Final Project Report

Healthcare Patient Management System using MongoDB

Ragini Gupta

Website URL: http://localhost:5173/

1. Problem Statement

Healthcare providers must manage a wide range of patient data, including personal information, medical history, test results, prescriptions, doctor notes, and appointment logs. This data is often semi-structured and varies significantly from patient to patient, making it difficult to model and scale using traditional relational databases.

This project aims to design and develop a Healthcare Patient Management System that utilizes MongoDB as the core database. MongoDB, being a flexible, document-oriented NoSQL database, is well-suited for managing complex and dynamic medical records. It supports a schema-less design, allows for efficient querying, and handles large volumes of heterogeneous data with ease.

2. Objectives

- ✅ Emphasize the practical use of MongoDB for managing complex and variable healthcare data.
- ✅ Provide hands-on experience with NoSQL data modeling and query optimization.
- ✅ Enable secure operations such as adding, searching, updating, and deleting patient records.
- ✅ Generate clinical and administrative insights using MongoDB's aggregation framework.
- ✅ Allow full flexibility in frontend/backend stack (React + Express + Node + MongoDB used here).

3. Key Modules Implemented

- 📌 Insert New Patient Record

Form-based interface to add patients. Fields include name, DOB, gender, emergency contact, allergies, chronic conditions, medications, and more. Data stored in the `patients` collection in MongoDB.

- 📌 Search Patient Record

Doctors can search by name, ID, or partial keywords using MongoDB's `$regex` and text indexes. Implemented pagination and filters via React Query.

- 📌 Update Patient Record

Embedded forms for updating health metrics, appointments, and medication data. Uses MongoDB operators such as `$set`, `$push`, `$pull` to modify nested arrays.

- 📌 Delete Patient Record (In Progress)

Securely deletes record via unique patient ID. Confirmation prompt appears before deletion. Audit log feature will log deleted ID and user for compliance.

- 📌 Optimized Search

Indexes created on frequently queried fields (`name`, `email`, `doctorId`). Query performance analyzed via `.explain()` in MongoDB.

- 📌 Aggregation & Analytics

Health metrics visualization using Chart.js. Aggregated data like average blood pressure, weight trends, etc.

4. Technology Stack

Layer	Technology	Justification
Frontend	React + Tailwind + DaisyUI	Responsive, customizable UI with component reuse.
Backend	Node.js + Express	Handles routing, JWT authentication, and REST APIs.
Database	MongoDB (Mongoose)	Schema-less design ideal for flexible medical data.

5. Security Features

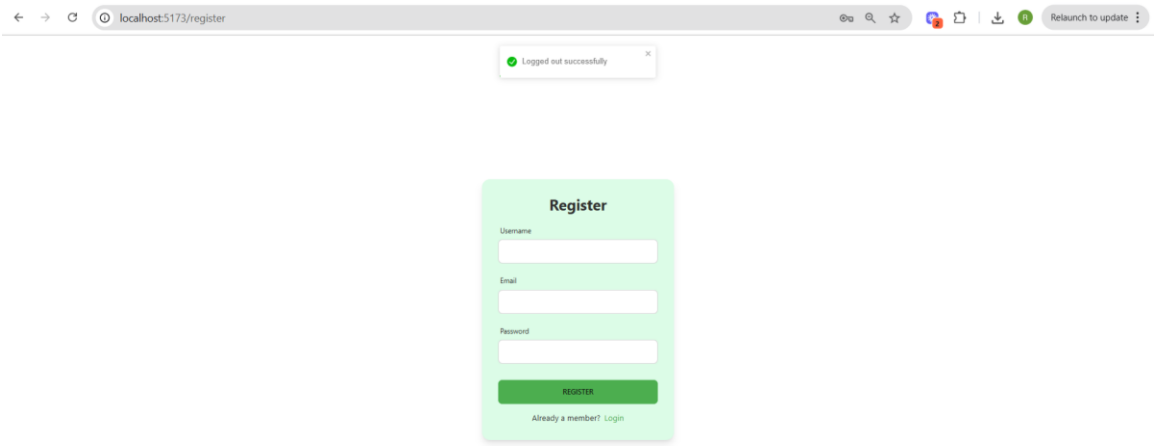
- 🔒 JWT-based authentication using Redux state.
- 🔒 Protected routes using `<ProtectedRoute>` wrapper.
- 🔒 Planned: Role-based access control and audit logging.

