POLLNOW

I.E.S. Hermanos Machado

Ciclo Formativo de Grado Superior en Desarrollo de Aplicaciones Web

Proyecto Intermodular

Pollnow

Autor: Raimundo Palma Méndez
Tutor: David de Vega Rodríguez

Dos Hermanas (Sevilla), Mayo - 2026

Proyecto Intermodular
Ciclo Formativo de Grado Superior en Desarrollo de Aplicaciones Web

Título: POLLNOW

Mayo - 2026

Autor: Raimundo Palma Méndez

Tutor: David de Vega Rodríguez
I.E.S. Hermanos Machado

Índice

Descripción del problema

Actualmente la organización de eventos digitales y entregas de premios se ha popularizado enormemente gracias al auge de los creadores de contenido y las comunidades en internet. Sin embargo, existe una carencia tecnológica significativa cuando se intenta profesionalizar la gestión de estos eventos sin la necesidad de recurrir a desarrollos a medida extremadamente caros o complejos. El problema en cuestión es que las herramientas de encuestas convencionales no están diseñadas para soportar la jerarquía lógica que requiere una gala de premios profesional.

Esta falta de especialización en el mercado obliga a los usuarios a utilizar sistemas incompletos que no garantizan la integridad del proceso. Por un lado, las plataformas gratuitas de formularios no tienen lo necesario para prevenir el fraude en votaciones masivas, y por otro, no ofrecen una experiencia de usuario inmersiva que refleje el prestigio de una gala. Además, no existe ninguna plataforma de este tipo que contenga tanto accesibilidad como seguridad para sus usuarios, implementando votaciones anónimas que sean imposibles de duplicar y cookies seguras. Y por último, el mercado actual no ayuda a los usuarios que necesitan un entorno de gestión que incluya moderación, analíticas en tiempo real y un sistema de soporte integrado.

Estudio de mercado

El análisis del entorno actual nos muestra un mercado dividido entre soluciones extremadamente simples y plataformas corporativas de alto coste. En el primer grupo encontramos herramientas de uso general (como Google Forms) que si bien son accesibles y gratuitas, resultan insuficientes para gestionar un modelo de datos relacional complejo donde interactúan eventos, categorías y nominados de forma coherente. Estas herramientas tampoco ofrecen un panel de administración orientado a la moderación de contenido, lo que limita su uso en entornos profesionales o de alta visibilidad pública.

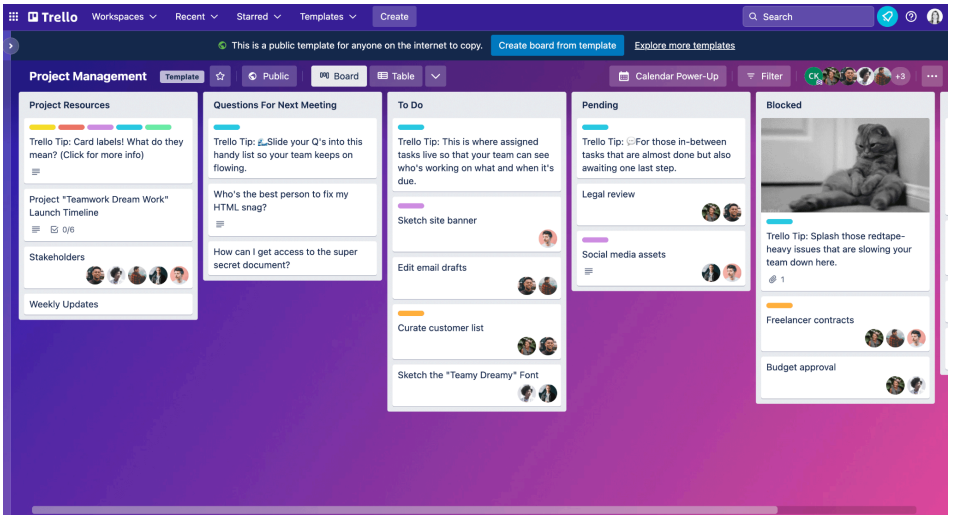
Pollnow se posiciona como una solución innovadora a este problema tan de nicho bajo un modelo SaaS (Software as a Service). Integrando tecnologías modernas como Next.js y un sistema de pagos flexible a través de Stripe, la plataforma permite democratizar el acceso a herramientas de nivel profesional a un bajo coste.

Herramienta de gestión del proyecto

Para la planificación y la gestión del proyecto he elegido Trello, una plataforma de gestión de tareas basada en el método Kanban. Nos permite organizar tareas mediante tableros y tarjetas, esta herramienta me ha ayudado mucho en la planificación y en la asignación de tareas de manera sencilla e intuitiva.

Características principales

- **Tableros, listas y tarjetas** para organizar tareas de forma visual.
- **Sistema Kanban** para representar fases del trabajo.
- **Etiquetas personalizables** para clasificar tareas por prioridad o categoría.
- **Checklists** para desglosar subtareas dentro de una tarea principal.
- **Fechas de entrega y recordatorios** para cumplir plazos establecidos.
- **Comentarios y archivos adjuntos** para documentar información relevante.
- **Sincronización en tiempo real** permitiendo seguimiento continuo por parte del profesor.
- **Acceso multiplataforma** (web o móvil).
- **Integraciones con otras herramientas** como GitHub, Google Drive o Slack.
- **Historial básico de actividad** para monitorizar cambios y avances.



Fuente de la imagen: Meistertask

Justificación de la elección

Para la gestión y organización del proyecto he elegido Trello por todo lo mencionado anteriormente y por los siguientes motivos:

- **Simplicidad y usabilidad:** La interfaz es clara y accesible lo que me permite gestionar el proyecto de una manera super sencilla.
- **Orientación Kanban:** Perfecto para seguir el avance de tareas del proyecto.
- **Flexibilidad:** Se adapta tanto a proyectos individuales como a proyectos en equipo.
- **Organización clara de etapas y tareas.**
- **Portabilidad:** Disponible tanto en ordenador como en móvil (para cuando estoy fuera de casa).

Configuración inicial del proyecto

- **TO-DO**……(Tareas pendientes)
- **WORKING ON**…….(Tareas en proceso)
- **NEXT STEP**……..…(Para tareas ya iniciadas pero pendientes de actualización)
- **ISSUES**……..……..…(Problemas por solucionar)
- **DONE**……..……..(Completado)
- **IDEAS**……..……...……..(Nuevas funcionalidades posibles para el proyecto)

Cada tabla tiene sus tareas, y estas a su vez tienen subtareas, fechas límite, comentarios y etiquetas por categoría (backend, frontend, diseño, documentación...)

Objetivos del proyecto

Objetivo general

El objetivo principal de este proyecto es desarrollar y desplegar una plataforma web bajo un modelo de software como servicio completa llamada Pollnow, diseñada específicamente para la creación, gestión y análisis de eventos de votación con formato de gala de premios digital, integrando un sistema de suscripción y un panel de administración avanzado para garantizar un ciclo de vida de eventos profesionales y seguros para todos los públicos.

Objetivos específicos

- **Implementar una buena arquitectura:** Utilicé el App Router de Next.js y React Server Components para optimizar el rendimiento y la carga de datos del sistema.

- **Diseñar un modelo de datos relacional robusto:** Usando Prisma creé un esquema capaz de gestionar la persistencia de usuarios, eventos, categorías de votación y participantes de forma eficiente y muy simplificada.

- **Garantizar la integridad de las votaciones:** Desarrollé un sistema de votación anónima que utiliza hashes de identidad y cookies para prevenir votos duplicados sin comprometer la privacidad de los usuarios que voten.

- **Desarrollar un sistema de monetización:** Integré Stripe para gestionar pagos, suscripciones y webhooks que me permiten diferenciar entre rangos de servicio gratuitos y de pago.

- **Proporcionar herramientas de análisis y soporte:** Creé un componente de estadísticas generales para cada evento y un sistema de soporte técnico mediante tickets para mejorar la experiencia de los organizadores.

- **Facilitar la moderación de contenidos:** Implementé un panel de administración propio que permite a los moderadores y administradores aprobar o rechazar eventos y gestionar absolutamente todo el contenido de la web.

Objetivos de negocio

OBJ-001: Gestión de usuarios del sistema

Campo	Valor
Código	OBJ-001
Nombre	Gestión de usuarios del sistema
Versión	1.0
Autor / Fuente	Raimundo Palma Méndez
Descripción	El sistema deberá permitir registrar, gestionar y autenticar a los usuarios para que puedan interactuar con la plataforma, crear eventos con categorias y nominados y personalizar sus datos.
Importancia	Alta
Estado	Aprobado

OBJ-002: Gestión de tickets

Campo	Valor
Código	OBJ-002
Nombre	Gestión de tickets
Versión	1.0
Autor / Fuente	Raimundo Palma Méndez
Descripción	El sistema deberá permitir a los usuarios abrir tickets de soporte y los administradores podrán responder a los chats.
Importancia	Alta
Estado	Aprobado

OBJ-003: Gestión de solicitudes

Campo	Valor
Código	OBJ-003
Nombre	Gestión de solicitudes
Versión	1.0
Autor / Fuente	Raimundo Palma Méndez
Descripción	El sistema permitirá a los administradores aceptar o cancelar solicitudes de publicación de eventos, al igual que revisar reportes de otros usuarios hacia otros eventos.
Importancia	Media
Estado	Aprobado

OBJ-004: Gestión de eventos

Campo	Valor
Código	OBJ-004
Nombre	Gestión de eventos
Versión	1.0
Autor / Fuente	Raimundo Palma Méndez
Descripción	El sistema deberá permitir que los usuarios registrados puedan crear sus propios eventos, modificando categorías y nominados.
Importancia	Alta
Estado	Aprobado

OBJ-005: Sistema de búsqueda y filtros avanzados

Campo	Valor
Código	OBJ-005
Nombre	Sistema de búsqueda y filtros avanzados
Versión	1.0
Autor / Fuente	Raimundo Palma Méndez
Descripción	El sistema permitirá filtrar y buscar eventos en base al título, etiquetas, categorías en específico, eventos populares…
Importancia	Muy alta
Estado	Aprobado

OBJ-006: Sistema de exploración de eventos

Campo	Valor
Código	OBJ-006
Nombre	Sistema de exploración de eventos
Versión	1.0
Autor / Fuente	Raimundo Palma Méndez
Descripción	El sistema deberá mostrar una sección pública con todos los eventos publicados por los demás usuarios.
Importancia	Media
Estado	Aprobado

OBJ-007: Seguridad y autenticación

Campo	Valor
Código	OBJ-007
Nombre	Seguridad y autenticación
Versión	1.0
Autor / Fuente	Raimundo Palma Méndez
Descripción	El sistema deberá implementar autenticación segura y control de accesos para proteger datos de usuarios y operaciones críticas.
Importancia	Alta
Estado	Aprobado

OBJ-008: Optimización y rendimiento de la plataforma

Campo	Valor
Código	OBJ-008
Nombre	Optimización y rendimiento de la plataforma
Versión	1.0
Autor / Fuente	Raimundo Palma Méndez
Descripción	El sistema deberá proporcionar tiempos de carga bajos, experiencia fluida y buena indexación en diferentes dispositivos.
Importancia	Media
Estado	Aprobado

OBJ-009: Persistencia y gestión de datos

Campo	Valor
Código	OBJ-009
Nombre	Persistencia y gestión de datos
Versión	1.0
Autor / Fuente	Raimundo Palma Méndez
Descripción	El sistema deberá almacenar y mantener información de usuarios, eventos, categorías y nominados de manera estructurada y escalable.
Importancia	Alta
Estado	Aprobado

OBJ-010: Despliegue en producción

Campo	Valor
Código	OBJ-010
Nombre	Despliegue en producción
Versión	1.0
Autor / Fuente	Raimundo Palma Méndez
Descripción	El sistema deberá ser desplegado y accesible en producción mediante servicios cloud (Vercel) garantizando su disponibilidad en internet.
Importancia	Media
Estado	Aprobado

Requisitos Funcionales (RF)

RF-01: Gestión de usuarios

Campo	Valor
Código	RF-01
Versión	1.0
Objetivos asociados	OBJ-001, OBJ-007
Descripción	El sistema deberá permitir: Registrar usuarios, Iniciar sesión, Cerrar sesión, Editar perfil…
Actores	Usuario registrado

RF-02: Gestión de eventos de votaciones

Campo	Valor
Código	RF-02
Versión	1.0
Objetivos asociados	OBJ-002, OBJ-003, OBJ-009
Descripción	El sistema deberá permitir: Crear y Editar eventos, Añadir categorías y participantes a cada evento, generar imágenes de participantes con IA (si es premium), Revisar estadísticas avanzadas de cada evento, Colaborar en equipo en un mismo evento con diferentes usuarios…
Actores	Admin, Usuario

RF-03: Gestión de categorías en evento

Campo	Valor
Código	RF-03
Versión	1.0
Objetivos asociados	OBJ-003, OBJ-001
Descripción	El sistema deberá permitir que el propietario de un evento añada nuevas categorías, editarlas, y eliminarlas a su gusto.
Actores	Autor del evento (Usuario), Admin

RF-04: Gestión de nominados en evento

Campo	Valor
Código	RF-04
Versión	1.0
Objetivos asociados	OBJ-004, OBJ-001
Descripción	El sistema deberá permitir que el propietario de un evento añada nuevos nominados, editarlos, crear imágenes con IA y eliminarlos a su gusto.
Actores	Usuario registrado

RF-05: Sistema de búsqueda y filtros avanzados

Campo	Valor
Código	RF-05
Versión	1.0
Objetivos asociados	OBJ-005
Descripción	El sistema deberá permitir la búsqueda de eventos aplicando filtros como etiquetas, temas, o categorías…
Actores	Usuario anónimo, Usuario registrado

RF-06: Sistema de exploración de eventos

Campo	Valor
Código	RF-06
Versión	1.0
Objetivos asociados	OBJ-006
Descripción	El sistema deberá mostrar eventos públicos en una página general de eventos publicados.
Actores	Usuario

RF-07: Persistencia y almacenamiento

Campo	Valor
Código	RF-07
Versión	1.0
Objetivos asociados	OBJ-009
Descripción	El sistema deberá almacenar y gestionar datos en una base persistente para usuarios, eventos, categorías, nominados...
Actores	Sistema

RF-08: Autenticación y control de acceso

Campo	Valor
Código	RF-08
Versión	1.0
Objetivos asociados	OBJ-007
Descripción	El sistema deberá permitir autenticación mediante: Email y contraseña, Conexiones seguras de terceros como Google.
Actores	Usuario registrado

Requisitos No Funcionales (RNF)

RNF-01: Disponibilidad

Campo	Valor
Código	RNF-01
Versión	1.0
Objetivos asociados	OBJ-010
Descripción	El sistema deberá estar disponible públicamente en un entorno web. Pudiendo acceder gracias al dominio *pollnow.es*

RNF-02: Seguridad

Campo	Valor
Código	RNF-02
Versión	1.0
Objetivos asociados	OBJ-007
Descripción	El sistema deberá proteger datos sensibles mediante cifrado y control de acceso.

