivo que permita a los usuarios disfrutar de juegos sencillos, pero bien diseñados, con una experiencia homogénea y personalizable.

### Características principales

* **Catálogo selecto de minijuegos propios**   
  Desarrollar varios minijuegos utilizando React partiendo solo de la idea, con mecánicas clásicas y accesibles, adaptadas para jugar desde PC o dispositivos móviles.
* **Interfaz visual atractiva y responsive**  
  Utilizando HTML5 y CSS3, diseñar una página que se adapta a diferentes tamaños de pantalla, con un diseño claro y moderno que facilita la navegación entre juegos y secciones.
* **Sistema de gestión de usuarios y sesiones**  
  Mediante localStorage y una API backend desarrollada con Express y Node.js, que permita que los usuarios puedan registrarse, iniciar sesión y mantener su progreso y estadísticas guardadas temporalmente.
* **Registro y visualización de estadísticas**  
  Cada juego incluye un sistema para almacenar datos como partidas jugadas, victorias, tiempos o puntuaciones, que los usuarios pueden consultar para motivar la mejora continua.
* **Arquitectura modular y escalable**  
  La estructura del proyecto es diseñada para que sea sencillo añadir nuevos juegos o funcionalidades en el futuro sin necesidad de reescribir grandes partes del código.

### Justificación técnica

* **Uso de React**  
  Permite crear interfaces dinámicas y reactivas, facilitando la reutilización de componentes y la gestión eficiente del estado de la aplicación.
* **Backend con Express y Node.js**  
  Una solución ligera y escalable para gestionar la API de estadísticas y usuarios, con buena integración con el frontend.
* **Almacenamiento local**  
  LocalStorage empleado para mantener la sesión activa y guardar datos momentáneos, reduciendo la dependencia del servidor y mejorando la experiencia offline parcial.

## Objetivos

El proyecto *Ángel Arcade: Minijuegos Web* tiene como finalidad principal el diseño y desarrollo de una plataforma web que permita a los usuarios acceder a una colección de minijuegos desarrollados con tecnologías actuales. Para lograr este fin, se establecen los siguientes objetivos generales y específicos:

### Objetivo general

* Desarrollar una aplicación web funcional y atractiva que sirva como plataforma de minijuegos accesibles desde el navegador, utilizando el framework React y aplicando los conocimientos adquiridos durante el ciclo formativo.

### Objetivos específicos

* Diseñar una estructura modular y escalable que permita añadir nuevos juegos fácilmente sin alterar la base del proyecto.
* Implementar distintos minijuegos con mecánicas variadas (ej. Solitario, Buscaminas, Memory, Tower Defense), cada uno como componente independiente en React.
* Gestionar las estadísticas de los usuarios (tiempos, victorias, puntuaciones) mediante una API REST sencilla y persistente.
* Aplicar buenas prácticas de programación: separación de componentes, reutilización de código, uso de hooks personalizados, etc.
* Garantizar que la aplicación sea visualmente clara, intuitiva y adaptable a distintos tamaños de pantalla.
* Almacenar la sesión del usuario y sus progresos mediante almacenamiento local (localStorage) y comunicación con la base de datos.
* Realizar pruebas funcionales de cada componente para asegurar el correcto funcionamiento de los juegos y de la plataforma en general.
* Evaluar posibles ampliaciones y mejoras, como el registro de usuarios con autenticación real, clasificación global de puntuaciones o inclusión de nuevos juegos.

### Objetivos opcionales

* Integrar una **banda sonora o efectos sonoros** básicos en los juegos para mejorar la experiencia del usuario.
* Añadir un sistema de **logros o recompensas** que motive la rejugabilidad.
* Implementar **modo oscuro y claro** con cambio dinámico de tema.
* Añadir un sistema de **mensajes o notificaciones** (por ejemplo, avisos de victoria, errores o cambios en la sesión).

## Beneficios obtenidos

* Plataforma educativa y de entretenimiento con buena usabilidad y diseño moderno.
* Mejor comprensión y dominio de tecnologías web actuales.
* Aprendizaje sobre creación y uso de APIs y otras tecnologías o frameworks como React y Express.
* Posibilidad de evolución futura con nuevas funcionalidades y juegos.
* Proyecto con un valor añadido al preservar y renovar la tradición de las páginas de minijuegos web adaptadas a la era post-Flash.

## Análisis de proyectos existentes y aspectos innovadores de Ángel Arcade

En el ámbito de las páginas web dedicadas a minijuegos existen múltiples plataformas populares, como Minijuegos.com, Kongregate, Armor Games o CrazyGames, que ofrecen una gran variedad de juegos accesibles desde el navegador. Sin embargo, muchos de estos sitios presentan ciertas limitaciones o enfoques distintos que hacen que Ángel Arcade aporte valor diferencial y novedades importantes:

