

AUTOMATIC ANALYSIS OF TEXT DOCUMENTS IN DISTANCE LEARNING SYSTEMS BASED ON DEEP NEURAL NETWORK MODELS

Viunenko Oleksandr

Ph.D. in Economic Sciences, Associate Professor, Sumy National Agrarian University, Ukraine

ORCID ID: 0000-0002-8835-0704

The field of text document analysis and distance learning is constantly evolving, fueled by both academic research and real-world applications. Universities and research labs are exploring new deep learning architectures and developing advanced methods to solve complex text analysis tasks that are used by various industries to extract value from their growing repositories of text data. At the same time, distance learning has become an important component of modern education, allowing students to access lectures, training materials, and assessments remotely. The growth in the volume and complexity of text data in various industries highlights the urgent need for the implementation of efficient and accurate methods for document analysis. Traditional approaches to text analysis often prove insufficient to handle the modern scale and complexity of text documents. In contrast, the integration of deep neural networks and cloud services shows promise in the field of document analysis automation. This approach allows users to extract samples, generate explanations, and have an interactive dialogue with neural network models. Therefore, the development of a web service that will be able to automatically analyze text documents, provide the user with an interactive explanation of the analysis results, and improve student engagement in the learning process is a pressing task.

1. Review of recent research and publications. The growing volume of digital text data creates both challenges and opportunities in various fields. Automated text document analysis has become an important tool for extracting information, summarizing data, and discovering patterns in large document collections. This section reviews recent advances in text document analysis, with a particular emphasis on the use of deep neural networks.

The emergence of deep neural networks, especially large language models such as GPT (Generative Pre-trained Transformer), has revolutionized natural language processing tasks. These models, trained on large text arrays, demonstrate exceptional capabilities in understanding and manipulating human language. Using the capabilities of DNN (Deep Neural Networks), text document analysis can reach new levels of complexity, enabling tasks such as topic modeling, sentiment analysis, information retrieval, and automated generalization [9].

Text document analysis encompasses a number of methods aimed at extracting meaningful information from text data. These methods can be roughly divided into the following groups:

- information retrieval, focuses on finding the right documents in a collection based on a user query. Traditional approaches involve keyword matching, while DNNs can provide more sophisticated semantic search capabilities;
- text summarization, summarization methods aim to reduce a document to a shorter version while preserving its key points. DNNs can be used to create annotations that capture key information and maintain a coherent narrative flow;
- Topic modeling, uncovers hidden thematic structures in a collection of documents. DNNs can be used to discover new topics, track the evolution of topics over time, and discover relationships between different topics;
- sentiment analysis, aimed at determining the emotional tone or opinion expressed in a text document. DNNs are excellent at classifying sentiments as positive, negative, or neutral, and can even identify the nuances of emotional states;
- Named object recognition involves identifying and classifying named objects in a document, such as people, organizations, and geographic locations. DNNs can achieve high accuracy in named object recognition tasks, allowing them to extract structured information from text data;
- Text classification classifies documents based on predefined labels. DNNs can be used to develop robust classifiers capable of handling complex and multifaceted text data.

By applying these methods, text document analysis enables researchers, companies, and individuals to unlock hidden value in vast document collections [12].

Deep neural networks have become a powerful tool for text analysis due to their ability to learn complex patterns from large data sets. There are several types of DNNs whose architectures are well suited for text analysis tasks.

Recurrent neural networks are a type of deep neural network specifically designed to process sequential data, such as text. Due to their architecture, they are able to store and use information from previous stages of the sequence, which allows them to effectively capture context and long-term dependencies in text data. Long short-term memory networks are a special type of architecture equipped with mechanisms to solve the vanishing gradient problem, which is a common problem that limits the ability of DNNs to learn long-term dependencies. LSTMs (Long Short-Term Memory) excel at tasks that require the analysis of long sequences, making them well suited for tasks such as document generalization and topic modeling [2].

Transformer is a relatively new DNN architecture that has achieved significant success in various natural language processing tasks. Unlike RNNs, transformers process entire text sequences simultaneously, which allows them to capture relationships between words at any position in the sequence. The ability to process in parallel makes transformers highly efficient for tasks such as machine translation and text generalization.

The considered types of DNN architectures can be used for various tasks of text document analysis. Such tasks include: text summarization, topic modeling, sentiment analysis, information retrieval, named object recognition and text classification [13]. Recent research uses LSTMs and transformers to create summaries that not only capture key factual information, but also maintain a coherent narrative structure. These models are trained on large datasets of annotations written by humans, which allows them to learn the intricacies of generalization of different types of text documents. LSTMs and transformers are investigated for the task of topic modeling. These models can effectively detect hidden topics in collections of documents, track their evolution over time and reveal relationships between different topics. In particular, researchers are exploring the possibility of integrating domain-specific knowledge into DNNs to improve the accuracy of topic modeling in specialized fields. DNNs excel at sentiment analysis, outperforming traditional lexicon-based approaches. Convolutional neural networks and LSTMs are commonly used to analyze text and classify sentiment as positive, negative, or neutral. In addition, research is moving toward detecting more nuanced emotional states in text data [14].

Information retrieval has been revolutionized by providing semantic search capabilities. Modern DNNs are able to go beyond simple keyword matching and understand the underlying meaning of user queries, leading to more relevant and accurate document retrieval. Artificial neural networks achieve high accuracy in named object recognition tasks. These models are trained on large datasets annotated with named entities, which allows them to identify and classify different types of entities in text documents with high accuracy. DNNs are also used to develop robust text classifiers capable of handling complex and multifaceted text data. Convolutional neural networks and transformers show promising results in document classification based on predefined categories such as topic, genre, or mood. Although DNNs have revolutionized text document analysis, a critical evaluation is needed to identify areas for improvement and future research directions. Strengths include:

- high accuracy, as DNNs excel at learning complex patterns from large data sets, leading to better performance in various tasks compared to traditional methods;
- scalability, DNNs can work effectively with large collections of documents, making them suitable for analyzing huge arrays of text data;
- The versatility and diverse architectures of DNNs allow them to be adapted to various text analysis tasks, offering a flexible and powerful approach.

Significant limitations worth noting are:

- interpretability, the inner workings of complex DNN models can be opaque, making it difficult to understand how they produce specific results. Lack of interpretability can hinder trust and limit their application in certain areas;
- bias, DNNs are prone to inheriting biases present in the training data they are fed. Reducing bias requires careful data selection and the development of fair and unbiased training procedures;
- computational cost, training and running DNN models, especially large LLMs (Large Language Models) such as GPT-4, can be a very expensive process requiring significant computational resources [7].

By overcoming these limitations, DNNs have enormous potential to further revolutionize text document analysis and extract even deeper insights from the ever-growing sea of text data [12]. Overall, DNNs represent a transformative force in text document analysis, enabling researchers and practitioners to extract valuable insights from vast collections of text data. By overcoming current limitations and developing promising directions, DNNs have the potential to unlock even greater opportunities in this critically important field.

2 Software Product Analysis and Analogues. The first service for text document analysis is Google Natural Language AI [1]. A powerful tool that provides developers with natural language understanding technology. It allows developers to extract information from unstructured text using Google machine learning. This service is part of Google Cloud and offers a variety of functions for text analysis. One of the key features of Google Natural Language AI is its ability to extract, analyze and store text. It can train high-quality custom machine learning models without a single line of code using AutoML. This allows developers to upload their training data and test custom models without a single line of code. Google Natural Language AI applies natural language understanding to applications using the Natural Language API. The API includes functions such as sentiment analysis, entity analysis, entity sentiment analysis, content classification and syntax analysis. The sentiment analysis function examines a given text and determines the prevailing emotional sentiment in the text, especially to determine the author's attitude as positive, negative or neutral. On the other hand, entity analysis checks the text for known entities and returns information about those entities. Entity sentiment analysis combines these two functions, returning information about known entities and determining the prevailing emotional assessment of an entity in the text. Parsing is one of the functions of the Natural Language Processing API that extracts linguistic information by breaking down a given text into individual sentences and tokens, providing further analysis at the level of these tokens. Content classification analyzes the content of the text and returns the corresponding content category. Therefore, Google Natural Language AI is a comprehensive tool for analyzing text documents, offering a wide range of functions and capabilities that can be used for various tasks in the field of natural language understanding.

The next natural language processing service is Amazon Comprehend. It uses machine learning to find inferences and relationships in text. It is part of the Amazon Web Services suite and is designed to analyze and extract inferences from text [4].

Amazon Comprehend uses a pre-trained model to learn and analyze a document or set of documents. The model is continuously trained on a large body of text, eliminating the need for users to provide training data. The service identifies the language of the text, identifies key phrases, places, people, brands, or events, understands how positive or negative the text is, and analyzes the text using tokenization and parts of speech. It can also automatically organize a collection of text files into topics.

Amazon Comprehend analyzes documents, including identifying elements such as objects, key phrases, language, sentiment, and other common components. Objects include the names of people, places, objects, and locations mentioned in a document. Key phrases are meaningful phrases that occur in the text. Personal information includes data that can identify an individual, such as an address, bank account number, or phone number. Additionally, the dominant language of the document is determined. The mood of the document is classified as positive, neutral, negative, or mixed, with the target sentiment

associated with specific objects in the text. Syntax includes identifying parts of speech for each word in the document. Amazon Comprehend can process any text file in UTF-83 format. It also performs real-time analysis for small workloads or runs asynchronous analysis jobs for large sets of documents. Users can use pre-trained models provided by the web service or create their own models to classify and recognize objects. Amazon Comprehend can store user data to continuously improve the quality of its pre-trained models. In addition to these features, Amazon Comprehend supports different languages depending on the specific functionality. The dominant language detection feature can analyze documents and identify the dominant language from a large set of languages. Thus, Amazon Comprehend is a robust tool for analyzing text documents, offering a wide range of features and capabilities for performing various natural language processing tasks.

Each existing service has its own advantages and disadvantages, making them suitable for different usage scenarios and preferences. AWS Intelligent Document Processing offers the most complete set of features and integrations, making it ideal for users deeply integrated into the AWS ecosystem. Google Cloud Natural Language provides robust NLP capabilities and seamless integration with other Google cloud services. Microsoft Azure Text Analytics offers broad language support and competitive performance, making it an excellent choice for organizations already using Azure services. However, there is a significant drawback to these services that makes them less suitable for mass use - this is a complex interface and the lack of an interactive form of interaction, such as a chat interface. The main part of the users of the above web services are developers who use the API interface to integrate some of the functionality into their applications. Creating your own web service for analyzing text documents solves this problem. This service can be specially designed to address the specific needs of the general audience. It enables users to not only receive text analysis, but also interact with data in the most convenient and accessible way. The integration of deep neural networks will allow for more accurate and high-quality text analysis results that surpass the capabilities of existing services.

