PLAN DE IMPLANTACION WEB VELADA VI

LUIS MIGUEL HERRÁ ALPUENTE
https://kokoworks.es

Índice de contenidos

1. Descripción general del proyecto

Infovelada.com es el sitio web oficial del evento anual de boxeo amateur entre streamers hispanohablantes organizado por Ibai Llanos, celebrado el 25 de julio de 2026 en el Estadio La Cartuja de Sevilla, la Velada del Año VI.

El sitio actúa como hub central para el seguimiento del evento: votaciones en tiempo real sobre los combates, resultados, ranking post-evento y acceso a información sobre los participantes. La vida útil activa del proyecto es de aproximadamente dos meses, desde la presentación oficial hasta aproximadamente un mes después del evento.

1.1. Escala y requisitos de rendimiento

Parámetro	Valor estimado
Usuarios concurrentes máximos	150.000 – 350.000
Votos totales durante el evento	5.000.000
Combates a votar	7
Vida útil activa	~2 meses
Pico secundario de tráfico	Post-evento (ranking compartible)

Requisito crítico	El sistema de votación debe mantenerse operativo bajo carga máxima sin degradación visible. Un fallo durante la retransmisión en directo con millones de espectadores tendría impacto inmediato.

2. Stack tecnológico

El stack es una decisión fija de proyecto. Se detallan a continuación las justificaciones técnicas de cada elección.

2.1. Framework: Astro JS

Astro aplica hidratación parcial mediante el modelo de islas (Island Architecture): el JavaScript se carga únicamente en los componentes que lo requieren de forma estricta, dejando el resto del contenido como HTML estático. Para un sitio de evento donde la mayoría del contenido es estático y la performance es determinante para SEO y tiempo de carga inicial, esta arquitectura es la opción óptima frente a frameworks SPA tradicionales.

2.2. Lenguaje: TypeScript

Integración nativa con Astro. Aporta facilidad en implantación para debugging, seguridad de tipos, legibilidad del código y mantenibilidad a largo plazo. Relevante especialmente considerando que el proyecto puede ser tomado como base en ediciones futuras por desarrolladores distintos a los actuales.

2.3. Estilos: Tailwind CSS v4

Compatible con TypeScript y Astro. Permite desarrollar interfaces de usuario con rapidez mediante clases utilitarias, sin necesidad de gestionar hojas de estilos adicionales. La versión 4 es la actual y estable para este stack.

2.4. Gestor de paquetes: pnpm

Seleccionado sobre npm por dos razones principales:

- Velocidad en instalaciones: relevante si es necesario levantar nuevas instancias con tiempo de arranque mínimo.
- Seguridad: se han detectado vulnerabilidades en versiones recientes de npm que hacen preferible pnpm para este proyecto.

2.5. Hosting: Vercel

Plataforma de despliegue continuo con edge network global. Permite servir contenido estático y funciones serverless desde el edge más próximo al usuario, reduciendo la latencia tanto para España como para LATAM. El despliegue automático desde repositorio Git simplifica el proceso de publicación, incluyendo la generación del JSON de resultados post-evento.

3. Base de datos

3.1. Tecnología elegida: PostgreSQL en cloud

Se requiere una base de datos relacional gestionada con capacidad de escalar automáticamente durante picos de carga. Los dos candidatos evaluados son Neon y Supabase.

3.2. Evaluación de candidatos

Criterio	Neon (preferido)	Supabase (alternativa)
Modelo	Serverless, pago por compute real	PostgreSQL gestionado
Coste en reposo	Escala a cero. Sin coste residual tras cierre	Tier gratuito, pero pausa el proyecto por inactividad
Riesgo para este proyecto	Ninguno conocido relacionado con el ciclo de vida	Podría valorarse la pausa por inactividad, pero impacto menor.
Escalado en picos	Automático sin intervención manual	Réplicas y failover disponibles en tiers de pago
Backups	Automáticos en tiers de pago	Disponibles en tiers superiores
Audiencia distribuida (ES + LATAM)	A evaluar	Múltiples instancias y réplicas por región

Decisión pendiente Con Equipo	Comparar el coste real de cada plataforma durante exactamente 2 meses con el volumen estimado. Priorizar integridad de datos garantizada sobre ahorro marginal. Backups automáticos diarios son requisito no negociable en cualquier opción elegida.

3.3. Esquema de base de datos

El esquema está diseñado para minimizar la complejidad y maximizar el rendimiento en operaciones de escritura masiva y concurrente.

Tablas de votación

Una tabla por boxeador. Cada tabla contiene únicamente dos campos:

- votos_favor: contador de votos a favor.
- votos_totales: contador de votos totales (equivalente matemático para calcular porcentaje).

No se añaden índices porque cada tabla tendrá muy pocas filas, las consultas son directas sobre una fila concreta, y los índices supondrían un leve overhead sin beneficio real a esta escala.

