BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION COMPUTER SCIENCE

DIPLOMA THESIS

Information Extraction from Romanian Documents

Supervisor
Lect. Dr. Lupea Mihaiela Ana

Author
Mocanu Dragoș Marius

2025

UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

LUCRARE DE LICENŢĂ

Extragerea Informațiilor din Documente în Limba Română

Conducător științific
Lect. Dr. Lupea Mihaiela Ana

Absolvent
Mocanu Dragoș Marius

2025

Abstract

This paper presents an application for extracting meaningful information from Romanian texts. It uses different NLP techniques, such as keyword extraction with RAKE, TextRank and dependency relations, all adapted and implemented specifically for Romanian language.

Using a layered architecture, the application supports document upload or text insertion and it provides keyword extraction, NER, relation analysis, knowledge graph construction, and question answering, by implementing different Romanian-specific challenges, including lemmatization, stopword filtering, and diacritic normalization.

All components have been implemented from scratch, designed for Romanian. The originality lies in the system's capacity to combine statistical and linguistic methods for a complex analysis task.

The application was tested on different types of Romanian documents, with results showing the strengths and weaknesses of each algorithm. For example, RAKE provides fast results, while TextRank delivers more semantically relevant keywords. The dependency graph and question-answering module provides a deeper understanding of those texts.

Overall, this work is a valuable tool for under-resourced languages such as Romanian. The application supports educational use cases, offering a clear, extensible interface for document analysis.

This work is the result of my own activity, and I confirm I have neither given, nor received unauthorized assistance for this work

Contents

Chapter 1

Introduction

Nowadays, the volume of Romanian documents is growing exponentially, making it more difficult for most people to efficiently extract relevant information. Automatic text analysis tools have become essential for facilitating the better understanding of complex texts, especially in languages such as Romanian, where NLP tools are underdevolped. This thesis presents an application designed to analyze documents written in Romanian, extracting keywords, their relationships and answering some key questions, making the texts easier to understand. [12].

1.1 Motivation

Romanian language is often overlooked in NLP research due to its complex grammar and the fact that it is not widely used, tools for automatic keyword extraction are usually created for English. As a Romance language, Romanian has high morphological variability, requiring custom lemmatization and normalization for successful processing.

Motivated by these things, my work addresses the need for:

- Easily accessible keyword and concept extraction tools for Romanian documents.
- An interface that helps people in understanding Romanian texts more efficiently.
- Structured information from unstructured text.

1.2 Problem Statement

Despite the availability of numerous keyword extraction tools, most are made for languages like English and aren't very good for Romanian. In particular, very few tools handle Romanian morphology, syntax, or diacritics properly. Moreover, existing methods often rely solely on word frequency, TF–IDF scores, or position-based heuristics [17], frequently overlooking the syntactic and grammatical relationships between words that are crucial in morphologically rich languages like Romanian.

This project focuses exclusively on Romanian texts and implementing a NLP application that integrates multiple components for deeper document understanding. It combines RAKE [16] and TextRank [11] to ensure reliable keyword extraction and supplements them with named entity recognition and knowledge graph construction. In addition, the application is able to answer questions such as WHO, WHAT, WHEN, WHERE and WHY.

1.3 Related Work

Keyword extraction has evolved considerably since the introduction of basic statistical models like TF–IDF [17]. The 2004 TextRank algorithm by Rada Mihalcea and Paul Tarau [11] introduced a graph-based approach to ranking words by importance. RAKE, published in 2010 [16], remains one of the most widely-used statistical co-occurrence methods due to its simplicity and domain-independence.

Recent reviews such as [3, 4, 13] highlight a wide array of hybrid, supervised, and deep learning-based alternatives, while guides such as [20] compare practical implementation scenarios. However, few studies focus on morphologically rich, low-resource languages like Romanian. For Romanian specifically, ensemble models for dependency parsing [7] and dedicated NLP resources [12, 15] have begun to appear only recently.

Named entity recognition in Romanian doesn't have the best accuracy. Romanian-specific datasets such as RoLargeSum [2] and benchmarking efforts like LiRo [14] are the foundation for building more robust NLP applications.

Thus, this thesis is part of a growing effort to bring modern NLP capabilities to underrepresented languages through the adaptation and integration of proven methods.

1.4 Outline

The next chapters offers detailed information about the created tool for Romanian texts. Chapter 2 offers a theoretical foundation of the NLP techniques used.

It details preprocessing methods, such as tokenization and lemmatization, along with the principles behind keyword extraction using RAKE and TextRank, syntactic parsing, and knowledge graph construction. Chapter 3 provides a description of the application's core functionalities. It presents how each module processes the input text and contributes to generating structured insights. Chapter 4 focuses on implementation specifics. It describes the system's architecture, backend NLP pipeline, frontend interface, and the design of the modules. Chapter 5 illustrates how the application performs on a real input example. It displays and analyzes the outputs of each module, including keyword candidates, recognized entities, relations, and extracted answers. Chapter 6 concludes the thesis with a summary of contributions, an overview of encountered challenges, and directions for future enhancements.

I declare that I used ChatGPT to rephrase some parts of the text, in order to make the text more readable.

Chapter 2

Theoretical Aspects

This chapter presents the theoretical aspects of NLP techniques, used for extracting semantic and syntactic information from text. It focuses on core methods such as text preprocessing, keyword extraction, dependency parsing, and the construction of knowledge graphs. The challenges of processing morphologically rich languages such as Romanian have been increasingly addressed in recent research [12, 15].

2.1 NLP and Preprocessing Foundations

Natural Language Processing is a subfield of artificial intelligence and computational linguistics, dealing with the automated analysis and understanding of human language. It integrates techniques from linguistics, computer science, and statistics to model textual data. Preprocessing is the foundational stage of any NLP system, aiming to standardize and simplify raw text before advanced processing is applied.

2.1.1 Tokenization

Tokenization is the process of segmenting text into smaller units called tokens. Tokens are typically words, punctuation marks, or meaningful subwords. Let T denote a raw text string. Tokenization is a function:

$$\text{tokenize} : T \rightarrow \{w_1, w_2, ..., w_n\}$$

where w_i represents the i-th token extracted from the input.

Consider the Romanian sentence:

„Inteligența artificială influențează societatea modernă."

After tokenization, the result would be:

$$\{\texttt{Inteligența}, \texttt{artificială}, \texttt{influențează}, \texttt{societatea}, \texttt{modernă}\}$$

2.1.2 Stopword Removal

Stopwords are high-frequency words that don't have a significant meaning. These include words such as prepositions, articles, and conjunctions. In Romanian, examples include „și", „în", „pe", „de", and „cu". Stopword removal is applied to reduce noise and computational complexity.

Let S be the set of stopwords, and W be the list of tokens. The operation:

$$\text{filter\_stopwords}(W, S) = \{w \in W \mid w \notin S\}$$

produces the filtered list. For example:

„Lucrare de licență"

After filtering the stopwords, the result would be:

$$\{\texttt{Lucrare}, \texttt{licență}\}$$

The preposition "de" was removed. Stopword filtering is typically implemented using curated lists such as those provided by NLTK [5] or spaCy [1].

2.1.3 Lemmatization

Lemmatization is the process of reducing inflected or derived words to their canonical form, called a lemma. This operation enhances semantic uniformity and improves keyword grouping by normalizing words to their dictionary form.

Let w be a token, then the lemmatization function is:

$$\text{lemma}(w) = l, \quad \text{where } l \text{ is the canonical form of } w$$

In Romanian, lemmatization must account for gender, case, number, and tense. For example:

Word	**Lemma**
„copiilor"	„copil"
„inteligenței"	„inteligență"
„modernă"	„modern"

Table 2.1: Examples of Romanian lemmatization

Accurate lemmatization is critical in morphologically complex languages, where inflected forms can significantly outnumber their lemmas. Neural-based morphological models and parsers adapted for Romanian have improved accuracy in this area [7].

2.1.4 Normalization and Diacritics

In Romanian, diacritics (ă, î, â, ș, ț) are essential for correct orthography. Many sources omit or misrepresent them, leading to inconsistencies in preprocessing. Text normalization is the process of standardizing text, typically including:

- Conversion to lowercase
- Unicode normalization
- Replacement of diacritics with standard forms (e.g., „ş" to „s")

Let f be a normalization function, then for any text T:

$$f(T) = \text{lowercase}(\text{normalize\_unicode}(T))$$

This step ensures that words with and without diacritics are treated uniformly. For instance, „Stiinta" and „Știința" should be normalized to the same token.

