A Project Work - Phase I on

Automated Word Formatter

Submitted in partial fulfillment of the requirements for the award of the

Bachelor of Technology

in

Department of Computer Science and Engineering
(Artificial Intelligence and Machine Learning)

by

N. Krishna Kalyan	**20241A6638**
Abhinav Aka	**20241A6605**
Raya Samuel Sajit	**20241A6647**

Under the Esteemed guidance of

Dr. M. KiranKumar

Professor

Department of Computer Science and Engineering

(Artificial Intelligence and Machine Learning)

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Approved by AICTE, Autonomous under JNTUH, Hyderabad)

Bachupally, Kukatpally, Hyderabad-500090

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY
(Autonomous)

Hyderabad-500090

CERTIFICATE

This is to certify that the Project Work - Phase I entitled "Automated Word Formatter" is submitted by **N. Krishna Kalyan (20241A6638), Aka Abhinav (20241A6605) and Raya Samuel Sajit (20241A6647)** in partial fulfillment of the award of degree in BACHELOR OF TECHNOLOGY in Computer Science and Engineering (Artificial Intelligence and Machine Learning) during Academic year 2023-2024.

Internal Guide	Head of the Department
Dr. M. Kiran Kumar	Dr. G. Karuna

External Examiner

ACKNOWLEDGEMENT

There are many people who helped us directly and indirectly to complete our Project Work - Phase I successfully. We would like to take this opportunity to thank one and all. First, we would like to express our deep gratitude towards our internal guide **Dr. M. Kiran Kumar, Associate Professor,** Department of Computer Science and Engineering (Artificial Intelligence and Machine Learning), for his support in the completion of our dissertation. We wish to express our sincere thanks to **Dr. G. Karuna,** Head of the Department**,** and to our principal **Dr. J. PRAVEEN,** for providing the facilities to complete the dissertation. We would like to thank Project Coordinator, Dr. R. P. Ram Kumar, our faculty members and friends for their help and constructive criticism during the period. Finally, we are very much indebted to our parents for their moral support and encouragement to achieve goals.

N. Krishna Kalyan (20241A6638)
Aka Abhinav (20241A6605)
Raya Samuel Sajit (20241A6647)

DECLARATION

We hereby declare that the Project Work - Phase I titled **"Automated Word Formatter"** is the work done during the period from **17th July 2023 to 23th November 2023** and is submitted in the partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering (Artificial Intelligence and Machine Learning) from Gokaraju Rangaraju Institute of Engineering and Technology (Autonomous under Jawaharlal Nehru Technology University, Hyderabad). The results embodied in this Project Work - Phase I have not been submitted to any other University or Institution for the award of any degree or diploma.

N. Krishna Kalyan (20241A6638)
Aka Abhinav (20241A6605)
Raya Samuel Sajit (20241A6647)

ABSTRACT

Researches face a lot of problem while documenting their Manuscripts. In absence of proper formatting, the research paper may get rejected, even if it has quality content. The aim of project is to create an application to streamline the process of transforming documents into various recognized academic or professional styles, ensuring consistency and precision. The proposed application is useful for a wide range of usecases, from Research papers to Academic Thesis, which effortlessly manages titles, headings, text, tables, figures, equations and also presents the content in an organized and professional manner. Additionally, it automates the handling of Citations and References, significantly enhancing the document preparation efficiently across various formatting standards. The proposed application is constructed by modifying the nodes of the document tree structure. With this versatile application, valuable time and effort can be saved while ensuring the documents adhere to the highest standards of formatting.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

Acronym	Full Form
E3S	Electronic Entertainment Experience
IEEE	Institute of Electrical and Electronics Engineers
MVT	Model-View-Template

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Automated Word Formatter

In today's academic and professional world, the correct formatting of documents according to research paper standards is crucial for success. Whether you're a student, researcher, or professional, your work often demands adherence to specific guidelines, such as those set by renowned organizations like the Electronic Entertainment Experience (E3S). Achieving impeccable formatting, however, can be a daunting and time-consuming task, leading to frustration and potential errors. To address this challenge, we present the Automated Word Formatter, a web-based platform designed to simplify the process of correctly formatting your documents. At its core, the Automated Word Formatter streamlines the way you format word documents according to research paper standards, such as those outlined by the E3S. By providing users with a seamless and efficient solution, this platform empowers individuals to focus on the substance of their work rather than getting lost in the intricacies of formatting. The Automated Word Formatter offers an intuitive and user-friendly interface, making it accessible to a wide range of users, from students and researchers to professionals across various fields. The key objective is to eliminate the hassle and complexities associated with formatting, ultimately saving time and reducing the risk of errors.