The last of the web services reviewed is Lexalytics. A leader in text analytics and natural language processing. It offers a set of software libraries that transform unstructured text into usable data and insights. These libraries are suitable for data analysts and architects who need full access to the underlying technology or who need to deploy locally for security or privacy reasons [16].

At the core of Lexalytics' offering is text analytics technology. Lexalytics text analytics works by breaking down sentences and phrases into their constituent parts, then evaluating the role and meaning of each part using complex programming rules and machine learning algorithms. This forms the basis for numerous natural language processing features, including named object recognition, categorization, and sentiment analysis.

Lexalytics' text analytics technology is designed to analyze four main aspects: identifying the subject of the utterance, the topic of discussion, the statements about these topics, and the emotions expressed. To do this, Lexalytics uses text mining tools that allow it to extract useful information and make contextual inferences from large volumes of raw text, such as social media comments, online reviews, and news articles. In addition to its core text analytics technology, Lexalytics offers a number of services to help customers get the most out of their text analytics solutions. These services include entity configuration, custom categorization, sentiment tuning, and the development of custom machine learning models. For example, Lexalytics can create and customize lists of entities tailored to a specific industry, including products, brands, and even addresses. It can also improve the accuracy of categorization and topic extraction by creating custom taxonomies.

When certain words and phrases have special meaning in a particular business or industry, Lexalytics can customize its sentiment scoring systems to reflect these specifics. Machine learning models are used to solve complex problems, providing more cost-effective solutions. These models are trained to perform very specific tasks, such as recognizing objects by ambiguous company names or categorizing food products and sauces.

Lexalytics offers a comprehensive set of tools and services for text analysis and natural language processing. Its capabilities range from basic text analytics to more advanced features such as sentiment analysis, named object recognition, and custom machine learning models. This makes it a reliable tool for analyzing text documents.

While Google Natural Language AI, Amazon Comprehend, and Lexalytics offer powerful text analysis capabilities, they also have certain limitations that may hinder their use by non-developers and regular users.

First, these services require a certain level of technical knowledge to use effectively. They are designed with developers in mind, so their interfaces and documentation are often complex and full of technical terms. This can make them inaccessible to regular users who don't have programming or data science experience.

Second, these services often require manual configuration and tuning to achieve optimal results. For example, sentiment analysis may need to be adjusted to accurately reflect sentiment in a specific industry or context. This requires an understanding of the underlying technology and can be time-consuming.

Third, these services usually lack a user-friendly interactive interface. Users often have to work with code or complex dashboards to analyze their text documents, which can be difficult for non-technical users.

So, while existing services like Google Natural Language AI, Amazon Comprehend, and Lexalytics offer powerful capabilities, their complexity and lack of a user-friendly interface pose significant challenges for ordinary users. Developing a new web service designed with ease of use and interactivity in mind could address these issues and make text analysis capabilities accessible to everyone.

3. Problem statement. The proliferation of digital text documents across industries has necessitated the development of advanced tools for effective analysis and understanding. With the advent of deep neural network models, such as OpenAI's GPT-4, the potential for automated text analysis has reached new heights. In this context, the development of a web service that uses such models to provide interactive explanations of text documents opens up vast opportunities for both research and practical application. The main problem is the complexity and inaccessibility of existing text document analysis services, such as Google Natural Language AI, Amazon Comprehend, and Lexalytics. These services, while powerful, are designed for developers and require a certain level of technical knowledge to use effectively. To achieve optimal results, they often require manual configuration and tuning, which can be time-consuming and difficult for non-technical users. In addition, these services lack a user-friendly interactive interface, which makes it difficult for ordinary users to effectively analyze text documents. Therefore, the current task is to develop a web service for analyzing text documents that is easy to use and accessible to ordinary users, not just developers. This service should offer a chat interface that allows users to interact with the system in a natural, conversational manner, asking questions and receiving answers in real time. This will make the process of text analysis more interesting and intuitive, removing the barriers that are often associated with more technical systems.

The problem is to develop and implement a web service that can seamlessly analyze text documents and provide users with deep explanations. The main goals of such a web service will be: 1) integrate state-of-the-art deep neural network models into the web service to facilitate document analysis and interpretation; 2) create an intuitive user interface that allows users to upload documents, interact with the system, and retrieve meaningful data without unnecessary effort; 3) implement a robust data management system to efficiently process user documents, queries, and responses, ensuring seamless integration with external services; 4) implement methods to transform text documents and user queries into semantically meaningful representations for efficient information retrieval and context generation; 5) ensure the scalability of the system to serve a large number of users and documents while maintaining high performance and responsiveness.

Integrating modern deep neural network models, in particular the GPT-4 model, into a web service is an important aspect for providing automated analysis and interpretation of text documents. This process includes several key steps to ensure effective interaction between the web service and the neural network model, as well as generating accurate and context-relevant responses.

The first step in model integration is to select models that are suitable for the text analysis task. The GPT-4 model stands out as a powerful language model capable of understanding and generating human-like text. Other models, such as BERT (Bidirectional Encoder Representations from Transformers), can also be considered depending on the specific requirements [8].

Once you have selected your models, you need to integrate with the OpenAI API to access their capabilities. This involves configuring API endpoints to communicate with the model, handling authentication, and configuring query parameters such as model version and token limits. Before passing text data to the model, some preprocessing may be required to ensure compatibility and optimal performance. This may include tokenization, removing special characters, and handling any linguistic nuances or formatting issues. When analyzing text documents, it is important to provide the model with context so that it can generate meaningful responses. This can be done by segmenting the document into smaller chunks or providing relevant background information to guide the analysis process. After the model has processed the input text, the generated responses need to be analyzed, formatted, and presented to the user in a clear and understandable form. This may include post-processing steps such as summarizing, highlighting key points, or creating explanations.

As with any external API integration, it is important to have error handling mechanisms in place to handle issues such as network outages, API speed limits, or unexpected model errors. Implementing retry strategies and fallback mechanisms can help ensure service reliability.

To maintain responsiveness and scalability, performance optimization techniques such as caching frequently used responses, batching queries, or using asynchronous processing for long-running tasks can be applied. By effectively integrating deep neural network models into a web service, users can take advantage of advanced text analysis capabilities, including generalization, sentiment analysis, object recognition, and question answering. The integration forms the foundation of the web service, allowing it to provide accurate and insightful explanations of text documents in an interactive manner.

Creating an intuitive and engaging user interface is essential to enable users to easily upload documents, interact with the system, and extract meaningful information from their text documents. User interaction encompasses various aspects of a web service, including uploading documents, submitting queries, and presenting results.

User interaction with the system consists of the following elements:

- The document upload interface should be visually clear and attractive to help users easily upload their PDF documents. It should provide feedback on the progress and success of the upload. Implement validation checks to ensure that only PDF documents are accepted for upload. Provide informative error messages if the upload fails due to incorrect file format, size limit, or other issues, guiding users on how to resolve the issue.
- The chat interface should be intuitive, resembling a conversation with the system. Users should be able to type their queries into a text input field and easily submit them. Provide hints or suggestions to help users understand what types of queries they can ask or how to effectively interact with the system. Provide real-time feedback when submitting a query, such as letting them know when the system is processing the query and when a response will be ready.
- Displaying the results of the analysis in an interactive way, for example, using a chat interface or dynamically updated visualizations. This allows users to interact with the results and seek further clarification if necessary. Highlighting key ideas or important sections of the document that are relevant to the user's query. This will help users quickly understand the most relevant information;
- Present the user interaction flow in a logical sequence, seamlessly guiding users through the various stages of loading, querying, and exploring results. Develop navigation elements such as the

top navigation bar and backlinks to help users easily navigate between different sections of the web service. Maintain design and interaction patterns across the interface to ensure a cohesive user experience;

- The interface should be accessible to users with disabilities by implementing features such as screen reader compatibility, keyboard navigation, and alternative text for visual elements. Optimize the interface for different devices and screen sizes to ensure the same experience on desktop, tablet, and mobile devices.

Effective data management is crucial to ensuring the smooth operation of a web service, especially when processing user documents, requests, and responses. Data management encompasses several key components, including storage, retrieval, and integration with external services. Using third-party services to store PDF documents uploaded by users. After a document is uploaded, the service generates a unique URL, which is then stored along with additional metadata in a relational database. This metadata includes information such as the user ID, document title, upload timestamp, and any relevant tags or categories associated with the document. The relational database serves as a central repository for managing user data, consisting of several three main tables. A table for storing user information, including usernames, hashed passwords, and authentication tokens. A table for storing uploaded documents, with a one-to-many relationship to the previous table to associate documents with their owners. That table for storing queries and responses also has a one-to-many relationship to link queries to the documents they were made to.

Ensuring data consistency, security, and scalability is a priority in the data management process. To achieve this, the web service must perform a number of additional tasks:

- User data, including passwords and document content, is encrypted both in transit and at rest to prevent unauthorized access;
- Access control is used to restrict access to user data based on their authentication and authorization levels. Only authenticated users can access their own documents;
- The data management system is designed to scale horizontally to accommodate growing user bases and document collections. This involves distributing data across multiple servers and using load balancing techniques to efficiently distribute incoming requests.
- Regular backups of user data are performed to prevent data loss in the event of equipment failures or other unforeseen circumstances. In addition, there are robust recovery procedures for rapid data recovery in the event of a failure;

Overall, effective data management ensures that a web service can securely store, retrieve, and process user data, enabling seamless interaction and analysis of text documents. By integrating with external services, a web service can leverage their capabilities to provide users with a comprehensive and robust document analysis experience.

Semantic understanding is a key aspect of the web service, providing accurate interpretation of the meanings of text documents and user queries. The web service uses the OpenAI Embeddings API to transform text documents and user queries into semantic vectors. By transforming text into a vector space, the web service can perform semantic operations such as similarity comparison and contextual analysis.

When a user downloads a PDF document, its contents are processed to extract textual information. This text is then passed through the Embeddings model to create a semantic vector representation. The vector encapsulates the semantic meaning of the document, reflecting its key concepts, topics, and word relationships. Similarly, when a user submits a query, it is also transformed into a semantic vector that reflects the semantic content of the query and serves as the basis for extracting relevant information from the document database.

Semantic search methods are used to match semantic vectors of user queries with the content of documents, involving the calculation of similarity scores between vectors using methods such as cosine similarity. Documents with vectors most similar to the query vector are found and considered as potential context for generating answers. Once relevant documents are identified, their semantic

vectors are used to provide context to deep neural network models. By incorporating this context into the analysis process, the models can generate more accurate and contextually relevant answers to user queries [19]. Semantic understanding underlies the ability of a web service to analyze text documents and provide meaningful explanations. Using the Embeddings model and semantic search methods, the service can effectively bridge the semantic gap between user queries and document content, allowing for more accurate and in-depth analysis.

