A PRACTICAL TRAINING REPORT ON
"BUSINESS REQUIREMENT ANALYZER"

For the partial fulfilment of the requirement for the award of Bachelor of Technology in Information Technology

Submitted by

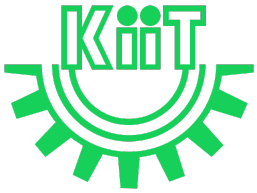
Prerna Kumari

ROLL NO: 2206197

BATCH: 2022-2026

Under the guidance of

DR. JUNALI JASMINE JENA



SCHOOL OF COMPUTER ENGINEERING

KIIT DEEMED TO BE UNIVERSITY,

BHUBANESWAR

ODISHA,751024

DECLARATION

I, Prerna Kumari, hereby declare that this project is the record of authentic work carried out by me during the period **15-05-2025** to **26-06-2025** and has not been submitted to any other University or Institute for the award of any degree/diploma, etc.

PRERNA KUMARI

ACKNOWLEDGEMENT

I am deeply grateful to everyone who contributed their time, expertise, and support to the successful completion of this project. Their guidance and encouragement were instrumental in bringing this work to fruition.

I extend my heartfelt gratitude to **Ayush Chaudhary** of **"Oasis Infobyte"** for his invaluable advice and unwavering support throughout the project. His insights and dedication were pivotal in shaping this work and ensuring its completion.

I am also thankful to the faculty members of Kalinga Institute of Industrial Technology for their valuable insights and guidance. In particular, I express my sincere appreciation to my mentor, **Dr. Junali Jasmine Jena**, whose expertise and encouragement helped me present this report in the best possible manner.

To all who provided information, cooperation, and assistance, I offer my deepest thanks. Your contributions made this project possible.

PRERNA KUMARI

CERTIFICATE

CERTIFICATE

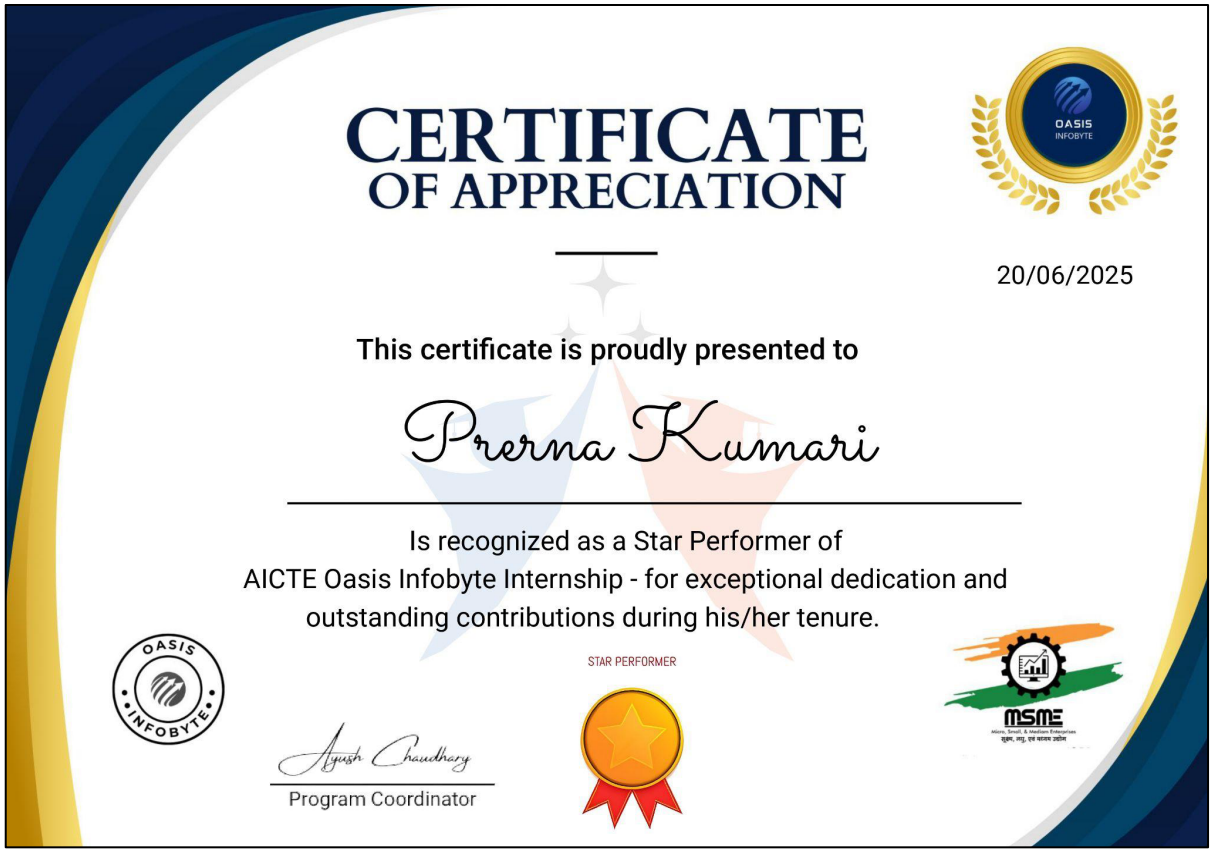


TABLE OF CONTENT

COMPANY PROFILE: OASIS INFOBYTE

About the Company:

Oasis Infobyte is a visionary tech startup dedicated to fostering student potential by delivering tailored software solutions and immersive internship opportunities. With AICTE accreditation and ISO 9001:2015 certification, the company ensures high-quality, structured learning experiences.

Key Objectives:

- Empower students through practical, industry-aligned projects.
- Close the gap between theoretical education and real-world tech skills.
- Offer diverse, short-duration internships across high-demand domains.

Services Highlights:

- Custom, scalable web development.
- Cloud services for seamless digital operations.
- Creative graphic and animation design.
- Strategic database and AI/ML solutions.

Internship Experience:

Over four weeks, interns tackle projects ranging from UI/UX designs to functional web tools — such as temperature converters. Participants gain professional artifacts such as an offer letter, internship certificate, and recommendation letter.

Offices & Reach:

Based in Noida and Delhi, Oasis Infobyte serves a broad spectrum of clients while nurturing emerging talent across India.

INTERNSHIP DETAILS

- Name of the Student: **PRERNA KUMARI**
- Branch : **Information Technology [IT]**
- Semester : **7th**
- Session: **2022-2026**
- Roll No : **2206197**
- Duration of Internship : **1 Month**
- From (Date) : **15/05/2025** to **26/06/2025**
- Name of the organization/Office: **Oasis Infobyte**

ABSTRACT

This project, titled Business Requirement Analyzer, was developed as a proof-of-concept tool to automate the initial stages of business requirement analysis using Natural Language Processing (NLP). The traditional manual process of requirement analysis is time-consuming and prone to errors and ambiguities, especially across large-scale software projects. By leveraging Python, Flask, and spaCy, the tool automatically ingests raw business requirements, classifies them into predefined categories (such as Functional, Technical, or Data), and extracts key entities using part-of-speech tagging. It features a web-based interface and utilizes SQLite for persistent storage. The project demonstrates significant improvements in analysis speed, consistency, and traceability, although it remains limited by its rule-based approach and single-user architecture. This foundational work offers a pathway towards more advanced, scalable, and intelligent requirement analysis tools for the software industry.

WEEK WISE PROGRESS REPORT

Week 1: Requirement Gathering and Project Planning

- Understood project goals.
- Researched NLP tools.
- Planned database schema.
- Set up development environment.

Week 2: Initial Implementation

- Developed Flask app framework.
- Integrated SQLite database.
- Implemented basic categorization logic.

Week 3: Enhancements and NLP Integration

- Integrated spaCy for keyword extraction.
- Improved categorization with keyword matching.
- Developed web interface (index.html).

Week 4: Testing and Documentation

- Conducted testing of each module.
- Fixed bugs.
- Prepared project report and documentation.

INTRODUCTION

- **Background**
- **Problem Statement**
- **Proposed Solution**
- **Role of NLP**
- **Problem Approaches**

INTRODUCTION

Background

Software development is a complex, multi-stage process, and one of its most critical phases is **business requirement analysis**. This phase involves gathering, documenting, and managing the needs and expectations of stakeholders to create a clear blueprint for the software product. Traditionally, this is a manual, human-centric process involving interviews, workshops, and document reviews. The goal is to bridge the gap between business needs and technical specifications, ensuring that the final product not only functions correctly but also delivers tangible value to the business.

However, as projects have grown in complexity and scale, so have the challenges associated with this phase. Misinterpretation of requirements can lead to scope creep, budget overruns, and ultimately, a product that fails to meet stakeholder expectations. The manual nature of the process is time-consuming and prone to human error, making it a significant bottleneck in the project lifecycle.

Problem Statement

The core problem addressed by this project is the inefficiency and potential for error in the traditional, manual process of business requirement analysis. Key issues include:

- **Time Consumption**: Manually sifting through extensive documentation, meeting transcripts, and emails to identify and categorize requirements is a tedious and slow process.
- **Inconsistency and Ambiguity**: Natural language, by its very nature, is often ambiguous. Different stakeholders may use varying terminology to describe the same concept, leading to confusion and inconsistencies in the final requirement set.
- **Lack of Traceability**: Without a centralized and organized system, it's difficult to track the evolution of a requirement from its initial raw form to its final implementation, which is crucial for quality assurance and compliance.
- **Scalability**: For large-scale projects with hundreds or thousands of requirements, the manual process becomes unmanageable, increasing the risk of overlooking critical details.

