# Admin → **admin@example.com / admin123** User → **user@example.com / user123**

# Hotel Booking App — Mini DPR (Detailed Project Report)

## 1) Project Overview

**Name:** Hotel Booking App  
**Primary Users:** Guests/Users and Admins  
**Stack:** Frontend – React (Vite) + Tailwind; Backend – Python (Flask + SQLAlchemy + Flask-JWT-Extended); DB – PostgreSQL 15+  
**Focus:** Database-centric design with strong data integrity, constraints, views, functions, and admin/user analytics.

### 1.1 Goals

* Discover hotels & room types, view details, book rooms.
* Auth with roles; admins can book on behalf of guests/users.
* Dashboards with current bookings & history; edit/cancel.
* Admin CRUD for hotels/room types/rooms; occupancy logic.
* Query/analytics: per-user history, per-hotel bookings by user, per-user per-hotel counts.
* Search and filtering across the app.

### 1.2 Non‑Goals (v1)

* Payments & invoicing
* Multi-room in a single booking (v2)
* Dynamic pricing/yield management (v2)

## 2) Functional Requirements

### 2.1 Public/Homepage

* Search bar (hotel name/city/date range/capacity).
* Card grid of room types across hotels: thumbnail, price, capacity, **Details**, **Book Now**.

### 2.2 Details Page

* Hotel info, amenities, images.
* Room types list (price, capacity, current availability), **Book Now** opens booking form.

### 2.3 Booking

* Users must be logged in.
* Admins can book “for” someone (fields: Guest name, optional user lookup).
* Over‑booking prevented at DB level; if none available → error.
* Edit/Cancel by owner; admin can modify anything (audited).

### 2.4 Dashboards

* **User:** Upcoming bookings, history, edit/cancel if eligible; “How many times I booked Hotel X?” quick stat.
* **Admin:** All bookings, filters (by hotel/user/date/status), quick actions; analytics widgets.

### 2.5 Admin Management

* CRUD: Hotels, Room Types, Rooms (physical rooms).
* Status toggles: room active/maintenance; hotel computed occupancy (auto Occupied if all rooms booked).
* Queries: user history; per-hotel bookings by different users.

### 2.6 Security/Access

* JWT auth; role: user or admin.
* Users can see only their bookings; admins see all.

## 3) High-Level Architecture

* **React (Vite)** SPA; Router + protected routes; components for cards/modals/forms; Tailwind UI.
* **Flask API** (REST): Blueprints for auth, hotels, rooms, bookings, admin, reports.
* **PostgreSQL** as source of truth. Heavy logic in DB (constraints, exclusion constraints, views, functions).
* **Migrations:** Alembic/Flask‑Migrate.
* **Storage:** (v1) Images via URLs (seeded/sample).

## 4) Database Design (PostgreSQL)

Target: PostgreSQL 15+; install extension btree\_gist for range constraints.

### 4.1 Entities

* **users**(id, full\_name, email, phone, role, password\_hash, created\_at, updated\_at)
* **hotels**(id, name, city, country, address, description, star\_rating, amenities jsonb, created\_at, updated\_at)
* **hotel\_images**(id, hotel\_id, url, alt\_text, is\_primary)
* **room\_types**(id, hotel\_id, name, capacity, base\_price, description, amenities jsonb, active, created\_at, updated\_at)
* **rooms**(id, hotel\_id, room\_type\_id, room\_number, status, active, created\_at, updated\_at)
* **bookings**(id, booked\_by\_user\_id, guest\_user\_id nullable, guest\_name, hotel\_id, room\_type\_id, room\_id, check\_in, check\_out, status, num\_guests, total\_amount, currency, created\_at, updated\_at)
* **booking\_status\_log**(id, booking\_id, changed\_by\_user\_id, from\_status, to\_status, note, changed\_at)

### 4.2 Core Constraints & Rules

* Unique: users.email, (rooms.hotel\_id, rooms.room\_number).
* Checks: check\_in < check\_out; num\_guests <= room\_types.capacity; status in allowed set.
* Foreign keys with ON UPDATE CASCADE and sensible ON DELETE (e.g., RESTRICT on hotels/rooms referenced by bookings).
* **No overlapping bookings per room** enforced by exclusion constraint over a generated tstzrange (Postgres):