* **Diversidad y control de juegos**  
  Las plataformas existentes suelen incluir un catálogo muy amplio, a menudo con juegos desarrollados por terceros, lo que puede generar una experiencia inconsistente en cuanto a calidad, diseño y jugabilidad. En cambio, Ángel Arcade se centra en ofrecer un conjunto selecto de minijuegos desarrollados por el propio autor, garantizando una experiencia uniforme, cuidada y optimizada.
* **Tecnología moderna y accesibilidad**  
  Mientras que muchos juegos clásicos en la web usaban tecnologías ya obsoletas como Adobe Flash (que dejó de ser soportado en 2020), Ángel Arcade se construye íntegramente con tecnologías actuales como React para el frontend y Node.js con Express para el backend, asegurando compatibilidad con navegadores modernos y facilidad de mantenimiento y ampliación.
* **Integración de estadísticas y perfil de usuario**  
  A diferencia de muchos sitios que solo ofrecen jugar, Ángel Arcade incorpora un sistema personalizado de estadísticas y progreso para cada usuario, almacenado a través de una API propia y gestionado con sesiones locales. Esto añade un componente motivacional y competitivo que mejora la experiencia del jugador.
* **Proyecto personal y educativo**  
  Al ser un proyecto desarrollado como Trabajo de Fin de Grado, Ángel Arcade destaca por la intención de aprendizaje y mejora continua, lo que permite iterar rápidamente y adaptar el proyecto según feedback y nuevas ideas, algo que los grandes portales comerciales no pueden hacer tan ágilmente.
* **Potencial para ampliaciones personalizadas**  
  El enfoque modular y basado en React facilita la incorporación futura de nuevos minijuegos, así como funcionalidades adicionales como ranking global, modos multijugador o integración con redes sociales, lo que abre un amplio abanico de posibilidades para innovar y evolucionar el proyecto más allá de su versión inicial.

En resumen, Ángel Arcade no solo ofrece una alternativa moderna y cuidada a las páginas clásicas de minijuegos, sino que además aporta un enfoque personal y didáctico, con tecnologías actuales, integración de estadísticas y una arquitectura preparada para crecer, lo que le otorga un claro valor añadido y potencial de innovación en este sector.

# Requisitos del Proyecto

Para el correcto desarrollo del proyecto *Ángel Arcade: Minijuegos Web*, se han identificado y definido los requisitos funcionales y no funcionales necesarios para garantizar una plataforma robusta, accesible y técnicamente completa. Estos requisitos aseguran que el producto final cumpla con los objetivos planteados, mantenga la calidad técnica y proporcione una experiencia satisfactoria para el usuario.

## Requisitos funcionales

Los requisitos funcionales definen las características y comportamientos que la aplicación debe implementar. Se dividen en varios bloques relacionados con el funcionamiento de los juegos, la interacción del usuario y la gestión de datos:

* 1. Gestión de usuarios:
  + 1.1. El sistema debe permitir el registro de nuevos usuarios.
  + 1.2. El sistema debe permitir el inicio de sesión con credenciales válidas.
  + 1.3. El usuario debe permanecer autenticado mientras dure la sesión mediante almacenamiento local.
* 2. Interfaz de navegación:
  + 2.1. El sistema debe ofrecer una interfaz visual clara y adaptativa que permita acceder a los diferentes juegos desde el menú principal.
  + 2.2. Cada juego debe ser accesible desde un enlace o botón específico.
* 3. Minijuegos:
  + 3.1. La aplicación debe incluir una colección de al menos 8 minijuegos desarrollados con React.
  + 3.2. Cada juego debe ser completamente funcional, con sus reglas internas implementadas correctamente.
  + 3.3. Deben contemplarse distintos géneros de juego (lógicos, estrategia, memoria, etc.).
* 4. Sistema de estadísticas:
  + 4.1. El sistema debe registrar estadísticas individuales por usuario para cada juego (partidas jugadas, victorias, tiempo, puntuación, etc.).
  + 4.2. Las estadísticas deben visualizarse desde una sección específica dentro de la plataforma.
  + 4.3. Los datos deben mantenerse actualizados tras cada partida y almacenarse mediante una API REST.
* 5. Gestión de logros y progresos (opcional):
  + RF5.1. El sistema podrá incluir un sistema de logros para motivar la rejugabilidad.
  + RF5.2. Los logros desbloqueados se reflejarán en el perfil del usuario.
* 6. Persistencia de datos:
  + RF6.1. Los datos de sesión y estadísticas deben almacenarse utilizando localStorage y complementarse con llamadas a una API backend desarrollada con Express.

## Requisitos no funcionales

Estos requisitos aseguran la calidad, escalabilidad, rendimiento y accesibilidad del sistema:

