# AI DocQuery

**Final Year Project**

**Session 2020-2024**

A project submitted in partial fulfillment of the degree of

BS in Computer Science

Department of Computer Science

Faculty of Computer Science & Information Technology

The Superior University, Lahore

| | |
|---|---|
| Type (Nature of project) | [ ✓ ] **D**evelopment [ ] **R**esearch [ ] **R**&**D** |
| Area of specialization | MERN Stack |
| FYP ID | FYP-BCSM-F23-027 |

**Project Group Members**

| Sr.# | Reg. # | Student Name | Email ID | *Signature |
|---|---|---|---|---|
| (i) | BCSM-F20-146 | Anum Iqbal | bcsm-f20-146@superior.edu.pk | |
| (ii) | BCSM-F20-140 | Jahazaib Ashfaq | bcsm-f20-140@superior.edu.pk | |
| (iii) | BCSM-F20-238 | Amir Ali | bcsm-f20-238@superior.edu.pk | |

*The candidates confirm that the work submitted is their own and appropriate credit has been given where reference has been made to work of others

## Plagiarism Free Certificate

This is to certify that, I **Anum Iqbal** S/D of **Muhammad Iqbal**, group leader of FYP under registration no FYP-**BCSM-F23-027** at Computer Science Department, The Superior University, Lahore. I declare that my FYP report is checked by my supervisor.

Date: Name of Group Leader: Anum Iqbal Signature: _____

Name of Supervisor: Mr. Talha Amjad

Designation: Lecturer

Signature: _____

HoD: Dr. Arfan Ud Din

Signature: _____

# Project Report

## AI DocQuery Hub

### Change Record

| Author(s) | Version | Date | Notes | Supervisor's Signature |
|---|---|---|---|---|
| | 1.0 | | <Original Draft> | |
| | | | <Changes Based on Feedback from Supervisor> | |
| | | | <Changes Based on Feedback From Faculty> | |
| | | | <Added Project Plan> | |
| | | | <Changes Based on Feedback from Supervisor> | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

# APPROVAL

.

**PROJECT SUPERVISOR**

Comments: _____

_____

_

Name:_____

Date:_____ Signature:_____

.

**PROJECT MANAGER**

Comments: _____

_____

_

Date:_____ Signature:_____

**HEAD OF THE DEPARTMENT**

Comments: _____

_____

_

Date:_____ Signature:_____

# Dedication

*This work is dedicated to my unwavering commitment to excellence, fueled by the passion to innovate and create solutions that make a meaningful impact. It stands as a testament to the countless hours of dedication poured into every line of code, every design element, and every decision made with meticulous care.*

*I extend heartfelt gratitude to my unwavering supporters – my family, friends, and mentors – who have stood by me with unwavering encouragement and belief in my abilities. Their support has been the guiding light, propelling me forward even in the face of challenges.*

*This work is dedicated to the future – to the endless possibilities it holds, the challenges it presents, and the boundless opportunities for innovation and progress. May this endeavor inspire and empower others to pursue their dreams with determination and resilience.*

# Acknowledgements

*I am really thankful to my supervisor, whose guidance, expertise, and unwavering support have been invaluable throughout this journey. Their mentorship has not only enriched my understanding of the subject matter but also inspired me to push the boundaries of my capabilities.*

*Their constructive feedback, encouragement, and patience have played a significant role in shaping this project. I am truly fortunate to have had the opportunity to learn from their wisdom and experience.*

*Their dedication to fostering growth and fostering a collaborative environment has been instrumental in my development as a researcher. I extend my sincerest thanks for their generosity, professionalism, and unwavering commitment to excellence.*

# Executive Summary

**AI DocQuery**, developed within the MERN Stack framework, introduces a groundbreaking approach to information retrieval. Powered by ChatGPT APIs, this platform streamlines the user experience by allowing seamless uploads of PDF files for intuitive and efficient document-based queries.

## Key Highlights:

**Innovative User Experience:** AI DocQuery redefines user interaction by enabling direct uploads of all files. Users can effortlessly ask questions about their documents, creating a user friendly and efficient information retrieval process.

**ChatGPT API Integration:** The heart of AI DocQuery lies in its integration with ChatGPT APIs, ensuring a sophisticated natural language processing experience. The chatbot comprehends and responds to user queries, providing detailed and context aware information extracted from the uploaded documents.

**MERN Stack Framework**: Developed within the robust MERN Stack framework, AI DocQuery inherits the advantages of a secure and scalable environment. This integration enhances the overall performance and reliability of the platform, ensuring a seamless user experience.

**Elimination of Search Complexities:** Traditional search methods often prove cumbersome, especially when users struggle to express queries in a way that search engines comprehend. AI DocQuery bridges this gap by offering an intuitive and tailored search experience, significantly reducing the time and effort required for information retrieval.

**Tailored to Individual Needs:** AI DocQuery revolutionizes how users interact with textual data. By providing detailed answers and extracting additional relevant information from uploaded files,

the platform ensures that information retrieval is not only efficient but also tailored to individual user needs.

**Offline Mode Capability:** In future, AI DocQuery stands out by offering an offline mode, allowing users to access and interact with their uploaded documents even when not connected to the internet. This feature enhances flexibility and ensures uninterrupted usage, catering to users in various connectivity scenarios.

# Table of Contents

Plagiarism Free Certificate ..... 2
Dedication ..... 5
Acknowledgements ..... 6
Executive Summary ..... 7
Table of Contents ..... 9
List of Figures ..... 11
List of Tables ..... 12
Chapter 1 ..... 2
Introduction ..... 2
1.1. Background ..... 3
1.2. Motivations and Challenges ..... 3
1.3. Goals and Objectives ..... 4
1.4. Literature Review/Existing Solutions ..... 4
1.5. Gap Analysis ..... 5
1.6. Proposed Solution ..... 5
1.7. Project Plan ..... 6
1.7.1. Work Breakdown Structure ..... 7
1.7.2. Roles & Responsibility Matrix ..... 9
1.7.3. Gantt Chart ..... 10
1.8. Report Outline ..... 10
1.9. Empathy Map ..... 12
Chapter 2 ..... 13
Software Requirement Specifications ..... 13
1.1. Introduction ..... 14
1.1.2. Document Conventions ..... 15
1.1.3. Intended Audience and Reading Suggestions ..... 16
1.1.4. Product Scope ..... 18
1.2. Overall Description ..... 19
1.2.1. Product Perspective ..... 19
1.2.2. Product Features ..... 19
1.2.3. User Classes and Characteristics ..... 20
1.2.4. Operating Environment ..... 20
1.2.5. Design and Implementation Constraints ..... 20
1.2.6. User Documentation ..... 21
1.2.7. Assumptions and Dependencies ..... 21
1.3. External Interface Requirements ..... 21
1.3.1. User Interfaces ..... 21
1.3.2. Hardware Interfaces ..... 22
1.3.3. Software Interfaces ..... 24
1.3.4. Communications Interfaces ..... 25
Chapter 3 ..... 28
Use Case Analysis ..... 28

3.1. Use Case Model ..... 29
3.2. Use Case Descriptions ..... 30
Chapter 4 ..... 31
System Design ..... 31
4.1. Architecture Diagram ..... 32
4.2. Domain Model ..... 33
4.3. Entity Relationship Diagram with data dictionary ..... 34
4.4. Class Diagram ..... 36
4.5. Sequence / Collaboration Diagram ..... 37
4.6. Operation contracts ..... 38
4.7. Activity Diagram ..... 39
4.8. State Transition Diagram ..... 39
4.9. Component Diagram ..... 40
4.10. Deployment Diagram ..... 41
4.11. Data Flow diagram ..... 42
Chapter 5 ..... 43
Implementation ..... 43
5.1. Important Flow Control/Pseudo codes ..... 44
5.2. Components, Libraries, Web Services and stubs ..... 50
5.3. Deployment Environment ..... 51
5.4. Tools and Techniques ..... 51
Tools ..... 51
Techniques ..... 51
5.5. Best Practices / Coding Standards ..... 52
5.6. Version Control ..... 52
Chapter 6 ..... 53
Testing and Evaluation ..... 53
6.1. Use Case Testing ..... 54
6.2. Equivalence partitioning ..... 55
6.3. Boundary value analysis ..... 55
6.4. Data flow testing ..... 56
6.5. Unit testing ..... 56
6.6. Integration testing ..... 56
6.7. Performance testing ..... 57
6.8. Stress Testing ..... 57
Chapter 7 ..... 58
Summary, Conclusion and Future Enhancements ..... 58
7.1. Project Summary ..... 59
7.2. Achievements and Improvements ..... 59
7.3. Critical Review ..... 60
7.4. Lessons Learnt ..... 60
7.5. Future Enhancements/Recommendations ..... 60

# List of Figures


| | | |
|---|---|---|
| 1.9 | Empthy Map | 12 |
| 3.1 | Use Case Model | 29 |
| 4.1 | Architecture Diagram | 32 |
| 4.2 | Domain Model | 33 |
| 4.4 | Class Diagram | 36 |
| 4.5 | Sequence / Collaboration Diagram | 37 |
| 4.6 | Operation Contract | 38 |
| 4.7 | Activity Diagram | 39 |
| 4.8 | State Transition Diagram | 39 |
| 4.9 | Component Diagram | 40 |
| 4.10 | Deployment Diagram | 41 |
| 4.11 | Data Flow Diagram | 42 |

# List of Tables


| | | |
|---|---|---|
| 1.7.2 | Roles and Responsibility Matrix | 9 |
| 1.7.3 | Gantt Chart | 10 |
| 4.3 | Entity Relationship Diagram with data dictionary | 34 |
| 4.6 | Operation Contract | 22 |
| 5.1 | Important Flow Control / Pseudo Codes | 44 |
| 5.1.1 | Document Upload and Processing | 44 |
| 5.1.2 | Chatbot Integration | 46 |
| 5.1.3 | Maintains Uploaded Document History | 48 |

# Chapter 1

# Introduction

# **Chapter 1:** Introduction

In today's dynamic digital landscape, users encounter challenges in efficiently retrieving specific information from extensive documents or PDFs. Traditional search methods often prove cumbersome, requiring users to articulate queries precisely. Recognizing this need for a more user friendly solution, we introduce **AI DocQuery**. Developed within the **MERN Stack** framework, AI DocQuery where users can directly upload PDF files. This innovative solution, powered by ChatGPT APIs, allows users to ask natural language questions about their documents, eliminating the complexities associated with conventional search engines. By understanding user intent and extracting relevant information from uploaded files.

## 1.1. Background

AI DocQuery has emerged as a solution to simplify document-based queries in the ecommerce landscape. Fueled by the need for a more efficient way to retrieve information from various file formats, AI DocQuery was developed within the robust MERN (MongoDB, Express.js, React, Node.js) stack framework. Traditional search methods that prompted us to integrate ChatGPT APIs for a user-friendly experience.

