

02 - Arquitectura Técnica - Tactic Neo Eleven

Decisiones de Stack Tecnológico

Resumen de la Arquitectura

Frontend (Next.js) ↔ Backend API (Next.js API Routes) ↔ Base de Datos (PostgreSQL/Supabase)

↕

Hosting (Vercel) + Auth (Supabase)

Base de Datos: PostgreSQL con Supabase

¿Por qué PostgreSQL?

Decisión: PostgreSQL como motor de base de datos principal

Justificación:

- ✓ **Relaciones complejas:** El proyecto tiene múltiples entidades interrelacionadas (jugadores, contratos, partidos, estadísticas, lesiones)
- ✓ **Integridad referencial:** FOREIGN KEYS garantizan consistencia de datos críticos
- ✓ **Consultas avanzadas:** JOINS complejos para reportes y estadísticas
- ✓ **Escalabilidad probada:** Soporta millones de registros sin problemas
- ✓ **ACID compliance:** Transacciones seguras para operaciones críticas
- ✓ **JSON support:** Flexibilidad para campos como "posiciones secundarias"

Alternativas descartadas:

- ✗ **MongoDB:** Menos eficiente para relaciones complejas, más complejo para reportes
- ✗ **MySQL:** Funcionalidades JSON menos maduras que PostgreSQL
- ✗ **SQLite:** Limitaciones de concurrencia para aplicación web

¿Por qué Supabase?

Decisión: Supabase como plataforma de base de datos

Justificación:

- ✓ **PostgreSQL nativo:** Aprovecha toda la potencia de PostgreSQL
- ✓ **Autenticación integrada:** Sistema de usuarios sin implementación adicional
- ✓ **Real-time subscriptions:** Updates automáticos en la UI
- ✓ **Row Level Security:** Seguridad granular por roles de usuario

- ☒ **API REST automática:** Endpoints generados automáticamente
- ☒ **Dashboard administrativo:** Gestión visual de datos
- ☒ **Backups automáticos:** Protección de datos sin configuración
- ☒ **Experiencia previa positiva:** El desarrollador ya ha trabajado con Supabase

Alternativas descartadas:

- ☒ **PostgreSQL self-hosted:** Requiere mantenimiento, backups, seguridad manual
- ☒ **Firebase:** NoSQL no adecuado para este tipo de relaciones
- ☒ **PlanetScale:** MySQL, menos funcionalidades que PostgreSQL
- ☒ **AWS RDS:** Más complejo, requiere configuración adicional de auth

Frontend: Next.js 14/15 con React

¿Por qué Next.js?

Decisión: Next.js como framework principal de frontend

Justificación:

- ☒ **Multiplataforma:** Responsive design funciona en PC, tablet, móvil
- ☒ **SSR/SSG:** Rendimiento superior, mejor SEO
- ☒ **API Routes:** Backend integrado para lógica personalizada
- ☒ **File-based routing:** Organización intuitiva de páginas
- ☒ **Optimizaciones automáticas:** Image optimization, code splitting
- ☒ **TypeScript nativo:** Mejor developer experience y menos errores
- ☒ **PWA ready:** Posibilidad de app móvil nativa
- ☒ **Ecosystem maduro:** Amplia comunidad y librerías disponibles

Alternativas descartadas:






- ☒ **React SPA puro:** Peor rendimiento inicial, más configuración
- ☒ **Vue/Nuxt:** Menor experiencia del desarrollador con Vue
- ☒ **Angular:** Overhead excesivo para el proyecto
- ☒ **Svelte/SvelteKit:** Ecosystem menos maduro

¿Por qué React?

Decisión: React como librería de UI

Justificación:

- ☒ **Experiencia del desarrollador:** Dominio técnico completo








-  **Component-based:** Reutilización perfecta para elementos como "carta de jugador"
-  **Estado predictable:** Gestión clara del estado de la aplicación
-  **Ecosystem gigante:** Librerías para cualquier necesidad específica
-  **Testing maduro:** Jest, React Testing Library bien establecidos
-  **Hooks:** Lógica reusable entre componentes

Hosting: Vercel




¿Por qué Vercel?

