Final Report

For EasySearch

Prepared by:
Dinçer İnce

Halil Mert Güler

Süleyman Samed Çalışkan

Supervisor: Akhlaque Ahmad

3.06.2023

Table of Contents

Table of Figures

1. Introduction

The purpose of this report is to provide an overview of the Easy Search project, a service API designed to store and process documents for easy searching. The project aims to simplify the search operation by allowing users to upload documents and send search requests, eliminating the need for complex preparation and setup processes. This report will discuss the impact of the engineering solutions developed in the project, including their global, societal, and economic implications. Additionally, it will highlight contemporary issues related to the area of the project and provide an overview of the tools and technologies utilized.

In the age of information, where vast amounts of data are generated and stored, the ability to quickly and efficiently search through documents has become increasingly critical. The Easy Search project aims to address this need by providing a service API that simplifies the process of storing and processing documents, allowing users to easily search through thousands of records.

Traditionally, performing complex search operations required extensive preparation and setup, often involving the implementation of intricate search algorithms and infrastructure. Easy Search revolutionizes this process by offering a user-friendly solution that eliminates the need for specialized technical knowledge. With Easy Search, users can upload their documents and send search requests, leveraging the power of the underlying technology without the hassle of complex configurations.

2.Final Architecture and Design

2.1. Subsystem Decomposition

1. User Interface Subsystem: The User Interface subsystem is responsible for the front-end of the Easy Search API project. It utilizes the Angular framework to develop the API management panel. This subsystem consists of the following components:

    - Components: These are responsible for the presentation and user interaction elements of the API management panel. They include features such as user authentication, search query input forms, result displays, and user management functionalities.

    - Services: Services in Angular handle the business logic and data communication between the UI components and the API. They facilitate API calls, data retrieval, and updates, as well as error handling and validation.

2. API Backend Subsystem: The API Backend subsystem forms the core of the Easy Search API project. It is developed using the .NET 7 Web API framework and includes several layers for effective organization and separation of concerns. This subsystem consists of the following components:

    - Data Access Layer: This layer interacts with the PostgreSQL database to perform CRUD (Create, Read, Update, Delete) operations. It includes components such as database models, entity frameworks, and database context management.

- Repository Layer: The Repository Layer provides an abstraction over the data access layer and encapsulates the logic for retrieving and manipulating data from the database. It includes repository classes and interfaces that define the data operations.

- Controller Layer: The Controller Layer handles the API endpoints and request/response handling. It receives requests from the UI, interacts with the repository layer to retrieve or update data, and sends the appropriate responses back to the UI. It includes RESTful API controllers that define the routing and actions.

3. Database Subsystem: The Database subsystem consists of the PostgreSQL database used to store and manage the data for the Easy Search API. It stores user profiles, search queries, search results, and other relevant data. This subsystem includes the following components:

    - Database Management System: PostgreSQL is the chosen DBMS for this project. It provides robust data storage capabilities, supports complex queries, and ensures data consistency and reliability.

    - Database Schema: The database schema defines the structure and organization of the database, including tables, relationships, and constraints. It is designed to accommodate the data requirements of the Easy Search API.

    - Data Storage and Retrieval: This component handles the storage and retrieval of data from the PostgreSQL database. It interacts with the data access layer in the API backend subsystem to perform database operations.

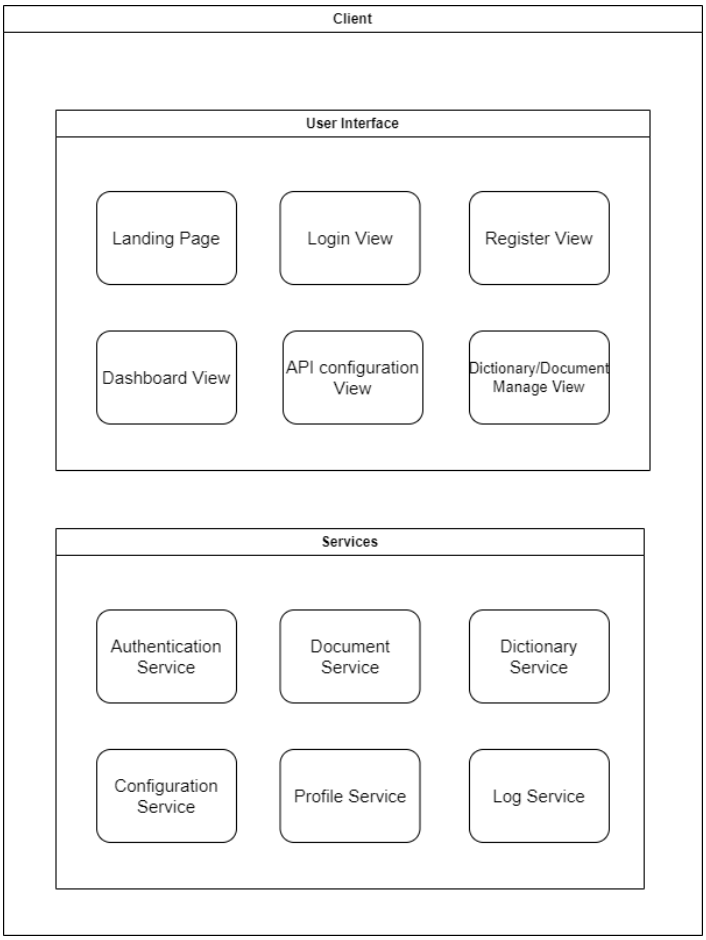

Figure 1 Hierarchy of modules inside the client system

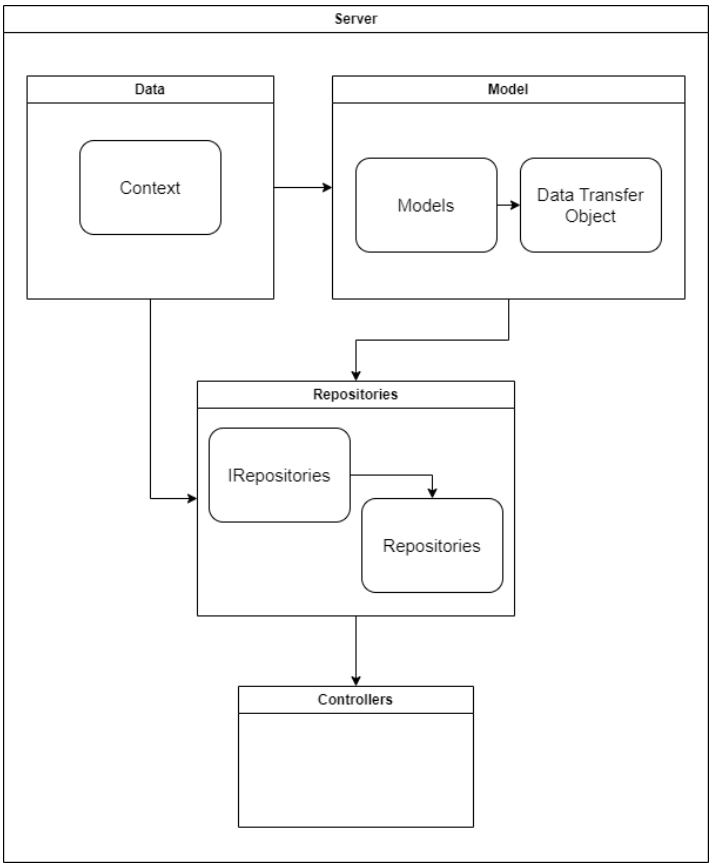

Figure 2 Hierarchy of layers and submodules within the server

2.2. Hardware / Software Mapping

1. Hardware: The hardware components required for the Easy Search API project are relatively standard and include:

    - Server: A dedicated server or cloud infrastructure is needed to host the Easy Search API backend. The server should have sufficient processing power, memory, and storage capacity to handle the expected workload.