This platform operates on a simple premise: a user uploads a word document, and the Automated Word Formatter takes care of the rest. By leveraging advanced algorithms and pre-defined formatting styles, the system automatically refines the document's layout, ensuring it adheres to the exacting standards expected in the world of research papers. Whether it's managing fonts, margins, headings, or reference lists, the Automated Word Formatter excels in producing professional and standardized documents. The automated formatting tool caters to a variety of requirements and guidelines, with a primary focus on the stringent standards set forth by the E3S. This ensures that your work aligns with the expectations of prestigious institutions and publications. We understand that conformity to these standards is essential not only for readability but also for fostering credibility and trust in your research. Furthermore, the Automated Word Formatter is underpinned by a robust back-end system that securely processes your documents. Your data is treated with the utmost care and confidentiality, allowing you to trust our platform for your sensitive research work. In a nutshell, the Automated Word

Formatter is your ultimate partner in document formatting, simplifying what was once a cumbersome and time-intensive task. It allows you to maintain your focus on the core of your work, all while ensuring that your research paper adheres to the highest standards. Whether you're new to academic writing or a seasoned professional, our platform welcomes you to experience the future of document formatting. Say goodbye to the stress of document formatting; say hello to the Automated Word Formatter, your gateway to impeccable and effortless research paper formatting.

1.2 Objective of the Project

To Implement a python-based word formatting tool which automates manual editing to save time and effort.

The approach used is to classify the elements of the document.

Applying a pre-defined style to the categorized elements through manipulation of the Document Tree Structure.

1.3 Methodology

The objectives are achieved by performing the following steps

Step 1: Saving the Document in the MongoDB Cloud Database

Figure 1.1 Document Upload

In this initial step as shown in fig 1.1, is to upload the document to the MongoDB database after which the document undergoes ingestion and is securely stored within the application's MongoDB cloud database. Renowned for its robust web framework, the application seamlessly integrates with MongoDB, a sophisticated cloud-based database system. This strategic storage choice is particularly beneficial when handling unstructured data types, such as documents. By employing this MongoDB cloud database, the application ensures

seamless accessibility and efficient processing of the document, setting the stage for subsequent steps in the formatting.

Step 2: Data Extraction Using the Python Docx Module

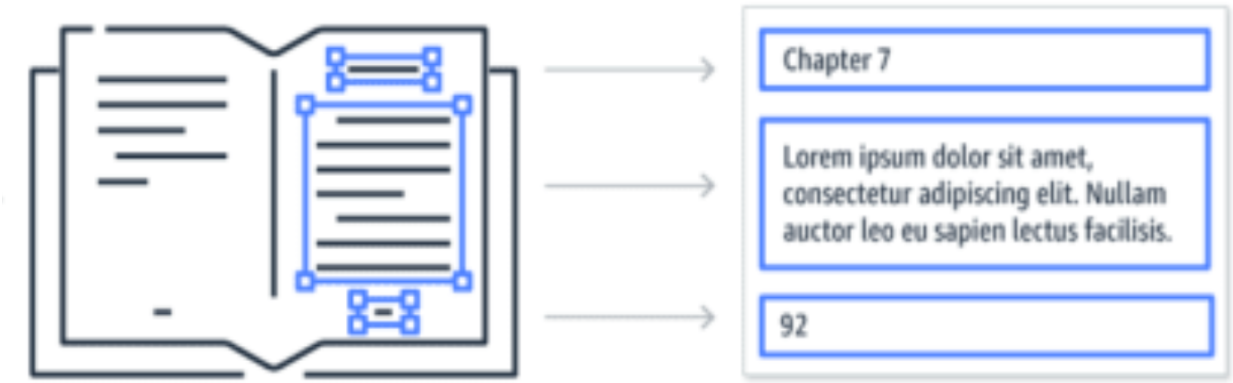

Figure 1.2 Data Extraction

Once securely storing the document in the database, the subsequent phase, illustrated in Figure 1.2, centers around the pivotal process of data extraction. This crucial step is meticulously orchestrated through the adept use of the Python Docx module—an exceptionally versatile library crafted for the precise parsing of Word documents. Revered for its robust functionality, this module takes on the intricate task of unraveling both the textual nuances and structural intricacies woven into the document's fabric. Navigating through the document's complexities, the Python Docx module seamlessly retrieves essential content, laying the groundwork for subsequent stages of sophisticated processing and analysis. This refined methodology not only safeguards the document's integrity but also establishes the framework for deploying advanced techniques and algorithms in the ensuing phases of the document formatting pipeline.

