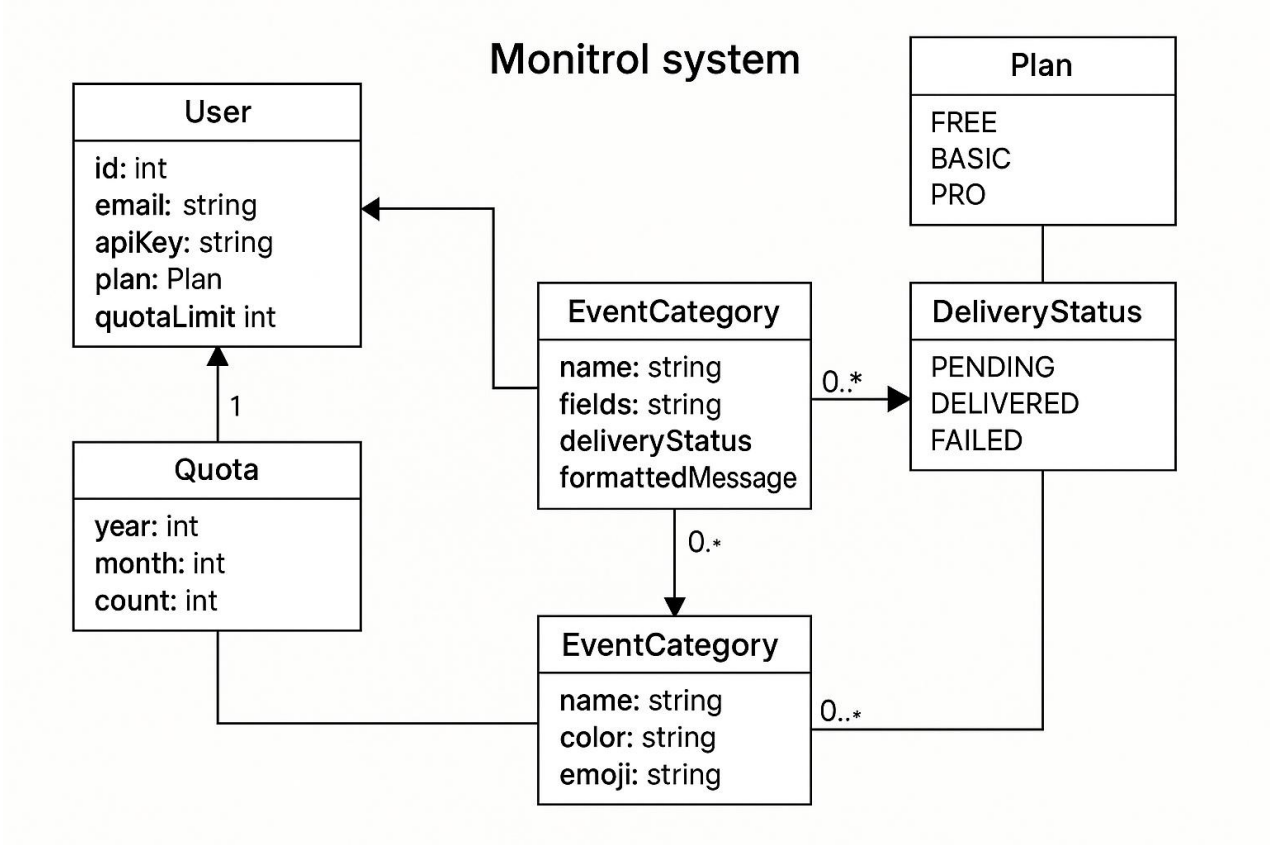
Monitrol - Architecture Documentation

Overview

This document presents the architectural design of Monitrol, a modern fullstack event monitoring SaaS platform built with Next.js, PostgreSQL, and integrated third-party services.

