

MediZyra: A Cyber-Resilient, Offline-First Hospital Care Coordination Platform with Role-Based Triage Workflows

Dinesh Kumar¹, Pragyan Kumar², Salman M. Khan³

¹Assistant Professor at Bhagwan Parshuram Institute of Technology ^{2,3} Students at Bhagwan Parshuram Institute of Technology ¹dineshkumar@bpitindia.com
²kumarpragyan87@gmail.com ³salmanmk0807@gmail.com

Abstract-- Healthcare institutions increasingly face cyber threats that disrupt appointment scheduling, patient record access, and clinical workflows. This paper presents MediZyra Healthcare Solutions, a fully offline-first, role-based hospital care coordination platform designed to maintain operational continuity even when internet-connected infrastructure is compromised or unavailable. The system is implemented using React 19 and Vite for the frontend, an Express.js API layer for role-based authentication and triage management, and MongoDB for persistent local data storage. Three distinct portal roles -- Patient, Admin, and Doctor -- enforce a structured care workflow from patient care request through admin triage to doctor-led clinical closure. The platform demonstrates cyber resilience through local-only data operation, status-rule enforcement, and role-hierarchy logic that mirrors real-world hospital process control. Experimental results confirm correct workflow enforcement across all roles, near-zero latency in local operation, and suitability as both an academic demonstration of resilience-focused design and a practical prototype for lightweight hospital coordination.

Index Terms-- Cyber Resilience, Healthcare IT, Role-Based Access Control, React 19, MongoDB, Express.js, Hospital Management System, Offline-First Architecture, Triage Workflow, JWT Authentication

INTRODUCTION

Healthcare has become one of the most heavily targeted sectors for cyberattacks globally. At least 81 of England's 236 NHS trusts were hit during the 2017 WannaCry ransomware outbreak, which disrupted appointment records, clinical systems, and day-to-day care operations on a massive scale [1]. In the United States, the Department of Health and Human Services (HHS) reported that cyberattacks against hospitals and health systems more than doubled between 2016 and 2021 [2]. At that point, building digitally resilient systems stopped being just a technical ambition -- it became a matter of patient safety.

The trouble is, most hospital management systems in use today were designed with the assumption that the internet would always be there. They depend on cloud databases and third-party APIs for even the most basic functions. When that connection gets cut -- whether through ransomware, infrastructure failure, or a deliberate attack -- everything grinds to a halt.

This paper presents MediZyra Healthcare Solutions, a prototype hospital coordination platform built around the idea that the system should never need the internet to do its job. It brings together specialist browsing, care request submission, admin triage, and doctor-led clinical closure in one locally deployable web application. The key difference from conventional HMS platforms is that MediZyra keeps everything -- the user interface, the API, and the database -- on a single machine. There is no outward-facing network dependency, which significantly shrinks the attack surface during deployment, training exercises, and actual emergency scenarios.

The primary contributions of this work are:

- **Three-Role Care Workflow:** A Patient-Admin-Doctor role hierarchy where each user can only do what their role permits, enforced at the server level with strict status-transition rules.

- **Offline-First Architecture:** The entire system -- React 19 + Vite frontend, Express.js API, and local MongoDB storage -- runs without any cloud dependency whatsoever.
- **Structured Appointment Lifecycle:** Appointments follow a defined state machine (Requested → Confirmed → Completed / Cancelled), and at every stage the system checks that the right person is making the change.
- **Cyber Resilience Demonstration:** By keeping everything local, the system reduces the available attack surface and maintains data persistence -- an approach validated through academic evaluation in a healthcare context.

IDENTIFICATION, RESEARCH, AND IDEA COLLECTION

The project followed a step-by-step process: first understanding where current healthcare IT systems actually break down, and then working toward a coordination platform that could keep running even when connectivity is gone. Here is how each stage unfolded:

- **[1] Identification of Problems in Existing Systems:** The starting point was studying what actually goes wrong when cloud-dependent HMS platforms come under attack. Ransomware incidents make the picture clear -- once connectivity is lost, staff lose access to records, appointment booking stops working, and the only fallback is pen and paper, which brings its own problems in the form of transcription errors and slower care delivery.
- **[2] Research on Existing Solutions:** Several well-known platforms were looked at, including OpenEMR, Bahmni, and a range of SaaS-based HMS products. Each has real strengths in the right setting, but none of them were designed to work when the network simply is not there. Cloud connectivity or distributed infrastructure is baked into all of them, which rules them out for a deployment where local-only resilience is the whole point.
- **[3] Requirement Analysis:** From those shortcomings, a working set of requirements took shape. The system would need role-specific workflows for patients, doctors, and admins; local data storage through MongoDB; an appointment lifecycle with enforced status transitions; and a React-based interface that works just as well when the internet is completely absent.
- **[4] Exploration of Offline-First Architecture:** Rather than treating offline capability as something to bolt on later, the architecture was designed around it from the start. Running the frontend, API, and database together on one local machine means there is no external dependency to cut and no outward-facing network surface for ransomware to target. The resilience here is structural, not an afterthought.
- **[5] Technology Selection:** React 19 paired with Vite was chosen for the frontend, Express.js for the API layer, and MongoDB for local persistence. The stack was picked because each piece runs cleanly on a single machine, and none of them need anything outside of it to function properly.

Put together, these steps shaped MediZyra into what it is: a hospital coordination platform that runs locally, enforces role boundaries, and stays fully operational without ever needing to reach the internet.