Our platform goes beyond the norm by offering offline functionality, ensuring accessibility even without an internet connection. In the following sections, we will explore the key features that set AI DocQuery apart, how users interact with their textual data.

## 1.2. Motivations and Challenges

The motivation behind the creation of AI DocQuery stems from the desire to simplify and enhance the user experience in retrieving information from various document formats. Traditional search methods often proved challenging, leading us to explore innovative solutions.

By integrating ChatGPT APIs and leveraging the MERN stack framework, we aimed to provide users with a seamless platform for document-based queries.

However, developing AI DocQuery posed its set of challenges. Ensuring for maintaining the security of user data, and optimizing performance within the MERN stack. In future, accommodating various file formats and implementing an effective offline mode required careful consideration. Despite these challenges, the drive to deliver a groundbreaking solution that addresses user needs and surpasses existing limitations has been the guiding force behind the development of AI DocQuery.

## 1.3. Goals and Objectives

The primary goal of AI DocQuery is to redefine the landscape of document-based queries by providing users with a seamless and efficient platform. We aim to simplify information retrieval from diverse file formats, ranging from PDFs through direct file uploads.

Our key objectives include enhancing user interaction with textual data, ensuring the security and scalability of the platform within the MERN stack framework, and integrating ChatGPT APIs. AI DocQuery aspires to eliminate the complexities associated with traditional search methods, offering a user-friendly experience. In future, we aim to cater to the diverse needs of users by facilitating offline access, making document retrieval accessible anytime, anywhere.

## 1.4. Literature Review/Existing Solutions

Current document-based information retrieval relies on keyword searches, often requiring precise queries and leading to user frustration. Traditional tools like Microsoft Word and search engines struggle to comprehend natural language, highlighting a gap in user friendly solutions. AI advancements show promise, but a comprehensive platform seamlessly integrating user friendly uploads.

This literature gap motivates the development of AI DocQuery, leveraging ChatGPT APIs and the

MERN Stack framework to revolutionize document retrieval, addressing current challenges and setting a new standard.

## 1.5. Gap Analysis

The solution tools we currently use to find information in documents aren't very good at understanding what users need. This can be frustrating. Even though there have been improvements in using computers to understand human language and improve search results, there isn't a single platform that does this really well.

AI DocQuery is a new solution that tries to fix this problem. Its lets users upload documents directly and asks questions in normal language. It can understand what users mean and gives relevant answers. It uses technology like ChatGpt and the MERN Stack framework to do this. AI DocQuery wants to change how we find information in documents online, making it easier and more natural. The fact that there's a gap in existing tools shows why platforms like AI DocQuery are important. They focus on making it easier for people to find what they need online.

## 1.6. Proposed Solution

The proposed solution, AI DocQuery, addresses the existing challenges in document-based information retrieval by offering a comprehensive and user centric approach. AI DocQuery introduces a seamless platform where users can directly upload PDF files, eliminating the need for precise queries. Powered by advanced ChatGPT APIs, users to ask questions about their documents in everyday language.

AI DocQuery stands out by combining user-friendly document uploads. The chatbot not only comprehends user queries but also extracts relevant information from the uploaded documents, delivering detailed responses that are sensitive to the context of the conversation. This combined method makes finding information easier and faster, connecting what users want with what search engines can do.

Using the strong MERN Stack framework, AI DocQuery is secure, scalable, and dependable. This solution makes finding documents easier and improves the user experience by making it more personalized, efficient, and easy to use. AI DocQuery changes how we search for documents, providing a smart and new way to handle today's information needs.

## 1.7. Project Plan

The project plan for the development of AI DocQuery focuses on creating a user-friendly platform for retrieving document-based information. It starts with a detailed review solutions to understand their challenges and shortcomings.

The requirements analysis phase will focus on defining user stories, functional requirements, and nonfunctional requirements. System design will follow, encompassing architectural, database, and UI/UX design.

Development comes next, using the MERN stack technologies like Node.js, Express.js, React.js, and MongoDB for both backend and frontend tasks. Testing and performance evaluation will ensure the robustness and efficiency of the system, incorporating unit testing, integration testing, and performance testing.

Extensive documentation, including end-user documentation, application administration documentation, and system administrator documentation, will accompany the development phases. A change control system will be implemented to manage and evaluate change requests effectively.

Our toolkit includes Git for version control, ChatGPT APIs for advanced natural language processing, and libraries for parsing PDF documents. We'll use an agile approach to development, allowing us to adapt to changing project needs quickly.

This project plan offers a structured path to realize the objectives of AI DocQuery, aiming to deliver a smooth and user-friendly document retrieval platform on time.

### 1.7.1. Work Breakdown Structure

**1. Literature Review**

1.1 Research Existing Solutions

1.2 Evaluate Competing Systems

1.3 Summarize Challenges from Existing Literature

**2. Requirements Analysis**

2.1 Collect User Requirements

2.2 Document Functional Requirements

2.3 Document Non-functional Requirements

**3. System Design**

3.1 Define System Architecture

3.2 Design Database Schema

3.3 UI Mockup Design

**4. Implementation**

**4.1 Implement UI/UX Design**

Begin by creating a user-friendly interface design that aligns with the project's goals and user needs. This involves creating wireframes, mockups, and prototypes to visualize the user journey and interaction flow.

**4.2 Setup React.js Environment**

Set up the React.js development environment, including installing necessary dependencies, configuring build tools like Webpack, and organizing the project structure to facilitate efficient development.

### 4.3 Develop React.js Components

Develop reusable and modular React.js components based on the UI/UX design. This includes creating components for navigation, input forms, data display, and other UI elements to build the frontend of the application.

### 4.4 Setup Node.js Server

Configure and set up a Node.js server environment to serve the React.js frontend and handle backend logic. This involves installing Node.js, setting up Express.js for routing, and implementing middleware for request handling.

### 4.5 Develop MongoDB connection

Establish a connection to MongoDB to store and retrieve data for the application. Develop schemas and models to define the structure of the data and implement CRUD operations to interact with the database from the Node.js backend.

## 5. Testing & Performance Evaluation

5.1 Conduct Backend Unit Tests

5.2 Test System Integration

5.3 Evaluate System Performance

## 6. Documentation

6.1 Prepare User Manuals

6.2 Document System Administration Procedures

6.3 Compile Technical Documentation

## 7. Change Control System

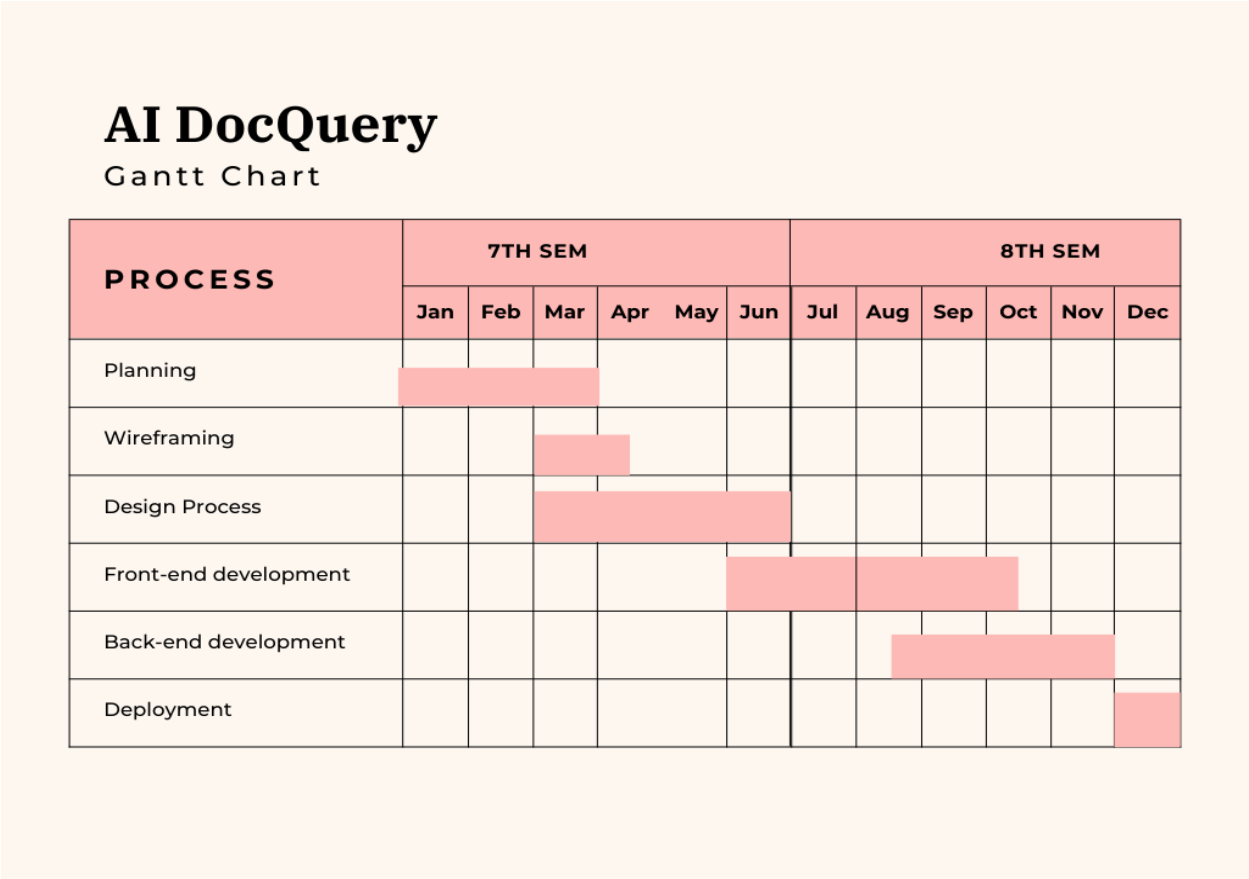
7.1 Assess Change Requests

7.2 Implement Git Workflow

### 1.7.2. Roles & Responsibility Matrix

| | | |
|---|---|---|
| **Literature Review** | Research Solutions | Evaluate Competitors |
| **Requirements Analysis** | UI Mockup Design | Collect User Requirements, Document Requirements |
| **System Design** | UI/UX Design | Define Architecture, Design Database Schema |
| **Implementation** | Implement UI/UX Design | Setup React.js Environment, Develop React.js Components, Setup Node.js Server, Develop MongoDB Connection |
| **Testing & Evaluation** | UI/UX Testing | Conduct Unit Test, System Integration, Performance Evaluation |
| **Documentation** | Prepare User Manuals, Document Administration Procedures, Compile Technical Docs | Prepare User Manuals, Document Administration Procedures, Compile Technical Docs |
| **Change Control System** | Implement Git Workflow | Access Change Requests, implement Git Workflow |

### 1.7.3. Gantt Chart




## 1.8. Report Outline

This report is structured to provide a comprehensive overview of the AI DocQuery project, aiming to revolutionize document-based information retrieval in the website domain. The outline follows a logical sequence to effectively communicate the project's background, motivations, challenges, goals, and methodologies.