CREATE EXTENSION IF NOT EXISTS btree\_gist;  
  
-- Enum via CHECK for easy migrations  
CREATE TABLE users (  
 id BIGSERIAL PRIMARY KEY,  
 full\_name TEXT NOT NULL,  
 email CITEXT UNIQUE NOT NULL,  
 phone TEXT,  
 role TEXT NOT NULL CHECK (role IN ('user','admin')),  
 password\_hash TEXT NOT NULL,  
 created\_at TIMESTAMPTZ NOT NULL DEFAULT now(),  
 updated\_at TIMESTAMPTZ NOT NULL DEFAULT now()  
);  
  
CREATE TABLE hotels (  
 id BIGSERIAL PRIMARY KEY,  
 name TEXT NOT NULL,  
 city TEXT NOT NULL,  
 country TEXT NOT NULL,  
 address TEXT,  
 description TEXT,  
 star\_rating INT CHECK (star\_rating BETWEEN 1 AND 5),  
 amenities JSONB DEFAULT '{}'::jsonb,  
 created\_at TIMESTAMPTZ NOT NULL DEFAULT now(),  
 updated\_at TIMESTAMPTZ NOT NULL DEFAULT now()  
);  
  
CREATE TABLE hotel\_images (  
 id BIGSERIAL PRIMARY KEY,  
 hotel\_id BIGINT NOT NULL REFERENCES hotels(id) ON DELETE CASCADE,  
 url TEXT NOT NULL,  
 alt\_text TEXT,  
 is\_primary BOOLEAN NOT NULL DEFAULT false  
);  
  
CREATE TABLE room\_types (  
 id BIGSERIAL PRIMARY KEY,  
 hotel\_id BIGINT NOT NULL REFERENCES hotels(id) ON DELETE CASCADE,  
 name TEXT NOT NULL,  
 capacity INT NOT NULL CHECK (capacity > 0),  
 base\_price NUMERIC(12,2) NOT NULL CHECK (base\_price >= 0),  
 description TEXT,  
 amenities JSONB DEFAULT '{}'::jsonb,  
 active BOOLEAN NOT NULL DEFAULT true,  
 created\_at TIMESTAMPTZ NOT NULL DEFAULT now(),  
 updated\_at TIMESTAMPTZ NOT NULL DEFAULT now()  
);  
  
CREATE TABLE rooms (  
 id BIGSERIAL PRIMARY KEY,  
 hotel\_id BIGINT NOT NULL REFERENCES hotels(id) ON DELETE CASCADE,  
 room\_type\_id BIGINT NOT NULL REFERENCES room\_types(id) ON DELETE RESTRICT,  
 room\_number TEXT NOT NULL,  
 status TEXT NOT NULL DEFAULT 'available' CHECK (status IN ('available','maintenance','inactive')),  
 active BOOLEAN NOT NULL DEFAULT true,  
 created\_at TIMESTAMPTZ NOT NULL DEFAULT now(),  
 updated\_at TIMESTAMPTZ NOT NULL DEFAULT now(),  
 UNIQUE (hotel\_id, room\_number)  
);  
  
CREATE TABLE bookings (  
 id BIGSERIAL PRIMARY KEY,  
 booked\_by\_user\_id BIGINT NOT NULL REFERENCES users(id) ON DELETE RESTRICT,  
 guest\_user\_id BIGINT REFERENCES users(id) ON DELETE SET NULL,  
 guest\_name TEXT, -- required if booking on behalf of non-registered guest  
 hotel\_id BIGINT NOT NULL REFERENCES hotels(id) ON DELETE RESTRICT,  
 room\_type\_id BIGINT NOT NULL REFERENCES room\_types(id) ON DELETE RESTRICT,  
 room\_id BIGINT REFERENCES rooms(id) ON DELETE RESTRICT,  
 check\_in TIMESTAMPTZ NOT NULL,  
 check\_out TIMESTAMPTZ NOT NULL,  
 status TEXT NOT NULL DEFAULT 'confirmed' CHECK (status IN ('pending','confirmed','checked\_in','checked\_out','cancelled')),  
 num\_guests INT NOT NULL CHECK (num\_guests > 0),  
 total\_amount NUMERIC(12,2) NOT NULL CHECK (total\_amount >= 0),  
 currency TEXT NOT NULL DEFAULT 'USD',  
 created\_at TIMESTAMPTZ NOT NULL DEFAULT now(),  
 updated\_at TIMESTAMPTZ NOT NULL DEFAULT now(),  
 CONSTRAINT chk\_time\_valid CHECK (check\_in < check\_out)  
);  
  
