Cairo University
Faculty of Computers and Artificial Intelligence
Computer Sciences Department

e-book SPHERE

Supervised by

Dr. Manar ElKady

TA. Menna Asfour

Implemented by

20200104	Eman Ramadan Mostafa Mansour
20200632	Yasmin Ahmed Omar Abdallah
20200209	Ziad Mysara Yousef Mohamed
20200034	Ahmed Magdy Fahmy Mohamed
20200360	Omar Mahmod Abdelhamied Hassan

Graduation Project
Academic Year 2023-2024
Final Year Documentation

Table of Content

Table of Figures

Table of Lists

Table of Abbreviations

VADER	Valence Aware Dictionary and sEntiment Reasoner	OCR	Optical Character Recognition
NLP	Natural Language Processing	BERT	Bidirectional Encoder Representations from Transformers
API	Application Programming Interface	PDF	Portable Document Format
NLTK	Natural Language Toolkit	EPUB	electronic publication
AI	Artificial Intelligence	Dotx	Word Open XML Document Template

Chapter 1: Introduction

1.1 Motivation

The proliferation of digital content and the growing trend towards eBooks have revolutionized how readers access and consume literature. However, authors and readers face numerous challenges in creating, managing, and discovering eBooks. Authors often struggle with intuitive tools for eBook creation, formatting, and publishing, while readers find it difficult to discover and access high-quality eBooks that match their interests. This project, eBook Sphere, aims to bridge this gap by providing a comprehensive platform for authors to create and publish eBooks and for readers to explore and enjoy them.

1.2 Problem definition

The primary problem addressed by eBook Sphere is the lack of an integrated, user-friendly platform that caters to both authors and readers. Existing platforms either focus solely on eBook creation without adequate publishing and discovery features or provide limited tools for readers to find relevant content. Authors need a seamless way to create, edit, and publish eBooks, while readers require advanced search and recommendation systems to find content that suits their preferences.

1.3 Project Objective (suggested solution)

The objective of eBook Sphere is to develop a unified platform that supports the entire lifecycle of eBooks, from creation to consumption. The platform will offer authors a suite of tools for creating, formatting, and publishing eBooks, complete with template selection and export options in various formats. For readers, the platform will provide advanced search functionalities, including semantic search powered by deep Learning, to enhance the discovery of relevant eBooks. The solution aims to improve the user experience for both authors and readers, facilitating better content creation and discovery.

1.4 Gantt chart of project time plan

The project time plan is divided into distinct phases, each with specific milestones and deliverables. The Gantt chart below outlines the timeline for project analysis, implementation, and testing:

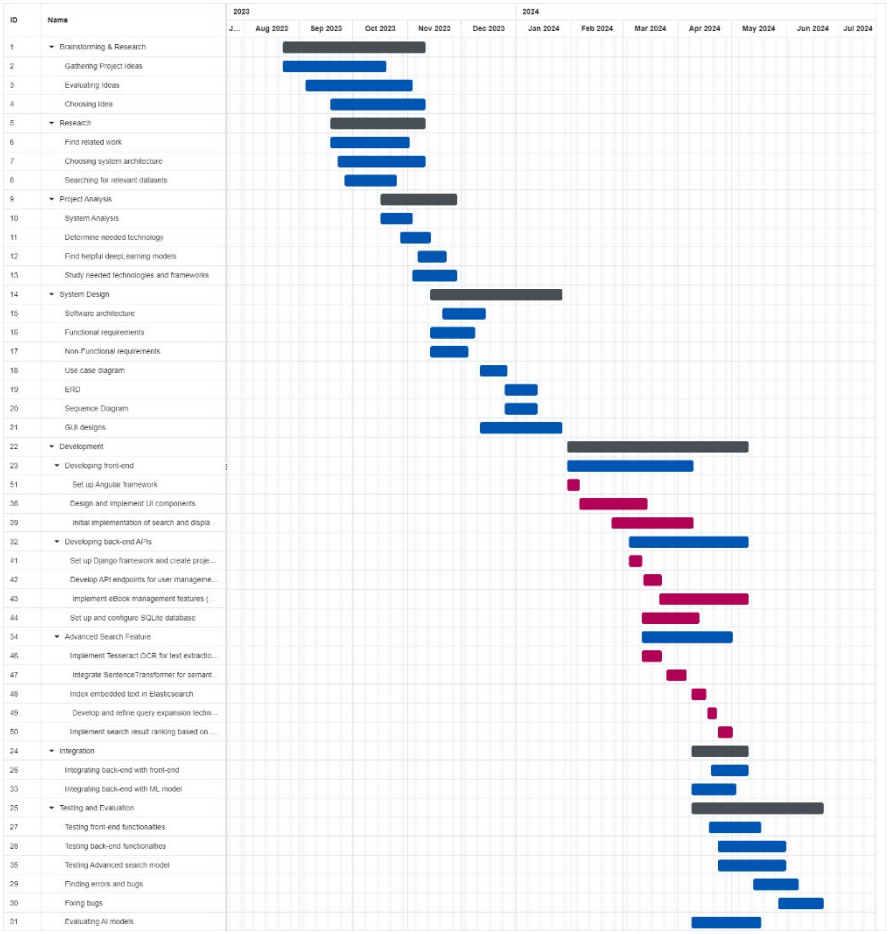

Figure 1 "Gantt chart"

1.5 Project development methodology

The E-book Sphere project employs the Waterfall development methodology, which is well-suited for the structured and systematic development of our eBook creation and editing platform. This linear and sequential approach ensures that each phase is completed thoroughly before moving to the next, providing a solid framework for our project.

1. Requirement Analysis:

- In this phase, we gathered and documented all the requirements for E-book Sphere, including features such as text editing, formatting, previewing, keyword identification, and support for multiple eBook formats (PDF, EPUB, ODT, TXT, Docx, Dotx). We also considered user roles (authors, readers, and administrators) and their specific needs. This comprehensive requirement specification forms the foundation for the entire project.

2. System Design:

   - Based on the documented requirements, we designed the architecture and components of the E-book Sphere platform. This included creating detailed system design documents that outlined the overall structure, user interface design, database schemas, and the integration of deep learning techniques for search. The design phase also involved selecting the appropriate technologies, such as Angular for the front-end, Django for the back-end, and SQLite for the database.

3. Implementation:

   - During this phase, we developed the E-book Sphere platform according to the design specifications. This involved coding the software components, integrating the system modules, and ensuring that all features, such as text editing, formatting, previewing, and multi-format support, were implemented effectively. We utilized Python for back-end development and integrated BERT and Synsets for enhanced search capabilities.

4. Integration and Testing:

   - We rigorously tested the E-book Sphere platform to verify that it met all requirements and functioned correctly. This included unit testing, integration testing, system testing, and acceptance testing to ensure that all features worked as intended and provided a seamless user experience. Specific attention was given to the search functionality, ensuring accurate and efficient document retrieval using our advanced search engine.

❖ By following the Waterfall methodology, the E-book Sphere project ensured a clear structure and thorough documentation, which reduced the likelihood of major changes or revisions late in the development process. This methodical approach allowed us to deliver a robust and user-friendly eBook creation and editing platform.

1.6 The used tools in the project (SW and HW)

1.6.1 Environment:

- **VSCode:** development environment

- **Github:** for Version Control with backend
- **Postman:** for testing API's

1.6.2 Front-end:

- **HTML/CSS/JavaScript**: Core technologies for building the structure, styling, and interactivity of the website.
- **Angular**: A platform and framework for building single-page client applications using TypeScript.
- **Bootstrap**: A front-end framework for developing responsive and mobile-first websites.

1.6.3 Back-end:

- **Python**: The primary programming language used for back-end development.
- **Django REST Framework**: A powerful and flexible toolkit for building Web APIs, allowing the back-end to communicate effectively with the front-end.
- **SQLite (db.sqlite3)**: A lightweight, disk-based database used for storing and managing data in development.

1.6.4 Machine learning:

- **Tesseract OCR**: An open-source Optical Character Recognition engine used for text extraction from documents.
- **SentenceTransformer**: A Python framework for sentence embeddings, which converts text into dense vector representations.
- **WordNet**: A lexical database for the English language that provides synonyms and antonyms for query expansion.
- **Google Generative AI (Gemini)**: A generative AI model used to enhance query expansion with additional related terms.
- **BERT (Bidirectional Encoder Representations from Transformers)**: A transformer-based machine learning technique for natural language processing pre-training, used for embedding text.
- **ElasticSearch**: Used as part of the search engine for indexing and retrieving document embeddings based on semantic similarity.

1.7 Report Organization (summary of the rest of the report)

The rest of this report is organized as follows:

- **Chapter 2: Related Work**

  Reviews similar eBook creation platforms and identifies key differences and advantages of E-book Sphere.

- **Chapter 3: System Analysis**

  Details the functional and non-functional requirements of the system and provides use case diagrams.

- **Chapter 4: System Design**

  Describes the system's architecture, including component and class diagrams, sequence diagrams, ERD, and GUI design.

- **Chapter 5: Implementation and Testing**

  Discusses the challenges faced during implementation, solutions employed, detailed implementation process, algorithms used, and API implementation.

- **Summary**

  The E-book Sphere project aims to revolutionize the eBook creation process by providing an all-in-one, user-friendly platform for authors. The project addresses common challenges in eBook creation such as text formatting, keyword identification, and multi-format support. By employing a Waterfall development methodology, the project ensures a structured and systematic approach with distinct phases of development, leading to a thoroughly tested and reliable product. Utilizing robust software tools like Django REST Framework, Angular, and deep learning techniques, along with high-performance hardware, the project is set to deliver a comprehensive solution. The E-book Sphere platform not only simplifies eBook creation but also enhances reader engagement, contributing to the growth of digital literature.

Chapter 2: Related Work

2.1 The closest examples

In the realm of eBook creation and editing, several platforms and tools have emerged to aid authors and publishers. Here, we discuss some of the most notable examples that are closest to the E-book Sphere project in terms of functionality and objectives.

1. **Venngage**

versatile graphic design platform specializing in infographic creation. With a user-friendly interface and a diverse range of templates, it allows users to easily design and visualize information for various purposes. From creating eye-catching social media posts to detailed business reports, Venngage simplifies the design process, making it accessible for both professionals and beginners.

2. **Visme**

Visme is a versatile design platform providing templates for infographics, presentations, and social media graphics. Notably, it doubles as an eBook maker, offering an intuitive interface and diverse design elements for users to easily create visually appealing digital books.

3. **The Verge**:

The Verge is a prominent online platform known for its tech and culture coverage, offering articles, reviews, and news on a variety of topics. While it primarily serves as an information hub, it does not function as an eBook reading platform. Users can explore a wide range of content and stay updated on the latest trends in technology and culture, but The Verge does not provide specific features for reading eBooks.

2.2 Main Differences

Features	Our System	Venngage	Visme	The Verge
Make eBook	✓	✓	✓	✗
Read eBook	✓	✗	✗	✓
Search by word	✓	✗	✓	✗
Rating	✓	✗	✗	✓
Comment	✓	✗	✗	✓
Easy to use	✓	✓	✗	✓
Download	✓	✓	✓	✓
Complete Words	✓	✗	✗	✗
All Service is Free	✓	✓	✗	✗

Table 1 "Related Work Summary"

To summarize the similarity and differences between the features of our system and the others

❖ **Main Differences between our system and theirs:**

It's free, unlike most apps that require payment for advanced uses.
It combines writing an e-book and reading it, unlike all platforms, as no site does both.
You can comment or rate a book, and you can search for everything you want using a word, and everything related to the word will be displayed, there is a process of completing the speech, downloading the book, and then printing it.

Chapter 3: System analysis

3.1 Functional requirements

➢ **For User**

- Sign up: Users can register for the service.
- Sign in: Registered users can log in to their accounts.
- View & edit profile information: User can view and edit his profile information.
- Change password: User can change his password.

- Forgot password: User can reset his password using mail.
- create eBook: Users can generate a new eBook.
- Open an existing eBook: Users can open an existing eBook.
- Choose template: User can choose template to create his eBook.
- open eBook Maker: User can open eBook Maker.
- preview template: User can preview the template before selecting it.
- Edit eBook: Users can make changes to their eBooks.
- Format eBook: Ability to format the content/layout of the eBook.
- Save eBook: Users can save their created eBooks.
- Preview and print eBook: Users can preview how the eBook will appear and print it if he needs.
- Export eBook: Users can export the created eBook in different formats like .epub, .pdf, .odt, .docx, .dotx, txt.
- Publish eBook: Users can publish their eBooks.
- Select Categories: Ability to choose categories for eBook when creating publish request.
- view eBooks: User can view eBooks.
- Show his own eBooks: User can see his own eBooks which are created in our website .
- Show Dashboard: User can see his dashboard that contains the number of eBooks, and for selected book he can see the number of readers, the number of comments, info graph for positive and negative comments, info graph for progress of reading this book.
- Collaborate with other users: User can collaborate with other users and Author can collaborate with users.
- filtering eBooks: User can filter eBooks by categories.
- Show eBook details: User can see the details of the eBook.
- Read eBooks: Users can read the eBooks.
- Download eBooks: Users can download the eBooks as pdf.
- Feedback: User can add comment on an eBook and rate it.
- Add to favorite: User can add eBook to his favorite list.
- Remove from favorite: User can remove eBook from his favorite list.
- Show his favorite eBooks: User can see his favorite eBooks.
- View Reading progress: User can see his reading progress.

- search in website: The user can search about main function in the website.
- Advanced Book Search: The user can search by book name or Author name and by using deep learning.
- Contact Us: User can contact with admin for any query.

➢ **For Admin**

- Review eBook: The administrator can evaluate eBooks and choose to either accept or reject them for display on the website's "Read" section.
- Control comments: The administrator can control the comments on the eBooks.
- Control Categories: The administrator can add/Remove the categories.
- Control templates: The administrator can add/Remove the templates.
- Control Users: The administrator can add/Remove the users.
- Control eBooks: The administrator can add/Remove the eBooks.

3.2 Non-functional requirements

1. **Performance:**
   - ***Response Time*:** Ensure the system responds promptly to user actions (e.g., loading eBooks, generating previews) within acceptable time limits.
   - ***Scalability:*** The platform should accommodate an increasing number of users and eBooks without compromising performance.
   - ***Reliability:*** The system should be reliable, minimizing downtime or errors during peak usage.
   - ***Usability:*** The interface should be user-friendly and accessible, allowing users to perform tasks with minimal effort (e.g., fewer mouse clicks).
2. **Security:**
   - User information encryption: Ensuring user data remains secure and encrypted to prevent unauthorized access.
   - Error Handling and Logging: Implement effective error handling mechanisms to inform users about errors gracefully. Also, maintain logs for troubleshooting and analysis purposes.

- Email Verification: The password reset functionality employs secure token-based authentication over HTTPS, verifies user identity via email, and includes measures to ensure robust security and protect against unauthorized access.

3. **Extensibility**
   - Flexible to add and develop more features to keep up with the current users' needs or to fix a specific problem.

3.3 Use Case Diagram

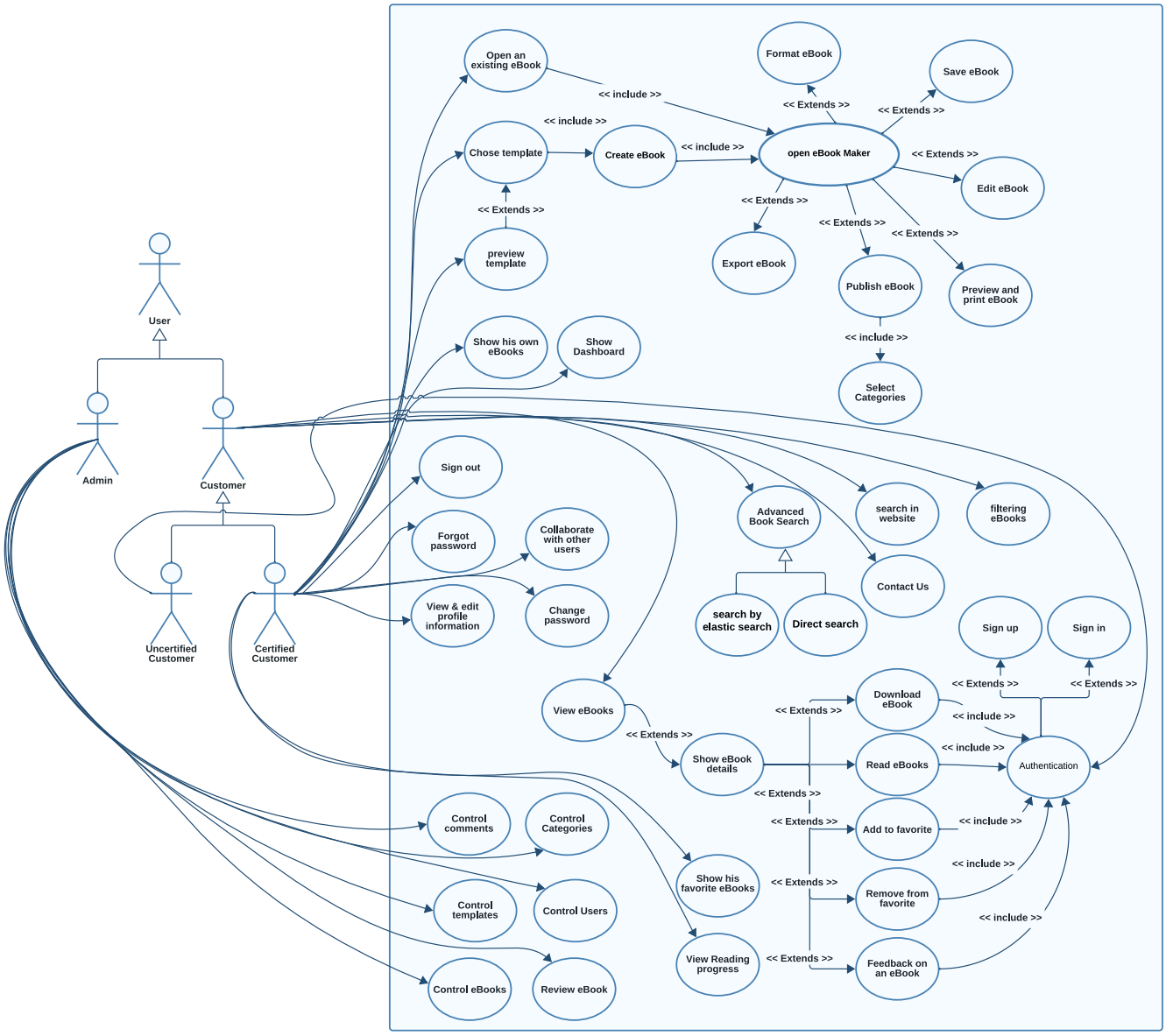

Figure 2 "Use Case Diagram"

Chapter 4: System Design

4.1 System Component Diagram

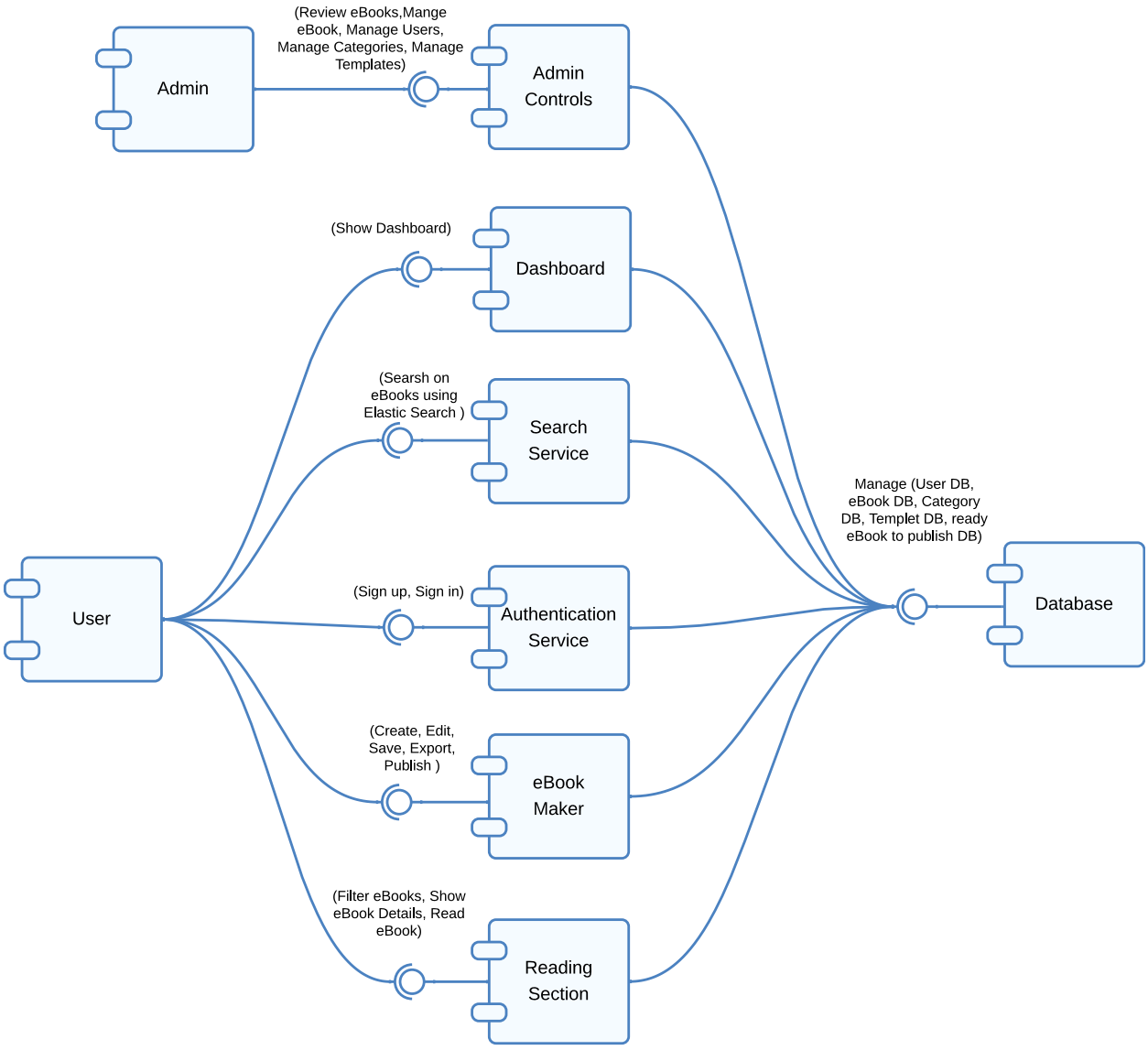

Figure 3 "System Component Diagram"

