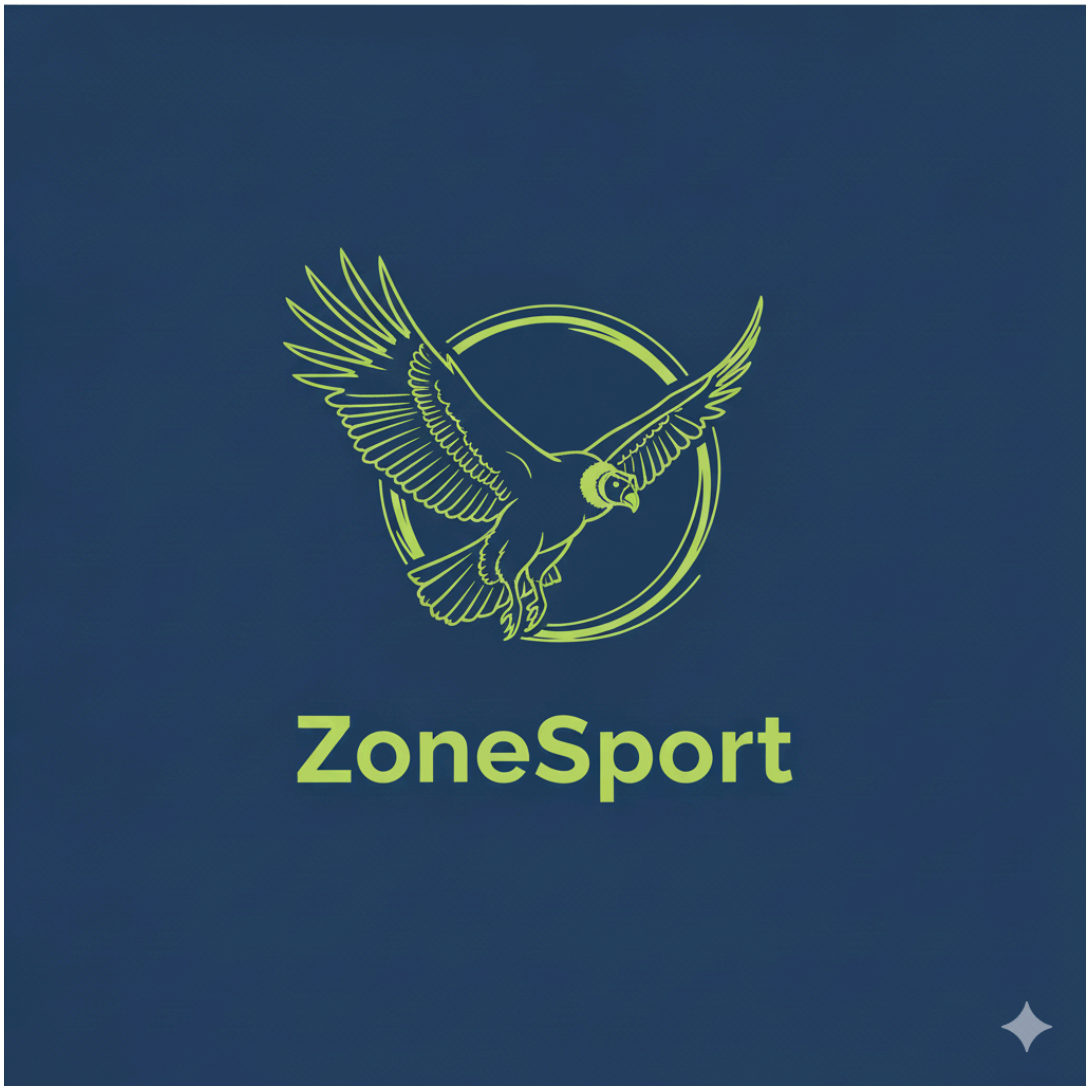
# 📄 Documentación Final del Proyecto: ZoneSport




## 0. Identidad de Marca y Estilo

Esta identidad está diseñada para funcionar en el contexto de Antioquia, Colombia.

| Aspecto | Detalle |
|---|---|
| **Slogan** | **"Organiza, Compite, Clasifica. El deporte en Antioquia tiene un nuevo centro."** |
| **Paleta de Colores** | **1. Azul Profundo (\#007ACC):** [Primario] – Simboliza confianza, tecnología y deportividad. Funciona bien como color de acción en modos claro y oscuro. |

| Aspecto | Detalle |
|---|---|
| | **2. Verde Lima (\#8BC34A):** [Secundario/Acento] – Representa energía, crecimiento y victoria. Ideal para estados de éxito, notificaciones y elementos de *ranking*. |
| **Tipografía Principal** | **Poppins** (Google Fonts): Moderna, geométrica y legible. |
| **Tipografía Monospace** | **JetBrains Mono** (Google Fonts): Clara y legible, ideal para la alineación de datos en tablas de resultados. |

