Software Requirements Specification (SRS) for MongoDB Query Generator

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) defines the functional and non-functional requirements for the MongoDB Query Generator, referred to as "MongoDB Copilot." The tool enables authenticated users to interact with any MongoDB database using natural language queries, which are translated into executable MongoDB queries. It simplifies database interactions while ensuring security, flexibility, and robust error handling.

1.2 Scope

MongoDB Copilot is a web-based application that allows authenticated users to:

- Connect to any MongoDB database using a connection URL.
- Input natural language queries for read, write, update, and delete operations.
- Preview, confirm, and execute generated queries with explanations.
- Reverse mutation operations via mandatory undo functionality.
- Whitelist the application's static IP for database access.
- Access the system securely via OAuth authentication with session management.
- Operate within per-user rate limits to ensure fair usage.

The tool targets developers, database administrators, and non-technical users, offering a secure, user-friendly alternative to manual query writing. It supports MongoDB deployments including Atlas, self-hosted, Docker, and on-premises instances.

1.3 Definitions, Acronyms, and Abbreviations

- **MongoDB**: A document-oriented NoSQL database.
- **Natural Language Query**: A query in plain English or other human languages.
- **LLM**: Large Language Model, used for query generation and explanation.
- **SRS**: Software Requirements Specification.
- **MongoDB Copilot**: The MongoDB Query Generator tool.

- **Mutation**: Operations that modify data (`insert` , `update` , `delete`).
- **Static IP Proxy**: A server with a fixed IP address to route MongoDB connections.
- **OAuth**: Open Authorization protocol for secure authentication.
- **JWT**: JSON Web Token for session management.

1.4 References

- MongoDB Documentation: MongoDB Official Site
- React Documentation: React Docs
- Express Documentation: Express Docs
- OAuth 2.0: OAuth Official Site
- IEEE Standard for Software Requirements Specifications (IEEE Std 830-1998)

1.5 Overview

This SRS is organized as follows:

- **Introduction**: Outlines the document's purpose, scope, and structure.
- **Overall Description**: Describes the product's perspective, functions, users, and constraints.
- **Specific Requirements**: Details functional and non-functional requirements.
- **Verification and Validation**: Specifies how requirements will be verified.
- **Appendices**: Includes glossary and index.

2. Overall Description

2.1 Product Perspective

MongoDB Copilot is a standalone web application that integrates with MongoDB databases to enable natural language querying for authenticated users. It leverages Large Language Models (LLMs) to interpret user inputs, generate MongoDB queries, and provide plain English explanations. Unlike MongoDB Compass, which is desktop-based and lacks robust mutation safety, MongoDB Copilot offers web accessibility, OAuth authentication, rate limiting, mandatory undo functionality, and IP whitelisting guidance, making it suitable for diverse deployment scenarios.

2.2 Product Functions

The primary functions include:

- **Authentication**: Secure user login via OAuth and session management with JWT.
- **Rate Limiting**: Enforce per-user query limits to prevent abuse.

- **Connection Management**: Connect to MongoDB databases via user-provided URLs through a static IP proxy.
- **Schema Introspection**: Dynamically extract database schema.
- **Query Generation**: Translate natural language into MongoDB queries.
- **Query Explanation**: Provide plain English explanations for generated queries.
- **Query Preview**: Display queries and their impact before execution.
- **Execution Confirmation**: Require user approval for all queries, with emphasis on mutations.
- **Undo Functionality**: Enable reversal of mutation operations.
- **IP Whitelisting Guidance**: Provide instructions for users to allow the proxy's static IP.
- **Result Display**: Present query results clearly.

2.3 User Classes and Characteristics

User Class	Description	Needs
Developers	Skilled in programming and database management	Secure authentication, automated query generation, undo capabilities, query explanations
Database Administrators	Manage database operations	Simplified query creation, safety, IP whitelisting instructions, rate limits
Non-Technical Users	Limited technical expertise	Intuitive interface, minimal learning curve, guided setup, clear explanations

2.4 Operating Environment

- **Frontend**: Web-based, built with React, accessible via browsers (Chrome, Firefox, Safari).
- **Backend**: Node.js with Express, hosted on Vercel serverless with a static IP proxy (e.g., Fly.io or AWS EC2).
- **Database**: Any MongoDB instance (Atlas, self-hosted, Docker, on-prem).
- **LLM**: Open-source (e.g., Mistral-8x7b-Instruct via HuggingFace Inference API) or hosted (e.g., OpenAI GPT-4o).
- **Authentication**: OAuth 2.0 providers (e.g., Google, GitHub) with JWT-based session management.