4. Web service design. When designing a web service, one of the fundamental decisions is the choice of a web framework. This choice will significantly affect the development process, the capabilities of the service, and its performance. Three popular web frameworks that stand out in the modern landscape are Next.js, Nuxt, and SvelteKit. Each of these frameworks has its own unique advantages and may be more suitable for certain types of projects.

Next.js is a robust framework based on the React library, offering server-side rendering and static site generation. It is known for its flexibility and compatibility with a wide range of browsers [10]. Nuxt is a more independent framework based on the Vue.js library. It provides more defined conventions and standards, which can simplify and speed up the development process [11]. SvelteKit is a meta-framework built on the lightweight Svelte library. It is distinguished by its performance and code simplicity thanks to a reactive data model [15].

To make an objective comparison between Next.js, Nuxt, and SvelteKit, several important parameters were taken into account. The results of this comparison are presented in Table 1.

Table 1. Comparison of Next.js, Nuxt and SvelteKit frameworks

Function	Next.js	Nuxt.js	SvelteKit
Base library	React	View	Svelte
Routing	Directory-based, API-based	Directory-based, Nuxt-specific routes	Directory-based
Rendering	Server, Client, Static	Server, Client, Universal	Server, Client, Static
Expansion	Custom Next.js API, third-party React libraries	Nuxt modules, third-party Vue libraries	SvelteKit adapters, third-party Svelte libraries
Deployment	Vercel, Netlify, AWS Amplify	Netlify, AWS Amplify, Vercel	Vercel, Netlify, AWS Amplify
Advantages	Large community, rich feature set, well documented	Flexibility, ease of use, integration with the Nuxt ecosystem	Light weight, high performance, svelte components
Disadvantages	More difficult learning curve, larger package size	Smaller community, fewer third-party libraries	Smaller ecosystem, less documentation
Using	E-commerce, SaaS, marketing sites	Blogs, forums, single-page web applications	Mobile web applications, high-performance websites

Of all the parameters listed, Next.js comes out on top in most categories. This framework is based on the React library, which is one of the most popular for developing web applications, and has a significant developer community. Next.js's native API and support for a large number of third-party libraries simplify the process of integrating new features. In addition, the framework has a large user community, an extensive set of functionalities, and well-structured documentation, which makes it an advantage for developing web services.

TypeScript was chosen as the primary development language for this project. While JavaScript initially offers a faster development cycle, the advantages of TypeScript outweigh the disadvantages in a project of this scale. Static typing will greatly improve the maintainability of the code and reduce the likelihood of errors when executing a complex web service. In addition, the excellent tooling support for TypeScript will improve the development experience.

The next important step after choosing a framework and programming language is determining how you will upload and store documents, particularly PDFs. There are several services for this, each with its own set of features and considerations.

One of the most well-known options is Amazon S3, a widely used cloud storage service that offers scalable, reliable, and secure storage for a variety of data types, including documents. With Amazon S3, users can upload their PDF documents directly to the cloud, and the service provides a unique URL for each stored file. This URL can be stored in a database for easy retrieval and management [3].

Another option to consider is Google Cloud Storage, which provides features similar to Amazon S3. It offers scalable storage for large objects, including PDFs, with the ability to access and manage them programmatically using APIs. Google Cloud Storage also provides high availability and durability for stored data [6].

Alternatively, Microsoft Azure Blob Storage is another contender in this space. Azure Blob Storage allows users to store large amounts of unstructured data, including PDF documents, and provides features such as encryption, access control, and integration with other Azure services [5].

Each storage service has its own advantages and disadvantages. Amazon S3, for example, is widely used and offers a wide range of features, including version control, lifecycle management, and access control policies. Its “pay-as-you-go” pricing model makes it suitable for projects of all sizes, although costs can increase as usage increases.

On the other hand, Google Cloud Storage provides tight integration with other Google Cloud Platform services, which can be useful if a project uses other Google cloud services. It also offers features such as object versioning and object lifecycle management. In terms of cost, Google Cloud Storage is generally competitive, and depending on specific requirements, it can offer cost-saving options such as Nearline and Coldline storage classes.

Microsoft Azure Blob Storage, as part of the Azure ecosystem, offers seamless integration with other Azure services. It provides features such as Azure Storage Explorer for easy management, Blob versioning, and Blob lifecycle management. Pricing may vary based on storage reservation options and access level selection.

The decision on which service to use should consider not only the technical aspects, but also factors such as support, documentation, and the ecosystem of tools and services available on each platform. Additionally, the availability of client libraries and SDKs to integrate with the chosen service can simplify the development process.

Amazon S3 stands out among the three services due to its extensive capabilities, robust scalability, and broad integration capabilities into the AWS ecosystem. It offers high reliability, security, and availability, making it suitable for a variety of applications and workloads. Additionally, its pay-as-you-go pricing policy provides efficiency and flexibility. Meanwhile, while Google Cloud Storage and Microsoft Azure Blob Storage offer similar capabilities, Amazon S3’s leadership is clear due to its comprehensive feature set, reliability, and recognized reputation in the industry.

The choice of a database management system for a web service depends on various factors, such as the nature of the data, the expected traffic, and the specific requirements of the web service. In the context of the proposed web service, three popular open source relational database management systems are considered - MySQL, MySQL, due to its wide range of features, community support, and easy configuration, is often considered the optimal all-purpose system for most web applications. It is characterized by high performance, reliability, and ease of use. MySQL supports a wide range of data types and provides strong security features, making it a suitable choice for web services that process sensitive user data, such as personal information and PDF documents.

PostgreSQL is known for its high level of customization and adherence to SQL standards. It supports a wide range of data types, including those not found in MySQL. PostgreSQL's syntax is close to standard SQL, which can be useful for transferring skills to other database management systems.

Table 2. *Comparison of Amazon S3, Google Cloud Storage and Microsoft Azure Blob Storage web services*

Parameter	Amazon S3	Google Cloud Storage	Microsoft Azure Blob Storage
Scalability	Highly scalable with virtually unlimited capacity	High scalability with region selection	High scalability with selectable temperature settings
Security	Offers encryption, access control, and data integrity	Provides encryption, access control, and IAM policies	Offers encryption, access control, and integration with Azure AD
Accessibility	Provides high availability with uptime SLA	Offers high availability with a robust network infrastructure	Provides high availability with service level agreements
Integration with other services	Integrates well with the AWS ecosystem and third-party tools	Has strong integration with Google Cloud Platform services	Provides seamless integration with other Azure services
Cost of efficiency	Offers competitive pricing with pay-as-you-go pricing	Competitive prices with a choice of economy classes	Cost varies depending on storage types and access level
Object lifecycle management	Provides lifecycle policies for automated data management	Offers asset lifecycle management to optimize costs	Supports BLOB lifecycle management for automated policies

SQLite is a lightweight database management system that does not require a server process, which provides faster performance. It is a popular choice for applications that require embedded databases instead of a client-server architecture. However, SQLite may not be the best choice for web services that expect high traffic or need to handle complex queries, as it lacks some of the advanced features provided by MySQL and PostgreSQL.

MySQL is the leader among these DBMSs due to its ease of use, high performance, scalability, and extensive feature set. Its support for partitioning makes it an excellent choice for working with large databases, such as the one required for the described web service. In addition, the extensive support of the MySQL community ensures that help is readily available when needed. Thus, MySQL is a suitable choice for the described web service, given its requirements and the nature of the data it processes.

In addition to storing text data and metadata in relational databases such as MySQL, vector databases are needed to efficiently manage and query high-dimensional semantic vectors created from documents. When choosing a vector database, several factors must be considered, including scalability, query performance, and ease of integration with existing architecture.

The choice of vector database depends on several factors, including the specific requirements of the service, the scale of the data, and the desired performance. The three vector databases under consideration - Pinecone, Milvus, and Weaviate - have their own strengths and tradeoffs.

Pinecone is a managed solution that provides real-time search capabilities and is easy to use. It is especially suitable for startups and companies that prioritize rapid development and scalability. Pinecone's managed solution is useful for web services that require seamless integration with minimal configuration and maintenance efforts.

Milvus is an open source tool that provides fine-grained control over performance and customization. Developers who are comfortable working with open source tools and who need more control over their database may prefer Milvus. It may be a good fit for a web service that requires specific customizations and has developers who want to manage the database themselves.

Weaviate is another open source option that efficiently manages relationships between data entities. It can be a good choice for text analysis services that involve not only documents but also entities and their relationships. Weaviate provides a GraphQL interface for accessing data and integrates well with various frontend development frameworks. However, the vector search functionality in Weaviate may be less developed compared to Pinecone or Milvus.

The choice between Pinecone, Milvus and Weaviate depends on specific requirements, namely: ease of use, scalability, cost, customization. Each of these systems offers different capabilities and functionality, which makes them more suitable for different usage scenarios.

Pinecone seems to us to be the most suitable solution for a vector database for a learning web service. Pinecone's managed service approach simplifies the setup and maintenance process, making it an attractive choice for rapid deployment and integration into the existing Next.js framework. The simple API aligns well with the development environment and facilitates efficient integration with the GPT model used in the web service, facilitating seamless data exchange and enabling efficient use of advanced text analysis capabilities. The cost-effectiveness aligns well with the potential growth of the service, and integration with machine learning frameworks extends the capabilities of advanced text analysis. Therefore, the combination of ease of use, scalability, cost-effectiveness, and integration with machine learning makes Pinecone a reasonable choice for a text document analysis web service.

5. Database design. Database design is a critical step in the development of information systems, as it defines the data structure that provides efficient storage, retrieval, and manipulation of information. The quality of database design determines its performance, scalability, data integrity, and ease of maintenance. The theoretical framework underlying database design includes the concepts of data modeling, normalization, integrity assurance, and transaction management.

Web service users need a system that supports secure login, document uploads, and interactive queries with the GPT-4 model. The main information needs identified include:

- user authentication and authorization;
- storage and search for downloaded documents;
- generation and storage of semantic vectors for documents and user queries;
- effective search and extraction of information based on semantic vectors;

The main business processes include user registration and login, document upload, document analysis, and interactive queries. The functional requirements arising from these processes are as follows:

- secure and efficient user management system;
- integration with Amazon S3 for document storage;
- using the OpenAI API to create semantic vectors and interactive responses;
- a robust search engine to facilitate quick retrieval of relevant document content;

The types of data stored in a database are divided into the following categories:

- information related to user accounts, such as username, email address, and user profile data;
- metadata about uploaded documents, including URLs from Amazon S3, document names, upload timestamps, and user associations;
- user queries and responses from the GPT-4 model, as well as associated metadata such as timestamps and session IDs.

Database design should provide scalability to accommodate growing user and document volumes. Performance optimization strategies include indexing key fields, efficient data retrieval algorithms, and appropriate use of vector databases for storage and insert queries.