After establishing the core preprocessing techniques, we now turn our attention to keyword extraction methods. The first approach analyzed is RAKE, which relies on co-occurrence and statistical scoring.

2.2 RAKE

RAKE (Rapid Automatic Keyword Extraction) is an unsupervised, domain-independent algorithm used to extract key phrases from individual documents [16]. It does not require training data and relies on the statistical co-occurrence and frequency of words in a text. RAKE works in several steps: phrase delimitation, word co-occurrence analysis, scoring, and candidate ranking [19, 10].

2.2.1 Algorithm Overview

RAKE identifies candidate keywords by first eliminating stopwords and punctuation, then grouping the remaining words into candidate phrases. It constructs a

co-occurrence graph based on word adjacency, computes two values for each word — its degree and frequency — and assigns each candidate phrase a score.

Let:

- $f(w)$ = frequency of word w
- $d(w)$ = degree of word w (number of co-occurring words)
- $s(w) = \frac{d(w)}{f(w)}$ = score of word w

Then, the score of a candidate phrase $P = \{w_1, w_2, ..., w_n\}$ is:

$$\text{RAKE}(P) = \sum_{i=1}^{n} s(w_i)$$

2.2.2 Example of Phrase Formation in Romanian

Consider the following sentence:

„Educația în era digitală este importantă pentru școala modernă."

After removing stopwords such as *„în"*, *„este"*, and *„pentru"*, RAKE identifies the following candidate phrases:

$$\{\texttt{educația}, \texttt{era digitală}, \texttt{școala modernă}\}$$

2.2.3 Word Co-occurrence Matrix

A co-occurrence matrix C is used to calculate the degree $d(w)$ of each word. Let W be the set of all unique words from the candidate phrases:

$$W = \left\{ \texttt{educația, era, digitală, școala, modernă} \right\}$$

Then the co-occurrence matrix is:

	educația	era	digitală	școala	modernă
educația	0	0	0	0	0
era	0	0	1	0	0
digitală	0	1	0	0	0
școala	0	0	0	0	1
modernă	0	0	0	1	0

Table 2.2: Co-occurrence matrix for RAKE candidate words

From the matrix above:

- `educația` has degree 0, frequency 1 $\Rightarrow s = 0$
- `era`, `digitală`, `școala`, `modernă` each have degree 1, frequency 1 $\Rightarrow s = 1$

Phrase scores:

$$\text{RAKE}(\texttt{educația}) = 0$$

$$\text{RAKE}(\texttt{era digitală}) = 1 + 1 = 2$$

$$\text{RAKE}(\texttt{școala modernă}) = 1 + 1 = 2$$

2.2.4 Advantages and Limitations

RAKE is computationally efficient and requires no external resources, making it ideal for quick keyword extraction on short to medium-length documents. However, it has several limitations:

- It does not account for syntactic or semantic information.
- Candidate phrases are highly dependent on stopword selection.
- Performance degrades on noisy or unstructured text.

Despite these issues, RAKE remains widely used in information retrieval due to its simplicity and ease of adaptation to multiple languages, including Romanian.

While RAKE emphasizes simplicity and efficiency, TextRank offers a more sophisticated, graph-based method for identifying important keywords. The following section explores its principles and advantages.

2.3 TextRank

TextRank is an unsupervised, graph-based ranking model introduced by Rada Mihalcea and Paul Tarau [11], used extensively in natural language processing for keyword and sentence extraction. The algorithm is inspired by PageRank, the famous algorithm used by Google to rank web pages. TextRank constructs a graph of words based on co-occurrence and determines word importance through iterative scoring. Unlike RAKE, it incorporates graph connectivity structure and recursively propagates influence across nodes.

2.3.1 Algorithm Overview

TextRank operates by:

1. Preprocessing the text (POS Tagging, Tokenization, Lemmatization, Normalization).
2. Selecting candidate lemmas (usually for nouns and adjectives).
3. Creating a graph where each lemma is a node.
4. Adding edges between nodes if the lemmas co-occur within a sentence.
5. Initializing all node scores randomly and applying the iterative scoring formula until convergence. Convergence is obtained when, for all vertices, the difference between the scores of a vertex for 2 successive iterations is less than a threshold.

Let $G = (V, E)$ be an undirected graph:

- V is the set of lemmas.
- E is the set of edges between co-occurring lemmas.

The TextRank score $S(V_i)$ for each lemma V_i is computed iteratively:

$$S(V_i) = (1 - d) + d \cdot \sum_{V_j \in \text{In}(V_i)} \frac{w_{ji}}{\sum_{V_k \in \text{Out}(V_j)} w_{jk}} S(V_j)$$

where:

- d is the damping factor (typically 0.85)
- $\text{In}(V_i)$ are lemmas linking to V_i (i.e., nodes that have an edge pointing to V_i)
- $\text{Out}(V_j)$ are lemmas that V_j links to (i.e., nodes that receive edges from V_j)
- w_{ji} is the edge weight (e.g., co-occurrence frequency or 1 for unweighted)

2.3.2 Example in Romanian

Consider the sentence:

„Educația în era digitală este importantă pentru școala modernă.”

We keep only nouns and adjectives:

{educația, era, digitală, importantă, școala, modernă}

With a window size of 2, we create edges between adjacent terms. The resulting co-occurrence graph is:

- educația – era
- era – digitală
- digitală – importantă
- importantă – școala
- școala – modernă

2.3.3 Word Co-occurrnce Matrix

	educația	era	digitală	importantă	școala	modernă
educația	0	1	0	0	0	0
era	1	0	1	0	0	0
digitală	0	1	0	1	0	0
importantă	0	0	1	0	1	0
școala	0	0	0	1	0	1
modernă	0	0	0	0	1	0

Table 2.3: Co-occurrence matrix used by TextRank

After convergence, lemmas with the highest scores are selected. Multi-word phrases are formed by combining adjacent high-ranking lemmas in the original text. For this sentence, the final keywords might be:

{era digitală, școala modernă}

2.3.4 Advantages and Limitations

TextRank offers several advantages:

- Works without external resources or supervised training.
- Language-independent with proper preprocessing.

However, it also has limitations:

- Sensitive to window size and POS tagging accuracy.
- Slower than RAKE on large texts due to iterative updates.

Despite these problems, TextRank is a powerful and flexible approach, often outperforming simpler techniques in accuracy.

Keyword extraction alone does not capture the full syntactic structure of a sentence. To gain deeper linguistic insight, we next examine dependency parsing and grammatical relationships.

2.4 Dependency Parsing and Grammatical Relations

Dependency parsing is a syntactic analysis technique that establishes directed binary relations between words in a sentence. The output is a tree structure in which each word (except the root) is connected to a syntactic head via a dependency label. This method allows for the explicit modeling of grammatical roles such as subjects and objects, providing a structured and linguistically rich representation of a sentence [18, 7].

2.4.1 Fundamental Concepts

Let a sentence be composed of a sequence of tokens $T = \{t_1, t_2, \ldots, t_n\}$. A dependency parse can be defined as a set of directed edges:

$$D = \{(h_i, d_i, r_i) \mid h_i \in T \cup \{\text{ROOT}\}, d_i \in T, r_i \in \text{Rel}\}$$

where:

- h_i is the head token
- d_i is the dependent token
- r_i is the dependency relation label (e.g., `nsubj`, `dobj`)

Each word is the dependent of exactly one head, and the resulting structure is a rooted, directed, acyclic tree.

2.4.2 Types of Dependency Relations

Below are common dependency relation labels, with formal definitions and Romanian examples.

- **nsubj (nominal subject):** A noun that serves as the subject of a clause.

  Example: „**Copilul** citește o carte."
  Dependency: (`citește`, `copilul`, `nsubj`)

- **dobj (direct object):** A noun that is the object of a verb.

  Example: „Copilul citește **o carte**."
  Dependency: (`citește`, `carte`, `dobj`)

- **amod (adjectival modifier):** An adjective modifying a noun.

  Example: „Cartea **interesantă** este pe masă."
  Dependency: (`cartea`, `interesantă`, `amod`)

- **advmod (adverbial modifier):** An adverb modifying a verb or adjective.

  Example: „El aleargă **repede**."
  Dependency: (`aleargă`, `repede`, `advmod`)

- **obl (oblique nominal):** A non-core argument or modifier.

