SAMPLE PROJECT REPORT

IMPORTANT NOTICE

This is just a sample project report.
DO NOT SUBMIT THIS DOCUMENT.

It is only for showing how project reports look like.

Note: Front page, certificate of originality, and resume will be added in the original project submissions.

Watermarks Notice: Original projects will NOT contain any kind of watermarks. Watermarks are added only for preventing people from submitting this sample document.

Personal Document Scraper: Solving File Retrieval Issues

Acknowledgement

I would like to express my heartfelt gratitude to my guide, **Mr. Nitin Gupta**, for his invaluable support, insightful guidance, and unwavering encouragement throughout the course of this project, **Personal Document Scraper**. His profound expertise, dedication, and patience have been vital in helping me navigate through the challenges and complexities of this work.

From the inception of the idea to its execution, Mr. Gupta's constructive feedback and continuous motivation pushed me to refine my approach and reach deeper levels of understanding. His insightful suggestions have not only enhanced the quality of this project but have also enriched my knowledge and broadened my perspective on the subject.

I am especially thankful for his willingness to take the time to review my work, offer detailed explanations, and inspire confidence in my abilities. His mentorship went beyond the confines of technical guidance; it was his constant reassurance that motivated me to persevere, even during the most demanding phases of the project.

Completing this project was no small feat, and it would not have been possible without the intellectual stimulation, timely advice, and support provided by Mr. Gupta. His encouragement has left an indelible mark on my academic journey and will serve as a foundation for my future endeavors.

Thank you, **Mr. Nitin Gupta**, for your generous spirit and for being a source of inspiration throughout this process. Your belief in my work has been truly empowering, and I am grateful for the privilege of having you as my guide.

Personal Document Scraper: Solving File Retrieval Issues

August 14, 2025

SYNOPSIS

Table of Contents

Abstract

In today's digital age, individuals accumulate vast amounts of documents and files on their personal computers, ranging from work-related documents to personal photos and everything in between. However, with this accumulation comes the challenge of effectively managing and retrieving these files when needed. All too often, users find themselves struggling to locate specific documents amidst the sea of files stored on their devices, leading to frustration, wasted time, and decreased productivity. The Personal Document Scraper software project aims to address this pressing issue by offering a comprehensive solution to the problem of file retrieval. This project proposes the development of an advanced software application capable of automatically crawling, indexing, and retrieving documents from a user's file system, thereby streamlining the document retrieval process and enhancing user productivity.

1 Introduction

In the era of information overload, managing digital documents has become a significant challenge for individuals and organizations alike. As users accumulate an ever-growing number of files on their personal computers, ranging from text documents and spreadsheets to PDFs and multimedia files, the task of locating specific files when needed becomes increasingly daunting. Traditional methods of file organization, which rely heavily on manual categorization and hierarchical folder structures, often prove inadequate in the face of this digital deluge.

The Personal Document Scraper software project seeks to revolutionize the way users interact with their digital documents by providing a robust solution to the problem of file retrieval. At its core, the Personal Document Scraper is designed to automate the process of document organization and indexing, thereby enabling users to quickly and effortlessly locate the files they need. The software leverages advanced web crawling and text extraction techniques to systematically traverse a user's file system, extract meaningful content from various file formats, and index this content in a powerful search engine.

Imagine a scenario where users no longer need to remember the exact location or filename of a document they are searching for. Instead, they can simply enter a keyword or phrase into a search bar, and within moments, the software retrieves all relevant documents containing that keyword, presenting them in an organized and easily accessible manner. This functionality not only alleviates the frustration and inefficiency associated with manual file searching but also empowers users to reclaim valuable time that would otherwise be spent sifting through countless folders and directories.

The significance of this software cannot be overstated. By automating the tedious task of file organization and indexing, the Personal Document Scraper addresses a critical pain point faced by users in their daily interactions with digital documents. Whether it is retrieving an important work document, locating a cherished family photo, or accessing an old email attachment, the Personal Document Scraper puts the power of instant retrieval at the user's fingertips.

Moreover, the increasing prevalence of remote work and distributed teams has amplified the need for efficient document retrieval solutions. Users often need to access documents quickly and reliably, regardless of their physical location or the device they are using. The Personal Document Scraper is designed with this flexibility in mind, offering seamless compatibility across a wide range of operating systems and devices.

In essence, the Personal Document Scraper represents a paradigm shift in how users manage and retrieve their digital documents. By harnessing the latest advancements in web crawling, text extraction, and search indexing technologies, this software offers a comprehensive and user-friendly solution to the age-old problem of file retrieval.

2 Problem Statement

The exponential growth of digital content has led to significant challenges in file management and retrieval. Individuals and organizations are inundated with a vast array of documents, emails, multimedia files, and other digital content stored across various devices and platforms. This overwhelming volume of data often results in inefficient file management practices, making it difficult for users to locate specific documents when needed.

Traditional methods of file organization, such as hierarchical folder structures and manual categorization, are no longer sufficient to handle the diversity and quantity of digital content. Users frequently encounter frustration and wasted time as they search through cluttered directories and poorly named files. This inefficiency not only affects personal productivity but also impacts business operations, where timely access to information is critical.

Moreover, the lack of systematic organization strategies leads to documents becoming ”lost” within the labyrinth of the file system. Important documents may be buried deep within nested folders, forgotten over time, or mislabeled, further complicating retrieval efforts. This disorganization can have serious consequences, including missed deadlines, impaired decision-making, and reduced collaborative efficiency.

With the rise of remote work, the need for effective document retrieval solutions has become even more pressing. Remote workers require quick and reliable access to documents from any location and device. However, traditional file searching methods are inadequate to meet this demand, leading to delays and decreased overall efficiency.

To address these challenges, there is a clear need for an advanced, user-friendly solution that automates the process of file organization and indexing. The Personal Document Scraper aims to fill this gap by providing a comprehensive software solution that streamlines document retrieval, enhances productivity, and reduces the frustration associated with manual file searching.

3 Objectives

The Personal Document Scraper project aims to address the inefficiencies in file retrieval by developing a comprehensive, automated document management system. The primary objectives of this project are as follows:

3.1 Automate Document Organization

- Develop a document crawling system that can systematically traverse a user's file system to locate and retrieve documents across various formats and directories.
- Implement text extraction algorithms capable of parsing content from different file types, including PDFs, Word documents, spreadsheets, and multimedia files, to extract meaningful, searchable text.

3.2 Enhance Search and Retrieval Efficiency

- Create a powerful indexing mechanism to store extracted text and metadata, enabling rapid and accurate search results.
- Design a user-friendly search interface that allows users to easily enter keywords or phrases and quickly retrieve relevant documents.

3.3 Maintain Data Security and Integrity

- Implement robust security measures to protect user data during the crawling, indexing, and retrieval processes, ensuring user privacy and data integrity.
- Ensure the system is reliable and robust, minimizing the risk of data loss or system failures.

3.4 Provide Integration Capabilities

- Develop APIs for integration with external systems, such as cloud storage providers, to expand the system's functionality and interoperability.
- Ensure seamless interaction with databases for storing metadata and indexing information, facilitating easy maintenance and scalability.

3.5 Adhere to Constraints and Standards

- Design the system within the technical constraints of hardware and software limitations, optimizing resource utilization.
- Ensure the system complies with relevant legal and regulatory requirements concerning data privacy and security.
- Operate within allocated budget constraints, optimizing cost-efficiency without compromising on functionality and performance.

4 Architecture

The Personal Document Scraper project utilizes a client-server architecture to efficiently manage and retrieve digital documents. This architecture ensures scalability, flexibility, and robust performance across various devices and platforms. The key components of the architecture are as follows:

4.1 Client-Server Model

The system is divided into two primary components: the client-side and the server-side. This separation of concerns allows for a modular design, where each component handles specific tasks, enhancing maintainability and scalability.

4.2 Client-Side Components

- **User Interface:** The client-side includes a user-friendly interface through which users can perform searches and retrieve documents. This interface is designed to be intuitive and responsive, providing a seamless user experience across desktops, laptops, and mobile devices.

- **Search Query Handling:** When a user enters a search query, the client application sends this query to the server for processing. The client is responsible for managing user inputs and displaying search results in an organized manner.

4.3 Server-Side Components

- **Web Crawler:** The server hosts the web crawler component, which systematically traverses the user's file system, identifying and retrieving documents. The web crawler is designed to handle various file formats and directories, ensuring comprehensive coverage of the file system.

- **Text Extraction Engine:** This component processes the retrieved documents, extracting searchable text from various file formats such as PDFs, Word documents, spreadsheets, and multimedia files. The text extraction engine uses advanced algorithms to parse and extract meaningful content.

- **Indexing Engine:** Extracted text and metadata are indexed by the indexing engine. This component creates a structured index that facilitates rapid search and retrieval of documents based on user queries.

- **Search Engine:** The search engine processes incoming queries from the client, searching the index for relevant documents. It uses efficient search algorithms to ensure quick and accurate retrieval of results.

- **APIs and Integration:** The server provides APIs for integration with external systems, such as cloud storage providers. This allows the system to expand its functionality and interact with other services seamlessly.

- **Security and Data Protection:** Robust security measures are implemented on the server-side to protect user data during the crawling, indexing, and retrieval processes. This includes encryption, authentication, and access control mechanisms.

4.4 Communication Protocols

- **HTTP/HTTPS:** The communication between the client and server components primarily uses HTTP/HTTPS protocols. This ensures secure and reliable data transmission over the network.
- **RESTful APIs:** The system employs RESTful APIs for interaction between client and server, allowing for flexible and scalable integration with other systems and services.

4.5 Database Management

- **Metadata Storage:** The system uses a database to store metadata associated with each document, such as file names, sizes, types, and last modified timestamps. This metadata is crucial for efficient indexing and retrieval.
- **Index Storage:** The indexes created by the indexing engine are stored in a high-performance database, optimized for rapid search and retrieval operations.

4.6 Scalability and Performance

The architecture is designed to be scalable, allowing the system to handle increasing volumes of documents and user queries without compromising performance. The use of efficient algorithms, robust indexing techniques, and scalable storage solutions ensures that the system remains responsive and reliable as the user base grows.

4.7 Conclusion

The client-server architecture of the Personal Document Scraper project provides a solid foundation for efficient document management and retrieval. By separating the client and server responsibilities, the system achieves a modular and scalable design that enhances user experience, performance, and maintainability.

5 Software and Tools Used

The Personal Document Scraper project utilizes various software and tools to implement its functionality effectively. Below is a list of the key components along with the rationale behind their selection:

- **Programming Language:** Rust

  Reasons for Selection: Rust is ideal for this project due to its memory safety guarantees and high performance, ensuring reliable and efficient handling of concurrent tasks like web crawling and indexing. Additionally, its strong type system and minimal runtime make it suitable for building a robust backend system.

- **Search Engine:** MeiliSearch

  Reasons for Selection: MeiliSearch is selected for its speed, relevance, and ease of integration. It provides powerful full-text search capabilities, enabling users to quickly retrieve relevant documents based on their queries. MeiliSearch's simple RESTful API and robust indexing mechanism make it a suitable choice for implementing the search functionality in the Personal Document Scraper.

- **Text Extraction Library:** Apache Tika

  Reasons for Selection: Apache Tika is a comprehensive text extraction library that supports a wide range of document formats, including images, audio files, and rich text formats. Its versatility and extensive format support make it an ideal choice for handling diverse document types within the Personal Document Scraper project.

- **Database Management System:** SQLite

  Reasons for Selection: SQLite is chosen as the secondary database management system for its lightweight nature, simplicity, and ease of integration. It is enough to keep the configuration and users details, status of documents, filepath, last modifies time, crawled or not, indexed or not. It offers a self-contained, serverless, zero-configuration database engine, SQLite's reliability, ACID compliance, and wide adoption make it a pragmatic choice for managing the project's data storage requirements. It is enough to keep the status of documents, Filepath, last modifies, crawled or not, indexed or not.

6 Platform and Specifications

The Personal Document Scraper project is designed to be compatible with the following platforms and specifications:

6.1 Platform Compatibility

The project can be run on the following platforms:

- **Linux:** The Personal Document Scraper is fully compatible with Linux distributions, offering native support and optimal performance.
- **Windows with WSL:** While the project is primarily developed for Linux environments, it can also be run on Windows using the Windows Subsystem for Linux (WSL). This allows users to seamlessly execute Linux-based applications on Windows without the need for virtual machines or dual-boot setups.

6.2 Hardware Requirements

To run the Personal Document Scraper efficiently, the following hardware specifications are recommended:

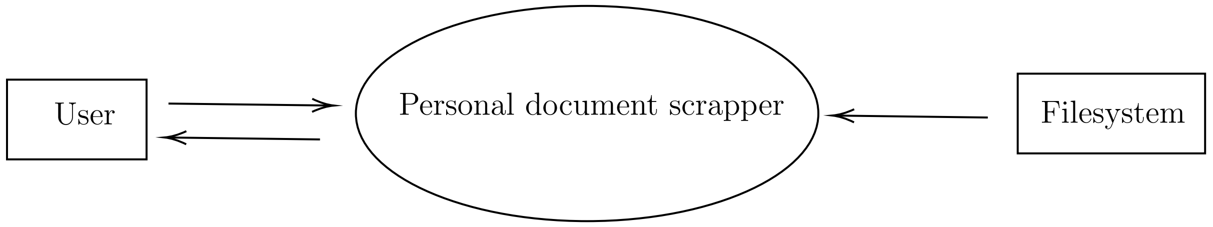
- **RAM:** Minimum 2 GB of RAM is required to ensure smooth operation of the application.
- **Processor:** A processor with a minimum of 2 cores and a clock speed of 2 GHz is recommended to handle the computational tasks involved in document crawling, text extraction, and indexing operations efficiently.

These platform and hardware specifications ensure optimal performance and compatibility of the Personal Document Scraper across different environments and configurations.

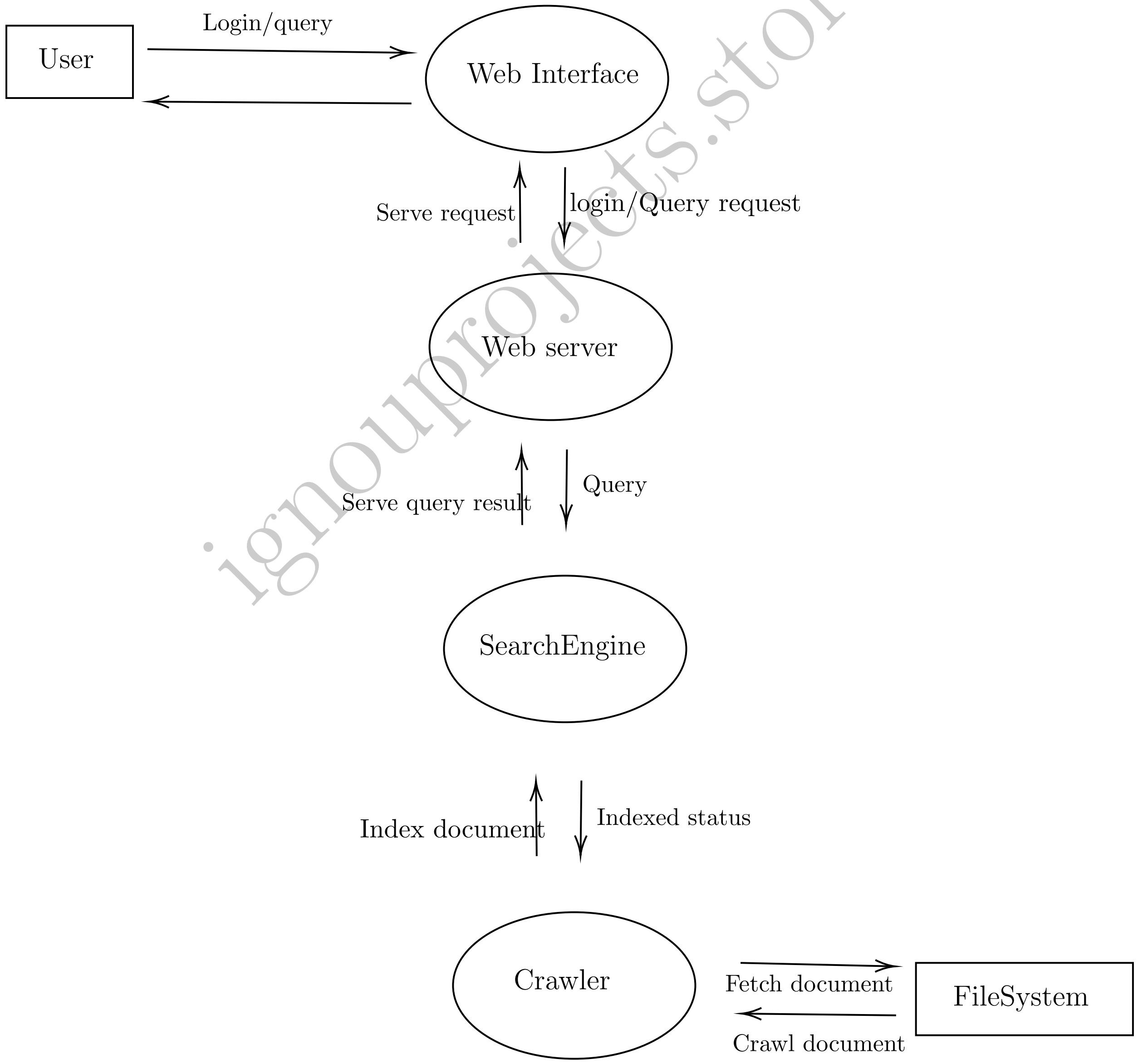
7 Data Flow Diagram (DFD)

The Data Flow Diagram (DFD) below illustrates the flow of data within the Personal Document Scraper system, depicting the processes involved in handling user queries, document crawling, text extraction, indexing, and retrieving documents.

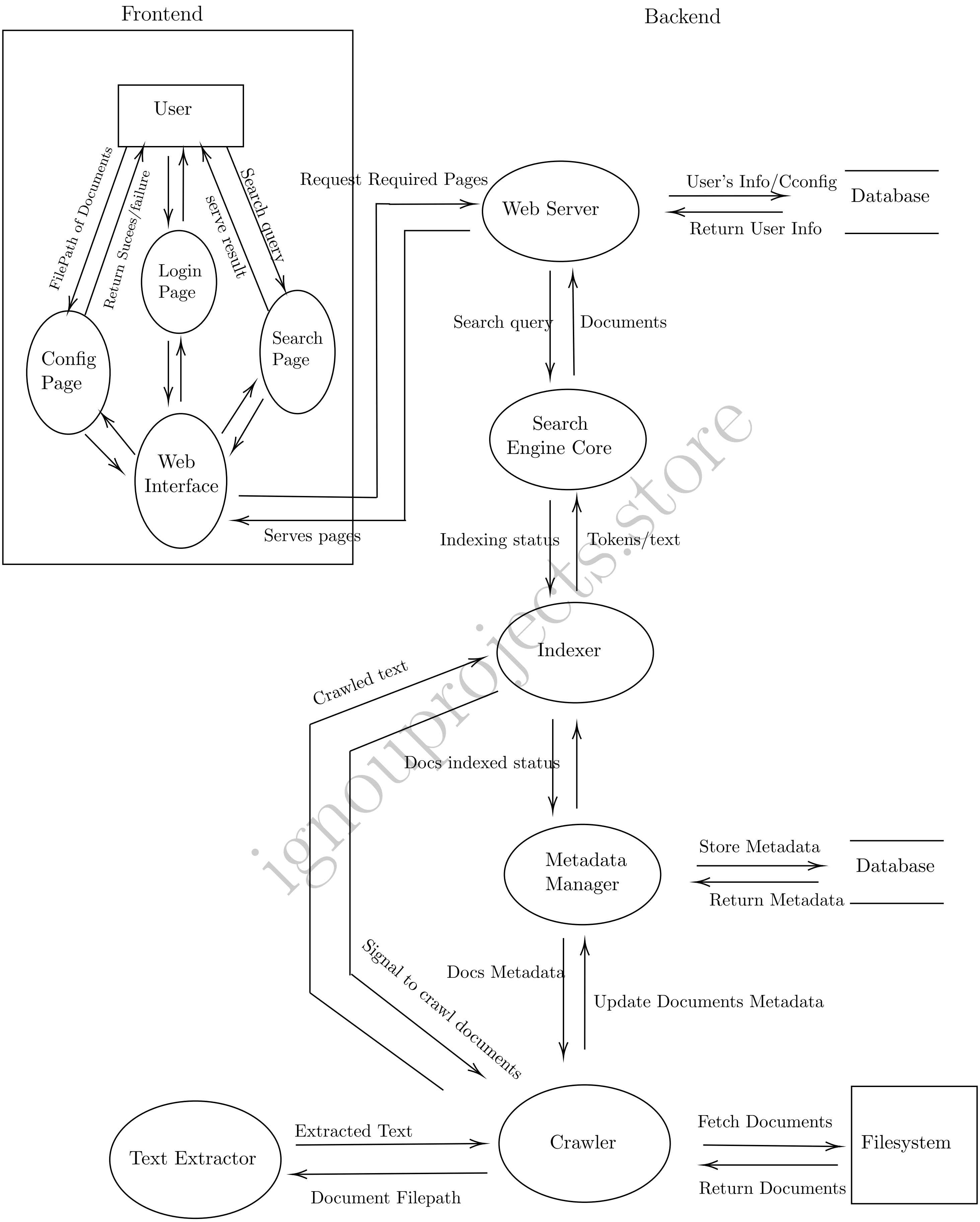
7.1 Level 0 (Context Diagram)



7.2 Level 1 DFD



7.3 Level 2 DFD



8 Data structures and Class Diagram

Object: User

Attribute	**Type**	**Notes**
UserID	INT	Primary key
Username	VARCHAR	User's username
PasswordHash	VARCHAR	Hashed password for security
Email	VARCHAR	User's email address
CreatedAt	DATETIME	Timestamp when user account was created
UpdatedAt	DATETIME	Timestamp when user account was last updated

Table 1: User Object Schema

Object: Query

Attribute	**Type**	**Notes**
QueryID	INT	Primary key
QueryKeyword	VARCHAR	Keyword or phrase used in the query

Table 2: Query Object Schema

Object: Document

Attribute	**Type**	**Notes**
DocumentID	INT	Primary key for the document
Name	VARCHAR	Name of the document
Path	VARCHAR	Path to the document location
FileType	VARCHAR	Type of the document file
LastModified	DATETIME	Last modified timestamp of the document
IsCrawled	BOOLEAN	Indicates whether the document has been crawled
IsIndexed	BOOLEAN	Indicates whether the document has been indexed
ExtractedText	CHAR[]	Text extracted from the document (if applicable)

Table 3: Document Object Schema

Object: QueryResult

Attribute	Type	Notes
QueryID	INT	Foreign key referencing the Query object
QueryResult	VARCHAR	Result of the query execution

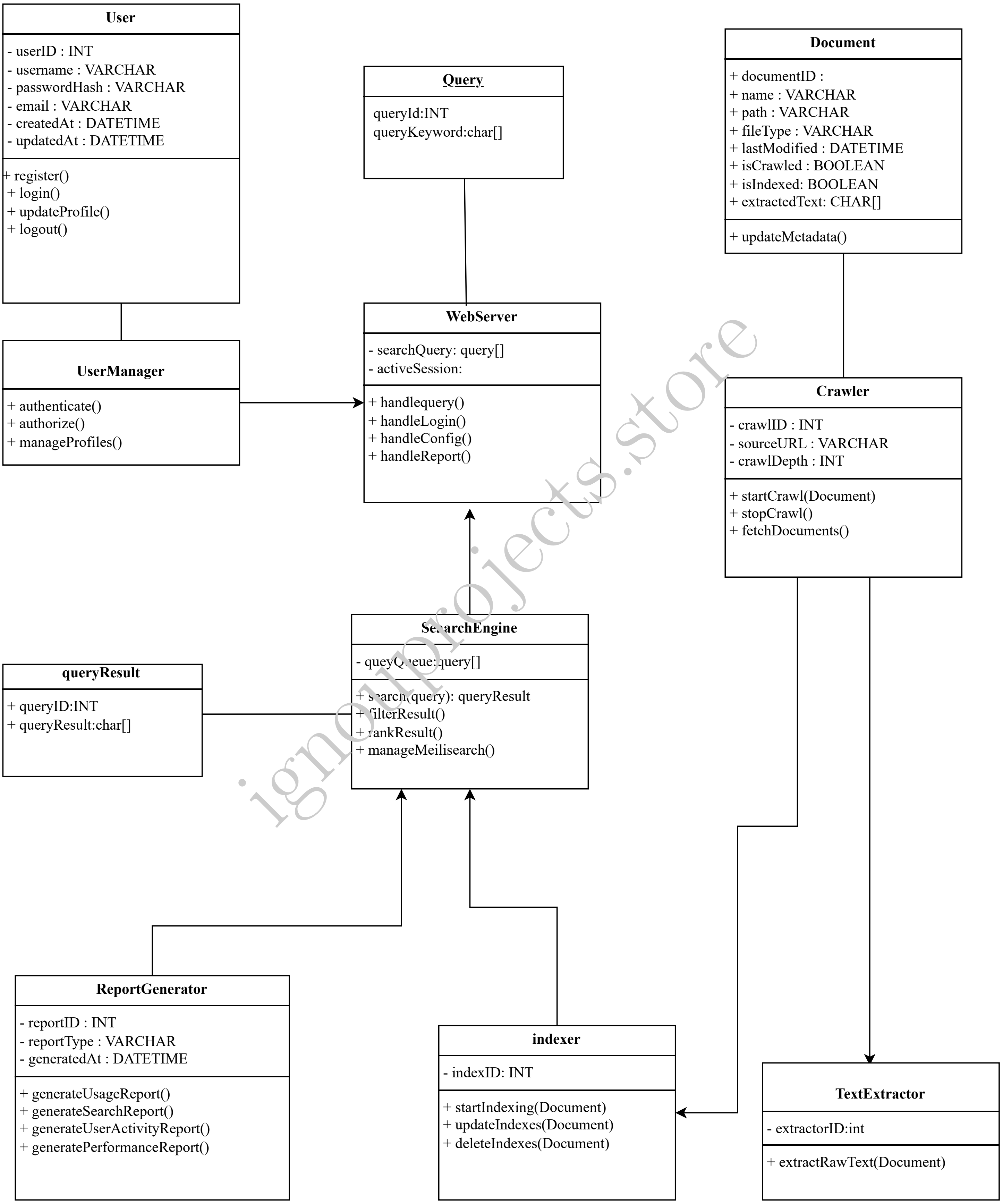
Table 4: QueryResult Object Schema

Object: Report

Attribute	Type	Notes
ReportID	INT	Unique identifier for the report
ReportType	VARCHAR	Type of the report
GeneratedAt	DATETIME	Timestamp when the report was generated
TotalUsers	INT	Total number of users in the system
ActiveUsers	INT	Number of active users during the reporting period
TotalQueries	INT	Total number of queries made
MostSearchedKeyword	VARCHAR	The most frequently searched keyword
AverageQueryTimeMS	INT	Average time taken to serve queries in milliseconds
TotalIndexedDocuments	INT	Total number of documents that have been indexed
TotalCrawledDocuments	INT	Total number of documents that have been crawled
PeakUsageTime	DATETIME	Time of day when the system experiences peak usage
ErrorRate	FLOAT	Percentage of queries that resulted in errors
AverageDocumentsPerUser	FLOAT	Average number of documents accessed per user

Table 5: UsageReport Object Schema

Class Diagram:



9 Modules and their Description

The Personal Document Scraper project consists of the following modules:

1. User Management Module
2. Crawling Module
3. Text Extraction Module
4. Indexing Module
5. Search Module
6. Metadata Module
7. Reporting Module

The Personal Document Scraper project is organized into several modules, each responsible for specific functionalities. Below are the descriptions of these modules:

9.1 User Management Module

The User Management module handles user authentication, authorization, and user profile management. It allows users to register, log in, and manage their profiles. This module ensures secure access to the system and assigns appropriate roles and permissions to users.

9.2 Crawling Module

The Crawling module is responsible for fetching documents from various sources, such as file systems, websites, or external repositories. It employs web crawling techniques to systematically navigate through web pages and retrieve documents for processing.

9.3 Text Extraction Module

The Text Extraction module extracts text content from different types of documents, including PDFs, Word documents, PowerPoint presentations, and other common file formats. It utilizes libraries such as Apache Tika to parse and extract text from documents accurately.

9.4 Indexing Module

The Indexing module indexes the extracted text content to facilitate efficient and fast search operations. It builds an index of keywords and their corresponding document references, enabling users to quickly retrieve relevant documents based on their search queries.

9.5 Search Module

The Search module provides a user-friendly interface for users to search for documents based on keywords or phrases. It utilizes the indexed data to perform fast and accurate searches, returning relevant documents matching the user's query.

9.6 Metadata Module

This module is primarily utilized by the Indexing Module and Crawling Module. It ensures that metadata is accurately updated and managed. The module plays a crucial role in maintaining the integrity and relevance of document metadata. This facilitates efficient indexing and crawling operations.

9.7 Reporting Module

The Reporting module generates various reports and analytics based on user interactions and system activities. It provides insights into document usage, search patterns, user behaviors, and other relevant metrics, helping administrators make informed decisions and optimize system performance.

These modules collectively form the backbone of the Personal Document Scraper system, enabling efficient document retrieval, text analysis, and search capabilities for users.

10 Types of Report Generation

The Reporting module in the Personal Document Scraper project supports various types of report generation to provide insights into document usage, search patterns, user behaviors, and system activities. Below are the types of reports that can be generated:

10.1 Usage Reports

Usage reports provide statistical data on the usage of documents within the system. These reports include metrics such as the number of documents accessed, the frequency of document access, and user engagement metrics.

10.2 Search Reports

Search reports analyze search patterns and trends within the system. They provide information on the most commonly searched keywords, popular search queries, and the effectiveness of search results in meeting user needs.

10.3 User Activity Reports

User activity reports track the actions and behaviors of users within the system. They include information on user login/logout activities, document downloads, search history, and other user interactions.

10.4 Performance Reports

Performance reports evaluate the performance of the Personal Document Scraper system. They include metrics such as system uptime, response times, indexing speed, and overall system efficiency.

11 Validations

The Personal Document Scraper project implements various validations to ensure data integrity, security, and usability. Below are the validations performed within the system:

11.1 User Authentication and Authorization

Validation of user credentials during login to ensure only authorized users gain access to the system. Role-based access control (RBAC) validations to enforce appropriate permissions and access levels for different user roles.

11.2 Document Integrity

Validation of document integrity during the crawling and extraction process to ensure accurate retrieval and parsing of document content. Verification of document metadata and checksums to detect any tampering or corruption.

11.3 Search Query Validation

Validation of search queries to ensure they comply with the specified format and criteria. Filtering and sanitization of search inputs to prevent unintended or malicious search behavior.

11.4 Indexing Consistency

Validation of indexed data to maintain consistency and accuracy in search results. Regular checks and updates to the index to reflect changes in document content or metadata.

11.5 Data Backup and Recovery

Validation of data backup and recovery processes to ensure data reliability and availability. Regular backups and restoration tests to verify the integrity and effectiveness of backup mechanisms.

11.6 Access Control

Validation of access control mechanisms to prevent unauthorized access to sensitive data and system resources. Regular audits and reviews of access control policies and configurations.

11.7 Error Handling

Validation of error handling mechanisms to gracefully handle unexpected errors and exceptions. Logging and monitoring of error messages to facilitate troubleshooting and system diagnostics.

These validations help maintain the integrity, security, and usability of the Personal Document Scraper system, ensuring reliable operation and protection of sensitive data.

12 Limitations

While the Personal Document Scraper project offers valuable features and functionalities, it also has certain limitations that users and administrators should be aware of. Below are the limitations of the project:

12.1 Platform Compatibility

The project is primarily developed for Linux environments and may have limited compatibility with other operating systems. While it can be used on Windows with the Windows Subsystem for Linux (WSL), full functionality and performance may not be guaranteed on non-Linux platforms.

12.2 File Format Support

The project may have limitations in supporting certain file formats for document crawling and text extraction. While efforts are made to support a wide range of formats, compatibility with specialized or proprietary formats may be limited.

12.3 Performance Scalability

The performance of the system may degrade when handling a large volume of documents or concurrent user requests. Scalability issues may arise under heavy load conditions, impacting response times and system throughput.

12.4 Resource Requirements

The project may require significant computational resources, such as memory, CPU, and storage, especially when processing and indexing large documents or datasets. Limited hardware resources may affect system performance and responsiveness.