* 1. Tecnología y arquitectura:
  + 1.1. El frontend debe estar desarrollado con React, empleando una arquitectura modular basada en componentes reutilizables.
  + 1.2. El backend debe estar implementado con Node.js y Express, ofreciendo endpoints para la gestión de estadísticas y usuarios.
  + 1.3. La comunicación entre frontend y backend debe realizarse mediante peticiones HTTP a través de fetch o librerías equivalentes.
* 2. Usabilidad y diseño:
  + 2.1. La interfaz debe ser responsive, permitiendo su uso desde ordenadores y dispositivos móviles.
  + 2.2. El diseño debe seguir principios de claridad visual, accesibilidad y navegación intuitiva.
* 3. Escalabilidad:
  + 3.1. La estructura del proyecto debe permitir la incorporación de nuevos juegos sin necesidad de modificar la base del sistema.
  + 3.2. Los componentes y funciones deben ser lo suficientemente desacoplados para permitir refactorizaciones y mejoras futuras.
* 4. Rendimiento:
  + 4.1. Los juegos deben cargarse rápidamente y responder fluidamente a la interacción del usuario.
  + 4.2. La gestión de estado y renderizado debe ser eficiente, especialmente en dispositivos con recursos limitados.
* 5. Seguridad básica:
  + 5.1. Los datos de usuario no deben exponerse públicamente ni enviarse sin control a la API.
  + 5.2. Las rutas del backend deben manejar errores y validar los datos recibidos.
* 6. Mantenibilidad y control de versiones:
  + 6.1. El proyecto debe estar gestionado mediante un sistema de control de versiones (Git) y alojado en un repositorio en línea (GitHub).
  + 6.2. Debe existir una documentación mínima del código y comentarios explicativos en las partes más relevantes.

# Diseño técnico

En el desarrollo de Ángel Arcade: Minijuegos Web, se optó por un stack tecnológico moderno y ampliamente adoptado en la industria, que permitiera construir una aplicación modular, escalable y de fácil mantenimiento. Este apartado describe las tecnologías utilizadas, la arquitectura funcional del sistema y la organización general del proyecto tanto en el cliente como en el servidor.

## Tecnologías utilizadas

La selección del software se centró en herramientas de código abierto, multiplataforma y bien documentadas, que facilitan el desarrollo ágil y eficiente:

* React: Biblioteca JavaScript para construir interfaces de usuario mediante componentes reutilizables. Su uso del Virtual DOM permite un rendimiento óptimo en la renderización dinámica de los distintos minijuegos y componentes de la aplicación.
* HTML5 y CSS3: Estándares fundamentales para la estructura y presentación del contenido web. HTML5 aporta semántica y accesibilidad, mientras que CSS3 se emplea para estilos visuales modernos, diseño responsive y animaciones.
* Node.js y Express: Node.js proporciona un entorno de ejecución JavaScript en servidor, y Express es el framework minimalista utilizado para crear una API REST que gestiona la comunicación entre frontend y base de datos.
* SQLite y SQLiteStudio: SQLite se utiliza como sistema de gestión de base de datos ligero y eficiente, almacenando de manera persistente la información de usuarios y estadísticas. SQLiteStudio facilita la administración y visualización de la base de datos durante el desarrollo.
* npm (Node Package Manager): Gestor de paquetes para instalar y mantener dependencias del proyecto, desde librerías auxiliares hasta herramientas de desarrollo.
* CORS (Cross-Origin Resource Sharing): Middleware configurado en el backend para permitir solicitudes del frontend, especialmente útil en entornos de desarrollo con orígenes diferentes.
* localStorage: API del navegador utilizada para guardar temporalmente datos como la sesión del usuario y configuraciones de juego, mejorando la experiencia al evitar consultas constantes al servidor.
* Git y GitHub: Herramientas para control de versiones y hospedaje remoto, esenciales para la gestión ordenada del código, ramas de desarrollo y respaldo continuo del proyecto.

### Diseño de la base de datos

Para almacenar la información de los usuarios y las estadísticas de los distintos minijuegos, se ha diseñado una base de datos relacional basada en SQLite. La base de datos está compuesta por una tabla principal de usuarios y varias tablas de estadísticas específicas para cada juego.

#### Tabla users

Esta tabla contiene los datos básicos de los usuarios registrados en el sistema.

| **Campo** | **Tipo** | **Descripción** |
| --- | --- | --- |
| **id** | **INTEGER PK AUTOINCREMENT** | **Identificador único del usuario.** |
| **username** | **TEXT UNIQUE** | **Nombre único del usuario.** |
| **password** | **TEXT NOT NULL** | **Contraseña del usuario (hashed).** |

#### Tablas de estadísticas

Cada juego dispone de su propia tabla de estadísticas, vinculada a la tabla users mediante el campo id que actúa como clave foránea. Esto permite mantener separadas las estadísticas por juego y almacenar diferentes métricas según las particularidades de cada uno.

| **Tabla** | **Descripción** |
| --- | --- |
| **stats\_2048** | **Guarda el mejor puntaje alcanzado en el juego 2048.** |
| **stats\_buscaminas** | **Almacena estadísticas desglosadas por dificultad: partidas jugadas, victorias y mejores tiempos para fácil, normal y difícil.** |
| **stats\_chess** | **Registra el número de victorias jugando con blancas y negras.** |
| **stats\_memory** | **Contabiliza partidas jugadas, mejor puntuación y pares descubiertos en el juego Memory.** |
| **stats\_snake** | **Registra la puntuación más alta y partidas jugadas en Snake.** |
| **stats\_solitario** | **Guarda partidas jugadas, victorias y mejor tiempo en Solitario.** |
| **stats\_tictactoe** | **Lleva cuenta de victorias con 'O', victorias con 'X' y empates.** |
| **stats\_towerdefense** | **(No implementado aún) Contendrá partidas jugadas, victorias y oleada alcanzada en Tower Defense.** |