# 1. Visión y Alcance

| Aspecto | Detalle |
|---|---|
| **Visión General** | Crear una plataforma web eficiente para la **Gestión Centralizada de Eventos Deportivos** (ligas y torneos) en el área de Antioquia. El enfoque es la organización, la **introducción de resultados** y la **generación dinámica de Tablas de Clasificación**, eliminando por completo las funcionalidades de "Red Social". |
| **Tecnologías Clave** | **Frontend:** Next.js (App Router) + TypeScript. **Backend: NestJS** + **PostgreSQL**. **Infraestructura: Docker** para desarrollo y despliegue. |
| **Core del Producto** | **Tablas de Clasificación Dinámicas y Oficiales** con métricas personalizadas por deporte. |

# 2. Arquitectura de Desarrollo y Producción

## 2.1. Arquitectura de Despliegue (Opción A: Gratuita y Escalable)

| Componente | Tecnología | Entorno de Despliegue Gratuito |
|---|---|---|
| **Frontend** | Next.js (SSR/CSR) | **Vercel** (Plan Hobby) |
| **Backend (API)** | **NestJS** (Server API) | **Render** (Free Web Services) |
| **Base de Datos** | **PostgreSQL** (TypeORM) | **Render Postgres** (Free Databases) |

## 2.2. Desarrollo Local con Docker 🐳

Se usará **Docker Compose** para orquestar los servicios en el entorno local.

| Servicio Docker | Imagen Base | Puerto | Función |
|---|---|---|---|
| db | postgres:16-alpine | 5432:5432 | Base de datos PostgreSQL para desarrollo. |
| backend | node:20-slim | 3000:3000 | Servidor NestJS. |
| frontend | node:20-slim | 3001:3001 | Aplicación Next.js. |

## 2.3. Ventajas de las Tecnologías Clave (Nuevo)

| Tecnología | Rol en el Proyecto | Ventajas Principales (Simplificadas) |
|---|---|---|
| **Vercel** | Hosting y CI/CD del Frontend | **Despliegues Automatizados:** Integración directa con GitHub para CI/CD continuo. **Rendimiento Global:** Uso de CDN y *Edge Functions* para baja latencia. **Escalabilidad y Seguridad:** Escalado automático, SSL y protección (DDoS) integrados. |
| **Next.js** | Framework Frontend | **Rendimiento Óptimo:** Soporte nativo para Server-Side Rendering (SSR) y Static Site Generation (SSG). **Developer Experience:** Enrutador de aplicaciones (App Router) y tipado estático con TypeScript. |
| **NestJS** | Framework Backend (API) | **Estructura Profesional:** Basado en TypeScript y Arquitectura de Módulos (como Angular). **Robustez:** Facilita la implementación de seguridad (JWT, Guards, Rate Limiting) y la arquitectura orientada a microservicios. |
| **PostgreSQL** | Base de Datos Relacional | **Integridad de Datos:** Garantiza la coherencia ACID, crucial para las Tablas de Clasificación y resultados. **Fiabilidad:** Base de datos robusta y escalable, con soporte nativo para JSON (para reglas de clasificación dinámicas). |

# 3. Implementación y Seguridad

## 3.1. Gestión de Variables de Entorno

| Variable | Uso | Entorno | Comentarios |
|---|---|---|---|
| **NODE_ENV** | Define si es development o production. | Ambos | Crucial para la lógica condicional. |
| **DATABASE_URL** | String de conexión completo de PostgreSQL. | Ambos | **Secreto en producción**. |

| Variable | Uso | Entorno | Comentarios |
|---|---|---|---|
| **JWT_SECRET** | Clave secreta para firmar tokens de autenticación. | Backend | Debe ser una **cadena larga, aleatoria y compleja**. |
| **NEXT_PUBLIC_API_URL** | URL pública del Backend de NestJS. | Frontend | Permite al *frontend* saber dónde buscar la API. |

### 3.2. Seguridad en NestJS (Backend) 🔒

- **Protección de Rutas:** Implementar Guards para verificar el **JWT** en todas las rutas privadas.
- **Validación de Esquemas:** Usar Pipes con class-validator para validar rigurosamente todos los *payloads* de entrada (DTOs).
- **Prevención de Ataques Comunes:** Configurar **CORS**, usar **Helmet** (contra XSS, *clickjacking*), e implementar **Rate Limiting** (contra DoS).

### 3.3. Conexión Segura a PostgreSQL

- **Desarrollo (Local):** Conexión insegura a través de Docker Compose.
- **Producción (Render): FORZAR la conexión SSL** en la configuración de TypeORM en NestJS, y almacenar DATABASE_URL directamente en la configuración de **Variables de Entorno** de Render.