13 Future Scope and Features

The Personal Document Scraper project has significant potential for expansion and enhancement to offer more advanced functionalities and improve user experience. Below are some future scope areas and features that could be considered for addition:

13.1 AI-Based Text Processing

Integrate artificial intelligence (AI) and natural language processing (NLP) techniques to enhance text extraction and indexing capabilities. Implement advanced algorithms for semantic analysis, entity recognition, and context-aware search to provide more accurate and relevant search results.

13.2 Advanced Search Filters

Enhance the search functionality with advanced filters and facets to refine search results based on metadata attributes, document types, or user-defined criteria. Implement dynamic filtering options and faceted navigation to facilitate precise and targeted document retrieval.

13.3 Personalized Recommendations

Utilize machine learning algorithms to analyze user behavior and preferences and provide personalized document recommendations. Implement recommendation systems that suggest relevant documents based on user interactions, search history, and content preferences.

13.4 Collaborative Document Management

Introduce collaborative document management features to enable users to collaborate on document creation, editing, and sharing. Implement version control, document commenting, and real-time collaboration features to streamline document workflows and enhance productivity.

13.5 Mobile and Cross-Platform Compatibility

Develop mobile applications and ensure cross-platform compatibility to enable users to access and interact with the system from various devices and platforms. Implement responsive design principles and native mobile app development for seamless user experience across devices.

13.6 Integration with External Systems

Integrate with external systems and third-party services to extend the functionality and interoperability of the Personal Document Scraper project. Implement APIs, webhooks, and data connectors to facilitate seamless data exchange and integration with other enterprise systems.

14 Bibliography

14.1 Online Resources

- **The Rust Reference**: Rust reference documentation. Available at: https://doc.rust-lang.org/stable/reference/

- **MeiliSearch API Documentation**: MeiliSearch RESTful API documentation. Available at: https://docs.meilisearch.com/

- **SQLite Documentation**: SQLite official documentation. Available at: https://www.sqlite.org/docs.html

- **Apache Tika Documentation**: Apache Tika documentation. Available at: https://tika.apache.org/documentation.html

- **Stack Overflow**: Online community for programming-related QA. Available at: https://stackoverflow.com/

These resources provide valuable information and insights relevant to the development and implementation of the Personal Document Scraper project.

PROJECT REPORT

Table of Contents

1 Introduction

In today's digital age, individuals and organizations generate and accumulate a vast and diverse array of documents, including PDFs, Word files, spreadsheets, images, and plain text notes. This growing collection often spans years, stored across multiple devices and cloud services. Despite advancements in cloud storage and document management systems, accessing specific information from personal archives remains a daunting challenge for most users. Unlike the web, which benefits from powerful search engines such as Google or Bing, personal document collections often lack efficient and intuitive tools for comprehensive search and retrieval. This is the gap that the Personal Document Scraper project aims to fill.

The Personal Document Scraper is a groundbreaking initiative designed to empower users to efficiently organize, search, and retrieve information from their local document repositories. By leveraging cutting-edge technologies, such as Apache TIKA for robust file scraping and Meilisearch for high-performance indexing, this project brings the sophistication of web-scale search engines to personal document collections. Apache TIKA facilitates the extraction of metadata and text content from a wide range of document formats, while Meilisearch ensures rapid indexing and query processing to deliver near-instant search results. Together, these tools form the backbone of a solution that transforms fragmented and inaccessible document repositories into highly searchable and structured knowledge bases.

One of the key features of the Personal Document Scraper is its focus on user-centric design and accessibility. The tool is designed to provide an intuitive interface that requires minimal technical expertise, making it accessible to a broad range of users. Advanced search capabilities, such as keyword-based searches, phrase matching, and filters for metadata (e.g., date, author, file type), allow users to pinpoint relevant documents with precision. The system is also optimized for performance, ensuring that even large-scale personal document repositories can be indexed and searched efficiently.

Beyond search functionality, the Personal Document Scraper emphasizes privacy and control—critical aspects that differentiate it from traditional web scraping and cloud-based solutions. Unlike cloud services, which may expose sensitive data to external servers, the Personal Document Scraper operates locally on the user's device, ensuring that documents remain secure and private. This local-first approach aligns with the growing demand for tools that respect user privacy while delivering robust functionality.

Additionally, the project addresses the need for interoperability and extensibility. By supporting a wide range of document formats and offering integration with existing file storage systems, the tool ensures seamless integration into diverse workflows. Future iterations of the project may include features such as real-time indexing, collaborative search capabilities, and AI-driven recommendations for related documents.

In summary, the Personal Document Scraper project is a transformative solution that bridges the gap between traditional document management and modern search engine technology. By enabling users to efficiently search, retrieve, and organize their personal documents, the project addresses critical pain points in personal information management. With its emphasis on privacy, performance, and user-centric design, the Personal Document Scraper has the potential to redefine how individuals and organizations interact with their personal archives, ultimately empowering them with better control and accessibility over their digital knowledge.

2 Objectives

The primary objective of the **Personal Document Scraper** project is to develop an efficient and user-friendly tool for searching personal document collections. Specifically, the project aims to:

1. **Provide a Local Search Solution**: Enable users to seamlessly search through their personal archives, such as PDFs, Word files, and text notes, mimicking the functionality of popular web search engines.

2. **Leverage Advanced Technologies**: Use Apache TIKA for file scraping and Meilisearch for fast indexing, ensuring that users can quickly and accurately search through their document collections.

3. **Ensure Privacy and Control**: Differentiate personal document scraping from traditional web scraping by emphasizing privacy, data security, and the user's control over their local information.

4. **Improve Accessibility and Organization**: Enhance the management and retrieval of personal documents by offering an intuitive search interface that organizes documents and makes them easily searchable.

5. **Provide Better Information Retrieval**: Empower users to access their documents quickly and efficiently, improving overall productivity and information management.

By achieving these objectives, the Personal Document Scraper will facilitate enhanced search capabilities for personal documents and contribute to more effective personal information management.

3 About Technologies and Tools

The **Personal Document Scraper** project utilizes a combination of powerful technologies and tools to ensure efficient document scraping, fast searching, and smooth project management. Below is a list of the key technologies and tools used in the development of this project:

- **Rust**
- **Meilisearch**
- **Apache TIKA**
- **Git**
- **GitHub**
- **CI/CD**

Now, let's discuss each of these tools in detail:

1. **Rust**: Rust is the core programming language used in this project. It is known for its performance, memory safety, and concurrency. Rust was chosen because it offers a high degree of control over system resources, which is essential when managing large volumes of data. Additionally, its strong type system helps prevent errors during compilation, leading to safer and more reliable code. Rust's memory management is done without a garbage collector, making it ideal for projects that require efficiency and high-performance computing, such as handling large document collections.

2. **Meilisearch**: Meilisearch is an open-source, fast, and typo-tolerant search engine that excels in providing an exceptional search experience. It was selected for its ability to deliver rapid indexing and searching of large document datasets, making it an ideal choice for the Personal Document Scraper project. Meilisearch is lightweight, easy to integrate, and optimized for full-text search, allowing for quick retrieval of relevant information.

   One of the standout features of Meilisearch is its ability to handle complex queries with ease, including support for filters, sorting, and faceting, which enhances the user experience by allowing users to refine their search results effectively. Additionally, Meilisearch is designed to be highly customizable, enabling developers to tailor the search experience to meet specific user needs.

   Its speed and relevance in search results make it perfect for indexing and querying personal document collections, ensuring that users can find what they need without delay. The choice of Meilisearch was driven by its performance, simplicity, and the flexibility it provides for handling various types of documents, as well as its active community and ongoing development, which ensure that it remains a cutting-edge solution for search functionality.

3. **Apache TIKA**: Apache TIKA is a versatile content extraction toolkit designed for scraping and parsing a wide range of document formats, such as PDFs, Word files, and plain text documents. TIKA was chosen for its extensive format support and its ability to efficiently extract both text and metadata. This tool guarantees that all documents within the user's archive are accurately processed, enabling effective indexing and searching. It streamlines the intricate task of managing diverse file types, ensuring a seamless user experience.

4. **Git**: Git is a distributed version control system that allows for efficient tracking of changes in the project codebase. It was chosen because of its speed, flexibility, and reliability in managing large projects. Git enables collaborative work, tracks history, and allows easy branching and merging of code. This is particularly important for projects that involve multiple developers or frequent changes to the code. Git's widespread adoption in the development community made it the clear choice for this project.

5. **GitHub**: GitHub is a cloud-based platform used for hosting Git repositories, enabling collaborative development, version control, and issue tracking. It provides a centralized space where the project code can be stored, reviewed, and shared. GitHub also offers tools for collaboration, such as pull requests and code reviews, which are essential for maintaining code quality. The decision to use GitHub was based on its powerful collaboration features, integration with CI/CD tools, and its ability to manage large codebases with ease.

6. **CI/CD**: Continuous Integration/Continuous Deployment (CI/CD) tools are used to automate the process of testing, building, and deploying the project. CI/CD pipelines ensure that any changes made to the code are automatically tested, built, and deployed without manual intervention. This approach improves the quality and stability of the code while allowing for rapid iterations and faster deployment of new features or bug fixes. The use of CI/CD tools in this project was aimed at improving the overall development workflow, ensuring high-quality code, and minimizing human errors during deployment.

By leveraging these technologies and tools, the Personal Document Scraper project ensures efficient development, robust performance, and effective document management, providing a powerful solution for personal document search and retrieval.

4 System Analysis

System analysis is a fundamental process that helps in comprehending, designing, and improving the systems. It involves examining various aspects of a system to ensure its functionality, performance, and alignment with user requirements. The different types of system analysis methods contribute uniquely to the overall process, and these are outlined below:

1. **Requirements Analysis:** This phase is centered around gathering and documenting the user's needs and expectations for the system. It typically involves discussions with stakeholders, user surveys, and careful observation to understand the primary objectives the system must achieve.

2. **Feasibility Study:** A feasibility study investigates the viability of a proposed system, focusing on technical, economic, operational, and legal factors. This analysis helps determine whether the project is worth pursuing and if it can be successfully implemented.

3. **Data Analysis:** This method focuses on understanding the data within a system. It involves analyzing how data is collected, processed, and utilized to ensure the data structure is efficient and meets the system's goals.

4. **Structured Systems Analysis:** Structured techniques such as Data Flow Diagrams (DFDs) and Entity-Relationship Diagrams (ERDs) are employed in this analysis to model the data flow and interactions between system components, ensuring a detailed understanding of system processes.

5. **User Interface (UI) Analysis:** UI analysis emphasizes the design and usability of the system's interface. It ensures that the system is easy to use, intuitive, and provides a seamless experience for the user.

6. **Cost-Benefit Analysis:** A cost-benefit analysis evaluates the financial implications of a system. This includes assessing the cost of development, maintenance, and operation, and comparing it with the expected benefits or returns the system will provide.

Project Overview

The **Personal Document Scraper** project is aimed at developing a tool that can efficiently scrape, index, and search personal documents stored locally on a user's system. It intends to replicate the search functionality seen in web engines like Google, but for documents stored on a personal machine, making it easier for users to organize and retrieve information from their archives.

Key components of the project include:

- **Document Scraping:** The system will extract content from various document formats, such as PDFs, DOCX, and text files, using advanced content extraction tools.

- **Indexing:** Extracted data is indexed to enable quick searches and easy retrieval of relevant information.

- **Search Interface:** A simple and user-friendly interface will allow users to search their document collection efficiently.

The system will also ensure that user data remains secure and private, with appropriate encryption and access control measures.

4.1 Software Development Life Cycle (SDLC)

The Software Development Life Cycle (SDLC) is a systematic, structured approach to software development that defines the stages involved in the development of software from its initial planning phase through to its delivery and maintenance. The SDLC provides a methodical framework for planning, designing, developing, testing, and maintaining software, ensuring that each stage is completed before moving on to the next. This approach helps in reducing risks, improving quality, and managing project timelines and budgets effectively.

The SDLC consists of several key phases, each with specific objectives:

1. **Requirement Gathering and Analysis:** This initial phase involves collecting requirements from stakeholders, users, and other relevant parties. The goal is to understand the needs and expectations for the software. Detailed analysis is performed to identify functional and non-functional requirements, which form the basis for designing the system.

2. **System Design:** In the system design phase, the software architecture is defined, and the design is structured. This includes designing both high-level architecture and low-level components. The design phase covers database structure, user interfaces, and system interfaces. The aim is to create a clear blueprint for the development team to follow during the coding phase.

3. **Implementation (Coding):** The implementation phase is where the actual coding begins. The design documents created in the previous phase are translated into a working system by developers. The software is built according to the defined specifications, and individual modules or components are developed and integrated.

4. **Testing:** After the system is built, it undergoes rigorous testing to identify and fix any issues or bugs. This phase includes unit testing, integration testing, system testing, and user acceptance testing (UAT). The goal is to ensure that the system meets the specified requirements and functions correctly in all scenarios.

5. **Deployment:** Once the software is tested and deemed ready for release, it is deployed into the production environment. This phase involves setting up the system on user machines or servers and making it available to end users. Deployment may occur in stages, depending on the complexity and size of the software.

6. **Maintenance:** After the software is deployed, it enters the maintenance phase, where it is monitored for performance, updated to address any new issues or user feedback, and improved over time. This phase ensures that the system remains relevant, secure, and operational throughout its life cycle.

The SDLC offers several benefits, such as structured project management, clear documentation, and improved communication among stakeholders. It helps ensure that software is developed efficiently, meets the user's needs, and is of high quality. Additionally, it allows for risk management through careful planning and testing before deployment. By following these well-defined stages, the SDLC reduces the chances of errors and miscommunication, making the development process more predictable and manageable.

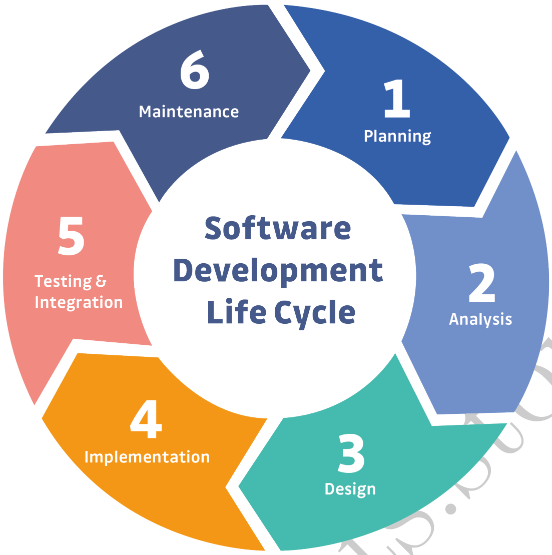


Figure 1: The Software Developement Cycle

4.2 System Analysis: Personal Document Scrapper

Introduction

System analysis is a cornerstone of software development, providing the foundation for building a robust, efficient, and user-centric application. For the **Personal Document Scrapper** project, this phase involves understanding the requirements, designing the system structure, and ensuring alignment with user needs. The objective is to create a **web interface** that allows users to securely log in and search their personal document collections with ease, functioning like a personalized search engine. By leveraging modern technologies, the system aims to deliver an intuitive and seamless document-searching experience.

This document explores the comprehensive analysis, requirements, design, and data flow aspects of the system, offering a detailed perspective on the project's execution plan.

Project Overview

- **Project Name:** Personal Document Scrapper
- **Objective:** Develop a powerful and efficient tool to search personal document collections stored locally.
- **Tech Stack:**
  - **Programming Language:** Rust
  - **Tools:** Apache TIKA for document parsing, Meilisearch for indexing and searching
- **Deployment:** Web-based interface accessible via a browser.
- **Database:** Metadata storage with indexing by Meilisearch.
- **Target Users:** Individuals seeking efficient, secure, and intuitive management of personal documents.

Importance of System Analysis in the Project

System analysis ensures that the application meets the expectations of its end users and operates effectively in real-world scenarios. Here are the core reasons why system analysis is critical to this project:

1. **Understanding Requirements:** A deep dive into the user's needs ensures that the features designed align with expectations, including secure login capabilities and efficient search across large collections of documents.
2. **Establishing Feasibility:** Analyzing whether the chosen technologies (Rust, Meilisearch, and Apache TIKA) can effectively meet the functional and performance requirements.
3. **Structured Design:** Designing a well-thought-out architecture avoids future reworks and ensures scalability. This includes decisions about the client-server model, document crawling processes, and search indexing.

4. **Mitigating Risks:** Proactively identifying challenges, such as security vulnerabilities or performance bottlenecks, minimizes project delays and enhances user trust.

User Requirements

The **user requirements** capture what functionalities users expect from the system. These include:

- **User Authentication:** Users can securely sign up, log in, and manage their credentials. Multi-factor authentication ensures robust security.
- **Document Search:** Ability to search using keywords, phrases, or file names. Results should be displayed with relevance and speed.
- **Document Management:** Automatic scraping of documents from the user's host device. Options for organizing, deleting, or categorizing documents.
- **Performance and Speed:** Deliver fast search results, even with large document repositories.
- **Support for Multiple Formats:** Compatible with popular formats such as PDF, DOCX, and plain text files.
- **User-Friendly Interface:** An intuitive and responsive design ensures accessibility for non-technical users.
- **Data Security:** Robust encryption protocols safeguard user data.

System Architecture

The **Personal Document Scrapper** follows a client-server architecture with the following components:

- **Frontend (Client-Side):** Built with **HTML**, **CSS**, and **JavaScript**, ensuring responsiveness. Provides secure login/logout functionality, a search bar for document queries, and interfaces for managing and configuring document settings.
- **Backend (Server-Side):** Developed in **Rust** to ensure high performance and low memory overhead. Responsibilities include user authentication, communication with the database and search engine, and document scraping and indexing using Apache TIKA.
- **Database:** Stores user credentials (securely encrypted) and metadata about documents (e.g., titles, formats).
- **Search Engine (Meilisearch):** Indexes document content for efficient keyword-based search.
- **Document Scraping:** Apache TIKA parses and extracts content from supported file formats for indexing.
- **Deployment Environment:** Hosted in the cloud for scalability. Deployed using **Docker** to simplify scaling and management

- **User Data:** Includes username, password, email. Stored securely with encrypted storage and hashed passwords.
- **Document Metadata:** Includes document title, format, size, and location. Stored securely for search indexing.
- **Search Index:** Built by Meilisearch, containing extracted keywords and phrases. Enables quick lookup for user queries.

4.3 Data Flow Diagram (DFD)

Data Flow Diagrams (DFDs) are a vital tool for visualizing and analyzing the flow of information within a system. They help in understanding how data moves between processes, data stores, and external entities, thereby facilitating effective system design and communication among stakeholders. A DFD provides a graphical representation of the system, breaking down complex processes into manageable components and highlighting the logical flow of data.

Components of a DFD

A DFD primarily consists of the following components:

- **Processes:** Represented by circles or rounded rectangles, processes depict activities that transform input data into output.
- **Data Stores:** Depicted by open rectangles, these are repositories where data is stored and retrieved.
- **Data Flows:** Shown as arrows, they indicate the movement of data between entities, processes, and data stores.
- **External Entities:** Represented by rectangles, these are sources or destinations of data outside the system boundary.

Significance of DFDs

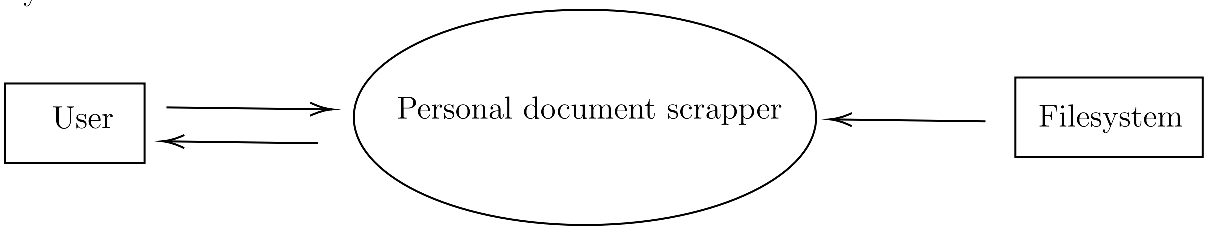
DFDs are significant because they:

- Simplify complex systems by breaking them into smaller, more comprehensible parts.
- Provide a clear, visual representation of data processes and flows, aiding communication among developers, clients, and stakeholders.
- Serve as a blueprint for system development, ensuring consistency in design and implementation.

The following subsections detail the different levels of the Data Flow Diagram for the Personal Document Scraper system.

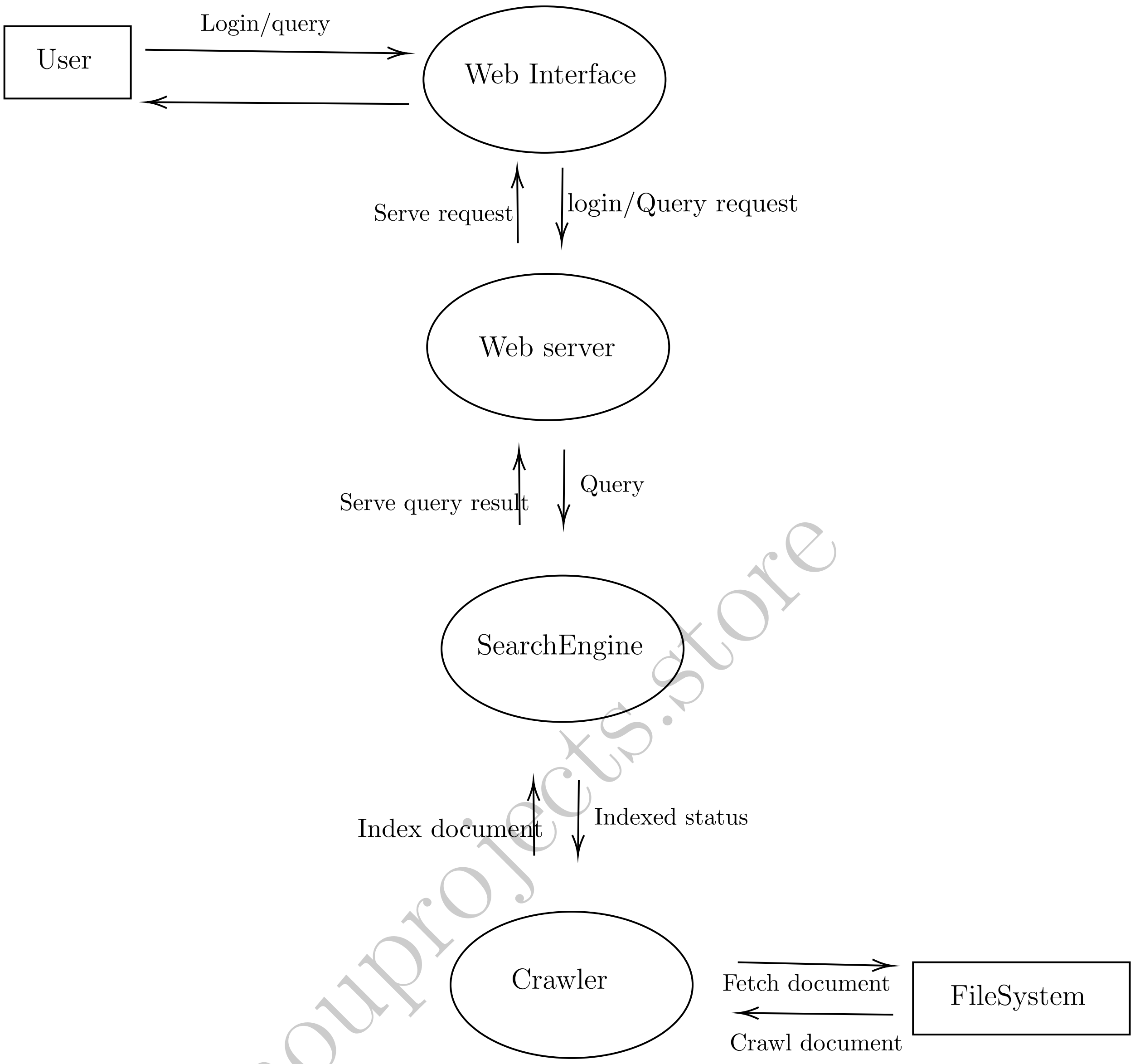
Level 0 (Context Diagram)

The Level 0 DFD, or Context Diagram, provides a high-level view of the Personal Document Scraper system. It represents the entire system as a single process and identifies external entities interacting with the system. The diagram shows the flow of data between the system and its environment.



Level 1 DFD

The Level 1 DFD breaks down the primary process into its main sub-processes, detailing the key functions of the system. This level illustrates how the system handles user queries, document crawling, text extraction, indexing, and document retrieval, while highlighting the interaction between processes and data stores.


Login/query
User
Web Interface
Serve request
login/Query request
Web server
Serve query result
Query
SearchEngine
Index document
Indexed status
Crawler
Fetch document
FileSystem
Crawl document

Level 2 DFD

The Level 2 DFD provides a granular breakdown of the processes identified in Level 1. It delves deeper into the functionalities and sub-processes within the Personal Document Scraper system. By expanding on the major processes, this level offers a comprehensive view of the data flow, internal interactions, and dependencies.

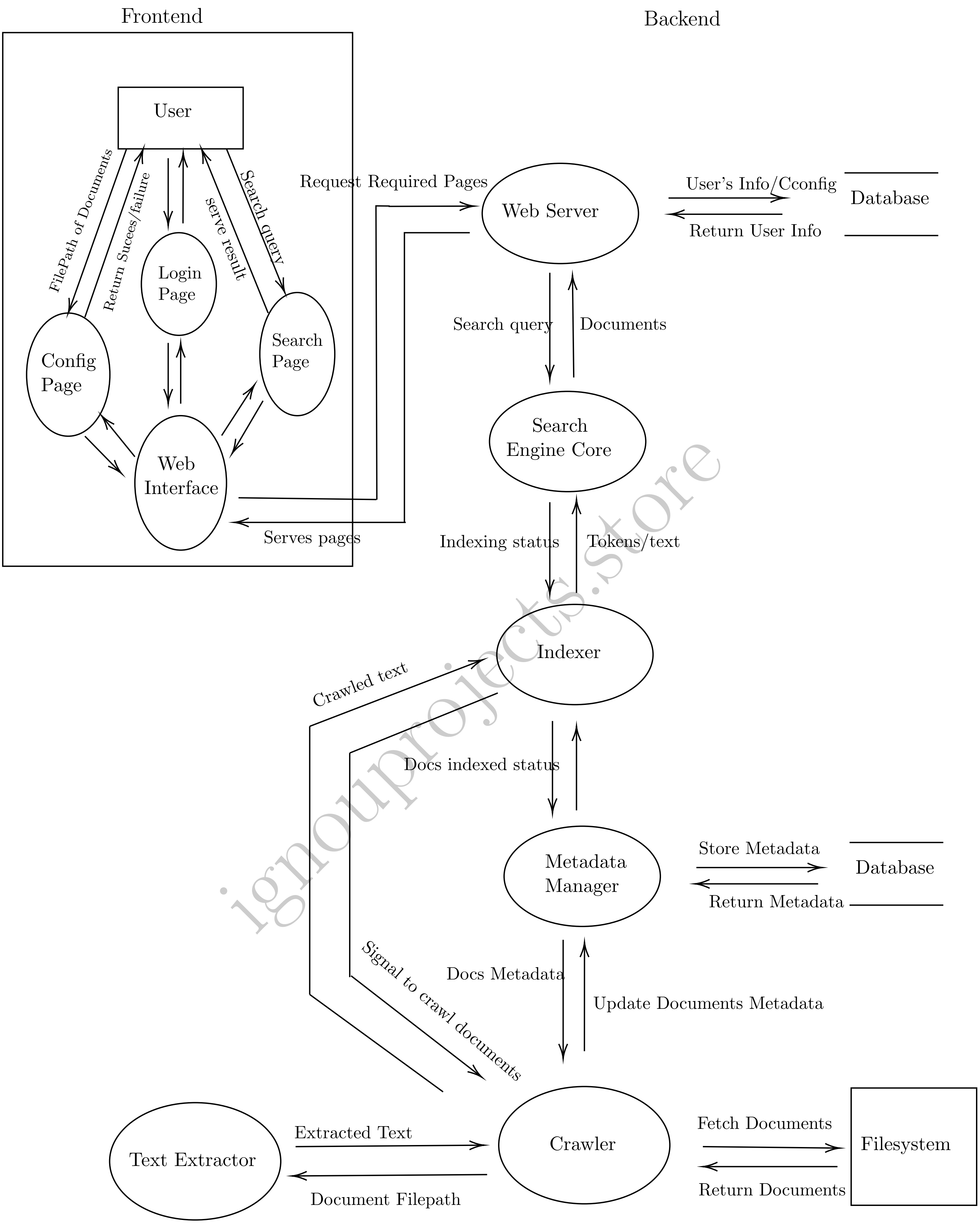
Detailed Processes in Level 2 DFD

- **Document Crawling:** This sub-process involves traversing file systems, cloud storage, or external sources to locate relevant documents. Specific methods include:
  - Recursive directory traversal for local storage.
  - API-based fetching for cloud platforms.
  - Web scraping for online document repositories.
- **Text Parsing and Extraction:** Once documents are identified, this sub-process handles the extraction of meaningful data. It includes:
  - Format-specific parsing (e.g., PDF, DOCX, TXT).
  - Content cleaning to remove metadata or formatting artifacts.
  - Natural Language Processing (NLP) techniques to extract entities and keywords.
- **Indexing:** The extracted data is indexed for efficient retrieval. This involves:
  - Creating inverted indexes for fast search operations.
  - Storing metadata (e.g., document size, creation date) alongside the indexed content.
  - Organizing data in a structure optimized for query handling.
- **Query Handling:** This sub-process interprets user queries and retrieves relevant documents. It includes:
  - Query parsing to understand user intent.
  - Searching the index for matches based on keywords or semantic similarity.
  - Ranking and filtering results to present the most relevant documents.
- **Feedback and Learning:** The system incorporates feedback mechanisms to improve accuracy and user satisfaction over time. For example:
  - Adjusting ranking algorithms based on user-selected results.
  - Incorporating user preferences for prioritizing certain document types or sources.

Importance of Level 2 DFD

By breaking down these sub-processes, the Level 2 DFD serves as a blueprint for developers and system architects. It ensures:

- **Clarity:** Provides a detailed understanding of how individual components interact within the system.
- **Error Identification:** Helps identify potential bottlenecks or inconsistencies in the system design.
- **Optimization Opportunities:** Highlights areas for performance improvement, such as optimizing data flow or storage mechanisms.
- **Communication:** Facilitates effective collaboration between technical teams and stakeholders by presenting a detailed yet understandable view of the system.


Frontend
Backend
User
FilePath of Documents
Return Succes/failure
Login Page
serve result
Search query
Config Page
Search Page
Web Interface
Request Required Pages
Serves pages
Web Server
User's Info/Cconfig
Return User Info
Database
Search query
Documents
Search Engine Core
Indexing status
Tokens/text
Indexer
Crawled text
Docs indexed status
Metadata Manager
Store Metadata
Return Metadata
Database
Signal to crawl documents
Docs Metadata
Update Documents Metadata
Text Extractor
Extracted Text
Document Filepath
Crawler
Fetch Documents
Return Documents
Filesystem

Class Diagram

Introduction to Class Diagrams

A class diagram is a type of static structure diagram in the Unified Modeling Language (UML) that represents the system's objects, their attributes, methods, and relationships. It provides a blueprint for the system's design by showing how entities (classes) within the system interact and are interconnected. Class diagrams are crucial in object-oriented system development as they establish the core structure, data flow, and behavior of the system.

Purpose of Class Diagram

In the context of the Personal Document Scraper project, the class diagram serves to:

- Provide a clear representation of the system's data model, showing the relationships and dependencies between various entities.
- Facilitate communication among developers and stakeholders by presenting a shared understanding of the system's design.
- Serve as a reference for implementation, ensuring consistency and adherence to design principles.
- Identify areas of complexity or potential inefficiencies in the system design.

Overview of the Class Diagram for Personal Document Scraper

The class diagram for the Personal Document Scraper system encompasses the following key components:

- **User:** Represents the application users, their credentials, and preferences.
- **Document:** Represents the uploaded documents, including metadata and extracted content.
- **SearchEngine:** Handles indexing and querying of document content using Meilisearch.
- **AuthenticationManager:** Manages user login, registration, and session handling.
- **DocumentProcessor:** Processes uploaded documents for text extraction and metadata retrieval.
- **Dashboard:** Represents the user interface component for viewing and managing documents.

Detailed Description of Classes

- **User Class:**
  - **Attributes:**
    - `userId: String` – Unique identifier for the user.
    - `username: String` – The username chosen by the user.

