Data Models & API Design - Curated Events Platform

Database Schema Design

Core Entity Relationships

```mermaid
erDiagram
    User ||--o{ Event : creates
    User ||--o{ EventAttendance : attends
    User ||--o{ Review : writes
    User ||--o{ SocialConnection : has
    Event ||--o{ EventAttendance : has
    Event ||--o{ Review : receives
    Event ||--o{ CurationWorkflow : undergoes
    Event ||--o{ EventMedia : contains
    Curator ||--o{ CurationReview : performs
    Organization ||--o{ Event : hosts
    Venue ||--o{ Event : hosts
    Category ||--o{ Event : categorizes
```

1. User Domain Models

User Entity

```sql
```

```sql
-- PostgreSQL Schema
CREATE TABLE users (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   email VARCHAR(255) UNIQUE NOT NULL,
   email_verified BOOLEAN DEFAULT FALSE,
   password_hash VARCHAR(255),
   created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   deleted_at TIMESTAMP WITH TIME ZONE,

   -- Profile Information
   display_name VARCHAR(100) NOT NULL,
   first_name VARCHAR(50),
   last_name VARCHAR(50),
   avatar_url TEXT,
   bio TEXT,
   birth_date DATE,

   -- Location
   location_city VARCHAR(100),
   location_state VARCHAR(100),
   location_country VARCHAR(100),
   coordinates POINT, -- PostGIS point type

   -- Preferences
   timezone VARCHAR(100) DEFAULT 'UTC',
   language VARCHAR(10) DEFAULT 'en',
   currency VARCHAR(3) DEFAULT 'USD',

   -- Privacy Settings
   profile_visibility user_visibility_enum DEFAULT 'public',
   allow_friend_requests BOOLEAN DEFAULT TRUE,
   show_attendance BOOLEAN DEFAULT TRUE,

   -- System Fields
   is_active BOOLEAN DEFAULT TRUE,
   is_verified BOOLEAN DEFAULT FALSE,
   verification_level INTEGER DEFAULT 0,

   CONSTRAINT valid_email CHECK (email ~* '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$')
);

CREATE TYPE user_visibility_enum AS ENUM ('public', 'friends', 'private');
```

```
-- Indexes
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_location ON users USING GIST(coordinates);
CREATE INDEX idx_users_created_at ON users(created_at);
```

User Preferences

```sql
CREATE TABLE user_preferences (
  user_id UUID PRIMARY KEY REFERENCES users(id) ON DELETE CASCADE,

  -- Event Preferences
  preferred_categories TEXT[] DEFAULT '{}',
  max_distance_km INTEGER DEFAULT 50,
  price_range_min DECIMAL(10,2) DEFAULT 0,
  price_range_max DECIMAL(10,2),
  preferred_times time_preference_enum[] DEFAULT '{any}',

  -- Notification Preferences
  email_notifications BOOLEAN DEFAULT TRUE,
  push_notifications BOOLEAN DEFAULT TRUE,
  sms_notifications BOOLEAN DEFAULT FALSE,
  notification_frequency notification_freq_enum DEFAULT 'daily',

  -- Discovery Preferences
  show_suggested_events BOOLEAN DEFAULT TRUE,
  show_trending_events BOOLEAN DEFAULT TRUE,
  show_friends_activity BOOLEAN DEFAULT TRUE,

  updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE TYPE time_preference_enum AS ENUM ('morning', 'afternoon', 'evening', 'night', 'any');
CREATE TYPE notification_freq_enum AS ENUM ('immediate', 'daily', 'weekly', 'never');
```

User Reputation System

```sql
```

```
CREATE TABLE user_reputation (
  user_id UUID PRIMARY KEY REFERENCES users(id) ON DELETE CASCADE,

  -- Reputation Scores (0-1000)
  overall_score INTEGER DEFAULT 100,
  creator_score INTEGER DEFAULT 100,
  reviewer_score INTEGER DEFAULT 100,
  community_score INTEGER DEFAULT 100,

  -- Activity Metrics
  events_created INTEGER DEFAULT 0,
  events_attended INTEGER DEFAULT 0,
  reviews_written INTEGER DEFAULT 0,
  reviews_helpful INTEGER DEFAULT 0,
  community_flags_accurate INTEGER DEFAULT 0,

  -- Quality Metrics
  avg_event_rating DECIMAL(3,2),
  avg_review_helpfulness DECIMAL(3,2),
  response_rate DECIMAL(3,2),
  cancellation_rate DECIMAL(3,2),

  -- Badges and Achievements
  badges TEXT[] DEFAULT '{}',
  achievements JSONB DEFAULT '{}',

  last_calculated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

  CONSTRAINT valid_scores CHECK (
    overall_score BETWEEN 0 AND 1000 AND
    creator_score BETWEEN 0 AND 1000 AND
    reviewer_score BETWEEN 0 AND 1000 AND
    community_score BETWEEN 0 AND 1000
  )
);
```

