Full-Stack Take-Home (4 Hours): Appointment Booking

Goal: Build and deploy a minimal appointment booking app for a small clinic. Deliver a working **hosted URL** plus repo in 4 hours.

Core User Stories (Must-Have)

As a Patient

1. I can **register** and **log in**.
2. I can **see available slots** for the next 7 days.
3. I can **book a slot** (and not double-book an already-taken slot).
4. I can **see my bookings**.

As an Admin

1. I can **log in as admin** (seeded user is fine).
2. I can **see all bookings**.

Non-Functional

- Basic input validation and error handling.
- Auth (JWT or session) with **role-based access** (patient vs admin).
- Persistent storage (SQLite/Postgres/Mongo acceptable).
- **Deploy** both API and UI to a free tier and share **live links**.

API Requirements

Create a REST API (JSON) with these endpoints:

- `POST /api/register` – `{name, email, password}` → 201 on success.
- `POST /api/login` – `{email, password}` → 200 with `{token, role}`.
- `GET /api/slots?from=YYYY-MM-DD&to=YYYY-MM-DD` – available slots (30-min blocks between 9:00–17:00 local time).
- `POST /api/book` – `{slotId}` → 201; must prevent double booking.
- `GET /api/my-bookings` – auth: patient; returns list.

- `GET /api/all-bookings` – auth: admin; returns all.

Constraints

- Prevent booking if another booking exists for the same `slotId`.
- Return sensible HTTP codes and JSON error shapes:
  `{ "error": { "code": "SLOT_TAKEN", "message": "..." } }`
- Time zone: assume server local or UTC; be consistent and document.

Suggested Schema (example)

- `users(id, name, email [unique], password_hash, role ['patient'|'admin'], created_at)`
- `slots(id, start_at [UTC ISO], end_at [UTC ISO], created_at)`
- `bookings(id, user_id, slot_id [unique], created_at)`

You may generate slots on the fly for the next 7 days or seed them on server start.

Frontend Requirements

- Framework - ReactJS. Keep styling minimal.
- Pages/flows:
  - Register / Login
  - Patient Dashboard: list available slots → Book → My Bookings
  - Admin Dashboard: All Bookings
- Persist auth across refresh.
- Show API errors in-UI (toasts or inline messages).
- Simple routing and loading states.

Deliverables (Quality Bar)

1. **Live App URL(s)**
   - **Frontend URL** (e.g., Vercel/Netlify/Cloudflare Pages).
   - **API URL** (e.g., Render/Railway/Fly.io).
   - Provide **test credentials**:

        - Patient: `patient@example.com` / `Passw0rd!`
        - Admin: `admin@example.com` / `Passw0rd!` (seed this user)

2. **Public Git Repository**
    - Clear commit history (no code dumps at T+3:59).
    - A top-level **`README.md`** that includes:
        - Tech stack choices (and 2–3 sentences of trade-offs)
        - How to run locally (one command per service)
        - Environment variables required
        - Deployment steps taken (exact commands or screenshots)
        - Known limitations and what you'd do with 2 more hours

3. **Architecture Notes (1 page in README)**
    - Folder structure rationale
    - Auth + RBAC approach
    - Concurrency/atomicity for booking
    - Error handling strategy

4. **Quick Verification Script**
    - Either a **Postman collection** or 4–6 **`curl`** commands in the README to:
        - Register → Login → Get Slots → Book → Get My Bookings

5. **Basic Tests (Optional but Scored)**
    - 3–5 backend unit/integration tests **or** a minimal e2e flow.

6. **Security Hygiene**
    - Hash passwords, do not log secrets, CORS allowlist (frontend origin), basic rate-limit or brute-force guard (even simple).