    - `passwordHash: String` – Hashed version of the user's password.
    - `email: String` – User's email address for communication and password recovery.
  - **Methods:**
    - `registerUser()` – Registers a new user in the system.
    - `login()` – Authenticates a user with their credentials.
    - `updateProfile()` – Allows users to modify their account details.

- **Document Class:**
  - **Attributes:**
    - `documentId: String` – Unique identifier for each document.
    - `fileName: String` – Name of the uploaded file.
    - `fileType: String` – Format of the document (e.g., PDF, DOCX).
    - `uploadDate: Date` – Timestamp for when the document was uploaded.
    - `metadata: Map<String, String>` – Key-value pairs of document metadata.
    - `content: String` – Extracted text content from the document.
  - **Methods:**
    - `extractContent()` – Extracts text and metadata using Apache TIKA.
    - `storeDocument()` – Saves the document to the database.
    - `deleteDocument()` – Deletes the document from the system.

- **SearchEngine Class:**
  - **Attributes:**
    - `index: Object` – Represents the Meilisearch index.
  - **Methods:**
    - `indexDocument()` – Adds a document to the search index.
    - `queryIndex(query: String)` – Searches for documents based on a query string.
    - `updateIndex()` – Updates the search index when documents are modified.

- **AuthenticationManager Class:**
  - **Attributes:**
    - `sessionId: String` – Identifier for the current user session.
  - **Methods:**
    - `authenticateUser()` – Validates login credentials.
    - `logout()` – Ends the user session.
    - `resetPassword()` – Facilitates password recovery.

- **DocumentProcessor Class:**
  - **Attributes:**

    - `processorId: String` – Identifier for the document processing instance.
  - **Methods:**
    - `parseDocument()` – Parses a document to extract text and metadata.
    - `validateFileType()` – Ensures the uploaded file is of a supported format.

- **Dashboard Class:**
  - **Attributes:**
    - `recentActivity: List` – List of recent uploads or searches.
  - **Methods:**
    - `renderDashboard()` – Generates the user dashboard view.
    - `listDocuments()` – Displays a list of uploaded documents.
    - `searchDocuments()` – Integrates with the SearchEngine to display search results.

Relationships in the Class Diagram

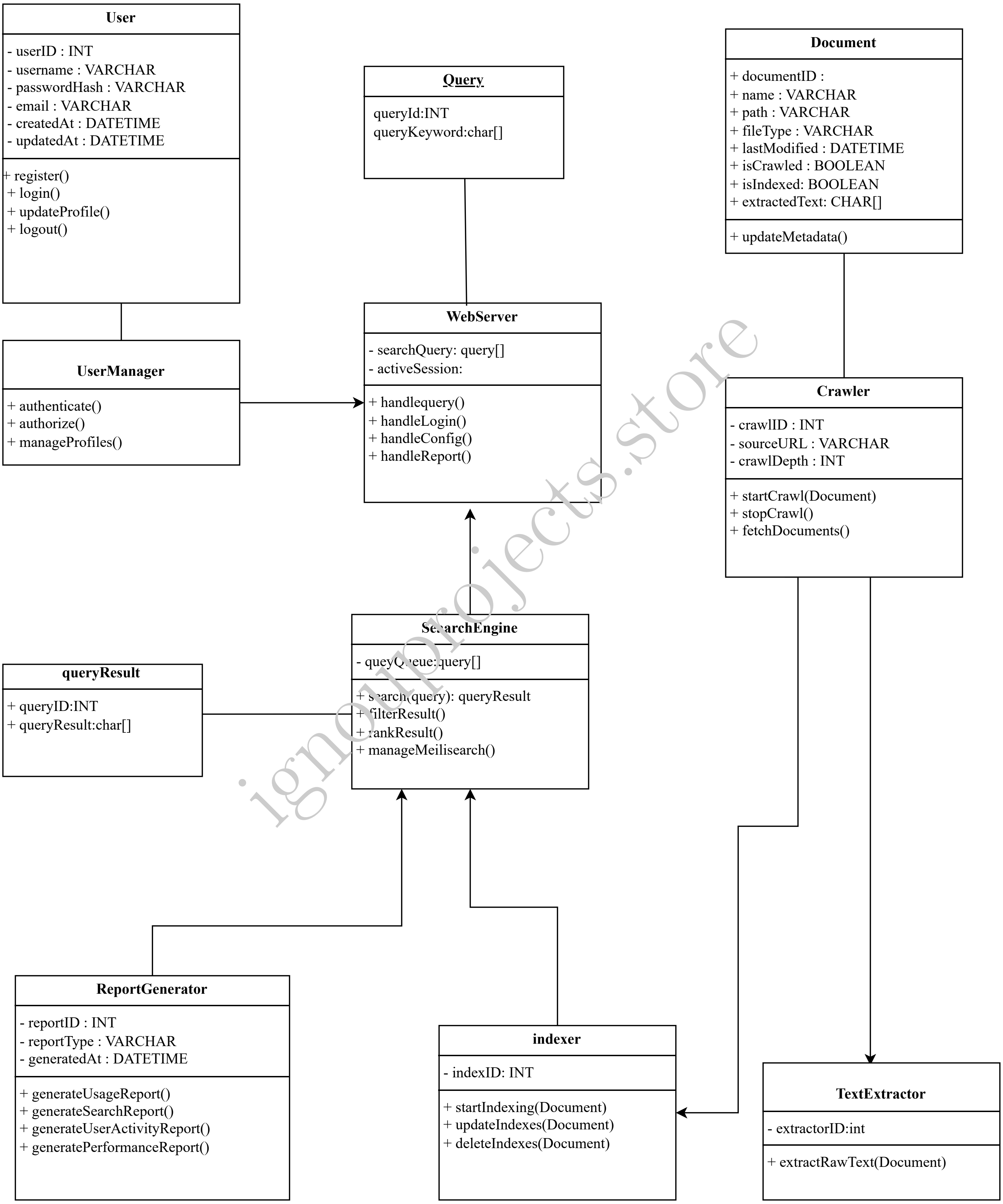
The relationships between these classes are represented as:

- **Associations:** The User class is associated with the Document class as each user can upload multiple documents.
- **Aggregation:** The Dashboard aggregates functionality from multiple classes, including DocumentProcessor and SearchEngine.
- **Dependencies:** The AuthenticationManager class depends on the User class for validating credentials.

Conclusion

The class diagram forms the backbone of the Personal Document Scraper's object-oriented design. By detailing the structure and relationships of the system's core components, it ensures that the implementation aligns with the project's functional and non-functional requirements.

Class Diagram:



Technical Specifications

1. **User Authentication:** Password hashing with industry-standard algorithms, token-based session management, and optional two-factor authentication.

2. **Search Engine Optimization:** Meilisearch will be optimized for fast response.

3. **Document Parsing:** Apache TIKA will support multiple formats seamlessly.

4. **Deployment:** Docker containers ensure portability. Hosted on a scalable cloud service.

5. **Security Best Practices:** Secure API endpoints, end-to-end encryption of data in transit and at rest.

Risk Analysis

- **Data Security Risks:** Countered by encryption, regular audits, and secure APIs.

- **Performance Risks:** Mitigated through load testing and efficient indexing algorithms.

- **User Adoption Risks:** Addressed via user training and accessible design.

Conclusion

The **Personal Document Scrapper** is designed to provide an intuitive, secure, and high-performance solution for managing personal document collections. With comprehensive system analysis, user-focused design, and cutting-edge technologies, the project is well-positioned to meet user needs effectively while maintaining robust security and scalability.

By breaking down each component in the analysis, this document ensures clarity and focus in the development process, aligning technical decisions with user expectations.

4.4 SDLC Waterfall Model

The Waterfall model is a linear and sequential approach to software development, where each phase must be completed before the next phase begins. This model is named for its cascading effect, where progress flows downwards through distinct phases, resembling a waterfall. The Waterfall model is characterized by its structured approach, making it easy to understand and manage. It is particularly effective for projects with well-defined requirements and where changes are minimal during the development process.

The origins of the Waterfall model can be traced back to the 1970s, when it was first introduced by Dr. Winston W. Royce in a paper titled "Managing the Development of Large Software Systems." Although Royce did not explicitly name it the Waterfall model, his description of a sequential development process laid the groundwork for what would later be known as the Waterfall methodology. Over the years, the model gained popularity due to its simplicity and clarity, becoming one of the first formalized approaches to software development.

The Waterfall model consists of distinct phases, including requirements analysis, system design, implementation, testing, deployment, and maintenance. Each phase has specific deliverables and milestones, ensuring that the project progresses in a systematic manner. This model is particularly beneficial for projects where requirements are well understood from the outset, allowing for a clear path from conception to completion.

4.4.1 Phases of the Waterfall Model

The Waterfall model consists of several key phases:

1. **Requirements Analysis:** In this initial phase, all the requirements of the system are gathered and documented. This includes understanding user needs, system functionalities, and constraints. A comprehensive requirements specification is created to guide the subsequent phases.

2. **System Design:** Based on the requirements gathered, the system architecture and design are developed. This phase outlines how the system will be structured, including hardware and software specifications, user interfaces, and data models.

3. **Implementation:** During the implementation phase, the actual coding of the system takes place. Developers write the code according to the design specifications, and the system components are integrated to form a complete system.

4. **Testing:** After implementation, the system undergoes rigorous testing to identify and fix any defects or issues. This phase ensures that the system meets the specified requirements and functions correctly.

5. **Deployment:** Once testing is complete, the system is deployed to the production environment. Users can start using the system, and any necessary training or support is provided.

6. **Maintenance:** The final phase involves ongoing maintenance and support for the system. This includes fixing any issues that arise, implementing updates, and making enhancements based on user feedback.

4.4.2 Why We Chose the Waterfall Model for the Personal Document Scraper

The Waterfall model was chosen for the Personal Document Scraper project due to its structured approach and clear phase delineation. Given the project's well-defined requirements and the need for a reliable and efficient tool for document management, the Waterfall model allows for thorough documentation and planning at each stage. This ensures that all user requirements are met and that the system is built with a focus on quality and performance. Additionally, the linear nature of the Waterfall model facilitates easier project management and tracking of progress, making it an ideal choice for this project.

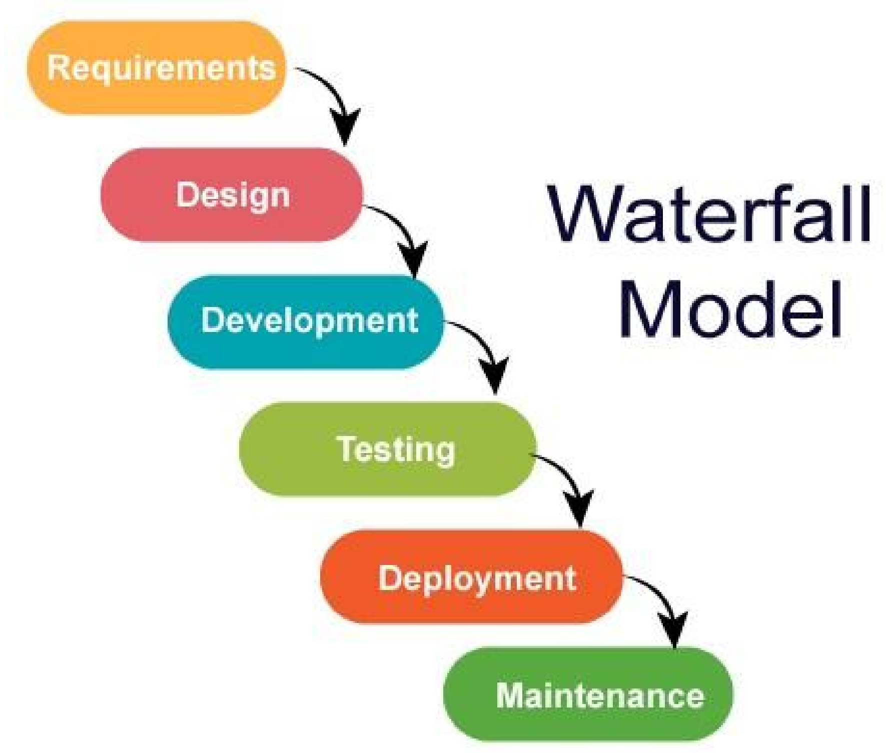


Figure 2: The Waterfall Model in Software Development

4.5 Waterfall Model for Personal Document Scraper

Requirement Analysis

The initial phase of the Waterfall Model involves gathering and analyzing all necessary requirements for the project. The primary requirements for the Personal Document Scraper project are as follows:

- A robust system for document scraping, indexing, and searching.
- Secure user authentication and access control.
- A user-friendly interface for document search and retrieval.
- Support for various document types (e.g., PDF, Word, plain text).
- High-performance processing and fast search capabilities.
- Version control and collaboration support.

System Design

Based on the analyzed requirements, the design for the project is outlined with the following key components:

- **Backend:** Implemented using Rust for performance and memory safety.
- **Document Processing Layer:** Uses Apache TIKA for content extraction.
- **Search Engine:** Integrates Meilisearch for indexing and query handling.
- **Version Control:** Managed through Git with repositories hosted on GitHub.
- **CI/CD Pipelines:** Implemented to streamline development, testing, and deployment.
- **Web Interface:** Provides login and search functionalities for users.

Implementation

The implementation phase involves coding and developing the system in sequential phases:

- **Backend Development (Rust):** Implementing core functionalities, including document handling and user authentication.
- **Integration with Apache TIKA:** Creating modules for extracting text and metadata from documents.
- **Search Engine Integration (Meilisearch):** Setting up indexing mechanisms and query endpoints.
- **User Interface Creation:** Developing the web interface for user interaction.
- **Version Control and Collaboration:** Using Git and GitHub for tracking changes and managing contributions.

Verification

Verification ensures each module meets project specifications through thorough testing:

- **Unit Testing (Rust code):** Verifies individual functions and modules work correctly.
- **Integration Testing:** Ensures seamless operation between Apache TIKA, Meilisearch, and the backend.
- **User Interface Testing:** Checks that the web interface is responsive and user-friendly.
- **Performance Testing:** Measures the speed and accuracy of document processing and search results.
- **Continuous Integration/Deployment (CI/CD):** Automates testing to ensure code quality and prevent integration issues.

Deployment

Deployment makes the project available for real-world use:

- **Hosting:** Deploying the web interface and backend on a server or cloud platform.
- **Documentation:** Providing comprehensive user and developer documentation.
- **Security Measures:** Implementing protocols to protect user data and secure access.

Maintenance

Post-deployment maintenance involves:

- **Bug Fixes and Patches:** Addressing reported issues and ongoing monitoring.
- **Feature Enhancements:** Adding new features or optimizations based on user feedback.
- **CI/CD Workflow Updates:** Keeping CI/CD processes updated for smooth development.

Project Closure

The project closure phase ensures the formal completion of the project. Key activities include:

- **Final Testing and Validation:** Conducting comprehensive testing to verify that all functionalities meet the project requirements.
- **Documentation Handover:** Delivering final project documentation to stakeholders for future reference.

- **Lessons Learned:** Documenting insights and lessons learned during the project to inform future development practices.
- **Project Review and Evaluation:** Assessing the overall success and performance of the project against initial goals and metrics.
- **Sign-off:** Obtaining formal approval and project sign-off from stakeholders.

4.6 Software Requirements Specification - Personal Document Scraper

Introduction

The purpose of this document is to comprehensively define the requirements, technical specifications, and design considerations for the development of the **Personal Document Scraper** project. This document serves as a guide for developers, stakeholders, and project managers to ensure the application meets its functional and non-functional requirements while providing a seamless user experience. The Personal Document Scraper system aims to offer a secure, robust, and efficient platform for managing and searching personal documents.

This document highlights the project's purpose, scope, stakeholders, requirements, and technical specifications. By systematically detailing the development process, this document also provides a blueprint for maintaining system integrity and achieving project milestones within set timelines.

Project Overview

The Personal Document Scraper is a cutting-edge web-based application designed to empower users to manage, search, and interact with their personal document collections. The application supports various file formats, ranging from text documents to PDFs and multimedia files, ensuring versatility in its functionality.

Leveraging state-of-the-art tools and technologies, the system utilizes **Rust** for backend efficiency, **Apache TIKA** for comprehensive content extraction, and **Meilisearch** for rapid and precise search capabilities. The application prioritizes security, performance, and user experience, ensuring a robust solution for document management.

Key features of the project include:

- Secure and intuitive user authentication mechanisms.
- Advanced text extraction and metadata analysis for diverse file formats.
- Real-time document indexing and search for enhanced productivity.
- An elegant and user-friendly interface for seamless interaction.

Scope

The Personal Document Scraper is designed to cater to a broad range of users, from professionals managing office documents to individuals organizing personal files. The system offers:

- Secure upload, storage, and retrieval of documents in multiple formats.
- Intelligent content extraction for meaningful text and metadata retrieval.
- Full-text search capabilities powered by advanced indexing and query optimization techniques.
- A robust system architecture designed for scalability, reliability, and efficiency.

- Customizable search filters and sorting options for personalized user experiences.

The scope of the project includes the development, deployment, and maintenance of the application to ensure it remains adaptable to evolving user needs and technological advancements.

Stakeholders

Stakeholders are the entities or individuals directly or indirectly impacted by the project. For the Personal Document Scraper, they include:

- **End Users:** These are the primary users of the system, including individuals and organizations looking to efficiently manage their personal or professional document repositories.
- **Project Developers:** Responsible for the design, coding, testing, and deployment of the application. Their role is critical to translating requirements into a functional system.
- **System Administrators:** Handle the system's operational aspects, such as server management, security updates, and technical support to maintain uninterrupted service.
- **Project Managers:** Oversee the entire project lifecycle, ensuring timely delivery of milestones, resource allocation, and adherence to the defined requirements.

Functional Requirements

The core functionalities of the Personal Document Scraper are pivotal in meeting the system's objectives. They include:

- **User Authentication:** Secure user login and session management to ensure data privacy.
- **Document Upload:** Support for uploading documents in various formats, such as DOCX, PDF, TXT, and more.
- **Text Extraction:** Using Apache TIKA for robust content extraction, converting documents into searchable text while retrieving associated metadata.
- **Indexing and Search:** Leveraging Meilisearch for indexing document content, ensuring quick and accurate search responses.
- **Filters and Sorting:** Enable users to refine search results based on document type, upload date, or relevance.
- **Dashboard Management:** A central hub where users can view and manage uploaded documents and recent activities.

Non-Functional Requirements

To ensure the system's quality and reliability, the following non-functional requirements are defined:

- **Performance:** The system must handle up to 1,000 concurrent users and process document searches within an average response time of 500ms.
- **Security:** All user data must be encrypted during transmission and storage, adhering to best practices in cybersecurity.
- **Reliability:** The system should maintain 99.9% uptime and provide robust error-handling mechanisms.
- **Scalability:** Designed to accommodate an increasing number of users and expanding document collections without performance degradation.
- **Usability:** A clean and intuitive interface ensuring ease of use for non-technical users.

Technical Specifications

The development of the Personal Document Scraper involves modern technologies and tools to ensure a robust and efficient application:

- **Programming Language:** Rust is used for backend development, ensuring high performance and safety.
- **Content Extraction Tool:** Apache TIKA is employed for parsing and extracting text and metadata from various document formats.
- **Search Engine:** Meilisearch is utilized for indexing and searching document content, providing real-time and relevant results.
- **Version Control:** Git for collaborative development, with repositories hosted on GitHub.
- **Deployment Pipeline:** CI/CD pipelines ensure automated testing, integration, and deployment for consistent delivery.
- **Database:** A lightweight database to store user information and document metadata.

User Interface

A seamless user interface is a critical component of the Personal Document Scraper. The interface provides:

- A secure and responsive login page for user authentication.
- A dashboard displaying uploaded documents, recent activities, and system statistics.
- A search bar with advanced filtering options for refined query results.

- Search results presented in an organized layout, showing document previews, metadata, and quick-access options.

Conclusion

The Software Requirements Specification for the Personal Document Scraper outlines the foundational aspects of the project. By leveraging advanced tools and technologies, the system aims to deliver a powerful, efficient, and secure document management platform. This document serves as a roadmap to guide development and ensure the project's success.

5 Project Timeline

5.1 Gantt Chart

A Gantt chart is a type of bar chart that represents a project schedule. It is named after its inventor, Henry L. Gantt, who designed it in the 1910s. The Gantt chart is a visual tool that illustrates the start and finish dates of the various elements of a project. It is widely used in project management to plan, coordinate, and track specific tasks within a project.

5.1.1 Description

A Gantt chart typically includes:

- **Tasks:** The individual activities or work items that need to be completed.
- **Timeline:** A horizontal time axis that shows the duration of the project, divided into days, weeks, or months.
- **Bars:** Horizontal bars that represent the duration of each task. The length of the bar corresponds to the start and end dates of the task.
- **Dependencies:** Arrows or lines that indicate the relationships between tasks, showing which tasks must be completed before others can begin.

5.1.2 Usage

Gantt charts are used for:

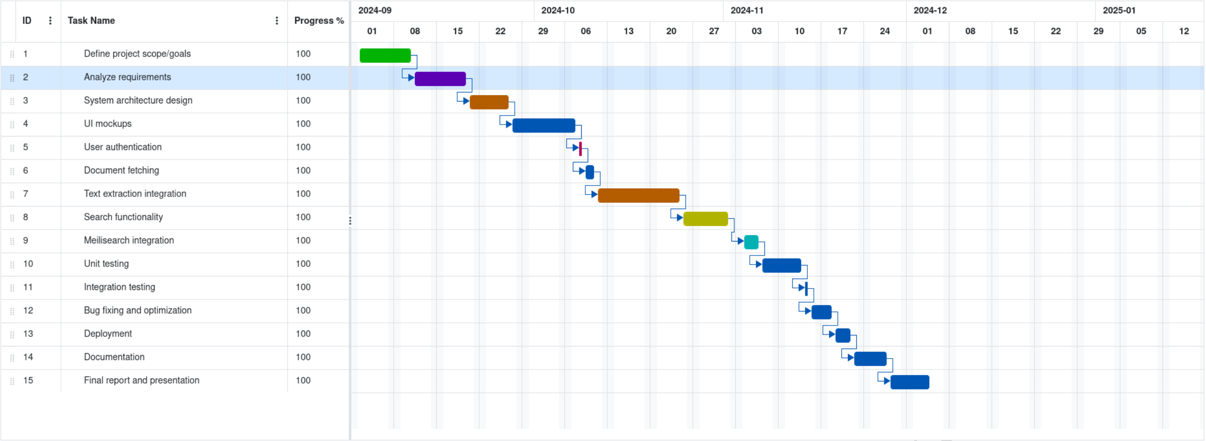
- **Planning:** Defining the tasks that need to be completed and their respective timelines.
- **Scheduling:** Assigning start and end dates to tasks and ensuring that resources are allocated efficiently.
- **Tracking:** Monitoring the progress of tasks and the overall project to ensure that it stays on schedule.
- **Communication:** Providing a clear visual representation of the project timeline to stakeholders, team members, and clients.

5.1.3 Why We Use Gantt Charts

Gantt charts are used because they:

- **Improve Visibility:** Provide a clear and concise visual representation of the project timeline, making it easy to see the status of each task.
- **Enhance Coordination:** Help in coordinating tasks and resources, ensuring that dependencies are managed effectively.
- **Facilitate Communication:** Serve as a communication tool to keep all stakeholders informed about the project schedule and progress.

- **Aid in Time Management:** Assist in identifying potential delays and bottlenecks, allowing for timely interventions to keep the project on track.



ID	Task Name	Progress %
1	Define project scope/goals	100
2	Analyze requirements	100
3	System architecture design	100
4	UI mockups	100
5	User authentication	100
6	Document fetching	100
7	Text extraction integration	100
8	Search functionality	100
9	Meilisearch integration	100
10	Unit testing	100
11	Integration testing	100
12	Bug fixing and optimization	100
13	Deployment	100
14	Documentation	100
15	Final report and presentation	100

Figure 3: Gantt Chart for the Project Timeline

6 System Design

System design represents a crucial phase within the Software Development Life Cycle (SDLC), during which the architectural and technical blueprint of a software system is established. This phase outlines how the different components and modules of the system will interact to fulfill the defined requirements and objectives. Below is an overview of the main aspects and procedures involved in system design.

6.1 Key Aspects of System Design

- **Architectural Design**: Establish the overall system structure and choose suitable architectural patterns such as client-server, microservices, or monolithic designs. This phase includes planning the interactions between primary components and defining data flow.
- **Component Design**: Divide the system into smaller, manageable parts or modules, specifying the roles and functions of each. This ensures modularity and clear interfaces for easier development and maintenance.
- **Data Design**: Develop data storage strategies, database schemas, and data models that ensure effective data management. This includes focusing on data integrity, normalization to reduce redundancy, and scalability.
- **User Interface Design**: Design user-friendly and intuitive UI layouts. Prioritize usability, accessibility, and responsive design to cater to various devices and user preferences.
- **Algorithm Design**: Plan and define the algorithms and data structures that will be implemented for data processing, searching, sorting, and optimizing various tasks within the system.

6.2 Steps in System Design

1. **Requirement Analysis**: Begin by thoroughly analyzing and understanding the project's requirements and limitations. Collect input from stakeholders to ensure clarity and completeness.
2. **System Architecture**: Create a high-level architectural outline depicting the major system components and their interactions. Select architectural patterns and decide on deployment methods, whether on-premise or cloud-based.
3. **Detailed Design**: Break down the architecture into detailed modules. For each component, specify its interface, role, and interactions with other modules within the system.
4. **Data Design**: Plan the database structure, including tables, relationships, and data storage solutions. Include strategies for data migration and maintenance to ensure consistent data handling.
5. **UI/UX Design**: Create wireframes or prototypes for the user interface. Map out layouts, navigation flows, and user interactions while ensuring alignment with branding and user expectations.

6. **Algorithm and Code Design**: Design and document algorithms and data structures necessary for implementing the system's functions. Establish coding standards and best practices for uniformity.

7. **Security Design**: Identify potential security threats and weaknesses. Incorporate security features such as authentication, authorization, encryption, and input validation to safeguard the system.

8. **Performance Design**: Optimize code and database queries for better performance. Plan for caching mechanisms, load balancing, and scalability to accommodate higher user loads and data volume.

9. **Testing and Validation**: Formulate a comprehensive testing plan and create test cases for each module. Ensure the design is aligned with the testing strategy and validated through rigorous testing.

10. **Documentation**: Compile thorough documentation covering the entire system design, including architectural diagrams, data models, interface details, and coding guidelines for developers and stakeholders.

11. **Review and Validation**: Conduct reviews of the design with peers and stakeholders. Gather feedback, make necessary revisions, and ensure the design aligns with project objectives and user expectations.

6.3 Modules of the Personal Document Scraper Project

1. User Management Module

The User Management Module is designed to securely manage user accounts and access control.

- **User Registration and Login**: Provides secure user registration with encryption to protect credentials and supports multi-factor authentication (MFA) for additional security.
- **Authentication and Authorization**: Uses token-based authentication (e.g., JWT) for secure session management and implements role-based access control (RBAC) for differentiated user roles.
- **Profile Management**: Allows users to update profiles, change passwords, and manage settings such as notification preferences.
- **Session Management**: Monitors active sessions, implements timeout for inactive users, and provides session management tools for administrators.

2. Crawling Module

The Crawling Module handles the automated discovery of directories and files for document collection.

- **Recursive Path Traversal**: Explores directories recursively to find files that match user-defined criteria (e.g., extensions, dates).
- **Path Filtering and Exclusions**: Implements pattern-matching filters to exclude directories or files based on user-defined rules.
- **Crawling Scheduler**: Integrates with scheduling tools for automated crawling at specific intervals, with status monitoring.
- **Concurrency and Performance**: Uses multithreading to handle large structures efficiently and limits processed files to maintain system performance.

3. Text Extraction Module

This module extracts readable content from files using specialized tools and libraries.

- **Text Extraction**: Utilizes libraries like `textextract` to extract content from various formats (e.g., PDF, DOCX, TXT).
- **Format Support and Extensibility**: Includes support for OCR for scanned documents and is extensible to new formats with additional libraries.
- **Error Handling and Logging**: Logs extraction errors for analysis and troubleshooting, with detailed error reports.
- **Preprocessing**: Ensures document readiness by performing format validation and sanitization before extraction.

4. Indexing Module

The Indexing Module organizes extracted text for efficient search and retrieval.

- **Text Indexing**: Uses indexing engines like Meilisearch to build structured indexes for fast full-text searches.
- **Data Structuring**: Organizes data using tokens and inverted indexes, optimizing search response times.
- **Incremental and Batch Indexing**: Supports incremental updates for new documents and batch re-indexing for bulk updates.
- **Scalability and Optimization**: Includes compression and deduplication to optimize index size and performance.

5. Search Module

This module allows users to query the indexed documents with various features.

- **User Interface for Searching**: Offers a web-based interface with advanced search operators, filters, and result previews.
- **Full-Text Search**: Implements keyword and phrase searches with highlighted results.
- **Relevance and Ranking**: Uses ranking algorithms to sort results based on keyword relevance and other factors.
- **Faceted Search and Filtering**: Provides filters for narrowing down search results by attributes such as date and file type.

6. Metadata Module

This module manages additional data associated with documents to enhance searches.

- **Metadata Extraction**: Collects metadata such as file type, size, and timestamps during crawling.
- **Metadata Storage**: Ensures metadata is stored with main content for quick retrieval.
- **Metadata Enrichment**: Allows user-defined metadata and integrates external data sources for enriched information.

7. Reporting Module

Generates reports and analytics on system usage and document statistics.

- **Usage Analytics**: Tracks user activity and search trends to optimize system performance.
- **Document Statistics**: Provides reports on document counts and system operations over time.

- **Search Performance Metrics**: Monitors search response times and flags performance issues.
- **Exportable Reports**: Allows exporting reports in formats like CSV and PDF for external analysis.

6.4 Database Design

1. Primary Database - Meilisearch (Key-Value Database)

The **primary database** used in the system is **Meilisearch**, which handles full-text searching of the content within the uploaded documents. Meilisearch utilizes an **inverted index** to optimize search performance, making it an ideal choice for indexing and retrieving document content based on user queries.

6.4.1 Inverted Index in Meilisearch

An **inverted index** is a data structure that stores a mapping of terms (words) to the documents in which they appear. This allows for rapid searches by directly linking terms to their respective documents rather than scanning the entire document content. When a document is indexed, the terms (words) within the document are extracted and stored in a way that allows for efficient querying.

Advantages of the Inverted Index:

- **Fast Full-Text Search**: The inverted index enables quick searching across large volumes of document content.
- **Efficient Query Execution**: Searching for terms in indexed documents is much faster than scanning document content.
- **Scalability**: The index can easily scale as more documents are added.
- **Relevance Ranking**: The inverted index can be combined with ranking algorithms to return more relevant search results.
- **Storage Efficiency**: Storing only terms and document references saves space compared to storing full document content.
- **Support for Complex Queries**: Boolean searches, phrase searches, and partial matching are supported.

Algorithm and Data Structure

The inverted index in Meilisearch is built using a combination of algorithms and data structures:

- **Tokenization**: Documents are broken down into individual terms (tokens) using language-specific tokenization rules. For example, the sentence ”The quick brown fox” would be tokenized into [”the”, ”quick”, ”brown”, ”fox”].

- **Normalization**: Tokens are normalized to handle variations (e.g., case-folding, removing accents) to improve search accuracy. For example, "Café" and "cafe" would be normalized to the same term.
- **B-tree Data Structure**: The index uses B-trees to store and retrieve term-to-document mappings efficiently. B-trees provide O(log n) time complexity for lookups and maintain sorted order of terms.

Key-Value Structure

The key-value structure in Meilisearch looks like this:

```
{
    "term1": {
        "doc_id1": [position1, position2, ...],
        "doc_id2": [position1, position3, ...],
        ...
    },
    "term2": {
        "doc_id3": [position1, position4, ...],
        "doc_id4": [position2, position5, ...],
        ...
    }
}
```

For example, if we have two documents:

```
Doc1 (id: 1): "The quick brown fox"
Doc2 (id: 2): "The brown dog"

The index would look like:
{
    "the": {
        "1": [0],
        "2": [0]
    },
    "quick": {
        "1": [1]
    },
    "brown": {
        "1": [2],
        "2": [1]
    },
    "fox": {
        "1": [3]
    },
    "dog": {
        "2": [2]
    }
}
```

This structure allows for:

- Fast term lookups
- Position-aware phrase searching
- Relevance scoring based on term frequency
- Efficient boolean query operations

Meilisearch is responsible for indexing the documents and enabling fast search queries. It stores document identifiers (IDs) and their indexed content, allowing users to search for specific words or phrases.

