þÿ# Medicare Project Complete Documentation

1. Project Description

1.1 Project Name

Medicare Healthcare Management System

1.2 Project Objective

The project is a full-stack healthcare platform designed to digitize patient-facing and staff-facing hospital workflows in one web application. Its primary objective is to provide:

- Public healthcare information and service discovery.
- Secure authentication and role-based access for admin, doctor, and patient users.
- Operational modules for appointments, billing, telehealth, records, communication, and analytics.
- A modern, responsive web interface with real-time updates and dashboard experiences.

1.3 Functional Scope

The system supports:

- Public website pages (home, services, departments, doctors, emergency, contact, home healthcare, pharmacy, diagnostics).
- Account lifecycle (register, login, token refresh, role persistence).
- Role-protected dashboards:
- `admin`: user and system management, incidents, tasks, reports, notifications.
- `doctor`: schedules, patient views, lab workflow, billing view, staff chat.
- `patient`: personal dashboard, appointment history, lab results, prescriptions, billing.
- Appointment booking and appointment lifecycle management.
- Telehealth session creation, join/leave, issue tracking, and completion.
- Billing and billing integration endpoints.
- Clinical data modules (patients, lab results, vitals, prescriptions).
- Engagement and analytics modules for services and departments.
- Real-time notifications and simulated live operational streams through Socket.IO.

2. Application Architecture

2.1 High-Level Architecture

- Frontend: React + Vite SPA (`client/`)
- Backend: Node.js + Express API (`server/`)
- Database: MongoDB accessed via Mongoose
- Realtime: Socket.IO (server + client)

2.2 Runtime Flow

1. User accesses React SPA routes.
2. Frontend calls REST API endpoints on the Express server.
3. Backend validates JWT and role permissions for protected endpoints.
4. Backend reads/writes MongoDB collections via Mongoose models.
5. Realtime updates are emitted via Socket.IO for selected dashboard metrics and notifications.

3. Tools and Technologies Used

This section lists the key technologies used in development and explains their purpose in this project.

3.1 Core Platform and Language

- JavaScript: primary language in both frontend and backend.
- Node.js: backend runtime.
- npm: dependency and script management.

3.2 Frontend Technologies (client)

- React 19: component-based UI architecture.
- React DOM: browser rendering for React.
- Vite: dev server and build tooling.
- React Router DOM: SPA routing and protected navigation.
- Tailwind CSS (+ PostCSS + Autoprefixer): utility-first styling pipeline.
- Framer Motion: UI animations and transitions.
- Axios + Fetch wrappers: API request handling.
- Socket.IO Client: real-time client communication.
- React Grid Layout: configurable dashboard layouts.
- React Hot Toast: notifications/toasts.
- React Icons, Heroicons, Lucide React: icon systems.
- Chart.js + React ChartJS 2 + Recharts: data visualizations.
- React Calendar + React Datepicker: date and schedule interactions.
- DnD Kit (`@dnd-kit/*`): drag-and-drop UX.
- Fuse.js: client-side fuzzy search behavior.
- HTML2Canvas + jsPDF + xlsx + FileSaver: export/report features.
- Leaflet: map/location visuals.
- QRCode React: QR generation.
- React Speech Recognition: voice interaction utility.
- React Tooltip, React Window: tooltip and list optimization utilities.

3.3 Backend Technologies (server)

- Express 5: REST API framework.
- Mongoose: ODM and schema/model layer for MongoDB.
- JSON Web Token (`jsonwebtoken`): authentication tokens.
- bcrypt / bcryptjs: password hashing.
- CORS: cross-origin access control.
- Helmet: security headers.
- express-rate-limit: request throttling.
- compression: response compression.
- dotenv: environment variable management.
- Socket.IO: realtime server push.
- node-fetch, axios: server-side HTTP integrations.

3.4 Testing, Quality, and Development Tools

- ESLint: linting for code quality.
- Nodemon: backend auto-restart during development.
- Mocha + Chai + Chai HTTP + Supertest: backend/API testing.
- Jest + Testing Library (`@testing-library/*`): frontend unit/integration tests.
- Cross-env: cross-platform environment variable support in scripts.
- Babel (`@babel/*`): test/build transpilation support.

3.5 Build and Deployment Related Files

- `railway.toml`: deployment configuration (Railway style deployment metadata).
- `start-dev.bat`: local helper script to run development services.