2.5 Design and Implementation Constraints

- Use React for the frontend and Node.js/Express for the backend.
- Use the `mongodb` Node.js driver for database interactions.
- Implement OAuth 2.0 for authentication and JWT for session management.
- Enforce rate limits using Express middleware (e.g., `express-rate-limit`).
- Support both `mongodb://` and `mongodb+srv://` connection strings.
- Use a static IP proxy to address Vercel's dynamic IP limitations.
- Ensure secure handling of user-provided database credentials.

- Implement mandatory undo functionality for all mutations.
- Provide clear IP whitelisting instructions compatible with MongoDB Atlas and self-hosted setups.
- Include query explanations generated by the LLM.

2.6 Assumptions and Dependencies

- Users have valid MongoDB connection URLs and access permissions.
- MongoDB instances are accessible via the proxy's static IP or within the user's network.
- OAuth providers (e.g., Google, GitHub) are available for authentication.
- LLM services require internet connectivity unless using a local model.
- Users are responsible for managing database security and IP whitelisting.
- The `mongodb` Node.js driver supports all targeted MongoDB versions.

3. Specific Requirements

3.1 External Interface Requirements

- **User Interfaces**: Web interface built with React, styled with Tailwind CSS, including:
  - OAuth login button for authentication.
  - Forms for MongoDB URL and query input.
  - Query preview, explanation, and confirmation components.
  - IP whitelisting instructions and rate limit status display.
- **Hardware Interfaces**: Standard web browsers on desktop/mobile devices.
- **Software Interfaces**:
  - MongoDB driver: `mongodb` for Node.js.
  - LLM APIs: Mistral via HuggingFace Inference API or OpenAI GPT-4o.
  - Proxy server: Node.js/Express on Fly.io or AWS EC2.
  - Authentication: OAuth 2.0 libraries (e.g., `passport` for Node.js).
  - Rate limiting: `express-rate-limit` middleware.
- **Communication Interfaces**: HTTPS for web requests, MongoDB wire protocol for database communication, OAuth 2.0 for authentication.

3.2 Functional Requirements

3.2.1 Authentication and Session Management

- **FR1**: Implement OAuth 2.0 authentication supporting providers like Google and GitHub.
- **FR2**: Generate and store JWTs for session management upon successful login.
- **FR3**: Require authentication for all API endpoints except public routes (e.g., login page).
- **FR4**: Provide a logout feature to invalidate JWT sessions.

- **FR5**: Store user sessions in a MongoDB collection with expiration (e.g., 24 hours).

3.2.2 Rate Limiting

- **FR6**: Enforce per-user rate limits on query generation and execution (e.g., 100 queries/hour).
- **FR7**: Store rate limit data in a MongoDB collection, tied to authenticated user IDs.
- **FR8**: Display remaining quota in the React UI (e.g., "50 queries left this hour").
- **FR9**: Return a 429 status code with a user-friendly message when limits are exceeded.

3.2.3 Connection Management

- **FR10**: Allow authenticated users to input a MongoDB connection URL (`mongodb://` or `mongodb+srv://`) via a React form.
- **FR11**: Validate URL format and connectivity using the `mongodb` Node.js driver.
- **FR12**: Route connections through a static IP proxy (e.g., `54.212.123.45`).
- **FR13**: Display error messages for connection failures with IP whitelisting instructions.

3.2.4 Schema Introspection

- **FR14**: Extract schema information (collections, field names, types) upon successful connection using `listCollections()` and `findOne()` .
- **FR15**: Handle databases with no collections or documents gracefully, prompting users to check their database.
- **FR16**: Cache schema temporarily in memory for performance during a session.