6. Future Enhancements

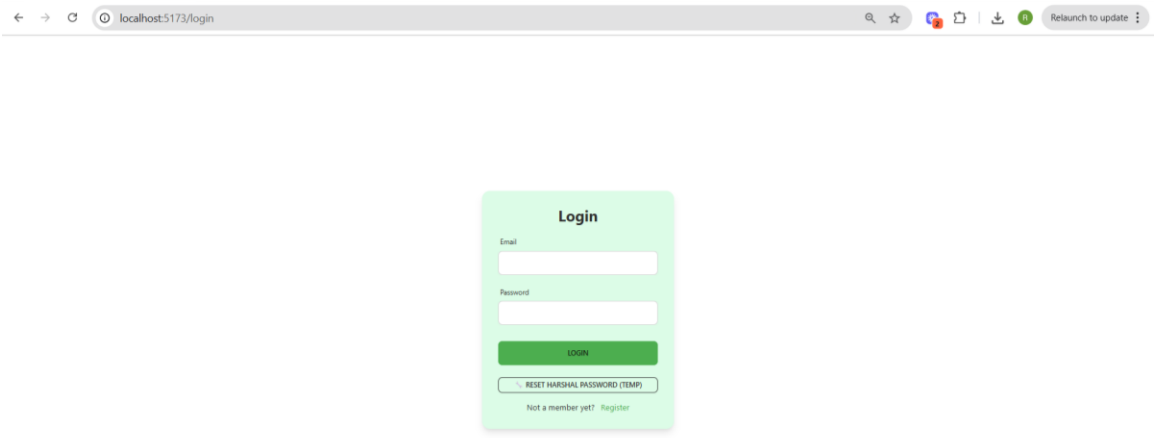
- 🚀 Implement deletion audit logging using a separate `logs` collection.
- 🚀 Add role-based dashboards (Admin vs Doctor).
- 🚀 Export patient records to PDF.
- 🚀 Mobile-responsive views and push notifications.