4. Database Documentation

4.1 Database Type and Purpose

- Database type: MongoDB (NoSQL document database)
- ODM: Mongoose
- Default local database name: `healthcare_db` (from server connection fallback)

Purpose of database usage:

- Persist user identities, healthcare records, appointments, telehealth sessions, billing, settings, operational logs, and engagement analytics.
- Support both transactional workflows (appointments, billing, prescriptions) and content-driven/public modules (departments, services, home content, emergency content).

4.2 Database Structure

The database is organized as multiple Mongoose collections, each mapped to a domain model in `server/models/`.

4.2.1 Core Clinical and Identity Collections

- `User`: credentials, role (`admin|doctor|patient`), profile basics, gamification points.
- `Patient`: patient demographics, medical history, allergies, medications, insurance, emergency contact.
- `Doctor`: professional profile, specialization, licensing, availability, ratings.
- `Appointment`: booked consultation details, patient and doctor snapshot data, timing, status, prescription linkage.
- `Prescription`: medication/treatment records tied to patient and prescriber.
- `LabResult`: diagnostic result documents with status and follow-up metadata.
- `Vitals`: physiological measurements and recording metadata.
- `Billing`: invoices, line items, payment status/method, insurance claim details.
- `BillingIntegration`: additional billing/payment processing integration records.
- `TelehealthSession`: virtual consultation sessions linked to appointment/patient/doctor.

4.2.2 Operations and Administration Collections

- `Notification`
- `ActivityLog`
- `Task`
- `Incident`
- `Report`
- `CalendarEvent`
- `Settings` (singleton-style system configuration)
- `StaffMessage`
- `Agent`
- `Branch`
- `Feedback`
- `Contact`
- `Certification`

4.2.3 Public Content and Service Experience Collections

- `Department`
- `DepartmentAnalytics`

- `DepartmentAnnouncement`
- `DepartmentChatLog`
- `DepartmentEngagement`
- `DepartmentHighlight`
- `DepartmentInsurance`
- `DepartmentRecommendation`
- `DepartmentReview`
- `DepartmentSchedule`
- `Service`
- `ServiceAnalytics`
- `ServiceCompare`
- `ServiceFavorite`
- `ServiceQuizResult`
- `ServiceReview`
- `HomeContent`
- `HomeHealthcareService`
- `HomeHealthcareTeam`
- `HomeHealthcareStory`
- `HomeHealthcarePackage`
- `HomeHealthcareCertification`
- `HomeHealthcareAward`
- `HomeHealthcareBlogPost`
- `HomeHealthcarePress`
- `HomeHealthcarePartner`
- `EmergencyMetric`
- `EmergencyIncident`
- `EmergencyHospital`
- `EmergencyChecklist`
- `PharmacyItem`
- `Cart`
- `Order`

4.2.4 Engagement and Behavior Tracking Collections

- `FavoriteDoctor`
- `ArticleRead`
- `WellnessEnrollment`
- `ChallengeJoin`
- `MedicationRefillRequest`
- `MoodLog`
- `SymptomCheck`
- `ReminderLog`
- `InsuranceView`
- `NewsletterSubscriber`
- `OfferClaim`
- `PricingCalcResult`

4.3 Schema Explanation (Key Models)

4.3.1 `User` Schema

Core fields:
- `name`, `email` (unique), `password`
- `role` enum: `admin`, `doctor`, `patient`
- `points` numeric accumulator

Role is used by middleware to enforce route-level access control.

4.3.2 `Patient` Schema

Includes:
- `userId` reference to `User`
- Personal and demographic data (`dateOfBirth`, `gender`, blood group, contact)
- Structured nested data for address, emergency contact, history, medications, insurance

Purpose: patient-centric clinical profile consumed by appointments, billing, lab, and vitals modules.

4.3.3 `Doctor` Schema

Includes:
- `userId` reference to `User`
- medical profile (`specialization`, `licenseNumber`, `experience`, `education`)
- operational availability slots
- consultation fee and ratings

Purpose: doctor directory and scheduling context for appointments/telehealth.

4.3.4 `Appointment` Schema

Includes:
- embedded patient and doctor snapshot blocks
- date/time fields and appointment `status` lifecycle
- consultation details and optional `prescriptionId` reference
- indexes for patient/date and doctor/date queries

Purpose: core scheduling and consultation workflow entity.

4.3.5 `Billing` Schema

Includes:
- foreign references: `patientId`, `appointmentId`, `doctorId`
- invoice line `items`, subtotal/tax/discount/total
- payment and insurance claim states

Purpose: billing pipeline and payment tracking for completed services.