Decisión: Vercel como plataforma de hosting

Justificación:

-  **Next.js nativo:** Optimizado específicamente para Next.js
-  **Deploy automático:** Push a Git → Deploy automático
-  **Edge functions:** Rendimiento global optimizado
-  **Preview branches:** Testing de features sin afectar producción
-  **Analytics integrado:** Métricas de rendimiento sin configuración
-  **Escalabilidad automática:** Maneja picos de tráfico transparentemente
-  **Experiencia previa:** El desarrollador ya ha usado Vercel exitosamente

Alternativas descartadas:






-  **Netlify:** Menos optimizado para Next.js que Vercel
-  **AWS/Digital Ocean:** Requiere configuración de servidor, DevOps
-  **Railway/Render:** Menos maduro para aplicaciones React avanzadas

Autenticación: Supabase Auth

¿Por qué Supabase Auth?

Decisión: Sistema de autenticación integrado de Supabase

Justificación:

-  **Integración nativa:** Se conecta directamente con la base de datos
-  **Row Level Security:** Permisos granulares automáticos
-  **Múltiples providers:** Email, Google, GitHub si se necesita después
-  **JWT tokens:** Estándar de la industria, seguro
-  **Reset password:** Funcionalidades básicas incluidas

- ☒ **Session management:** Manejo automático de sesiones
- ☒ **Role-based access:** Perfecto para el sistema de roles (presidente, entrenador, etc.)

UI/UX: Tailwind CSS

¿Por qué Tailwind CSS?

Decisión: Tailwind CSS para el diseño de la interfaz

Justificación:

- ☒ **Utility-first:** Desarrollo rápido y consistente
- ☒ **Responsive nativo:** Mobile-first design por defecto
- ☒ **Customización total:** Themes personalizados para Tactic Neo Eleven
- ☒ **Bundle size:** Solo incluye estilos realmente usados
- ☒ **Developer experience:** Autocomplete, no context switching
- ☒ **Consistencia:** Sistema de colores y espaciados uniforme
- ☒ **Componentes reutilizables:** Fácil crear biblioteca de componentes

Alternativas descartadas:

- ☒ **CSS modules:** Más verboso, menos productive
- ☒ **Styled-components:** Runtime overhead
- ☒ **Bootstrap:** Menos moderno, más rígido
- ☒ **Chakra UI:** Buena opción pero menos control granular

Gestión de Estado: React Built-in + SWR

¿Por qué esta combinación?

Decisión: useState/useReducer + SWR para data fetching

Justificación:

- ☒ **Simplicidad inicial:** useState para estado local simple
- ☒ **SWR para server state:** Cache automático, revalidación, optimistic updates
- ☒ **Real-time:** SWR + Supabase real-time subscriptions
- ☒ **No over-engineering:** Evita complejidad innecesaria de Redux
- ☒ **Escalabilidad:** Se puede añadir Zustand/Redux después si es necesario

Evolución futura si es necesario:

- **Zustand:** Para estado global simple

- **Redux Toolkit:** Solo si la complejidad lo justifica

Herramientas de Desarrollo

TypeScript

Decisión: TypeScript en todo el proyecto

Justificación:

- ☒ **Type safety:** Menos errores en runtime
- ☒ **Better DX:** Autocomplete, refactoring seguro
- ☒ **Documentation:** Los tipos sirven como documentación
- ☒ **Escalabilidad:** Esencial para proyectos grandes

ESLint + Prettier

Decisión: Linting y formatting automático

Justificación:

- ☒ **Código consistente:** Mismo estilo siempre
- ☒ **Menos errores:** ESLint catch errores comunes
- ☒ **Productivity:** Auto-format al guardar

Git + GitHub

Decisión: Control de versiones con Git, repositorio en GitHub

Justificación:

- ☒ **Historial completo:** Cada cambio documentado
- ☒ **Branching:** Features en ramas separadas
- ☒ **Collaboration ready:** Preparado para múltiples desarrolladores
- ☒ **Integration:** CI/CD con Vercel integrado