Tabla de usuarios (decisión pendiente)

Almacenaría un booleano por combate por usuario para garantizar integridad del voto ante borrado de cookie. El problema de escala es significativo:

- 5 millones de usuarios × 7 combates = 35 millones de filas.
- Las escrituras concurrentes masivas tienen el mismo problema de contención que los votos.
- Sin Redis aplicado también a esta tabla, el problema persiste.

Recomendación: no implementar salvo que la integridad del voto individual sea declarada requisito crítico por el equipo, en cuyo caso debe aplicarse la misma estrategia de mitigación de contención que para los votos.

4. Estrategia de concurrencia y caché

Se definen tres niveles de complejidad según el volumen estimado de escrituras concurrentes. Debería de valorarse el nivel según subfase de producción.

4.1. Niveles de complejidad

Nivel	Condición	Solución
Nivel 1	< 500 votos concurrentes garantizado	Solo base de datos. El lock de UPDATE no tiene impacto apreciable. No aplica a este proyecto.
Nivel 2	> 500 votos concurrentes posibles	Counter sharding: múltiples filas por boxeador, actualización de fila aleatoria. Viable como solución intermedia.
Nivel 3 (este proyecto)	> 500 votos concurrentes confirmado	Redis con operaciones atómicas INCR. Sin locks, consistencia garantizada.

4.2. Arquitectura con Redis (Nivel 3)

El flujo de un voto en producción es el siguiente:

1. El usuario emite su voto desde el frontend.
2. El servidor realiza una escritura atómica en Redis mediante INCR. Respuesta inmediata al cliente.
3. Un cronjob en Vercel sincroniza periódicamente el estado de Redis a PostgreSQL (cada N votos o cada X segundos).
4. PostgreSQL actúa como fuente de verdad (source of truth). Redis es la capa de escritura rápida.

Lectura de porcentajes

En condiciones normales, la lectura se realiza siempre desde Redis. Si Redis cae o se reinicia, el sistema aplica un fallback automático a PostgreSQL con warm-up obligatorio: al arrancar, carga el estado actual de la DB en Redis antes de aceptar escrituras.

Warm-up crítico	Durante su ejecución, mostrar porcentajes a cero tras un reinicio de Redis sería un fallo visible y grave. El warm-up desde DB debe ejecutarse antes de aceptar cualquier voto.

Optimistic update en frontend

El cliente actualiza el porcentaje localmente en el momento del voto, sin esperar confirmación del servidor. Cuando la respuesta llega ya coincide con lo mostrado en pantalla. Esto elimina la percepción de latencia independientemente del tiempo de respuesta real del servidor.

4.3. Decisión sobre activación de Redis

Modelo de coste Redis	Estrategia recomendada
Coste mensual fijo	Activar el último mes completo antes del evento para amortizar el coste.
Coste on-demand (Valkey u otros)	Activar únicamente la semana del evento.

Decisión pendiente	*Comparar Valkey y otras opciones on-demand con Redis fijo para determinar cuándo activar la capa de caché.*

Lógicamente, si se toma la decisión de tomar coste mensual fijo, deberá aplicarse lo antes posible previo inicio del evento, sino estaríamos infrautilizando lo pagado.

5. Autenticación y sistema anti-doble-voto

5.1. Decisión: sin OAuth de Twitch

La integración de login con Twitch fue evaluada y descartada. Los motivos son:

- Añade fricción al proceso de voto, reduciendo la tasa de participación.
- Requiere gestión de sesiones adicional.
- No aporta beneficio funcional más allá de la coherencia temática con el evento.

5.2. Estrategia a valorar: cookie HttpOnly con firma HMAC

En el primer voto del usuario, el servidor genera una firma HMAC del fingerprint del dispositivo. El fingerprint se calcula combinando IP, user-agent y timestamp del primer voto, hasheados con una clave secreta del servidor. Esta firma se devuelve como cookie HttpOnly con expiración larga.

En cada voto posterior, el servidor verifica la firma HMAC sin consultar ninguna base de datos ni Redis. El coste operativo es prácticamente cero.

Valoración de la solución

Aspecto	Ventajas	Limitaciones
Participación	Sin registro previo. Máxima tasa de participación.	—
Privacidad / RGPD	Sin almacenamiento de datos personales identificables. Cumplimiento trivial de RGPD y LOPD.	—
Rendimiento	Sin escrituras adicionales en DB ni Redis.	—
Integridad del voto	Efectiva para el caso general.	Si el usuario borra las cookies, puede votar de nuevo.
Impacto estadístico	—	En proporción a 5M votantes, el impacto es ruido estadístico. Riesgo conocido y aceptado.

El fingerprint se almacena únicamente en el frontend (cookie firmada), nunca en base de datos. Esto es lo que diferencia esta solución de la tabla de usuarios y lo que elimina el coste de escritura asociado.

En caso de no implantación: Podríamos almacenar como se ha planteado anteriormente los usuarios, mediante una tabla con sus votos y no permitir el voto doble, sí el cambio de voto. Simplemente, se plantea esta solución para motivar un mayor número de usuarios siempre y cuando se valore el impacto de dicho riesgo conocido.

