Event Booking System — Project Documentation

Contents

Overview

An event management and booking web application built with Laravel. Organizers create and manage events; Attendees register, browse upcoming events, and book spots. Includes authentication, CRUD, raw SQL reporting, manual capacity validation, automated testing, and advanced waitlist features.

Quick Start

Requirements

- PHP ≥ 11
- SQLite/MySQL

Installation & Setup

```
# Clone the repo
        git clone https://github.com/ayakoneko/EventBookingLaravel.git
        cd EventBookingLaravel

# Install dependencies
        composer install
        npm install && npm run dev

# Set up environment
        cp .env.example .env
        php artisan key:generate

# Configure DB in .env
        DB_DATABASE=eventbooking
        DB_USERNAME=root
        DB_PASSWORD=secret

# Run migrations and seeders
        php artisan migrate –seed

# Run in Server
        php artisan serve

# Run Laravel task scheduler
        php artisan schedule:work
```

Project Summary (Role-based)

Roles & Access

- User types: Organizer, Attendee, Guest
- Auth: Email/password login & logout
- Navbar: Show name + role
- Middleware:
  - EnsureUserIsAttendee → attendee-only features
  - EnsureUserIsOrganizer → organizer-only features

Guest (Unauthenticated)

- Browse upcoming events (8 per page, paginated)
- View event details (title, description, time, location, organizer, remaining spots)
- Register with privacy consent
- Login/logout

