🩺 Project Title: Healthcare Patient Management System Using MongoDB

📝 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.

The system should support operations like adding, searching, updating, and deleting patient records, with optimized queries for performance and aggregation features for clinical and administrative insights. Students are free to use any frontend (e.g., React, Vue, Angular) and backend (e.g., Node.js, Python, Java) technology stacks, but **MongoDB is mandatory** as the database.

🎯 Objectives:

- Emphasize practical use of MongoDB for managing complex and variable healthcare data.
- Allow flexibility in frontend and backend technology choices.
- Provide hands-on experience with NoSQL data modeling and query optimization.
- Enable generation of medical insights and reports using MongoDB's aggregation framework.

📦 Core Modules:

Insert New Patient Record

- Add a new patient entry to the system.
- Fields may include: patient ID, name, age, gender, contact info, allergies, medical history, current prescriptions, and doctor notes.
- Data stored in a `patients` collection in MongoDB.

Search Patient Record

- Search by patient ID, name, condition, or visit date.
- Use MongoDB query operators like `$regex`, `$in`, `$or`, and text indexes.
- Implement pagination and sorting for efficient results display.

Update Patient Record

- Modify existing patient data, such as new diagnoses, updated prescriptions, or added lab reports.
- Use MongoDB operators like `$set`, `$push`, `$pull`, and `$addToSet` to update embedded arrays and nested fields.

Delete Patient Record

- Remove a patient record securely using a unique identifier (e.g., patient ID).
- Include verification prompts and logs for compliance and auditing.

Optimize Search

- Utilize MongoDB indexing on commonly searched fields (e.g., patient name, diagnosis).
- Monitor query performance using `.explain()` and refactor queries for speed.

Aggregation and Analytics

- Use MongoDB Aggregation Framework to generate healthcare insights:
    - Number of patients per condition
    - Most prescribed medications
    - Average patient age per department
    - Frequency of visits per month
- Optionally visualize these analytics using frontend libraries (e.g., Chart.js, D3.js).

Authentication Module (Optional)

- Implement admin or healthcare staff login.
- Role-based access control (e.g., doctor, nurse, receptionist).
- Use JWT tokens or session-based authentication.

Data Validation & Error Handling

- Validate data entry for formats (e.g., phone numbers, email, date of birth).
- Provide clear error messages for invalid or incomplete forms.

🛠️ Suggested Technology Stack (Flexible):

- **Frontend:** React, Vue, Angular, Bootstrap, HTML/CSS/JS
- **Backend:** Node.js (Express), Python (Flask/Django), Java (Spring Boot), Go, etc.
- **Database:** MongoDB (Required)

💼 Project Title: Job Portal System Using MongoDB

📝 Problem Statement:

Modern job portals must manage a large volume of diverse data: job listings, company details, candidate profiles, resumes, applications, and communications. This data is often semi-structured—resumes vary in format, job requirements differ by role, and application workflows are dynamic—making traditional relational databases inflexible and harder to scale.

This project aims to design and implement a **Job Portal System** using **MongoDB** as the core database. As a document-oriented NoSQL database, MongoDB is ideal for handling unstructured and variable data, enabling quick searches, flexible schemas, and scalable performance.

The system should support the core operations of posting and managing job listings, registering candidates and employers, searching for jobs or candidates, and tracking applications. Students are free to choose any frontend (e.g., React, Angular, Vue) and backend (e.g., Node.js, Python, Java) technologies, but **MongoDB must be used for the database layer**.

🎯 Objectives:

- Demonstrate efficient use of MongoDB for modeling real-world job and resume data.
- Allow flexibility in technology stack for frontend and backend components.
- Provide experience with advanced querying, indexing, and aggregation in MongoDB.
- Build a functional system for matching candidates with job opportunities using NoSQL data strategies.

📦 Core Modules:

Insert New Job Post / Resume

- Allow employers to create job listings with fields like job title, company, location, description, requirements, and salary.
- Allow job seekers to create profiles with resume details, skills, education, and work history.
- Store job posts in a `jobs` collection and resumes in a `candidates` or `resumes` collection.

Search Jobs / Candidates

- Enable users to search job posts by title, location, skills, salary range, or company.
- Enable recruiters to search candidate profiles by skills, experience, or keywords.
- Use advanced MongoDB queries with `$regex`, `$text`, `$or`, and filters.

- Support pagination and sorting for large result sets.

Update Job Post / Profile

- Employers can edit job descriptions, update application deadlines, or close listings.
- Candidates can update their resumes, add new skills, or modify experience.
- Use MongoDB’s `$set` and `$push` operators for efficient partial updates.

Delete Job Post / Resume

- Allow users to delete job listings or candidate profiles securely.
- Ensure confirmation prompts and logical soft deletes if needed.