7. UI Mockup

Register page

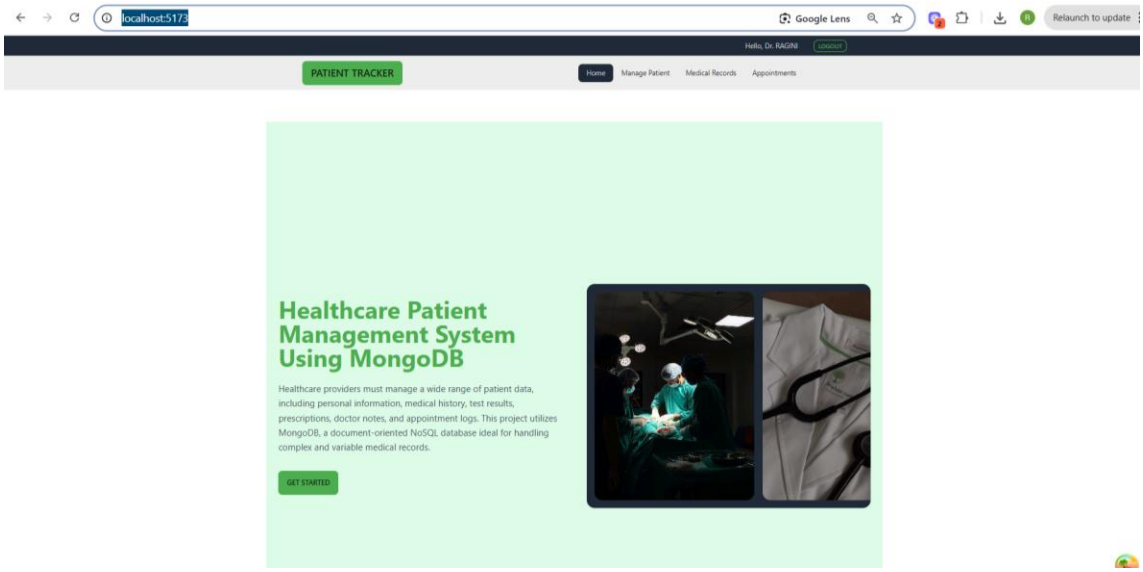


Login page

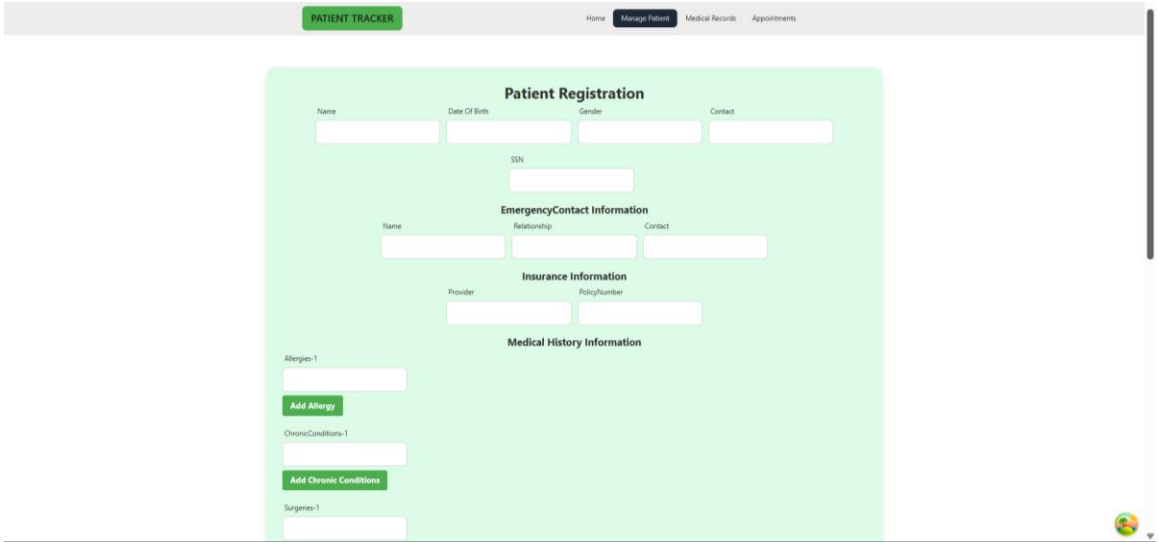