Arquitectura de Carpetas

Estructura del Proyecto

```
tactic-neo-eleven/  
├── docs/           # Documentación del proyecto  
├── src/  
│   ├── app/       # Next.js App Router  
│   │   ├── (dashboard)/ # Rutas del dashboard  
│   │   └── auth/    # Páginas de autenticación
```

└─ api/	# API Routes
└─ components/	# Componentes reutilizables
└─ ui/	# Componentes básicos (botones, inputs)
└─ jugadores/	# Componentes específicos de jugadores
└─ partidos/	# Componentes de partidos
└─ layout/	# Componentes de layout
└─ lib/	# Utilidades y configuraciones
└─ supabase/	# Cliente de Supabase
└─ validations/	# Esquemas de validación (Zod)
└─ utils/	# Funciones auxiliares
└─ types/	# Definiciones de tipos TypeScript
└─ hooks/	# Custom hooks de React
└─ supabase/	# Migraciones y configuración de DB
└─ public/	# Assets estáticos
└─ tests/	# Tests unitarios e integración

Justificación de la Estructura

- **Separación por feature:** Componentes agrupados por funcionalidad
- **Colocation:** Archivos relacionados cerca unos de otros
- **Escalabilidad:** Fácil añadir nuevas features sin reorganizar
- **DX:** Fácil encontrar y mantener código

Consideraciones de Seguridad

Row Level Security (RLS)

- **Políticas granulares:** Cada usuario solo ve sus datos
- **Roles diferenciados:** Presidente ve todo, entrenador solo su área
- **Audit trail:** Registro de quién modifica qué y cuándo

Validación de Datos

- **Cliente + Servidor:** Validación en ambos lados
- **Zod schemas:** Validaciones tipadas y reutilizables
- **Sanitización:** Limpieza de inputs para evitar XSS

Autenticación Segura

- **JWT tokens:** Estándar de la industria
- **Refresh tokens:** Sesiones seguras de larga duración
- **Rate limiting:** Protección contra ataques de fuerza bruta

Consideraciones de Rendimiento

Optimizaciones de Base de Datos

- **Índices estratégicos:** En campos de búsqueda frecuente
- **Query optimization:** Consultas eficientes con JOINS optimizados
- **Connection pooling:** Gestión eficiente de conexiones

Optimizaciones de Frontend

- **Code splitting:** Carga solo el código necesario
- **Image optimization:** Next.js Image component
- **Caching:** SWR cache + Vercel edge caching
- **Lazy loading:** Componentes cargados bajo demanda

Plan de Monitoring y Observabilidad

Métricas de Aplicación

- **Vercel Analytics:** Performance metrics automáticas
- **Error tracking:** Integración con Sentry (fase futura)
- **User analytics:** Hotjar o similar (fase futura)

Métricas de Base de Datos

- **Supabase Dashboard:** Query performance, conexiones activas
- **Custom metrics:** Queries lentas, operaciones críticas

Backup y Recuperación

Estrategia de Backup

- **Supabase automático:** Backups diarios automáticos
- **Git history:** Código siempre recuperable
- **Export functionality:** Permitir export de datos del usuario

Plan de Recuperación

- **RTO (Recovery Time Objective):** < 1 hora
- **RPO (Recovery Point Objective):** < 24 horas
- **Procedimientos documentados:** Pasos claros de recuperación

Justificación de Decisiones No Tomadas

¿Por qué NO microservicios?

- Complejidad innecesaria para el tamaño del proyecto
- Un monolito bien estructurado es más eficiente para este caso
- Facilita desarrollo y deployment

¿Por qué NO GraphQL?

- REST es suficiente para las necesidades del proyecto
- SWR maneja eficientemente el caching de REST
- Menor curva de aprendizaje

¿Por qué NO Docker?

- Vercel maneja deployment automáticamente
- Supabase es managed service
- Complejidad adicional sin beneficio claro

Conclusión: Esta arquitectura proporciona una base sólida, escalable y mantenible para Tactic Neo Eleven, equilibrando simplicidad inicial con capacidad de crecimiento futuro.