## 4. Fases del Proyecto (SDLC)

El proyecto seguirá un ciclo de vida de desarrollo de software (SDLC) iterativo y ágil.

| Fase | Duración Estimada | Entregables Clave |
|---|---|---|
| **1. Planeación (Actual)** | 1 Semana | Documento de Requisitos Finalizado, Diseño de Base de Datos (Esquema), *Wireframes* de Rutas Principales. |
| **2. Arquitectura & Setup** | 2 Semanas | Configuración de Docker/Docker Compose, Setup de Repositorios (Git/GitHub), Módulos Base de NestJS (Auth, User), Setup de Next.js (Tailwind, TypeScript). |
| **3. Desarrollo Core** | 4-6 Semanas | Implementación de todos los Modelos (TypeORM) y Endpoints CRUD, Lógica del Algoritmo de Clasificación, Despliegue Inicial de Pruebas (Render/Vercel). |
| **4. Pruebas y QA** | 2 Semanas | Pruebas Unitarias/Integración (Jest), Pruebas de Carga/Stress, Pruebas de Usabilidad (UX) por parte de usuarios finales. |

| Fase | Duración Estimada | Entregables Clave |
|---|---|---|
| **5. Despliegue y Lanzamiento** | 1 Semana | Configuración Final de Producción (SSL, dominio), Monitoreo y *Hotfixes*. |

# 5. Estrategia de Pruebas (Testing)

Se implementará una estrategia de pruebas por capas para asegurar la calidad y la robustez del sistema.

| Tipo de Prueba | Herramientas | Enfoque |
|---|---|---|
| **Pruebas Unitarias** | Jest / Vitest | Módulos individuales de NestJS (servicios, *controllers*) y componentes de Next.js. **Crucial para el Algoritmo de Clasificación.** |
| **Pruebas de Integración** | Supertest (en NestJS) | Asegurar que los *endpoints* de la API (DB \leftarrow TypeORM \leftarrow NestJS) funcionen correctamente con datos simulados. |
| **Pruebas End-to-End (E2E)** | Cypress / Playwright | Simular el flujo de usuario completo (ej. Registro \rightarrow Creación de Evento \rightarrow Ingreso de Resultados \rightarrow Ver Clasificación). |
| **Pruebas de Carga** | k6 / Artillery | Simular el tráfico esperado de usuarios de Antioquia para asegurar que Render y PostgreSQL soporten la carga (RNF de Escalabilidad). |

# 6. Especificación Técnica del Backend

## 6.1. Estructura de Datos Central (PostgreSQL Models)

| Modelo (Tabla) | Descripción | Campos Clave y Tipos de Datos |
|---|---|---|
| **User** | Atletas, organizadores, administradores. | id (PK), email, password (hashed), role (ENUM: ATHLETE, ORGANIZER, ADMIN). |
| **Sport** | Catálogo de deportes soportados. | id (PK), name, classification_rules (JSON/TEXT para definir la lógica de puntos, *wins*, etc.). |
| **Event / Tournament** | Entidad principal. | id (PK), name, sport_id (FK), organizer_id (FK). |
| **Match / Result** | Partidos jugados y resultados. | id (PK), event_id (FK), score_a, |

| Modelo (Tabla) | Descripción | Campos Clave y Tipos de Datos |
|---|---|---|
| | | score_b, status (PLAYED, SCHEDULED). |
| **Classification** | **Tabla derivada** del ranking actual. | event_id (FK), team_id (FK), points, wins, goals_for, goals_against. |

### 6.2. Requisitos de la API (Endpoints de NestJS)

| Módulo | Endpoint | Método | Funcionalidad Clave | Autenticación Requerida |
|---|---|---|---|---|
| **Autenticación** | /auth/login | POST | Inicio de sesión y retorno de JWT. | Pública |
| **Eventos (Público)** | /events | GET | Listado de eventos con filtros. | Pública |
| | /events/:id/classification | GET | **Tabla de Clasificación Dinámica** del evento. | Pública |
| **Gestión Eventos** | /events | POST | Creación de un nuevo evento. | JWT + Role: ORGANIZER/ADMIN |
| | /events/:id/results | POST | **Ingreso de resultados** de un partido. | JWT + Role: ORGANIZER/ADMIN |

# 7. Flujo de Despliegue (CI/CD)

El flujo se automatiza a través de las integraciones nativas de Vercel y Render con GitHub.

| Paso | Acción | Herramienta |
|---|---|---|
| **1. Push** | El código se sube al repositorio de GitHub. | Git/GitHub |
| **2. Despliegue de Frontend** | **Vercel** detecta cambios y ejecuta el *build* de Next.js. | Vercel |
| **3. Despliegue de Backend** | **Render** detecta cambios y ejecuta el Dockerfile. | Render |
| **4. Conexión** | El **Frontend** (Vercel) consume la API del dominio de Render. | Vercel ➡️ Render |

# 8. Requisitos No Funcionales (RNF)

- **Seguridad:** Implementación de JWT, bcrypt para contraseñas, y conexión SSL forzada a PostgreSQL.
- **Performance:** Tiempos de respuesta rápidos (<500ms) para la carga de clasificaciones.
- **Escalabilidad Futura:** Arquitectura modular (NestJS) y uso de Docker para facilitar la migración a planes de pago si el tráfico supera los límites gratuitos.
- **Usabilidad (UX):** Interfaz basada en shadcn/ui y **Tailwind CSS** para un diseño responsivo y accesible en cualquier dispositivo.