2. Event Domain Models

Event Entity

```
sql
```

```sql
CREATE TABLE events (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  creator_id UUID NOT NULL REFERENCES users(id),
  organization_id UUID REFERENCES organizations(id),

  -- Basic Information
  title VARCHAR(200) NOT NULL,
  slug VARCHAR(250) UNIQUE NOT NULL,
  description TEXT NOT NULL,
  short_description VARCHAR(500),

  -- Categorization
  category_id UUID NOT NULL REFERENCES categories(id),
  subcategory VARCHAR(100),
  tags TEXT[] DEFAULT '{}',

  -- Schedule
  start_time TIMESTAMP WITH TIME ZONE NOT NULL,
  end_time TIMESTAMP WITH TIME ZONE NOT NULL,
  timezone VARCHAR(100) NOT NULL,
  is_all_day BOOLEAN DEFAULT FALSE,
  recurrence_rule TEXT, -- RRULE format for recurring events

  -- Location
  venue_id UUID REFERENCES venues(id),
  venue_name VARCHAR(200),
  address JSONB, -- Structured address
  coordinates POINT,
  is_online BOOLEAN DEFAULT FALSE,
  online_details JSONB, -- Meeting links, platform info

  -- Ticketing
  is_free BOOLEAN DEFAULT TRUE,
  ticket_price DECIMAL(10,2),
  currency VARCHAR(3) DEFAULT 'USD',
  ticket_url TEXT,
  external_ticket_provider VARCHAR(100),
  capacity INTEGER,
  requires_approval BOOLEAN DEFAULT FALSE,
  age_restriction INTEGER,

  -- Media
  cover_image_url TEXT NOT NULL,
    gallery_urls TEXT[] DEFAULT '{}',
    video_url TEXT,

    -- Status and Visibility
    status event_status_enum DEFAULT 'draft',
    visibility event_visibility_enum DEFAULT 'public',
    featured BOOLEAN DEFAULT FALSE,

    -- Engagement Metrics (denormalized for performance)
    view_count INTEGER DEFAULT 0,
    interested_count INTEGER DEFAULT 0,
    attending_count INTEGER DEFAULT 0,
    share_count INTEGER DEFAULT 0,
    save_count INTEGER DEFAULT 0,

    -- Timestamps
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    published_at TIMESTAMP WITH TIME ZONE,
    deleted_at TIMESTAMP WITH TIME ZONE,

    -- Constraints
    CONSTRAINT valid_time_range CHECK (end_time > start_time),
    CONSTRAINT valid_capacity CHECK (capacity IS NULL OR capacity > 0),
    CONSTRAINT valid_price CHECK (ticket_price IS NULL OR ticket_price >= 0)
);

CREATE TYPE event_status_enum AS ENUM ('draft', 'pending_review', 'approved', 'rejected', 'published', 'cancell
CREATE TYPE event_visibility_enum AS ENUM ('public', 'private', 'unlisted');

-- Indexes
CREATE INDEX idx_events_creator ON events(creator_id);
CREATE INDEX idx_events_category ON events(category_id);
CREATE INDEX idx_events_start_time ON events(start_time);
CREATE INDEX idx_events_location ON events USING GIST(coordinates);
CREATE INDEX idx_events_status ON events(status);
CREATE INDEX idx_events_tags ON events USING GIN(tags);
CREATE INDEX idx_events_text_search ON events USING GIN(to_tsvector('english', title || ' ' || description));
```

Event Attendance

```
sql
```

```
CREATE TABLE event_attendance (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    event_id UUID NOT NULL REFERENCES events(id) ON DELETE CASCADE,
    user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,

    -- Attendance Status
    status attendance_status_enum NOT NULL,
    tickets_quantity INTEGER DEFAULT 1,

    -- Timestamps
    registered_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    status_changed_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    checked_in_at TIMESTAMP WITH TIME ZONE,

    -- Additional Data
    registration_source VARCHAR(50), -- 'web', 'mobile', 'social_share'
    notes TEXT,
    metadata JSONB DEFAULT '{}',

    UNIQUE(event_id, user_id)
);

CREATE TYPE attendance_status_enum AS ENUM ('interested', 'attending', 'maybe', 'not_attending', 'checked_in

-- Indexes
CREATE INDEX idx_attendance_event ON event_attendance(event_id);
CREATE INDEX idx_attendance_user ON event_attendance(user_id);
CREATE INDEX idx_attendance_status ON event_attendance(status);
```

3. Curation Domain Models

Curation Workflow

```
sql
```

```sql
CREATE TABLE curation_workflows (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  event_id UUID NOT NULL REFERENCES events(id) ON DELETE CASCADE,

  -- Current State
  current_stage curation_stage_enum NOT NULL DEFAULT 'ai_prescreening',
  overall_status curation_status_enum NOT NULL DEFAULT 'pending',

  -- AI Pre-screening Results
  ai_screening_completed_at TIMESTAMP WITH TIME ZONE,
  ai_screening_passed BOOLEAN,
  ai_scores JSONB, -- Detailed AI assessment scores
  ai_flags TEXT[] DEFAULT '{}',
  ai_confidence DECIMAL(3,2),

  -- Human Review
  assigned_curator_id UUID REFERENCES users(id),
  assigned_at TIMESTAMP WITH TIME ZONE,
  human_review_completed_at TIMESTAMP WITH TIME ZONE,
  human_review_passed BOOLEAN,
  curator_notes TEXT,
  curator_quality_score INTEGER, -- 1-10 scale

  -- Community Oversight
  community_flags_count INTEGER DEFAULT 0,
  community_reviews_count INTEGER DEFAULT 0,
  community_average_rating DECIMAL(3,2),

  -- Final Results
  final_quality_score DECIMAL(3,2),
  quality_badges TEXT[] DEFAULT '{}',
  rejection_reason TEXT,

  -- Timestamps
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  completed_at TIMESTAMP WITH TIME ZONE,

  UNIQUE(event_id)
);
```

```
CREATE TYPE curation_stage_enum AS ENUM ('ai_prescreening', 'human_review', 'community_oversight', 'comp
CREATE TYPE curation_status_enum AS ENUM ('pending', 'approved', 'rejected', 'needs_revision', 'flagged');
```

Curation Reviews (Detailed Audit Trail)

```sql
CREATE TABLE curation_reviews (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  workflow_id UUID NOT NULL REFERENCES curation_workflows(id) ON DELETE CASCADE,
  reviewer_id UUID NOT NULL REFERENCES users(id),
  reviewer_type reviewer_type_enum NOT NULL,

  -- Review Details
  stage curation_stage_enum NOT NULL,
  decision review_decision_enum NOT NULL,
  quality_score INTEGER, -- 1-10 scale

  -- Detailed Feedback
  content_quality_score INTEGER,
  accuracy_score INTEGER,
  relevance_score INTEGER,
  presentation_score INTEGER,

  -- Notes and Flags
  notes TEXT,
  flags TEXT[] DEFAULT '{}',
  improvement_suggestions TEXT,

  -- Timestamps
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  review_duration_seconds INTEGER,

  CONSTRAINT valid_quality_scores CHECK (
    quality_score IS NULL OR (quality_score BETWEEN 1 AND 10)
  )
);

CREATE TYPE reviewer_type_enum AS ENUM ('ai_system', 'human_curator', 'community_member');
CREATE TYPE review_decision_enum AS ENUM ('approve', 'reject', 'needs_revision', 'flag_for_attention');
```