4.2 System Class Diagram

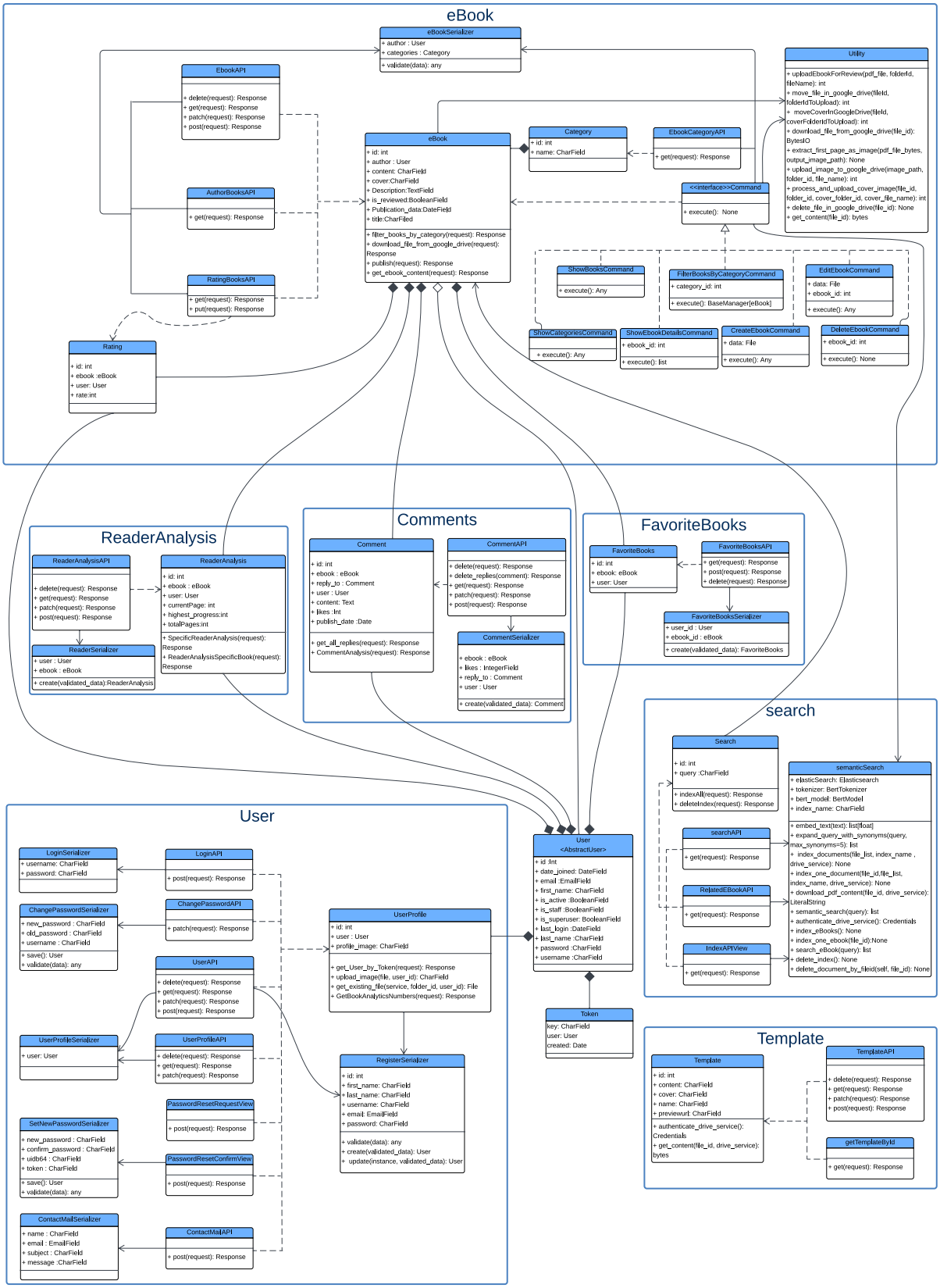

Figure 4 "System Class Diagram"

4.3 Sequence Diagrams

4.3.1 Publish

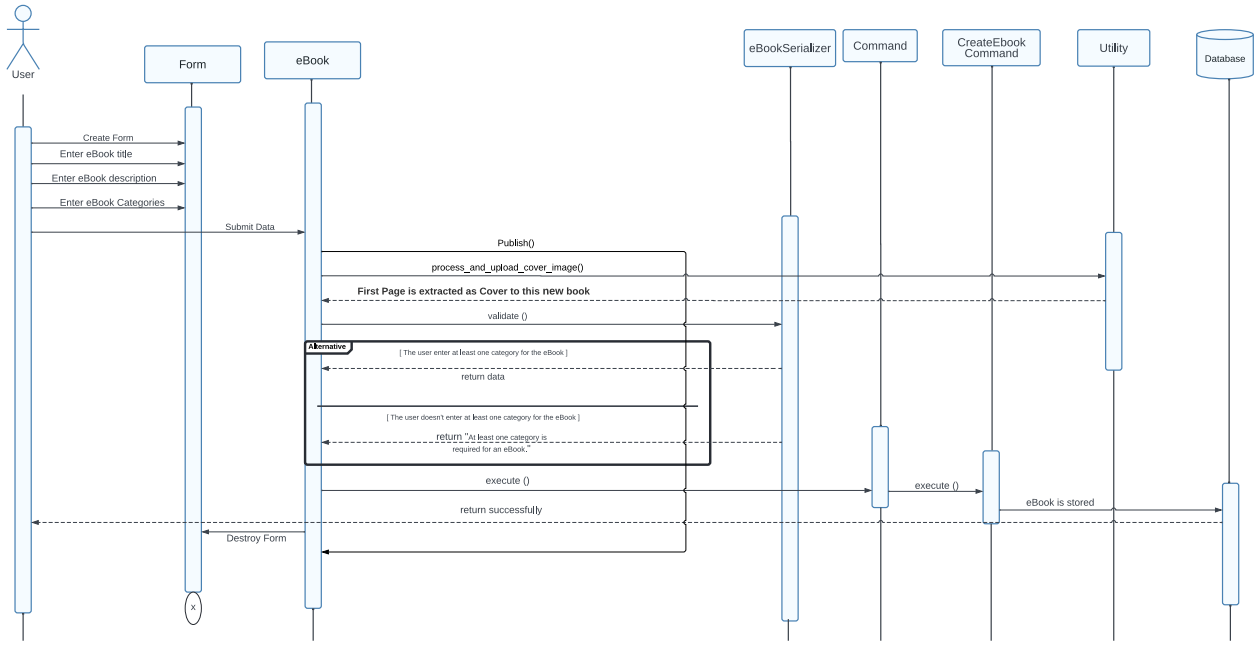

Figure 5 "Sequence Diagram: Publish"

4.3.2 Admin

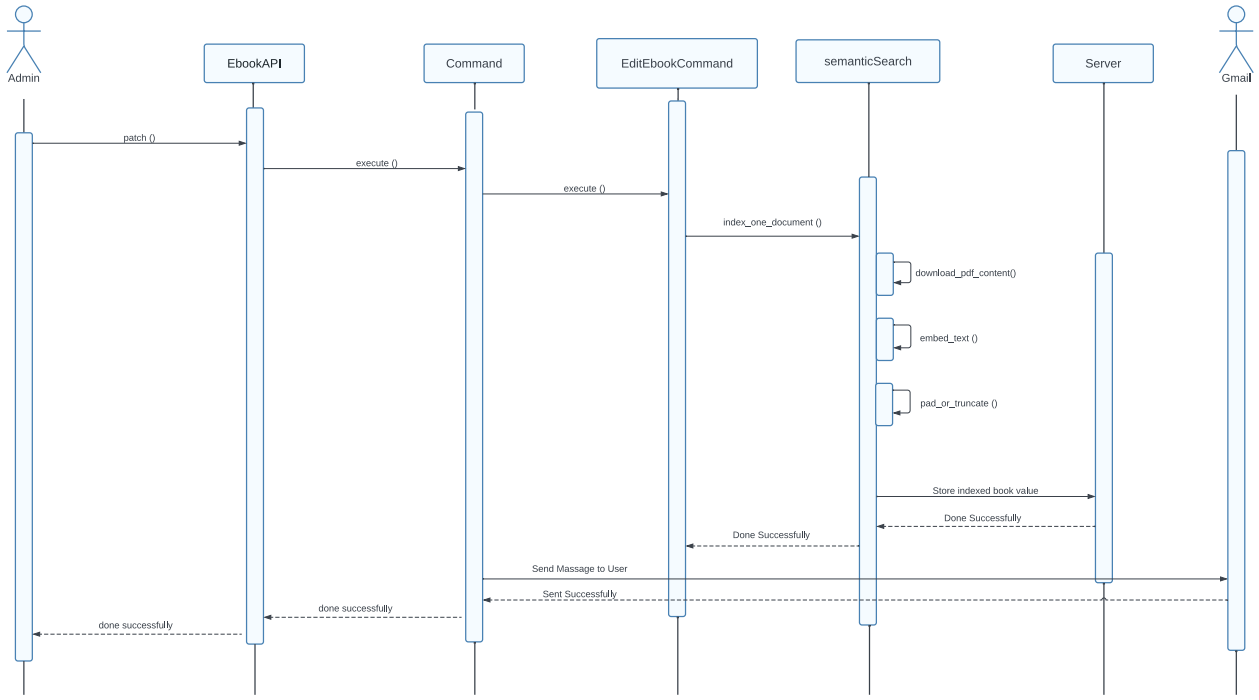

Figure 6 "Sequence Diagram: Admin"

4.3.3 Indexing

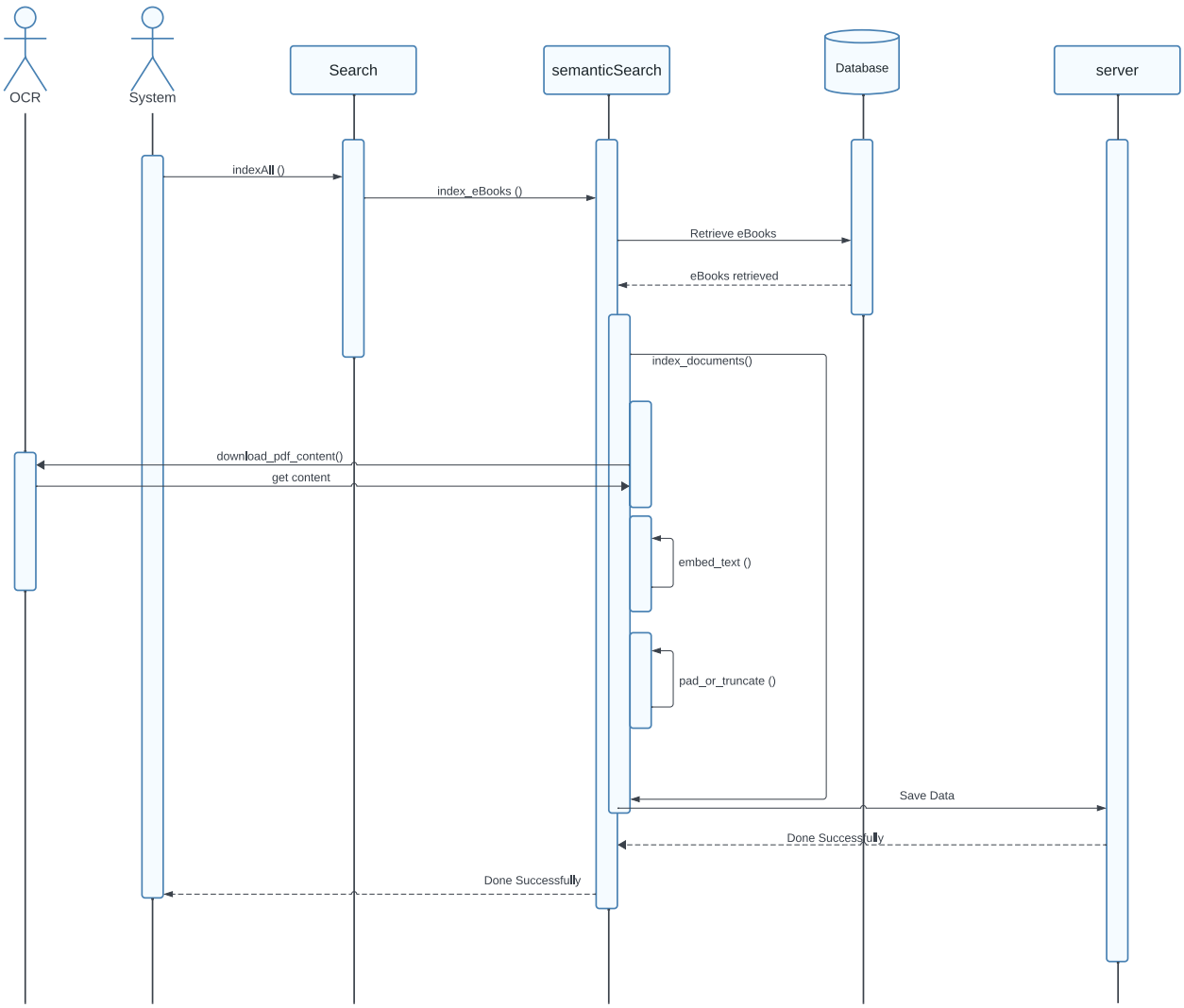

Figure 7 "Sequence Diagram: Indexing"

4.3.4 Elastic Search

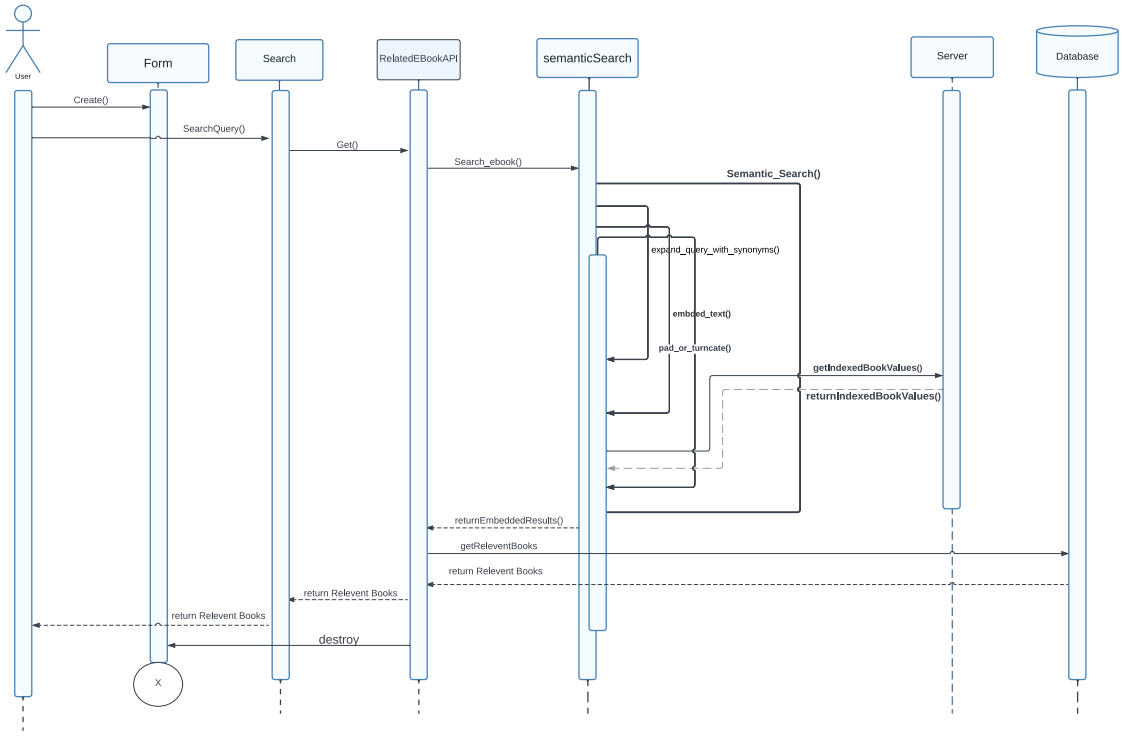

Figure 8 "Sequence Diagram: Elastic Search"

4.3.5 eBook and comment analysis

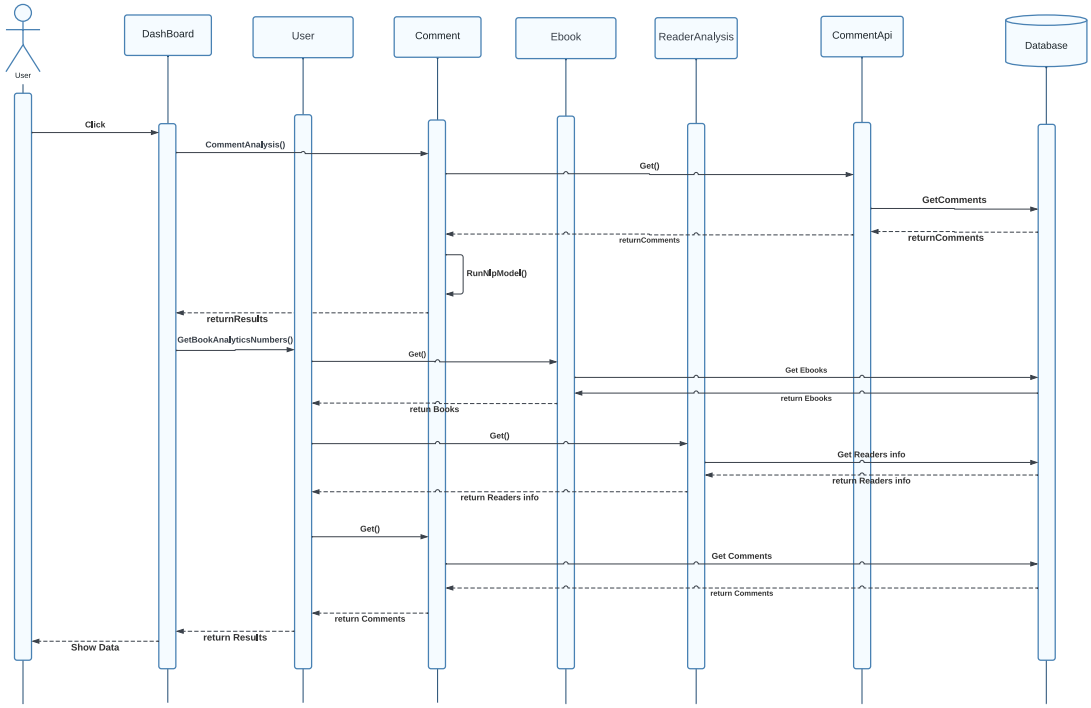

Figure 9 "Sequence Diagram: eBook and comment analysis"

4.4 Project ERD

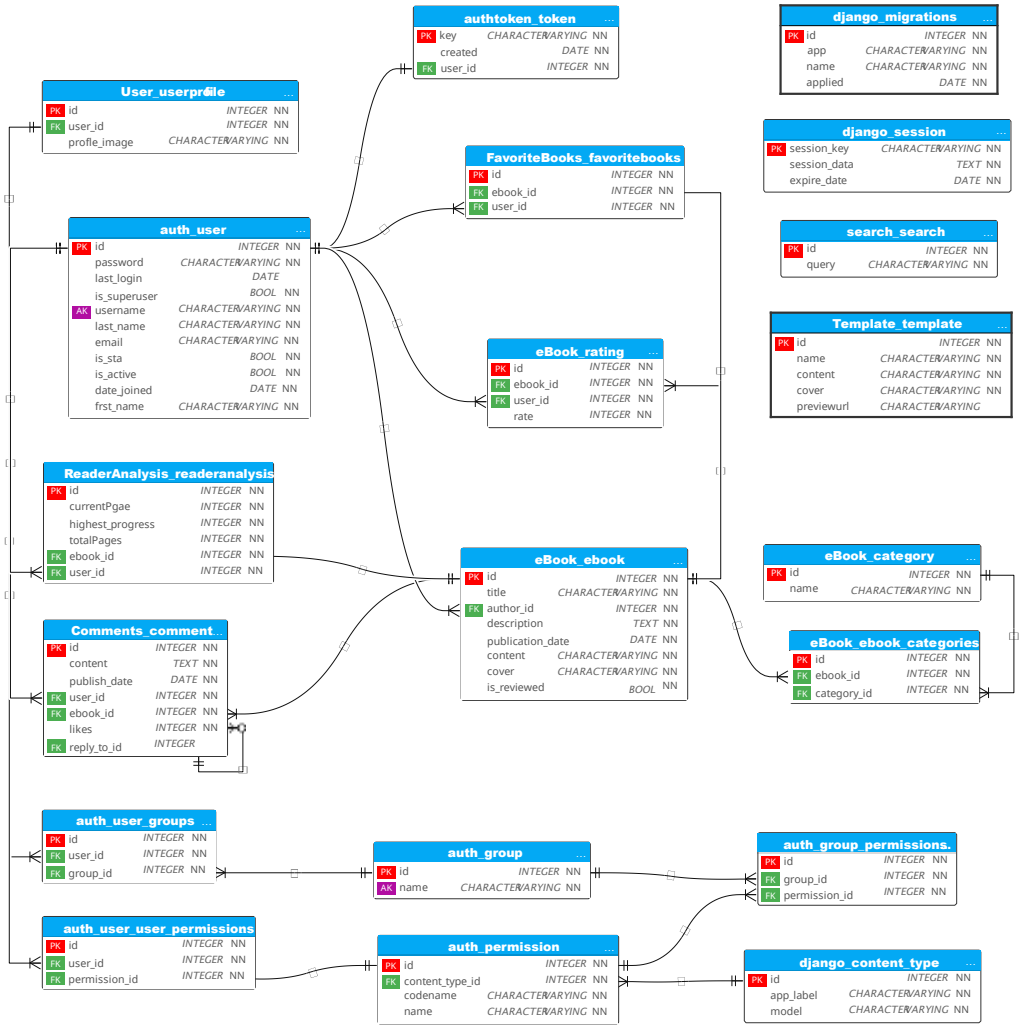

Figure 10 "Project ERD"