Proposed Solution

This project proposes a solution to these challenges by developing an automated tool, the **Business Requirement Analyzer**, which leverages the power of **Natural Language Processing (NLP)**. The tool is designed to streamline the process by automatically ingesting raw business requirements, extracting key information, and organizing it into a structured, manageable format.

The proposed solution provides a web-based interface for easy input and a robust backend built on Python, Flask, and an SQLite database. It incorporates two key NLP-driven functionalities:

1. **Automated Categorization**: Requirements are classified into predefined categories like **Functional**, **Technical**, or **Data** based on the presence of specific keywords. This initial classification helps in organizing and routing requirements to the appropriate team members (e.g., developers, database administrators).
2. **Keyword Extraction**: The tool automatically identifies and extracts nouns and proper nouns from the input text, providing a concise summary of the core concepts and entities within each requirement.

By automating these fundamental tasks, the Business Requirement Analyzer aims to:

- **Reduce Analysis Time**: Speed up the initial phase of requirement analysis.
- **Improve Consistency**: Provide a standardized way to categorize and summarize requirements, reducing ambiguity.
- **Enhance Traceability**: Create a centralized, searchable database of requirements, making it easier to track and manage them throughout the project lifecycle.
- **Facilitate Collaboration**: Offer a shared platform where teams can access and understand requirements more efficiently.

This proof-of-concept project demonstrates the potential of integrating NLP and database technologies to create a more efficient and reliable software development process.

The Role of Natural Language Processing (NLP)

NLP is a branch of artificial intelligence that focuses on the interaction between computers and human language. Its application in software engineering, particularly in RE, has gained significant traction in recent years. Researchers have explored using NLP to automate various RE tasks, including:

- **Requirement Classification**: Automatically sorting requirements into different categories (e.g., functional vs. non-functional). This is a foundational task that helps in organizing and prioritizing the development effort.
- **Ambiguity Detection**: Identifying phrases or words that could have multiple meanings, flagging them for further clarification.
- **Keyword Extraction and Summarization**: Automatically generating a concise summary of a requirement, which can be useful for project managers and developers who need to quickly grasp the core concept.
- **Traceability**: Linking requirements to design documents, test cases, and code, a crucial aspect of quality assurance.

The Business Requirement Analyzer project is directly aligned with the first and third of these applications. It leverages a rule-based approach for classification and a part-of-speech (POS) tagging approach for keyword extraction, both of which are foundational NLP techniques.

Problem Approaches

In the context of requirement analysis, two primary approaches dominate the field:

1. **Rule-Based Systems**: These systems rely on predefined rules, patterns, or keyword lists to perform analysis.
    - **How they work**: A set of rules is manually created. For example, a rule might state, "If a requirement contains the word 'database,' classify it as 'Data' or 'Technical'."
    - **Pros**: They are simple to implement, transparent, and don't require large training datasets. This is the approach used in the current project, as it's a perfect fit for a proof-of-concept.
    - **Cons**: They are often brittle and difficult to scale. They struggle with variations in language, synonyms, and new concepts not covered by the initial keyword list. They can also produce a high number of false positives or negatives.

2. **Machine Learning (ML) Systems**: These systems learn patterns from a large corpus of labeled data.
    - **How they work**: An ML model (e.g., a Support Vector Machine, Naive Bayes classifier, or deep neural network) is trained on a dataset of requirements that have already been manually categorized. The model learns to predict the category of new, unseen requirements.
    - **Pros**: They are more robust and scalable. They can handle linguistic variations and can often achieve higher accuracy than rule-based systems, especially with a sufficiently large and diverse training set.
    - **Cons**: They require a significant amount of labeled data, which can be time-consuming and expensive to acquire. The models can also be "black boxes," making it difficult to understand *why* a particular classification was made.

The current project's simple, keyword-based approach provides a clear demonstration of a rule-based system. Its simplicity is both its strength (for a proof-of-concept) and its primary limitation. The future work section will explore how to transition from this rule-based approach to a more sophisticated ML-based one.