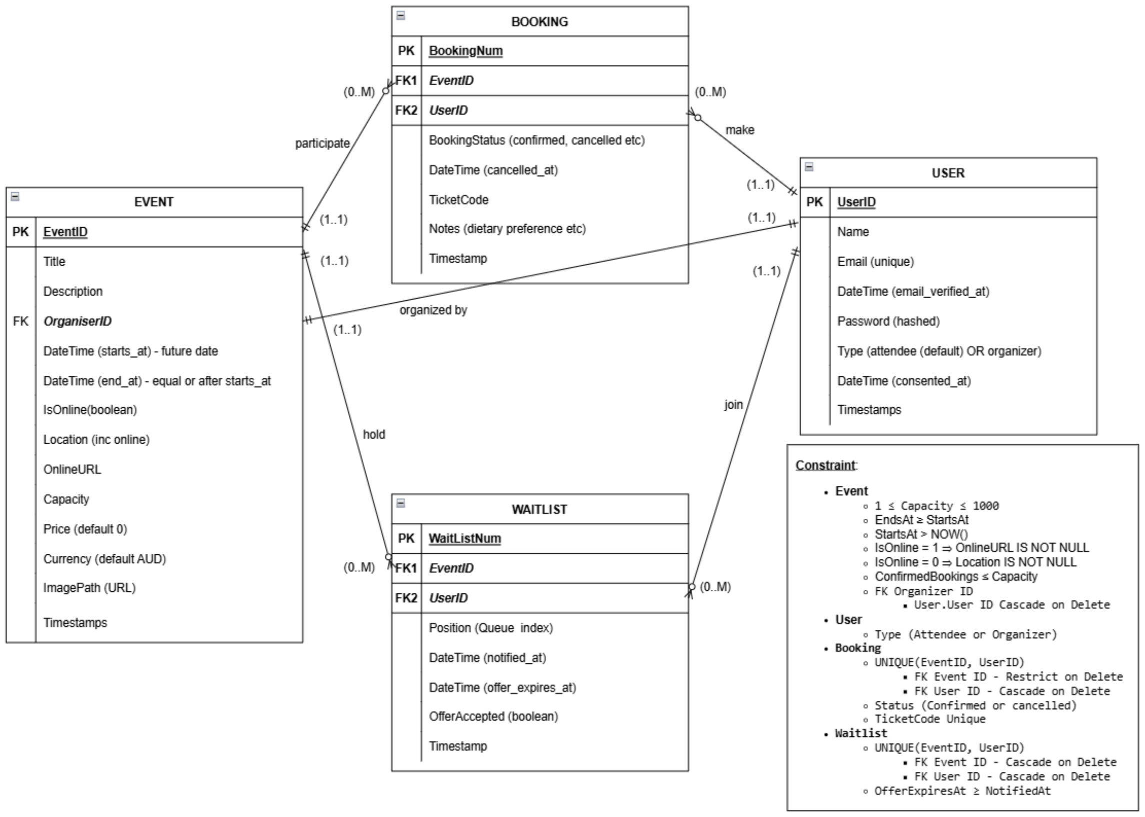
Attendee

- Register/Login with consent checkbox
- Book/Cancel seat
  - Book if seats available
  - No double-booking
  - Cancel anytime → frees seat & triggers waitlist offer
- Waitlist Join/Leave
  - Join if event full
  - Position shown, leave anytime
- Claim Timed Offer
  - Email with token + expiry
  - Claim seat on event page
  - Race-safe, single-use
- Dashboards
  - My Bookings → active upcoming events (cancel allowed)
  - My Waitlists → show queue position & active offers

Organizer

- Event CRUD
  - Create/update/delete with validation (capacity, price, schedule)
  - Cannot reduce capacity below confirmed bookings
- Automated Waitlist Offers
  - Trigger on cancellation or increased capacity
  - FIFO queue, one offer at a time
  - Laravel Scheduler runs WaitlistSweeper to expire offers & promote next
- Dashboards (Using raw SQL query)
  - Event dashboard → capacity, bookings, remaining spots, waitlist count
  - Admin waitlist view → see queue, active offers, expirations

Domain Model



<u>Constraint</u>:

- **Event**
  - 1 ≤ Capacity ≤ 1000
  - EndsAt ≥ StartsAt
  - StartsAt > NOW()
  - IsOnline = 1 ⇒ OnlineURL IS NOT NULL
  - IsOnline = 0 ⇒ Location IS NOT NULL
  - ConfirmedBookings ≤ Capacity
  - FK Organizer ID
    - User.User ID Cascade on Delete
- **User**
  - Type (Attendee or Organizer)
- **Booking**
  - UNIQUE(EventID, UserID)
    - FK Event ID - Restrict on Delete
    - FK User ID - Cascade on Delete
  - Status (Confirmed or cancelled)
  - TicketCode Unique
- **Waitlist**
  - UNIQUE(EventID, UserID)
    - FK Event ID - Cascade on Delete
    - FK User ID - Cascade on Delete
  - OfferExpiresAt ≥ NotifiedAt

User

Types: attendee, organizer. Registration requires explicit consent to Terms/Privacy. Role governs access to event CRUD and admin views.

Event

Fields: title, description, starts_at, ends_at, is_online, location, online_url, capacity, price_cents, image_path, organizer_id.

Booking

Represents an attendee reservation with status (confirmed/cancelled), ticket_code, cancelled_at.

Waitlist

Queue per event with ordered positions; supports join/leave, admin view, notifications and timed offers.

HTTP Surface (High-level)

Public

- GET /events — List upcoming (8 per page)
- GET /events/{event} — Event details

Authenticated Attendee

- POST /events/{event}/book — Book a seat
- DELETE /bookings/{booking} — Cancel booking (may trigger next waitlist offer)
- GET /my-bookings — View my bookings
- POST /events/{event}/waitlist — Join waitlist
- GET /my-waitlist — View my waitlist
- DELETE /events/{event}/waitlist — Leave waitlist

Organizer

- GET/POST/PUT/DELETE /events — CRUD (owner only)
- GET /organizer/dashboard — Dashboard for owned events
- GET /events/{event}/waitlist/admin — Admin view for owned event, waitlist management

Validation & Business Rules

Event Creation (EventController@store / @update)

Rules:

- title: required, string, max:100
- description: nullable, string, max:1000
- starts_at: required, date, date_format:Y-m-d\TH:i, after:now
- ends_at: nullable, date, date_format:Y-m-d\TH:i, after_or_equal:starts_at
- is_online: required, in:0,1
- location: required_if:is_online,0 | nullable, string, max:255
- online_url: required_if:is_online,1 | nullable, url, max:255
- capacity: required, integer, between:1,1000
- price_cents: required, integer, min:0
- image_path: nullable, string, max:2048

Custom messages:

- starts_at.after: "The start time must be in the future."
- ends_at.after_or_equal: "The end time must be after the start time."
- location.required_if: "Location is required for in-person events."
- online_url.required_if: "Online URL is required when the event is online."