Optimize Search

- Use **compound indexes** and **text indexes** on frequently searched fields like `jobTitle`, `skills`, and `location`.
- Analyze query execution using `.explain()` and optimize for performance.

Aggregation and Analytics

- Use MongoDB Aggregation Framework to generate insights such as:
    - Number of jobs posted per industry
    - Most in-demand skills
    - Average salary range by job role

    - Application trends over time

- Display analytics using charts or graphs in the frontend.

Authentication Module (Optional)

- Separate login for **Employers** and **Candidates**.
- Use JWT or session-based authentication.
- Role-based access for managing posts or applying to jobs.

Application Tracking

- Allow candidates to apply for jobs.
- Enable employers to view, filter, and track applications.
- Store application data in a dedicated `applications` collection linking job and candidate IDs.

Data Validation & Error Handling

- Validate entries like email, phone number, required fields, and resume formats.
- Handle errors gracefully and provide clear feedback to users.

🛠️ Suggested Technology Stack (Flexible):

- **Frontend:** React, Vue.js, Angular, Tailwind, Bootstrap

- **Backend:** Node.js (Express), Python (Flask/Django), Java (Spring Boot)
- **Database:** MongoDB (Required)

🍽️ Project Title: Restaurant Menu & Order Management System Using MongoDB

📝 Problem Statement:

Restaurants today need digital systems to manage dynamic menus, customer orders, dietary preferences, customizations, feedback, and order tracking. Traditional relational databases often struggle with evolving menu structures and nested, semi-structured order data (e.g., multiple items, modifiers, custom notes).

This project focuses on building a **Restaurant Menu & Order Management System** using **MongoDB** as the primary database. MongoDB’s flexible, document-based structure is ideal for storing complex menu items, orders with nested line items, and customer data. It enables seamless scalability, rapid search, real-time updates, and detailed analytics.

The system should allow restaurant staff to manage menu items, process orders, track order statuses, and generate reports. Students can choose any frontend (e.g., React, Angular, Vue) and backend (e.g., Node.js, Python, Java) stack—but **MongoDB is mandatory** for the database layer.

🎯 Objectives:

- Highlight the use of MongoDB for managing flexible, nested, and dynamic restaurant data.
- Provide students the freedom to implement their preferred frontend/backend stack.
- Offer practical experience in building real-world NoSQL data models.
- Emphasize query optimization and use of MongoDB's aggregation and indexing features.

📦 Core Modules:

Insert New Menu Item or Order

- Add new items to the menu with fields like name, category, price, ingredients, tags (e.g., vegan, spicy), and availability.
- Insert new customer orders with multiple items, quantities, customization (e.g., extra cheese), and notes.
- Menu items stored in a `menu` collection; orders stored in an `orders` collection.

Search Menu

- Search and filter menu items by name, category, dietary tags, ingredients, or price range.

- Use MongoDB queries with `$regex`, `$in`, `$and`, and compound filters.
- Allow dynamic menu browsing with pagination and sorting.

Update Menu Item / Order

- Update menu item details like pricing, availability, or category.
- Modify in-progress orders (e.g., adding/removing items, changing status).
- Use MongoDB's `$set`, `$push`, and `$pull` operators for efficient updates.

Delete Menu Item / Order

- Remove outdated menu items or cancel orders.
- Support soft delete or archival for analytics purposes.
- Ensure confirmation and role-based deletion control.

Optimize Search

- Implement indexing on fields like `name`, `category`, and `tags` in the `menu` collection.
- Optimize order history lookup using indexes on `customerId`, `timestamp`, or `status`.

Aggregation and Analytics

- Use MongoDB's aggregation framework for:
    - Sales reports (daily, weekly, item-wise)
    - Most ordered dishes

    - Category-wise revenue breakdown
    - Peak ordering hours

- Present analytics using frontend libraries (e.g., Chart.js, D3.js).

Authentication Module (Optional)

- Admin login to manage menu and staff.
- Staff login for order handling.
- Use JWT or session-based authentication with role-based access.

Order Status & Tracking

- Track each order's status: *Placed*, *In Preparation*, *Ready*, *Delivered*.
- Display order progress in real time.
- Use MongoDB to update statuses and store timestamps.

Data Validation & Error Handling

- Validate inputs like price formats, required fields, and allowed tags.
- Handle errors gracefully with informative feedback to staff or users.

🛠️ Suggested Technology Stack (Flexible):

- **Frontend:** React, Vue, Angular, Tailwind CSS, Bootstrap

- **Backend:** Node.js (Express), Python (Flask/Django), Java (Spring Boot)
- **Database:** MongoDB (Required)