CREATE TABLE booking\_status\_log (  
 id BIGSERIAL PRIMARY KEY,  
 booking\_id BIGINT NOT NULL REFERENCES bookings(id) ON DELETE CASCADE,  
 changed\_by\_user\_id BIGINT NOT NULL REFERENCES users(id) ON DELETE RESTRICT,  
 from\_status TEXT,  
 to\_status TEXT NOT NULL,  
 note TEXT,  
 changed\_at TIMESTAMPTZ NOT NULL DEFAULT now()  
);  
  
-- Generated range column for exclusion (PG 12+)  
ALTER TABLE bookings  
 ADD COLUMN stay tstzrange GENERATED ALWAYS AS (tstzrange(check\_in, check\_out, '[)')) STORED;  
  
-- Prevent overlapping stays for the same room, excluding cancelled  
CREATE INDEX IF NOT EXISTS bookings\_stay\_gist\_idx ON bookings USING gist (room\_id, stay);  
ALTER TABLE bookings  
 ADD CONSTRAINT no\_overlap\_per\_room EXCLUDE USING gist (  
 room\_id WITH =,  
 stay WITH &&  
 ) WHERE (status <> 'cancelled');  
  
-- Performance indexes  
CREATE INDEX IF NOT EXISTS idx\_bookings\_hotel ON bookings(hotel\_id);  
CREATE INDEX IF NOT EXISTS idx\_bookings\_user ON bookings(guest\_user\_id);  
CREATE INDEX IF NOT EXISTS idx\_bookings\_period ON bookings USING gist (stay);

### 4.3 DB Functions & Triggers

**Find available room** for a room type and period:

CREATE OR REPLACE FUNCTION fn\_find\_available\_room(p\_hotel\_id BIGINT, p\_room\_type\_id BIGINT,  
 p\_start TIMESTAMPTZ, p\_end TIMESTAMPTZ)  
RETURNS BIGINT LANGUAGE plpgsql AS $$  
DECLARE r\_id BIGINT;  
BEGIN  
 SELECT r.id INTO r\_id  
 FROM rooms r  
 WHERE r.hotel\_id = p\_hotel\_id  
 AND r.room\_type\_id = p\_room\_type\_id  
 AND r.active = true AND r.status = 'available'  
 AND NOT EXISTS (  
 SELECT 1 FROM bookings b  
 WHERE b.room\_id = r.id  
 AND b.status IN ('pending','confirmed','checked\_in')  
 AND tstzrange(b.check\_in, b.check\_out, '[)') && tstzrange(p\_start, p\_end, '[)')  
 )  
 ORDER BY r.id  
 LIMIT 1;  
 RETURN r\_id; -- NULL if none  
END; $$;

**Create booking** (atomic, server‑side):

CREATE OR REPLACE FUNCTION sp\_create\_booking(  
 p\_booked\_by BIGINT,  
 p\_guest\_user BIGINT,  
 p\_guest\_name TEXT,  
 p\_hotel BIGINT,  
 p\_room\_type BIGINT,  
 p\_check\_in TIMESTAMPTZ,  
 p\_check\_out TIMESTAMPTZ,  
 p\_num\_guests INT,  
 p\_total NUMERIC,  
 p\_currency TEXT DEFAULT 'USD'  
) RETURNS BIGINT LANGUAGE plpgsql AS $$  
DECLARE v\_room BIGINT; v\_id BIGINT;  
BEGIN  
 IF p\_check\_in >= p\_check\_out THEN RAISE EXCEPTION 'Invalid stay window'; END IF;  
 SELECT fn\_find\_available\_room(p\_hotel, p\_room\_type, p\_check\_in, p\_check\_out) INTO v\_room;  
 IF v\_room IS NULL THEN RAISE EXCEPTION 'No rooms available for the selected dates'; END IF;  
  
