Health Management App — Full-stack scaffold

This document contains a ready-to-run scaffold for a **Health Management Web App** based on your spec. It includes a lightweight **Node + Express backend** (SQLite) and a simple **React frontend** (Vite). Paste files into a repo, run the commands in the README, and you'll have a working starting point.

Repo layout (what's included below)

```
/README.md
/backend/package.json
/backend/server.js
/backend/db.js
/backend/init.sql
/backend/seed.sql
/backend/.env.example
/frontend/package.json
/frontend/vite.config.js
/frontend/index.html
/frontend/src/main.jsx
/frontend/src/App.jsx
/frontend/src/api.js
/frontend/src/styles.css
```

README.md

~~~
# Health Management App (Scaffold)

## What this provides
- Node + Express backend using SQLite (file `data.db`).
- REST API for Patients, Diseases, Alarms, and Patient-Disease associations.
- Pre-seeded disease data for Diabetes, Hypertension, Asthma and 20+ diseases.
- Simple React (Vite) frontend with tabbed navigation: Dashboard, Patients,
Diseases, Alarms, Add Patient.

## Requirements
- Node 18+ and npm

## Setup Backend
```bash
cd backend
~~~

```
npm install
cp .env.example .env   # edit if needed
node server.js
```

Server runs at http://localhost:4000

Setup Frontend

```
cd frontend
npm install
npm run dev
```

Frontend runs at http://localhost:5173 (by default)

Notes

- This scaffold is intentionally simple to keep it lightweight. You can swap SQLite for Postgres later.
- The AI chatbot is a placeholder UI component; integrate an LLM API (OpenAI/GPT) and secure your keys before production.

  ```
  ```

/backend/package.json

```json
{
  "name": "health-backend",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "better-sqlite3": "^8.1.0",
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "body-parser": "^1.20.2",
    "nanoid": "^4.0.0"
  }
}
```

/backend/env.example

```
PORT=4000
DB_FILE=./data.db
```

/backend/init.sql

```sql
-- Schema for Health Management App
PRAGMA foreign_keys = ON;

CREATE TABLE IF NOT EXISTS diseases (
  id TEXT PRIMARY KEY,
  name TEXT NOT NULL,
  category TEXT,
  severity TEXT,
  description TEXT,
  symptoms TEXT,
  precautions TEXT,
  diet_recommendations TEXT,
  foods_to_avoid TEXT,
  care_instructions TEXT
);

CREATE TABLE IF NOT EXISTS patients (
  id TEXT PRIMARY KEY,
  name TEXT NOT NULL,
  dob TEXT,
  gender TEXT,
  phone TEXT,
  allergies TEXT,
  emergency_contact TEXT,
  medical_history TEXT
);

CREATE TABLE IF NOT EXISTS patient_diseases (
  id TEXT PRIMARY KEY,
  patient_id TEXT NOT NULL,
  disease_id TEXT NOT NULL,
  diagnosis_date TEXT,
  status TEXT,
  FOREIGN KEY(patient_id) REFERENCES patients(id) ON DELETE CASCADE,
  FOREIGN KEY(disease_id) REFERENCES diseases(id) ON DELETE CASCADE
);
```

```
CREATE TABLE IF NOT EXISTS alarms (
  id TEXT PRIMARY KEY,
  patient_id TEXT NOT NULL,
  medicine_name TEXT,
  dose TEXT,
  frequency TEXT,
  times TEXT,
  start_date TEXT,
  end_date TEXT,
  notes TEXT,
  active INTEGER DEFAULT 1,
  FOREIGN KEY(patient_id) REFERENCES patients(id) ON DELETE CASCADE
);
```

/backend/seed.sql

```
-- Minimal seed data (diabetes, hypertension, asthma) -- IDs are simple strings
here
INSERT OR IGNORE INTO diseases
(id,name,category,severity,description,symptoms,precautions,diet_recommendations,foods_to_avoid,c
VALUES
('d1','Diabetes','Endocrine','High','Chronic metabolic disease with high blood
sugar.','Increased thirst; frequent urination; fatigue','Monitor sugar;
exercise; medication adherence','Low sugar, high fiber foods; regular
meals','Sugary drinks; excessive
sweets','Regular blood sugar monitoring; follow up with physician'),
('d2','Hypertension','Cardiovascular','High','High blood pressure that increases
heart disease risk.','Headache; dizziness; blurred
vision','Low-salt diet; avoid smoking','Reduce salt; increase potassium-rich
vegetables','High-salt processed foods','Monitor BP regularly; lifestyle
modifications'),
('d3','Asthma','Respiratory','Moderate','Chronic inflammatory disease of the
airways.','Wheezing; shortness of breath;
cough','Avoid triggers; use inhaler as prescribed','Avoid known allergens;
balanced diet','Cold air; strong
fragrances','Use controller medications; action plan');