4.5 System Architecture

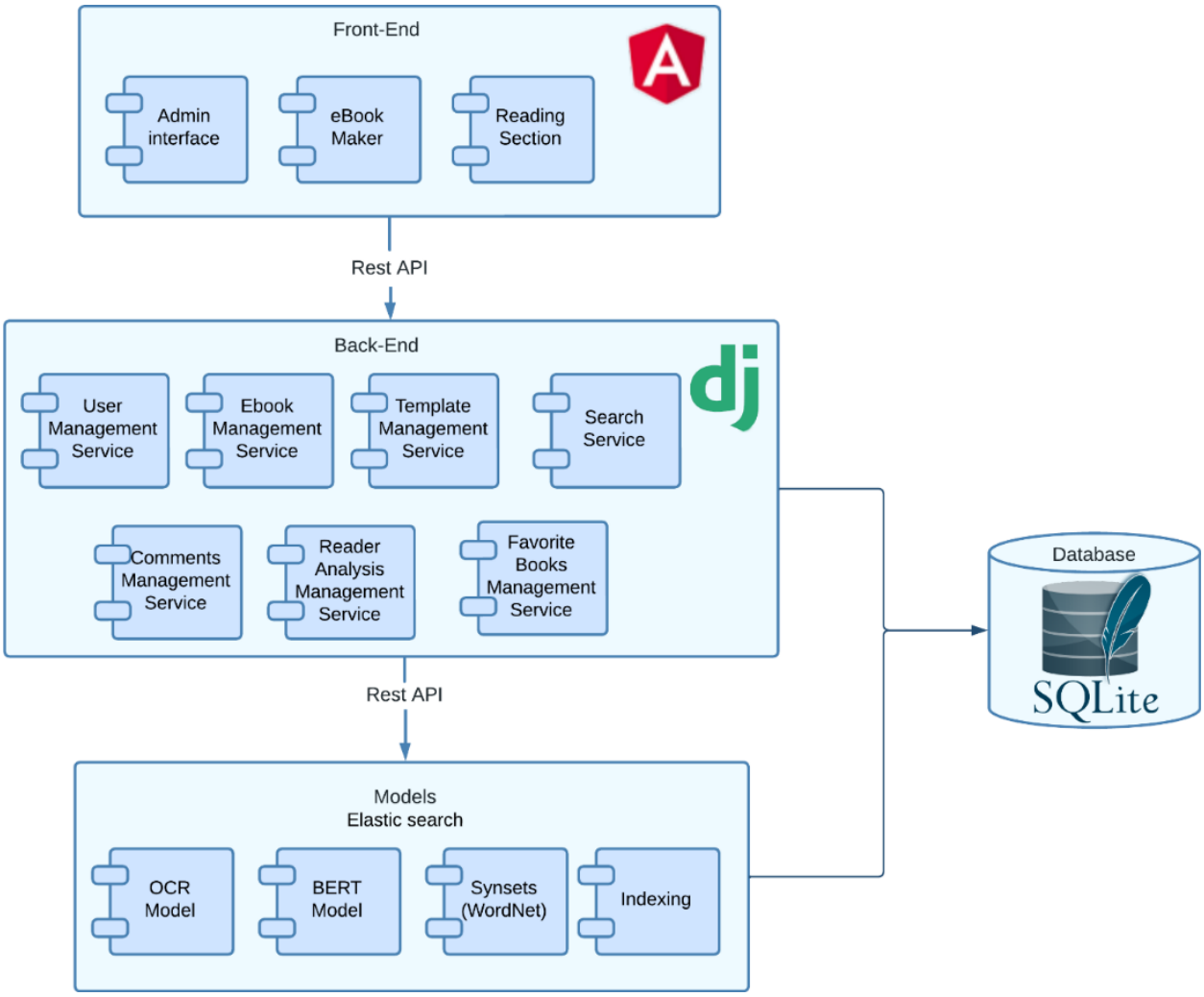

Figure 11 "System Architecture"

The system architecture of the E-book Sphere project is designed as a three-layered structure to ensure efficient and effective eBook creation, editing, and searching capabilities. Each layer has specific components that work together to provide a seamless user experience. The layers and their components are outlined below:

4.5.1 Front-End Layer

This layer is responsible for the user interface and interaction, providing a responsive and intuitive platform for users.

- **Admin Interface**: Provides administrative functionalities to manage users, eBooks, and system settings.
- **eBook Maker**: A feature-rich interface that allows users to create, edit, and format eBooks.

- **Reading Section**: A user-friendly reading interface where users can read eBooks, add comments, and interact with the content.

4.5.2 Back-End Layer

This layer handles the server-side operations and services that support the front-end functionalities.

- **User Management Service**: Manages user accounts, authentication, and authorization.
- **eBook Management Service**: Handles operations related to eBook creation, editing, and storage.
- **Template Management Service**: Manages templates used in eBook creation for consistent formatting and styling.
- **Search Service**: Implements the search functionality, leveraging models and indexing to provide relevant search results.
- **Comments Management Service**: Manages user comments on eBooks, enabling interaction and feedback.
- **Reader Analysis Service**: Analyzes reading habits and preferences to provide personalized recommendations and insights.
- **Favorite Books Management Service**: Allows users to mark and manage their favorite books.

4.5.3 Models and Search Layer

This layer comprises advanced models and search capabilities that enhance the functionalities provided by the back-end services.

- **OCR Model (Tesseract OCR)**: Extracts text from image-based eBooks to make them searchable.
- **BERT Model**: Used for embedding queries and documents, capturing the contextual meaning of words to improve search accuracy.
- **Synsets(WordNet, Gemini)**: Expands user queries with synonyms to enhance search comprehensiveness.
- **Indexing :**Ensures efficient and accurate document retrieval by indexing eBooks based on their content.

4.6 System GUI Design

Here is some main section in our website:

4.6.1 Home:

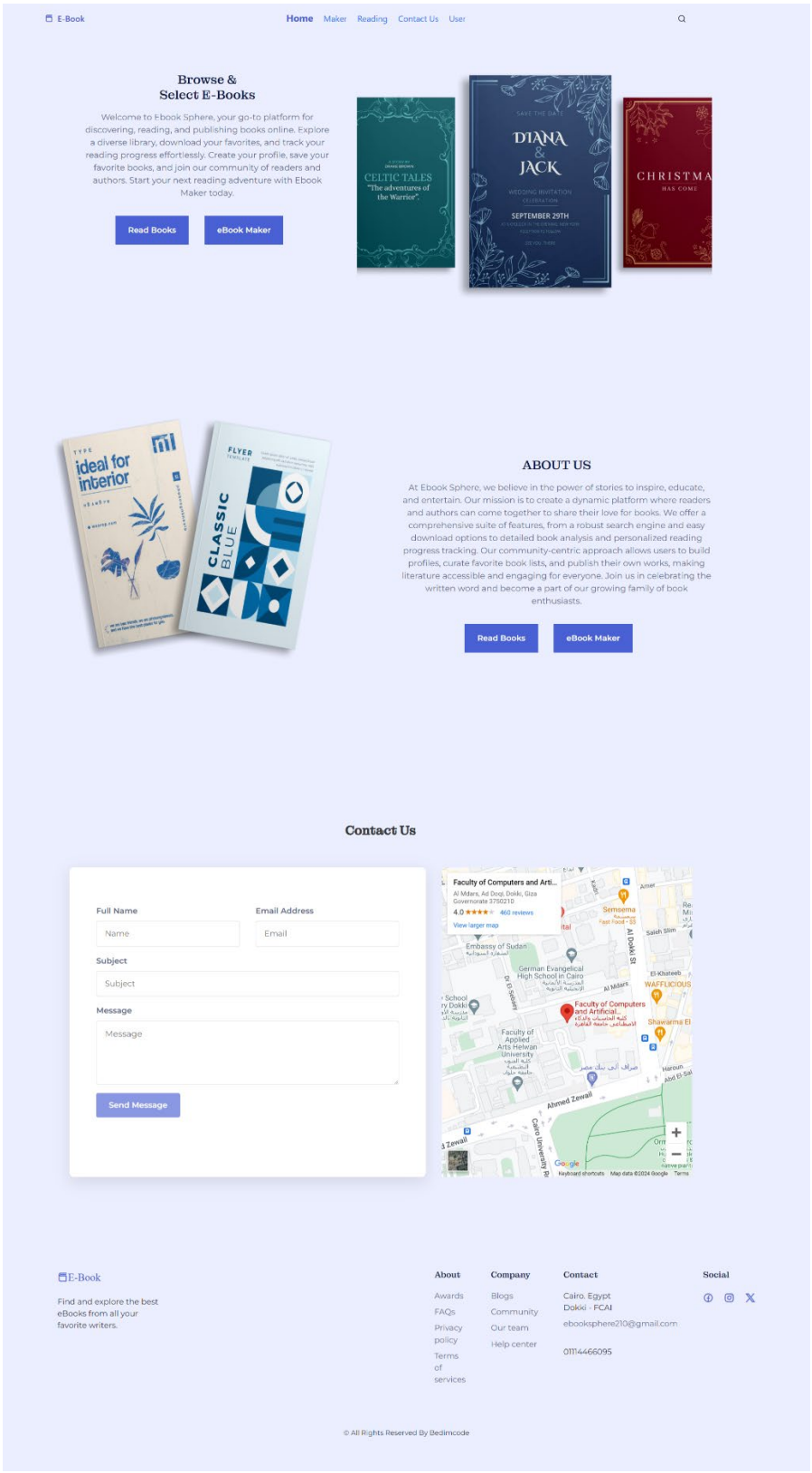

Figure 12 "GUI Design: Home"

4.6.2 Reading Section:

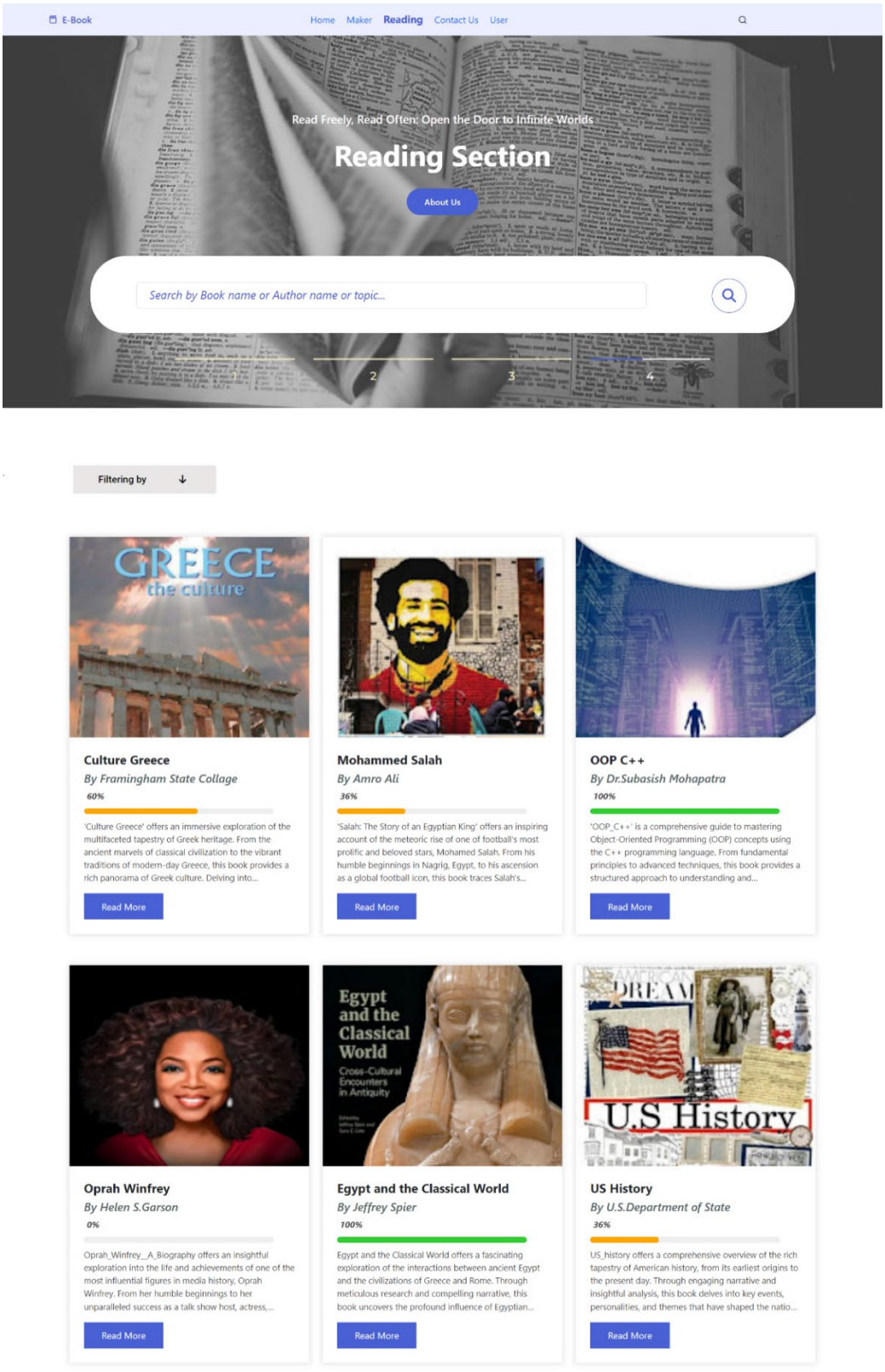

Figure 13 "GUI Design: Reading Section"

4.6.3 Search:

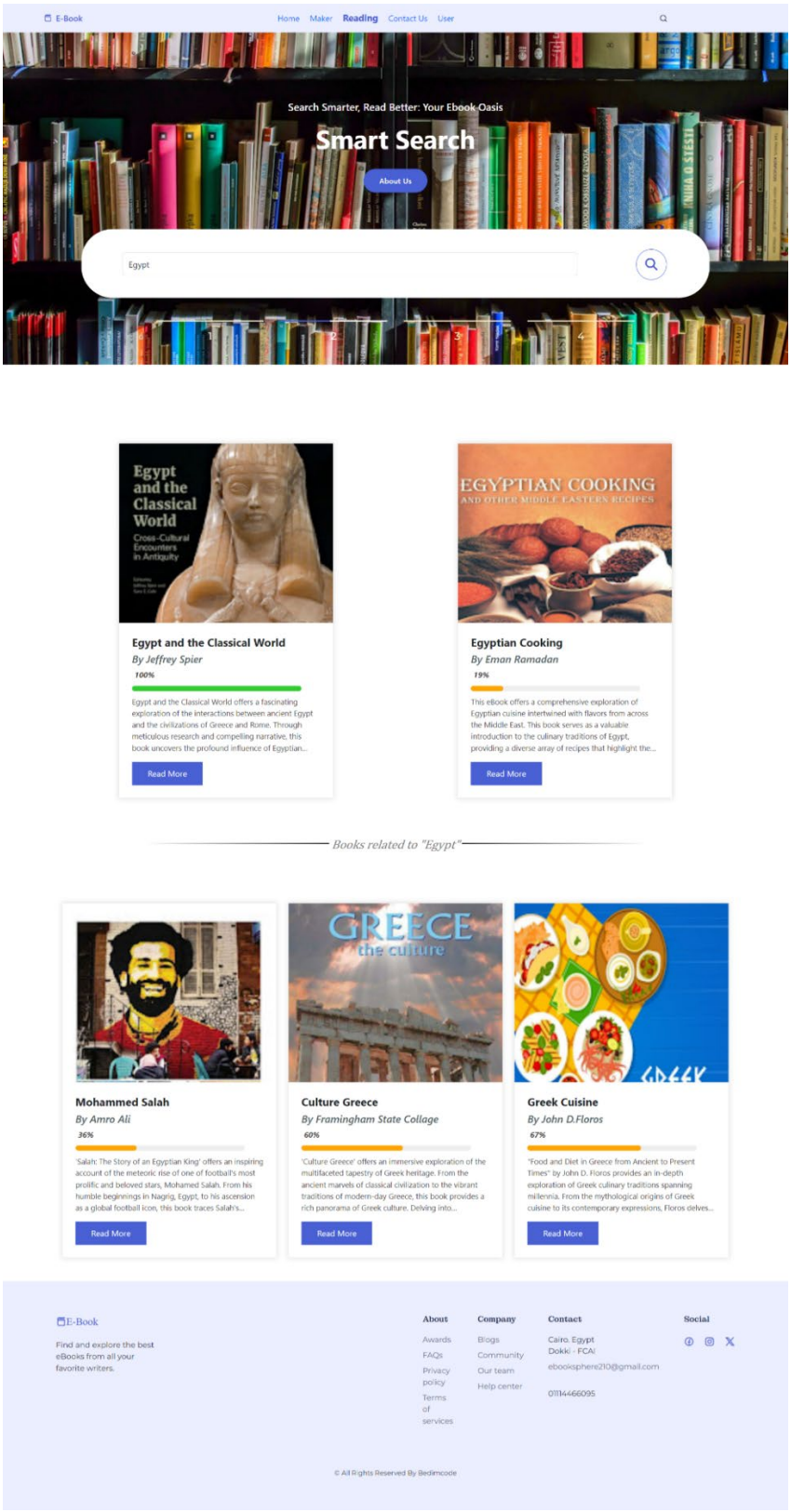

Figure 14 "GUI Design: Search"

4.6.4 eBook Description:

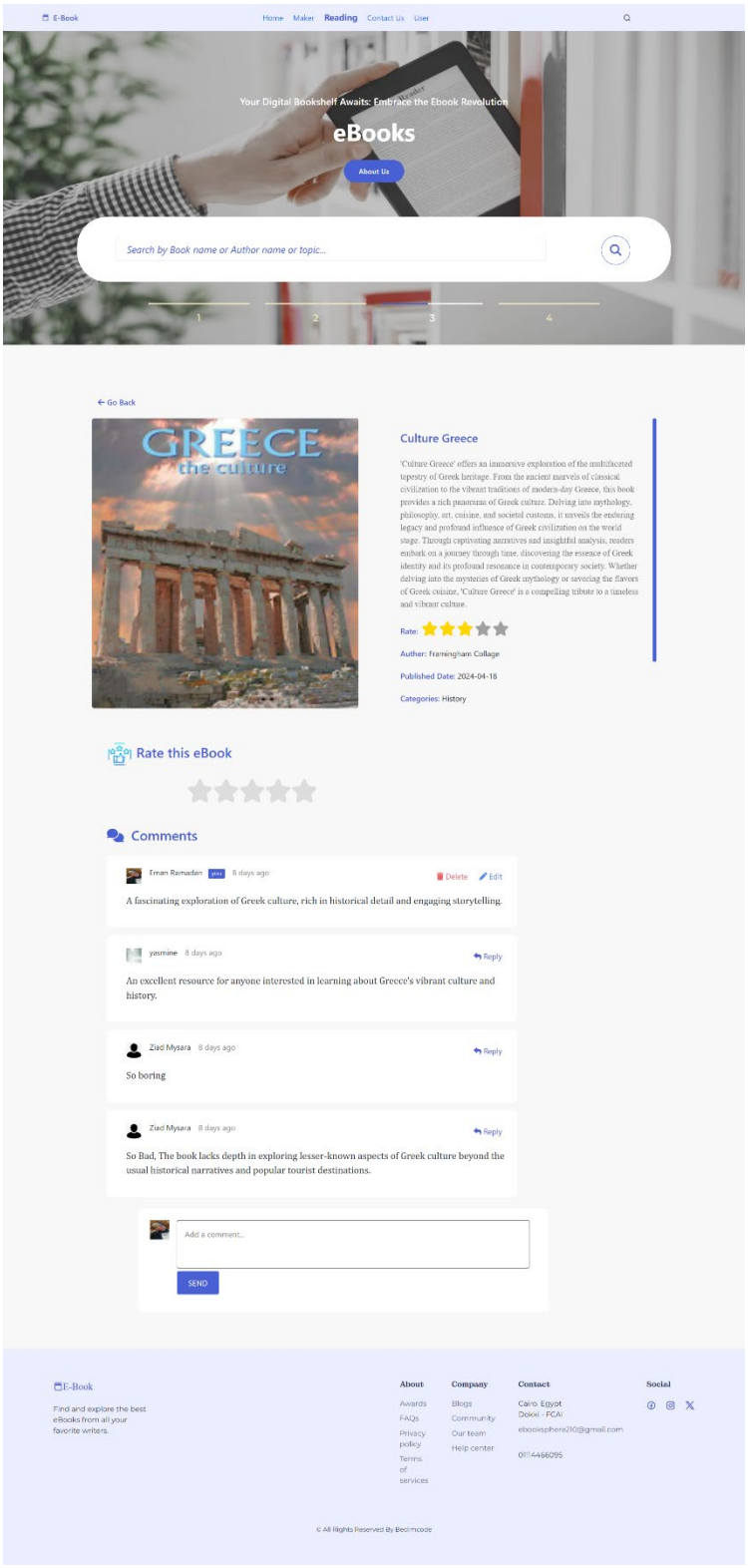

Figure 15 "GUI Design: eBook Description"