  Example: „Ea locuiește **în Cluj**."
  Dependency: (`locuiește`, `Cluj`, `obl`)

2.4.3 Dependency Tree Structure

A dependency tree can be formally represented as a directed graph $G = (T, D)$, where each token is a node, and the arcs correspond to labeled dependencies. The root of the tree corresponds to the main verb or predicate of the sentence.

For the sentence:

„Inteligența artificială transformă industria modernă."

the dependency tree includes arcs such as:

- (`transformă`, `inteligența`, `nsubj`)
- (`inteligența`, `artificială`, `amod`)
- (`transformă`, `industria`, `dobj`)
- (`industria`, `modernă`, `amod`)

This type of structure is especially useful in morphologically rich languages such as Romanian, where word order may vary, but syntactic functions are preserved [12, 15].

2.4.4 Advanced Perspectives

In more recent approaches, dependency parsing is performed using neural networks, transformers, and prompt-based models. The work of [8] introduces the Structuralized Prompt Template, which uses LLMs (Large Language Models) for syntactic parsing with minimal supervision. Similarly, surveys such as [6] provide a taxonomy of unsupervised dependency parsing models.

Building upon dependency structures, the next logical step involves converting syntactic relationships into semantic triplets. This process is fundamental for constructing knowledge graphs from text.

2.5 Knowledge Graphs and Semantic Triplets

2.5.1 Overview

A **knowledge graph (KG)** is a structured representation of information in which entities are represented as nodes and relationships between them as edges. In natural language processing, KGs are often constructed by extracting relational triplets from text, in the form:

$$(s, r, o)$$

where:

- s is the subject (head noun),
- r is the relation a verb),
- o is the object (dependent noun or phrase).

Example: „Profesorul explică lecția elevului.”
Triplet extracted: (`profesorul`, `explică`, `lecția`)

Modern Romanian NLP platforms such as RELATE [15] and benchmarks like LiRo [14] emphasize the need for syntactic structure in semantic representation, especially for under-resourced languages.

2.5.2 Challenges and Limitations

- **Coreference resolution:** Identifying referents like „el" or „ea" across sentences.
- **Negation:** Differentiating between „predă lecția" and „nu predă lecția".
- **Semantic ambiguity:** The verb „susține" may mean „to support" or „to deliver (a talk)".

To mitigate these challenges, KG construction pipelines often incorporate additional tools such as coreference resolvers, negation detectors, and semantic role labelers.

Chapter 3

Application

This chapter presents the practical implementation of the concepts introduced previously. It describes the main functionalities of the developed application, detailing how each module contributes to the extraction and structuring of information from Romanian texts.

3.1 Overview of Functionality

In figure 3.1 illustrates the application's workflow, emphasizing the core modules that will be discussed in the following chapter.

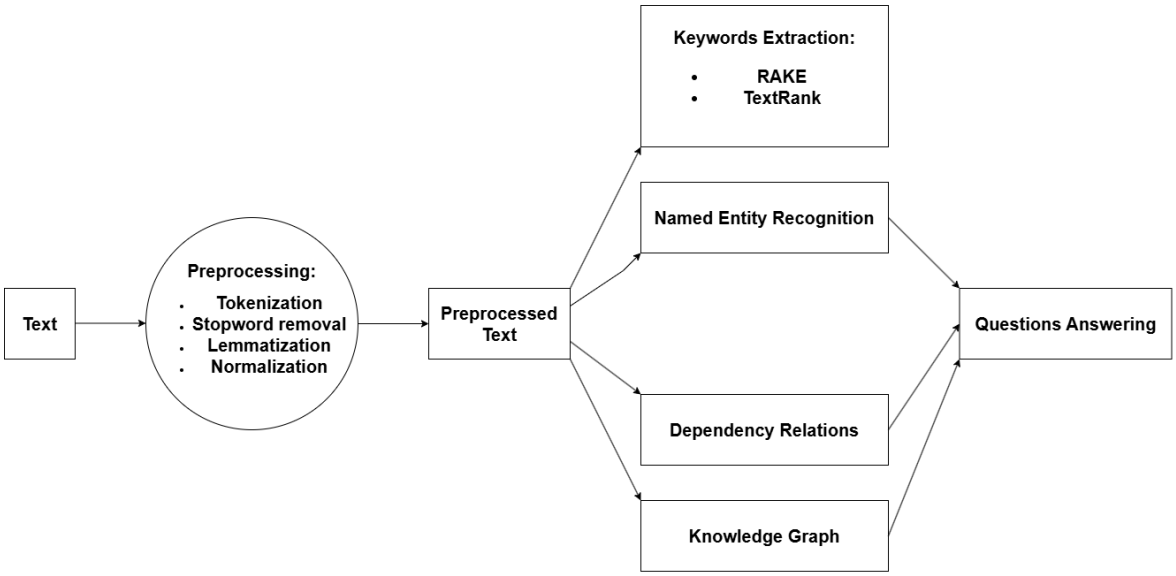

Figure 3.1: System Architecture

The application is designed to extract keywords and analyze syntactic structures from texts written in Romanian. It offers users the possibility to input data in three

ways:

- By writing or pasting text directly into a dedicated text box.
- By uploading a plain text file.
- By uploading a document in portable document format (PDF).

Once the input is received, the system automatically initiates the full analysis process. The text is first extracted and normalized, then passed through a series of natural language processing stages, which include stopword removal, lemmatization, keyword scoring, named entity recognition, dependency parsing, and relation extraction.

3.2 Text Intake and Preprocessing

The first component of the processing pipeline deals with extracting raw text from the input. If the user uploads a PDF file, the document is parsed page by page and converted into plain text. For TXT files, content is read line by line and concatenated into a full string.

After extraction, the system applies normalization steps. All characters are transformed to lowercase, and diacritics are harmonized to avoid inconsistencies (e.g., ”ş” is replaced with ”s”).

The clean text is then split into sentences and tokens. Each token is further analyzed for its grammatical category and morphological features. The tokens are matched against a stopword list. Those found to be stopwords are removed to reduce noise and improve the quality of keyword extraction.

Next, each token that remains is lemmatized, transformed into its base form. This allows different grammatical variations of the same concept to be grouped under a single representation. For instance, “copiilor”, “copilului”, and “copii” are all lemmatized to “copil”.

This preprocessing stage produces a normalized and semantically coherent version of the input text, prepared for scoring and syntactic analysis.

3.3 Keyword Extraction

Once the text is preprocessed and lemmatized, it is passed to two independent modules that identify and score keywords. These modules operate in parallel and return two separate lists. The first method is based on co-occurrence statistics, while the second employs a graph-based ranking strategy.

3.3.1 RAKE

The RAKE module works by isolating sequences of words that don't contain stopwords. These sequences are considered candidate phrases. Each phrase is splited into individual words and two values are calculated:

- **Frequency:** how many times the word appears across all candidate phrases.
- **Degree:** the sum of the number of words it co-occurs with in those phrases.

The score of each word is computed as the ratio between its degree and frequency. For a full phrase, the score is the sum of the scores of its component words. For example, if the phrase "inteligență artificială" contains words that appear together frequently and in multiple phrases, their score will be higher.

This score-based ranking promotes longer expressions that have more meaning. The application filters out very short phrases and those with extremely low scores. The final RAKE result is a list of high-scoring keyphrases, ordered from most to least relevant.

3.3.2 TextRank

In the TextRank module, the system starts by selecting the most semantically significant words, nouns and adjectives, and builds a co-occurrence graph. In this graph, each word is a node, and edges are created between nodes that appear close to one another in the text, within a fixed-size window.

The graph is undirected and unweighted at first, but edge weights are introduced later based on the frequency of co-occurrence. Then, a recursive scoring algorithm iteratively updates the importance score of each word based on the scores of its neighbors.

After convergence, each word has a final score. These are sorted, and top keywords are selected.

Both RAKE and TextRank produce distinct sets of keyphrases, offering different perspectives on the content: one based on frequency patterns, and the other on structural relationships between words.

3.4 Named Entity Recognition

After keywords are extracted, the system checks whether any of them match recognized named entities. This is done using a module that detects known types such as people, locations, organizations, or temporal expressions.

For each keyword, a comparison is made against the list of detected entities. If there is an exact match, the keyword is annotated with a semantic label such as `PER` for people, `LOC` for places, or `ORG` for institutions. These labels are then included in the final result, allowing the user to distinguish between generic terms and specific, real-world entities.

