**Software Design**

**Document**

**for**

Behavioral Cognition Project

**Version 2.0 approved**

**Prepared by Deep Bhakta, Yizhang Cao, Iris Ha, Erick Agustin Herrera, Shant Hovagimian, Daniel Ontiveros, Jorge Pena, Arturo Rodriguez, David Santini, Izeth Torres**

Behavioral Cognition

**September 22, 2023**

**Table of Contents**

Table of Contents................................................................................................................... <pg 2>

Revision History..................................................................................................................... <pg 4>

1. Introduction................................................................................................................ <pg 5>
   1. Purpose........................................................................................................... <pg 5>
   2. Document Conventions……………………................................................. <pg 5>
   3. Intended Audience and Reading Suggestions................................................ <pg 5>
   4. System Overview........................................................................................... <pg 5>
2. Design Considerations............................................................................................... <pg 6>
   1. Assumptions and dependencies..................................................................... <pg 6>
   2. General Constraints........................................................................................<pg 6>
   3. Goals and Guidelines.....................................................................................<pg 6>
   4. Development Methods................................................................................... <pg 7>
3. Architectural Strategies.............................................................................................. <pg 7>
4. System Architecture................................................................................................... <pg 7>
5. Policies and Tactics.................................................................................................... <pg 8>
   1. Specific Products Used.................................................................................. <pg 8>
   2. Requirements traceability............................................................................... <pg 8>
   3. Testing the software....................................................................................... <pg 9>
6. Detailed System Design............................................................................................. <pg 9>

6.x Name of Module........................................................................................... <pg 9>

6.x.1 Responsibilities...................................................................................... <pg 9>

6.x.2 Constraints............................................................................................. <pg 9>

6.x.3 Composition........................................................................................... <pg 9>

6.x.4 Uses/Interactions.................................................................................... <pg 9>

6.x.5 Resources............................................................................................... <pg 9>

6.x.6 Interface/Exports.................................................................................... <pg 9>

1. Detailed Lower level Component Design

7.x Name of Class or File......................................................................... <pg 10>

7.x.1 Classification.................................................................................... <pg 10>

7.x.2 Processing Narrative(PSPEC).......................................................... <pg 10>

7.x.3 Interface Description........................................................................ <pg 10>

7.x.4 Processing Detail.............................................................................. <pg 10>

7.x.4.1 Design Class Hierarchy................................................................... <pg 10>

7.x.4.2 Restrictions/Limitations.................................................................. <pg 10>

7.x.4.3 Performance Issues........................................................................ <pg 10>

7.x.4.4 Design Constraints......................................................................... <pg 10>

7.x.4.5 Processing Detail For Each Operation........................................... <pg 10>

1. User Interface
   1. Overview of User Interface.............................................................. <pg 11>
   2. Screen Frameworks or Images......................................................... <pg 11>
   3. User Interface Flow Model............................................................... <pg 11>
2. Database Design

9.1 Overview of User Interface………………………………………….…… <pg 11>

9.2 Screen Frameworks or Images…………………………………………… <pg 11>

9.3 User Interface Flow Model………………………………………………. <pg 12>

1. Requirements Validation and Verification..................................................... <pg 12>
2. Glossary.......................................................................................................... <pg 13>
3. References...................................................................................................... <pg 14>

**Revision History**

| Name | Date | Reason For Changes | Version |
| --- | --- | --- | --- |
| Iris Ha | 12/5/23 | Starting Documentation | 1.0 |
| Yizhang Cao | 5/1/2024 | Revising Documentation | 2.0 |
|  |  |  |  |
|  |  |  |  |

**1. Introduction**

**1.1 Purpose**

Software design document is for lp-toolkit version 1.0. Document will go over the thought process of developing lp-toolkit and the architecture.

**1.2 Document Conventions**

* Coding Standards: We adhere to the PEP 8 standards for Python, which ensures readability and uniformity across our Python codebase. JavaScript code follows Airbnb's style guide, a widely adopted set of rules that promotes code clarity and bug prevention.
* Documentation Standards: All technical documentation is written in Markdown format. This choice supports easy editing, version control compatibility, and straightforward conversion to other formats like HTML or PDF.
* Diagram Conventions: We use Unified Modeling Language (UML) diagrams for architectural and system flow visualizations. Sequence diagrams are employed to illustrate the interaction between system components over time, providing a clear, step-by-step representation of the processes.