The first stage of database design for a text document analysis web service should provide a clear understanding of the requirements and expectations. This knowledge will be used in the subsequent stages of database schema development to ensure that the system meets user needs and operates effectively under expected loads. The next steps involve translating these requirements into a detailed database schema and implementing the necessary infrastructure to support the defined business processes. The second stage of database design for a text document analysis web service involves infological modeling. This stage focuses on creating a high-level data model that encapsulates the information needs of the organization. The model is abstract and independent of specific technologies, focusing on entities, their attributes, and business rules.

The main information objects defined for the web service are User, Document, and Message. Each entity represents a critical aspect of the business process and has specific properties associated with it. User represents the individuals who use the web service. Attributes: User ID, Email Address, Profile Image URL. Document represents a PDF document uploaded by the user. Attributes: Document ID, Document Name, URL where the document is stored, Amazon S3 unique key, Upload Status, Creation Date, Update Date. Message represents the interaction between the user and the GPT model. Attributes: Message ID, Message Text Content, Creation Date, Update Date, ID to determine whether the message is a user or system message.

The next step is conceptual modeling, which transforms the infological model into a more abstract and generalized structure that reflects the basic principles and logic of the system. The main tool for this stage is the ER model, which clearly reflects the entities, their attributes, and the relationships between them.

Each user can upload many PDF files, but each file can only be associated with one user. This maintains data integrity by ensuring that a file is always clearly associated with the specific user who uploaded it and allows for efficient searching for files associated with a specific user. The Document entity will have an attribute that will contain the ID of the user who uploaded the file.

A one-to-many relationship between the User and Document entities. Each user can upload many PDF files, but each file can only be associated with one user. This maintains data integrity by ensuring that a file is always clearly associated with the specific user who uploaded it and provides efficient search for files associated with a specific user. The Document entity will have a `userId` attribute (a foreign key) that contains the ID of the user who uploaded the file. Figure 1 shows a generalized infologic database model.

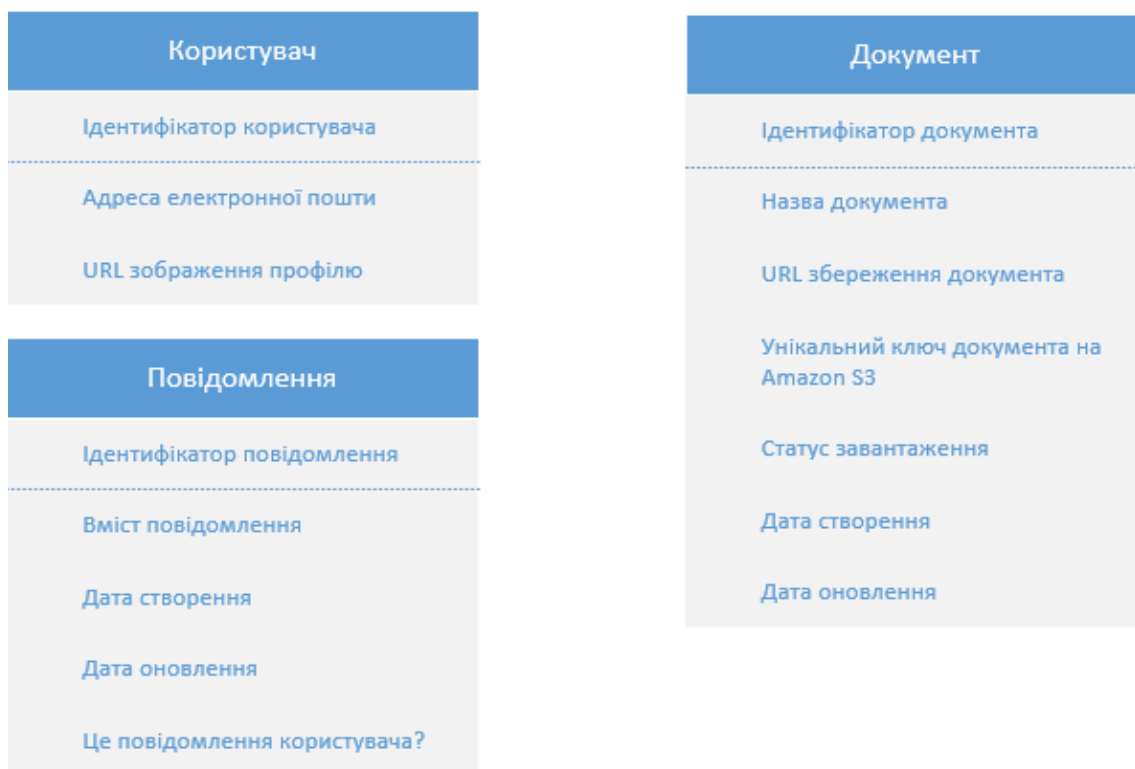


Fig. 1. Infological database model

A one-to-many relationship between the Document and Message entities. A file can have many messages associated with it, but a message can only be associated with one file. This allows a user to maintain a history of their interaction with the GPT model associated with a particular file. Provides context for messages, allowing users to understand what a particular message is about. The Message entity will have a fileId attribute (a foreign key) that will contain the ID of the file it is associated with.

One-to-many relationship between the User and Message entities. Each user can create many messages, but a single message can only be created by one user. This corresponds to the main use case where users interact with the GPT model. Maintains data integrity by ensuring that each message is clearly associated with the user who created it. Allows you to track user activity and analyze their interactions with the model. The Message entity will have a userId attribute (a foreign key) that will contain the ID of the user who created it.

The detailed relationships between entities in the conceptual database model described provide a flexible and efficient structure for storing web service data. The choice of one-to-many relationships corresponds to the system usage scenarios. The model is the basis for further logical and physical database design. Figure 2 shows the conceptual model of the web service database.

Based on the conceptual model described earlier, we can now move on to logical modeling. This stage involves transforming the abstract entities and relationships of the conceptual model into a clear and detailed structure that can be implemented in a specific database management system.

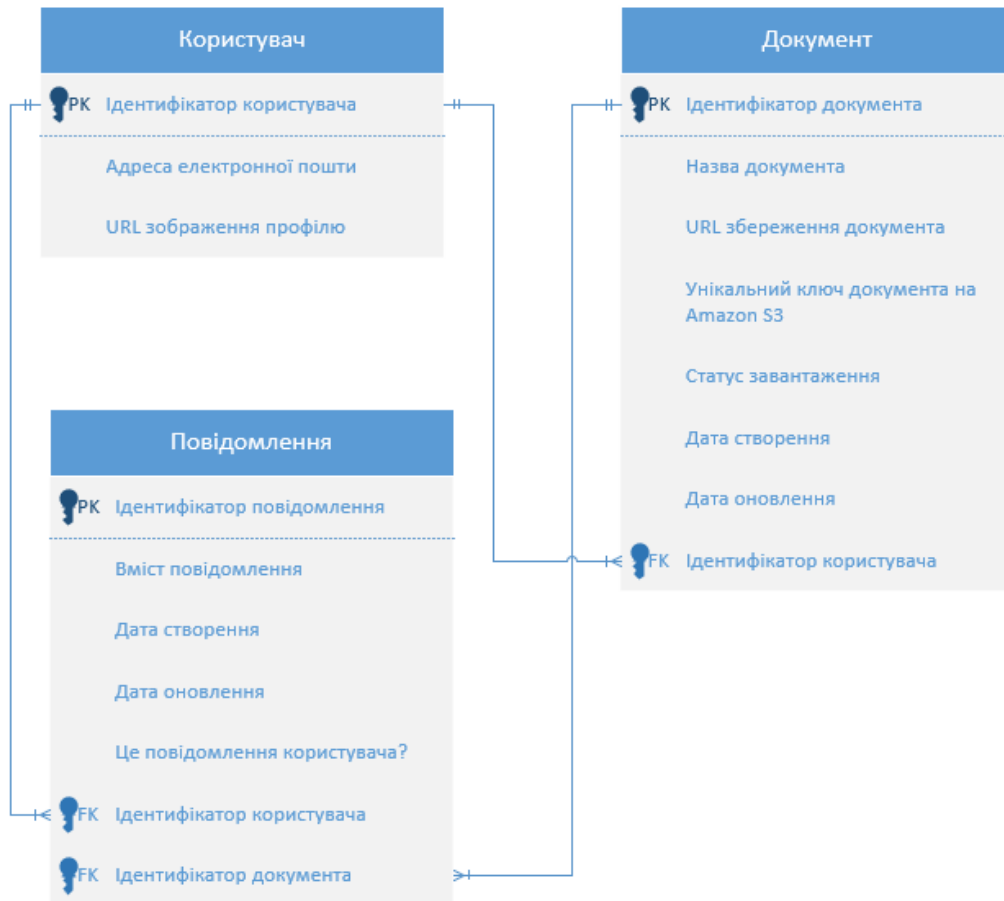


Fig. 2. Conceptual database model

The logical modeling stage involves transforming the abstract entities and relationships of the conceptual model into a clear and detailed structure that can be implemented in a specific database management system (DBMS).

In this logical model, three tables are created:

- 1) User:
 - id: unique user identifier - primary key;
 - email: email address - unique key;
 - imageUrl: The URL of the profile image.
- 2) File:
 - id: unique file identifier - primary key;
 - name: file name;
 - url: file URL;
 - key: unique Amazon S3 key;
 - uploadStatus: upload status;
 - createdAt: creation date;
 - updatedAt: update date;
 - userId: foreign key to the User table.
- 3) Message:
 - id: unique message identifier - primary key;
 - text: text content of the message;
 - isUserMessage: is it a user message?
 - createdAt: creation date;

- updatedAt: update date;
- userId: foreign key to the User table;
- fileId: foreign key to the File table.

Relationships between tables:

- one-to-many: a user can have many files (userId in File refers to id in User);
- one-to-many: a file can have many messages (fileId in Message refers to id in File);
- one-to-many: a user can create many messages (userId in Message refers to id in User).

Data types used in the database:

- varchar: used for short text strings;
- enum: used for enumerated values;
- datetime: used to represent dates and times;
- text: used for long text strings;
- boolean: used for logical values.

Figure 3 shows the logical model of the database.