4.3.6 `TelehealthSession` Schema

Includes:
- references to appointment/patient/doctor
- session status progression
- start/end/duration, meeting URL, issue logs, participant tracking
- indexes for retrieval by appointment, patient, doctor, and time

Purpose: virtual consultation lifecycle management.

4.3.7 `Department` and `Service` Schemas

- `Department` stores departmental metadata, doctors, occupancy, wait times, service tags.

- `Service` stores public-facing service content, delivery details, pricing metadata, FAQ, testimonials, portfolio and analytics tie-ins.

Purpose: content-driven discovery experience and conversion/engagement tracking.

4.4 Entity Relationships

Main relationships:
- `User` 1-1 `Patient` (via `Patient.userId`)
- `User` 1-1 `Doctor` (via `Doctor.userId`)
- `Patient` 1-many `Appointment`, `LabResult`, `Vitals`, `Billing`
- `Doctor` 1-many `Appointment`, `Vitals`, `Billing`, `TelehealthSession`
- `Appointment` 1-1/1-many linked domain records (`Prescription`, `TelehealthSession`, `Billing`)
- `User` 1-many `Notification`, `ActivityLog`, and engagement logs
- `Department` and `Service` 1-many analytics/interaction collections

MongoDB references (`ObjectId + ref`) are used where cross-collection joins/population are needed; nested subdocuments are used for grouped attributes that are naturally read/written together.

4.5 How the Database is Used in the Application

Typical flow examples:

1. Registration/Login
- User registers via `/api/auth/register`, creating a `User` document.
- On login, JWT is issued and role is stored client-side.

2. Appointment Booking
- Public booking endpoint stores an `Appointment`.
- Staff/role users later manage status/reschedule through protected endpoints.

3. Clinical Tracking
- `LabResult` and `Vitals` are stored and retrieved through patient/doctor workflows.

4. Billing Processing
- `Billing` document is created from care events; payment status updates and integration endpoints process settlements.

5. Public Content + Engagement
- Public pages pull data from `Department`, `Service`, `HomeContent`, and emergency/home-healthcare collections.
- User actions are logged to engagement collections for analytics.

6. Dashboard Realtime Updates
- Socket.IO pushes live metrics/events; persisted notifications and records remain in MongoDB.

5. API and Module Map

Backend route groups mounted in `server/index.js` include:

- `/api/auth`
- `/api/admin`
- `/api/appointments`
- `/api/patients`
- `/api/doctors`

- `/api/billing`
- `/api/billing-integration`
- `/api/telehealth`
- `/api/calendar`
- `/api/feedback`
- `/api/activity`
- `/api/orders`
- `/api/cart`
- `/api/vitals`
- `/api/notifications`
- `/api/user`
- `/api/certifications`
- `/api/public`
- `/api/engagements`
- `/api/services`
- `/api/service-engagements`
- `/api/department-engagements`
- `/api/lab-results`
- `/api/staff-chat`
- `/api/schedule`
- plus support/health/insights/contact related endpoints under `/api/*`.

Authentication model:
- Public and protected routes are separated.
- `verifyToken` + role middleware enforce access for protected resources.

6. Web Application Site Map (Navigation Structure)

6.1 Public Routes

- `/` -> Home / Hero
- `/services` -> Services
- `/departments` -> Departments
- `/doctors` -> Doctors directory
- `/contact` -> Contact and support
- `/emergency` -> Emergency module
- `/login` -> Login
- `/register` -> Registration
- `/book-appointment` -> Appointment booking
- `/pharmacy` -> Pharmacy
- `/diagnostic` -> Diagnostic services
- `/home-healthcare` -> Home healthcare content
- `/viewdashboard` -> Dashboard preview page

6.2 Protected Routes

- Patient:
- `/dashboard/patient`
- `/patient/lab-results`
- `/patient/prescriptions`
- `/patient/billing`

- `/appointment-history`

- Doctor:
- `/dashboard/doctor`
- `/doctor/patients`
- `/doctor/schedule`
- `/doctor/lab-results`
- `/doctor/chat`
- `/doctor/billing`

- Admin:
- `/dashboard/admin`

6.3 Navigation Flow Summary

1. Unauthenticated users browse public pages and can book appointments.
2. Authentication routes grant role-based session state.
3. ProtectedRoute checks role and redirects unauthorized users.
4. Each role enters dedicated dashboard paths and related sub-pages.