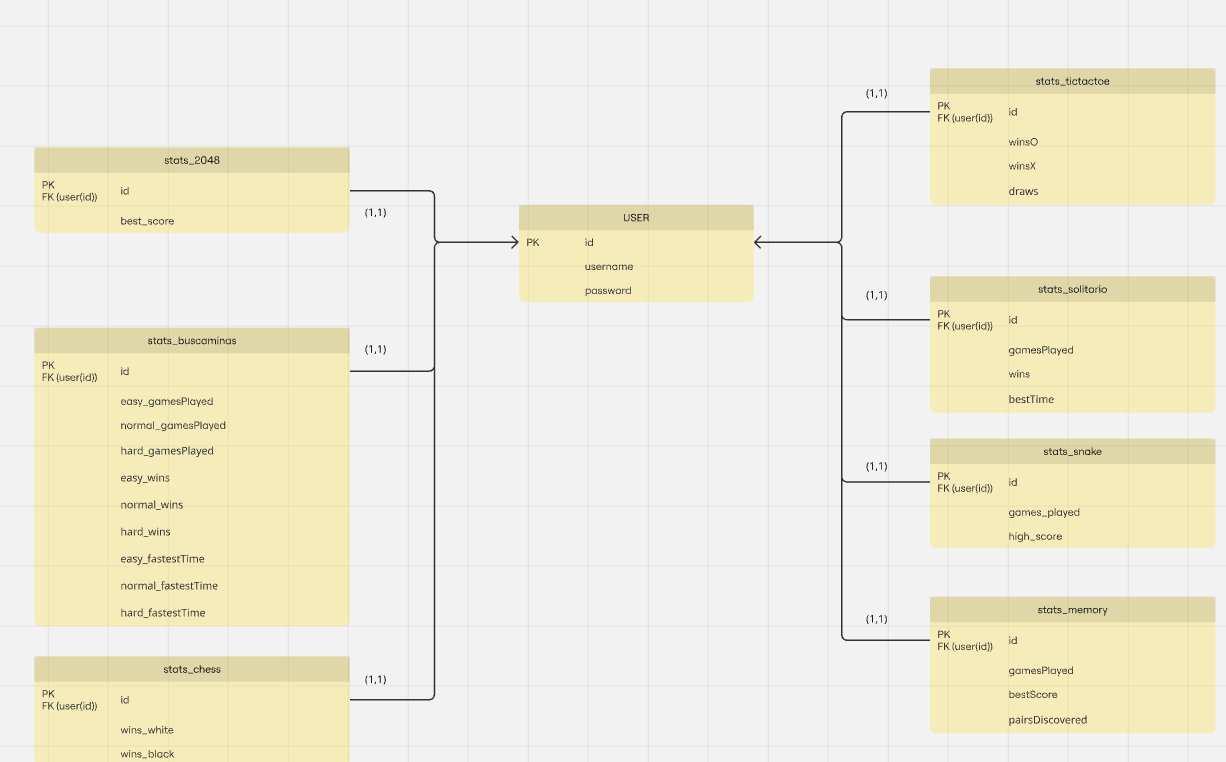
#### Estructura de las tablas de estadísticas

**A continuación, se muestra un resumen de las columnas más importantes de cada tabla:**

| **Tabla** | **Campos principales** |
| --- | --- |
| **stats\_2048** | **id (FK), best\_score** |
| **stats\_buscaminas** | **id (FK), easy\_gamesPlayed, normal\_gamesPlayed, hard\_gamesPlayed, easy\_wins, normal\_wins, hard\_wins, easy\_fastestTime, normal\_fastestTime, hard\_fastestTime** |
| **stats\_chess** | **id (FK), wins\_white, wins\_black** |
| **stats\_memory** | **id (FK), gamesPlayed, bestScore, pairsDiscovered** |
| **stats\_snake** | **id (FK), high\_score, games\_played** |
| **stats\_solitario** | **id (FK), gamesPlayed, wins, bestTime** |
| **stats\_tictactoe** | **id (FK), winsO, winsX, draws** |
| **stats\_towerdefense** | **id (FK), gamesPlayed, wins, waveReached (aún sin implementación completa)** |

#### Relaciones entre tablas

* **Cada tabla de estadísticas utiliza el campo id como clave primaria y clave foránea, que referencia al id de la tabla users.**
* **Esta estructura permite almacenar estadísticas individualizadas por usuario y juego, manteniendo una separación clara y facilitando consultas específicas.**



#### Justificación del diseño

Separar las estadísticas en tablas independientes según cada minijuego facilita:

* La extensión y el mantenimiento de la base de datos al añadir nuevos juegos o campos.
* La consulta y gestión específica de datos sin afectar otras partes.
* La adaptación de cada tabla a las necesidades particulares de cada juego (por ejemplo, diferentes tipos de estadísticas).

Este modelo modular es eficiente y coherente con la arquitectura del proyecto, facilitando su escalabilidad futura.