RNF-03: Rendimiento

Campo	Valor
Código	RNF-03
Versión	1.0
Objetivos asociados	OBJ-008
Descripción	La plataforma deberá proporcionar tiempos de carga bajos y respuesta fluida incluso con filtros activos.

RNF-04: Usabilidad

Campo	Valor
Código	RNF-04
Versión	1.0
Objetivos asociados	OBJ-005
Descripción	La interfaz deberá ser intuitiva para usuarios novatos o sin experiencia previa en la creación de eventos tipo gala de premios.

RNF-05: Escalabilidad

Campo	Valor
Código	RNF-05
Versión	1.0
Objetivos asociados	OBJ-009
Descripción	El sistema deberá poder ampliarse gradualmente sin reestructurar su arquitectura.

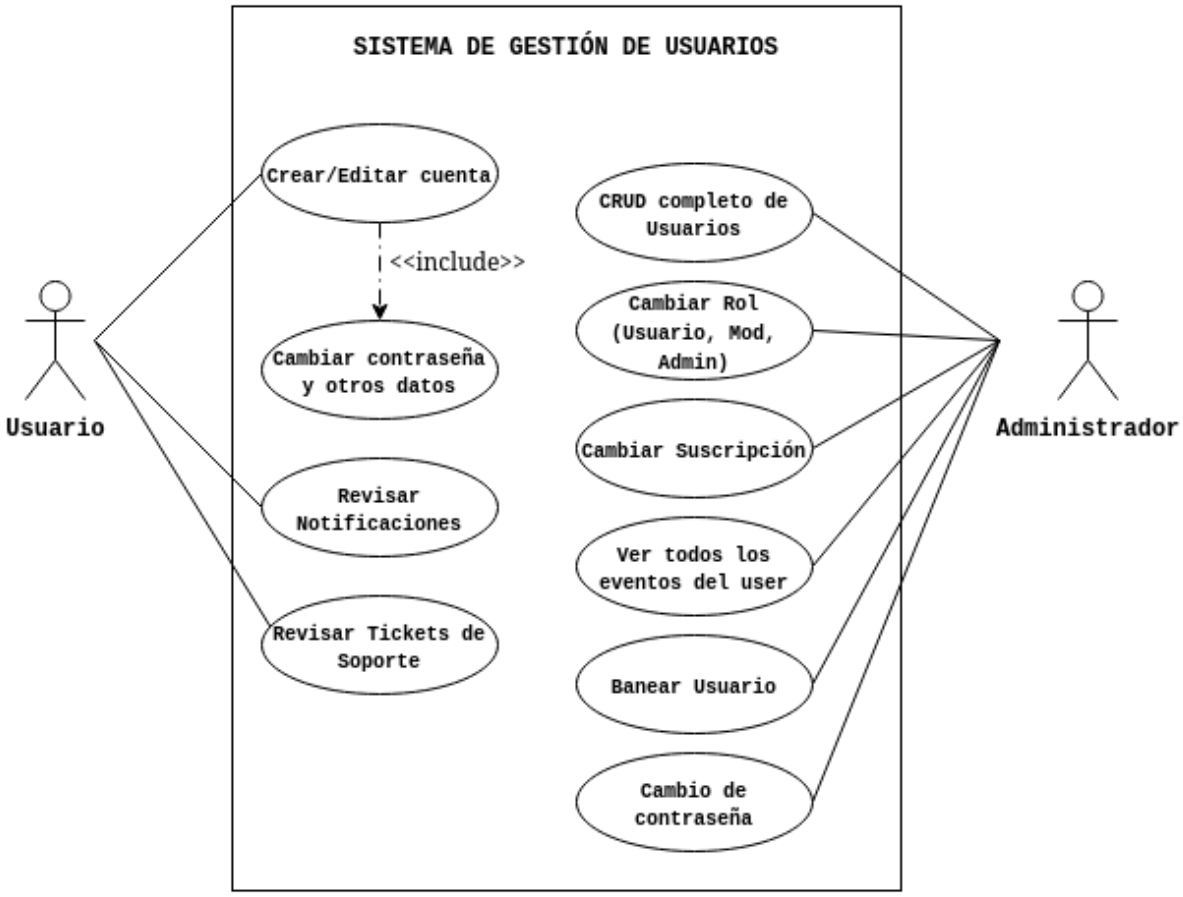
Matriz de trazabilidad

RF / RNF y OBJ	001	002	003	004	005	006	007	008	009	010
RF-01	■						■			
RF-02		■	■						■	
RF-03			■							
RF-04	■			■						
RF-05					■					
RF-06						■				
RF-07		■							■	
RF-08	■						■			
RNF-01										■
RNF-02							■			
RNF-03								■		
RNF-04					■					
RNF-05		■							■	

Casos de uso

- **Sistema de gestión de usuarios**

Los usuarios en este caso pueden crear o editar su cuenta, cambiar la contraseña, revisar notificaciones del sistema y revisar los tickets de soporte. Mientras que los administradores pueden acceder a un CRUD completo de los usuarios del sistema, cambiar de roles a los usuarios, cambiar la suscripción, ver todos los eventos creados por cada usuario, banear a los usuarios y cambiarles la contraseña.



- **Sistema de gestión de eventos de votación**

Los usuarios pueden crear, editar o eliminar sus eventos (también pueden editar eventos en los que colabore de otros usuarios si el creador del evento lo permite), puede solicitar la publicación de sus eventos y reportar eventos ajenos para avisar a los moderadores de posibles incidencias. Por otro lado los administradores pueden acceder a un CRUD completo de los eventos registrados en el sistema, revisar reportes creados por otros usuarios, sancionar a usuarios, eliminar eventos, revisar solicitudes de publicación de eventos y aprobar o denegar dichas solicitudes.

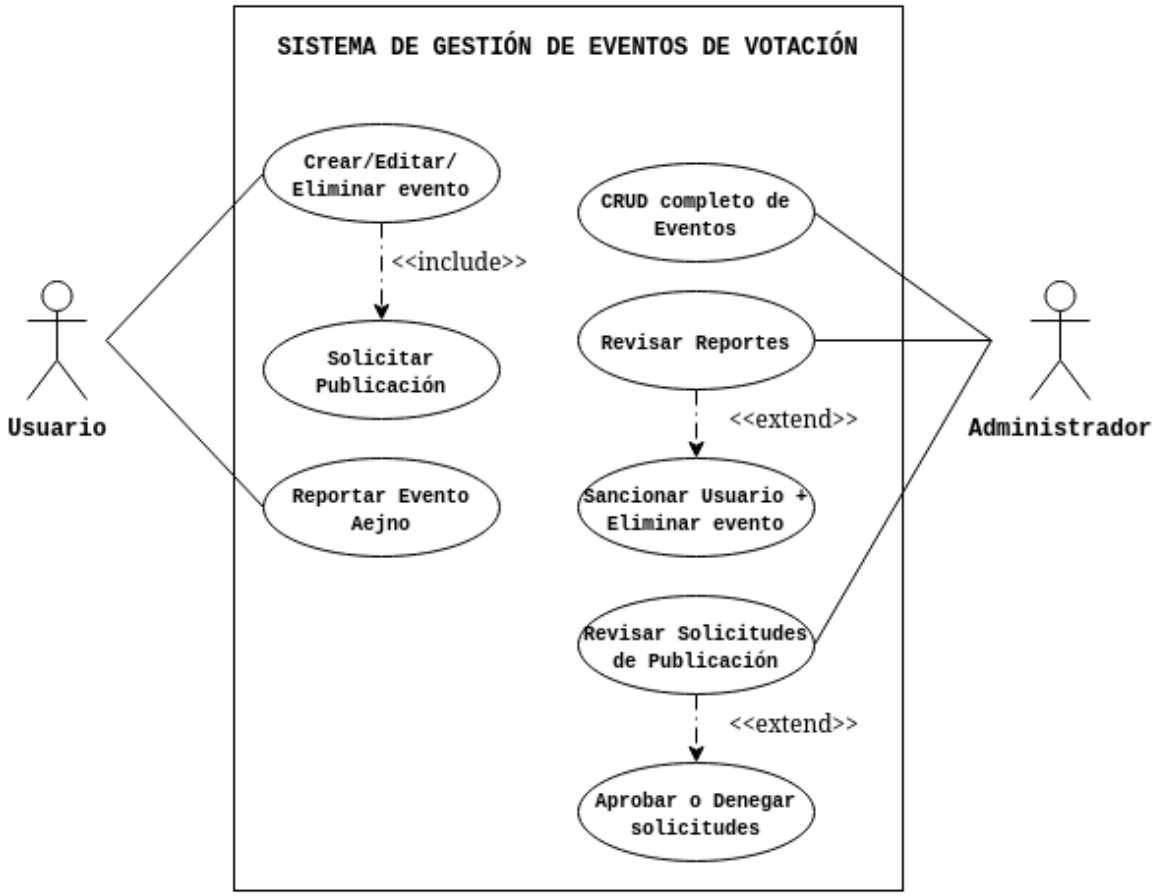


Diagrama de clases

Este diagrama representa la estructura lógica del sistema de Pollnow. Siguiendo el flujo de diseño lógico, este diagrama traslada los casos de uso y la interfaz de usuario hacia una arquitectura de software basada en entidades, atributos y sus relaciones.

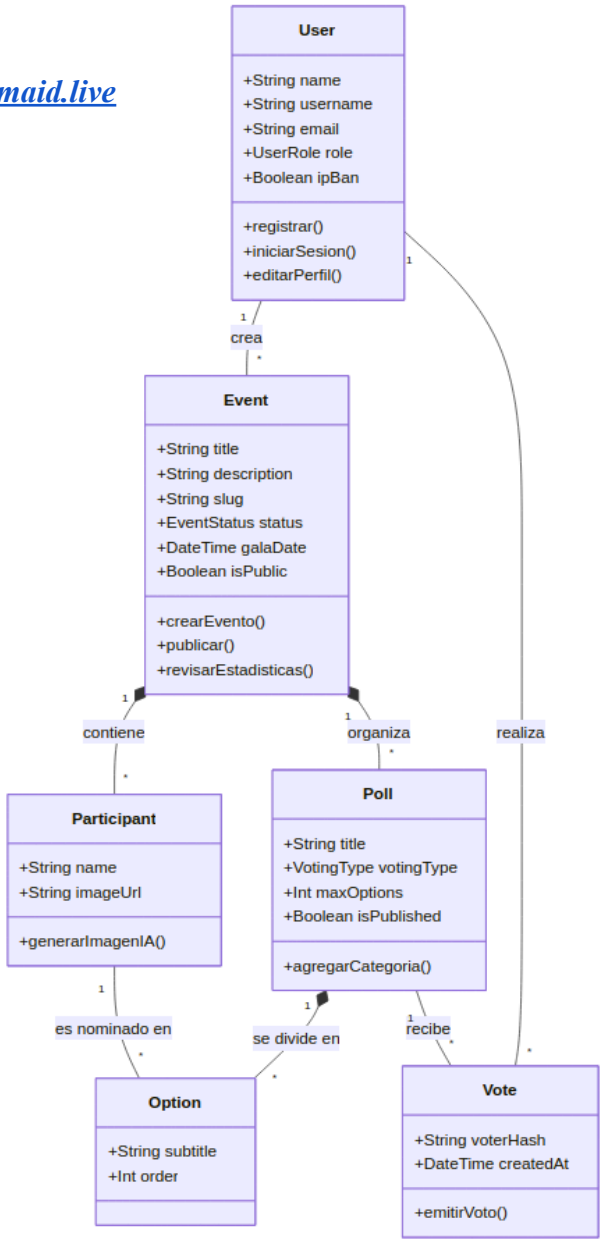
Representación visual del sistema

El siguiente diagrama* detalla las entidades principales del proyecto, incluyendo la gestión de usuarios, la creación de eventos de gala y el sistema de participación.

Ver diagrama completo en *Mermaid.live*

*Esto es un diagrama simplificado, en la realidad para este proyecto es mucho más grande y no cabe en la página.

El diagrama completo puede verse en el enlace compartido arriba de esta aclaración.



Descripción de las clases principales

- **User (Usuario):** Gestiona la autenticación y el control de acceso (RF-01, RF-08). Incluye atributos para el control de seguridad como *ipBan* y *role* (User, Moderator, Admin).

- **Event (Evento):** Es lo más importante del sistema (RF-02). Administra el estado de la gala (*DRAFT*, *PENDING*, *APPROVED*) y su visibilidad pública.

- **Poll (Encuesta/Categoría):** Representa las diferentes categorías o premios dentro de una gala (RF-03). Permite configurar el tipo de votación (simple, múltiple limitado o ilimitado).

- **Participant (Participante/Nominado):** Almacena la información de los participantes que pueden ser votados. Incluye la lógica para la generación de imágenes mediante IA (RF-04).