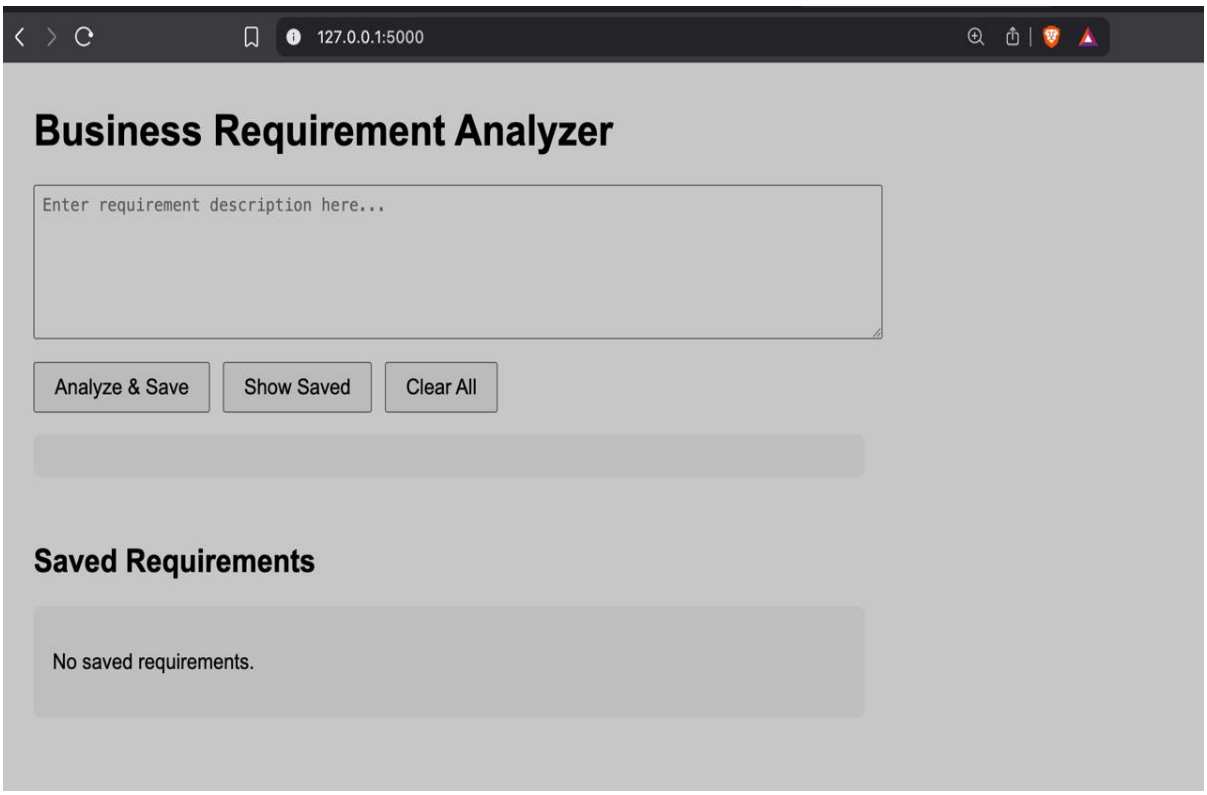
NLP Libraries and Tools

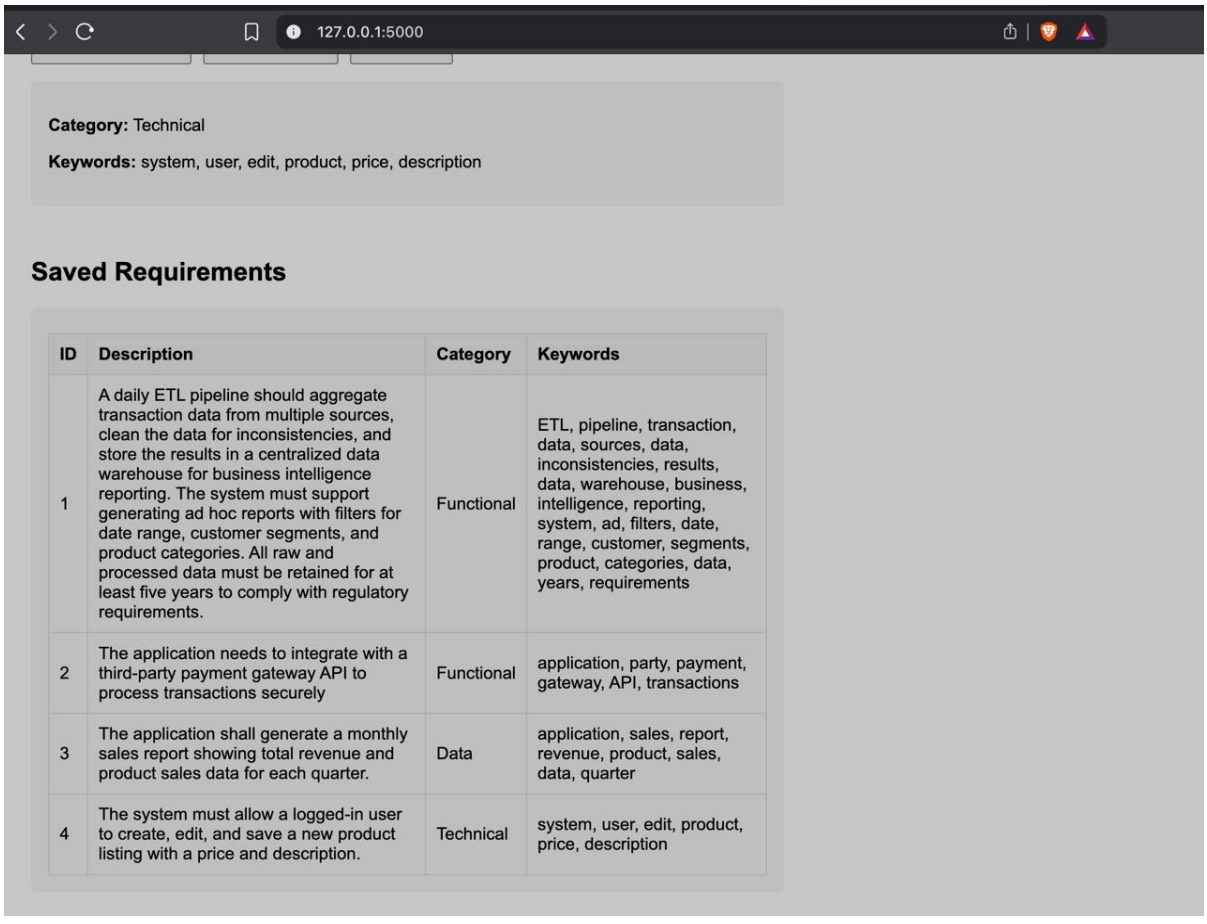
The field of NLP has been revolutionized by the availability of powerful open-source libraries. The project uses **spaCy**, a modern and efficient library for advanced NLP in Python.