The application also displays these grouped by category, allowing the user to see all people, organizations, or places mentioned in a document. This separation enhances document navigation and makes proper nouns more visible in dense texts.

3.5 Syntactic Relation Extraction

In addition to keywords, the application identifies syntactic relations between words. This is done by analyzing each sentence's structure and extracting labeled relations between tokens.

For every pair of related words, the system notes:

- The **head word** (typically a verb or main noun).
- The **dependent word** (adjective, subject, object, etc.).
- The **relation type**, such as `amod` for adjectival modifier or `nsubj` for nominal subject.

These are grouped into a structured set of connections and then formatted as a graph. Each node represents a word, and each edge is labeled with the grammatical relation between the nodes. This allows users to see how keywords are connected in the sentence, revealing patterns like adjective-noun pairs or subject-verb-object structures.

The system displays this structure as a list of every identified relation. This visualization helps users understand the syntactic flow of the sentence and how key terms relate grammatically.

3.6 Question-Answer Component

To further facilitate comprehension, the application extracts answers to five specific types of questions:

- **Who?** – Derived from subject-verb pairs in the sentence.
- **What?** – Focused on main objects or actions performed.
- **Where?** – Based on named location entities or locative phrases.
- **When?** – Extracted from date expressions or temporal modifiers.
- **Why?** – Inferred from causal patterns, such as the use of „pentru că", „deoarece", or „fiindcă".

The system scans each sentence and matches these patterns to extract relevant short answers. For example, if a sentence contains a subject and a verb, it may form a `who–does what` type of answer. Temporal expressions and prepositional phrases are scanned for `where` and `when` answers.

All identified answers are organized into five groups and shown in a separate tab. This provides a quick summary of essential information and helps readers find what matters most without reading the full text.

3.7 Knowledge Graph Construction

To enhance document interpretability, the application includes a module for generating a knowledge graph. This graph is built from triplets extracted directly from the dependency structure of each sentence.

The application searches for structures where a subject and an object are both connected to the same verb. If a sentence contains, for example, a relation like:

„Profesorul explică lecția."

The system extracts the triplet:

(profesorul, explică, lecția)

Each triplet is interpreted as a subject–predicate–object unit and added to the knowledge graph. The graph is internally represented as a list of nodes (words or phrases) and edges (relations).

These relations are then transformed into a graph where:

- Nodes represent the entities or concepts (e.g., „profesorul", „lecția").
- Edges represent the relationship between them (e.g., „explică").

The resulting graph allows users to explore document meaning beyond keywords. It highlights what actions are performed, by whom, and upon what — forming a structured conceptual map of the content.

3.8 Final Display and Output Summary

Once all modules have completed processing, the system returns a full structured output. This includes:

- Extracted keywords from RAKE and TextRank.
- Named entity categories and their associated words.
- Dependency relations.
- Knowledge graph triplets.
- Answers to the five guiding questions.

All information is displayed in a clear, interactive interface where users can click tabs to switch views.

Chapter 4

Implementation Details

This chapter provides an overview of the technical implementation of the application. It covers the entire architecture, describing the roles of each module and presents the libraries and frameworks used. The application was implemented with modularity and efficiency, to support Romanian text processing.

4.1 System Architecture Overview

The application uses a client-server architecture, where the frontend serves as the user interface layer and the backend executes all computational linguistic analysis tasks. Communication between these layers is done asynchronously via HTTP requests using the JSON format. The backend exposes a REST API implemented with FastAPI, and the frontend consumes these APIs using asynchronous fetch calls. The system is modular, allowing future extensibility.

4.2 Backend

The backend is entirely implemented in Python 3.11 and structured across several modules. Each module handles a separate NLP component or service:

4.2.1 API Layer (FastAPI)

The entry point of the backend is implemented using **FastAPI**, a modern, async-ready Python web framework. FastAPI provides OpenAPI schema generation, form/file handling, and asynchronous request processing.

Middleware is added using `fastapi.middleware.cors.CORSMiddleware` to allow cross-origin communication from any domain.

4.2.2 Document Processing and Text Extraction

PDF file processing is performed using the **PyMuPDF** (`fitz`) library. This allows accurate text extraction from documents, preserving paragraph structure. TXT files are parsed directly via UTF-8 decoding. The module `pdf_processor.py` encapsulates PDF handling functionality.

4.2.3 Text Normalization Utilities

Text normalization is using Unicode normalization (via Python's `unicodedata` standard library) to remove diacritics and convert text to lowercase. It also standardizes tokens before scoring and matching.

4.2.4 Stopword Management

It is using the **NLTK** library. It also adds domain-specific terms like "etc", "ie", "fig", etc. This customized stopword list improves keyword extraction by eliminating non-informative tokens.

4.2.5 Keyword Extraction Algorithms

The application implements two independent keyword extraction algorithms, both coded from scratch for customization and control:

RAKE

- Uses custom logic to split text into phrases using stopwords as boundaries.
- Computes *degree-to-frequency* scores for each word and aggregates them per phrase.
- Ranking is done based on combined word scores.

The algorithm operates on preprocessed and lemmatized tokens and selects candidate phrases of up to 3 words that do not contain stopwords. It ensures that only meaningful content words are considered.

TextRank

- Based on a graph representation where nodes are tokens (nouns and adjectives).
- Implemented using **NumPy** for matrix operations.

- Co-occurrence matrix is built using a sliding window over the token sequence.
- The PageRank algorithm is applied iteratively until convergence using a sparse adjacency matrix.

Tokens are filtered by part-of-speech (POS). A co-occurrence graph is created, and each token receives a centrality score. The top k tokens are returned as keywords.

4.2.6 NER

Named Entity Recognition is performed using **spaCy** and the `ro_core_news_lg` model, which supports Romanian-specific entity labels such as:

- `PER` – Persons
- `LOC` – Locations
- `ORG` – Organizations
- `DATE`, `TIME`, `GPE`, etc.

4.2.7 Dependency Parsing and Relation Extraction

Dependency relations between words are extracted using spaCy's syntactic parser. Tokens are linked to their syntactic heads with relations such as `nsubj`, `dobj`, `amod`, etc.

4.2.8 Knowledge Graph Triplet Extraction

The application extracts semantic triples of the form `(subject, predicate, object)`.

All triplets are de-duplicated, normalized, and converted into a knowledge graph structure. The graph is built from scratch using Python dictionaries and serialized as JSON. Each node is a token, and each labeled edge represents a semantic relationship.

4.2.9 Question-Answer Pair Extraction

The system attempts to extract answers to the standard WH-questions — WHO, WHAT, WHERE, WHEN, and WHY — using a combination of syntactic and semantic clues:

- **WHO**: Identified using `nsubj` or `nsubjpass` dependencies linked to verbs.

- **WHAT**: Constructed from predicate-object pairs derived from extracted knowledge triples.
- **WHERE, WHEN**: Extracted using NER labels like `LOC`, `GPE`, `DATE`, and `TIME`, and by filtering syntactic roles like `obl`, `prep`.
- **WHY**: Detected using Romanian causal patterns such as "pentru că", "fiindcă", "deoarece", located through substring matching.

After extraction, each list of answers is deduplicated and filtered using string containment heuristics to retain only the most specific and relevant results.

4.2.10 Output Generation and Error Handling

Once all processing steps are complete, the backend constructs a structured JSON response.

If input validation fails, for instance, when the input is empty, the file format is unsupported, or decoding fails, the API returns a structured error message. These errors are handled by the frontend, which displays user-friendly alerts without breaking the UI state.

4.3 Frontend

The frontend is a single-page application (SPA) developed using **ReactJS**. It provides users with an interactive and responsive interface for submitting documents and visualizing all results returned by the backend.

The application communicates asynchronously with the backend via `fetch` POST requests, sending data that may include plain text or uploaded files.

4.3.1 Styling and Layout

Bootstrap 5 is used for all styling and layout components.

Core Bootstrap components used include:

- Tabs (for separating views such as keywords, graphs, entities)
- Cards (for displaying individual keyword items with scores)

- Alerts (for displaying errors and notifications)
- Buttons and Input Groups (for file uploads and manual text entry)

4.3.2 Input Handling

The main input area accepts both:

- Plain text, entered directly in a component.
- Uploaded documents (`.pdf`, `.txt`) using the native file picker.

Input is sent using multipart/form-data. Loading indicators are shown during backend processing to improve user experience.