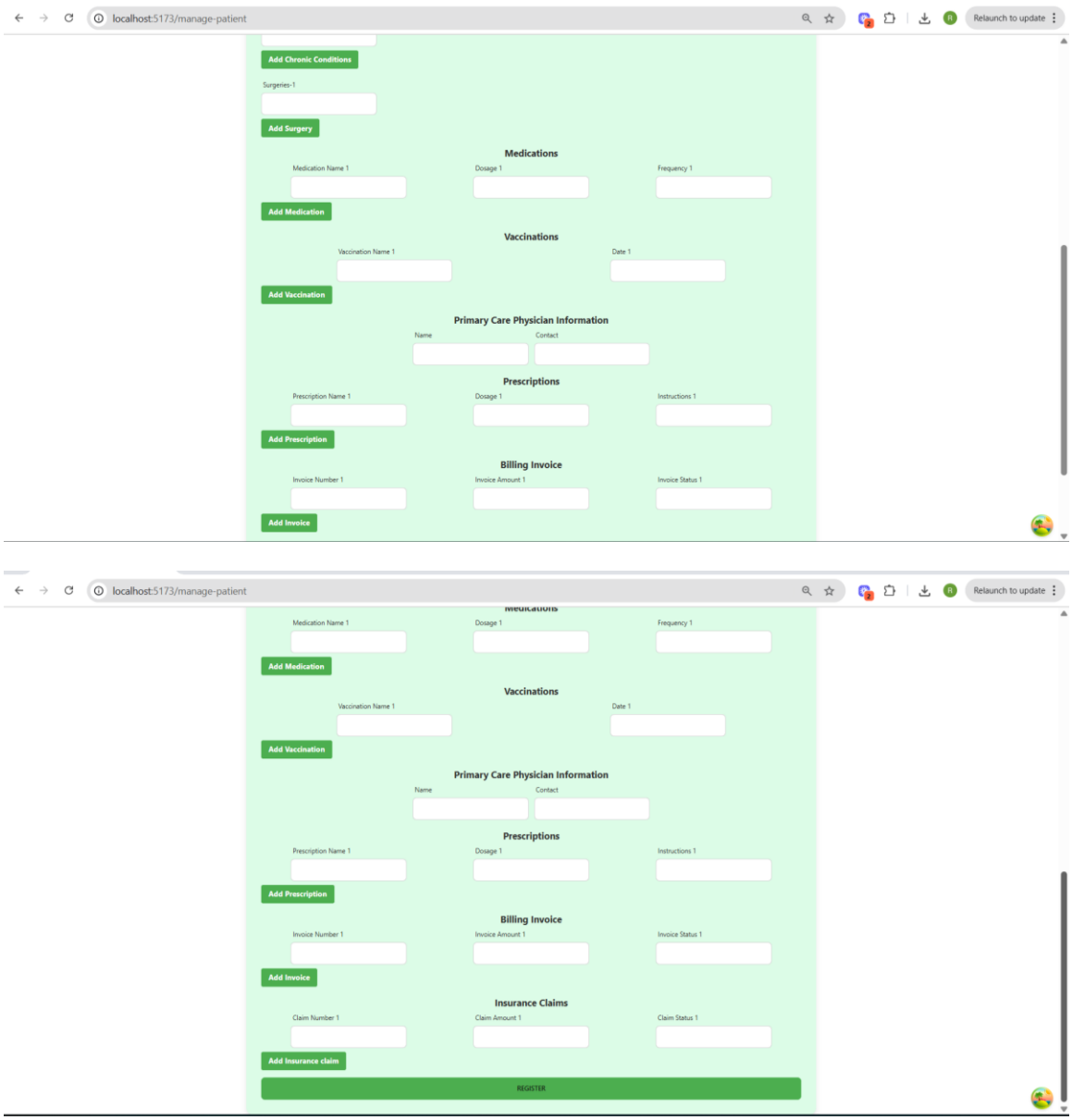
• Landing Page with carousel and green theme