-- Add a few more example diseases (abbreviated)
INSERT OR IGNORE INTO diseases
(id,name,category,severity,description,symptoms,precautions,diet_recommendations,foods_to_avoid,c
VALUES
('d4','Malaria','Infectious','High','Parasitic infection transmitted by
mosquitoes.','Fever; chills; sweating','Use nets; avoid mosquito
exposure','Hydration; light, bland diet','Alcohol; certain fatty foods','Seek
prompt antimalarial treatment'),
('d5','Dengue
Fever','Infectious','High','Mosquito-borne viral infection causing high
fever.','High fever; joint pain; rash','Avoid mosquito bites; hydration','Oral
rehydration; light foods','NSAIDs (avoid due to bleeding risk)','Seek medical
care for warning signs');
```

/backend/db.js (Node helper using better-sqlite3)

```
const Database = require('better-sqlite3');
const fs = require('fs');
const path = require('path');

function init(dbFile, initSqlFile, seedSqlFile) {
  const firstTime = !fs.existsSync(dbFile);
  const db = new Database(dbFile);
  // enable foreign keys
  db.pragma('foreign_keys = ON');

  const initSql = fs.readFileSync(initSqlFile, 'utf8');
  db.exec(initSql);

  if (seedSqlFile && fs.existsSync(seedSqlFile)) {
    const seedSql = fs.readFileSync(seedSqlFile, 'utf8');
    db.exec(seedSql);
  }

  return db;
}

module.exports = { init };
```

/backend/server.js

```
require('dotenv').config();
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const { init } = require('./db');
const { nanoid } = require('nanoid');
const path = require('path');

const PORT = process.env.PORT || 4000;
const DB_FILE = process.env.DB_FILE || './data.db';

const app = express();
app.use(cors());
app.use(bodyParser.json());

const db = init(DB_FILE, path.join(__dirname, 'init.sql'), path.join(__dirname,
'seed.sql'));

// Utility to run queries
function rows(sql, params=[]) { return db.prepare(sql).all(params); }
function row(sql, params=[]) { return db.prepare(sql).get(params); }
function run(sql, params=[]) { return db.prepare(sql).run(params); }

// --- Diseases ---
app.get('/api/diseases', (req,res)=>{
  const q = req.query.q;
  let results;
  if (q) {
    const like = `%${q}%`;
    results = rows('SELECT * FROM diseases WHERE name LIKE ? OR symptoms LIKE ?
OR category LIKE ? LIMIT 200', [like, like, like]);
  } else {
    results = rows('SELECT * FROM diseases LIMIT 500');
  }
  res.json(results);
});

app.get('/api/diseases/:id', (req,res)=>{
  const d = row('SELECT * FROM diseases WHERE id = ?', [req.params.id]);
  if (!d) return res.status(404).json({error:'Not found'});
  res.json(d);
});

app.post('/api/diseases', (req,res)=>{
  const id = nanoid();
  const p = req.body;
  run(`INSERT INTO diseases
(id,name,category,severity,description,symptoms,precautions,diet_recommendations,foods_to_avoid,c
VALUES (?,?,?,?,?,?,?,?,?,?)`,

[id,p.name,p.category,p.severity,p.description,p.symptoms,p.precautions,p.diet_recommendations,p.
  res.json({id, ...p});
});
```

```
// --- Patients ---
app.get('/api/patients', (req,res)=>{
  const q = req.query.q;
  if (q) {
    const like = `%${q}%`;
    return res.json(rows('SELECT * FROM patients WHERE name LIKE ? OR phone
LIKE ? LIMIT 200', [like,like]));
  }
  res.json(rows('SELECT * FROM patients LIMIT 500'));
});

app.get('/api/patients/:id', (req,res)=>{
  const p = row('SELECT * FROM patients WHERE id = ?', [req.params.id]);
  if (!p) return res.status(404).json({error:'Not found'});
  // include diseases and alarms
  const diseases = rows('SELECT d.* FROM diseases d JOIN patient_diseases pd ON
pd.disease_id = d.id WHERE pd.patient_id = ?', [req.params.id]);
  const alarms = rows('SELECT * FROM alarms WHERE patient_id = ?',
[req.params.id]);
  p.diseases = diseases;
  p.alarms = alarms;
  res.json(p);
});

app.post('/api/patients', (req,res)=>{
  const id = nanoid();
  const p = req.body;
  run(`INSERT INTO patients
(id,name,dob,gender,phone,allergies,emergency_contact,medical_history) VALUES
(?,?,?,?,?,?,?,?)`,