STUDIES AND FINDINGS

A. System Methodology

Formalizing Requirements

Looking closely at real-world healthcare cyberattacks -- especially WannaCry and the damage it caused across the NHS -- one thing became clear: these systems did not fail because they lacked features. They failed because they could not function without an internet connection. With that understanding, the following requirements were given priority:

- **Offline-First Local Operation:** Every part of the system must work without internet access, so that patient data and clinical workflows remain usable even when connectivity is disrupted.
- **Role-Hierarchy Enforcement:** Each role -- Patient, Admin, and Doctor -- should only be able to do what it is supposed to do. No one should be able to step outside their designated authority, and the system must enforce that.
- **Status-Transition Rules:** The appointment lifecycle needs to follow a logical sequence, and the system has to prevent nonsensical transitions -- you should not be able to cancel a consultation that has already been completed, or

skip the confirmation step.

- **Persistent Local Storage:** All records -- appointments, patient information, doctor profiles -- need to survive system restarts. If the machine reboots or the browser closes, nothing should be lost.

Technology Stack Rationale

The technology choices were driven by a simple goal: keep everything local, keep it responsive, and avoid pulling in unnecessary external dependencies.

- **Frontend Framework (React 19 + Vite):** React gives us a component-based, reactive interface, and Vite makes the development loop fast with near-instant hot module replacement. React Router DOM handles page navigation within a single-page application, so there are no full-page reloads.
- **API Layer (Express.js):** A lightweight Node.js framework for implementing RESTful endpoints handling authentication, triage updates, doctor note persistence, and role-validation logic.
- **Data Persistence (MongoDB):** A MongoDB instance running locally stores all patient, appointment, doctor, and admin records. Data persists through restarts, so nothing is lost when the system cycles.
- **Build Tooling (Vite):** Vite provides fast Hot Module Replacement during development and produces optimised builds for production, all served locally without needing a cloud CDN.

B. System Modules

1. Patient Module

The Patient Module handles all aspects of the patient-facing care journey, from specialist discovery through appointment tracking and clinical record review.

- [1] Self-registration and secure email/password login via the locally running Express API.
- [2] Browse and search specialists by name, specialty, city, or symptom focus area.
- [3] Submit care requests by selecting date, time slot, consultation mode, and describing symptoms.
- [4] View complete appointment history including admin intake notes, doctor consultation summaries, prescriptions, and follow-up plans.
- [5] Upcoming visits filtered to display only Requested and Confirmed appointments, excluding Cancelled or Completed records.

2. Admin Module

The Admin Module provides centralised triage coordination and hospital-side configuration control, serving as the operational hub between patient requests and clinical delivery.

- [6] Review all incoming patient care requests with contact details, care reason, and symptom descriptions.
- [7] Edit appointment date, time slot, consultation mode, and intake notes prior to confirmation.
- [8] Confirm Requested appointments, advancing them through the triage queue.
- [9] Cancel Requested or Confirmed appointments; cancellation of Completed cases is explicitly prohibited.
- [10] Manage specialist profiles, consultation fees, availability slots, and linked doctor portal accounts.
- [11] Review patient messages submitted via the Contact support form.

3. Doctor Module

The Doctor Module is where clinical ownership gets enforced. Only the doctor who is assigned to a case can close it out, and once a consultation is marked complete, no one can go back and change it.

- [12] Doctors see only their own appointments -- nothing from other doctors shows up.
- [13] When they are ready to see a patient, they can confirm a Requested appointment.
- [14] To mark a visit as Completed, the doctor must first enter a consultation summary -- the system will not let

them skip that step.

- [15] They can add prescriptions and set a follow-up date as part of wrapping up the case.
- [16] Once a case is completed, it is permanently locked. No one can cancel or alter a finished consultation.

4. *Specialist Directory Module*

The Specialist Directory is the public-facing page where patients go to find the right doctor before submitting a care request. Each specialist listing shows their specialty, clinic name, years of experience, location, consultation fee, the services they offer, and when they are next available. Patients can search by symptom, specialty, or city, and jump straight into the care request form from any specialist's profile card.

C. *System Architecture Diagram*

MediZyra follows a strict Three-Tier Architectural Pattern, with all tiers operating locally on a single machine to guarantee offline-first resilience.

The Local Three-Tier Schematic

- **Presentation Layer:** A React 19 + Vite Single-Page Application (SPA) serving all three role portals (Patient, Admin, and Doctor) together with the public-facing marketing site, running entirely within the local browser environment.
- **Application Logic Layer:** An Express.js API handling authentication, role verification, appointment lifecycle state transitions, doctor note persistence, and admin triage operations via RESTful endpoints.
- **Data Persistence Layer:** A locally running MongoDB instance storing all patient records, appointments, specialist profiles, prescriptions, and admin data with full persistence across restarts.

Appointment Lifecycle State Machine

The appointment status machine enforces the following deterministic operational lifecycle:

- **Care Request:** The patient picks a specialist, chooses a date and time slot, describes their symptoms, and submits. The appointment starts in the Requested state.
- **Admin Triage:** An admin reviews the incoming request, makes adjustments if needed, adds intake notes, and then either confirms it or cancels it.
- **Doctor Confirmation:** The assigned doctor sees the confirmed appointment on their dashboard and confirms that they are ready to go ahead with the consultation.
- **Clinical Closure:** After the consultation, the doctor enters a summary, adds any prescriptions, and sets a follow-up date. The status then moves to Completed, and at that point, the record is permanently locked.

C.1 *Database Schema*

On the database side, MediZyra uses four main MongoDB collections. The users collection holds accounts for patients, admins, and doctors, each tagged with their role. The appointments collection tracks the full life of every care request - from the initial status through intake notes, clinical summaries, and prescriptions stored as embedded arrays. The specialists collection connects doctor profiles to their user accounts and keeps track of availability, fees, and services offered. Finally, the messages collection stores anything patients submit through the contact form. Relationships between collections are handled with ObjectId references to keep everything consistent.