**1. Introduction**

1.1 Background

1.2 Motivations and Challenges

1.3 Goals and Objectives

1.4 Literature Review/Existing Solutions

**2. Project Overview**

2.1 Innovative User Experience

2.2 ChatGPT API Integration

2.3 MERN Stack Framework

2.4 Elimination of Search Complexities

2.5 Tailored to Individual Needs

2.6 Gap Analysis

**3. System Architecture and Implementation**

3.1 Proposed Solution

3.2 MERN Stack Integration

3.3 Offline Mode Feature

3.4 System Architectural Design

3.5 Implementation Tools and Techniques

3.6 Project Plan

3.6.1 Work Breakdown Structure (WBS)

3.6.2 Gantt Chart

**4. Scope and Features**

4.1 UserFriendly Document Uploads

4.2 Natural Language Query Processing

4.3 ContextAware Responses

4.4 MERN Stack Integration

4.5 Efficient Information Retrieval

**5. Conclusion and Future Outlook**

5.1 Project Summary

5.2 Achievements and Challenges

5.3 Future Enhancements

**6. Appendix**

6.1 Roles & Responsibility Matrix

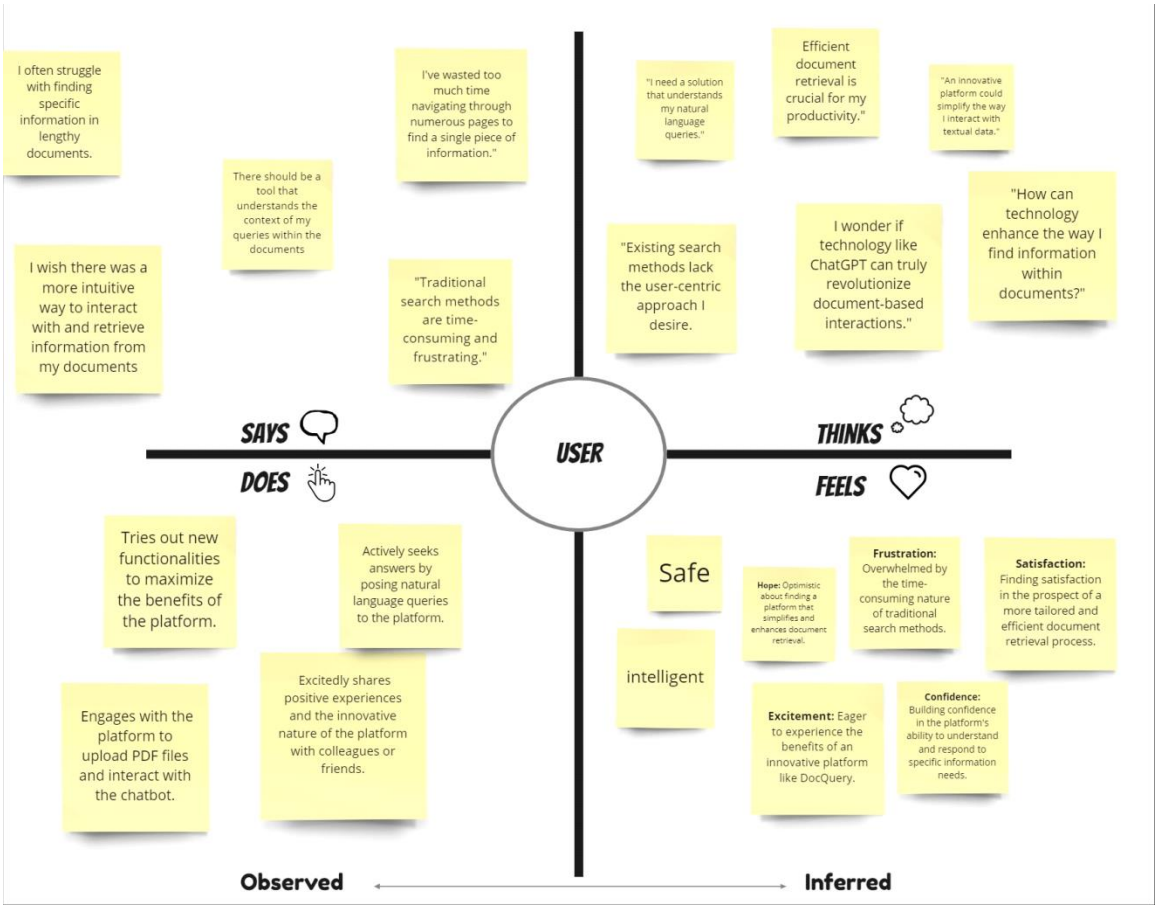
6.2 Empathy Map

6.3 HighLevel Architecture Diagram

6.4 References

This report outlines a coherent flow of information, allowing readers to delve into specific sections while comprehending the holistic journey of the AI DocQuery project.

## 1.9. Empathy Map



# Chapter 2

# Software Requirement Specifications

# **Chapter 2:** Software Requirement Specifications

## 1.1. Introduction

### 1.1.1. Purpose

The purpose of this Software Requirements Specification (SRS) document is to outline the requirements for the AI DocQuery platform. AI DocQuery is in a MERN Stack framework, focusing on efficient document-based information retrieval. This SRS details the software requirements for the initial release of AI DocQuery, providing a comprehensive understanding of the product's scope and functionality.

The document encompasses the entire system, addressing the integration of ChatGPT APIs, the MERN Stack framework, and the offline mode feature, ensuring a user centric and seamless document retrieval experience.

### 1.1.2. Document Conventions

Throughout this Software Requirements Specification (SRS) document, the following conventions are employed to enhance clarity and understanding:

1. **Naming Conventions:**

Proper nouns and specific terms related to the AI DocQuery platform are presented in Title Case (e.g., AI DocQuery, ChatGPT, MERN Stack).

2.**Acronyms and Abbreviations:**

All acronyms and abbreviations are defined upon their first occurrence in the document and subsequently used consistently (e.g., SRS for Software Requirements Specification).

3. **Formatting Styles:**

Code or technical terms are presented in a monospaced font (e.g., Node.js, React.js).

4. **Section Numbering:**

The document follows a hierarchical section numbering system (e.g., 1, 1.1, 1.1.1) for easy navigation and reference.

5. **User Interface Elements:**

User interface elements, such as buttons, menus, and dialog boxes, are italicized for distinction (e.g., Click the Submit button).

6. **Emphasis:**

Key terms, important notes, and emphasis are presented in bold (e.g., Important: Ensure proper data validation).

7. **External References:**

References to external documents, websites, or resources are provided in standard hyperlink format when applicable.

These conventions aim to standardize the presentation of information, ensuring a consistent and accessible format for readers and stakeholders.

### 1.1.3. Intended Audience and Reading Suggestions

This Software Requirements Specification (SRS) document is designed for a diverse audience, including but not limited to:

**Intended Audience:**

**1. Development Team:**

Developer involved in the design, implementation, and testing of the AI DocQuery platform.

**2. Project Managers:**

Individuals responsible for overseeing the project, ensuring it aligns with organizational goals, and meeting specified requirements.

**3. Quality Assurance (QA) Team:**

QA professionals engaged in testing and validating the functionality and performance of the AI DocQuery system.

**4. Users and EndUsers:**

Individuals who will interact with the AI DocQuery platform for document based information retrieval within an ecommerce setting.

## Reading Suggestions:

1. **Overview:**

Begin with the Introduction section to gain an overall understanding of the AI DocQuery project.

2. **Specific Roles:**

Developers should focus on sections related to system architecture, implementation, and technical details.

Project managers should pay attention to the project plan, including the Work Breakdown Structure (WBS) and Gantt Chart.

QA professionals can refer to the testing and performance evaluation sections for testing requirements.

3. **Stakeholders and Users:**

Business stakeholders and users should focus on the features and scope sections to understand the user centric aspects of the AI DocQuery platform.

4. **Appendix:**

1. **Roles & Responsibility Matrix**

A detailed breakdown of tasks and responsibilities assigned to team members throughout the project.

2. **Empathy Map**

Visual representation illustrating the thoughts, feelings, and behaviors of potential users to aid in user centric design decisions.

3. **High Level Architecture Diagram**

Visual representation showcasing the interconnected components of the AI DocQuery system, illustrating the flow of data and key architectural elements.

4. **Gantt Chart**

Timeline with milestones, tasks, and responsible team members for effective project management and tracking.

5. **References**

Citations and resources consulted during the preparation of this Software Requirements Specification (SRS) document.

### 1.1.4. Product Scope

The product scope for AI DocQuery encompasses the following key aspects:

1. **Document Based Information Retrieval:**

Users can upload PDF files, enabling seamless and intuitive document-based queries.

2. **Natural Language Processing (AI) Integration:**

Powered by ChatGPT APIs, the platform comprehends and responds to user queries in natural language, ensuring a sophisticated information retrieval experience.

3. **Mern Stack Framework Integration:**

Developed within the Mern Stack framework, AI DocQuery inherits a secure and scalable environment, enhancing overall performance and reliability.

4. **Elimination of Search Complexities:**

AI DocQuery simplifies the search experience by offering an intuitive and tailored approach, reducing the time and effort required for information retrieval.

5. **Tailored to Individual Needs:**

The platform provides detailed answers and extracts relevant information from uploaded files, ensuring a personalized and efficient information retrieval process.

6.**MERN Stack Integration:**

Developed within the MERN (MongoDB, Express.js, React.js, Node.js) stack, ensuring a modern, efficient, and scalable architecture.

7. **Offline Mode Feature:**

AI DocQuery includes an offline mode, allowing users to access and retrieve information even in scenarios with limited or no internet connectivity.

## 1.2. Overall Description

### 1.2.1. Product Perspective

AI DocQuery is a standalone platform developed within the MERN Stack framework. It operates independently, serving as a user centric solution for document-based queries within the ecommerce landscape. AI DocQuery is designed to function autonomously, streamlining document retrieval processes.

### 1.2.2. Product Features

The key features of AI DocQuery include:

**Document-Based Information Retrieval:**

Users can seamlessly upload PDF files, enabling intuitive and efficient document-based queries.

**MERN Stack Framework:**

Developed within the Merit Stack framework, AI DocQuery inherits a secure and scalable environment, ensuring reliable performance.

**Elimination of Search Complexities:**

AI DocQuery streamlines the search process by providing a user-friendly and customized method, minimizing the time and effort required for information retrieval.