Additional behavior:

- If is_online=true, location is normalised to "Online"
- Ownership set to current organizer on create.
- Delete is blocked if any bookings exist; otherwise redirect with success.

Event Booking (BookingController@store)

Rules & gates:

- Uniqueness: user_id must be unique in bookings for the given event where status=confirmed (prevents duplicate confirmed bookings).
- Capacity: if event->isFull() and user is not the current offeree (activeOffer), request fails with 'Sorry, this event is full.'
- Hold/Offer: if an active offer exists for another user, block with a message indicating the offer expiry timestamp.
- On success: firstOrNew(event_id,user_id), set status=confirmed, clear cancelled_at, assign ticket_code if missing, and save.
- If the user had a waitlist entry: delete it and decrement positions behind to compact the queue.

Booking Cancellation (BookingController@destroy)

Rules:

- Flip confirmed → cancelled within a transaction and set cancelled_at=now().
- If no active offer exists for the event, notify (via WaitlistMail) the first waitlisted user by setting notified_at=now() and offer_expires_at=now()+10 minutes.

Join Waitlist (WaitlistController@store)

Rules:

- Eligibility: Only when remaining seats == 0 OR when there is an active offer for another user.
- Block if user already has a confirmed booking for the event.
- If already on waitlist: respond with success copy 'You are already on the waitlist.' (idempotent UI).
- Assign next position = max(position)+1 and create the entry.

Leave Waitlist (WaitlistController@destroy)

Rules:

- Delete the current user's waitlist entry then decrement positions of all entries behind it to keep the queue contiguous.

Error Semantics

- 401 Unauthorized — guest attempting protected action
- 403 Forbidden — authenticated but not the organizer/owner
- 409 Conflict — duplicate waitlist or booking when full / other conflicts
- 422 Unprocessable Entity — validation failures
- 302 Found — redirects to login for guests on protected routes

Automated Feature Testing

Guest Access

- Can view events, paginate listings, and see event details.
- Redirected when accessing protected routes.

Attendee Actions

- Register, login/logout, book events, view bookings.
- Validation: Cannot double-book or book full events.

Organiser Actions

- Login and access dashboard.
- Create, update, and delete their own events.
- Restricted from editing or deleting others' events or events with bookings.