4.6.5 Templates:

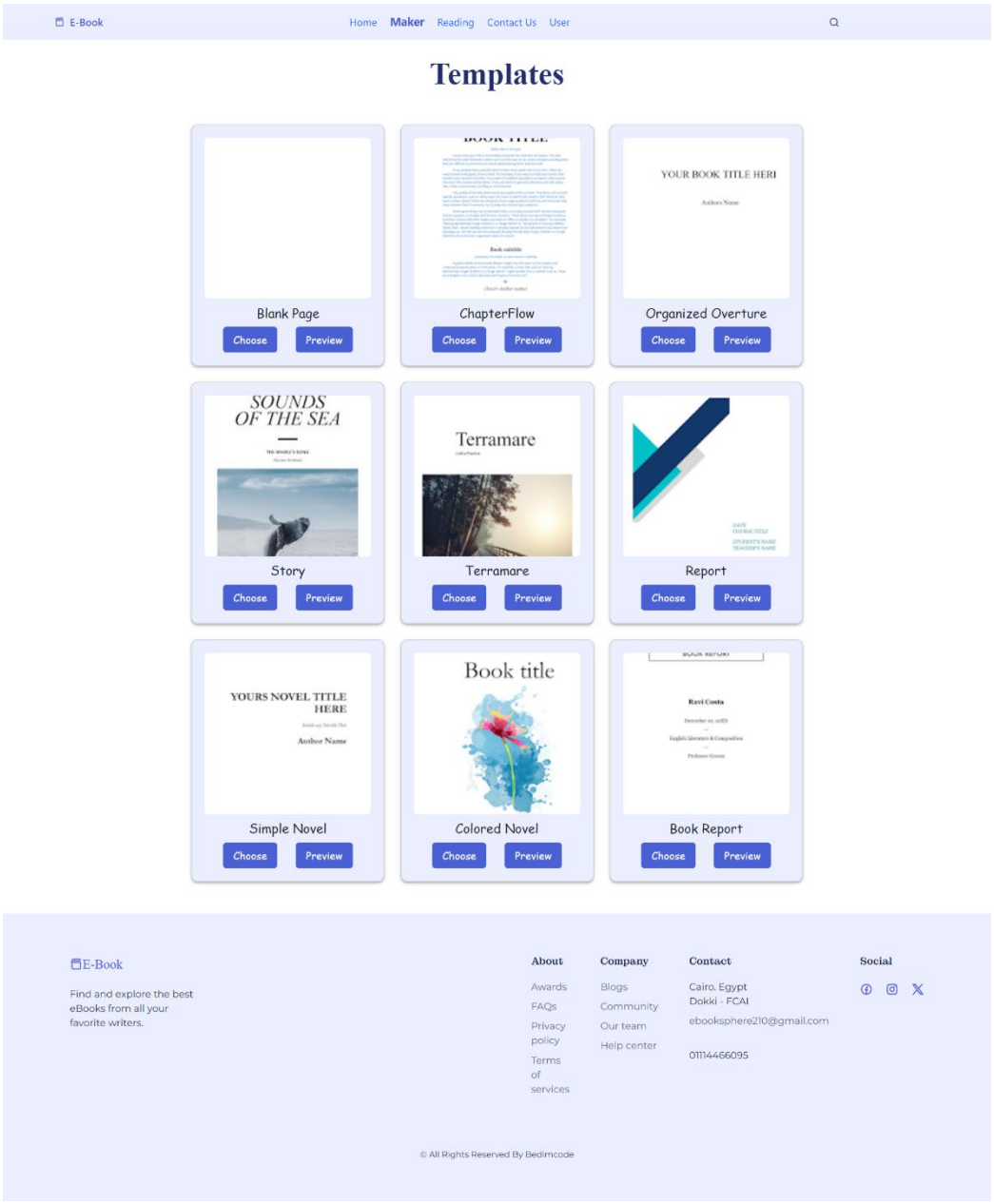

Figure 16 "GUI Design: Templates"

4.6.6 eBook Maker:

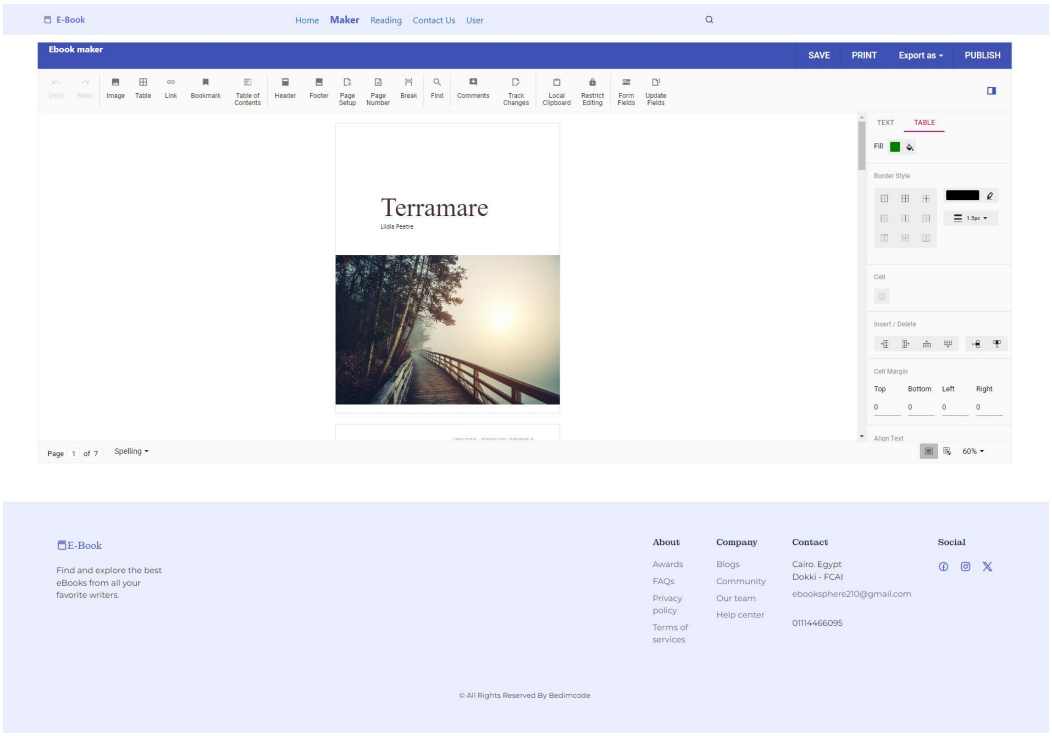

Figure 17 "GUI Design: eBook Maker"

4.6.7 Dashboard:

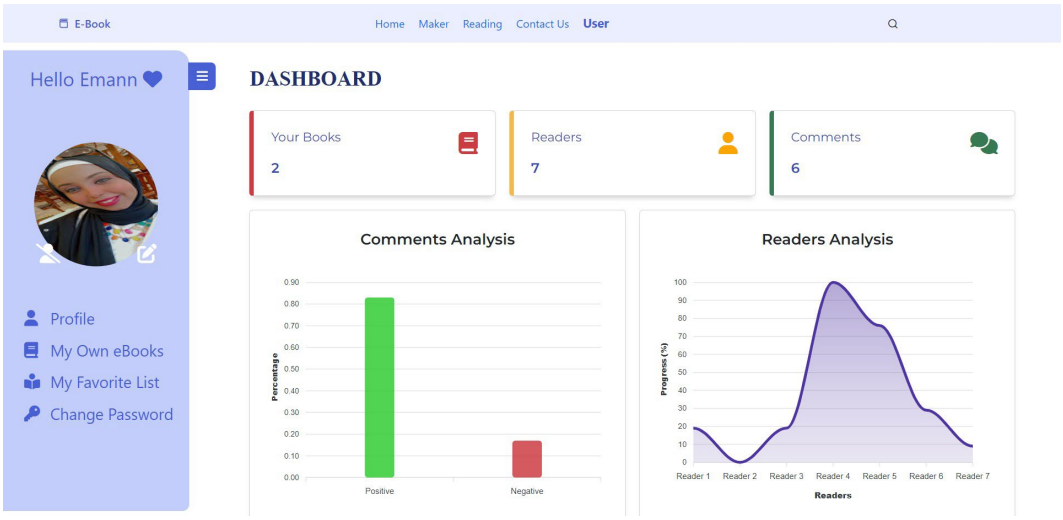

Figure 18 "GUI Design: Dashboard"

4.6.8 Admin Section:

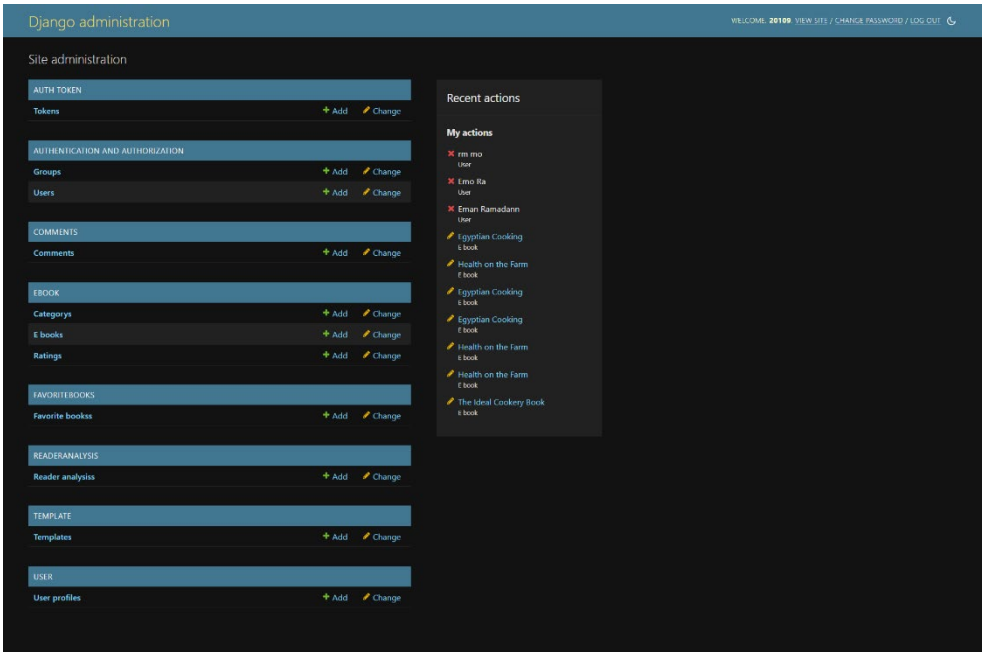

Figure 19 "GUI Design: Admin Section"

Chapter 5: Implementation and testing

5.1 Overview

5.1.1 Environment:

- **VSCode:** devolepment environment
- **Github:** for Version Control with backend
- **Postman:** for testing API's

5.1.2 Front-end:

- **HTML/CSS/JavaScript**: Core technologies for building the structure, styling, and interactivity of the website.
- **Angular**: A platform and framework for building single-page client applications using TypeScript.
- **Bootstrap**: A front-end framework for developing responsive and mobile-first websites.

5.1.3 Back-end:

- **Python**: The primary programming language used for back-end

development.

- **Django REST Framework**: A powerful and flexible toolkit for building Web APIs, allowing the back-end to communicate effectively with the front-end.
- **SQLite (db.sqlite3)**: A lightweight, disk-based database used for storing and managing data in development.

5.1.4 Machine learning:

- **Tesseract OCR**: An open-source Optical Character Recognition engine used for text extraction from documents.
- **SentenceTransformer**: A Python framework for sentence embeddings, which converts text into dense vector representations.
- **WordNet**: A lexical database for the English language that provides synonyms and antonyms for query expansion.
- **Google Generative AI (Gemini)**: A generative AI model used to enhance query expansion with additional related terms.
- **BERT (Bidirectional Encoder Representations from Transformers)**: A transformer-based machine learning technique for natural language processing pre-training, used for embedding text.
- **ElasticSearch**: Used as part of the search engine for indexing and retrieving document embeddings based on semantic similarity.

5.2 Challenges

During the development of the eBook Sphere project, several challenges were encountered:

- **Ineffective Search Results**: The initial implementation of the search functionality using ontology yielded poor results with an accuracy of less than 50%. The search required exact class names to return relevant results, making it inefficient and unreliable.

- **User Experience**: The need for exact match queries hindered the user experience, as users often did not know the exact terms or class names.

- **Performance Issues**: Handling large volumes of eBook data efficiently while providing fast and relevant search results posed significant technical challenges.

- **Limited Synonym Generation:** Utilizing WordNet alone for query expansion did not provide the desired level of accuracy, as the synonyms generated were often not comprehensive enough.

5.3 Solution

To address these challenges, the search engine was enhanced using a combination of BERT, Elasticsearch, and Google's generative AI (Gemini) alongside WordNet for better query expansion. This solution significantly improved the search functionality:

Search Engine Overview:

- The search engine is designed to efficiently index and search documents using advanced natural language processing (NLP) techniques and Elasticsearch.
- The implementation consists of two main phases: **indexing** and **query handling**.

Indexing Phase:

- **Text Extraction**: The engine uses Tesseract OCR and SentenceTransformer to extract text from documents.
- **Semantic Embedding**: The extracted text is semantically embedded using SentenceTransformer, creating embeddings that capture the meaning and context of the text.
- **Indexing**: These embeddings are then indexed in Elasticsearch, enabling fast and accurate document retrieval.

Query Handling Phase:

- **Query Expansion** User queries are expanded using both WordNet and Google's generative AI (Gemini) to enhance the search scope and ensure relevant results. This combination allows for a more comprehensive set of synonyms and related terms, improving accuracy.
- **Semantic Embedding of Queries**: The expanded queries are semantically embedded using the same SentenceTransformer model.
- **Matching**: The embedded queries are matched with the indexed document embeddings in Elasticsearch to find and return the most relevant results based on semantic similarity.

Outcome:

- This approach ensures efficient and accurate document retrieval beyond simple keyword matching.
- The use of BERT and Elasticsearch significantly improved the accuracy and relevance of the search results, with an F-measure of 70.77%.

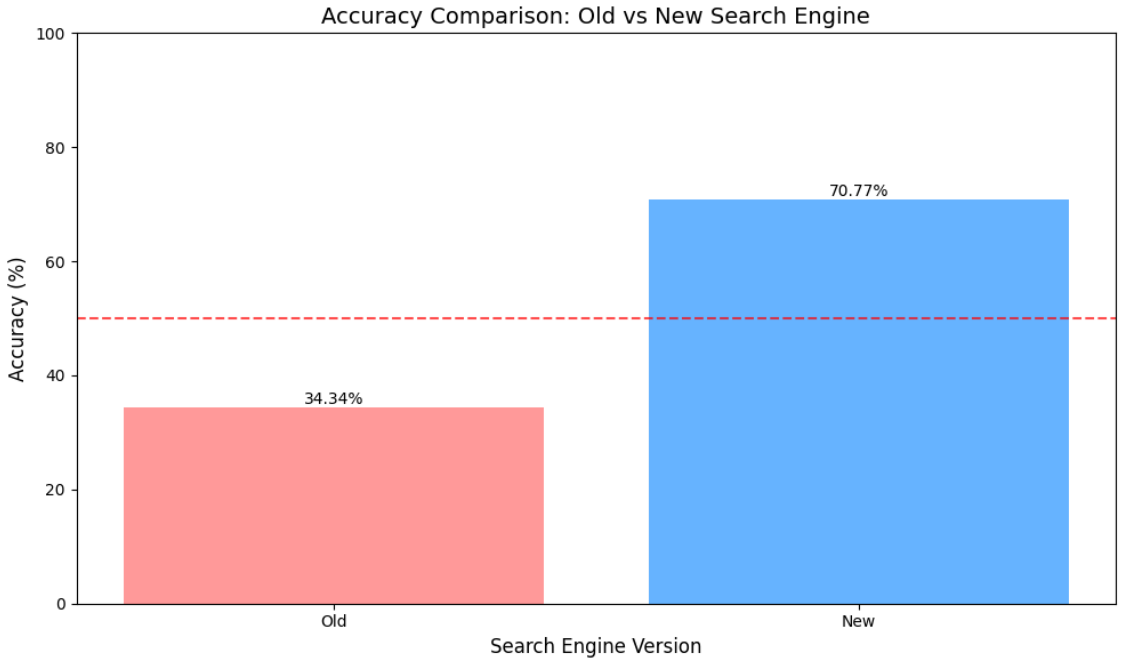

Figure 20 "Accuracy Comparison"

By leveraging advanced NLP techniques and the generative capabilities of Gemini alongside Elasticsearch, the new search engine provides a robust solution to the challenges faced in the initial implementation, ensuring a better user experience and more reliable search functionality.

5.4 ML Implementation and Testing

5.4.1 Search Engine

The Search Engine in this project is designed to efficiently index and search documents using a combination of advanced natural language processing (NLP) techniques and Elasticsearch. The implementation

consists of two main phases: indexing and query handling. The following sections detail each step of these phases.

- **Indexing Phase:**

1. **Document Processing**:

   - The documents are processed using various tools including Tesseract for Optical Character Recognition (OCR) and SentenceTransformer for text embeddings.

   - Pytesseract is configured to extract text from images and PDFs, converting them into a searchable format.

```
import pytesseract
pytesseract.pytesseract.tesseract_cmd = r'C:\\Program
Files\\Tesseract-OCR\\tesseract.exe'
```

2. **Text Embedding:**

   The "SentenceTransformer" model 'paraphrase-MiniLM-L6-v2' is used to generate embeddings for the text. This model helps in creating semantic representations of the text which are crucial for accurate search results.

```
from sentence_transformers import SentenceTransformer

class semanticSearch:
    def __init__(self, tesseract_path=None):
        self.model = SentenceTransformer('paraphrase-MiniLM-L6-v2')
        if tesseract_path:
            pytesseract.pytesseract.tesseract_cmd = tesseract_path
```

3. **Elasticsearch Configuration:**

   - Elasticsearch is set up with basic authentication and a specific index (ebook) where all processed documents are stored.

   - The "semanticSearch" class initializes the Elasticsearch client and sets up the connection parameters including the host, user credentials, and the index name.

```
from elasticsearch import Elasticsearch

class semanticSearch:
    def __init__(self, es_host='http://localhost:9200',
es_user='elastic', es_password='123456', index_name='ebook'):
        self.es = Elasticsearch([es_host], basic_auth=(es_user,
es_password), timeout=400)
        self.index_name = index_name
```

4. Indexing Documents:

- Each document is tokenized using BERT Tokenizer (Bidirectional Encoder Representations from Transformers).
- The tokenized text is then passed through the BERT model to obtain contextual embeddings.
- The embeddings are stored in Elasticsearch, making the documents searchable based on their semantic content rather than just keywords.

```
from transformers import BertTokenizer, BertModel

class semanticSearch:
    def index_documents(self, file_list, drive_service):
        for file in file_list:
            pdf_title = file['name']
            pdf_id = file['id']
            try:
                pdf_content = self.download_pdf_content(pdf_id,
drive_service)
                if pdf_content:
                    text_to_embed = f"{pdf_title}\n{pdf_content}"
                    embedded_text = self.embed_text(text_to_embed)
                    padded_text_vector =
self.pad_or_truncate(embedded_text, expected_dim=768)
                    embedded_text_list = padded_text_vector.tolist()

                    document_body = {
                        'filename': pdf_title,
                        'text': pdf_content,
```

```
                        'text_vector': embedded_text_list,
                        'fileId': pdf_id
                    }

                    self.es.index(index=self.index_name,
body=document_body)
                    print(f"Indexed document: {pdf_title}")
                else:
                    raise RuntimeError(f"Failed to download PDF
content for {pdf_title}")
            except Exception as e:
                raise RuntimeError(f"Error occurred while indexing
{pdf_title}: {e}")
```

- **Query Handling Phase:**

1. **Query Expansion:**

   - User queries are processed to extract keywords and relevant phrases.
   - To improve search accuracy and relevance, user queries are expanded using both WordNet and Google's generative AI (Gemini). This combination enhances the search scope by generating a comprehensive set of synonyms and related terms.

     The query expansion code starts by importing the necessary libraries:

```
import google.generativeai as genai
from nltk.corpus import wordnet as wn
```

The "expand_query_with_synonyms" function begins by initializing a set to store synonyms:

```
def expand_query_with_synonyms(self, query):
    synonyms = set()
    for synset in wn.synsets(query):
        synonyms.update(synset.lemma_names())
```

This part of the function uses WordNet to find synonyms for the query term. Each synonym set (synset) related to the query is iterated over, and the synonyms are added to the set.

Next, we configure the Google Generative AI API:

```
    genai.configure(api_key="YOUR_API_KEY")

    generation_config = {
        "temperature": 1,
        "top_p": 0.95,
        "top_k": 64,
        "max_output_tokens": 8192,
        "response_mime_type": "text/plain",
    }

    model = genai.GenerativeModel(
        model_name="gemini-1.5-flash",
        generation_config=generation_config,
    )
```

The API configuration includes parameters such as “temperature”, “top_p”, “top_k”, and “max_output_tokens”, which influence the behavior of the model. A generative model instance is created using these settings.