**1.3 Intended Audience and Reading Suggestions**

Document’s audience is for developers, users and testers for this software. This document will give a more detailed overview of the project, giving information about its goal, overall architecture and technical information.

**1.4 System Overview**

There is a frontend, backend, huggingface and langchain containers. Frontend displays a chat box on the browser as the user interface. Backend connects to the frontend, MongoDB, and langchain. The langchain container will have the vectorizer, langchain and other mirco-services. Huggingface container is an LLM implemented.

**2. Design Considerations**

**2.1 Assumptions and Dependencies**

lp-toolkit can be used based on the assumption that the user has docker installed to execute the program in one command in the terminal/command line.They will have their own api keys from the 2 implemented models.

**2.2 General Constraints**

The system must operate efficiently under the load of up to 10,000 concurrent users.

The interface must be compatible with major browsers like Chrome, Firefox, and Edge.

**2.3 Goals and Guidelines**

lp-toolkit's overarching ambition is to embody accessibility, expedited learning curves, and comprehensive inclusivity—focusing on being effortlessly navigable, swiftly adoptable, and encompassing all necessary components for users. Striving to be user-friendly, it aims to minimize complexities, ensuring a seamless onboarding experience. Embracing the "batteries included" ethos, it aspires to offer a holistic package, minimizing external dependencies, and empowering users with the necessary tools and resources readily available within the toolkit environment.

#### Batteries included

Any LLM(langchain, llama, openAI, etc.) can be used by using a simple API call. This allows users to easily use the software with minimal LLM knowledge.

**2.4 Development Methods**

Dynamic development in a research project signifies an iterative and adaptable approach, embracing flexibility and evolution as foundational principles. It involves an agile methodology, allowing for continuous refinement, exploration, and adaptation of project components as new insights, technologies, or findings emerge. This approach enables the project to respond swiftly to changing requirements, novel discoveries, or user feedback, fostering innovation and ensuring that the project stays at the forefront of advancements in its field.

**2.5 System Overview**

Technology Stack Details: The project leverages Docker to encapsulate different parts of our application in containers, promoting both isolation and portability. The backend, written in Python, makes use of frameworks like Flask to handle requests, whereas the frontend relies on React, providing a dynamic user experience.

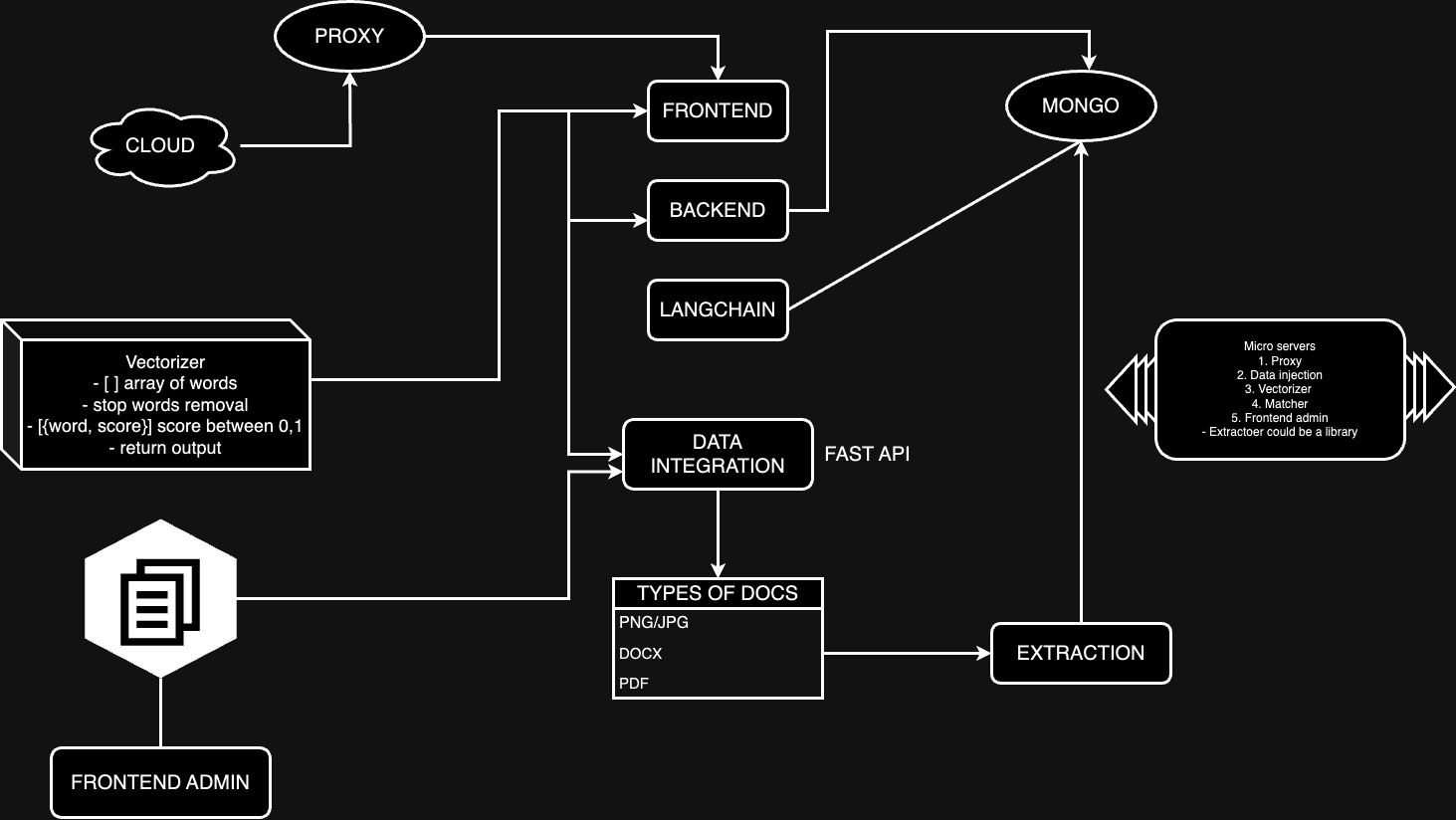
Component Interaction: There's a strong interface between frontend, backend, and LangChain microservices through well-defined RESTful APIs. These APIs facilitate data exchanges and process synchronization across containers, ensuring that the frontend dynamically reflects the system's backend logic.