3.2.5 Query Generation

- **FR17**: Accept natural language queries via a React text input field.
- **FR18**: Use an LLM to generate MongoDB queries based on the introspected schema and user input, making API calls from the Express backend.
- **FR19**: Support operations:
  - Read: `find` , `aggregate` .
  - Write: `insertOne` , `insertMany` .
  - Update: `updateOne` , `updateMany` .
  - Delete: `deleteOne` , `deleteMany` .

3.2.6 Query Explanation

- **FR20**: Generate a plain English explanation for each generated query using the LLM.
- **FR21**: Display the explanation in the React UI alongside the query (e.g., "This query finds all users from Hyderabad with age greater than 25").
- **FR22**: Allow users to toggle the explanation visibility to reduce clutter.

3.2.7 Query Preview and Confirmation

- **FR23**: Display generated queries in a readable JSON or MongoDB shell format within a React component.
- **FR24**: For all queries, provide:
  - Preview of affected documents (e.g., `find(filter).limit(5)`).
  - Estimated impact (e.g., `countDocuments(filter)` for number of documents).
- **FR25**: Require explicit user confirmation via "Execute" and "Cancel" buttons in the React UI for all queries.
- **FR26**: For mutation queries, highlight potential risks (e.g., "This will delete 1,000 documents") using styled alerts.

3.2.8 Query Execution

- **FR27**: Execute confirmed queries on the connected database via the proxy using the `mongodb` driver.
- **FR28**: Handle execution errors with user-friendly messages rendered in the React UI.
- **FR29**: Display read results in tabular or JSON format using React components (e.g., a data grid).
- **FR30**: Show mutation outcomes (e.g., `modifiedCount`, `deletedCount`) in a summary component.

3.2.9 Safety and Validation

- **FR31**: Validate query syntax to ensure it is valid MongoDB JSON using a Node.js parser.
- **FR32**: Prevent destructive queries (e.g., `deleteMany({})`) unless explicitly approved.
- **FR33**: **Mandatory Undo Functionality**:
  - For `insert`: Store inserted document IDs in a session cache and allow deletion via a React "Undo" button.
  - For `update`: Save original documents in a temporary MongoDB collection before applying changes, restorable via "Undo."
  - For `delete`: Move documents to a backup collection (soft delete) before permanent removal, restorable via "Undo."
  - Provide a UI option to "Undo" within a configurable time window (e.g., 5 minutes) in the React interface.
- **FR34**: Log all mutation operations (query, user, timestamp) in a MongoDB collection for auditing.

3.2.10 User Guidance for IP Whitelisting

- **FR35**: Provide a dedicated React component with step-by-step instructions for whitelisting the proxy's static IP.
- **FR36**: Include instructions for:
  - MongoDB Atlas IP whitelisting.

    - Self-hosted MongoDB firewall configuration.
- **FR37**: Display the proxy's static IP (e.g., `54.212.123.45`) prominently in the React UI and error messages.
- **FR38**: Offer a "Test Connection" button in the React UI to verify IP whitelisting before query generation.

3.3 Non-Functional Requirements

Requirement	Description
NFR1 Performance	Query generation and explanation within 5 seconds, execution within 10 seconds.
NFR2 Scalability	Handle 100 concurrent authenticated users without degradation.
NFR3 Security	Encrypt communications (HTTPS), secure OAuth/JWT handling, no storage of MongoDB URLs.
NFR4 Usability	Intuitive React interface, clear IP whitelisting guidance, accessible query explanations.
NFR5 Maintainability	Modular Node.js/Express codebase with comprehensive documentation.
NFR6 Reliability	99.9% uptime for proxy server, robust error handling in React and Express.
NFR7 Compliance	Adhere to OAuth 2.0 standards and data privacy regulations (e.g., GDPR for user data).