6. Cierre automático de votaciones

6.1. Mecanismo: cronjob en Vercel

El cierre de votación se implementa mediante un cronjob en Vercel programado a hora fija predefinida, cinco minutos antes del inicio de cada combate según el horario oficial del evento.

6.2. Justificación frente a alternativas

Opción	Evaluación
Cronjob en Vercel (elegida)	Fiable, predecible y auditable. Puede realizarse mediante una instancia programada, mediante github API o de manera manual por cualquiera de los integrantes del proyecto.
Twitch EventSub API	Descartada: no tiene granularidad suficiente para detectar el inicio exacto de un combate concreto. Requeriría lógica adicional de interpretación dependiente de una API externa.

Gestión del retraso en el evento

El único riesgo conocido de esta solución es el posible retraso del evento real respecto al horario oficial, habitual en este tipo de veladas. Sin embargo, este riesgo es auditable y gestionable manualmente si fuera necesario: basta con intervenir en el cronjob o activar el cierre de forma manual desde el panel de Vercel.

La solución simple y fiable es preferible a una solución técnicamente más elaborada cuando el contexto lo justifica y el riesgo residual es manejable.

7. Ranking post-evento

7.1. Concepto

Al cierre del evento se genera una funcionalidad de ranking personal, similar conceptualmente a Spotify Wrapped. Para cada usuario muestra cuántos de los 7 combates acertó, su posición relativa respecto a la media de la comunidad, y genera una tarjeta o imagen compartible en redes sociales.

El potencial viral de esta funcionalidad es alto durante las horas post-evento, que coinciden con el segundo pico de tráfico más elevado del proyecto.

7.2. Implementación técnica

Al cierre del evento se ejecuta un script (manual o automatizado vía GitHub API) que:

5. Lee el estado final de PostgreSQL.
6. Genera un JSON estático con los resultados definitivos de todos los combates.
7. Publica el JSON en el repositorio, desencadenando un despliegue automático en Vercel.

A partir de ese momento, todas las lecturas de resultados se sirven desde el JSON estático en el edge de Vercel. No se realizan llamadas a base de datos ni a Redis. El coste de lectura es cero y la latencia es mínima. Puede realizarse también de forma manual poco después de la finalización del evento, de guardar el email podría mandarse a los usuarios (teniendo en cuenta que esto aumentaría el coste general de la aplicación; necesita la tabla users y un servicio de mensajería).

Cálculo del resultado individual

El cálculo del acierto de cada usuario se realiza íntegramente en el frontend, comparando el booleano almacenado en la cookie del usuario con el JSON de resultados. No se realizan llamadas al servidor para este cálculo, siempre y cuando no se implemente la tabla de usuarios, aunque sea con token anónimo.

Riesgo del proceso de publicación

Riesgo identificado	Si el deploy del JSON es manual, existe riesgo humano en el momento de mayor presión post-evento. Mitigación recomendada: automatizar el script de generación como cronjob de cierre que realice el commit automáticamente vía GitHub API, con posibilidad de override manual si es necesario.

8. Funcionalidades adicionales propuestas

Funcionalidad	Tipo de cambio	Estado
Cards de combatientes con links a entrenamientos	Frontend. Coste mínimo.	Alta probabilidad. Mejora el engagement pre-evento y genera tráfico orgánico.
Sección de hype con cuenta atrás	Frontend. Reutiliza componentes de ediciones anteriores.	Alta probabilidad de implementación.
FAQ reutilizada de edición anterior	Contenido estático. Coste mínimo.	Directamente trasladable.
Integración Ticketmaster API o entradas.com para entradas	Backend + frontend. Depende de la plataforma de venta de entradas del evento.	Pendiente de confirmación de requisitos.

9. Decisiones pendientes de equipo

Las siguientes decisiones requieren validación formal antes de iniciar el desarrollo de los módulos afectados.

Decisión	Opciones	Criterio de desempate
Plataforma de base de datos	Neon vs Supabase	Coste real comparado a 2 meses con el volumen estimado. Priorizar integridad garantizada.
Tabla de usuarios en DB	Implementar vs no implementar	Solo si la integridad del voto individual es requisito crítico explícito.
Modelo de Redis	On-demand (Valkey u otros) vs mensual fijo	Coste concreto de cada opción para determinar ventana de activación.
Deploy del JSON de resultados	Manual vs automatizado vía GitHub API	Riesgo humano post-evento vs complejidad de automatización.
API de venta de entradas	Ticketmaster vs entradas.com u otra plataforma	Confirmar plataforma del evento antes de iniciar integración.
Funcionalidades nuevas (ranking, cards, hype)	Implementar vs diferir	Todas con coste mínimo. Requieren validación formal del equipo.

10. Resumen de arquitectura del sistema

El siguiente diagrama describe el flujo completo de un voto durante el evento, () implica optatividad, su implementación depende de decisiones de equipo:

Si quieres saber más sobre mí u otros proyectos consulta: https://kokoworks.es