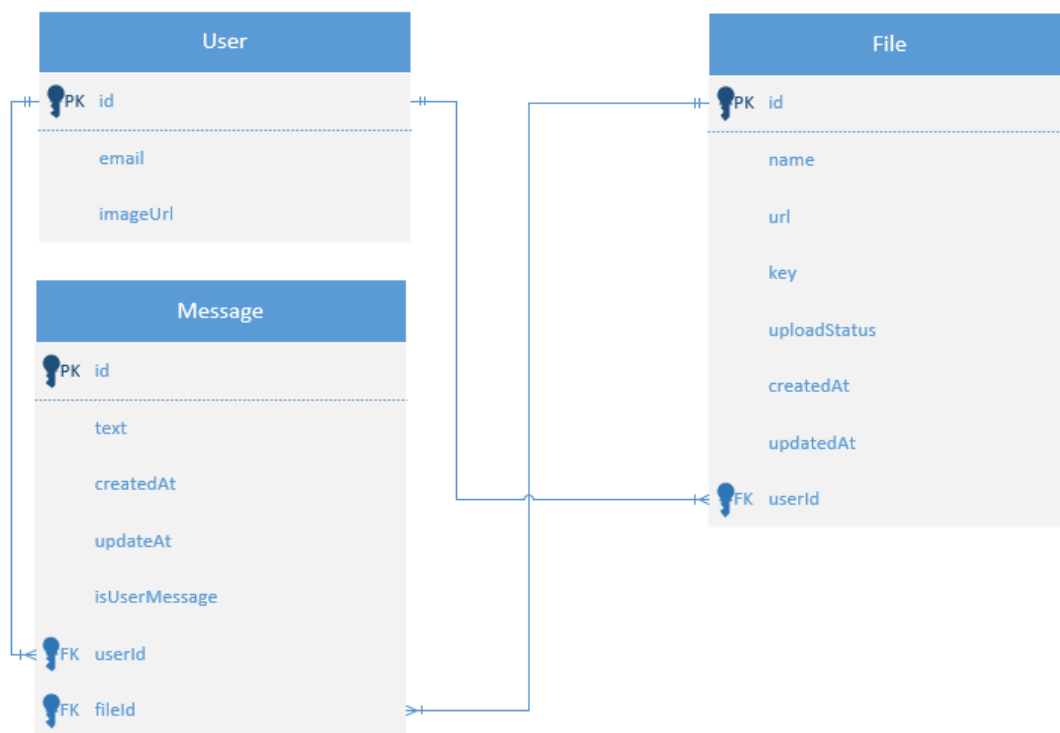


Fig. 3. Logical database model

The logical database model described above provides a detailed and realistic structure for the web service. It defines the tables, attributes, their data types, and the relationships between them. The model serves as the basis for the subsequent physical design, where the specific technical details of the database implementation in the chosen DBMS will be defined.

The next step is the physical database design, where the specific implementation details will be defined. The logical database model described earlier serves as the basis for the physical design, where the specific technical implementation details in the chosen DBMS will be defined. This project uses Prisma, a database tool in TypeScript. Using Prisma with physical database design provides several benefits. Abstracting from the database, Prisma allows you to interact with the database independently of the specific DBMS, simplifying development and maintenance. Automatic code generation, generates code for data access, CRUD (Create Read Update Delete) operations, and

interaction with relationships, which saves development time. Type safety, helps prevent errors when working with data.

Using Prisma, you can create tables according to your logical model. Prisma automatically generates code to implement the relationships between tables described in the schema. The code makes it easy to access and update related data. Figure 4 shows the created database in the DBeaver GUI.

User	
id	varchar(191)
email	varchar(191)
imageUrl	varchar(191)

Message	
id	varchar(191)
text	text
userId	varchar(191)
fileId	varchar(191)
isUserMessage	tinyint(1)
createdAt	datetime(3)
updatedAt	datetime(3)

File	
id	varchar(191)
name	varchar(191)
url	varchar(191)
key	varchar(191)
uploadStatus	enum('PENDING','PROCESSING','FAILED','SUCCESS')
createdAt	datetime(3)
updatedAt	datetime(3)
userId	varchar(191)

Fig. 4. Database in the DBeaver GUI

Physical database design using Prisma ensures efficient implementation of the logical model. Choosing a database, configuring Prisma, and using Prisma Client allow you to create a flexible and scalable database for your web service.

6. Designing a web service architecture. A well-defined, multi-tiered architecture is crucial for creating a maintainable, scalable, and secure web service. The text document analysis service will use a multi-tiered approach, separating functionality into separate modules with well-defined interfaces. This approach promotes modularity, simplifies development, and facilitates future maintenance.

The outer layer, the presentation layer, serves as the user interface. Built using the Next.js web framework, this layer is responsible for rendering user interface elements and handling user interactions. The Next.js framework offers server-side rendering capabilities, which improves initial page load time and SEO optimization. The presentation layer will display features such as a landing page with a description of the features, user authentication options, a document upload interface, and a chat window for interacting with the GPT-4 model. This layer will interact with the business logic layer via an API interface to perform user actions and retrieve the necessary data to render the user interface.

The business logic layer acts as the brain of the web service, managing the core functionality. Implemented in the TypeScript programming language, this layer receives user requests from the presentation layer and organizes interactions with other layers. After a user logs in, the business logic layer validates credentials and retrieves information about the user. It enforces authorization rules, ensuring that users can only access the documents and features they have downloaded based on their permissions. When a user downloads a PDF document, the business logic layer interacts with the data access layer to store the document metadata in a MySQL database. It then initiates the process of uploading the document to Amazon S3 cloud storage, obtaining the URL where the document is stored. After the document is uploaded, the business logic layer interacts with the OpenAI API to transform the document content into a semantic vector using Embeddings technology. This vector captures the document's values in a high-dimensional space. It then interacts with the service layer to store the semantic vector in the Pinecone vector database for use in semantic search. When a user initiates a chat with a GPT-4 model, the business logic layer captures the user's queries and stores them in a database. It transforms the user's query into a semantic vector and uses semantic search in the Pinecone database to find matching documents based on their semantic similarity. Finally, it retrieves the matching document context and passes it along with the user's query to the GPT-4 model via the service layer. The GPT-4 model generates a response based on the provided context.

The data access layer serves as the link between the business logic layer and the presentation layer. It handles all interactions with the MySQL database, including storing and retrieving user information, document metadata, and user queries. This layer ensures data integrity by enforcing data types, constraints, and properly managing database connections.

The service layer serves as an abstraction layer for interacting with external services such as OpenAI and Pinecone. It encapsulates the communication logic for these services, hiding key implementation details from the business logic layer. This facilitates loose coupling and simplifies future changes or integration with alternative services. The service layer uses the corresponding APIs provided by OpenAI and Pinecone to perform tasks such as generating document embeddings and storing/retrieving semantic vectors.

Thanks to this multi-tiered architecture, the web service achieves modularity, maintainability, and a clear separation of responsibilities. Each tier focuses on its specific responsibility, resulting in a more robust and scalable system.

A multi-tier architecture provides the foundation for a web service, with separate layers responsible for specific functions. To provide a connection between the user interface and the business logic layer, the service uses a well-defined application programming interface. The API acts as a contract, defining how external applications can interact with the service's functionality. There are several approaches to developing APIs, each with its own strengths and weaknesses. The most common include: RESTful, SOAP (Simple Object Access Protocol), and GraphQL APIs.

RESTful APIs follow the Representational State Transfer architectural style. REST defines a set of recommendations for how requests are made, namely using HTTP verbs such as GET, POST, PUT, DELETE, and how responses are formatted, most often in JSON format. RESTful APIs are resource-oriented, meaning they treat data as resources identified by URIs. This approach aligns well with web services, where documents and user data can be considered resources.

SOAP APIs use XML markup language for both requests and response messages. They offer a more structured approach compared to REST, but can be more complex to implement and less performant.

GraphQL APIs allow clients to request specific data fields from the server, potentially reducing the amount of data transferred. However, they can be more complex to implement and more secure.

When choosing an API design for a text document analysis web service, a RESTful API is the most suitable option due to a number of advantages:

- standardization, RESTful APIs adhere to an established set of principles, which makes them easier to understand and integrate with various client applications;
- resource-oriented design, consistent with the functionality of the service, where documents and user data act as resources that can be manipulated using CRUD operations;
- ease of use, RESTful APIs are generally considered easier to develop and understand compared to SOAP or GraphQL, making them a good choice for this project.

However, implementing a traditional RESTful API from scratch can be cumbersome, especially when dealing with complex data structures and ensuring type safety throughout the development process. While REST defines a common structure and communication style, it does not enforce data types or offer automatic code generation. This can lead to errors during development and requires manual effort to synchronize user interface code and business logic.

To address this issue, the tRPC library was developed. tRPC is based on RESTful principles, offering additional features that simplify development and improve API experience. tRPC specifically addresses the shortcomings mentioned above [18].

tRPC applies type definitions to both request and response data. This acts as a safeguard against runtime errors caused by incorrect data types being passed between the frontend and backend. It acts as a built-in mechanism to detect potential problems early in the development process. The library automates the generation of client code based on server-side API definitions. This eliminates

the need for developers to write and maintain boilerplate code for both sides, saving time and reducing the risk of inconsistencies between frontend and backend implementations.

Functionality is represented as procedures. These procedures directly reflect the functionality implemented at the business logic level. This provides a clear and granular way for client applications to interact with the backend, making the API easier to understand and use.

In summary, tRPC acts as a bridge between the chosen RESTful API style and the actual service implementation. It takes advantage of RESTful API benefits such as standardization and resource orientation, while offering additional features for more robust and developer-friendly development. By providing type safety, code generation, and a procedural approach, tRPC helps create a well-structured and maintainable API.

After defining the core technologies and libraries that will be used to develop the web service, the next step is to integrate with external services such as OpenAI API and Pinecone. These services play an important role in the functioning of the web service, providing the ability to analyze text documents and store semantic vectors accordingly.

The `openai` library is used to interact with the OpenAI API. The library provides a simple interface for using the GPT-4 model, allowing you to make API requests, receive responses, and handle errors. It also allows you to use Embeddings technology to transform the text of a document into a semantic vector used for semantic search.

Interaction with Pinecone occurs through the `pinecone-database` library. Pinecone is a vector database that allows you to store and search semantic vectors. The `pinecone-database` library provides a simple interface for interacting with Pinecone, allowing you to store semantic vectors, perform semantic searches, and retrieve results.