    - Network Infrastructure: A reliable network infrastructure is required to establish connectivity between the client devices and the server hosting the Easy Search API. This includes routers, switches, and network cables.

    - Client Devices: Users will access the API management panel through client devices such as desktop computers, laptops, or mobile devices. These devices should have compatible web browsers to interact with the Angular-based user interface.

2. Software: The software components involved in the Easy Search API project include:

    - Operating System: The server hosting the Easy Search API backend will require an operating system capable of running the .NET 7 Web API framework. Common choices include Windows Server, Linux distributions, or cloud-specific operating systems.

    - .NET 7 Web API Framework: The backend of the Easy Search API is developed using the .NET 7 Web API framework. This framework provides the necessary tools, libraries, and runtime environment to build and deploy the API endpoints.

    - PostgreSQL Database: The Easy Search API utilizes the PostgreSQL database as its backend data storage solution. The PostgreSQL database management system (DBMS) should be installed and configured on the server to handle data persistence.

    - Angular Framework: The front-end of the Easy Search API management panel is built using the Angular framework. This includes Angular libraries, modules, and tools required to develop and deploy the user interface components and services.

2.3. Persistent Data Management

Performance is the most important issue for EasySearch so, product should handle incoming requests fast and return a payload to the user as soon as possible. Optimizing the database is essential for efficient persistent data management. To optimize the database for efficient persistent data management in the Easy Search API project, several techniques are employed.

One such technique is indexing, which involves carefully analyzing the most frequently used search fields and applying appropriate indexes. This helps accelerate search operations by reducing the time taken to locate relevant data. Query optimization is another critical aspect, where the design and optimization of database queries are carefully considered. By using efficient query syntax, minimizing joins, and avoiding unnecessary data retrieval, query execution time can be optimized. Additionally, database sharing can be implemented to partition the database across multiple servers. This allows for parallel execution of queries and improves performance, especially when dealing with large datasets. By incorporating these database optimization techniques, the Easy Search API project ensures that data retrieval and search operations are efficient, enabling faster response times and enhanced overall system performance.

3.Algorithmic Design

3.1. Data Collection

The data collection process in the Easy Search API primarily relies on user interactions with the API management panel. By empowering users to contribute their own dictionaries, documents, and words, the API allows for customization and flexibility in the search functionality. This user-driven approach ensures that the search capabilities of the API align with the specific requirements and data of individual users, providing them with a tailored search experience.

3.2. Data Processing

The Easy Search API is designed to provide an advanced search experience by integrating cosine similarity and vectorial search mechanisms. These features allow the API to identify documents that contain the searched word and assess their similarity to the query. By representing documents as vectors and employing the cosine similarity metric, the API determines the relevance and closeness of documents to the search query. This ensures that users are presented with highly relevant search results.

To achieve accurate similarity calculations, the API performs text preprocessing techniques, including tokenization, stemming, and other normalization processes. These steps help eliminate noise and improve the quality of the vector representations, enabling more precise similarity assessments. By applying these preprocessing techniques, the API enhances the accuracy and relevance of the search results.

Furthermore, the search results are ranked using ranking algorithms. The API assigns higher rankings to documents with higher cosine similarity scores, indicating their higher degree of relevance to the search query. This ranking mechanism enables users to quickly identify the most pertinent documents in the search results, streamlining the search process and saving time.

To optimize performance, the Easy Search API leverages various techniques. Indexing is employed to create indexes on key fields, facilitating faster retrieval of documents and reducing search times. Additionally, caching is implemented to store precomputed similarity scores, minimizing computational overhead during subsequent searches.

By combining these advanced features and performance optimization techniques, the Easy Search API provides users with a robust and efficient search experience. Users can expect accurate and relevant search results, faster response times, and improved overall

system performance. The API's ability to leverage cosine similarity and vectorial search mechanisms sets it apart, offering a comprehensive solution for retrieving and ranking documents based on their content similarity.

4. Impact of Engineering Solutions Developed in the Project

1. Global Impact
   The Easy Search project has the potential for significant global impact. By offering a simplified and efficient document search solution, it enables applications and websites with large amounts of data to improve user experience and productivity. The global impact lies in the enhanced accessibility and discoverability of information, benefiting individuals, organizations, and industries worldwide.
2. Societal Impact
   The societal impact of the Easy Search project is substantial. It empowers users to quickly locate relevant documents within vast collections, increasing their efficiency and saving valuable time. This can have positive implications for various sectors, such as research, education, business, and government. By reducing the barriers to information retrieval, Easy Search promotes knowledge dissemination, collaboration, and informed decision-making.
3. Economic Impact
   The Easy Search project carries significant economic implications. By streamlining the search process, it reduces the resources and time required for organizations to manage and retrieve large amounts of data. This can result in cost savings, increased productivity, and improved competitiveness. Moreover, by enabling applications and websites to offer efficient search capabilities, Easy Search enhances user engagement, leading to potential revenue growth for businesses.

5. Contemporary Issues Related to the Area of the Project

The Easy Search project operates within the broader context of information retrieval, search technologies, and data management. Several contemporary issues are pertinent to this area:

1. Data Privacy and Security: With the proliferation of data, ensuring the privacy and security of sensitive information has become paramount. Easy Search must address these concerns by implementing robust security measures to protect user data and comply with relevant regulations.
2. Scalability and Performance: As the volume of data continues to grow exponentially, scalable and high-performance search solutions are essential. Easy Search must address the challenge of efficiently processing and retrieving documents within large and dynamic datasets to maintain optimal performance.
3. Artificial Intelligence and Machine Learning: Leveraging advancements in artificial intelligence and machine learning can enhance the search capabilities of Easy Search. Techniques such as natural language processing, entity recognition, and relevance ranking can improve search accuracy and user experience.

