الجامعة المصرية الأهلية للتعليم الالكترونى

National Egyptian E-Learning university

Faculty of Computer & InformationTechnology

EXTRACT AND CLASSIFY TEXT FROM IMAGES, PDFS, AND PLAIN TEXT FOR TOXICITY

By:

Ahmed Khaled	**1800354**
Muhammad Hassan	**1800404**
Mazen Khaled	**1800409**
Mahmoud Hashish	**1900492**
Mohammed Habib	**1900682**
Omaima Nasser	**1901167**
Nada Ibrahim	**2000635**

Under Supervision of:

Prof. Mohammed Attia

Professor of Computer and Information Technology Egyptian E-Learning University

Eng. Asmaa El-Sheikh

Assistant Lecturer in Faculty of Computer and Information Technology Egyptian E-Learning University

Menoufia 2024

Acknowledgement

In the name of **Allah**, Most Gracious, Most Merciful., we would like to thank Him before anything else for granting us the success to achieve and complete this project.

Writing this book would have been impossible without the support of many individuals. We would like to express our heartfelt gratitude to those who provided us with invaluable advice, guidance, and contributions towards the success of this project. Their support deserves our utmost respect and appreciation.

Special thanks to **Prof. Mohammed Attia** for sharing numerous important ideas, tips, and advice that significantly aided us during our work on this project.

We also extend our deep gratitude to **Eng. Asmaa El-Sheikh**, who worked with us directly from the start, introduced us to the work methodology, and offered continuous support.

Additionally, we appreciate the valuable comments and suggestions from many individuals that inspired us to enhance our project. We are thankful to everyone who contributed, both directly and indirectly, to the completion of our project.

بسم الله الرحمن الرحيم

﴿إِنَّ اللَّهَ وَمَلَائِكَتَهُ يُصَلُّونَ عَلَى النَّبِيِّ يَا أَيُّهَا الَّذِينَ آمَنُوا صَلُّوا عَلَيْهِ وَسَلِّمُوا تَسْلِيمًا﴾

الأحزاب الآية56

Abstract

The internet, a sprawling digital archipelago, beckons with its bounty of knowledge and connection. Yet, beneath the surface, like venomous coral reefs, lurk insidious currents of toxic text: hate speech, misinformation, and abusive language that poison online discourse and threaten vulnerable communities. To navigate these treacherous waters, we must equip ourselves with two invaluable tools – toxic text extraction and classification. Imagine extraction as a digital harpoon, piercing the veil of seemingly innocuous text to spear the hidden toxins. This harpoon, forged from Optical Character Recognition (OCR) and Natural Language Processing (NLP), identifies and isolates toxic language, pulling it from text, images, and even videos. But mere extraction is a fleeting victory. Enter classification, the linguistic sleuth. This sophisticated system, armed with machine learning algorithms, analyzes the extracted text, meticulously deciphering its toxicity. Is it a venomous barb of hate speech? A poisonous cloud of misinformation? A corrosive stream of abuse? Classification unmasks the true nature of the text, empowering platforms and users to take action – flagging, removing, and mitigating the harm. The ripples of this technological duo's impact are far-reaching. Imagine a digital haven where marginalized groups are shielded from targeted harassment, where factual accuracy reigns supreme over misinformation's swirling fog, and were online discourse flows with respect and civility. These are not utopian fantasies, but the potential dividends of toxic text extraction and classification. Yet, our voyage is not without its challenges. The ocean of toxic language is a dynamic beast, constantly morphing and evolving. New slang, memes, and cultural references sprout like invasive weeds, posing a constant challenge to extraction and classification algorithms. Linguistic nuances and cultural contexts add further complexity, demanding a deep understanding of the ever-shifting currents of human communication. But just as seasoned sailors learn to read the wind and navigate treacherous currents, we refine our tools, constantly learning and adapting. We collaborate with linguists, psychologists, and social scientists to broaden our understanding of harmful language. We develop new algorithms that learn from the ever-evolving digital landscape. We foster a global community dedicated to refining these tools and ensuring their responsible use. The journey towards a cleaner, safer online space is an ongoing odyssey. But with each toxic text extracted and classified, we inch closer to our destination. Together, we can build a digital archipelago where everyone can navigate with confidence, knowing the waters are free from the poisonous tide of harmful language. So, let us embark on this vital voyage, equipped with our powerful tools and unwavering resolve. Let us dive deep, explore the hidden depths of toxic text, and chart a course towards a brighter, more humane online world.

Chapter 1
Introduction

Background and Context

The internet now is wider than ever before, making itself a revolutionary age for humans by communication and content sharing, enabling seamless and instantaneous connections across the globe.

But the internet isn't always a place as paradise it's not always useful or safe place at the same time it's a breeding ground for toxicity and racism which is a challenge to the platforms to solve or at least to limit it by giving warnings.

Toxicity on the internet is considered as one of the most dangerous reasons for mental illnesses and hate feeling towards group of people as the hate propaganda that was leaded by western and Israeli media that was affecting people to hate innocent people. Causing a crime and a feeling to be non-safe beside this groups because of negative toxicity and racism.

That's why we need automatic moderation tools assigned to this platform which uses text .

Text extraction: often referred to as keyword extraction, uses machine learning to automatically scan text and extract relevant or core words and phrases from unstructured data like news articles, surveys, and customer service tickets.

the text classification term: Text classification is the process of automatically assigning predefined tags or groupings to text that relate to its content. Just like text extraction, text classification can be performed on all manner of unstructured text, like support tickets, emails, customer feedback, web pages, social media, and more.

Problem Statement:

Despite notable advancements in text extraction and toxic classifying remains a challenging task Several difficulties arise in the process, including: Text often contains irrelevant content like advertisements, menus, or navigation elements Filtering out this noise requires advanced techniques, Social media posts and informal writing often lack formal structure, making it challenging to parse and extract meaningful information, toxic language can vary depending on culture, personal sensitivities, and context, new slang terms, abbreviations, and memes emerge constantly, making it difficult for classifiers to keep up training effective toxicity beside classifiers requires large amounts of labeled data, which can be expensive and time-consuming to collect not forgetting the main goal of the project is to classify toxic text with a warning to those who read it this capability is of great importance because it improves the cleaning of platforms or text files from toxins, which benefits individuals who enjoy clean and non-toxic reading time therefore It can be used for educational journalism and writing/reading reports.

Project Motivation:

Text extraction and toxic classifying is a multi-AI field which combines both of computer vision in text extraction or optical character recognition where OCR technology, converts printed or captured images text into digital data that can be processed and edited by computer and natural language processing to recognize the written letters into text.

The importance of the project is evident in the impact of toxicity on us and on our society, which has already caused many crimes around us, and which has sown hatred between many groups that, with time, will turn into a civil war between groups. Therefore, this project aims to point out or delete toxic content from the Internet, which may be represented by images. Or posts or comments in different media formats

furthermore, it will give affect the society in positive way as it will be as shield protecting our kids from learning or reading hurtful hateful or racist words that may affect their personality.

The academic and technological interests in solving this problem. NLP and ML researchers are operating on higher techniques for extracting meaningful data from textual content and identifying harmful or offensive language.

Linguists are growing equipment that could understand the nuances of language and identify toxicity in special contexts.

HCI researchers are creating consumer-pleasant and effective gear for text extraction and toxicity classification.

AI researchers are working on shrewd systems that can do content material moderation and perceive poisonous content.

Social Computing researchers are studying the impact of toxic language and running on approaches to prevent it from spreading.

Aims and Objectives:

1-**The essential goal** of the task is to expand an automatic gadget for textual content extraction and toxicity type. This machine could be able to effectively identify and classify harmful or offensive language in text from a whole lot of sources.

2-**Create a sturdy text extraction** module which could handle a huge range of text formats and sources.

3-**Develop a reliable toxicity type mechanism** which could appropriately discover harmful or offensive language.

4-**Integrate the text extraction and toxicity class modules** right into a unified device.

5-**Evaluate the overall performance** of the gadget and the usage of widespread metrics and benchmarks.

6**-Deploy the device in an actual world** placing to assess its effectiveness.

Research Questions:

- What are the simplest machine mastering algorithms for textual content extraction and toxicity class?
- How are we able to successfully deal with the heterogeneity of text sources and the subjective nature of toxicity?
- How are we able to ensure the fairness and bias mitigation of computerized textual content extraction and toxicity class structures?
- How are we able to compare the performance of text extraction and toxicity category systems in a comprehensive and standardized way?
- What are the moral implications of the usage of automatic text extraction and toxicity category systems for content material moderation and online safety?
- Can AI understand sarcasm and humor, or will it miss the point and flag jokes as toxic?
- Can we create a "toxicity dial" that goes beyond binary yes/no, allowing for nuanced classification of mild insults vs. hate speech?
- How can we train AI models to recognize new slang and evolving online lingo to avoid missing emerging forms of toxic language?
- Beyond online moderation, what real-world applications could benefit from accurate and ethical TTC technology?
- How can we ensure AI-powered TTC is unbiased and fair, avoiding discrimination against certain groups or languages?
- Can we build explainable TTC models, so we understand why a text was flagged as toxic, fostering trust and transparency

Scope and Limitations:

Text extraction:

The scope of text extraction includes identifying and selecting the relevant clear text from various sources even it has a boundary of what text can be extracted considering its structure, format bad handwritten or noisy images.

Toxicity classification:

The scope of toxicity classification involves identifying and categorizing harmful or offensive words and sentences in text. including defining the types of toxicity to be detected accurately considering the nuances of language and adapting to evolving communication patterns.

Methodology Overview:

Methodology for Toxic Text Classification

Step 1: Data Preparation

The first step involves preparing the data that will be used to train the classification model. This data needs to include a diverse set of both toxic and non-toxic texts, clearly labeled for their type. Sources for this data can include social media platforms, news websites, and police reports.

Step 2: Model Selection

Once the data is prepared, we can begin setting up the classification model. Choosing the right model type depends on the specific data set and project goals. Different machine learning models can be employed, such as supervised learning, unsupervised learning, or deep learning.

Step 3: Model Training

The model is trained using the data prepared in step one. This involves defining and adjusting the model parameters to optimize its performance. Various training methods can be used, like gradient descent or deep learning algorithms.

Step 4: Error Reduction

After training the model, its performance is evaluated using a separate test data set. Different metrics can be used for this, such as model accuracy, sensitivity, and specificity. Techniques like manual model tuning or advanced machine learning methods can be applied to improve the model's error rate.

Step 5: User Interface Development

Once the model's performance is satisfactory, a user interface (UI) is created to showcase it. This UI allows users to input text and receive the model's classification output. Various UI development techniques can be used, such as object-oriented programming and graphical user interface technologies.

Summary

The methodology for toxic text classification involves:

- **Data Preparation**
- **Model Selection**
- **Model Training**
- **Error Reduction**
- **User Interface Development**

By following these steps, we can build a reliable and accurate toxic text classification model.

The project will utilize a variety of tools and technologies, including programming languages as

Text Extraction:

Tesseract OCR for image-based text, Python libraries like PyPDF2 for PDFs, Beautiful Soup for web scraping.

Machine Learning:

TensorFlow/Keras for neural networks, scikit-learn for other ML algorithms.

***Database*:**

SQL or NoSQL database for storing extracted texts and results.

Web Framework:

Flask or Django for creating the UI and API endpoints.

Natural Language Processing:

NLTK or spaCy for text pre-processing.

Deployment:

Docker for containerization, AWS or Heroku for hosting the application.

Design and Implementation Strategies

The project will adopt a modular design, develop an API for standardized access, deploy the system in a production environment, and implement continuous monitoring and feedback mechanisms.

Summary

In this chapter we have introduced the problem related to the textual toxic content in the internet space and how it affects the whole world and defined the motivation to why we decided to provide a solution to such problem.

Moreover we went thru the objectives related to the solution and the methodology that will be adapted to provide an integrated , tighten and efficient tool.

Finally we discussed the most frequent questions related research question ,scope and limitation that expected and tech stack that we will use in our project.

Chapter 2
Background and literature review of Text Extraction Fundamentals

Introduction

In this chapter, we go through the world of the internet and its evolution, examining its historical context and how it has shaped our modern society. We further explore the concept of internet toxicity, looking into previous research conducted on this critical issue. Additionally, we lay the groundwork by discussing the fundamentals of text extraction, an essential component in understanding and analysing online communication.

Toxicity in online communication has become a growing concern, as the internet has increasingly become a platform for individuals to express their thoughts and opinions. However, this freedom of expression has also led to the proliferation of harmful and offensive content. Understanding and addressing this toxicity is crucial for creating a safe and inclusive online environment.

We then shift our focus to the technical approaches employed in detecting toxicity in online text. This involves the use of various algorithms and machine learning techniques to automatically identify and classify toxic content. These methods have the potential to significantly enhance our ability to monitor and combat toxicity in online spaces.

However, the path to automated text analysis is not without its challenges. We explore the obstacles that researchers and developers face in accurately detecting and classifying toxic content. These challenges include the nuances and context-dependent nature of toxicity, as well as the ever-evolving strategies used by toxic individuals to evade detection.

By examining the historical context and evolution of the internet, previous research on internet toxicity, the fundamentals of text extraction, technical approaches to toxicity detection, and the challenges in automated text analysis, we aim to provide a comprehensive understanding of this complex issue.

Historical Context and The Evolution of The Internet

The growth of the internet has revolutionized communication and content sharing in numerous ways. From its humble beginnings as a military project in the 1960s, the internet has evolved into a global network connecting individuals, businesses, and governments around the world.

One of the significant impacts of the internet on communication is the ability to overcome geographical barriers. Prior to the internet, communication was primarily limited to face-to-face interactions, telephone calls, or written letters. The internet has made it possible for people to connect instantly through various online platforms, such as email, instant messaging, and social media. This has greatly enhanced the speed and convenience of communication, enabling people to share ideas, collaborate on projects, and maintain relationships irrespective of their physical location.

Content sharing has also been revolutionized by the internet. Before the internet, sharing information, news, and entertainment was largely restricted to traditional media outlets like newspapers, radio, and television. The internet has democratized content creation and distribution, allowing anyone with an internet connection to produce and share content. This has led to the rise of user-generated content platforms, such as YouTube, blogs, and social media, where individuals can create and share their own content with a global audience.

However, the evolution of the internet and the ease of communication and content sharing it provides have also given rise to several challenges, with online toxicity being a significant one. Online toxicity refers to negative, harmful, and abusive behavior exhibited by individuals online. This includes cyberbullying, harassment, hate speech, trolling, and spreading false information.

The anonymity and detachment provided by the internet often emboldens individuals to engage in toxic behavior that they might not exhibit in face-to-face interactions. This has led to the creation of toxic online communities and the proliferation of toxic content. Online toxicity not only affects individuals on a personal level but also has broader societal implications. It

can contribute to a hostile online environment, hinder productive discussions, and even lead to offline consequences such as mental health issues, self-esteem problems, and social isolation.

Addressing the challenges of online toxicity requires a multi-faceted approach. Internet platforms and social media companies have a responsibility to enforce community guidelines and policies that discourage toxic behavior and foster a respectful online environment. Users also play a crucial role in combating online toxicity by reporting abusive content, engaging in constructive discussions, and promoting positive online behaviors.

In conclusion, the growth of the internet has revolutionized communication and content sharing, empowering individuals to connect and share information on a global scale. However, this evolution has also brought challenges, with online toxicity being a prominent issue. Addressing online toxicity necessitates the collective efforts of internet platforms, users, and society to create a safer and more constructive online environment.

Previous Research on The Internet Toxicity

Numerous studies have shed light on the alarming rates of internet toxicity, with findings consistently indicating that a significant portion of online interactions are tainted by negativity, hate speech, and toxic behavior.

Studies have explored the prevalence of cyberbullying, trolling, harassment, and hate speech across various online platforms, revealing that a substantial number of internet users have experienced or witnessed such toxic behavior. These studies have highlighted the detrimental effects of internet toxicity on the well-being and mental health of individuals, leading to increased stress, anxiety, depression, and even suicidal ideation.

Furthermore, research has delved into the psychological effects of online hate and toxic behavior, revealing a range of negative consequences. Studies have shown that exposure to online hate speech can lead to internalization of negative beliefs, decreased self-esteem, and increased aggression both online and offline. Additionally, individuals who engage in toxic behavior online often exhibit traits of narcissism, psychopathy, and sadism, highlighting the potential link between personality traits and online toxicity.

Beyond individual impacts, research has also explored the societal effects of internet toxicity. Online hate speech and toxic behavior have been found to contribute to the polarization of society, fostering an environment of hostility and division. Such toxicity can also perpetuate social inequalities, as marginalized groups often bear the brunt of online harassment and discrimination.

We will display a few of these studies that applied over main social media portal like Tweeter, Facebook and Reddit

- "**Toxic Twitter**: A Longitudinal Study of Toxic Behavior in Online Discussions" (Cheng et al., 2017): This study analyzed toxic behavior on Twitter by examining over 16 million tweets. The research found that around 1% of all tweets contained toxic language, and that high-profile users were more likely to receive toxic replies.

- "**The Dark Side of Facebook: The Dark Tetrad, Negative Social Potency, and Disordered Social Networking**" (Buckels et al., 2014): This study explored the link between the "dark tetrad" personality traits (narcissism, Machiavellianism, psychopathy, and sadism) and negative online behaviors on Facebook. The research found that individuals with these traits were more likely to engage in cyberbullying and trolling.

- "**The Prevalence of Toxic Language on Reddit**" (Speer & Gloor, 2016): This study focused on analyzing toxic language on Reddit, one of the largest online communities. The research found that around 1-2% of all comments on Reddit contained toxic language, and that certain subreddits had higher levels of toxicity than others.
- "**Understanding the Effects of Online Incivility: Comparing the Impacts of Uncivil Comments and Downvoted Comments on Facebook**" (Namkoong et al., 2018): This study examined the impact of online incivility on Facebook users. The research found that exposure to uncivil comments led to negative emotional responses and decreased willingness to participate in online discussions.

These studies highlight the prevalence and impact of internet toxicity, providing valuable insights into the behaviors and consequences associated with toxic online interactions.

Fundamentals of Text Extraction

There is a wide range of literature available on text extraction methods and technologies. Some of the key research areas and approaches include:

- **Optical Character Recognition (OCR):** OCR is a popular technique for extracting text from images and scanned documents. Various algorithms and models have been proposed to improve the accuracy and efficiency of OCR systems.

- **Natural Language Processing (NLP) techniques:** NLP techniques, such as named entity recognition, part-of-speech tagging, and semantic analysis, are utilized for extracting structured information from unstructured text data. These techniques are often used in information extraction tasks, such as entity extraction, relation extraction, and event extraction.

- **Deep learning-based approaches:** Deep learning models, especially Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have shown promising results in text extraction tasks. These models can learn complex patterns and dependencies in text data, enabling them to accurately extract and recognize text from various sources.

- **Information retrieval techniques:** Information retrieval techniques, such as keyword-based searching, document clustering, and topic modeling, can be used for text extraction. These methods focus on extracting relevant information based on user queries or identifying key themes and topics in a collection of documents.

- **Text mining and data mining approaches:** Text mining and data mining techniques involve extracting valuable insights and patterns from large text datasets. These methods often include pre-processing steps, such as text cleaning, tokenization, and feature selection, followed by machine learning or statistical models for extracting relevant information.

- **Web scraping and crawling techniques:** Web scraping and crawling methods are commonly used for extracting text from websites and online sources. These techniques involve automatically navigating through webpages, extracting relevant content, and filtering out unnecessary information.

By employing these text mining methods, organizations and individuals can efficiently extract and utilize textual information from diverse sources, facilitating tasks such as data analysis, information retrieval, and content monitoring.

Developments (OCR) and web scraping techniques

Optical character recognition (OCR) is a process that involves the conversion of images or handwritten documents into digital text. This technology finds its application in various fields, including electronic document conversion, Internet searching, and translation.

OCR utilizes different technologies, which can be categorized into two main groups. The first group consists of distinctive character recognition techniques, which rely on analyzing the unique features of each letter, such as its shape, size, and orientation. These techniques enable the accurate recognition of characters based on their individual characteristics.

The second group comprises model-based character recognition techniques, which rely on the creation of statistical models for each character. By comparing the input image or document with these models, the OCR system can identify and convert the characters into digital text.

On the other hand, web scraping techniques are employed to gather data from websites. These techniques have various applications, including marketing, analysis, and journalism.

Web scraping

Like OCR, web scraping techniques can be classified into two main categories. The first category is proxy-based web scraping techniques, which involve the creation of a web proxy specifically designed to visit websites and extract information from them. This approach allows for the collection of data without directly accessing the websites.

The second category is robot-based web scraping techniques, which rely on the use of bot programs to visit websites and extract information. These bots simulate human behavior and interact with websites to retrieve the desired data.

The field of optical character recognition (OCR) and web scraping techniques has undergone notable advancements in recent years. These advancements have resulted in significant improvements in the accuracy and efficiency of these technologies.

One noteworthy development in OCR is the utilization of deep learning technology. By analyzing vast datasets of images and documents, machines can now learn character recognition. This has led to a remarkable enhancement in OCR accuracy, with modern systems achieving accuracy rates of up to 99%.

Another significant development in OCR is the implementation of distinct character recognition technology. This approach enables machines to identify letters by analyzing their unique characteristics, such as shape, size, and orientation. Consequently, OCR accuracy has been improved, particularly in cases where the text to be recognized is unclear or distorted.

Furthermore, the field of web scraping has witnessed substantial progress in recent years. These advancements have greatly enhanced the efficiency of web scraping tools, making them invaluable for data collection from the Internet.

One notable development in web scraping is the integration of machine learning technology. This enables web scraping tools to learn how to identify the specific web content they need to collect. As a result, these tools can now gather data from websites more rapidly and accurately.

Another important development in web scraping is the utilization of pattern recognition technology. This empowers web scraping tools to identify patterns within web content. Consequently, the efficiency of web scraping tools has been improved, particularly in cases where web content is disorganized or unclear.

Overall, the field of optical character recognition and web scraping technologies has experienced significant advancements in recent years. These developments have greatly enhanced the accuracy and efficiency of these techniques, making them indispensable in various domains.

Toxicity in Online Communication

Online platforms have become an increasingly prominent means of communication. Despite the obvious benefits to the expanded distribution of content, the last decade has resulted in disturbing toxic communication, such as cyberbullying and harassment. Nevertheless, detecting online toxicity is challenging due to its multi-dimensional, context sensitive nature. As exposure to online toxicity can have serious social consequences, reliable models and algorithms are required for detecting and analyzing such communication across the vast and growing space of social media. Online social media platforms are arguably among the most culturally significant technological innovations of the 21st century. The many benefits include widespread distribution of content that crosses geographical boundaries, enabling interaction and exchanges that are almost free of physical restrictions except infrastructure. Communities have sprung up around every conceivable special interest, from science to travel, from politics to child-rearing. The easy spread of data, information and knowledge was expected to enhance informed decision-making, cultural exchanges, and coordination of activities online and in the physical world. Unfortunately, social media has also dramatically enhanced the reach and volume of harmful content, including misinformation, conspiracy, extremism, harassment, violence, and other forms of socially toxic material. While social media platforms attempt to counter such harmful content, their efforts are largely ineffective and thus have the potential to create unintended negative impact. The effectiveness of moderation is likely to be biased by the platforms' economic interests or political and regulatory considerations. Or perhaps failure is simply due to a lack of effective tools and adequate investments. Regardless of the reason, moderation in human content has produced relatively unsatisfactory results, Although the political and public health climate of 2020 has encouraged society to adopt technological and specifically AI-based solutions, success has also been limited.

The problem of detecting toxicity is not a problem of pure computer science or artificial intelligence. To identify toxicity, it is necessary to understand the broader

context that goes beyond the situation and analyze domain-specific content, with reference to applied human values, social norms and culture at the individual, group, and community levels.

Toxicity detection is a multidisciplinary problem that relies on theory, experimental models, and knowledge to guide classification.

Toxicity detection takes the form of two problems: discovering the toxic source(s) and identifying the vulnerable victim. For both problems, we need more sophisticated natural language processing (NLP) and machine learning (ML) methods to detect and use features that indicate toxicity. Since the meaning of content is assigned based on the belief system of the source and target, semantic meaning must be computationally represented separately,

Hateful commenting, also known as 'toxicity', frequently takes place within news stories in social media. Yet, the relationship between toxicity and news topics is poorly understood. To analyze how news topics relate to the toxicity of user comments, we classify topics of 63,886 online news videos of a large news channel using a neural network and topical tags used by journalists to label content. We scored 320,246 user comments from those videos for toxicity and compare how the average toxicity of comments varies by topic. Findings show that topics like Racism, Israel-Palestine, and War & Conflict have more toxicity in the comments, and topics such as Science & Technology, Environment & Weather, and Arts & Culture have less toxic commenting. Qualitative analysis reveals five themes: Graphic videos, Humanistic stories, History and historical facts, Media as a manipulator, and Religion. We also observe cases where a typically more toxic topic becomes non-toxic and where a typically less toxic topic becomes "toxicities" when it involves sensitive elements, such as politics and religion. Findings suggest that news comment toxicity can be characterized as topic-driven toxicity that targets topics rather than as vindictive toxicity that targets users or groups. Practical implications suggest that humanistic framing of the news story (i.e., reporting stories through real everyday people) can reduce toxicity in the comments of an otherwise toxic.

Pictures and analysis are available in the following source.

https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0228723#pone.0228723.ref003

Connecting with people anywhere in the world, we have access to more knowledge than we could ever consume, and from gaming to Netflix, we have unlimited avenues of entertainment. Why, then, does the Internet – a force for social connection, knowledge, and entertainment – breed a culture of negativity

The term "online toxicity" encompasses rude, aggressive, and degrading attitudes and behavior that are exhibited on online platforms. It can range from excessive use of profanity to outright hate speech. It is generally observed in the context of online interactions between one or more individuals. The prevalence of the issue, coupled with its potentially detrimental consequences, has led to increasing concern in the past few years. A starting point for addressing this negative behavior has been to ask why it occurs in the first place. From there, behavioral scientists have attempted to develop interventions to cut it off. We still have a long way to go in that regard, with online toxicity running rampant on social media and in online gaming communities. However, through increased awareness of the consequences of toxic behavior and interventions based on screening algorithms and behavioral science, progress has been made.

Challenges in Automated Text Analysis

Handling unstructured text and evolving language patterns can create several challenges. Here are some key difficulties:

- **Lack of standardization:** Unstructured text can vary significantly in terms of formatting, grammar, and vocabulary. This lack of standardization makes it challenging to extract meaningful information and derive insights from the text.

- **Ambiguity and context dependency:** Language is inherently ambiguous, and the meaning of words or phrases can vary depending on the context. Understanding the intended meaning requires considering the broader context, which can be complex and challenging to capture accurately.

- **Slang and colloquialisms:** Language patterns constantly evolve, with new slang terms, colloquial expressions, and abbreviations emerging regularly. These linguistic nuances can be difficult to interpret, especially for automated systems that may not be updated frequently enough to capture the latest language trends.

- **Understanding idioms and figurative language:** Unstructured text often contains idioms, metaphors, and other forms of figurative language. Interpreting and understanding these expressions requires a deep understanding of the cultural and contextual factors associated with them.

- **Handling misspellings and typos:** Unstructured text, especially user-generated content, often contains misspellings, typos, and grammatical errors. Dealing with these variations and normalizing the text can be

challenging, as they can affect the accuracy of language processing algorithms.

- **Dealing with noise and irrelevant information:** Unstructured text can contain noise, such as irrelevant or redundant information, advertisements, or spam. Filtering out this noise and focusing on the relevant content is crucial for accurate analysis and understanding.

- **Adapting to evolving language patterns:** Language patterns and usage change over time. This poses a challenge for natural language processing systems that may struggle to adapt quickly to new language trends and patterns.

Addressing these challenges requires advanced natural language processing techniques, machine learning algorithms, and constant monitoring and updating of language models to stay up to date with evolving language patterns.

Moreover, there are limitations in current technologies for dealing with these challenges like:

- **Lack of adaptability:** Current technologies often struggle to adapt to evolving language patterns and slang. They may rely on static language models that become outdated over time and fail to capture the latest linguistic trends accurately.

- **Insufficient context understanding:** Existing technologies may have limited capabilities to understand the context of unstructured text accurately. This limitation hinders their ability to provide relevant and meaningful responses, especially in complex or ambiguous situations.

- **Difficulty in handling informal language:** Slang and colloquial language can be challenging for current technologies to handle. They may struggle to interpret non-standard language patterns and may produce incorrect or irrelevant responses when faced with informal language input.

- **Bias and inaccuracy:** Due to the lack of diverse and representative training data, current technologies may exhibit biases and inaccuracies when dealing with unstructured text. This can result in the propagation of stereotypes, misinterpretation of intent, and inadequate understanding of user queries.

- **Limited generalization:** Existing technologies may lack the ability to generalize well across different domains, languages, or cultures. They may perform better on specific types of unstructured text but struggle when faced with unfamiliar contexts, languages, or cultural nuances.

Overall, while current technologies have made significant advancements in handling unstructured text, evolving language patterns, and slang, there are still limitations that need to be addressed for more accurate and comprehensive understanding and generation of unstructured text.

Advancements in Natural Language Processing (NLP)

Natural language processing (NLP) refers to the branch of computer science—and more specifically, the branch of artificial intelligence or AI—concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.

Text extraction using natural language processing:

Text extraction using natural language processing In Natural Language Processing (NLP) or Natural Language Understanding (NLU), Information extraction is the technique to locate and extract relevant and important information from the structured text.

As the field of artificial intelligence advances, so does machine learning's ability to interpret and extract information from human language. This is especially important in the field of natural language processing (NLP), where machines are tasked with understanding unstructured text data. There are several natural language processing techniques that can be used to extract information from text or unstructured data, and here are some examples.

them. These techniques can be used to extract information such as entity names, locations, quantities, and more. With the help of natural language processing, computers can make sense of the vast amount of unstructured text data that is generated every day, and humans can reap the benefits of having this information readily available. Industries such as healthcare, finance, and ecommerce are already using natural language processing techniques to extract information and improve business processes.

Let's look at a few natural language processing techniques for extracting information from unstructured text:

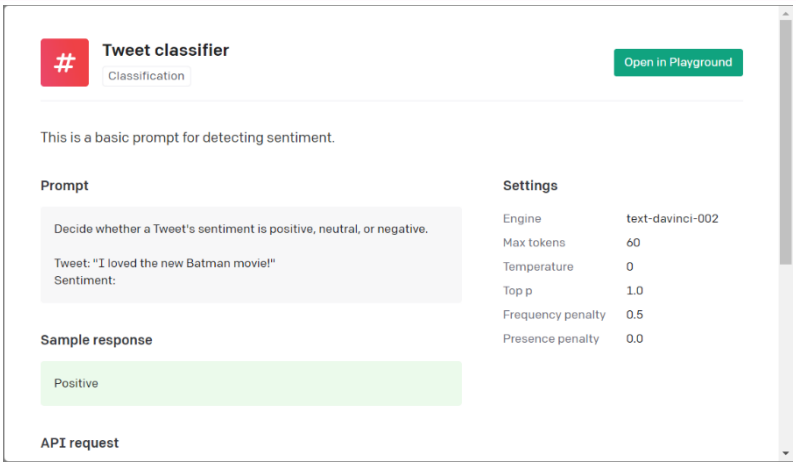
1. Named Entity Recognition using spaCy

Named entity recognition (NER) is a task that is concerned with identifying and classifying named entities in textual data. Named entities can be a person, organization, location, date, time, or even quantity. SpaCy is a popular Natural Language Processing library that can be used for named entity recognition and number of other NLP tasks. It comes with pretrained models that can identify a variety of named entities out of the box, and it offers the ability to train custom models on new data or new entities.

For the most part, NER models are trained on a per-token basis. That is, for each word in a sentence, the model predicts whether or not that word is a named entity that we want to fine. In spaCy, this is done using a bi-LSTM neural network that takes as input a sequence of words, and for each word it predicts whether it is a named entity. It then uses the information from the words around it to make a more informed prediction.

We can use things such as part-of-speech tags, dependency parse trees, and entity type information to help the bi-LSTM neural network make more accurate predictions as it works to learn the relationship between language and named entities.