[id,p.name,p.dob,p.gender,p.phone,p.allergies,p.emergency_contact,p.medical_history]);
  res.json({id, ...p});
});

// --- Patient-Disease associations ---
app.post('/api/patient_diseases', (req,res)=>{
  const id = nanoid();
  const p = req.body; // patient_id, disease_id, diagnosis_date, status
  run('INSERT INTO patient_diseases
(id,patient_id,disease_id,diagnosis_date,status) VALUES (?,?,?,?,?)',
[id,p.patient_id,p.disease_id,p.diagnosis_date,p.status]);
  res.json({id, ...p});
});

app.get('/api/patient_diseases/:patient_id', (req,res)=>{
  const list = rows('SELECT pd.*, d.name, d.severity FROM patient_diseases pd
JOIN diseases d ON d.id = pd.disease_id WHERE patient_id = ?',
[req.params.patient_id]);
  res.json(list);
});

// --- Alarms ---
app.get('/api/alarms', (req,res)=>{
  const patient_id = req.query.patient_id;
  if (patient_id) return res.json(rows('SELECT * FROM alarms WHERE patient_id
= ?', [patient_id]));
  res.json(rows('SELECT * FROM alarms LIMIT 500'));
});

app.post('/api/alarms', (req,res)=>{
  const id = nanoid();
  const a = req.body; // patient_id, medicine_name, dose, frequency, times,
start_date, end_date, notes
  run('INSERT INTO alarms
(id,patient_id,medicine_name,dose,frequency,times,start_date,end_date,notes,active)
VALUES (?,?,?,?,?,?,?,?,?,?)',
[id,a.patient_id,a.medicine_name,a.dose,a.frequency,a.times,a.start_date,a.end_date,a.notes,a.act
1:1]);
  res.json({id,...a});
});

// --- quick stats for dashboard ---
app.get('/api/stats', (req,res)=>{
  const totalPatients = row('SELECT COUNT(*) AS c FROM patients').c;
  const totalDiseases = row('SELECT COUNT(*) AS c FROM diseases').c;
  const activeAlarms = row('SELECT COUNT(*) AS c FROM alarms WHERE active =
1').c;
  res.json({totalPatients,totalDiseases,activeAlarms});
});

app.listen(PORT, ()=>{
  console.log(`Health backend running on http://localhost:${PORT}`);
});
```

/frontend/package.json

```
{
  "name": "health-frontend",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview"
  },
  "dependencies": {
    "react": "18.2.0",
    "react-dom": "18.2.0",
    "axios": "1.4.0",
    "react-router-dom": "6.14.1"
  },
  "devDependencies": {
    "vite": "5.1.0",
    "@vitejs/plugin-react": "4.0.0"
  }
}
```

/frontend/vite.config.js

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react()],
  server: { port: 5173 }
})
```

/frontend/index.html

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Health Management App</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

/frontend/src/main.jsx

```
import React from 'react'
import { createRoot } from 'react-dom/client'
import App from './App'
import './styles.css'

createRoot(document.getElementById('root')).render(<App />)
```

/frontend/src/api.js

```
import axios from 'axios'
const API = axios.create({ baseURL: 'http://localhost:4000/api' })
export default API
```

