Full Stack Developer Assessment

Estimated Time: 3.5 – 4.5 hours

Stack: Next.js 14+ (App Router), Tailwind CSS, SQLite, TypeScript

Overview

You are building a lightweight **Concierge Itinerary Proposal System** for Exclusive Resorts. A concierge can build a curated trip itinerary for a member, send it as a proposal via email, and the member can review, approve, and "pay" to lock it in before their trip.

This reflects a real workflow problem in luxury travel — we want to see how you think about product, data modeling, UX, and clean code under time pressure.

The Scenario

> **A member named James Whitfield is arriving at Villa Punta Mita, Mexico on March 15 and departing March 22.** The concierge team wants to build him a personalized itinerary proposal and get his sign-off before he arrives.

You are given:

- A simulated member record (hardcoded or seeded in the DB)
- A hardcoded reservation (arrival/departure + destination) to display context
- A set of itinerary categories to build from

What You'll Build

1 Concierge Dashboard (Front End)

A single-page dashboard the concierge uses to:

- See the member's upcoming trip — destination, arrival, departure dates displayed prominently at the top
- Build an itinerary proposal by adding line items from predefined categories (see below)
- Each line item has: category, title, description, date/time, estimated price
- Preview the proposal before sending
- Send the proposal — this triggers a "send email" action (you do not need a real email service; log to console or write to a `sent_emails` table, and display a success state in the UI)
- View all sent proposals with their current status (draft, sent, approved, paid)

2 Member Experience (Front End)

A separate route (e.g. `/proposal/[id]`) that simulates what the member sees when they click the link in their email:

- Beautifully presented itinerary with all line items, dates, and pricing
- Total cost clearly shown
- **Approve** button — moves proposal to approved
- **Pay & Lock In** button — moves proposal to paid and shows a confirmation screen
- Proposal should feel premium — this is a luxury brand

3 API / Backend

RESTful or Next.js Route Handlers covering:

- `GET /api/reservations` — return the member's current reservation
- `POST /api/proposals` — create a new proposal (draft)
- `GET /api/proposals` — list all proposals with status
- `GET /api/proposals/[id]` — get a single proposal with line items
- `PATCH /api/proposals/[id]` — update status (sent, approved, paid)
- `POST /api/proposals/[id]/send` — mark as sent + log the "email"

4 Database (SQLite)

Design and implement a simple schema. At minimum:

```
members — id, name, email reservations — id, member_id,
destination, villa, arrival_date, departure_date proposals — id,
reservation_id, status, created_at, sent_at proposal_items — id,
proposal_id, category, title, description, scheduled_at, price
sent_emails — id, proposal_id, to_email, sent_at, body_preview
```

Seed the DB with the member and reservation described above.

Itinerary Categories

Use these categories as the building blocks for line items. Display them as selectable cards or a dropdown in the concierge UI:

Category	Icon Suggestion	Example Activities
Dining	🍽️	Private chef dinner, restaurant reservation
Activities	🏄	Surf lesson, snorkeling, ATV tour
Wellness	💆	Spa treatment, yoga session, massage
Excursions	⛵	Whale watching, sailing charter, cultural tour
Transport	🚗	Airport transfer, private car, helicopter
Experiences	🌅	Sunset cocktails, bonfire on the beach, tequila tasting

Requirements & Constraints

- TypeScript throughout
- Tailwind CSS for all styling — no component libraries unless you bring in shadcn/ui (acceptable)
- Next.js App Router preferred (Pages Router acceptable if you explain why)

- SQLite for persistence — use better-sqlite3, Prisma with SQLite, or Drizzle with SQLite
- No real payment processing — a button that moves status to `paid` is sufficient
- No real email sending — write to a DB table and/or console log
- A `README.md` explaining how to run the project locally (one command ideally)

Evaluation Criteria

We are looking at the following, roughly in order of importance:

Area	What We're Looking For
Problem Thinking	Does the data model make sense? Does the workflow feel right?
UI/UX Quality	Does the concierge dashboard feel functional and efficient? Does the member view feel premium and luxurious?
Code Quality	Clean, readable TypeScript. Sensible component structure. No spaghetti.
API Design	Are routes logical, consistent, and do they handle errors gracefully?
Completeness	Does the full loop work — create → send → approve → pay?
README / Communication	Can you explain your decisions clearly and concisely?

Deliverables

1. **GitHub repo** (public or shared with us) with your full solution
2. **README.md** that includes:
   - How to install and run locally
   - Any assumptions you made
   - What you would improve given more time
   - What you found most interesting or challenging

3. **A short Loom or recorded screen walkthrough** (5–10 min) demoing the full flow and talking through your key decisions — this is important to us

Stretch Goals (only if you finish early)

> **These are completely optional.** Do not sacrifice the core requirements for these.

- Add a notes/message field the concierge can include with the proposal (rendered in the member view)
- Allow the concierge to edit a draft before sending
- Show a timeline view of the itinerary (day-by-day) in the member view
- Add optimistic UI updates so status changes feel instant
- Animate the proposal approval / payment confirmation screen
- Support multiple members / reservations in the UI

Tips & Notes

> - **We're a luxury travel company.** The member-facing view should feel that way — think clean whitespace, elegant typography, subtle imagery or gradients. Tailwind gives you everything you need.
> - **The concierge view should be efficient** — they're professionals who move fast. Prioritize clarity over decoration here.
> - **Don't overthink the database.** SQLite is fine. Normalize just enough to make the queries clean.
> - **The Loom walkthrough carries real weight** — we want to hear you think out loud.
> - **If something is broken or incomplete, call it out in your README.** Honesty matters more than pretending it's perfect.

Submission

Please bring your GitHub link and Loom walkthrough to your in-person interview.

We'll review and discuss your work together during the interview.

Questions before then? Reach out — we want you to succeed.

Good luck. We're excited to see how you think.