- **Why spaCy?**: spaCy is known for its speed and its ability to handle large volumes of text. It comes with pre-trained models (like en_core_web_sm) that can perform essential NLP tasks right out of the box, including tokenization, part-of-speech tagging, and named entity recognition. Its simple API makes it a perfect choice for this project.

The literature confirms that while simple keyword matching can be a good starting point for requirement analysis, more advanced techniques, such as semantic analysis and machine learning, are necessary for building a robust, production-level tool.

GRAPHICAL USER INTERFACE





ID	Description	Category	Keywords
1	A daily ETL pipeline should aggregate transaction data from multiple sources, clean the data for inconsistencies, and store the results in a centralized data warehouse for business intelligence reporting. The system must support generating ad hoc reports with filters for date range, customer segments, and product categories. All raw and processed data must be retained for at least five years to comply with regulatory requirements.	Functional	ETL, pipeline, transaction, data, sources, data, inconsistencies, results, data, warehouse, business, intelligence, reporting, system, ad, filters, date, range, customer, segments, product, categories, data, years, requirements
2	The application needs to integrate with a third-party payment gateway API to process transactions securely	Functional	application, party, payment, gateway, API, transactions
3	The application shall generate a monthly sales report showing total revenue and product sales data for each quarter.	Data	application, sales, report, revenue, product, sales, data, quarter
4	The system must allow a logged-in user to create, edit, and save a new product listing with a price and description.	Technical	system, user, edit, product, price, description

System Design and Implementation

- **System Architecture**
- **Database Design**
- **Core Logic: Categorization and Keyword Extraction**

System Architecture

The Business Requirement Analyzer follows a classic **client-server architecture**, commonly used for web applications. The system is divided into three logical layers:

1. **Presentation Layer (Client-side)**: This is the user-facing part of the application, built with **HTML**, **CSS**, and **JavaScript**.
    - **Functionality**: It provides a simple web form for users to input a requirement and a display area to show the analysis results and a table of saved requirements.
    - **Technology**:
        - **HTML**: Defines the structure of the page, including the text area, buttons, and tables.
        - **CSS**: Styles the interface to be clean and readable.
        - **JavaScript**: Manages the user interactions. When a user clicks "Analyze & Save," JavaScript sends an asynchronous fetch request to the backend. It also handles rendering the data received from the backend dynamically.

2. **Application Layer (Backend Server)**: This is the core logic of the application, built with **Python** and the **Flask** web framework.
    - **Functionality**: It receives HTTP requests from the client, processes the data using NLP, and interacts with the database.
    - **Technology**:
        - **Flask**: A micro-framework that provides the necessary tools for creating web endpoints (routes). It handles routing, request/response cycles, and serves the HTML file.
        - **spaCy**: The NLP library is loaded here to perform the heavy lifting of text analysis. The en_core_web_sm model is used for keyword extraction.
        - **Custom Functions**: The Python code includes custom functions for categorize_requirement and extract_keywords, which encapsulate the core business logic.