4.3.3 Tabbed Output Interface

The frontend organizes results using tabs, each linked to a specific section of the backend response:

- **RAKE**: Displays a list of keyphrases.
- **TextRank**: Similar list of keyphrases generated via TextRank.
- **Named Entities (NER)**: Entities grouped by type.
- **Relations**: Displays a list containing all identified dependency relations.
- **Knowledge Graph (KG)**: Visualization of a graph, consisting of subject–predicate–object relations.
- **Q&A**: Lists of extracted WHO/WHAT/WHEN/WHERE/WHY answers.

4.3.4 Graph Visualization

The Knowledge Graph is visualized using a **force-directed layout**, implemented using a JavaScript graph rendering libraries such as **D3.js** or `react-force-graph`.

The graph is interactive and support:

- Zoom and pan functionality.
- Node dragging for rearrangement.
- Hovering nodes to see the exact words.

The graph is rendered based on the `nodes` and `links` fields returned from the backend, using unique IDs and labeled edges.

4.3.5 Responsiveness and Error Display

During API calls, a loading spinner or message is displayed to indicate ongoing processing. In case of errors, such as unsupported formats, empty inputs, or decoding issues, Bootstrap alerts notify the user with messages. These alerts are dismissible and do not interrupt the state of the application.

4.4 Design

4.4.1 Use Case Diagram

Figure 4.1 illustrates the main interactions between the user and the application.

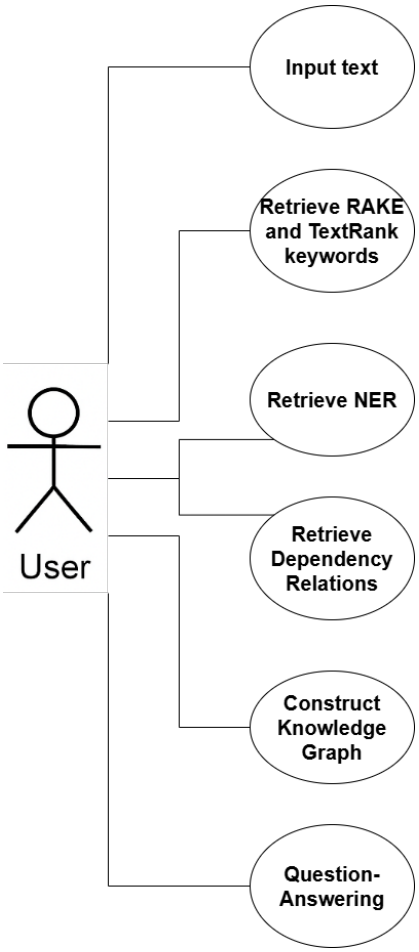

Figure 4.1: Use Case Diagram

4.4.2 Class Diagram

The class diagram of the application illustrates the relationships between classes.

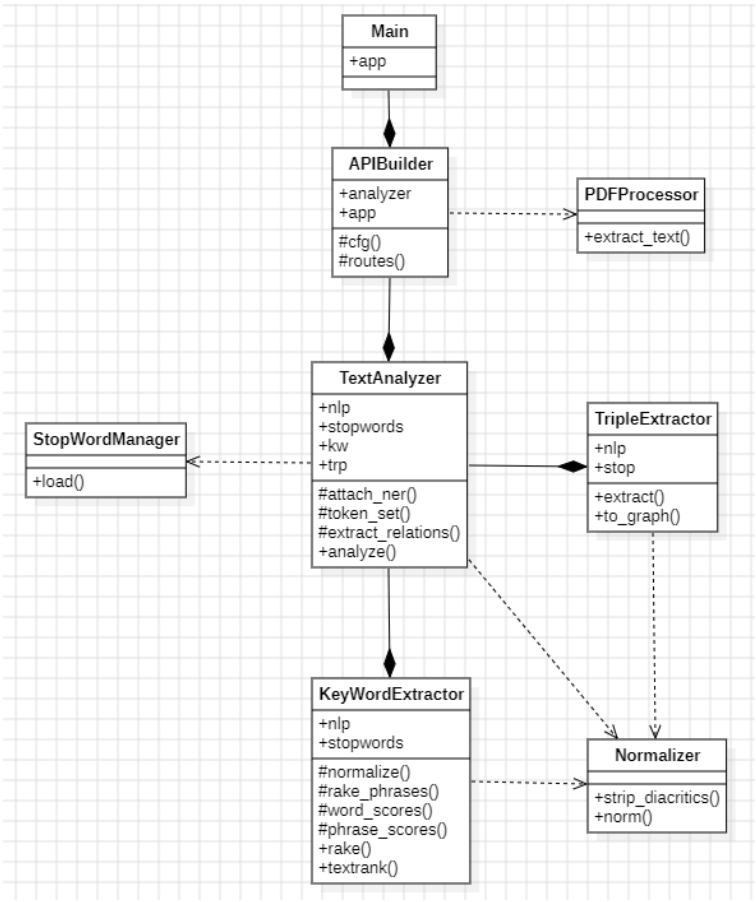

Figure 4.2: Class Diagram

4.4.3 Sequence Diagram

The sequence diagram shows how TextRank keywords are extracted from user input through system components.

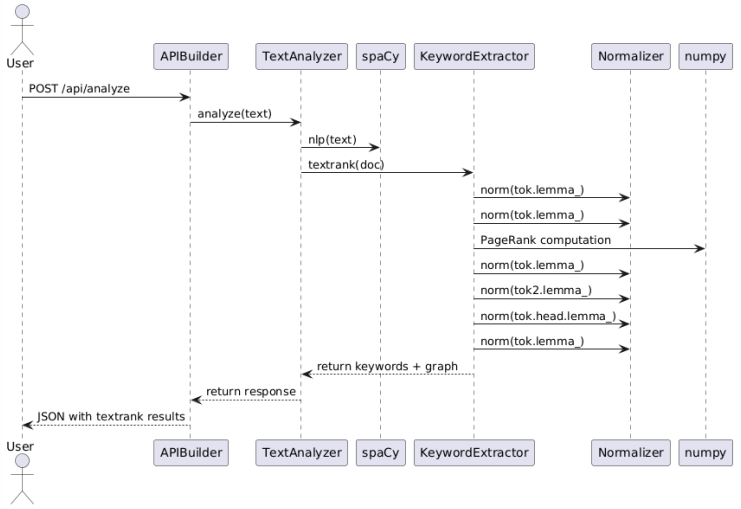

Figure 4.3: Sequence Diagram for TextRank

4.5 Architecture and Data Flow

The application maintains a clean and modular data flow between frontend and backend components. The sequential operations are:

1. The user provides input via direct text or document upload in the frontend.
2. The frontend sends the input as a `multipart/form-data` POST request to the backend API.
3. The backend performs the following in order:
   (a) Extracts text from PDF or decodes TXT.
   (b) Normalizes and tokenizes the text.
   (c) Performs keyword extraction using RAKE and TextRank.
   (d) Applies named entity recognition.
   (e) Parses dependency relations.
   (f) Extracts semantic triplets and builds the knowledge graph.
   (g) Generates structured answers to WHO/WHAT/WHERE/WHEN/WHY questions.
4. A comprehensive JSON response is returned.
5. The frontend parses this response and updates each tab accordingly.
6. The user explores the results.

This workflow ensures that the backend performs all computational tasks efficiently, while the frontend remains responsive and user-focused.

4.6 Testing and Validation

The system was manually tested across multiple scenarios:

- Uploading various PDF files.
- Verifying the tokenization and lemmatization.
- Verifying RAKE and TextRank outputs.
- Inspecting the KG to see its accuracy.
- Validating named entity grouping by category.
- Assessing WH-question answer relevance.

On the frontend, tests ensured:

- Proper display and handling of errors like unsupported file types.

Chapter 5

Evaluation and Results

This chapter presents the results obtained by the application on an example text, a short description of a project developed by Microsoft and Stanford.

5.1 Example

5.1.1 Input Text

Cercetătorii de la Microsoft și Universitatea Stanford au lansat în luna aprilie un sistem de recunoaștere vocală avansat.

Proiectul a fost desfășurat în laboratoarele din Seattle și București, folosind date colectate în anul 2023.

Sistemul a fost testat pe mai multe limbi naturale, iar acuratețea a crescut cu 25% față de versiunile anterioare.

Algoritmul a fost refăcut pentru a reduce latența în conversații, deoarece timpul de răspuns reprezintă un factor critic în aplicațiile vocale.