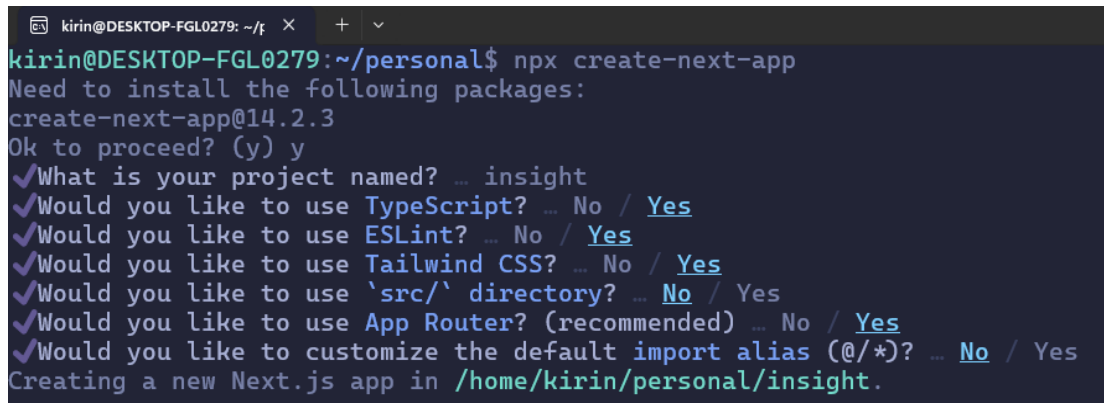
Integration with these external services occurs at the business logic level. When a user uploads a document, the document content is converted into a semantic vector using Embeddings via the OpenAI API. This vector is then stored in Pinecone using the `pinecone-database` library [17].

When a user initiates a chat with a GPT-4 model, the user's query is also converted into a semantic vector, and matching documents are found using semantic search in Pinecone. The information is then passed to the GPT-4 model as context for generating a response.

Thus, integration with OpenAI API and Pinecone is an important part of the web service architecture. It allows you to use the powerful capabilities of these services to analyze text documents and interact with users based on this analysis.

Therefore, the web service should have a multi-tiered architecture that ensures modularity, scalability, security, and ease of maintenance. RESTful API with tRPC library is used for clear and concise interaction between frontend and backend. Integration with OpenAI API and Pinecone allows you to analyze text documents, generate semantic vectors, and use them for semantic search and chat with the GPT-4 model.

7. Software Implementation. The first step in software implementation is to configure the Next.js application. To create the Next.js application, the `npx create-next-app` command-line tool is used (Figure 5).



```

kirin@DESKTOP-FGL0279: ~/personal$ npx create-next-app
Need to install the following packages:
create-next-app@14.2.3
Ok to proceed? (y) y
✓What is your project named? ... insight
✓Would you like to use TypeScript? ... No / Yes
✓Would you like to use ESLint? ... No / Yes
✓Would you like to use Tailwind CSS? ... No / Yes
✓Would you like to use `src/` directory? ... No / Yes
✓Would you like to use App Router? (recommended) ... No / Yes
✓Would you like to customize the default import alias (@/*)? ... No / Yes
Creating a new Next.js app in /home/kirin/personal/insight.

```

Fig. 5. Initializing the Next.js application using the command line tool

The `npx create-next-app` command starts an interactive process for creating a new Next.js project. It prompts you to choose a project name and define some settings, such as using TypeScript, ESLint, Tailwind CSS, a directory structure with a `src` directory, configuring App Router routing, and importing using aliases.

After answering the questions, a directory with the project name is created and all necessary dependencies are installed (Figure 6). This tool allows you to quickly start working on a new Next.js application with pre-configured parameters, which significantly simplifies the initial development stage. In particular, setting up the environment includes configuring Babel for JavaScript code transformation, configuring Webpack for module processing and code optimization, as well as preparing to work with CSS and other static resources.

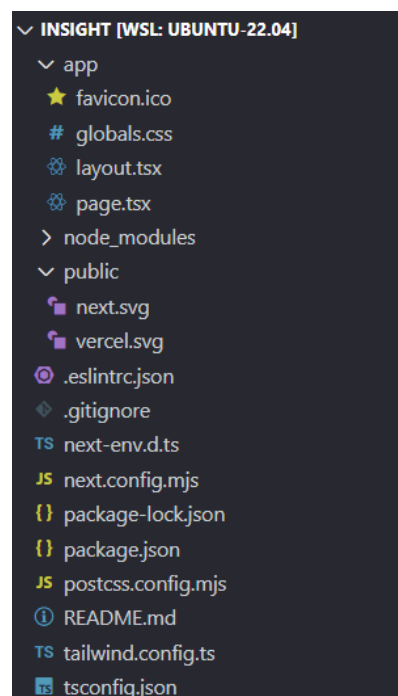


Fig. 6. Structure of a standard Next.js application

To test the Next.js application, you can run it in developer mode using the `npm run dev` script. By default, the application runs on a local server using port 3000 (Figure 7). In developer mode, the Next.js application is automatically reloaded whenever the code changes. This is achieved using the hot-reloading feature, which significantly improves development efficiency, as developers do not need to manually restart the server or refresh the page in the browser after making changes to the code.

```

o kirin@DESKTOP-FGL0279:~/personal/insight$ npm run dev

> insight@0.1.0 dev
> next dev

  ▲ Next.js 14.2.3
  - Local:      http://localhost:3000

✓ Starting...
Attention: Next.js now collects completely anonymous telemetry regarding usage.
This information is used to shape Next.js' roadmap and prioritize features.
You can learn more, including how to opt-out if you'd not like to participate in this anonymous program, by visiting the following URL:
https://nextjs.org/telemetry

✓ Ready in 2.2s
o Compiling / ...
✓ Compiled / in 3.9s (532 modules)
GET / 200 in 4946ms
✓ Compiled in 986ms (250 modules)
o Compiling /favicon.ico ...
✓ Compiled /favicon.ico in 4.1s (303 modules)
GET /favicon.ico 200 in 4189ms

```

Fig. 7. Running the application in developer mode

The application becomes available for viewing in a browser at <http://localhost:3000>, where you can see the results of your work in real time (Figure 8). This allows developers to easily track and test the changes they make, providing a more interactive and user-friendly development process.

It is important to note that Developer Mode is intended for development and testing purposes only. It is not optimized for performance or security, and is therefore not suitable for use in real-world environments.

To prepare your Next.js application for deployment in a production environment, you use the `npm run build` command. This command starts the build process, which includes minifying and optimizing your code, creating optimized versions of assets such as images and CSS, and generating static files if your application uses static site generation. After the build process is complete, the compiled application will be placed in the `out` folder, ready to be deployed to the server.

This approach ensures maximum application performance and reliability, as all resources will be optimized for fast loading and efficient operation under real-world load conditions. In addition, Next.js supports various deployment strategies, including deployment on platforms without server code, which adds additional flexibility and scalability for different types of web applications.

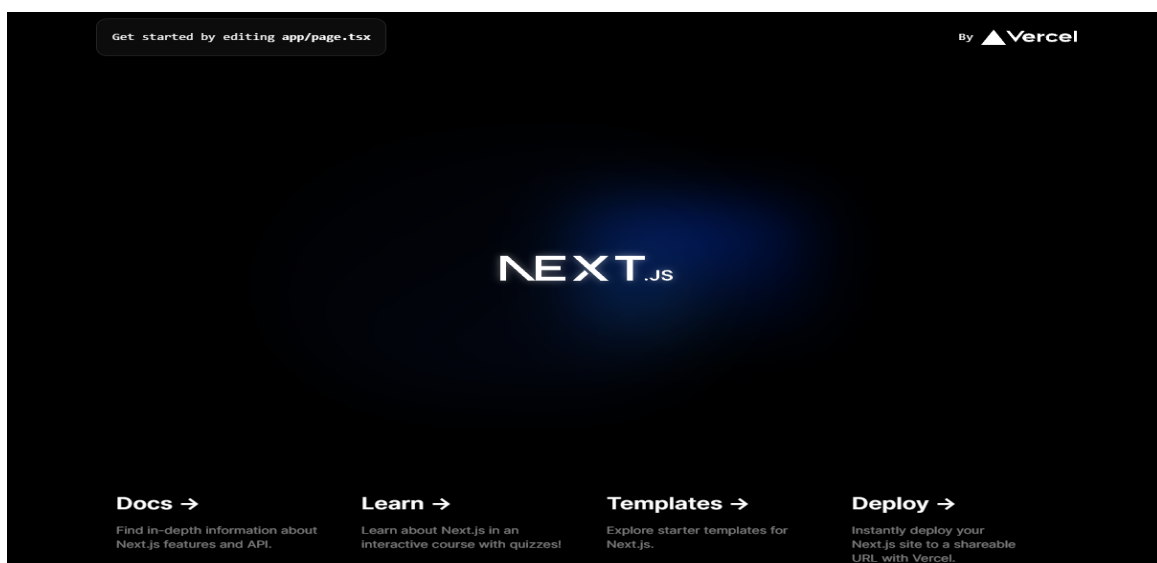


Fig. 8. Home page of a web application created using the `create-next-app` utility

The next step is to implement a web service user authorization system. Authorization is the process of determining whether a person is who they claim to be. After verification, the user gains access to a secure system. The process of connecting authorization through the Kinde service consisted of several stages that ensured the integration of security mechanisms and settings for managing user access to the web application. The unique Client ID and Client Secret identifiers obtained after registering the Kinde application, as well as additional environment variables, are added to the project's local environment file - .env.

The second step is to configure the OAuth 2.0 protocol used for authorization. In this context, it is necessary to define authorization parameters such as Authorization Endpoint and Token Endpoint. Also, configure the redirect URL that is needed to return the user to the web application after successful authentication (Figure 9).

Applications
SNAU AI

Quick start

Details

Authentication

Tokens

APIs

Callback URLs

Application homepage URI

http://localhost:3000

The homepage link to your application. E.g. https://app.yourapp.com. Opens on logo-click and for other back link actions.

Application login URI

http://localhost:3000

The default login route for resolving session issues. Allows users to redirect back to the application sign-in page via the /auth endpoint. E.g. https://app.yourapp.com/api/auth/login

Allowed callback URLs

http://localhost:3000/api/auth/kinde_callback

Add each URL on a new line

These are the redirect URIs where the user can be sent after authenticating

Allowed logout redirect URLs

http://localhost:3000

Add each URL on a new line

These are the URLs where the user can be sent after signing out

Fig. 9. *Configuring the OAuth 2.0 protocol*

The next step is to integrate with the web application. This includes adding the necessary libraries and modules to work with OAuth 2.0 (Figure 10), as well as implementing the authentication logic, creating a login and logout page, and configuring access token handling and refresh.

