Scenario

- Build a minimal healthcare system with **Patients**, **Physicians**, and **Prescription Drugs**.
- Patients may see multiple physicians across locations.
- Physicians can only view **their own patients** and the prescriptions they authored.
- A dashboard or a report of top N drugs used across all patients (admin function)

Instructions

- You may seek assistance from LLMs/ GPTs of your choice but ensure that it is explainable and not verbose.
- Assignments are progressive and completeness will be rewarded

Overall Success Criteria

- End-to-end flows work via UI (login → dashboards → create/view prescriptions).
- DB constraints + indexes for common lookups (see below).
- Runs with docker compose up and a one-page README with commands.

Assignment 1: API + SQL

Goal: Build two endpoints and the SQL behind them.

Brief

- **Entities**: patients, physicians, drugs, prescriptions.
- **Implement**:
  - POST /prescriptions: validate (patient_id, physician_id, drug_id, quantity, sig).
  - GET /analytics/top-drugs?from&to&limit=10: return top-N drugs in date range.
- **RBAC** (lightweight): Accept a header X-Role: physician|patient|admin.
  - Physicians may only create prescriptions for their linked patients.
  - Patients cannot call POST; can call GET but results must be scoped to their own prescriptions.
  - Admin unrestricted.

Deliverables

- Go handlers + minimal router.
- SQL schema + seed (SQL file ok; Postgres preferred).
- A single test (table-driven) for the top-N query or handler.
- Minimal README with run commands (Docker optional).

Success criteria

- Correct HTTP semantics, input validation, clear error messages.
- One performant aggregate query for top-N (with suggested index).
- Simple, readable code; passes go test ./... if tests included.

Assignment 2: Frontend-Only

Goal: Show React skill and product thinking without backend.

Brief

- Build a small React app (Vite/CRA) with:
- Login screen (fake auth; store role in local state).
- “Top Drugs” report screen: table + simple bar chart.
- Filters: date range, limit; call a provided mock (you give them a JSON file or a small mock server).
- Role-aware UI: physician sees only their patients’ data (simulated by filtering data using physician Id from auth state), patient sees only their own.

Deliverables

- React app, simple state management (Context or hooks).
- Clean components, empty/loading/error states.
- One accessibility touch (focus management or ARIA labels).
- README with *npm i && npm start*.

Success criteria

- Sensible component decomposition, clean state flow.
- Good UX polish for a tiny app; chart renders accurately.
- Clear role-based filtering on the client (given mock constraints).

Assignment 3: DevOps

- Postgres + API + Web via Docker Compose.
- Migration in compose.
- .env.example for secrets; never commit real secrets.