Step 3: Classification of Elements as Headings, Non-Headings, Tables or Images.

Figure 1.3 Classification of Elements

As demonstrated in fig 1.3 once the document's content extracted, the system employs the XGBoost classifier, a machine learning algorithm, to categorize elements within the document. The classifier differentiates between headings (such as titles, section headings, or subsections), non-headings (regular text, paragraphs, etc.), Tables and Images. This classification process is critical for identifying where specific formatting styles should be applied.

Step 4: Applying Predefined Styles Using the Python Regex Module

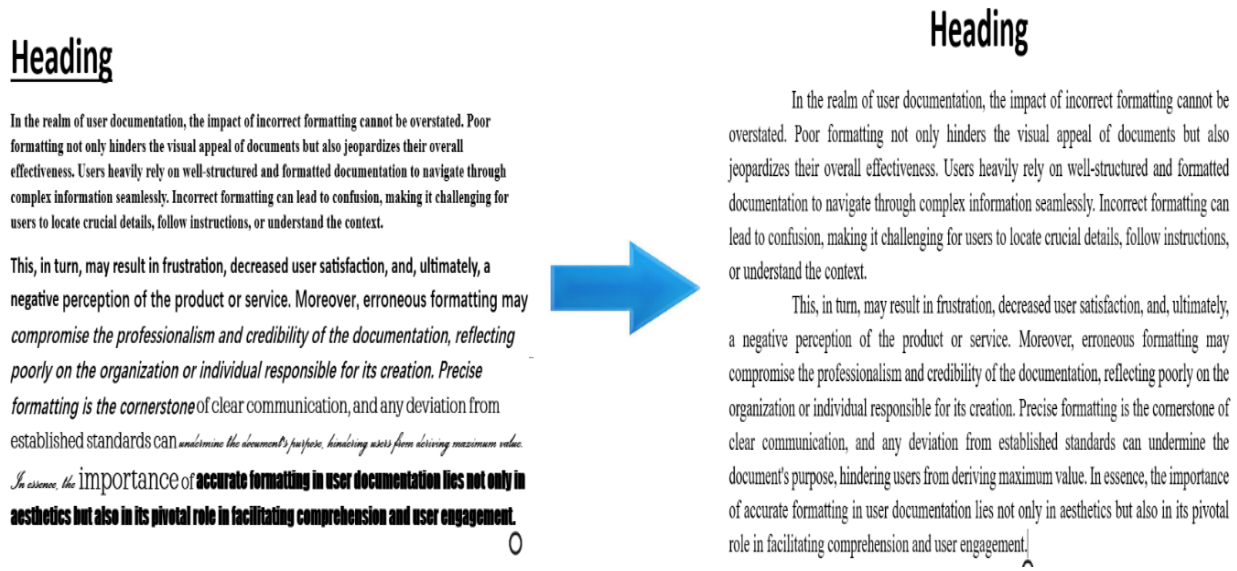

Figure 1.4 style application

In Figure 1.4, following the successful categorization of document elements, the system seamlessly incorporates the Python regex module into the workflow. This module, leveraging regular expressions (regex) within Python, plays a pivotal role in identifying and locating specific patterns embedded within the document content. These discerned patterns correspond to predefined formatting styles crucial for various elements identified during the classification process. The strategic use of regex empowers the system to efficiently navigate through the document, accurately identifying formatting cues essential for adhering to the rigorous standards set forth in research paper guidelines. The flexible and precise nature of regex enhances the system's ability to apply nuanced formatting styles to diverse elements, contributing significantly to the document's overall coherence and professionalism.