# Implementación y estructura del código

La codificación del proyecto Ángel Arcade: Minijuegos Web se organizó de manera clara y modular, dividiendo la aplicación en frontend y backend para facilitar el desarrollo, mantenimiento y escalabilidad.

## Cliente (Frontend)

El frontend está desarrollado íntegramente con React y organizado en la carpeta src/, que contiene todo el código fuente de la aplicación:

Estructura principal del cliente: App.js

El archivo App.js es el punto central de la aplicación en React. Desde aquí se controla la navegación interna, el usuario activo, y se gestiona la carga dinámica de cada minijuego o componente funcional, sin utilizar un sistema tradicional de rutas como React Router.

### App.js

* Gestión de sesión del usuario:
  + Al iniciar la aplicación, se comprueba si existe un usuario guardado en localStorage. Si es así, se establece automáticamente como usuario activo mediante useState y useEffect.
  + También se implementa un botón de "Cerrar sesión" que limpia la sesión eliminando el user de localStorage.
* Selección de juegos:
  + Se usa el estado activeGame para determinar qué juego mostrar. Este se actualiza cuando el usuario hace clic en una GameCard.
  + Todos los juegos están registrados en el objeto gameComponents, que asocia claves (como "snake" o "solitario") con su respectivo componente React.
* Pantalla principal de selección:
  + Si no hay ningún juego seleccionado (activeGame es null), se muestra un menú visual con tarjetas (GameCard) representando cada minijuego, con su imagen e invocación del juego correspondiente.
  + También se incluye un botón para acceder al menú de estadísticas generales (GameStatsMenu).
* Renderizado condicional:
  + Si el usuario no ha iniciado sesión, se muestra el formulario LoginRegister.
  + Si hay sesión activa, se muestra el título del proyecto, un saludo con el nombre del usuario, un botón de cerrar sesión, y el juego activo o el menú principal.

Esta lógica sencilla y efectiva evita el uso de rutas externas y permite mantener todo dentro de un solo punto de control.

Componentes destacados conectados con App.js:

* LoginRegister: Permite iniciar sesión o registrarse.
* GameCard: Componente reutilizable para mostrar las tarjetas de selección de minijuegos.
* GameStatsMenu: Muestra las estadísticas globales de los juegos asociados al usuario.
* 2048, Solitario, Buscaminas, TowerDefenseGame, etc.: Juegos individuales cargados dinámicamente según la elección del usuario.

Esta estructura modular permite un mantenimiento y expansión del sistema sencilla, facilitando la incorporación de nuevos juegos o secciones futuras.

### Componente LoginRegister.jsx

El componente LoginRegister.jsx gestiona tanto el inicio de sesión como el registro de nuevos usuarios. Su funcionalidad se adapta dinámicamente en función del estado isRegistering, permitiendo una única interfaz para ambas acciones.

#### Funcionalidad

Estado del formulario: Utiliza useState para controlar los campos de usuario, contraseña, repetir contraseña (solo en registro), errores y visibilidad de contraseñas.

Validación básica

Comprueba que las contraseñas coincidan antes de enviar el formulario en modo registro.

Comunicación con el backend

Realiza una petición POST a /api/auth/login o /api/auth/register en función del modo activo. Los datos del usuario (id y nombre de usuario) se almacenan en localStorage al autenticarse correctamente.

Cambio de modo

Permite alternar entre registro e inicio de sesión con un solo clic, cambiando los campos mostrados y el comportamiento del botón de envío.

Interfaz mejorada

Incluye botones para mostrar/ocultar contraseñas con iconos, mejorando la experiencia de usuario.

### Componente GameStatsMenu.jsx

El componente GameStatsMenu.jsx gestiona la visualización dinámica de estadísticas y logros para distintos minijuegos disponibles en la plataforma. Gracias a su estructura modular y reactiva, permite al usuario explorar su rendimiento en cada juego y ver los logros desbloqueados.

Selección de pestaña  
Utiliza el hook useState para controlar qué juego está actualmente seleccionado por el usuario (mediante el estado selectedTab). La interfaz muestra botones para alternar entre juegos como Tic Tac Toe, 2048, Buscaminas, Ajedrez, Memory, Snake y Solitario.

Carga de estadísticas desde la API  
Al cambiar de pestaña o al cargar el componente, se ejecuta un efecto (useEffect) que realiza una petición GET al backend, solicitando las estadísticas del juego seleccionado mediante la ruta correspondiente (por ejemplo, /api/statsSolitario/:id). Los datos recuperados se almacenan en el estado stats para ser utilizados en la interfaz.

Renderizado condicional de tablas  
Cada juego cuenta con su propia tabla de estadísticas, renderizada condicionalmente en función del juego activo. Estas tablas muestran diferentes datos según el juego (victorias, empates, puntuación máxima, tiempo récord, etc.), presentados de manera clara en una tabla HTML.

Sistema de logros dinámico  
Junto a las estadísticas, se muestran logros relacionados con el juego seleccionado. Cada logro se representa como un objeto con un título y una condición de desbloqueo basada en los datos del estado stats. Estos logros permiten al usuario visualizar su progreso de forma visual e intuitiva, fomentando la rejugabilidad.