2. Secondary Database - SQLite (Relational Database for User and Metadata Information)

The **secondary database** is **SQLite**, which is used to store structured data related to users and document metadata. SQLite is a lightweight, serverless, and relational database that is ideal for smaller-scale applications like this one, where the need for a full-fledged database server is not required.

6.4.2 Why SQLite?

- **Lightweight**: SQLite is embedded directly within the application, requiring no server.
- **Relational Model**: It organizes data in tables, allowing efficient handling of user data and metadata.
- **Fast for Small to Medium-Sized Applications**: SQLite performs well for the scope of this project, where moderate data volume and simple queries are expected.

In this project, the SQLite database is accessed using `rusqlite`, a Rust library for interacting with SQLite databases. `rusqlite` provides an easy-to-use interface for interacting with SQLite from Rust applications, making it a great choice for this use case.

Metadata Table Design

The **Metadata Table** stores information about each document, such as its name, file path, type, and the current status of crawling and indexing.

Here is the schema for the **Metadata Table** in the SQLite database:

```
CREATE TABLE Metadata (
    DocumentID INTEGER PRIMARY KEY,
    Name VARCHAR(255),
    Path VARCHAR(255),
    FileType VARCHAR(50),
    LastModified DATETIME,
    IsCrawled BOOLEAN,
    IsIndexed BOOLEAN,
    ExtractedText TEXT  -- Stored as JSON or serialized array in SQLite
);
```

Metadata Table Example

Here are two examples of how data in the **Metadata Table** might look in **JSON** format:

Example 1 (Document has been crawled and indexed):

```
{
  "DocumentID": 1,
  "Name": "Project_Report.pdf",
  "Path": "/user/documents/Project_Report.pdf",
  "FileType": "PDF",
  "LastModified": "2024-11-12T14:30:00",
  "IsCrawled": true,
  "IsIndexed": true,
  "ExtractedText": [
    "This is an example of a project report document.",
    "It contains sections on design, implementation, and testing."
  ]
}
```

Example 2 (Document has not been crawled or indexed yet):

```
{
  "DocumentID": 2,
  "Name": "Meeting_Notes_2024.docx",
  "Path": "/user/documents/Meeting_Notes_2024.docx",
  "FileType": "DOCX",
  "LastModified": "2024-11-10T08:45:00",
  "IsCrawled": false,
  "IsIndexed": false,
  "ExtractedText": []
}
```

6.4.3 Storing Data in SQLite

The data would be stored in the **Metadata Table** using SQL commands executed through the `rusqlite` Rust library. Below is an example of how data is inserted into the database using `rusqlite`.

First, we establish a connection to the SQLite database and prepare the necessary SQL statements for inserting data into the `Metadata` table.

```
use rusqlite::{params, Connection, Result};

fn insert_document(conn: &Connection, document_id: i32, name: &str, path: &str, file_
    conn.execute(
        "INSERT INTO Metadata (DocumentID, Name, Path, FileType, LastModified, IsCraw
        VALUES (?1, ?2, ?3, ?4, ?5, ?6, ?7, ?8)",
        params![document_id, name, path, file_type, last_modified, is_crawled, is_ind
    )?;
    Ok(())
```

```
}

fn main() -> Result<()> {
    let conn = Connection::open("documents.db")?; // Opening SQLite database

    // Example insertion for a crawled and indexed document
    insert_document(&conn, 1, "Project_Report.pdf", "/user/documents/Project_Report.p
                    r#"["This is an example of a project report document.", "It conta

    // Example insertion for a document that has not been crawled or indexed
    insert_document(&conn, 2, "Meeting_Notes_2024.docx", "/user/documents/Meeting_Not

    Ok(())
}
```

In this example:

- `Connection::open("documents.db")`: This opens a connection to the SQLite database file `documents.db`.
- `conn.execute(...)`: Executes an SQL `INSERT` statement to store the document metadata in the `Metadata` table.
- `params!`: A macro that binds the values to the placeholders `?1`, `?2`, etc., in the SQL statement.

The inserted data includes:

- `DocumentID`: The unique identifier for the document.
- `Name`: The document name.
- `Path`: The location of the document.
- `FileType`: The type of the document file (e.g., PDF, DOCX).
- `LastModified`: The timestamp of the last modification of the document.
- `IsCrawled`: A boolean indicating whether the document has been crawled.
- `IsIndexed`: A boolean indicating whether the document has been indexed.
- `ExtractedText`: A JSON array or serialized string that holds the extracted text from the document.

The `rusqlite` library simplifies the interaction with SQLite in Rust by managing database connections, executing SQL commands, and handling query results.

Conclusion

The **database design** for the **Personal Document Scrapper** utilizes two databases:

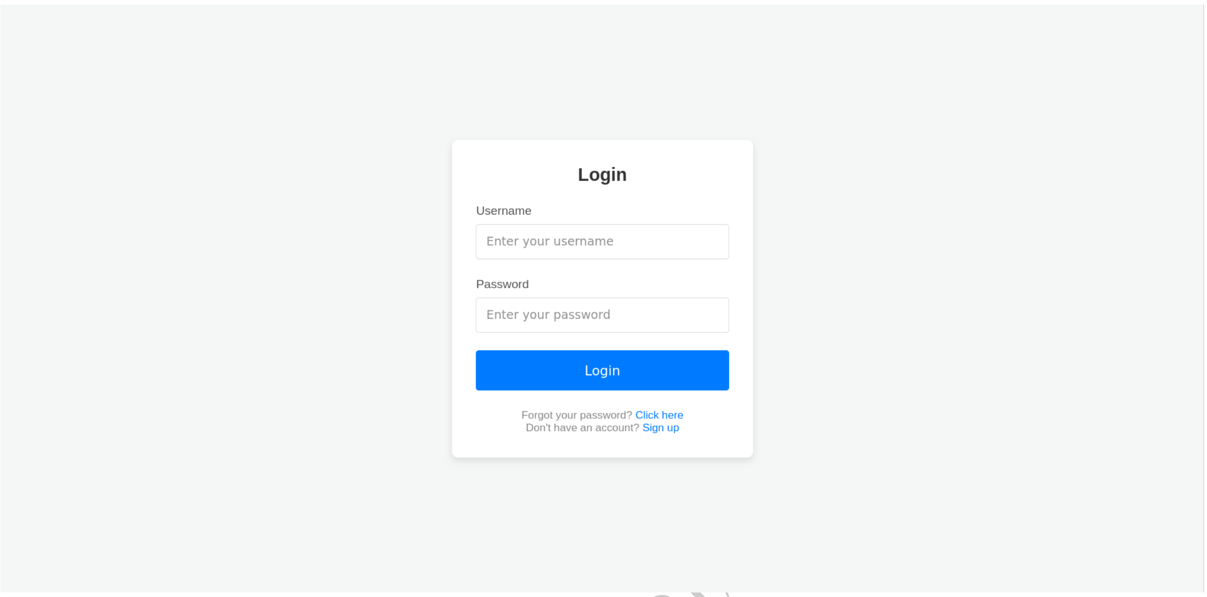
- **Meilisearch**: The **primary database** for full-text search and indexing of document content using an **inverted index**. This enables fast and scalable searching based on document content, providing a seamless search experience for users.
- **SQLite**: The **secondary database** for storing user data and document metadata. SQLite's relational model ensures efficient management of structured data, with the added benefit of being lightweight and easy to embed within the application.

This combination of databases allows for efficient storage, retrieval, and search of document metadata and content, ensuring the system's performance and scalability.

6.5 User Interface

Login Page

The login page provides a secure entry point for users to access the Personal Document Scraper system. It features a clean and intuitive interface with fields for username/email and password entry.



Login

Username

Enter your username

Password

Enter your password

Login

Forgot your password? Click here
Don't have an account? Sign up

Figure 4: Login Page Interface

Home Page

The home page serves as the central dashboard for users after logging in. It provides quick access to key features and displays important system statistics and recent activities.

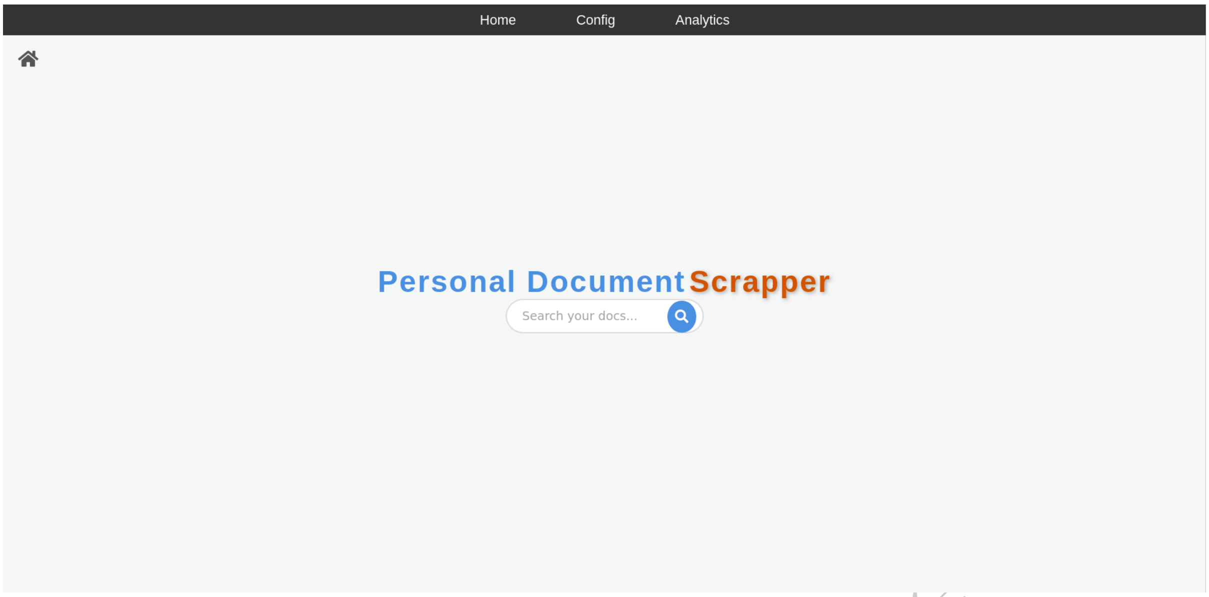
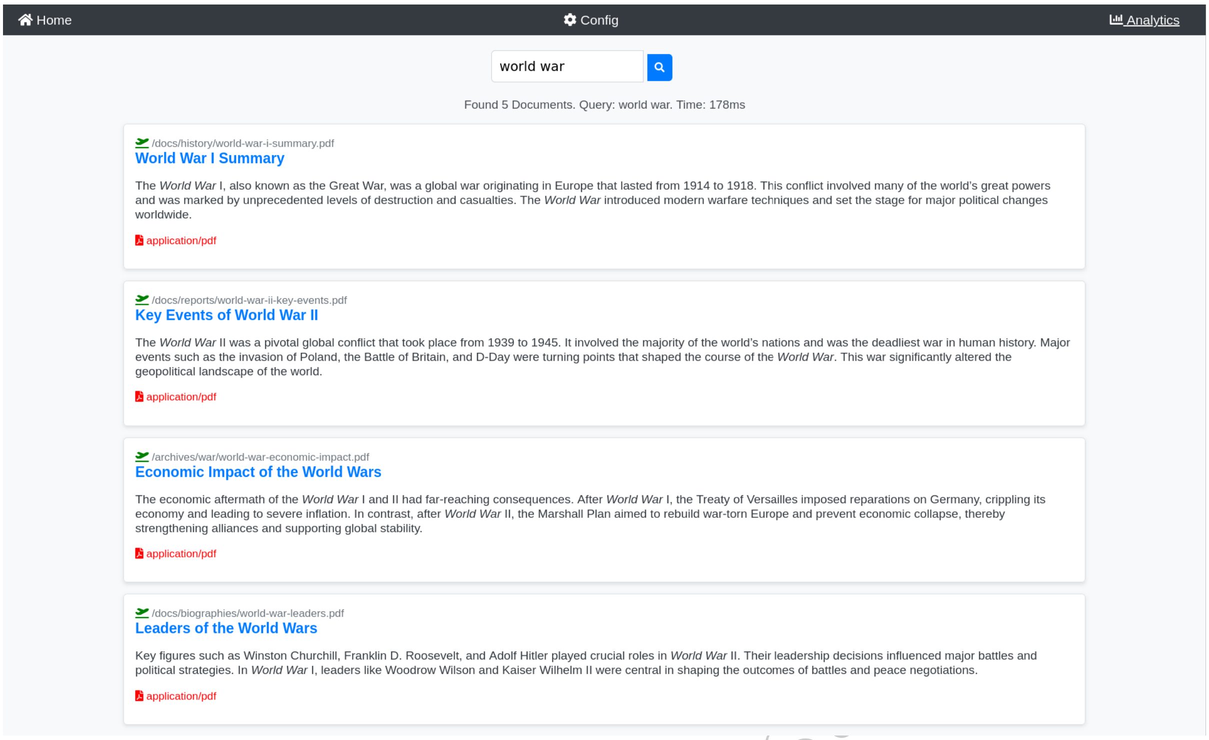


Figure 5: Home Page Dashboard

Search Page

The search page offers powerful document search capabilities with an intuitive interface. Users can perform full-text searches across their documents with advanced filtering options.



Found 5 Documents. Query: world war. Time: 178ms

/docs/history/world-war-i-summary.pdf
World War I Summary

The *World War* I, also known as the Great War, was a global war originating in Europe that lasted from 1914 to 1918. This conflict involved many of the world's great powers and was marked by unprecedented levels of destruction and casualties. The *World War* introduced modern warfare techniques and set the stage for major political changes worldwide.

application/pdf

/docs/reports/world-war-ii-key-events.pdf
Key Events of World War II

The *World War* II was a pivotal global conflict that took place from 1939 to 1945. It involved the majority of the world's nations and was the deadliest war in human history. Major events such as the invasion of Poland, the Battle of Britain, and D-Day were turning points that shaped the course of the *World War*. This war significantly altered the geopolitical landscape of the world.

application/pdf

/archives/war/world-war-economic-impact.pdf
Economic Impact of the World Wars

The economic aftermath of the *World War* I and II had far-reaching consequences. After *World War* I, the Treaty of Versailles imposed reparations on Germany, crippling its economy and leading to severe inflation. In contrast, after *World War* II, the Marshall Plan aimed to rebuild war-torn Europe and prevent economic collapse, thereby strengthening alliances and supporting global stability.

application/pdf

/docs/biographies/world-war-leaders.pdf
Leaders of the World Wars

Key figures such as Winston Churchill, Franklin D. Roosevelt, and Adolf Hitler played crucial roles in *World War* II. Their leadership decisions influenced major battles and political strategies. In *World War* I, leaders like Woodrow Wilson and Kaiser Wilhelm II were central in shaping the outcomes of battles and peace negotiations.

application/pdf

Figure 6: Search Result Interface

Configuration Page

The configuration page allows users to customize system settings and preferences. Users can manage crawling paths, file type filters, and indexing options.

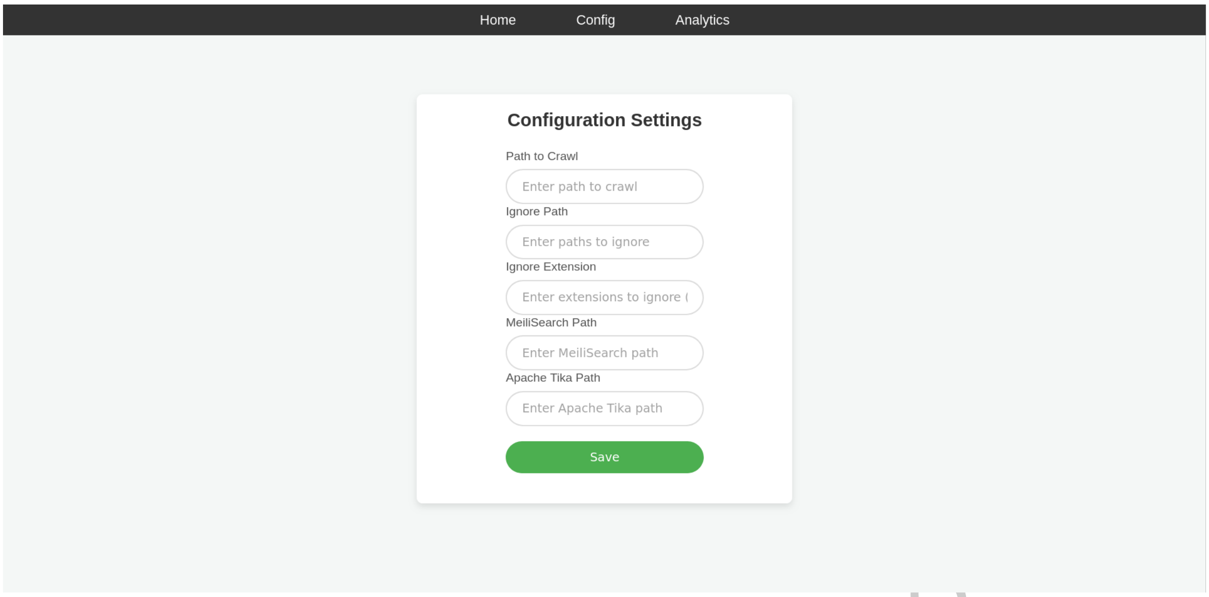


Figure 7: Configuration Settings Interface

7 Program Code

This section provides an overview of the program code for the Personal Document Scraper project. The codebase is divided into client-side files, server-side files, and raw code. Below, we present the directory structure, classify components based on their functionality, and provide explanations of key files.

7.1 Directory Structure

The following is the directory structure of the project, showcasing its organization:

```
.
|-- Cargo.lock
|-- Cargo.toml
|-- dependency_tree.txt
|-- project_tree.txt
|-- readme.md
|-- sample_files
|   `-- sample.pdf
|-- src
|   |-- analytics.rs
|   |-- auth.rs
|   |-- backup.rs
|   |-- crawler.rs
|   |-- health_checker.rs
|   |-- indexer.rs
|   |-- lib.rs
|   |-- logger.rs
|   |-- main.rs
|   |-- metadata_manager.rs
|   |-- notify.rs
|   |-- schedular.rs
|   |-- search.rs
|   |-- textract.rs
|   `-- web_server.rs
|-- static
|   |-- config.html
|   |-- index.html
|   |-- login.html
|   |-- logo.webp
|   |-- script.js
|   |-- search.css
|   |-- search.html
|   |-- search.js
|   `-- styles.css
`-- templates
    |-- login.html
    |-- login.html.tera
    `-- test.html

4 directories, 33 files
```

7.2 Client-Side Files

The client-side files handle the user interface and interactivity of the web application. These files are located in the 'static' and 'templates' directories.

7.2.1 Static Files

The static directory contains assets such as HTML, CSS, JavaScript, and images:

- **config.html, index.html, login.html, search.html:** HTML files rendering different pages of the application.
- **script.js, search.js:** JavaScript files enabling interactivity and client-side logic.
- **styles.css, search.css:** CSS files defining the styling of web pages.
- **logo.webp:** A branding asset used as the application logo.

7.2.2 Template Files

The 'templates' directory contains HTML templates, some of which utilize the Tera templating engine:

- **login.html, login.html.tera:** Templates for the login page, with one version designed for the Tera engine.
- **test.html:** A sample template for testing purposes.

7.3 Server-Side Files

The server-side logic is implemented in Rust and resides in the 'src' directory. These modules form the backend of the application, handling requests, processing data, and managing the system's state.

7.3.1 Core Configuration Files

- **Cargo.toml:** Defines project metadata, dependencies, and build configurations.
- **Cargo.lock:** Locks dependency versions to ensure reproducible builds.
- **main.rs:** The entry point of the application, responsible for initializing the server and managing its lifecycle.
- **lib.rs:** Acts as a central library file exposing shared functionalities across modules.

7.3.2 Key Modules

- **analytics.rs:** Tracks and reports analytics for system operations.
- **auth.rs:** Implements authentication logic, such as user login and session validation.
- **backup.rs:** Provides functionality for creating and restoring data backups.
- **crawler.rs:** Handles document crawling within directories or web sources.
- **health_checker.rs:** Monitors application health and uptime.
- **indexer.rs:** Manages indexing for documents using Meilisearch.
- **logger.rs:** Logs system events and errors for debugging and monitoring.
- **metadata_manager.rs:** Extracts and organizes metadata from uploaded documents.
- **notify.rs:** Implements notifications for system updates or alerts.
- **schedular.rs:** Automates tasks such as periodic crawling or backups.
- **search.rs:** Processes and handles search queries, including sorting and filtering results.
- **textract.rs:** Focuses on extracting text and metadata from various file formats.
- **web_server.rs:** Implements the web server and handles routing of HTTP requests.

7.3.3 Sample Files

The ‘sample_files‘ directory contains files used for testing and validation:

- **sample.pdf:** A sample document for testing document scraping capabilities.

7.3.4 Documentation and Metadata Files

- **readme.md:** A markdown file containing an overview of the project, setup instructions, and usage notes.