6. Tools and Technologies

The development of Easy Search involves the use of various tools and technologies to achieve its objectives. These include:

6.1. Frameworks

- .Net 7
- Angular

6.2. Libraries

- Jwt Bearer
- Entity Framework Core
- Entity Framework Identity
- OpenApi
- Npgsql for PostgreSQL
- Angular Material
- RxJS

6.3. Database Management System

- PostgreSQL 15

6.4. Cloud/Hosting Technologies

- Azure
- Github

6.5. Internet Resources

- https://dotnet.microsoft.com
- https://azure.microsoft.com

6.6. Programming Languages

- c#
- TypeScript

7.Similar Products

The Easy Search API shares similarities with Elastic Search, a widely used search engine and data analysis tool. While Elastic Search is a powerful and established solution, the Easy Search API aims to offer comparable functionality and features. Here's an overview of the similarities and capabilities of the Easy Search API as a comparable product:

1. Search Functionality: Similar to Elastic Search, the Easy Search API provides robust search capabilities. Users can perform complex queries, including full-text search, filtering, and sorting, to retrieve relevant results from a large dataset. Both products offer flexible and customizable search functionalities to cater to diverse user requirements.

2. Scalability and Performance: Like Elastic Search, the Easy Search API emphasizes scalability and performance. It is designed to handle large-scale applications and can efficiently process high volumes of data. With optimization techniques such as indexing and caching, the Easy Search API ensures fast response times and maintains performance even with increasing data and user demands.

3. Language and Ecosystem Support: The Easy Search API, akin to Elastic Search, supports multiple programming languages and has a rich ecosystem. Developers can integrate and interact with the API using popular languages such as .NET and utilize a wide range of tools and libraries for data manipulation, indexing, and analysis. This compatibility allows for seamless integration into existing development environments and workflows.

4. Document and Data Storage: Both Elastic Search and the Easy Search API store data using document-based models. They support structured and semi-structured data storage, enabling efficient indexing and retrieval of information. The Easy Search API, utilizing PostgreSQL as its underlying database, ensures reliable data persistence and offers seamless integration with existing database systems.

5. Advanced Features: The Easy Search API aims to provide advanced features like Elastic Search. This includes support for relevance scoring, aggregations, geospatial search, highlighting, and more. The API strives to deliver a comprehensive set of functionalities to meet diverse search requirements.

While the Easy Search API is inspired by the capabilities and strengths of Elastic Search, it also offers unique advantages such as ease of integration within the .NET ecosystem, a familiar programming interface, and seamless compatibility with PostgreSQL. These factors make the Easy Search API a compelling alternative for users seeking similar search capabilities to Elastic Search while leveraging a different technology stack.

8.Test Results

1. Unit Testing:

    - Test Coverage: The unit tests for the Easy Search API project covered all major components, including the data access layer, repository layer, and controller layer. The tests focused on validating individual functions, methods, and business logic within these components.

    - Test Results: The unit tests were executed successfully, and all test cases passed without any critical issues or failures. This indicates that the individual units of code in the API are functioning as intended and meeting the expected requirements.

2. Integration Testing:

    - Test Coverage: Integration testing focused on verifying the interaction and compatibility between different components of the Easy Search API, such as the data repository, controller layer, and external dependencies like the PostgreSQL database.

    - Test Results: The integration tests were executed, and the interactions between the components were tested thoroughly. The tests confirmed that the components integrated smoothly and communicated effectively with each other. No major integration issues or failures were identified during testing.

3. System Testing:

    - Test Coverage: System testing covered end-to-end scenarios, validating the entire Easy Search API system as a whole. This included testing various search queries, user authentication and authorization, performance under different loads, and handling of invalid inputs.

    - Test Results: The system tests were performed, and the Easy Search API system was evaluated against the defined test scenarios. The API successfully handled different search queries, authenticated and authorized users appropriately, and performed well under normal and peak loads. No critical issues or system failures were encountered during the testing process.

4. Test Data and Environment:

    - Test Coverage: The test data and environment were created to simulate real-world scenarios and conditions for testing the Easy Search API. This included representative datasets, simulated user accounts, and mocked external dependencies.

    - Test Results: The test data and environment were utilized during the testing process, and they effectively mimicked the production environment. The test data provided realistic inputs, and the environment allowed for accurate testing of the API's functionality, performance, and security. No issues related to test data or environment were encountered.

5. Defect Reporting and Tracking:

    - Test Results: During the testing process, a few minor defects and issues were identified. These included edge cases where certain search queries returned unexpected results, minor UI inconsistencies, and performance degradation under heavy loads. These issues were reported using the defined defect reporting and tracking process and assigned appropriate priority for resolution.