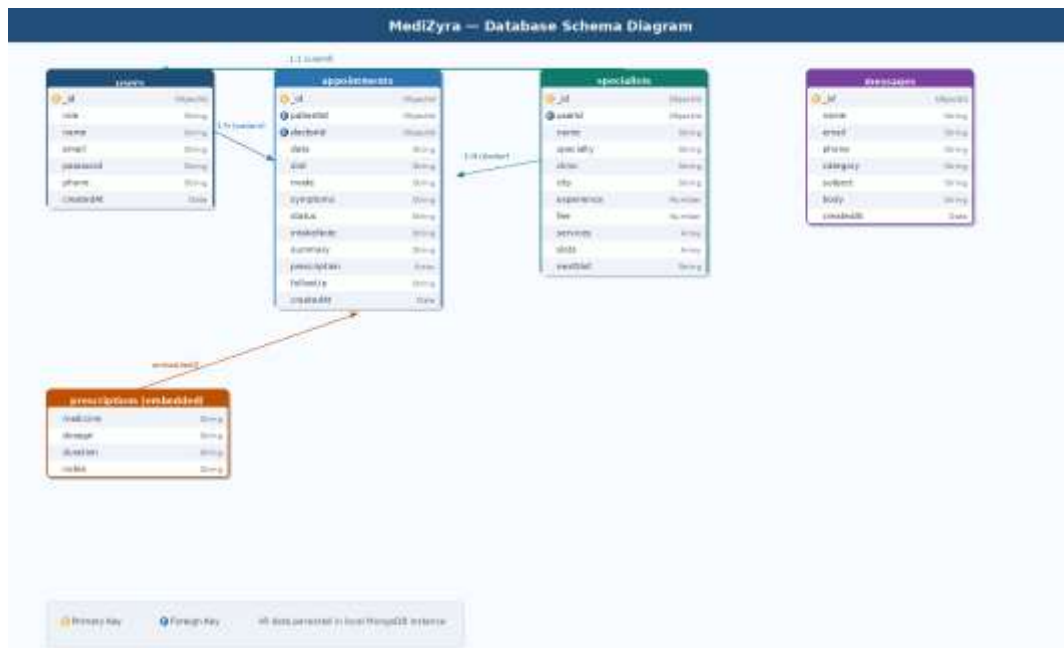


Figure 8: MediZyra MongoDB Database Schema -- Collections and Relationships

C.2 Appointment Workflow Diagram

The workflow diagram lays out how each actor -- Patient, Admin, Doctor, and the System (MongoDB) -- participates in the appointment lifecycle. It starts when a patient submits a care request, putting the appointment in Requested status. The admin picks it up from the triage board and either confirms or cancels it. If confirmed, the doctor sees it on their dashboard, confirms they are ready, conducts the consultation, and then closes the case by saving their notes and any prescriptions. Once marked Completed, that record is locked for good. MongoDB records every status change, so nothing is lost even if the system restarts.