/frontend/src/App.jsx

```
import React, { useEffect, useState } from 'react'
import API from './api'

function Tab({label, active, onClick}){return <button className={active? 'tab
active':'tab'} onClick={onClick}>{label}</button>}

export default function App(){
  const [tab,setTab] = useState('Dashboard')
  const [stats,setStats] = useState({totalPatients:0,totalDiseases:
0,activeAlarms:0})
  const [patients,setPatients] = useState([])
  const [diseases,setDiseases] = useState([])
  const [alarms,setAlarms] = useState([])
  const [form,setForm] = useState({name:'',phone:''})

  useEffect(()=>{ fetchStats(); fetchDiseases(); fetchPatients(); },[])

  async function fetchStats(){
    const r = await API.get('/stats'); setStats(r.data)
  }
  async function fetchPatients(){ const r = await API.get('/patients');
setPatients(r.data) }
  async function fetchDiseases(){ const r = await API.get('/diseases');
```

```
setDiseases(r.data) }

  async function addPatient(e){
    e.preventDefault();
    await API.post('/patients', form); setForm({name:'',phone:''});
fetchPatients(); setTab('Patients')
  }

  return (
    <div className="app">
      <header>
        <h1>Health Management</h1>
      </header>
      <nav className="tabs">
        <Tab label="Dashboard" active={tab==='Dashboard'}
onClick={()=>setTab('Dashboard')} />
        <Tab label="Patients" active={tab==='Patients'}
onClick={()=>setTab('Patients')} />
        <Tab label="Diseases" active={tab==='Diseases'}
onClick={()=>setTab('Diseases')} />
        <Tab label="Alarms" active={tab==='Alarms'}
onClick={()=>setTab('Alarms')} />
        <Tab label="Add Patient" active={tab==='AddPatient'}
onClick={()=>setTab('AddPatient')} />
      </nav>

      <main>
        {tab==='Dashboard' && (
          <section>
            <h2>Dashboard</h2>
            <div className="stats">
              <div className="stat">Total Patients<br/
><strong>{stats.totalPatients}</strong></div>
              <div className="stat">Total Diseases<br/
><strong>{stats.totalDiseases}</strong></div>
              <div className="stat">Active Alarms<br/
><strong>{stats.activeAlarms}</strong></div>
            </div>
            <h3>Recent Patients</h3>
            <ul>{patients.slice(0,8).map(p=> <li key={p.id}>{p.name} — {p.phone}
</li>)}</ul>
          </section>
        )}

        {tab==='Patients' && (
          <section>
            <h2>Patients</h2>
            <input placeholder="Search by name or phone" onChange={async e=>{
```

```
const q=e.target.value; const r = await API.get('/patients?
q='+encodeURIComponent(q)); setPatients(r.data)}} />
            <ul>{patients.map(p=> <li key={p.id}><strong>{p.name}</strong> —
{p.phone}</li>)}</ul>
          </section>
        )}

        {tab==='Diseases' && (
          <section>
            <h2>Diseases</h2>
            <input placeholder="Search diseases" onChange={async e=>{ const
q=e.target.value; const r = await API.get('/diseases?
q='+encodeURIComponent(q)); setDiseases(r.data)}} />
            <ul>{diseases.map(d=> <li key={d.id}><strong>{d.name}</strong> —
{d.category} — {d.severity}</li>)}</ul>
          </section>
        )}

        {tab==='Alarms' && (
          <section>
            <h2>Alarms</h2>
            <button onClick={async ()=>{ const r = await API.get('/alarms');
setAlarms(r.data)}}>Load Alarms</button>
            <ul>{alarms.map(a=> <li key={a.id}>{a.medicine_name} – {a.times}</
li>)}</ul>
          </section>
        )}

        {tab==='AddPatient' && (
          <section>
            <h2>Add Patient</h2>
            <form onSubmit={addPatient}>
              <input required placeholder="Full name" value={form.name}
onChange={e=>setForm({...form,name:e.target.value})} />
              <input placeholder="Phone" value={form.phone}
onChange={e=>setForm({...form,phone:e.target.value})} />
              <button type="submit">Add</button>
            </form>
          </section>
        )}
      </main>

      <footer>
        <small>Professional, medical-grade UI (scaffold)</small>
      </footer>
    </div>
  )
}
```

/frontend/src/styles.css

```
body{font-family:system-ui,Arial;margin:0;background:#f6f8fa;color:#111}
.app{max-width:980px;margin:18px auto;padding:12px;background:white;border-
radius:8px;box-shadow:0 6px 24px rgba(0,0,0,0.06)}
header h1{margin:0}
.tabs{display:flex;gap:8px;padding:12px 0}
.tab{padding:8px 12px;border-radius:8px;border:1px solid
#ddd;background:#fff;cursor:pointer}
.tab.active{background:#0ea5a4;color:white;border:none}
.stats{display:flex;gap:12px;margin:12px 0}
.stat{flex:1;padding:12px;border-radius:8px;border:1px solid #eee}
main section{padding:8px}
input{padding:8px;border-radius:6px;border:1px solid #ccc;margin:6px 0}
button{padding:8px 12px;border-radius:
6px;border:none;background:#0ea5a4;color:white}
```

Next steps & tips

- To add the AI Chatbot (Dr. AI), create a new React component that posts to a `/api/chat` endpoint. Implement server-side proxy to call the external LLM API with proper request limits and safety checks.
- Replace SQLite with Postgres for production and use environment variables to manage DB connection.
- Add authentication and role-based access (doctors / admins) before storing sensitive medical data publicly.

If you'd like, I can now: - Provide this scaffold as a ZIP file ready to download, or - Push to a GitHub repo (if you give me a repo name and grant access via a link), or - Generate more detailed models (e.g., OpenAPI spec, unit tests, or a Figma-style UI spec).

Which would you like me to do next?