4. Social Domain Models

Social Connections

sql

```sql
CREATE TABLE social_connections (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    requester_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    addressee_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,

    -- Connection Details
    connection_type connection_type_enum NOT NULL,
    status connection_status_enum NOT NULL DEFAULT 'pending',

    -- Timestamps
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    accepted_at TIMESTAMP WITH TIME ZONE,
    blocked_at TIMESTAMP WITH TIME ZONE,

    -- Metadata
    connection_source VARCHAR(50), -- How they connected
    mutual_friends_count INTEGER DEFAULT 0,

    UNIQUE(requester_id, addressee_id),
    CONSTRAINT no_self_connection CHECK (requester_id != addressee_id)
);

CREATE TYPE connection_type_enum AS ENUM ('friend', 'follow', 'block');
CREATE TYPE connection_status_enum AS ENUM ('pending', 'accepted', 'declined', 'blocked');
```

Activity Feed

sql

```sql
CREATE TABLE activity_feed (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,

    -- Activity Details
    activity_type activity_type_enum NOT NULL,
    entity_type VARCHAR(50) NOT NULL, -- 'event', 'user', 'review', etc.
    entity_id UUID NOT NULL,

    -- Content
    title VARCHAR(200) NOT NULL,
    description TEXT,
    image_url TEXT,
    action_url TEXT,

    -- Targeting
    visibility activity_visibility_enum DEFAULT 'friends',
    target_users UUID[], -- Specific users who should see this

    -- Engagement
    likes_count INTEGER DEFAULT 0,
    comments_count INTEGER DEFAULT 0,
    shares_count INTEGER DEFAULT 0,

    -- Timestamps
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    expires_at TIMESTAMP WITH TIME ZONE,

    -- Additional metadata
    metadata JSONB DEFAULT '{}'
);

CREATE TYPE activity_type_enum AS ENUM (
    'event_created', 'event_attending', 'event_completed', 'event_review',
    'friend_joined', 'achievement_earned', 'milestone_reached'
);
CREATE TYPE activity_visibility_enum AS ENUM ('public', 'friends', 'private');
```

API Design Specifications

API Architecture Overview

The platform uses a **GraphQL Federation** approach with REST fallbacks for simple operations. This provides:

- **Flexibility**: Clients can request exactly the data they need
- **Performance**: Reduces over-fetching and under-fetching
- **Type Safety**: Strong typing with automatic documentation
- **Real-time**: GraphQL subscriptions for live updates

Authentication & Authorization

```typescript
// JWT Token Structure
interface JWTPayload {
  sub: string;         // User ID
  email: string;
  role: UserRole;
  permissions: string[];
  iat: number;         // Issued at
  exp: number;         // Expires at
  aud: string;         // Audience (web/mobile)
}

// API Key Structure for Organizations
interface APIKey {
  id: string;
  organizationId: string;
  name: string;
  permissions: APIPermission[];
  rateLimit: number;
  expiresAt?: Date;
}
```

1. User API

GraphQL Schema

```graphql
```

```graphql
# User Types
type User {
  id: ID!
  email: String!
  displayName: String!
  avatar: String
  bio: String
  location: Location
  preferences: UserPreferences!
  reputation: UserReputation!
  socialConnections: [SocialConnection!]!
  eventsCreated(filter: EventFilter): [Event!]!
  eventsAttending(filter: EventFilter): [Event!]!
  reviews: [Review!]!
  createdAt: DateTime!
  updatedAt: DateTime!
}

type UserPreferences {
  categories: [String!]!
  maxDistance: Int!
  priceRange: PriceRange!
  preferredTimes: [TimePreference!]!
  notifications: NotificationSettings!
}

type UserReputation {
  overallScore: Int!
  creatorScore: Int!
  reviewerScore: Int!
  communityScore: Int!
  badges: [String!]!
  achievements: JSON
  stats: ReputationStats!
}

# Queries
type Query {
  me: User
  user(id: ID!): User
  users(filter: UserFilter, pagination: Pagination): UserPage!
  searchUsers(query: String!, filter: UserFilter): [User!]!
}
```

```
# Mutations
type Mutation {
  # Authentication
  login(input: LoginInput!): AuthResult!
  register(input: RegisterInput!): AuthResult!
  refreshToken(token: String!): AuthResult!
  logout: Boolean!

  # Profile Management
  updateProfile(input: UpdateProfileInput!): User!
  updatePreferences(input: UpdatePreferencesInput!): UserPreferences!
  uploadAvatar(file: Upload!): String!

  # Social Features
  sendFriendRequest(userId: ID!): SocialConnection!
  acceptFriendRequest(connectionId: ID!): SocialConnection!
  unfriend(userId: ID!): Boolean!
  blockUser(userId: ID!): Boolean!
}

# Subscriptions
type Subscription {
  userActivityFeed: ActivityItem!
  friendRequests: SocialConnection!
  notifications: Notification!
}
```

REST Endpoints (Fallback/Simple Operations)

```
typescript
```

```
// User Authentication REST API
POST   /api/auth/login
POST   /api/auth/register
POST   /api/auth/refresh
POST   /api/auth/logout
GET    /api/auth/me

// User Profile REST API
GET    /api/users/:id
PUT    /api/users/:id
DELETE /api/users/:id
POST   /api/users/:id/avatar
GET    /api/users/:id/events
GET    /api/users/:id/reviews

// Example Response Formats
interface LoginResponse {
  success: boolean;
  data: {
    user: User;
    tokens: {
      accessToken: string;
      refreshToken: string;
      expiresIn: number;
    };
  };
  meta: {
    timestamp: string;
    requestId: string;
  };
}
```