- **Vote (Voto):** Registra la participación. Para garantizar la integridad se relaciona con un *voterHash* (para usuarios anónimos) o un *userId* (para usuarios registrados).

Relaciones y lógica de negocio

Con este diseño implementé relaciones de composición (rombo negro) entre el *Event* y sus *Polls* y *Participants*, ya que estos no tienen sentido en el sistema si el evento principal es eliminado.

La relación entre *User* y *Event* es una asociación simple de 1 a N, permitiendo que un usuario gestione múltiples eventos de votación. Por último, la clase *Option* actúa como el intermediario que permite vincular a un nominado (*Participant)* específico con una categoría (*Poll*) concreta, manteniendo la trazabilidad de los nominados.

Modelo relacional

Mapeo relacional

He utilizado un enfoque de Mapeo Objeto Relacional (ORM) usando Prisma. Siguiendo las reglas de traducción técnica, he aplicado las siguientes transformaciones:

- **Tablas:** Cada modelo definido en el archivo "schema.prisma" se traduce en una tabla física en SQL.
- **Columnas y tipos:** Los atributos se mapean a tipos de datos específicos de PostgreSQL (por ejemplo, en vez de "String" decimos "VARCHAR")
- **Identificadores:** Se utilizan UUID como claves primarias (PK).

Transformación del dominio de datos

Lo más importante (usuarios, eventos, participantes y demás) se ha estructurado en tablas independientes y relacionadas según su jerarquía lógica:

1. **Entidades independientes:** Las tablas *User* y *Account* son los usuarios y sus credenciales, utilizando enums para definir el *UserRole* (ADMIN, MODERATOR, USER).

2. **Entidades dependientes:** La tabla *Event* depende de *User* (relación 1:N). El UUID del creador se relaciona con la tabla evento como clave foránea (FK).
    - La tabla *Participant* y *Poll* dependen directamente de *Event*. Tendrán eventId como atributo.

3. **Resolución de la votación:** La tabla *Option* actúa como la tabla intermedia que vincula a un participante con una categoría específica (*Poll*). Finalmente, la tabla *Vote* registra la acción, vinculando el UUID del usuario (si está logueado) o un hash único para votos anónimos.

Esquema SQL simplificado

```sql
-- Definición de Enums
CREATE TYPE "UserRole" AS ENUM ('USER', 'MODERATOR', 'ADMIN');
CREATE TYPE "EventStatus" AS ENUM ('DRAFT', 'PENDING', 'APPROVED', 'DENIED');

-- Tabla de Usuarios
CREATE TABLE "User" (
    "id" TEXT PRIMARY KEY,
    "name" TEXT NOT NULL,
    "email" TEXT UNIQUE NOT NULL,
    "role" "UserRole" DEFAULT 'USER',
    "createdAt" TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Tabla de Eventos
CREATE TABLE "Event" (
    "id" TEXT PRIMARY KEY,
    "userId" TEXT NOT NULL,
    "title" VARCHAR(100) NOT NULL,
    "status" "EventStatus" DEFAULT 'DRAFT',
    "galaDate" TIMESTAMP NOT NULL,
    FOREIGN KEY ("userId") REFERENCES "User"("id") ON DELETE CASCADE
);

-- Tabla de Votos
CREATE TABLE "Vote" (
    "id" TEXT PRIMARY KEY,
    "pollId" TEXT NOT NULL,
    "voterHash" TEXT NOT NULL,
    "userId" TEXT,
    FOREIGN KEY ("pollId") REFERENCES "Poll"("id") ON DELETE CASCADE,
    FOREIGN KEY ("userId") REFERENCES "User"("id") ON DELETE SET NULL
);
```

Integridad y restricciones

- **ON DELETE CASCADE:** Si un usuario borra un evento, todas las categorías, participantes y votos asociados se eliminan automáticamente, evitando datos huérfanos.
- **Unique Constraints:** Se garantiza que un usuario no pueda votar dos veces en la misma categoría mediante una restricción única en la combinación de pollId y voterHash.

Código completo de Prisma

A continuación comparto todo el código de mi esquema de Prisma (que se encuentra en *prisma/schema.prisma*).

```
generator client {
        provider = "prisma-client-js"
}

datasource db {
        provider  = "postgresql"
        url     = env("DATABASE_URL")
        directUrl = env("DATABASE_URL_UNPOOLED")
}

// ===================================
// ENUMS
// ===================================

enum UserRole {
        USER
        MODERATOR
        ADMIN
}

enum EventStatus {
        DRAFT
        PENDING
        APPROVED
        DENIED
}

enum VotingType {
        SINGLE
        MULTIPLE
        LIMITED_MULTIPLE
}

enum ReportReason {
        SPAM
        INAPPROPRIATE_CONTENT
        HARASSMENT
        SCAM
        OTHER
}

enum InvitationStatus {
        PENDING
        ACCEPTED
        REJECTED
}

enum NotificationType {
        SYSTEM
        COLLABORATION
}

enum RaffleStatus {
        ACTIVE
        CLOSED
        WINNER_SELECTED
}
```

```
// ====================================
// MODELOS DE USUARIO Y CUENTA
// ====================================
model User {
        id                      String          @id @default(uuid())
        name                    String
        username                String          @unique
        email                   String          @unique
        emailVerified           DateTime?
        passwordHash            String?
        image                   String?
        role                    UserRole        @default(USER)
        ipBan                   Boolean         @default(false)
        stripeCustomerId        String?         @unique
        stripeSubscriptionId String?           @unique
        stripePriceId           String?
        subscriptionStatus    String?           @default("free")
        subscriptionEndDate   DateTime?
        cancelAtPeriodEnd      Boolean          @default(false)
        welcomeBonusApplied   Boolean          @default(false)
        emailNotifications    Boolean          @default(true)
        emailCollaborations   Boolean          @default(true)
        createdAt               DateTime        @default(now())
        updatedAt               DateTime        @updatedAt
        accounts                Account[]
        events                  Event[]
        sessions                Session[]
        votes                   Vote[]                          @relation(name: "UserVotes")
        notifications           Notification[]                  @relation("AdminNotifications")
        receivedNotifications Notification[]        @relation("UserNotifications")
        supportChats            SupportChat[]                   @relation(name:
"UserSupportChats")
        chatMessages            ChatMessage[]                  @relation(name:
"UserChatMessages")
        moderationLogs          ModerationLog[]                @relation(name: "AdminActions")
        reports                 Report[]                       @relation(name: "UserReports")
        eventLikes              EventLike[]
        eventVotes              EventVote[]
        collaborations          EventCollaborator[]            @relation("UserCollaborations")
        sentInvitations         CollaboratorInvitation[]  @relation("SentInvitations")
        receivedInvitations   CollaboratorInvitation[]  @relation("ReceivedInvitations")
        wonRaffles              Raffle[]                       @relation("RaffleWinner")
}


model Account {
        id              String  @id @default(uuid())
        userId          String
        type            String
        provider        String
        providerAccountId String
        refresh_token   String? @db.Text
        access_token    String? @db.Text
        expires_at      Int?
        token_type      String?
        scope           String?
        id_token        String? @db.Text
        session_state   String?
        user User @relation(fields: [userId], references: [id], onDelete: Cascade)
        @@unique([provider, providerAccountId])
        @@index([userId])
}

model Session {
        id              String    @id @default(uuid())
        sessionToken String    @unique
        userId          String
        expires         DateTime
        user            User     @relation(fields: [userId], references: [id], onDelete:
Cascade)
        @@index([userId])
}
model VerificationToken {
        id      String    @id @default(uuid())
        email   String
        token   String    @unique
```

```
        expires DateTime
        @@unique([email, token])
}

model PasswordResetToken {
        id      String   @id @default(uuid())
        email   String
        token   String   @unique
        expires DateTime

        @@unique([email, token])
}

// ====================================
// MODELOS DE EVENTOS Y VOTACIONES
// ====================================

model Event {
        id              String          @id @default(uuid())
        userId          String
        title           String          @db.VarChar(100)
        description     String          @db.Text
        slug            String          @unique
        accessKey       String          @unique @default(uuid())
        isPublic        Boolean         @default(false)
        tags            String[]
        galaDate        DateTime
        isAnonymousVoting Boolean      @default(true)
        createdAt       DateTime        @default(now())
        updatedAt       DateTime        @updatedAt
        status          EventStatus     @default(DRAFT)
        reviewReason    String?
        defaultCanEditSettings   Boolean @default(false)
        defaultCanRegenerateKey  Boolean @default(false)
        defaultCanDeleteEvent Boolean @default(false)
        defaultCanManageNominees Boolean @default(true)
        defaultCanManagePolls Boolean @default(true)
        defaultCanViewStats   Boolean @default(true)
        user            User            @relation(fields: [userId], references: [id])
        participants    Participant[]
        polls           Poll[]
        reports         Report[]
        moderationLogs ModerationLog[]
        likes           EventLike[]
        eventVotes      EventVote[]
        collaborators   EventCollaborator[]
        invitations     CollaboratorInvitation[]

        @@index([userId])
        @@index([slug])
        @@index([status])
        @@index([createdAt])
}

model Participant {
        id              String    @id @default(uuid())
        name            String    @db.VarChar(100)
        imageUrl        String?   @db.Text
        eventId         String
        createdAt    DateTime @default(now())
        nominations Option[]
        event           Event   @relation(fields: [eventId], references: [id], onDelete:
Cascade)

        @@index([eventId])
        @@index([createdAt])
}


model Poll {
        id              String @id @default(uuid())
        eventId         String
        title           String @db.VarChar(100)
        description    String? @db.Text
        votingType      VotingType @default(SINGLE)
        maxOptions      Int     @default(2)
```

```
        isPublished    Boolean @default(false)
        order          Int     @default(0)
        createdAt      DateTime    @default(now())
        updatedAt      DateTime    @updatedAt
        event          Event   @relation(fields: [eventId], references: [id], onDelete:
Cascade)
        options        Option[]
        votes          Vote[]

        @@index([eventId])
}

model Option {
        id             String          @id @default(uuid())
        pollId         String
        subtitle       String?         @db.VarChar(100)
        participantId  String
        order          Int             @default(0)
        participant    Participant    @relation(fields: [participantId], references: [id], onDelete:
Cascade)
        poll           Poll          @relation(fields: [pollId], references: [id], onDelete: Cascade)
        votes          VoteOption[]

        @@unique([pollId, participantId])
}

model Vote {
        id             String          @id @default(uuid())
        pollId         String
        createdAt      DateTime        @default(now())
        voterHash      String
        userId         String?
        user           User?           @relation(name: "UserVotes", fields: [userId], references: [id])
        poll           Poll            @relation(fields: [pollId], references: [id], onDelete: Cascade)
        voteOptions    VoteOption[]

        @@unique([pollId, voterHash])
        @@index([userId])
        @@index([createdAt])
}

model VoteOption {
        id      String @id @default(uuid())
        voteId   String
        optionId String
        option   Option @relation(fields: [optionId], references: [id], onDelete: Cascade)
        vote    Vote   @relation(fields: [voteId], references: [id], onDelete: Cascade)

        @@unique([voteId, optionId])
}
```

```
// ====================================
// MODELOS DE MODERACIÓN Y SOPORTE
// ====================================

model Report {
        id              String          @id @default(uuid())
        reporterId      String
        eventId         String
        reason          ReportReason
        details         String          @db.Text
        isReviewed      Boolean         @default(false)
        createdAt       DateTime        @default(now())
        reviewedAt      DateTime?
        reporter        User            @relation(name: "UserReports", fields: [reporterId], references:
[id])
        event           Event           @relation(fields: [eventId], references: [id])

        @@index([eventId])
        @@index([reporterId])
        @@index([isReviewed])
        @@index([createdAt])
        @@index([isReviewed, createdAt])
}

model ModerationLog {
        id              String    @id @default(uuid())
        adminId         String
        actionType      String
        targetType      String
        targetId        String
        details         String?   @db.Text
        createdAt       DateTime  @default(now())
        admin           User      @relation(name: "AdminActions", fields: [adminId], references: [id])
        eventId         String?
        event           Event?    @relation(fields: [eventId], references: [id])

        @@index([adminId])
        @@index([targetId])
        @@index([eventId])
        @@index([createdAt])
        @@index([actionType])
}

model Notification {
        id              String          @id @default(uuid())
        message         String
        link            String?
        isRead          Boolean         @default(false)
        createdAt       DateTime        @default(now())
        adminUserId   String
        userId          String?
        type            NotificationType @default(SYSTEM)
        invitationId String?
        admin   User   @relation("AdminNotifications", fields: [adminUserId], references:
[id])
        targetUser User? @relation("UserNotifications", fields: [userId], references: [id])

        @@index([adminUserId])
        @@index([userId])
        @@index([isRead])
        @@index([createdAt])
        @@index([type])
}
```

```
// ====================================
// MODELOS DE COLABORACIÓN
// ====================================

model EventCollaborator {
        id      String   @id @default(uuid())
        eventId   String
        userId String
        createdAt DateTime @default(now())
        canEditSettings   Boolean?
        canRegenerateKey  Boolean?
        canDeleteEvent Boolean?
        canManageNominees Boolean?
        canManagePolls Boolean?
        canViewStats   Boolean?
        event Event @relation(fields: [eventId], references: [id], onDelete: Cascade)
        user  User  @relation("UserCollaborations", fields: [userId], references: [id], onDelete: Cascade)

        @@unique([eventId, userId])
        @@index([eventId])
        @@index([userId])
}

model CollaboratorInvitation {
        id              String          @id @default(uuid())
        eventId         String
        invitedById    String
        invitedUserId String
        status          InvitationStatus @default(PENDING)
        createdAt       DateTime        @default(now())
        respondedAt    DateTime?
        event           Event @relation(fields: [eventId], references: [id], onDelete:
Cascade)
        invitedBy    User @relation("SentInvitations", fields: [invitedById], references:
[id])
        invitedUser User @relation("ReceivedInvitations", fields: [invitedUserId], references: [id])

        @@unique([eventId, invitedUserId])
        @@index([eventId])
        @@index([invitedUserId])
        @@index([status])
}

model SupportChat {
        id              String          @id @default(uuid())
        userId          String
        adminId         String?
        isClosed        Boolean         @default(false)
        createdAt       DateTime        @default(now())
        lastMessageAt  DateTime        @updatedAt
        user     User            @relation(name: "UserSupportChats", fields: [userId], references: [id])
        messages   ChatMessage[]

        @@index([userId])
        @@index([isClosed])
        @@index([lastMessageAt])
}

model ChatMessage {
        id      String   @id @default(uuid())
        chatId  String
        senderId   String
        content    String
        createdAt DateTime @default(now())
        chat    SupportChat @relation(fields: [chatId], references: [id], onDelete: Cascade)
        sender    User     @relation(name: "UserChatMessages", fields: [senderId], references: [id])

        @@index([chatId])
}
```

```
// ====================================
// LIKES Y VOTOS DE EVENTOS
// ====================================

model EventLike {
        id      String   @id @default(uuid())
        eventId   String
        userId String
        createdAt DateTime @default(now())
        event Event @relation(fields: [eventId], references: [id], onDelete: Cascade)
        user  User  @relation(fields: [userId], references: [id], onDelete: Cascade)

        @@unique([eventId, userId])
        @@index([eventId])
        @@index([userId])
}

model EventVote {
        id      String   @id @default(uuid())
        eventId   String
        userId String
        value  Int
        createdAt DateTime @default(now())
        event Event @relation(fields: [eventId], references: [id], onDelete: Cascade)
        user  User  @relation(fields: [userId], references: [id], onDelete: Cascade)

        @@unique([eventId, userId])
        @@index([eventId])
        @@index([userId])
}

// ====================================
// PROMOCIONES Y SORTEOS
// ====================================

model PromotionConfig {
        id             String   @id @default("singleton")
        isActive       Boolean  @default(false)
        planSlug       String   @default("plus")
        durationDays    Int     @default(30)
        updatedAt      DateTime @updatedAt
}

model Raffle {
        id             String         @id @default(uuid())
        title          String         @db.VarChar(150)
        description    String         @db.Text
        deadline       DateTime
        showCounter    Boolean        @default(true)
        maxParticipants Int?
        condition      String         @default("all_users")
        status         RaffleStatus @default(ACTIVE)
        winnerId       String?
        winner         User?       @relation("RaffleWinner", fields: [winnerId], references:
[id])
        bannerText     String?        @db.VarChar(300)
        showInBanner   Boolean        @default(false)
        createdAt      DateTime       @default(now())
        updatedAt      DateTime       @updatedAt

        @@index([status])
        @@index([deadline])
}
```

```
model AnnouncementBar {
        id      String   @id @default("global")
        text    String   @db.VarChar(300)
        link    String?  @db.VarChar(500)
        linkText  String?  @db.VarChar(100)
        isActive  Boolean  @default(false)
        updatedAt DateTime @updatedAt
}
```