User Registration

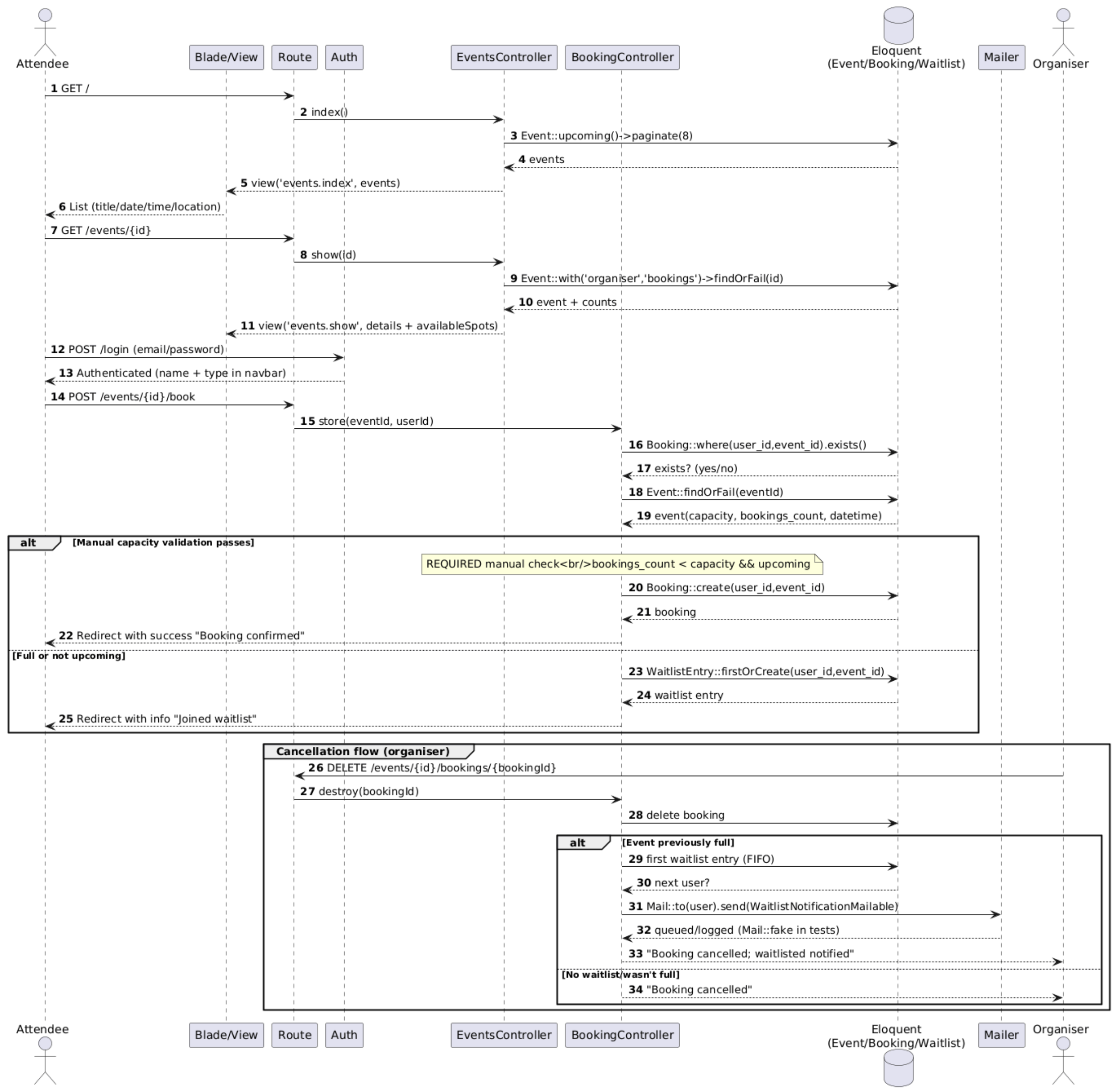
- Cannot register without agreeing to Privacy Policy.

Advanced Feature - Waitlist

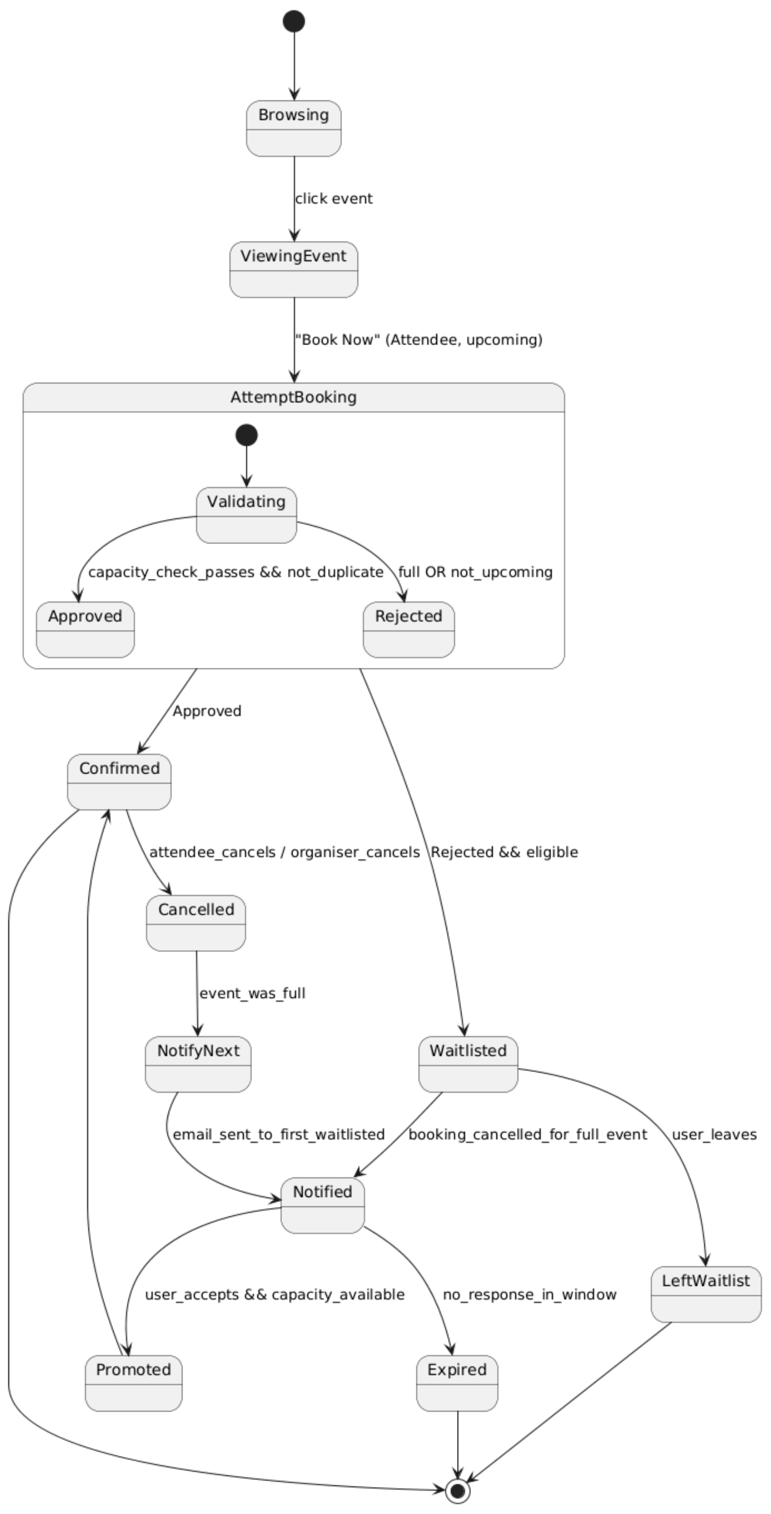
- Can join a waitlist if an event is full.
- Can view and leave their waitlists.
- Can view the waitlist for their events (restricted to the event owner).
- Cannot join waitlist if seats are available or if already booked/joined.
- Leaving waitlist compacts queue positions.