**3. Architectural Strategies**

Docker is used to separate the frontend, backend, and langchain onto their own containers. This allows the use of JavaScript for the front and backend while using python in langchain. MongoDB is used for its noSQL database which makes the data flexible and easy to work with and allows for scalability.

**4. System Architecture**

When the user inputs a query into the chat box, it is sent over from the cloud to a proxy which directs it to the frontend, backend, or langchain. The frontend is what the user sees in their browser and it is written in html, CSS, and JavaScript. The backend is connected to the frontend, MongoDB, and connects to langchain through an API call. The langchain container has the vectorizer and langchain. The vectorizer rates user queries on a scale of 0-1 based on the importance of the word; it converts text into a numerical representation. Langchain digests information and returns generative text.



Additional Detail: The system architecture involves three main components: Frontend, Backend, and LangChain. Each of these components runs in its own Docker container to ensure scalability and isolation of processes. The Frontend utilizes React for dynamic user interface elements, communicating with the Backend via RESTful APIs. The Backend, developed in Node.js, handles business logic, interacts with MongoDB for data persistence, and manages communication with the LangChain component for processing natural language queries.

**5. Policies and Tactics**

**5.1 Choice of which specific products used**

Visual Studio Code. MongoDB. Docker. JavaScript. LangChain. Tailwind.

**5.2 Plans for ensuring requirements traceability**

Docker is essential for our container management, providing an isolated environment for each part of our application. Visual Studio Code is recommended for development due to its robust support for both Python and JavaScript. MongoDB is chosen for its schema-less nature, which offers flexibility in handling the unstructured data that is typical in natural language processing. LangChain is utilized for its advanced capabilities in processing and generating language-based data.

**5.3 Plans for testing the software**

Using Jest to unit test critical functions and modules. Playwright for end-to-end testing to ensure user facing functions are working. CD/CI to automatically test and push the repository,pull requests,etc..

**6. Detailed System Design**

**6.x Name of Component (Module)**

**6.x.1 Responsibilities**

This module processes natural language inputs, leveraging advanced algorithms to understand and generate human-like responses.

**6.x.2 Constraints**

The module's performance and scalability are heavily dependent on the responsiveness of LangChain's external APIs and the computational resources available.

**6.x.3 Composition**

It integrates several smaller services that handle specific aspects of language understanding and response generation.

**6.x.4 Uses/Interactions**

It serves as a bridge between the user's queries and the data processing backends, ensuring that responses are accurate and timely.