**Tailored to Individual Needs:**

The platform provides detailed answers and extracts relevant information from uploaded files, ensuring a personalized information retrieval process.

**MERN Stack Integration:**

Developed within the MERN (MongoDB, Express.js, React.js, Node.js) stack, AI DocQuery benefits from a modern, efficient, and scalable architecture.

### 1.2.3. User Classes and Characteristics

AI DocQuery caters to the following user classes:

1. **End Users:**

   Individuals interacting with the platform to upload documents, ask questions, and retrieve information.

2. **Developers:**

   Individuals involved in the development, maintenance, and enhancement of the AI DocQuery platform.

### 1.2.4. Operating Environment

AI DocQuery operates in a web-based environment, accessible through standard web browsers. It is designed to function on diverse devices, ensuring compatibility with various operating systems.

### 1.2.5. Design and Implementation Constraints

**Internet Connectivity:**

We plan to launch this service in the future. While the offline mode offers basic functionality without an internet connection, to fully utilize the service, a stable internet connection is recommended.

**File Formats:**

We plan to launch this service in the future. While PDFs are the primary supported format, additional support for other file formats may be introduced in future releases.

**User Interface Compatibility:**

The platform is optimized for modern web browsers and may experience limitations on older browsers.

### 1.2.6. User Documentation

**User Interface Compatibility:**

The platform is optimized for modern web browsers and may experience limitations on older browsers.

### 1.2.7. Assumptions and Dependencies

**ChatGPT API Stability:**

The functionality of natural language processing relies on the stability and availability of the ChatGPT API.

**MERN Stack Framework:**

Assumption of continuous support and updates for the MERN Stack framework.

**User Internet Connectivity:**

The effectiveness of the platform is dependent on users having a stable internet connection for optimal performance.

## 1.3. External Interface Requirements

AI DocQuery is designed as a standalone platform, operating independently to revolutionize document-based information retrieval. While it can integrate with existing systems, its main focus is to function autonomously, offering a seamless and user-centric solution for efficient document queries.

### 1.3.1. User Interfaces

1. **Document Upload Interface:**

   Users interact with an intuitive interface for seamless PDF file uploads, enabling document-based queries.

2. **Chat Interface:**

The platform features a chat interface powered by natural language processing, allowing users to ask questions and receive context aware responses.

3. **Results Display:**

The system presents retrieved information in a clear and userfriendly manner, ensuring an efficient information consumption experience.

### 1.3.2. Hardware Interfaces

AI DocQuery operates as a web-based platform, minimizing hardware dependencies for users. The platform's components are hosted on dedicated servers, and users access the system through standard computing devices with internet connectivity.

1. **Supported Device Types:**

DocQuery is compatible with a range of computing devices, including:

- Desktop computers
- Laptops
- Tablets
- Smartphones

2. **Nature of Data and Control Interactions:**

**Data Interactions:**

- Bidirectional data flow between the software product and hardware components, allowing users to upload and retrieve documents.
- Efficient data processing to handle document uploads and natural language processing tasks.

**Control Interactions:**

- Control signals from user inputs (e.g., document uploads, queries) to the software for processing.

- Responsive control mechanisms for real time interactions within the platform.

**3. Communication Protocols:**

- AI DocQuery utilizes standard internet communication protocols for seamless interactions. These may include:
- HTTP/HTTPS for web-based communication.
- WebSocket's for real time communication in specific features.

**4. Optimized for Web Browsers:**

The platform is accessed through web browsers, ensuring compatibility with popular choices such as:

- Google Chrome
- Opera
- Mozilla Firefox
- Safari
- Microsoft Edge

**5. Technical Requirements for Development:**

Developers engage with the hardware components using standard development environments for:

**Node.js** for backend development

**React.js** for frontend development

**Express.js** for server connection

**MongoDB** to store data

Understanding the logical and physical characteristics of these hardware interfaces ensures that **AI DocQuery** can adapt to various devices and communication protocols, providing users with a versatile and accessible platform for document based information retrieval.

### 1.3.3. Software Interfaces

**DocQuery** interacts with various software components to provide a robust and integrated experience. The key software interfaces are outlined below:

1. **MERN Stack Framework:**

   DocQuery is developed within the Merit Stack framework, leveraging its secure and scalable environment. This includes interactions with the Merit Stack's core components for seamless integration.

2. **ChatGPT APIs:**

   DocQuery relies on ChatGPT APIs for sophisticated natural language processing. This interface enables the chatbot to comprehend user queries and provide detailed, context aware responses.

3. **Node.js for Backend Development:**

   Node.js is employed for building the backend of DocQuery, handling server side logic, and facilitating efficient data processing during document uploads and queries.

4. **React.js for Frontend Development:**

   React.js is used for creating the user interfaces, ensuring a dynamic and responsive frontend that enhances the overall user experience.

5. **MongoDB:**

   MongoDB serves as the database management system, storing user data, documents, and relevant information. **DocQuery** interacts with **MongoDB** for efficient data retrieval and storage.

6. **PDF Parsing Libraries:**

PDF parsing libraries compatible with Node.js, such as pdf parse, are integrated to extract text and information from uploaded PDF files during document processing.

7. **Web Browsers (Client Side):**

DocQuery is accessed through standard web browsers, ensuring compatibility and a consistent user experience across popular browsers.

### 1.3.4. Communications Interfaces

**AI DocQuery** employs various communication interfaces to facilitate interactions between the platform and external entities. The communication requirements are detailed below:

1. **Web Browser Communication:**

**Protocols:** HTTP/HTTPS

**Purpose:** DocQuery communicates with users through standard web browsers, utilizing HTTP/HTTPS protocols for secure and reliable data exchange. This includes interactions with the platform's user interfaces.

2. **ChatGPT API Communication:**

**Protocols:** HTTPS

**Purpose:** Interaction with ChatGPT APIs involves secure communication over HTTPS, ensuring the confidentiality and integrity of data exchanged between DocQuery and the natural language processing service.

3. **Document Upload and Retrieval:**

**Protocols:** HTTP/HTTPS

**Purpose:** The platform communicates with external servers to handle document uploads and retrievals. Secure HTTP/HTTPS protocols are employed to maintain the integrity and confidentiality of user data.

**4. Error Reporting and Logging:**

**Protocols:** HTTP/HTTPS

**Purpose:** In case of errors or issues, the platform may communicate with external servers for error reporting and logging. HTTP/HTTPS protocols ensure secure transmission of error data.

**5 .Optimization:**

Efficient data transfer rates for quick document uploads, queries, and results retrieval, ensuring a responsive user experience.

## 2.4 System Features

## 2.4.1 Document Upload and Processing

### 2.4.1.1 Description and Priority:

Allow users to upload documents in various formats.

Priority: High

### 2.4.1.2 Stimulus/Response Sequences:

**User Action:** User clicks on the "Upload Document" button.

**System Response:** AI DocQuery prompts users to select a file; once selected, the document is uploaded and processed.

### 2.4.1.3 Functional Requirements:

REQSF11: The system shall support uploading documents in PDF format.

REQSF12: Upon document upload, the system shall extract text and information using PDF parsing libraries.

REQSF13: The system shall display a confirmation message upon successful document processing.

## 2.4.2.1 Description and Priority:

Enable users to ask natural language questions about uploaded documents.

Priority: High

#### 2.4.2.2 Stimulus/Response Sequences:

**User Action:** User types a natural language query.

**System Response:** DocQuery processes the query using ChatGPT APIs and provides a context aware response.

#### 2.4.2.3 Functional Requirements:

REQSF21: The system shall process natural language queries using ChatGPT APIs.

REQSF22: DocQuery shall understand and interpret user intent from the natural language queries.

REQSF23: The system shall provide detailed and context aware responses to user queries.

## 2.5 Other Nonfunctional Requirements

### 2.5.1 Performance Requirements

The system should respond to user queries within 5 seconds under normal operating conditions.

### 2.5.2 Safety Requirements

AI DocQuery should not perform any actions that could lead to data loss or system instability.

### 2.5.3 Software Quality Attributes

The system should provide a user-friendly interface for seamless interaction.

## 2.6 Other Requirements

No additional requirements at this time.

# Chapter 3

# Use Case Analysis

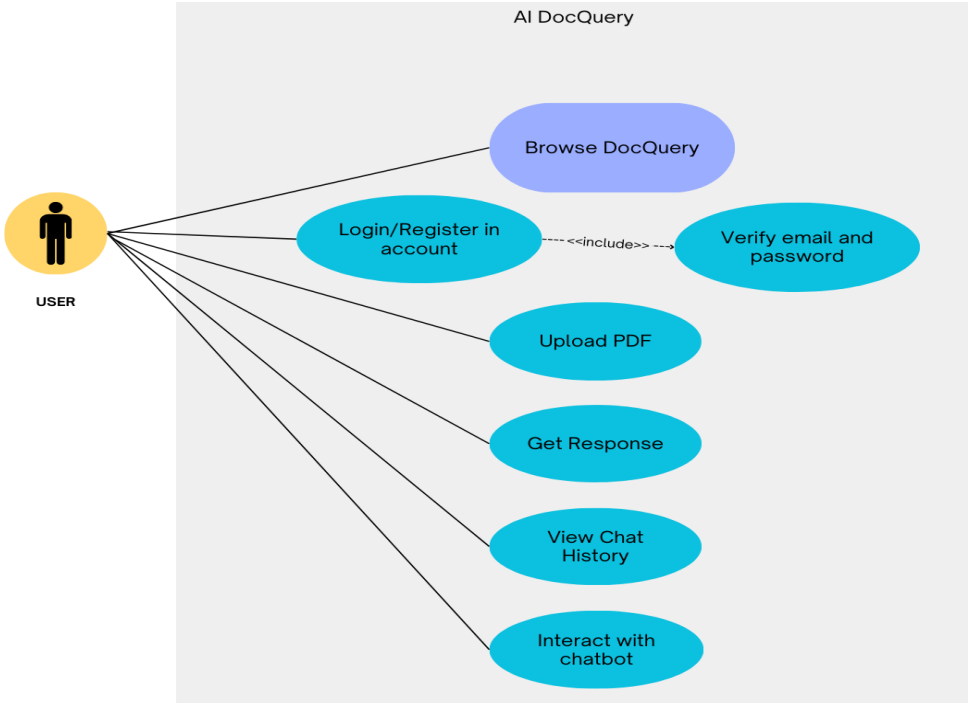
# **Chapter 3:** System Analysis

AI DocQuery is a software application that interacts with users through conversational interfaces, typically text-based, to provide information or perform tasks. Here's a breakdown of its components and functions:

1. **User Interface**
2. **Dialog Management**
3. **Knowledge Base/Backend**
4. **Integration**
5. **Machine Learning (ML)**
6. **Analytics and Monitoring**
7. **Security and Privacy**
8. **Deployment**
9. **Maintenance and Updates**

A successful DocQuery system requires careful design and integration of these components to deliver a seamless and practical user experience.