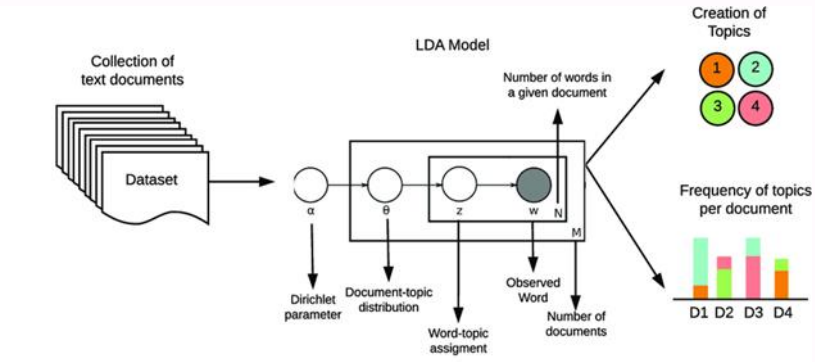
2. Sentiment analysis with GPT-3



GPT-3 is an autoregressive language model used for a wide variety of tasks including sentiment analysis. When given a sentence, GPT-3 will analyze the sentiment and generate a prediction. The predictions are made by considering the context of the sentence as well as the word choices. An example would be a text document that contains strong negative connotations such as "hate" or "I'm not a fan of them" which is likely to be predicted as having a negative sentiment. GPT-3 is not only able to predict the sentiment of a sentence, but it can also generate an explanation for its prediction. This makes GPT-3 a powerful tool for sentiment analysis, as it can provide not only a prediction, but also an explanation for that prediction. This can clarify why a particular sentence was predicted to have a certain sentiment and can also help in troubleshooting data science errors.

Sentiment analysis is already used for things such as social media monitoring, market research, customer support, product reviews, and many other places where people talk about their opinions. We've had a ton of success building these applications like this one for Twitter.

3. Topic modelling with Latent Dirichlet Allocation



Latent Dirichlet allocation (LDA) is a topic modeling technique that is used to discover hidden topics in text such as long documents or news articles. It does this by representing each document as a mixture of topics, and each topic is represented as a mixture of words. LDA is an unsupervised learning algorithm, which means that it does not require training on new labeled data. This makes it a powerful tool for discovering hidden structure in data that can be used quickly. LDA allows you to find out what topics are being talked about in a document, and how often those topics are mentioned. It can also be used to find out what words are associated with each topic.

With the rise of social media, online platforms have become hubs for self-expression and connection worldwide. However, some users take advantage of the anonymity afforded by the internet to spread harmful information, including hate speech, cyberbullying, and child abuse images. As platforms aim to cultivate safe and inclusive communities, social media moderation has become essential. Natural language processing (NLP), a branch of artificial intelligence focused on human language, offers an automated solution for detecting and eliminating harmful content at scale.

NLP allows computers to analyze, understand and generate human language. It uses machine learning algorithms trained on huge datasets to recognize patterns in language and classify content. This makes NLP well-suited for content moderation, where platforms must quickly identify and remove hate speech, spam, terrorist propaganda and other inappropriate material at massive scale.

Simple keyword matching and algorithmic techniques struggle to capture nuanced meanings and linguistic complexities. They often fail to detect coded hate speech or miss harmful content conveyed through ambiguous euphemisms. But advanced NLP, especially neural networks, and deep learning, provide sophisticated solutions for handling natural language in all its complexity.

Context-aware NLP considers relationships between words and how meaning changes based on context. The phrase "Let's kill it!" could be fine when referring to a challenging work task but threatening when directed at a person. Contextual NLP reduces false positives and addresses emerging patterns in harmful speech, even when hateful language isn't used directly.

However, human reviewers still provide essential oversight and feedback. While AI handles initial screening at scale, experts review edge cases and samples of decisions. This hybrid system, employed by AI-leaders like Appen, combines speed and accuracy, handling high volumes of posts but also nuanced, complex cases. The result is moderation that is fast, precise, and constantly improving to address new challenges.

With social media's widespread use, NLP has become crucial infrastructure for online communities. When implemented responsibly, NLP helps platforms curb harm while enabling authentic discourse and connection. The algorithms are continuously being learned in pursuit of kinder, more inclusive digital spaces where all voices can be heard.

Machine Learning Techniques in Text Analysis

The rise of machine learning (ML) has revolutionized our ability to analyze and understand textual data. By leveraging various algorithms and frameworks, we can now extract valuable insights from vast collections of text, empowering diverse applications across numerous fields. In this concise overview, we shall explore several key ML techniques utilized in text analysis:

Text Classification:

Technique: Algorithms for supervised learning (Naive Bayes, Support Vector Machines and varieties of Decision Trees as well Neural Networks) are all trained on labeled datasets. They separate text into predefined categories or classes.

Applications: Spam detection, sentiment analysis and topic categorization.

Strengths: Efficiently categorizes large datasets, tackling spam and sentiment analysis head-on.

Weaknesses: Reliant on well-labeled data, vulnerable to biases and misinterpretations in nuanced language

.

Named Entity Recognition (NER):

Technique: Sequence labeling algorithms are used to identify and categorize entities (names, places or organizations) that appear in text. Typically, they utilize conditional random fields or recurrent neural networks.

Applications: Information extraction, entity linking.

Strengths: Pinpoints important entities like locations and organizations, enriching information extraction.

Weaknesses: Accuracy plummets with complex domain-specific jargon or informal language.

Sentiment Analysis:

Technique: Supervised learning models for sentiment classifiers are designed to guess what tone some piece of text takes (positive or negative).

Applications: Social media monitoring, customer feedback analysis.
Strengths: Gauges public opinion and brand perception, guiding marketing and customer service strategies.

Weaknesses: Can be easily misled by sarcasm, irony, and cultural nuances, leading to misinterpretations.

Text Clustering:

Technique: K-Means, hierarchical clustering and other unsupervised learning algorithms classify documents or sentences of similar content into groups.

Applications: Document categorization, topic discovery.

Strengths: Uncovers hidden patterns and groups related documents, aiding in efficient information retrieval.
Weaknesses: Sensitive to outliers and noise in data, potentially grouping unrelated texts together.

Topic Modeling:

Technique: Topics in a collection of documents are identified using LDA and other probabilistic models.

Applications: Document summarization, content recommendation.

Strengths: Discovers latent themes and trends within vast document collections, driving content recommendations and trend analysis. **Weaknesses**: Overlooks subtle relationships between topics, susceptible to generating misleading themes in ambiguous data.

Text Generation:

Technique: Generative models (RNNs and Transformers) are trained to produce text like human beings.

Applications: Chatbots, content creation, language translation.

Strengths: Mimics human writing styles, fueling chatbots and creative writing tools.

Weaknesses: Prone to generating nonsensical or misleading text, especially when trained on low-quality data.

Text Summarization:

Technique: The difference is that extractive summarization selects important sentences from a document; abstractive creates concise summary on its own.

Applications: Document summarization, news article summarization.

Strengths: Condenses key points from lengthy documents, saving time and effort.

Weaknesses: Extractive summarization can miss crucial details, while abstractive methods can invent information.

Future Directions and Emerging Technologies

The latest technology is shaping the future of digital safety.

The scale of harm online is growing, and bad actors are becoming more sophisticated when perpetrating such harm. The Internet Watch Foundation (IWF), which works to tackle child sexual abuse material (CSAM) online, found 252,194 URLs containing or advertising CSAM in 2021, a 64% increase from 2020.

When it comes to terrorist content, platforms have been weaponized to live stream terrorist attacks from the Christchurch massacre, the synagogue in Halle, and, more recently, the Buffalo rampage. There has also been a concerning growth in cyberbullying, with the U.S. having the highest rate of racially motivated bullying online.

With almost 3.8 billion people now online globally, the digital world offers significant benefits, but also poses the risks of harmful content.

The Global Alliance for Responsible Media (GARM), created by the World Federation of Advertisers, is scaling its impact by partnering with the World Economic Forum's platform for Media, Entertainment and Culture to improve

the safety of digital environments, addressing harmful and misleading media while protecting consumers and brands.

Client-side scanning (CSS) broadly refers to systems that scan message content (e.g. pictures, text or video) for matches to a database of illegal or objectionable content before the message is sent to the intended recipient on an encrypted channel. An example of this is anti-virus software used to prevent your computer from being infected by malware.

In recent discussions about tackling CSAM and other illegal material, CSS has become a hot topic as it is seen by some to find this material without breaking the technology behind end-to-end encryption (E2EE). There are two main methods for CSS: the first is on-device and the second is on a remote server.

Ian Stevenson, CEO of Cya comb and Chair of the Online Safety Tech Industry Association (OSTIA), states: “Over the past few months we have seen some really good quality exploration of what is possible with client-side and split or multi-party compute technologies. This technology doesn’t tamper with the encrypted part of the system, leaving all its protections intact. There is no ‘back door.’ Instead, these technologies act as a border check for content entering and leaving the encrypted domain and do so in a way that maintains the privacy of the user. The content they are sending or receiving cannot be identified, tracked, or matched by any third party”.

Many organizations, such as Access Now and the Internet Society, have voiced their concerns about CSS, however. Access Now wrote a letter to the European Commission highlighting the risks to privacy, security, and expression. Stevenson and other experts (including the heads of GCHQ and NCSC) do not share these concerns. Stevenson says that these technical capabilities for matching and blocking known CSAM provides excellent privacy protection for users and suggests that metadata leakage from mainstream E2EE apps is far more of a threat to privacy than these newly developed systems.

In addressing fears that certain governments could use CSS technologies to suppress free speech or identify dissidents, he suggests that these should be considered in the context of existing threats, rather than in isolation.

“Autocratic governments intent on blocking content are unlikely to be very concerned about protecting privacy and, therefore, could easily mandate application of various solutions that exist today. The additional risk arising from deploying these new technologies is very small, with huge potential benefit to society,” he says.

Australia’s safety Commissioner, Julie Inman Grant, argues that there is a need to look at safety, privacy, and security as the three pillars of digital trust. “It is important to balance safety with privacy and security – they are not mutually exclusive and healthy tensions amongst these imperatives can lead to much better outcomes… But, to continue to pit privacy against safety or as values that are mutually exclusive is totally missing the point – in many cases reported to us, particularly in image-based abuse, privacy and safety are mutually reinforcing,” she says.

Artificial Intelligence and Natural Language Processing (NLP) Models

Artificial Intelligence (AI) systems can help increase the speed and scalability of content moderation by automating content moderation processes, as well as the detection of a range of harmful content through Natural Language Processing (NLP) models. One of the big challenges to its advancement, however, according to Bertie Vidgen, CEO and Co-Founder of Rewire, is that every platform is different – they have different users, different kinds of content, different media, different hazards, and different norms. This is a huge problem for developers because the traditional one-size-fits-all approach in software development just doesn’t work.

“Over the past two years, we’ve seen the emergence of incredibly powerful models that can do ‘zero shot’ and ‘few shot’ learning. Practically, these advances mean that software can achieve very high performance with relatively little data. We have a way to go, but this has opened exciting new possibilities to create scalable AI that is fully customized to each platform,

without the huge costs and development timelines that would otherwise be needed," Vidgen says.

There is still skepticism and distrust amongst some around the use of AI for online safety given a lot of software has struggled to handle issues such as nuance, intent, context, and jokes. Increasing reliability, flexibility, cost-efficiency, and accuracy of AI – together with human supervision to create effective feedback loops – will help increase its uptake.

Justin Davis, CEO and Co-Founder of Spectrum Labs, highlights how better detection of toxic behavior paves the way for measurement and transparency tools that help online platforms make better policy decisions and create better user experiences. "When that's combined with the ability to identify and encourage healthy behavior, trust and safety teams can align with the customer experience and product teams in a data-driven way to reinforce #SafetyByDesign principles," Davis says.

He believes investing in NLP and AI tools today will help the industry stay ahead of the curve against emerging threats and drive the growth of healthier communities online.

Summary

In this chapter we discussed the growth of the internet and its impact on communication and content sharing and explored how this evolution has led to the current challenges of online toxicity, then we reviewed studies on the prevalence and impact of internet toxicity and examined research on the psychological and societal effects of online hate and toxic behavior.

We also surveyed existing literature on text extraction methods and technologies and reviewed advancements in Optical Character Recognition (OCR) and web scraping technologies. We have analyzed studies on the nature and forms of toxicity in online spaces and discussed the subjective nature of toxicity and its cultural dependencies.

Moreover, we reviewed existing systems and algorithms for detecting toxic language and

compared various machine learning and NLP methodologies used in toxicity classification.

We investigated challenges in handling unstructured text, evolving language patterns, and slang and discussed limitations in current technologies for dealing with these challenges then we surveyed the latest advancements in NLP related to text extraction and understanding and discuss the integration of NLP in detecting and classifying toxic content.

Finally, we reviewed the application of various machine learning algorithms in text analysis and compared the effectiveness of different algorithms in the context of toxicity detection and Identify gaps in the current literature and suggest areas for future research and discussed emerging technologies and methodologies that could enhance toxicity detection and text extraction.

Chapter 3
Analysis and Design

INTRODUCTION

In this chapter we will describe the problem and define the solution we proposed, then we will identify the system analysis component that contains functional and non-functional requirements, Context Diagram, Use case table ,Test Cases and Sequence diagram, Finally we will define the database entities and relationship between each other's.

PROBLEM DESCRIPTION

Toxic language is a common problem in online environments; it uses negativity as a weapon against people, groups, and even companies. These hurtful remarks, which range from hate speech to cyberbullying, cause a variety of wounds, including emotional pain, silence, and a decline in trust. Due to their shortcomings in precision, understanding of context, and flexibility, current

solutions find it difficult to tackle this evil. We suggest a state-of-the-art text extraction and classification system as a remedy. This system will surpass constraints by pulling text from many sources, utilizing AI to analyze the meaning of every word, and providing useful information on toxicity levels. Picture a safe refuge on the internet where communities flourish, negativity is recognized and reported, and dialogue is cleaned up. Together, we will create that future by addressing this poisonous text problem head-on.

SCOPE

The proposed system is designed especially for users or social media developers to help them clear the website or the files from toxicity making it clear and safe place

Our algorithm recognizes offensive language and adjusts to linguistic changes to identify everyday threats and hate speech. However, slang and memes may still find their way in.

This condenses both elements' breadth and limits into a succinct, captivating phrase. Please feel free to modify it to fit the unique features of your system.

PROPOSED SOLUTION

- Contextual analysis: Understand the meaning of words within their surrounding text and intent to reduce false positives from sarcasm or figurative language.
- Sentiment analysis: Identify the overall sentiment of text to pinpoint hateful or harmful expressions even if they don't contain explicit keywords.
- Machine learning and AI: Continuously train and improve algorithms on diverse datasets of toxic language to adapt to evolving trends and slang.
- Reporting and flagging tools: Encourage users to report toxic content and provide clear guidelines on appropriate behavior.

- Moderation tools: Make moderation tools easily accessible and equip moderators with training and resources to handle sensitive situations effectively.
- Educational campaigns: Promote awareness about online safety, responsible communication, and the impact of harmful language.
- Collaborative lexicon development: Work with linguists and communities to identify and incorporate evolving slang and meme terminology into detection models.
- Contextual understanding of trends: Train algorithms to recognize the context and intent behind slang and memes, differentiating between playful humor and harmful expressions.
- Human-in-the-loop systems: Combine automated detection with human review and feedback to ensure accurate interpretation of nuanced language and cultural references.

DATA GATHERING

Prior to implementing the proposed model, a crucial step involves gathering representative and dependable data to construct a robust model for text classification and toxicity detection. The collected data will be categorized as follows:

Primary Data:

- Text Samples: We will solicit text samples from various sources, including:
- Online forums, social media platforms, and comment sections.
- Publicly available datasets specifically designed for toxicity detection.
- Internal datasets, if available, containing text that has been previously labeled toxicity.

•Annotations: Expert annotators will carefully label the collected text samples, identifying and classifying instances of toxicity. This will provide the ground truth necessary for training and evaluating the model.

Secondary Data:

- External Resources: We will leverage existing text classification and toxicity detection datasets to augment the primary data and potentially enhance model performance. These datasets may include:
- Large-scale, publicly available datasets like the Toxic Comment Classification Challenge dataset from Kaggle.
- Datasets from academic research or industry efforts focusing on toxicity detection.

SYSTEM ANALYSIS

- **Actors**
  - **User**
  - **Admin**
- **Function Requirements**.
  - **User Signup.**

    User will be able to enter some data like first name, last name, and email to be able to access their own functionality through a dashboard.
  - **User Login.**

    The User will be able to enter their mail and password into the site to go their dashboard page.
  - **User validation.**

The system check and validate User's credentials and if correct they are logged in.

- **Forgot password for User.**

  User will receive an email containing a link to reset their password on their account.

- **Update Password in database.**

  Once the user fills in the new password and confirms the password the new password will be updated in the database.

- **Classify plain text .**

  the user can write/paste down any plain text in the text box and will proceed directly to the classifier

- **Upload desired file.**

  The User's information page should include upload button which allow the user to upload the file that needs to get classified.

- **Extract text from file.**

  The user can get the text out of the file he provided by ocr in image files and python libraries in documents as docx and pdf

- **Classify extracted files text.**

  After the user gets the text from the previous function he can proceed to classify the extracted text

- **Delete user submission.**

  user can delete his submission]in history page.

- **Get Results.**

  The system gets the results from module with percentage of toxic words in the text or file submitted by the user

- **Add user history.**

  users will be able to add any classified submission to their history list.

- **Print Report.**

Enter the ID of the report that you want to print.

- **Added the wrong email in the forget password search.**

If the user added the wrong email in the forget password search bar error message will be shown.

- **Non-Function Requirements**

- **Product requirement:**
  - **Reliability Requirements**
    - system shall be available 24/7 to ensure users can access it at any time.
    - The system shall maintain 99.9% uptime, with minimal downtime for maintenance or updates.

  - **Portability Requirements**

- The system shall be accessible through commonly used web browsers (Chrome, Firefox, Safari, Edge).
- The system shall function on diverse devices, including desktops, laptops, tablets, and smartphones.
- Performance Requirements
- The system shall process text extraction and classification within 5 seconds for typical text inputs.

•The system shall scale to accommodate a growing number of users and concurrent requests without significant performance degradation.

- **Usability Requirements**

- The system shall offer an intuitive and user-friendly interface for both technical and non-technical users.
- Clear instructions and guidance shall be provided for all system functionalities.
- The system shall provide informative feedback to users regarding the results of text classification, including a visual representation of toxicity levels.

- **Organizational requirement:**
  - **Software Delivery Requirements.**
    - The complete project will be delivered on 1/6/2024.
  - **Software Implementation Requirements.**
    - system shall adhere to industry-standard security measures to protect user data and privacy.
    - The system shall follow accessibility guidelines to enable use by people with disabilities
  - **Software Standards Requirements**
    - The website must maintain a high level of security to keep user data secure.

- The website must be able to load pages quickly and efficiently. External requirement:

➢ **Interoperability Requirements**

- The website or website functionalities must be accessible through other websites.

➢ **Ethical Requirements**

- The system shall provide an API for integration with other software systems and platforms.

➢ **Legislative Requirements**

- Privacy Requirements
  - The system shall collect only the minimum data necessary for its functionality..
- Safety Requirements
  - Sensitive user data must be protected from any malicious activity.
  - The system shall be resilient to cyberattacks and data breaches.

SYSTEM MODELS

- **Context Diagram.**

The website component sends the information of the user to the system, when the user enters plain text or uploads a document containing text to be analyzed. The website component seamlessly passes this text to the TEXTC system, ready for processing.

When the user submits the query, the System performs the following steps in the background:

- The System retrieves the text/document that are submitted by the user. The submission could be plain text, image, doc type.
- The System extracts the user submission as plain text. Then saves it to the data base query so user can access it anytime.
- The System returns the classified text results. The classified results typically include percentage and type of the classified toxic text .

Also for the user to use the Text extraction and toxic classification system he should upload readable clear image / doc to the system through the website and then we apply the Ai analyze model on the text to classify and return it to the website.

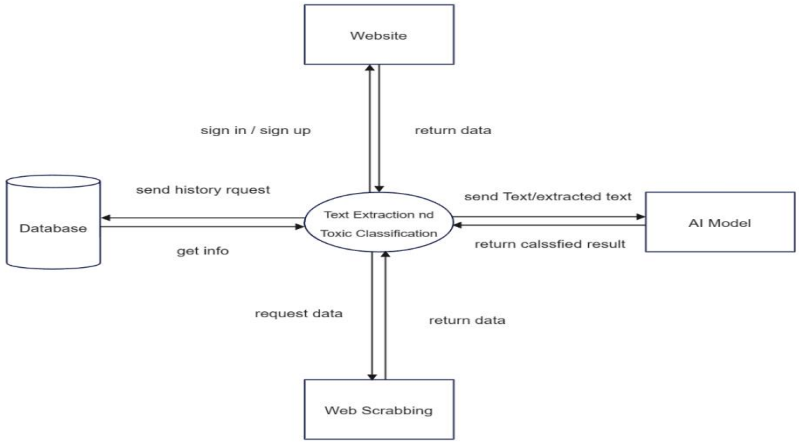


Figure 1: Context Diagram

- **Data Flow Diagram**

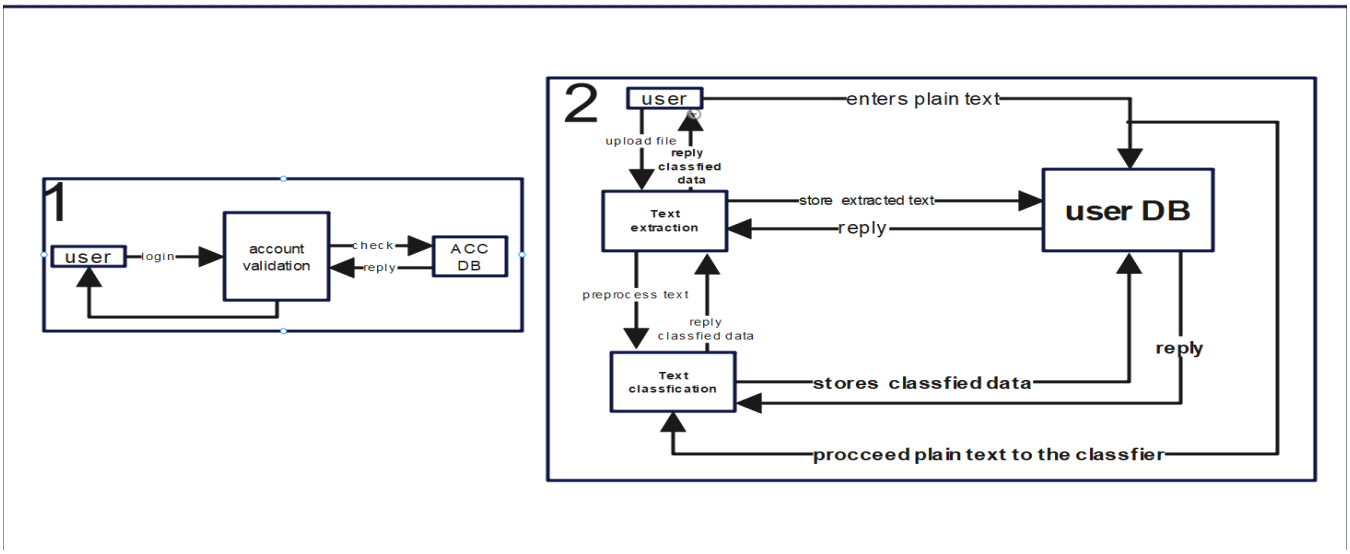


Figure 2: Data Flow Diagram

- **Use Case Diagram.**

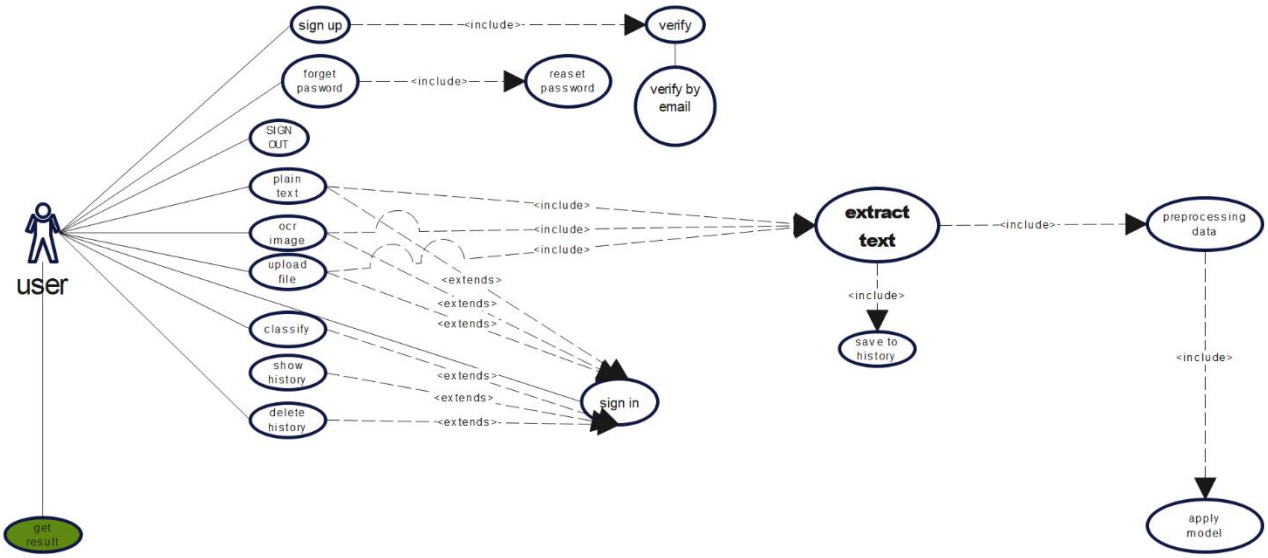


Figure 3: Use Case Diagram

- **Class Diagram**

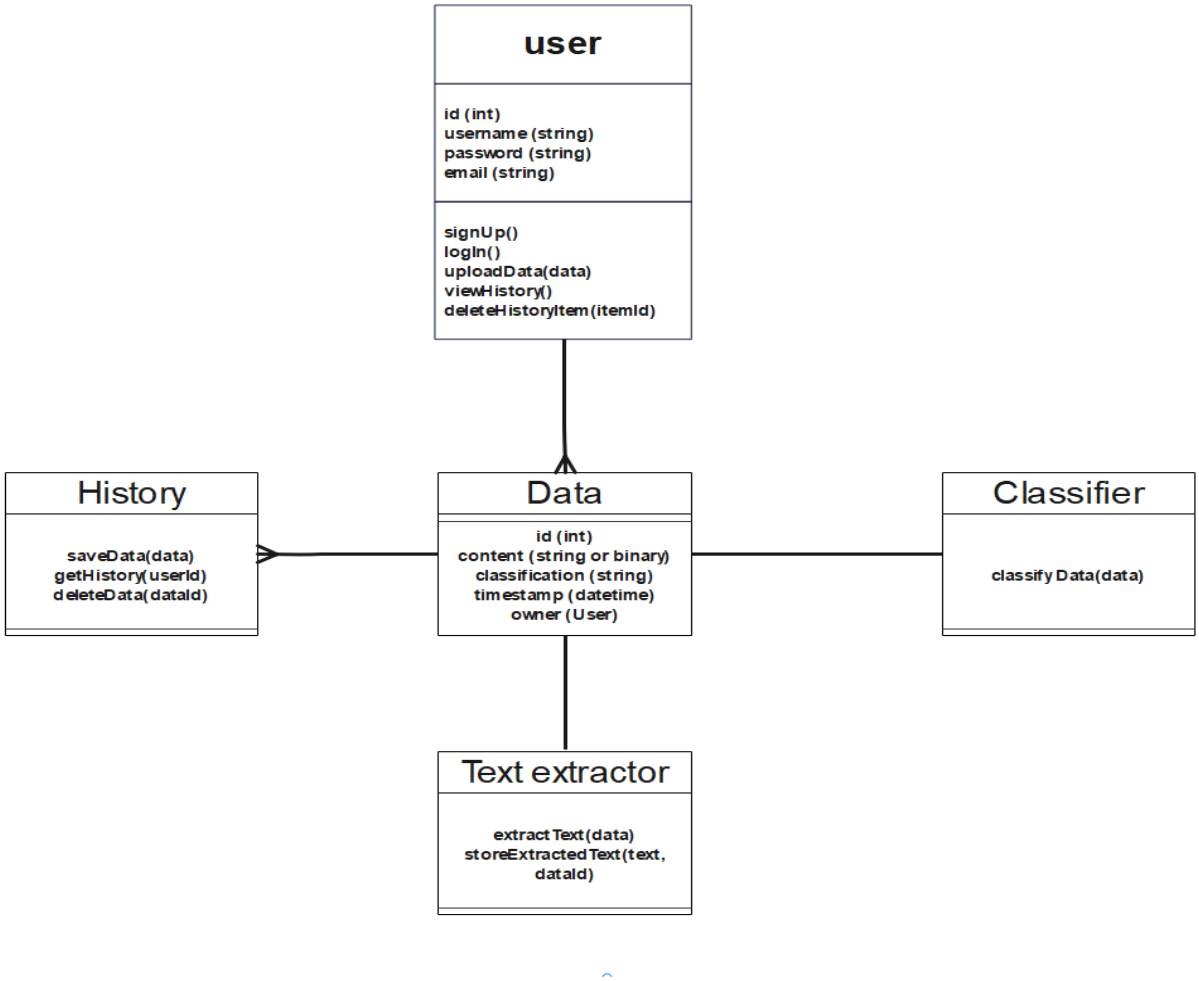


Figure 1: Class Diagram

- **ERD**

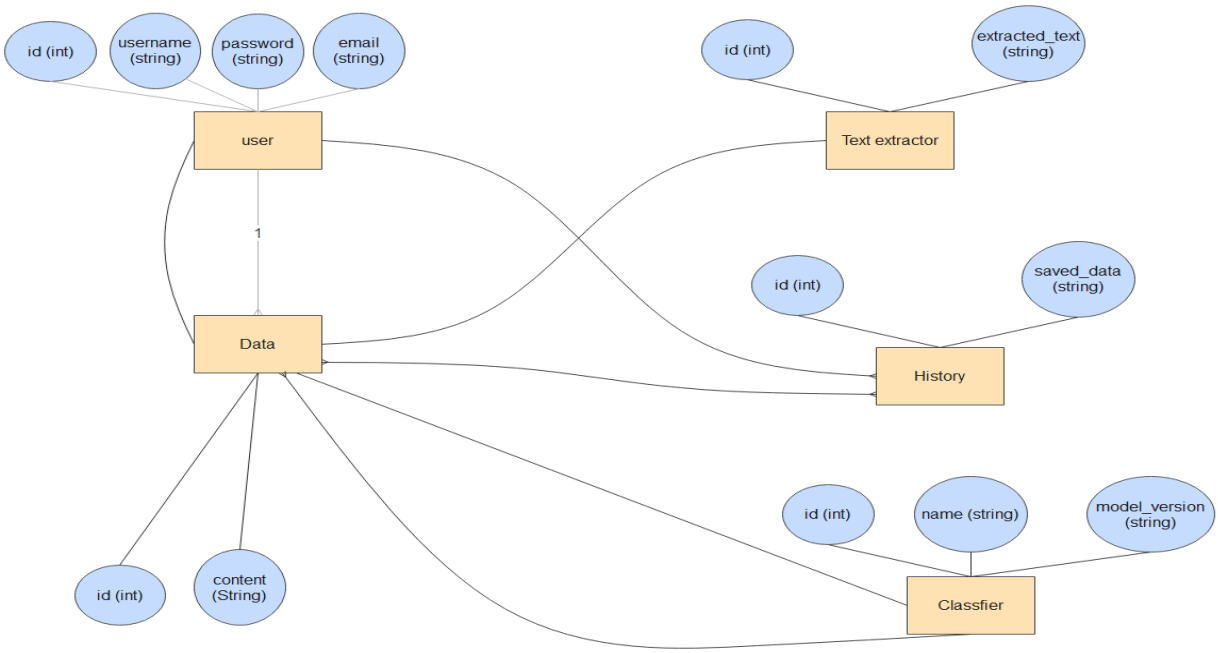


FIGURE 2: ERD

- **Use Case Tables**

Use Case ID	**1**
Use Case Name	**Sign up**
Actors	**User**
Goal	Create a new user account
Preconditions	None
Main Success Scenario	User enters username, password, and email. System verifies information and creates account. System sends confirmation email.
Extensions	Username already exists: System prompts for a different username. Email already exists: System prompts for a different email. Email sending fails: System displays an error message.
Postconditions	User account is created and active.