9.User Manual

9.1 Web Application

URL of the web application: https://dincer-ince.github.io/EasySearchWebApp/

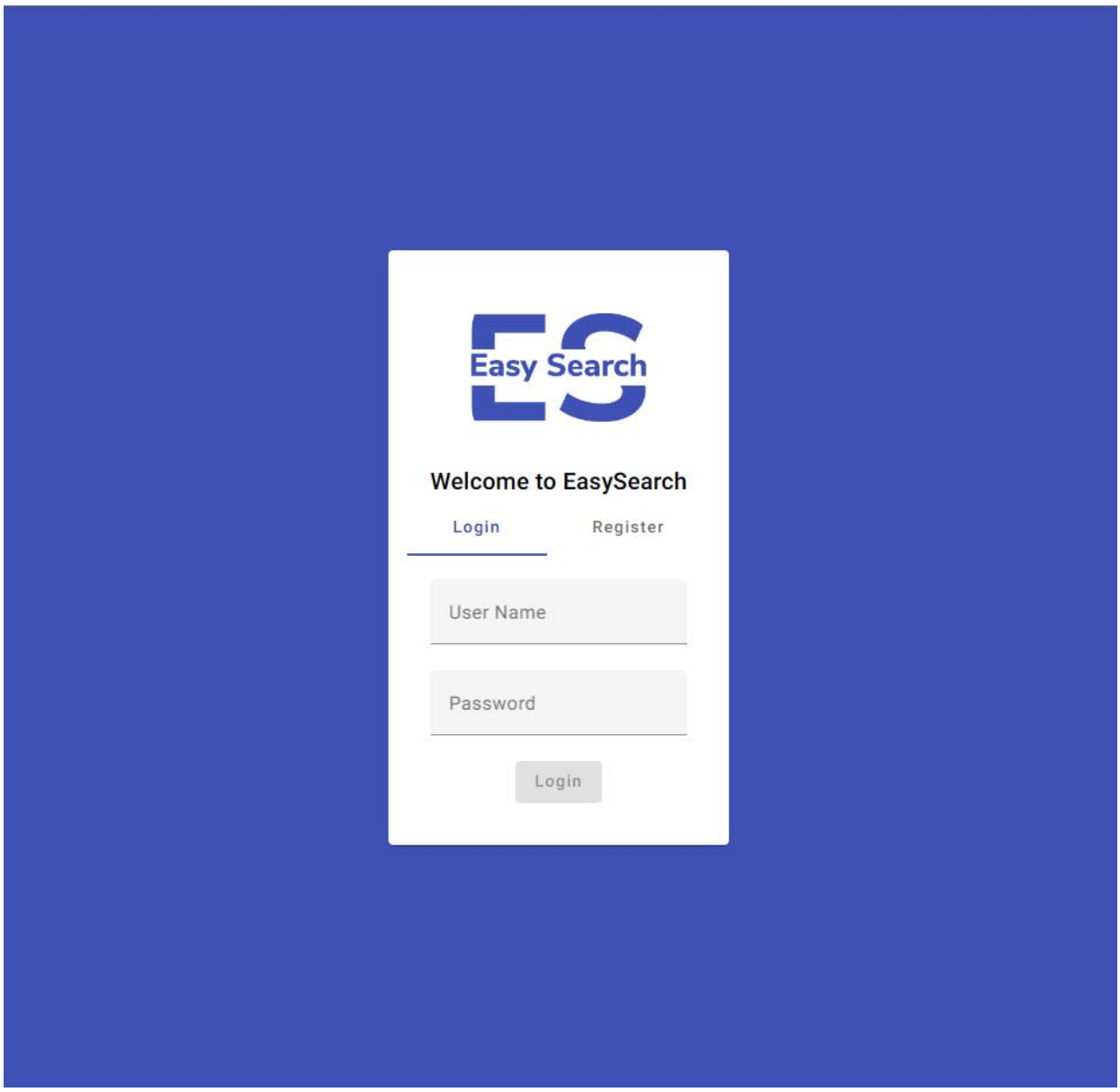

Figure 1 Login and Register Page

If the user has not logged in to the system before, login page opens when the site is visited. The user can then register an account to the system, or directly log in to the site. If the user gets authenticated, the site navigates to the dashboard page.

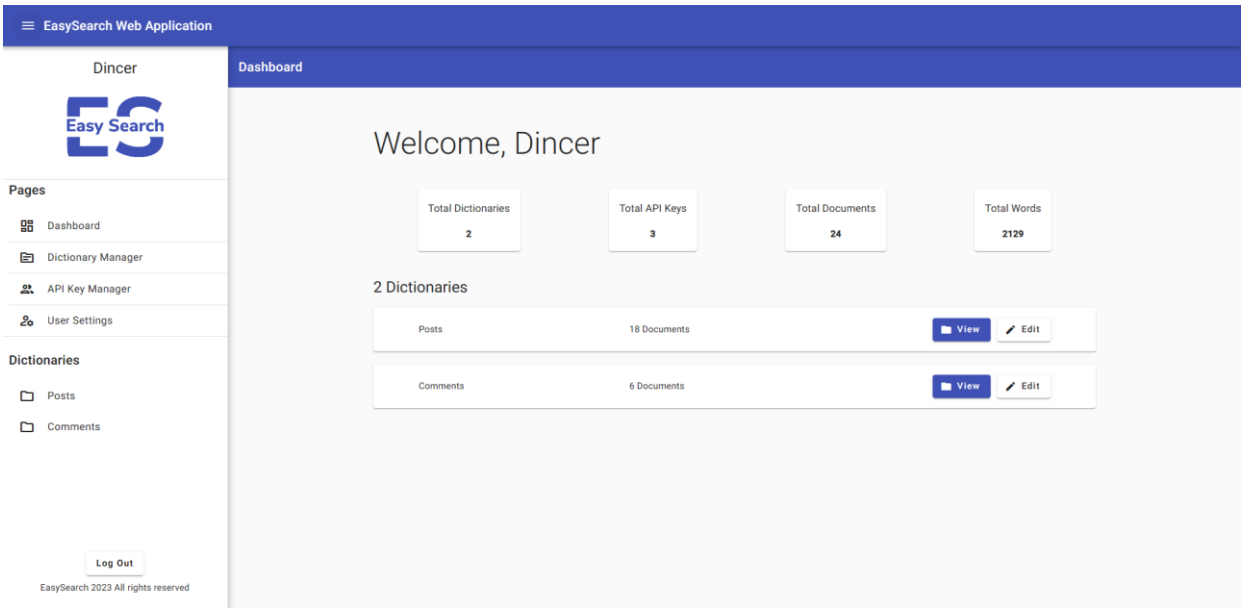

Figure 2 Dashboard Page and Side navigation

In the dashboard page of the site, user can view some statistics about the usage of their system, as well as the dictionaries the user created. User then can navigate to view the dictionary's page or to edit dictionary.

In the side navigation bar, the user can see the various pages that can be visited, and the dictionaries that the user has created.