## 3.1. Use Case Model



## 3.2. Use Case Descriptions

**Login:** The user can log in to the system using their credentials (username/email and password).

**Register:** The user can register for a new account by providing necessary details such as username, email, and password.

**Verify Email:** After registration, the user receives a verification email. They can verify their email address by clicking on the verification link provided in the email.

**Upload PDF:** The user can upload a PDF document to the system. They select the PDF file from their device and upload it.

**View History:** The user can view their interaction history, which includes their chat messages exchanged with the chatbot.

**Interact with Chatbot**: The user can interact with the chatbot by sending messages. The chatbot responds to user queries, provides assistance, and offers relevant information based on the conversation.

**Upload PDF (Again)**: Similar to the initial upload, the user can upload another PDF document to the system at any time.

# Chapter 4

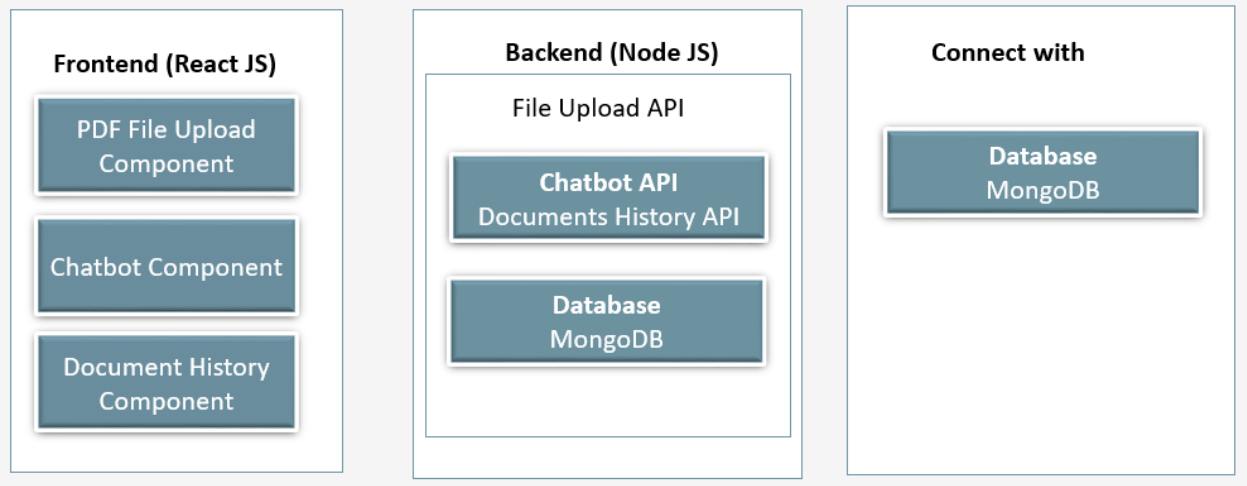
# System Design

# **Chapter 4:** System Design

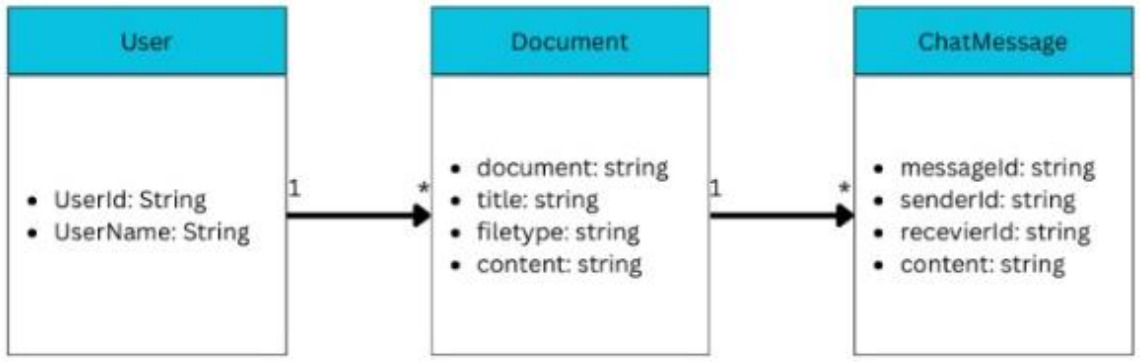
System Design Based On: -

- Architecture Design
- Domain Model
- Entity Relationship Diagram
- Class Diagram
- Sequence / Collaboration Diagram
- Operation Contracts
- Activity Diagram
- State Transition Diagram
- Component Diagram
- Deployment Diagram
- Data Flow Diagram

## 4.1. Architecture Diagram



## 4.2. Domain Model




**User:** Represents a user of the system. Each user has a unique userId and a username.

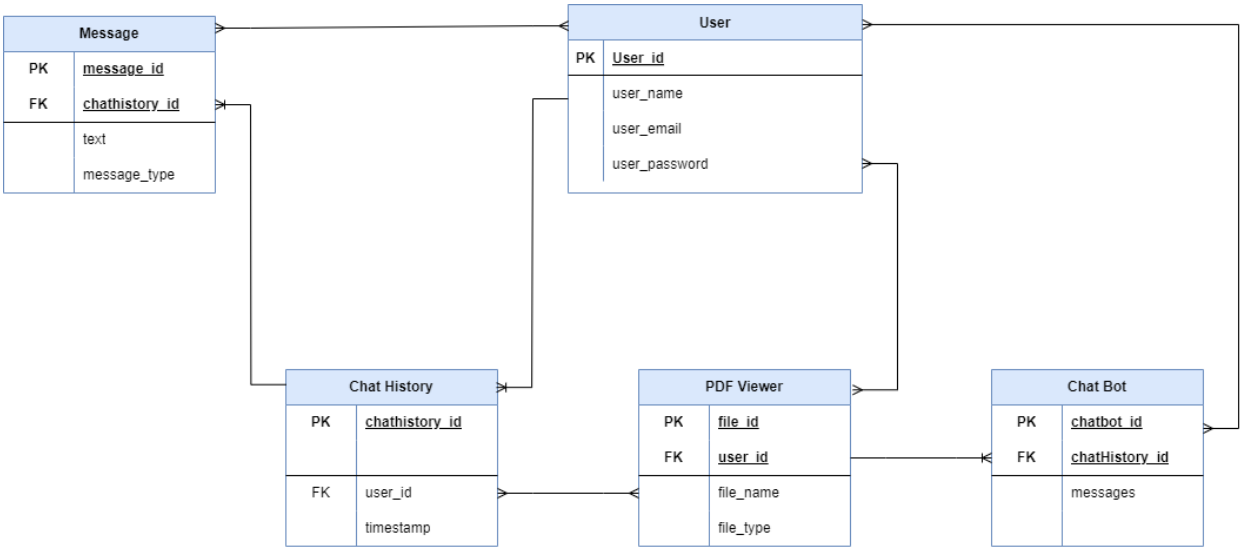
**Document:** Represents a document uploaded by a user. Each document has a unique documentId, a title, fileType, and content.

**ChatMessage:** Represents a message exchanged between users. Each message has a unique messageId, a senderId, a receiverId, and content.

**The relationships:**

- Each User can upload multiple Documents.
- Each User can send multiple ChatMessages, and each ChatMessage is sent by a single User .
- Each User can receive multiple ChatMessages, and each ChatMessage is received by a single User.
- This domain model provides a clearer depiction of the entities and their relationships in the system.

## 4.3. Entity Relationship Diagram with data dictionary




**Entities:**

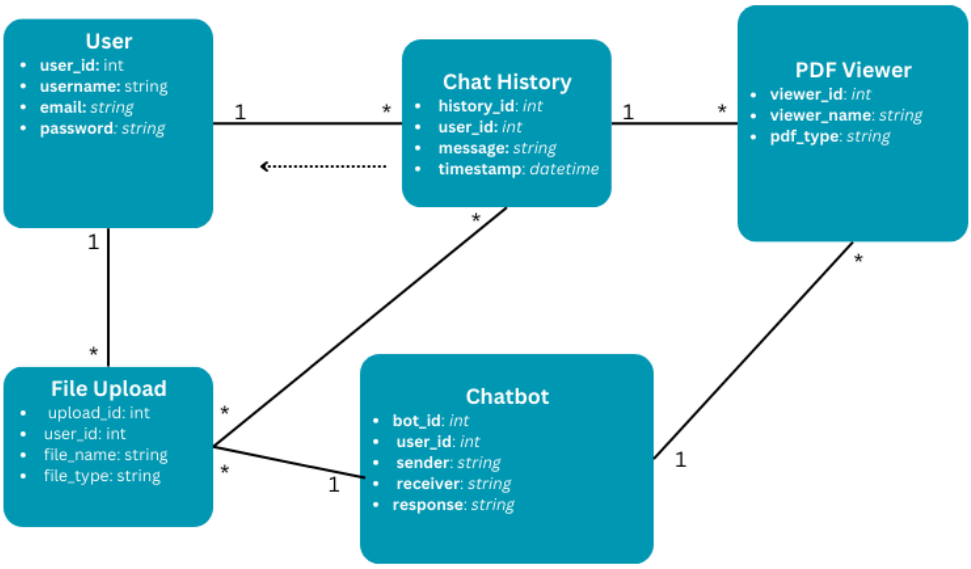
1. User
2. Chat History
3. PDF Viewer
4. Chatbot
5. Message

**Attributes:**

1. **User:**
    - User ID (Primary Key)
    - User Name
    - User Message
2. **Chat History:**
    - Chat History ID (Primary Key)
    - User ID (Foreign Key)
    - User Message or PDF (depending on the type of interaction)
    - Date and Time

3. **PDF Viewer:**
    - PDF Viewer ID (Primary Key)
    - Name of PDF Viewer
    - Type of PDF
    - Chatbot ID (Foreign Key)
    - Chat History ID (Foreign Key)
4. **Chatbot:**
    - Chatbot ID (Primary Key)
    - User ID (Foreign Key)
    - Message Sender
    - Message Receiver (assuming this is the bot)
    - Bot Response
    - Chat History ID (Foreign Key)
5. **Upload file:**
    - Message ID (Primary Key)
    - Chathistory ID (Foreign Key)
    - Text
    - Message Type

## 4.4. Class Diagram




This class diagram represents the structure of the system with four main classes: User, ChatHistory, PdfViewer, and FileUpload.

**User:** Represents the users of the system. Each user can have multiple interactions with the system, such as viewing chat history, uploading files, and interacting with the chatbot.