Home page and Naviagtion bar



• Patient registration with multiple fields

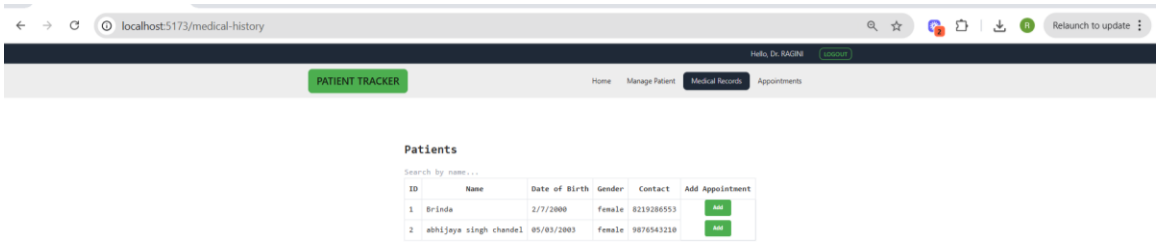




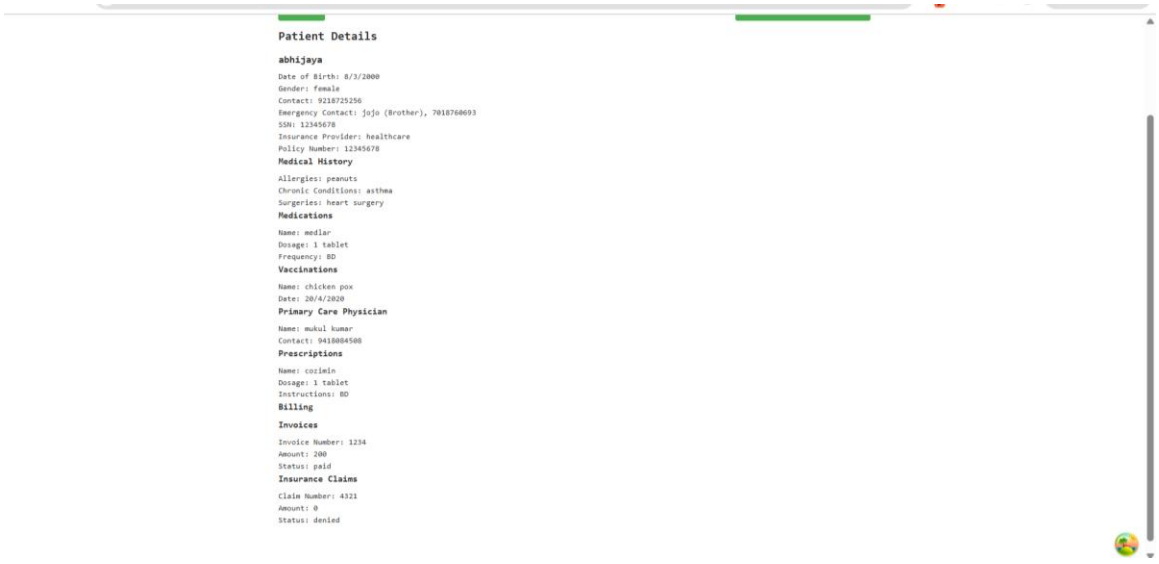
• Appointment Scheduler

Medical records for the patients for the logged in doctor (Doctor test)

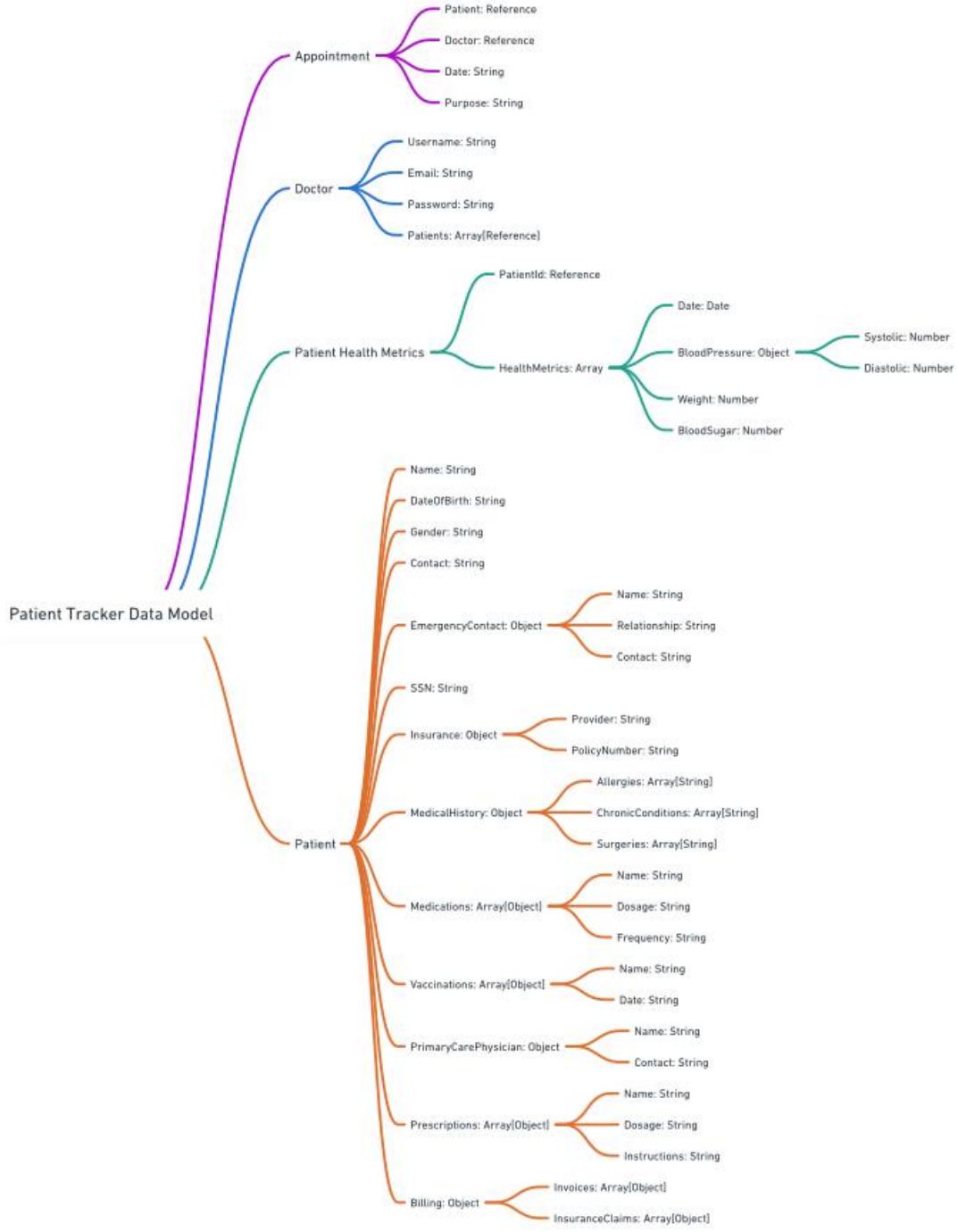
Medical records for patients for another logged in doctor (Doctor test2)