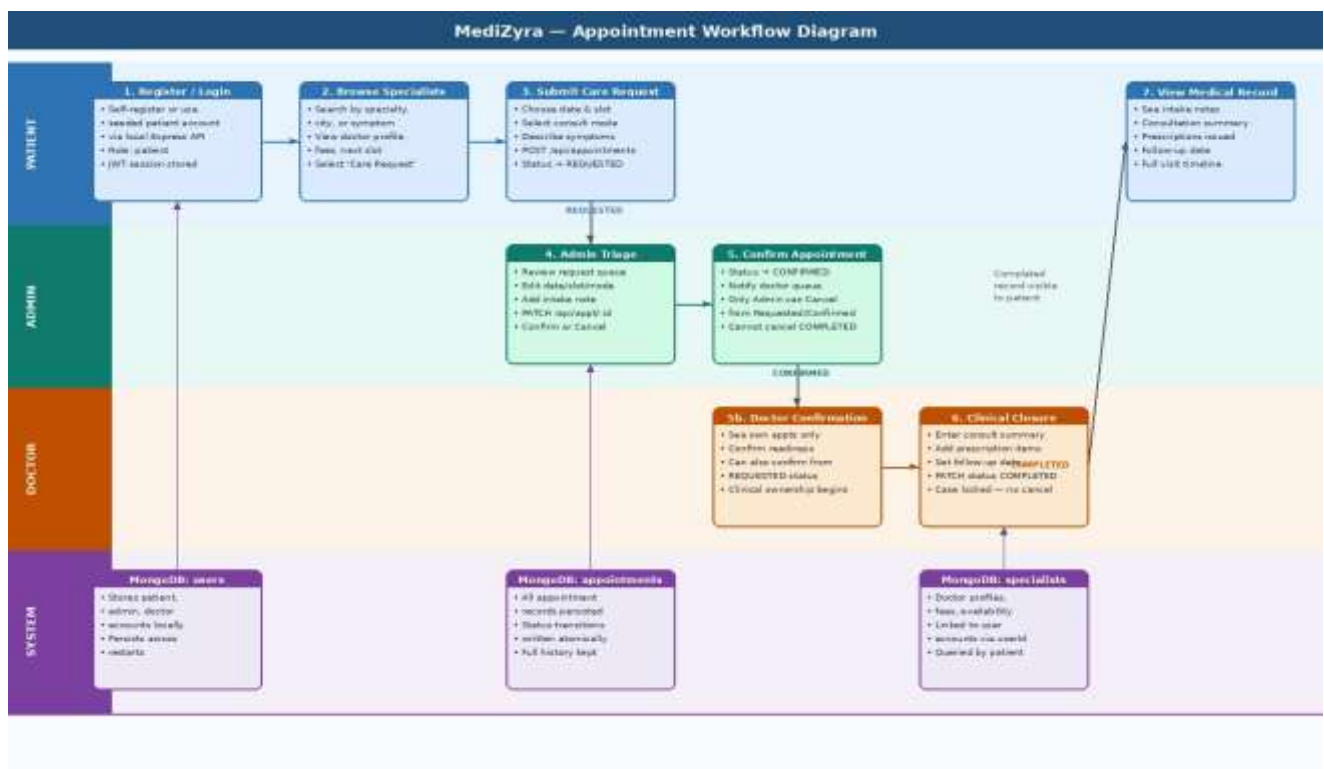


Figure 9: Appointment Lifecycle Workflow -- Role-Based Status Transitions

C.3 System Flowchart

The system flowchart maps the complete decision path from user entry through authentication, role branching, and role-specific operations to final MongoDB persistence. A new visitor either self-registers (local Express API writes to MongoDB users collection) or logs in to receive a JWT session. The system routes the authenticated user to their role portal: Patient (browse, request, view records), Admin (triage, confirm/cancel, manage profiles), or Doctor (confirm, consult, complete). All operations write through the Express.js API to the local MongoDB instance, supporting offline-first operation throughout the entire workflow.

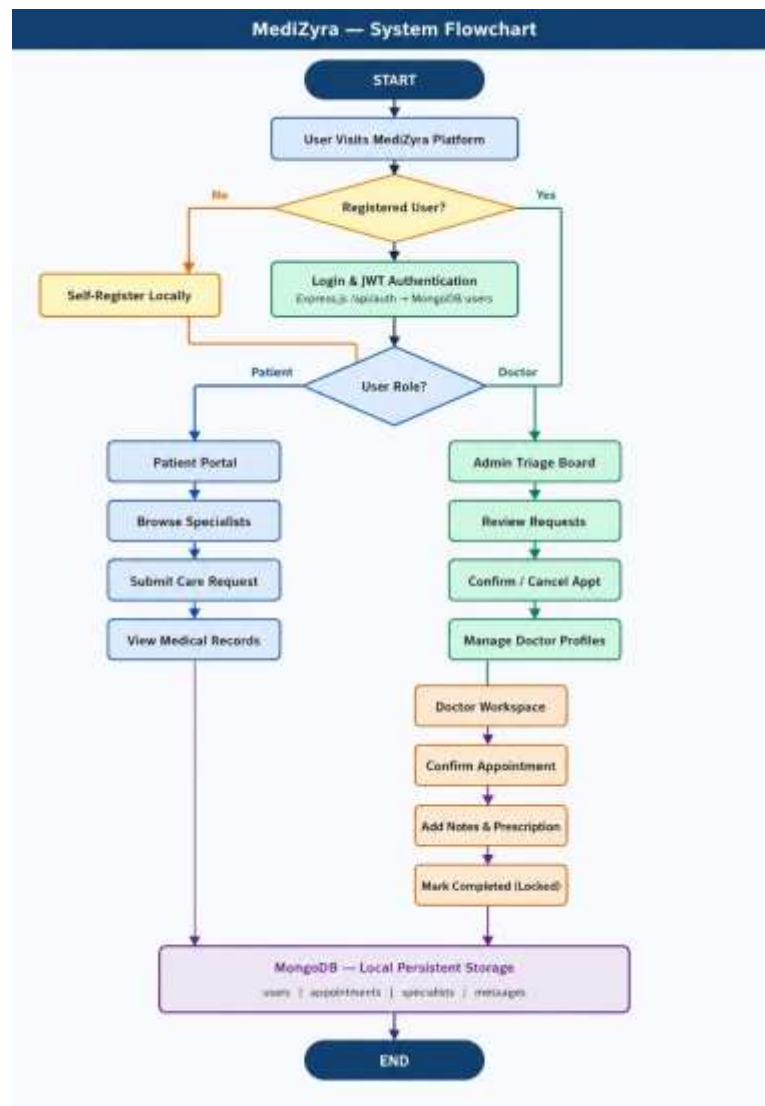


Figure 10: System Flowchart -- User Entry, Role-Based Branching, and MongoDB Persistence

D. System Flow

The working flow of MediZyra is designed to enforce clinical process integrity while maintaining operational simplicity:

- Patient registers locally or signs in using portal credentials via the login page.
- Patient browses the Specialist Directory and identifies the appropriate doctor by specialty, location, or symptom area.
- Patient submits a care request: selects a slot, consultation mode, and describes symptoms.
- Admin reviews the incoming request on the triage board, edits details, adds intake notes, and either confirms or cancels.
- Doctor views the confirmed appointment, reviews patient details, and confirms readiness to proceed.

- Doctor completes the consultation, saves clinical notes, prescriptions, and the follow-up date.
- Patient views the complete visit record -- including intake notes, consultation summary, prescriptions, and follow-up plan -- in their patient portal.

SYSTEM INTERFACE SCREENSHOTS

4.1 Landing Page -- Care Operations Overview

The MediZyra landing page provides a live care dashboard overview with key operational metrics and a high-level triage flow overview, enabling administrators and visitors to immediately assess system activity upon access.