 INSERT INTO bookings (booked\_by\_user\_id, guest\_user\_id, guest\_name, hotel\_id, room\_type\_id, room\_id,  
 check\_in, check\_out, status, num\_guests, total\_amount, currency)  
 VALUES (p\_booked\_by, p\_guest\_user, p\_guest\_name, p\_hotel, p\_room\_type, v\_room,  
 p\_check\_in, p\_check\_out, 'confirmed', p\_num\_guests, p\_total, p\_currency)  
 RETURNING id INTO v\_id;  
  
 RETURN v\_id;  
END; $$;

**Auto log status changes:**

CREATE OR REPLACE FUNCTION trg\_log\_status() RETURNS TRIGGER AS $$  
BEGIN  
 IF TG\_OP = 'UPDATE' AND NEW.status IS DISTINCT FROM OLD.status THEN  
 INSERT INTO booking\_status\_log(booking\_id, changed\_by\_user\_id, from\_status, to\_status)  
 VALUES (NEW.id, NEW.booked\_by\_user\_id, OLD.status, NEW.status);  
 END IF;  
 RETURN NEW;  
END; $$ LANGUAGE plpgsql;  
  
CREATE TRIGGER bookings\_log\_status  
AFTER UPDATE ON bookings  
FOR EACH ROW EXECUTE FUNCTION trg\_log\_status();

### 4.4 Views for Queries/Analytics

* **v\_hotel\_occupancy** (computed status: Occupied if available=0 for all active rooms of any room type):

CREATE OR REPLACE VIEW v\_hotel\_occupancy AS  
SELECT h.id AS hotel\_id, h.name,  
 SUM(CASE WHEN r.active AND r.status='available' THEN 1 ELSE 0 END) AS total\_available\_rooms,  
 SUM(CASE WHEN r.active THEN 1 ELSE 0 END) AS total\_rooms,  
 CASE WHEN SUM(CASE WHEN r.active AND r.status='available' THEN 1 ELSE 0 END) = 0 THEN 'Occupied' ELSE 'Available' END AS hotel\_status  
FROM hotels h  
LEFT JOIN rooms r ON r.hotel\_id = h.id  
GROUP BY h.id, h.name;

* **v\_roomtype\_availability (date‑bound)** — available count for a requested window:

-- counts rooms of a type that are free for the window now -> now is parameterized in API  
-- Illustrative query (to be parameterized):  
-- SELECT COUNT(\*) FROM rooms r WHERE r.room\_type\_id=$1 AND NOT EXISTS (  
-- SELECT 1 FROM bookings b WHERE b.room\_id=r.id AND b.status IN ('pending','confirmed','checked\_in')  
-- AND tstzrange(b.check\_in,b.check\_out,'[)') && tstzrange($start,$end,'[)'));

* **v\_user\_booking\_counts\_per\_hotel**:

CREATE OR REPLACE VIEW v\_user\_booking\_counts\_per\_hotel AS  
SELECT b.guest\_user\_id AS user\_id, b.hotel\_id, COUNT(\*) AS bookings\_count  
FROM bookings b  
WHERE b.status <> 'cancelled' AND b.guest\_user\_id IS NOT NULL  
GROUP BY b.guest\_user\_id, b.hotel\_id;

* **Admin report:** bookings by user for a hotel:

-- SELECT u.full\_name, u.email, COUNT(\*) AS bookings  
-- FROM bookings b JOIN users u ON u.id = b.guest\_user\_id  
-- WHERE b.hotel\_id = $hotel AND b.status <> 'cancelled'  
-- GROUP BY u.full\_name, u.email ORDER BY bookings DESC;

### 4.5 Sample Queries Required by Spec

* **Admin → query a user’s booking history:**

SELECT b.\* FROM bookings b WHERE (b.guest\_user\_id = $user OR b.booked\_by\_user\_id = $user)  
ORDER BY b.check\_in DESC;

* **Admin → bookings by different users for a hotel:**

SELECT u.id, u.full\_name, COUNT(\*) AS total  
FROM bookings b LEFT JOIN users u ON u.id = b.guest\_user\_id  
WHERE b.hotel\_id = $hotel AND b.status <> 'cancelled'  
GROUP BY u.id, u.full\_name ORDER BY total DESC;

