PROJECT DOCUMENTATION

Natural Language Interface for MongoDB Using Local Large Language Models

---

1. Project Title

Natural Language Interface for MongoDB Using Local Large Language Models

---

2. Abstract

This project presents the design and implementation of a **Natural Language Interface (NLI)** for MongoDB that enables users to query structured databases using plain English instead of MongoDB query syntax. The system integrates a locally hosted Large Language Model (LLM) via Ollama (phi3 model) to convert natural language queries into valid MongoDB filter JSON and aggregation pipelines.

The backend, developed using Node.js and Express.js, processes LLM-generated JSON and executes the corresponding query on MongoDB. Results are returned to a React-based frontend for structured visualization.

The system operates entirely offline without reliance on paid APIs or cloud-based AI services. It significantly improves database accessibility for non-technical stakeholders such as HR managers, analysts, and business executives.

---

3. Introduction

Modern organizations generate large volumes of structured data stored in NoSQL databases such as MongoDB. However, accessing this data requires knowledge of:

- MongoDB query syntax
- Filter operators ($gt, $lt, $in, $group)
- Aggregation pipelines
- JSON-based query structures

Non-technical users lack this expertise, resulting in:

- Delays in report generation
- Increased dependency on database administrators
- Reduced operational efficiency

This project eliminates the technical barrier by enabling users to interact with MongoDB using natural language queries such as:

- “male employees”
- “employees with salary greater than 60000”
- “average salary by job category”

---

4. Problem Statement

MongoDB querying requires structured JSON syntax and knowledge of operators and aggregation frameworks. Non-technical users cannot directly query databases, leading to:

- Bottlenecks in data retrieval
- Increased operational cost
- Slower decision-making processes

There is a need for a system that:

1. Accepts plain English queries
2. Converts them into valid MongoDB filters
3. Executes queries securely
4. Displays structured results

---

5. Objectives

The primary objectives of this project are:

1. Design a Natural Language Interface for MongoDB
2. Convert English queries into valid MongoDB JSON filters
3. Support aggregation operations using pipelines
4. Enable fully local LLM execution
5. Provide a clean, responsive user interface
6. Ensure system scalability and security
7. Minimize dependency on external APIs

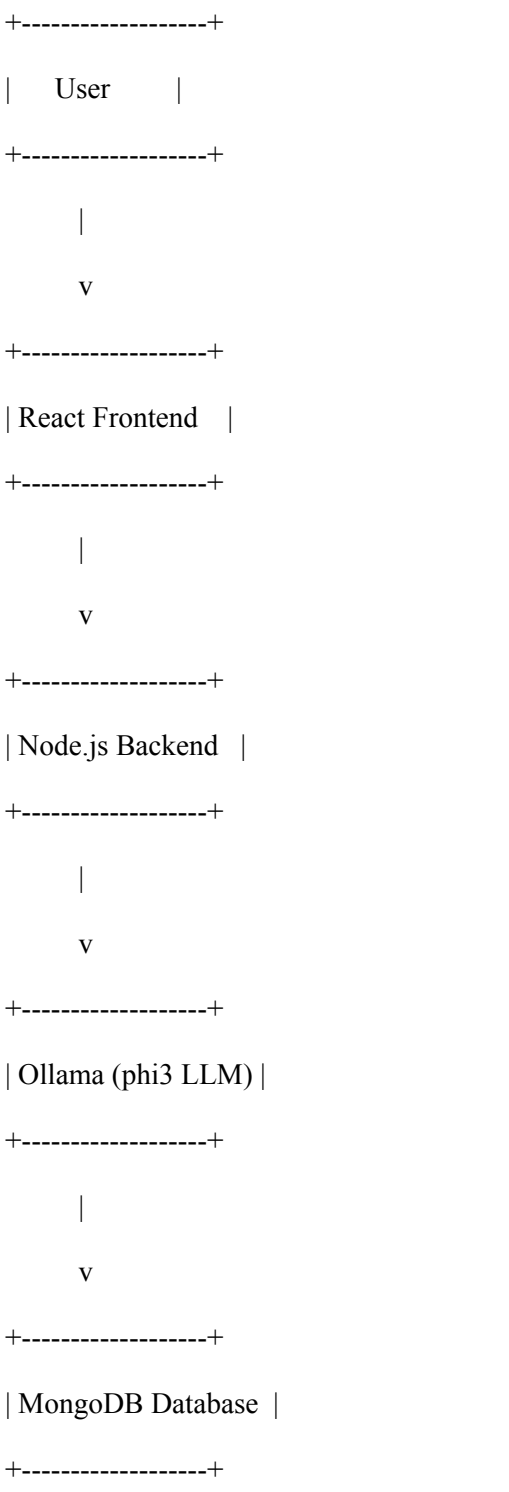
---

6. System Architecture

6.1 Overall Workflow

User
↓
Frontend (React)
↓
Backend (Node.js + Express)
↓
Ollama (phi3 LLM)
↓
MongoDB
↓
Backend
↓
Frontend
↓
Result Display