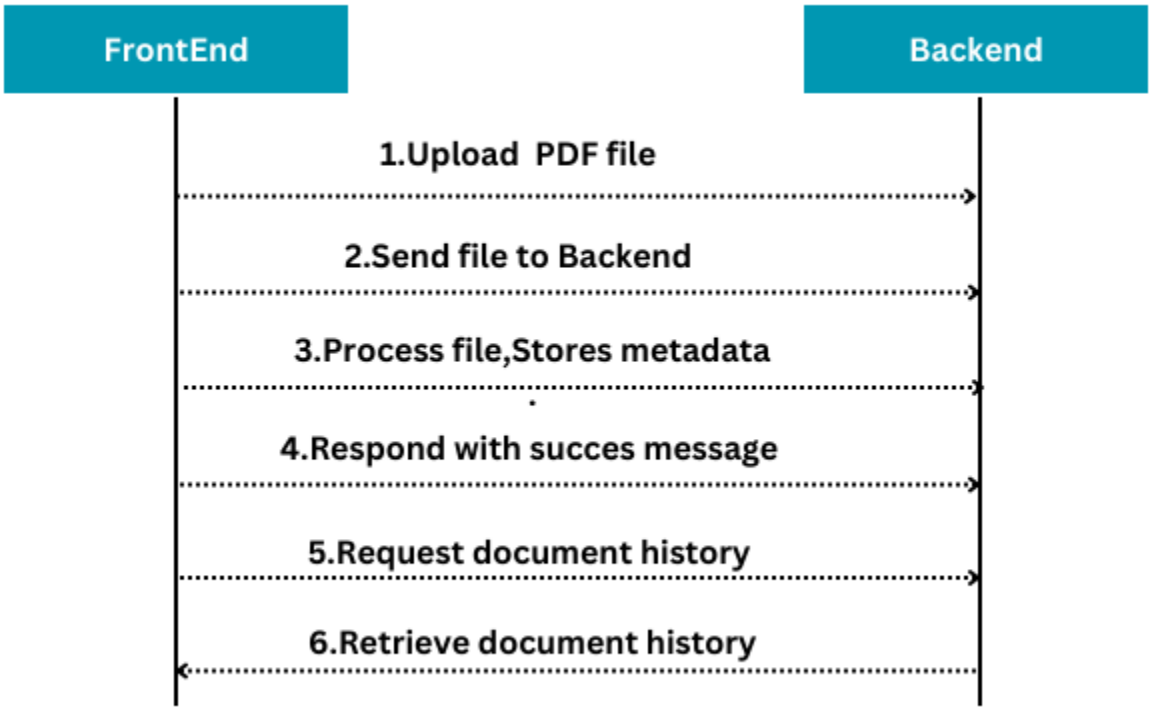
**ChatHistory**: Stores the chat history of each user. It is associated with the User class, indicating that each user can have multiple chat history entries.

**PdfViewer**: Represents the functionality to view PDF files uploaded by users. It is associated with the User class, indicating that each user can upload and view PDF files.

**FileUpload**: Represents the functionality to upload files. It is associated with the User class, indicating that each user can upload files to the system.

The associations between the classes indicate the relationships between them. For example, the one-to-many relationship between User and ChatHistory indicates that one user can have multiple chat history entries, but each chat history entry belongs to only one user. Similarly, the other associations follow similar relationships between the classes.

## 4.5. Sequence / Collaboration Diagram



## 4.6. Operation contracts

Here are the operation contracts for the main operations in your project:

**1. User Registration:**

**Preconditions**: The user provides valid registration details including username, email, and password.

**Postconditions:** A new user account is created in the system, and the user is logged in automatically.

**2. User Login:**

**Preconditions:** The user provides valid login credentials including username/email and password.

**Postconditions:** The user is authenticated and granted access to their account dashboard.

**3. Email Verification:**

**Preconditions**: The user requests an email verification link after registration.

**Postconditions**: An email verification link is sent to the user's provided email address.

**4. PDF Upload:**

**Preconditions:** The user selects a PDF file to upload.

**Postconditions**: The PDF file is successfully uploaded to the system.

**5. View Chat History:**

**Preconditions**: The user accesses the chat history section in the application.

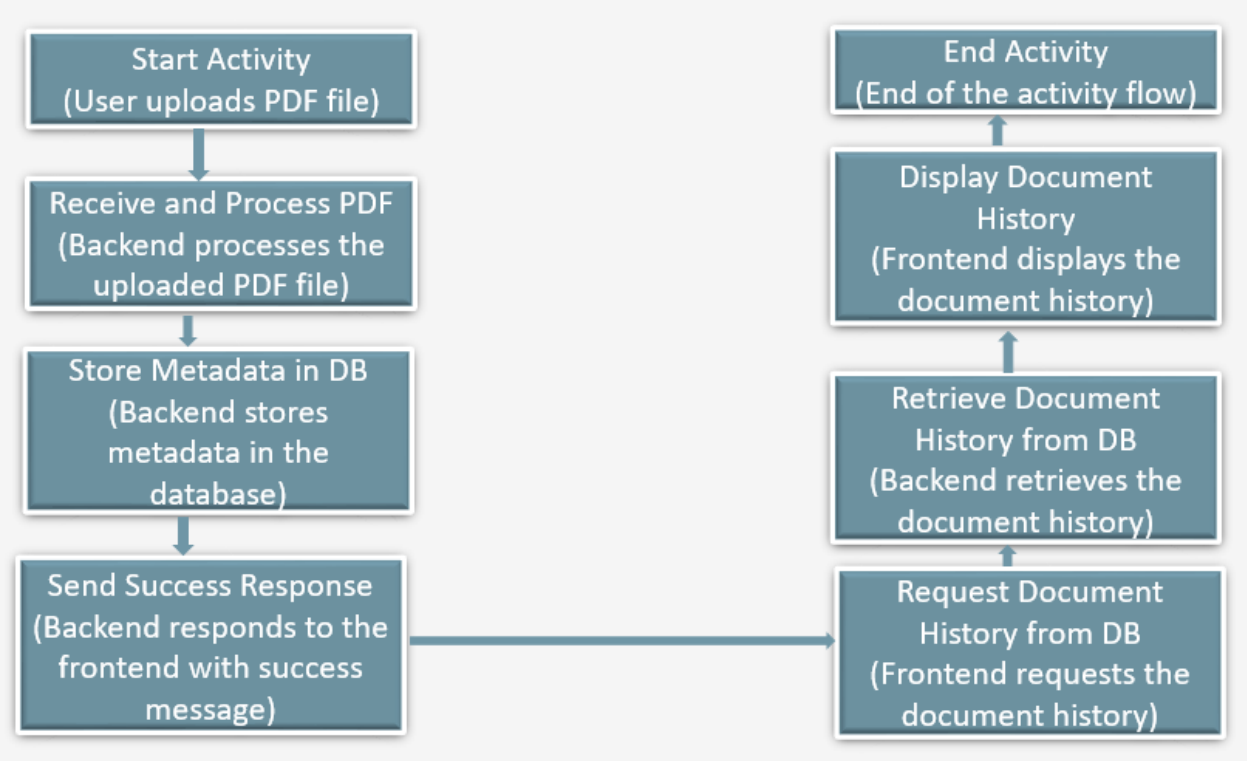
**Postconditions:** The user can view a list of past chat interactions with the chatbot.

**6. Interact with Chatbot:**

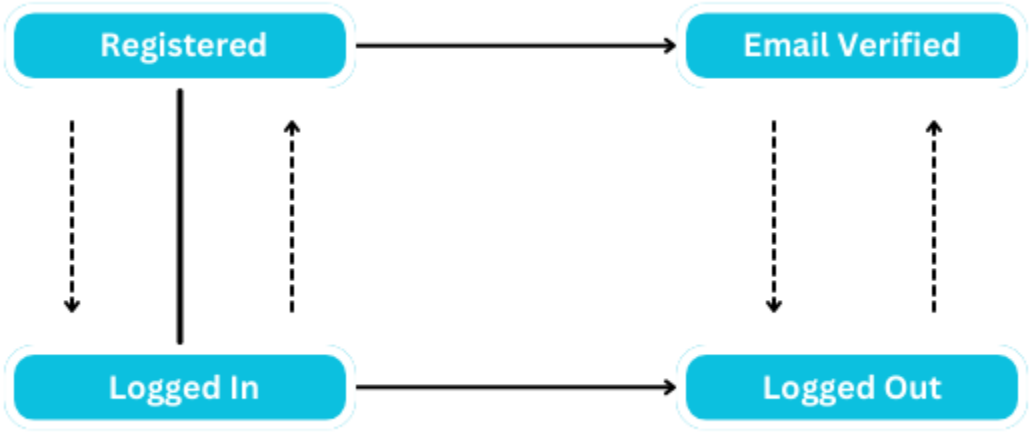
**Preconditions:** The user accesses the chat interface in the application.

**Postconditions:** The user can send messages to the chatbot and receive responses.

## 4.7. Activity Diagram



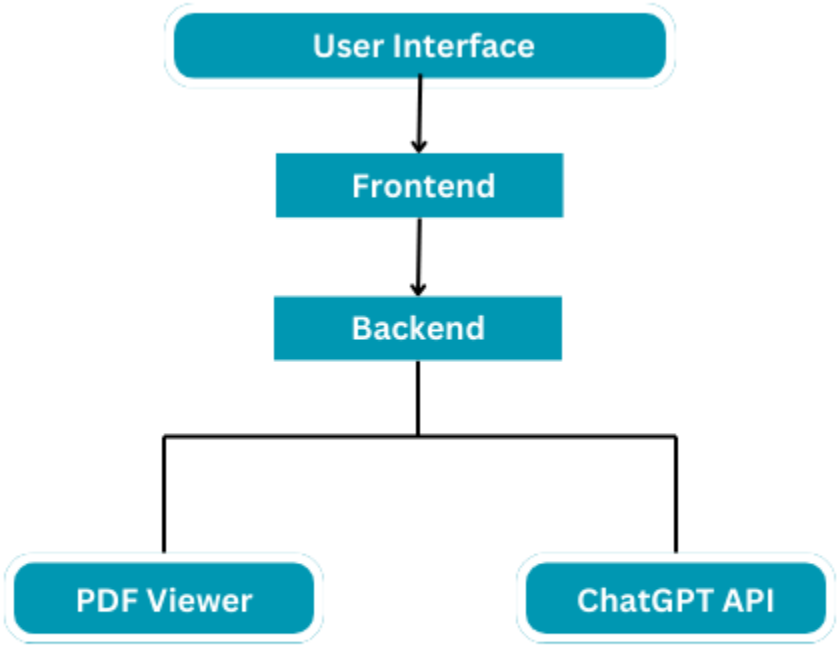

## 4.8. State Transition Diagram




- Registered: Initial state when a user signs up but hasn't verified their email.
- Email Verified: Transition occurs when the user verifies their email.