Escalabilidad  
La arquitectura del componente permite ampliar fácilmente la funcionalidad para nuevos juegos en el futuro, añadiendo nuevas entradas al array TABS, sus rutas de API correspondientes y nuevas secciones de renderizado.

#### Interfaz amigable

El diseño presenta una interfaz de usuario limpia y organizada, con pestañas fácilmente accesibles y tablas bien estructuradas. El sistema de logros añade una dimensión motivadora al uso de la aplicación, convirtiendo el seguimiento de progreso en una experiencia gratificante para el usuario.

## Servidor (Backend)

### Introducción

El proyecto cuenta con un backend desarrollado en Node.js usando el framework Express, que se encarga de gestionar la lógica relacionada con la autenticación de usuarios y la persistencia de estadísticas de cada minijuego. La arquitectura se basa en una API REST que conecta con una base de datos SQLite para almacenar los datos de forma local.

### Estructura del servidor

El servidor principal (index.js) configura los middlewares necesarios para el correcto funcionamiento:

app.use(cors()); // Permite peticiones desde el frontend

app.use(express.json()); // Parsea peticiones JSON

Además, monta las rutas específicas para cada funcionalidad:

app.use('/api/auth', authRoutes);

app.use('/api/statsSnake', statsSnakeRoutes);

app.use('/api/statsChess', statsChessRoutes);

El servidor escucha en el puerto 3001, a la espera de peticiones del frontend hecho en React.

### Rutas y controladores

Cada juego tiene su propio archivo de rutas que define los endpoints necesarios.  
Por ejemplo, el endpoint de estadísticas de Snake (/api/statsSnake/:userId) permite:

* GET: consultar estadísticas de un usuario. Si no existen, se crea un registro nuevo.
* POST: actualizar estadísticas existentes o crear el registro si aún no existe.

Ejemplo simplificado de un endpoint tipo GET:

**router.get('/:userId', (req, res) => {**

**statsSnakeModel.getStatsByUser(userId, (err, row) => {**

**if (!row) {**

**statsSnakeModel.createStatsForUser(userId, () => {**

**res.json({ userId, highScore: 0, gamesPlayed: 0});**

**});**

**} else {**

**res.json(row);**

**}**

**});**

**});**

Esto permite que la aplicación React siempre obtenga datos válidos al cargar una partida.

### Base de datos y modelos

Se utiliza SQLite como base de datos local para simplificar el despliegue y gestión. Cada tipo de estadísticas se almacena en una tabla distinta (como stats\_snake, stats\_chess, etc.).

El modelo de cada juego contiene funciones reutilizables como:

* getStatsByUser: consulta por ID.
* createStatsForUser: crea un nuevo registro.
* updateStats: actualiza los campos deseados o crea el registro si no existe.

Ejemplo de modelo (statsSnakeModel):

db.run(`

CREATE TABLE IF NOT EXISTS stats\_snake (

id TEXT PRIMARY KEY,

high\_score INTEGER DEFAULT 0,

games\_played INTEGER DEFAULT 0

)

`);

Y su método para actualizar:

const updateStats = (userId, high\_score, games\_played, callback) => {

db.run(

"UPDATE stats\_snake SET high\_score = ?, games\_played =? WHERE id =?",

[high\_score, games\_played, userId],

callback

);

};

### Conexión con el frontend

Desde el cliente en React, las peticiones se realizan usando fetch o librerías similares hacia la API, con rutas como:

fetch(`/api/statsSnake/${userId}`)

fetch(`/api/statsSnake/${userId}`, { method: 'POST', body: JSON.stringify(data)})

Esto permite mantener sincronizados los datos entre el frontend y el backend, y almacenar el progreso o resultados de forma persistente.

# Pruebas

## Objetivo de las pruebas

El propósito de este apartado es asegurar que todas las funcionalidades implementadas en la aplicación funcionan correctamente, tanto en el frontend (interfaz de usuario y lógica de los juegos) como en el backend (gestión de usuarios y estadísticas). Se han realizado pruebas manuales para validar el comportamiento esperado del sistema en diferentes situaciones.

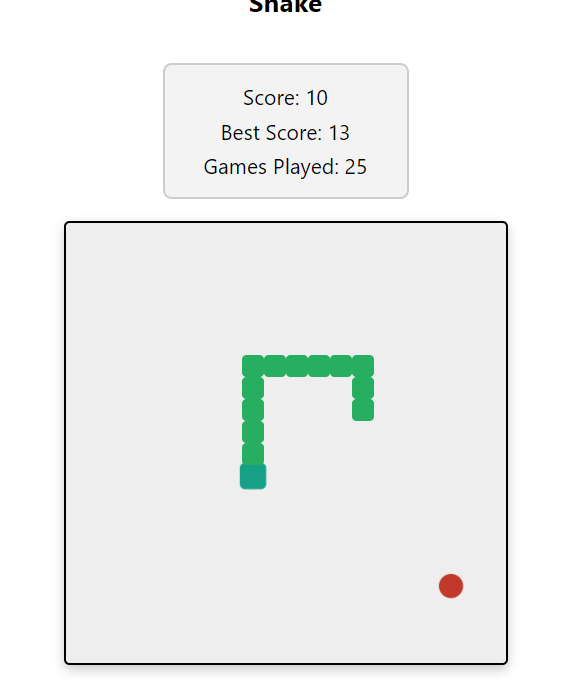
## Tipos de pruebas realizadas

### Pruebas funcionales de los minijuegos

Se han jugado partidas completas en cada uno de los minijuegos para comprobar su funcionamiento básico y avanzado:

#### Snake

* + Inicia correctamente y responde a las teclas de dirección. ✅
  + El marcador aumenta al comer comida. ✅



* + Al finalizar la partida, se actualiza la puntuación máxima y las partidas jugadas en la base de datos. ✅

Captura de pantalla de un celular con letras

El contenido generado por IA puede ser incorrecto.

En la siguiente partida los datos recibidos de la base de datos son estos:

Datos recibidos: { highScore: 16, gamesPlayed: 2 }

#### Buscaminas

* + Las minas se generan aleatoriamente. ✅

Gráfico, Gráfico de dispersión, Gráfico de cajas y bigotes

El contenido generado por IA puede ser incorrecto.Gráfico, Gráfico de dispersión, Gráfico de cajas y bigotes

El contenido generado por IA puede ser incorrecto.

* + Se detecta correctamente cuándo se gana o se pierde. ✅

Gráfico, Gráfico de dispersión, Gráfico de cajas y bigotes

El contenido generado por IA puede ser incorrecto.Imagen que contiene Calendario

El contenido generado por IA puede ser incorrecto.

* + Se guarda la duración de la partida y el resultado en las estadísticas del usuario. ✅

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

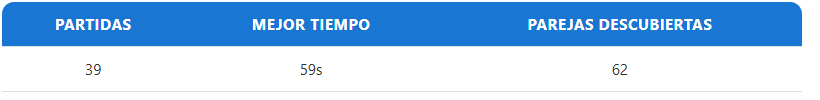
#### Memory

* + Las cartas se barajan correctamente al empezar. ✅

Teclado de computadora

El contenido generado por IA puede ser incorrecto.Imagen que contiene electrónica, decorado, teclado, luz

El contenido generado por IA puede ser incorrecto.

* + El número de parejas descubiertas se contabiliza y guarda. ✅
  + Se guarda el mejor tiempo y el número de partidas. ✅

#### Solitario

* + Se pueden mover cartas entre columnas, del mazo y a las fundaciones. ✅

Imagen que contiene diferente, diversos, montón, llenado

El contenido generado por IA puede ser incorrecto.

* + El sistema reconoce automáticamente una partida ganada. ✅Imagen de la pantalla de un video juego

    El contenido generado por IA puede ser incorrecto.
  + Se actualizan correctamente el número de partidas, victorias y tiempos. ✅

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

#### Tic Tac Toe

* + Se alternan los turnos correctamente entre X y O. ✅

Una captura de pantalla de un celular con texto e imágenes

El contenido generado por IA puede ser incorrecto.Imagen de la pantalla de un celular con texto e imágenes

El contenido generado por IA puede ser incorrecto.Un reloj con números romanos

El contenido generado por IA puede ser incorrecto.

* + El juego detecta empates y victorias. ✅

Imagen que contiene Forma

El contenido generado por IA puede ser incorrecto.Una captura de pantalla de un celular con texto e imágenes

El contenido generado por IA puede ser incorrecto.Imagen de la pantalla de un video juego

El contenido generado por IA puede ser incorrecto.

* + Se guardan estadísticas por usuario. ✅

Texto

El contenido generado por IA puede ser incorrecto.

#### 2048

* + Las teclas de dirección funcionan correctamente. ✅

Imagen de la pantalla de un celular con letras

El contenido generado por IA puede ser incorrecto.Un teclado de computadora

El contenido generado por IA puede ser incorrecto.Una calculadora junto a un teclado

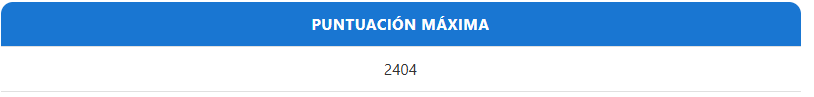
El contenido generado por IA puede ser incorrecto.Un teclado de computadora

El contenido generado por IA puede ser incorrecto.Una calculadora junto a un teclado

El contenido generado por IA puede ser incorrecto.

Orden de movimientos: derecha, arriba, abajo, izquierda

* + Se combinan correctamente los bloques. ✅
  + Se guarda la mejor puntuación alcanzada. ✅



### Pruebas del backend y la API REST

Se han utilizado Postman y el navegador para probar manualmente los endpoints. Los principales puntos comprobados fueron:

* **GET** /api/statsSnake/:userId
  + Devuelve estadísticas si existen.

[GET] Estadísticas encontradas para userId=1: { id: '1', high\_score: 13, games\_played: 25 }

* + Si no existen, las crea con valores por defecto.

