Events Module — Software Specification and Implementation Plan

1. Executive summary

Purpose: Deliver a production-ready Events module that enables discovery, booking, ticket issuance, validation, transfers and resale for live events across Canada and the United States.

Scope: Web frontend, Native mobile apps (iOS / Android), backend services, integrations and operational requirements.

High-level goals:

• Provide reliable, low-latency discovery and booking UX on web and native.

• Ensure secure payments and ticket issuance with QR and mobile wallet support.

• Provide organizer tools for event creation, ticketing rules and reporting.

• Comply with regional regulations and tax rules.

2. Architecture overview

Overview:

The architecture is modular and service-oriented. Frontend clients communicate with an API gateway that routes requests to dedicated services. MongoDB is the primary datastore. Redis provides inventory holds and rate limiting. Stripe is the payment provider. Notification services deliver email and SMS.

Primary services:

• Auth Service (JWT roles and scopes)

• Event Service (CRUD, search, filters)

• Order Service (holds, idempotency, order lifecycle)

• Payment Service (Stripe PaymentIntent orchestration)

• Ticket Service (issue QR, wallet passes, transfers, resale)

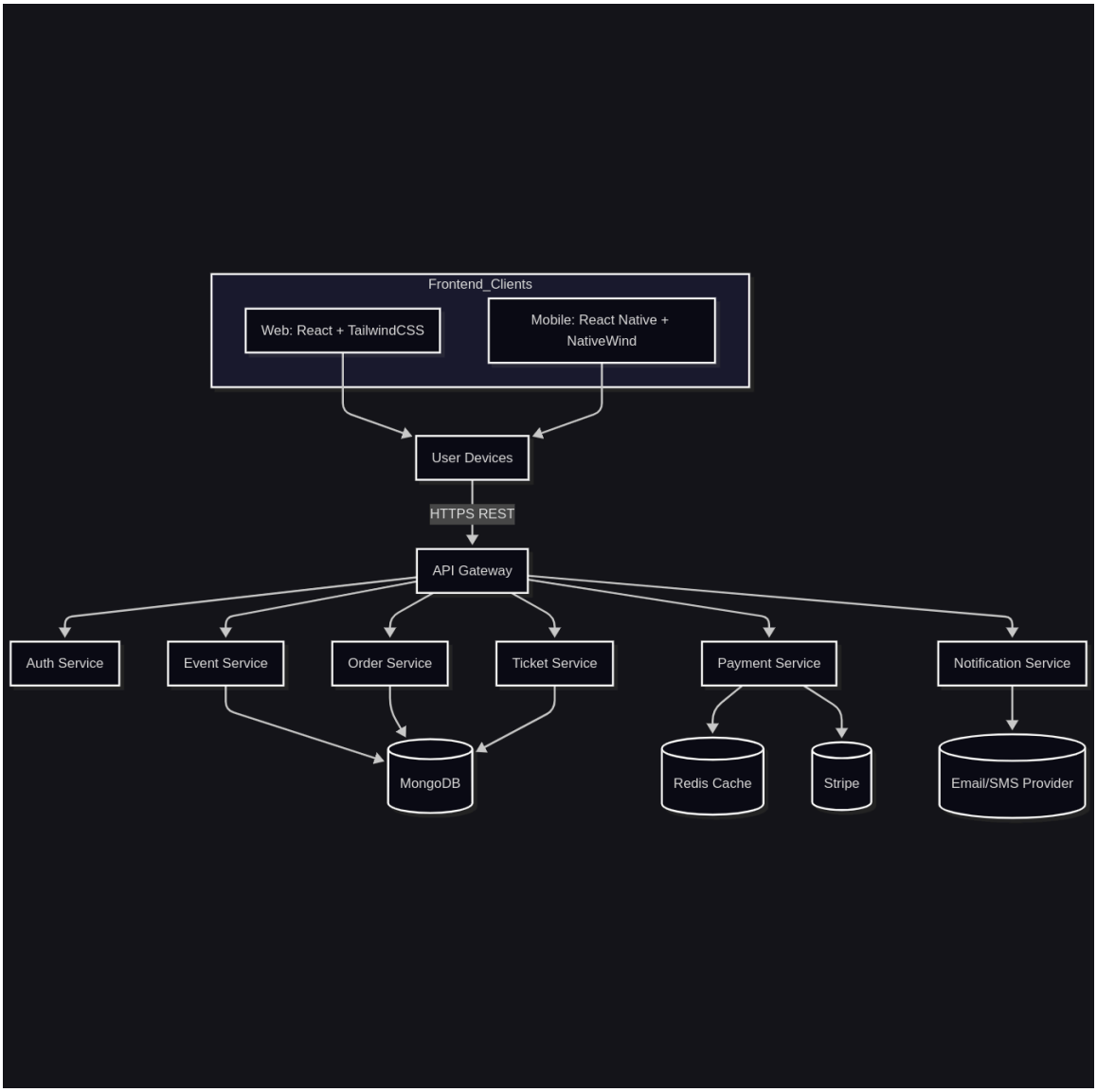
• Validation Service (scanner endpoints, offline sync)