**6.x.5 Resources**

Requires high network bandwidth and CPU allocation to manage the data-intensive tasks efficiently.

**6.x.6 Interface/Exports**

Provides several API endpoints that facilitate the interaction with frontend applications and other back-end modules.

**7. Detailed Lower level Component Design**

**7.x Name of Class or File**

**7.x.1 Classification**

Serves as the central controller that manages request routing and error handling.

**7.x.2 Processing Narrative (PSPEC)**

Upon receiving a request, this class determines the appropriate service or handler based on the request type and content.

**7.x.3 Interface Description**

Exposes RESTful API endpoints for user interaction, acting as the entry point for the application.

**7.x.4 Processing Detail**

Implements robust error handling to manage exceptions and provide meaningful error messages to the end-users.

**7.x.4.1 Design Class Hierarchy**

Sits at the top of the backend service hierarchy, directing requests to the appropriate components.

**7.x.4.2 Restrictions/Limitations**

Designed to handle specific types of requests; cannot process unsupported or malformed requests.

**7.x.4.3 Performance Issues**

Optimizations are required to handle high volumes of simultaneous requests without degradation in response times.

**7.x.4.4 Design Constraints**

Must comply with strict security protocols to prevent unauthorized data access.

**7.x.4.5 Processing Detail For Each Operation**

Each function within the class is documented with specific details on input, process, and output, ensuring clarity and maintainability.

**8. Database Design**

Database used is MongoDB; a NoSQL database that offers high flexibility and scalability. The following are details of the database design, including the structure of collections and documents, key fields, indexing strategies, and relationships between different data entities. This database design aims to support the project's current needs while allowing for flexibility and scalability as the project grows and evolves.