To implement the authentication logic, appropriate route handlers are created in the web application. The login handler redirects the user to the Kinde authorization page. This is achieved by generating a URL with the appropriate parameters.

```

● kirin@DESKTOP-FGL0279:~/personal/diplom$ npm install @kinde-oss/kinde-auth-nextjs
changed 6 packages, and audited 598 packages in 8s

149 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
○ kirin@DESKTOP-FGL0279:~/personal/diplom$

```

Fig. 10. Installing the @kinde-oss/kinde-auth-nextjs library

Login handler code listing:

```

import { NextRequest } from "next/server";
import { handleAuth } from "@kinde-oss/kinde-auth-nextjs/server";
export async function GET(request: NextRequest, { params }: any) {
  const endpoint = params.kindeAuth;
  return handleAuth(request, endpoint);
}

```

The redirect handler processes the response from Kinde after successful authentication. This handler receives the authorization code that was used to obtain access and refresh tokens via the Token Endpoint. The redirect handler code listing is:

```

export const appRouter = router({
  authCallback: publicProcedure.query(async () => {
    const { getUser } = getKindeServerSession();
    const user = getUser();
    if (!user?.id || !user?.email)
      throw new TRPCError({ code: "UNAUTHORIZED" });
    const dbUser = await db.user.findFirst({
      where: {
        id: user.id,
      },
    });
    if (!dbUser) {
      await db.user.create({
        data: {
          id: user.id,
          email: user.email,
        },
      });
    }
    return { success: true };
  }),
});

```


To protect certain routes of a web application, it is necessary to check the presence and validity of an access token. This allows you to limit access to resources only to authorized users. In the absence of a valid token, the user is redirected to the login page. Code listing:

```
trpc.authCallback.useQuery(undefined, {
  onSuccess: ({ success }) => {
    if (success) {
      router.push(origin ? `${origin}` : "/dashboard");
    }
  },
  onError: (error) => {
    if (error.data?.code === "UNAUTHORIZED") {
      router.push("/sign-in");
    }
  },
});
```

To log in and register in the system, the `LoginLink` and `RegisterLink` components from the `kinde-oss/kinde-auth-nextjs` library are used. Code listing for login and registration:

```
<div className="flex gap-3 items-center">
  <LoginLink className={buttonVariants({ variant: "ghost", size: "sm" })}>
    {i18n("navbar.auth.sign-in")}
  </LoginLink>
  <RegisterLink className={buttonVariants({ size: "sm" })}>
    {i18n("navbar.auth.sign-up")}
  </RegisterLink>
</div>
```

Figures 11 and 12 show authorization to the system using the Kinde service.




Ласкаво просимо!

Введіть свою адресу електронної пошти, і ми надішлемо вам код для входу.

Електронна пошта

[Продовжити](#)

Або

 [Продовжити з Google](#)

Немає облікового запису? [Створити](#)

Fig. 11. Authorization page

Thus, the process of connecting authorization through the Kinde service consists of sequential stages, each of which is aimed at ensuring reliable and secure integration of authorization mechanisms into the web application.

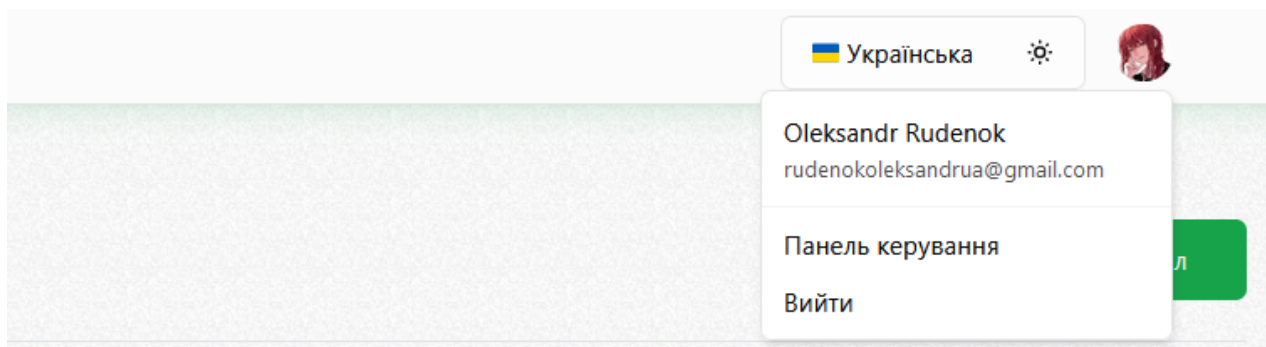


Fig. 12. Result of successful authorization

The next step is to implement a file upload system. But first, you need to set up the Pinecone vector database. First, you need to register on their website and create an account. After that, the service provides access to the management interface and API keys. After registration, you need to create a new index in Pinecone. The index defines the structure in which the data vectors will be stored and processed (Figure 13).

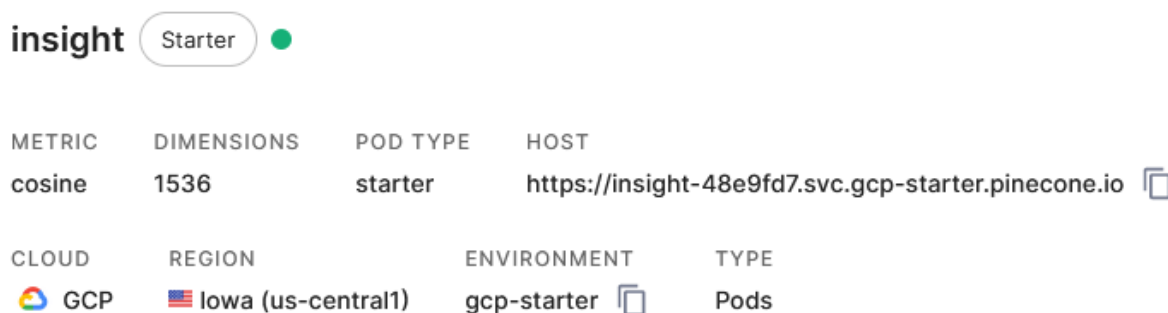


Fig. 13. Pinecone vector database index

To connect to Pinecone, SDKs or libraries are usually used that provide a convenient interface for working with vectors. For example, the pinecone-database/pinecone library, supported by the JavaScript and TypeScript programming languages, allows you to integrate Pinecone with your application directly via the API. The code listing for integrating Pinecone with a web service:

```
import { Pinecone } from "@pinecone-database/pinecone";
export const pinecone = new Pinecone({
  apiKey: process.env.PINECONE_API_KEY!,
  environment: "gcp-starter",
});
export const pineconeIndex = pinecone.Index("insight");
```

The PDF document upload process consists of two parts: client and server. The client part is implemented using the react-dropzone library. The code listing for the handleDrop function that handles the upload process in the UploadFile component is given below, and the full code listing for the UploadFile component is given in Appendix B.


```
async function handleDrop(file: File[]) {
  setIsUploading(true);
  const progressInterval = simulateProgress();
  const res = await startUpload(file);
  if (!res) {
    setIsUploadingError(true);
    return toast({
      title: i18n("toast-error.title"),
      description: i18n("toast-error.description"),
      variant: "destructive",
    });
  }
  const [fileResponse] = res;
  const { key } = fileResponse;
  if (!key) {
    setIsUploadingError(true);
    return toast({
      title: i18n("toast-error.title"),
      description: i18n("toast-error.description"),
      variant: "destructive",
    });
  }
  clearInterval(progressInterval);
  setUploadProgress(100);
  polling({ key });
}
```

The `handleDrop` function is an asynchronous function that performs the process of uploading a file and also handles any errors that may occur during this process. It consists of several steps, each of which is responsible for a specific aspect of uploading and processing files. Below is a detailed description of each step of this function.

The function calls `setIsUploading(true)` to set the "uploading" state, which can be used to display an upload indicator to the user.

The `simulateProgress()` function is called, which creates an interval to simulate the progress of the file being uploaded. This step is necessary to improve the user experience by displaying a visual progress indicator.

The asynchronous function `startUpload(file)` is called, which initiates the process of uploading the file to the server. It waits for a response from the server, which is stored in the variable `res`.

If the response from the `res` server is undefined or erroneous, an upload error state is set using `setIsUploadingError(true)` and an error message is displayed via the `toast()` function.

If the response from the server contains information about the file, the file key `key` is extracted from it. If the key is missing, an error state is set and an appropriate error message is displayed. After successfully receiving the file key, the interval that was responsible for simulating the download progress is stopped. The download progress is set to 100%, indicating the completion of the download process. The polling function is called, which starts polling the server to obtain the status of the downloaded file, using the received file key.

Figure 14 shows the PDF document file download window with a download indicator showing the current file download progress.

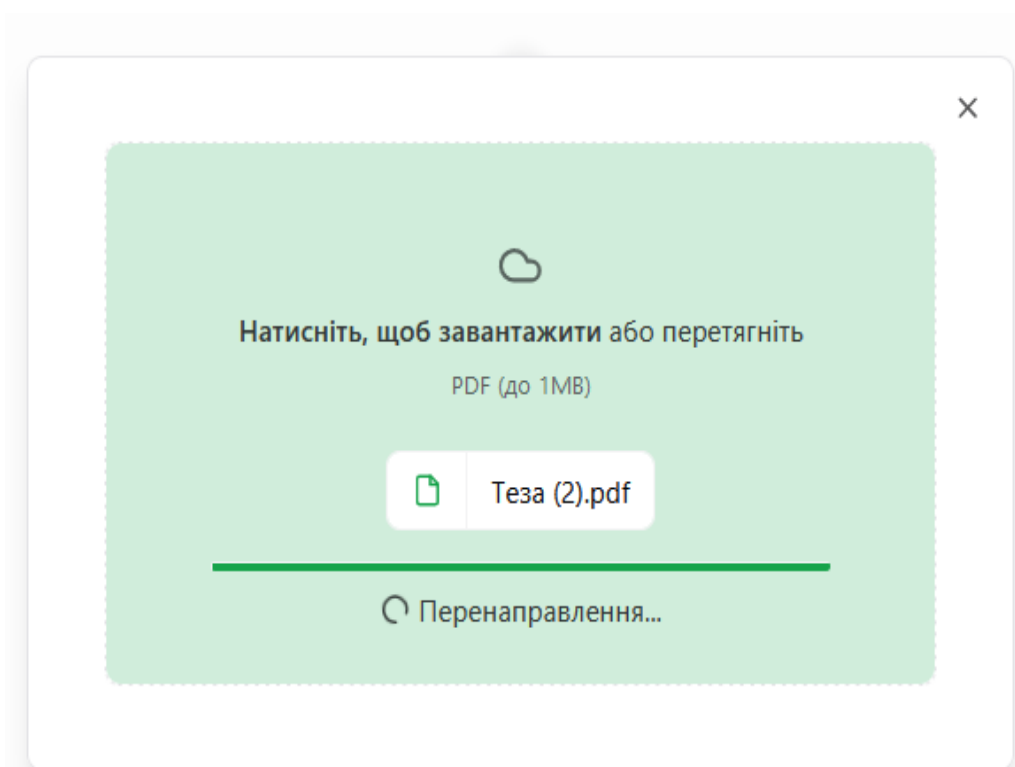


Fig. 14. PDF document download process

This function provides a complete file upload cycle, including informing the user about progress and handling possible errors, making it an important part of the file upload mechanism in a web application.

On the backend, a file router object `ourFileRouter` is created, which defines a route for downloading PDF files with a file size limit of 2 MB. The middleware performs user authorization checks using the Kinde session. If the user is not authorized, an "Unauthorized" error is raised.