• Notification Service (email, SMS, push)

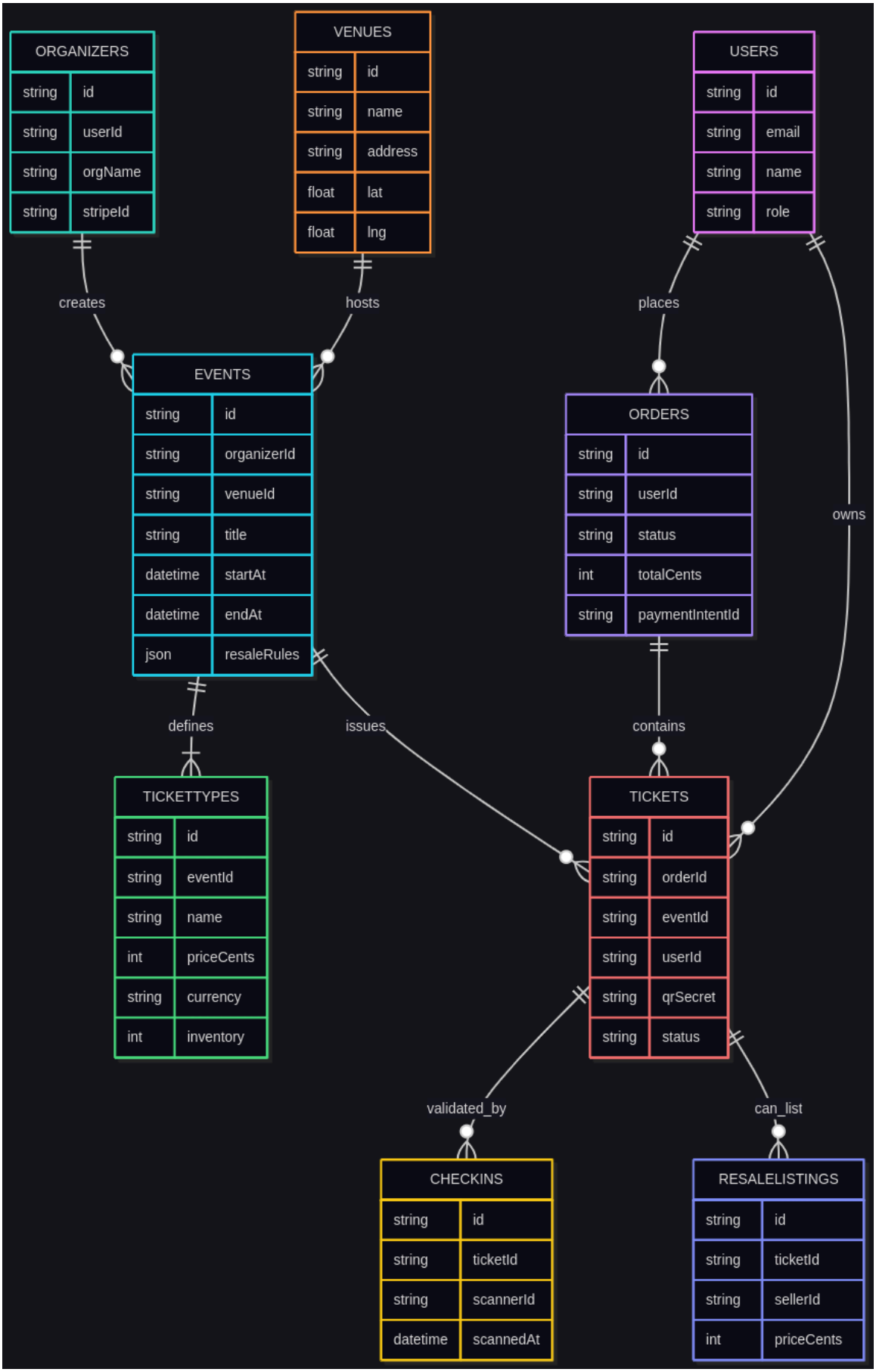
• Analytics/Reporting Service (read replicas, ETL)