Código fuente extraído del archivo schema.prisma del proyecto.

Arquitectura y tecnologías

En este proyecto decidí establecer un modelo de fullstack unificado basado en el App Router de Next.js para así aprovechar al máximo todas las ventajas que nos ofrece Next.js al crear un proyecto front y back en un único repositorio.

Patrón arquitectónico

La web se organiza siguiendo el principio de "Separación de responsabilidades" (Separation of Concerns), pero aprovechando las capacidades híbridas de los React Server Components. El flujo de datos se divide en las siguientes capas lógicas:

- **Client y Server components:** Uso React y Tailwind CSS para construir interfaces interactivas. Los Server Components se encargan de la recuperación de datos inicial, mientras que los Client Components gestionan la interactividad compleja (como el flujo de votación o los gráficos de estadísticas).

- **Server actions:** En lugar de una API REST tradicional, las reglas de negocio y mutaciones se convierten en Server Actions localizados en "*src/app/lib/*-actions.ts*". Aquí está la lógica de validación (Zod), el procesamiento de votos y la gestión de eventos.

- **Prisma y PostgreSQL:** Prisma ORM es el intermediario entre la lógica del servidor y la base de datos relacional PostgreSQL permitiendo consultas tipadas y seguras.

- **Servicios externos:** La arquitectura se extiende mediante integraciones de terceros para funcionalidades críticas, como por ejemplo Stripe para la facturación, Resend para el ciclo de vida de correos electrónicos, Framer Motion para animaciones, Lucide React para los iconos y NextAuth para la gestión de sesiones y seguridad.

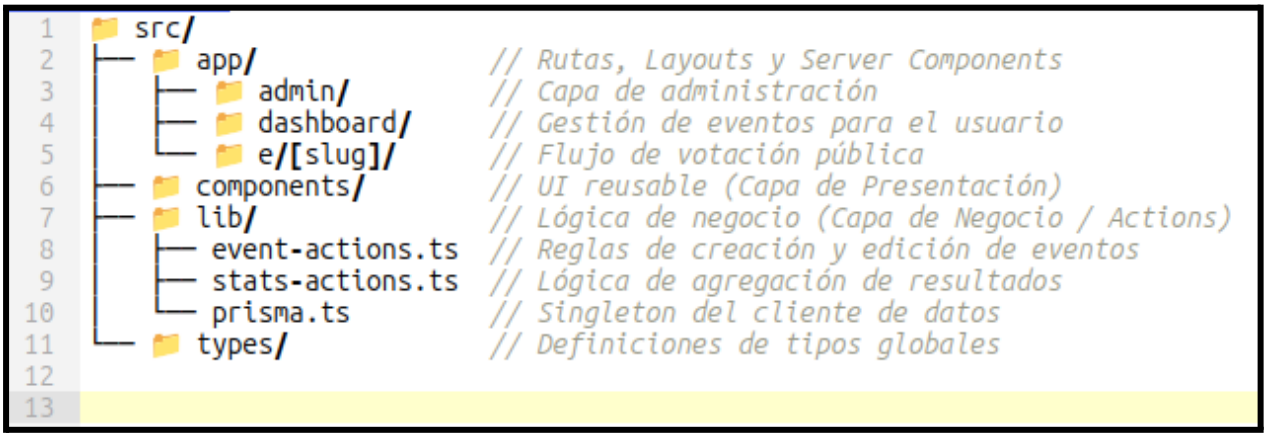
Stack tecnológico

Estas son las tecnologías que he elegido para crear este proyecto.

Capa arquitectónica	Tecnología	Función
Framework principal	**Next.js 16+**	Organización completa de rutas, renderizado y servidor.
Lenguaje	**TypeScript**	Garantizar un código robusto mediante tipado estático.
Base de datos	**PostgreSQL**	Almacenamiento relacional de usuarios, eventos, votos y auditorías.
ORM	**Prisma**	Gestión de migraciones y acceso a los datos.
Autenticación	**NextAuth.js**	Gestión de sesiones, login con OAuth (Google), credenciales y protección de rutas.
Pagos y suscripción	**Stripe**	Procesamiento de suscripciones de pago y manejo de webhooks.
Interfaz de usuario	**Tailwind CSS**	Estilos rápidos y consistentes mediante diseño basado en utilidades.
Animaciones	**Framer Motion**	Mejora en la experiencia de usuario y visualización de galas con animaciones.

Estructura de carpetas

El código del proyecto lo he organizado de forma que la jerarquía de archivos refleja las responsabilidades definidas. Esta organización me permite que el proyecto sea escalable y fácil de mantener con el tiempo, separando lo que es la lógica del servidor de la interactividad del cliente.



*La estructura de carpetas completa está disponible en el **README.md** de mi proyecto en github.*

Cronograma de implementación

Tras definir la arquitectura y el modelo de datos de Pollnow voy a explicar la planificación que tuve para desarrollar el proyecto, quise tener un enfoque ágil basado en Sprints de dos semanas (a raíz de lo que aprendí en mi primer año del curso con la filosofía SCRUM). De esta manera convertí los casos de uso en tareas técnicas medibles que no superaran los 3 días de ejecución para garantizarme a mi mismo un flujo de trabajo constante y sin bloqueos.

Metodología de planificación

La implementación la organicé siguiendo un orden lógico de dependencias. Base de datos, lógica de negocio (Server Actions) e interfaz de usuario (Frontend). Para la gestión visual de estas tareas utilicé el tablero de mi Trello configurado previamente con el método Kanban.

Desglose de tareas por Sprints

El desarrollo lo dividí en cinco etapas o “sprints” principales (con una duración de una a dos semanas aproximadamente por cada sprint). Empecé el día 18 de noviembre del 2025 y hasta el día de hoy ya ha recibido más de 220 commits.

Sprint 1: Cimientos y Autenticación	(1 semana)
BDD-01	Configuración inicial de PostgreSQL y despliegue del esquema base con Prisma.
ACT-01	Implementación de flujos de NextAuth (Registro, Login, Google OAuth y verificación de email).
FRN-01	Maquetación de las páginas de acceso y landing page con Tailwind CSS.

Sprint 2: Gestión de Eventos y Participantes	(2 semanas)
ACT-02	Programación de Server Actions para el CRUD de eventos, categorías (polls) y participantes.
ACT-03	Integración de lógica para la generación de imágenes de participantes mediante IA.
FRN-02	Desarrollo del Dashboard de usuario y el panel de edición de galas con interactividad React.

Sprint 3: Motor de Votación y Estadísticas	(1 semana)
ACT-04	Desarrollo del motor de votación anónima basado en voter hashes y cookies de sesión.
ACT-05	Lógica de agregación de resultados y generación de estadísticas en tiempo real.
BDD-02	Implementación del sistema de persistencia de votos con restricciones de integridad.

Sprint 4: Monetización y Administración (3 semanas)	
ACT-06	Integración de la API de Stripe para suscripciones y webhooks.
ACT-07	Sistema de tickets de soporte y notificaciones internas.
FRN-03	Desarrollo del panel de administración para moderación de eventos y gestión de usuarios.

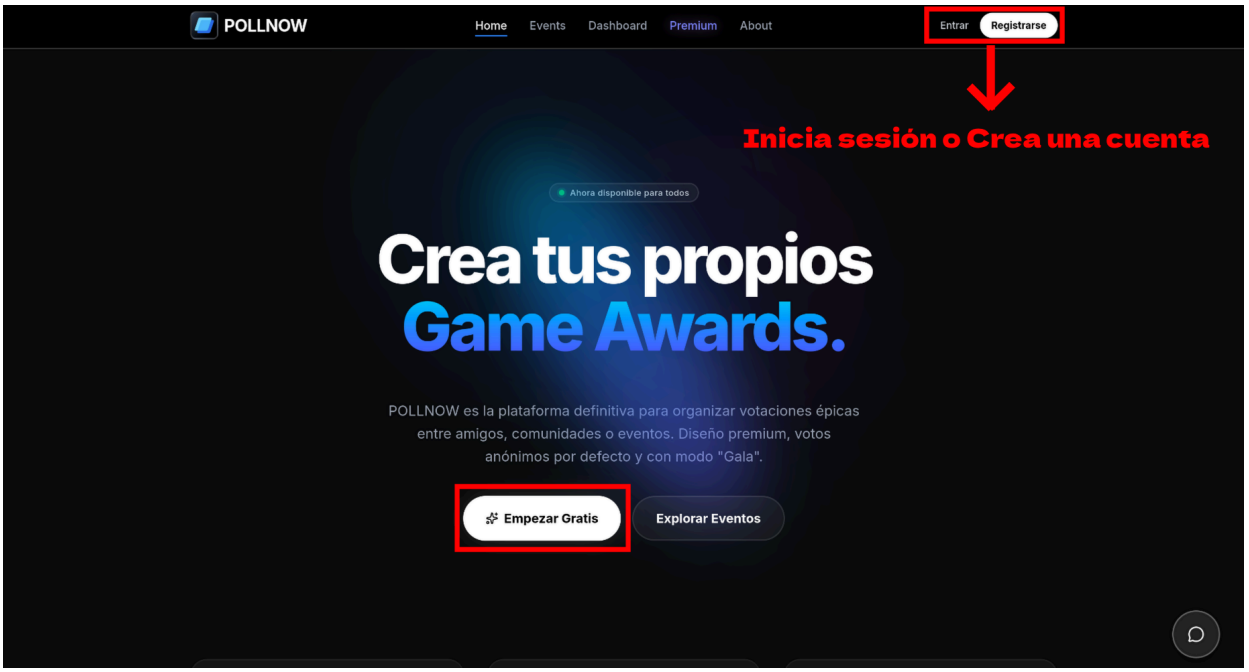
Sprint 5: Refinamiento y Despliegue (2 semanas)	
FRN-04	Pulido de UX con animaciones (usando la librería Framer Motion).
TST-01	Pruebas unitarias y control de acceso usando el middleware de Next.js.
DEP-01	Despliegue final en producción usando Vercel.