Clicking on the particular patient for more details



Data Model



1. Appointment Model

patient: Reference to Patient (ObjectId)

doctor: Reference to Doctor (ObjectId)

date: String

purpose: String

2. Doctor Model

username: String

email: String (unique, required)

password: String (required)

patients: Array of references to Patients (ObjectId)

3. Patient Health Metrics Model

patientId: Reference to Patient (ObjectId, required)

healthMetrics: Array of Health Metric objects

date: Date (required)

bloodPressure: Object (systolic: Number, diastolic: Number)

weight: Number

bloodSugar: Number

4. Patient Model

name: String

dateOfBirth: String

gender: String

contact: String

emergencyContact: Object (name, relationship, contact)

ssn: String

insurance: Object (provider, policyNumber)

medicalHistory: Object (allergies: Array, chronicConditions: Array, surgeries: Array)

medications: Array of medication objects (name, dosage, frequency)

vaccinations: Array of vaccination objects (name, date)

primaryCarePhysician: Object (name, contact)

prescriptions: Array of prescription objects (name, dosage, instructions)

billing: Object (invoices: Array, insuranceClaims: Array)

The diagram illustrates the following entities and their relationships:

Doctor: Manages the patient profiles. They also can customize their dashboard to tailor the user experience.

Patient Profile: Central entity that contains both personal and medical details of the patient. It's also linked to the data

visualization component.

Personal Details: Information such as name, age, contact details, etc.

Medical Details: Information about a patient's health, medical history, medications, treatments, etc.

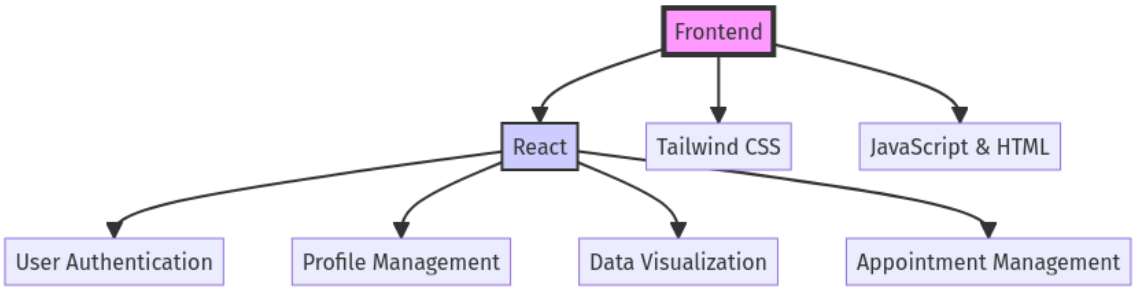
Data Visualization: This component is linked to the patient profile and displays patient data and health trends.

Authentication Details: Contains login credentials and authentication-related information for the patient.

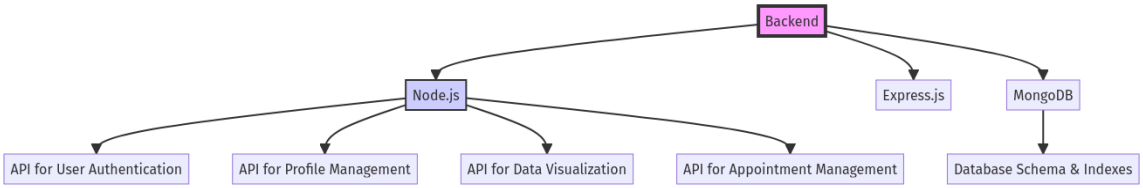
Authentication Module: Ensures authentication details are secured and manages the log in process.

Architectural Design

1. Frontend Architecture



2. Backend Architecture



Functional Requirements

Doctor's Dashboard Viewing:

UI Interaction: I will access the dashboard displaying a daily schedule with a listview. Each appointment is clickable for more details.