The function then initiates a chat session and sends a message to the model to generate additional related words:

```
    chat_session = model.start_chat(history=[])
    response = chat_session.send_message(f"What are some words
related to {
        query} and the response should be a list?")
```

Finally, the function combines the generated synonyms with those from WordNet and the original query, forming an expanded query:

```
    result = response.text + " ".join(list(synonyms)) + " " + query
    return result
```

2. Query Embedding:

- Similar to document processing, the query is embedded using the `SentenceTransformer` model to obtain a semantic representation.
- The method `embed_text` converts the query into an embedding vector.

```
class semanticSearch:
    def embed_text(self, text):
        return self.model.encode(text, convert_to_tensor=True)
```

3. Search Execution:

- The search is performed by matching the query embeddings with the document embeddings stored in Elasticsearch.
- The Elasticsearch client handles the retrieval of documents that have the highest semantic similarity to the query.

```
def semantic_search(self, query):

    combined_query = self.expand_query_with_synonyms(query)
    #print(combined_query)
    query_vector = self.embed_text(combined_query)
    padded_text_vector = self.pad_or_truncate(query_vector,
expected_dim=768)
    query_vector = padded_text_vector.tolist()

    es_query = {
        "query": {
            "script_score": {
                "query": {"match_all": {}},
                "script": {
                    "source": "cosineSimilarity(params.query_vector,
'text_vector') + 1.0",
                    "params": {"query_vector": query_vector}
                }
            }
        }
    }
```

```
    search_results = self.es.search(index=self.index_name,
body=es_query)
    unique_hits = []
    encountered_texts = set()

    for hit in search_results['hits']['hits']:
        if '_source' in hit and 'fileId' in hit['_source']:
            text_title = hit['_source']['fileId']
            if text_title not in encountered_texts:
                #print(f"Matched document:
{hit['_source']['filename']} with score: {hit['_score']}")
                unique_hits.append(hit)
                encountered_texts.add(text_title)

    return unique_hits
```

4. **Result Handling:**

- The search results are formatted and returned to the user, ensuring that the most relevant documents are presented based on the semantic match rather than just keyword occurrence.

```
class RelatedEBookAPI(APIView):
    def get(self, request):
        query = request.GET.get('query')
        if query:
            try:
                results =
semantic_search_instance.search_eBook(query)
                eBooks_data = []

                for hit in results:
                    # Filter eBook instances by filename

                    eBooks = eBook.objects.filter(
                        content=hit['_source']['fileId'])
                    print(eBooks)
                    if eBooks:  # Check if any eBooks are found
                        serializer = eBookSerializer(eBooks,
many=True)
                        serialized_data_with_score = [
                            {"score": hit['_score'], "eBook":
eBook_data} for eBook_data in serializer.data]
```

```
eBooks_data.extend(serialized_data_with_score)

                sorted_eBooks = sorted(
                    eBooks_data, key=lambda x: x['score'],
reverse=True)
                sorted_eBooks_without_score = [
                    eBook_data['eBook'] for eBook_data in
sorted_eBooks]
                serialized_ebooks = []
                for ebook in sorted_eBooks_without_score:
                    ebook = eBook.objects.get(id=ebook['id'])
                    author_instance = ebook.author
                    categories = ebook.categories.all()
                    serialized_ebook = eBookSerializer(ebook).data
                    serialized_ebook['categories'] =
CategorySerializer(
                        categories, many=True).data
                    serialized_ebook['author'] = RegisterSerializer(
                        author_instance).data
                    serialized_ebooks.append(serialized_ebook)
                return Response(serialized_ebooks,
status=status.HTTP_200_OK)

            except RuntimeError as e:
                return Response({"status": "failed", "message":
str(e)}, status=status.HTTP_400_BAD_REQUEST)
        else:
            return Response({"error": "No search query provided"},
status=status.HTTP_400_BAD_REQUEST)
```

- **Summary**

  The search engine in the eBook Sphere project leverages Tesseract OCR and SentenceTransformer to extract and semantically embed text from documents, which are then indexed in Elasticsearch. User queries are processed by expanding them with synonyms using both WordNet and Google's generative AI (Gemini) and embedding them. These embeddings are then matched with indexed document embeddings to find and return the most relevant results based on semantic similarity. This approach ensures efficient and accurate document retrieval beyond simple keyword matching, significantly

improving search accuracy and relevance. The search engine achieved an F-measure of 70.77%, reflecting its high level of accuracy.

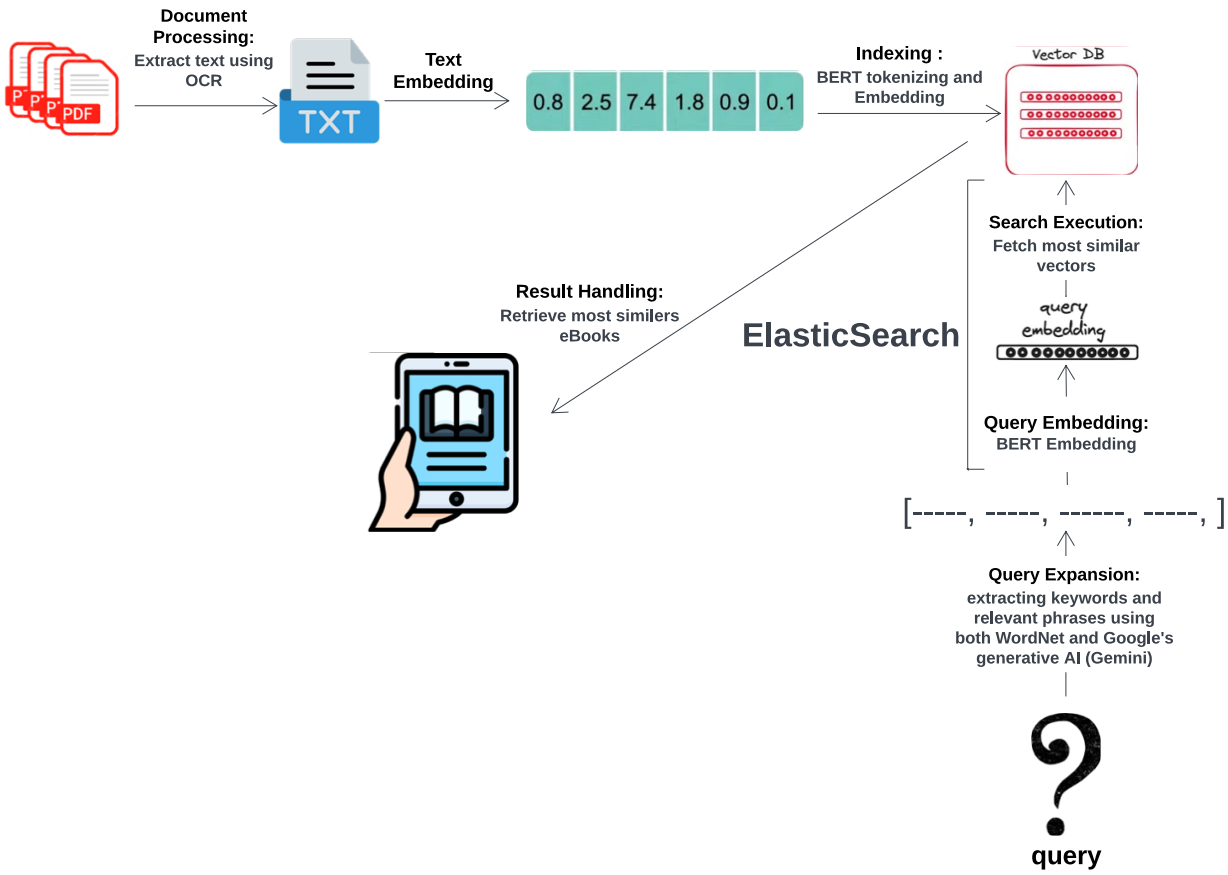

Figure 21 "Semantic Search Opreation"

- **Here's the flowchart representing the implementation of the search engine:**
    - **Document Processing:** Text extraction from images and PDFs using Tesseract.
    - **Text Embedding:** Generating semantic embeddings using SentenceTransformer.
    - **Indexing Documents:** Tokenizing documents with BERT and storing embeddings in Elasticsearch.
    - **Query Expansion:** Extracting keywords and expanding them with synonyms using both WordNet and Google's generative AI (Gemini).

- **Query Embedding:** Converting the user query into an embedding vector .
- **Search Execution:** Matching query embeddings with document embeddings in Elasticsearch.
- **Result Handling:** Formatting and presenting the search results to the user.

- **Semantic Search Testing and Evaluation:**
  - The semantic search functionality, which utilizes BERT embeddings and WordNet synsets, was thoroughly tested to evaluate its accuracy and effectiveness.
  - The performance of the semantic search was measured using the F-measure, a metric that combines precision and recall to provide a single performance score.
  - The semantic search achieved an F-measure of 70.77%, indicating a high level of accuracy in retrieving relevant eBook results based on user queries.

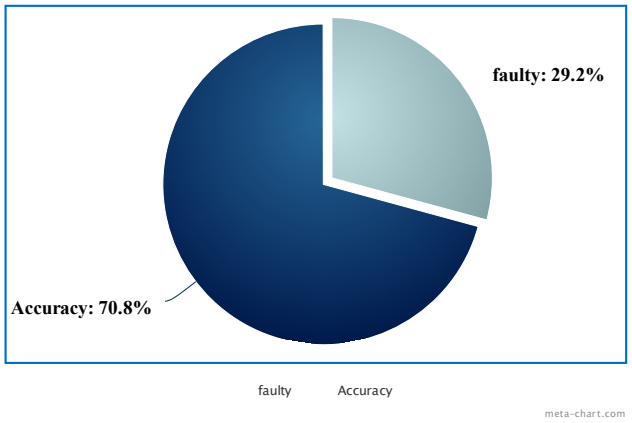

Figure 22 "Accuracy pie chart"

This testing result demonstrates that the implemented search algorithm effectively understands and processes user queries, returning accurate and contextually relevant results. The high F-measure score reflects the robustness of the search functionality and its ability to meet user expectations for finding eBooks on the platform.

5.4.2 Comment Analysis

In this section, we discuss the implementation and testing of the machine learning model used for comment analysis. The primary objective is to analyze user comments on specific eBooks and classify them as positive or negative. The following is an overview of the machine learning approach and the results obtained.

1. **Machine Learning Model**

   For the sentiment analysis of user comments, we employed a machine learning model based on the VADER (Valence Aware Dictionary and sEntiment Reasoner) sentiment analysis tool from the NLTK library. VADER is specifically designed to analyze text sentiment in social media contexts, making it suitable for our application.

2. **Implementation**

   The core implementation of the comment analysis involves the use of the VADER sentiment analyzer. The model processes each comment and assigns a sentiment score, which is then classified as positive, negative, or neutral based on predefined thresholds.

3. **Sentiment Analysis Endpoint**

   The sentiment analysis is implemented in the `CommentAnalysis` function. This function takes a `book_id` parameter, retrieves the associated comments, and analyzes their sentiment using the VADER analyzer.

3.1. Setting Up the VADER Analyzer

First, we download the VADER lexicon and initialize the sentiment analyzer:

```
from nltk import download
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Download the VADER lexicon
download('vader_lexicon')

# Initialize the VADER sentiment analyzer
sia = SentimentIntensityAnalyzer()
```

This code ensures that the VADER lexicon, which contains the sentiment scores for various words and phrases, is available for use. The "SentimentIntensityAnalyzer" is then initialized to perform the sentiment analysis.

3.2.Defining the Sentiment Classification Function

Next, we define a function to classify the sentiment scores returned by the VADER analyzer:

```
# Function to classify comments
def classify_sentiment(score):
    if score['compound'] >= 0.05:
        return 'positive'
    elif score['compound'] <= -0.05:
        return 'negative'
    else:
        return 'neutral'
```

The "classify_sentiment" function takes the sentiment score as input and classifies it into three categories:

- **Positive**: If the compound score is greater than or equal to 0.05.
- **Negative**: If the compound score is less than or equal to -0.05.
- **Neutral**: If the compound score is between -0.05 and 0.05.

3.3.Analyzing the Comments

The comments are then analyzed using the VADER sentiment analyzer, and their sentiments are classified:

```
positive_count = 0
negative_count = 0

for comment in comments:
    sentiment = sia.polarity_scores(comment.content)
    sentiment_class = classify_sentiment(sentiment)
    if sentiment_class == 'positive':
        positive_count += 1
    elif sentiment_class == 'negative':
        negative_count += 1
```

In this loop, each comment's content is analyzed to determine its sentiment scores. The "sia.polarity_scores" function returns a dictionary of sentiment scores, including the compound score. The "classify_sentiment" function is

then used to categorize the sentiment as positive, negative, or neutral. The counts of positive and negative comments are updated accordingly.

4. **Comment Analysis Testing and Evaluation**

To evaluate the performance of our sentiment analysis model, we conducted extensive testing using a labeled dataset of user comments. The model's performance was assessed using the F1 score, a metric that balances precision and recall. Our model achieved an F1 score of 80.98%, indicating a high level of accuracy in classifying comments as positive or negative.

The high F1 score of 80.98% demonstrates the effectiveness of our sentiment analysis model in accurately interpreting user sentiments from their comments. This level of accuracy ensures that the insights derived from the analysis are reliable and can be used to make informed decisions regarding eBook content and user preferences.

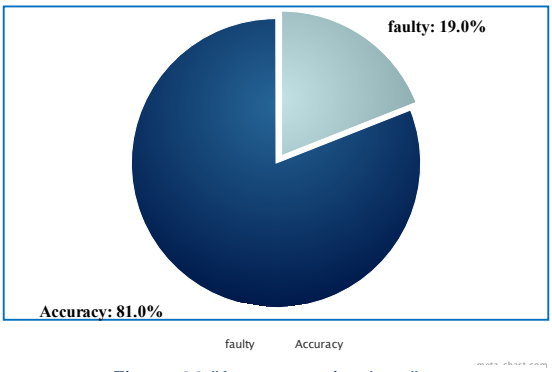

Figure 23 "Accuracy pie chart"

5.5 Other Implementation and Testing

5.5.1 Test Cases

Sign Up Test:

- **Objective:** Verify that users can register successfully.
- **Steps:** Submit the registration form with valid data and check for successful account creation.
- **Expected Result:** User is redirected to the last open page.

Test Cases:

1. **Test 1:**
   - **Input:** Empty
   - **Output:** Show messages "First Name is required", "Last Name is required", "UserName is required", "Email is required", "Password is required".
2. **Test 2:**

- **Input:**
  - **First Name:** Ziad
  - **Last Name:** Mysara
  - **Username:** ZiadMysara
  - **Email:** Ziad.Mysara
  - **Password:** Ziad_2024
- **Output:** Invalid email format ,error message "Email is invalid".

3. **Test 3:**
   - **Input:**
     - **First Name:** Ziad
     - **Last Name:** Mysara
     - **Username:** ZiadMysara
     - **Email:** ZiadMysara@gmail.com
     - **Password:** 123456
   - **Output:** Invalid password format error message "Password must contain characters and numbers and special characters, Password must be at least 8 characters long".
4. **Test 4:**
   - **Input:**
     - **First Name:** Ziad
     - **Last Name:** Mysara
     - **Username:** already exists name
     - **Email:** ZiadMysara@gmail.com
     - **Password:** Ziad_2024
   - **Output:** Show message "Username already exists".
5. **Test 5:**
   - **Input:**
     - **First Name:** Ziad
     - **Last Name:** Mysara
     - **Username:** ZiadMysara
     - **Email:** ZiadMysara@gmail.com
     - **Password:** Ziad_2024
   - **Output:** Account is successfully created, and the user is redirected to the last open page.

Sign In Test:

- **Objective:** Verify that users can log in successfully.
- **Steps:** Submit the login form with valid credentials and check for successful login.
- **Expected Result:** User is redirected to the last open page.

Test Cases:

1. **Test 1:**
   - **Input:** Empty

- **Output:** show messages "Username is required", "Password is required"

2. **Test 2:**
   - **Input:**
     - **Username:** ZiadMysara
     - **Password:** IncorrectPassword
   - **Output:** Invalid credentials error message; "Username or password is incorrect".
3. **Test 3:**
   - **Input:**
     - **Username:** ZiadMysara
     - **Password:** CorrectPassword
   - **Output:** successful login, User is redirected to the last open page.

Forget Password Test:

- **Objective:** Verify that users reset Password successfully.
- **Steps:** Submit the ForgetPasswordByEmail form with valid credentials.
- **Expected Result:** Password reset email sent.

Test Cases:

1. **Test 1:**
   - **Input:** Empty
   - **Output:** show messages "Email is required"
2. **Test 2:**
   - **Input:**
     - **Email:** ZiadMysara@gmail.com
   - **Output:** show messages "Password reset email sent".

Reset Password Test:

- **Objective:** Verify that users can rest password successfully.
- **Steps:** Submit the reset password form with valid data and within 5 minutes.
- **Expected Result:** Show message "Password has been reset".

Test Cases:

1. **Test 1:**
   - **Input:** Empty
   - **Output:** Show messages "Password is required"
2. **Test 2:**
   - **Input:**

- **New Password:** Ziad_2024
- **Conform Password:** Ziad_2024_2
- **Time:** less than 5 minutes.

- **Output:** "Passwords must match" error message.

3. **Test 3:**
   - **Input:**
     - **New Password:** Ziad_2024
     - **Conform Password:** Ziad_2024
     - **Time:** more than 5 minutes.
   - **Output:** "Link is invalid or expired. Please try again." error message.
4. **Test 4:**
   - **Input:**
     - **New Password:** Ziad_2024
     - **Conform Password:** Ziad_2024
     - **Time:** less than 5 minutes.
   - **Output:** Show message "Password has been reset.".

Change Profile info Test:

- **Objective:** Verify that users Change Profile info successfully.
- **Steps:** Submit the Change Profile info form with valid data and check for successful account creation.
- **Expected Result:** Show message "Profile updated successfully!"

Test Cases:

5. **Test 1:**
   - **Input:** Empty form
   - **Output:** Show messages "First Name is required", "Last Name is required", "UserName is required", "Email is required".
6. **Test 2:**
   - **Input:**
     - **First Name:** Ziad
     - **Last Name:** Mysara
     - **Username:** ZiadMysara
     - **Email:** Ziad.Mysara
   - **Output:** Show message "Email is invalid".
7. **Test 3:**
   - **Input:**
     - **First Name:** Ziad
     - **Last Name:** Mysara25
     - **Username:** ZiadMysara
     - **Email:** ZiadMysara@gmail.com
   - **Output:** Show message "Last Name must be only characters".
8. **Test 4:**

- **Input:**
  - **First Name:** Ziad
  - **Last Name:** Mysara
  - **Username:** ZiadMysara
  - **Email:** ZiadMysara@gmail.com
- **Output:** Account is successfully updated, Show message "Profile updated successfully!".

Change Password Test:

- **Objective:** Verify that users can change password successfully.
- **Steps:** Submit the change password form with valid data and check for successful account creation.
- **Expected Result:** Show message "Password changed successfully!".

Test Cases:

9. **Test 1:**
   - **Input:** Empty
   - **Output:** Show messages "Username is required", "Password is required"
10. **Test 2:**
    - **Input:**
      - **Username:** ZiadMysara
      - **Old Password:** wrong old password
      - **New Password:** Ziad_2024
      - **Conform Password:** Ziad_2024
    - **Output:** "Old password is incorrect" error message.
11. **Test 3:**
    - **Input:**
      - **Username:** ZiadMysara
      - **Old Password:** true old password
      - **New Password:** Ziad_2024
      - **Conform Password:** Ziad_2024_2
    - **Output:** "Passwords must match" error message.
12. **Test 4:**
    - **Input:**
      - **Username:** ZiadMysara
      - **Old Password:** true old password
      - **New Password:** Ziad_2024
      - **Conform Password:** Ziad_2024
    - **Output:** Show message "Password Changed successfully!".

Spelling Test:

- **Objective:** Verify that users write sentence correctly.

- **Steps:** Write sentence.
- **Expected Result:** No red underlines.

Test Cases:

1. **Test 1:**
    - **Input:** Sentence with a misspelled word.
    - **Output:** Red underline appears under the misspelled word.
2. **Test 2:**
    - **Input:** Correctly spelled sentence.
    - **Output:** No red underlines.

Save eBook Test:

- **Objective:** Verify that users can save their eBook.
- **Steps:** Choose save directory and write Title name.
- **Expected Result:** the eBook saved.

Test Cases:

1. **Test 1:**
    - **Input:** No directory selected.
    - **Output:** show message "No directory selected".
2. **Test 2:**
    - **Input:** Choose invalid save directory and make Title name Empty.
    - **Output:** show message "Please enter ebook title".
3. **Test 3:**
    - **Input:** Choose invalid save directory and write Title name.
    - **Output:** the eBook saved and show message "Document saved successfully".

eBook Export Test:

- **Objective:** Verify that users can export their eBooks.
- **Steps:** Select an eBook and choose the export option.
- **Expected Result:** eBook is exported in the selected format.

Test Cases:

1. **Test 1:**
    - **Input:** Select "Export as PDF." and title wasn't entered before.
    - **Output:** Show message "Enter your eBook Title" .
2. **Test 2:**
    - **Input:** Select "Export as ePub." and title was entered before.
    - **Output:** eBook is exported as an ePub file.

Publish eBook Test:

- **Objective:** Verify that users can publish a new eBook successfully.
- **Steps:** Submit the eBook publish form with valid data.
- **Expected Result:** eBook is published and listed on the user's profile.

Test Cases:

1. **Test 1:**
   - **Input:**
     - **Title:** Empty
     - **Description:** Sample description
     - **Select Categories:** Science
   - **Output:** show message "Please enter ebook title".
2. **Test 2:**
   - **Input:**
     - **Title:** Sample eBook
     - **Description:** Empty
     - **Select Categories:** Science
   - **Output:** show message "Please enter description".
3. **Test 3:**
   - **Input:**
     - **Title:** Sample eBook
     - **Description:** Sample description
     - **Select Categories:** Empty
   - **Output:** show message "Please select at least one category".
4. **Test 4:**
   - **Input:**
     - **Title:** Sample eBook
     - **Description:** Sample description
     - **Select Categories:** Science
   - **Output:** eBook is published and listed on the user's profile and show message "Document published successfully and it will be reviewed by our team".