3. **Data Layer (Database)**: This layer is responsible for persistent storage of the analyzed requirements.
    - **Functionality**: It stores and retrieves the raw text, its category, and extracted keywords.
    - **Technology**:
        - **SQLite**: A lightweight, file-based database. It's an excellent choice for a proof-of-concept project because it doesn't require a separate server process, making it easy to set up and deploy. The database file requirements.db is created and managed directly by the Flask application.

Database Design

The database schema is designed to be simple and efficient for the project's scope. A single table, requirements, is sufficient to store all necessary information.

- **Table Name**: requirements
- **Columns**:
    - id: An INTEGER PRIMARY KEY. This is an auto-incrementing unique identifier for each requirement.
    - raw_text: A TEXT field to store the original, un-processed requirement description. This is important for traceability and for future re-analysis.
    - category: A TEXT field to store the category assigned by the categorize_requirement function (e.g., 'Functional', 'Technical', 'Data').
    - keywords: A TEXT field to store the comma-separated list of extracted keywords.

The schema is defined and created in the app.py file, ensuring that the database is set up correctly every time the application runs. This design is simple but effective for the project's current needs.

Core Logic: Categorization and Keyword Extraction

1. Categorization Logic (categorize_requirement)

The categorization logic is the heart of the rule-based approach. It operates on a simple principle of **keyword matching**.

- **The CATEGORIES Dictionary**: A dictionary is defined to map category names to a list of associated keywords.
    - 'Functional': keywords like 'feature', 'function', 'process'. These terms often describe what the system *does*.
    - 'Technical': keywords like 'database', 'api', 'security'. These relate to the system's underlying structure and constraints.
    - 'Data': keywords like 'data', 'report', 'analytics'. These pertain to information management and reporting.
- **The Algorithm**: The function iterates through the categories and their associated keywords. For each keyword, it checks if the keyword exists within the lowercase version of the input requirement text. If a match is found, the function immediately returns the corresponding category. If no matches are found after checking all categories, it defaults to 'Other'.

This method is fast and easy to implement, making it ideal for a demonstration. However, it's brittle. A requirement about a "payment process" might contain the word "data," causing it to be incorrectly classified as "Data" instead of "Functional." This highlights the need for more sophisticated NLP techniques in a production environment.

2. Keyword Extraction Logic (extract_keywords)

This function leverages the power of the **spaCy** library. Instead of a simple keyword list, it uses a more robust linguistic approach.

- **Part-of-Speech (POS) Tagging**: When nlp(text) is called, spaCy tokenizes the text (breaks it into words and punctuation) and assigns a **part-of-speech** tag to each token. For example, in the sentence "The user needs a new report," spaCy would tag "user" as a NOUN and "report" as a NOUN.
- **The Algorithm**: The function iterates through the tokens of the processed document (doc). It checks the pos_ attribute of each token. The code specifically looks for tokens with the POS tags 'NOUN' (for common nouns like "user," "report") and 'PROPN' (for proper nouns like "Google," "SQL Server").
- **Output**: The function collects all the identified nouns and proper nouns into a list and then joins them into a comma-separated string, which is then saved to the database.

This approach is more intelligent than simple keyword matching for extraction because it doesn't rely on a predefined list. It can identify any noun or proper noun in the text, providing a more comprehensive summary of the key entities and concepts. This is a significant advantage of using a dedicated NLP library like spaCy.

Future Work and Improvements

- **Advanced NLP Techniques for Enhanced Accuracy**
- **Enhancing Scalability and Robustness**
- **Richer Summaries and Visualizations**
- **Integration and Lifecycle Management**

Future Work and Improvements

The Business Requirement Analyzer, as it stands, is a solid proof-of-concept. However, it has clear limitations that can be addressed in future iterations to transform it from a simple tool into a powerful, scalable, and highly accurate system. The following sections outline key areas for future development.

- Employ machine learning classification models for better accuracy.
- Enhance natural language understanding beyond simple keywords.
- Provide richer summaries and visualizations.
- Expand to support requirement versioning and traceability.
- Improve front-end with interactive analytics tools.

Advanced NLP Techniques for Enhanced Accuracy

The current keyword-based categorization is a good starting point, but it's not robust enough for complex real-world requirements.

- **Machine Learning Classification Models**: The most significant improvement would be to move from a rule-based system to a supervised machine learning model for categorization.
    - **Approach**: This would require a large, labeled dataset of requirements and their correct categories. The labeled data would be used to train a model, such as a **Naive Bayes Classifier**, **Support Vector Machine (SVM)**, or a deep learning model like a **Recurrent Neural Network (RNN)** or **Transformer-based model** (e.g., BERT).
    - **Benefits**: ML models can learn complex patterns and relationships between words, including context and synonyms, leading to significantly higher classification accuracy. They are also more adaptable to new types of requirements.