2. Event API

GraphQL Schema

```
graphql
```

```
type Event {
  id: ID!
  title: String!
  slug: String!
  description: String!
  shortDescription: String

  # Categorization
  category: Category!
  subcategory: String
  tags: [String!]!

  # Creator and Organization
  creator: User!
  organization: Organization

  # Schedule
  startTime: DateTime!
  endTime: DateTime!
  timezone: String!
  isAllDay: Boolean!
  recurrenceRule: String

  # Location
  venue: Venue
  address: Address
  coordinates: Coordinates
  isOnline: Boolean!
  onlineDetails: OnlineEventDetails

  # Ticketing
  isFree: Boolean!
  ticketPrice: Money
  ticketUrl: String
  capacity: Int
  requiresApproval: Boolean!
  ageRestriction: Int

  # Media
  coverImage: String!
  gallery: [String!]!
  video: String
```

```graphql
  # Status and Curation
  status: EventStatus!
  visibility: EventVisibility!
  curationWorkflow: CurationWorkflow
  qualityScore: Float
  qualityBadges: [String!]!

  # Engagement
  viewCount: Int!
  interestedCount: Int!
  attendingCount: Int!
  shareCount: Int!
  saveCount: Int!

  # Relationships
  attendance(userId: ID): EventAttendance
  attendees(filter: AttendeeFilter): [EventAttendance!]!
  reviews: [Review!]!
  similarEvents: [Event!]!

  # Timestamps
  createdAt: DateTime!
  updatedAt: DateTime!
  publishedAt: DateTime
}

# Queries
type Query {
  event(id: ID, slug: String): Event
  events(filter: EventFilter, sort: EventSort, pagination: Pagination): EventPage!
  searchEvents(query: SearchInput!): EventSearchResult!
  recommendedEvents(userId: ID, limit: Int = 20): [Event!]!
  trendingEvents(location: LocationFilter, timeframe: TimeFrame): [Event!]!
  featuredEvents(limit: Int = 10): [Event!]!
}

# Mutations
type Mutation {
  # Event Management
  createEvent(input: CreateEventInput!): Event!
  updateEvent(id: ID!, input: UpdateEventInput!): Event!
  deleteEvent(id: ID!): Boolean!
  publishEvent(id: ID!): Event!
  cancelEvent(id: ID!, reason: String): Event!
  # Attendance
  rsvpToEvent(eventId: ID!, status: AttendanceStatus!, tickets: Int = 1): EventAttendance!
  checkInToEvent(eventId: ID!): EventAttendance!

  # Engagement
  likeEvent(eventId: ID!): Boolean!
  saveEvent(eventId: ID!): Boolean!
  shareEvent(eventId: ID!, platform: SocialPlatform!): ShareResult!

  # Media
  uploadEventImage(eventId: ID!, file: Upload!, type: ImageType!): String!
  reorderEventGallery(eventId: ID!, imageOrder: [String!]!): Boolean!
}

# Subscriptions
type Subscription {
  eventUpdates(eventId: ID!): Event!
  eventAttendanceUpdates(eventId: ID!): EventAttendance!
  newEventsInArea(location: LocationFilter!): Event!
}
```

Advanced Search API

typescript

```
// Elasticsearch-powered search endpoint
POST /api/search/events
{
  "query": {
    "text": "jazz music",
    "location": {
      "latitude": 40.7128,
      "longitude": -74.0060,
      "radius": "10km"
    },
    "filters": {
      "categories": ["music", "arts"],
      "dateRange": {
        "start": "2024-01-01T00:00:00Z",
        "end": "2024-12-31T23:59:59Z"
      },
      "priceRange": {
        "min": 0,
        "max": 50
      },
      "isFree": null,
      "hasAvailableTickets": true,
      "qualityScore": {
        "min": 4.0
      }
    },
    "sort": [
      { "field": "relevance", "order": "desc" },
      { "field": "startTime", "order": "asc" }
    ],
    "personalization": {
      "userId": "user-123",
      "boost": {
        "categories": 1.5,
        "location": 1.2,
        "social": 1.3
      }
    }
  },
  "pagination": {
    "page": 1,
    "size": 20
  },
  "aggregations": [
    "categories",
    "priceRanges",
    "timeSlots",
    "venues"
  ]
}

interface EventSearchResponse {
  results: Event[];
  totalCount: number;
  facets: {
    categories: { name: string; count: number }[];
    priceRanges: { range: string; count: number }[];
    timeSlots: { slot: string; count: number }[];
    venues: { venue: Venue; count: number }[];
  };
  suggestions: string[];
  searchTime: number;
  personalizationApplied: boolean;
}
```

3. Curation API

GraphQL Schema

```
graphql
```

```graphql
type CurationWorkflow {
  id: ID!
  event: Event!
  currentStage: CurationStage!
  overallStatus: CurationStatus!

  # AI Assessment
  aiScreening: AIScreeningResult

  # Human Review
  assignedCurator: User
  humanReview: HumanReviewResult

  # Community Oversight
  communityReviews: [CommunityReview!]!
  communityFlags: [CommunityFlag!]!

  # Final Results
  finalQualityScore: Float
  qualityBadges: [String!]!
  rejectionReason: String

  # Timeline
  createdAt: DateTime!
  completedAt: DateTime
  estimatedCompletionTime: DateTime
}

type AIScreeningResult {
  passed: Boolean!
  confidence: Float!
  scores: AIScores!
  flags: [String!]!
  completedAt: DateTime!
}

type AIScores {
  completeness: Float!
  contentQuality: Float!
  imageQuality: Float!
  spamProbability: Float!
  duplicateRisk: Float!
}
```

```
# Queries (Admin/Curator Only)
type Query {
  curationQueue(stage: CurationStage, priority: Priority): [CurationWorkflow!]!
  curationWorkflow(id: ID!): CurationWorkflow
  curationStats(timeframe: TimeFrame): CurationStats!
  curatorPerformance(curatorId: ID!, timeframe: TimeFrame): CuratorStats!
}

# Mutations (Admin/Curator Only)
type Mutation {
  # Curator Actions
  claimCurationTask(workflowId: ID!): CurationWorkflow!
  submitCurationReview(input: CurationReviewInput!): CurationWorkflow!
  escalateCurationIssue(workflowId: ID!, reason: String!): Boolean!

  # Community Actions
  flagEvent(eventId: ID!, reason: FlagReason!, details: String): CommunityFlag!
  submitCommunityReview(eventId: ID!, rating: Int!, feedback: String): CommunityReview!

  # Admin Actions
  overrideCurationDecision(workflowId: ID!, decision: CurationStatus!, reason: String!): CurationWorkflow!
  retriggerAIScreening(eventId: ID!): Boolean!
}
```