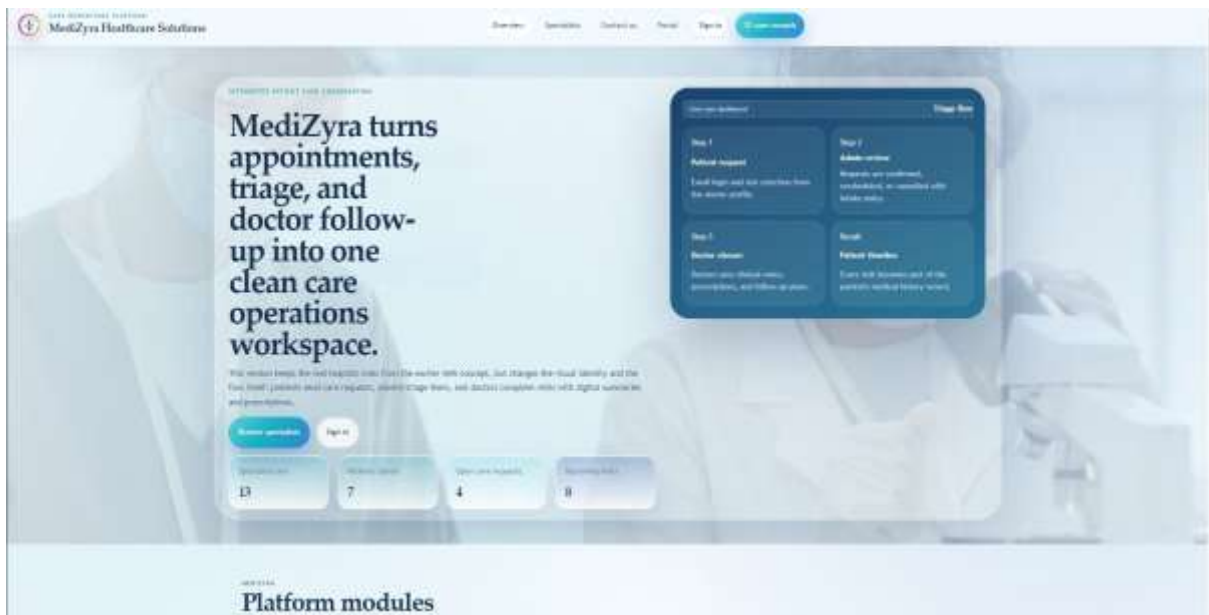


Figure 1: MediZyra Landing Page with Live Care Dashboard and Triage Flow Overview

4.2 Specialist Directory

The Specialist Directory interface enables patients to search across all registered specialists by specialty, city, or symptom area. Each specialist card displays their specialty badge, clinic location, consultation fee, experience, and available services, with a direct link to initiate a care request.

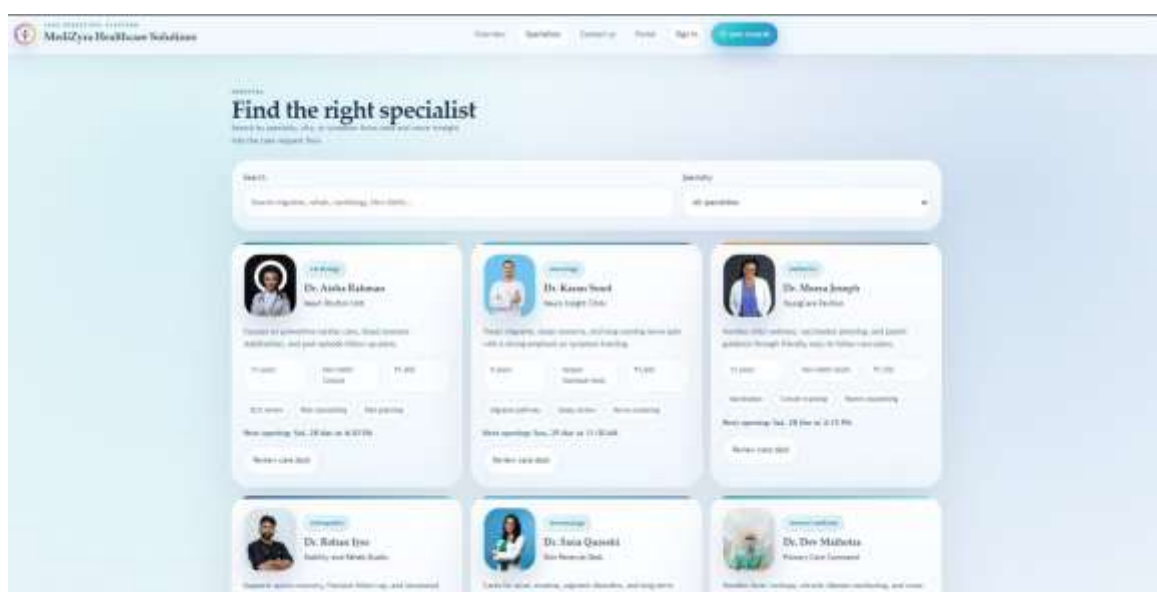


Figure 2: Specialist Directory -- Search by Specialty, City, or Symptom Area with Care Request Integration