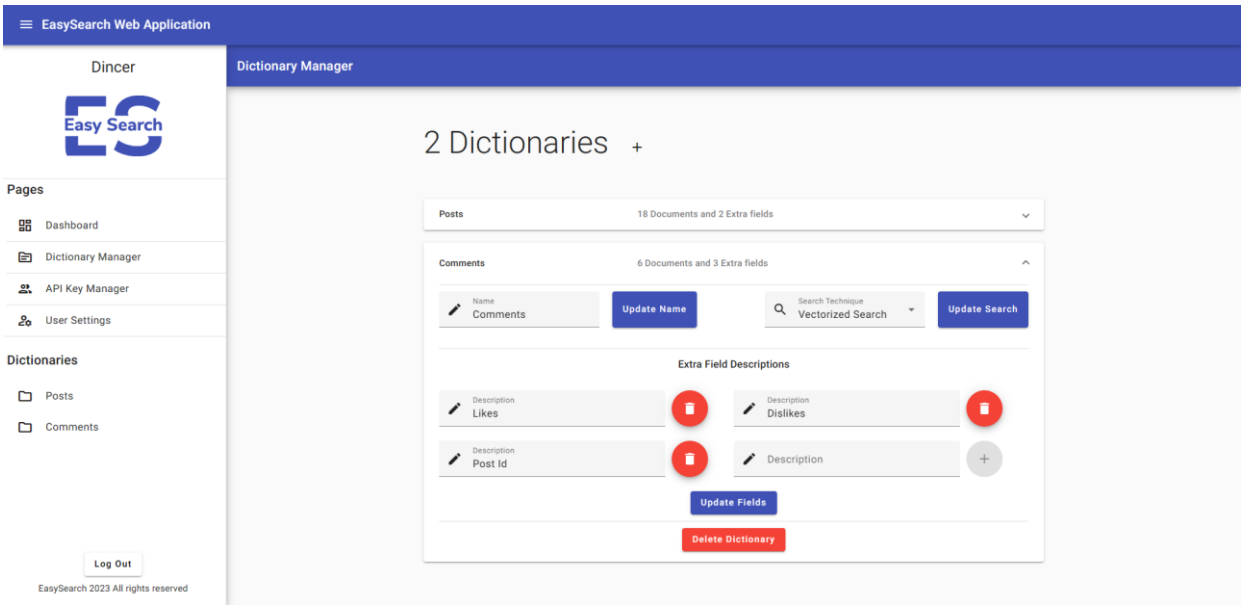

Figure 3 Dictionary Manager Page

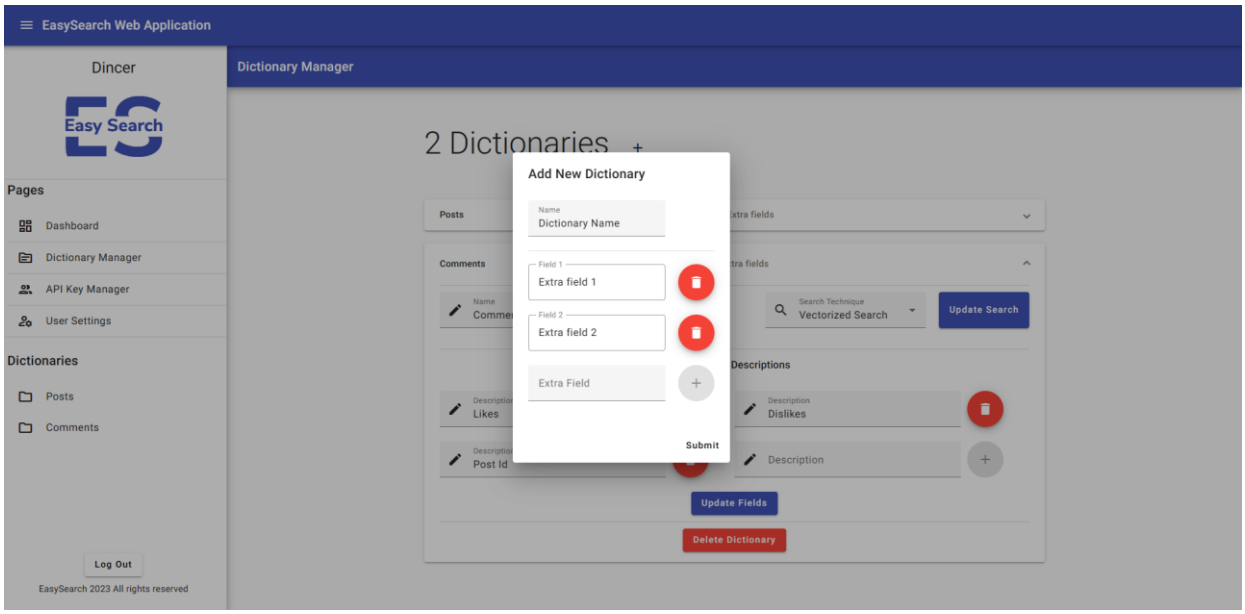

Figure 4 Add Dictionary Dialog

In the Dictionary Manager Page the user can manage or create their dictionaries.

In the figure 3, there is an example dictionary opened, the user can change the name of the dictionary, change their preferred search technique of the dictionary, and edit the extra field number and descriptions of the documents contained in the dictionary.

In the figure 4, there is the dialog for adding a new dictionary. The user can set the number of fields to whatever they need to store with their document.

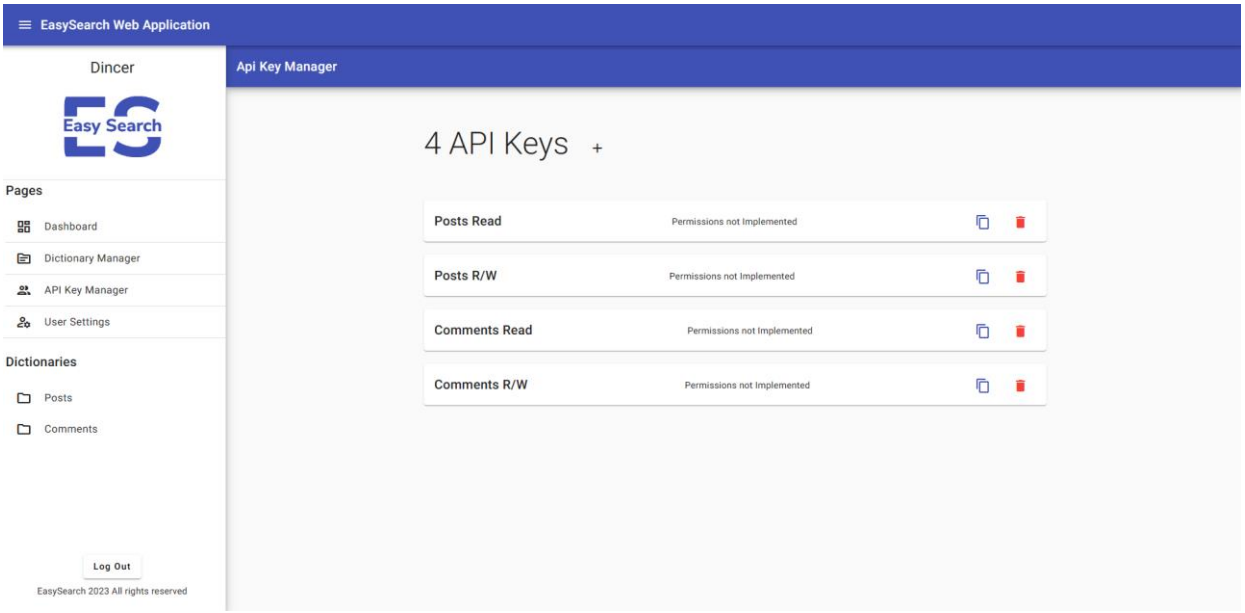

Figure 5 API Key Manager Page

In the figure 5, there are the API Keys that the user has created. The user can copy their values, or delete them. The API keys are used for Authenticating the API calls the user requests.

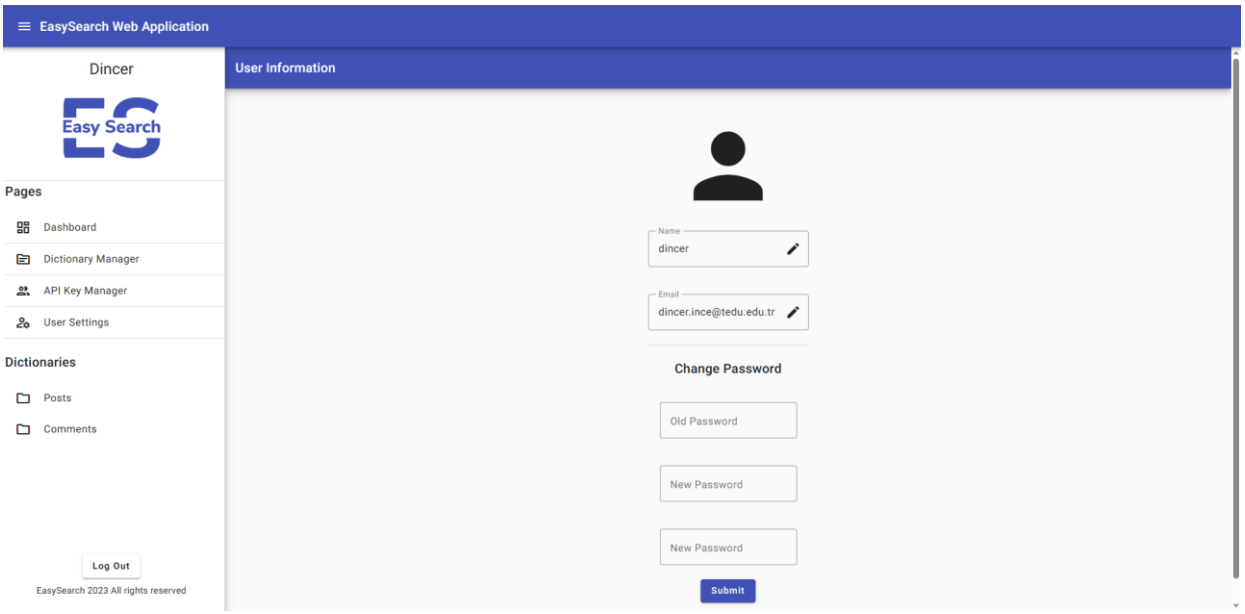

Figure 6 User Settings Page

In the User Settings page, the user can view and edit the information stored by the system. The user can change their email, password, and their name.