Use Case ID	**2**
Use Case Name	Login
Actors	User
Goal	Authenticate and access the system
Preconditions	User has a valid account
Main Success Scenario	User enters username and password. System validates credentials. System grants access to the system.
Extensions	Invalid credentials: System displays an error message. Account is locked: System displays an error message.
Postconditions	User is logged in and can interact with the system.

Use Case ID	**4**
Use Case Name	Forget Password
Actors	User
Goal	Reset password for a forgotten one
Preconditions	User has a valid account
Main Success Scenario	User selects "Forgot Password" option. User enters their email address. System sends a password reset link to the email. User clicks the link and enters a new password. System resets the password. System displays a confirmation message.
Extensions	Invalid email: System displays an error message. Link expired: System prompts for email again. Reset fails: System displays an error message.
Postconditions	Password is reset and can be used for login.

Use Case ID	**5**
Use Case Name	Enter Plain Text
Actors	User
Goal	Submit plain text for processing and storage
Preconditions	User is logged in
Main Success Scenario	User accesses the designated text input area. User types or pastes plain text into the text box. User clicks "Submit" or "Process" button. System validates text input . System processes plain text (e.g., extracts information, classifies, stores). System displays results or confirmation.
Extensions	Invalid text input: System displays an error message and prompts for correction
Postconditions	Plain text is processed, stored, and available for further use.

Use Case ID	**6**
Use Case Name	Upload Data
Actors	User
Goal	Submit data for classification and storage
Preconditions	User is logged in
Main Success Scenario	User selects a data file to upload. System validates data format. System extracts text (if applicable).System classifies data. System stores data and extracted text. System displays classification results.
Extensions	Invalid data format: System displays an error message. Text extraction fails: System logs error and continues. Classification fails: System logs error and displays a generic message. Storage failure: System displays an error message.
Postconditions	Data is stored, classified, and available for viewing and retrieval.

Use Case ID	**7**
Use Case Name	View Extracted Text
Actors	User
Goal	View the extracted text from a data item
Preconditions	User is logged in and viewing a data item
Main Success Scenario	User selects the "View Extracted Text" option. System retrieves the extracted text for the data item. System displays the extracted text in a readable format.
Extensions	No extracted text available: System displays a message indicating no text.
Postconditions	User can view the extracted text.

Use Case ID	**8**
Use Case Name	View History
Actors	User
Goal	View past uploaded data and classification results
Preconditions	User is logged in
Main Success Scenario	User selects "History" option. System retrieves historical data for the user. System displays data items and classification labels
Extensions	No historical data: System displays a message indicating no history.
Postconditions	User can view historical data and results.

Use Case ID	**9**
Use Case Name	Delete History Item
Actors	User
Goal	Remove a data item from history
Preconditions	User is logged in and viewing history
Main Success Scenario	User selects a data item to delete. System confirms deletion. User confirms deletion. System removes data item from history.
Extensions	User cancels deletion: System returns to history view. Deletion fails: System displays an error message.
Postconditions	Data item is removed from history.

Use Case ID	**10**
Use Case Name	logout
Actors	User
Goal	Terminate the current session and revoke access to the system.
Preconditions	User is logged in.
Main Success Scenario	User clicks the "Logout" button or link. System invalidates the user's session data. System redirects the

	user to the login page or a public landing page. System displays a confirmation message (optional).
Extensions	
Postconditions	User is no longer authenticated.

SEQUENCE DIAGRAM

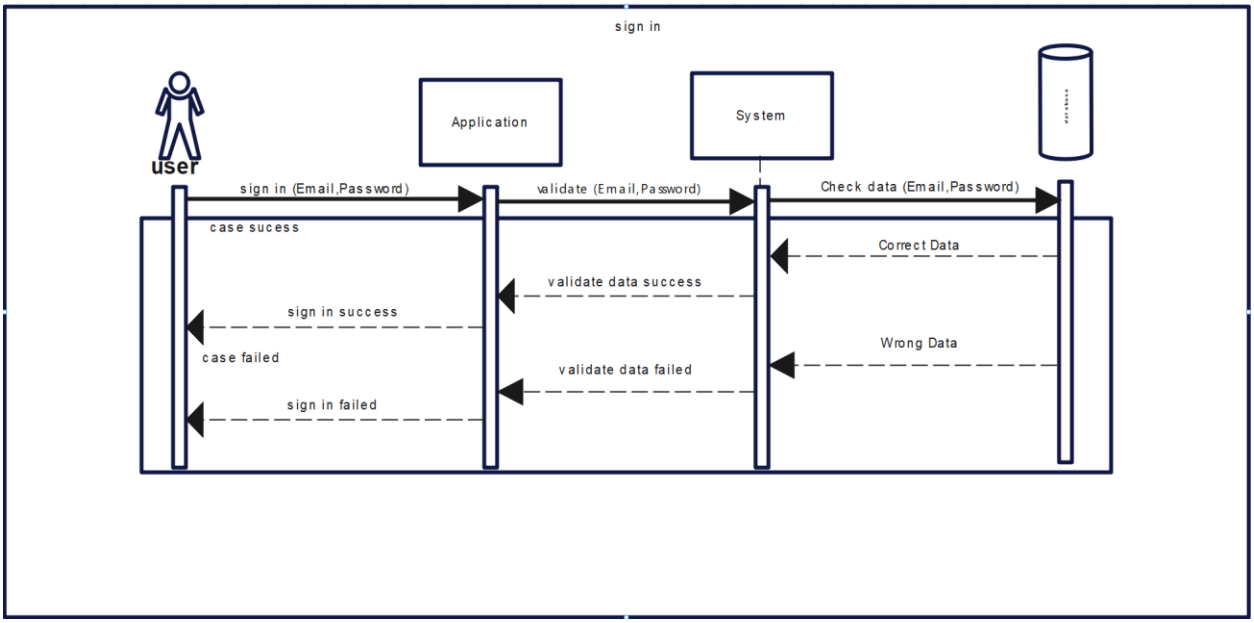


FIGURE 5: SIGN IN

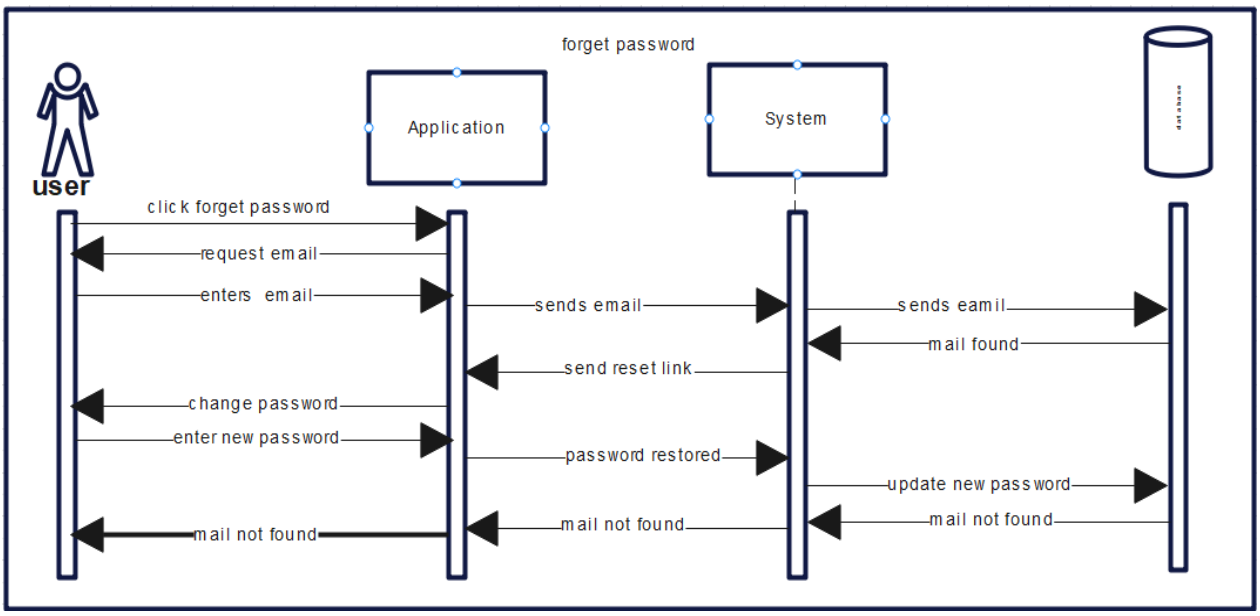


FIGURE 6: FORGET PASSWORD

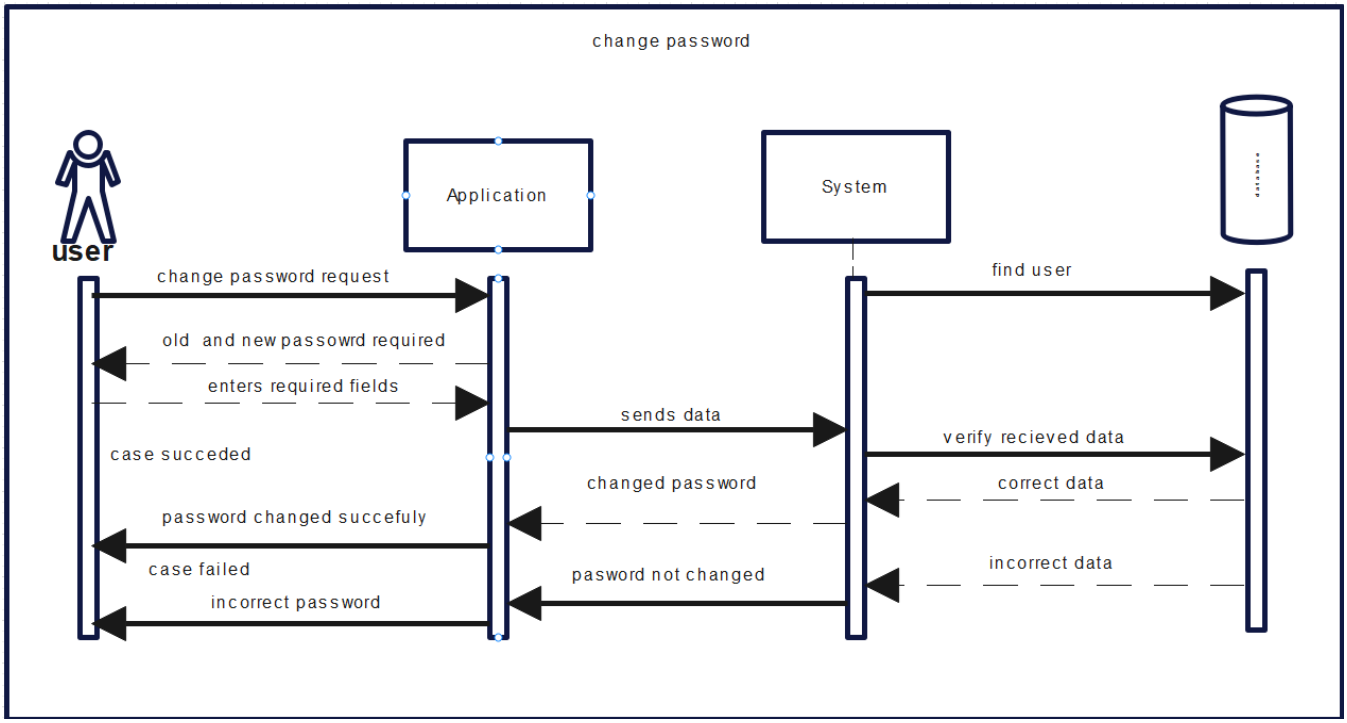


FIGURE 7: CHANGE PASSWORD

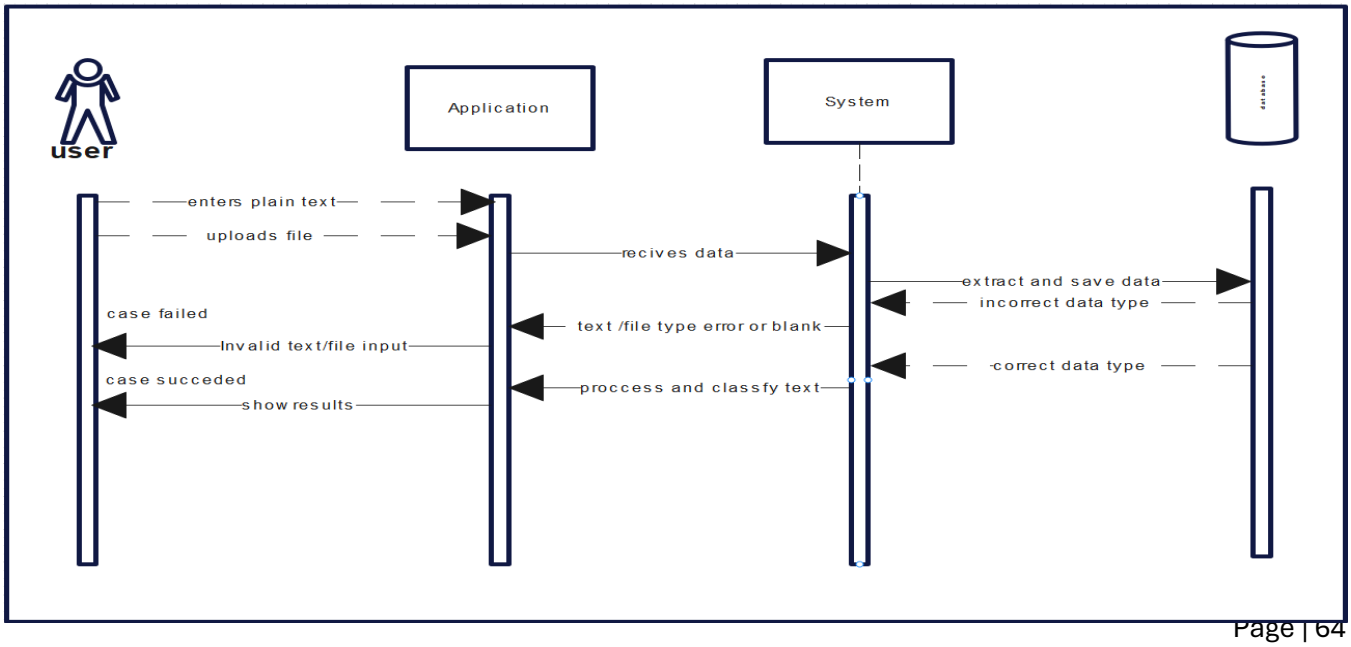


FIGURE 8: CLASSIFICATON PROCESS

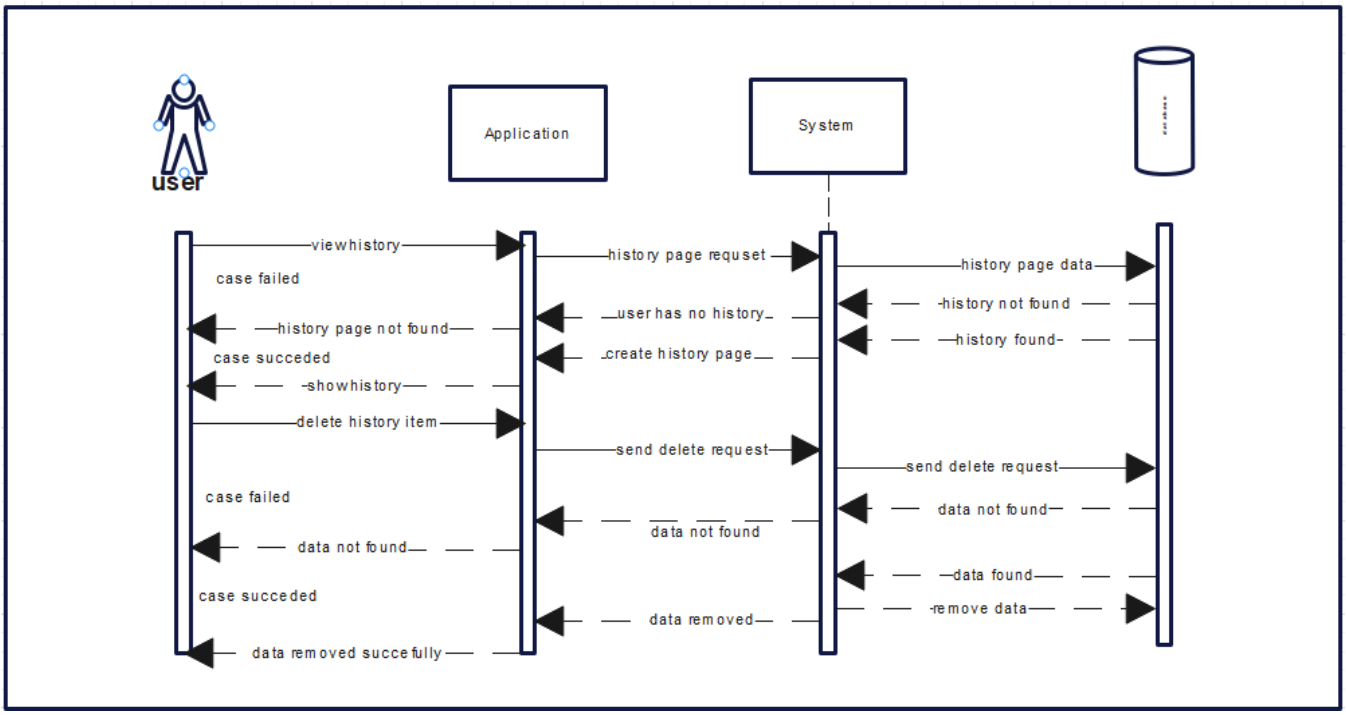


FIGURE 9: HISTORY FUNCTIONS

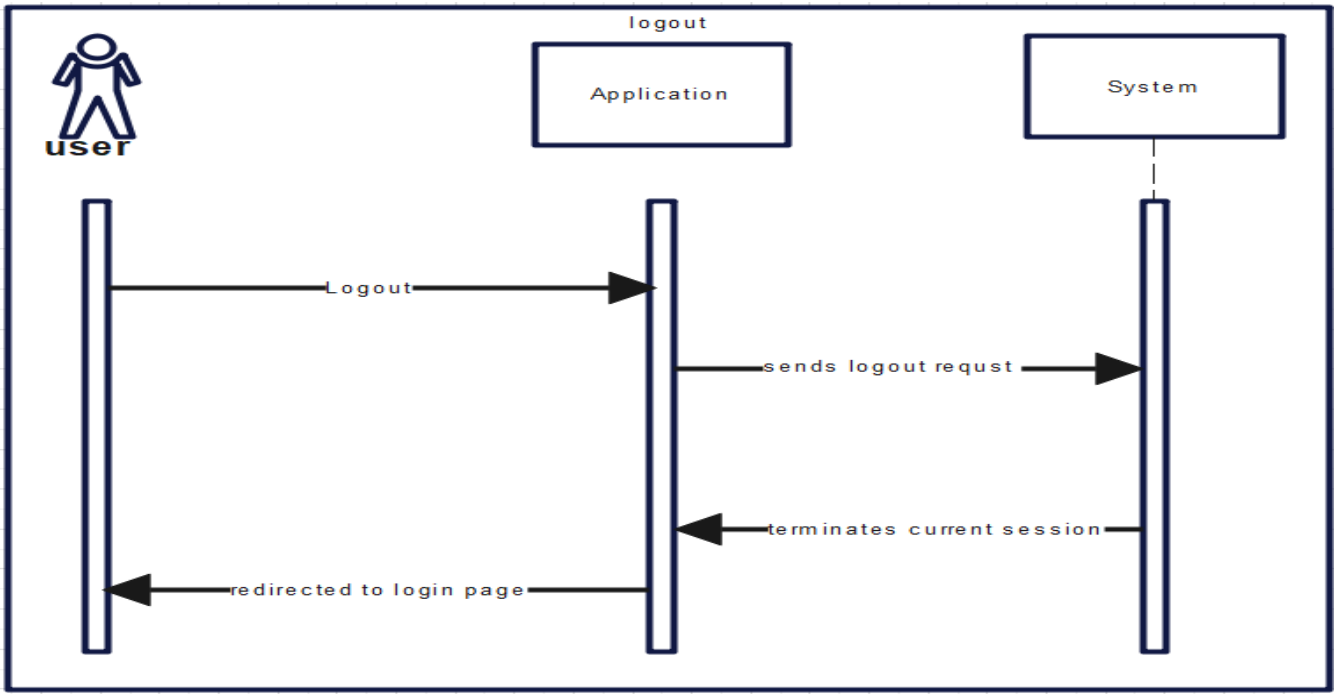


FIGURE 10 : LOG OUT

FUNCTIONAL ANALYSIS

Function ID	Function Name	Description	Input Data	Output Data	Dependencies
1	User Signup	Creates a new user account	First name, last name, email, password	Confirmation message	-
2	User Login	Authenticates user credentials and grants access	Email, Password	Access token, user data	User Registration
4	Password Reset	Resets a forgotten password	Email address	Password reset link	User Registration
5	Data Upload	Accepts data files or plain text input for processing	Data file (or text content), data format (if applicable)	Data ID, classification results, extracted text (if applicable)	Text Extractor, Classifier
6	Text Extraction	Extracts plain text from supported data formats	Data content	Extracted text	-

7	View Extracted Text	Displays extracted text for a given data item	Data ID	Extracted text	Data Storage
8	Data Classification	Classifies data based on content or extracted text	Data content or extracted text	Classification label	Classifier
9	Data Storage	Stores data, extracted text, and classification results	Data content, extracted text, classification label, user ID	Confirmation message	Database
10	History Management	Stores and retrieves user's data history	Data ID, user ID	Historical data items	

DATA PRIVILEGE

- **User**
- **Admin**

Order	Table Name	Data privilege
1.	User	Login
2.	User	Signup
3.	User	Forgot password
4.	User	View extracted text
5.	User	Upload filed
6.	User	Delete history
7.	User	Classify text

Order	Table Name	Data privilege
1.	Admin	Login
2.	Admin	Forget Password
3.	Admin	Search Users
4.	Admin	View Users
5.	Admin	Create Users
6.	Admin	Edit Users

7.	Admin	Delete Users

DATABASE DESIGN

User

Field Name	Data Type	Width	Value
ID	Number	8	Not Null, primary key
Full Name	Character	50	Not Null
Email	Character	50	Not Null
Password	Character	20	Not Null
Text	Character	5000	Not Null
file	Document / img / sound	200 mb	Not Null

Admin

Field Name	Data Type	Width	Value
ID	Number	8	Not Null, primary key
Full Name	Character	50	Not Null
Email	Character	50	Not Null
Password	Character	20	Not Null

CONCLUSION

In this chapter we have discussed the problem we covered and the solution we proposed with its scope and how will we gather the data will be used to train our model then we moved to the technical system details and displayed the architecture of the application thru couple of diagrams which are:

- **Context Diagram.**
- **Data Flow Diagram**
- **Use Case Diagram.**
- **Class Diagram**

- **ERD**
- **Use Case Tables**
- **Sequence Diagram**

Finally, we described the database and its entities and relationships as well.

Chapter 4
System Design

4. Introduction:

In this chapter we will discuss the design phase and how it will look like in the proposed system and its attributes and actions and capabilities. During the Design Phase, the system is designed to satisfy the requirements identified in the previous phases.

The requirements identified in the Requirements Analysis Phase then transformed into a System Design Document that accurately describes the design of the system and that can be used as an input to system development in the next phase.

4.1. System Architecture:

In this system architecture we defined that the client portal will be a web application that will be access the database , AI model and business logic layer throw HTTP request, It will implemented using HTML and CSS

System Design

- **Client Layer:**
  - The client layer will be developed using HTML and CSS.
  - HTML will be used to structure the web pages and define the content.
  - CSS will be used to style and layout the web pages.
- **Backend Layer:**
  - The backend layer will be developed using Django, a Python web framework.
  - Django will handle the server-side logic and interact with the database.
- **AI Model:**

- An AI model will be developed using Python and OCR (Optical Character Recognition) techniques.
- Python libraries like OpenCV, Tesseract, or Pytesseract can be used for OCR.
- The AI model will be trained to recognize and extract text from images.

- **Database:**

  - Django's built-in Object-Relational Mapping (ORM) will be used to interact with the SQL Server database.
  - The database will store user information, processed data, and any other relevant data.
  - The system will use SQL Server as the database management system.

- **System Architecture:**
  - The client layer will send requests to the Django backend.
  - Django will handle the requests, process the data, and interact with the AI model for OCR.
  - The AI model will extract text from images and provide the processed data back to Django.
  - Django will store the processed data in the SQL Server database.
  - The client layer can then retrieve the processed data from the database.

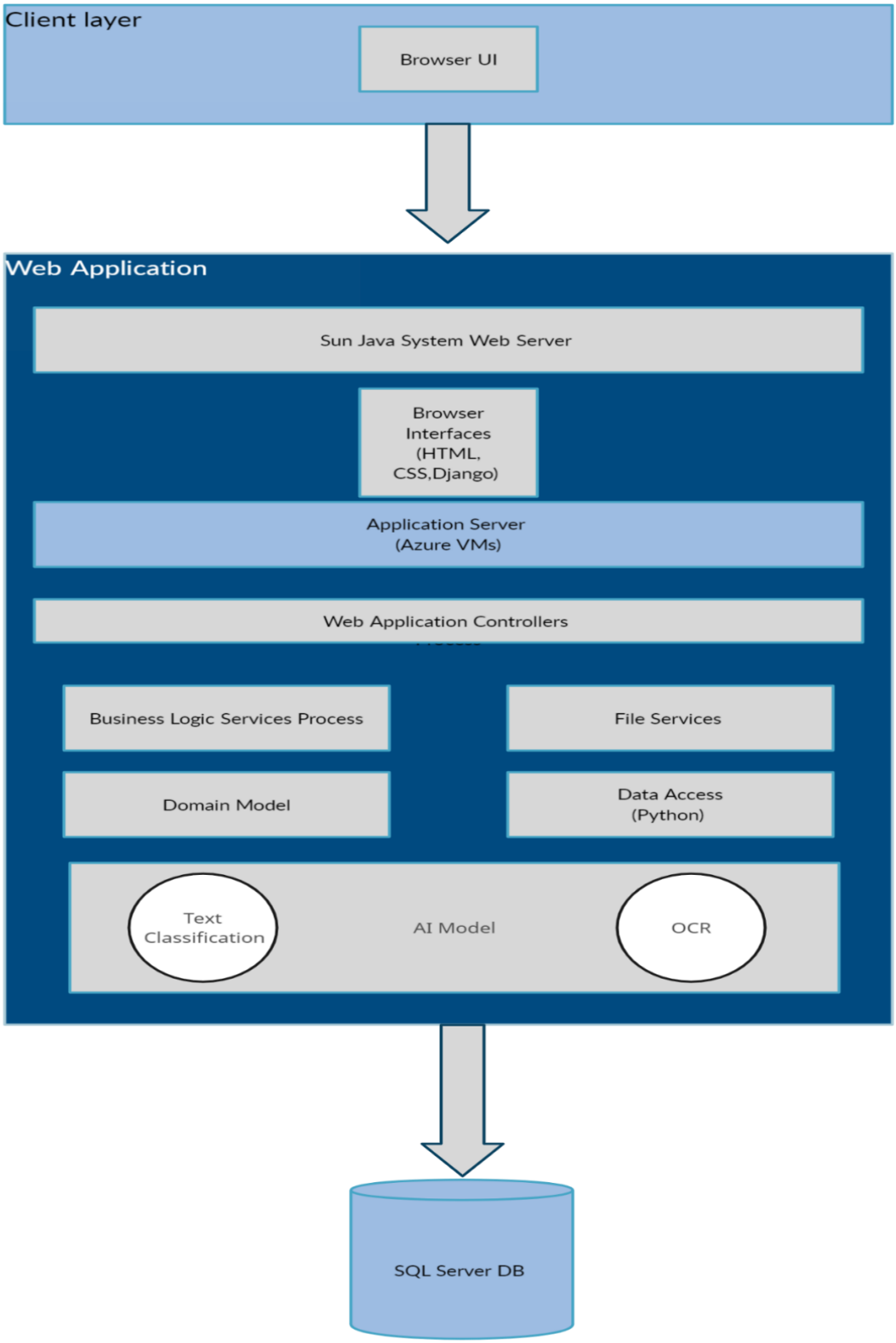


Figure 1: System Architecture.

- **Security:**
  - The system should implement proper security measures to protect user data and prevent unauthorized access.
  - Django provides built-in security features like CSRF protection, user authentication, and authorization.
  - Secure communication protocols, such as HTTPS, should be used to encrypt data transmission.
- **Scalability and Performance:**
  - The system should be designed to handle many users and perform efficiently.
  - Proper database indexing and query optimization techniques should be implemented.
  - Caching mechanisms, such as Redis, can be used to improve performance.
  - Load balancing and horizontal scaling can be considered to distribute the workload.
- **Testing and Deployment:**
  - Unit testing and integration testing should be performed to ensure the system functions as expected.
  - Continuous Integration/Continuous Deployment (CI/CD) pipelines can be set up for automated testing and deployment.
  - Deployment can be done on cloud platforms like AWS, Azure, or GCP, considering factors like scalability and availability.

Note: The above system design is a general overview and can be further customized based on specific requirements and constraints.

4.2. Database Design:

1. **User**

FIELD NAME	DATA TYPE	VALUE
ID	INT	NOT NULL , PK
FULL-NAME	NVARCHAR(100)	NOT NULL
EMAIL	VARCHAR(50)	NOT NULL
PASSWORD	VARCHAR(50)	NOT NULL
IS-ADMIN	BIT	NOT NULL

2. **Admin**

FIELD NAME	DATA TYPE	VALUE
ID	INT	NOT NULL , PK
FULL-NAME	NVARCHAR(100)	NOT NULL
EMAIL	VARCHAR(50)	NOT NULL
PASSWORD	VARCHAR(50)	NOT NULL
IS-ADMIN	BIT	NOT NULL

4.3. User Interface Design:

User Interface Design focuses on anticipating what users might need to do and ensuring that the interface has elements that are easy to access, understand, and use to facilitate those actions. When designing a user interface, there are many design considerations and principles that are considered to ensure that the interface meets the needs of users and are effective in achieving its intended purpose. Here are some key considerations and principles:

Usability: User interface design focuses on the needs and preferences of users. This means that the design is intuitive and easy to use.

Accessibility: This includes the use of appropriate color contrast, and the ability to navigate the interface using keyboard controls.

Responsiveness: Responsive user interface design for various screen sizes and devices, including desktops, laptops, tablets, and smartphones. The interface is designed with a flexible layout that adapts to different screen sizes.