Step 5: Adjusting Page Layout Using the Docx Module

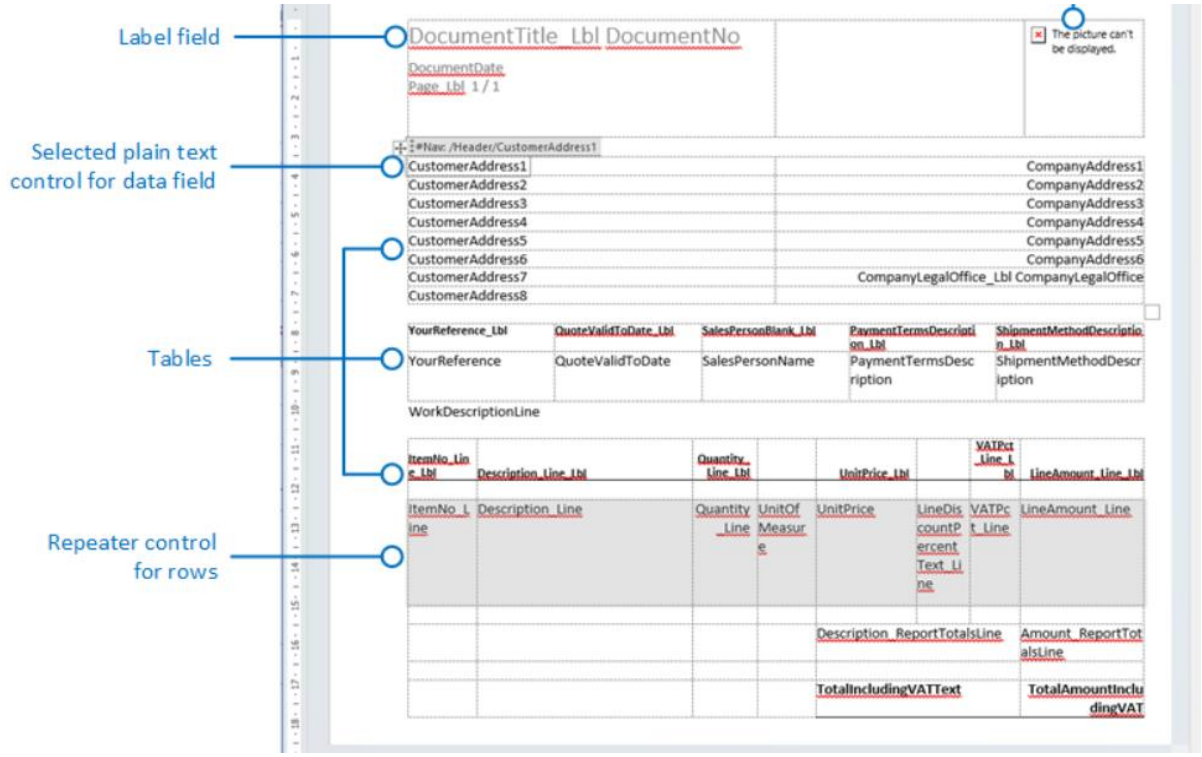

Figure 1.5 Layout Adjustment

As illustrated in fig1.5, the final phase of the methodology involves using the Python Docx module is once again this time for page layout adjustments. Elements such as margins, indentation, and alignment are modified to adhere to the desired research paper formatting standards. This ensures that the entire document, including page layout and content, is in line with the established guidelines.

Step 6: Saving the Formatted Document in a Database and Providing User Access

Once the document has undergone the formatting process, the final step involves storing the formatted version in the application's database, ensuring it is securely archived and easily accessible. This archival serves as a safeguard, preserving the document for future use and reference.

