James Kirby A. Orias **ITE 18 -EJ1**

APP PLAN DOCUMENT

1. Comprehensive App Plan

1.1. Core Vision and Use Cases

- Public users can browse events by category, venue, and date, view event details, and purchase tickets.
- Registered users can manage their profile, view their tickets, and access QR or booking references.
- Admin users can manage events, venues, categories, and see ticket sales and payment summaries.
- The system records all tickets and payments, ensuring ticket availability is respected.

Primary actors:

- User (authenticated attendee)
- Admin (system manager)

Main flows:

1. Guest browsing and registration
2. User login and profile management
3. Event discovery and search
4. Ticket purchase and payment
5. Admin event and catalog management
6. Admin reporting (ticket sales, revenue, capacity utilization)

1.2. High-Level Architecture

Backend:

- Laravel API serving JSON (RESTful endpoints).
- Models: User, Admin, Event, Category, Venue, Ticket, Payment.

- Authentication:
    - User auth via Laravel Sanctum (token-based or session depending on UI).
    - Admin auth via a separate guard/table (admins).
- Business logic:
    - Ticket availability based on Event.total_tickets - number of issued tickets.
    - Payment records per ticket with status tracking.

Frontend (Next.js UI folder you already have):

- Next.js app consuming the Laravel API.
- Pages/sections:
    - Landing page with featured events and categories.
    - Event listing and detail pages.
    - Authentication pages (login/register) for users and admins (separate route/area).
    - User dashboard (my tickets, profile).
    - Admin dashboard (events, venues, categories, sales).

Infrastructure:

- Database: MySQL/MariaDB (already implied by XAMPP).
- Storage: Local disk for uploads (event images), with option to move to S3 later.
- Caching: Laravel cache (for event lists, categories, venues).
- Logging: Laravel logs for errors and payment callbacks.

1.3. Functional Modules and Features

1.3.1. Authentication and Authorization

- User registration:
    - Fields: username, email, password.
    - Email uniqueness enforced.
- User login:
    - Token (Sanctum) or session-based depending on integration with Next.js.

- Admin login:
    - Separate credentials in admins table with role and is_active flag.
    - Admin guard and middleware for admin routes.
- Role-based access:
    - Public: browse events and categories only (no purchase history).
    - User: purchase tickets, access user-specific endpoints.
    - Admin: CRUD for events, venues, categories; view sales and user ticket info.

1.3.2. Event Catalog

- Category management (admin-side):
    - Create, edit, archive/soft-delete categories.
    - Fields: category_name, icon, description.
- Venue management (admin-side):
    - Create, edit venues with capacity constraints.
    - Fields: venue_name, address, capacity.
- Event management (admin-side):
    - CRUD for events:
        - admin_id (owner/creator).
        - event_name, event_date, event_time.
        - category_id, venue_id.
        - description, base_price, image_url, total_tickets.
    - Validation:
        - Event date/time must be in the future upon creation.
        - total_tickets <= venue.capacity.
    - Event status can be inferred (upcoming, past) from date/time.
- Event discovery (public/user):
    - List all upcoming events, with filters:
        - By category, by venue, by date range.

- Event detail page:
    - Shows description, location, remaining tickets (derived), price, image.

1.3.3. Ticketing

- Ticket purchase flow:
    1. User selects an event and quantity (if you support quantity > 1 per order; current Ticket model is one record per ticket; you can still loop to create multiple).
    2. Backend checks available tickets using Event.getAvailableTicketsAttribute.
    3. Reserve tickets (create Ticket records linked to user and event).
    4. Initiate payment (online or offline depending on your chosen gateway).
    5. On successful payment, mark Payment.status as success and keep tickets active.
    6. On failed/canceled payment, either:
        - Delete tickets, or
        - Mark them as not valid / release back to inventory.
- Ticket model responsibilities:
    - Associates User and Event.
    - Holds purchase_date.
    - Has one Payment.
    - Can be extended with:
        - Unique ticket_code / QR_code.
        - seat_number (if you want seat management later).
        - status (active, canceled, refunded).
- User ticket management:
    - List user's tickets with event summary and payment status.
    - Ticket detail endpoint with event info and QR/confirmation.

1.3.4. Payments

- Payment creation:
    - When a ticket (or tickets) is created, create a Payment record:
        - ticket_id
        - amount
        - payment_date
        - payment_method (e.g., card, PayPal, etc.)
        - status (pending, success, failed, refunded).
- Integration with gateway:
    - Create payment initialization endpoint (e.g., POST /api/payments/initiate).
    - Webhook or callback route to update Payment.status.
- Reporting:
    - Aggregate revenue per event, per category, per period.
    - Admin can filter payments by status and date.

1.3.5. Admin Dashboard Functions

- Event dashboard:
    - List all events with:
        - number of tickets sold
        - remaining capacity
        - total revenue (sum of successful payments).
- Category and venue dashboards:
    - Simple CRUD.
    - Stats like number of events per category, events per venue.
- User management:
    - View users and tickets purchased.
- Admin auditing:
    - last_login_at tracking is already in Admin.
    - Optionally track which admin created/updated events.

1.4. Non-Functional Requirements

- Security:
    - Use Sanctum for API protection.
    - Hash passwords (already configured via casts in User and Admin).
    - Validate all input; rate-limit login and payment endpoints.
- Performance:
    - Use pagination for event and ticket lists.
    - Use eager loading to avoid N+1 problems (events with category, venue, tickets).
- Reliability:
    - Transaction management around ticket + payment record creation to avoid partial writes.
    - Graceful handling of payment failures; idempotent webhook handling.
- Maintainability:
    - Clear separation of layers:
        - Controllers, Services (business logic), Models, Resources/Transformers.