4. Real-time Features API

WebSocket Connection Management

```typescript
```

```
// WebSocket Event Types
interface WebSocketMessage {
  type: 'subscribe' | 'unsubscribe' | 'message' | 'error' | 'heartbeat';
  channel?: string;
  data?: any;
  timestamp: string;
  requestId?: string;
}

// Real-time Channels
enum RealtimeChannels {
  EVENT_UPDATES = 'event:updates',
  USER_NOTIFICATIONS = 'user:notifications',
  ATTENDANCE_CHANGES = 'event:attendance',
  ACTIVITY_FEED = 'user:activity',
  CURATION_UPDATES = 'curation:updates'
}

// WebSocket Connection Handler
class RealtimeAPI {
  connect(authToken: string): WebSocket;
  subscribe(channel: string, filters?: any): void;
  unsubscribe(channel: string): void;
  send(message: WebSocketMessage): void;

  // Event Handlers
  onMessage(callback: (message: WebSocketMessage) => void): void;
  onError(callback: (error: Error) => void): void;
  onReconnect(callback: () => void): void;
}
```

5. Analytics API

Event Tracking

```typescript
```

```
// Analytics Event Tracking
POST /api/analytics/track
{
  "events": [
    {
      "type": "event_view",
      "eventId": "event-123",
      "userId": "user-456",
      "sessionId": "session-789",
      "properties": {
        "source": "search_results",
        "position": 3,
        "searchQuery": "jazz music"
      },
      "timestamp": "2024-01-15T10:30:00Z"
    }
  ]
}

// Analytics Query API
POST /api/analytics/query
{
  "metrics": ["views", "attendance", "engagement"],
  "dimensions": ["category", "location", "time"],
  "filters": {
    "eventId": ["event-123", "event-456"],
    "dateRange": {
      "start": "2024-01-01",
      "end": "2024-01-31"
    }
  },
  "granularity": "day"
}
```

Data Access Patterns

1. CQRS (Command Query Responsibility Segregation)

```
typescript
```

```
// Command Side - Write Operations
interface EventCommand {
  createEvent(command: CreateEventCommand): Promise<EventId>;
  updateEvent(command: UpdateEventCommand): Promise<void>;
  deleteEvent(command: DeleteEventCommand): Promise<void>;
}

// Query Side - Read Operations
interface EventQuery {
  getEvent(id: EventId): Promise<Event>;
  searchEvents(criteria: SearchCriteria): Promise<EventSearchResult>;
  getEventsByCreator(userId: UserId): Promise<Event[]>;
}

// Event Sourcing for Audit Trail
interface EventStore {
  appendEvents(streamId: string, events: DomainEvent[]): Promise<void>;
  getEvents(streamId: string, fromVersion?: number): Promise<DomainEvent[]>;
}
```

2. Repository Pattern

typescript

```typescript
// Generic Repository Interface
interface Repository<T, ID> {
  findById(id: ID): Promise<T | null>;
  findAll(criteria?: FilterCriteria): Promise<T[]>;
  save(entity: T): Promise<T>;
  delete(id: ID): Promise<void>;
  count(criteria?: FilterCriteria): Promise<number>;
}

// Event Repository Implementation
class EventRepository implements Repository<Event, EventId> {
  constructor(
    private db: DatabaseConnection,
    private cache: CacheService,
    private search: SearchService
  ) {}

  async findById(id: EventId): Promise<Event | null> {
    // Try cache first
    const cached = await this.cache.get(`event:${id}`);
    if (cached) return cached;

    // Query database
    const event = await this.db.query(
      'SELECT * FROM events WHERE id = $1 AND deleted_at IS NULL',
      [id]
    );

    if (event) {
      // Cache for future requests
      await this.cache.set(`event:${id}`, event, { ttl: 300 });
    }

    return event;
  }

  async searchEvents(criteria: SearchCriteria): Promise<EventSearchResult> {
    // Use Elasticsearch for complex searches
    return await this.search.search('events', {
      query: this.buildSearchQuery(criteria),
      sort: criteria.sort,
      pagination: criteria.pagination
    });
  }
}
```

3. Caching Strategy

typescript

```
// Multi-layer Caching Strategy
interface CacheStrategy {
  // L1: Application Memory Cache (fastest)
  memoryCache: Map<string, any>;

  // L2: Redis Cache (shared across instances)
  redisCache: RedisClient;

  // L3: CDN Cache (for static content)
  cdnCache: CDNService;
}

class EventCacheService {
  private readonly TTL = {
    EVENT_DETAIL: 300,      // 5 minutes
    EVENT_LIST: 60,         // 1 minute
    USER_PROFILE: 600,      // 10 minutes
    SEARCH_RESULTS: 30,     // 30 seconds
    STATIC_CONTENT: 86400   // 24 hours
  };

  async getEvent(id: string): Promise<Event | null> {
    // L1 Cache
    if (this.memoryCache.has(`event:${id}`)) {
      return this.memoryCache.get(`event:${id}`);
    }

    // L2 Cache
    const cached = await this.redisCache.get(`event:${id}`);
    if (cached) {
      this.memoryCache.set(`event:${id}`, cached);
      return cached;
    }

    return null;
  }

  async setEvent(id: string, event: Event): Promise<void> {
    // Cache invalidation strategy
    const cacheKeys = [
      `event:${id}`,
      `events:creator:${event.creatorId}`,
      `events:category:${event.categoryId}`,
      `search:events:*` // Wildcard invalidation
    ];

    // Update all cache layers
    this.memoryCache.set(`event:${id}`, event);
    await this.redisCache.setex(`event:${id}`, this.TTL.EVENT_DETAIL, event);

    // Invalidate related caches
    await this.invalidateKeys(cacheKeys);
  }
}
```