Diagrams

Sequence Diagram


Attendee
Blade/View
Route
Auth
EventsController
BookingController
Eloquent (Event/Booking/Waitlist)
Mailer
Organiser
1 GET /
2 index()
3 Event::upcoming()->paginate(8)
4 events
5 view('events.index', events)
6 List (title/date/time/location)
7 GET /events/{id}
8 show(id)
9 Event::with('organiser','bookings')->findOrFail(id)
10 event + counts
11 view('events.show', details + availableSpots)
12 POST /login (email/password)
13 Authenticated (name + type in navbar)
14 POST /events/{id}/book
15 store(eventId, userId)
16 Booking::where(user_id,event_id).exists()
17 exists? (yes/no)
18 Event::findOrFail(eventId)
19 event(capacity, bookings_count, datetime)
alt [Manual capacity validation passes]
REQUIRED manual check<br/>bookings_count < capacity && upcoming
20 Booking::create(user_id,event_id)
21 booking
22 Redirect with success "Booking confirmed"
[Full or not upcoming]
23 WaitlistEntry::firstOrCreate(user_id,event_id)
24 waitlist entry
25 Redirect with info "Joined waitlist"
Cancellation flow (organiser)
26 DELETE /events/{id}/bookings/{bookingId}
27 destroy(bookingId)
28 delete booking
alt [Event previously full]
29 first waitlist entry (FIFO)
30 next user?
31 Mail::to(user).send(WaitlistNotificationMailable)
32 queued/logged (Mail::fake in tests)
33 "Booking cancelled; waitlisted notified"
[No waitlist/wasn't full]
34 "Booking cancelled"

State Diagram


Browsing
click event
ViewingEvent
"Book Now" (Attendee, upcoming)
AttemptBooking
Validating
capacity_check_passes && not_duplicate
full OR not_upcoming
Approved
Rejected
Approved
Confirmed
attendee_cancels / organiser_cancels
Rejected && eligible
Cancelled
event_was_full
NotifyNext
Waitlisted
email_sent_to_first_waitlisted
booking_cancelled_for_full_event
user_leaves
Notified
LeftWaitlist
user_accepts && capacity_available
no_response_in_window
Promoted
Expired