3.4 System Features

Feature	Description	Rationale
OAuth Authentication	Secure user login via Google/GitHub OAuth with JWT sessions	Ensures authorized access
Rate Limiting	Enforce per-user query limits (e.g., 100/hour)	Prevents abuse, ensures fair usage
Natural Language Querying	Translate plain English into MongoDB queries	Simplifies database interaction
Query Explanation	Provide plain English query explanations	Enhances user understanding
Schema-Agnostic Operation	Adapt to any MongoDB schema dynamically	Enhances flexibility
Mandatory User Confirmation	Require approval for all queries	Prevents accidental data changes

Feature	Description	Rationale
Query Preview and Impact Estimation	Show query and impact before execution	Builds trust and ensures correctness
Mandatory Undo Functionality	Allow reversal of mutations	Mitigates errors and data loss
IP Whitelisting Guidance	Provide clear instructions for IP access	Ensures connectivity for users

3.5 User Guidance for IP Whitelisting

To ensure MongoDB Copilot can connect to the user's database, the application provides detailed instructions for whitelisting the proxy's static IP address (`54.212.123.45`). These instructions are displayed in a dedicated React component and included in connection error messages.

3.5.1 MongoDB Atlas IP Whitelisting

1. **Log in to MongoDB Atlas**:
   - Navigate to MongoDB Atlas and sign in.
2. **Access Network Access Settings**:
   - In the left sidebar, click "Network Access" under the "Security" section.
3. **Add IP Address**:
   - Click the "Add IP Address" button.
   - Enter the proxy's static IP: `54.212.123.45` .
   - Optionally, add a comment (e.g., "MongoDB Copilot Proxy").
   - Set an expiration (e.g., permanent or temporary for testing).
4. **Save Changes**:
   - Click "Confirm" to save the IP whitelist entry.
5. **Verify Connectivity**:
   - Use the "Test Connection" button in MongoDB Copilot's React UI to confirm access.

3.5.2 Self-Hosted MongoDB Firewall Configuration

1. **Identify Firewall Settings**:
   - Access the server hosting MongoDB (e.g., AWS EC2, DigitalOcean, on-prem).
   - Locate the firewall configuration (e.g., `ufw` , `iptables` , or cloud provider security groups).
2. **Allow Inbound Traffic**:
   - Add a rule to allow inbound traffic on the MongoDB port (default: `27017`) from `54.212.123.45` .
   - Example for `ufw` :

```
sudo ufw allow from 54.212.123.45 to any port 27017
```

    - Example for AWS Security Group:
        - Add an inbound rule: Type "Custom TCP," Port `27017`, Source `54.212.123.45/32`.
3. **Restart Firewall** (if required):
    - Apply changes (e.g., `sudo ufw reload`).
4. **Test Connection**:
    - Use MongoDB Copilot's "Test Connection" button to verify access.

3.5.3 Error Handling

- If connection fails due to IP restrictions, display in the React UI:
    - Error message: "Connection failed. Please whitelist our proxy IP: 54.212.123.45."
    - Link to detailed whitelisting instructions.
- Provide a "Copy IP" button for easy access to `54.212.123.45`.

3.6 Verification and Validation

- **VR1**: Verify functional requirements through unit, integration, and end-to-end tests using Jest (React) and Mocha/Chai (Express).
- **VR2**: Validate OAuth authentication by testing login/logout with Google and GitHub providers.
- **VR3**: Test rate limiting by simulating user queries and verifying 429 responses.
- **VR4**: Validate undo functionality by simulating mutations and confirming reversals in test MongoDB instances.
- **VR5**: Verify query explanation feature by comparing LLM-generated explanations to expected outputs.
- **VR6**: Test IP whitelisting instructions with MongoDB Atlas and self-hosted setups.
- **VR7**: Conduct performance, security, and usability tests for non-functional requirements.
- **VR8**: Perform user acceptance testing (UAT) to ensure user needs are met.

4. Appendices

4.1 Glossary

- **MongoDB**: A document-oriented NoSQL database.
- **Natural Language Processing (NLP)**: Technology for understanding human language.
- **Large Language Model (LLM)**: AI model for generating text.
- **Mutation**: Database operations that modify data (`insert`, `update`, `delete`).
- **Static IP Proxy**: A server with a fixed IP to route MongoDB connections.

- **Undo Functionality**: Ability to reverse mutation operations.
- **OAuth**: Protocol for secure user authentication.
- **JWT**: JSON Web Token for session management.

4.2 Index