SMART TRIAGE & HOSPITAL MANAGEMENT SYSTEM

Mission-Aligned Instruction Prompt for GitHub Copilot

This document defines the exact mission, scope, UX expectations, architectural intent, and functional requirements for building a Smart Triage & Hospital Management System (STRA-System) for a public hospital in Nairobi, Kenya. The goal is to ensure Copilot generates correct, mission-aligned, production-grade React.js frontend code with exceptional UI/UX quality and seamless backend integration.

1. CORE MISSION (NON-NEGOTIABLE)

1. Eliminate long patient queues and inefficient waiting processes
2. Ensure accurate, real-time patient data capture and transfer
3. Enable urgency-based triage using RED / YELLOW / GREEN classification
4. Improve trust in public hospitals by delivering fast, reliable care
5. Build technology that actually works in real hospital conditions (offline-first, high load, limited resources)

2. USER ROLES & UX EXPECTATIONS

1. **Nurses:** Fast triage input, minimal typing, visual vitals capture, color-coded urgency, offline support
2. **Doctors:** Clean patient queue, full vitals history, prescriptions, diagnostics, zero clutter
3. **Admin:** Operational dashboards, KPIs, staffing, utilization, system-wide visibility
4. **Pharmacy:** Prescription flow, stock levels, alerts, fast fulfillment

3. FRONTEND TECH & DESIGN DIRECTIVES

1. Framework: React.js (functional components, hooks)
2. Mobile-first, fully responsive (tablets used heavily in wards)
3. Design must feel premium, intentional, and human-centered
4. Use clear visual hierarchy, spacing, cards, and color psychology
5. RED / YELLOW / GREEN must be instantly recognizable
6. Zero unnecessary screens, every UI element must serve a purpose
7. Optimized for speed: minimal clicks, fast load, cached data

4. KEY FUNCTIONAL FLOWS TO IMPLEMENT

1. Patient Registration → Vitals Capture → Symptom Checklist → Automatic Triage Score
2. Urgency Classification → Department Queue Assignment → Live Queue Updates
3. Doctor View → Diagnosis → Prescription → Disposition (Admit / Discharge / Transfer)
4. Admin Dashboard → Resources, Staff Hours, Patient Volume, Utilization
5. Pharmacy → Medication Dispense → Inventory Update → Alerts

5. DATA & SYSTEM INTEGRITY RULES

1. Never lose patient data (offline-first with sync)
2. Frontend must strictly follow backend API contracts
3. All patient actions must be auditable
4. UI must gracefully handle slow or unstable internet
5. Assume high patient volume and staff pressure environments

6. DESIGN PHILOSOPHY (VERY IMPORTANT)

1. This is not a demo or school project
2. This system must feel like it was built with care, empathy, and deep thought
3. Design should reduce stress, not add cognitive load
4. Every screen should answer: 'What do I need to do next?'
5. If a feature does not help patient flow, safety, or clarity, it does not belong

7. INSTRUCTIONS TO COPILOT

1. Generate clean, scalable, production-ready React.js code
2. Prioritize UI/UX excellence over shortcuts
3. Assume backend is Node.js with REST APIs
4. Design components for reuse and clarity
5. Use realistic hospital data models
6. Do not oversimplify logic or flows
7. Always align code with the mission defined above