3. System diagrams

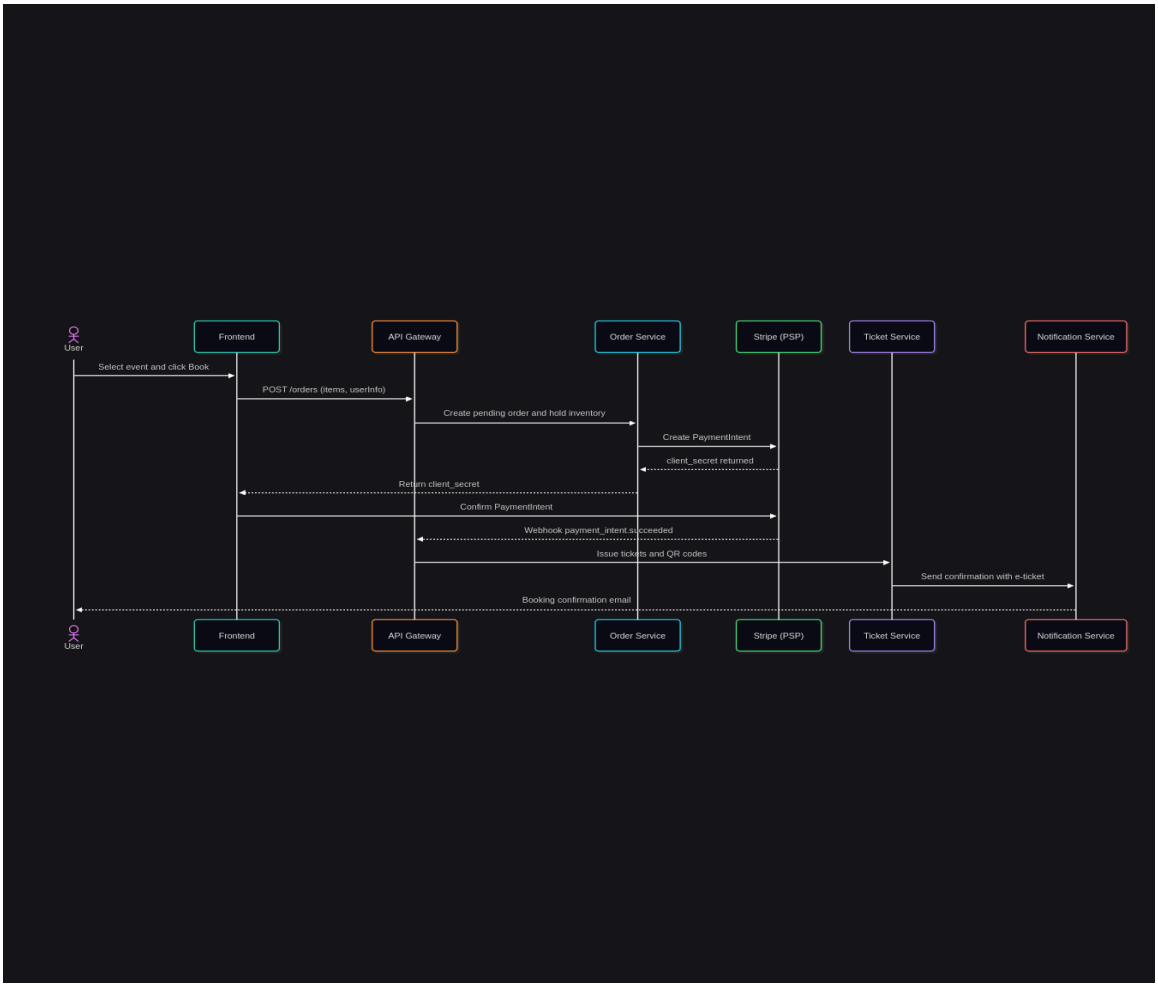
System architecture:



Entity relationship:


ORGANIZERS
string id
string userId
string orgName
string stripeId
VENUES
string id
string name
string address
float lat
float lng
USERS
string id
string email
string name
string role
creates
hosts
places
owns
EVENTS
string id
string organizerId
string venueId
string title
datetime startAt
datetime endAt
json resaleRules
ORDERS
string id
string userId
string status
int totalCents
string paymentIntentId
defines
issues
contains
TICKETTYPES
string id
string eventId
string name
int priceCents
string currency
int inventory
TICKETS
string id
string orderId
string eventId
string userId
string qrSecret
string status
validated_by
can_list
CHECKINS
string id
string ticketId
string scannerId
datetime scannedAt
RESALELISTINGS
string id
string ticketId
string sellerId
int priceCents

Booking & payment sequence:



4. API deliverable and repository task

```
openapi: 3.1.0

info:

 title: Events Module API

 version: 1.0.0

 description: API for Events discovery, booking, ticket issuance,
validation, resale and transfers.

servers:
```

```yaml
 - url: https://api.domain.com/v1

components:

 securitySchemes:

   BearerAuth:

     type: http

     scheme: bearer

     bearerFormat: JWT

 schemas:

   Error:

     type: object

     properties:

       code:

         type: string

       message:

         type: string

   Event:

     type: object

     properties:

       id:

         type: string

       title:

         type: string

       description:

         type: string

       startAt:
```

```yaml
        type: string
        format: date-time
      endAt:
        type: string
        format: date-time
      venue:
        $ref: '#/components/schemas/Venue'
      ticketTypes:
        type: array
        items:
          $ref: '#/components/schemas/TicketType'
      images:
        type: array
        items:
          type: string
      resaleRules:
        type: object
  Venue:
    type: object
    properties:
      id: { type: string }
      name: { type: string }
      address: { type: string }
      lat: { type: number }
      lng: { type: number }
```

```
  TicketType:
    type: object
    properties:
      id: { type: string }
      name: { type: string }
      priceCents: { type: integer }
      currency: { type: string }
      inventory: { type: integer }
      type: { type: string, enum: ["GA","RESERVED"] }
      seatMapId: { type: string, nullable: true }
  OrderRequest:
    type: object
    required: [items, userInfo, idempotencyKey]
    properties:
      idempotencyKey: { type: string }
      items:
        type: array
        items:
          type: object
          properties:
            ticketTypeId: { type: string }
            quantity: { type: integer }
            seatInfo:
              type: object
      userInfo:
```

```
        type: object
        properties:
          name: { type: string }
          email: { type: string }
      paymentMethodId:
        type: string
  OrderResponse:
    type: object
    properties:
      orderId: { type: string }
      status: { type: string }
      amountCents: { type: integer }
paths:
 /events:
  get:
    summary: Search / list events
    parameters:
      - name: q
        in: query
        schema: { type: string }
      - name: city
        in: query
        schema: { type: string }
      - name: from
        in: query
```

```
        schema: { type: string, format: date-time }
      - name: to
        in: query
        schema: { type: string, format: date-time }
      - name: category
        in: query
        schema: { type: string }
      - name: lat
        in: query
        schema: { type: number }
      - name: lng
        in: query
        schema: { type: number }
      - name: radiusKm
        in: query
        schema: { type: number }
    responses:
      '200':
        description: list
        content:
          application/json:
            schema:
              type: array
              items: { $ref: '#/components/schemas/Event' }
/events/{id}:
```

```yaml
  get:
    summary: Get event by id
    parameters:
      - name: id
        in: path
        required: true
        schema: { type: string }
    responses:
      '200':
        content:
          application/json:
            schema: { $ref: '#/components/schemas/Event' }
      '404':
        content:
          application/json:
            schema: { $ref: '#/components/schemas/Error' }
/orders:
  post:
    summary: Create an order (hold inventory, return client secret)
    security:
      - BearerAuth: []
    requestBody:
      required: true
      content:
        application/json:
```

```
          schema: { $ref: '#/components/schemas/OrderRequest' }
    responses:
      '201':
        description: Created
        content:
          application/json:
            schema: { $ref: '#/components/schemas/OrderResponse' }
      '409':
        description: Conflict (inventory)
 /payments/confirm:
   post:
    summary: Confirm payment (webhook style from client) - server
finalizes order
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              orderId: { type: string }
              paymentIntentId: { type: string }
    responses:
      '200':
        description: payment processed
```

```
/webhooks/stripe:
  post:
    summary: Stripe webhooks
    requestBody:
      required: true
      content:
        application/json:
          schema: { type: object }
    responses:
      '200': { description: ok }
/tickets/{ticketId}:
  get:
    summary: Get ticket detail (for My Tickets)
    parameters:
      - name: ticketId
        in: path
        required: true
    responses:
      '200':
        content:
          application/json:
            schema:
              type: object
              properties:
                id: { type: string }
```

```yaml
                qrSecret: { type: string }
                seatInfo: { type: object }
/tickets/issue:
  post:
    summary: Issue ticket (internal, called after payment webhook)
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              orderId: { type: string }
    responses:
      '201':
        description: ticket(s) created
/tickets/validate:
  post:
    summary: Validate a ticket QR (scanner apps)
    security:
      - BearerAuth: []
    requestBody:
      required: true
      content:
        application/json:
```

```
          schema:
            type: object
            required: [qrSecret, scannerId]
            properties:
              qrSecret: { type: string }
              scannerId: { type: string }
              scannedAt: { type: string, format: date-time }
    responses:
      '200':
        description: validation result
        content:
          application/json:
            schema:
              type: object
              properties:
                status: { type: string, enum: [VALID, INVALID,
ALREADY_CHECKED_IN] }
                ticketId: { type: string }
                checkedInAt: { type: string, format: date-time }
 /tickets/{ticketId}/transfer:
   post:
     summary: Initiate transfer of ticket
     security:
       - BearerAuth: []
     requestBody:
```

```
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              toEmail: { type: string }
    responses:
      '200':
        description: transfer pending
/tickets/{ticketId}/resell:
  post:
    summary: List ticket on resale marketplace
    security:
      - BearerAuth: []
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              priceCents: { type: integer }
    responses:
      '201':
```

```
        description: listing created
/users/{userId}/bookings:
  get:
    summary: Get bookings for a user
    security:
      - BearerAuth: []
    parameters:
      - name: userId
        in: path
        required: true
    responses:
      '200':
        content:
          application/json:
            schema:
              type: array
              items:
                type: object
                properties:
                  orderId: { type: string }
                  tickets: { type: array }
/organizers/{orgId}/events:
  post:
    summary: Organizer creates an event
    security:
```

```
      - BearerAuth: []
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Event'
    responses:
      '201': { description: created }
```