1. Entity Relationship Diagram ERD

The database schema consists of four main entities with the following relationships:



Core Entities:

- **User**: Central entity storing user information, API keys, and subscription plans
- **EventCategory**: User-defined categories for organizing events
- **Event**: Main event records with delivery status tracking **Quota**: Monthly
- usage tracking per user

Key Relationships:

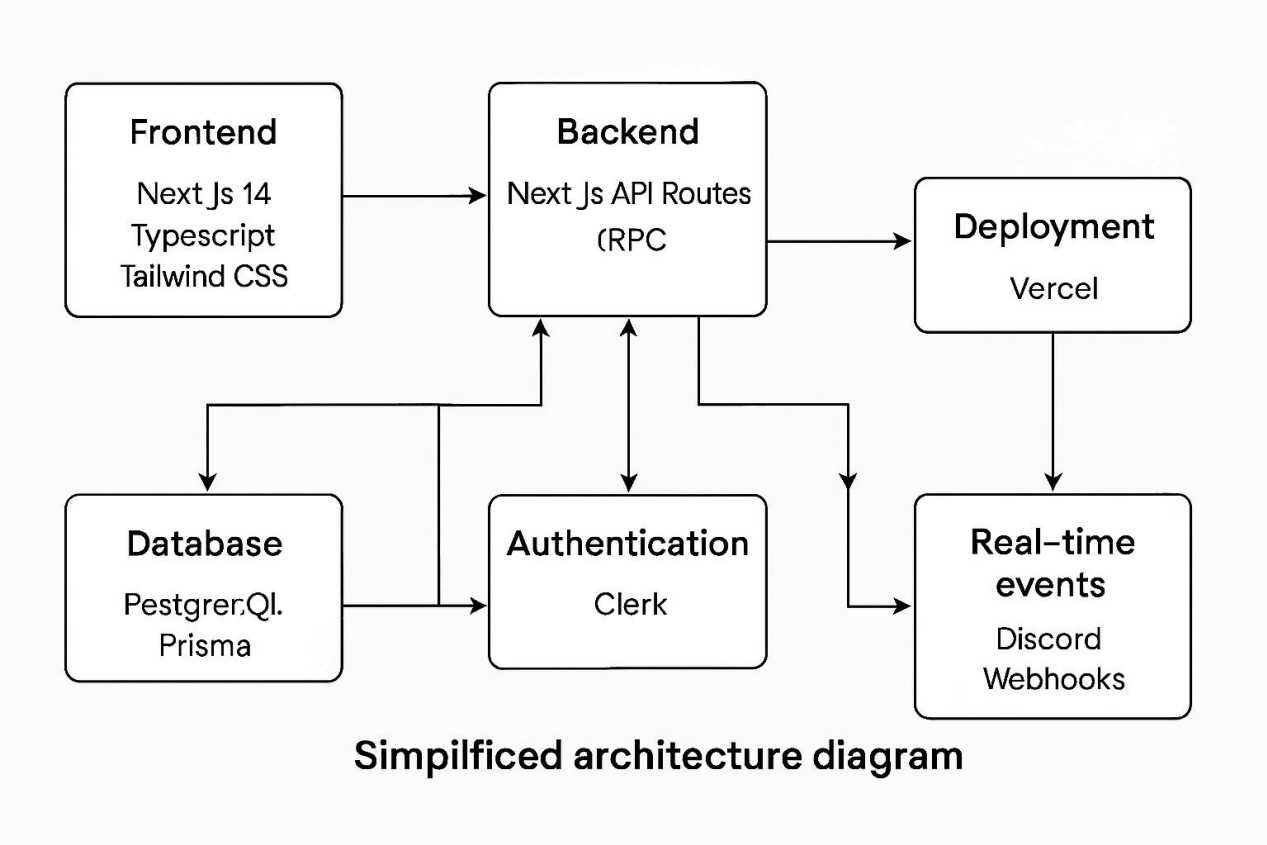
- User → EventCategory One-to-Many)
- User → Event One-to-Many)
- User → Quota One-to-One)
- EventCategory → Event One-to-Many)

Enums:

- **Plan**: FREE, PRO
- **DeliveryStatus**: PENDING, DELIVERED, FAILED

2. System Architecture Flowchart

The system follows a typical SaaS architecture pattern:



User Authentication: Clerk handles secure user registration and login

Dashboard Interface: Next.js provides responsive UI for event management

Event Processing: Events are stored in PostgreSQL and processed for notifications

Real-time Notifications: Discord webhooks deliver instant event alerts

Payment Processing: Stripe manages subscription billing for PRO features

3. UML Class Diagram

The object-oriented design includes:

Main Classes:

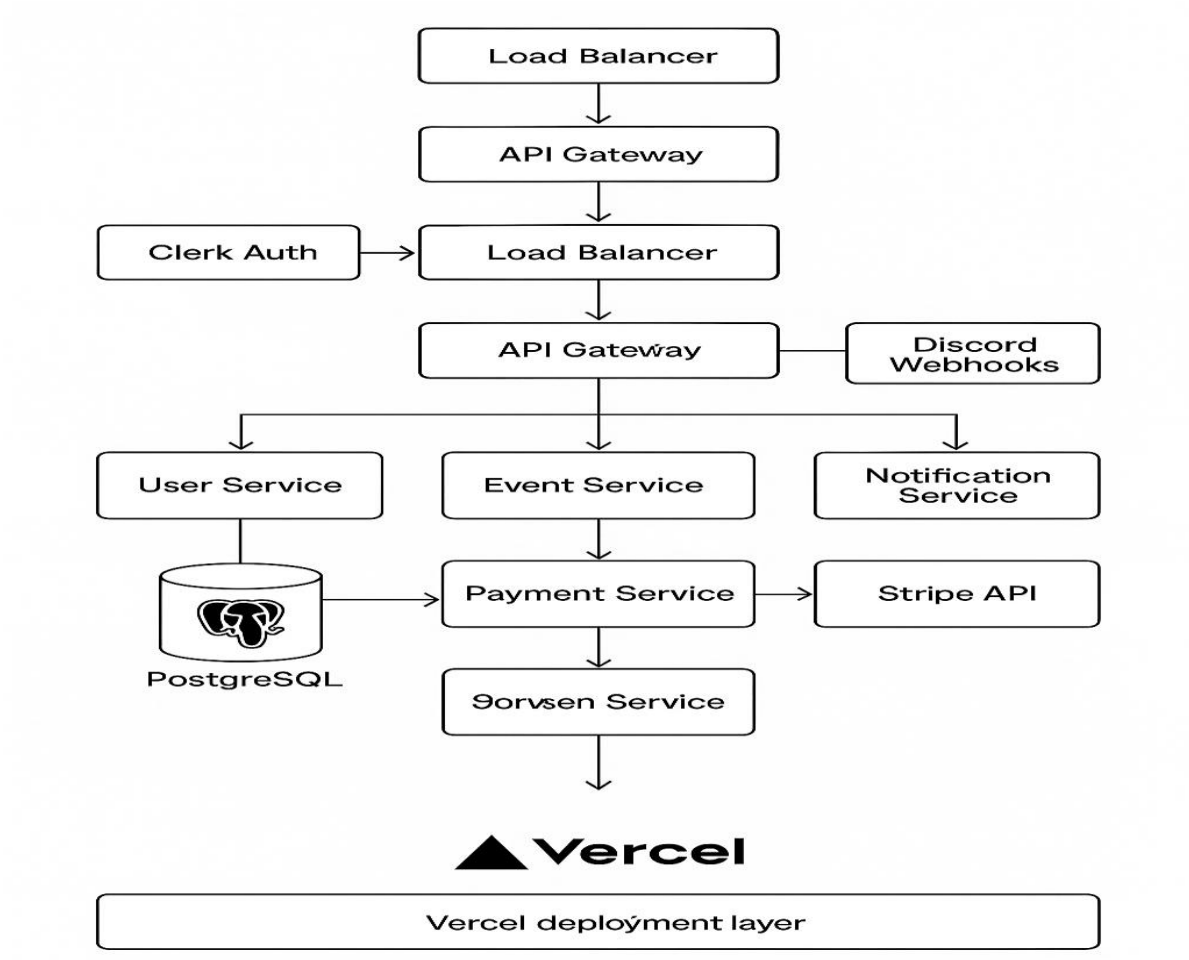
- **User Class**: Manages user data, authentication, and subscription information
- **EventCategory Class**: Handles event categorization with visual attributes
- **Event Class**: Core event data with JSON field storage and delivery tracking **Quota Class**: Tracks
- monthly usage limits per user

Design Patterns:

- Composition relationships between User and related entities
- Enum-based status management for plans and delivery states

4. Microservice Architecture

While currently implemented as a monolithic Next.js application, the system could be architected as microservices:

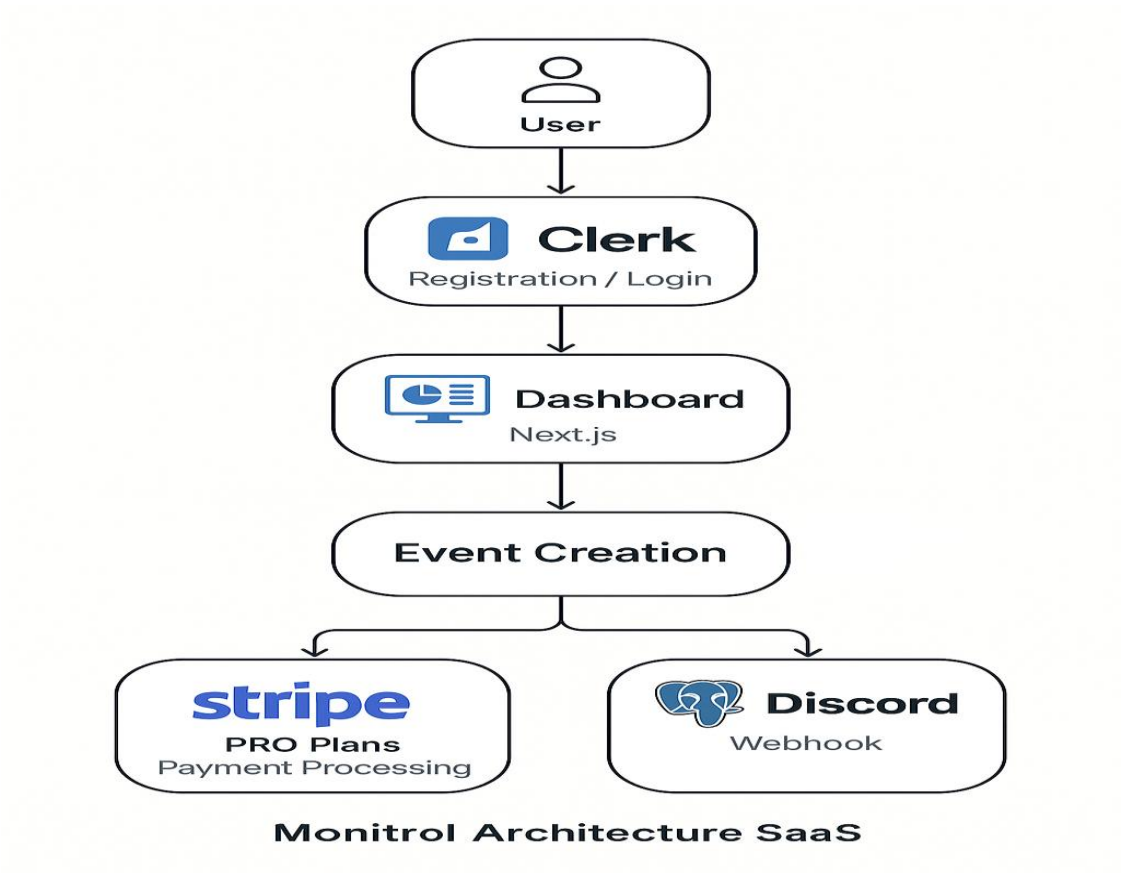


Proposed Services:

- **User Service**: Handle user management and authentication
- **Event Service**: Process and store event data
- **Notification Service**: Manage Discord webhook delivery **Payment Service**:
- Handle Stripe integration and billing

Infrastructure:

- API Gateway for request routing
- PostgreSQL as shared data store
- External service integrations Clerk, Stripe, Discord) Vercel
- deployment platform



Technical Stack

- **Frontend**: Next.js 14, TypeScript, Tailwind CSS
- **Backend**: Next.js API Routes, tRPC
- **Database**: PostgreSQL with Prisma ORM
- **Authentication**: Clerk
- **Payments**: Stripe
- **Notifications**: Discord Webhooks

- **Deployment**: Vercel

Scalability Considerations

The current monolithic architecture provides simplicity for development and deployment, while the database schema supports horizontal scaling through proper indexing and relationships.
Future microservice migration could improve scalability and service isolation.