4.3.1: The Register page includes a form where users can enter their personal information to create a new account. The form includes fields for user full name, email address, and password. The form includes sign-in options for users if they have an account

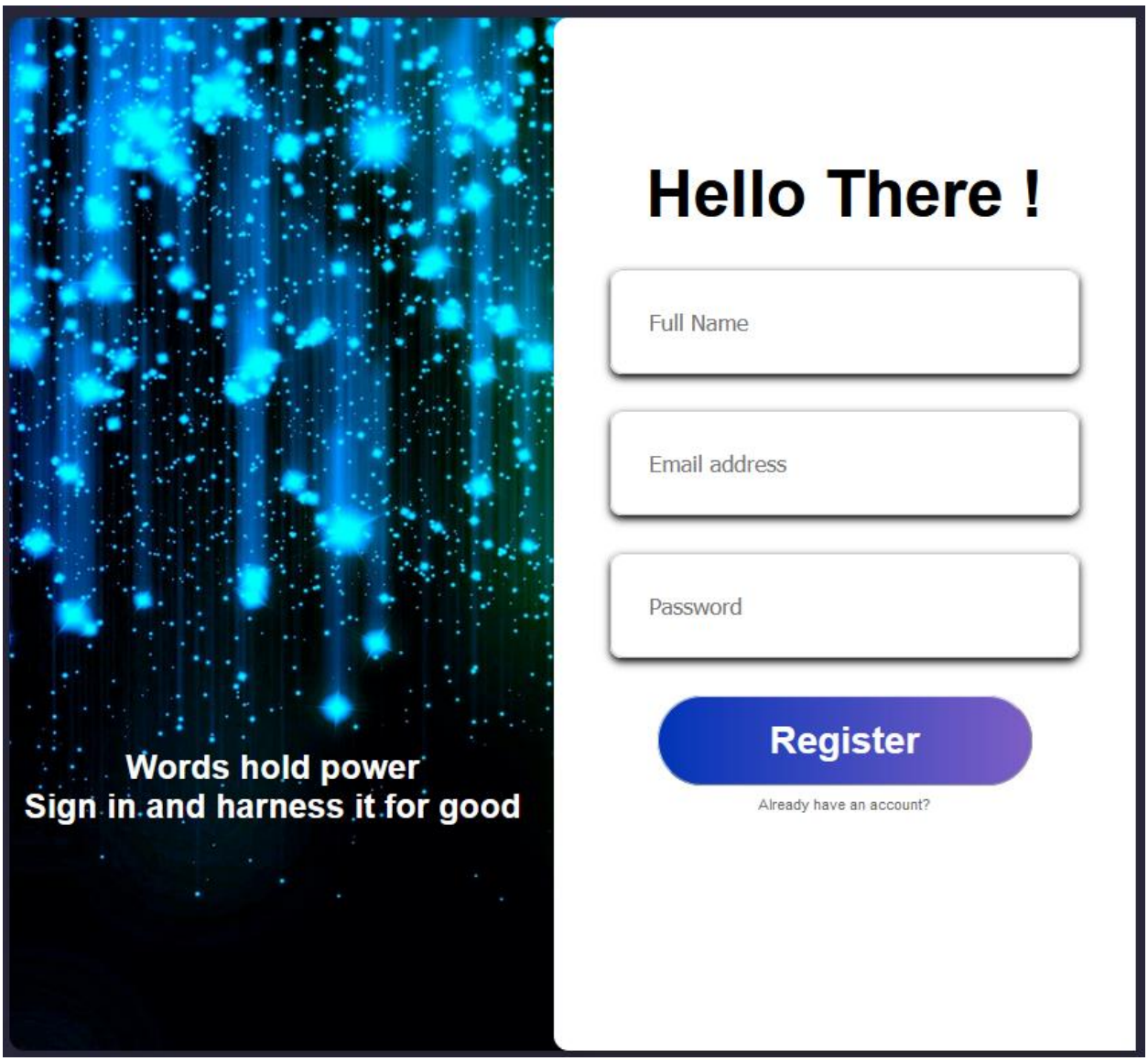


Figure 1: Register Form

4.3.2. user Sign In: The Sign-in page includes a form where user can enter their login credentials, such as an email address and password. The form also includes options for users to reset their password if forgotten or create a new account if they don't have one

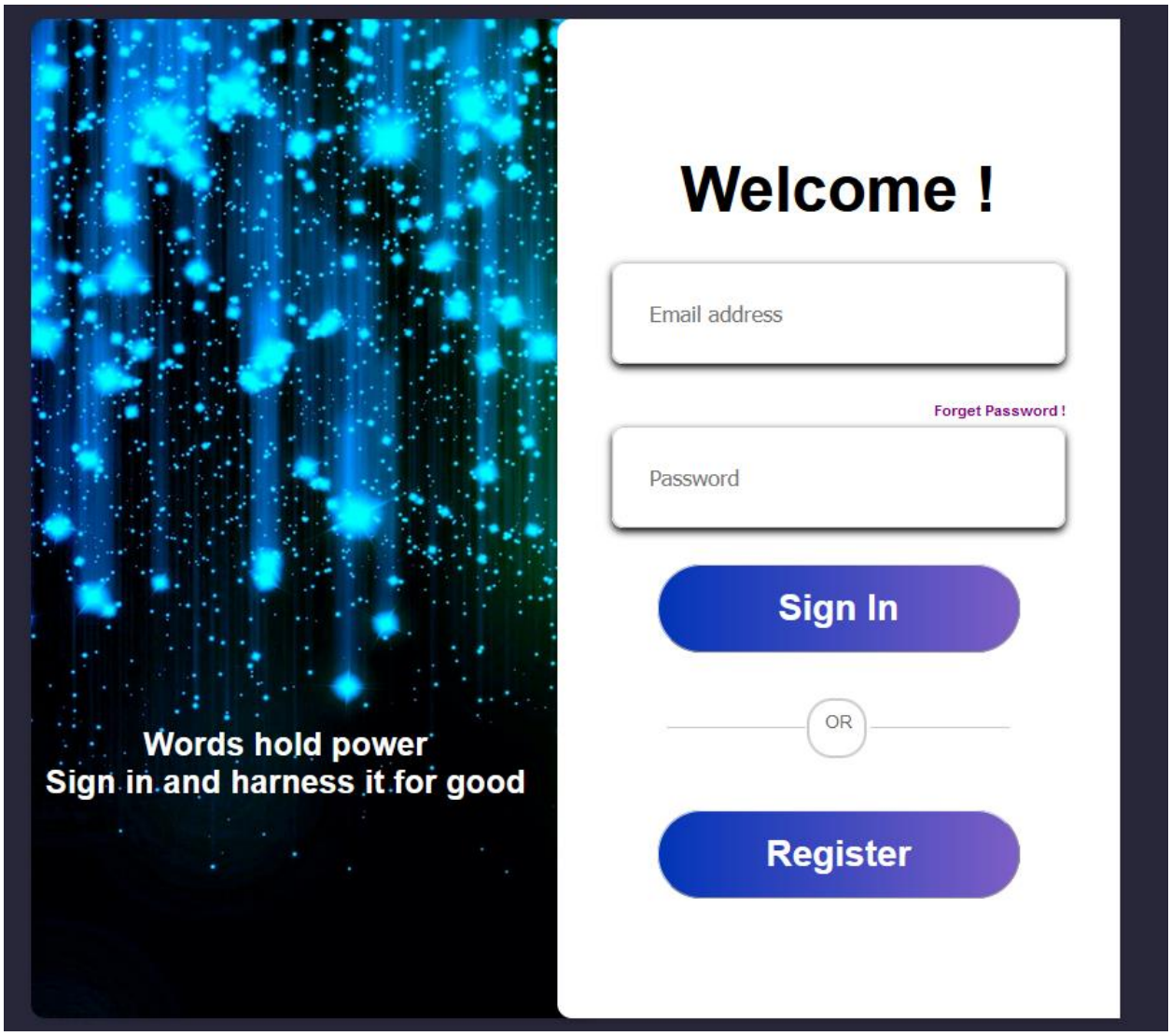


Figure 2: Sign In Form

4.3.3: Forget Password: The system's "forgotten password" page includes a form where users can reset their password if they forget it. The form asks the user to enter the email address associated with their account. Once the user submits this information, the system will send an email to the user with the instructions

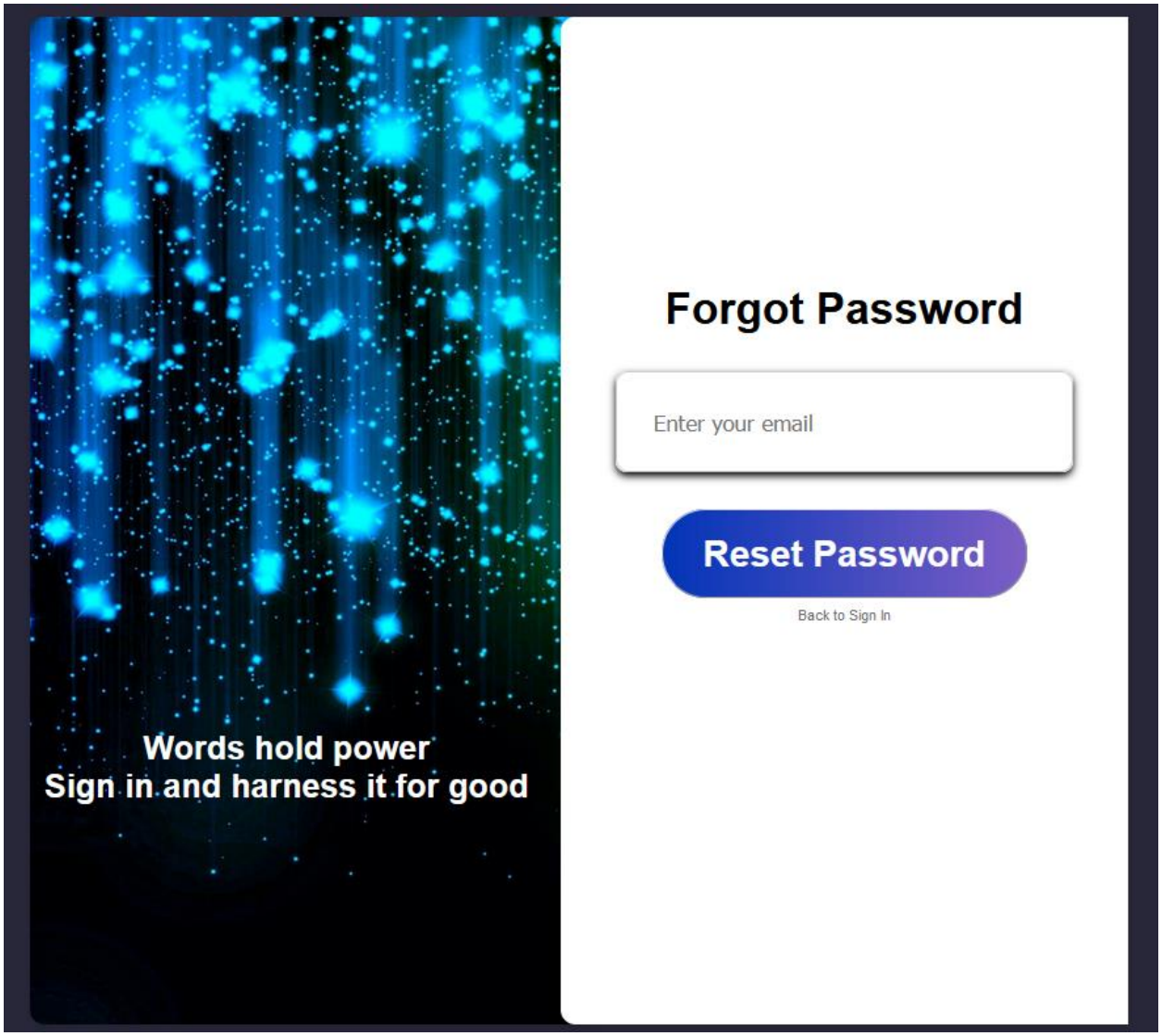


Figure 3: Forget Password

4.3.4: Main Screen: The main screen includes a text box , upload file, clear and submit buttons where users can enter their text, read image or files text and classify and clear it

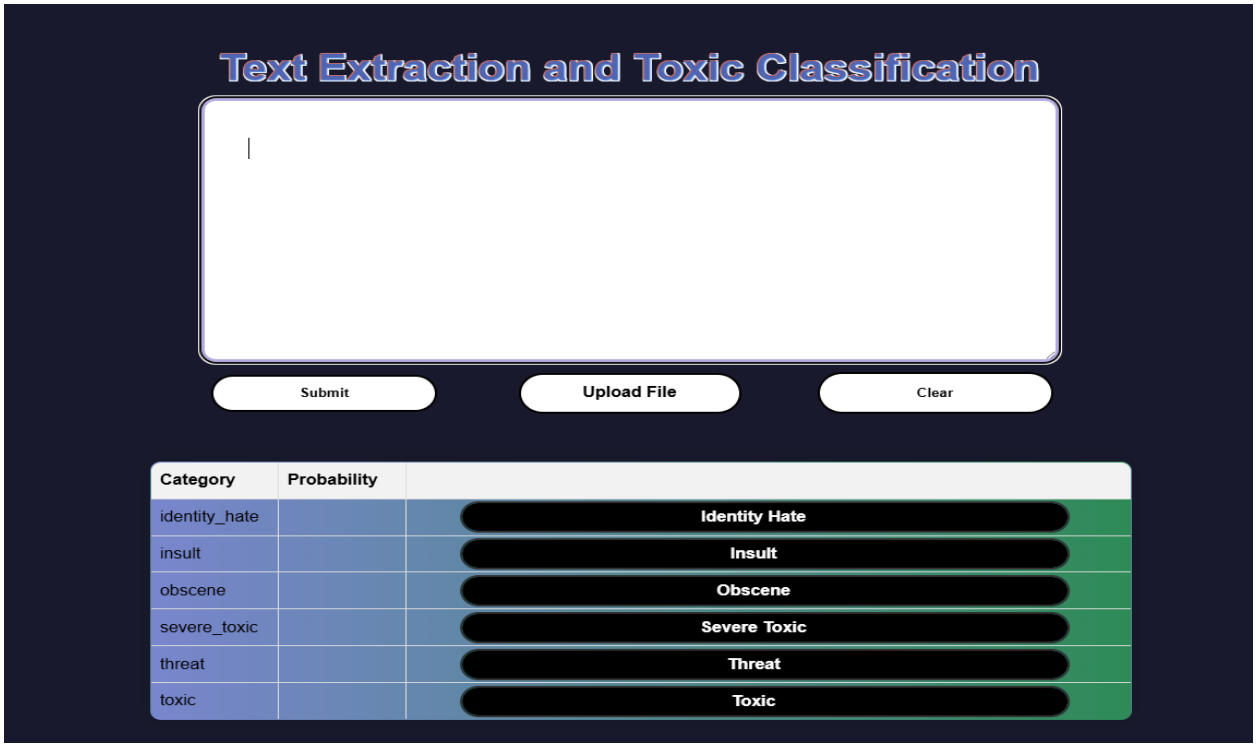


Figure 4: Main Screen

4.3.5: Enter Text : Text box have text area requiring user to enter the text he desires to classify

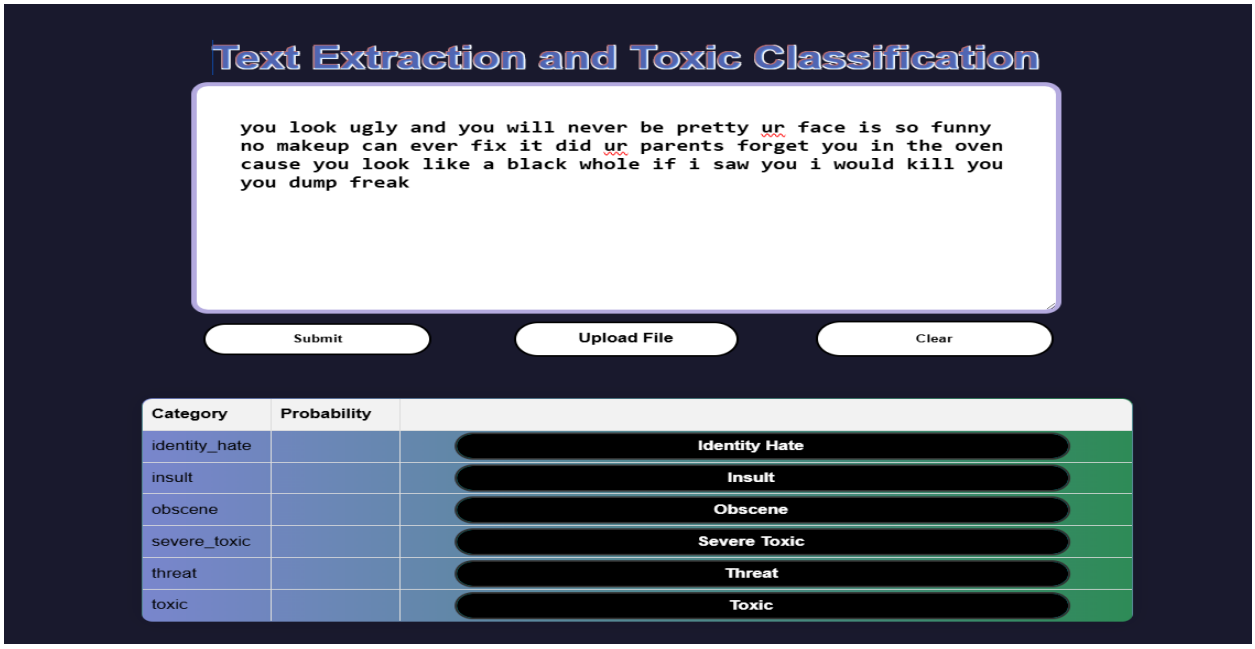


Figure 5: Enter Text

4.3.6: Classify Plain Text : The text user entered gets processed by the model classifying it giving the percentage and type of the toxic text if ever found

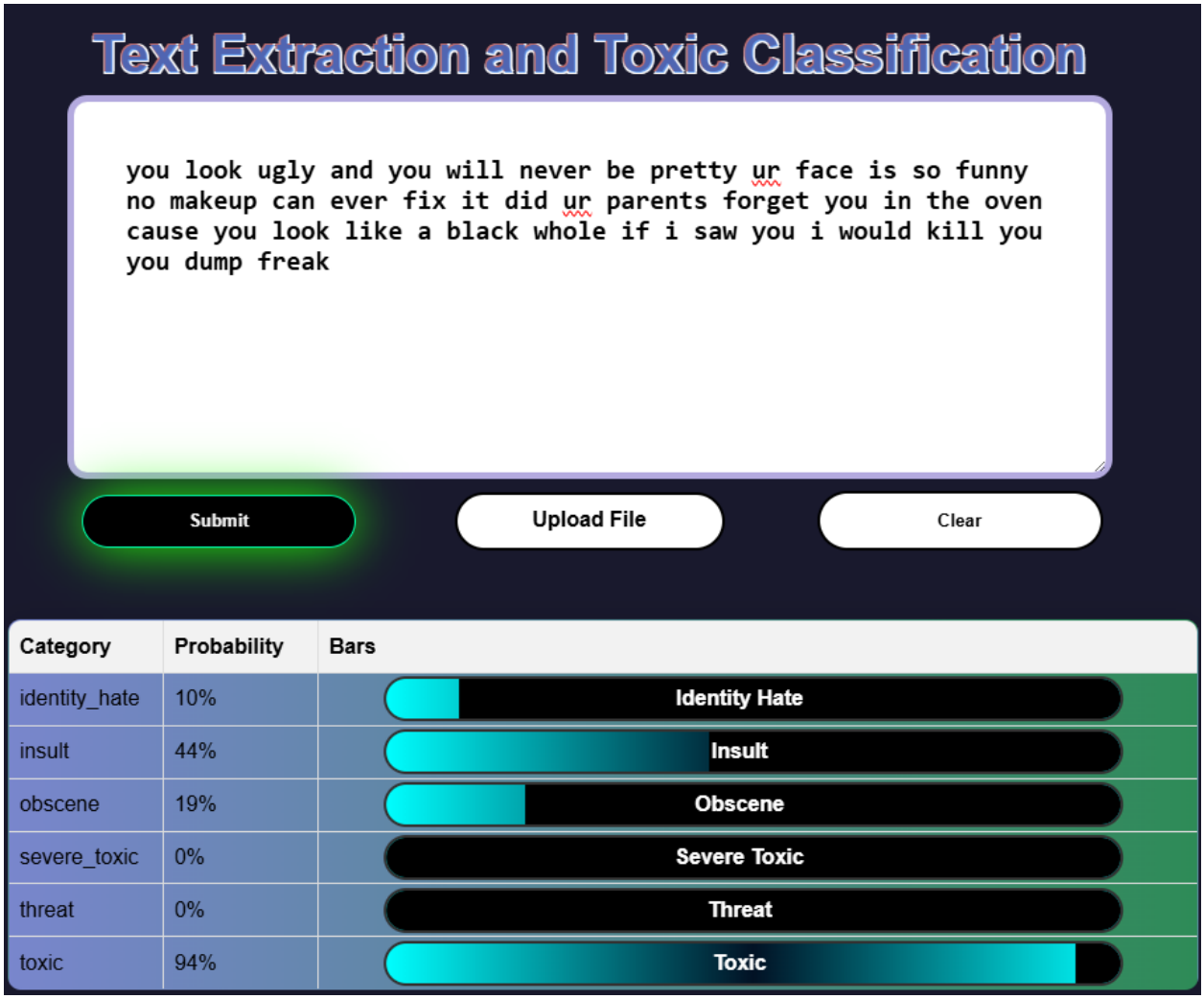
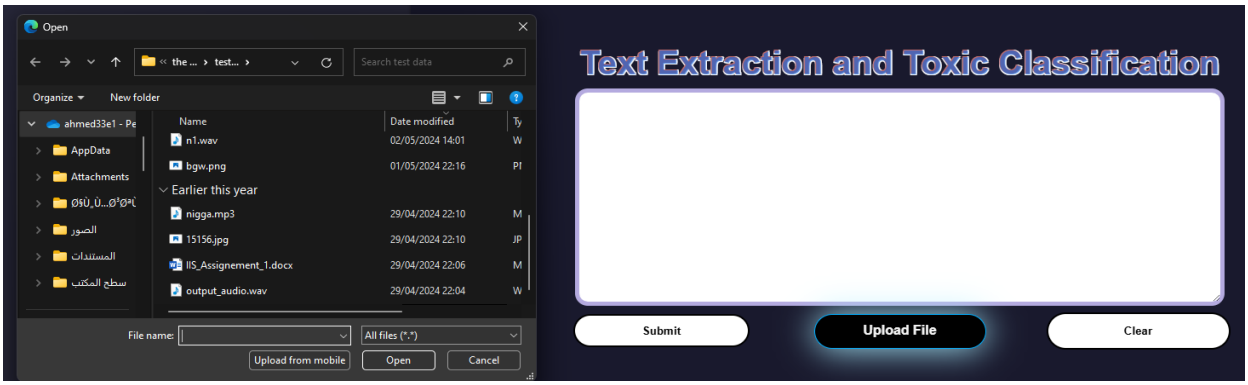


Figure 6: Classify Plain Text Results

4.3.7: Upload File : middle button allows the user to upload an any allowed file type to extract the text I in and get displayed into the text box



(user can drag and drop files)

Figure 7.1: Upload Files

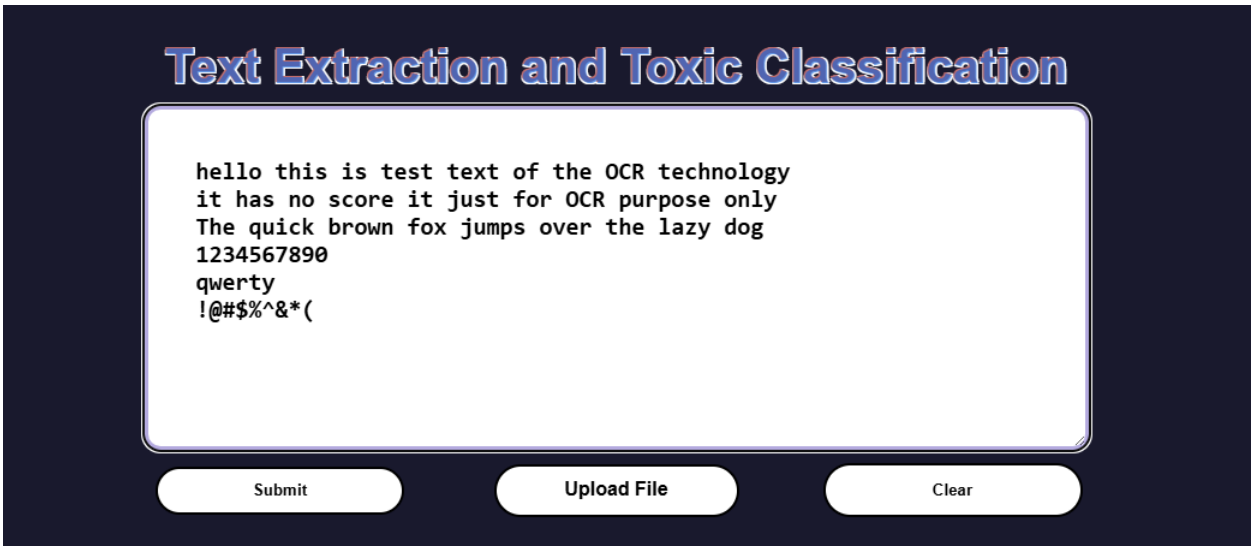


Figure 7.2: Image OCR (Image Text Extraction)

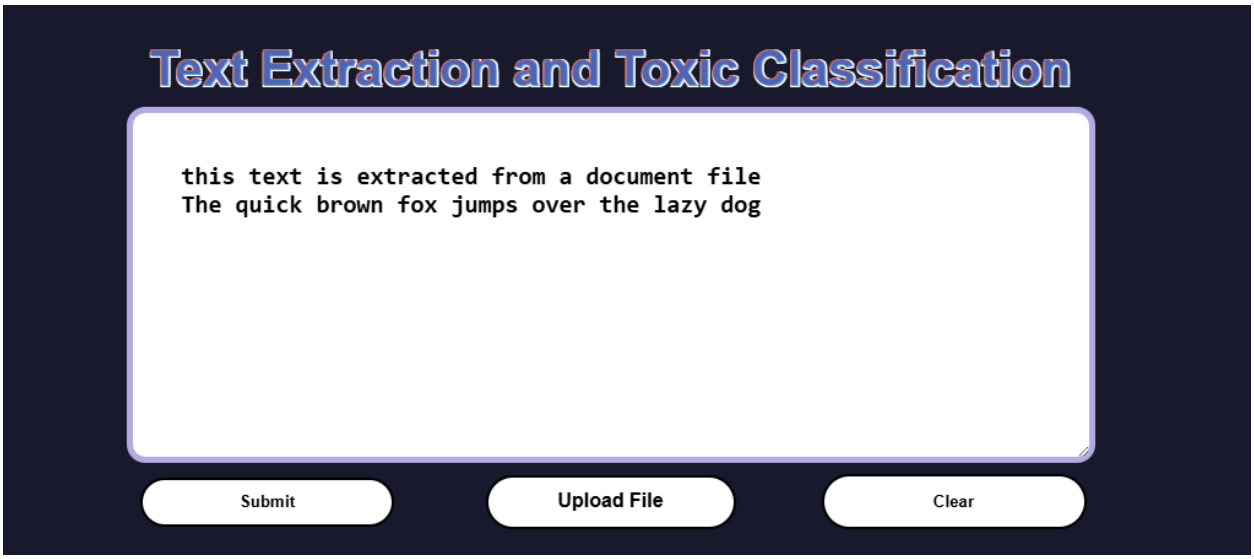


Figure 7.3: Document Text Extraction

4.3.8: Clear: A button in the right corner once clicked it clears the text in the text box allowing the user to enter new text

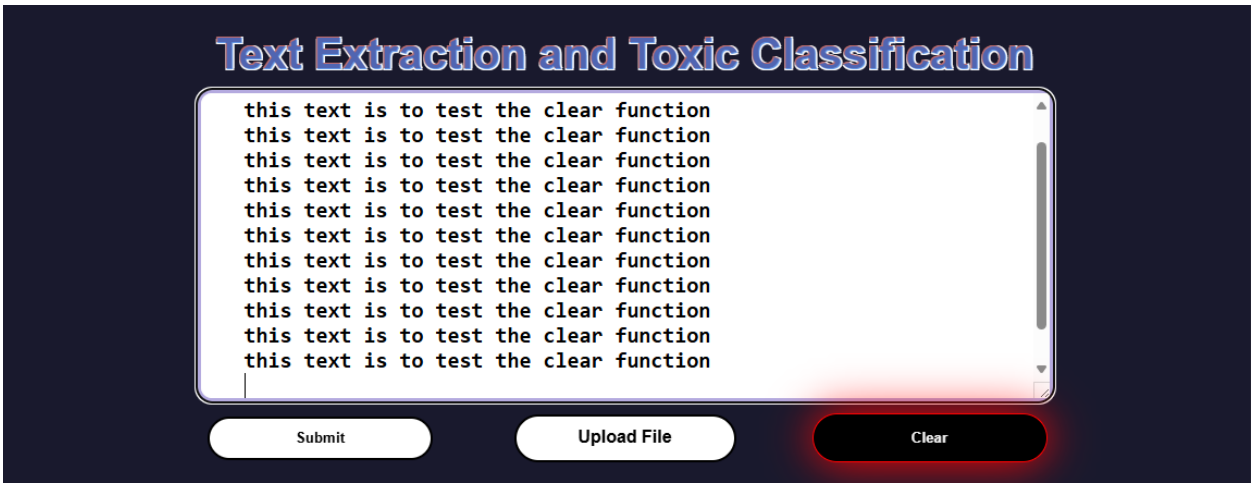


Figure 8.1: Clear button

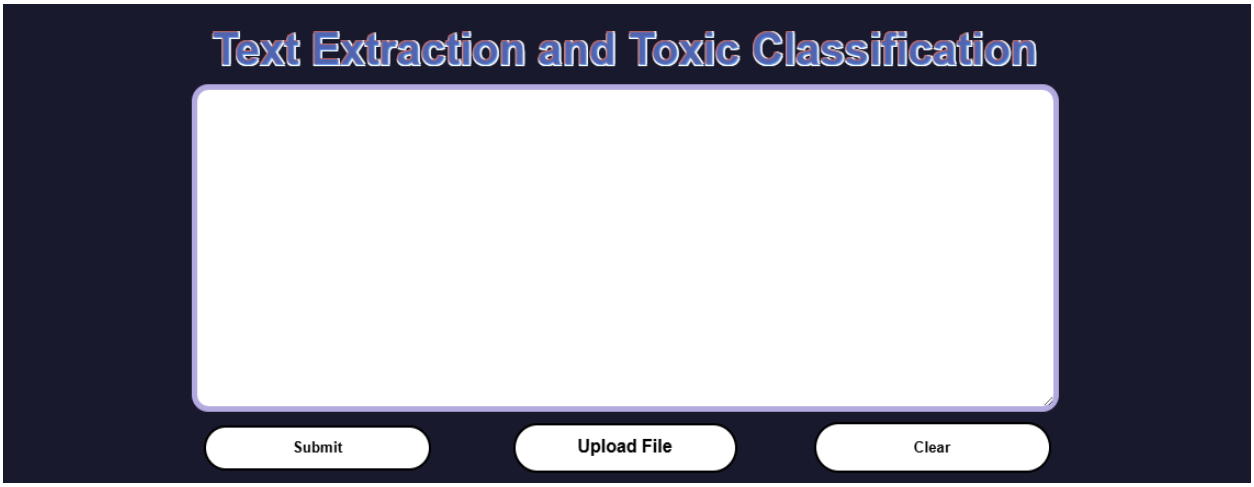


Figure 8.2: Text box after clicking the clear button

4.3.9: History: a history contains the user recent 10 submits at max currently due to system cap that he can re-use it just by clicking on it each element has the average of the classified data user can press it to get the detailed scores user is able to delete the history one by one or just all the history