5. Backend tasks

Implementation tasks:

1. Implement Auth service with JWT roles and scopes: attendee, organizer, scanner, admin.

2. Implement Event read and Event create endpoints for organizers with validation and image upload support.

3. Implement Orders POST endpoint with inventory hold logic using Redis. Ensure idempotency via idempotency keys.

4. Implement PaymentIntent creation endpoint integrated with Stripe; include metadata linking to orderId and userId.

5. Implement webhook handler at /webhooks/stripe with signature verification and processing of payment events.

6. Implement ticket issuance job triggered by payment success. The job generates secure QR secrets, creates ticket records in MongoDB, and enqueues notification tasks for email and wallet pass generation.

7. Set up development and staging environments for backend services with test Stripe keys and local Redis/Mongo instances.

6. Frontend tasks

Web team deliverables:

• Implement Events listing, Event details, Booking flow, Booking summary and Booking success screens. Fetch data from API sandbox endpoints. Integrate Stripe Elements for card input and Payment Intent confirmation.

Native team deliverables:

• Implement Events listing and Event details screens. Implement QR viewer, Add to Wallet flow, and integrate ReactNativeStripe (Stripe SDK) for payments. Use NativeWind for consistent styling and shared design tokens.

Shared frontend deliverables:

• Provide a component library with EventCard, TicketView, FilterBar, SeatPicker, and shared design tokens. Publish the library to an internal package registry. Ensure localization and accessibility are integrated.