Add Comment Test:

- **Objective:** Verify that users can add Comment successfully.
- **Steps:** write massage on comment box and click send.
- **Expected Result:** Comment added and displayed correctly.

Test Cases:

1. **Test 1:**
   - **Input:** Empty
   - **Output:** Show message "Message is required".

2. **Test 2:**
    - **Input: “z”**.
    - **Output:** Show message “Message must be at least 4 characters long”.
3. **Test 2:**
    - **Input:** “fascinating” and click send.
    - **Output:** Comment added and displayed correctly.

Contact Us Test:

- **Objective:** Verify that users can contact the admin successfully.
- **Steps:** Submit the contact form with a valid message.
- **Expected Result:** Message is sent successfully, and a confirmation is shown.

Test Cases:

1. **Test 1:**
    - **Input:** Empty form
    - **Output:** "Send Message" button is disabled until the form is filled completely and Show messages “Full Name is required”, “Email is required”, “Subject is required”, “Message is required”.
2. **Test 2:**
    - **Input:**
        - **Full Name:** Ziad Mysara
        - **Email Address:** ziad.mysara
        - **Subject:** congratulations
        - **Message:** this is great website.
    - **Output:** Show message “Email is invalid”.
3. **Test 3:**
    - **Input:**
        - **Full Name:** Ziad Mysara2551
        - **Email Address:** ziadmysara@gmail.com
        - **Subject:** congratulations
        - **Message:** this is great website
    - **Output:** Show message “Full Name must be only characters”.
4. **Test 4:**
    - **Input:**
        - **Full Name:** Ziad Mysara
        - **Email Address:** ziadmysara@gmail.com
        - **Subject:** congratulations
        - **Message:** this is great website.
    - **Output:** Show message “Massage sent successfully!”.

Search Test:

- **Objective:** Verify that the search functionality returns relevant results based on user queries.
- **Steps:** Enter different search queries and evaluate the relevance and accuracy of the search results.
- **Expected Result:** The search results are relevant to the entered query and displayed correctly.

Test Cases:

1. **Test 1:**
    - **Input:** Egypt
    - **Output:** The search results include eBooks with titles or descriptions containing "Egypt" and result that related to "Egypt"
2. **Test 2:**
    - **Input:** Cook
    - **Output:** The search results include eBooks with titles or descriptions containing "Cook" and result that related to "Cook"

5.5.2 Applying the Test Cases

Sign Up Test:

1. **Test 1:**

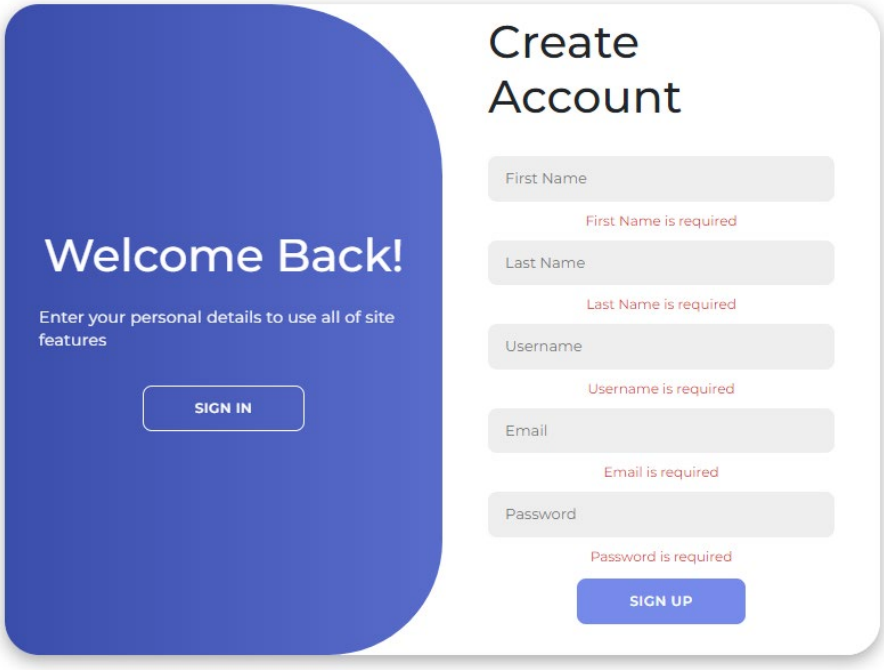

Figure 24 "Sign Up Test1"

2. **Test 2:**

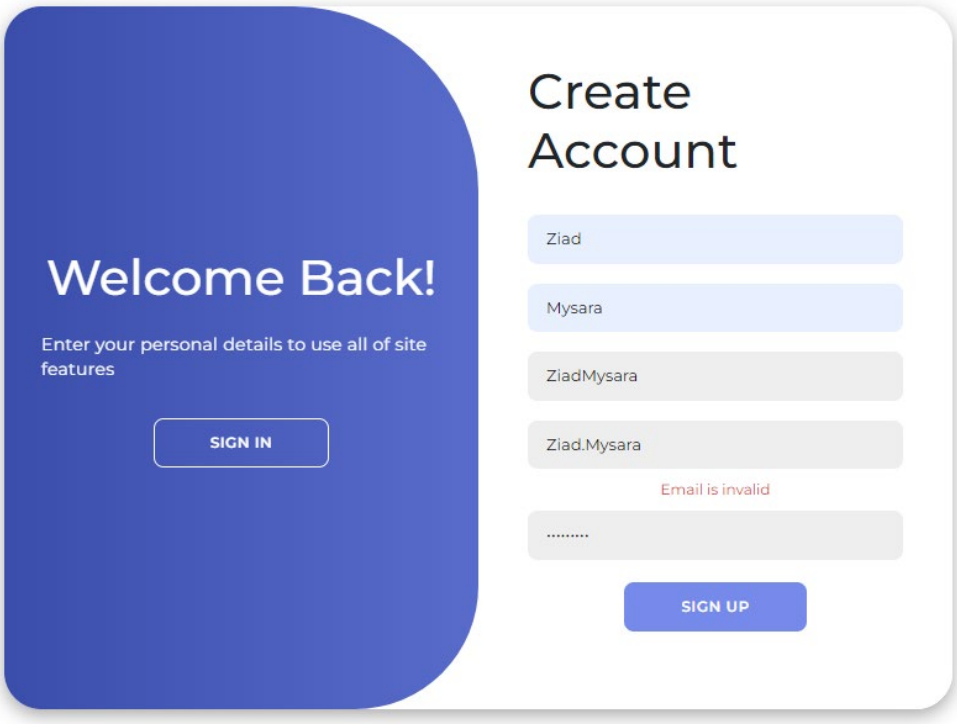

Figure 25 "Sign Up Test2"

3. **Test 3:**

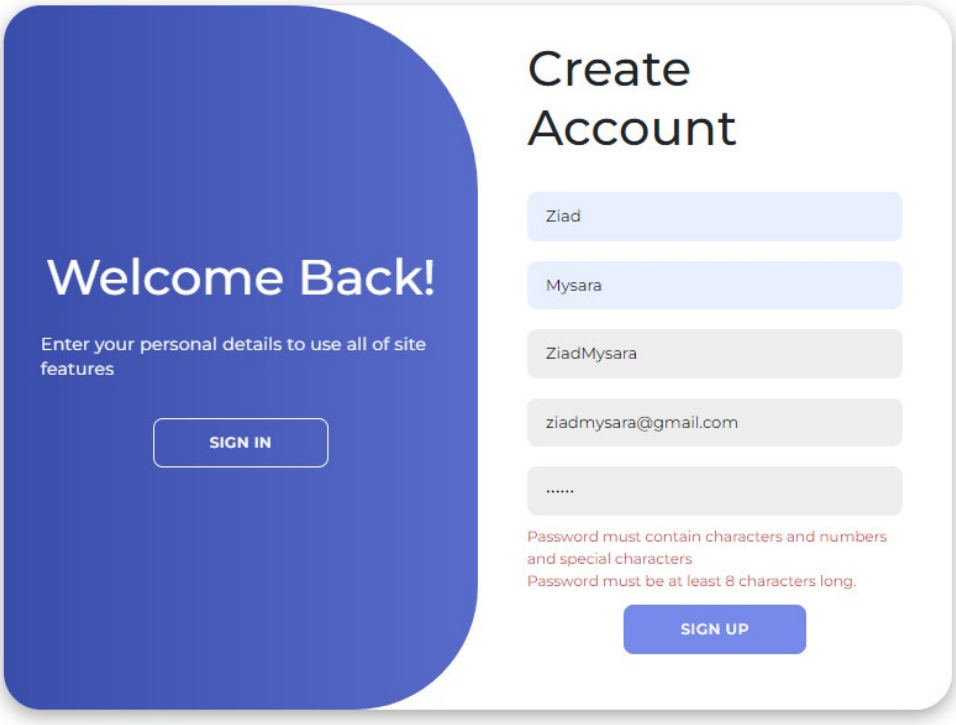

Figure 26 "Sign Up Test3"

4. **Test 4:**

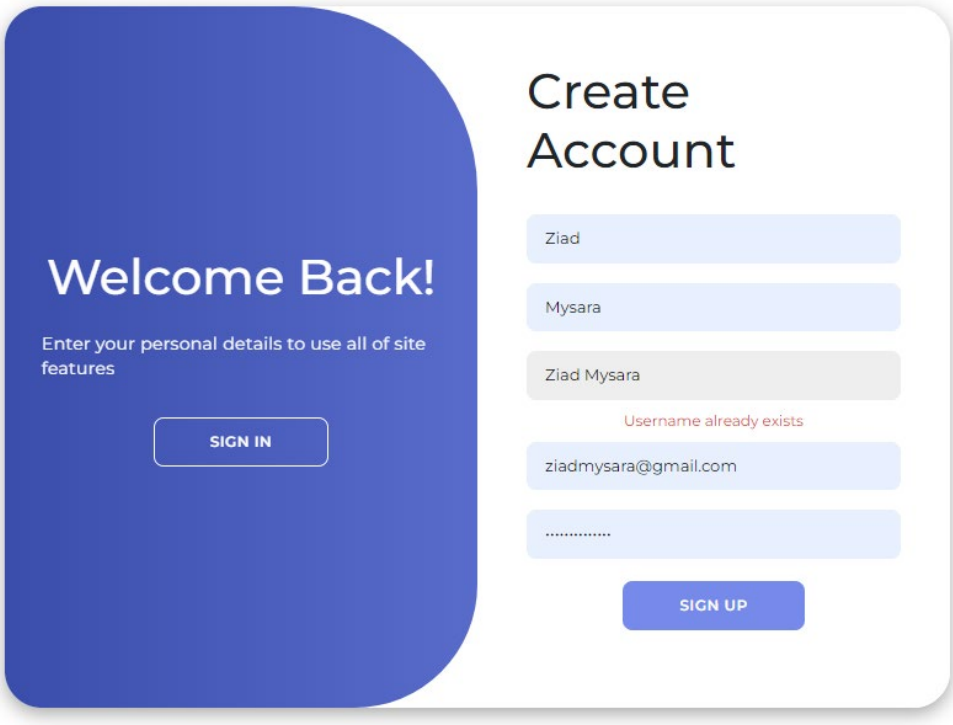

Figure 27 "Sign Up Test4"

5. **Test 5:**

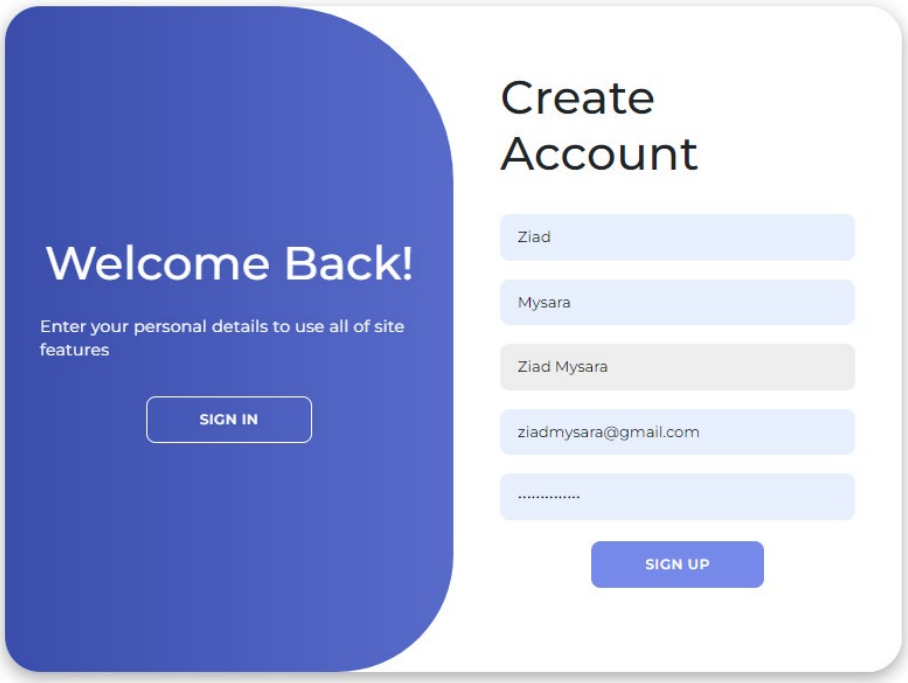

Figure 28 "Sign Up Test5"

Sign In Test:

1. **Test 1:**

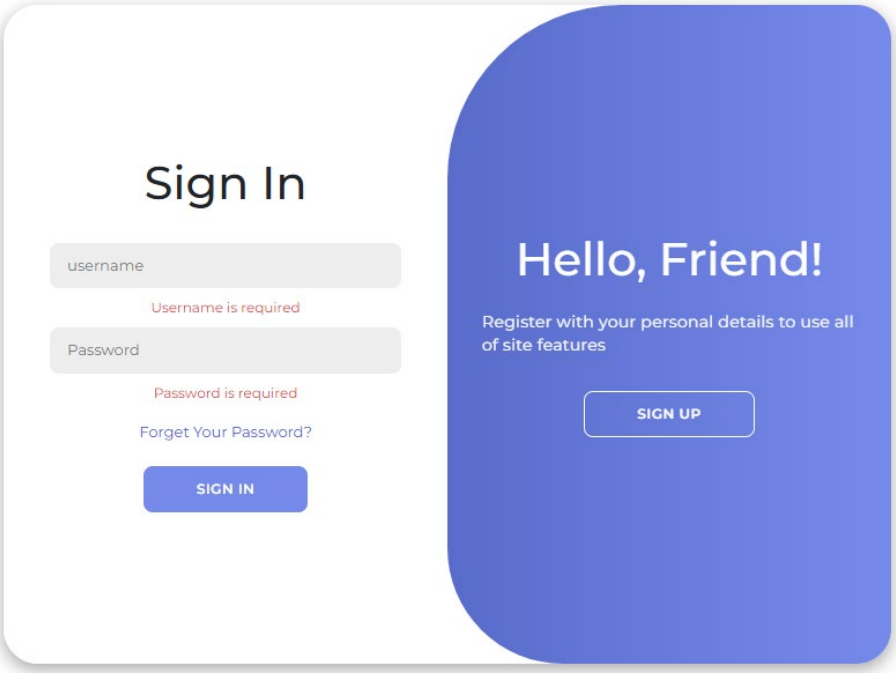

Figure 29 "Sign In Test1"

2. **Test 2:**

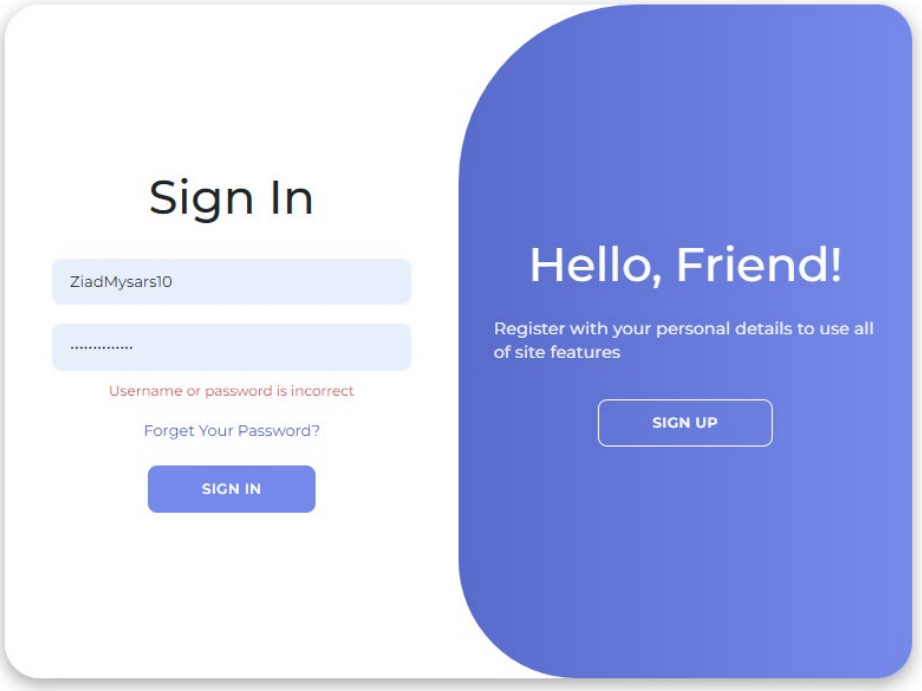

Figure 30 "Sign In Test2"

3. **Test 3:**

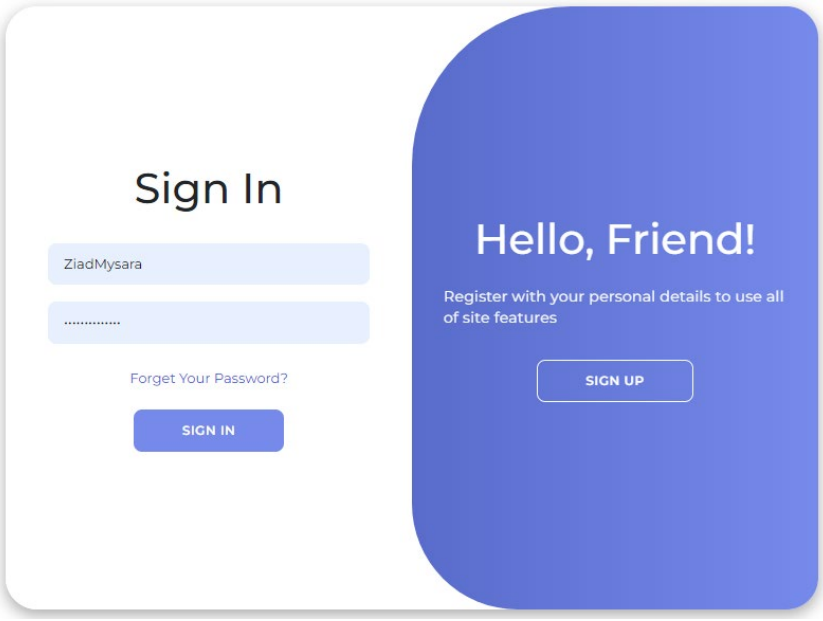

Figure 31 "Sign In Test3"

Forget Password Test:

1. **Test 1:**

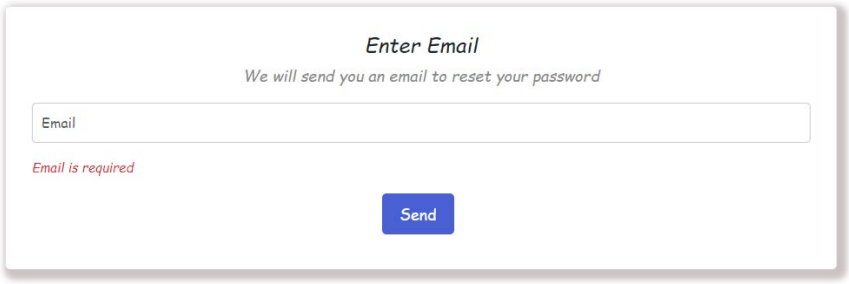

Figure 32 "Forget Password Test1"

2. **Test 2:**

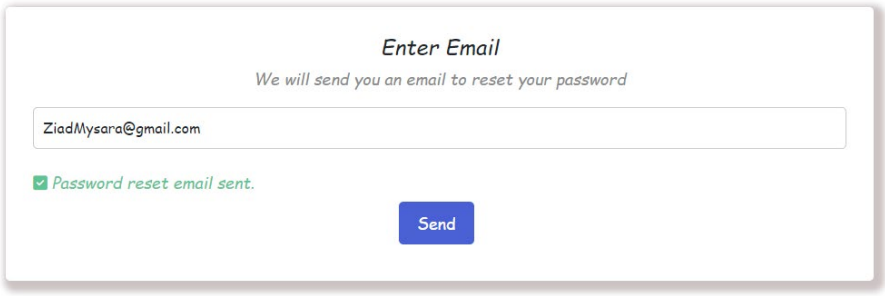

Figure 33 "Forget Password Test2"

Reset Password Test:

1. **Test 1:**

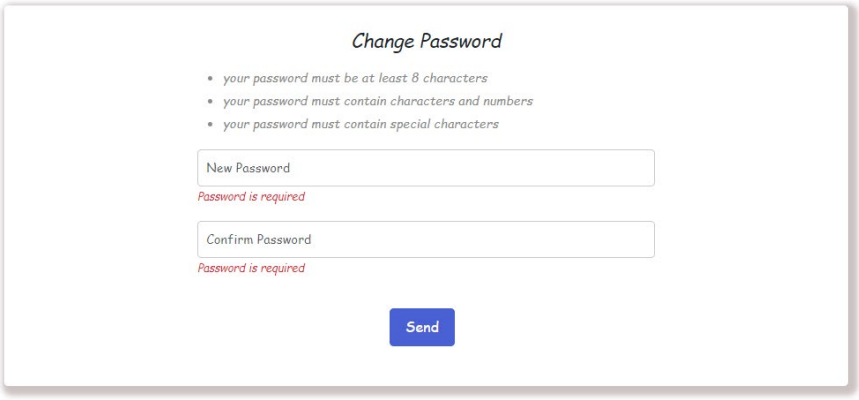

Figure 34 "Reset Password Test1"