De asemenea, arhitectura modelului a fost optimizată pentru că procesarea se face pe dispozitive mobile, unde resursele hardware sunt limitate.

5.1.2 RAKE Output

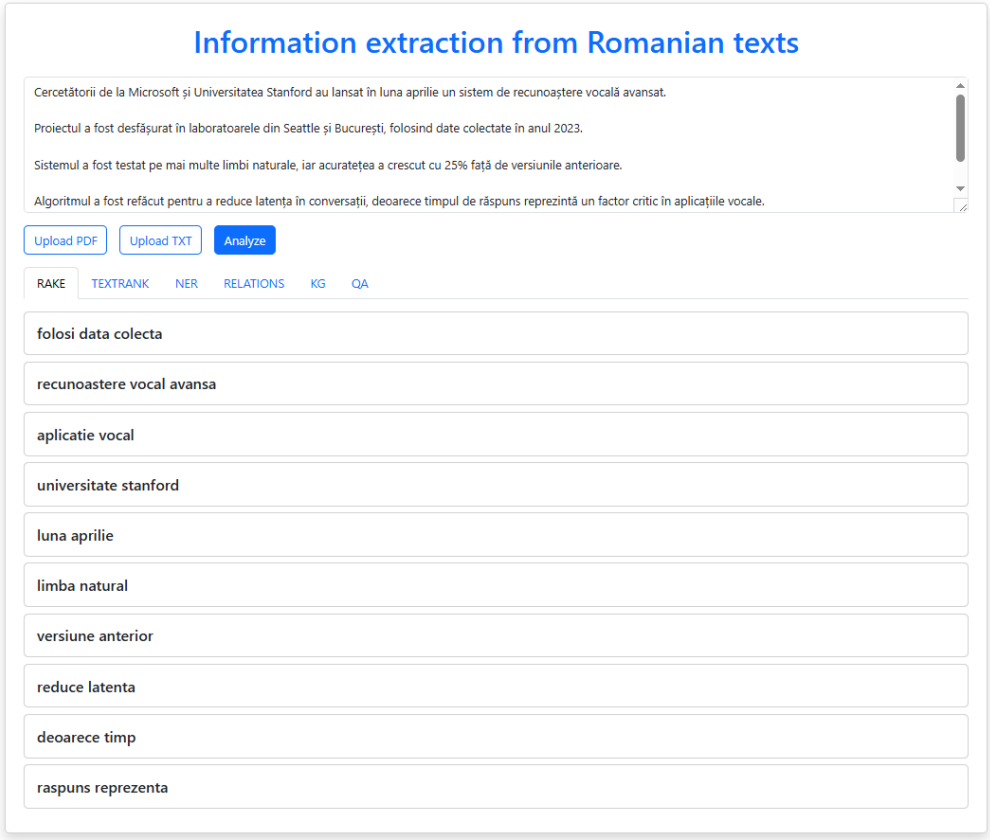

Figure 5.1: RAKE keyword extraction from source text

The RAKE algorithm extracted lemmatized key multi-word phrases. Examples include *folosi data colecta*, *recunoastere vocal avansa*, *aplicatie vocal*, and *universitate stanford*.

5.1.3 TextRank Output

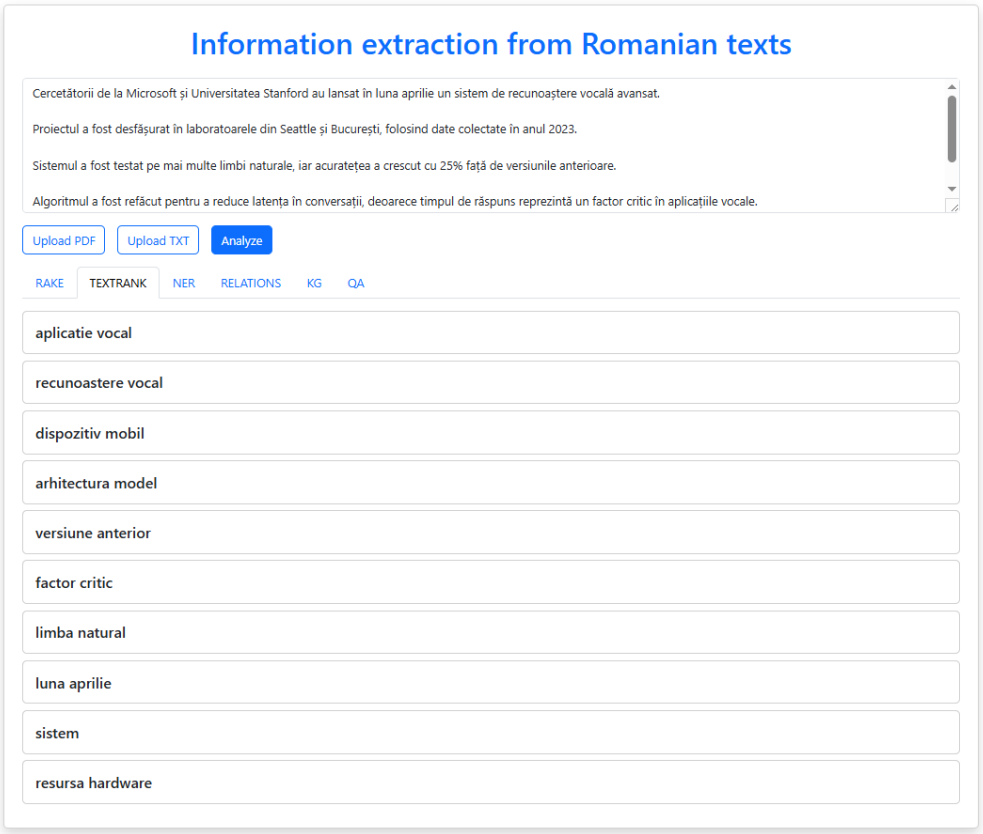

Figure 5.2: TextRank keyword extraction from source text

Unlike RAKE, which favors phrase-level co-occurrence, TextRank is based on graph centrality and prefers concepts that are well-connected across the sentence structure. TextRank's performance is improved by lemmatization, ensuring uniform scoring of word variants (e.g., *limbi* $\rightarrow$ *limba*).

The outputs of the two algorithms are partially overlapping: *recunoastere vocală*, *aplicatie vocală*, and *versiune anterior*, confirming the effectiveness of both algorithms.

5.1.4 NER Output

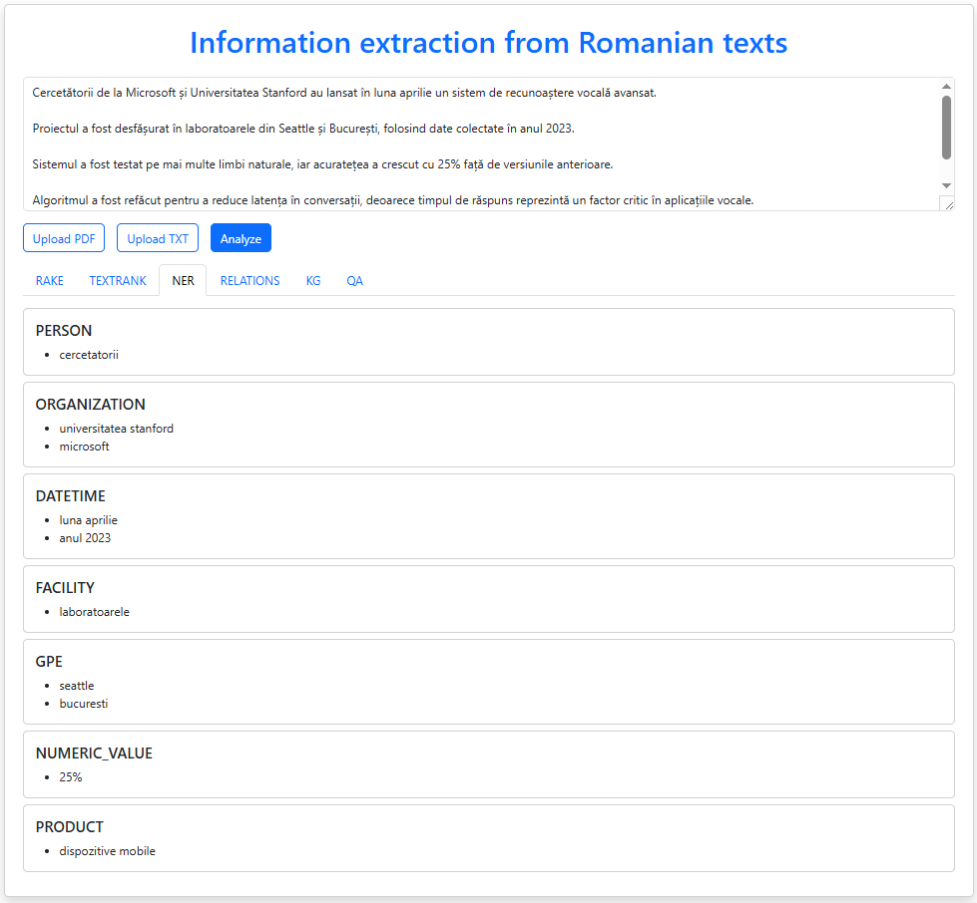

Figure 5.3: Named Entity Recognition results

The Named Entity Recognition module successfully identified and categorized the following entities:

- **PERSON:** *cercetătorii*
- **ORGANIZATION:** *universitatea stanford*, *microsoft*
- **DATETIME:** *luna aprilie*, *anul 2023*
- **FACILITY:** *laboratoarele*
- **GPE (Geo-Political Entity):** *seattle*, *București*
- **NUMERIC_VALUE:** *25%*
- **PRODUCT:** *dispozitive mobile*

5.1.5 Syntactic Relation Extraction

The application extracts dependency relations using spaCy's dependency parser. This information is useful for understanding structural links between words. Below is the list of relations identified in the input text:

Source	Label	Target
universitate	cc	si
cercetator	conj	universitate
universitate	nmod	stanford
luna	case	in
lansa	nmod:tmod	luna
luna	nmod	aprilie
sistem	det	un
lansa	obj	sistem
recunoastere	case	de
sistem	nmod	recunoastere
recunoastere	amod	vocal
recunoastere	amod	avansa
folosi	punct	,
desfasura	advcl	folosi
folosi	obj	data
data	acl	colecta
colecta	obl	an
testa	nsubj:pass	sistem
limba	case	pe
limba	det	mult
testa	obl	limba
limba	amod	natural
versiune	case	fata
25%	nmod	versiune
versiune	amod	anterior
reduce	mark	pentru
reduce	mark	a
reface	advcl	reduce
reduce	obj	latenta
reduce	obl	conversatie
reprezenta	punct	,
reprezenta	mark	deoarece
reprezenta	nsubj	timp
raspuns	case	de
timp	nmod	raspuns

Source	Label	Target
reface	advcl	reprezenta
factor	det	un
reprezenta	xcomp	factor
factor	amod	critic
aplicatie	case	in
factor	nmod	aplicatie
aplicatie	amod	vocal
optimiza	nsubj:pass	arhitectura
arhitectura	nmod	model
dispozitiv	case	pe
face	obl	dispozitiv
dispozitiv	amod	mobil
limita	nsubj	resursa
resursa	amod	hardware
dispozitiv	acl	limita

Table 5.1: Syntactic dependencies extracted from the input text

5.1.6 Knowledge Graph Visualization

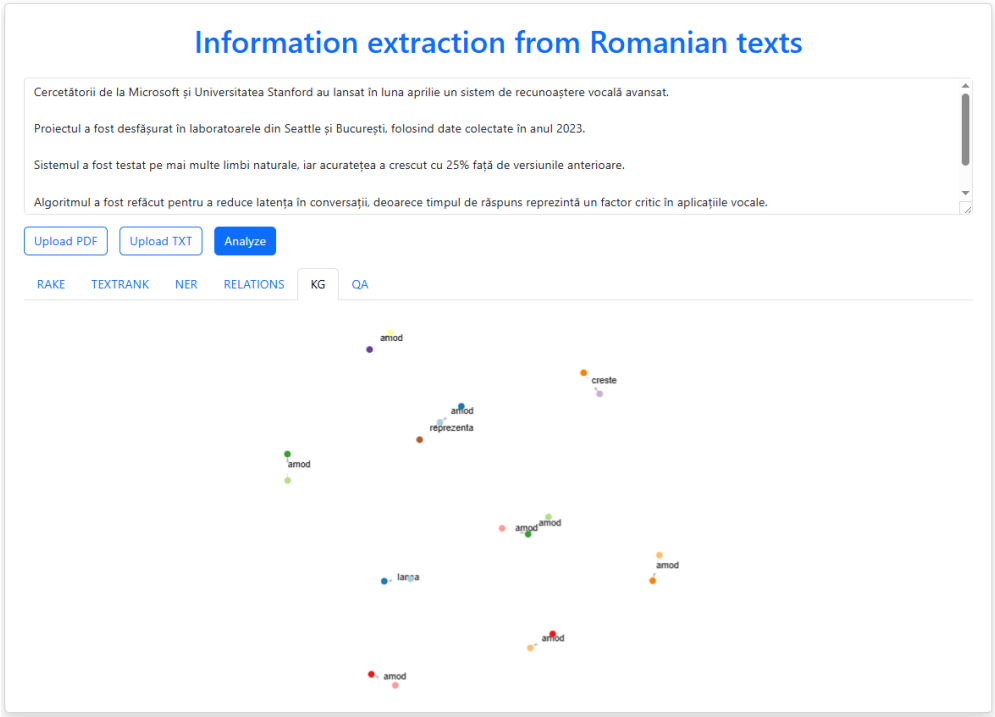

Figure 5.4: Knowledge Graph extracted from syntactic triplets

The knowledge graph is built from subject–predicate–object triplets. In the example above, we can see the triplets:

- *cercetator* → *lansa*→*sistem* – marking the main action performed by the subject.
- *acuratete* → *creste*→*25*
- *timp* → *reprezenta* → *factor*

Also the knowledge graph was designed to contain dependency relations of type `amod`, some examples would be:

- *factor* → *critic*
- *versiune* → *anterior*
- *resursa* → *hardware*
- *dispozitiv* → *mobil*

The graph representation enhances interpretability, allowing users to visualize dependencies across clauses. These graphs are also useful for downstream tasks like reasoning, summarization, and question-answering.

5.1.7 Question Answering Results

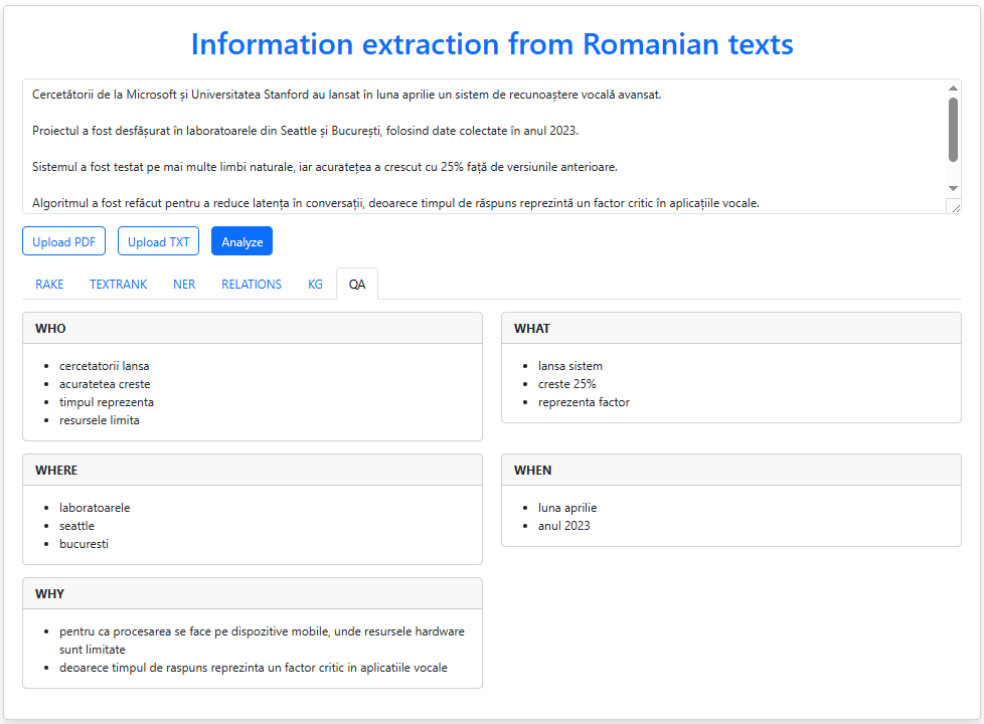

Figure 5.5: Automatically extracted answers to WHO/WHAT/WHERE/WHEN/WHY

- **Who?**
  - cercetatorii lansa
  - acuratetea creste
  - timpul reprezenta
  - resursele limita
- **What?**
  - lansa sistem
  - creste 25%
  - reprezenta factor
- **Where?**
  - laboratoarele
  - seattle
  - bucuresti

- **When?**
  - luna aprilie
  - anul 2023
- **Why?**
  - pentru ca procesarea se face pe dispozitive mobile, unde resursele hardware sunt limitate
  - deoarece timpul de raspuns reprezinta un factor critic in aplicatiile vocale

This answers offers a summary in such a way that users can understand the main idea without reading the full text.