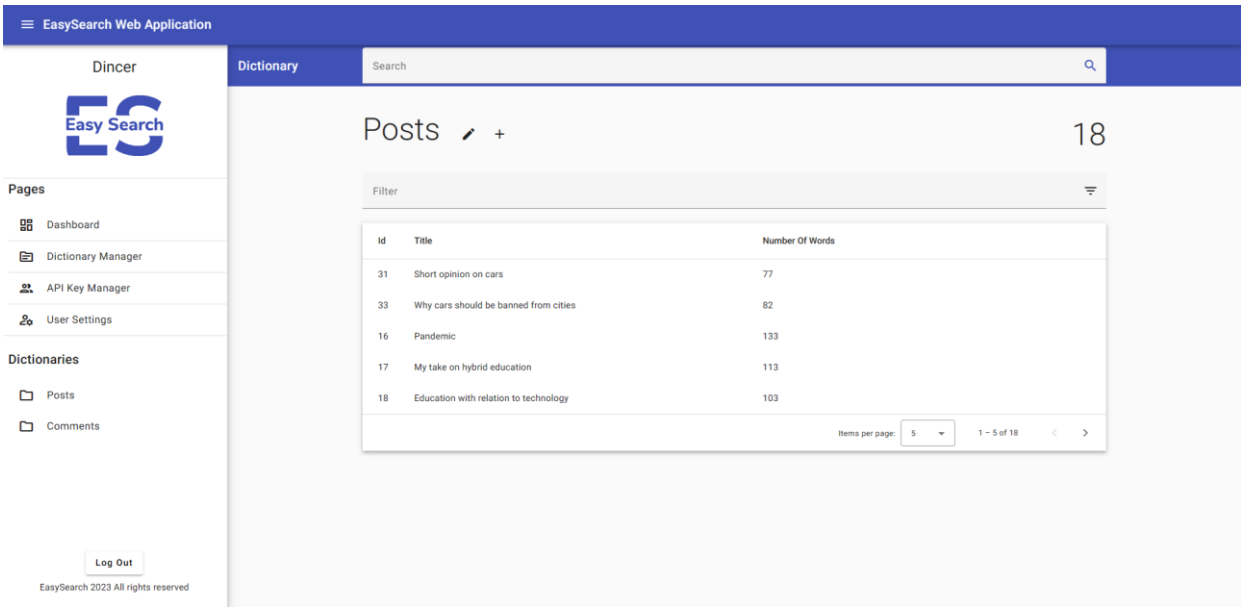

Figure 7 Dictionary Page

In the Dictionary Page, the user can see the documents inside of that dictionary. The documents are presented in a table that can be sorted and filtered.

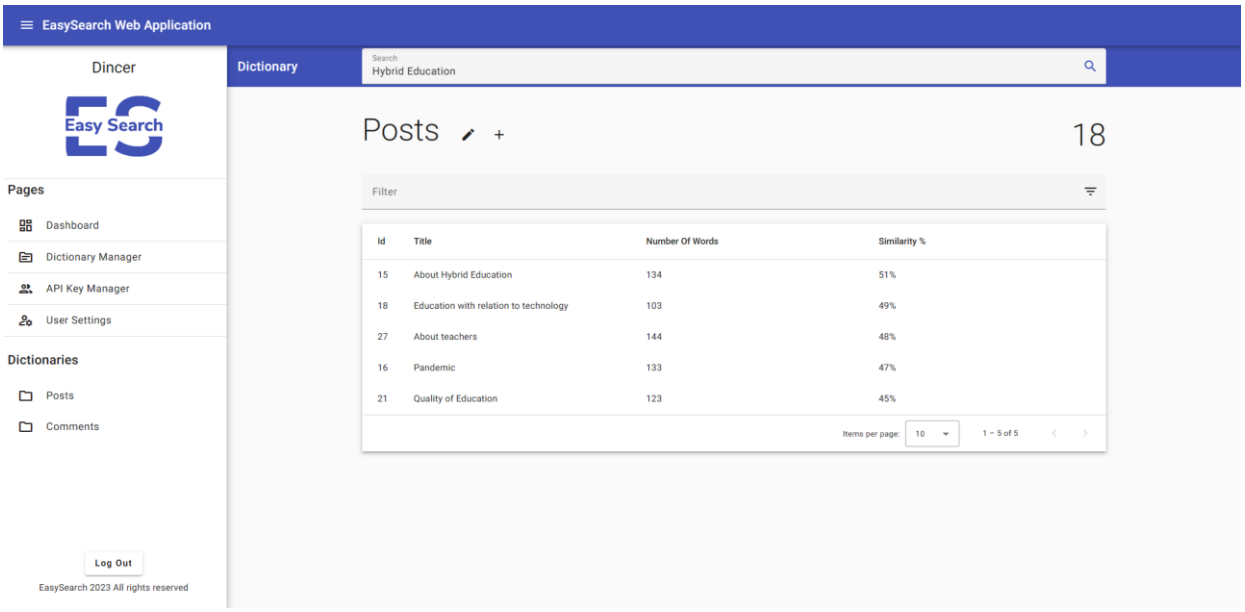

Figure 8 Dictionary Search

In the dictionary Page the user can perform search operation, and get the results sorted based on their similarity to the query. The search operation is performed based on the preferred search technique of the dictionary.