After the upload is complete, it is checked whether the file already exists in the database. If the file exists, nothing is done. It is also checked whether the user has exceeded the 2 file limit. If the limit is exceeded, an error is thrown. If the upload is successful, the file is added to the database with the status "PROCESSING".

The file is downloaded from an S3 bucket and loaded as a PDF using `PDFLoader`. Each page of the PDF file is processed separately, creating an array of documents. Each document is tagged with metadata and stored in a Pinecone vector database using `OpenAIEmbeddings`. Upon successful storage, the file status is updated to "SUCCESS". In case of an error, the status is updated to "FAILED".

Thus, this code provides a comprehensive process for loading, processing, and storing PDF files, ensuring security and user data management. A complete listing of the PDF document loading code is provided in Appendix B.

By opening the vector database index on the Pinecone website, you can observe two semantic vectors that were generated from a successfully downloaded PDF document (Figure 15) [16].

1	ID	VALUES
	469ae519-24...	-0.00479211472, 0.000525163254, 0.0118783638, -0.0297408234, 0.00484394, 0.00926636718, -0.0300172251,
	SCORE	-0.0247
2	METADATA	
	fileId:	"clw1z3qb30001z7zwi4jlev1k"
	text:	"МІНІМІЗАЦІЯ ЗАТРИМОК У ВЕБДОДАТКАХ ВИКОРИСТОВУЮЧИ ГРАНИЧНІ ТЕХНОЛОГІЇ\n\nРуденок О. А., студ. 2 с.т. курсу ІСТ
2	ID	VALUES
	71c920e8-e1a...	-0.00479211472, 0.000525163254, 0.0118783638, -0.0297408234, 0.00484394, 0.00926636718, -0.0300172251,
	SCORE	-0.0247
2	METADATA	
	fileId:	"clwavv4hs000128cnlus4h8w9"
	text:	"МІНІМІЗАЦІЯ ЗАТРИМОК У ВЕБДОДАТКАХ ВИКОРИСТОВУЮЧИ ГРАНИЧНІ ТЕХНОЛОГІЇ\n\nРуденок О. А., студ. 2 с.т. курсу ІСТ

Fig. 15. Semantic vectors generated from a downloaded PDF document

A semantic vector consists of a unique identifier, a numeric value, and metadata, which in turn have a unique identifier for the PDF document from which the vector was created and the text content of that document.

The next stage of web service development is the implementation of user message processing and providing interactive responses. The code listing for user message processing:

```
const results = await vectorStore.similaritySearch(message, 1, {
  fileId: file.id,
});
const prevMessages = await db.message.findMany({
  where: { fileId },
  orderBy: { createdAt: "asc" },
  take: 5,
});
const formattedPrevMessages = prevMessages.map((msg) => ({
  role: msg.isUserMessage ? "user" : "assistant",
  content: msg.text,
}));
const context = `PREVIOUS CONVERSATION:${formattedPrevMessages.map(
  (message) => {
    if (msg.role === "user") return `User:${msg.content}\n`;
    return `Assistant:${msg.content}\n`;
  }
)}CONTEXT:${results.map((r) => r.pageContent)
.join("\n\n")}USER INPUT:${message}`;
const response = await openai.chat.completions.create({
  // model: "gpt-3.5-turbo-0125",
  model: "gpt-4o",
  temperature: 0.5,
  stream: true,
  messages: [
    {
      role: "system",
```

```

    content: "Your name is SNAU AI. Use the following context snippets (or the previous
conversation, if necessary) to answer the user's question in markup format. Answer in the language
of the user's question.",
  },
  {
    role: "user",
    content: context,
  },
],
});

```

```

const stream = OpenAIStream(response, {
  async onCompletion(completion) {
    await db.message.create({
      data: { text: completion, isUserMessage: false, fileId, userId },
    });
  },
});
return new StreamingTextResponse(stream);
} catch (error) {
  console.error("Error searching for similar messages:", error);
  return new Response("InternalServerError", { status: 500 });
}

```

This code processes POST requests to add user messages to the database, performs a similarity search for vectors in the file using Pinecone, and generates a response using a GPT-4 model based on previous conversations and user-entered text (Figure 16).

The software implementation of the web service includes the development and integration of the main modules, in particular algorithms for automatic text analysis, query processing mechanisms, and result generation.

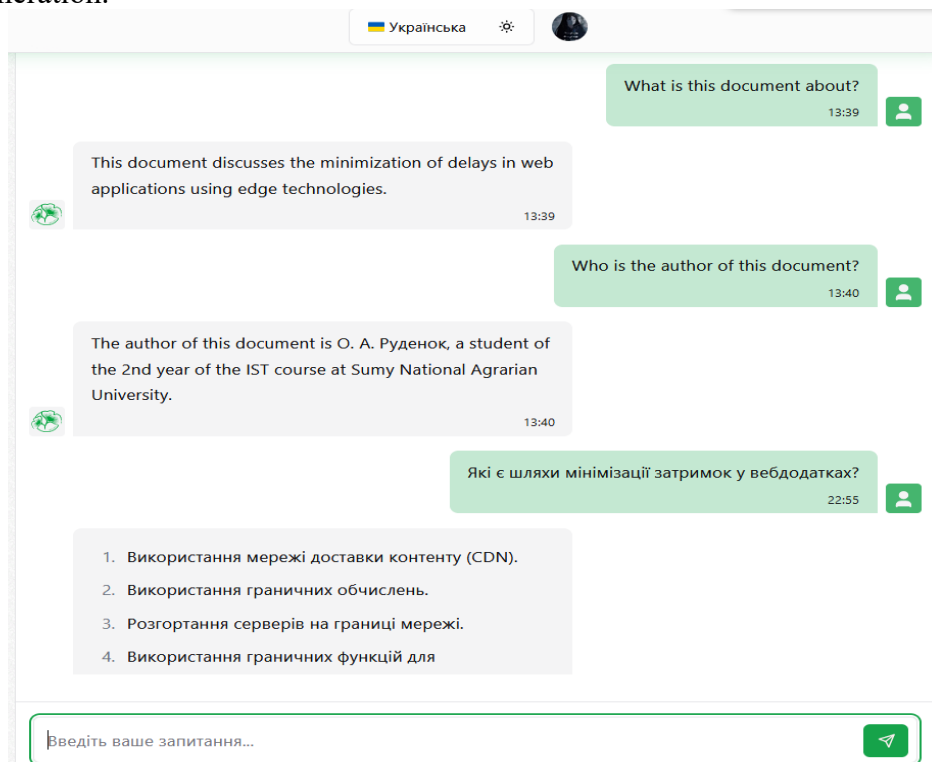


Fig. 16. Getting interactive answers to user questions regarding information in the uploaded file

Conclusions. As a result of the research, a functional web service was created for automated analysis of text documents, which can be used in various applied tasks related to the processing and analysis of text information. The developed web service is ready for implementation in distance learning systems to assist students in analyzing their electronic lectures and notes. This will help improve the quality of the educational process and facilitate learning activities. Also, to increase the efficiency and expand the capabilities of the web service, it is advisable to expand the functionality by adding modules for more detailed text analysis, such as emotion detection, automatic generalization and classification of text data. In addition, optimization of algorithms and the use of more powerful hardware will significantly increase the speed of data processing. Integration with other software products and services can provide greater compatibility and expand the possibilities of using the web service, which is an important aspect for its further development.

REFERENCES

1. Onyshchenko K., Daniel Ya., Kamenev R. Analysis of Natural Language Processing Methods. Information Systems and Technologies IST-2020: International Scientific Conference, Koblevo, September 12, 2020. Kharkiv, 2020. pp. 186–190.
2. Tkachenko K., Zuenko O. Using a multilayer LSTM neural network in the process of recognizing printed texts. Digital platform: information technologies in the socio-cultural sphere. 2022. Vol. 5, No. 1.
3. Amazon S3 - Cloud Object Storage - AWS. Amazon Web Services, Inc. URL: <https://aws.amazon.com/s3/> (access date: 05/13/2024).
4. Analyze insights in text with Amazon Comprehend. Amazon Web Service. URL: <https://aws.amazon.com/getting-started/hands-on/analyze-sentiment-comprehend/> (accessed: 07.05.2024).
5. Azure Blob Storage. Microsoft Azure. URL: <https://azure.microsoft.com/en-us/products/storage/blobs> (accessed: 12.05.2024).
6. Cloud Storage. Google Cloud. URL: <https://cloud.google.com/storage> (accessed: 13.05.2024).
7. GPT-4. OpenAI. URL: <https://openai.com/index/gpt-4-research/> (accessed: 07.05.2024).
8. Horev R. BERT Explained: State of the art language model for NLP. Towards Data Science. URL: <https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270> (accessed: 10.05.2024).
9. Li H. Deep learning for natural language processing: advantages and challenges. National Science Review. 2018. Vol. 5, No. 1. P. 24–26.
10. Next.js by Vercel - The React Framework. Next.js by Vercel - The React Framework. URL: <https://nextjs.org> (accessed: 10.05.2024).
11. Nuxt: The Intuitive Vue Framework. Nuxt. URL: <https://nuxt.com/> (accessed: 12.05.2024).
12. Rezaeenour J., Ahmadi M., Jelodar H. Systematic review of content analysis algorithms based on deep neural networks. Springer Link. URL: <https://link.springer.com/article/10.1007/s11042-022-14043-z> (access date: 05/07/2024).
13. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions / L. Alzubaidi et al. Journal of Big Data. URL: <https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00444-8> (accessed: 07.05.2024).
14. Suissa O., Elmalech A., Zhitomirsky-Geffet M. Text analysis using deep neural networks in digital humanities and information science. ResearchGate. URL: https://www.researchgate.net/publication/352904410_Text_analysis_using_deep_neural_networks_in_digital_hummanities_and_information_science (accessed 05/07/2024).
15. SvelteKit • Web development, streamlined. SvelteKit • Web development, streamlined. URL: <https://kit.svelte.dev> (date of access: 10.05.2024).
16. Text Analytics. Lexalytics. URL: <https://www.lexalytics.com/technology/text-analytics/> (accessed: 07.05.2024).
17. The vector database to build knowledgeable AI | Pinecone. The vector database to build knowledgeable AI | Pinecone. URL: <https://www.pinecone.io/> (access date: 05/18/2024).
18. tRPC - Move Fast and Break Nothing. End-to-end typesafe APIs made easy. | tRPC. tRPC - Move Fast and Break Nothing. End-to-end typesafe APIs made easy. | tRPC. URL: <https://trpc.io/> (access date: 05/17/2024).
19. Tripathi R. What are Vector Embeddings. Pinecone. URL: <https://www.pinecone.io/learn/vector-embeddings/> (accessed: 07.05.2024).