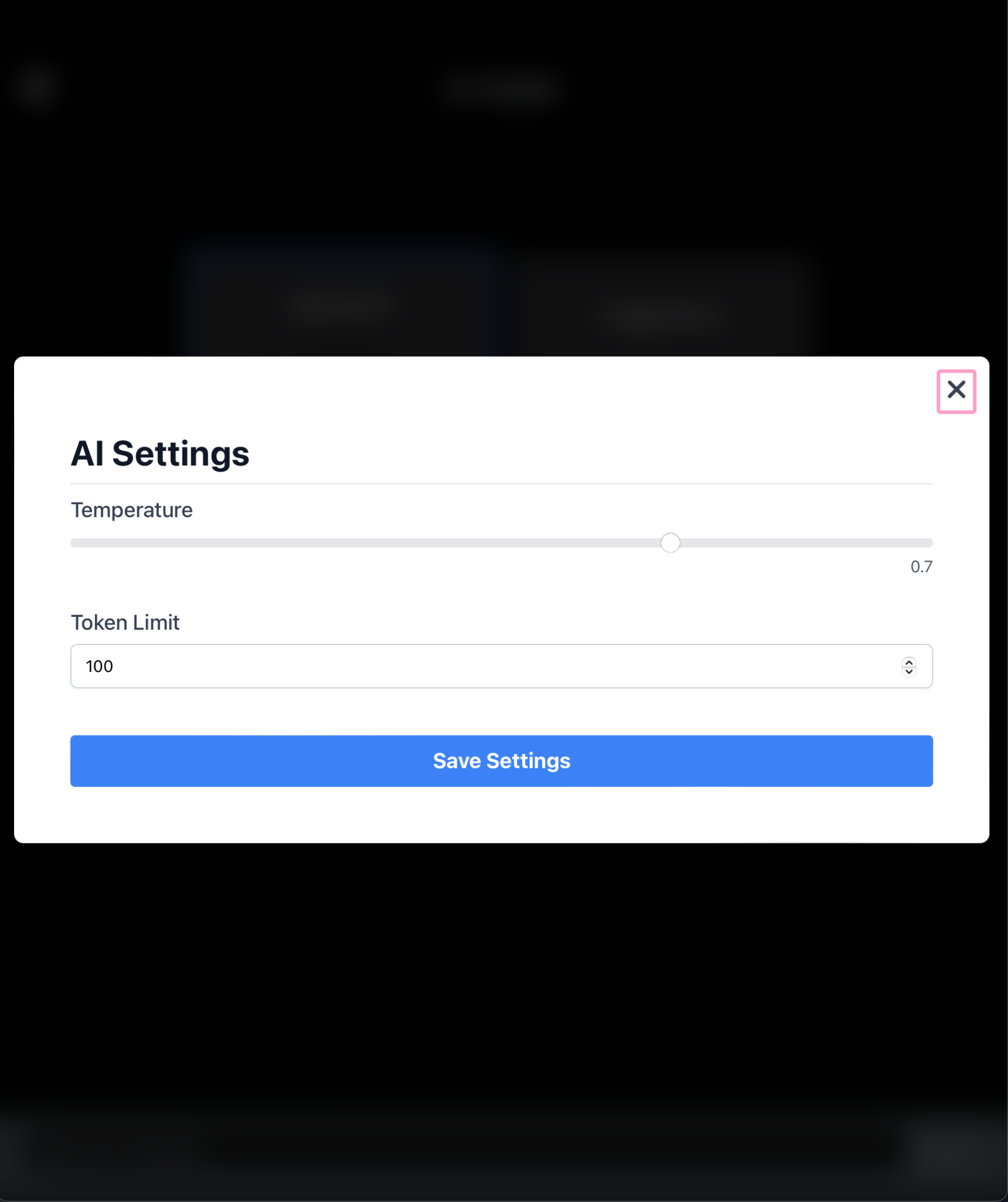
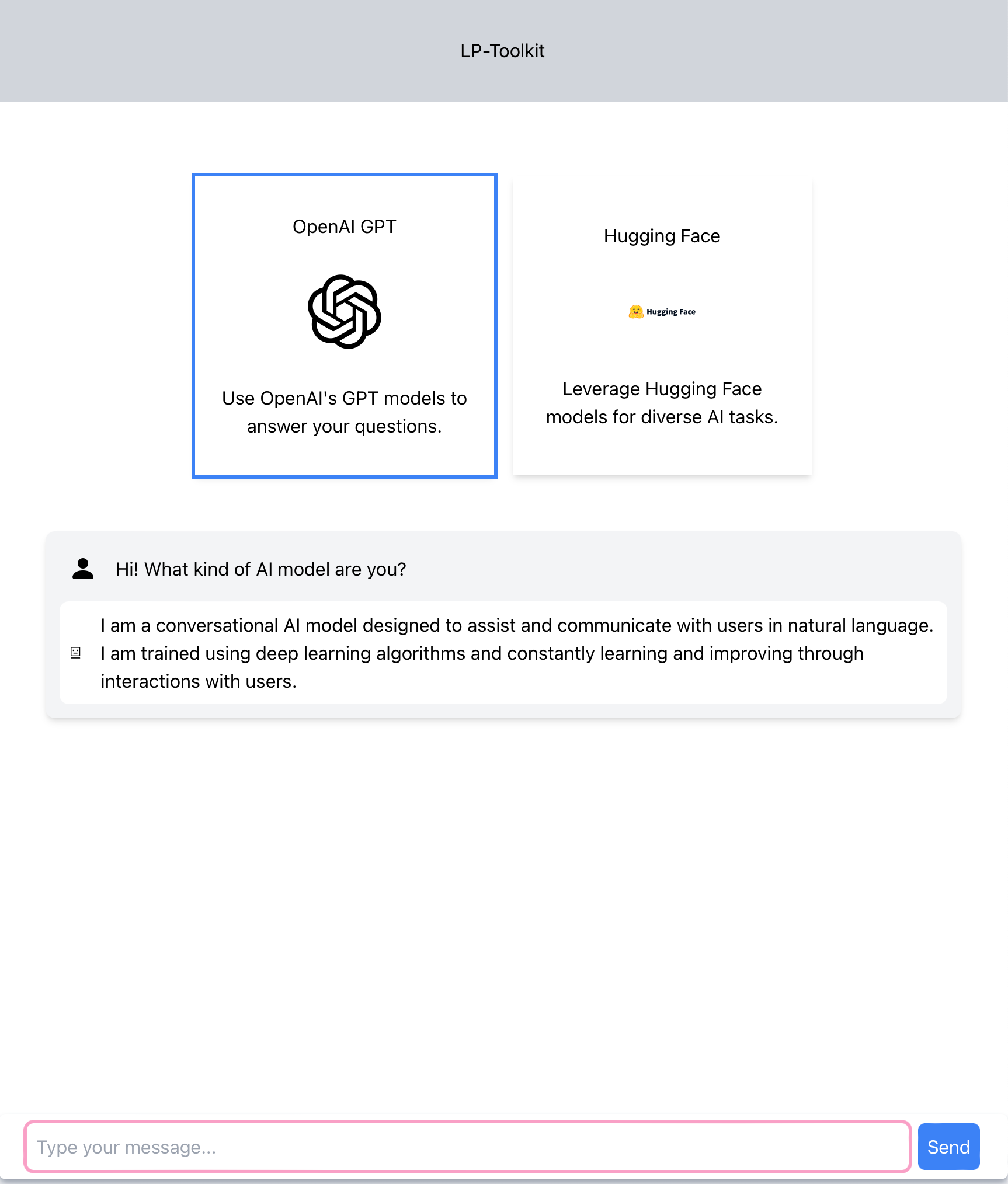
MongoDB is utilized for its schema-less structure, which provides flexibility in handling the diverse data types and structures encountered in language processing. The database schema includes collections such as Users, Sessions, and Queries. Indexing is implemented on frequently queried fields like username and session\_id to enhance performance.