Free/Open Hosting Options (Use Any)

Frontend (static): Vercel, Netlify, Cloudflare Pages
Backend (server): Render Free, Railway Free, Fly.io Free
Database:

- Postgres: **Neon** or **Railway Postgres** (free tiers)
- SQLite: fine for demo if your host supports persistent storage (else mount a volume or generate slots in memory)
- Mongo: **MongoDB Atlas Free**

Tip: Popular "quick path" is **Vercel (frontend)** → **Render (Express API)** → **Neon (Postgres)**. All have free tiers.

Timeboxing Guidance (so it's doable in 4 hours)

- **0:00–0:20** – Repo, issue checklist, envs, choose stack, scaffold.
- **0:20–1:30** – Backend: auth, slots, booking (unique constraint), basic validation.
- **1:30–2:30** – Frontend: auth flow, list slots, book, my bookings, admin table.
- **2:30–3:20** – Deploy DB + API + frontend; wire envs; CORS; smoke tests.
- **3:20–3:50** – README, curl/Postman, seed users, fix rough edges.
- **3:50–4:00** – Final QA: create new patient, book, refresh, admin views.

Evaluation Rubric (100 points)

A. Working Features (35 pts)

- Auth + RBAC (10)
- Slot listing and booking w/ double-book prevention (15)
- Patient and Admin views (10)

B. Code Quality & Architecture (25 pts)

- Clear structure, separation of concerns (10)
- Clean, readable code & naming (5)
- Error handling & input validation (5)
- Sensible schema & transactions/constraints (5)

C. DevOps/Deployment (20 pts)

- Live links work; envs managed; CORS correct (10)
- Reproducible deploy steps in README (5)
- Sensible use of free/open services (5)

D. UX & DX Polish (10 pts)

- Usable flows, loading & error states (5)
- Helpful README & verification steps (5)

E. Stretch/Extras (10 pts)

- Basic tests (3–5 meaningful) (5)
- Dockerfiles or CI lint/test (3)
- Rate limit/login throttle or password rules (2)

Deductions

- Missing hosted link (-20)
- Default admin password in repo (-10)
- Double booking possible (-10)
- Secrets committed (-15)

Guardrails & Constraints

- You may use scaffolding (Create React App, Vite, Next.js, NestJS, Express generator).
- You may use an ORM (Prisma/TypeORM/Sequelize) or query builder (Knex).
- No heavy UI kits required; Tailwind/Bootstrap ok.
- Keep infra simple (single region; free tiers).
- If you run out of time, **document trade-offs plainly**.

Acceptance Criteria (we will check)

- Can register & log in as patient; can book a visible slot; “My Bookings” updates.
- Second attempt to book the same slot by anyone fails with a clear error.

- Admin login shows all bookings.
- Refreshing the page keeps me logged in.
- README lets us run locally within 10 minutes.
- Hosted links are reachable for 72 hours after submission.

Suggested Tech Path (example stacks)

Node + Express + Postgres + React

- API: Express + Prisma, JWT auth, unique index on `bookings.slot_id`
- DB: Neon (free Postgres)
- API host: Render Free
- Frontend: React + Vite on Vercel

Alt: Next.js (pages/api) + Supabase (Postgres & Auth) deployed to Vercel.
(If using hosted Auth, still enforce roles and booking uniqueness on the DB.)

Seed & Testing Data

- Seed an **admin** user on server start (env-driven credentials) or via a script.
- Generate 7 days of slots at 30-min intervals between 09:00–17:00.
  Example: `2025-08-07T09:00:00Z … 2025-08-13T17:00:00Z` (document TZ choice).

Submission Checklist (paste this in your README)

- Frontend URL: _____
- API URL: _____
- Patient: `patient@example.com` / `Passw0rd!`
- Admin: `admin@example.com` / `Passw0rd!`
- Repo URL: _____
- Run locally: `README` steps verified
- Postman/curl steps included

- Notes on trade-offs & next steps

How We'll Interview Around It (15–20 mins follow-up)

- Walkthrough of architecture decisions and trade-offs.
- Why you chose your data model and how you ensured no double booking.
- What you'd build next with 2 more hours.