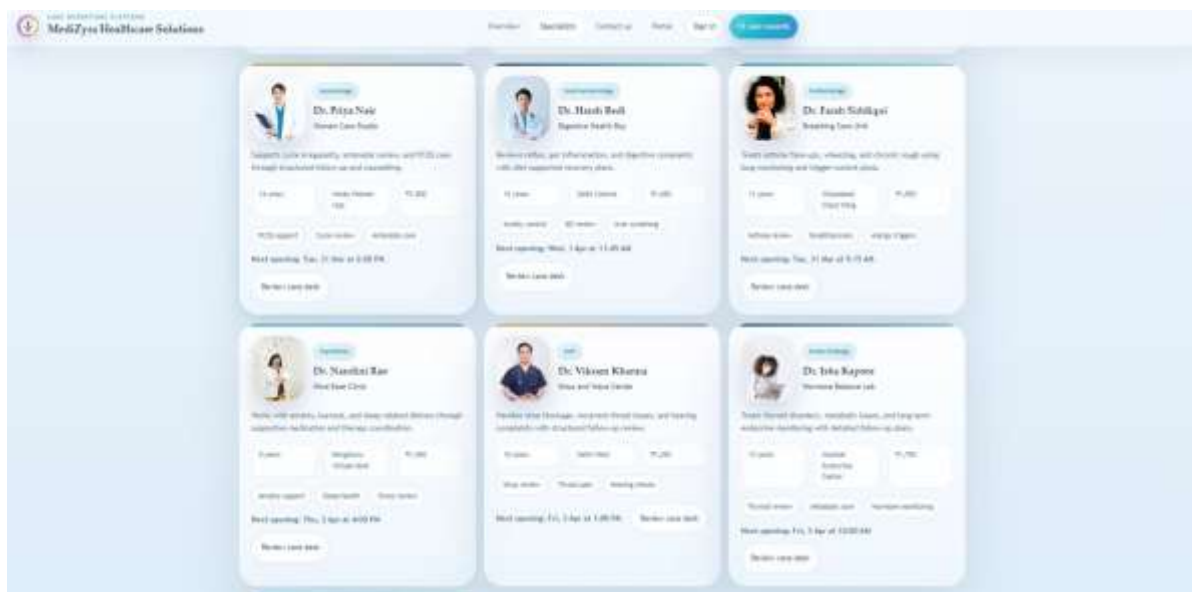


Figure 3: Specialist Directory (continued) -- Multi-Specialty Listings Including Gynecology, Gastroenterology, Pulmonology, Psychiatry, ENT, and Endocrinology

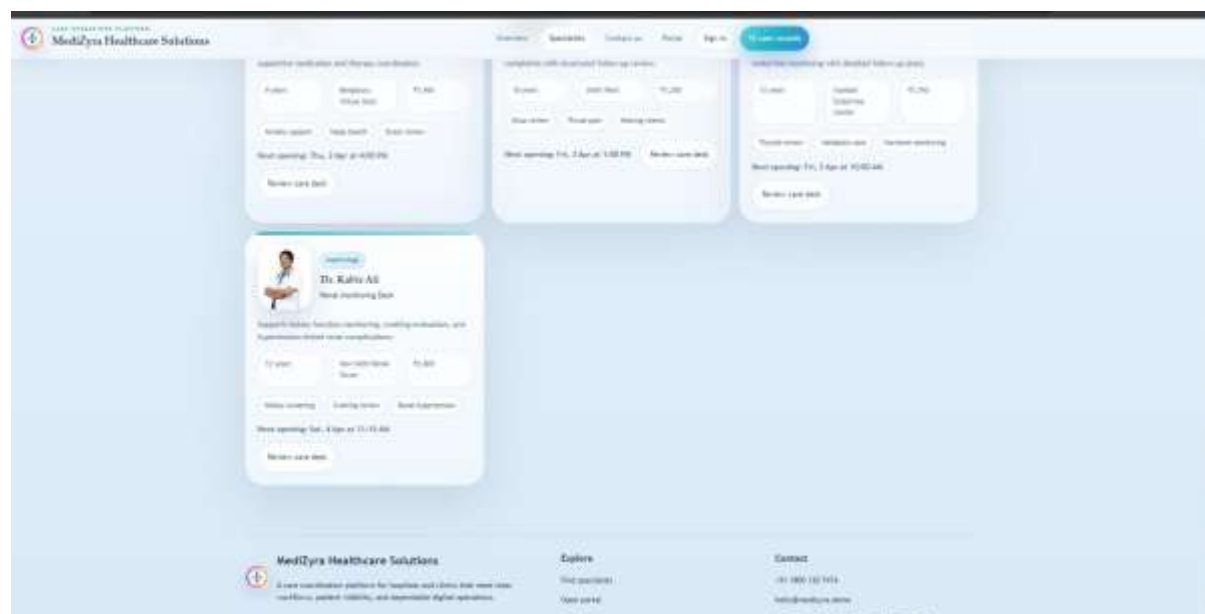
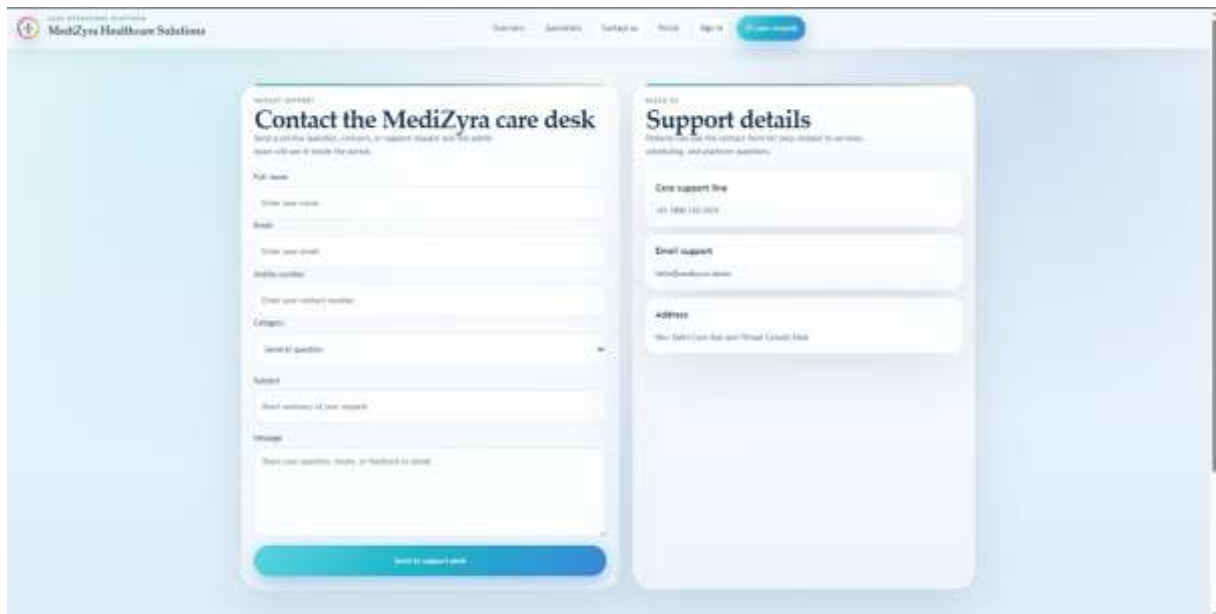