**9. User Interface**

**9.1 Overview of User Interface**

The user will be able to see a chat interface with 2 of the implemented AI models to choose from. Once the user chooses one and enters a query the chosen model will answer the user’s query. By clicking on the AI at any time it will change to that model to respond.

**9.2 Screen Frameworks or Images**

****

**9.3 User Interface Flow Model**

A detailed description of the flow between different UI components, how users navigate from one part to another, including error handling and user feedback mechanisms.

**10. Requirements Validation and Verification**

Testing Strategy: A multi-tiered approach to testing includes unit tests for individual components using Jest, which ensures that each function performs as expected in isolation. Integration tests via Playwright assess the interactions between components, ensuring that they work together seamlessly. Continuous Integration and Deployment setups are configured to run these tests automatically upon code submissions to the repository, facilitating early detection of integration issues.

**11. Glossary**

1. **Agile Methodology**
   * *Definition*: A project management methodology that uses short development cycles called sprints to focus on continuous improvement in the development of a product or service.
2. **Backend**
   * *Definition*: The server-side component of a software application or website, which handles the logic, database interactions, authentication, and server configuration.
3. **CI/CD (Continuous Integration/Continuous Deployment)**
   * *Definition*: A method to frequently deliver apps to customers by introducing automation into the stages of app development. The main concepts attributed to CI/CD are continuous integration, continuous delivery, and continuous deployment.
4. **Docker**
   * *Definition*: A software platform that allows developers to quickly deploy apps as portable, self-sufficient containers that can run on the cloud or on-premises.
5. **Frontend**
   * *Definition*: The part of a website or software application that the user interacts with directly. It is typically built using technologies like HTML, CSS, and JavaScript.
6. **Git**
   * *Definition*: A distributed version control system that lets multiple developers safely work on the same project without interfering with each other's changes.
7. **JSON (JavaScript Object Notation)**
   * *Definition*: A lightweight data-interchange format that is easy for humans to read and write and for machines to parse and generate. JSON is often used for transmitting data in web applications between clients and servers.
8. **Microservices Architecture**
   * *Definition*: An architectural style that structures an application as a collection of small autonomous services, modeled around a business domain.
9. **REST** (Representational State Transfer)
   * *Definition*: An architectural style for distributed hypermedia systems, such as the World Wide Web, emphasizing scalability of component interactions, uniform interfaces, and a stateless communication protocol.
10. **Webhook**
    * *Definition*: A method used by applications to provide other applications with real-time information. A webhook delivers data to other applications as it happens, meaning you get data immediately.

**12. References**

1. **Docker Documentation.** (n.d.). Retrieved Month Day, Year, from https://docs.docker.com/

Official Docker documentation providing detailed guidelines on containerization concepts, setup instructions, and best practices for Docker files and commands.

1. **MongoDB Manual.** (n.d.). Retrieved Month Day, Year, from https://docs.mongodb.com/manual/

MongoDB’s official manual, including details on installation, configuration, and best practices for database design and performance tuning.

1. **Node.js Documentation.** (n.d.). Retrieved Month Day, Year, from https://nodejs.org/en/docs/

Documentation for Node.js, offering insights into its asynchronous programming model and various APIs supporting backend development.

1. **Stack Overflow. (n.d.)**. Discussions on containerization with Docker. Retrieved Month Day, Year, from https://stackoverflow.com/questions/tagged/docker

A collection of community discussions and problem-solving related to the use of Docker in development environments, providing practical insights and troubleshooting tips.