7.4 Dependency Tree

The following is the dependency tree for the project, generated by the ‘cargo tree‘ command. It lists all the dependencies used in the application, along with their versions and sub-dependencies:

```
personal_document_scrapper v0.1.0 (/home/kcubeterm/project/personal_document_scrapper)
|-- chrono v0.4.38
|   |-- iana-time-zone v0.1.61
|   `-- num-traits v0.2.19
|       [build-dependencies]
|       `-- autocfg v1.4.0
|-- meilisearch-sdk v0.27.1
|   |-- async-trait v0.1.83 (proc-macro)
|   |   |-- proc-macro2 v1.0.89
|   |   |   `-- unicode-ident v1.0.13
|   |   |-- quote v1.0.37
|   |   |   `-- proc-macro2 v1.0.89 (*)
|   |   `-- syn v2.0.87
|   |       |-- proc-macro2 v1.0.89 (*)
|   |       |-- quote v1.0.37 (*)
|   |       `-- unicode-ident v1.0.13
|   |-- bytes v1.8.0
|   |-- either v1.13.0
|   |   `-- serde v1.0.214
|   |       `-- serde_derive v1.0.214 (proc-macro)
|   |           |-- proc-macro2 v1.0.89 (*)
|   |           |-- quote v1.0.37 (*)
|   |           `-- syn v2.0.87 (*)
|   |-- futures v0.3.31
|   |   |-- futures-channel v0.3.31
|   |   |   |-- futures-core v0.3.31
|   |   |   `-- futures-sink v0.3.31
|   |   |-- futures-core v0.3.31
|   |   |-- futures-executor v0.3.31
|   |   |   |-- futures-core v0.3.31
|   |   |   |-- futures-task v0.3.31
|   |   |   `-- futures-util v0.3.31
|   |   |       |-- futures-channel v0.3.31 (*)
|   |   |       |-- futures-core v0.3.31
|   |   |       |-- futures-io v0.3.31
|   |   |       |-- futures-macro v0.3.31 (proc-macro)
|   |   |       |   |-- proc-macro2 v1.0.89 (*)
|   |   |       |   |-- quote v1.0.37 (*)
|   |   |       |   `-- syn v2.0.87 (*)
|   |   |       |-- futures-sink v0.3.31
|   |   |       |-- futures-task v0.3.31
|   |   |       |-- memchr v2.7.4
|   |   |       |-- pin-project-lite v0.2.15
|   |   |       |-- pin-utils v0.1.0
|   |   |       `-- slab v0.4.9
|   |   |           [build-dependencies]
|   |   |           `-- autocfg v1.4.0
|   |   |-- futures-io v0.3.31
|   |   |-- futures-sink v0.3.31
|   |   |-- futures-task v0.3.31
|   |   `-- futures-util v0.3.31 (*)
|   |-- futures-io v0.3.31
|   |-- iso8601 v0.6.1
|   |   `-- nom v7.1.3
|   |       |-- memchr v2.7.4
|   |       `-- minimal-lexical v0.2.1
|   |-- jsonwebtoken v9.3.0
|   |   |-- base64 v0.21.7
|   |   |-- ring v0.17.8
|   |   |   |-- cfg-if v1.0.0
|   |   |   |-- getrandom v0.2.15
|   |   |   |   |-- cfg-if v1.0.0
|   |   |   |   `-- libc v0.2.161
|   |   |   |-- spin v0.9.8
|   |   |   `-- untrusted v0.9.0
|   |   |   [build-dependencies]
|   |   |   `-- cc v1.1.34
|   |   |       `-- shlex v1.3.0
|   |   |-- serde v1.0.214 (*)
|   |   `-- serde_json v1.0.132
|   |       |-- itoa v1.0.11
|   |       |-- memchr v2.7.4
|   |       |-- ryu v1.0.18
|   |       `-- serde v1.0.214 (*)
|   |-- log v0.4.22
|   |-- meilisearch-index-setting-macro v0.27.1 (proc-macro)
|   |   |-- convert_case v0.6.0
|   |   |   `-- unicode-segmentation v1.12.0
|   |   |-- proc-macro2 v1.0.89 (*)
|   |   |-- quote v1.0.37 (*)
|   |   |-- structmeta v0.3.0
|   |   |   |-- proc-macro2 v1.0.89 (*)
|   |   |   |-- quote v1.0.37 (*)
|   |   |   |-- structmeta-derive v0.3.0 (proc-macro)
|   |   |   |   |-- proc-macro2 v1.0.89 (*)
|   |   |   |   |-- quote v1.0.37 (*)
|   |   |   |   `-- syn v2.0.87 (*)
|   |   |   `-- syn v2.0.87 (*)
|   |   `-- syn v2.0.87 (*)
|   |-- pin-project-lite v0.2.15
|   |-- reqwest v0.12.9
|   |   |-- base64 v0.22.1
|   |   |-- bytes v1.8.0
|   |   |-- futures-core v0.3.31
|   |   |-- futures-util v0.3.31 (*)
|   |   |-- h2 v0.4.6
|   |   |   |-- atomic-waker v1.1.2
|   |   |   |-- bytes v1.8.0
|   |   |   |-- fnv v1.0.7
|   |   |   |-- futures-core v0.3.31
|   |   |   |-- futures-sink v0.3.31
|   |   |   |-- http v1.1.0
|   |   |   |   |-- bytes v1.8.0
|   |   |   |   |-- fnv v1.0.7
|   |   |   |   `-- itoa v1.0.11
|   |   |   |-- indexmap v2.6.0
|   |   |   |   |-- equivalent v1.0.1
|   |   |   |   |-- hashbrown v0.15.1
|   |   |   |   `-- serde v1.0.214 (*)
|   |   |   |-- slab v0.4.9 (*)
|   |   |   |-- tokio v1.41.0
|   |   |   |   |-- bytes v1.8.0
|   |   |   |   |-- libc v0.2.161
|   |   |   |   |-- mio v1.0.2
|   |   |   |   |   `-- libc v0.2.161
|   |   |   |   |-- parking_lot v0.12.3
|   |   |   |   |   |-- lock_api v0.4.12
|   |   |   |   |   |   `-- scopeguard v1.2.0
|   |   |   |   |   |   [build-dependencies]
|   |   |   |   |   |   `-- autocfg v1.4.0
|   |   |   |   |   `-- parking_lot_core v0.9.10
|   |   |   |   |       |-- cfg-if v1.0.0
|   |   |   |   |       |-- libc v0.2.161
|   |   |   |   |       `-- smallvec v1.13.2
|   |   |   |   |-- pin-project-lite v0.2.15
|   |   |   |   |-- signal-hook-registry v1.4.2
|   |   |   |   |   `-- libc v0.2.161
|   |   |   |   |-- socket2 v0.5.7
|   |   |   |   |   `-- libc v0.2.161
|   |   |   |   `-- tokio-macros v2.4.0 (proc-macro)
|   |   |   |       |-- proc-macro2 v1.0.89 (*)
|   |   |   |       |-- quote v1.0.37 (*)
|   |   |   |       `-- syn v2.0.87 (*)
|   |   |   |-- tokio-util v0.7.12
|   |   |   |   |-- bytes v1.8.0
|   |   |   |   |-- futures-core v0.3.31
|   |   |   |   |-- futures-sink v0.3.31
|   |   |   |   |-- pin-project-lite v0.2.15
|   |   |   |   `-- tokio v1.41.0 (*)
|   |   |   `-- tracing v0.1.40
|   |   |       |-- pin-project-lite v0.2.15
|   |   |       `-- tracing-core v0.1.32
|   |   |           `-- once_cell v1.20.2
|   |   |-- http v1.1.0 (*)
|   |   |-- http-body v1.0.1
|   |   |   |-- bytes v1.8.0
|   |   |   `-- http v1.1.0 (*)
|   |   |-- http-body-util v0.1.2
|   |   |   |-- bytes v1.8.0
|   |   |   |-- futures-util v0.3.31 (*)
|   |   |   |-- http v1.1.0 (*)
|   |   |   |-- http-body v1.0.1 (*)
|   |   |   `-- pin-project-lite v0.2.15
|   |   |-- hyper v1.5.0
|   |   |   |-- bytes v1.8.0
|   |   |   |-- futures-channel v0.3.31 (*)
|   |   |   |-- futures-util v0.3.31 (*)
|   |   |   |-- h2 v0.4.6 (*)
|   |   |   |-- http v1.1.0 (*)
|   |   |   |-- http-body v1.0.1 (*)
|   |   |   |-- httparse v1.9.5
|   |   |   |-- itoa v1.0.11
|   |   |   |-- pin-project-lite v0.2.15
|   |   |   |-- smallvec v1.13.2
|   |   |   |-- tokio v1.41.0 (*)
|   |   |   `-- want v0.3.1
|   |   |       `-- try-lock v0.2.5
|   |   |-- hyper-rustls v0.27.3
|   |   |   |-- futures-util v0.3.31 (*)
|   |   |   |-- http v1.1.0 (*)
|   |   |   |-- hyper v1.5.0 (*)
|   |   |   |-- hyper-util v0.1.10
|   |   |   |   |-- bytes v1.8.0
|   |   |   |   |-- futures-channel v0.3.31 (*)
|   |   |   |   |-- futures-util v0.3.31 (*)
|   |   |   |   |-- http v1.1.0 (*)
|   |   |   |   |-- http-body v1.0.1 (*)
|   |   |   |   |-- hyper v1.5.0 (*)
|   |   |   |   |-- pin-project-lite v0.2.15
|   |   |   |   |-- socket2 v0.5.7 (*)
|   |   |   |   |-- tokio v1.41.0 (*)
|   |   |   |   |-- tower-service v0.3.3
|   |   |   |   `-- tracing v0.1.40 (*)
|   |   |   |-- rustls v0.23.16
|   |   |   |   |-- once_cell v1.20.2
|   |   |   |   |-- ring v0.17.8 (*)
|   |   |   |   |-- rustls-pki-types v1.10.0
|   |   |   |   |-- rustls-webpki v0.102.8
|   |   |   |   |   |-- ring v0.17.8 (*)
|   |   |   |   |   |-- rustls-pki-types v1.10.0
|   |   |   |   |   `-- untrusted v0.9.0
|   |   |   |   |-- subtle v2.6.1
|   |   |   |   `-- zeroize v1.8.1
|   |   |   |-- rustls-pki-types v1.10.0
|   |   |   |-- tokio v1.41.0 (*)
|   |   |   |-- tokio-rustls v0.26.0
|   |   |   |   |-- rustls v0.23.16 (*)
|   |   |   |   |-- rustls-pki-types v1.10.0
|   |   |   |   `-- tokio v1.41.0 (*)
|   |   |   |-- tower-service v0.3.3
|   |   |   `-- webpki-roots v0.26.6
|   |   |       `-- rustls-pki-types v1.10.0
|   |   |-- hyper-util v0.1.10 (*)
|   |   |-- ipnet v2.10.1
|   |   |-- log v0.4.22
|   |   |-- mime v0.3.17
|   |   |-- once_cell v1.20.2
|   |   |-- percent-encoding v2.3.1
|   |   |-- pin-project-lite v0.2.15
|   |   |-- rustls v0.23.16 (*)
|   |   |-- rustls-pemfile v2.2.0
|   |   |   `-- rustls-pki-types v1.10.0
|   |   |-- rustls-pki-types v1.10.0
|   |   |-- serde v1.0.214 (*)
|   |   |-- serde_urlencoded v0.7.1
|   |   |   |-- form_urlencoded v1.2.1
|   |   |   |   `-- percent-encoding v2.3.1
|   |   |   |-- itoa v1.0.11
|   |   |   |-- ryu v1.0.18
|   |   |   `-- serde v1.0.214 (*)
|   |   |-- sync_wrapper v1.0.1
|   |   |   `-- futures-core v0.3.31
|   |   |-- tokio v1.41.0 (*)
|   |   |-- tokio-rustls v0.26.0 (*)
|   |   |-- tokio-util v0.7.12 (*)
|   |   |-- tower-service v0.3.3
|   |   |-- url v2.5.3
|   |   |   |-- form_urlencoded v1.2.1 (*)
|   |   |   |-- idna v1.0.3
|   |   |   |   |-- idna_adapter v1.2.0
|   |   |   |   |   |-- icu_normalizer v1.5.0
|   |   |   |   |   |   |-- displaydoc v0.2.5 (proc-macro)
|   |   |   |   |   |   |   |-- proc-macro2 v1.0.89 (*)
|   |   |   |   |   |   |   |-- quote v1.0.37 (*)
|   |   |   |   |   |   |   `-- syn v2.0.87 (*)
|   |   |   |   |   |   |-- icu_collections v1.5.0
|   |   |   |   |   |   |   |-- displaydoc v0.2.5 (proc-macro) (*)
|   |   |   |   |   |   |   |-- yoke v0.7.4
|   |   |   |   |   |   |   |   |-- stable_deref_trait v1.2.0
|   |   |   |   |   |   |   |   |-- yoke-derive v0.7.4 (proc-macro)
|   |   |   |   |   |   |   |   |   |-- proc-macro2 v1.0.89 (*)
|   |   |   |   |   |   |   |   |   |-- quote v1.0.37 (*)
|   |   |   |   |   |   |   |   |   |-- syn v2.0.87 (*)
|   |   |   |   |   |   |   |   |   `-- synstructure v0.13.1
|   |   |   |   |   |   |   |   |       |-- proc-macro2 v1.0.89 (*)
|   |   |   |   |   |   |   |   |       |-- quote v1.0.37 (*)
|   |   |   |   |   |   |   |   |       `-- syn v2.0.87 (*)
|   |   |   |   |   |   |   |   `-- zerofrom v0.1.4
|   |   |   |   |   |   |   |       `-- zerofrom-derive v0.1.4 (proc-macro)
|   |   |   |   |   |   |   |           |-- proc-macro2 v1.0.89 (*)
|   |   |   |   |   |   |   |           |-- quote v1.0.37 (*)
|   |   |   |   |   |   |   |           |-- syn v2.0.87 (*)
|   |   |   |   |   |   |   |           `-- synstructure v0.13.1 (*)
|   |   |   |   |   |   |   |-- zerofrom v0.1.4 (*)
|   |   |   |   |   |   |   `-- zerovec v0.10.4
|   |   |   |   |   |   |       |-- yoke v0.7.4 (*)
|   |   |   |   |   |   |       |-- zerofrom v0.1.4 (*)
|   |   |   |   |   |   |       `-- zerovec-derive v0.10.3 (proc-macro)
|   |   |   |   |   |   |           |-- proc-macro2 v1.0.89 (*)
|   |   |   |   |   |   |           |-- quote v1.0.37 (*)
|   |   |   |   |   |   |           `-- syn v2.0.87 (*)
|   |   |   |   |   |   |-- icu_normalizer_data v1.5.0
|   |   |   |   |   |   |-- icu_properties v1.5.1
|   |   |   |   |   |   |   |-- displaydoc v0.2.5 (proc-macro) (*)
|   |   |   |   |   |   |   |-- icu_collections v1.5.0 (*)
|   |   |   |   |   |   |   |-- icu_locid_transform v1.5.0
|   |   |   |   |   |   |   |   |-- displaydoc v0.2.5 (proc-macro) (*)
|   |   |   |   |   |   |   |   |-- icu_locid v1.5.0
|   |   |   |   |   |   |   |   |   |-- displaydoc v0.2.5 (proc-macro) (*)
|   |   |   |   |   |   |   |   |   |-- litemap v0.7.3
|   |   |   |   |   |   |   |   |   |-- tinystr v0.7.6
|   |   |   |   |   |   |   |   |   |   |-- displaydoc v0.2.5 (proc-macro) (*)
|   |   |   |   |   |   |   |   |   |   `-- zerovec v0.10.4 (*)
|   |   |   |   |   |   |   |   |   |-- writeable v0.5.5
|   |   |   |   |   |   |   |   |   `-- zerovec v0.10.4 (*)
|   |   |   |   |   |   |   |   |-- icu_locid_transform_data v1.5.0
|   |   |   |   |   |   |   |   |-- icu_provider v1.5.0
|   |   |   |   |   |   |   |   |   |-- displaydoc v0.2.5 (proc-macro) (*)
|   |   |   |   |   |   |   |   |   |-- icu_locid v1.5.0 (*)
|   |   |   |   |   |   |   |   |   |-- icu_provider_macros v1.5.0 (proc-macro)
|   |   |   |   |   |   |   |   |   |   |-- proc-macro2 v1.0.89 (*)
|   |   |   |   |   |   |   |   |   |   |-- quote v1.0.37 (*)
|   |   |   |   |   |   |   |   |   |   `-- syn v2.0.87 (*)
|   |   |   |   |   |   |   |   |   |-- stable_deref_trait v1.2.0
|   |   |   |   |   |   |   |   |   |-- tinystr v0.7.6 (*)
|   |   |   |   |   |   |   |   |   |-- writeable v0.5.5
|   |   |   |   |   |   |   |   |   |-- yoke v0.7.4 (*)
|   |   |   |   |   |   |   |   |   |-- zerofrom v0.1.4 (*)
|   |   |   |   |   |   |   |   |   `-- zerovec v0.10.4 (*)
|   |   |   |   |   |   |   |   |-- tinystr v0.7.6 (*)
|   |   |   |   |   |   |   |   `-- zerovec v0.10.4 (*)
|   |   |   |   |   |   |   |-- icu_properties_data v1.5.0
|   |   |   |   |   |   |   |-- icu_provider v1.5.0 (*)
|   |   |   |   |   |   |   |-- tinystr v0.7.6 (*)
|   |   |   |   |   |   |   `-- zerovec v0.10.4 (*)
|   |   |   |   |   |   |-- icu_provider v1.5.0 (*)
|   |   |   |   |   |   |-- smallvec v1.13.2
|   |   |   |   |   |   |-- utf16_iter v1.0.5
|   |   |   |   |   |   |-- utf8_iter v1.0.4
|   |   |   |   |   |   |-- write16 v1.0.0
|   |   |   |   |   |   `-- zerovec v0.10.4 (*)
|   |   |   |   |   `-- icu_properties v1.5.1 (*)
|   |   |   |   |-- smallvec v1.13.2
|   |   |   |   `-- utf8_iter v1.0.4
|   |   |   `-- percent-encoding v2.3.1
|   |   `-- webpki-roots v0.26.6 (*)
|   |-- serde v1.0.214 (*)
|   |-- serde_json v1.0.132 (*)
|   |-- thiserror v1.0.68
|   |   `-- thiserror-impl v1.0.68 (proc-macro)
|   |       |-- proc-macro2 v1.0.89 (*)
|   |       |-- quote v1.0.37 (*)
|   |       `-- syn v2.0.87 (*)
|   |-- time v0.3.36
|   |   |-- deranged v0.3.11
|   |   |   |-- powerfmt v0.2.0
|   |   |   `-- serde v1.0.214 (*)
|   |   |-- itoa v1.0.11
|   |   |-- num-conv v0.1.0
|   |   |-- powerfmt v0.2.0
|   |   |-- serde v1.0.214 (*)
|   |   |-- time-core v0.1.2
|   |   `-- time-macros v0.2.18 (proc-macro)
|   |       |-- num-conv v0.1.0
|   |       `-- time-core v0.1.2
|   |-- uuid v1.11.0
|   |   `-- getrandom v0.2.15 (*)
|   `-- yaup v0.3.1
|       |-- form_urlencoded v1.2.1 (*)
|       |-- serde v1.0.214 (*)
|       `-- thiserror v1.0.68 (*)
|-- mime_guess v2.0.5
|   |-- mime v0.3.17
|   `-- unicase v2.8.0
|   [build-dependencies]
|   `-- unicase v2.8.0
|-- reqwest v0.11.27
|   |-- base64 v0.21.7
|   |-- bytes v1.8.0
|   |-- encoding_rs v0.8.35
|   |   `-- cfg-if v1.0.0
|   |-- futures-core v0.3.31
|   |-- futures-util v0.3.31 (*)
|   |-- h2 v0.3.26
|   |   |-- bytes v1.8.0
|   |   |-- fnv v1.0.7
|   |   |-- futures-core v0.3.31
|   |   |-- futures-sink v0.3.31
|   |   |-- futures-util v0.3.31 (*)
|   |   |-- http v0.2.12
|   |   |   |-- bytes v1.8.0
|   |   |   |-- fnv v1.0.7
|   |   |   `-- itoa v1.0.11
|   |   |-- indexmap v2.6.0 (*)
|   |   |-- slab v0.4.9 (*)
|   |   |-- tokio v1.41.0 (*)
|   |   |-- tokio-util v0.7.12 (*)
|   |   `-- tracing v0.1.40 (*)
|   |-- http v0.2.12 (*)
|   |-- http-body v0.4.6
|   |   |-- bytes v1.8.0
|   |   |-- http v0.2.12 (*)
|   |   `-- pin-project-lite v0.2.15
|   |-- hyper v0.14.31
|   |   |-- bytes v1.8.0
|   |   |-- futures-channel v0.3.31 (*)
|   |   |-- futures-core v0.3.31
|   |   |-- futures-util v0.3.31 (*)
|   |   |-- h2 v0.3.26 (*)
|   |   |-- http v0.2.12 (*)
|   |   |-- http-body v0.4.6 (*)
|   |   |-- httparse v1.9.5
|   |   |-- httpdate v1.0.3
|   |   |-- itoa v1.0.11
|   |   |-- pin-project-lite v0.2.15
|   |   |-- socket2 v0.5.7 (*)
|   |   |-- tokio v1.41.0 (*)
|   |   |-- tower-service v0.3.3
|   |   |-- tracing v0.1.40 (*)
|   |   `-- want v0.3.1 (*)
|   |-- hyper-tls v0.5.0
|   |   |-- bytes v1.8.0
|   |   |-- hyper v0.14.31 (*)
|   |   |-- native-tls v0.2.12
|   |   |   |-- log v0.4.22
|   |   |   |-- openssl v0.10.68
|   |   |   |   |-- bitflags v2.6.0
|   |   |   |   |-- cfg-if v1.0.0
|   |   |   |   |-- foreign-types v0.3.2
|   |   |   |   |   `-- foreign-types-shared v0.1.1
|   |   |   |   |-- libc v0.2.161
|   |   |   |   |-- once_cell v1.20.2
|   |   |   |   |-- openssl-macros v0.1.1 (proc-macro)
|   |   |   |   |   |-- proc-macro2 v1.0.89 (*)
|   |   |   |   |   |-- quote v1.0.37 (*)
|   |   |   |   |   `-- syn v2.0.87 (*)
|   |   |   |   `-- openssl-sys v0.9.104
|   |   |   |       `-- libc v0.2.161
|   |   |   |       [build-dependencies]
|   |   |   |       |-- cc v1.1.34 (*)
|   |   |   |       |-- pkg-config v0.3.31
|   |   |   |       `-- vcpkg v0.2.15
|   |   |   |-- openssl-probe v0.1.5
|   |   |   `-- openssl-sys v0.9.104 (*)
|   |   |-- tokio v1.41.0 (*)
|   |   `-- tokio-native-tls v0.3.1
|   |       |-- native-tls v0.2.12 (*)
|   |       `-- tokio v1.41.0 (*)
|   |-- ipnet v2.10.1
|   |-- log v0.4.22
|   |-- mime v0.3.17
|   |-- native-tls v0.2.12 (*)
|   |-- once_cell v1.20.2
|   |-- percent-encoding v2.3.1
|   |-- pin-project-lite v0.2.15
|   |-- rustls-pemfile v1.0.4
|   |   `-- base64 v0.21.7
|   |-- serde v1.0.214 (*)
|   |-- serde_json v1.0.132 (*)
|   |-- serde_urlencoded v0.7.1 (*)
|   |-- sync_wrapper v0.1.2
|   |-- tokio v1.41.0 (*)
|   |-- tokio-native-tls v0.3.1 (*)
|   |-- tower-service v0.3.3
|   `-- url v2.5.3 (*)
|-- rocket v0.5.1
|   |-- async-stream v0.3.6
|   |   |-- async-stream-impl v0.3.6 (proc-macro)
|   |   |   |-- proc-macro2 v1.0.89 (*)
|   |   |   |-- quote v1.0.37 (*)
|   |   |   `-- syn v2.0.87 (*)
|   |   |-- futures-core v0.3.31
|   |   `-- pin-project-lite v0.2.15
|   |-- async-trait v0.1.83 (proc-macro) (*)
|   |-- atomic v0.5.3
|   |-- binascii v0.1.4
|   |-- bytes v1.8.0
|   |-- either v1.13.0 (*)
|   |-- figment v0.10.19
|   |   |-- pear v0.2.9
|   |   |   |-- inlinable_string v0.1.15
|   |   |   |-- pear_codegen v0.2.9 (proc-macro)
|   |   |   |   |-- proc-macro2 v1.0.89 (*)
|   |   |   |   |-- proc-macro2-diagnostics v0.10.1
|   |   |   |   |   |-- proc-macro2 v1.0.89 (*)
|   |   |   |   |   |-- quote v1.0.37 (*)
|   |   |   |   |   |-- syn v2.0.87 (*)
|   |   |   |   |   `-- yansi v1.0.1
|   |   |   |   |   [build-dependencies]
|   |   |   |   |   `-- version_check v0.9.5
|   |   |   |   |-- quote v1.0.37 (*)
|   |   |   |   `-- syn v2.0.87 (*)
|   |   |   `-- yansi v1.0.1
|   |   |       `-- is-terminal v0.4.13
|   |   |           `-- libc v0.2.161
|   |   |-- serde v1.0.214 (*)
|   |   |-- toml v0.8.19
|   |   |   |-- serde v1.0.214 (*)
|   |   |   |-- serde_spanned v0.6.8
|   |   |   |   `-- serde v1.0.214 (*)
|   |   |   |-- toml_datetime v0.6.8
|   |   |   |   `-- serde v1.0.214 (*)
|   |   |   `-- toml_edit v0.22.22
|   |   |       |-- indexmap v2.6.0 (*)
|   |   |       |-- serde v1.0.214 (*)
|   |   |       |-- serde_spanned v0.6.8 (*)
|   |   |       |-- toml_datetime v0.6.8 (*)
|   |   |       `-- winnow v0.6.20
|   |   `-- uncased v0.9.10
|   |       `-- serde v1.0.214 (*)
|   |       [build-dependencies]
|   |       `-- version_check v0.9.5
|   |   [build-dependencies]
|   |   `-- version_check v0.9.5
|   |-- futures v0.3.31 (*)
|   |-- indexmap v2.6.0 (*)
|   |-- log v0.4.22
|   |-- memchr v2.7.4
|   |-- multer v3.1.0
|   |   |-- bytes v1.8.0
|   |   |-- encoding_rs v0.8.35 (*)
|   |   |-- futures-util v0.3.31 (*)
|   |   |-- http v1.1.0 (*)
|   |   |-- httparse v1.9.5
|   |   |-- memchr v2.7.4
|   |   |-- mime v0.3.17
|   |   |-- spin v0.9.8
|   |   |-- tokio v1.41.0 (*)
|   |   `-- tokio-util v0.7.12 (*)
|   |   [build-dependencies]
|   |   `-- version_check v0.9.5
|   |-- num_cpus v1.16.0
|   |   `-- libc v0.2.161
|   |-- parking_lot v0.12.3 (*)
|   |-- pin-project-lite v0.2.15
|   |-- rand v0.8.5
|   |   |-- libc v0.2.161
|   |   |-- rand_chacha v0.3.1
|   |   |   |-- ppv-lite86 v0.2.20
|   |   |   |   `-- zerocopy v0.7.35
|   |   |   |       |-- byteorder v1.5.0
|   |   |   |       `-- zerocopy-derive v0.7.35 (proc-macro)
|   |   |   |           |-- proc-macro2 v1.0.89 (*)
|   |   |   |           |-- quote v1.0.37 (*)
|   |   |   |           `-- syn v2.0.87 (*)
|   |   |   `-- rand_core v0.6.4
|   |   |       `-- getrandom v0.2.15 (*)
|   |   `-- rand_core v0.6.4 (*)
|   |-- ref-cast v1.0.23
|   |   `-- ref-cast-impl v1.0.23 (proc-macro)
|   |       |-- proc-macro2 v1.0.89 (*)
|   |       |-- quote v1.0.37 (*)
|   |       `-- syn v2.0.87 (*)
|   |-- rocket_codegen v0.5.1 (proc-macro)
|   |   |-- devise v0.4.2
|   |   |   |-- devise_codegen v0.4.2 (proc-macro)
|   |   |   |   |-- devise_core v0.4.2
|   |   |   |   |   |-- bitflags v2.6.0
|   |   |   |   |   |-- proc-macro2 v1.0.89 (*)
|   |   |   |   |   |-- proc-macro2-diagnostics v0.10.1 (*)
|   |   |   |   |   |-- quote v1.0.37 (*)
|   |   |   |   |   `-- syn v2.0.87 (*)
|   |   |   |   `-- quote v1.0.37 (*)
|   |   |   `-- devise_core v0.4.2 (*)
|   |   |-- glob v0.3.1
|   |   |-- indexmap v2.6.0
|   |   |   |-- equivalent v1.0.1
|   |   |   `-- hashbrown v0.15.1
|   |   |-- proc-macro2 v1.0.89 (*)
|   |   |-- quote v1.0.37 (*)
|   |   |-- rocket_http v0.5.1
|   |   |   |-- cookie v0.18.1
|   |   |   |   |-- percent-encoding v2.3.1
|   |   |   |   `-- time v0.3.36
|   |   |   |       |-- deranged v0.3.11
|   |   |   |       |   `-- powerfmt v0.2.0
|   |   |   |       |-- itoa v1.0.11
|   |   |   |       |-- num-conv v0.1.0
|   |   |   |       |-- powerfmt v0.2.0
|   |   |   |       |-- time-core v0.1.2
|   |   |   |       `-- time-macros v0.2.18 (proc-macro) (*)
|   |   |   |   [build-dependencies]
|   |   |   |   `-- version_check v0.9.5
|   |   |   |-- either v1.13.0
|   |   |   |-- futures v0.3.31
|   |   |   |   |-- futures-channel v0.3.31 (*)
|   |   |   |   |-- futures-core v0.3.31
|   |   |   |   |-- futures-io v0.3.31
|   |   |   |   |-- futures-sink v0.3.31
|   |   |   |   |-- futures-task v0.3.31
|   |   |   |   `-- futures-util v0.3.31
|   |   |   |       |-- futures-core v0.3.31
|   |   |   |       |-- futures-sink v0.3.31
|   |   |   |       |-- futures-task v0.3.31
|   |   |   |       |-- pin-project-lite v0.2.15
|   |   |   |       `-- pin-utils v0.1.0
|   |   |   |-- http v0.2.12 (*)
|   |   |   |-- hyper v0.14.31
|   |   |   |   |-- bytes v1.8.0
|   |   |   |   |-- futures-channel v0.3.31 (*)
|   |   |   |   |-- futures-core v0.3.31
|   |   |   |   |-- futures-util v0.3.31 (*)
|   |   |   |   |-- http v0.2.12 (*)
|   |   |   |   |-- http-body v0.4.6 (*)
|   |   |   |   |-- httparse v1.9.5
|   |   |   |   |-- httpdate v1.0.3
|   |   |   |   |-- itoa v1.0.11
|   |   |   |   |-- pin-project-lite v0.2.15
|   |   |   |   |-- socket2 v0.5.7 (*)
|   |   |   |   |-- tokio v1.41.0
|   |   |   |   |   |-- libc v0.2.161
|   |   |   |   |   |-- mio v1.0.2 (*)
|   |   |   |   |   |-- pin-project-lite v0.2.15
|   |   |   |   |   `-- socket2 v0.5.7 (*)
|   |   |   |   |-- tower-service v0.3.3
|   |   |   |   |-- tracing v0.1.40 (*)
|   |   |   |   `-- want v0.3.1 (*)
|   |   |   |-- indexmap v2.6.0 (*)
|   |   |   |-- log v0.4.22
|   |   |   |-- memchr v2.7.4
|   |   |   |-- pear v0.2.9 (*)
|   |   |   |-- percent-encoding v2.3.1
|   |   |   |-- pin-project-lite v0.2.15
|   |   |   |-- ref-cast v1.0.23 (*)
|   |   |   |-- smallvec v1.13.2
|   |   |   |-- stable-pattern v0.1.0
|   |   |   |   `-- memchr v2.7.4
|   |   |   |-- state v0.6.0
|   |   |   |-- time v0.3.36 (*)
|   |   |   |-- tokio v1.41.0 (*)
|   |   |   `-- uncased v0.9.10
|   |   |       [build-dependencies]
|   |   |       `-- version_check v0.9.5
|   |   |-- syn v2.0.87 (*)
|   |   |-- unicode-xid v0.2.6
|   |   `-- version_check v0.9.5
|   |-- rocket_http v0.5.1
|   |   |-- cookie v0.18.1 (*)
|   |   |-- either v1.13.0 (*)
|   |   |-- futures v0.3.31 (*)
|   |   |-- http v0.2.12 (*)
|   |   |-- hyper v0.14.31 (*)
|   |   |-- indexmap v2.6.0 (*)
|   |   |-- log v0.4.22
|   |   |-- memchr v2.7.4
|   |   |-- pear v0.2.9 (*)
|   |   |-- percent-encoding v2.3.1
|   |   |-- pin-project-lite v0.2.15
|   |   |-- ref-cast v1.0.23 (*)
|   |   |-- serde v1.0.214 (*)
|   |   |-- smallvec v1.13.2
|   |   |-- stable-pattern v0.1.0 (*)
|   |   |-- state v0.6.0
|   |   |-- time v0.3.36 (*)
|   |   |-- tokio v1.41.0 (*)
|   |   `-- uncased v0.9.10 (*)
|   |-- serde v1.0.214 (*)
|   |-- serde_json v1.0.132 (*)
|   |-- state v0.6.0
|   |-- tempfile v3.13.0
|   |   |-- cfg-if v1.0.0
|   |   |-- fastrand v2.1.1
|   |   |-- once_cell v1.20.2
|   |   `-- rustix v0.38.39
|   |       |-- bitflags v2.6.0
|   |       `-- linux-raw-sys v0.4.14
|   |-- time v0.3.36 (*)
|   |-- tokio v1.41.0 (*)
|   |-- tokio-stream v0.1.16
|   |   |-- futures-core v0.3.31
|   |   |-- pin-project-lite v0.2.15
|   |   `-- tokio v1.41.0 (*)
|   |-- tokio-util v0.7.12 (*)
|   |-- ubyte v0.10.4
|   |   `-- serde v1.0.214 (*)
|   `-- yansi v1.0.1 (*)
|   [build-dependencies]
|   `-- version_check v0.9.5
|-- rocket_dyn_templates v0.2.0
|   |-- normpath v1.3.0
|   |-- notify v6.1.1
|   |   |-- crossbeam-channel v0.5.13
|   |   |   `-- crossbeam-utils v0.8.20
|   |   |-- filetime v0.2.25
|   |   |   |-- cfg-if v1.0.0
|   |   |   `-- libc v0.2.161
|   |   |-- inotify v0.9.6
|   |   |   |-- bitflags v1.3.2
|   |   |   |-- inotify-sys v0.1.5
|   |   |   |   `-- libc v0.2.161
|   |   |   `-- libc v0.2.161
|   |   |-- libc v0.2.161
|   |   |-- log v0.4.22
|   |   |-- mio v0.8.11
|   |   |   |-- libc v0.2.161
|   |   |   `-- log v0.4.22
|   |   `-- walkdir v2.5.0
|   |       `-- same-file v1.0.6
|   |-- rocket v0.5.1 (*)
|   `-- walkdir v2.5.0 (*)
|-- rusqlite v0.26.3
|   |-- bitflags v1.3.2
|   |-- fallible-iterator v0.2.0
|   |-- fallible-streaming-iterator v0.1.9
|   |-- hashlink v0.7.0
|   |   `-- hashbrown v0.11.2
|   |       `-- ahash v0.7.8
|   |           |-- getrandom v0.2.15 (*)
|   |           `-- once_cell v1.20.2
|   |           [build-dependencies]
|   |           `-- version_check v0.9.5
|   |-- libsqlite3-sys v0.23.2
|   |   [build-dependencies]
|   |   |-- pkg-config v0.3.31
|   |   `-- vcpkg v0.2.15
|   |-- memchr v2.7.4
|   `-- smallvec v1.13.2
|-- serde v1.0.214 (*)
|-- sha2 v0.10.8
|   |-- cfg-if v1.0.0
|   |-- cpufeatures v0.2.14
|   `-- digest v0.10.7
|       |-- block-buffer v0.10.4
|       |   `-- generic-array v0.14.7
|       |       `-- typenum v1.17.0
|       |       [build-dependencies]
|       |       `-- version_check v0.9.5
|       `-- crypto-common v0.1.6
|           |-- generic-array v0.14.7 (*)
|           `-- typenum v1.17.0
```

```
|-- tera v1.20.0
|   |-- chrono v0.4.38 (*)
|   |-- chrono-tz v0.9.0
|   |   |-- chrono v0.4.38 (*)
|   |   `-- phf v0.11.2
|   |       `-- phf_shared v0.11.2
|   |           `-- siphasher v0.3.11
|   |   [build-dependencies]
|   |   `-- chrono-tz-build v0.3.0
|   |       |-- parse-zoneinfo v0.3.1
|   |       |   `-- regex v1.11.1
|   |       |       |-- regex-automata v0.4.8
|   |       |       |   `-- regex-syntax v0.8.5
|   |       |       `-- regex-syntax v0.8.5
|   |       |-- phf v0.11.2 (*)
|   |       `-- phf_codegen v0.11.2
|   |           |-- phf_generator v0.11.2
|   |           |   |-- phf_shared v0.11.2 (*)
|   |           |   `-- rand v0.8.5
|   |           |       `-- rand_core v0.6.4
|   |           `-- phf_shared v0.11.2 (*)
|   |-- globwalk v0.9.1
|   |   |-- bitflags v2.6.0
|   |   |-- ignore v0.4.23
|   |   |   |-- crossbeam-deque v0.8.5
|   |   |   |   |-- crossbeam-epoch v0.9.18
|   |   |   |   |   `-- crossbeam-utils v0.8.20
|   |   |   |   `-- crossbeam-utils v0.8.20
|   |   |   |-- globset v0.4.15
|   |   |   |   |-- aho-corasick v1.1.3
|   |   |   |   |   `-- memchr v2.7.4
|   |   |   |   |-- bstr v1.10.0
|   |   |   |   |   `-- memchr v2.7.4
|   |   |   |   |-- log v0.4.22
|   |   |   |   |-- regex-automata v0.4.8
|   |   |   |   |   |-- aho-corasick v1.1.3 (*)
|   |   |   |   |   |-- memchr v2.7.4
|   |   |   |   |   `-- regex-syntax v0.8.5
|   |   |   |   `-- regex-syntax v0.8.5
|   |   |   |-- log v0.4.22
|   |   |   |-- memchr v2.7.4
|   |   |   |-- regex-automata v0.4.8 (*)
|   |   |   |-- same-file v1.0.6
|   |   |   `-- walkdir v2.5.0 (*)
|   |   `-- walkdir v2.5.0 (*)
|   |-- humansize v2.1.3
|   |   `-- libm v0.2.11
|   |-- lazy_static v1.5.0
|   |-- percent-encoding v2.3.1
|   |-- pest v2.7.14
|   |   |-- memchr v2.7.4
|   |   |-- thiserror v1.0.68 (*)
|   |   `-- ucd-trie v0.1.7
|   |-- pest_derive v2.7.14 (proc-macro)
|   |   |-- pest v2.7.14 (*)
|   |   `-- pest_generator v2.7.14
|   |       |-- pest v2.7.14 (*)
|   |       |-- pest_meta v2.7.14
|   |       |   |-- once_cell v1.20.2
|   |       |   `-- pest v2.7.14 (*)
|   |       |   [build-dependencies]
|   |       |   `-- sha2 v0.10.8
|   |       |       |-- cfg-if v1.0.0
|   |       |       |-- cpufeatures v0.2.14
|   |       |       `-- digest v0.10.7
|   |       |           |-- block-buffer v0.10.4 (*)
|   |       |           `-- crypto-common v0.1.6
|   |       |               |-- generic-array v0.14.7 (*)
|   |       |               `-- typenum v1.17.0
|   |       |-- proc-macro2 v1.0.89 (*)
|   |       |-- quote v1.0.37 (*)
|   |       `-- syn v2.0.87 (*)
|   |-- rand v0.8.5 (*)
|   |-- regex v1.11.1
|   |   |-- aho-corasick v1.1.3 (*)
|   |   |-- memchr v2.7.4
|   |   |-- regex-automata v0.4.8 (*)
|   |   `-- regex-syntax v0.8.5
|   |-- serde v1.0.214 (*)
|   |-- serde_json v1.0.132 (*)
|   |-- slug v0.1.6
|   |   `-- deunicode v1.6.0
|   `-- unic-segment v0.9.0
|       `-- unic-ucd-segment v0.9.0
|           |-- unic-char-property v0.9.0
|           |   `-- unic-char-range v0.9.0
|           |-- unic-char-range v0.9.0
|           `-- unic-ucd-version v0.9.0
|               `-- unic-common v0.9.0
|-- thiserror v1.0.68 (*)
`-- tokio v1.41.0 (*)
```

7.5 Raw Code

7.6 templates/login.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login</title>
</head>
<body>
    <h1>Login</h1>
    <form action="/login" method="post">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username" required><br><br>
        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required><br><br>
        <button type="submit">Login</button>
    </form>
</body>
</html>
```

7.7 templates/test.html

```html
<!-- templates/test.html -->
<!DOCTYPE html>
<html lang="en">
<body>
    <h1>Test Template</h1>
    <p>This is a test template.</p>
</body>
</html>
```

7.8 src/main.rs

```rust
use personal_document_scrapper::web_server::{index, login_page, login,
↪  config_page, handle_query};
use personal_document_scrapper::crawler::Crawler;
use personal_document_scrapper::indexer::Indexer;
use personal_document_scrapper::metadata::MetadataManager;
use personal_document_scrapper::scheduler::Scheduler;
use personal_document_scrapper::logger::Logger;
use personal_document_scrapper::notification::NotificationManager;
use personal_document_scrapper::backup::BackupManager;
use personal_document_scrapper::health_checker::HealthChecker;

use clap::{App, Arg};
use rocket::routes;
use rocket::fs::FileServer;
use tokio::task;
use std::process::{Command, Stdio};
use std::sync::Arc;
use std::thread;
use std::time::Duration;

#[rocket::main]
async fn main() -> Result<(), Box<dyn std::error::Error>> {
    // Parse CLI arguments
    let matches = App::new("Personal Document Scrapper")
        .version("1.0")
        .about("A system to scrape, index, and analyze personal
        ↪  documents.")
        .arg(
            Arg::new("meilisearch_url")
                .short('m')
                .long("meilisearch")
                .value_name("URL")
                .default_value("http://localhost:7700")
                .about("URL of the Meilisearch instance")
        )
        .arg(
            Arg::new("database_path")
                .short('d')
                .long("database")
                .value_name("PATH")
                .default_value("./sample_files/database.db")
                .about("Path to the SQLite database file")
        )
        .arg(
            Arg::new("static_dir")
                .short('s')
                .long("static")
                .value_name("DIR")
                .default_value("static")
                .about("Path to the directory containing static files")
        )
        .get_matches();

    // Extract CLI argument values
    let meilisearch_url = matches.value_of("meilisearch_url").unwrap().to_string();
    let database_path = matches.value_of("database_path").unwrap().to_string();
    let static_dir = matches.value_of("static_dir").unwrap().to_string();

    let logger = Logger::new("logs/system.log").expect("Failed to initialize logger");
    logger.info("Starting the Personal Document Scrapper system...");

    // Start Meilisearch in a separate thread
    let meilisearch_url_clone = meilisearch_url.clone();
    let meilisearch_thread = thread::spawn(move || {
        logger.info("Starting Meilisearch...");
        let meilisearch_process = Command::new("meilisearch")
            .arg("--master-key")
            .arg("masterKey")
            .stdin(Stdio::null())
            .stdout(Stdio::null())
            .stderr(Stdio::null())
            .spawn();

        match meilisearch_process {
            Ok(_) => logger.info("Meilisearch started successfully."),
            Err(e) => logger.error(&format!("Failed to start Meilisearch: {}", e)),
        }
    });