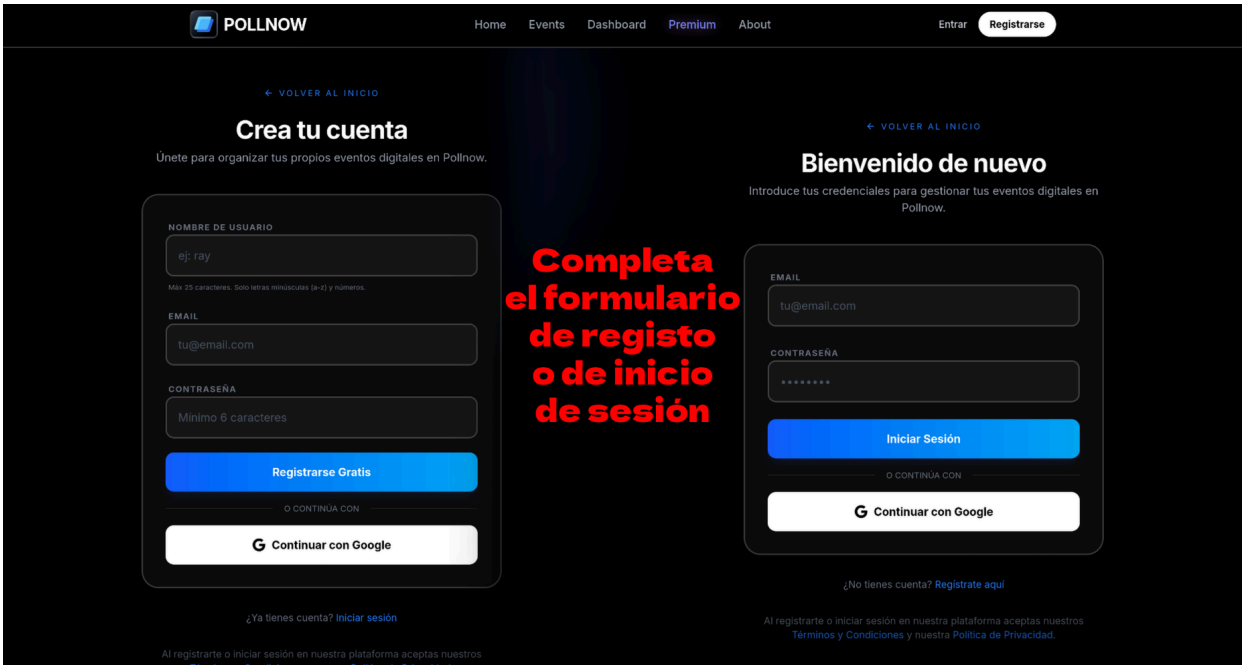
Visualización del flujo

Para que todo estuviera en orden cada tarea del proyecto la iba desplazando por las columnas del tablero de gestión según el estado en el que se encontrara dicha tarea durante el desarrollo. Una tarea la consideraba finalizada (DONE) únicamente cuando el código había sido verificado con TypeScript y el esquema de base de datos (schema.prisma) había sido migrado correctamente y sin fallos.

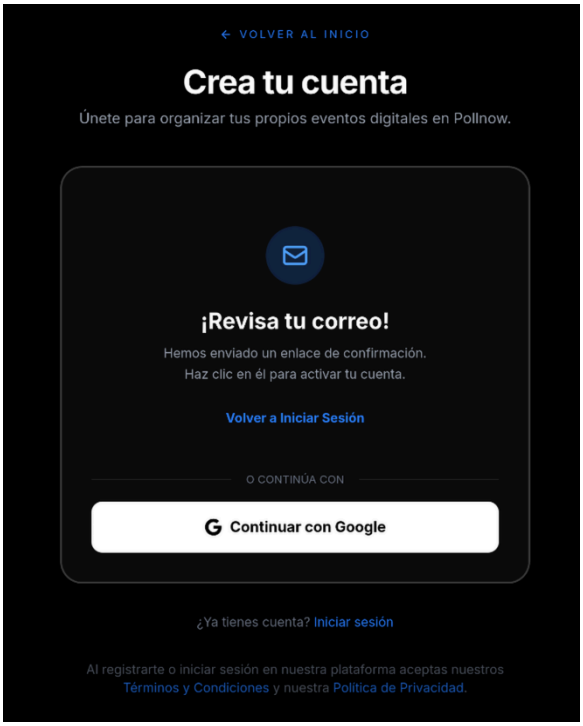
Manual básico de uso para el usuario

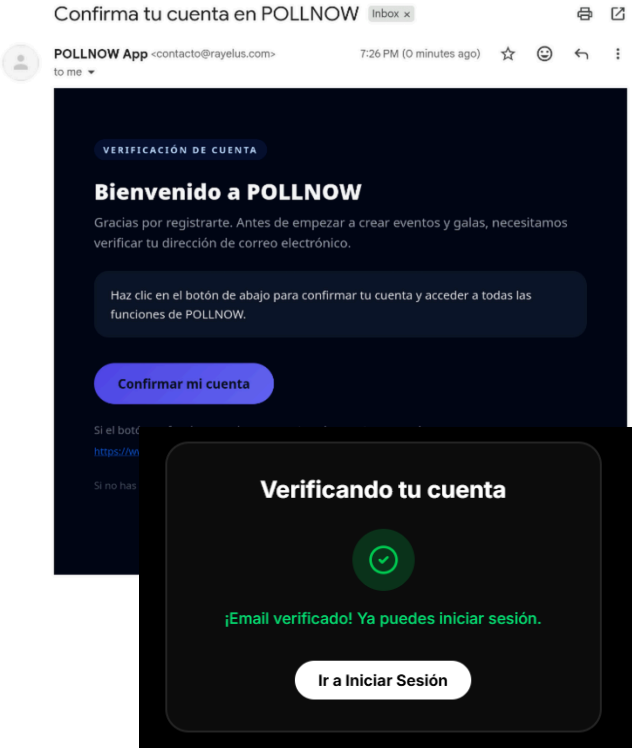
Cuando entras a la web esto es lo primero que encuentras, desde aquí puedes acceder a la mayoría (por no decir todas) las páginas más importantes de la web. No es necesario que te crees una cuenta para votar en eventos, pero puedes comenzar creando una siguiendo estos pasos.



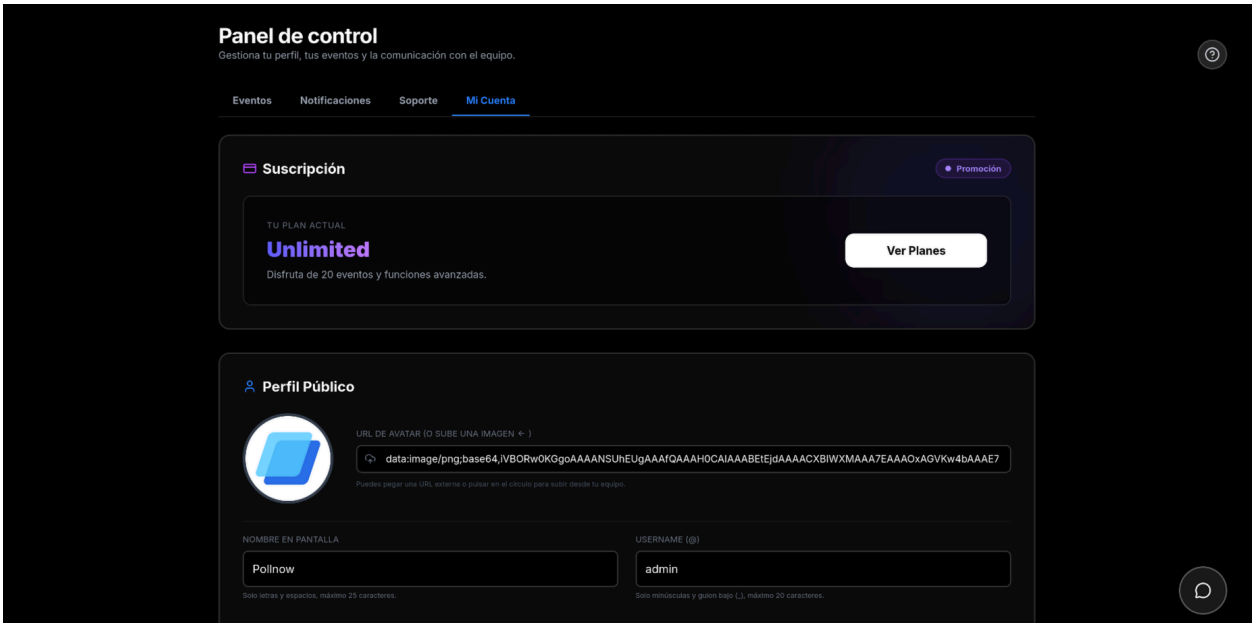


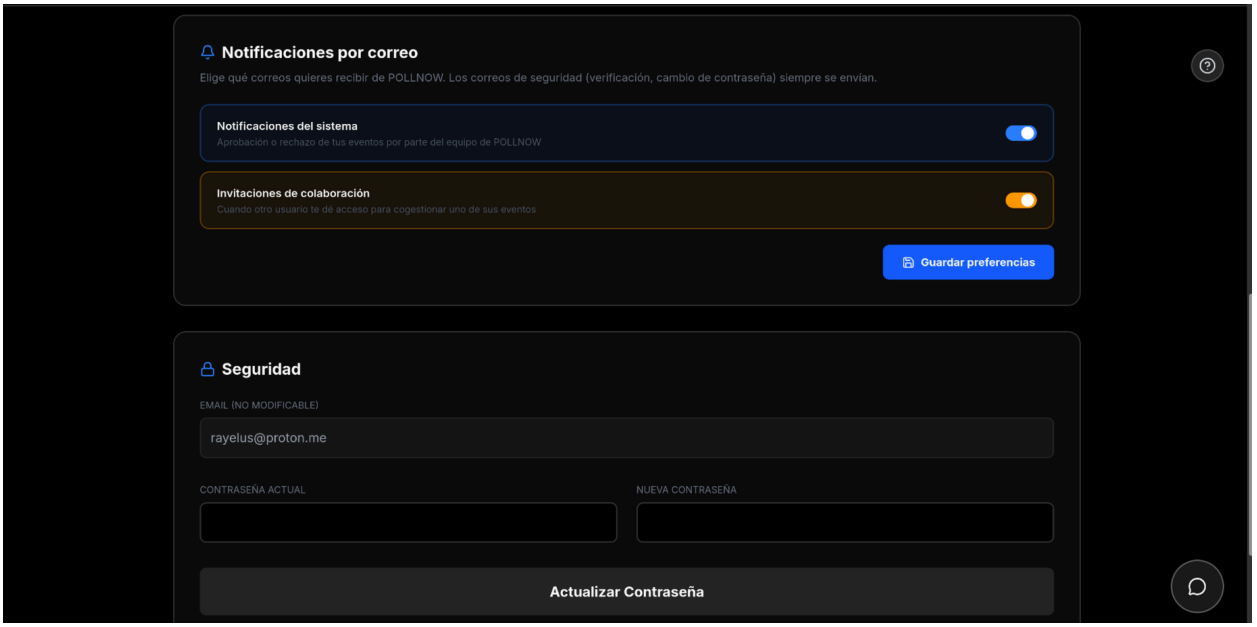
Una vez inicies sesión podrás acceder a tu perfil y empezar a crear tus propios eventos. En el caso de que hayas creado una cuenta nueva será necesario que verifiques tu correo electrónico.





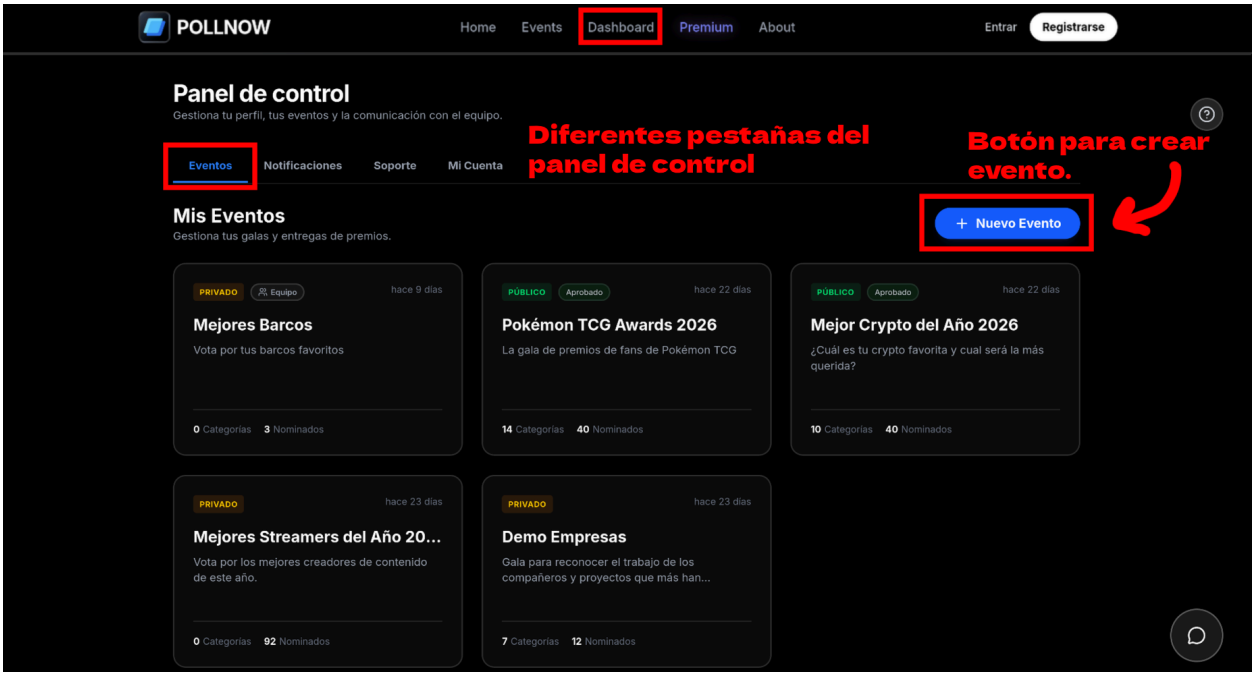
En tu perfil podrás encontrar tu información, preferencias, contraseña y plan de suscripción.