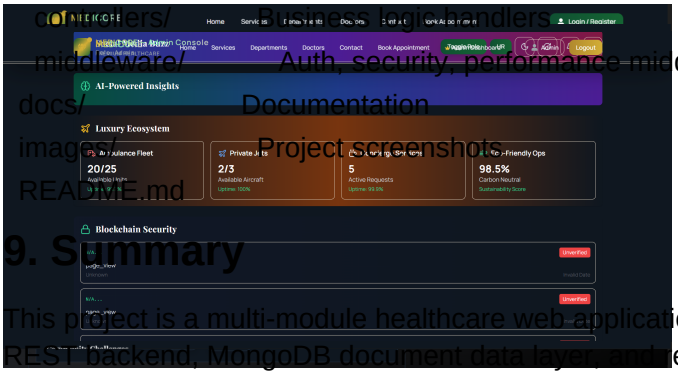
7. Screenshots (Web Application Visual References)

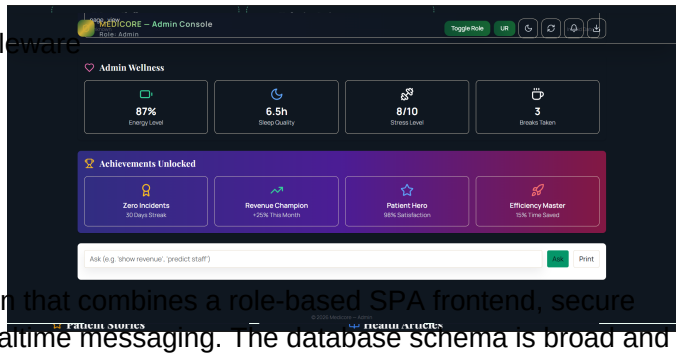
All screenshots below are captured in sequence and arranged in page order for documentation review and hardcopy submission.

7.1 Ordered Full Website Screenshots

8. Project Structure (Current)

```
medicare/
  client/              React frontend
  server/              Express backend
    models/              Mongoose schemas/collections
    routes/              API route modules
    controllers/         Business logic handlers
    middleware/          Auth, security, performance middleware
  docs/                Documentation
    images/              Project screenshots
  README.md
```



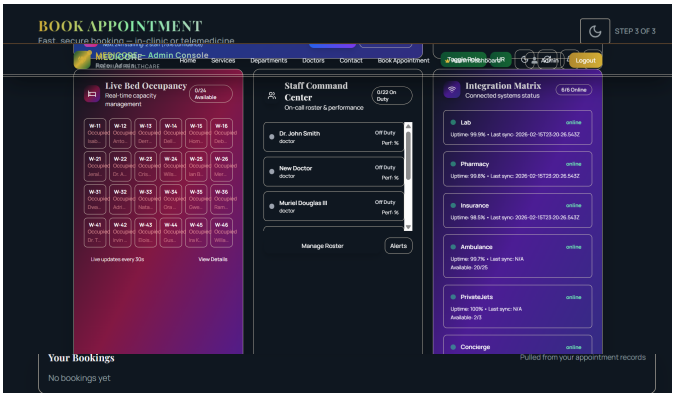


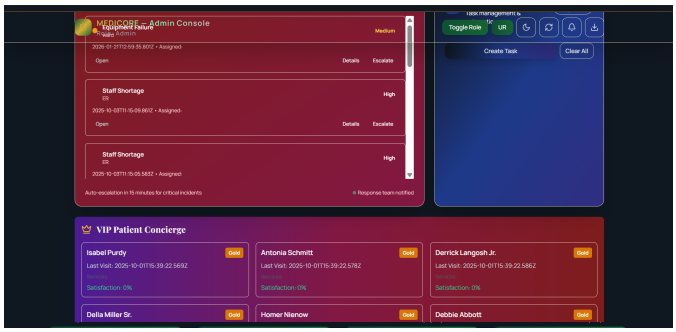
9. Summary

This project is a multi-module healthcare web application that combines a role-based SPA frontend, secure REST backend, MongoDB document data layer, and realtime messaging. The database schema is broad and modular, covering identity, clinical, billing, operations, and public engagement needs. The sitemap and screenshots in this file provide both structural and visual understanding of the system.

085 Manual Screenshot

088 Manual Screenshot





085 Manual Screenshot

088 Manual Screenshot