[GET] No existe registro para userId=5, creando uno nuevo...

[GET] Registro creado para userId=5, devolviendo valores por defecto.

* **POST** /api/statsSnake/:userId
  + Crea o actualiza correctamente las estadísticas con los valores recibidos.

[POST] /api/statsChess/5 - Datos recibidos: { wins\_white: 1, wins\_black: 0 }

[POST] Stats existentes para userId=5, actualizando...

[POST] Stats actualizadas correctamente para userId=5.

Esto se ha replicado con el resto de rutas de estadísticas: /statsBuscaminas, /statsMemory, /statsSolitario, etc.

### Pruebas de persistencia de datos

Tras reiniciar la aplicación o cerrar sesión, los datos del usuario se mantienen correctamente gracias al uso de localStorage y la base de datos SQLite. Se ha comprobado que las estadísticas se guardan de forma permanente, sin perderse entre sesiones.

### Pruebas de autenticación

* Registro e inicio de sesión funcionan correctamente.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.Calendario

El contenido generado por IA puede ser incorrecto.

* Se impide el registro de usuarios con nombres repetidos.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

* Se requiere poner dos veces la misma contraseña para poder registarse.  
  Interfaz de usuario gráfica, Aplicación

  El contenido generado por IA puede ser incorrecto.
* La sesión se guarda en localStorage y se mantiene al recargar la página.

Aplicación, Teams

El contenido generado por IA puede ser incorrecto.

* Cada usuario accede únicamente a sus estadísticas.

Interfaz de usuario gráfica, Texto, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

### Pruebas de la interfaz

* La navegación entre secciones es fluida.
* No hay errores en la consola del navegador.
* Los logros se actualizan correctamente.

## Resultados

Tras todas las pruebas manuales realizadas, se ha verificado que el funcionamiento general del sistema es correcto. Se detectaron algunos pequeños errores durante el desarrollo, como que trataba de guardar un campo con un nombre invalido en la tabla de stats\_chess y que el juego de snake empezaba nada más acabar y eso creaba alguna que otra molestia, que fueron solucionados a tiempo. El sistema responde como se espera en todos los casos evaluados.

# Posibles mejoras futuras

Aunque el proyecto cumple con los objetivos principales planteados, se han identificado varias áreas en las que se podría seguir mejorando tanto la funcionalidad como la experiencia del usuario. Estas mejoras podrían implementarse en futuras versiones del sistema:

Mejoras en los minijuegos

* **Incorporación de nuevos juegos**: Ampliar la colección con más minijuegos clásicos como Tetris, Sudoku, Pac-Man o Conecta 4.
* **Mayor dificultad progresiva**: En algunos juegos como Snake o 2048, se podría implementar una dificultad adaptativa o niveles.

## Mejoras en el sistema de estadísticas

* **Historial completo de partidas**: Permitir al usuario ver el historial detallado de sus partidas (fechas, duración, resultado).
* **Gráficas de progreso**: Visualizar la evolución de las estadísticas mediante gráficos.
* **Ranking global y por juego**: Implementar una clasificación entre usuarios para fomentar la competitividad.

## Mejora del sistema de logros

## Experiencia de usuario (UX/UI)

* **Diseño responsive completo**: Mejorar la adaptabilidad para dispositivos móviles y tablets.
* **Animaciones y sonidos**: Añadir efectos visuales y sonoros para hacer la experiencia más inmersiva.

## Backend y base de datos

* **Cambio a una base de datos más robusta**: Considerar el uso de PostgreSQL o MySQL para escalar el sistema si crece el número de usuarios.

## Infraestructura y despliegue

* **Despliegue online**: Subir la aplicación a un servidor (por ejemplo, Vercel + Render o Railway) para que sea accesible desde cualquier dispositivo.
* **Sistema de copias de seguridad de la base de datos**: Implementar backups periódicos automáticos para evitar la pérdida de datos.

# Bibliografía

* React – Biblioteca JavaScript para construir interfaces de usuario.  
  <https://reactjs.org/>
* Express – Framework web para Node.js.  
  <https://expressjs.com/>
* Node.js – Entorno de ejecución para JavaScript del lado del servidor.  
  <https://nodejs.org/>
* SQLite – Sistema de gestión de bases de datos ligero.  
  <https://www.sqlite.org/>
* npm – Gestor de paquetes para JavaScript.  
  <https://www.npmjs.com/>
* CORS – Recurso para entender el intercambio de recursos entre orígenes.  
  <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>
* HTML – Referencia y tutoriales.  
  <https://www.w3schools.com>
* CSS – Referencia y tutoriales.  
  <https://www.w3schools.com>
* JavaScript – Guía y documentación.  
  <https://developer.mozilla.org/es/docs/Web/JavaScript>
* Git – Sistema de control de versiones.  
  <https://git-scm.com/doc>
* GitHub – Plataforma de alojamiento de repositorios.  
  <https://docs.github.com/>
* Imágenes de cartas para el Solitario – Repositorio de cartas SVG.  
  <https://github.com/hanhaechi/playing-cards>