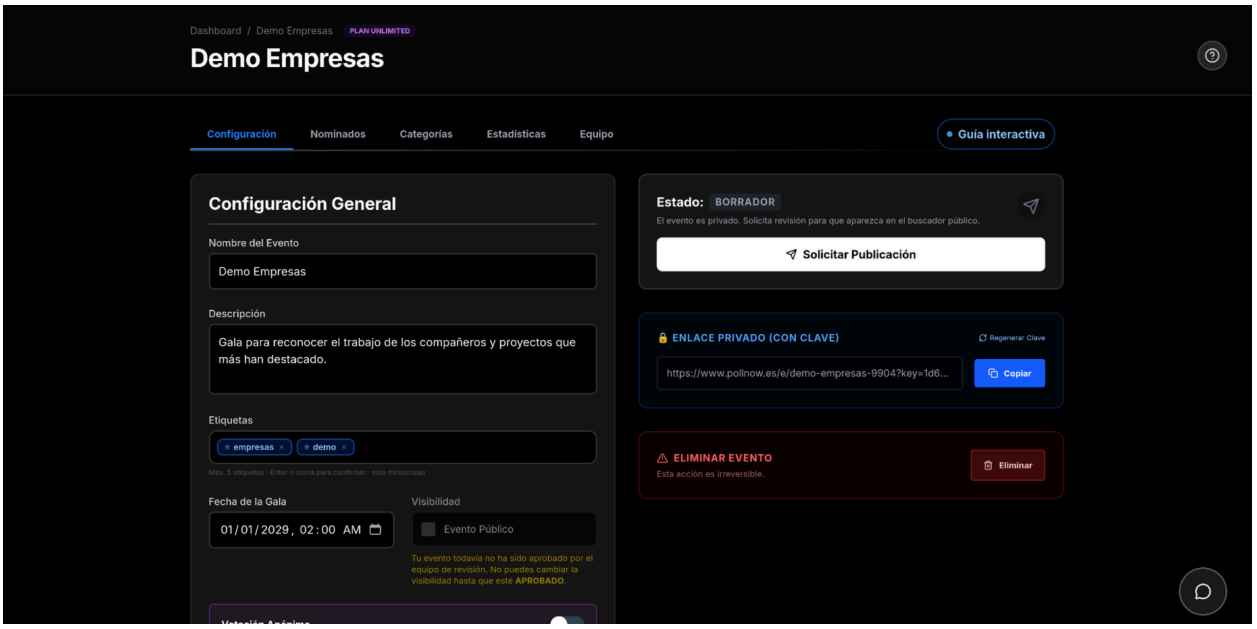


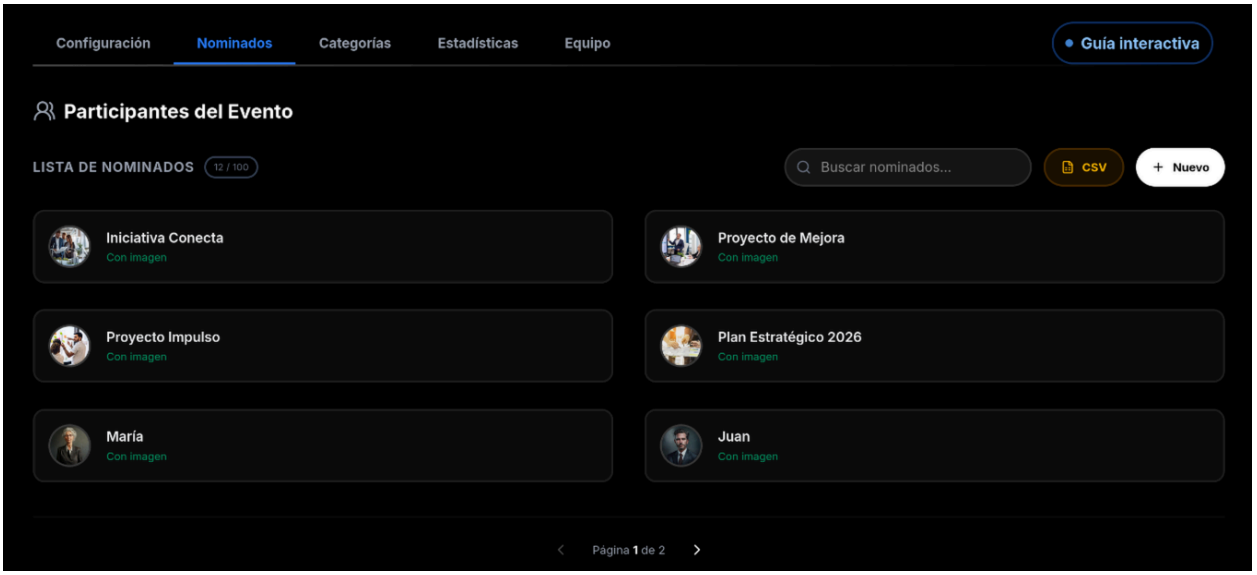
Lo primero que verás al iniciar sesión será la sección de eventos públicos de la comunidad. Aquí podrás buscar, filtrar y seleccionar los eventos que prefieras.

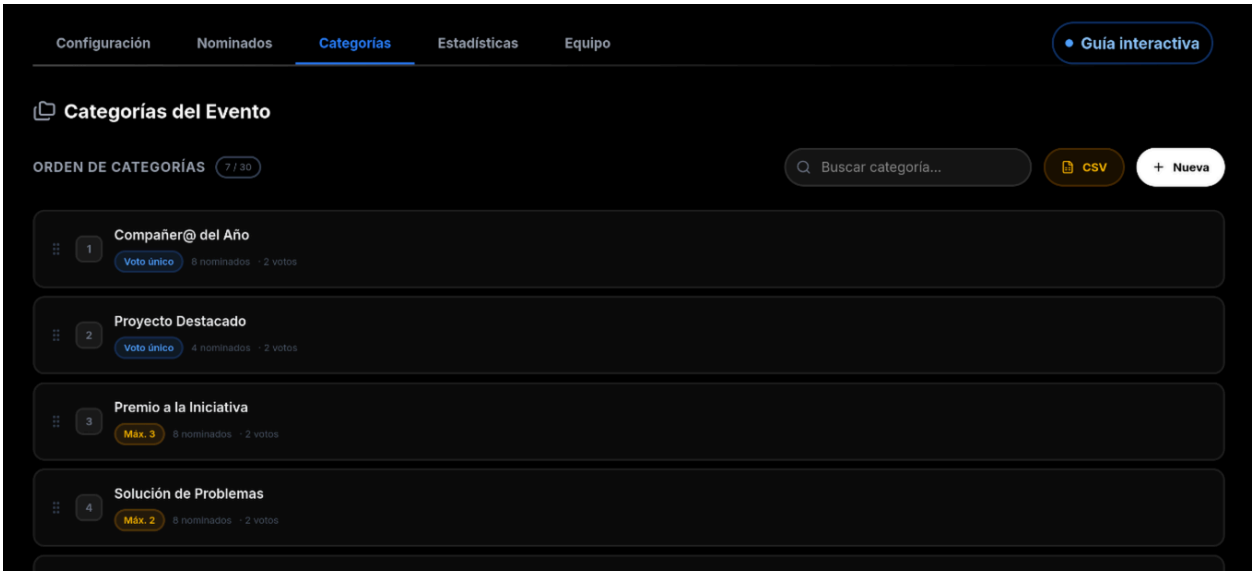
Por otro lado tendrás tu panel de control donde podrás gestionar tus eventos, notificaciones, chats de soporte y tu cuenta.



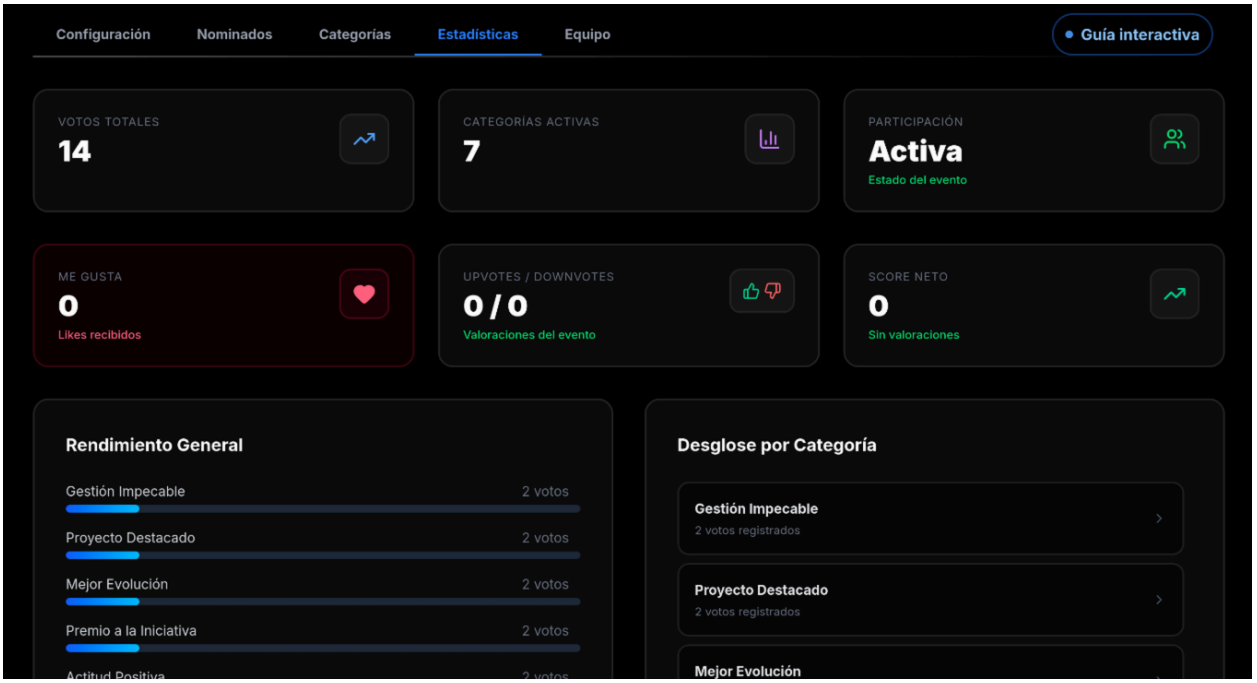
Cuando creas un evento puedes acceder a él para editarlo a tu gusto. Tanto la configuración general como los nominados, categorías y el equipo en el que colabora en la creación del evento.