4. Database Query Optimization

```sql
```

```sql
-- Optimized Event Search Query with Spatial Index
WITH nearby_events AS (
  SELECT e.id, e.title, e.start_time, e.coordinates,
         ST_Distance(e.coordinates, ST_MakePoint($longitude, $latitude)) as distance
  FROM events e
  WHERE ST_DWithin(e.coordinates, ST_MakePoint($longitude, $latitude), $radius_meters)
    AND e.status = 'published'
    AND e.start_time > NOW()
    AND ($category_filter IS NULL OR e.category_id = ANY($category_filter))
),
filtered_events AS (
  SELECT ne.*,
         ts_rank(to_tsvector('english', e.title || ' ' || e.description),
                 plainto_tsquery('english', $search_query)) as text_rank,
         (CASE
           WHEN e.featured THEN 1.5
           ELSE 1.0
         END) * (1.0 - (ne.distance / $max_distance)) as location_score
  FROM nearby_events ne
  JOIN events e ON ne.id = e.id
  WHERE ($search_query IS NULL OR
         to_tsvector('english', e.title || ' ' || e.description) @@ plainto_tsquery('english', $search_query))
)
SELECT *, (text_rank * 0.4 + location_score * 0.6) as final_score
FROM filtered_events
ORDER BY final_score DESC, start_time ASC
LIMIT $limit OFFSET $offset;

-- Composite Indexes for Performance
CREATE INDEX CONCURRENTLY idx_events_search_composite
ON events (status, start_time, category_id)
WHERE deleted_at IS NULL;

CREATE INDEX CONCURRENTLY idx_events_location_time
ON events USING GIST (coordinates, tsrange(start_time, end_time))
WHERE status = 'published';

CREATE INDEX CONCURRENTLY idx_events_fulltext
ON events USING GIN (to_tsvector('english', title || ' ' || description));
```

API Versioning & Backwards Compatibility

GraphQL Schema Evolution

```graphql
# Version 1.0 - Initial Schema
type Event {
  id: ID!
  title: String!
  description: String!
  startTime: DateTime!
}

# Version 1.1 - Additive Changes (Non-breaking)
type Event {
  id: ID!
  title: String!
  description: String!
  startTime: DateTime!
  # New fields added
  shortDescription: String    # Nullable, so non-breaking
  tags: [String!]! @since(version: "1.1")
  qualityScore: Float @since(version: "1.1")
}

# Version 2.0 - Breaking Changes
type Event {
  id: ID!
  title: String!
  description: String!
  # BREAKING: Changed from DateTime to custom type
  schedule: EventSchedule! @since(version: "2.0")
  # BREAKING: Removed field
  # startTime: DateTime! @deprecated(reason: "Use schedule.startTime instead")
}

type EventSchedule {
  startTime: DateTime!
  endTime: DateTime!
  timezone: String!
  recurrence: RecurrenceRule
}
```

REST API Versioning

typescript

```typescript
// URL-based versioning for major changes
// /api/v1/events
// /api/v2/events

// Header-based versioning for minor changes
// Accept: application/json; version=1.1

interface APIVersioning {
  supportedVersions: string[];
  defaultVersion: string;
  deprecationPolicy: {
    warningPeriod: number; // months
    sunsetPeriod: number;  // months
  };
}

// Backwards compatibility middleware
class VersioningMiddleware {
  async handleRequest(req: Request, res: Response, next: NextFunction) {
    const version = this.extractVersion(req);
    const transformer = this.getResponseTransformer(version);

    // Intercept response to transform based on version
    const originalSend = res.send;
    res.send = function(data) {
      const transformedData = transformer.transform(data);
      return originalSend.call(this, transformedData);
    };

    next();
  }
}
```

Error Handling & Validation

Comprehensive Error Schema

typescript

```typescript
// Standardized Error Response Format
interface APIError {
  error: {
    code: string;          // Machine-readable error code
    message: string;       // Human-readable error message
    details?: any;         // Additional error context
    timestamp: string;     // ISO 8601 timestamp
    requestId: string;     // Unique request identifier
    path?: string;         // GraphQL path or REST endpoint
    extensions?: {         // Additional metadata
      classification: 'CLIENT_ERROR' | 'SERVER_ERROR' | 'NETWORK_ERROR';
      retryable: boolean;
      documentation: string;
    };
  };
  meta: {
    version: string;
    rateLimit?: {
      remaining: number;
      resetTime: string;
    };
  };
}

// Error Codes Enum
enum ErrorCodes {
  // Authentication & Authorization
  UNAUTHORIZED = 'UNAUTHORIZED',
  FORBIDDEN = 'FORBIDDEN',
  TOKEN_EXPIRED = 'TOKEN_EXPIRED',

  // Validation Errors
  VALIDATION_ERROR = 'VALIDATION_ERROR',
  INVALID_INPUT = 'INVALID_INPUT',
  CONSTRAINT_VIOLATION = 'CONSTRAINT_VIOLATION',

  // Resource Errors
  NOT_FOUND = 'NOT_FOUND',
  ALREADY_EXISTS = 'ALREADY_EXISTS',
  RESOURCE_CONFLICT = 'RESOURCE_CONFLICT',

  // Business Logic Errors
  EVENT_FULL = 'EVENT_FULL',
  EVENT_CANCELLED = 'EVENT_CANCELLED',
  REGISTRATION_CLOSED = 'REGISTRATION_CLOSED',
  INSUFFICIENT_PERMISSIONS = 'INSUFFICIENT_PERMISSIONS',

  // External Service Errors
  PAYMENT_FAILED = 'PAYMENT_FAILED',
  EMAIL_DELIVERY_FAILED = 'EMAIL_DELIVERY_FAILED',
  SOCIAL_MEDIA_ERROR = 'SOCIAL_MEDIA_ERROR',

  // System Errors
  INTERNAL_ERROR = 'INTERNAL_ERROR',
  SERVICE_UNAVAILABLE = 'SERVICE_UNAVAILABLE',
  RATE_LIMIT_EXCEEDED = 'RATE_LIMIT_EXCEEDED'
}
```