* **User → how many times *I***\*\* booked Hotel X:\*\*

SELECT COUNT(\*)  
FROM bookings b  
WHERE b.hotel\_id = $hotel AND b.status <> 'cancelled'  
 AND (b.guest\_user\_id = $me OR b.booked\_by\_user\_id = $me);

## 5) API Design (Flask)

Base URL: /api

### 5.1 Auth

* POST /auth/register → create user
* POST /auth/login → JWT access+refresh
* POST /auth/logout (optional blacklist)

### 5.2 Public

* GET /hotels?search=&city=
* GET /hotels/:id (with images + room types)
* GET /room-types/:id
* GET /availability?hotel\_id=&room\_type\_id=&check\_in=&check\_out=

### 5.3 Bookings

* GET /bookings (user: own; admin: all, with filters user\_id, hotel\_id, status, from, to)
* POST /bookings (body: hotel\_id, room\_type\_id, check\_in, check\_out, num\_guests, [guest\_name|guest\_user\_id]) → calls sp\_create\_booking
* GET /bookings/:id
* PATCH /bookings/:id (edit dates/guests if policy allows; re-run availability)
* DELETE /bookings/:id (cancel)

### 5.4 Admin

* POST /hotels, PATCH /hotels/:id, DELETE /hotels/:id
* POST /room-types, PATCH /room-types/:id, DELETE /room-types/:id
* POST /rooms, PATCH /rooms/:id, DELETE /rooms/:id
* Reports:
  + GET /reports/user/:user\_id/bookings
  + GET /reports/hotel/:hotel\_id/user-bookings
  + GET /reports/hotel/:hotel\_id/occupancy

**AuthZ:** decorator checks role; in queries, non‑admins automatically filter by current\_user.id.

## 6) Frontend Pages & Components (React + Tailwind)

* **Layout:** Topbar (logo, search, login/register or user menu), optional Sidebar in dashboards.
* **HomePage:** Search + grid of RoomTypeCards (image, name, hotel, price, capacity, buttons).
* **HotelDetailPage:** Hotel info, carousel, RoomTypeList with availability by selected dates.
* **BookingModal/Form:** date pickers, guests, (admin) guest name/user lookup, price summary.
* **UserDashboard:** tabs: Upcoming, History, Stats (per‑hotel count).
* **AdminDashboard:** tabs: Bookings (filters), Hotels, Room Types, Rooms, Reports.
* **Auth:** Login/Register, Logout button.

**State:** React Query (TanStack) for API calls & caching; Zod/Yup for forms.  
**Routing:** React Router; protected routes by role.

## 7) Business Rules & Policies

* Booking can be edited if status IN ('pending','confirmed') and new dates available.
* Cancellation allowed before check\_in; after → admin required.
* A room in maintenance cannot be assigned; API rejects (fn\_find\_available\_room).
* A hotel is shown as **Occupied** in UI if v\_hotel\_occupancy.total\_available\_rooms = 0.

## 8) Testing Plan

* **DB unit tests** for functions/triggers: overlapping bookings, capacity checks, maintenance rooms.
* **API tests**: auth, ACL, happy/error paths for bookings.
* **UI tests**: booking flow, admin CRUD, search.

## 9) Project Plan & Milestones (Indicative)

* **Day 1–2:** Finalize schema; set up Flask project, Alembic migrations; seed data.
* **Day 3–4:** Implement bookings API (create/cancel/edit) using sp\_create\_booking; availability endpoint; unit tests.
* **Day 5–6:** Admin CRUD endpoints + reports; React scaffolding & auth pages.
* **Day 7–8:** Home/Details/Booking UI; dashboards; polish.
* **Day 9:** QA, README, sample data, demo video.

## 10) Deliverables

* Git repo with backend (Flask) and frontend (React) folders.
* SQL migrations; SQL script to create views/functions; seed SQL.
* Postman/Thunder tests.
* README with setup, ERD, and screenshots.

## 11) Next Steps

1. Confirm React + Tailwind is OK (we can switch to vanilla JS if required).
2. I’ll scaffold the repo (Flask API + Alembic + JWT + sample endpoints) and create initial migration with the schema above.
3. We’ll seed a few hotels/rooms and start wiring the Home → Details → Booking flow.