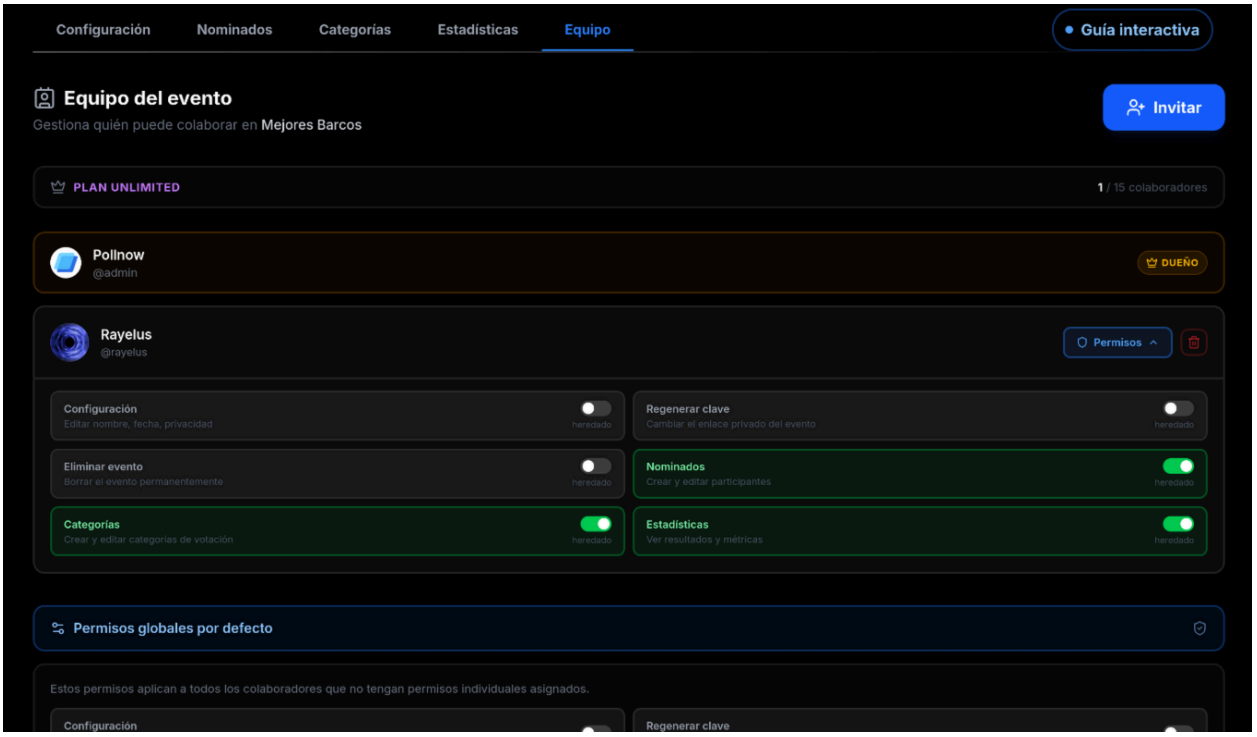




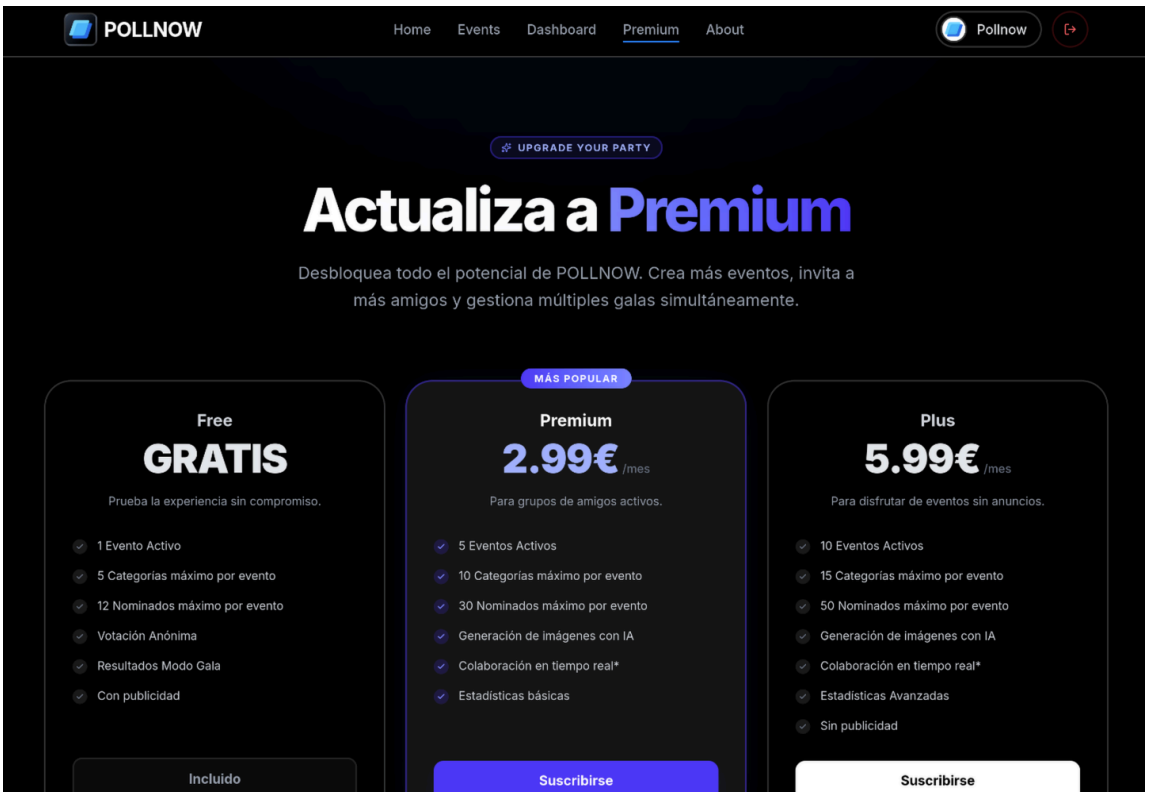
También puedes ver las estadísticas de dicho evento, los votos totales recibidos, categorías activas, me gusta, upvotes y downvotes (métricas para calcular la popularidad de los eventos públicos). Si tienes un plan de suscripción de nivel Unlimited o superior podrás desactivar el voto anónimo.

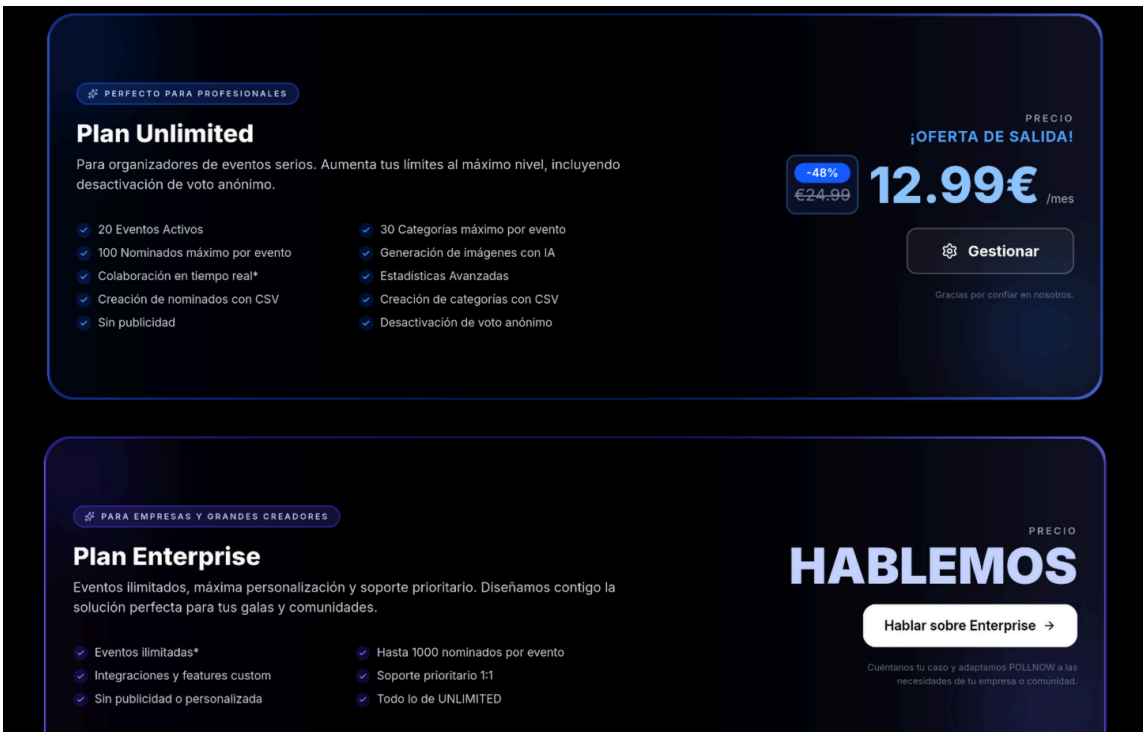


Y por último puedes gestionar el equipo con el que creas el evento de manera conjunta (esta función solo está disponible para los usuarios con un nivel de suscripción de nivel Premium o superior).

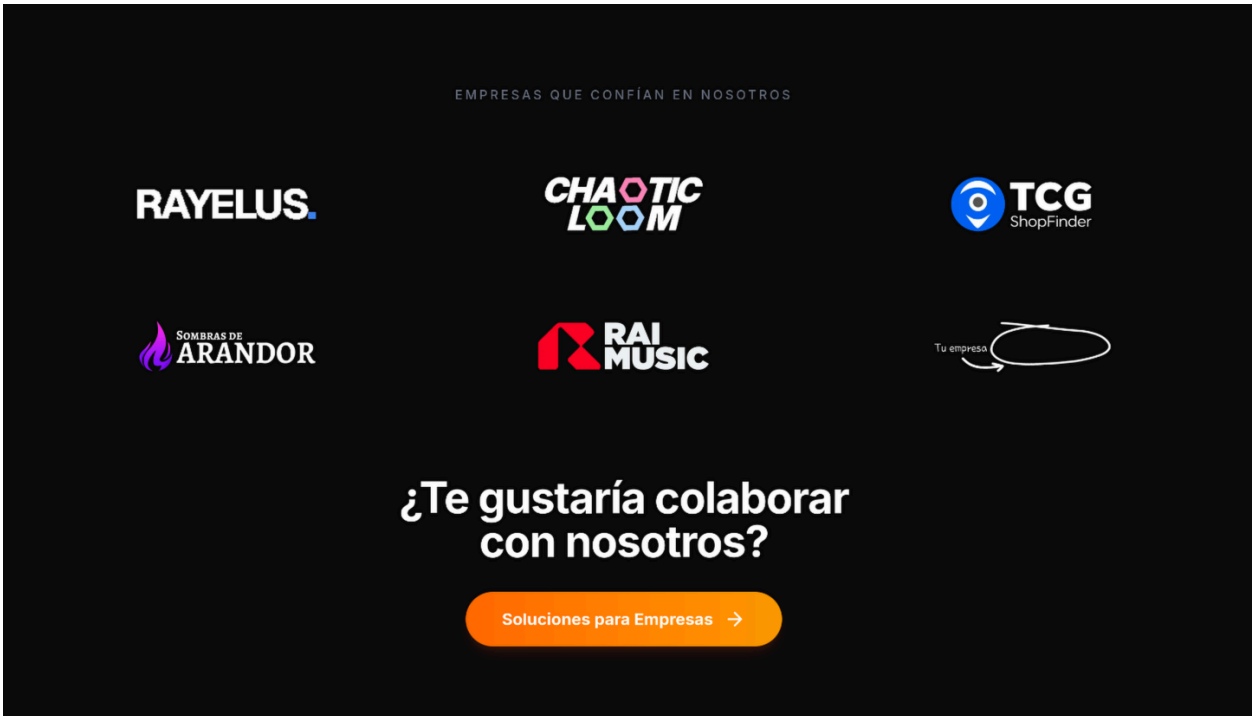


En la página de planes de suscripción podrás encontrar los diferentes niveles de pago que tiene la web.



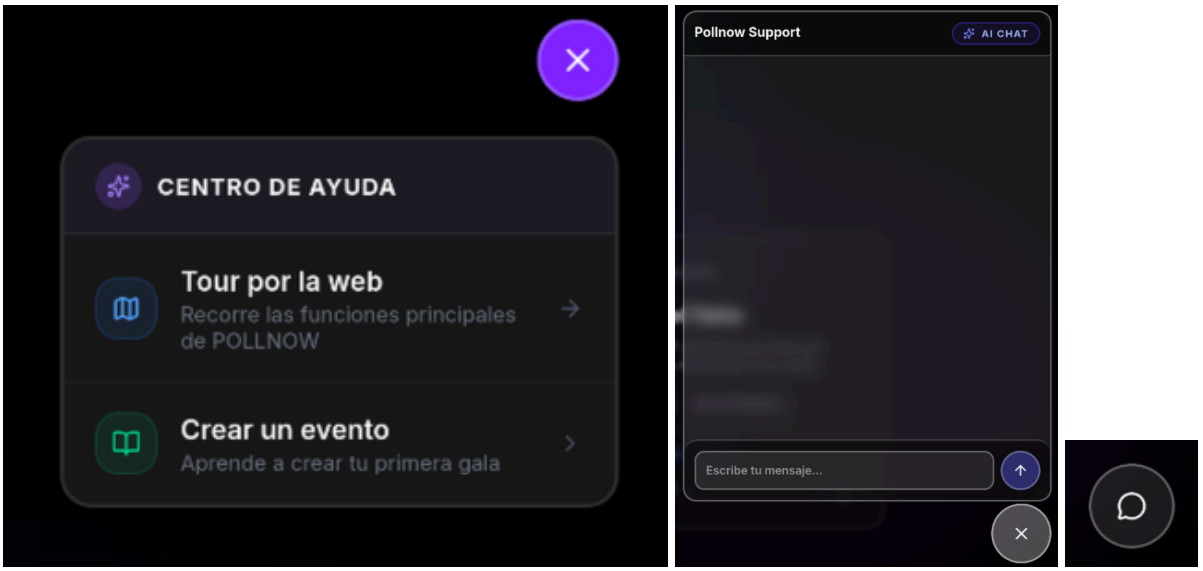


En el caso de que tuvieras una empresa Pollnow ofrece licencias en la propia web.



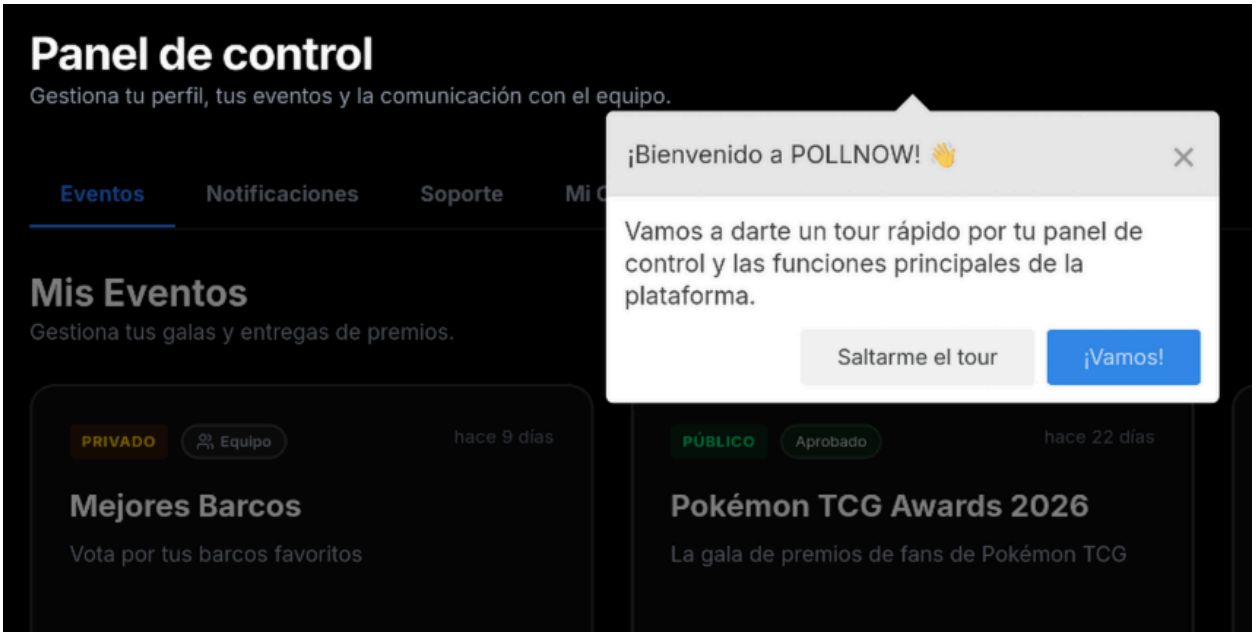
Una vez pulsas en "Soluciones para Empresas" este botón te llevará a una sección exclusiva para pequeñas y grandes empresas que buscan una solución similar a la que ofrecemos en la web.

Este botón de “Solicitar licencia” te llevará al formulario de contacto indicado para tramitar la solicitud dentro de la web.