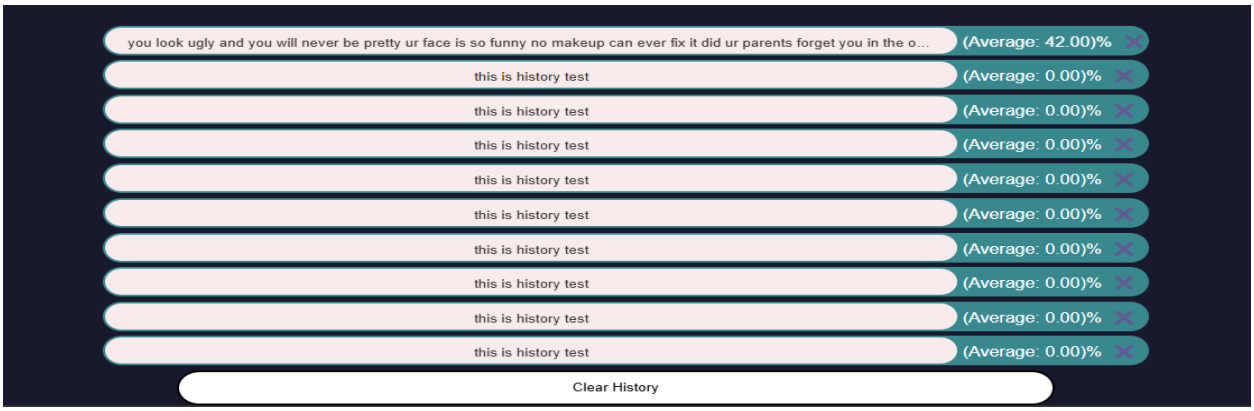


Figure 9.1 : the 10 history elements

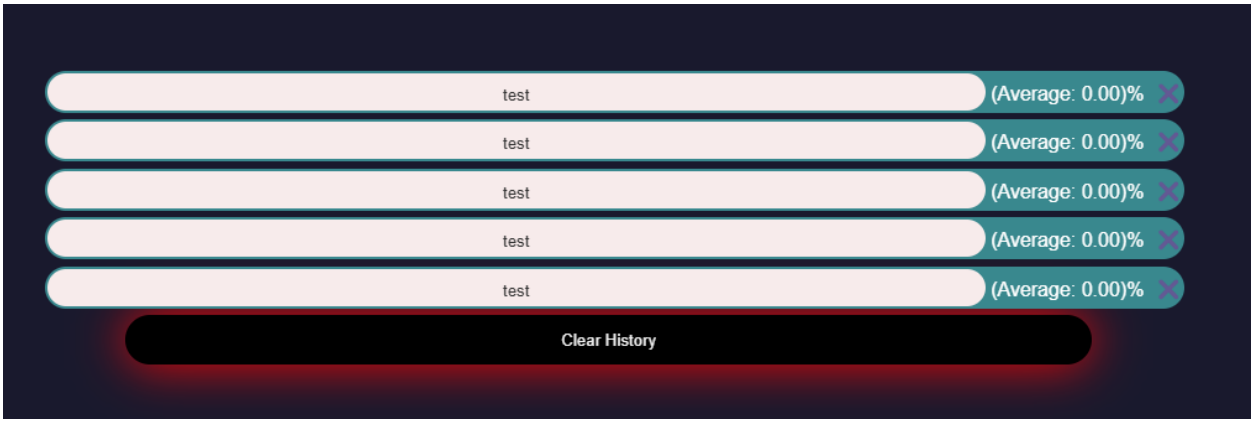


Figure 9.2: custom element remove (x)

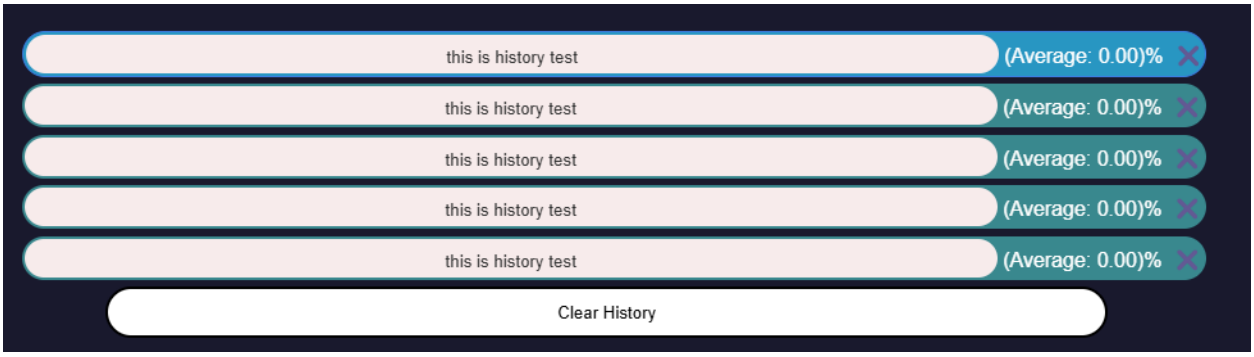


Figure 9.3: Removes all the history elements

4.3.10: Logout : when the user finally finishes all he wants he can just logout by using simple logout button on the top right

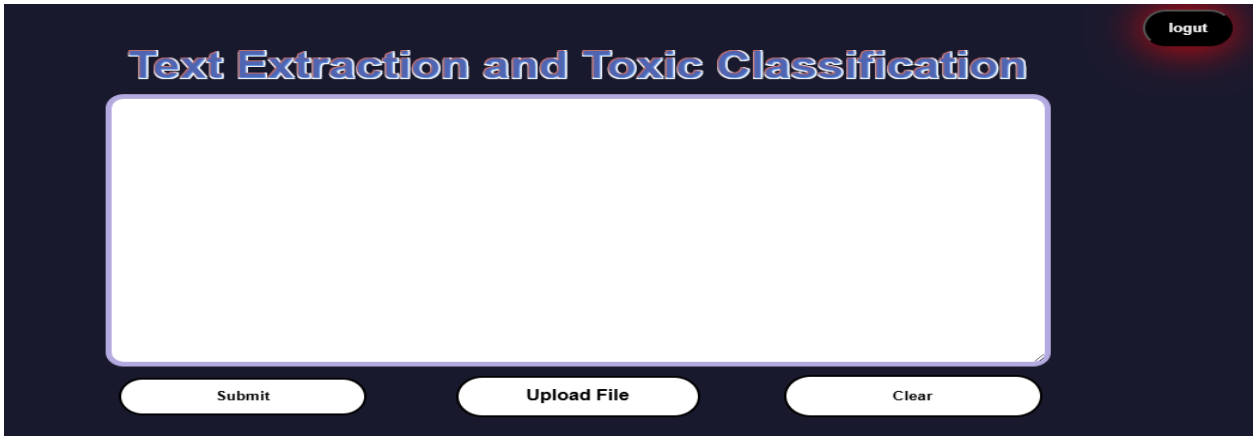


Figure 10 : the logout button

4.3.11: admin login : when the admin sign in with admin account the system automatically detect the account type and redirect it to the admin page where the admin can search create edit and delete users

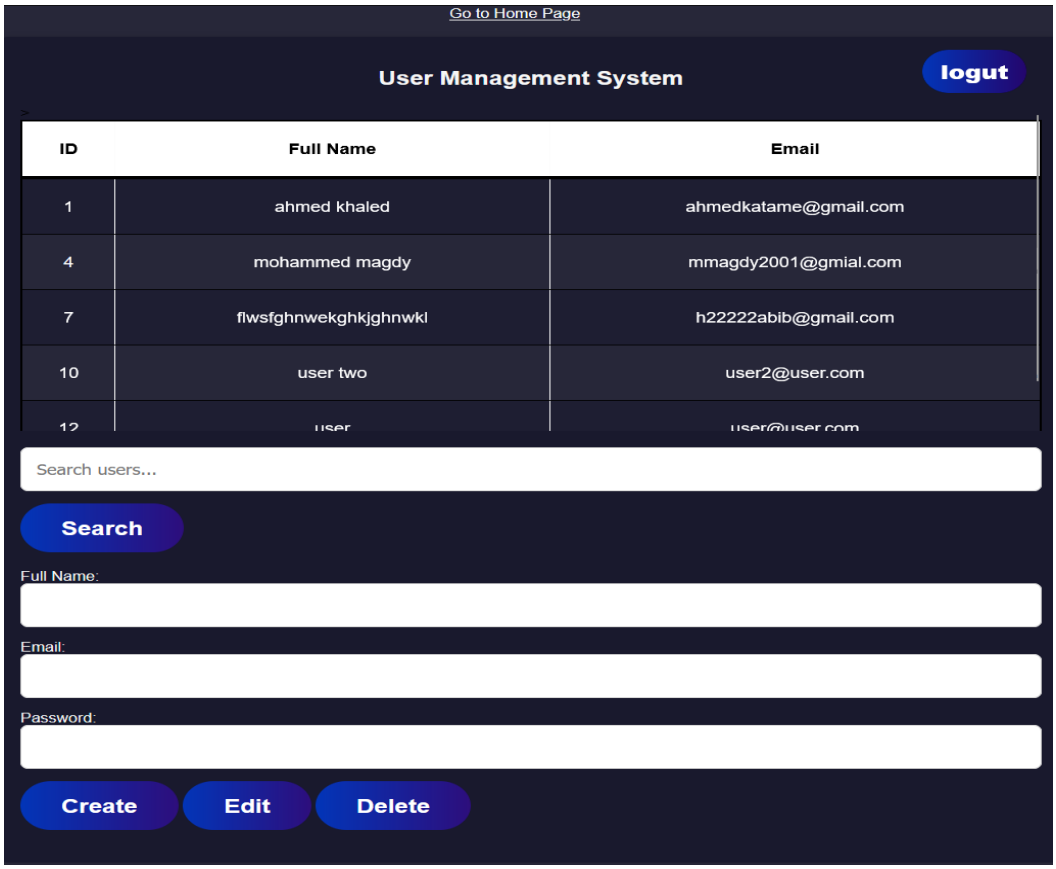


Figure 11 : the admin page

4.3.12: Search users : the admin is able to look at the users and their info admin can search user by their id (id is preferred cause it never changed and got displayed as white label over the search box) or email as its their primes after the admin enters the user id or email the user info will be displayed into boxes so admin can edit or delete the user

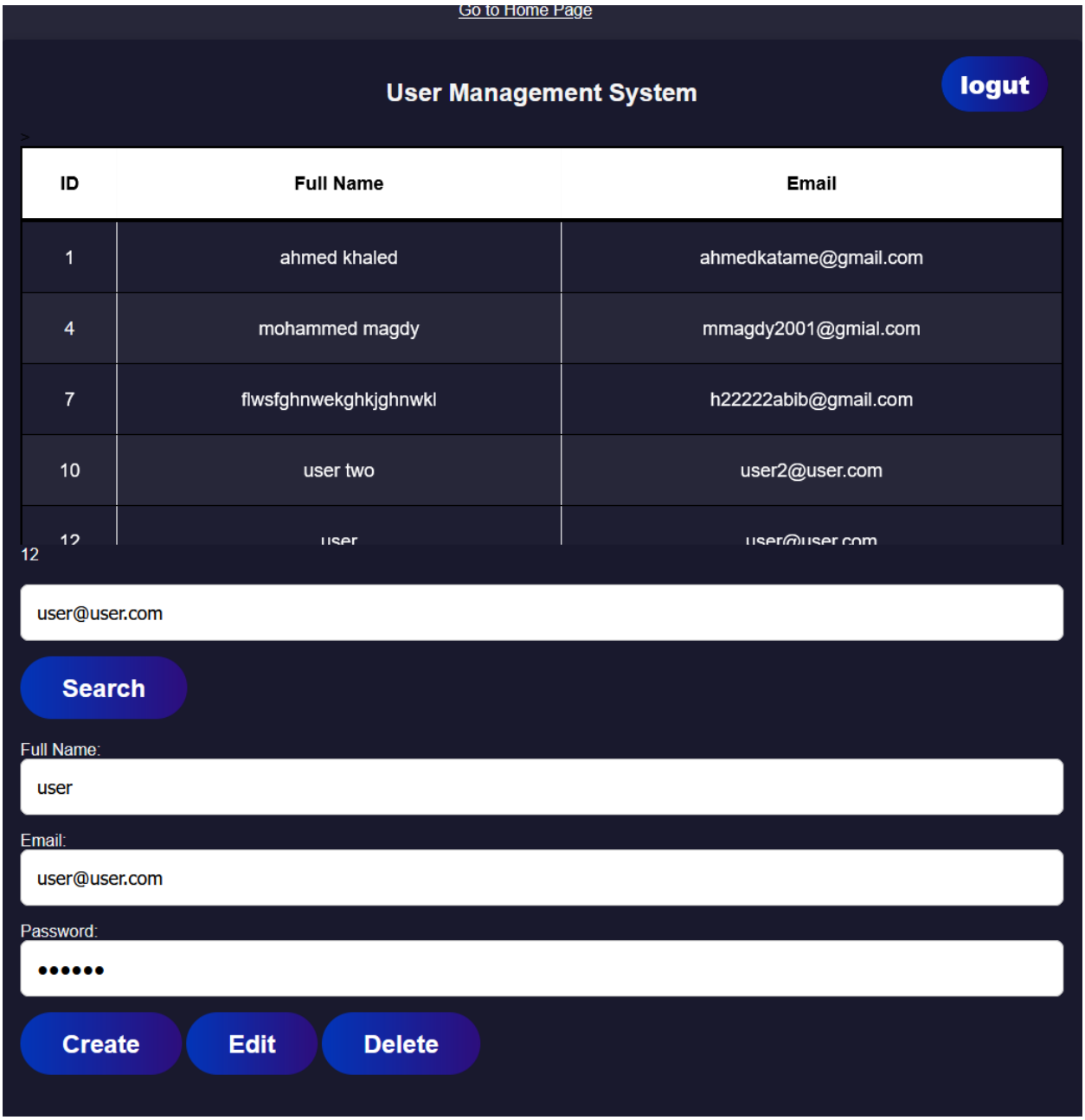


Figure 12: Search Function

4.3.13 : the admin can create users as they want from the admin page they just have to fill the text boxes and the user will be created Table will be refreshed with the new data

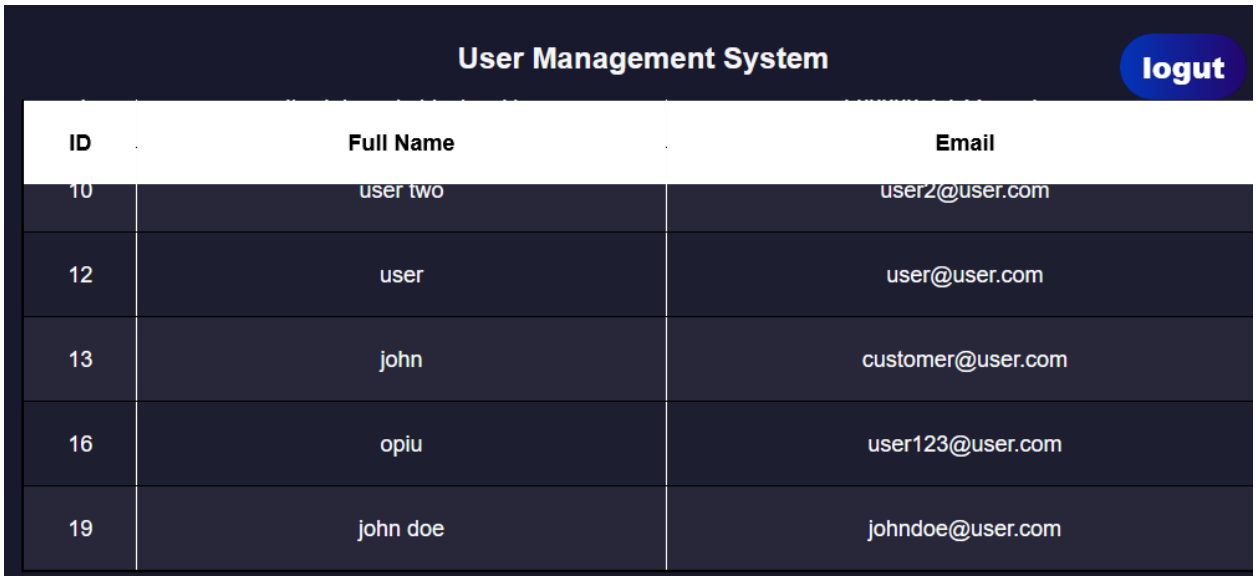


Figure 13.1 : Create User

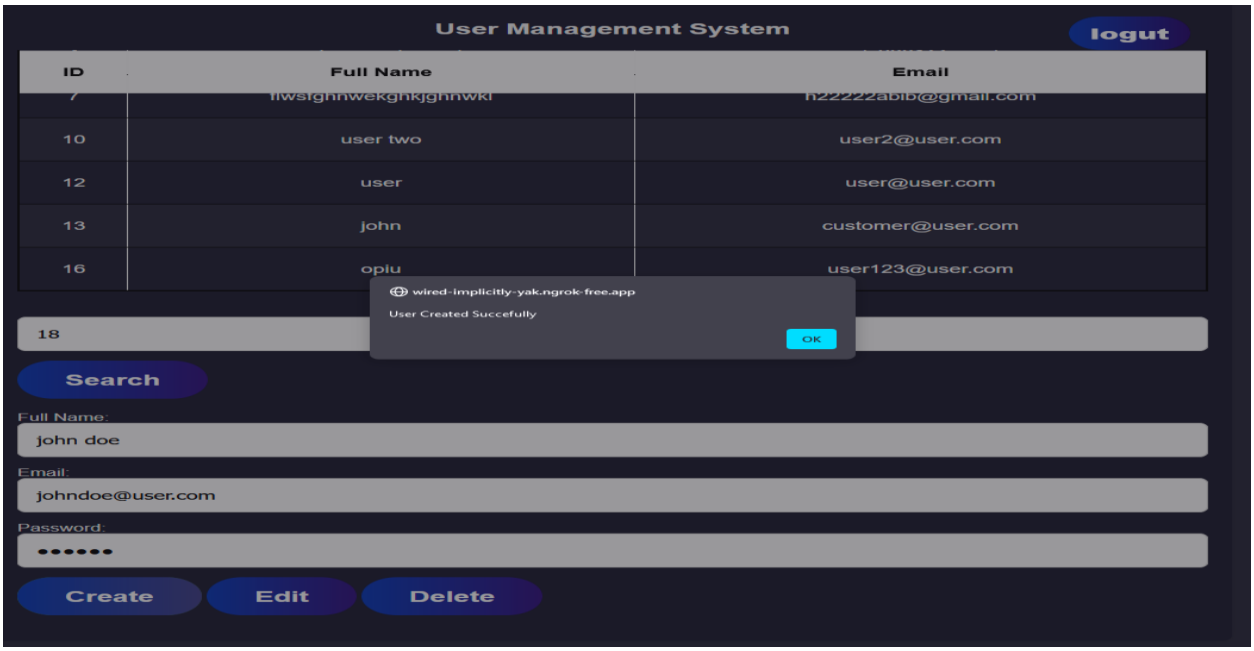


Figure 13.2 : User Populated Into The Tabel

4.3.14 : the admin can edit users as they want from the admin page so they can update any user data they just have to search the user by id or its email to get selected the selected user id will be displayed upon the search bar and its information will be displayed into the textboxes allowing the admin to edit it

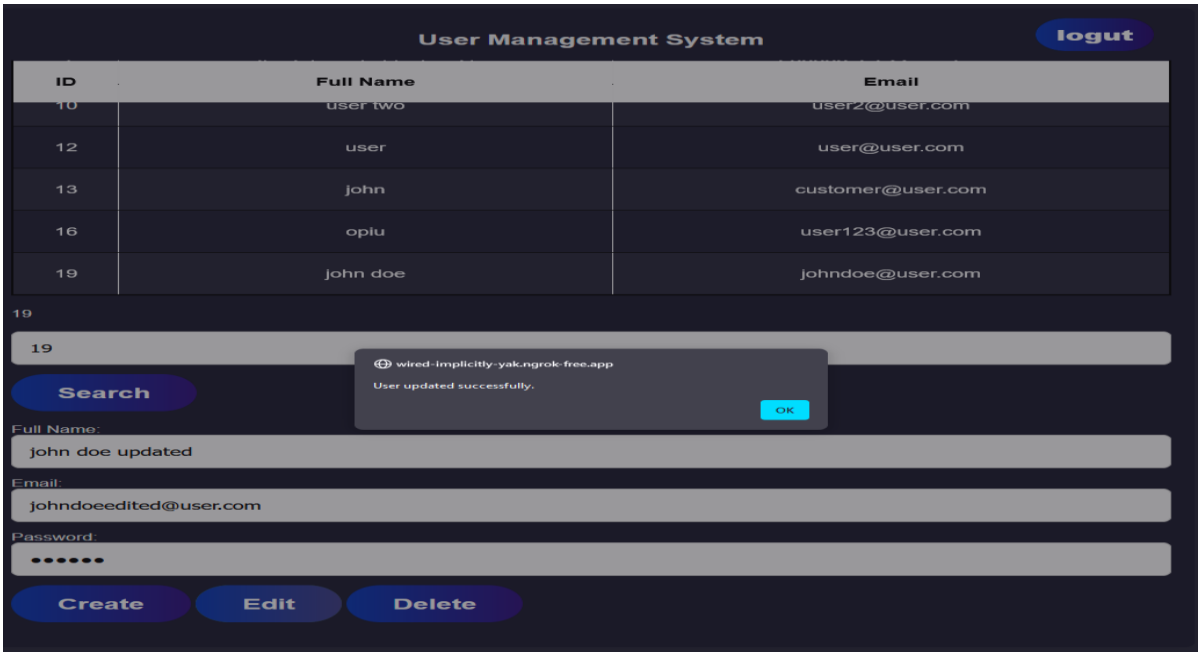


Figure 14.1: Update User

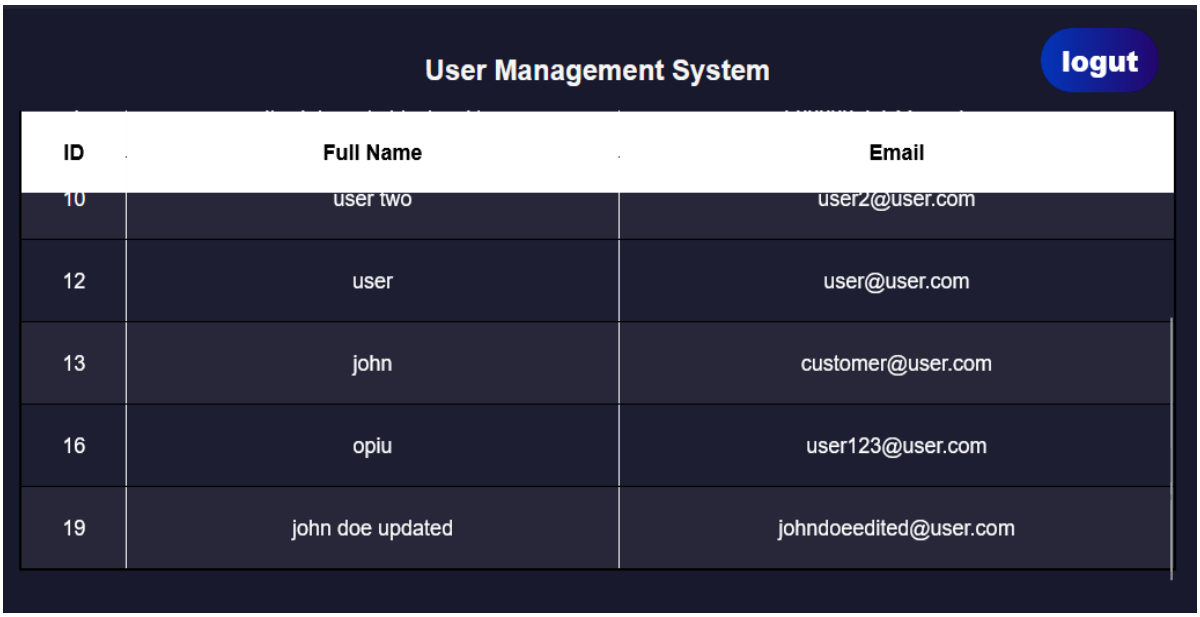


Figure 14.2 : User Data Updated

4.3.15 : the admin can delete users as they want from the admin page they just have to search the user by id or its email to get selected the selected user id will be displayed upon the search bar and the admin can delete it

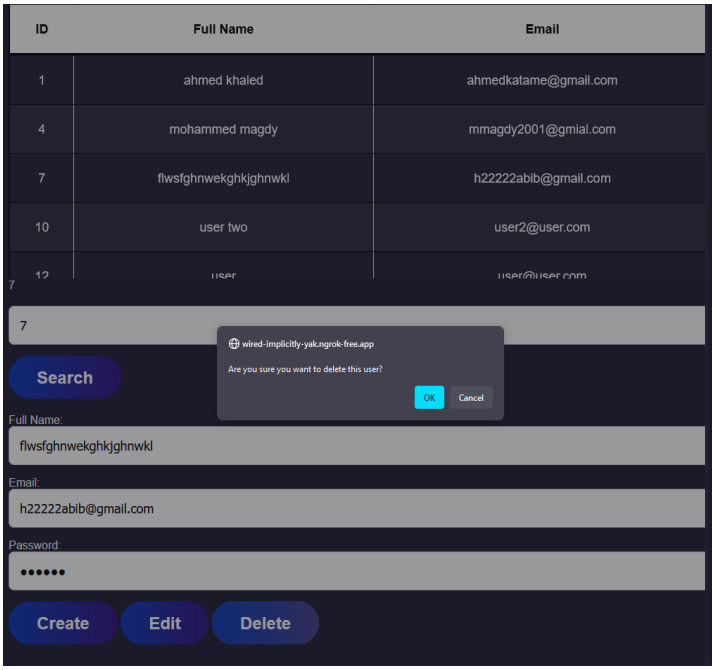


Figure 15.1 : Confirm Delete User

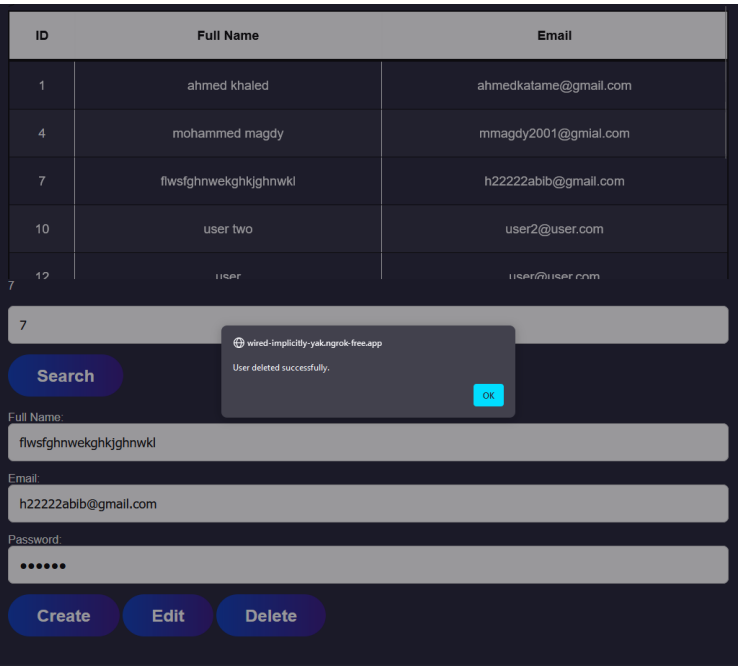


Figure 15.2 : Delete User Done

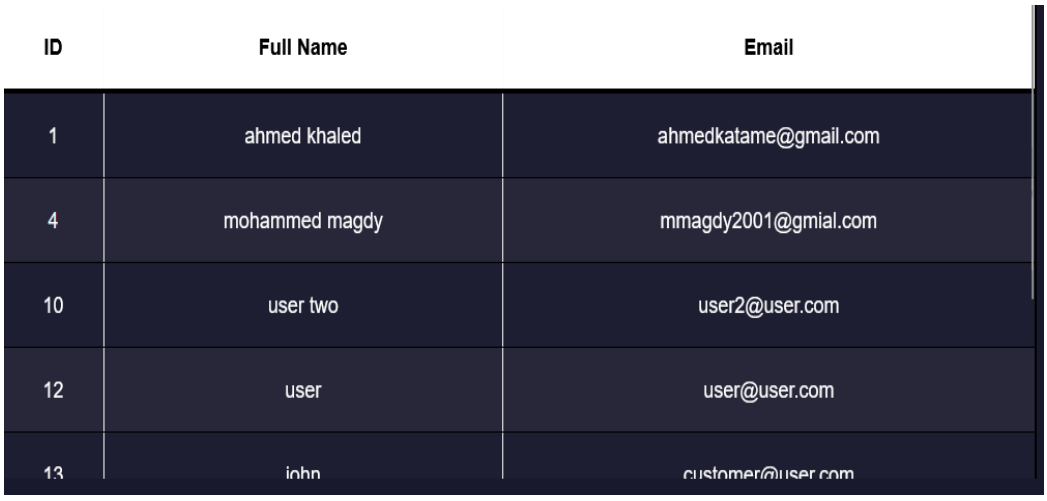
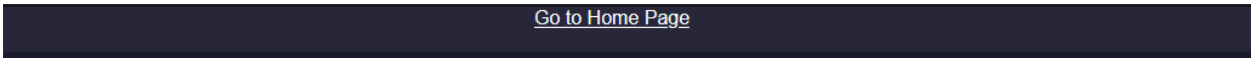


Figure 15.3: User Deleted

4.3.16 : the admin page has a hyperlink label that redirects the admins to the main page

To use and test the system functions

When they are redirected to the admin page there is a button appears only for



admins to get back to the admin page

Figure 16.1: Main Page Link For Admin

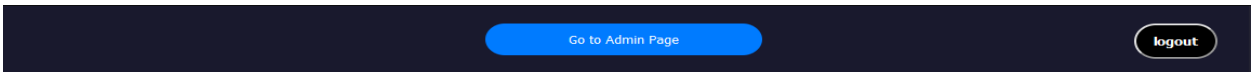


Figure 16.2: Back to Admin Page Button

4.3.17 : the admin can logout by the logout button on the upper right corner from the admin page or main page

Figure 17.1: Admin logout from Admin page

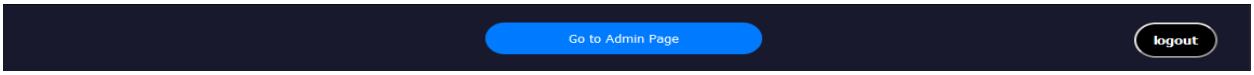


Figure 17.2: Admin logout from main page

Summary

In this chapter we have provided the system architecture and the interconnection between its component then we defined the database and its denormalized entities design.

We also moved to the UI design prototype and have created a joyful user journey using wireframes and interactive designs.

We covered the interaction for the mentioned design to set the user exceptions regarding the designs and how is the behavior of the system.

Finally we defined the exception handling for any unexpected behavior and how to establish a tight, comprehensive system

Chapter 5
Implementation

INTRODUCTION

In the expansive realm of deep learning implementation, the journey from inception to deployment traverses a multifaceted landscape of data orchestration, model crafting, and real-world integration. It commences with the meticulous curation of data, where a medley of sources, spanning CSV repositories to web-scraped treasures, converge into a harmonious ensemble of information. This amalgamation undergoes a rigorous preprocessing regimen, shedding extraneous artifacts and donning the guise of cleanliness through the purging of HTML detritus and the harmonization of casing. Tokenization then beckons, ushering the textual corpus into a realm of quantifiable entities, where words and sub words frolic as tokens, ready for the subsequent symphony of modelling.

The selection of the model is akin to a grandiose spectacle, with options ranging from the venerable Logistic Regression to the enigmatic depths of CNNs and Transformers. Each contender vies for supremacy, its architecture tailored to the nuances of the task at hand, be it the finesse of sentiment analysis or the grandeur of document classification. Once chosen, the model is meticulously sculpted, its layers intricately woven into a tapestry of computational prowess, with loss functions and optimizers serving as the artisan's brushstrokes.

Training, the crucible of neural refinement, transforms raw data into predictive mastery. Across epochs and batches, the model imbibes the essence of its domain, its weights and biases adjusting in harmony with the data's melody. Hyperparameters dance in a delicate ballet, their tuning an endeavour of art and science, as the model tiptoes on the tightrope between underfitting and overfitting.

Evaluation, the moment of reckoning, unfolds with the unveiling of metrics: accuracy, precision, recall—all scrutinized under the unforgiving gaze of validation sets. Here, the model's mettle is tested, its performance laid bare for all to behold. Error analysis becomes the harbinger of enlightenment, illuminating the shadows of misclassification and guiding the quest for refinement.

And then, deployment—the grand denouement of the saga. The model emerges from the crucible, embarking on its journey to the realms of production. An API, a conduit between the digital and the tangible, beckons forth, woven with the threads of Flask or Fast API, ready to interface with the clamouring masses. Monitoring stations stand sentinel, guardians of performance and sentinels against the encroachment of data drift and degradation.

In summation, the path from inception to deployment is not merely a journey—it is an epic saga, replete with trials and triumphs, challenges and conquests.

Implementation

Implementation in deep learning refers to the process of building, training, and deploying a deep learning model for a specific task or application. The implementation process for our text extraction and classification project involves several steps

Data Collection and Preprocessing

Data Collection:

Data collection and preprocessing are crucial steps in any machine learning project. The first task involves gathering relevant text data from various sources like CSV files, web scraping, or APIs. Preprocessing ensures that the data is clean and ready for modeling. This includes tasks like removing HTML tags, special characters, and handling punctuation. Tokenization splits the text into individual tokens, which could be words or sub words. Normalization standardizes the text to a consistent format, converting it to lowercase and applying stemming or lemmatization. Finally, vectorization converts text tokens into numerical vectors that machine learning models can process. Libraries like Pandas, NLTK, SpaCy, and TF-IDF Vectorizer are often used in these steps..