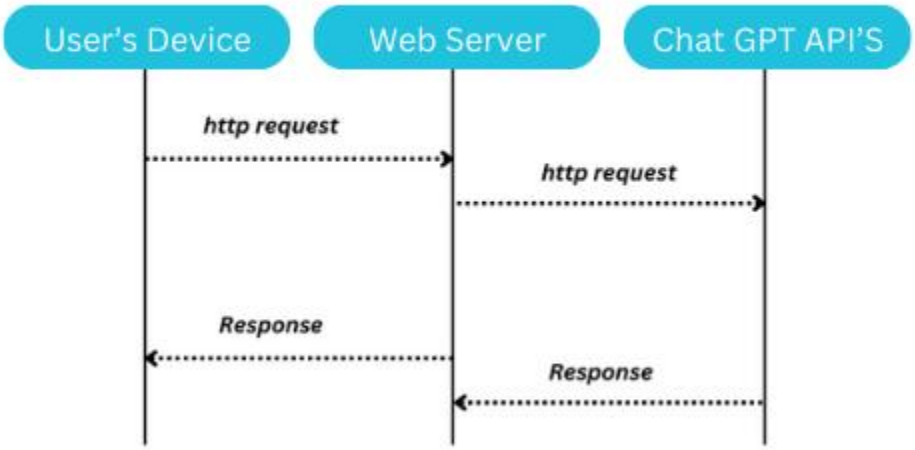
- Logged In: Transition occurs when the user successfully logs in.
- Logged Out: Transition occurs when the user logs out.

## 4.9. Component Diagram



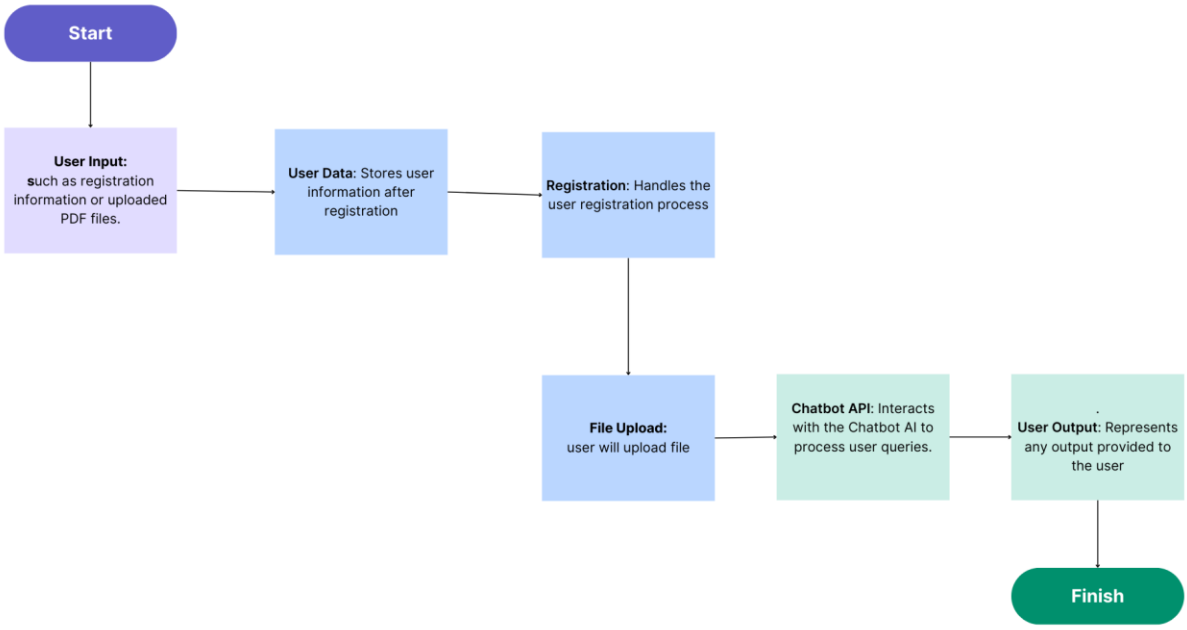

- User Interface: Represents the interface through which users interact with the system.
- Frontend: The user interface component responsible for rendering the interface and handling user interactions.
- Backend: The backend component responsible for processing user requests, managing data, and interacting with external components.
- PDF Viewer: Component for viewing PDF files uploaded by users.
- ChatGPT API: External API component responsible for natural language processing and generating responses for user queries.

## 4.10. Deployment Diagram




- The "User's Device" represents any device used by the user to interact with the system, such as a computer or mobile device.
- The "Web Server" hosts the web application where the user interacts with the system. It handles HTTP requests from the user's device.
- The "Web Server" hosts the web application where the user interacts with the system. It handles HTTP requests from the user's device.
- Responses from the ChatGPT API are sent back to the web server, which then forwards The "ChatGPT API" is an external service that the system communicates with to process natural language queries and generate responses.
- HTTP requests are sent from the user's device to the web server, where they are processed and forwarded to the ChatGPT API if necessary to the user's device for display.

## 4.11. Data Flow diagram




- User Input: Represents any input provided by the user, such as registration information or uploaded PDF files.
- User Data: Stores user information after registration.
- Registration: Handles the user registration process, including email verification.
- PDF Upload: Manages the process of uploading PDF files.
- Chatbot API: Interacts with the Chatbot AI to process user queries.
- Chatbot AI: Processes user queries using AI algorithms.
- User Output: Represents any output provided to the user, such as chatbot responses or other system messages.

# Chapter 5
# Implementation

# Chapter 5: Implementation

This chapter focuses on the detailed implementation aspects of the project. It covers the important flow control, components, libraries, web services, deployment environment, tools, techniques, best practices, coding standards, and version control used during the development process.

## 5.1. Important Flow Control/Pseudo codes

In the implementation of the project, several critical components employ specific flow control mechanisms and algorithms to achieve desired functionality. The following provides a concise overview of the high-level logic and pseudo codes used in these components:

### 5.1.1 Document Upload and Processing

The document upload and processing workflow involve the following steps:

```
import React, { useState } from "react";
import { FilePond } from "react-filepond";
import "filepond/dist/filepond.min.css";
import "../PdfViewer/pdfViewer.css";
import { usePdf } from "../Contexts/fileContext";
import { ImBin } from "react-icons/im";

const PdfViewer = () => {
  const [uploadedFiles, setUploadedFiles] = useState("");
  const {  extractTextFromPDF } = usePdf();

  // handler for upload the file
  const handleFileUpload = async (files) => {
    if (files.length > 0) {
      const uploadedFile = files[0];
      setUploadedFiles([...uploadedFiles, uploadedFile]);
      extractTextFromPDF(uploadedFile.file);
    }
  };

  // handler for delete the uploaded file
  const handleDelete = (file) => {
    const filteredFiles = uploadedFiles.filter((item) => item !== file);
    setUploadedFiles(filteredFiles);
  };

```

```
  return (
    <div className="pdf-container">
      <h3>PDF Viewer</h3>

      {!uploadedFiles.length ? (

        <FilePond onupdatefiles={handleFileUpload} />
      ) : (
        <div>
          {uploadedFiles.map((file, index) => (
            <div key={index}>
              <embed
                src={URL.createObjectURL(file.file)}
                width="100%"
                height="400vh"
                type="application/pdf"
              />
              <button
                className="delete-button"
                type="button"
                onClick={() => handleDelete(file)}
              >
                  <ImBin />
              </button>
            </div>
          ))}
        </div>
      )}
    </div>
  );
};

export default PdfViewer;
```

### 5.1.2 Chatbot Integration

The chatbot interaction follows a conversational flow using natural language processing (AI) techniques. Here's a simplified pseudo code snippet:

```
import React, { useState } from "react";
import "./chatbot.css";
import { FaLocationArrow } from "react-icons/fa";
import { usePdf } from "../Contexts/fileContext";
// import { useCurrentHistoryId } from "../Contexts/fileContext";
import axios from "axios";

const Chatbot = () => {
  const [userInput, setUserInput] = useState("");
  const [userReceiver, setUserReceiver] = useState("");
  const [response, setResponse] = useState(false);
  const { textFromPDF } = usePdf();
  // const { selectedMessage } = useCurrentHistoryId();

  const handleChat = async (e) => {
    e.preventDefault();
    console.log(userInput);

    if (!userInput.trim()) {
      return;
    }

    axios
      .post("http://localhost:5555/api/chat/getreply", {
        userInput,
        pdfContent: textFromPDF,
      })
      .then((response) => {
        setUserReceiver(response.data.message);
        setResponse(true);
        // setUserInput("");
      })
      .catch((error) => {
        console.error(error);
      });
  };

```

```
return (
  <div className="chatbot-container">
    <h3>Chatbot</h3>
    <div className="message-area">
      <div className="text-area">
        {response && (
          <div className="message-output">
            <div className="send">
              <p>{userInput}</p>
              {/* <p>{selectedMessage}</p> */}
            </div>
            <div className="receive">
              <p>{userReceiver}</p>
              {/* <p>{selectedMessage}</p> */}
            </div>
          </div>
        )}
        <div className="message-input">
          <input
            type="text"
            value={userInput}
            placeholder="Enter your question here! (max 1,000 characters)"
            onChange={(e) => setUserInput(e.target.value)}
          />
          <div className="send-button">
            <button onClick={handleChat}>
              <FaLocationArrow />
            </button>
          </div>
        </div>
      </div>
    </div>
  </div>
);
};

export default Chatbot;

```

### 5.1.3 Maintains Uploaded Document History

This presupposes the presence of a database framework designed to store chat history.

```jsx
import React, { useState, useEffect } from "react";
import "./chatHistory.css";
import axios from "axios";
import { useCurrentHistoryId } from "../Contexts/fileContext";

const ChatHistory = () => {
  const [userId] = useState("2fb29582e9e84140f4b626f6");
  const [historyData, setHistoryData] = useState([]);
  const [error, setError] = useState(null);
  const [loading, setLoading] = useState(false);
  const { setSelectedMessage } = useCurrentHistoryId();

  const fetchHistory = async () => {
    setLoading(true);

    try {
      const response = await axios.get(
        `http://127.0.0.1:5555/api/chat/getchathistoriesbyuserid?userId=${userId}`,
        {
          headers: {
            "Cache-Control": "no-cache",
          },
        }
      );
      if (
        response.data.userHistory &&
        Array.isArray(response.data.userHistory)
      ) {
        setHistoryData(response.data.userHistory);
        setError(null);
      } else {
        setError("No chat history available.");
      }
    } catch (error) {
      console.error("Error fetching chat history:", error);
      setError("Error fetching chat history. Please try again later.");
    } finally {
      setLoading(false);
    }
  };

  useEffect(() => {
    fetchHistory();
  }, []);

  const formatTimestamp = (timestamp) => {
    const date = new Date(Number(timestamp));
    return date.toLocaleString();
  };

  // const handleAlert = (id) => {
  //   alert(`ID: ${id}`);
  //   console.log(historyData);
  // };

  const handleSelectMessage = (message) => {
    setSelectedMessage(message);

  };
```

```
61    return (
62      <div className="chat-history-container">
63        <h3>Chat History</h3>
64        <br></br>
65        <div className="main">
66          {error && <p className="error">{error}</p>}
67          {historyData && historyData.length > 0 ? (
68            <div className="history-data">
69              {historyData.map((item, index) => (
70                <div
71                  key={index}
72                  className="message-container"
73                  onClick={() => handleSelectMessage(item._id)}
74                >
75                  <div className="message">{item.message}</div>
76                  <div className="timestamp">
77                    {formatTimestamp(item.timestamp)}
78                  </div>
79                </div>
80              ))}
81            </div>
82          ) : (
83            !loading && <p>No chat history available.</p>
84          )}
85        </div>
86      </div>
87    );
88  };
89
90  export default ChatHistory;
91
```

## 5.2. Components, Libraries, Web Services and stubs

### 5.2.1 Backend Components:

1. **PDF Processing Library:**

   **Library:** pdf-parse

   **Description:** Utilized for extracting text and metadata from uploaded PDF documents.

2. **Backend Framework:**

   **Framework**: Node.js and Express.js

   **Description:** Serving as the backend framework to handle HTTP requests, API endpoints, and interact with the database.