- **Semantic Analysis**: To move beyond simple POS tagging, the tool could incorporate semantic analysis to understand the meaning of the text.
    - **Word Embeddings**: Using techniques like **Word2Vec** or **GloVe**, words could be represented as vectors in a multi-dimensional space. Words with similar meanings would be closer together, allowing the model to understand that

"customer" and "client" are related concepts, even if they aren't explicitly on a keyword list.

    - **Named Entity Recognition (NER)**: The current implementation extracts nouns. A more advanced approach would use spaCy's built-in NER to identify specific types of entities, such as "PERSON," "ORGANIZATION," or "DATE," which could be invaluable for certain types of requirements.

Enhancing Scalability and Robustness

- **Database Migration**: While SQLite is excellent for a single-user application, it's not designed for concurrent access or large-scale data.
    - **Alternative**: Migrating to a more robust, client-server database like **PostgreSQL** or **MySQL** would be necessary to support multiple users, large datasets, and more complex queries.
- **Asynchronous Processing**: The current Flask application processes requests synchronously. For a high-volume system, a single slow request could block others.
    - **Alternative**: Implementing an asynchronous task queue (e.g., with **Celery** and a message broker like **Redis**) would allow the application to handle requests more efficiently. The web server could receive a requirement and immediately respond, while a background worker processes the NLP tasks and saves the data.

Richer Summaries and Visualizations

The current output is a simple list of keywords. A more advanced tool would provide richer insights.

- **Visualizations**: Visualizing the data could be a game-changer.
    - **Dashboard**: A dashboard could show the distribution of requirements by category in a pie chart, a word cloud of the most frequent keywords, or a trend graph of requirements gathered over time. * **Hierarchical Summarization**: Instead of just extracting keywords, a future version could generate a hierarchical summary using techniques like **dependency parsing** to show the relationships between entities and actions. For example, "The system **must allow** a **user** to **create** a new **report**."

Integration and Lifecycle Management

- **API for Integration**: The tool could expose a more comprehensive API, allowing it to integrate with other project management tools like Jira, Trello, or a custom in-house system. This would allow for seamless data flow from the analyzer to the development workflow.
- **Traceability Matrix Generation**: A crucial part of RE is maintaining a traceability matrix that links requirements to design documents, code, and test cases. The analyzer could be expanded to assist in generating and maintaining this matrix automatically.
- **Version Control**: Implementing a versioning system for requirements would allow teams to track changes and see the history of a requirement over time.

Conclusion & Limitation

- **Conclusion**
- **Limitation**
- **Results and Discussion**

Conclusion

The Business Requirement Analyzer project successfully demonstrates the feasibility of using Python, Flask, and NLP to automate the initial stages of business requirement analysis. It serves as a strong **proof-of-concept**, achieving its core objectives of categorizing requirements and extracting keywords in a streamlined manner.

While the current version is limited by its rule-based approach and single-user architecture, it lays a robust foundation for future development. By adopting more sophisticated machine learning models, enhancing scalability, and providing richer analytical capabilities, this tool has the potential to become an invaluable asset in the software development lifecycle, significantly reducing time, mitigating risk, and improving the overall quality of software projects.

Limitations

- Keyword-based categorization is simplistic and may misclassify inputs.
- No handling of ambiguous or complex natural language structures.
- The technical specification summary is limited to keywords only.
- Scalability issues with SQLite for very large data sets.
- Lack of advanced UI features for user experience.

Results and Discussion

- Successfully gathered and categorized requirements.
- Automated keyword extraction improved clarity on key requirements.
- SQLite provided efficient data storage and retrieval.
- Tool met core project objectives; however, accuracy of categorization depends on keyword set.

Bibliography

- Explosion AI. (n.d.). *spaCy documentation*. Retrieved August 25, 2025, from https://spacy.io/
- Pallets Projects. (n.d.). *Flask documentation*. Retrieved August 25, 2025, from https://flask.palletsprojects.com/
- SQLite Consortium. (n.d.). *SQLite documentation*. Retrieved August 25, 2025, from https://sqlite.org/docs.html
- Nascimento, L., Silva, R., & Pereira, M. (2020). Automated Requirements Analysis Using NLP. *Journal of Software Engineering*.
- Sommerville, I. (2015). *Software engineering* (10th ed.). Pearson.