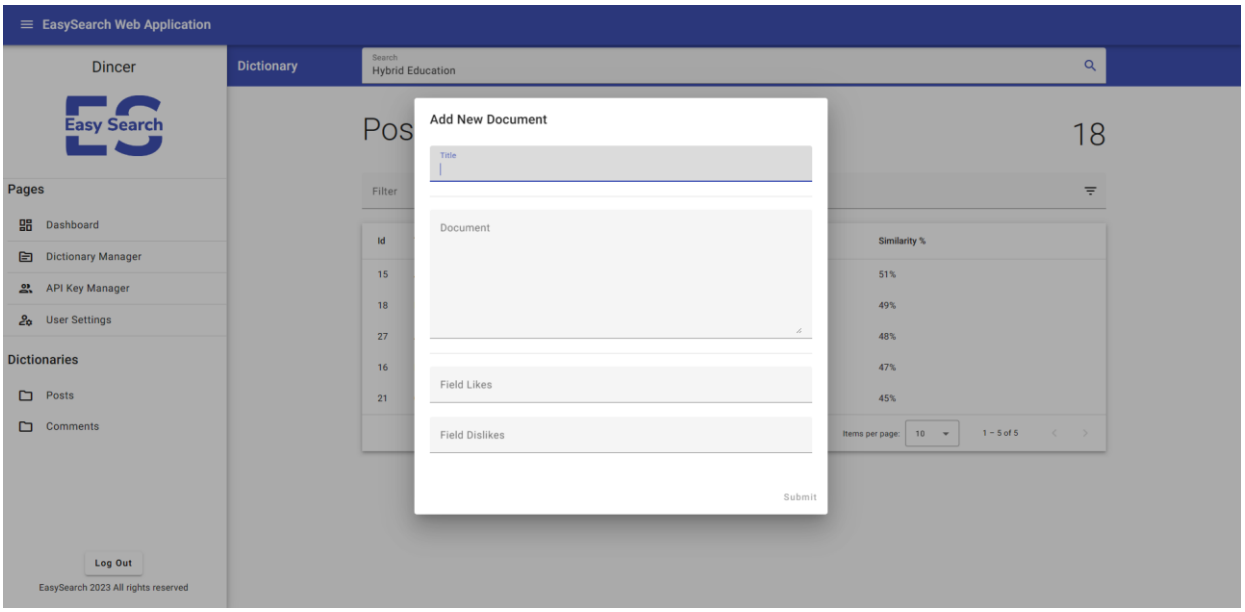

Figure 9 Add Dictionary dialog

Also in the same page the user can add new dictionary, with the dialog in the figure 9. The fields presented in the form comes from the field descriptions of the dictionary.

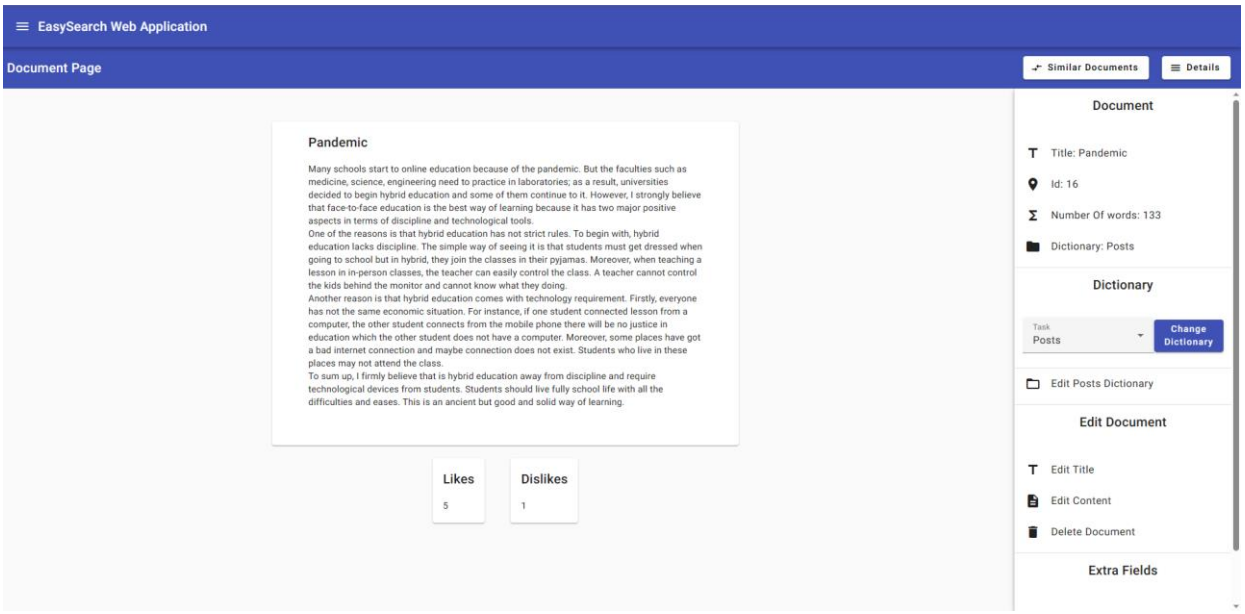

Figure 10 Document Page

After selecting a document from the dictionary page, the site navigates to the document page as seen in the figure 10. In the document page the user can view the document, as well as inspect the other information stored with it.

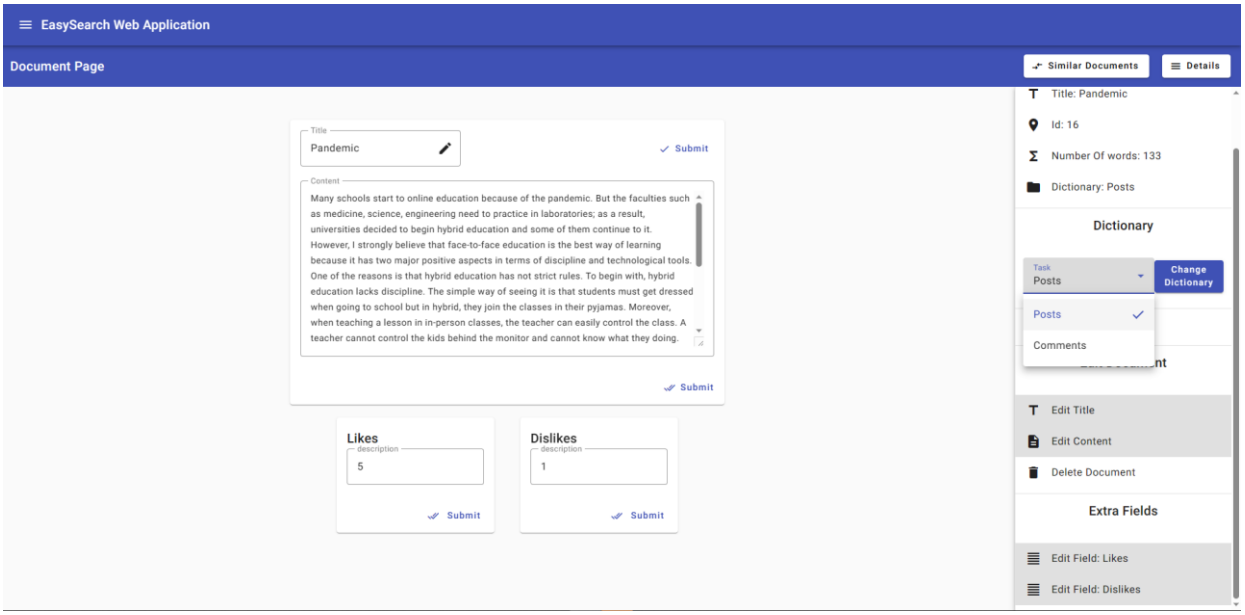

Figure 11 Document Editing Options

From the Side navigation bar in the document page, the user can edit the document by toggling the field to be edited. The toggling the field converts the field in the document view, to be an input which then the user can edit and submit.