We used a csv data for a variant of comments and text like Kaggle the data used is more than 150 thousand record of data with scores

```
def train_model():
    train = pd.read_csv('train.csv')
    COMMENT = 'comment_text'
    train[COMMENT].fillna("unknown", inplace=True)
```

In this code the data is read by panda library to read csv files such as train.csv the file that has all the training data inside of it

159542	ff9e91b29	I find this	0	0	0	0	0	0
159543	ffa33d312	Your	1	0	1	0	1	0
159544	ffa95244f2	maybe he	0	0	0	0	0	0
159545	ffad10433	scrap that	0	0	0	0	0	0
159546	ffaed63c4	You could	0	0	0	0	0	0
159547	ffb268f37	, 7 March	0	0	0	0	0	0
159548	ffb47123b	"	1	0	0	0	1	0
159549	ffb7b4c3d	Thank ve	0	0	0	0	0	0
159550	ffb93b0a0	Talkback:	0	0	0	0	0	0
159551	ffb998f97	2005	0	0	0	0	0	0
159552	ffba5332d	i agree/ o	0	0	0	0	0	0
159553	ffbc2db42	While abc	0	0	0	0	0	0
159554	ffbcd64a7	Prague	0	0	0	0	0	0
159555	ffbd331a3	I see this	0	0	0	0	0	0
159556	ffbdbb048	and i'm gc	1	0	1	0	1	0
159557	ffc2f40965	"	0	0	0	0	0	0
159558	ffc671f2ac	I'll be on I	0	0	0	0	0	0
159559	ffc7bbb17	It is my op	0	0	0	0	0	0
159560	ffca1e81a	Please stc	0	0	0	0	0	0
159561	ffca8d71d	Image:Ba	0	0	0	0	0	0
159562	ffcdcb718	"Editing	0	0	0	0	0	0
159563	ffd2e85b0	"	0	0	0	0	0	0
159564	ffd72e976	"	0	0	0	0	0	0
159565	ffe029a7c	"	0	0	0	0	0	0
159566	ffe897e7f	Catalan in	0	0	0	0	0	0
159567	ffe8b9316	The numb	0	0	0	0	0	0
159568	ffe987279	":::::And	0	0	0	0	0	0
159569	ffea4adee	You	0	0	0	0	0	0
159570	ffee36eab	Spitzer	0	0	0	0	0	0
159571	fff125370e	And it loo	0	0	0	0	0	0
159572	fff46fc426	"	0	0	0	0	0	0

Every record of the data has id and a comment contains the text model will be trained with

Every comment text has a score for each label

Data Preparing

Text Cleaning

When operating with nlp we have to clear and the clean the data we are using it to enhance the project
reducing the quantization on the model by eliminating un necessary data

- **Goal:** Remove noise and irrelevant parts from the text data.
- **Steps:**
  - Remove any irrelevant information, such as HTML tags, special characters, and stop words.
  - Handle punctuation, numbers, and irrelevant whitespace

We can do that by using nltk library in add to other custom stop words or characters to help cleaning the text

```python
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Ensure stopwords and wordnet are downloaded
nltk.download('stopwords')
nltk.download('wordnet')
default_stop_words = set(stopwords.words('english'))

# Custom stop words
custom_stop_words = set(["you", "human", "example"])

# Combine default and custom stop words
stop_words = default_stop_words.union(custom_stop_words)

lemmatizer = WordNetLemmatizer()
```

Stop words removal to make the model run smoother

Added custom words repeatedly used in the training data but don't have weight

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\katame\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\katame\AppData\Roaming\nltk_data...
```

Tokenization

Split text into individual tokens (words or sub words).

Using tokenization techniques to convert text into tokens. Popular tokenizers include those provided by NLTK, SpaCy, or the tokenizer from a specific model like BERT.

- **Objective:** The function aims to separate words and punctuation marks in a string, creating tokens that can be processed individually by a machine learning model.
- **Regex Usage:** The function utilizes regular expressions (regex) to identify punctuation marks and they are treated as separate tokens.

Steps:

1. **Regex Pattern Compilation:**
   - `re_tok = re.compile(f'([{string.punctuation}“”¨«»®´·º½¾¿¡§££’])')`
   - A regex pattern is compiled to match any character listed in the `string.punctuation` constant as well as additional punctuation marks provided in the string.
   - `string.punctuation` contains basic punctuation marks like `.,!?;:.`
2. **Regex Substitution:**
   - `re_tok.sub(r' \1 ', s):`
     - The `sub` method of the compiled regex replaces each punctuation mark in the string `s` with itself surrounded by spaces (`\1` refers to the matched punctuation).
   - This ensures that punctuation marks are treated as separate tokens rather than being attached to words.
3. **Splitting the String:**
   - `.split():`
     - After substitution, the string is split into a list of tokens using the default whitespace separator.
   - This results in a list where words and punctuation marks are individual elements.

```
def tokenize(s):
    re_tok = re.compile(f'([{string.punctuation}“”¨«»®´·º½¾¿¡§££’])')
    return re_tok.sub( repl: r' \1 ', s).split()
```

Model Selection

Model selection involves choosing an appropriate architecture for the text classification task. The choice depends on the complexity and nature of the problem. Traditional models like Logistic Regression and Naive Bayes are simple yet effective for many tasks. More complex architectures like Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformers (e.g., BERT) are used for tasks requiring deep understanding of text. Pre-trained models like BERT can be fine-tuned on specific tasks to leverage their vast learned knowledge. The selection impacts the model's performance, training time, and resource requirements.

```
models = []
for j in label_cols:
    m = LogisticRegression(C=4, penalty='l2', solver='liblinear')
    m.fit(trn_term_doc, train[j])
    models.append(m)

return models, vec, label_cols
```

Logistic Regression is a widely used statistical method for binary classification problems. Despite its name, it is a linear model suitable for predicting the probability that a given input belongs to a certain class. Here's a brief explanation of why Logistic Regression is a good choice for text classification tasks.

1. Simplicity and Interpretability: Logistic Regression is straightforward to implement and interpret. The output probabilities are easy to understand and provide clear insights into the classification decisions.

2. Efficiency: Logistic Regression is computationally efficient, making it suitable for large datasets and real-time applications. It scales well with the number of features, which is crucial when dealing with high-dimensional text data.

3. Effectiveness with TF-IDF: When combined with TF-IDF (Term Frequency-Inverse Document Frequency) vectorization, Logistic Regression performs well on text classification tasks. TF-IDF helps transform text data into numerical features by capturing the importance of words in the context of the entire dataset, which Logistic Regression can effectively use to distinguish between classes.

```
vec = TfidfVectorizer(ngram_range=(1, 2), tokenizer=tokenize, min_df=3, max_df=0.9, strip_accents='unicode',
                      use_idf=True, smooth_idf=True, sublinear_tf=True)
trn_term_doc = vec.fit_transform(train[COMMENT])
```

4. Linear Decision Boundary: Logistic Regression models the decision boundary as a linear function. This simplicity often leads to good generalization on test data, especially when the classes are linearly separable or nearly so.

5. Regularization: Logistic Regression in Scikit-learn supports regularization (e.g., L2 regularization). Regularization helps prevent overfitting, which is essential for models trained on text data with high dimensionality.

6. Probabilistic Output: The Logistic Regression model provides probabilistic outputs. This is useful in scenarios where you need to know the confidence level of the predictions, not just the class labels.

Logistic Regression is chosen for text classification due to its simplicity, efficiency, effectiveness with TF-IDF features, and the ability to provide probabilistic outputs. It is a linear model that generalizes well and can handle high-dimensional text data, making it a solid choice for this task.

4. Model Building & training

Model building involves constructing the chosen architecture using a suitable framework like TensorFlow, Keras, or PyTorch. This step includes defining the model layers, specifying the input and output formats, and compiling the model with appropriate loss functions, optimizers, and metrics. Building the model requires understanding the task requirements and designing the architecture to capture the necessary patterns in the data. It's crucial to experiment with different configurations to find the optimal structure. The process may involve building custom layers, handling complex data pipelines, and ensuring the model is efficient and scalable.

```
def train_model():
    train = pd.read_csv('train.csv')
    COMMENT = 'comment_text'
    train[COMMENT].fillna("unknown", inplace=True)

    vec = TfidfVectorizer(ngram_range=(1, 2), tokenizer=tokenize, min_df=3, max_df=0.9, strip_accents='unicode',
                          use_idf=True, smooth_idf=True, sublinear_tf=True)
    trn_term_doc = vec.fit_transform(train[COMMENT])

    label_cols = ['toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate']
    models = []
    for j in label_cols:
        m = LogisticRegression(C=4, penalty='l2', solver='liblinear')
        m.fit(trn_term_doc, train[j])
        models.append(m)

    return models, vec, label_cols
```

Training the model is the process of teaching the model to make accurate predictions by feeding it the training data. This involves iteratively updating the model's weights to minimize the loss function using algorithms like gradient descent. Hyperparameter tuning, such as adjusting learning rates, batch sizes, and epochs, is critical to achieving good performance. Training also includes validating the model on the validation set to monitor overfitting. Saving model checkpoints during training ensures that the best-performing models are preserved. This phase is computationally intensive and may require GPUs for faster processing.

In our toxic comments classification model, prediction refers to the process of using the trained model to identify and categorize new or unseen comments as toxic or non-toxic. This crucial step leverages the model's ability to generalize from the patterns it has learned during the training phase, allowing it to make informed predictions on new data.

Training Phase Recap

The model is initially trained on a comprehensive dataset of labeled comments, which includes both toxic and non-toxic examples. This dataset is meticulously prepared to ensure it captures a wide range of language nuances, including variations in spelling, grammar, slang, and context. The training process involves multiple iterations where the model adjusts its internal parameters to minimize the error in its predictions, guided by optimization algorithms like

```
# Predict text classification
def predict_text_classification(txt, model_data):
    array, vec, label_cols = model_data
    vtxt = vec.transform([txt])
    predsx = np.zeros((1, len(label_cols)))
    for i, m in enumerate(array):
        predsx[:, i] = m.predict_proba(vtxt)[:, 1]
    return predsx[0]
```

gradient descent.

Prediction Process

Once the model is trained, it can be deployed to classify new comments. The prediction process involves several key steps:

1. **Preprocessing**: New comments are first pre-processed to match the format and structure of the training data. This step includes tokenization, normalization and possibly encoding the text into a numerical format using techniques such as word embeddings or one-hot encoding.

2. **Feature Extraction**: The pre-processed comment is then passed through the model's layers. In a neural network, these layers typically include convolutional layers (for CNNs), recurrent layers (for RNNs), or attention mechanisms (for transformers), each designed to extract relevant features from the text. These features capture various aspects of the comment, such as syntax, semantics, and contextual information.

3. **Classification**: The extracted features are fed into the model's final output layer, which computes a probability distribution over the possible classes (toxic or non-toxic). This is often done using a softmax function that ensures the output probabilities sum to one. The class with the highest probability is

then selected as the predicted class for the input comment.

Factors Influencing Prediction Accuracy

The accuracy of the prediction depends on several critical factors:

1. **Quality and Diversity of Training Data**: A diverse and high-quality training dataset that accurately represents the different types of comments encountered in real-world scenarios significantly improves the model's generalization capabilities.

2. **Model Complexity and Architecture**: The choice of model architecture (e.g., simple logistic regression versus complex deep learning models like BERT) and its complexity (number of layers, parameters) play a crucial role in the model's ability to capture intricate patterns in the data.

3. **Algorithm Efficiency**: The efficiency and robustness of the prediction algorithm, including how well it handles edge cases and its ability to minimize false positives and negatives, directly impact the model's performance.

4. **Regularization and Overfitting**: Proper regularization techniques (like dropout, L2 regularization) and strategies to prevent overfitting (like cross-validation, early stopping) ensure that the model performs well on unseen data and is not just memorizing the training examples.

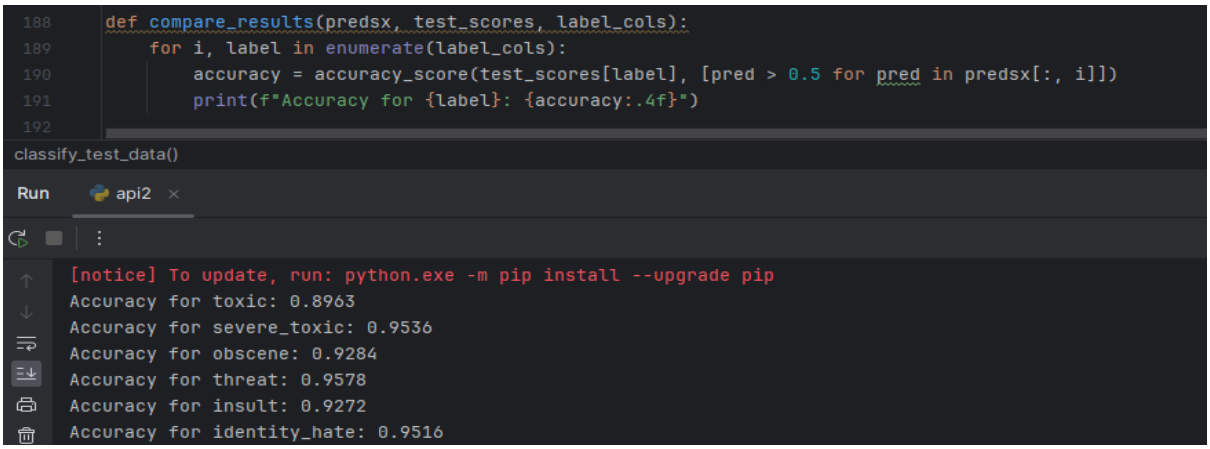
Continuous Improvement

To maintain and improve the accuracy of the toxic comments classification model, continuous monitoring and updating are essential. This involves regularly retraining the model with new data to adapt to evolving language trends and emerging toxic behaviours. Additionally, incorporating feedback from users and moderators can help refine the model's predictions and reduce instances of misclassification.

By leveraging advanced NLP techniques and continually refining the model, the toxic comments classification system can significantly contribute to creating a safer and more respectful online environment, enhancing user experience and community health.

Model Evaluation

Model evaluation assesses the trained model's performance using the test set. This step involves calculating metrics like accuracy, precision, recall, F1-score, and confusion matrix to gauge how well the model performs on unseen data. Evaluation helps identify the model's strengths and weaknesses. Performing error analysis on misclassified instances can provide insights for improving the model. It's essential to ensure that the model generalizes well and performs consistently across different subsets of the data. Tools like Scikit-learn provide utilities for comprehensive model evaluation.



The accuracy of our toxic comments classification model ranges between 89% and 95%, demonstrating its high effectiveness in identifying toxic content. Accuracy is a common performance metric used to evaluate the effectiveness of our model in classification tasks. It measures the proportion of correct predictions made by the model, i.e., the number of correct predictions divided by the total number of predictions made. In other words, accuracy quantifies the model's ability to correctly classify both positive and negative samples. Mathematically, accuracy can be defined as:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{False Positives} + \text{True Negatives} + \text{False Negatives}}$$

True Positives are the number of correctly predicted positive samples, False Positives are the number of negative samples that were incorrectly predicted as positive, True Negatives are the number of correctly predicted negative samples, and False Negatives are the number of positive samples that were incorrectly predicted as negative. This high accuracy level is crucial for maintaining a safe and respectful online environment, effectively filtering harmful content while

preserving legitimate discourse. Factors contributing to this performance include a diverse and comprehensive training dataset, advanced neural network architecture, and robust preprocessing techniques. Continuous updates and retraining ensure the model adapts to evolving language patterns.

Deployment

Deployment involves making the trained model available for use in a production environment. by developing an inference pipeline to handle real-time or batch predictions. Creating an API using frameworks like Fast API or Flask allows users to interact with the model. Deployment also involves setting up monitoring to track the model's performance and detect issues like data drift. It's crucial to ensure that the deployment is scalable and secure. Proper deployment practices ensure that the model delivers consistent and reliable results in real-world applications.

```
# Save model data
def save_model(model_data, filename="model_data.pkl"):
    with open(filename, 'wb') as file:
        pickle.dump(model_data, file)

# Load model data
def load_model(filename="model_data.pkl"):
    try:
        with open(filename, 'rb') as file:
            model_data = pickle.load(file)
        return model_data
    except (EOFError, FileNotFoundError) as e:
        print(f"Error: {e}")
        return None
```

Model Saving and Loading Functions

Save Model Data

The `save_model` function is designed to serialize and save the model data to a file. This is useful for preserving the trained model's state so that it can be easily reloaded and used for predictions at a later time, without the need to retrain the model. Here's a breakdown of its functionality:

- **Function Definition**: `def save_model(model_data, filename="model_data.pkl")`
  - `model_data`: The data representing the trained model, typically including the model itself, feature vectorizer, and any other relevant metadata.
  - `filename`: The name of the file where the model data will be saved. The default is `"model_data.pkl"`.
- **File Handling**: The function opens the specified file in write-binary mode (`'wb'`).
- **Serialization**: The `pickle.dump` method is used to serialize the `model_data` object and write it to the file.
- **Usage**: This function is called after training the model to save the model data to disk.

Load Model Data

The `load_model` function is used to deserialize and load the model data from a file. This allows for the retrieval of the trained model and its associated data, enabling the model to be used for predictions without needing to be retrained. Here's a breakdown of its functionality:

- **Function Definition**: `def load_model(filename="model_data.pkl")`
  - `filename`: The name of the file from which the model data will be loaded. The default is `"model_data.pkl"`.
- **File Handling**: The function tries to open the specified file in read-binary mode (`'rb'`).
- **Deserialization**: The `pickle.load` method is used to deserialize the content of the file and load it into the `model_data` object.
- **Exception Handling**: The function handles two specific exceptions:
  - `EOFError`: This exception is caught if the end of the file is reached unexpectedly, indicating that the file might be corrupted or incomplete.
  - `FileNotFoundError`: This exception is caught if the specified file does not exist, indicating that the model data has not been saved yet or the file path is incorrect.
- **Return Value**: If successful, the function returns the `model_data`. If an exception occurs, it prints an error message and returns `None`.
- **Usage**: This function is called during the application startup or whenever model predictions are required to load the previously saved model data.

```
from flask import Flask, request, jsonify, send_from_directory
from flask_cors import CORS
```

In this project, Flask is used to create a web-based API for toxic comment classification. Flask is a lightweight and flexible web framework for Python It allows us to handle HTTP requests, manage user authentication, and serve files. The endpoints defined in the Flask application enable functionalities like text classification, file processing, and user management, making it a versatile choice for deploying machine learning models in a web-accessible manner. Additionally, Flask's support for extensions like CORS facilitates seamless integration with front-end applications.

Home Page Functions

```python
def process_file(file_path, translator):
    file_extension = os.path.splitext(file_path)[-1].lower()
    if file_extension in ['.png', '.jpg', '.jpeg', '.bmp', '.gif']:
        img = cv2.imread(file_path)
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        _, thresh = cv2.threshold(gray, thresh: 0, maxval: 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
        extracted_text = pytesseract.image_to_string(thresh)
    elif file_extension == '.pdf':
        text = ""
        with open(file_path, 'rb') as file:
            reader = PyPDF2.PdfReader(file)
            for page in reader.pages:
                text += page.extract_text()
        extracted_text = text
    elif file_extension == '.docx':
        doc = Document(file_path)
        text = ""
        for paragraph in doc.paragraphs:
            text += paragraph.text + "\n"
        extracted_text = text
    elif file_extension in ['.wav']:
        recognizer = sr.Recognizer()
        with sr.AudioFile(file_path) as source:
            audio_data = recognizer.record(source)
        try:
            extracted_text = recognizer.recognize_google(audio_data)
        except sr.UnknownValueError:
            extracted_text = "Google Speech Recognition could not understand audio"
        except sr.RequestError as e:
            extracted_text = f"Could not request results from Google Speech Recognition service; {e}"
    else:
        extracted_text = None
```

1- Extract text from file：

The `process_file` function is a versatile tool designed to extract textual content from an array of file formats. Whether it's images, PDFs, Word documents, or audio files, this function seamlessly handles the complexities of digital media, transforming them into a unified text format ready for further analysis and interpretation.

Ease of Use: To streamline the user experience and simplify the process, the function automatically identifies the type of file uploaded. This eliminates the need for users to specify the file type, allowing them to upload any supported file with ease. Once uploaded, the extracted data is presented in a text area on the home page, where users can directly classify it or make edits before classification.

Supported File Types: The `process_file` function supports a variety of file types, including:

- **Documents:** PDF and DOCX formats.
- **Images:** PNG, JPG, JPEG, and more.
- **Speech Recognition:** WAV audio files.

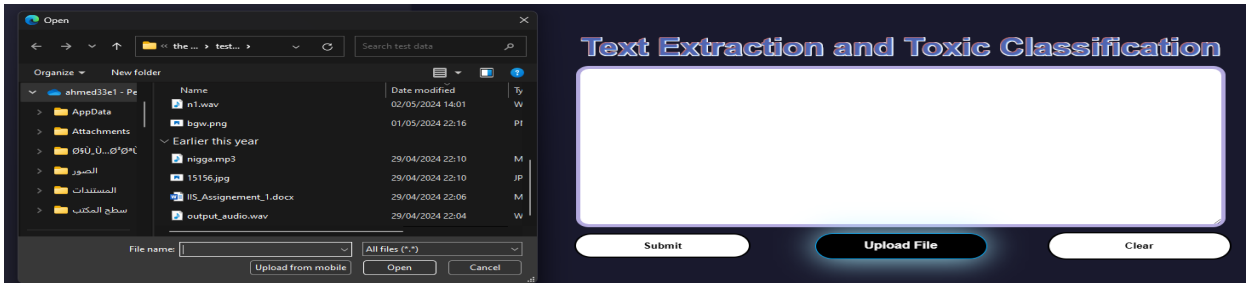
Scalability Considerations: While the current implementation supports multiple file types, it's important to note that handling additional formats may increase the

load on the model server. Therefore, careful consideration should be given to scalability when expanding the range of supported file types.

1- Web view

The design of the web app is desired to be simple easy to use and limit the options if the user into simple choices avoiding user case errors and mistakes

The user can upload or drag and drop the files he desires



after uploading the file it get processed into the process file function generating a text that will be sent to the end point flask end point it will receive the text data and send it to the front to be fetched so the data will be populated into the text area

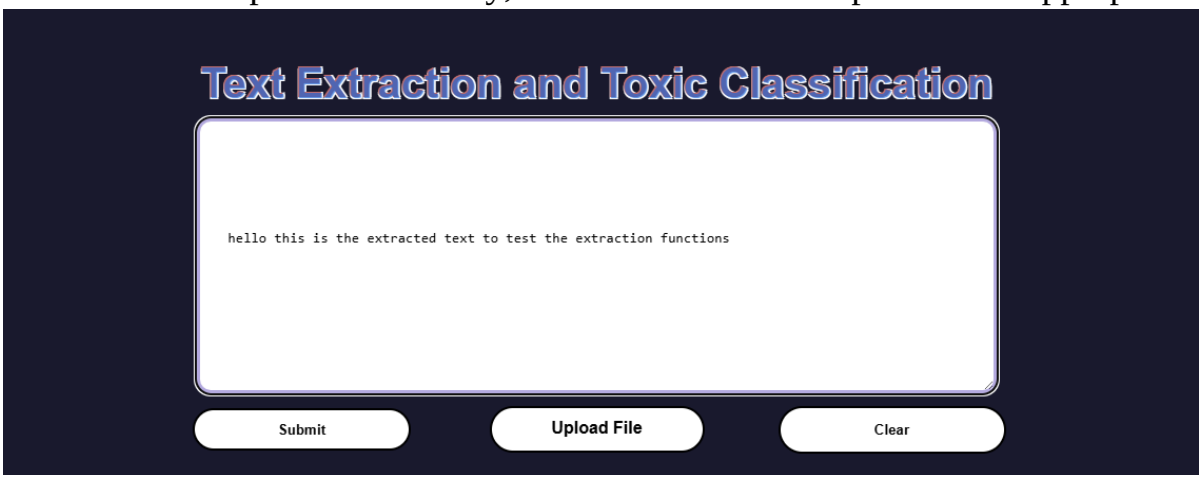
end point for process file function

```python
@app.route( rule: '/process_file', methods=['POST'])
def process_file_endpoint():
    if 'file' not in request.files:
        return jsonify({'error': 'No file part'})

    file = request.files['file']
    if file.filename == '':
        return jsonify({'error': 'No selected file'})

    if file:
        filename = secure_filename(file.filename)
        file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(file_path)
        result = process_file(file_path, translator)
        return jsonify(result)
```

Text get populated into text area user can edit or clear it by clear button

2- Text-Classification :

The `classify_text` function serves as a core component within the text classification framework, enabling the prediction of labels for input text data. Given the plain text (`txt`) or the text extracted via the `process_file` function, along with the essential model-related data (`model_data`), this function initiates the classification process. Initially, it ensures that the input text is appropriately



translated, if necessary, to maintain consistency in language processing. Subsequently, leveraging the `predict_text_classification` function, it generates predictions based on the processed text, utilizing the provided model. These predictions are then formatted to extract the probability scores for each classification label, providing valuable insights into the likelihood of the input text belonging to different categories. Ultimately, the function returns a dictionary containing these probability scores, thereby facilitating efficient and accurate label predictions for a diverse range of textual data.

```
3 usages
def classify_text(txt, model_data, translator):
    translated_text = translator.translate(txt, dest='en').text
    predsx = predict_text_classification(translated_text, model_data)
    _, _, label_cols = model_data
    return get_label_proba(predsx, label_cols)
```

In conclusion, the `classify_text` function represents a critical component within the text classification pipeline, enabling the efficient prediction of labels for input text data. By leveraging the provided model data and translation capabilities, it ensures robust and accurate classification results across various text sources. Through its seamless integration with the

classification framework, this function contributes to the overall effectiveness and reliability of the text classification system, empowering users to gain valuable insights from textual data with confidence and ease.

Once the user presses the submit button the app sends the text in the text area by js function to the backend point of the classification end point

The `/classify_text` endpoint serves as a crucial gateway for classifying textual data within the web application. Upon receiving a POST request, it extracts the input text from the request form data. Subsequently, the function invokes the `classify_text` method, passing the extracted text along with the model data and translation capabilities. This initiates the text classification process, where the input text undergoes prediction based on the pre-trained model. Upon successful classification, the function returns the classification result in JSON format. However, if any errors occur during processing, it gracefully handles them by returning an error message. Overall, this endpoint facilitates the seamless integration of text classification functionality within the web application, enabling users to obtain classification results effortlessly.

```python
@app.route( rule: '/classify_text', methods=['POST'])
def classify_text_endpoint():
    input_text = request.form.get('text')
    print("Received text:", input_text)
    if input_text:
        try:
            result = classify_text(input_text, model_data, translator)
            print("Classification result:", result)
            return jsonify(result)
        except Exception as e:
            print(f"Error processing request: {e}")
            return jsonify({'error': 'An error occurred while processing the request'})
    else:
        return jsonify({'error': 'Missing required form parameter "text"'})
```

After the data received from the end point the classification result get populated into a table of labels aside of percentages in text for each label and a progress bar for each

3- History:

In order to enhance user experience and streamline the reclassification process, a history feature has been implemented within the application. This history functionality allows users to conveniently reclassify previously processed data within the same session. By maintaining a record of the most recent classifications, users can easily refer back to and reevaluate prior results without the need to re-upload or reprocess the data. The history feature is designed to store up to 10 elements, providing users with quick access to recent classifications. Additionally, the system is architected to support potential expansion of the history capacity in future iterations of the application, ensuring scalability and adaptability to evolving user needs. With the history feature in place, users can navigate through their recent classification activities with ease, thereby improving efficiency and productivity within the application

In addition to providing users with the ability to access recent classifications, the history section incorporates interactive functionalities to further enhance user control and customization. Each entry in the history is accompanied by an "X" button, enabling users to conveniently remove individual history items with a single click. This granular control empowers users to manage their history list efficiently, allowing them to declutter and organize their classification records as needed.

Furthermore, for users seeking to clear their entire history at once, a dedicated "Clear History" button is prominently displayed within the interface. By clicking this button, users can swiftly erase all entries from their history list in one action. This comprehensive clearance option offers users a quick and efficient method to reset their classification history, providing a clean slate for future classification activities. Together, these interactive features within the history section ensure a seamless and intuitive user experience, allowing users to tailor their history management according to their preferences and workflow requirements.

incorporating additional insights into the history section, each classification entry is augmented with an average score representing the cumulative classification results associated with that particular text. This average score provides users with a concise summary of the classification tendencies for a given text over multiple classification instances.

By displaying the average classification score alongside each history element, users gain valuable context regarding the overall sentiment or classification pattern exhibited by the text across different classification sessions. This insight can aid users in assessing the consistency and reliability of the classification outcomes over time, allowing them to make more informed decisions based on the historical performance of the text.

Furthermore, the inclusion of the average score contributes to a more comprehensive understanding of the text's classification dynamics, enabling users to identify trends or fluctuations in sentiment or classification probabilities. This enhanced visibility into the historical classification data empowers users to derive deeper insights from their classification history, facilitating more informed decision-making and analysis within the application.

When clicking on any button of the history data it get placed into the text area ready to be classified again to see detailed classification results according to the table labels

after clicking the “x” button icon the element assigned to it gets removed from the history

when clicking the clear history all elements of the history get deleted

4- Admin Page:

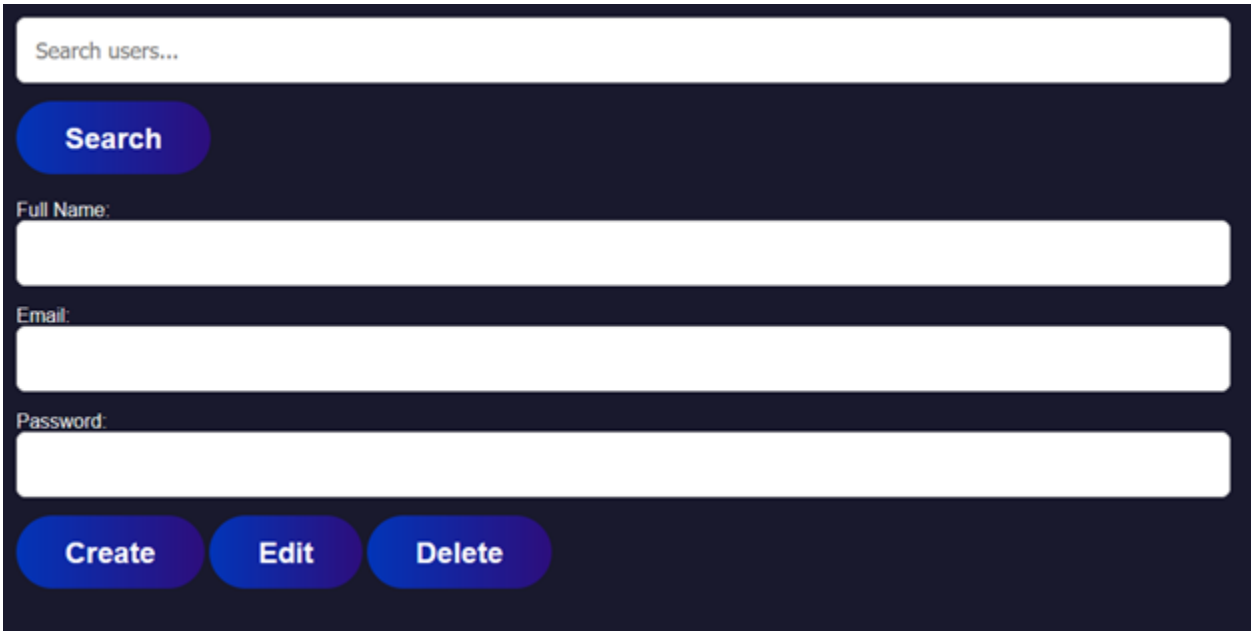
The admin page of the application includes a critical function, `get_users`, designed to retrieve and display user information from the database. When accessed via the `/users` endpoint with a GET request, this function initiates a connection to the database using the `create_connection` utility. Upon establishing a connection, it executes a SQL query to fetch all records from the "Users" table. The function then

ID	Full Name	Email
1	ahmed khaled	ahmedkatame@gmail.com
4	mohammed magdy	mmagdy2001@gmial.com
10	user two	user2@user.com
12	user	user@user.com
13	john	customer@user.com

constructs a list of user dictionaries by mapping each database row to a dictionary, utilizing the column names for key assignments. This structured data is subsequently converted to JSON format, providing a clear and readable response for the admin interface. The `get_users` function is automatically triggered to populate the user table on the front end, ensuring that the latest user information is readily available for administrative tasks. Additionally, robust error handling is implemented to manage any potential issues during data retrieval. In case of an error, the function logs a detailed error message and returns a generic error response to the client, ensuring smooth operation and user feedback. The integration of this functionality into the admin page facilitates efficient user management, allowing administrators to monitor and manage user data effectively and effortlessly.

5 - Admin Page Functions

the admin page offers comprehensive user management capabilities, including searching, creating, editing, and deleting user records. These functions ensure efficient and effective administration of the user database, providing a streamlined interface for maintaining user information. With built-in error handling and JSON responses, the application facilitates seamless user interactions and data management, enhancing overall system functionality and user experience.



Search User by ID or Email

The admin page allows administrators to search for users by their ID or email using the `/user` endpoint with a GET request. Upon receiving the request, the `get_user` function establishes a database connection and retrieves the provided user ID or email from the request arguments. Depending on the input, it constructs and executes an appropriate SQL query to fetch the user details. If a matching user is found, the function formats the user data into a dictionary and returns it in JSON format. If no user is found, or if neither ID nor email is provided, it returns an error message. This functionality is essential for quickly locating specific user records based on key identifiers.