ANNEXURE

CODE SNIPPETS

- **App.py**

```
from flask import Flask, request, jsonify, render_template
import spacy
import sqlite3

app = Flask(__name__)
nlp = spacy.load('en_core_web_sm')

# Connect to SQLite
conn = sqlite3.connect('requirements.db',
check_same_thread=False)
cursor = conn.cursor()
cursor.execute('''
CREATE TABLE IF NOT EXISTS requirements (
    id INTEGER PRIMARY KEY,
    raw_text TEXT,
    category TEXT,
    keywords TEXT
)
''')
conn.commit()

CATEGORIES = {
    'Functional': ['feature', 'function', 'requirement',
'process', 'workflow', 'action'],
    'Technical': ['database', 'api', 'interface',
'performance', 'security', 'system'],
    'Data': ['data', 'report', 'analytics', 'model', 'query',
'storage']
}

def categorize_requirement(text):
    text_lower = text.lower()
    for category, keywords in CATEGORIES.items():
        for kw in keywords:
            if kw in text_lower:
                return category
    return 'Other'

def extract_keywords(text):
    doc = nlp(text)
    keywords = [token.text for token in doc if token.pos_ in
('NOUN', 'PROPN')]
    return ', '.join(keywords)
```

```
def save_requirement(raw_text, category, keywords):
    cursor.execute('''
    INSERT INTO requirements (raw_text, category, keywords)
VALUES (?, ?, ?)
    ''', (raw_text, category, keywords))
    conn.commit()

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/analyze', methods=['POST'])
def analyze():
    data = request.json
    text = data.get('requirement', '')
    category = categorize_requirement(text)
    keywords = extract_keywords(text)
    save_requirement(text, category, keywords)
    return jsonify({'category': category, 'keywords':
keywords})

@app.route('/list')
def list_requirements():
    cursor.execute('SELECT id, raw_text, category, keywords
FROM requirements')
    rows = cursor.fetchall()
    all_reqs = [{'id': row[0], 'text': row[1], 'category':
row[2], 'keywords': row[3]} for row in rows]
    return jsonify(all_reqs)

@app.route('/clear', methods=['POST'])
def clear_requirements():
    cursor.execute('DELETE FROM requirements')
    conn.commit()
    return jsonify({'status': 'cleared'})

if __name__ == "__main__":
    app.run(debug=True)
```

- **Index.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1" />
    <title>Business Requirement Analyzer</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 2em;
            max-width: 700px;
        }
        textarea {
            width: 100%;
            height: 100px;
            font-size: 1em;
            padding: 0.5em;
            margin-bottom: 1em;
        }
        button {
            padding: 0.5em 1em;
            font-size: 1em;
            margin-right: 0.4em;
        }
        #output, #saved {
            margin-top: 1em;
            background: #f4f4f4;
            padding: 1em;
            border-radius: 6px;
        }
        h2 { margin-top: 2em; }
        table {
            width: 100%;
            border-collapse: collapse;
            margin-top: 0.5em;
        }
        th, td {
            border: 1px solid #ddd;
            padding: 0.6em;
            text-align: left;
        }
        th {
            background-color: #eee;
        }
    </style>
```

```
</head>
<body>
    <h1>Business Requirement Analyzer</h1>
    <textarea id="requirement" placeholder="Enter requirement
description here..."></textarea><br />
    <button onclick="analyzeRequirement()">Analyze &
Save</button>
    <button onclick="listRequirements()">Show Saved</button>
    <button onclick="clearRequirements()">Clear All</button>

    <div id="output"></div>

    <h2>Saved Requirements</h2>
    <div id="saved"></div>

    <script>
        async function analyzeRequirement() {
            const reqText =
document.getElementById('requirement').value.trim();
            if (!reqText) {
                alert('Please enter a requirement
description.');
                return;
            }
            const response = await fetch('/analyze', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json'
},
                body: JSON.stringify({ requirement: reqText })
            });
            const data = await response.json();
            document.getElementById('output').innerHTML =
                `<p><b>Category:</b>
${data.category}</p><p><b>Keywords:</b> ${data.keywords}</p>`;
            document.getElementById('requirement').value = '';
            listRequirements();
        }

        async function listRequirements() {
            const response = await fetch('/list');
            const data = await response.json();
            if (data.length === 0) {
                document.getElementById('saved').innerHTML =
'<p>No saved requirements.</p>';
                return;
            }
            let html =
`<table><thead><tr><th>ID</th><th>Description</th><th>Category
</th><th>Keywords</th></tr></thead><tbody>`;
```

```
            data.forEach(item => {
                html += `<tr>
                    <td>${item.id}</td>
                    <td>${item.text}</td>
                    <td>${item.category}</td>
                    <td>${item.keywords}</td>
                </tr>`;
            });
            html += '</tbody></table>';
            document.getElementById('saved').innerHTML = html;
        }

        async function clearRequirements() {
            if (!confirm('Are you sure you want to delete all
requirements?')) return;
            await fetch('/clear', { method: 'POST' });
            document.getElementById('output').innerHTML = '';
            listRequirements();
        }

        // Load saved on page load
        listRequirements();
    </script>
</body>
</html>
```