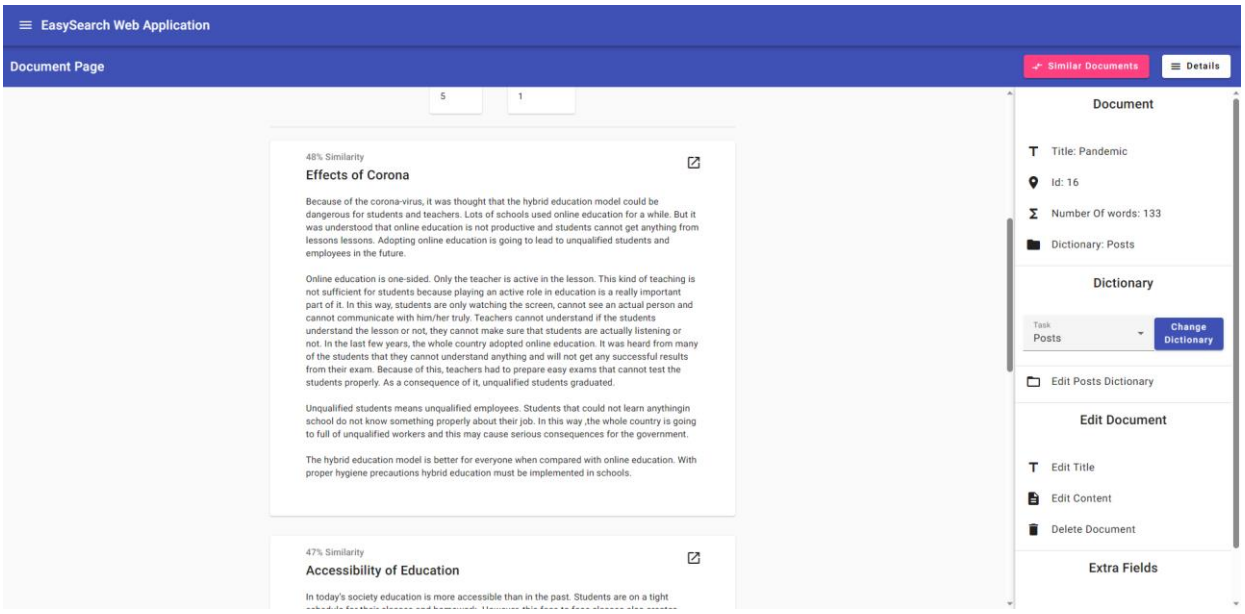

Figure 12 Similar Documents

Upon clicking the similar documents button, the page loads the similar documents contained in the same dictionary, on to the page. The results are loaded based on their similarity which the user can view, and the user can also navigate to the similar document with a button.

9.2 API

URL of the API: https://easysearchserver.azurewebsites.net/api/
Values presented inside curly brackets are variable values the user is expected to change.

9.2.1 Types used in the API

- Document
  - Id:integer
  - Title:string
  - RawDocument:string
  - NumberOfWords:integer
  - ExtraFields:string[]
  - DictionaryId:integer

- DocumentWithSimilarityModel
  - Id:integer
  - Title:string
  - RawDocument:string
  - NumberOfWords:integer
  - Similarity:double

- CreateDocumentDTO
  - DictionaryId:integer
  - Title:string
  - RawDocument:string
  - ExtraFields:string[]

- editContentDTO
    - text:string
    - id:integer

9.2.2 API Endpoints

- https://easysearchserver.azurewebsites.net/api/Documents/{DocumentId}
    - TYPE: GET
    - Inputs
        - DocumentId : Integer
            - Id of the document that is desired to be fetched
    - Returns
        - Document

- https://easysearchserver.azurewebsites.net/api/Documents/list/{DictionaryId}
    - TYPE: GET
    - Inputs
        - DictionaryId: Integer
            - Id of the dictionary for the list of documents to be fetched.
    - Returns
        - List of Documents

- https://easysearchserver.azurewebsites.net/api/Documents
    - TYPE: POST
    - Inputs
        - In body : CreateDocumentDTO
            - DTO of the Document object that is to be created in the dictionary.
    - Returns
        - Integer

- https://easysearchserver.azurewebsites.net/api/Documents/{DocumentID}
    - TYPE: DELETE
    - Inputs
        - DocumentID : Integer
    - Returns
        - Boolean
            - Whether the operation was successful.

- https://easysearchserver.azurewebsites.net/api/Documents/DocumentSimilarity/{ID1}/{ID2}
    - TYPE: GET
    - Inputs
        - ID1 : Integer
            - Id of the document

        - ID2 : Integer
            - Id of the document that is compared with
    - Returns
        - Double
            - The similarity value between these documents.

- https://easysearchserver.azurewebsites.net/api/Documents/similarDocuments/{length}/{DocumentId}
    - TYPE: GET
    - Inputs
        - Length : integer
            - Number of documents to be sent back
        - DocumentId : integer
            - The document that is compared with the rest of the dictionary.
    - Returns
        - List of DocumentWithSimilarity

- https://easysearchserver.azurewebsites.net/api/Documents/search/{dictionaryId}/{query}
    - Type: GET
    - Inputs
        - dictionaryId : Integer
            - The Id of the dictionary in which the search is performed.
        - Query : string
            - The search query
    - Returns
        - List of DocumentWithSimilarity

- https://easysearchserver.azurewebsites.net/api/Documents/query/{dictionaryId}/{fieldIndex}/{query}/{length}
    - TYPE: GET
    - Inputs
        - dictionaryId: integer
            - Id of the dictionary in which the document with the desired field is stored
        - fieldIndex: integer
            - Index of the field stored in the document that is desired to be queried.
        - query: string
            - The value that is desired to be found in the field of the document.
        - length: integer
            - Number of documents to be sent back.
    - Returns
        - List of Documents

- https://easysearchserver.azurewebsites.net/api/Documents/changeDocumentTitl

e?document_id="{documentId}"&title="{title}"

- TYPE: PUT
- Inputs
  - documentId: integer
    - The Id of the document that is desired to be edited
  - title:string
    - New title
- Returns
  - Boolean
    - Whether the operation was successful.

- https://easysearchserver.azurewebsites.net/api/Documents/changeDocumentRawDocument
  - TYPE: PUT
  - Inputs
    - In body : editContentDTO
  - Returns
    - Boolean
      - Whether the operation was successful

- https://easysearchserver.azurewebsites.net/api/Documents/editDocumentField?document_id="{documentId}"&index="{fieldIndex}"&text="{text}"
  - TYPE: PUT
  - Inputs
    - documentId: integer
      - Id of the documents that is desired to be edited.
    - index:integer
      - Index of the field that is desired to be edited in the document.
    - text:string
      - New field value
  - Returns
    - Boolean
      - Whether the operation was successful

10.References

- *Angular Documentation*. Angular. (n.d.). https://angular.io/docs

- BillWagner. (n.d.). .*Net documentation*. Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/

- *Cosine-Similarity*. TF-IDF transformations for text preprocessing. (n.d.). https://cosinesimilarity.org/content/tf-idf.html

- Jcjiang. (n.d.). *Entity Framework Documentation Hub*. Microsoft Learn. https://learn.microsoft.com/en-us/ef/

- *RxJS Documentation*. RxJS. (n.d.). https://rxjs.dev/guide/overview