Input Validation Schema

typescript

```
// Joi Validation Schemas
const EventValidationSchema = Joi.object({
  title: Joi.string()
    .min(3)
    .max(200)
    .required()
    .messages({
      'string.min': 'Event title must be at least 3 characters long',
      'string.max': 'Event title cannot exceed 200 characters',
      'any.required': 'Event title is required'
    }),

  description: Joi.string()
    .min(50)
    .max(5000)
    .required()
    .messages({
      'string.min': 'Event description must be at least 50 characters long'
    }),

  startTime: Joi.date()
    .iso()
    .min('now')
    .required()
    .messages({
      'date.min': 'Event start time must be in the future'
    }),

  endTime: Joi.date()
    .iso()
    .greater(Joi.ref('startTime'))
    .required()
    .messages({
      'date.greater': 'Event end time must be after start time'
    }),

  coordinates: Joi.object({
    latitude: Joi.number().min(-90).max(90).required(),
    longitude: Joi.number().min(-180).max(180).required()
  }).required(),

  capacity: Joi.number()
    .integer()
    .min(1)
    .max(100000)
    .optional(),

  ticketPrice: Joi.number()
    .precision(2)
    .min(0)
    .max(10000)
    .when('isFree', {
      is: false,
      then: Joi.required(),
      otherwise: Joi.forbidden()
    }),

  tags: Joi.array()
    .items(Joi.string().max(50))
    .max(10)
    .unique()
    .optional()
});

// GraphQL Input Validation
const validateGraphQLInput = (schema: Joi.Schema) => {
  return (target: any, propertyName: string, descriptor: PropertyDescriptor) => {
    const method = descriptor.value;
    descriptor.value = async function (...args: any[]) {
      const [, input] = args;
      const { error, value } = schema.validate(input, {
        abortEarly: false,
        stripUnknown: true
      });

      if (error) {
        throw new ValidationError('Input validation failed', error.details);
      }

      return method.apply(this, [args[0], value, ...args.slice(2)]);
    };
  };
};
```

Performance Optimization Strategies

Database Connection Pooling

typescript

```
// PostgreSQL Connection Pool Configuration
const poolConfig = {
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  host: process.env.DB_HOST,
  database: process.env.DB_NAME,
  port: parseInt(process.env.DB_PORT || '5432'),

  // Pool settings
  max: 20,                    // Maximum number of connections
  min: 5,                     // Minimum number of connections
  idleTimeoutMillis: 30000,   // Close idle connections after 30s
  connectionTimeoutMillis: 2000, // Fail fast if can't connect

  // Advanced settings
  acquireTimeoutMillis: 60000,
  createTimeoutMillis: 3000,
  destroyTimeoutMillis: 5000,
  reapIntervalMillis: 1000,
  createRetryIntervalMillis: 200
};

// Read/Write Splitting
class DatabaseManager {
  private writePool: Pool;
  private readPools: Pool[];

  constructor() {
    this.writePool = new Pool({ ...poolConfig, host: 'primary-db' });
    this.readPools = [
      new Pool({ ...poolConfig, host: 'replica-1' }),
      new Pool({ ...poolConfig, host: 'replica-2' }),
      new Pool({ ...poolConfig, host: 'replica-3' })
    ];
  }

  getWriteConnection(): Pool {
    return this.writePool;
  }

  getReadConnection(): Pool {
    // Round-robin load balancing
    const index = Math.floor(Math.random() * this.readPools.length);
    return this.readPools[index];
  }
}
```

Query Optimization Patterns

typescript

```
// DataLoader for N+1 Query Prevention
class EventDataLoader {
  private eventLoader = new DataLoader(async (eventIds: string[]) => {
    const events = await this.db.query(`
      SELECT * FROM events
      WHERE id = ANY($1) AND deleted_at IS NULL
    `, [eventIds]);

    // Maintain order matching input
    return eventIds.map(id =>
      events.find(event => event.id === id) || null
    );
  });

  private attendanceLoader = new DataLoader(async (eventIds: string[]) => {
    const attendance = await this.db.query(`
      SELECT event_id, COUNT(*) as count
      FROM event_attendance
      WHERE event_id = ANY($1) AND status = 'attending'
      GROUP BY event_id
    `, [eventIds]);

    return eventIds.map(id => {
      const record = attendance.find(a => a.event_id === id);
      return record ? parseInt(record.count) : 0;
    });
  });

  async getEvent(id: string): Promise<Event> {
    return this.eventLoader.load(id);
  }

  async getAttendanceCount(eventId: string): Promise<number> {
    return this.attendanceLoader.load(eventId);
  }
}

// Pagination with Cursor-based Strategy
interface CursorPagination {
  first?: number;    // Limit
  after?: string;    // Cursor for next page
  last?: number;     // Limit for reverse pagination
  before?: string;   // Cursor for previous page
}

class PaginationService {
  async getEventPage(criteria: SearchCriteria, pagination: CursorPagination) {
    const limit = pagination.first || 20;
    const cursor = pagination.after ?
      this.decodeCursor(pagination.after) : null;

    const query = `
      SELECT *, (start_time::text || '|' || id::text) as cursor
      FROM events
      WHERE ($1::timestamp IS NULL OR start_time > $1)
        AND ($2::uuid IS NULL OR id > $2)
        AND status = 'published'
      ORDER BY start_time ASC, id ASC
      LIMIT $3
    `;

    const events = await this.db.query(query, [
      cursor?.timestamp,
      cursor?.id,
      limit + 1 // Fetch one extra to determine if there are more pages
    ]);

    const hasNextPage = events.length > limit;
    const edges = events.slice(0, limit).map(event => ({
      node: event,
      cursor: this.encodeCursor({ timestamp: event.start_time, id: event.id })
    }));

    return {
      edges,
      pageInfo: {
        hasNextPage,
        hasPreviousPage: !!pagination.after,
        startCursor: edges[0]?.cursor,
        endCursor: edges[edges.length - 1]?.cursor
      }
    };
  }
}
```