3. **Database Management System:**

   **Database:** MongoDB

   **Description:** A MongoDB database for storing document and chat histories.

### 5.2.2 Frontend Components:

1. **Frontend Library:**

   **Library:** React.js

   **Description:** Building a dynamic and responsive user interface for seamless document interaction.

2. **State Management:**

   **Library:** Context API

   **Description:** Utilizing Context API for managing state across the React application.

### 5.2.3 Chatbot Integration:

Integrated into the backend for creating and training the chatbot, enhancing natural language understanding.

### 5.2.4 Stubs and Mocks:

Employing Figma for creating mock tests to simulate interactions with external services during the testing phase.

## 5.3. Deployment Environment

In configuring the deployment environment for our project, we've created a robust and secure setup that covers both the backend and frontend components. The backend, built with Node.js, is hosted on reliable cloud platforms such as AWS or Azure and integrates smoothly with our scalable MongoDB database. Our frontend is hosted on platforms like Netlify, and we manage continuous deployment through GitHub Actions.

## 5.4. Tools and Techniques

## Tools

- **Visual Studio Code** for coding
- **Git/GitHub** for version control
- **npm** for package management
- Backend relies on **Node.js** and **Express.js**
- **MongoDB** serves as the scalable database solution

## Techniques

- React.js powers dynamic interfaces, and Context API manages state across components.
- Chatbot functionality is integrated using the Rasa framework, enhancing natural language understanding.
- External services include Elasticsearch for efficient search and Firebase Cloud Messaging for real-time notifications.
- Testing is robust with Jest for mock testing, ensuring feature accuracy.
- Project management is facilitated by Trello, aiding in task organization, and Slack promotes seamless team communication.

## 5.5. Best Practices / Coding Standards

- We enforce a strict version control strategy, leveraging Git and GitHub effectively.
- Comprehensive and well-structured documentation is maintained in Markdown, providing valuable insights for developers and users alike.
- In terms of security, sensitive information is safeguarded through secure practices, and data validation is rigorously implemented to mitigate potential vulnerabilities.
- The use of environment variables ensures secure storage of confidential information.

## 5.6. Version Control

- **Tool:** Git and GitHub
- **Branching Strategy:** Disciplined feature branching
- **Commit Practices:** Regular commits with meaningful messages
- **Merging:** Feature branches merged into the main branch
- **Documentation:** Well-documented version history
- **Efficiency:** Contributes to an efficient and organized development workflow.

# Chapter 6

# Testing and Evaluation

# **Chapter 6:** Testing and Evaluation

Testing and evaluation are crucial steps in ensuring the quality and functionality of a product or service. By rigorously testing all features and functionalities, we can identify and address any potential issues or bugs. This process helps us to improve the user experience, enhance performance, and ensure the reliability of the service. Additionally, user feedback plays a vital role in this phase, providing valuable insights that help us refine and optimize the product further. Our goal is to deliver a robust and reliable solution that meets the needs and expectations of our users.

## 6.1. Use Case Testing

Use case testing focuses on validating the specific scenarios or situations in which users interact with the product. By identifying and testing these use cases, we can ensure that the product meets the functional requirements and performs as expected in real-world situations. This type of testing helps us to understand how users will navigate through the system, interact with its features, and achieve their goals.

For example, in the context of our document retrieval platform, use case testing might involve scenarios such as:

1. Searching for a specific document by its title or keywords.
2. Filtering search results based on various criteria like date, author, or file type.
3. Accessing and viewing documents in PDF formats.
4. Sharing documents with other users or exporting them to external storage.

## 6.2. Equivalence partitioning

Equivalence partitioning is a software testing technique used to divide the input data of a program into different partitions or sets that are considered equivalent. The main objective of this technique is to reduce the number of test cases while still ensuring adequate test coverage.

Here's how equivalence partitioning works:

1. Identify the input conditions or variables that the program uses.
2. Divide the input data into different partitions based on these conditions.
3. Select representative values from each partition to create test cases.
4. Test each test case to ensure that the program behaves correctly across the entire partition.

## 6.3. Boundary value analysis

Boundary value analysis is another software testing technique that focuses on testing the boundary or edge cases of input ranges. The main idea behind this technique is that errors often occur at the boundaries of input domains rather than in the middle of input ranges.

Here's how boundary value analysis works:

1. Identify the input conditions or variables that the program uses.
2. Determine the minimum and maximum boundary values for each input condition.
3. Select test cases using these boundary values, focusing on values that are just inside, on, and just outside the boundaries.
4. Test each boundary value to ensure that the program behaves correctly at these critical points.

## 6.4. Data flow testing

Integrating data flow testing into your project can bolster the resilience and dependability of your software. This testing method examines the movement and transformation of data within the application, revealing potential data-related challenges. It verifies data pathways from input to output, confirming accurate data processing and manipulation. By adopting data flow testing, you can pinpoint and rectify data discrepancies, flaws, or weaknesses, elevating the overall quality and trustworthiness of your application.

## 6.5. Unit testing

Unit testing is a crucial component in software development that focuses on testing individual units or components of a software application in isolation. By isolating each unit and testing its functionality independently, unit testing ensures that each part of the software performs as intended. This testing approach helps identify bugs early in the development cycle, making it easier and less costly to fix them. Implementing unit testing in my project can lead to improved code quality, increased developer confidence, and faster development cycles, ultimately contributing to a more robust and reliable software product.

## 6.6. Integration testing

Integration testing is a critical phase in the software testing process that focuses on verifying the interactions between different modules or components of an application. This testing approach ensures that individual units work together as expected when integrated, identifying any inconsistencies or issues that may arise from their interaction. By testing the interfaces and interactions between components, integration testing validates the overall functionality, performance, and reliability of the integrated system. Incorporating integration testing into your project helps ensure seamless communication between modules, reduces integration risks, and enhances the overall quality and stability of your software.

## 6.7. Performance testing

Performance testing is essential for evaluating the speed, responsiveness, and stability of a software application under various load conditions. This testing method assesses the system's performance metrics such as response time, throughput, and resource utilization to identify any bottlenecks or performance issues. By simulating different user loads and stress conditions, performance testing helps ensure that the application can handle the expected workload effectively without degradation in performance. Implementing performance testing in your project allows you to optimize the system's performance, identify potential scalability issues, and deliver a reliable and high-performing software solution to end-users.

## 6.8. Stress Testing

Stress testing assesses the system's resilience and dependability by subjecting it to extreme conditions. This method applies high loads or stress levels to the software to determine its limits and pinpoint any weaknesses or failures. By exceeding the system's typical operational limits, stress testing reveals potential issues like memory leaks, system crashes, or performance bottlenecks under intense strain. Including stress testing in your project helps detect and rectify vulnerabilities, ensuring the application remains stable and delivers reliable performance during peak usage and stressful conditions.

# Chapter 7

# Summary, Conclusion and Future Enhancements

# **Chapter 7:** Summary, Conclusion & Future Enhancements

## 7.1. Project Summary

AI DocQuery project sets out to revolutionize document-based information retrieval through innovative technology and a user-friendly interface. By incorporating ChatGPT APIs and the MERN Stack framework, the platform aims to streamline the user experience and provide tailored, efficient document queries. Testing and evaluation processes are crucial for ensuring the functionality and reliability of the platform, with a focus on use case testing, equivalence partitioning, boundary value analysis, data flow testing, and integration testing. Moving forward, key objectives may include enhancing performance, optimizing user interactions, and implementing additional features such as real-time notifications, advanced search functionalities, and enhanced security measures. Continuous improvement, user feedback integration, and adherence to best practices and coding standards will contribute to the success and scalability of the AI DocQuery platform.

## 7.2. Achievements and Improvements

Here's a list highlighting some achievements and improvements in your project:

1. **User Experience Enhancement:** Introduced intuitive design updates that have boosted user engagement and satisfaction levels.
2. **Performance Fine-Tuning:** Undertook optimizations to achieve quicker response times and elevate the system's overall efficiency.
3. **Feature Enrichment:** Incorporated new functionalities based on user input, enriching the project's scope and user appeal.
4. **Stability and Bug Resolution:** Addressed critical bugs and stability concerns, resulting in a more resilient and dependable application.
5. **Security Reinforcement:** Bolstered security protocols to safeguard user information and deter unauthorized access, fostering a secure user environment.
6. **Scalability Advancement:** Improved the system's scalability to cater to increasing user needs and higher loads without sacrificing performance.

7. **Third-Party Service Integration:** Achieved successful integration with external APIs and services, enhancing the project's versatility and capabilities.

## 7.3. Critical Review

The project has shown commendable progress in terms of user experience and performance optimization. However, one area that requires immediate attention is the offline accessibility of the application. Currently, the application's reliance on an internet connection limits its usability for users in areas with unstable or no internet access.

## 7.4. Lessons Learnt

From this experience, we've recognized the importance of providing offline capabilities to enhance user accessibility and satisfaction. Understanding the diverse needs of our user base is crucial, and adapting the application to cater to offline scenarios will be a priority in future developments.

## 7.5. Future Enhancements/Recommendations

1. **Offline Model Integration:**

   To address the connectivity issues faced by our users, we plan to integrate an offline mode in future updates. This will enable users to access and utilize the application's features without requiring an active internet connection.

2. **File Type Flexibility:**

   Currently, the application supports only PDF uploads. To enhance user flexibility and convenience, we aim to expand this feature to support uploads of various file types, allowing users to upload and manage a broader range of documents seamlessly.