Error Handling: If no appointments are scheduled, display a message "No appointments today".

Patient's Profile Update:

UI Interaction: I want to update patient profiles using a form with fields for medications and allergies. Each field includes validation to ensure accurate data entry.

Error Handling: On No data entry, display specific error messages (e.g., “Form not filled"). For update failures, show "Failed to update patient information".

Doctor's Data Visualization:

UI Interaction: I want to view patient health trends via interactive charts within the patient's profile.

Error Handling: If data is missing, display "Data not available". For retrieval errors, show "Unable to load health trends".

Patient's Appointment Booking:

UI Interaction: I want to view and manage appointment requests through an interactive list. I want to delete or edit appointments with a few clicks.

Error Handling: In case of booking errors, display "Unable to process the appointment request".

Doctor's Patient Search:

UI Interaction: I want to use a search bar to find patients by name or ID. Search results should be presented in a list, with options to view full profiles.

Error Handling: For no results, display "No patients found".

Doctor's Treatment Notes:

UI Interaction: After consultations, I want to add notes to patient profiles via a text editor. Options to edit or delete notes are available.

Doctor's Prescription Management:

UI Interaction: I want to prescribe medications along with instructions to the patient.

Patient's Historical Data Viewing:
UI Interaction: I want to access a comprehensive history of patient appointments, prescriptions, and notes in the patient profile.
Error Handling: For missing data, display "Data not available". If data loading fails, show "Error loading historical data".
Patient's Medical History Management:
UI Interaction: I want to update a patient's medical history, including "Allergies" (with an option for 'None'), "Chronic
Conditions" (e.g., Asthma), and "Surgeries" (e.g., Knee Replacement), through a structured and editable section in the patient's
profile.
Error Handling: The system validates entries for completeness and correctness, displaying error messages like "Invalid input" or
"Incomplete data" when necessary, and alerts on system failures with "Update failed, try again later".
Billing Invoice and Insurance Claims Management:
UI Interaction: I want to generate and manage billing invoices using a digital form. I want to submit insurance claims directly
through the system.
1.4. Non-Functional Requirements
Performance:
Load Time: Quick loading within seconds.
Response Time: Instantaneous system feedback for user actions.
Capacity: Handles thousands of concurrent users.
Reliability:
Uptime: At least 99.5% system availability.
Data Accuracy: Error-free data display and storage.
Backup & Recovery: Scheduled data backups with a recovery plan.
Security:
Data Encryption: Secure patient data during storage and transmission.
Access Control: Role-based data access.
Audit Trails: Logged user interactions for accountability.
User Experience (UX):
Intuitive Design: Easy navigation and understanding of system features.
Database Scalability:
Handles growing data without performance drops.
Maintainability:
Modular Design: Independent system modules for easy updates.
Documentation: Clear instructions for future development

Incorporating the core modules through the database using terminal

```
mongosh
 show dbs
use patient_data
show collections

To check existing
db.patients.findOne()


To insert new patients through database
db.patients.insertOne({
name:'Ragini',
dateOfBirth:'12/8/2005',



Search patient through database
Search $regex
sb.patients.find({ name: {$regex: "Ginny", $options: "i" } }).pretty()

Search $in $or $regex
db.patients.find({
 $or: [
  { name: { $regex: "ragini", $options: "i" } },  // case-insensitive name match
  { "medicalHistory.chronicConditions": { $in: ["Asthma", "Diabetes"] } },
  { "medicalHistory.allergies": { $in: ["Dust", "Rash"] } }
 ]
}).pretty()


Full text search with text index
db.patients.createIndex({
 name: "text",
 "medicalHistory.chronicConditions": "text",
 "medicalHistory.allergies": "text"
})
```

```
db.patients.find({
  $text: { $search: "ragini asthma rash" }
}).pretty()
```