2. **Test 2:**

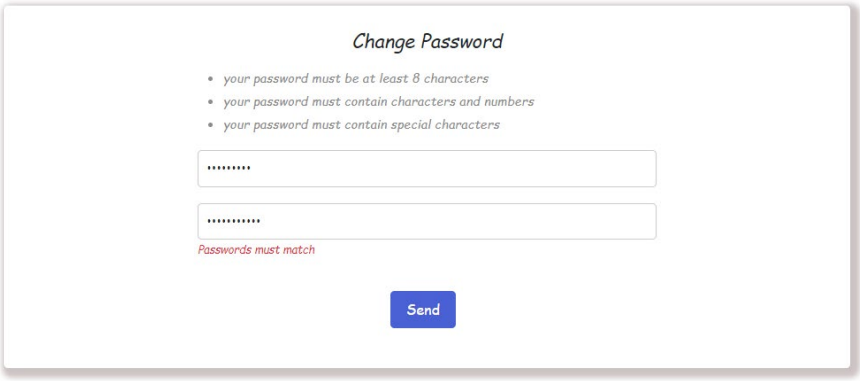

Figure 35 "Reset Password Test2"

3. **Test 3:**

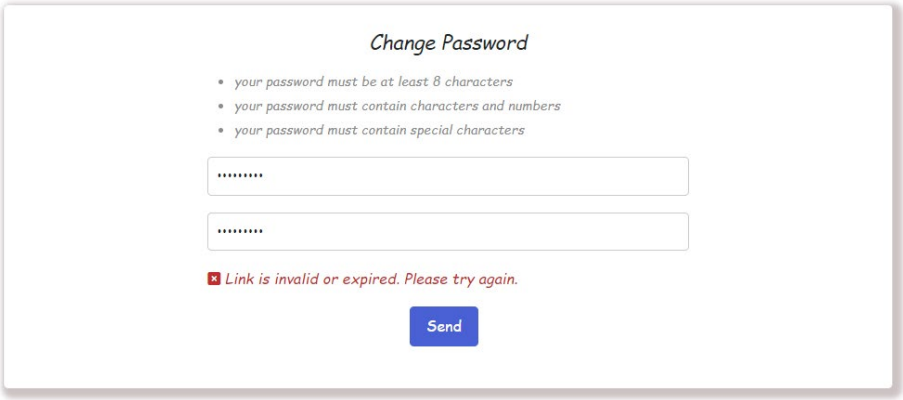

Figure 36 "Reset Password Test3"

4. **Test 4:**

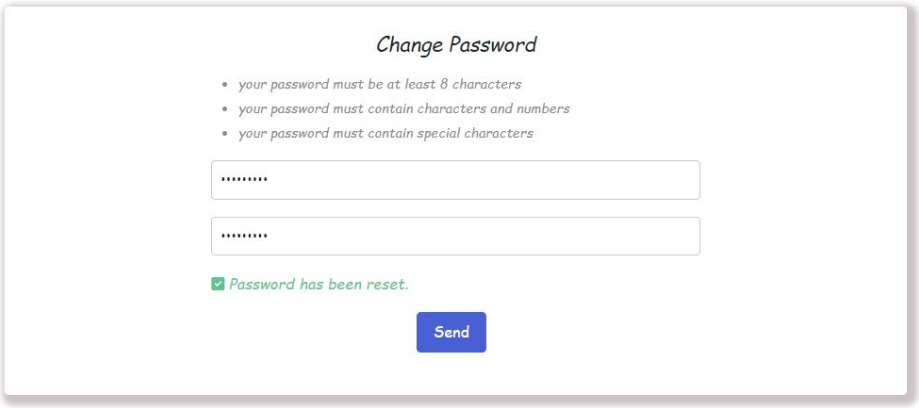

Figure 37 "Reset Password Test4"

Change Profile info Test:

1. **Test 1:**

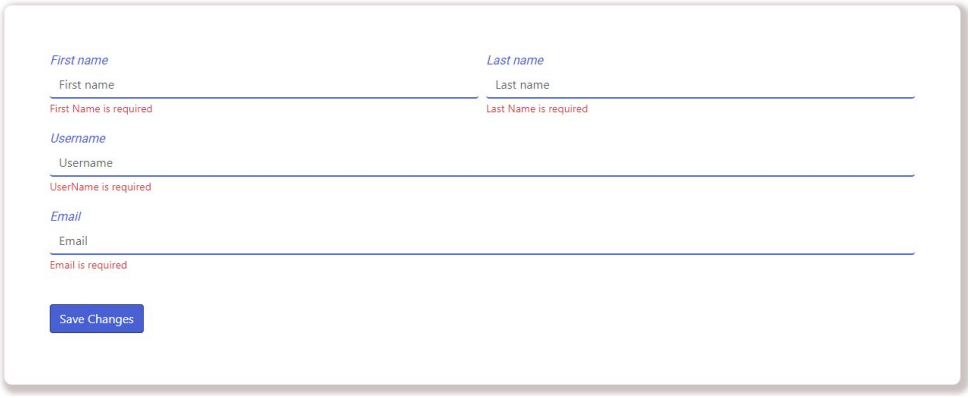

Figure 38 "Change Profile info Test1"

2. **Test 2:**

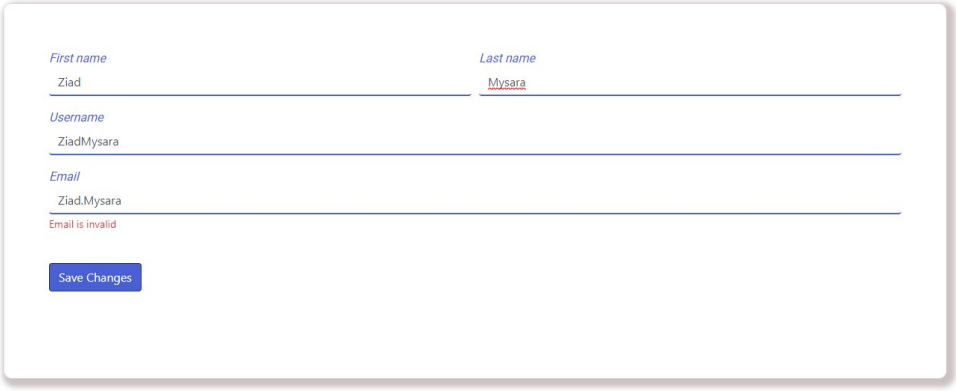

Figure 39 "Change Profile info Test2"

3. **Test 3:**

Figure 40 "Change Profile info Test3"

4. **Test 4:**

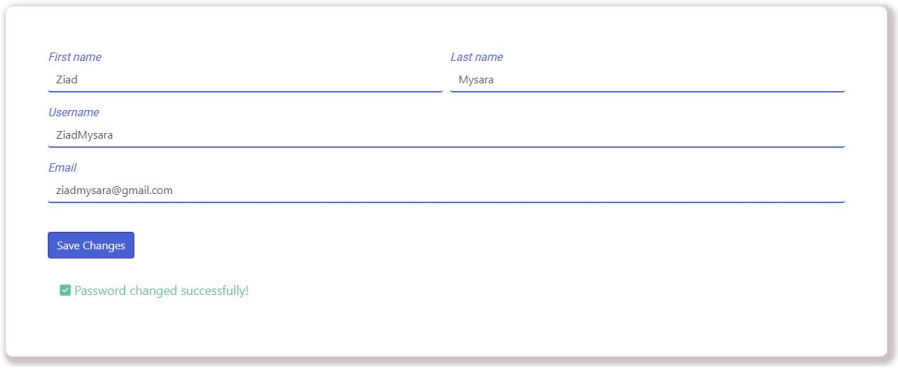

Figure 41 "Change Profile info Test4"

Change Password Test:

1. **Test 1:**

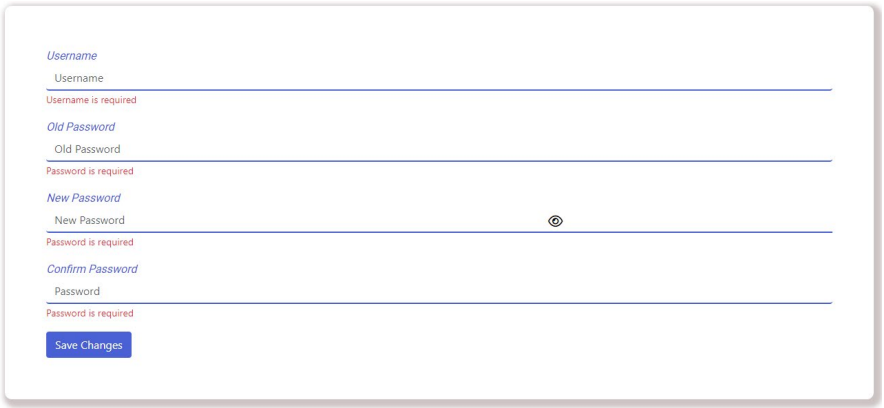

Figure 42 "Change Password Test1"

2. **Test 2:**

Figure 43 "Change Password Test2"

3. **Test 3:**

Figure 44 "Change Password Test3"

4. **Test 4:**

Figure 45 "Change Password Test4"

Spelling Test:

1. **Test 1:**

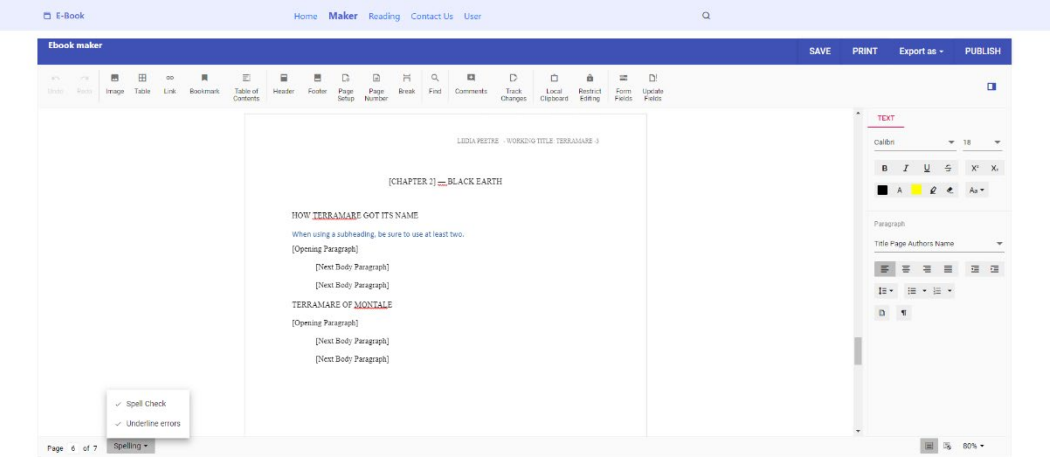

Figure 46 "Spelling Test1"

2. **Test 2:**

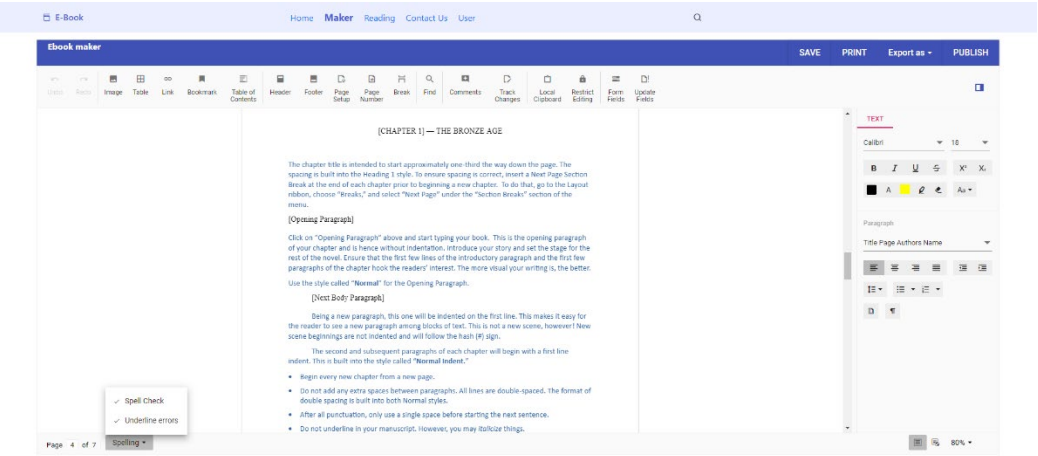

Figure 47 "Spelling Test2"

Save eBook Test:

1. **Test 1:**

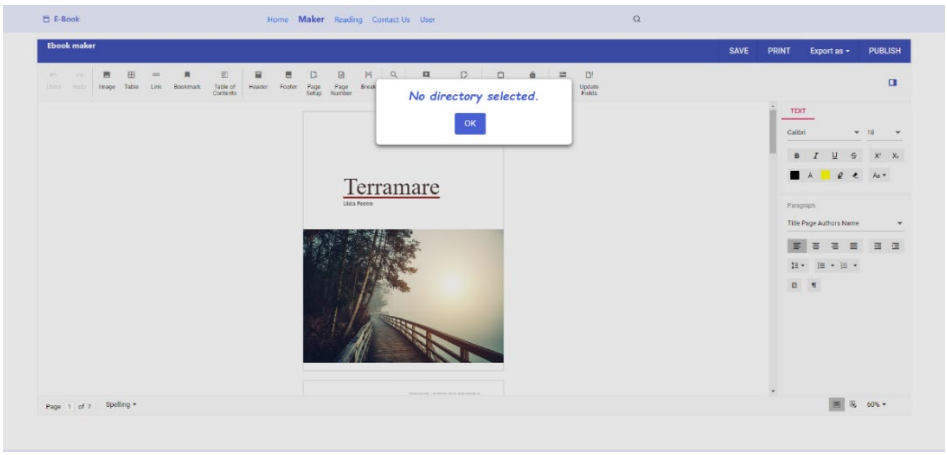

Figure 48 "Save eBook Test1"

2. **Test 2:**

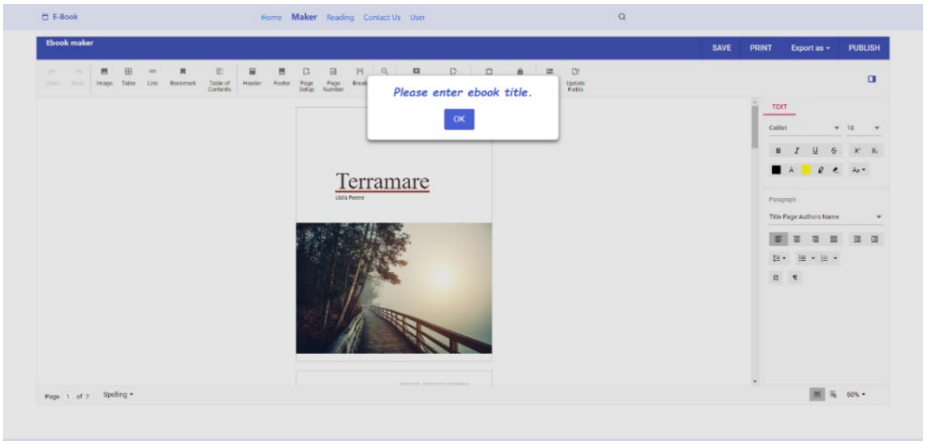

Figure 49 "Save eBook Test2"

3. **Test 3:**

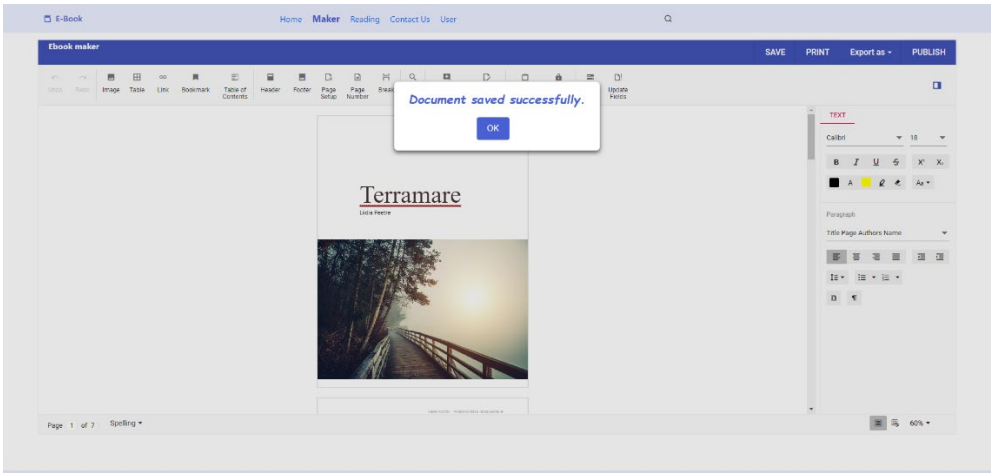

Figure 50 "Save eBook Test3"

eBook Export Test:

1. **Test 1:**

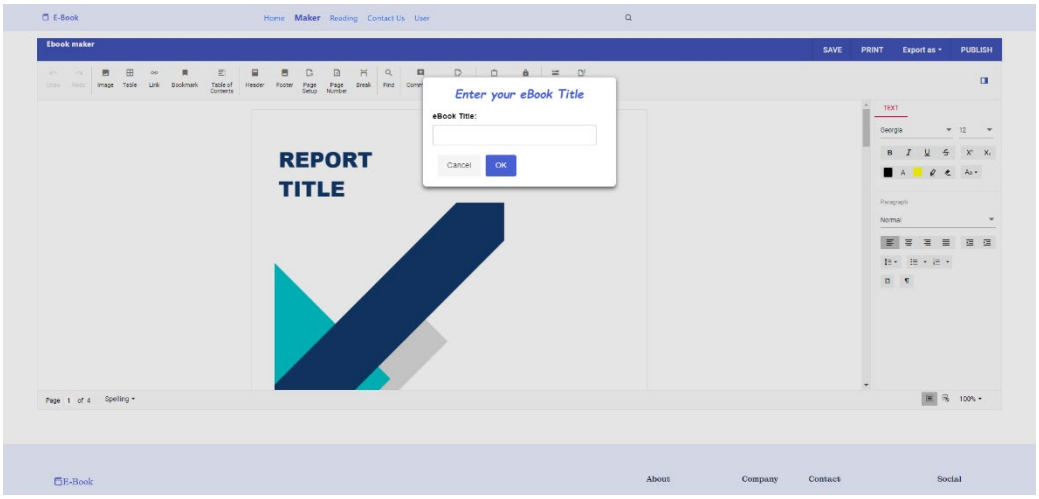

Figure 51 "eBook Export Test1"

2. **Test 2:**

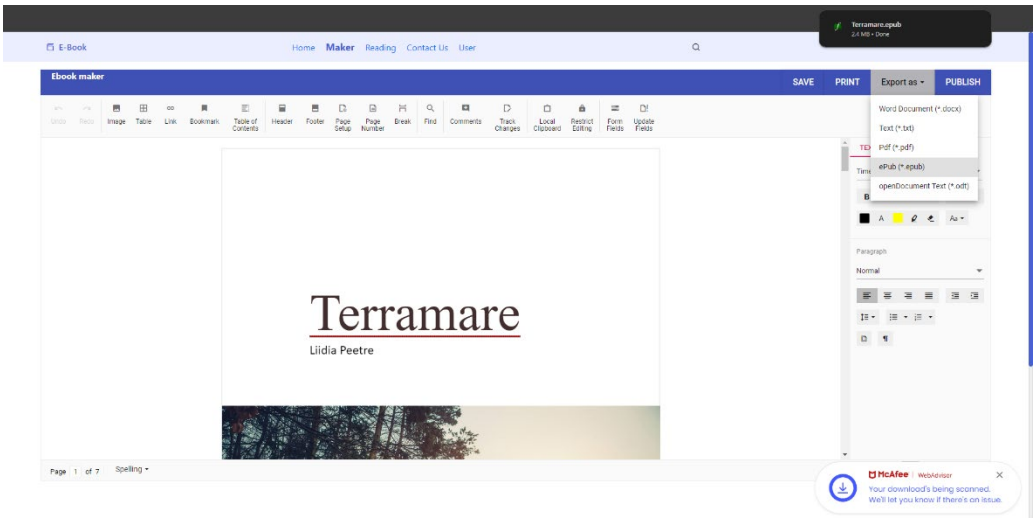

Figure 52 "eBook Export Test2"

Publish eBook Test:

1. **Test 1:**

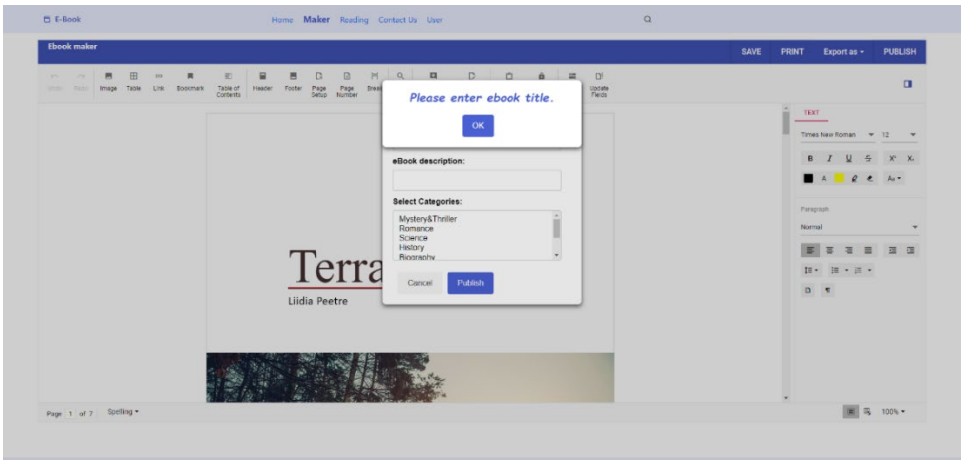

Figure 53 "Publish eBook Test1"