Figure 4: Specialist Directory Footer -- Nephrology Specialist and Platform Contact Details

4.3 Patient Support and Contact Page

The Contact Page provides patients with a dedicated support form alongside care desk contact information including phone number, email address, and physical address. This module enables direct patient-to- administration communication outside of the formal appointment workflow.

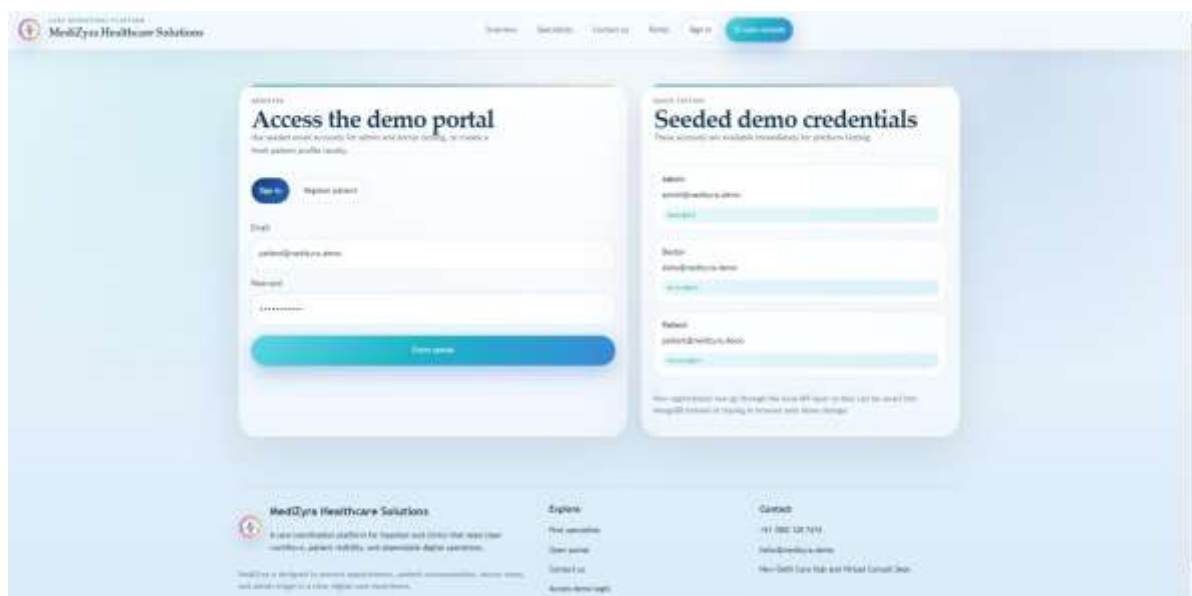


The image shows a web form titled "Contact the MediZyra care desk". It includes fields for "Full name", "Email", "Phone number", "Category" (with a dropdown), "Subject", "Message", and a "Send to support desk" button. To the right, there is a "Support details" section with "Care support line" (1800-123-7414), "Email support" (help@medizyra.com), and "Address" (New Delhi Care Hub and Patient Consult Desk).

Figure 5: Contact Page -- Patient Support Form with Care Desk Phone, Email, and Address Details

4.4 Portal Access and Seeded Demo Credentials

The Portal Login Page provides access to all three role portals with pre-seeded demo credentials for Admin, Doctor, and Patient roles, enabling immediate evaluation of the full workflow without prior account creation. All authentication is handled locally through the Express.js API integrated with the MongoDB users collection.



The image shows a web page titled "Access the demo portal" and "Seeded demo credentials". It includes fields for "Email" and "Password" for each role (Admin, Doctor, Patient) and a "Demo login" button. Below the login forms, there is a footer section with "MediZyra Healthcare Solutions" logo, "Explore" links (Home, About, Contact, Demo), and "Contact" information (Phone: 1800-123-7414, Email: help@medizyra.com, Address: New Delhi Care Hub and Patient Consult Desk).

Figure 6: Portal Login Page -- Seeded Credentials for Admin, Doctor, and Patient Roles with Local MongoDB Integration

4.5 Site Footer

The Site Footer provides quick navigation links, platform contact information, and a demo data reset option that enables evaluators to restore the system to a baseline state for repeated testing sessions.



Figure 7: Site Footer -- Explore Links, Contact Information, and Demo Data Reset Option

PEER REVIEW

Faculty members and fellow students at Bhagwan Parshuram Institute of Technology evaluated the system, looking at whether the design held together, whether the workflows behaved as expected, and whether the cyber resilience rationale was sound. Three things were examined closely: whether each role could only perform the actions it was supposed to, whether the appointment lifecycle actually enforced the kind of process control you would see in a real hospital, and whether the system truly worked without an internet connection.

The reviewers confirmed that the role boundaries worked as intended -- admins could not complete consultations, doctors could not cancel finished cases, and patients could not see other patients' records. They noted that the status-transition rules, especially the permanent lock on Completed appointments, reflected the kind of process discipline you would expect in a real clinical environment. The decision to keep everything running locally was praised as a clear and practical approach to resilience.