```python
@app.route( rule: '/user', methods=['GET'])
def get_user():
    try:

        connection = create_connection()
        cursor = connection.cursor()

        user_id = request.args.get('id')
        user_email = request.args.get('email')

        if user_id:
            query = f"SELECT * FROM users WHERE id = {user_id}"
        elif user_email:
            query = f"SELECT * FROM users WHERE email = '{user_email}'"
        else:
            return jsonify({"error": "Please provide either user ID or email."}), 400

        cursor.execute(query)
        user = cursor.fetchone()

        if user:

            user_data = {
                "id": user[0],
                "full_name": user[1],
                "email": user[2],
                "password": user[3]
            }
            return jsonify(user_data)
        else:
            return jsonify({"error": "User not found."}), 404
    except Exception as e:
        return jsonify({"error": str(e)}), 500
    finally:

        if connection:
            connection.close()
```

Create User

To facilitate the addition of new users, the admin page includes a `create_user` function accessible via the `/users` endpoint with a POST request. This function extracts the full name, email, and password from the request arguments and inserts these details into the "Users" table in the database. Upon successful insertion, the function commits the changes and returns a success message in JSON format. In case of any errors during the process, it returns an error message. This capability allows administrators to efficiently add new users to the system, ensuring that the user database remains up-to-date with minimal effort.

```python
@app.route( rule: '/users', methods=['POST'])
def create_user():
    try:

        full_name = request.args.get('full_name')
        email = request.args.get('email')
        password = request.args.get('password')
        print("hi")

        connection = create_connection()
        cursor = connection.cursor()

        cursor.execute( sql: "INSERT INTO Users (full_name, email, password) VALUES (?, ?, ?)",  *params: (full_name, email, password))
        connection.commit()

        print("User added successfully.")

        return jsonify({"message": "User created successfully"})
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

Edit User

The admin page provides a `update_user` function for editing existing user information, accessible via the `/users/<int:user_id>` endpoint with a PUT request. This function retrieves the user ID from the URL and the new full name, email, and password from the request body. It then constructs and executes an SQL query to update the user's details in the database. Upon successful completion, the function commits the changes and returns a success message in JSON format. Any errors encountered during the update process are captured and returned as error messages. This feature enables administrators to easily modify user details, maintaining the accuracy and relevance of user records.

```python
@app.route( rule: '/users/<int:user_id>', methods=['PUT'])
def update_user(user_id):
    try:
        connection = create_connection()
        cursor = connection.cursor()

        data = request.json
        full_name = data.get('full_name')
        email = data.get('email')
        password = data.get('password')

        cursor.execute( sql: "UPDATE Users SET full_name = ?, email = ?, password = ? WHERE ID = ?",  *params: (full_name, email, password, user_id))

        connection.commit()

        print(f"User with ID {user_id} updated successfully.")

        return jsonify({"message": "User updated successfully"})
    except Exception as e:
```

Delete User

For user management, the admin page includes a `delete_user` function, which can be invoked via the `/users/<int:user_id>` endpoint with a DELETE request. This function extracts the user ID from the URL and executes an SQL query to delete the corresponding user from the "Users" table. After successfully deleting the user, the function commits the changes and returns a success message in JSON format. If any errors occur during the deletion process, they are returned as error messages. This functionality provides administrators with the ability to remove users from the system as needed, ensuring that the user database is kept clean and current.