2. **Test 2:**

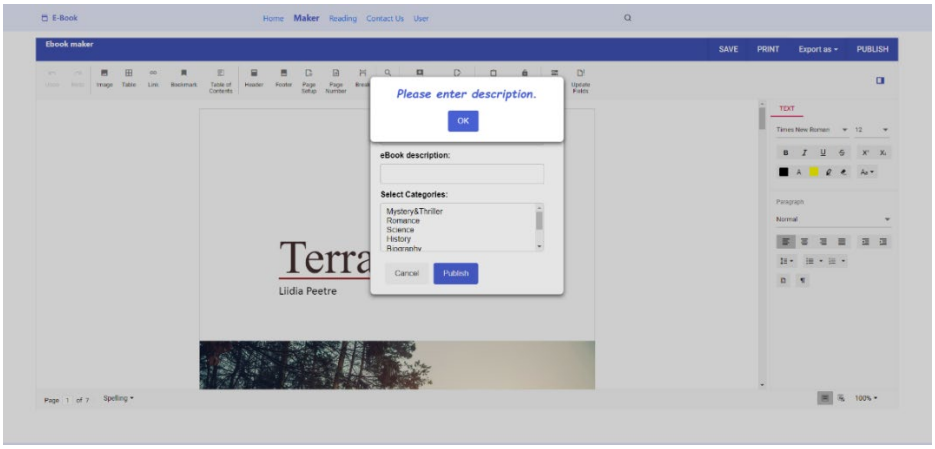

Figure 54 "Publish eBook Test2"

3. **Test 3:**

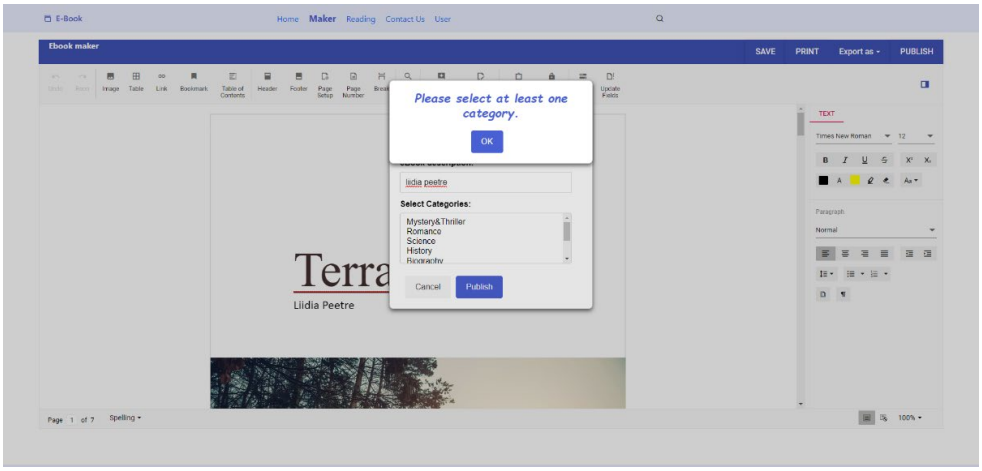

Figure 55 "Publish eBook Test3"

4. **Test 4:**

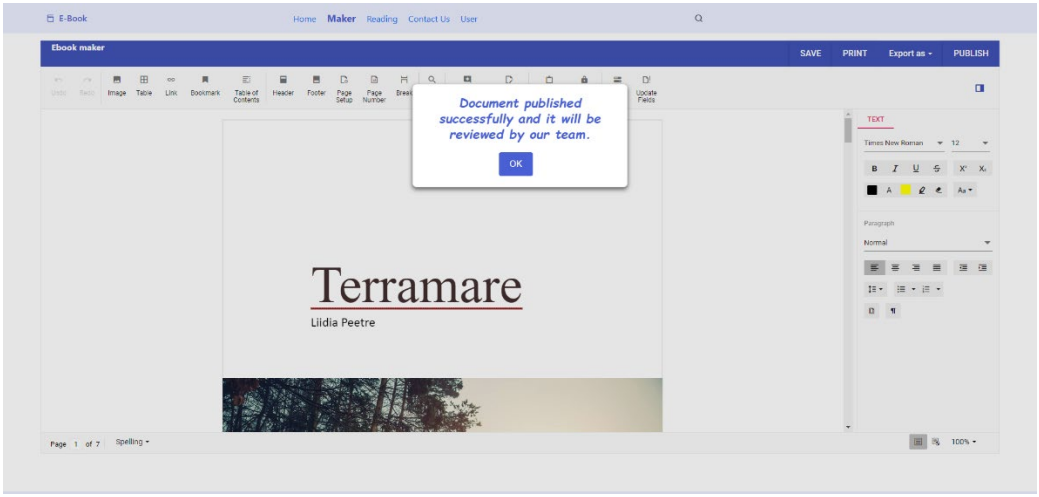

Figure 56 "Publish eBook Test4"

Add Comment Test:

1. **Test 1:**

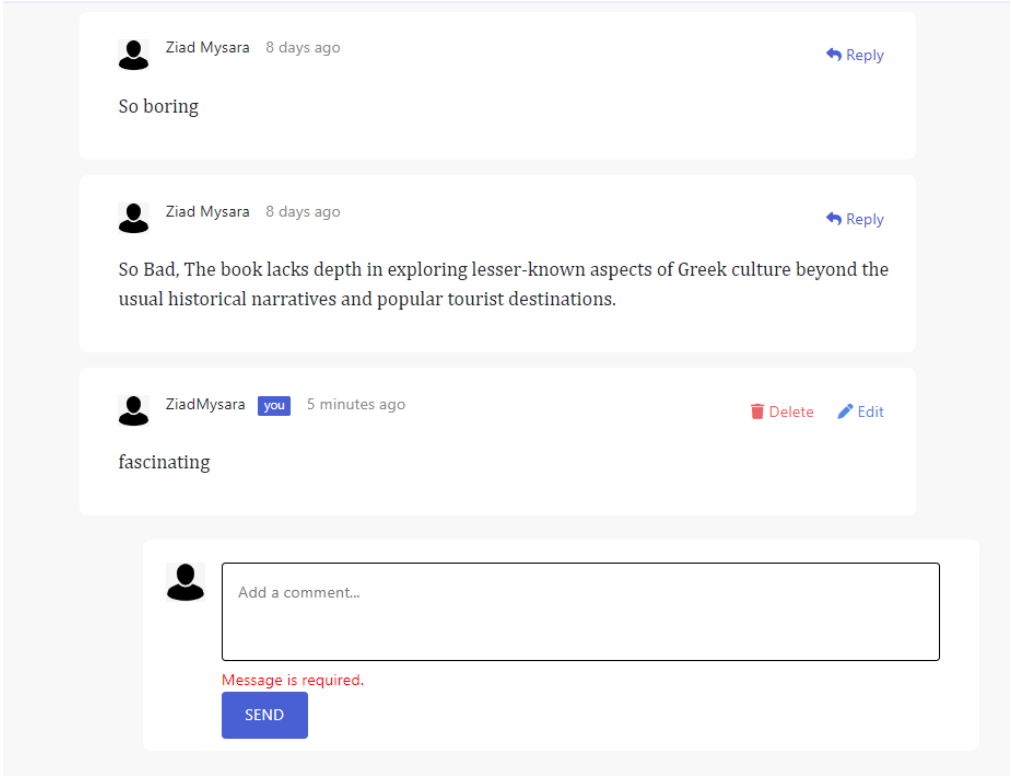

Figure 57 "Add Comment Test1"

2. **Test 2:**

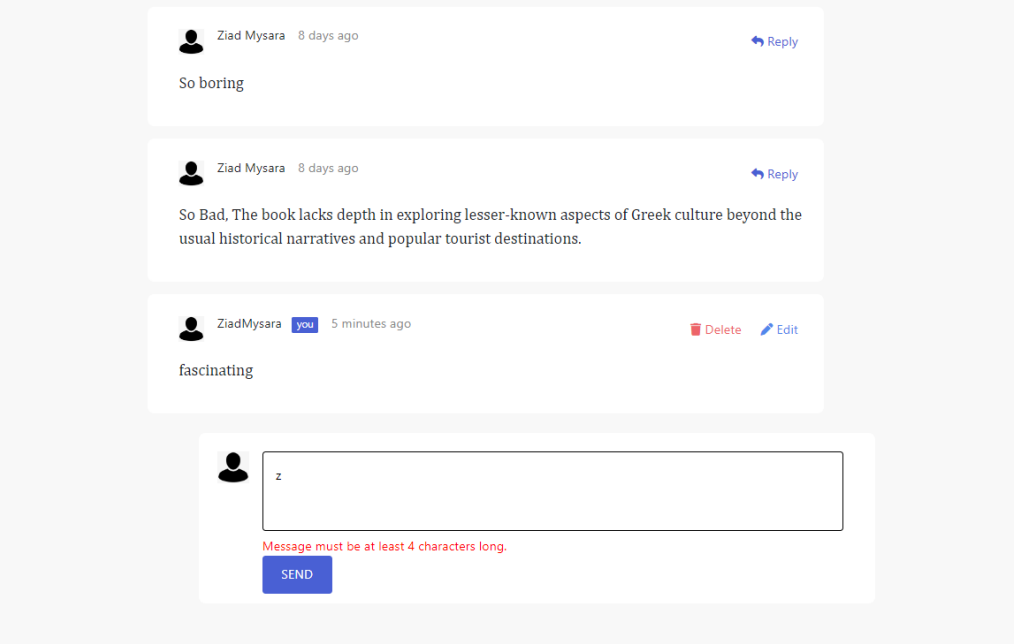

Figure 58 "Add Comment Test2"

3. **Test 3:**

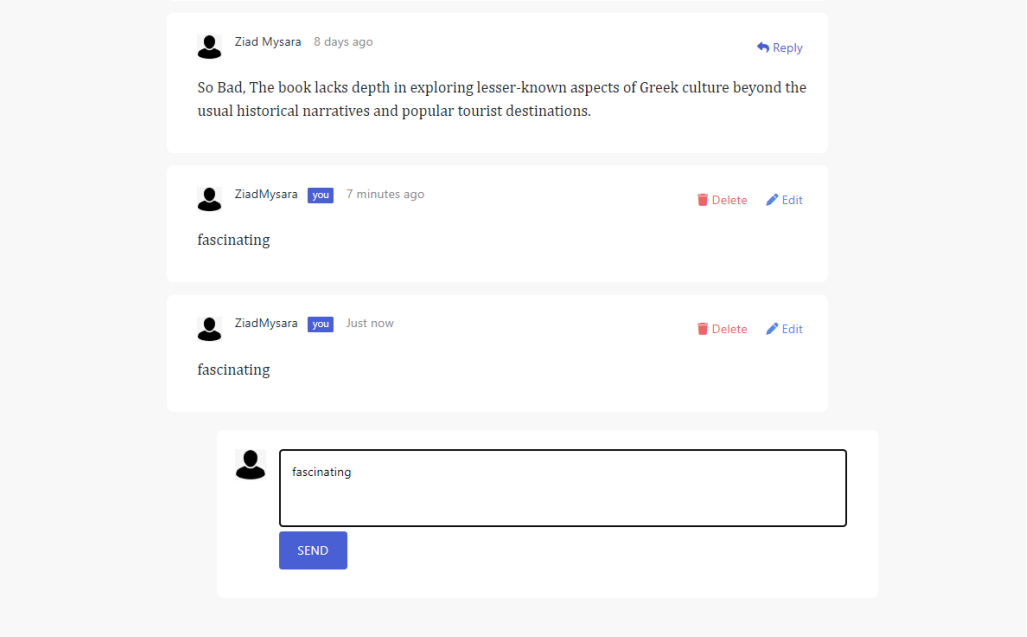

Figure 59 "Add Comment Test3"

Contact Us Test:

1. **Test 1:**

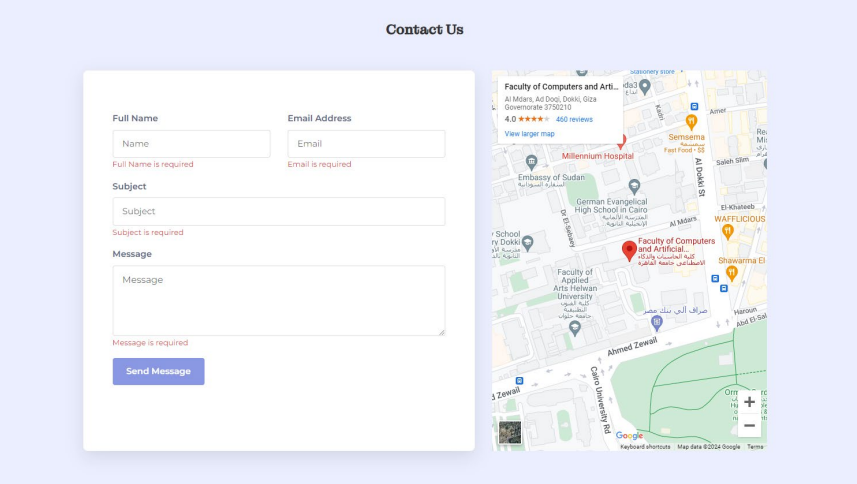

Figure 60 "Contact Us Test1"

2. **Test 2:**

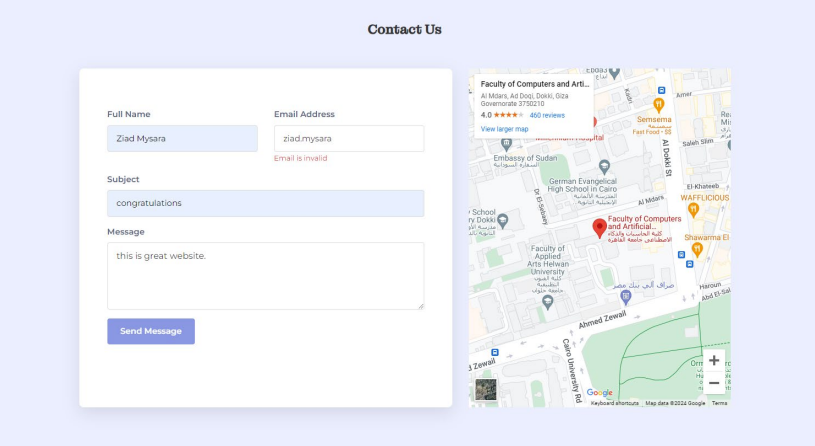

Figure 61 "Contact Us Test2"

3. **Test 3:**

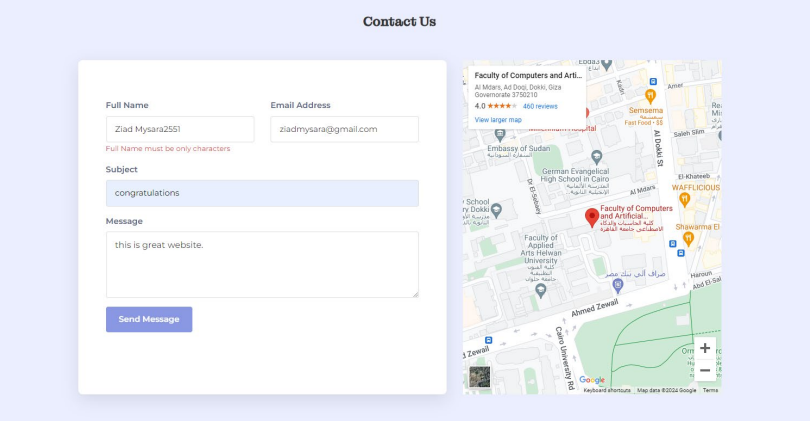

Figure 62 "Contact Us Test3"

4. **Test 4:**

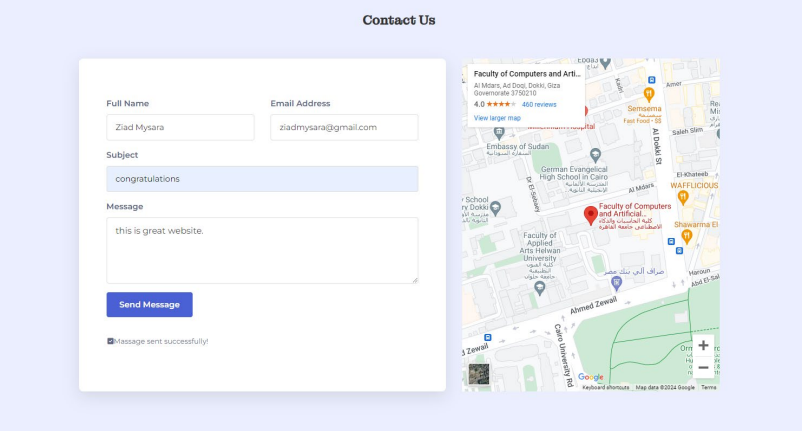

Figure 63 "Contact Us Test4"

Search Test:

1. **Test 1:**

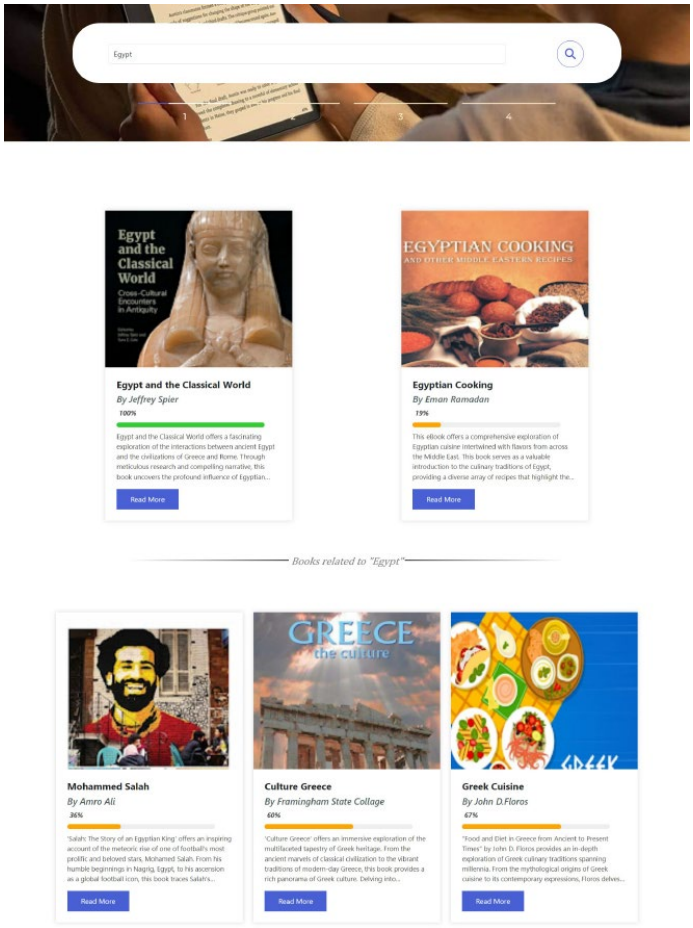

Figure 64 "Search Test1"

2. **Test 2:**

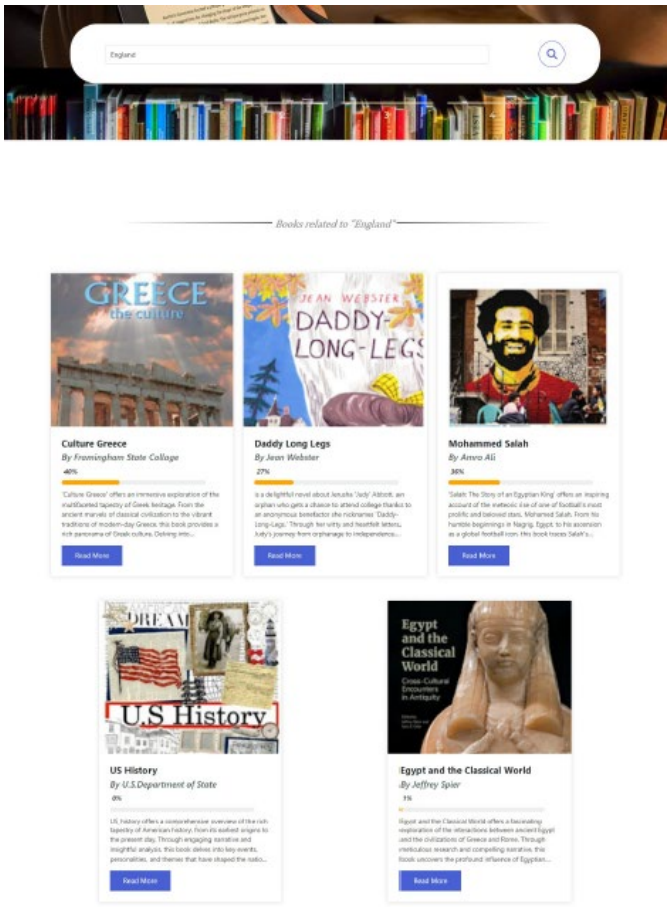

Figure 65 "Search Test2"

Reference

https://www.quora.com/What-are-the-challenges-that-e-book-sellers-face

https://www.godotmedia.com/4-challenges-that-a-new-ebook-author-should-be-prepared-to-face

http://dbpedia.org/ontology/

https://rapidapi.com/grammarbot/api/grammarbot

https://tarekraafat.github.io/autoComplete.js/#/

https://yes-pdf.com/

https://github.com/UB-Mannheim/tesseract/wiki

https://www.elastic.co/guide/en/elasticsearch/reference/current/zip-windows.html