    // Metadata manager instance
    let metadata_manager = Arc::new(MetadataManager::new(&database_path)?);

    // Start the crawler in a separate thread
    let crawler_manager = Arc::clone(&metadata_manager);
    let crawler_thread = thread::spawn(move || {
        let crawler = Crawler::new(
            vec!["./sample_files/documents".into()],
            vec!["./sample_files/tmp".into()],
        );
        crawler.start_crawler();
        logger.info("Crawler started.");
    });

    // Start the indexer in another thread
    let indexer_manager = Arc::clone(&metadata_manager);
    let indexer_thread = thread::spawn(move || {
        let indexer = Indexer::new(indexer_manager);
        if let Err(e) = indexer.start_indexing_thread() {
            logger.error(&format!("Failed to start indexer: {}", e));
        }
        logger.info("Indexer started.");
    });

    // Start periodic backup using the Scheduler
    let backup_scheduler = Scheduler::new(Duration::from_secs(3600)); // Every hour
    let backup_manager = BackupManager::new(&database_path, "./sample_files/backups");
    let backup_thread = thread::spawn(move || {
        backup_scheduler.schedule(move || {
            if let Err(e) = backup_manager.create_backup() {
                logger.error(&format!("Backup failed: {}", e));
            } else {
                logger.info("Backup created successfully.");
            }
        });
    });

    // Health checker monitoring
    let health_checker = HealthChecker::new(&meilisearch_url, &database_path);
    let health_checker_thread = thread::spawn(move || {
        tokio::runtime::Runtime::new()
            .unwrap()
            .block_on(async { health_checker.monitor_health(Duration::from_secs(30)).await });
    });

    // Start the Rocket web server in the main thread
    logger.info("Starting Rocket web server...");
    let rocket_thread = task::spawn(async move {
        let _rocket = rocket::build()
            .mount("/", routes![index, login_page, login, config_page, handle_query])
            .mount("/static", FileServer::from(&static_dir))
            .launch()
```

```rust
            .await;

        match _rocket {
            Ok(_) => logger.info("Rocket server started successfully."),
            Err(e) => logger.error(&format!("Rocket server failed to start: {}", e)),
        }
    });

    // Join all threads
    meilisearch_thread.join().expect("Meilisearch thread failed");
    crawler_thread.join().expect("Crawler thread failed");
    indexer_thread.join().expect("Indexer thread failed");
    backup_thread.join().expect("Backup thread failed");
    health_checker_thread.join().expect("Health checker thread failed");
    rocket_thread.await.expect("Rocket server thread failed");

    logger.info("System shutdown completed.");
    Ok(())
}
```

7.9 src/textract.rs

```rust
use std::fs::File;
use std::io::{self, Read};
use mime_guess::from_path;
use reqwest::blocking::Client;
pub struct TextExtractor {
    tika_server_url: String,
}

impl TextExtractor {
    pub fn new(tika_server_url: &str) -> Self {
        TextExtractor {
            tika_server_url: tika_server_url.to_string(),
        }
    }

    pub fn extract_raw_text(&self, file_path: &str) -> Result<String, String> {
        let mime_type = from_path(file_path).first_or_octet_stream().to_string();
        println!("Mime type: {}", mime_type);
        let mut file = File::open(file_path).map_err(|e| format!("Failed to open file: {}", e))?;
        let mut buffer = Vec::new();
        file.read_to_end(&mut buffer).map_err(|e| format!("Failed to read file: {}", e))?;

        let client = Client::new();
        let response = client
            .put(format!("{}/tika", self.tika_server_url))
            .header("Content-Type", mime_type)
            .header("Accept", "text/plain")
            .body(buffer)
            .send()
            .map_err(|e| format!("Failed to send request to Tika server: {}", e))?;

        if response.status().is_success() {
            let extracted_text = response.text().map_err(|e| format!("Failed to read response text: {}", e))?;
            Ok(extracted_text)
        } else {
            Err(format!("Tika server returned an error: {}", response.status()))
        }
    }
}
```

```
#[cfg(test)]
mod tests {
    use super::*;
    use std::fs::write;

    #[test]
    fn test_new_text_extractor() {
        let extractor = TextExtractor::new("http://localhost:9998");
        assert_eq!(extractor.tika_server_url, "http://localhost:9998");
    }

    #[test]
    fn test_extract_raw_text_success() {
        let test_file_path = "test.txt";
        write(test_file_path, "Test content").expect("Failed to write
        ↪  test file");
        let extractor = TextExtractor::new("http://localhost:9998");
        let result = extractor.extract_raw_text(test_file_path);
        assert!(result.is_ok());
        let _ = std::fs::remove_file(test_file_path);
    }

    #[test]
    fn test_extract_raw_text_file_not_found() {
        let extractor = TextExtractor::new("http://localhost:9998");
        let result =
        ↪  extractor.extract_raw_text("../sample_files/sample.txt");
        assert!(result.is_err());
        assert_eq!(result.unwrap_err(), "Failed to open file: No such
        ↪  file or directory (os error 2)");
        }


}
```

7.10 src/web_server.rs

```rust
use rocket::form::Form;
use rocket::http::{Cookie, CookieJar};
use rocket::response::Redirect;
use rocket::fs::{NamedFile};
use rocket::serde::json::Json;
use serde::Serialize;
use std::path::Path;
use crate::auth::{AuthManager, AuthError};
use crate::search::SearchClient;

#[derive(FromForm)]
struct LoginData {
    username: String,
    password: String,
}

#[derive(Serialize)]
struct SearchResult {
    message: String,
    results: Vec<String>,
}

#[post("/login", data = "<login_data>")]
pub fn login(login_data: Form<LoginData>, cookies: &CookieJar<'_>, auth_manager: &AuthManager) -> Redirect {
    match auth_manager.verify_user(&login_data.username, &login_data.password) {
        Ok(user) => {
            cookies.add(Cookie::new("user_id", user.id.to_string()));
            Redirect::to("/")
        }
        Err(AuthError::UserNotFound) | Err(AuthError::InvalidPassword) => {
            println!("Invalid username or password");
            Redirect::to("/login")
        }
        Err(AuthError::DatabaseError(e)) => {
            println!("Database error occurred: {}", e);
            Redirect::to("/login")
        }
    }
}

#[get("/login")]
pub async fn login_page() -> Option<NamedFile> {
    NamedFile::open(Path::new("static/login.html")).await.ok()
```

```
}

#[get("/")]
pub async fn index(cookies: &CookieJar<'_>) -> Option<NamedFile> {
    if cookies.get("user_id").is_some() {
        NamedFile::open(Path::new("static/index.html")).await.ok()
    } else {
        NamedFile::open(Path::new("static/login.html")).await.ok()
    }
}

#[get("/config")]
pub async fn config_page(cookies: &CookieJar<'_>) -> Option<NamedFile> {
    if cookies.get("user_id").is_some() {
        NamedFile::open(Path::new("static/config.html")).await.ok()
    } else {
        NamedFile::open(Path::new("static/login.html")).await.ok()
    }
}

#[get("/analytics")]
pub async fn analytics_page(cookies: &CookieJar<'_>) -> Option<NamedFile> {
    if cookies.get("user_id").is_some() {
        NamedFile::open(Path::new("static/analytics.html")).await.ok()
    } else {
        NamedFile::open(Path::new("static/login.html")).await.ok()
    }
}

#[get("/search?<query>")]
pub async fn handle_query(query: String, cookies: &CookieJar<'_>, search_client: &SearchClient) -> Json<SearchResult> {
    if cookies.get("user_id").is_none() {
        return Json(SearchResult {
            message: "User not logged in.".to_string(),
            results: vec![],
        });
    }

    match search_client.find_query(&query).await {
        Ok(results) => {
            let document_titles = results.hits.into_iter().map(|doc| doc.title).collect();
            Json(SearchResult {
                message: "Search successful".to_string(),
                results: document_titles,
            })
        }
        Err(_) => Json(SearchResult {
            message: "Error occurred during search.".to_string(),
            results: vec![],
        }),
    }
}

#[cfg(test)]
mod tests {
    use super::*;
    use rocket::local::blocking::Client;
    use rocket::{Build, Rocket};

    fn rocket_instance() -> Rocket<Build> {
        rocket::build()
            .mount(
                "/",
                routes![
                    login, login_page, index, config_page, analytics_page, handle_query
                ],
            )
    }

    #[test]
    fn test_login_page() {
        let client = Client::tracked(rocket_instance()).expect("valid rocket instance");
        let response = client.get("/login").dispatch();
        assert_eq!(response.status(), rocket::http::Status::Ok);
    }

    #[test]
    fn test_index_not_logged_in() {
        let client = Client::tracked(rocket_instance()).expect("valid rocket instance");
        let response = client.get("/").dispatch();
        assert_eq!(response.status(), rocket::http::Status::Ok);
    }

    #[test]
    fn test_config_page_not_logged_in() {
        let client = Client::tracked(rocket_instance()).expect("valid rocket instance");
        let response = client.get("/config").dispatch();
        assert_eq!(response.status(), rocket::http::Status::Ok);
    }

    #[test]
    fn test_analytics_page_not_logged_in() {
        let client = Client::tracked(rocket_instance()).expect("valid rocket instance");
        let response = client.get("/analytics").dispatch();
        assert_eq!(response.status(), rocket::http::Status::Ok);
    }

    #[test]
    fn test_handle_query_not_logged_in() {
        let client = Client::tracked(rocket_instance()).expect("valid rocket instance");
        let response = client.get("/search?query=test").dispatch();
        assert_eq!(response.status(), rocket::http::Status::Ok);
    }
}
```

7.11 src/backup.rs

```rust
use std::fs;
use std::io::{self, Write};
use std::path::Path;
use std::time::{SystemTime, UNIX_EPOCH};
use std::error::Error;
use tokio::task;

pub struct BackupManager {
    db_path: String,
    backup_dir: String,
    index_backup_dir: String,
}

impl BackupManager {
    pub fn new(db_path: &str, backup_dir: &str, index_backup_dir: &str) -> Self {
        Self {
            db_path: db_path.to_string(),
            backup_dir: backup_dir.to_string(),
            index_backup_dir: index_backup_dir.to_string(),
        }
    }

    pub async fn backup_database(&self) -> Result<String, Box<dyn Error>> {
        let timestamp = Self::get_timestamp();
        let backup_file = format!("{}/db_backup_{}.sqlite", self.backup_dir, timestamp);

        fs::copy(&self.db_path, &backup_file)?;
        println!("[BackupManager] Database backup created: {}", backup_file);
        Ok(backup_file)
    }

    pub async fn backup_indexes(&self) -> Result<String, Box<dyn Error>> {
        let timestamp = Self::get_timestamp();
        let backup_dir = format!("{}/indexes_{}", self.index_backup_dir, timestamp);

        fs::create_dir_all(&backup_dir)?;
        // Simulating Meilisearch index backup; replace with actual API calls as needed.
        println!("[BackupManager] Backing up Meilisearch indexes to: {}", backup_dir);

        // Placeholder for index backup logic
        fs::write(format!("{}/index_placeholder.json", backup_dir),
         ↪  "{}")?;
        Ok(backup_dir)
    }

    pub async fn restore_database(&self, backup_file: &str) -> Result<(),
     ↪  Box<dyn Error>> {
        fs::copy(backup_file, &self.db_path)?;
        println!("[BackupManager] Database restored from: {}",
         ↪  backup_file);
        Ok(())
    }

    pub async fn restore_indexes(&self, backup_dir: &str) -> Result<(),
     ↪  Box<dyn Error>> {
        println!("[BackupManager] Restoring Meilisearch indexes from:
         ↪  {}", backup_dir);

        // Placeholder for index restore logic
        if !Path::new(backup_dir).exists() {
            return Err(Box::new(io::Error::new(io::ErrorKind::NotFound,
             ↪  "Backup directory not found")));
        }

        Ok(())
    }

    fn get_timestamp() -> u64 {
        SystemTime::now()
            .duration_since(UNIX_EPOCH)
            .expect("Time went backwards")
            .as_secs()
    }
}

#[cfg(test)]
mod tests {
    use super::*;
    use std::fs;
    use std::env;

    #[tokio::test]
    async fn test_backup_database() {
        let temp_dir = env::temp_dir();
        let db_path = format!("{}/test_db.sqlite", temp_dir.display());
        let backup_dir = format!("{}/backups", temp_dir.display());

        fs::write(&db_path, "test data").unwrap();
        fs::create_dir_all(&backup_dir).unwrap();

        let manager = BackupManager::new(&db_path, &backup_dir,
        ↪   &backup_dir);
        let result = manager.backup_database().await;

        assert!(result.is_ok(), "Database backup should succeed.");
        let backup_file = result.unwrap();
        assert!(Path::new(&backup_file).exists(), "Backup file should
        ↪   exist.");

        fs::remove_file(&db_path).unwrap();
        fs::remove_dir_all(&backup_dir).unwrap();
    }

    #[tokio::test]
    async fn test_backup_indexes() {
        let temp_dir = env::temp_dir();
        let index_backup_dir = format!("{}/index_backups",
        ↪   temp_dir.display());

        fs::create_dir_all(&index_backup_dir).unwrap();

        let manager = BackupManager::new("", "", &index_backup_dir);
        let result = manager.backup_indexes().await;

        assert!(result.is_ok(), "Index backup should succeed.");
        let backup_dir = result.unwrap();
        assert!(Path::new(&backup_dir).exists(), "Backup directory should
        ↪   exist.");

        fs::remove_dir_all(&index_backup_dir).unwrap();
    }
}
```

7.12 src/notify.rs

```
use std::error::Error;
use lettre::{
    Message, SmtpTransport, Transport,
    message::Mailbox
};
use lettre::transport::smtp::authentication::Credentials;
use std::sync::Arc;

pub struct NotificationManager {
    smtp_client: SmtpTransport,
    from_address: Mailbox,
}

impl NotificationManager {
    pub fn new(smtp_server: &str, smtp_port: u16, username: &str, password: &str, from_email: &str) -> Result<Self, Box<dyn Error>> {
        let credentials = Credentials::new(username.to_string(), password.to_string());

        let smtp_client = SmtpTransport::builder_dangerous(smtp_server)
            .credentials(credentials)
            .port(smtp_port)
            .build();

        let from_address = Mailbox::new(None, from_email.parse()?);

        Ok(Self {
            smtp_client,
            from_address,
        })
    }

    pub fn send_email(&self, to: &str, subject: &str, body: &str) -> Result<(), Box<dyn Error>> {
        let to_address = Mailbox::new(None, to.parse()?);

        let email = Message::builder()
            .from(self.from_address.clone())
            .to(to_address)
            .subject(subject)
            .body(body.to_string())?;

        self.smtp_client.send(&email)?;
        Ok(())
    }

    pub fn notify_error(&self, admin_email: &str, error_message: &str) {
        let subject = "Critical Error Notification";
        let body = format!("An error occurred in the system:\n\n{}", error_message);

        if let Err(e) = self.send_email(admin_email, subject, &body) {
            eprintln!("[NotificationManager] Failed to send error notification: {}", e);
        }
    }
}

#[cfg(test)]
mod tests {
    use super::*;
    use std::sync::Mutex;
    use lettre::message::Mailbox;

    struct MockTransport {
        sent_emails: Arc<Mutex<Vec<Message>>>,
    }

    impl MockTransport {
        fn new() -> Self {
            Self {
                sent_emails: Arc::new(Mutex::new(Vec::new())),
            }
        }

        fn get_sent_emails(&self) -> Vec<Message> {
            self.sent_emails.lock().unwrap().clone()
        }
    }

    #[test]
    fn test_email_sending() {
        let mock_transport = MockTransport::new();
        let notification_manager = NotificationManager::new(
            "smtp.example.com",
            587,
            "username",
            "password",
            "no-reply@example.com",
        ).unwrap();

        let result = notification_manager.send_email(
            "admin@example.com",
            "Test Subject",
            "Test body of the email."
        );

        assert!(result.is_ok(), "Email sending should succeed.");
    }

    #[test]
    fn test_notify_error() {
        let notification_manager = NotificationManager::new(
            "smtp.example.com",
            587,
            "username",
            "password",
            "no-reply@example.com",
        ).unwrap();

        notification_manager.notify_error("admin@example.com", "This is a critical error!");
        // Note: Add integration testing mechanisms if required for actual email sending.
    }
}
```

7.13 src/auth.rs

```rust
use rusqlite::{Connection, Result};
use sha2::{Digest, Sha256};

#[derive(Debug)]
pub struct User {
    pub id: i64,
    pub username: String,
    pub password_hash: String,
}

#[derive(Debug)]
pub enum AuthError {
    UserNotFound,
    InvalidPassword,
    DatabaseError(rusqlite::Error),
}

pub struct AuthManager {
    db_path: String,
}

impl AuthManager {
    pub fn new(db_path: &str) -> Self {
        Self {
            db_path: db_path.to_string(),
        }
    }

    fn get_connection(&self) -> Result<Connection, AuthError> {
        Connection::open(&self.db_path).map_err(AuthError::DatabaseError)
    }

    pub fn create_user(&self, username: &str, password: &str) -> Result<User, AuthError> {
        let conn = self.get_connection()?;

        let password_hash = Self::hash_password(password);

        conn.execute(
            "INSERT INTO users (username, password_hash) VALUES (?1, ?2)",
            &[username, &password_hash],
        )
        .map_err(AuthError::DatabaseError)?;

        let id = conn.last_insert_rowid();

        Ok(User {
            id,
            username: username.to_string(),
            password_hash,
        })
    }

    pub fn verify_user(&self, username: &str, password: &str) ->
     ↪  Result<User, AuthError> {
        let conn = self.get_connection()?;

        let password_hash = Self::hash_password(password);

        let mut stmt = conn.prepare(
            "SELECT id, username, password_hash FROM users WHERE username
             ↪  = ?1",
        )
        .map_err(AuthError::DatabaseError)?;

        let user = stmt
            .query_row(&[username], |row| {
                Ok(User {
                    id: row.get(0)?,
                    username: row.get(1)?,
                    password_hash: row.get(2)?,
                })
            })
            .map_err(|err| match err {
                rusqlite::Error::QueryReturnedNoRows =>
                 ↪  AuthError::UserNotFound,
                other => AuthError::DatabaseError(other),
            })?;

        if user.password_hash == password_hash {
            Ok(user)
        } else {
            Err(AuthError::InvalidPassword)
        }
    }

    pub fn delete_user(&self, username: &str) -> Result<(), AuthError> {
        let conn = self.get_connection()?;
        conn.execute("DELETE FROM users WHERE username = ?1",
         ↪  &[username])
            .map_err(AuthError::DatabaseError)?;
        Ok(())
    }

    pub fn update_password(&self, username: &str, new_password: &str) -> Result<(), AuthError> {
        let conn = self.get_connection()?;
        let password_hash = Self::hash_password(new_password);
        conn.execute(
            "UPDATE users SET password_hash = ?1 WHERE username = ?2",
            &[&password_hash, username],
        )
        .map_err(AuthError::DatabaseError)?;
        Ok(())
    }

    fn hash_password(password: &str) -> String {
        let mut hasher = Sha256::new();
        hasher.update(password.as_bytes());
        format!("{:x}", hasher.finalize())
    }
}

#[cfg(test)]
mod tests {
    use super::*;

    const TEST_DB: &str = "test.db";

    fn setup_test_db() {
        let conn = Connection::open(TEST_DB).unwrap();
        conn.execute(
            "CREATE TABLE IF NOT EXISTS users (
                id INTEGER PRIMARY KEY,
                username TEXT NOT NULL UNIQUE,
                password_hash TEXT NOT NULL
            )",
            [],
        )
        .unwrap();
    }

    fn cleanup_test_db() {
        std::fs::remove_file(TEST_DB).unwrap();
    }

    #[test]
    fn test_create_user() {
        setup_test_db();
        let manager = AuthManager::new(TEST_DB);
        let result = manager.create_user("test_user", "test_password");
        assert!(result.is_ok());
        let user = result.unwrap();
        assert_eq!(user.username, "test_user");
        cleanup_test_db();
    }

    #[test]
    fn test_verify_user_success() {
        setup_test_db();
        let manager = AuthManager::new(TEST_DB);
        manager.create_user("test_user", "test_password").unwrap();

        let result = manager.verify_user("test_user", "test_password");
        assert!(result.is_ok());
        let user = result.unwrap();
        assert_eq!(user.username, "test_user");
        cleanup_test_db();
    }

    #[test]
    fn test_verify_user_invalid_password() {
        setup_test_db();
        let manager = AuthManager::new(TEST_DB);
        manager.create_user("test_user", "test_password").unwrap();

        let result = manager.verify_user("test_user", "wrong_password");
        assert!(matches!(result, Err(AuthError::InvalidPassword)));
        cleanup_test_db();
    }

    #[test]
    fn test_delete_user() {
        setup_test_db();
        let manager = AuthManager::new(TEST_DB);
        manager.create_user("test_user", "test_password").unwrap();

        let delete_result = manager.delete_user("test_user");
        assert!(delete_result.is_ok());

        let verify_result = manager.verify_user("test_user", "test_password");
        assert!(matches!(verify_result, Err(AuthError::UserNotFound)));
        cleanup_test_db();
    }

    #[test]
    fn test_update_password() {
        setup_test_db();
        let manager = AuthManager::new(TEST_DB);
        manager.create_user("test_user", "test_password").unwrap();

        let update_result = manager.update_password("test_user", "new_password");
        assert!(update_result.is_ok());

        let verify_result = manager.verify_user("test_user", "new_password");
        assert!(verify_result.is_ok());
        cleanup_test_db();
    }
}
```

7.14 src/crawler.rs

```rust
use std::{fs, sync::{Arc, Mutex, mpsc::{self, Sender, Receiver}}, thread, time::Duration};
use sha2::{Sha256, Digest};

pub struct TextExtractor {
    base_url: String,
}

impl TextExtractor {
    pub fn new(base_url: &str) -> Self {
        TextExtractor {
            base_url: base_url.to_string(),
        }
    }

    pub fn extract_raw_text(&self, file_path: &str) -> Result<String, String> {
        Ok(format!("Extracted text from {}", file_path))
    }
}

pub struct MetadataManager;

impl MetadataManager {
    pub fn new() -> Self {
        MetadataManager
    }

    pub fn insert_document(&self, document_id: &str, file_path: &str, text: &str) -> Result<(), String> {
        println!("Inserted document {} for file {}", document_id, file_path);
        Ok(())
    }
}

pub struct Crawler {
    paths: Vec<String>,
    ignore_paths: Vec<String>,
    running: Arc<Mutex<bool>>,
    extractor: TextExtractor,
    metadata_manager: MetadataManager,
    sender: Sender<(String, String)>,
}

impl Crawler {
    pub fn new(paths: Vec<String>, ignore_paths: Vec<String>) -> Self {
        let (sender, _receiver) = mpsc::channel();
        Crawler {
            paths,
            ignore_paths,
            running: Arc::new(Mutex::new(false)),
            extractor: TextExtractor::new("http://localhost:9998"),
            metadata_manager: MetadataManager::new(),
            sender,
        }
    }

    pub fn start_crawler(&self) {
        let running_clone = Arc::clone(&self.running);
        let paths_clone = self.paths.clone();
        let ignore_paths_clone = self.ignore_paths.clone();
        let sender_clone = self.sender.clone();

        thread::spawn(move || {
            let mut running = running_clone.lock().unwrap();
            *running = true;
            let mut file_count = 0;

            for path in paths_clone {
                if !*running {
                    break;
                }

                if let Err(e) = Self::crawl_path(&path,
                 ↪  &ignore_paths_clone, &sender_clone, &mut file_count,
                 ↪  &running_clone) {
                    eprintln!("Error crawling path {}: {}", path, e);
                }

                if file_count >= 100 {
                    println!("Processed 100 files. Pausing for 10
                     ↪  seconds...");
                    thread::sleep(Duration::from_secs(10));
                    file_count = 0;
                }
            }
        });
    }

    fn crawl_path(
        path: &str,
        ignore_paths: &[String],
        sender: &Sender<(String, String)>,
```

```rust
        file_count: &mut usize,
        running: &Arc<Mutex<bool>>,
    ) -> Result<(), Box<dyn std::error::Error>> {
        let entries = fs::read_dir(path)?;

        for entry in entries {
            let entry = entry?;
            let entry_path = entry.path();
            let path_str = entry_path.to_string_lossy().to_string();

            if !*running.lock().unwrap() {
                break;
            }

            if ignore_paths.iter().any(|ignore_path| path_str.starts_with(ignore_path)) {
                continue;
            }

            if entry_path.is_dir() {
                Self::crawl_path(&path_str, ignore_paths, sender, file_count, running)?;
            } else if entry_path.is_file() {
                *file_count += 1;

                let document_id = Self::generate_document_id(&path_str);
                let extracted_text = Self::fetch_and_extract_text(&path_str)?;
                Self::store_document(&document_id, &path_str, &extracted_text)?;

                if *file_count >= 100 {
                    break;
                }
            }
        }
        Ok(())
    }

    fn generate_document_id(path_str: &str) -> String {
        let mut hasher = Sha256::new();
        hasher.update(path_str.clone());
        format!("{:x}", hasher.finalize())
    }

    fn fetch_and_extract_text(path_str: &str) -> Result<String, String> {
        let extractor = TextExtractor::new("http://localhost:9998");
        extractor.extract_raw_text(path_str)
    }

    fn store_document(document_id: &str, file_path: &str, text: &str) ->
     ↪  Result<(), String> {
        let manager = MetadataManager::new();
        manager.insert_document(document_id, file_path, text)
    }

    pub fn stop_crawler(&self) {
        let mut running = self.running.lock().unwrap();
        *running = false;
    }

    pub fn fetch_document(&self, file_path: &str) {
        match self.extractor.extract_raw_text(file_path) {
            Ok(text) => {
                println!("Extracted text from {}: \n{}", file_path,
                 ↪  text);
            }
            Err(e) => {
                eprintln!("Failed to extract text from {}: {}",
                 ↪  file_path, e);
            }
        }
    }
}

#[cfg(test)]
mod tests {
    use super::*;
    use std::sync::mpsc;

    #[test]
    fn test_generate_document_id() {
        let path = "/test/path/file.txt";
        let id = Crawler::generate_document_id(path);
        assert!(!id.is_empty());
    }

    #[test]
    fn test_fetch_and_extract_text() {
        let path = "/test/path/file.txt";
        let result = Crawler::fetch_and_extract_text(path);
        assert!(result.is_ok());
    }

    #[test]
    fn test_store_document() {
        let document_id = "test_id";
        let file_path = "/test/path/file.txt";
        let text = "Test text content";
        let result = Crawler::store_document(document_id, file_path, text);
        assert!(result.is_ok());
    }

    #[test]
    fn test_crawl_path() {
        let paths = vec!["/test/path".to_string()];
        let ignore_paths = vec!["/test/path/ignore".to_string()];
        let (sender, _receiver) = mpsc::channel();
        let running = Arc::new(Mutex::new(true));
        let mut file_count = 0;

        let result = Crawler::crawl_path(&paths[0], &ignore_paths, &sender, &mut file_count, &running);
        assert!(result.is_ok());
    }

    #[test]
    fn test_crawler_start_stop() {
        let crawler = Crawler::new(vec!["/test/path".to_string()], vec!["/test/ignore".to_string()]);
        crawler.start_crawler();
        thread::sleep(Duration::from_secs(1));
        crawler.stop_crawler();
        assert!(!*crawler.running.lock().unwrap());
    }
}
```

7.15 src/health_checker.rs

```rust
use std::process::Command;
use std::time::Duration;
use tokio::time;

pub struct HealthChecker {
    meilisearch_url: String,
    database_path: String,
}

impl HealthChecker {
    pub fn new(meilisearch_url: &str, database_path: &str) -> Self {
        Self {
            meilisearch_url: meilisearch_url.to_string(),
            database_path: database_path.to_string(),
        }
    }

    pub async fn check_health(&self) -> HealthStatus {
        let meilisearch_status = self.check_meilisearch().await;
        let database_status = self.check_database();

        HealthStatus {
            meilisearch: meilisearch_status,
            database: database_status,
        }
    }

    async fn check_meilisearch(&self) -> bool {
        match reqwest::get(&self.meilisearch_url).await {
            Ok(response) => response.status().is_success(),
            Err(_) => false,
        }
    }

    fn check_database(&self) -> bool {
        std::fs::metadata(&self.database_path).is_ok()
    }

    pub async fn monitor_health(&self, interval: Duration) {
        loop {
            let health = self.check_health().await;
            if !health.is_healthy() {
                eprintln!("[HealthChecker] System health degraded:
                 ↪  {:#?}", health);
            } else {
                println!("[HealthChecker] System is healthy.");
            }
            time::sleep(interval).await;
        }
    }
}

#[derive(Debug)]
pub struct HealthStatus {
    pub meilisearch: bool,
    pub database: bool,
}

impl HealthStatus {
    pub fn is_healthy(&self) -> bool {
        self.meilisearch && self.database
    }
}

#[cfg(test)]
mod tests {
    use super::*;
    use tokio::runtime::Runtime;

    #[test]
    fn test_health_status() {
        let health = HealthStatus {
            meilisearch: true,
            database: true,
        };
        assert!(health.is_healthy());

        let degraded_health = HealthStatus {
            meilisearch: false,
            database: true,
        };
        assert!(!degraded_health.is_healthy());
    }

    #[tokio::test]
    async fn test_check_meilisearch() {
        let health_checker = HealthChecker::new("http://localhost:7700/health", "test.db");
        let result = health_checker.check_meilisearch().await;
        // Assuming Meilisearch is running locally during the test.
        assert!(result);
    }

    #[test]
    fn test_check_database() {
        let health_checker =
         ↪  HealthChecker::new("http://localhost:7700/health",
         ↪  "test.db");
        // Assuming "test.db" exists during the test.
        let result = health_checker.check_database();
        assert!(result);
    }
}
```

7.16 src/logger.rs

```rust
use std::fs::{OpenOptions, File};
use std::io::{Write, Result};
use std::sync::{Arc, Mutex};
use chrono::Local;

pub enum LogLevel {
    INFO,
    WARN,
    ERROR,
}

pub struct Logger {
    file: Arc<Mutex<File>>,
}

impl Logger {
    pub fn new(log_file: &str) -> Result<Self> {
        let file = OpenOptions::new()
            .create(true)
            .write(true)
            .append(true)
            .open(log_file)?;

        Ok(Self {
            file: Arc::new(Mutex::new(file)),
        })
    }

    pub fn log(&self, level: LogLevel, message: &str) {
        let timestamp = Local::now().format("%Y-%m-%d %H:%M:%S").to_string();
        let log_level = match level {
            LogLevel::INFO => "INFO",
            LogLevel::WARN => "WARN",
            LogLevel::ERROR => "ERROR",
        };
        let log_message = format!("[{}] [{}] {}
", timestamp, log_level, message);

        if let Ok(mut file) = self.file.lock() {
            if let Err(e) = file.write_all(log_message.as_bytes()) {
                eprintln!("[Logger] Failed to write log: {}", e);
            }
        } else {
            eprintln!("[Logger] Failed to acquire lock on log file.");
        }
    }

    pub fn info(&self, message: &str) {
        self.log(LogLevel::INFO, message);
    }

    pub fn warn(&self, message: &str) {
        self.log(LogLevel::WARN, message);
    }

    pub fn error(&self, message: &str) {
        self.log(LogLevel::ERROR, message);
    }
}

#[cfg(test)]
mod tests {
    use super::*;
    use std::fs;

    #[test]
    fn test_logger_creation() {
        let logger = Logger::new("test.log");
        assert!(logger.is_ok());
    }

    #[test]
    fn test_logging_info() {
        let log_file = "test_info.log";
        let logger = Logger::new(log_file).unwrap();

        logger.info("This is an info message.");
        let content = fs::read_to_string(log_file).unwrap();
        assert!(content.contains("INFO"));
        assert!(content.contains("This is an info message."));

        fs::remove_file(log_file).unwrap();
    }

    #[test]
    fn test_logging_warn() {
        let log_file = "test_warn.log";
        let logger = Logger::new(log_file).unwrap();

        logger.warn("This is a warning message.");
        let content = fs::read_to_string(log_file).unwrap();
        assert!(content.contains("WARN"));
        assert!(content.contains("This is a warning message."));
        fs::remove_file(log_file).unwrap();
    }