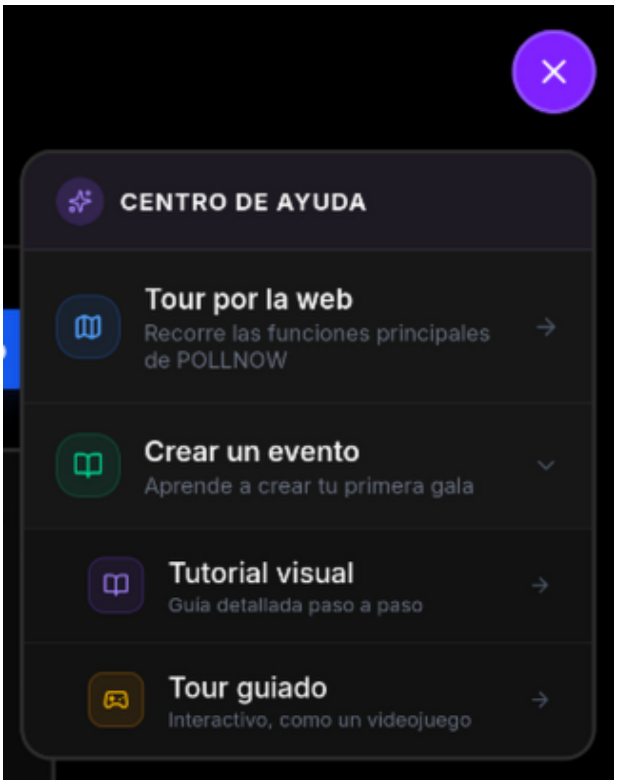


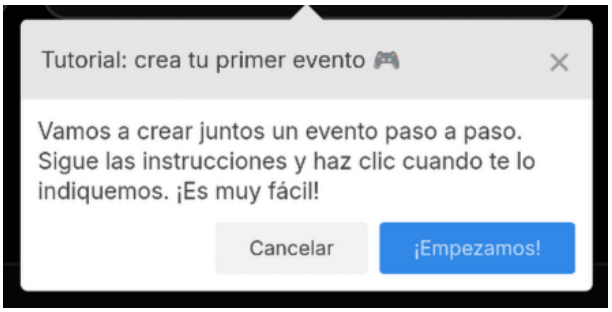
Si es tu primera vez creando un evento con Pollnow y no tienes mucha idea de como funciona puedes revisar el centro de ayuda y el chatbot con inteligencia artificial.

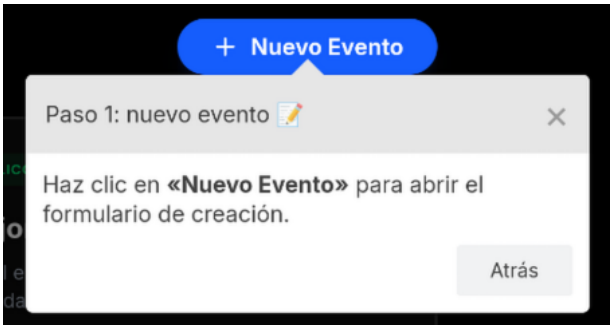
El centro de ayuda te permite realizar un tour guiado por toda la web.

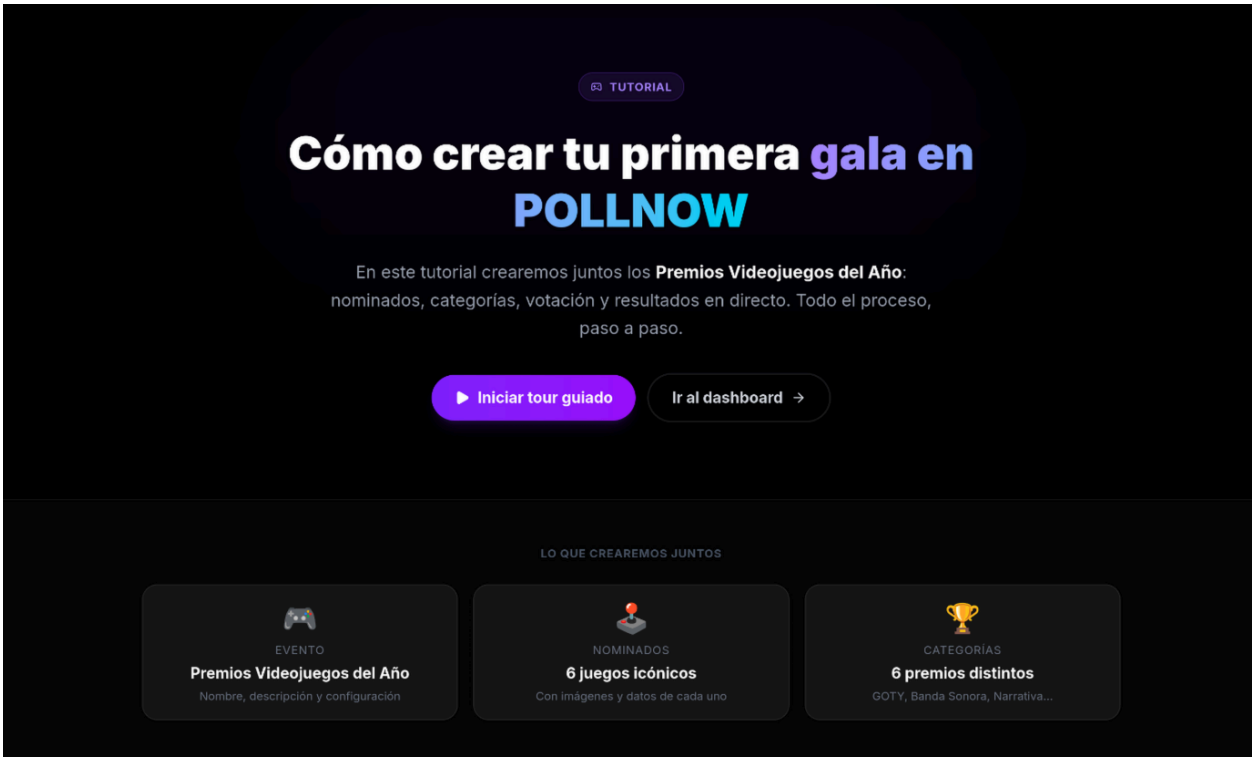


También puedes obtener más ayuda sobre crear tu propio evento con una página dedicada a mostrar un tutorial visual paso a paso o un tour guiado interactivo.

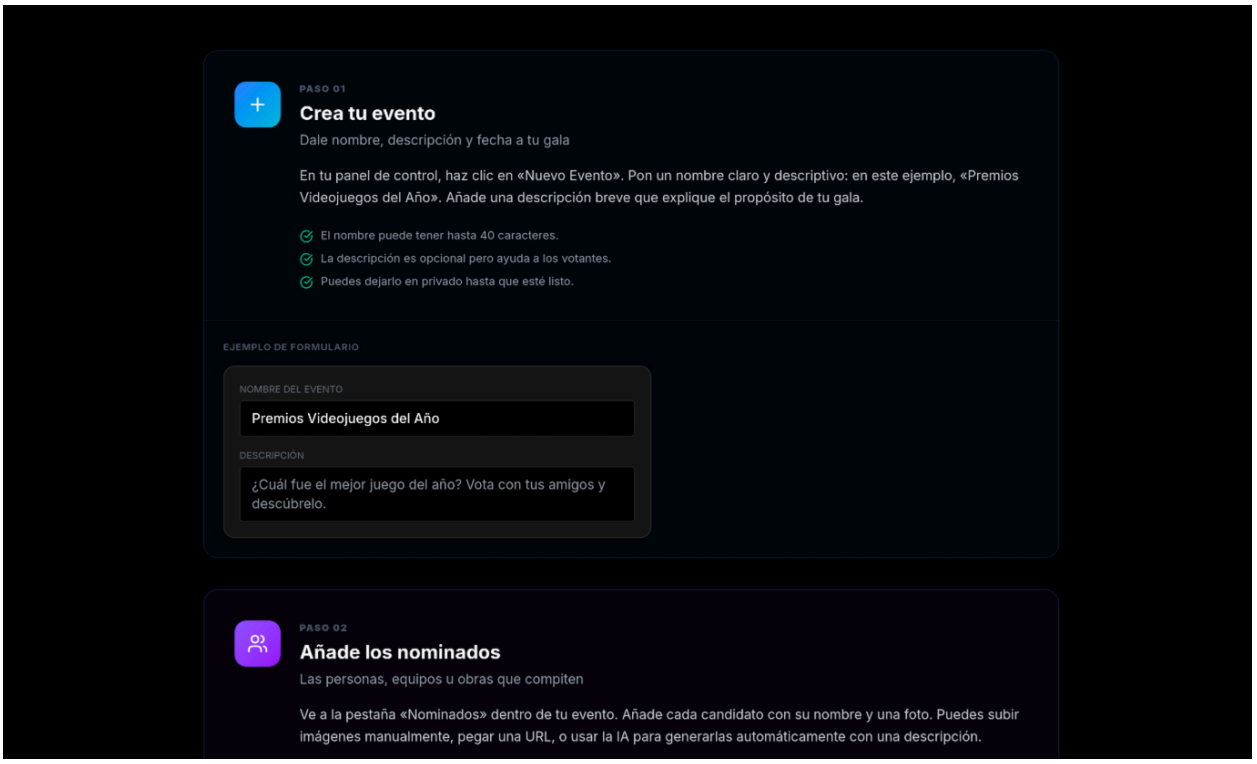








En la página de tutorial para crear tu propio evento te explica todo paso a paso para crear un buen evento.



Dificultades encontradas

Durante el desarrollo me enfrenté a varios problemas complicados, como son la creación de un panel de administración completamente desde cero y personalizado, la integración de Stripe para los pagos y suscripciones y la autenticación y sesiones de usuarios usando NextAuth.js.

Para el desarrollo del panel de administración completamente desde cero y de manera personalizada tuve que diseñar un sistema capaz de gestionar usuarios, eventos, reportes, chats y diferentes apartados de la plataforma de forma eficiente y segura. Esto implicó trabajar tanto en el frontend como en el backend organizando correctamente la arquitectura del proyecto y asegurando que toda la información se mostrara y actualizara de manera dinámica.

Por otro lado la integración de Stripe para gestionar pagos y suscripciones premium requería comprender el funcionamiento de los métodos de pago, la creación y gestión de planes de suscripción, así como el tratamiento seguro de la información y los eventos enviados mediante webhooks. También tuve que sincronizar correctamente el estado de las suscripciones con la base de datos del proyecto para habilitar o restringir funcionalidades premium dependiendo del tipo de usuario.

Y por último la autenticación y gestión de sesiones mediante NextAuth.js fue de las primeras cosas que se me complicó durante el desarrollo. Aunque la librería facilita gran parte del proceso, tuve que comprender en profundidad cómo funciona el sistema de autenticación, la persistencia de sesiones, la protección de rutas privadas y la gestión de permisos dependiendo del rol de cada usuario. Además tuve que resolver problemas relacionados con la sincronización de sesiones y el manejo de tokens.

Evidentemente también tuve que enfrentarme a problemas menores como la depuración de errores, la optimización del rendimiento, la organización del código y la toma de decisiones sobre qué tecnologías utilizar en cada situación (por suerte la documentación oficial de las tecnologías que usé está muy bien redactada y explicada).

Conclusiones

Este es el segundo proyecto “serio” al que me he enfrentado, he trabajado con sistemas de autenticación y gestión de sesiones, registro y administración de usuarios, envío de correos electrónicos con Resend, utilización de ORM para la interacción con bases de datos, así como con diferentes librerías y tecnologías modernas utilizadas actualmente en el desarrollo web. También experimenté con procesos de despliegue y puestas en producción. Esto me ayudó mucho para entender mejor el ciclo de vida completo de una aplicación real.

La idea comenzó dentro de mi grupo de amigos y la broma terminó llegando muy lejos, hasta el punto de crear mi primer software como servicio completo. Estoy muy feliz de que la programación y el desarrollo me permitan crear este tipo de proyectos en mi tiempo libre.

En definitiva, he aprendido y dado lo máximo en este proyecto y sobre todo he disfrutado mucho durante el proceso y el desarrollo del mismo.

Como siempre digo, si puedes imaginarlo puedes programarlo.

Fuentes de información y recursos utilizados

Las fuentes y recursos que he usado para realizar este proyecto han sido las siguientes.

Fuentes de información

- DeepSeek, Chat GPT, Gemini y Claude como IAs generativas que me han ayudado a corregir y comprender errores del código así como completar ciertas partes complicadas que tenía pendientes durante el desarrollo.
- Tema 4 Accesibilidad de Mercedes Florido Berrocal para consultar los principios a seguir en cuanto a accesibilidad web.
- Repositorio de Github de Jose Antonio Muñoz para consultar dudas concretas sobre Prisma o Next.js durante el desarrollo.
- Documentación oficial de React
- Documentación oficial de TypeScript
- Características básicas de Next.js
- Documentación oficial de Next.js y NextAuth para completar la autenticación y partes complejas del código.

Recursos utilizados

- Mermaid Live Editor para la generación de diagramas para la base de datos.
- Github y Git para el control de versiones del proyecto.
- Visual Studio Code y Google Antigravity como entornos de desarrollo.
- Figma para la creación de layouts y diseños de la web.

Agradecimientos

Me gustaría dedicar y agradecer este proyecto a todas las personas que me han apoyado y lo han hecho posible.

En primer lugar agradezco a mi tutor David de Vega Rodrígez por su orientación, apoyo y sobre todo por su dedicación durante todo el proceso, de igual forma me gustaría agradecer al profesorado por todo lo que nos han enseñado durante estos dos años de ciclo superior.

También quiero dar las gracias tanto al I.E.S Hermanos Machado como a la empresa SEIDOR Opentrends en la que hice las prácticas en empresa por darme la oportunidad de aprender cosas nuevas, colaborar en proyectos muy interesantes que me han servido para saber trabajar como es debido en un entorno profesional y finalmente por contar conmigo como desarrollador full-stack junior con un contrato indefinido al terminar mis prácticas.

Por último quiero agradecer a mi familia y amigos por todo su apoyo incondicional y motivación durante esta etapa. Sin ellos yo no estaría aquí hoy.

Este proyecto se lo dedico a todos ellos y especialmente a mi pareja Laura, por todo lo que ha hecho y hace por mí todos los días sin esperar nada a cambio.

A todos, muchas gracias.

Autor: Raimundo Palma Méndez

Presentación: POLLNOW - Presentación Proyecto Intermodular 2ºDAW (2025/2026)

Repo del proyecto en Github: https://github.com/Rayelus5/pollnow