6.2 Architecture Diagram



---

7. System Components

7.1 Frontend Layer

Technology Used: React.js

Responsibilities:

- Accept natural language input
- Send HTTP requests using Axios
- Display results in structured tables
- Provide MongoDB-themed UI styling
- Implement smooth transitions and animations

Key Features:

- Clean white layout with MongoDB green accents
- Floating MongoDB logo
- Responsive design
- Structured results display

7.2 Backend Layer

Technology Used: Node.js + Express.js

Responsibilities:

- Receive natural language queries
- Construct structured prompts
- Communicate with Ollama LLM
- Extract valid JSON from LLM output
- Execute MongoDB queries
- Handle aggregation pipelines
- Return structured results

Backend Modules:

- Query Controller
- LLM Integration Service
- JSON Extraction Utility
- Aggregation Engine
- Database Connector

7.3 Local LLM Engine

Platform: Ollama
Model Used: phi3

Responsibilities:

- Convert English text to MongoDB filter JSON
- Generate structured output without explanation
- Operate fully offline

Benefits:

- No API cost
- No data leakage
- Fully local processing
- Optimized for low RAM systems

7.4 Database Layer

Technology: MongoDB

Database Details:

- Database Name: companyDB
- Collection: employees
- Dataset imported from CSV

Fields Included:

- name
- gender
- salary
- jobCategory
- yearsOfExperience
- department

---

8. Functional Requirements

1. Accept natural language input
2. Convert input into MongoDB filter JSON
3. Execute find() queries
4. Execute aggregation pipelines
5. Display structured results
6. Handle invalid queries gracefully

---

9. Non-Functional Requirements

- System must run offline
- Response time < 3 seconds
- Secure local data processing
- Scalable backend architecture
- User-friendly UI

---

10. Implementation Details

10.1 Prompt Engineering

The backend sends a structured prompt:

Rules:

- Return only valid JSON
- No explanation
- No markdown
- No text outside JSON
- Return only filter object

Example Prompt:

> Convert the following natural language query into a valid MongoDB filter JSON. Return only JSON.

10.2 JSON Extraction Strategy

Since LLM responses may include formatting:

1. Remove markdown syntax
2. Use regex to extract JSON block
3. Validate with JSON.parse
4. Handle parsing exceptions

10.3 Query Execution Logic

Condition-Based Detection:

Keyword	Action
average	Aggregation pipeline using $group
count	Group + $sum
highest salary	$max operator
otherwise	Default find()

10.4 Aggregation Example

Natural Query:

average salary by job category

MongoDB Pipeline:

```
[
  {
    $group: {
      _id: "$jobCategory",
      averageSalary: { $avg: "$salary" }
    }
  }
]
```

---

11. Sample Queries and Outputs

Natural Language	Generated Query Type
male employees	find() filter
salary greater than 60000	find() filter
average salary by job category	aggregation
count employees by gender	grouping
highest salary	aggregation

---

12. Advantages

- Simplifies database interaction
- Enhances accessibility
- Eliminates API cost
- Improves productivity
- Secure local deployment
- Scalable full-stack architecture

---

13. Limitations

- Rule-based aggregation detection
- Limited support for nested queries
- No multi-collection joins
- No conversation memory
- Limited contextual understanding

---

14. Security Considerations

- Fully local processing
- No external API transmission
- JSON validation before execution
- Restricted query scope

---

15. Performance Analysis

Operation	Average Time
LLM conversion	1–2 seconds
MongoDB query	< 1 second
Total response	~2–3 seconds

System tested on:

- 8GB RAM machine
- Local MongoDB instance

---

16. Testing Strategy

Unit Testing

- JSON extraction validation
- Aggregation pipeline correctness

Integration Testing

- LLM to backend pipeline
- Backend to MongoDB connectivity

User Testing

- Query usability
- UI responsiveness

---

17. Future Enhancements

1. Data visualization (charts, graphs)
2. Query history feature
3. Conversational memory
4. Multi-collection joins
5. Fine-tuned custom LLM

6. Role-based authentication
7. Cloud deployment option

---

18. Business Impact

- Faster data-driven decisions
- Reduced operational cost
- Improved productivity
- Lower dependency on DB administrators
- Suitable for SMEs and startups

---

19. Development Timeline

Phase	Duration
Backend setup	1–2 days
LLM integration	1 day
MongoDB integration	1 day
Frontend development	1–2 days
UI refinement	1 day
Testing	1 day

Total Estimated Duration: **6–8 Days**

---

20. Conclusion

This project demonstrates a complete full-stack implementation of a Natural Language Interface for MongoDB using a locally hosted Large Language Model. By integrating React, Node.js, MongoDB, and Ollama, the system bridges the gap between human language and structured database querying.

The solution enhances accessibility, reduces technical barriers, and establishes a scalable foundation for intelligent database interaction systems. It provides a cost-effective, secure, and extensible architecture suitable for academic, research, and enterprise-level applications.