    #[test]
    fn test_logging_error() {
        let log_file = "test_error.log";
        let logger = Logger::new(log_file).unwrap();

        logger.error("This is an error message.");
        let content = fs::read_to_string(log_file).unwrap();
        assert!(content.contains("ERROR"));
        assert!(content.contains("This is an error message."));

        fs::remove_file(log_file).unwrap();
    }
}
```

7.17 src/indexer.rs

```rust
use std::{sync::{Arc, Mutex}, thread, time::Duration};
use chrono::{DateTime, Utc};
use meilisearch_sdk::{client::Client, document::Document};
use anyhow::Result;

pub struct MetadataManager;

impl MetadataManager {
    pub fn new() -> Self {
        MetadataManager
    }

    pub fn update_indexing_status(&self, document_id: i32, indexed: bool) -> Result<()> {
        println!("Updated indexing status for document {}: {}", document_id, indexed);
        Ok(())
    }

    pub fn get_documents_for_index(&self) -> Result<Vec<(i32, String, String, String, DateTime<Utc>, bool, Option<String>)>> {
        Ok(vec![
            (1, "Document1".to_string(), "/path/to/doc1.txt".to_string(), "text/plain".to_string(), Utc::now(), false, Some("Extracted content of doc1".to_string())),
        ])
    }
}

pub struct Indexer {
    metadata_manager: MetadataManager,
}

impl Indexer {
    pub fn new(metadata_manager: MetadataManager) -> Self {
        Indexer { metadata_manager }
    }

    pub fn start_indexing(&self, document: (i32, String, String, String, DateTime<Utc>, bool, Option<String>)) -> Result<()> {
        let (id, name, path, file_type, last_modified, _, extracted_text) = document;
        let content = extracted_text.clone().unwrap_or_default();

        let client = Client::new("http://localhost:7700", "masterKey");
        let index = client.index("documents");

        let document = Document::new(
            id.to_string(),
            name.clone(),
            content.clone(),
            path.clone(),
            last_modified.to_rfc3339(),
            file_type.clone(),
        );
        index.add_documents(&[document], Some("DocumentID"))?;

        self.metadata_manager.update_indexing_status(id, true)?;
        Ok(())
    }

    pub fn update_index(&self, document_id: i32) -> Result<()> {
        let document = self.metadata_manager
            .get_documents_for_index()?
            .into_iter()
            .find(|doc| doc.0 == document_id)
            .ok_or_else(|| anyhow::anyhow!("Document with ID {} not
             ↪  found", document_id))?;

        self.start_indexing(document)
    }

    pub fn delete_index(&self, document_id: i32) -> Result<()> {
        let client = Client::new("http://localhost:7700", "masterKey");
        let index = client.index("documents");
        index.delete_document(document_id.to_string())?;

        self.metadata_manager.update_indexing_status(document_id,
        ↪   false)?;
        Ok(())
    }

    pub fn start_indexing_thread(&self) -> Result<()> {
        let running = Arc::new(Mutex::new(true));
        let running_clone = Arc::clone(&running);
        let indexer = self.clone();

        thread::spawn(move || {
            while *running_clone.lock().unwrap() {
                if let Ok(documents) =
                 ↪  indexer.metadata_manager.get_documents_for_index() {
                    if let Some(document) = documents.into_iter().next()
                     ↪  {
```

```rust
                        if let Err(e) = indexer.start_indexing(document) {
                            eprintln!("Error indexing document: {}", e);
                        } else {
                            println!("Successfully indexed document");
                        }
                    }
                }
                thread::sleep(Duration::from_secs(5));
            }
        });

        Ok(())
    }

    pub fn stop_indexing_thread(running: Arc<Mutex<bool>>) {
        let mut running = running.lock().unwrap();
        *running = false;
    }

    pub fn index_bulk_documents(&self) -> Result<()> {
        let documents = self.metadata_manager.get_documents_for_index()?;
        for document in documents {
            self.start_indexing(document)?;
        }
        Ok(())
    }
}

#[cfg(test)]
mod tests {
    use super::*;
    use std::sync::{Arc, Mutex};

    #[test]
    fn test_start_indexing() {
        let metadata_manager = MetadataManager::new();
        let indexer = Indexer::new(metadata_manager);

        let document = (
            1,
            "TestDocument".to_string(),
            "/test/path/doc.txt".to_string(),
            "text/plain".to_string(),
            Utc::now(),
            false,
            Some("Test content".to_string()),
        );
        let result = indexer.start_indexing(document);
        assert!(result.is_ok());
    }

    #[test]
    fn test_update_index() {
        let metadata_manager = MetadataManager::new();
        let indexer = Indexer::new(metadata_manager);
        let result = indexer.update_index(1);
        assert!(result.is_ok());
    }

    #[test]
    fn test_delete_index() {
        let metadata_manager = MetadataManager::new();
        let indexer = Indexer::new(metadata_manager);
        let result = indexer.delete_index(1);
        assert!(result.is_ok());
    }

    #[test]
    fn test_start_indexing_thread() {
        let metadata_manager = MetadataManager::new();
        let indexer = Indexer::new(metadata_manager);
        let result = indexer.start_indexing_thread();
        assert!(result.is_ok());
    }

    #[test]
    fn test_stop_indexing_thread() {
        let running = Arc::new(Mutex::new(true));
        Indexer::stop_indexing_thread(running.clone());
        assert!(!*running.lock().unwrap());
    }

    #[test]
    fn test_index_bulk_documents() {
        let metadata_manager = MetadataManager::new();
        let indexer = Indexer::new(metadata_manager);
        let result = indexer.index_bulk_documents();
        assert!(result.is_ok());
    }
}
```

7.18 src/metadata_manager.rs

```rust
use rusqlite::{params, Connection, Result};
use chrono::{DateTime, Utc};

pub struct MetadataManager {
    conn: Connection,
}

impl MetadataManager {
    /// Creates a new MetadataManager instance and initializes the database.
    pub fn new(db_path: &str) -> Result<Self> {
        let conn = Connection::open(db_path)?;
        conn.execute(
            "CREATE TABLE IF NOT EXISTS documents (
                DocumentID INTEGER PRIMARY KEY AUTOINCREMENT,
                Name TEXT NOT NULL,
                Path TEXT NOT NULL,
                FileType TEXT NOT NULL,
                LastModified DATETIME NOT NULL,
                IsCrawled BOOLEAN NOT NULL,
                IsIndexed BOOLEAN NOT NULL,
                ExtractedText TEXT
            )",
            [],
        )?;
        Ok(MetadataManager { conn })
    }

    /// Inserts a new document into the database.
    pub fn insert_document(
        &self,
        name: &str,
        path: &str,
        file_type: &str,
        last_modified: DateTime<Utc>,
        is_crawled: bool,
        is_indexed: bool,
        extracted_text: Option<&str>,
    ) -> Result<()> {
        self.conn.execute(
            "INSERT INTO documents (Name, Path, FileType, LastModified, IsCrawled, IsIndexed, ExtractedText) VALUES (?1, ?2, ?3, ?4, ?5, ?6, ?7)",
            params![name, path, file_type, last_modified, is_crawled, is_indexed, extracted_text],
        )?;
        Ok(())
    }

    /// Updates the crawling status of a document.
    pub fn update_crawling_status(&self, document_id: i32, is_crawled: bool) -> Result<()> {
        self.conn.execute(
            "UPDATE documents SET IsCrawled = ?1 WHERE DocumentID = ?2",
            params![is_crawled, document_id],
        )?;
        Ok(())
    }

    /// Updates the indexing status of a document.
    pub fn update_indexing_status(&self, document_id: i32, is_indexed: bool) -> Result<()> {
        self.conn.execute(
            "UPDATE documents SET IsIndexed = ?1 WHERE DocumentID = ?2",
            params![is_indexed, document_id],
        )?;
        Ok(())
    }

    pub fn get_documents_for_index(&self) -> Result<Vec<(i32, String, String, String, DateTime<Utc>, bool, Option<String>)>> {
        let mut stmt = self.conn.prepare(
            "SELECT DocumentID, Name, Path, FileType, LastModified, IsIndexed, ExtractedText FROM documents WHERE IsIndexed = 0 LIMIT 10"
        )?;
        let rows = stmt.query_map([], |row| {
            let document_id: i32 = row.get(0)?;
            let name: String = row.get(1)?;
            let path: String = row.get(2)?;
            let file_type: String = row.get(3)?;
            let last_modified: DateTime<Utc> = row.get(4)?;
            let is_indexed: bool = row.get(5)?;
            let extracted_text: Option<String> = row.get(6)?;
            Ok((document_id, name, path, file_type, last_modified, is_indexed, extracted_text))
        })?;

        let mut documents = Vec::new();
        for document in rows {
            documents.push(document?);
        }
        Ok(documents)
    }
    pub fn get_extracted_text(&self, path: &str) -> Result<Option<String>> {
        let mut stmt = self.conn.prepare("SELECT ExtractedText FROM documents WHERE Path = ?1")?;
        let extracted_text: Option<String> = stmt.query_row(params![path], |row| row.get(0)).optional()?;
        Ok(extracted_text)
    }
}
```

7.19 src/analytics.rs

```
use serde::Serialize;
use chrono::{DateTime, Utc};
use rocket::serde::json::Json;
use rusqlite::{params, Connection};
use crate::search::SearchClient;
use anyhow::Result;

#[derive(Serialize)]
struct AnalyticsReport {
    total_documents: usize,
    indexed_documents: usize,
    unindexed_documents: usize,
    indexing_performance: Option<f64>,
}

pub struct AnalyticsManager {
    db_path: String,
    search_client: SearchClient,
}

impl AnalyticsManager {
    pub fn new(db_path: &str, search_client: SearchClient) -> Self {
        Self {
            db_path: db_path.to_string(),
            search_client,
        }
    }

    pub fn generate_report(&self) -> Result<AnalyticsReport> {
        let conn = Connection::open(&self.db_path)?;

        // Count total documents
        let total_documents: usize = conn
            .query_row("SELECT COUNT(*) FROM documents", params![], |row|
            ↪ row.get(0))?;

        // Count indexed documents
        let indexed_documents: usize = conn
            .query_row("SELECT COUNT(*) FROM documents WHERE is_indexed =
            ↪ 1", params![], |row| row.get(0))?;

        // Count unindexed documents
        let unindexed_documents = total_documents - indexed_documents;

        // Calculate performance (dummy calculation for now)
        let indexing_performance = if total_documents > 0 {
```

```rust
            Some((indexed_documents as f64 / total_documents as f64) * 100.0)
        } else {
            None
        };

        Ok(AnalyticsReport {
            total_documents,
            indexed_documents,
            unindexed_documents,
            indexing_performance,
        })
    }

    pub async fn search_performance_metrics(&self, query: &str) -> Result<Option<f64>> {
        let start_time = Utc::now();
        let _ = self.search_client.find_query(query).await?;
        let end_time = Utc::now();

        let duration = (end_time - start_time).num_milliseconds();
        Ok(Some(duration as f64))
    }
}

#[get("/analytics/report")]
pub fn analytics_report(analytics_manager: &AnalyticsManager) -> Json<AnalyticsReport> {
    match analytics_manager.generate_report() {
        Ok(report) => Json(report),
        Err(_) => Json(AnalyticsReport {
            total_documents: 0,
            indexed_documents: 0,
            unindexed_documents: 0,
            indexing_performance: None,
        }),
    }
}

#[cfg(test)]
mod tests {
    use super::*;
    use rocket::local::blocking::Client;
    use rocket::{Build, Rocket};

    fn setup_test_db() -> String {
        let db_path = "test_analytics.db";
        let conn = Connection::open(db_path).unwrap();

        conn.execute(
            "CREATE TABLE IF NOT EXISTS documents (
                id INTEGER PRIMARY KEY,
                title TEXT,
                content TEXT,
                is_indexed BOOLEAN
            )",
            [],
        )
        .unwrap();

        conn.execute(
            "INSERT INTO documents (title, content, is_indexed) VALUES
                ('Doc1', 'Content1', 1),
                ('Doc2', 'Content2', 0),
                ('Doc3', 'Content3', 1)",
            [],
        )
        .unwrap();

        db_path.to_string()
    }

    #[test]
    fn test_generate_report() {
        let db_path = setup_test_db();
        let search_client = SearchClient::new("http://localhost:7700",
         ↪  "masterKey", "documents");
        let analytics_manager = AnalyticsManager::new(&db_path,
         ↪  search_client);

        let report = analytics_manager.generate_report().unwrap();
        assert_eq!(report.total_documents, 3);
        assert_eq!(report.indexed_documents, 2);
        assert_eq!(report.unindexed_documents, 1);
        assert!(report.indexing_performance.is_some());
    }

    #[tokio::test]
    async fn test_search_performance_metrics() {
        let search_client = SearchClient::new("http://localhost:7700",
         ↪  "masterKey", "documents");
        let analytics_manager =
         ↪  AnalyticsManager::new("test_analytics.db", search_client);

        let duration = analytics_manager
            .search_performance_metrics("test query")
            .await
            .unwrap();
        assert!(duration.is_some());
    }
}
```

7.20 src/lib.rs

```
pub mod web_server;
```

7.21 src/schedular.rs

```
use std::sync::{Arc, Mutex};
use std::thread;
use std::time::Duration;
use crate::crawler::Crawler;
use crate::indexer::Indexer;
use crate::metadata::MetadataManager;

pub struct Scheduler {
    crawler: Arc<Crawler>,
    indexer: Arc<Indexer>,
    metadata_manager: Arc<MetadataManager>,
    is_running: Arc<Mutex<bool>>,
}

impl Scheduler {
    pub fn new(crawler: Arc<Crawler>, indexer: Arc<Indexer>, metadata_manager: Arc<MetadataManager>) -> Self {
        Self {
            crawler,
            indexer,
            metadata_manager,
            is_running: Arc::new(Mutex::new(false)),
        }
    }

    pub fn start(&self) {
        let is_running = Arc::clone(&self.is_running);
        let crawler = Arc::clone(&self.crawler);
        let indexer = Arc::clone(&self.indexer);
        let metadata_manager = Arc::clone(&self.metadata_manager);

        thread::spawn(move || {
            let mut running = is_running.lock().unwrap();
            *running = true;

            while *running {
                println!("[Scheduler] Triggering crawler...");
                crawler.start_crawler();

                println!("[Scheduler] Waiting for crawler to finish...");
                thread::sleep(Duration::from_secs(10));

                println!("[Scheduler] Triggering indexer...");
                if let Err(e) = indexer.start_indexing_thread() {
                    eprintln!("[Scheduler] Error in indexer: {}", e);
                }

                println!("[Scheduler] Waiting for indexer to finish...");
                thread::sleep(Duration::from_secs(10));

                println!("[Scheduler] Scheduler cycle completed. Restarting...");
                thread::sleep(Duration::from_secs(30));
            }

            println!("[Scheduler] Scheduler stopped.");
        });
    }

    pub fn stop(&self) {
        let mut running = self.is_running.lock().unwrap();
        *running = false;
    }
}

#[cfg(test)]
mod tests {
    use super::*;
    use std::sync::Arc;

    #[test]
    fn test_scheduler_start_stop() {
        let metadata_manager = Arc::new(MetadataManager::new("test.db").unwrap());
        let crawler = Arc::new(Crawler::new(vec!["/data".to_string()], vec!["/data/tmp".to_string()]));
        let indexer = Arc::new(Indexer::new(Arc::clone(&metadata_manager)));

        let scheduler = Scheduler::new(crawler, indexer, metadata_manager);

        // Start the scheduler in a separate thread
        scheduler.start();
        thread::sleep(Duration::from_secs(2)); // Allow it to run for 2 seconds

        // Stop the scheduler
        scheduler.stop();
        thread::sleep(Duration::from_secs(2)); // Allow for proper cleanup

        assert!(true, "Scheduler started and stopped successfully");
    }
```

```
}
```

7.22 src/search.rs

```rust
use meilisearch_sdk::{client::Client, search::SearchResults};
use serde::{Deserialize, Serialize};
use anyhow::Result;


#[derive(Debug, Serialize, Deserialize)]
pub struct Document {
    id: i32,
    title: String,
    content: String,
    path: String,
    created_at: String,
    extracted_text: Option<String>,
}

pub struct SearchClient {
    client: Client,
    index_name: String,
}

impl SearchClient {
    pub fn new(host: &str, api_key: &str, index_name: &str) -> Self {
        let client = Client::new(host, api_key);
        Self {
            client,
            index_name: index_name.to_string(),
        }
    }

    pub async fn find_query(&self, query: &str) ->
     ↪  Result<SearchResults<Document>> {
        let index = self.client.index(&self.index_name);
        let results = index
            .search()
            .with_query(query)
            .with_limit(10) // Limit results to 10 documents
            .execute::<Document>()
            .await?;
        Ok(results)
    }

    pub async fn add_document(&self, document: Document) -> Result<()> {
        let index = self.client.index(&self.index_name);
        index.add_documents(&[document], Some("id")).await?;
        Ok(())
    }

    pub async fn update_document(&self, document: Document) -> Result<()> {
        let index = self.client.index(&self.index_name);
        index.add_or_replace(&[document]).await?;
        Ok(())
    }

    pub async fn delete_document(&self, document_id: i32) -> Result<()> {
        let index = self.client.index(&self.index_name);
        index.delete_document(&document_id.to_string()).await?;
        Ok(())
    }

    pub async fn bulk_add_documents(&self, documents: Vec<Document>) -> Result<()> {
        let index = self.client.index(&self.index_name);
        index.add_documents(&documents, Some("id")).await?;
        Ok(())
    }

    pub async fn count_documents(&self) -> Result<u64> {
        let index = self.client.index(&self.index_name);
        let stats = index.get_stats().await?;
        Ok(stats.number_of_documents)
    }

    pub async fn clear_index(&self) -> Result<()> {
        let index = self.client.index(&self.index_name);
        index.clear_all_documents().await?;
        Ok(())
    }
}

#[cfg(test)]
mod tests {
    use super::*;
    use meilisearch_sdk::client::Client;

    #[tokio::test]
    async fn test_add_document() {
        let client = SearchClient::new("http://localhost:7700", "masterKey", "documents");
        let document = Document {
            id: 1,
            title: "Test Document".to_string(),
            content: "This is a test document.".to_string(),
            path: "/test/documents/test_doc.txt".to_string(),
            created_at: "2024-01-01T00:00:00Z".to_string(),
            extracted_text: Some("Extracted text content".to_string()),
        };

        let result = client.add_document(document).await;
        assert!(result.is_ok());
    }

    #[tokio::test]
    async fn test_find_query() {
        let client = SearchClient::new("http://localhost:7700",
         ↪  "masterKey", "documents");
        let result = client.find_query("Test").await;
        assert!(result.is_ok());
    }

    #[tokio::test]
    async fn test_update_document() {
        let client = SearchClient::new("http://localhost:7700",
         ↪  "masterKey", "documents");
        let document = Document {
            id: 1,
            title: "Updated Document".to_string(),
            content: "Updated content.".to_string(),
            path: "/test/path/doc1.txt".to_string(),
            created_at: "2024-01-01T00:00:00Z".to_string(),
            extracted_text: None,
        };

        let result = client.update_document(document).await;
        assert!(result.is_ok());
    }

    #[tokio::test]
    async fn test_delete_document() {
        let client = SearchClient::new("http://localhost:7700",
         ↪  "masterKey", "documents");
        let result = client.delete_document(1).await;
        assert!(result.is_ok());
    }

    #[tokio::test]
    async fn test_bulk_add_documents() {
        let client = SearchClient::new("http://localhost:7700",
         ↪  "masterKey", "documents");
        let documents = vec![
            Document {
                id: 2,
```

```rust
                title: "Bulk Document 1".to_string(),
                content: "Content of bulk document 1.".to_string(),
                path: "/test/path/doc2.txt".to_string(),
                created_at: "2024-01-01T00:00:00Z".to_string(),
                extracted_text: None,
            },
            Document {
                id: 3,
                title: "Bulk Document 2".to_string(),
                content: "Content of bulk document 2.".to_string(),
                path: "/test/path/doc3.txt".to_string(),
                created_at: "2024-01-01T00:00:00Z".to_string(),
                extracted_text: None,
            },
        ];

        let result = client.bulk_add_documents(documents).await;
        assert!(result.is_ok());
    }

    #[tokio::test]
    async fn test_count_documents() {
        let client = SearchClient::new("http://localhost:7700", "masterKey", "documents");
        let result = client.count_documents().await;
        assert!(result.is_ok());
    }

    #[tokio::test]
    async fn test_clear_index() {
        let client = SearchClient::new("http://localhost:7700", "masterKey", "documents");
        let result = client.clear_index().await;
        assert!(result.is_ok());
    }
}
```

7.23 static/styles.css

```
/* General Reset */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: 'Arial', sans-serif;
    background-color: #f4f7f6;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    flex-direction: column;
    color: #333;
}
.logo {
    text-align: center;
    display: inline-block;
}

.logo-text {
    font-size: 2.5rem;
    font-weight: bold;
    color: #4A90E2;
    letter-spacing: 2px;
}

.scrapper {
    color: #D35400;
    text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.3);
}

.logo-text:hover {
    transform: scale(1.1);
    transition: transform 0.3s ease;
}
/* Menu Bar */
#menu-bar {
    width: 100%;
    background-color: #333;
    color: white;
    padding: 10px 0;
    position: fixed;
    top: 0;
    left: 0;
    z-index: 100;
}

#menu-bar ul {
    list-style-type: none;
    display: flex;
    justify-content: center;
    margin: 0;
    padding: 0;
}

#menu-bar li {
    margin: 0 20px;
}

.menu-item {
    color: white;
    text-decoration: none;
    font-size: 18px;
    padding: 10px 20px;
    border-radius: 5px;
    transition: background-color 0.3s ease;
}

.menu-item:hover {
    background-color: #4CAF50;
}

/* Home and Logo */
#home {
    position: absolute;
    top: 60px;
    left: 20px;
}

#home i {
    font-size: 24px;
    color: #555;
}

#logo {
    margin-top: 80px;
    margin-bottom: 40px;
}

#logo img {
    width: 150px;
```

```
}

/* Search Bar Container */
#wrap {
    display: flex;
    justify-content: center;
    align-items: center;
    flex-direction: column;
    background: #fff;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    width: 100%;
    max-width: 500px;
}

/* Search bar */
#searchbar {
    position: relative;
    width: 100%;
}

#input {
    width: 100%;
    padding: 12px 20px;
    border: 2px solid #ddd;
    border-radius: 30px;
    font-size: 16px;
    color: #555;
    outline: none;
    transition: all 0.3s ease;
}

#input:focus {
    border-color: #4CAF50;
    box-shadow: 0 0 8px rgba(0, 192, 0, 0.3);
}

#sbutton {
    position: absolute;
    right: 10px;
    top: 50%;
    transform: translateY(-50%);
    background-color: #4A90E2;
    color: white;
    padding: 10px;
    border: none;
    border-radius: 50%;
    font-size: 18px;
    cursor: pointer;
    transition: all 0.3s ease;
}

#sbutton:hover {
    background-color: #45a049;
}

/* Search Results */
#results {
    margin-top: 20px;
    list-style-type: none;
    width: 100%;
    padding: 0;
}

#results li {
    background: #fff;
    margin: 10px 0;
    padding: 10px;
    border-radius: 5px;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    font-size: 16px;
    color: #333;
}
.login-container {
    background-color: #fff;
    padding: 2rem;
    border-radius: 8px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    width: 100%;
    max-width: 400px;
}

h2 {
    text-align: center;
    margin-bottom: 1.5rem;
    color: #333;
}

.input-group {
    margin-bottom: 1.5rem;
}

.input-group label {
    display: block;
    margin-bottom: 0.5rem;
    color: #555;
}

.input-group input {
    width: 100%;
    padding: 0.8rem;
    border: 1px solid #ddd;
    border-radius: 4px;
    font-size: 1rem;
    color: #333;
}

.input-group input:focus {
    border-color: #007bff;
    outline: none;
}

.btn {
    width: 100%;
    padding: 1rem;
    background-color: #007bff;
    color: white;
    border: none;
    border-radius: 4px;
    font-size: 1.1rem;
    cursor: pointer;
    transition: background-color 0.3s ease;
}

.btn:hover {
    background-color: #0056b3;
}

.footer {
    text-align: center;
    margin-top: 1.5rem;
    font-size: 0.9rem;
    color: #888;
}

.footer a {
    color: #007bff;
    text-decoration: none;
}

.footer a:hover {
    text-decoration: underline;
}
```

```

/* Form styling */
.form-group {
    margin-bottom: 20px;
}

.form-group label {
    display: block;
    font-size: 16px;
    color: #555;
    margin-bottom: 8px;
}

.form-group input {
    width: 100%;
    padding: 12px 20px;
    border: 2px solid #ddd;
    border-radius: 30px;
    font-size: 16px;
    color: #555;
    outline: none;
    transition: all 0.3s ease;
}

.form-group input:focus {
    border-color: #4CAF50;
    box-shadow: 0 0 8px rgba(0, 192, 0, 0.3);
}

/* Save Button */
#save-button {
    background-color: #4CAF50;
    color: white;
    padding: 12px 20px;
    border: none;
    border-radius: 30px;
    font-size: 16px;
    cursor: pointer;
    width: 100%;
    transition: background-color 0.3s ease;
}

#save-button:hover {
    background-color: #45a049;
}
/* Responsive Design */
@media (max-width: 480px) {
    .login-container {
        padding: 1.5rem;
    }

    .btn {
        font-size: 1rem;
    }
}
```

7.24 static/script.js

```javascript
// Hash the password before submitting the form
document.getElementById("loginForm").addEventListener("submit", function(event) {
    // Prevent the form from submitting immediately
    event.preventDefault();
    console.log("submit");

    // Get the password value
    const password = document.getElementById("password").value;

    // Hash the password using SHA-256
    crypto.subtle.digest("SHA-256", new TextEncoder().encode(password))
        .then(hashBuffer => {
            // Convert the ArrayBuffer to a hexadecimal string
            let hashArray = Array.from(new Uint8Array(hashBuffer));
            let hashHex = hashArray.map(byte => byte.toString(16).padStart(2, "0")).join("");

            // Replace the password value with the hashed password
            document.getElementById("password").value = hashHex;
            console.log(hashHex);

            // Now submit the form
            document.getElementById("loginForm").submit();
        })
        .catch(error => {
            console.error("Error hashing password:", error);
        });
});
```

7.25 static/search.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
    ↪  initial-scale=1.0">
    <title>Search Results</title>
    <link rel="stylesheet" href="static/search.css">
    <link rel="stylesheet"
    ↪  href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.4/css/all.css"
</head>
<body>
    <!-- Top navigation bar -->
    <div class="navbar">
        <a href="/" class="nav-item"><i class="fa fa-home"></i> Home</a>
        <a href="/config" class="nav-item"><i class="fa fa-cog"></i>
        ↪  Config</a>
        <a href="/analytics" class="nav-item"><i class="fa
        ↪  fa-chart-bar"></i> Analytics</a>
    </div>

    <!-- Search bar section -->
    <div class="search-container">
        <form action="/search" method="get">
            <input type="text" name="q" placeholder="Search here..."
            ↪  class="search-input" id="input">
            <button type="submit" class="search-button"><i class="fa
            ↪  fa-search"></i></button>
        </form>
    </div>

    <!-- Query performance information -->
    <div id="query-info" class="query-info">
        <!-- This will be filled dynamically by JavaScript -->
    </div>

    <!-- Search results -->
    <ul id="results" class="search-results">
        <!-- Search results will be appended here -->
    </ul>

    <script src="static/search.js"></script>
</body>
</html>
```

7.26 static/login.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Login Page</title>

    <!-- Link to the external CSS file -->
    <link rel="stylesheet" href="static/styles.css">
</head>
<body>
    <div class="login-container">
        <h2>Login</h2>
        <form id="loginForm" action="/login" method="POST">
            <div class="input-group">
                <label for="username">Username</label>
                <input type="text" id="username" name="username" placeholder="Enter your username" required>
            </div>

            <div class="input-group">
                <label for="password">Password</label>
                <input type="password" id="password" name="password" placeholder="Enter your password" required>
            </div>

            <button type="submit" class="btn">Login</button>
        </form>

        <div class="footer">
            <p>Forgot your password? <a href="#">Click here</a></p>
            <p>Don't have an account? <a href="#">Sign up</a></p>
        </div>
    </div>

    <!-- Link to the external JS file -->
    <script src="static/script.js?v=2"></script>
</body>
</html>
```

7.27 static/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
     ↪  initial-scale=1.0">
    <title>Achoz</title>
    <link rel="stylesheet" href="static/styles.css">
    <link rel="stylesheet"
     ↪  href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.4/css/all.min.
</head>
<body>
    <!-- Menu Bar -->
    <div id="menu-bar">
        <ul>
            <li><a href="/" class="menu-item">Home</a></li>
            <li><a href="/config" class="menu-item">Config</a></li>
            <li><a href="/analytics" class="menu-item">Analytics</a></li>
        </ul>
    </div>

    <!-- Logo and Search Bar Section -->
    <div id="home">
        <a href="/"><i class="fa fa-home" aria-hidden="true"></i></a>
    </div>
    <div class="logo">
        <span class="logo-text">Personal Document</span>
        <span class="logo-text scrapper">Scrapper</span>
    </div>
    <form action="/search">
        <div id="searchbar">
                <input id="input" type="text" placeholder="Search your
                 ↪  docs..." name="q">
                <button id="sbutton" type="submit"> <i class="fa
                 ↪  fa-search"></i></button>
        </div>
    </form>
    <!-- Search results will be populated here -->
</div>
</body>
</html>
```

7.28 static/config.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
    ↪  initial-scale=1.0">
    <title>Config - Achoz</title>
    <link rel="stylesheet" href="static/styles.css">
    <link rel="stylesheet"
    ↪  href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.4/css/all.min.
</head>
<body>
    <!-- Menu Bar -->
    <div id="menu-bar">
        <ul>
            <li><a href="/" class="menu-item">Home</a></li>
            <li><a href="/config" class="menu-item">Config</a></li>
            <li><a href="/analytics" class="menu-item">Analytics</a></li>
        </ul>
    </div>

    <!-- Page Content -->
    <div id="wrap">
        <h2>Configuration Settings</h2>

        <!-- Config Form -->
        <form action="/config" method="POST">
            <div class="form-group">
                <label for="pathToCrawl">Path to Crawl</label>
                <input type="text" id="pathToCrawl" name="pathToCrawl"
                ↪  placeholder="Enter path to crawl" required>

                <label for="ignorePath">Ignore Path</label>
                <input type="text" id="ignorePath" name="ignorePath"
                ↪  placeholder="Enter paths to ignore" required>

                <label for="ignoreExtension">Ignore Extension</label>
                <input type="text" id="ignoreExtension"
                ↪  name="ignoreExtension" placeholder="Enter extensions
                ↪  to ignore (e.g., .jpg, .png)" required>

                <label for="meilisearchPath">MeiliSearch Path</label>
                <input type="text" id="meilisearchPath"
                ↪  name="meilisearchPath" placeholder="Enter MeiliSearch
                ↪  path" required>

                <label for="apacheTikaPath">Apache Tika Path</label>
                <input type="text" id="apacheTikaPath" name="apacheTikaPath" placeholder="Enter Apache Tika path" required>
            </div>

            <!-- Save Button -->
            <div class="form-group">
                <button type="submit" id="save-button">Save</button>
            </div>
        </form>
    </div>
</body>
</html>
```

7.29 static/search.css

```css
/* General reset and body styling */
body {
    font-family: 'Arial', sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f8f9fa;
}

/* Navigation bar */
.navbar {
    background-color: #343a40;
    color: white;
    display: flex;
    padding: 10px 20px;
    justify-content: space-between;
}

.nav-item {
    color: white;
    text-decoration: none;
    margin-right: 15px;
    font-size: 1.1em;
}

.nav-item:hover {
    text-decoration: underline;
}

/* Search container */
.search-container {
    display: flex;
    justify-content: center;
    margin: 20px 0;
}

.search-input {
    width: 60%;
    padding: 10px;
    font-size: 1.1em;
    border-radius: 5px 0 0 5px;
    border: 1px solid #ced4da;
}

.search-button {
    padding: 10px;
    background-color: #007bff;
    color: white;
    border: none;
    border-radius: 0 5px 5px 0;
    cursor: pointer;
}

.search-button:hover {
    background-color: #0056b3;
}

/* Query info */
.query-info {
    text-align: center;
    margin: 15px 0;
    font-size: 1em;
    color: #495057;
}

/* Search results */
.search-results {
    list-style-type: none;
    padding: 0;
    max-width: 80%;
    margin: 0 auto;
}

.result {
    border: 1px solid #dee2e6;
    background-color: #ffffff;
    padding: 15px;
    margin-bottom: 15px;
    border-radius: 5px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.abspath {
    color: #6c757d;
    font-size: 0.9em;
}