7. QA & Operations tasks

QA and ops action items:

• Load test the booking flow and simulate on-sale traffic spikes. Validate Redis inventory behavior under concurrency.

• Penetration test payment and webhook flows. Confirm webhook signature verification and idempotency behavior.

• Implement structured logging and Sentry error tracking. Ensure logs include correlation_id for tracing across services.

• Configure monitoring and alerting dashboards for SLOs and error rates.

8. Documentation and onboarding

Documentation deliverables:

• Publish API documentation using Swagger UI based on events.yaml.

• Publish developer onboarding guide describing local environment setup, API keys, and flow to issue wallet passes.

• Publish design tokens and component usage guidelines for web and native teams.

9. Event core (embedded)

The following event excerpt contains the core endpoints required by the frontend and backend teams. Include the full Events YAML in the repository as events.yaml.

```yaml
openapi: 3.1.0
info:
  title: Events Module API
  version: 1.0.0
servers:
  - url: https://api.domain.com/v1
components:
  securitySchemes:
    BearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
paths:
  /events:
    get:
      summary: List and search events
  /events/{id}:
    get:
      summary: Get event by id
  /orders:
    post:
      summary: Create order and hold inventory
  /payments/create-intent:
    post:
      summary: Create PaymentIntent (Stripe)
  /webhooks/stripe:
    post:
      summary: Stripe webhook handler
  /tickets/issue:
    post:
      summary: Issue tickets after payment confirmation
  /tickets/validate:
    post:
      summary: Validate ticket QR (scanner app)
  /tickets/{ticketId}/transfer:
    post:
      summary: Transfer ticket to another user
  /tickets/{ticketId}/resell:
    post:
      summary: List ticket on resale marketplace
  /users/{userId}/bookings:
    get:
      summary: Get user bookings
```

10. Frontend — screen-by-screen summary

Events Listing (Web & Mobile): Featured carousel, filters, search, event grid.

Event Details: Hero image, metadata, ticket types list, book CTA.

Booking: Ticket selection, seat selection modal for reserved seating, add-ons.

Booking Summary: Detailed price breakdown including fees and taxes, payment CTA.

Booking Success: Confirmation, wallet add, share and manage booking.

My Bookings & Ticket View: List of bookings, QR display, transfer and resale actions if permitted.

11. Implementation notes and best practices

• Ensure all money values are handled as integer cents to avoid floating point issues.

• Use idempotency keys for order creation and payment endpoints to guard against duplicate charges.

• Protect webhook endpoints using signature verification and replay protection.

• Implement optimistic concurrency for inventory and seat assignments with Redis locks and MongoDB checks.

• Store consent records for marketing per CASL and PIPEDA requirements; support subject data requests and deletions.

12. References and Ideas

- Ticketmaster — https://www.ticketmaster.com / https://www.ticketmaster.ca
- Eventbrite — https://www.eventbrite.com / https://www.eventbrite.ca
- SeatGeek — https://seatgeek.com
- StubHub — https://www.stubhub.com
- Vivid Seats — https://www.vividseats.com
- Tickets.com — https://www.tickets.com
- Universe (Live Nation/Universe) — https://www.universe.com
- Ticketleap — https://www.ticketleap.com
- Showpass (Canada-focused) — https://www.showpass.com
- Ticket Tailor — https://www.tickettailor.com
- Yapsody — https://www.yapsody.com
- Brown Paper Tickets — https://www.brownpapertickets.com
- TicketNetwork — https://www.ticketnetwork.com
- TicketSwap — https://www.ticketswap.com
- Festicket — https://www.festicket.com
- Front Gate Tickets (festival-focused operations) — https://www.frontgatetickets.com
- Bandsintown — https://www.bandsintown.com
- Songkick — https://www.songkick.com
- Universe BoxOffice / BoxOffice by Universe (organizer/scan tools) — Play store listing / universe.com

- Front Gate (also provides festival ops & on-site systems) — https://www.frontgatetickets.com
- See Tickets — https://www.seetickets.com (operates in NA via localized portals)
- Tickets.com (MLBAM-owned tech + distribution) — https://www.tickets.com
- TicketTailor / TicketSource / Showpass alternatives (smaller self-serve tools) — tickettailor.com and showpass.com