5.2 User Manual

This section provides instructions on how to use the application. The tool is designed to help people use this application

5.2.1 Starting the Application

- Upload a document in `.txt` or `.pdf` format, or insert plain text into the input field.
- Click on the **Analyze** button to process the input.

5.2.2 Navigation Between Tabs

Once the text is analyzed, the user can see the information on each tab:

- **RAKE:** displays the top keyphrases extracted using the RAKE algorithm.
- **TextRank:** displays the top keywords extractedd using TextRank algorithm.
- **NER :** groups named entities into categories such as *PERSON*, *ORG*, *DATE*, *LOC*, etc.
- **Relations:** presents dependency relations.
- **Knowledge Graph:** a visual representation of the graph.
- **Question Answering:** summarizes the most relevant information in response to the questions **WHO, WHAT, WHERE, WHEN, WHY**.

5.2.3 Output Interpretation

- All keyword lists are ordered by relevance scores.
- The graph is interactive and zoomable for better exploration.

5.2.4 Limitations

- The application is optimized for Romanian texts.
- Results may vary for informal, noisy, or mixed-language input.

Chapter 6

Conclusions

This thesis presented a fully functional application for extracting, analyzing, and visualizing information from Romanian documents. By integrating traditional NLP methods (RAKE, TextRank), syntactic analysis, and graph-based representations, offering a practical solution for Romanian language, a morphologically rich and underrepresented language in computational linguistics.

6.1 Key Contributions

The most important achievements of this thesis are:

- A modular backend written in Python that preprocesses Romanian input through lemmatization, custom stopword filtering, and dependency parsing, following recommendations in [12].
- Implementation of two keyword extraction strategies: RAKE and TextRank, adapted for Romanian and enabling comparative evaluation as explored in [9].
- Integration of a relation extractor and a knowledge graph builder, which identify subject–predicate–object triples using SpaCy syntactic dependencies and present them as interactive graphs.
- A frontend built with React, providing a user-friendly interface for document upload, manual text input, and real-time exploration of NLP outputs.
- Full support for NER and a basic QA module that gives answers to “WHO”, “WHAT”, “WHEN”, “WHERE”, and “WHY” questions.

6.2 Challenges Encountered

Romanian has significant challenges in natural language processing due to its inflectional morphology, flexible word order, and context-dependent structures. These factors complicate tasks such as keyword extraction and dependency parsing. Several measures were taken to mitigate these issues:

- **Lemmatization and normalization:** Implemented using spaCy's Romanian models and an additional normalization step to unify word forms (e.g., verbs in different conjugations).
- **Stopword curation:** The default stopword list was manually extended based on analysis of texts, in accordance with best practices outlined in [12].
- **Diacritic handling:** Character normalization was used to address inconsistencies in text sources where Romanian diacritics were missing.

6.3 Future Work

Although the application is complete and usable for Romanian texts, there are several things that can be improved.

6.3.1 Automated Summary Generation

An useful update would be automatic summarization, particularly using sentence-level ranking based on TextRank, similar to the original formulation proposed by RadaMihalcea and Paul Tarau. This would allow users to receive short, coherent summaries of documents, complementing the keyword output. Integration of this feature aligns with practices explored in [9].

6.3.2 Backend API and Cloud Deployment

To support broader usage in educational platforms, the backend can be converted into a RESTful API (based on the current FastAPI structure) and deployed to the cloud, where users could create accounts and their upload history would be saved.

Bibliography

[1] Explosion AI. spacy: Industrial-strength natural language processing in python, 2024. Accessed May 2025. URL: https://spacy.io/.

[2] Andrei Avram, Radu Tudor Ionescu, et al. Rolargesum: A large dialect-aware romanian news dataset for summarization and keyword extraction. *arXiv preprint arXiv:2412.11317*, 2024. URL: https://arxiv.org/abs/2412.11317.

[3] Slobodan Beliga. Keyword extraction: A review of methods and approaches. *Information and Organizational Sciences*, 39:1–20, 2014. URL: https://hrcak.srce.hr/file/188694.

[4] Slobodan Beliga, Ana Meštrović, and Sanda Martinčić-Ipšić. An overview of graph-based keyword extraction methods and approaches. *Journal of Information and Organizational Sciences*, 39(1):1–20, 2015. URL: https://hrcak.srce.hr/140857, doi:10.31341/jios.39.1.1.

[5] Steven Bird, Ewan Klein, and Edward Loper. Natural language toolkit (nltk), 2024. Accessed May 2025. URL: https://www.nltk.org/.

[6] Wenjuan Han, Yong Jiang, Hwee Tou Ng, and Kewei Tu. A survey of unsupervised dependency parsing. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2522–2533, 2020. URL: https://aclanthology.org/2020.coling-main.227/, doi:10.18653/v1/2020.coling-main.227.

[7] Răzvan Ion et al. Ensemble romanian dependency parsing with neural networks. In *Proceedings of LREC*, pages 248–254, 2018. URL: https://aclanthology.org/L18-1248.pdf.

[8] Keunha Kim and Youngjoong Ko. Dependency parsing with the structuralized prompt template. *arXiv preprint arXiv:2502.16919*, 2025. URL: https://arxiv.org/abs/2502.16919.

[9] Nikita Mangwani. Keyphrase extraction from document using rake and textrank algorithms, 2020. URL: https://medium.com/@nikita-mangwani102737/keyphrase-extraction-from-document-using-rake-and-textrank-algorith

[10] MarkovML. Understanding the rake algorithm: A simple guide, 2024. Accessed May 2025. URL: https://www.markovml.com/blog/rake-algorithm.

[11] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into text. *Proceedings of EMNLP 2004*, pages 404–411, 2004. URL: https://aclanthology.org/W04-3252/.

[12] Ionuț Nițu and Mihai Dascălu. Natural language processing tools for romanian – going beyond a low-resource language. *ResearchGate Preprint*, 2024. URL: https://www.researchgate.net/publication/380779085.

[13] Tadashi Nomoto. Keyword extraction: A modern perspective. *SN Computer Science*, 4(92), 2023. URL: https://link.springer.com/article/10.1007/s42979-022-01481-7, doi:10.1007/s42979-022-01481-7.

[14] LiRo Project. Liro: Benchmark and leaderboard for romanian language tasks, 2021. NeurIPS Datasets and Benchmarks. URL: https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/5f93f983524def3dca464469d2cf9f3e-Abstract-round1.html.

[15] RELATE Project. Relate: A modern processing platform for romanian language. *ResearchGate Preprint*, 2024. URL: https://www.researchgate.net/publication/385353950.

[16] Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. Automatic keyword extraction from individual documents. In *Text Mining: Applications and Theory*, pages 1–20. John Wiley & Sons, Ltd, 2010. URL: https://onlinelibrary.wiley.com/doi/10.1002/9780470689646.ch1, doi:10.1002/9780470689646.ch1.

[17] Claude Sammut and Geoffrey I. Webb. Tf–idf. *Encyclopedia of Machine Learning*, pages 986–987, 2010. doi:10.1007/978-0-387-30164-8_832.

[18] Analytics Vidhya. Dependency parsing in natural language processing with examples, 2021. Accessed May 2025. URL: https://www.analyticsvidhya.com/blog/2021/12/dependency-parsing-in-natural-language-processing-with-examples/.

[19] Analytics Vidhya. Rake algorithm in natural language processing, 2021. URL: https://www.analyticsvidhya.com/blog/2021/10/rapid-keyword-extraction-rake-algorithm-in-natural-language-process

[20] Analytics Vidhya. Keyword extraction methods from documents in nlp, 2022. Accessed May 2025. URL: https://www.analyticsvidhya.com/blog/2022/03/keyword-extraction-methods-from-documents-in-nlp/.