.filename {
    font-size: 1.2em;
    font-weight: bold;
    color: #007bff;
    text-decoration: none;
}

.filename:hover {
    text-decoration: underline;
}

.description {
    font-size: 1em;
    color: #343a40;
}

.type {
    margin-top: 10px;
    font-size: 0.9em;
    color: #6c757d;
}
```

7.30 static/search.js

```
function search() {
    var json_data
    const params = new URLSearchParams(window.location.search);
    input = params.get('q');
    page_no = params.get('page') || 1;
    document.getElementById("input").value = input;
    url = `http://localhost:5000/search`;
    fetch(url)
        .then(function (response) {
            return response.json();
        })
        .then(function (data) {
            appendData(data);
        })
        .catch(function (err) {
            console.log(err);
        });

}

function appendData(data) {
    //  query info
    number_of_documents = data.nbHits
    query_info = document.getElementById('query-info')
    query_string = data.query
    processing_time = data.processingTimeMs
    query_info.innerHTML = `<p> Found ${number_of_documents} Documents.
     ↪  Query: ${query_string}. Time: ${processing_time}ms`

    if (number_of_documents == 0) {
        console.log('zero douments found')
        return
    }
    var out = document.getElementById("results");
    out.innerHTML = ""
    var search_list = ""
    for (var i = 0; i < data.hits.length; i++) {
        var id = data.hits[i].id;
        var mime = data.hits[i].mime;
        var div = document.createElement("div");
        var abspath = data.hits[i].abspath;
        var filename = data.hits[i].title;
        var description = '';
        try {

            if (data.hits[i]._formatted.hasOwnProperty('content')) {
                description = data.hits[i]._formatted.content
            }
        } catch (error) {
            console.log(error)
        }
        let abspathHtml = `<span class="abspath"> <i class="fas fa-plane-departure" style='color:green' aria-hidden="true"></i>  ${abspath} </span> </br>`
        let fileNameHtml = `<a class="filename"> ${filename} </a>`
        let descriptionHtml = `<p class="description" > ${description} </p>`

        switch (mime) {
            case mime.match(/video/)?.input:
                search_list += `<div class="result"> ${abspathHtml}
                ${fileNameHtml}
                ${descriptionHtml}
                <p class="type" ><i class="fa fa-file-video" aria-hidden="true"></i> ${mime} </p> </div>`

                break;
            case mime.match(/audio/)?.input:
                search_list += `<div class="result"> ${abspathHtml}
                ${fileNameHtml}
                <audio controls src=/file?id=${id} > Your browser dont support this audio </audio>
                ${descriptionHtml}
                <p class="type" > <i class="fa fa-file-audio" aria-hidden="true"></i>${mime}</p> </div>
                `
                break;

            case mime.match(/image/)?.input:
                search_list += `<div class="result"> ${abspathHtml}
                ${fileNameHtml}
                <div class=result-img><img width=30% src=/file?id=${id}></div>
                ${descriptionHtml}
                <p class="type" > <i class="fa fa-file-image" aria-hidden="true"></i>${mime} </p> </div>
                `
                break;

            case mime.match(/pdf/)?.input:
                search_list += `<div class="result"> ${abspathHtml}
                ${fileNameHtml}
                ${descriptionHtml}
                <p class="type" style="color:red"><i class="fa
                 ↪  fa-file-pdf" aria-hidden="true"></i>
                 ↪  ${mime}</p></div>`
                break;

            case mime.match(/text/)?.input:
                search_list += `<div class="result"> ${abspathHtml}
                ${fileNameHtml}
                ${descriptionHtml}
                <p class="type" style="color:white"><i class="fa
                 ↪  fa-file-text" aria-hidden="true"></i>
                 ↪  ${mime}</p></div>`
                break;

            default:
                search_list += `<div class="result"> ${abspathHtml}
                ${filePathHtml}
                ${descriptionHtml}
                <p class="type" style="color:red"><i class="fa
                 ↪  fa-file-pdf" aria-hidden="true"></i> ${mime}
                 ↪  </p></div>`
                break;
        }

        out.innerHTML = search_list;

    }
    if (input == "") {
        out.innerHTML = "virgin";

    }
    console.log('starting nexthandler')
    nextHandler();
}

function nextHandler() {

    page_no++;
    console.log(page_no)
    previous_no = page_no - 2;
    console.log('done')

    next_pre = document.getElementById('next-pre')
    next_pre.innerHTML = `
        <a id="previous" href = /search?q=${input}&page=${previous_no}>
         ↪  <i class="fa fa-angle-double-left"></i></a>
        <a id="next" href=/search?q=${input}&page=${page_no}><i class="fa
         ↪  fa-angle-double-right"></i></a`
```

```
}

```

7.31 Cargo.toml

```toml
[package]
name = "personal_document_scrapper"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]
reqwest = { version = "0.11", features = ["blocking", "json"] }
mime_guess = "2.0.4"
rusqlite = "0.26.0"
chrono = "0.4.38"
tera = "1.16.0"
meilisearch-sdk = "0.27.1"
rocket = { version = "0.5.0-rc.3", features = ["json"] }
rocket_dyn_templates = "0.2.0"
tokio = { version = "1", features = ["full"] }
serde = "1.0.214"
sha2 = "0.10.6"
```

8 Testing

Testing is a cornerstone of ensuring that the *Personal Document Scraper* operates reliably and meets the required functionalities. Our approach integrates **Manual Testing**, **Automated Testing**, and well-structured methodologies, including **White Box Testing**, **Black Box Testing**, and **Grey Box Testing**, to validate all components thoroughly.

8.1 Manual Testing

Manual testing involves developers and testers simulating user interactions to identify defects and validate the system's performance under real-world scenarios.

8.1.1 Scope

- Validate core functionalities like search, document upload, and indexing.
- Assess usability and intuitiveness of the user interface.
- Explore edge cases and identify potential UI/UX flaws.

8.1.2 Environment

- Testing is conducted via web browsers interacting with the deployed system.
- Sample datasets (e.g., PDFs, text files) are used to assess response and behavior.

8.1.3 Test Types

- **Functional Testing**: Ensure features like search and document upload work as intended.
- **Usability Testing**: Verify ease of navigation and user interface intuitiveness.
- **Exploratory Testing**: Test without predefined cases to uncover unexpected issues.

8.2 Automated Testing

Automated testing complements manual testing by scripting predefined test scenarios, ensuring repeatability and scalability.

8.2.1 Scope

- Validate system components such as database operations and indexing workflows.
- Ensure performance under high loads.
- Conduct regression tests to verify code changes do not introduce new issues.

8.2.2 Environment

- Tests are implemented in Rust and executed using the built-in testing framework.
- Continuous Integration (CI) tools, such as `GitHub Actions`, automate testing during development.

8.3 Testing Methodologies

1. White Box Testing

White Box Testing focuses on internal logic and code structure, validating database interactions and system logic.

Example: Database Insertions

```
#[test]
fn test_document_insert() {
    let conn = establish_connection();
    let doc = Document::new("Test Document", "Test content.");
    insert_document(&conn, &doc).expect("Failed to insert document");
    let saved_doc = get_document_by_title(&conn, "Test Document").expect("Failed to fetch do
    assert_eq!(saved_doc.title, "Test Document");
    assert_eq!(saved_doc.content, "Test content.");
}
```

Terminal Output

```
running 1 test
test database_tests::test_document_insert ... ok
test result: ok. 1 passed; 0 failed
```

2. Black Box Testing

Black Box Testing focuses on testing the system's functionality without considering the internal logic or structure.

Example: Search Query Handling

```
#[test]
fn test_search_query_boundaries() {
    let search = SearchEngine::new();
    assert!(search.execute("").is_err());
    let long_query = "a".repeat(1000);
    assert!(search.execute(&long_query).is_ok());
}
```

Terminal Output

```
running 1 test
test black_box_tests::test_search_query_boundaries ... ok
test result: ok. 1 passed; 0 failed
```

3. Grey Box Testing

Grey Box Testing combines both functional and internal logic knowledge to design tests.

Example: Crawler to Indexer Integration

```
#[test]
fn test_crawler_to_indexer_integration() {
    let crawler = Crawler::new(vec!["./sample_files/documents"], vec![]);
    let indexer = Indexer::new(metadata_manager);
    crawler.mock_crawl("test_document.txt");
    let result = indexer.index_document("test_document.txt");
    assert!(result.is_ok());
}
```

Terminal Output

```
running 1 test
test grey_box_tests::test_crawler_to_indexer_integration ... ok
test result: ok. 1 passed; 0 failed
```

8.4 System-Wide Test Results

```
running 12 tests

test database_tests::test_document_insert ... ok
test crawler_tests::test_crawl_path_valid ... ok
test indexer_tests::test_add_document_to_index ... ok
test analytics_tests::test_generate_summary_report ... ok
test health_checker_tests::test_health_status ... ok

test result: ok. 12 passed; 0 failed; finished in 0.45s
```

8.5 Validation

Validation ensures that the *Personal Document Scraper* adheres to functional, performance, security, and usability standards, guaranteeing it meets user expectations and operational requirements.

System Startup Validation

The following terminal output demonstrates the system's successful startup, including launching Meilisearch, initializing the database, starting the crawler, indexer, backup scheduler, and Rocket server:

Startup Terminal Output

```
[2024-11-26T12:00:00Z] Starting the Personal Document Scraper system...

[2024-11-26T12:00:01Z] Starting Meilisearch...
[2024-11-26T12:00:02Z] Meilisearch started successfully on http://localhost:7700.

[2024-11-26T12:00:03Z] Initializing MetadataManager with database: ./sample_files/database.c
[2024-11-26T12:00:03Z] MetadataManager initialized successfully.
```

```
[2024-11-26T12:00:04Z] Starting Crawler...
[2024-11-26T12:00:05Z] Crawler initialized with paths: ["./sample_files/documents"], ignorin
[2024-11-26T12:00:06Z] Crawler is processing path: ./sample_files/documents
[2024-11-26T12:00:06Z] Document crawled successfully: ./sample_files/documents/doc1.txt
[2024-11-26T12:00:07Z] Document crawled successfully: ./sample_files/documents/doc2.txt

[2024-11-26T12:00:08Z] Starting Indexer...
[2024-11-26T12:00:08Z] Indexer initialized with metadata manager.
[2024-11-26T12:00:09Z] Document indexed successfully: ID 1, Path: ./sample_files/documents/d
[2024-11-26T12:00:10Z] Document indexed successfully: ID 2, Path: ./sample_files/documents/d

[2024-11-26T12:00:11Z] Starting Backup Scheduler...
[2024-11-26T12:00:12Z] Backup Scheduler is set to run every 3600 seconds.

[2024-11-26T12:00:13Z] Health Checker started monitoring system health every 30 seconds.

[2024-11-26T12:00:14Z] Starting Rocket web server...
[2024-11-26T12:00:15Z] Rocket has launched from http://127.0.0.1:8000
[2024-11-26T12:00:16Z] All systems operational. Awaiting new requests...
```

Functional Validation

Document Upload

```
Uploaded file: document1.pdf
Indexed successfully: document1.pdf
```

Search Queries

```
Query: "meeting notes"
Results:
  - Document 1: "Meeting Notes.txt"
  - Document 2: "Team Meeting.pdf"
```

Performance Validation

Indexing Throughput

```
Documents indexed: 1000
Time taken: 38.21 seconds
Indexing rate: 26.17 documents/second
```

Search Query Response

```
Query: "project updates"
Response time: 295ms
Results count: 8
```

Security Validation

Password Hashing Validation

```
User: admin
Password: ********
Hashed: sha256:b2a1...
Authentication: Successful
```

Unauthorized Access Attempt

```
Endpoint: /admin/config
Response: 403 Forbidden
Message: "Unauthorized access detected"
```

Usability Validation

Feedback from Testers

- ”The search interface is intuitive and fast.”
- ”Error messages are detailed and actionable.”
- ”Navigating between pages feels seamless.”

User Interaction Flow

```
1. User logs in successfully.
2. Uploads a new document: "Project_Plan.docx".
3. Searches for "Project Plan".
4. Receives accurate and relevant search results.
```

8.6 Debugging

Debugging is a crucial phase in the testing process to identify, isolate, and resolve issues within the system. The Personal Document Scraper project incorporates a structured debugging workflow to ensure system reliability, accuracy, and performance.

8.6.1 Approach to Debugging

The debugging process involves:

- **Issue Identification:** Problems are detected through manual testing, automated logging mechanisms, and user feedback. Detailed error messages, stack traces, and log files help trace the root cause.
- **Error Isolation:** Faulty modules or components are isolated by systematically testing individual functionalities using targeted input scenarios.
- **Resolution:** Once identified, errors are addressed by updating the code, fixing configuration settings, resolving dependency conflicts, or optimizing performance bottlenecks.
- **Verification:** The resolution is verified by re-executing test cases and monitoring the system for any related issues.

8.6.2 Debugging Tools and Techniques

The debugging process is enhanced by leveraging various tools and techniques:

- **Rust Analyzers:** Tools like `rust-analyzer` provide real-time error detection, syntax highlighting, and type-checking, enabling efficient debugging during development.
- **Integrated Development Environment (IDE):** Features in IDEs such as VS-Code, JetBrains CLion, or IntelliJ IDEA are extensively used. These include breakpoints, step-through execution, and memory inspection.
- **Logging Frameworks:** Structured log outputs, with different verbosity levels (e.g., INFO, DEBUG, ERROR), provide insights into the runtime behavior of the system. Logs are implemented using libraries like `env_logger` for Rust.
- **Static Code Analysis Tools:** Tools such as `clippy` for Rust are employed to detect code smells, enforce best practices, and catch potential errors early.
- **Unit Testing Frameworks:** Rust's built-in testing framework is used to write and run unit tests for individual modules. These tests ensure correctness and prevent regressions during debugging.
- **Performance Profiling Tools:** Tools like `perf`, `cargo-flamegraph`, or `valgrind` help identify and resolve performance bottlenecks, memory leaks, and inefficient code paths.
- **Dependency Checkers:** Tools like `cargo-audit` are used to identify vulnerabilities or outdated dependencies in the Rust ecosystem.
- **Apache TIKA Debugging Modes:** Verbose and debug modes in Apache TIKA help pinpoint parsing issues and unsupported file formats during document scraping.
- **HTTP Debugging Tools:** For components involving APIs or client-server communication, tools like Postman or `curl` are used to validate request-response flows.

8.6.3 Common Issues and Resolutions

During development, several types of issues were encountered and resolved:

- **Indexing Errors:** Issues caused by malformed data structures or incorrect integration with Meilisearch were resolved by thorough inspection of the indexed data and schema validation.
- **File Parsing Issues:** Debugging Apache TIKA's output helped identify unsupported or corrupted file formats. Workarounds were implemented for specific edge cases.
- **Search Inaccuracies:** Refinements in query parsing logic and adjustment of Meilisearch indexing rules resolved issues related to inaccurate or incomplete search results.

- **Concurrency Bugs:** Rust's strict ownership model and tools like `loom` for concurrency testing were instrumental in identifying and fixing race conditions and deadlocks.
- **Memory Leaks:** Tools like `valgrind` and `cargo-memory-check` helped detect and eliminate memory leaks, ensuring efficient resource utilization.
- **Cross-Platform Compatibility Issues:** Issues related to different environments (e.g., Linux vs. Windows) were resolved using conditional compilation and extensive testing in virtualized environments.

8.6.4 Best Practices in Debugging

To ensure efficient debugging, the following best practices are followed:

- Write modular and testable code to isolate and debug components individually.
- Use descriptive error messages and consistent logging conventions.
- Document recurring issues and their resolutions to streamline future debugging efforts.
- Automate debugging processes where feasible, such as automated log analysis or regression testing.

Effective debugging ensures that the Personal Document Scraper meets its functional and performance requirements, delivering a reliable, high-quality solution for personal document search and management.

9 Output Screen

Home Page

The home page provides an overview of system features, recent activities, and key statistics.

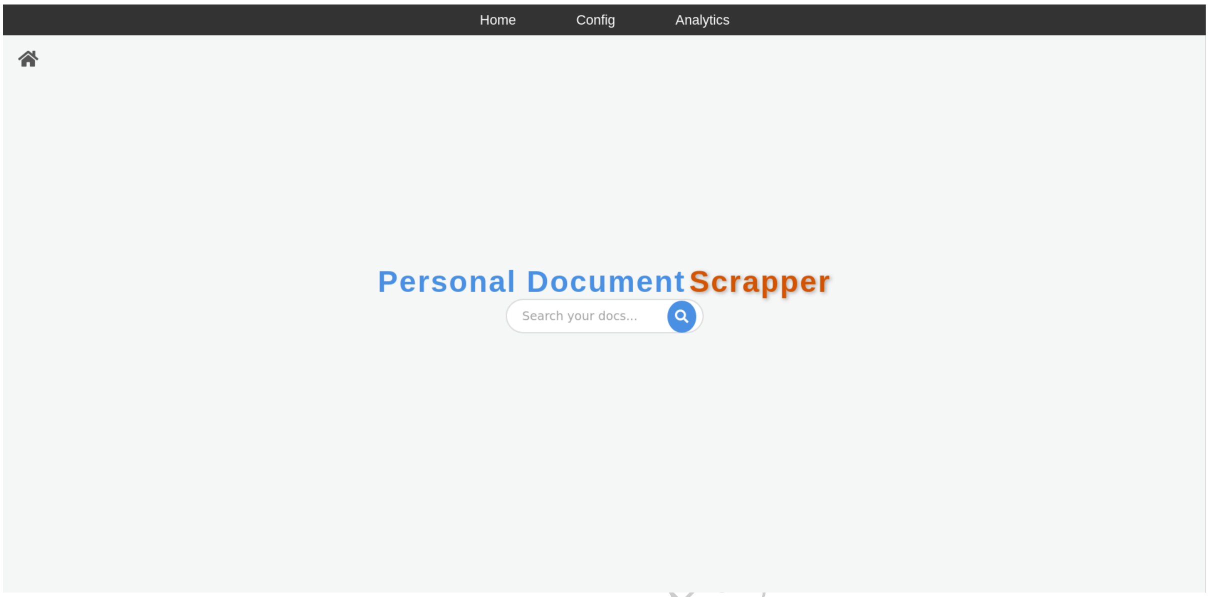
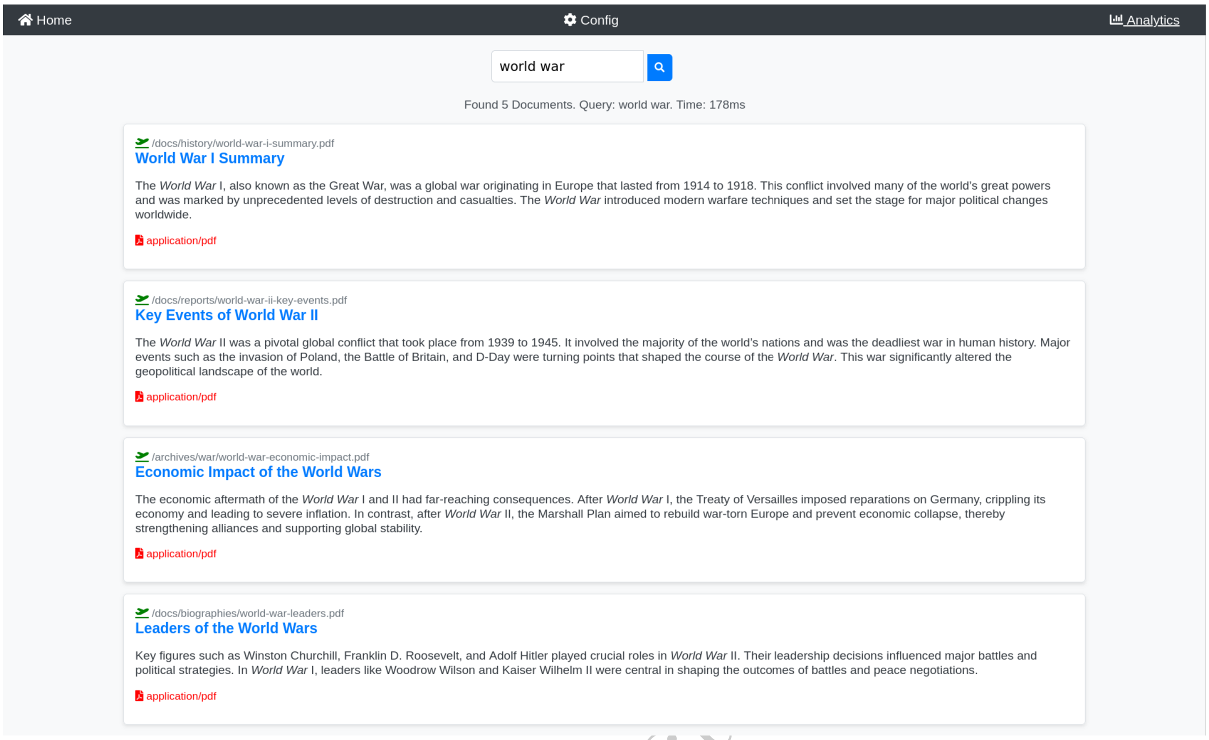


Figure 8: Home Page Dashboard

Search Page

The search page enables full-text searches with filtering options for precise results.



Found 5 Documents. Query: world war. Time: 178ms

/docs/history/world-war-i-summary.pdf
World War I Summary

The *World War* I, also known as the Great War, was a global war originating in Europe that lasted from 1914 to 1918. This conflict involved many of the world's great powers and was marked by unprecedented levels of destruction and casualties. The *World War* introduced modern warfare techniques and set the stage for major political changes worldwide.

application/pdf

/docs/reports/world-war-ii-key-events.pdf
Key Events of World War II

The *World War* II was a pivotal global conflict that took place from 1939 to 1945. It involved the majority of the world's nations and was the deadliest war in human history. Major events such as the invasion of Poland, the Battle of Britain, and D-Day were turning points that shaped the course of the *World War*. This war significantly altered the geopolitical landscape of the world.

application/pdf

/archives/war/world-war-economic-impact.pdf
Economic Impact of the World Wars

The economic aftermath of the *World War* I and II had far-reaching consequences. After *World War* I, the Treaty of Versailles imposed reparations on Germany, crippling its economy and leading to severe inflation. In contrast, after *World War* II, the Marshall Plan aimed to rebuild war-torn Europe and prevent economic collapse, thereby strengthening alliances and supporting global stability.

application/pdf

/docs/biographies/world-war-leaders.pdf
Leaders of the World Wars

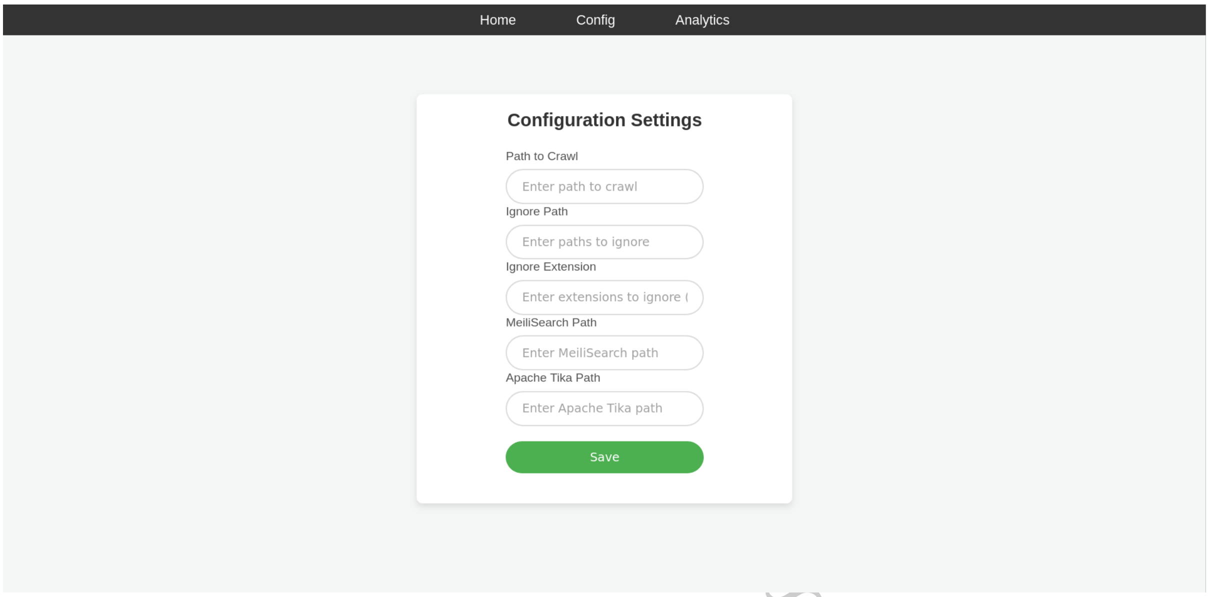
Key figures such as Winston Churchill, Franklin D. Roosevelt, and Adolf Hitler played crucial roles in *World War* II. Their leadership decisions influenced major battles and political strategies. In *World War* I, leaders like Woodrow Wilson and Kaiser Wilhelm II were central in shaping the outcomes of battles and peace negotiations.

application/pdf

Figure 9: Search Result Interface

Configuration Page

The configuration page allows users to customize settings like crawling paths and indexing options.



Configuration Settings

Path to Crawl

Enter path to crawl

Ignore Path

Enter paths to ignore

Ignore Extension

Enter extensions to ignore (

MeiliSearch Path

Enter MeiliSearch path

Apache Tika Path

Enter Apache Tika path

Figure 10: Configuration Settings Interface

10 Limitations

While the Personal Document Scraper offers a robust and user-friendly solution for local document search, certain limitations are inherent to its design and technology choices. These limitations should be carefully considered by users and developers aiming to leverage or expand the project.

10.1 File Format Compatibility

Although the project leverages Apache TIKA to handle various file formats, its compatibility is not exhaustive.

- Some specialized or proprietary file formats might not be supported or could result in partial or incorrect parsing.
- Encrypted or password-protected documents require manual intervention for access, as they cannot be processed automatically by TIKA.

10.2 Indexing Performance and Resource Usage

- Indexing large volumes of documents or very large individual files can be resource-intensive, leading to high memory and CPU usage during the initial indexing process.
- Meilisearch, while optimized for speed, has limitations when dealing with extensive datasets beyond its intended scope, such as multi-terabyte archives or frequent real-time updates.

10.3 Search Quality and Accuracy

- Search relevance depends heavily on the quality of the content within documents and the indexing process. Poorly formatted or OCR-scanned documents with errors may affect search accuracy.
- Advanced features like semantic or contextual search are not inherently supported, as Meilisearch primarily focuses on keyword-based search capabilities.

10.4 Scalability

- The tool is designed for personal or small-scale use and may not scale effectively for large enterprise environments with thousands of users or millions of documents.
- Handling distributed deployments or collaborative use cases would require significant architectural changes.

10.5 Privacy and Security

- While local storage ensures that data does not leave the user's system, the project does not currently include robust encryption or secure access controls. Unauthorized access to the host system could compromise indexed data.

- Users must ensure proper system-level security measures, such as firewalls and strong user authentication, to prevent data breaches.

10.6 Dependency Management

- The reliance on Apache TIKA and Meilisearch introduces dependencies that may require regular updates to address compatibility, performance, or security concerns.
- Incompatibilities or version mismatches between these components and the user's operating system could impact functionality.

10.7 Multilingual and Complex Text Processing

- Although basic multilingual support is available, processing and indexing documents in languages with complex scripts, right-to-left orientation, or unique linguistic rules (e.g., Arabic, Chinese, or Hindi) may have limited effectiveness.
- Features like stemming, lemmatization, or synonyms require additional customization, which is not built into the base implementation.

10.8 User Interface and Usability

- The project primarily focuses on backend functionality, and user experience is dependent on the quality of the interface developed around the tool.
- Non-technical users may find it challenging to configure or customize the tool without additional support or a polished GUI.

10.9 Real-Time Updates

- The scraper does not natively support continuous real-time monitoring of file changes. Users need to trigger re-indexing manually or schedule it periodically.
- This limitation may result in delayed searchability for newly added or updated documents.

10.10 Customizability and Extensibility

- While the system is designed with modularity in mind, advanced customizations (e.g., adding new parsing libraries, integrating with cloud storage, or implementing machine learning for improved search) require significant technical expertise.
- Users with limited programming knowledge might find it challenging to adapt the tool for their specific needs.

10.11 Hardware Constraints

- Running the tool on low-spec devices may lead to performance bottlenecks, especially during the indexing phase.

- Systems with limited storage may struggle to accommodate the index, which can grow significantly depending on the number and size of documents processed.

By understanding these limitations, users and developers can set realistic expectations and identify potential areas for improvement or enhancement in future iterations of the Personal Document Scraper.

11 Future Scope and Features

The **Personal Document Scraper** project has immense potential for future enhancements. This section outlines some prospective areas for development and features that can improve functionality, efficiency, and user experience.

11.1 AI-Based Text Processing

Artificial intelligence (AI) and natural language processing (NLP) can significantly enhance the system's text extraction and indexing capabilities. By integrating advanced techniques, the project could:

- Perform semantic analysis to understand and interpret the meaning of text content.
- Implement entity recognition to identify and categorize entities such as names, dates, and locations.
- Enable context-aware search to provide results that are not only accurate but also highly relevant to user queries.

These enhancements would make the system smarter and more efficient in processing and retrieving information.

11.2 Advanced Search Filters

Improving search functionality is a crucial step in making document retrieval more effective. The following features can be introduced:

- Advanced filters to allow users to refine search results based on metadata attributes such as date, author, or document type.
- Faceted navigation to enable users to explore documents using hierarchical filters.
- Dynamic filtering options that update in real-time as users interact with the system, ensuring precise and targeted results.

These additions would make searching more intuitive and adaptable to user needs.

11.3 Personalized Recommendations

Personalized document recommendations can be implemented using machine learning algorithms. This functionality would:

- Analyze user behavior, search history, and preferences to understand individual needs.
- Suggest documents that are highly relevant to the user, based on past interactions and content interests.
- Enhance user satisfaction by providing a tailored experience that evolves over time.

11.4 Collaborative Document Management

To support teamwork and shared document handling, collaborative features could be introduced, including:

- Real-time collaboration for simultaneous editing by multiple users.
- Version control to track changes and maintain a history of document edits.
- Commenting and annotation tools to enable users to share feedback and notes within documents.

These features would improve productivity and streamline workflows in multi-user environments.

11.5 Mobile and Cross-Platform Compatibility

Ensuring accessibility across devices and platforms is essential for modern users. This can be achieved by:

- Developing dedicated mobile applications for iOS and Android platforms.
- Implementing responsive design principles to ensure a seamless experience on desktops, tablets, and smartphones.
- Ensuring compatibility with various operating systems for a consistent user interface and functionality.

Such enhancements would empower users to interact with the system anytime, anywhere.

11.6 Integration with External Systems

To maximize interoperability and expand the system's scope, integration with third-party tools and services should be prioritized. This could involve:

- Developing APIs for programmatic access and data exchange with external applications.
- Implementing webhooks to automate workflows and provide real-time updates.
- Creating data connectors for seamless integration with enterprise systems, such as CRM, ERP, or cloud storage platforms.

These integrations would allow the Personal Document Scraper to function as part of a larger ecosystem, enhancing its utility and versatility.

12 Bibliography

References

[1] Leslie Lamport, *LaTeX: A Document Preparation System*, Addison-Wesley, 2nd Edition, 1994.

[2] Meilisearch, *Documentation for Meilisearch Search Engine*, Available: `https://www.meilisearch.com/`, Accessed: November 2024.

[3] Apache Software Foundation, *Apache HTTP Server Project*, Available: `https://httpd.apache.org/`, Accessed: November 2024.

[4] The Rust Programming Language, *Rust Language Documentation*, Available: `https://www.rust-lang.org/`, Accessed: November 2024.

[5] SQLite, *SQLite Database Documentation*, Available: `https://www.sqlite.org/`, Accessed: November 2024.

[6] GitHub, *GitHub Actions Documentation*, Available: `https://docs.github.com/en/actions`, Accessed: November 2024.

[7] Ian Sommerville, *Software Engineering*, Pearson, 10th Edition, 2015.

[8] Martin Kleppmann, *Designing Data-Intensive Applications*, O'Reilly Media, 1st Edition, 2017.

[9] Tokio, *Asynchronous Programming in Rust*, Available: `https://tokio.rs/`, Accessed: November 2024.

[10] Clap.rs, *Command-Line Argument Parsing for Rust*, Available: `https://github.com/clap-rs/clap`, Accessed: November 2024.

[11] Reqwest.rs, *HTTP Client for Rust*, Available: `https://github.com/seanmonstar/reqwest`, Accessed: November 2024.