Text Search Query with Pagination & Sorting

```
const page = 2;
const pageSize = 10;

db.patients.find(
  { $text: { $search: "asthma rash" } },
  { score: { $meta: "textScore" } }
)
.sort({ score: { $meta: "textScore" } })
.skip((page - 1) * pageSize)
.limit(pageSize)
.pretty()
```

```
db.patients.countDocuments({ $text: { $search: "asthma rash" } })
```

to modify existing patient data in MongoDB, you'll use update queries with operators like:
$set → update or add a value
$push → append to an array
$pull → remove from an array
$addToSet → add to array only if it doesn't already exist

```
db.patients.updateOne(
 { name: "Ragini" },
 {
  $set: {
   "prescriptions.0.name": "Amoxicillin",
   "prescriptions.0.dosage": "500mg",
   "prescriptions.0.instructions": "Twice daily after meals"
  }
 }
)

----------------
db.patients.updateOne(
 { name: "Ragini" },
 {
  $push: {
   "medicalHistory.chronicConditions": "Asthma",
   "medicalHistory.allergies": "Pollen"
  }
 }
)


---------------
db.patients.updateOne(
 { name: "Ragini" },
 {
  $addToSet: {
   labReports: {
    testName: "CBC",
    result: "Normal",
    date: new Date("2025-06-05")
   }
  }
 }
)


------------------------


db.patients.updateOne(
```

```
  { name: "Ragini" },
  {
    $pull: {
      medications: { name: "" }
    }
  }
)
```

```
db.patients.updateOne(
  { name: "Ragini" },
  {
    $set: {
      "doctorNotes": "Patient shows improvement after antibiotic therapy"
    },
    $push: {
      "medicalHistory.chronicConditions": "Sinusitis",
      labReports: {
        testName: "X-Ray Chest",
        result: "Clear",
        date: new Date("2025-06-05")
      }
    }
  }
)
```

```
db.patients.find({ name: "Ragini" }).pretty()
```

Delete

```
const patient = db.patients.findOne({ _id: ObjectId("6841c371bc5760b14d78e998") });
```

```
printjson(patient);

db.patients.deleteOne({ _id: ObjectId("6841c371bc5760b14d78e998") });
```

Optimize search

1. Create Indexes on Common Search Fields
For example, to speed up searches on patient name,
medicalHistory.chronicConditions, and medicalHistory.allergies, create indexes like:

```
db.patients.createIndex({ name: 1 })
db.patients.createIndex({ "medicalHistory.chronicConditions": 1 })

db.patients.createIndex({
  name: "text",
  "medicalHistory.chronicConditions": "text",
  "medicalHistory.allergies": "text"
})
```

Use .explain("executionStats") to Analyze Query Performance
To check how MongoDB executes your query and how efficient it is:

```
const query = {
  $text: { $search: "asthma rash" }
}

db.patients.find(query).explain("executionStats")
```

Look for:

executionStats.totalKeysExamined (should be low)

executionStats.totalDocsExamined (should be low)

winningPlan (shows what index is used)

stage — want "IXSCAN" for index scan, not "COLLSCAN" (collection scan)

Aggregation and Analytics

Number of Patients Per Condition

```
db.patients.aggregate([
 { $unwind: "$medicalHistory.chronicConditions" },
 { $group: {
    _id: "$medicalHistory.chronicConditions",
    patientCount: { $sum: 1 }
 }},
 { $sort: { patientCount: -1 } }
])
```

Most Prescribed Medications

```
db.patients.aggregate([
 { $unwind: "$medications" },
 { $group: {
    _id: "$medications.name",
    count: { $sum: 1 }
 }},
 { $sort: { count: -1 } },
 { $limit: 10 }
])
```

Average Patient Age Per Department

```
db.patients.aggregate([
 {
  $addFields: {
   birthDate: {
    $dateFromString: {
     dateString: "$dateOfBirth",
     format: "%d/%m/%Y"
    }
   }
  }
 },
 {
  $addFields: {
   age: {
    $dateDiff: {
     startDate: "$birthDate",
     endDate: "$$NOW",
     unit: "year"
    }
```

```
    }
   }
  },
  {
   $group: {
    _id: "$primaryCarePhysician.department",  // Replace with actual dept field
    averageAge: { $avg: "$age" }
   }
  }
])
```

Frequency of Visits Per Month

```
db.appointments.aggregate([
 {
  $group: {
   _id: {
    year: { $year: { $toDate: "$visitDate" } },
    month: { $month: { $toDate: "$visitDate" } }
   },
   visitCount: { $sum: 1 }
  }
 },
 {
  $sort: { "_id.year": 1, "_id.month": 1 }
 }
])
```

References

- HEMANGANI GitHub Base Repo
- MongoDB Mongoose Documentation
- React Query Docs
- DaisyUI & Tailwind CSS Docs