Week 3 Self-Evaluation — Booking System & Seat Selection

Scope recap

- Build seat map UI, locking, validation, and real-time updates.
- Complete booking flow: passenger info, booking creation, summary/review, history/dashboard.
- Guest services: checkout without registration, lookup, reference generation.
- Ticketing: PDF e-ticket with QR, download/email, branded template.

Status by task

- Seat map UI & validation: **Done** — `frontend/src/pages/search/SeatSelection.tsx` renders grouped seats with state styling, enforces selection limits, and blocks unavailable seats. (Uses 30s polling until lock acquired.)
- Seat locking mechanism: **Done (DB/JWT)** — `/api/seat-status/lock` uses pessimistic DB locking and JWT lock tokens (`backend/src/modules/seat-status/providers/seat-lock.provider.ts`, controller + service). **Redis not yet used** despite spec; locks expire by timestamp.
- Real-time seat updates: **Partial** — 30s polling on `SeatSelection` until lock; no WebSocket push yet.
- Seat selection validation logic: **Done** — Backend validates seat states/lock/duplicates and seat-to-passenger mapping in `backend/src/modules/booking/providers/booking.provider.ts`.
- Passenger info forms: **Done** — `frontend/src/pages/search/Checkout.tsx` collects per-seat passenger data and contact info with Zod + RHF validation.
- Booking creation/management: **Done (create)** — Public `POST /api/booking` consumes lock token, books seats atomically, and clears locks (`booking.controller.ts` / `booking.provider.ts`). No cancellation/expiration workflows yet.
- Booking summary/review UI: **Done** — Checkout shows trip details, selected seats, fare + fee total; routes to payment stub.
- Booking history/dashboard: **Done** — Users can view booking history and drill into booking details (UI wired to backend data).
- Guest checkout: **Partial** — Booking endpoint allows missing `userId`; contact info not persisted, no guest session UX.
- Guest booking lookup: **Done** — Guests can retrieve their bookings and view details (lookup now backed by real data).
- Booking reference generation: **Not done** — Uses UUID; no friendly reference codes.
- PDF e-ticket with QR + download/email + branded template: **Not started** — No PDF/QR/email implementation yet (`rg pdf` shows none).
- Booking detail email: **Done** — Confirmation email with booking details is sent after booking creation.
- Booking edit/cancel: **Done (pending only)** — Pending bookings can be edited or cancelled; cancelling releases seats back to available state.

Evidence highlights

- Seat lock flow: `backend/src/modules/seat-status/providers/seat-lock.provider.ts`, `seat-status.controller.ts`, `seat-status.service.ts`.
- Booking creation: `backend/src/modules/booking/providers/booking.provider.ts`, `booking.controller.ts`.
- Frontend seat selection & checkout: `frontend/src/pages/search/SeatSelection.tsx`, `frontend/src/pages/search/Checkout.tsx`.
- Booking history/detail: user dashboard now renders real booking list/detail.
- Guest lookup: guest booking lookup displays real booking data.
- Admin UX improvements (adjacent to Week 3): Trips/Routes/Buses pages now have filters, pagination, and error toasts; trip admin endpoint ordering fixed. Bus seat editor shows an interactive seat map, enforces unique seat codes per bus, and deleting a bus now soft-deletes its seats.
- Booking management: Pending bookings can be edited/cancelled; cancel frees seats.

Gaps & risks

- Redis/WebSocket real-time missing; locks rely on DB/JWT only.
- No booking cancellation/expiry, reference codes, or contact persistence; guest flow incomplete.
- Ticketing (PDF/QR/email) unimplemented.

Next steps

1. Swap lock storage to Redis and add WebSocket/Server-Sent Events for live seat status.
2. Implement booking lifecycle (expiry/cancel), friendly reference generator, and persist contact info for guests.
3. Add PDF/QR e-ticket generation with email delivery and download endpoint.