Emory ED Sepsis Surveillance Dashboard

A Vite + React prototype of the sepsis surveillance experience we shared with Emory's ED sepsis team. The goal is to provide a working front-end that mirrors the design exploration and can be wired into real patient feeds or analytics services.

Tech Stack

- **Framework**: React 18 with TypeScript, Vite for bundling, Tailwind v4 utility classes (see `src/index.css`).
- **UI primitives**: Radix UI (Select, Dialog, etc.), shadcn-inspired components under `src/components/ui`.
- **Charts**: Recharts (see `src/components/charts`) for patient trend lines.
- **State**: Local React state and memoization only—no global store yet.

Repository Layout

```
Design/
|- src/
|  |- App.tsx               -> Entry point with search, filtering, and detail dialog toggling.
|  |- mockData.json         -> Synthetic cohort with vitals, labs, SEP-1 protocol status.
|  |- components/
|  |    |- PatientTable.tsx   -> Renders ranked census list with condition bands.
|  |    |- PatientDetailDialog.tsx -> Detail workspace with vitals charts, SEP protocol, risk bre
|  |    \- charts/TrendLineChart.tsx -> Thin wrapper around Recharts LineChart.
|  |- constants/            -> Visualization ranges and tone helpers.
|  \- styles/ / ui/         -> Tailwind + design tokens and reusable UI atoms.
|- package.json
\- vite.config.ts
```

The Design directory is the working app; the sibling src/ folder in the repo root currently has no runtime code.

Getting Started

1. **Install dependencies**

   ```
   cd Design
   npm install
   ```

2. **Run locally**

```
npm run dev
```

Vite will print a local URL (default `http://localhost:5173`). Hot reload is enabled.

3. **Build for handoff**

   ```
   npm run build
   ```

   Outputs a static bundle under `Design/build/`.

Requirements: Node.js 18+ (tested with 20.x) and npm 9+.

Mock Data & Domain Assumptions

- `src/mockData.json` contains ten representative encounters generated from historical feature distributions (priority rank, sepsis score, vitals, SEP-1 transport method flags, etc.).
- `App.tsx` converts the raw JSON into richer `PatientDetail` objects with:
  - Synthetic patient names and assigned primary nurse (`PATIENT_NAMES`, `PRIMARY_NURSES` arrays).
  - Derived risk breakdown slices (vitals/labs/comorbidities/age from `RISK_PRESETS`).
  - SEP protocol checklists with status + timing windows.
  - Auto-generated trends when the JSON lacks explicit values (`generateTrends` blends SBP/DBP, lactate, HR trajectories).
- Condition bands shown in the table are derived via `getConditionBand()` (`src/components/PatientTable.ts` which maps sepsis scores to `critical`, `warning`, or `stable`.

Updating Data

1. Replace `src/mockData.json` with an export from Emory's data lake or API (ensure numeric types stay numeric).
2. If the schema changes, adjust `RawMockPatient` / `buildPatientDetail()` in `App.tsx`.
3. For live data, swap the static import with a fetch inside `useEffect`, then hydrate state with the API response.

UI Overview

- **Census Table** (`PatientTable`): Search by MRN or patient name, filter by primary nurse, view priority rank, last vital timestamp highlighting ($\geq$ 60 min amber, $\geq$ 90 min red), and temperature/heart rate snapshots.
- **Detail Workspace** (`PatientDetailDialog`):
  - Multi-metric trend cards (temperature, HR, lactate, SBP/DBP) using `TREND_Y_RANGES` to keep axes consistent.
  - Risk breakdown donut, comorbidity chips (`getComorbidityTone`) and SEP-1 protocol checklist with status chips.
  - Alerts automatically generated when deltas exceed thresholds defined near the top of `PatientDetailDialog.tsx`.
- **Legend & Status badge**: Condition color key plus active patient count + vital alerts summary at the top of `App.tsx`.

Working With Real Systems

1. **Data plumbing** – Replace the mock import with a hook that polls Emory's sepsis scoring service or EPIC feed, enforce authentication, and handle pagination for >50 patients.
2. **SEP-1 compliance** – Wire the protocol checklist to actual order/EHR timestamps and compute overdue states server-side.
3. **Alerts & notifications** – Emit toast/Badge states based on streaming vitals or HL7 messages; thresholds are centralized in `CHANGE_THRESHOLDS`.
4. **Design tokens** – Pull typography/colors from Emory's design system via CSS variables (currently IBM Plex fonts).

Suggested Next Steps for Emory's Team

1. Connect to the hospital integration engine (Cloverleaf, Redox, etc.) and validate the JSON contract end-to-end.
2. Add clinician authentication + auditing (Okta, Azure AD) before exposing PHI.
3. Expand testing: unit tests for the data transformers and visual regression testing for the triage table once designs stabilize.
4. Decide on deployment target (Vercel, Azure Static Web Apps, or an internal Kubernetes ingress) and create CI/CD around `npm run build`.

Reach out if additional documentation is needed during the Emory handoff.