Caching Invalidation Strategies

typescript

```typescript
// Event-driven Cache Invalidation
class CacheInvalidationService {
  constructor(
    private redis: RedisClient,
    private eventBus: EventBus
  ) {
    this.setupEventHandlers();
  }

  private setupEventHandlers() {
    this.eventBus.on('event.created', this.handleEventCreated.bind(this));
    this.eventBus.on('event.updated', this.handleEventUpdated.bind(this));
    this.eventBus.on('user.updated', this.handleUserUpdated.bind(this));
  }

  private async handleEventUpdated(event: EventUpdatedEvent) {
    const invalidationKeys = [
      `event:${event.eventId}`,
      `events:creator:${event.creatorId}`,
      `events:category:${event.categoryId}`,
      `search:events:*`,
      `recommendations:*`,
      `trending:events:*`
    ];

    // Parallel invalidation
    await Promise.all([
      this.invalidateKeys(invalidationKeys),
      this.invalidateSearchCache(event.eventId),
      this.updateRecommendationCache(event.eventId)
    ]);
  }

  private async invalidateKeys(patterns: string[]) {
    for (const pattern of patterns) {
      if (pattern.includes('*')) {
        // Handle wildcard patterns
        const keys = await this.redis.keys(pattern);
        if (keys.length > 0) {
          await this.redis.del(...keys);
        }
      } else {
        await this.redis.del(pattern);
      }
    }
  }
}
```

Security Considerations

Rate Limiting Implementation

```
typescript
```

```typescript
// Multi-tier Rate Limiting
interface RateLimitConfig {
  windowMs: number;      // Time window in milliseconds
  maxRequests: number;   // Max requests per window
  skipSuccessfulRequests?: boolean;
  skipFailedRequests?: boolean;
  keyGenerator?: (req: Request) => string;
}

const rateLimitTiers = {
  // Anonymous users
  anonymous: {
    windowMs: 15 * 60 * 1000, // 15 minutes
    maxRequests: 100
  },

  // Authenticated users
  authenticated: {
    windowMs: 15 * 60 * 1000,
    maxRequests: 1000
  },

  // Premium users
  premium: {
    windowMs: 15 * 60 * 1000,
    maxRequests: 5000
  },

  // API keys
  apiKey: {
    windowMs: 60 * 1000, // 1 minute
    maxRequests: 10000
  }
};

// Endpoint-specific rate limits
const endpointLimits = {
  'POST /api/events': { windowMs: 60000, maxRequests: 10 },
  'POST /api/auth/login': { windowMs: 300000, maxRequests: 5 },
  'GET /api/search/*': { windowMs: 60000, maxRequests: 100 }
};
```

Input Sanitization & XSS Prevention

typescript

```
// Comprehensive Input Sanitization
import DOMPurify from 'isomorphic-dompurify';
import validator from 'validator';

class InputSanitizer {
  sanitizeHTML(input: string): string {
    return DOMPurify.sanitize(input, {
      ALLOWED_TAGS: ['p', 'br', 'strong', 'em', 'u', 'ol', 'ul', 'li'],
      ALLOWED_ATTR: [],
      KEEP_CONTENT: true
    });
  }

  sanitizeString(input: string): string {
    return validator.escape(validator.trim(input));
  }

  validateEmail(email: string): boolean {
    return validator.isEmail(email) && email.length <= 255;
  }

  validateURL(url: string): boolean {
    return validator.isURL(url, {
      protocols: ['http', 'https'],
      require_protocol: true,
      require_valid_protocol: true
    });
  }

  sanitizeEventInput(input: CreateEventInput): CreateEventInput {
    return {
      ...input,
      title: this.sanitizeString(input.title),
      description: this.sanitizeHTML(input.description),
      shortDescription: input.shortDescription ?
        this.sanitizeString(input.shortDescription) : undefined,
      tags: input.tags?.map(tag => this.sanitizeString(tag))
    };
  }
}
```

SQL Injection Prevention

typescript

```typescript
// Parameterized Query Builder
class QueryBuilder {
  private query: string = '';
  private parameters: any[] = [];
  private parameterIndex: number = 1;

  select(fields: string[]): this {
    this.query += `SELECT ${fields.join(', ')} `;
    return this;
  }

  from(table: string): this {
    this.query += `FROM ${this.escapeIdentifier(table)} `;
    return this;
  }

  where(condition: string, value?: any): this {
    if (this.query.includes('WHERE')) {
      this.query += 'AND ';
    } else {
      this.query += 'WHERE ';
    }

    if (value !== undefined) {
      this.query += condition.replace('?', `${this.parameterIndex}`);
      this.parameters.push(value);
      this.parameterIndex++;
    } else {
      this.query += condition;
    }
    this.query += ' ';
    return this;
  }

  private escapeIdentifier(identifier: string): string {
    return `"${identifier.replace(/"/g, '""')}"`;
  }

  build(): { query: string; parameters: any[] } {
    return {
      query: this.query.trim(),
      parameters: this.parameters
    };
  }
}

// Usage Example
const searchEvents = async (criteria: SearchCriteria) => {
  const qb = new QueryBuilder()
    .select(['id', 'title', 'start_time', 'coordinates'])
    .from('events')
    .where('status = ?', 'published')
    .where('start_time > ?', new Date())
    .where('deleted_at IS NULL');

  if (criteria.category) {
    qb.where('category_id = ?', criteria.category);
  }

  if (criteria.location) {
    qb.where('ST_DWithin(coordinates, ST_MakePoint(?, ?), ?)',
      criteria.location.longitude,
      criteria.location.latitude,
      criteria.location.radius * 1000 // Convert to meters
    );
  }

  const { query, parameters } = qb.build();
  return await db.query(query, parameters);
};
```