There were also some constructive suggestions -- clearer labels on a few interface elements, better feedback during loading states, and some inline guidance to help first-time users find their way. All of these were folded into the final version of the system.

IMPROVEMENTS BASED ON REVIEW

The system was subjected to a structured peer-validation phase aimed at identifying workflow gaps, UI clarity issues, and architectural concerns. The following improvements were implemented following faculty review:

- **1. Role Hierarchy Clarification:** Added explicit distinction between Admin and Doctor confirmation capabilities. The transition Requested → Confirmed may be performed by either Admin or Doctor; however, the transition Confirmed → Completed is restricted exclusively to the assigned Doctor.
- **2. MongoDB Integration:** Migrated from browser-only local storage to a persistent local MongoDB instance via the Express.js API layer, enabling records to survive page refreshes, browser data clears, and system restarts.
- **3. Patient View Filtering:** Improved the upcoming visits filter to exclude Cancelled and Completed appointments, ensuring that patients see only actionable future care slots.
- **4. UI Responsiveness:** Enhanced loading state indicators and form validation feedback across all three role portals, improving usability for first-time evaluators.
- **5. Security Hardening:** Implemented JWT-based session management with role-encoded claims, ensuring that API endpoints validate both authentication and role authority on every protected request.

CONCLUSION

This research demonstrates the viability of a cyber-resilient, offline-first hospital care coordination platform as a practical response to the growing threat of ransomware and connectivity-disrupting cyberattacks in healthcare environments. MediZyra Healthcare Solutions achieves full three-role workflow operation -- from patient care request

through admin triage to doctor-led clinical closure -- without dependency on cloud infrastructure or internet connectivity.

By maintaining all operational components (React 19 frontend, Express.js API, and MongoDB persistence) within a single local machine environment, the system dramatically reduces the attack surface available to ransomware and other cyber threats that routinely disable internet-connected hospital systems. The role- hierarchy enforcement, status-transition rules, and local MongoDB persistence together demonstrate that continuity-focused design principles can be practically realised in an academic prototype context.

MediZyra serves both as a functional hospital coordination prototype and as a concrete demonstration of offline- first resilience thinking relevant to real-world healthcare cybersecurity planning. The platform's modular architecture further positions it as a viable foundation for extension into production-grade hospital management systems.

FUTURE ENHANCEMENTS

There are several directions this work could go next:

- **Encrypted Local Backups:** Scheduled AES-encrypted snapshots of the MongoDB database would allow recovery from ransomware that manages to encrypt the local data store, without needing cloud backup services.
- **Tamper-Evident Audit Logging:** An audit log that records every status change, login event, and data modification would support forensic investigation after a security incident and help align the system with healthcare regulations like HIPAA.
- **Offline Export of Continuity Packs:** Giving doctors and admins the ability to export appointment summaries, prescriptions, and patient timelines as digitally signed PDFs would be valuable during extended outages, especially in paper-based or air-gapped settings.
- **Progressive Web Application (PWA) Packaging:** Packaging MediZyra as a PWA with service worker caching would open up full mobile access to all three portals, even during active connectivity outages.
- **IoT Vitals Integration:** Connecting lightweight IoT devices -- like pulse oximeters and blood pressure monitors -- to the Doctor Module would allow pre-consultation data capture in offline clinical settings, giving doctors more information before the patient even walks in.

ACKNOWLEDGMENT

We wish to express our deepest appreciation to our project guide for the guidance, assistance, and constructive suggestions provided throughout the development of this project. It was through these suggestions that we could continuously improve the system's design, resilience rationale, and clinical workflow accuracy.

We also extend our heartfelt thanks to all faculty members at Bhagwan Parshuram Institute of Technology who provided the facilities and evaluative framework necessary to complete this research.

REFERENCES

- [1] National Audit Office, "Investigation: WannaCry Cyber Attack and the NHS," UK NAO, 2018. [Online]. Available: <https://www.nao.org.uk/reports/investigation-wannacry-cyber-attack-and-the-nhs/>
- [2] HHS ASPR, "Cybersecurity in Health Care: Protecting Patients from Attacks," U.S. Department of Health and Human Services, 2022. [Online]. Available: <https://aspr.hhs.gov/cyber/Pages/default.aspx>
- [3] CISA, "Stop Ransomware," Cybersecurity and Infrastructure Security Agency, 2023. [Online]. Available: <https://www.cisa.gov/Ransomware>
- [4] React Documentation, "React 19: A JavaScript Library for Building User Interfaces," Meta Open Source, 2024. [Online]. Available: <https://react.dev/>
- [5] Vite Documentation, "Vite -- Next Generation Frontend Tooling," Evan You, 2024. [Online]. Available:

<https://vite.dev/>

- [6] Express.js Foundation, "Express -- Fast, Unopinionated, Minimalist Web Framework for Node.js," OpenJS Foundation, 2023. [Online]. Available: <https://expressjs.com/>
- [7] MongoDB Inc., "MongoDB Documentation," 2024. [Online]. Available: <https://www.mongodb.com/docs/>
- [8] C. Kruse et al., "Cybersecurity in Healthcare: A Systematic Review of Modern Threats and Trends," Technology and Health Care, vol. 25, no. 1, pp. 1-10, 2017.
- [9] D. Kumar, Y. Joshi, M. Mandal, V. Pal, "Software-Defined Smart Parking System with Real-Time Tracking and Dynamic Allocation," IJSREM, vol. 10, issue 04, April 2026.
- [10] D. Kumar, D. Summi, S. Singh, M. Jain, "A RAG-Based Health Chatbot for Verifiable Medical Queries," IJSREM, vol. 10, issue 04, April 2026.