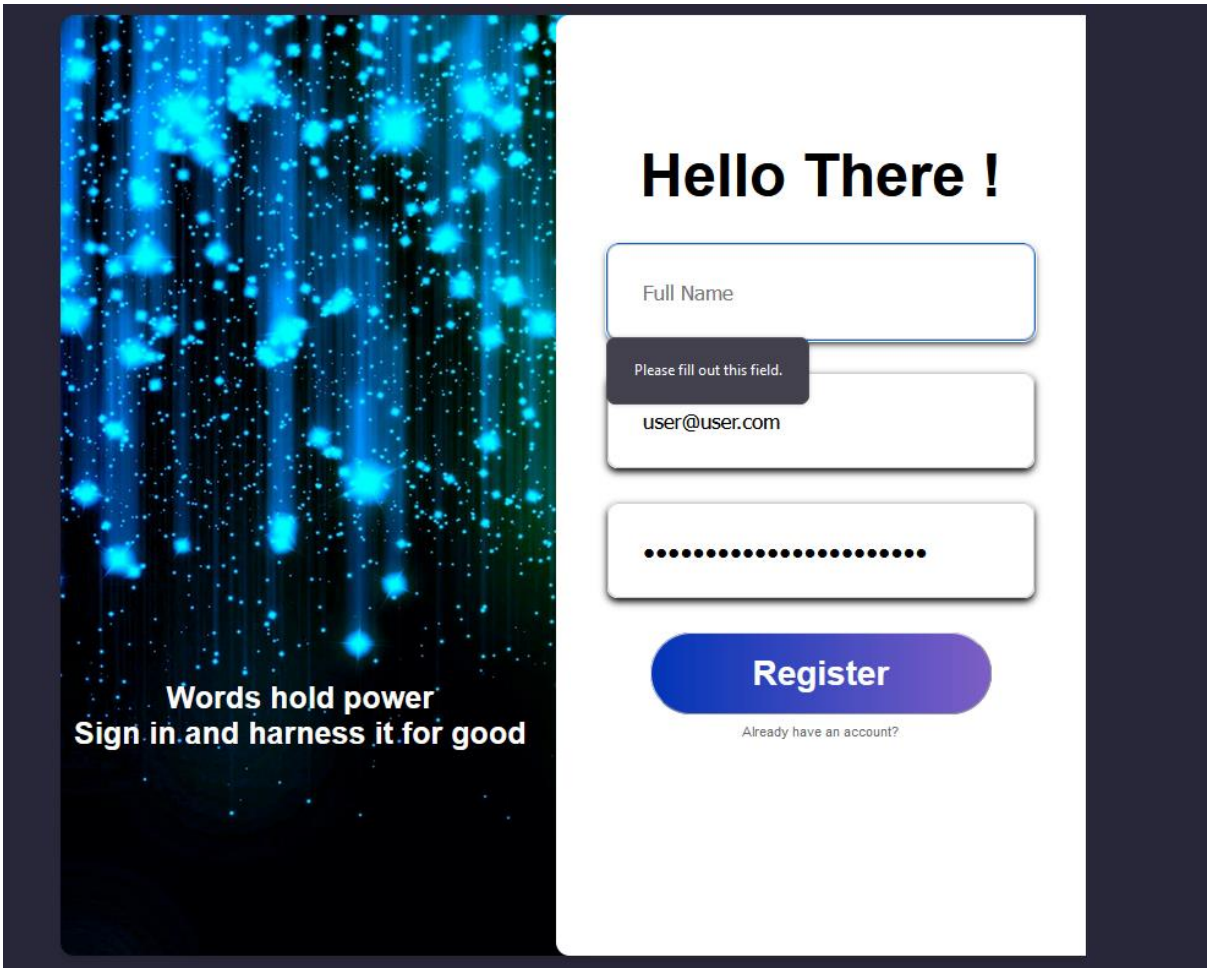
```python
@app.route( rule: '/users/<int:user_id>', methods=['DELETE'])
def delete_user(user_id):
    try:
        connection = create_connection()
        cursor = connection.cursor()

        cursor.execute( sql: "DELETE FROM Users WHERE ID = ?", *params: (user_id,))

        connection.commit()

        print(f"User with ID {user_id} deleted successfully.")

        return jsonify({"message": "User deleted successfully"})
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

Chapter 6
Testing And Evaluation

system testing and evaluation stand as formidable gatekeepers, ensuring the fortification of digital creations against the onslaught of user errors and unforeseen glitches. As we delve into the intricate web of software systems, we embark on a journey to scrutinize, refine, and fortify the digital edifices we construct. System testing, the vigilant sentinel of the software development lifecycle, is tasked with the noble duty of subjecting every component, every interaction, every pathway within the system to rigorous examination. Its mission: to ascertain the system's mettle, to validate its functionality. At the heart of system testing lies a profound commitment to the user experience, a dedication to crafting digital environments that not only function as intended but also embrace the fallibility of human interaction. For within the labyrinth of user interactions, lies a myriad of potential pitfalls – from the humble login screen to the sprawling expanse of the main page, each interaction point becomes a battleground where users and systems engage in a delicate dance of input and response. Consider, for instance, the humble act of user authentication – the gateway through which users traverse into the digital realm. Here, amidst the fields of usernames and passwords, lurk the Specters of user errors – erroneous data types, incomplete fields, and feeble passwords, each waiting to ensnare the unwary user. Through meticulous testing and evaluation, we endeavour to fortify this gateway, to erect barriers against the onslaught of user errors, and to guide users safely into the digital domain. But the realm of system testing extends far beyond the confines of authentication; it encompasses every facet of the digital experience. From the labyrinthine pathways of password recovery to the bustling thoroughfares of the main page, each interaction point presents its own set of challenges and opportunities. Here, users may stumble and falter – uploading files of unsupported formats, submitting forms with empty fields, or grappling with the complexities of forgotten passwords. In the crucible of system testing, we simulate these scenarios, orchestrating a symphony of user interactions and system responses, seeking to uncover weaknesses, anticipate pitfalls, and refine the digital experience. Through meticulous examination and iterative refinement, we sculpt digital landscapes that not only withstand the test of time but also embrace the inherent variability of human interaction. In essence, system testing and evaluation stand as bastions of quality assurance, guardians of user experience, and architects of digital resilience. Through their unwavering vigilance and meticulous scrutiny, they pave the way for the creation of software systems that not only function flawlessly but also inspire confidence, trust, and admiration in their users.

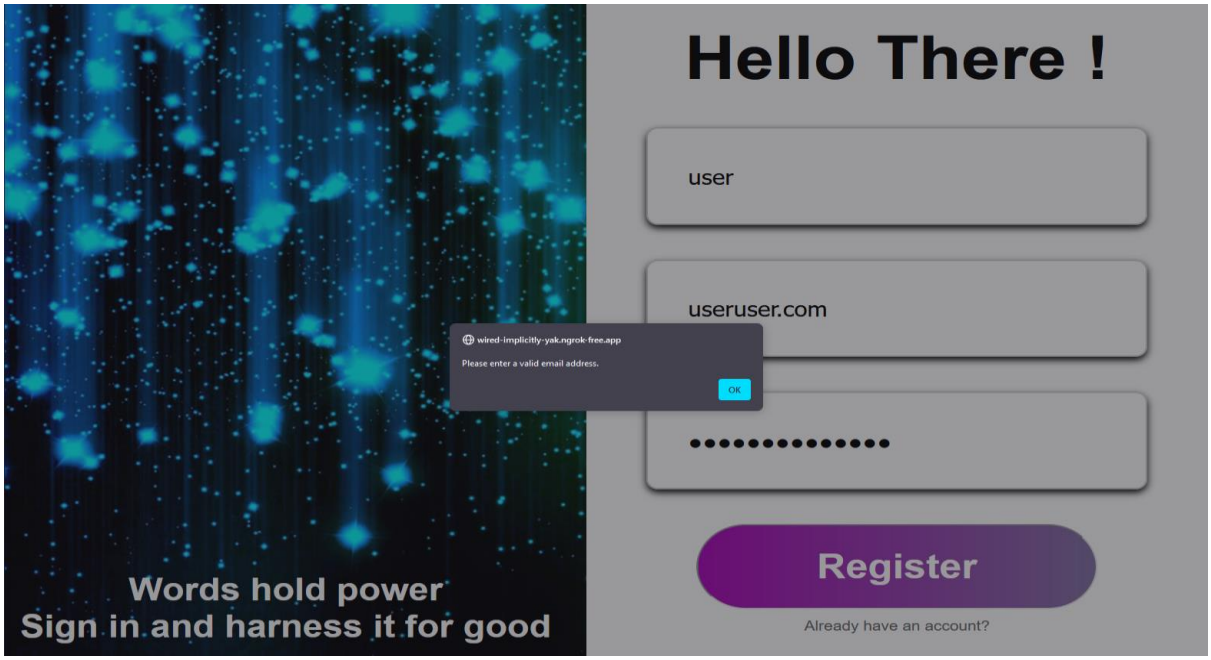
1-Registeration Testing : in this testing phase we will test the errors or the mistakes the user could make trying create an account for the first time

1.1 : Empty Fields : when the user leave one or more of the Fields empty the system will tell him to fill it with the right data



Empty Fields

1.2: wrong E-mail Typo (Bad Regex) : if the user entered an wrong typed text in email a bad regex for example ‘’useruser.com’’ the user will get a message him to enter valid email address



Bad E-mail

1.3: Already Registered : if the user entered an already registered email the system will tell him to and will suggest him to sign in by already have an account? Hyperlink label redirecting him to the sign in page

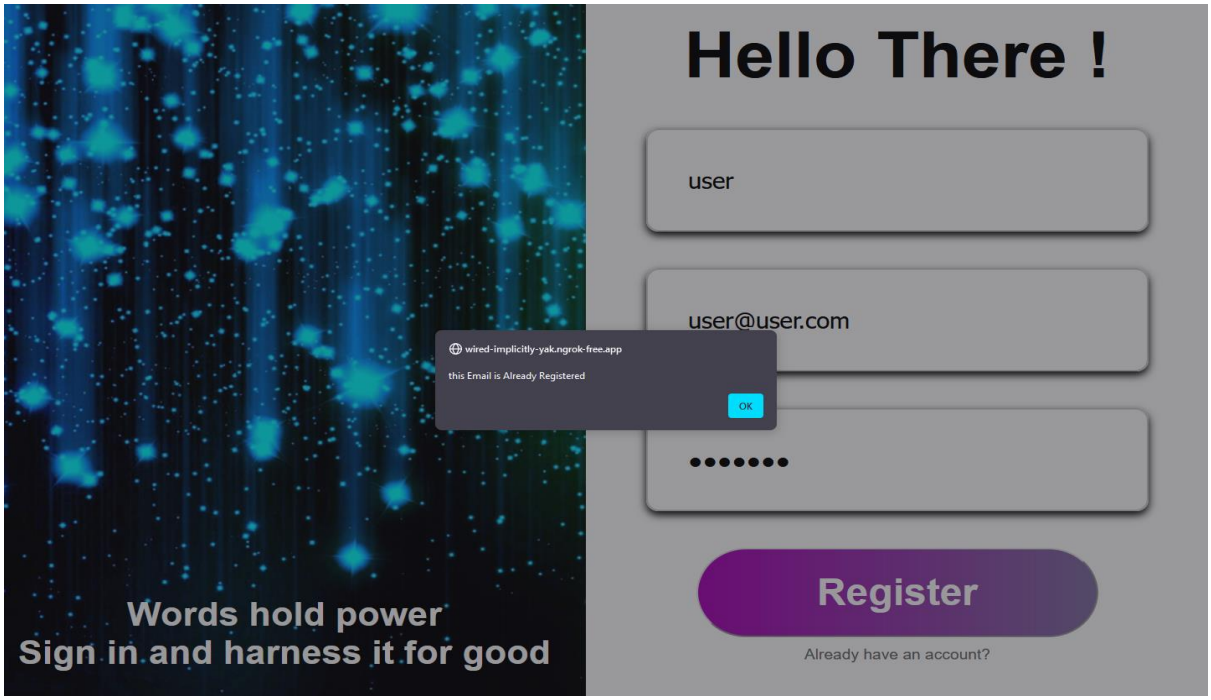
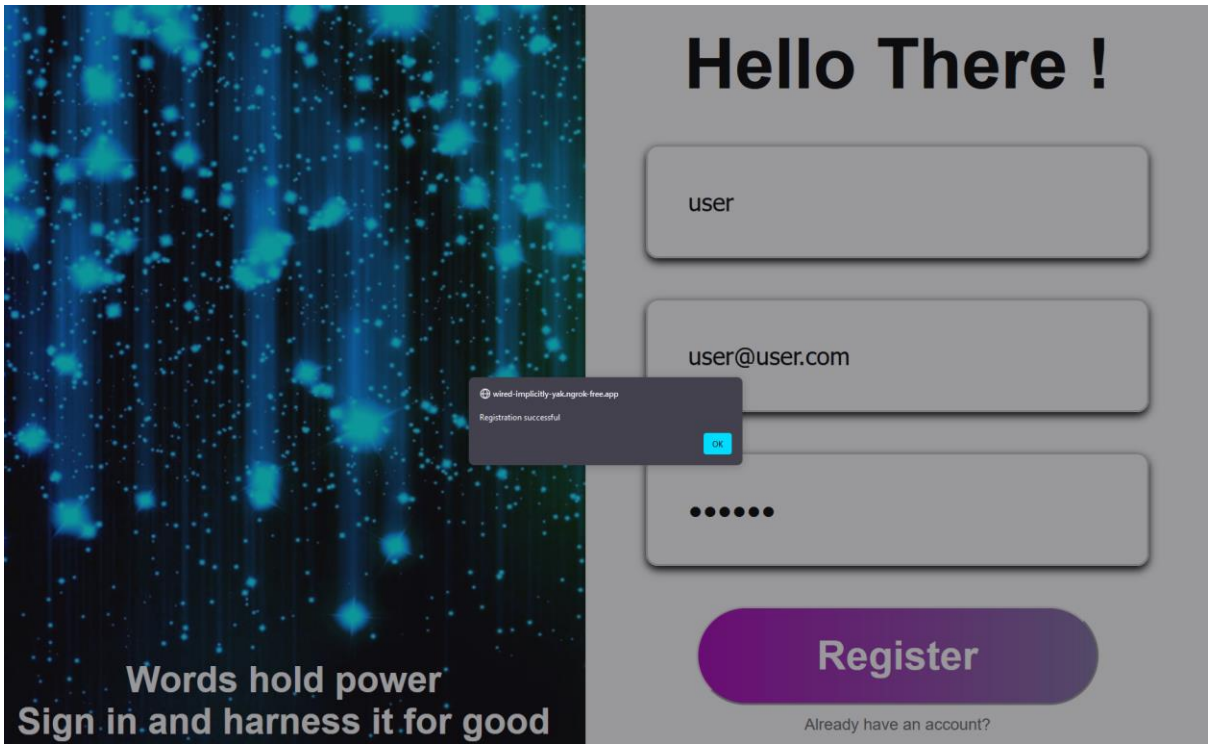


Figure 3 : Already Registered

1.4 : Registration success : here we will show how the user correctly make an account into the system correctly with right unused email

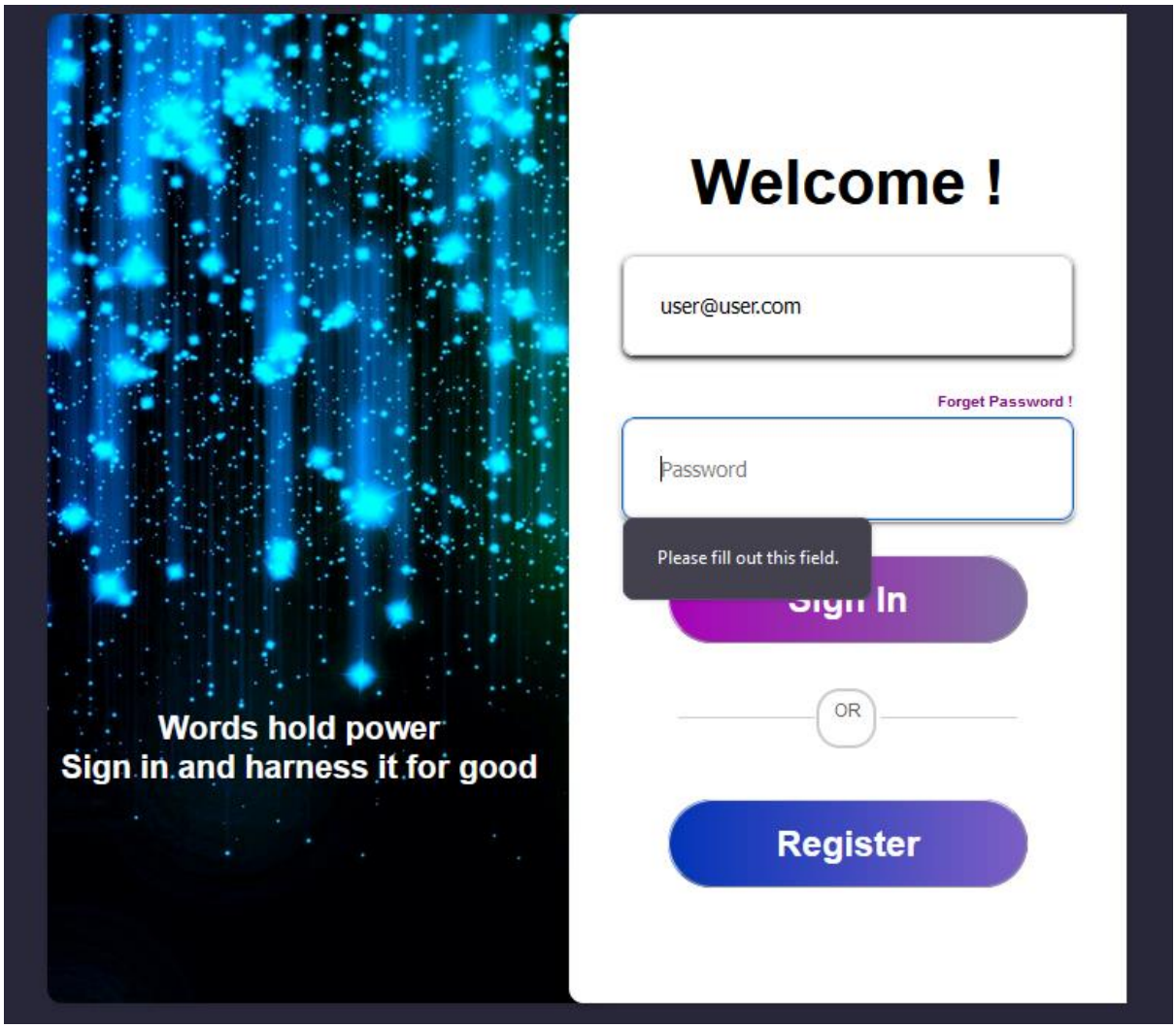
After so the user will be redirected to the main page



Registration successful

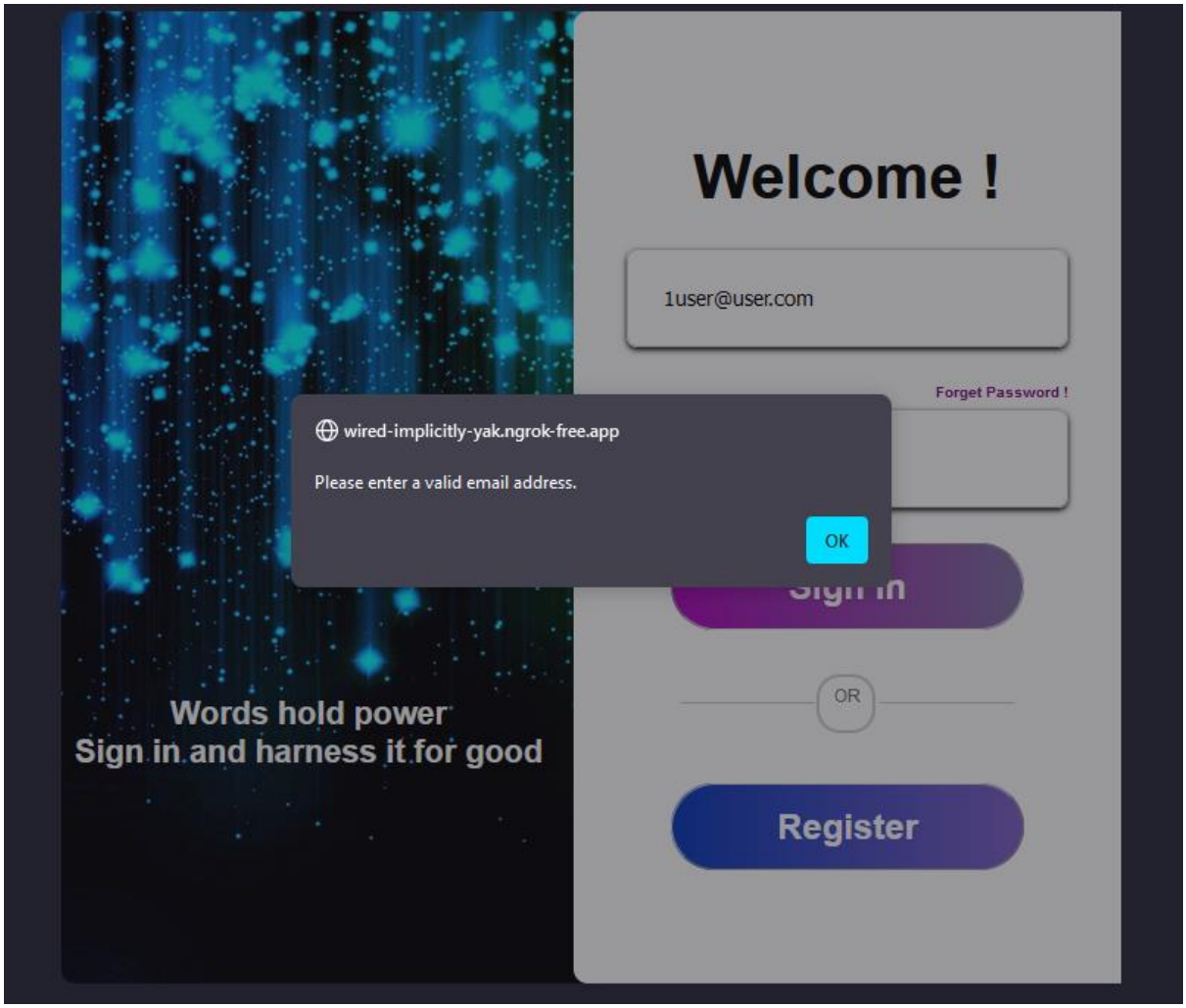
2-Sign in Testing : in this testing phase we will test the errors or the mistakes the user could make trying to enter the system and view to it how it could work well

2.1: Empty Fields : when the user leave one or more of the Fields empty the system will tell him to fill it with the right data



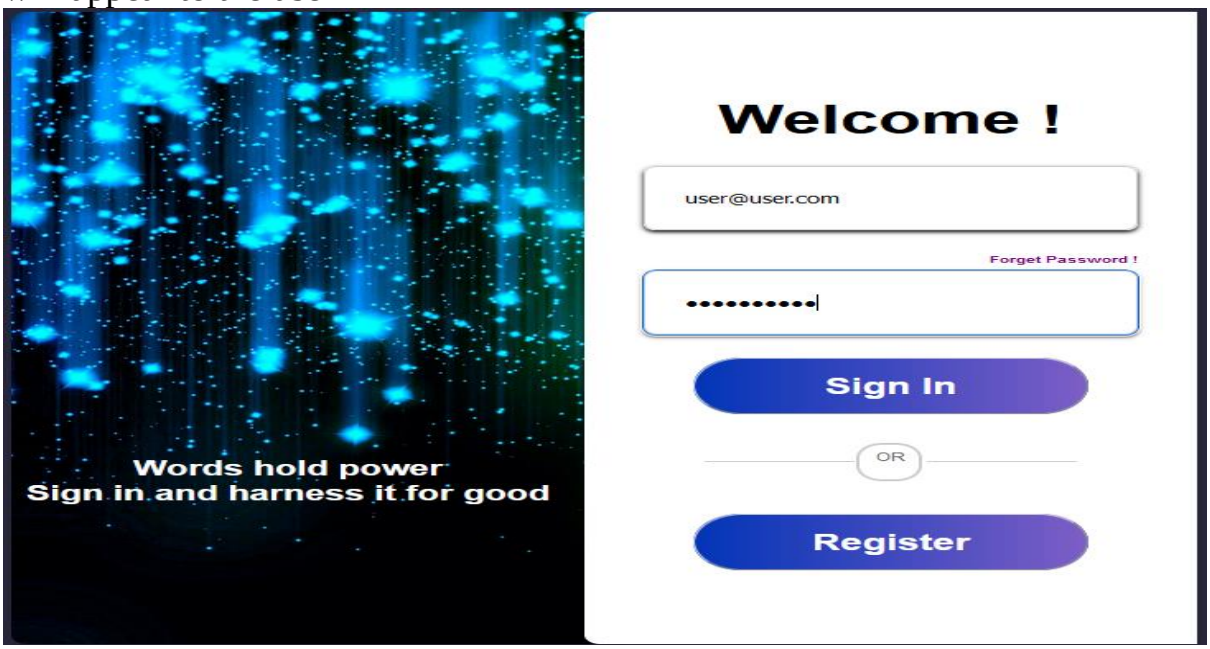
Empty Fields

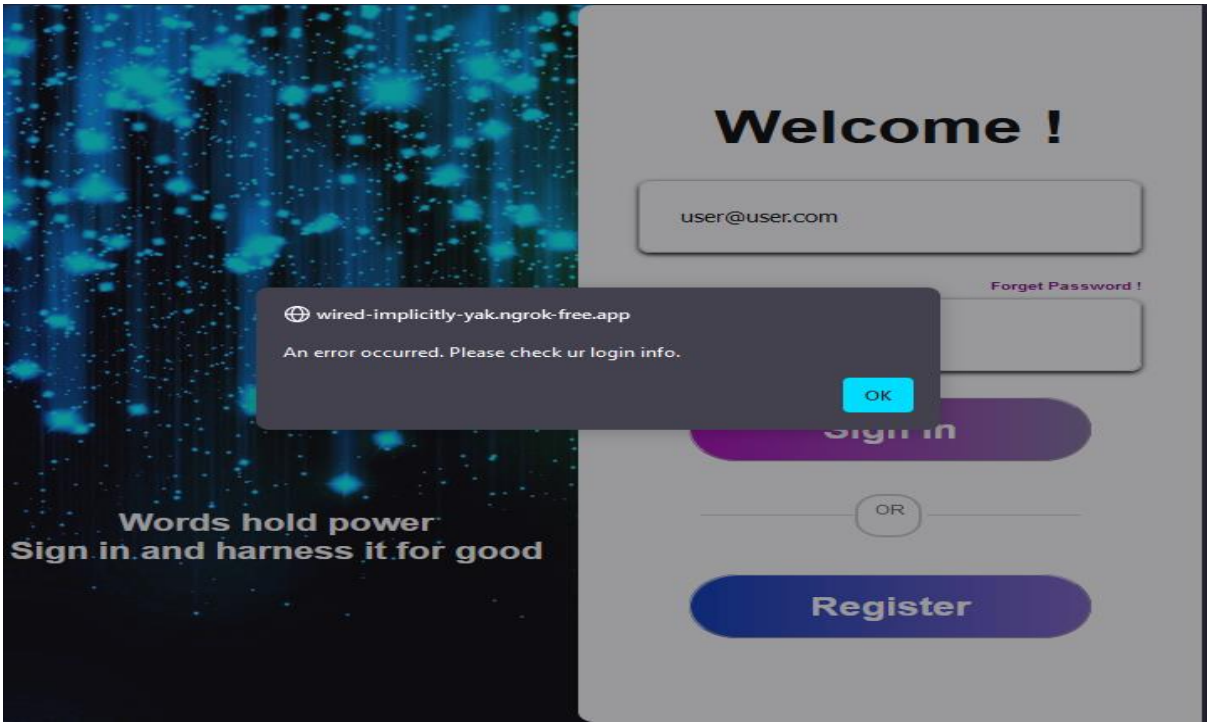
2.2: wrong E-mail Typo (Bad Regex) : if the user entered an wrong typed text in email a bad regex for example ‘’1user@user.com’’ the user will get a message telling him to enter valid email address



Bad E-mail

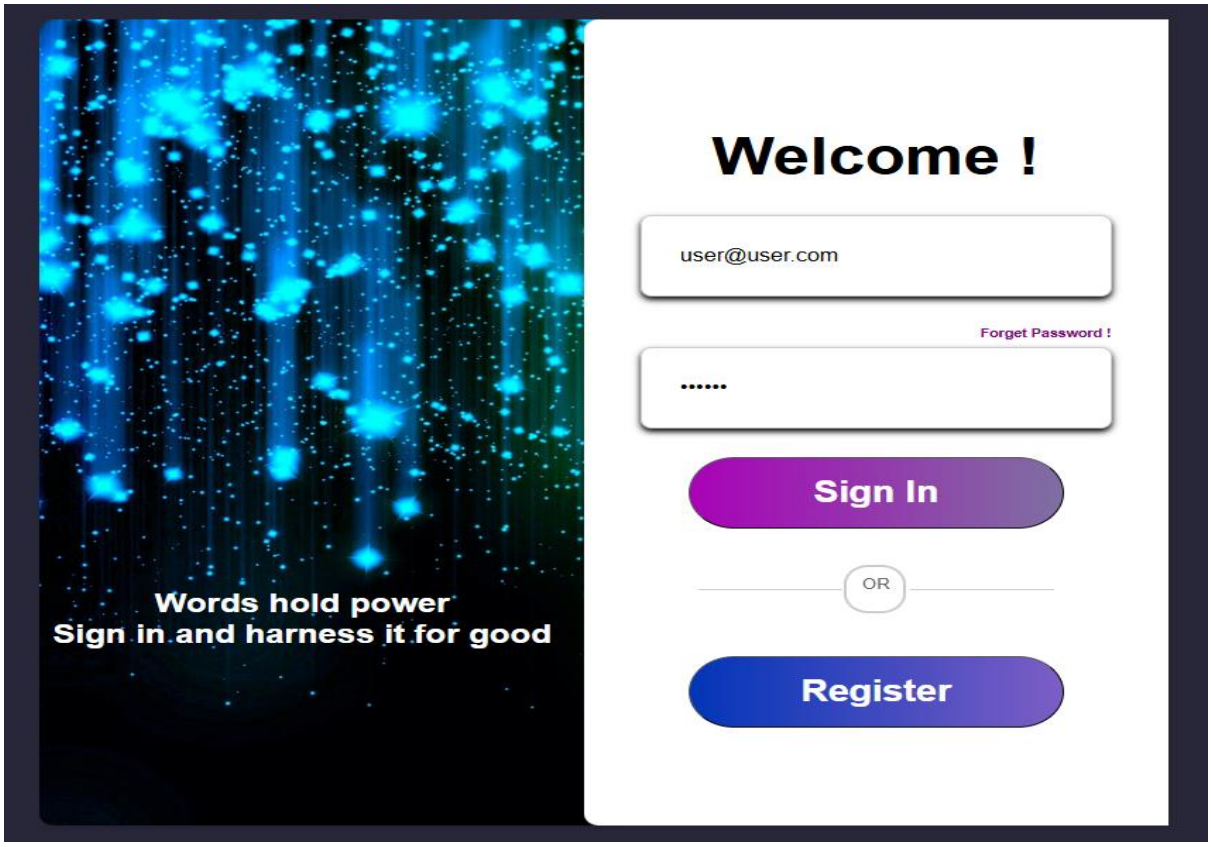
2.3: Wrong password or E-mail : when the user enters its credentials it get checked by the system in the database if the data is not matched an error message will appear to the user





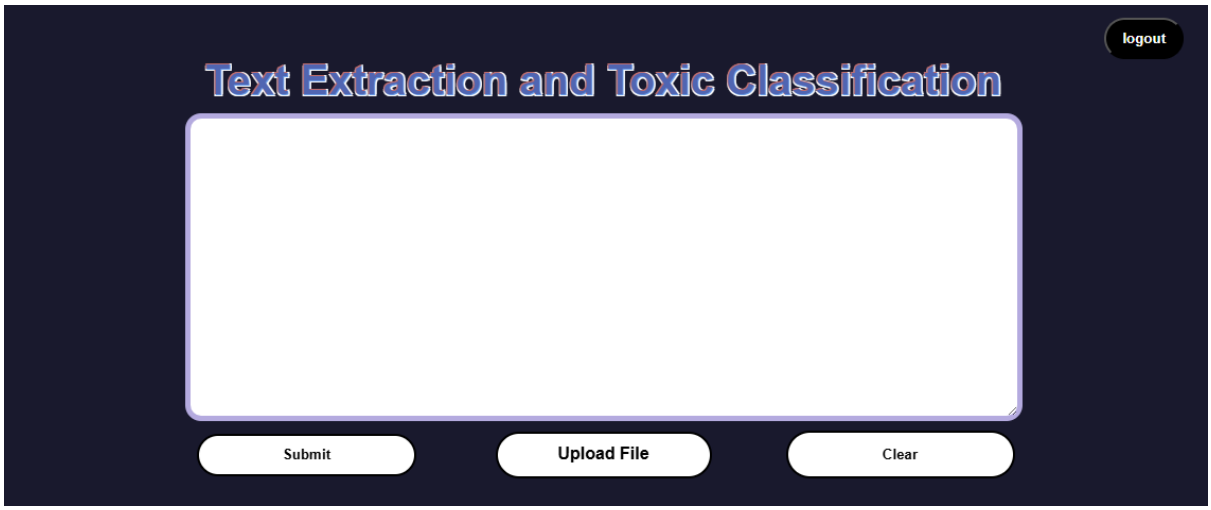
Wrong login info

2.4 : Sign in success : here we will show how the user gets into the System if the login info is correct!



Right login info

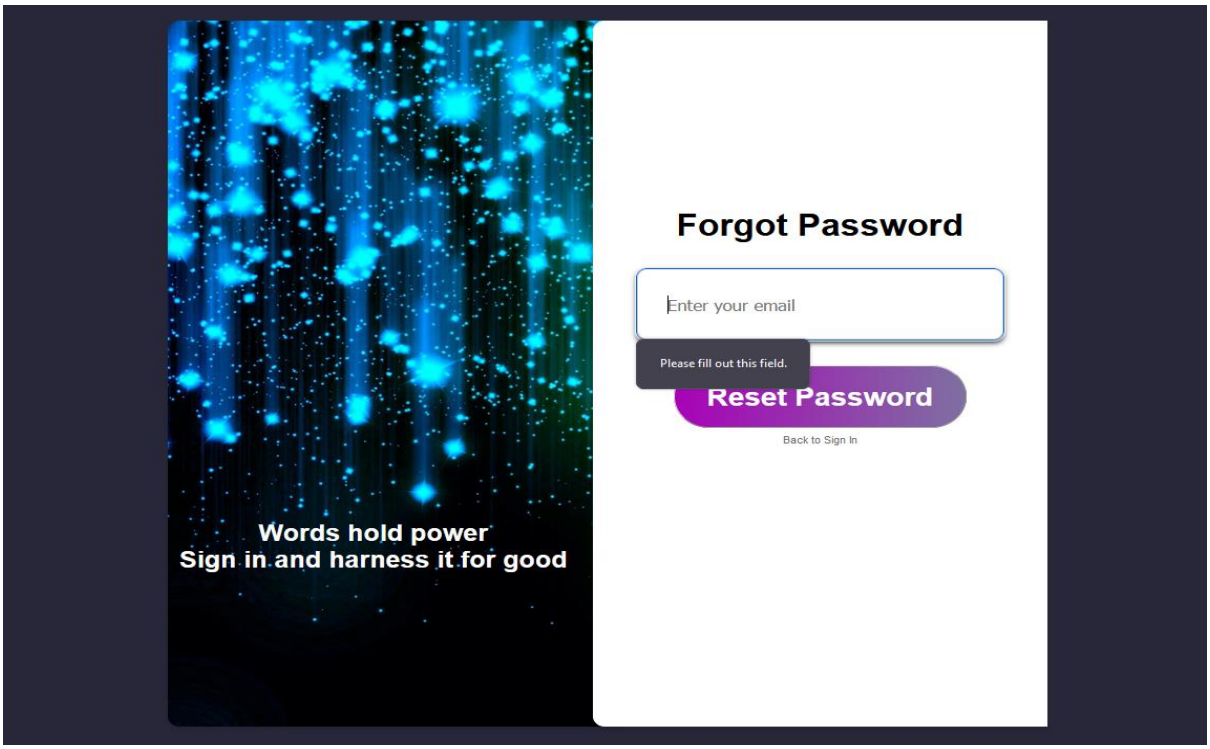
After pressing the sign in the user will get redirected to the main page



Main Page

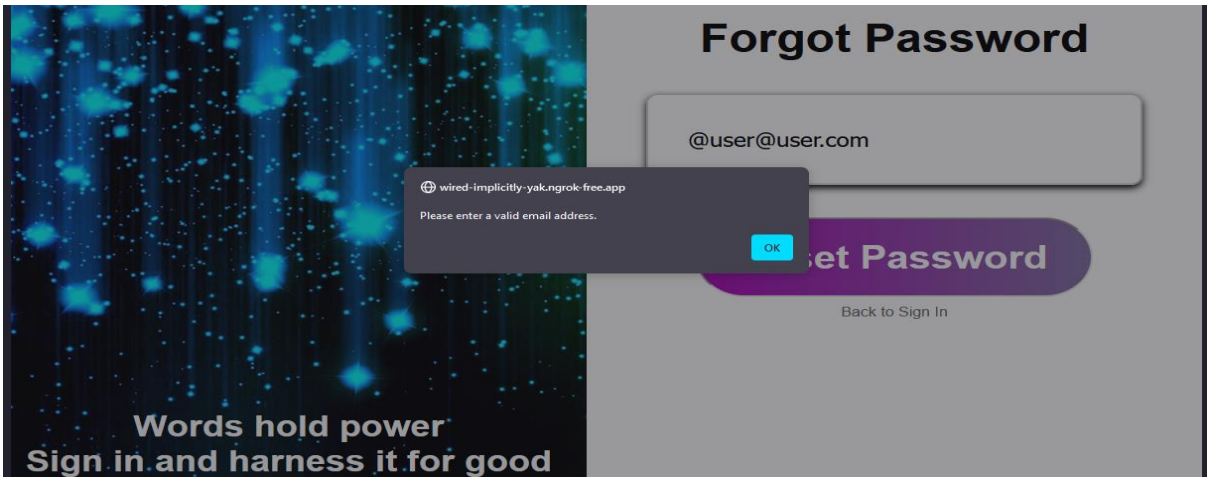
3-Forget Password Testing : in this testing phase we will test the errors or the mistakes the user could make trying to recover his forgotten password

3.1: Empty Fields : when the user leave one or more of the Fields empty the system will tell him to fill it with the right data



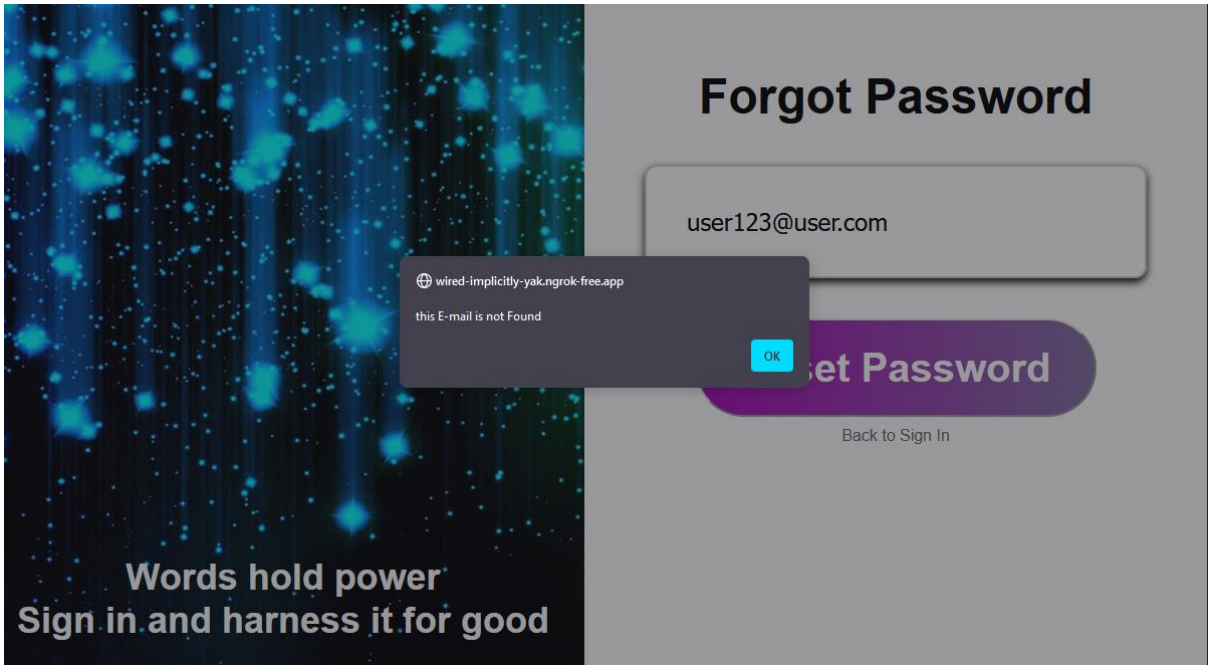
Empty Fields

3.2: wrong E-mail Typo (Bad Regex) : if the user entered an wrong typed text in email a bad regex for example ''@user@user.com'' the user will get a message telling him to enter valid email address



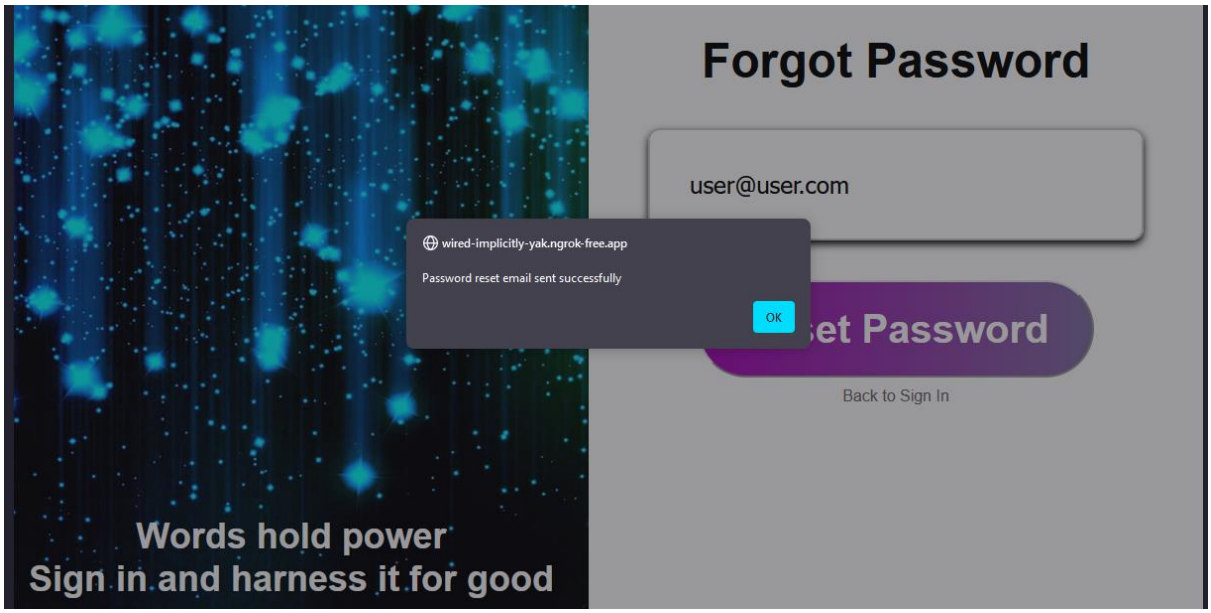
Bad E-mail

3.3: Mail not found! : if the user entered an e-mail address that is not into the system database the system will show a error message to the user



Not Existing E-mail

3.4 : Forget-Password success: this is how the system will response if the user entered the right data

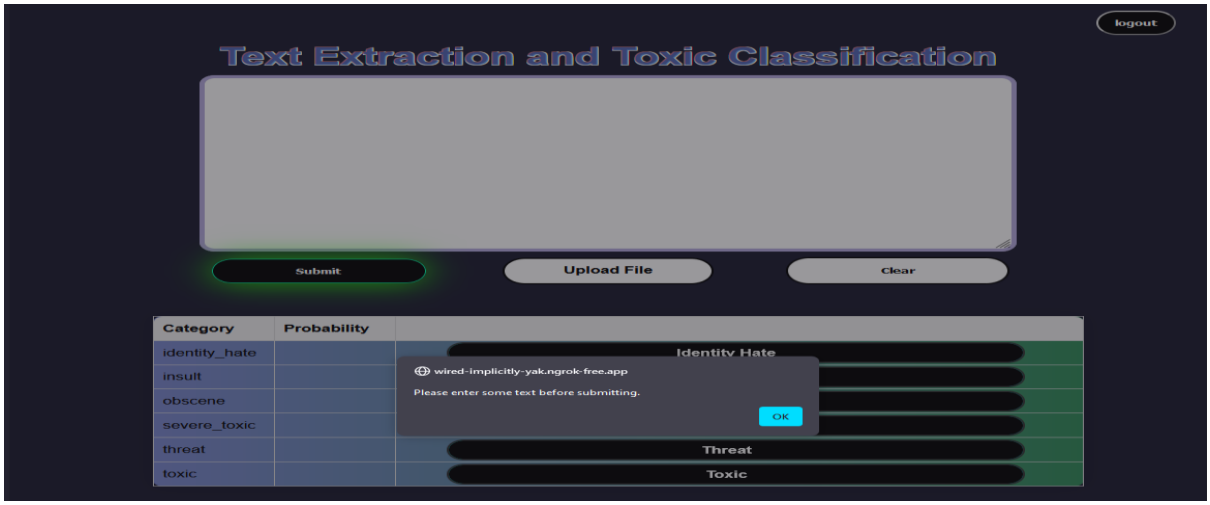


Right Forgot Password

4-Submiting Testing: in this testing phase we will test the errors or the mistakes the user could make trying to submit data into the classifier

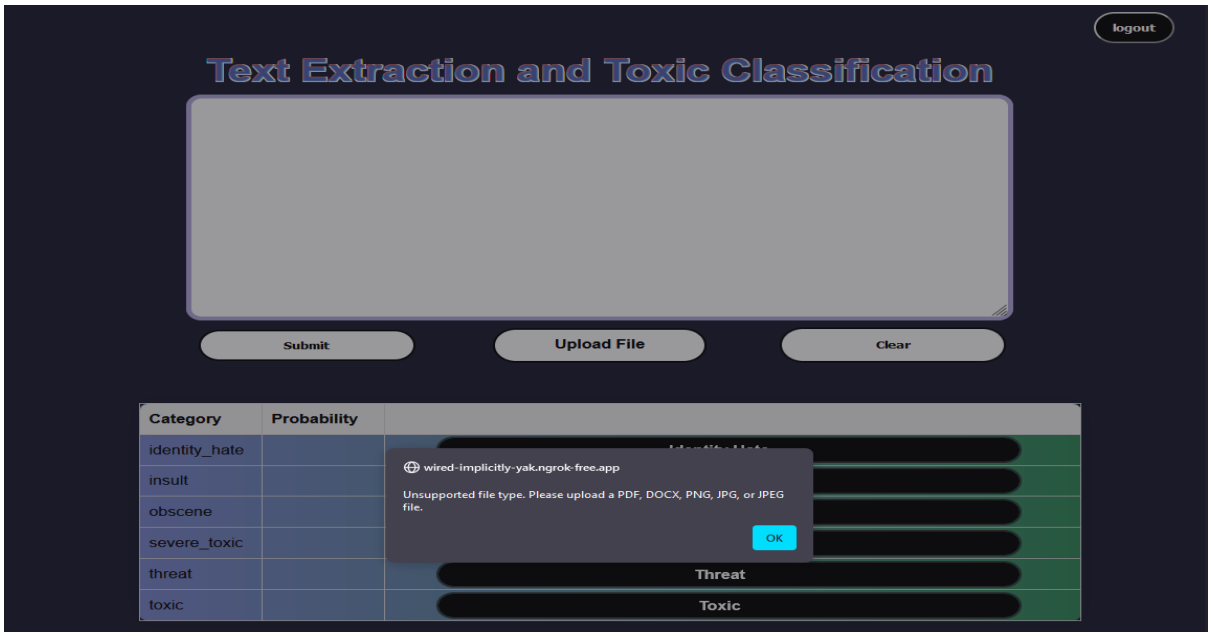
4.1: Empty Submission: when the user submits an empty data into the classifier

The system will show a message to alert the user to enter the data



Empty Text-Box

4.2: Wrong File: when the user uploads unsupported file format as mp4 for example



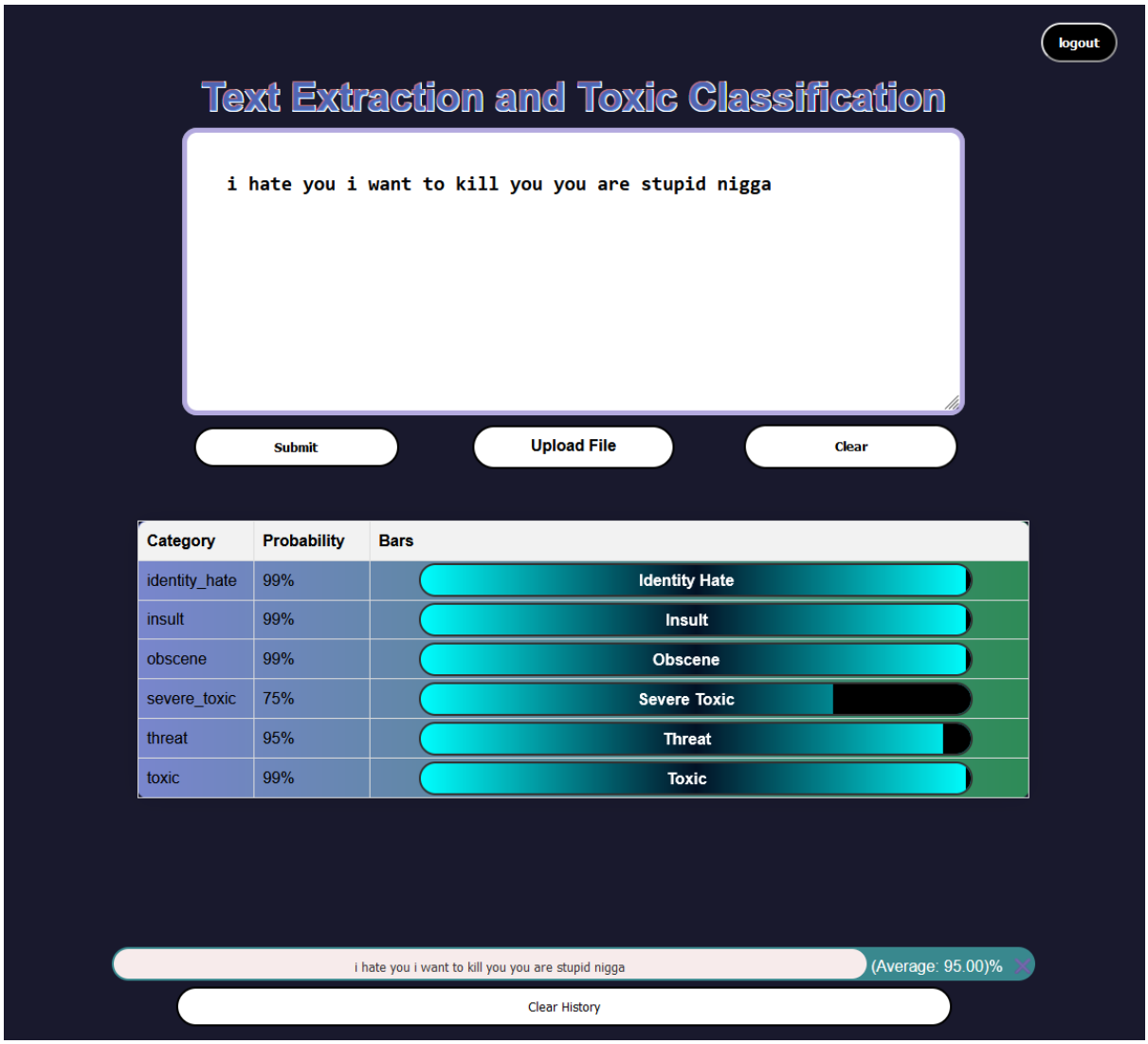
Unsupported file format

4.3 : Uploading & Classifying success: this is how the system will response if the user entered the right data

The extracted text from the file is in the text box

After the user presses the submit button the text will be classified into the table

And the recent classified data will be added to the history alongside with its average



classified result

Summary

This document outlines a series of testing scenarios for different functionalities within the system, including login, password recovery, and data submission to the classifier. It covers various error cases, such as incorrect email formats, empty fields, and unsupported file formats, with corresponding figures illustrating the expected behavior of the system. The testing scenarios demonstrate the system's ability to handle user input effectively, providing clear error messages and feedback where necessary. Additionally, it emphasizes successful login and data submission processes, highlighting the expected outcomes when users enter correct information. It's worth noting that the evaluation of admin functionality is omitted, as administrators are expected to be familiar with the system's operation and are not subject to typical user testing scenarios. Overall, the document serves as a comprehensive guide for testing the system's robustness and user-friendliness across different scenarios.

Chapter 7
Conclusion

Conclusion

Motivated by the pressing need to address toxicity in online interactions, exacerbated by the pervasive reliance on digital communication platforms, the project's primary objective was to create a tool capable of extracting textual content from diverse file types—including PDFs, Word documents, images, and audio files—and subsequently classify the extracted text for toxicity. This endeavor is particularly relevant given the exponential growth of social media, online forums, and other digital communication mediums where toxic behavior can proliferate unchecked.

The project's structured methodology commenced with data preparation, involving the collection and preprocessing of an extensive corpus of text data. This data formed the foundation for training various machine learning models, incorporating advanced natural language processing (NLP) techniques for text classification. The methodology also encompassed rigorous testing and error reduction phases to ensure the model's reliability and accuracy. A critical component of the project was the development of a user-friendly interface, designed to facilitate seamless interaction with the system for non-technical users.

The system architecture was meticulously designed to support the extraction and classification of text from multiple file formats. Key components included a text extraction module, an audio processing module, a classification module, and a user interface. The text extraction module leverages optical character recognition (OCR) for images and document parsing libraries for PDFs and Word documents, while the audio processing module converts speech to text using cutting-edge speech recognition technologies. The classification module employs sophisticated NLP models to categorize text based on toxicity, and the user interface, developed using modern web technologies, provides functionalities for uploading files, viewing extracted text, and displaying classification results. The implementation phase involved integrating these modules into a cohesive system, ensuring seamless interaction between components and utilizing a robust database backend for storing user data and classification history.

The testing phase was crucial, ensuring the system's functionality across various scenarios. Authentication testing verified that users could register, log in, and recover passwords without issues, while data submission testing confirmed that users could upload different file types and receive accurate text extraction and classification results. The classification accuracy was meticulously assessed, determining the performance of the NLP models in correctly identifying toxic content. The testing results were promising, indicating that the system performs reliably under various conditions and accurately classifies text for toxicity. The iterative testing process also identified areas for improvement, which were addressed to enhance system performance.

Significant effort was invested in creating a user-friendly interface that simplifies the complex functionalities of the system. Key features include file upload capabilities, a text area for displaying extracted text, and a results section for showing the toxicity classification results. Additionally, the history management feature stores up to ten recent classifications, with options to clear individual entries or the entire history, and displays average classification scores for each entry. This ensures that users can easily track their interactions with the system and manage their data efficiently.

The admin interface was developed to provide additional functionalities for managing users. Admins can view users, create new users, update existing user information, delete users, and search for users by ID or email. These functionalities facilitate efficient user management and ensure the smooth operation of the system. The design of the admin interface emphasizes ease of use and accessibility, ensuring that administrators can perform their tasks without difficulty.

Looking ahead, the project offers several avenues for future development. Scalability is a key consideration, with plans to expand the system's capacity to handle larger volumes of data and more concurrent users. Enhanced NLP models are also on the horizon, incorporating more sophisticated techniques to improve classification accuracy. Real-time processing capabilities are another area of interest, aiming to develop functionalities for real-time text extraction and classification. Additionally, integrating user feedback to continuously improve model performance and developing a mobile-friendly version of the application are important future directions.

In conclusion, this project has successfully developed a versatile system for text extraction and toxicity classification, addressing a critical need in online communication. The combination of robust data processing, advanced machine learning models, and a user-friendly interface ensures that the system is both effective and accessible. Through meticulous testing and iterative improvements, the project has laid a strong foundation for future enhancements and scalability.

The inclusion of admin functionalities ensures that the system can be effectively managed, maintaining user data integrity and providing control over the classification process. The history feature enhances user experience by allowing easy access to past classifications, further emphasizing the system's usability and efficiency.

Overall, this project represents a significant step forward in combating online toxicity, providing a practical tool for individuals and organizations alike. The insights gained and the technologies developed during this project pave the way for future innovations in the field of text analysis and content moderation. The system's scalability, adaptability, and potential for real-time processing make it a valuable asset in the ongoing effort to foster safer and more respectful online environments.

Chapter 7
References

References

Books

1. **"Speech and Language Processing" by Daniel Jurafsky and James H. Martin**

2. **"Deep Learning" by Ian Goodfellow, Yoshua Bengio, and Aaron Courville**

3. **"Natural Language Processing with Python" by Steven Bird, Ewan Klein, and Edward Loper**

4. **"Machine Learning Yearning" by Andrew Ng**

Journal Articles

1. **"Detection of Hate Speech and Offensive Language on Twitter" by Thomas Davidson et al.**
   - This paper presents methodologies for detecting hate speech and offensive language on social media platforms.
   - Link to the article

2. **"A Survey on Offensive Language Detection using Machine Learning and Deep Learning Techniques" by Federico Bianchi et al.**
   - A comprehensive survey on the current methods for detecting offensive language.
   - Link to the article

3. **"The Challenges of Detecting Hate Speech with Machine Learning" by Shirin Nilizadeh et al.**
   - This article discusses the various challenges faced in the automated detection of hate speech.
   - Link to the article

Technology and Tools

1. **Tesseract OCR Engine**
   - Tesseract is a popular open-source OCR engine that can be used for text extraction.
   - Link to the GitHub repository
2. **SpaCy**
   - SpaCy is an NLP library in Python that is efficient for text processing and extraction.
   - Link to the official website
3. **TensorFlow and PyTorch**
   - These are two of the most widely used deep learning frameworks, suitable for building and deploying models for toxic language detection.
   - TensorFlow
   - PyTorch
4. **NLTK (Natural Language Toolkit)**
   - NLTK is a leading platform for building Python programs to work with human language data.
   - Link to the official website

Online Resources

1. **PLOS ONE Article on Hate Speech Detection**
   - A detailed article on methods for detecting hate speech on social media.
   - Link to the article
2. **ScienceDirect Article on Hate Speech Detection**
   - A systematic review of hate speech detection using NLP.
   - Link to the article
3. **The Decision Lab on Online Toxicity**
   - An article discussing the impacts of online toxicity and methods for mitigation.
   - Link to the article