1.4 Architecture diagram

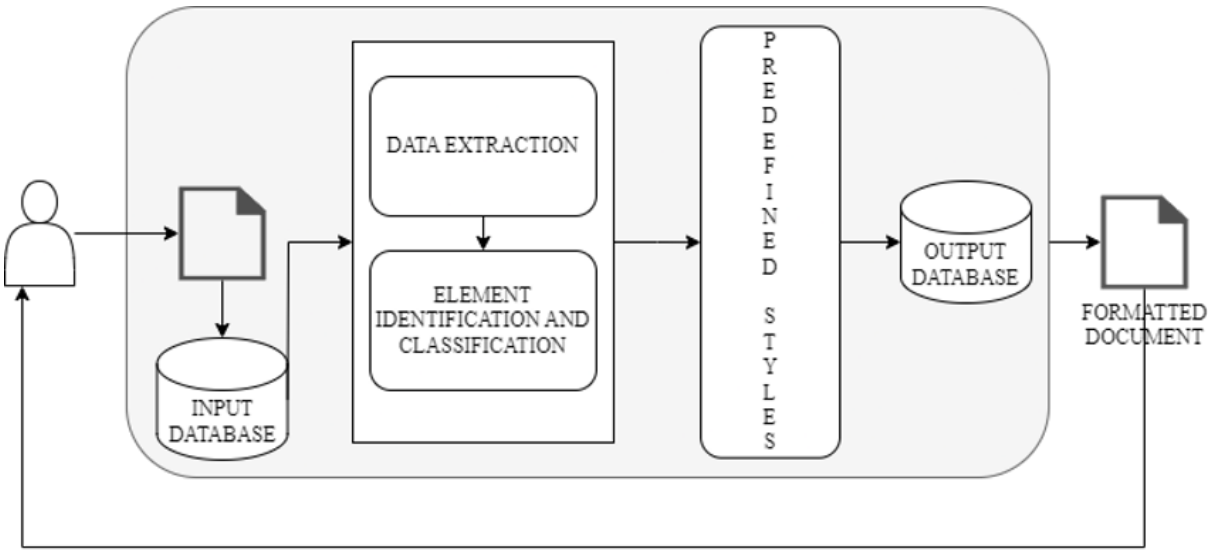

Figure 1.6 Architecture diagram

The structural design of the Automated Word Formatter ensures a fluid interaction between users and the underlying system. Users initiate the process by uploading Microsoft Word documents through the platform's user-friendly interface. These documents are securely stored in a MongoDB database, chosen for its adaptability and scalability, ensuring efficient document storage and retrieval.Once retrieved from MongoDB, the system engages in a detailed data extraction process facilitated by Python's python-docx module. This phase involves parsing and capturing both textual content and structural elements embedded within the document. Following this, an element identification stage takes place to discern between headings and non-headings, a crucial aspect for accurate formatting.

Machine learning becomes instrumental with the application of an XGBoost classifier for element classification. This dynamic approach ensures the differentiation of document components, laying the foundation for implementing predefined formatting styles. Each element is precisely formatted in accordance with established research paper standards.The now correctly formatted document seamlessly integrates back into the MongoDB database. This seamless cycle not only guarantees users access to the formatted document at their convenience but also sets the stage for generating a personalized download link.

The provision of a download link is a pivotal feature, directly connecting users to their formatted documents stored in the MongoDB database. This link acts as a bridge, allowing users to effortlessly retrieve their documents with a simple click. The overall result is an

efficient, user-centric process that aligns with the platform's commitment to delivering accurate and visually appealing document formatting.

1.5 Organization of the Report

This report consists of the overview of what all topics discussed in this entire report in a brief and concise manner with the sequence of topics presented.

Chapter 1: Introduction

This section discusses about the project and the use case of the project and how it is useful to the users and includes about the basic working of the overall project work flow.

Chapter 2: Literature Survey

This section discusses about the existing approaches to solve this problem and their drawbacks and the advantages. This section provides the required knowledge and a momentum to carry out the project.

Chapter 3: Proposed Methods

In this section we discussed about the logical sequence in which the problem is addressed and the methods that are adopted to solve the problem.

Chapter 4: Results and Discussions

This section contains information about the results obtained during the working of this project it contains the landing page an example of unformatted to formatted document and the step by step working results of the project.

Chapter 5: Conclusion and Future Enhancements

This section contains a robust solution for automating document formatting, ensuring precise style updates, and maintaining adherence to the highest standards. Users benefit from increased efficiency and accuracy in formatting, leading to time savings and elevated document quality. Additionally, it explores future aspects, providing a foundation for continual improvement and adaptation to evolving user needs.

CHAPTER 2

LITERATURE SURVEY

2.1 Existing Approaches

Nail Nasyrov and his team have introduced an innovative approach [1] for document formatting verification. In their work, they employ a decision tree algorithm, leveraging the efficiency of CatBoost. This algorithm meticulously classifies each element within documents as either syntactically correct or incorrect. The classification is based on a multitude of attributes, including but not limited to font size, alignment, line spacing, and more. The approach offers a systematic way to enhance the accuracy of document formatting v [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13]erification, showcasing the team's commitment to advancing the field.

[2] Dr. Venkatesan, along with a collaborative team, introduces a template matching technique designed for comparing the layouts of two documents. This innovative method efficiently identifies differences between documents and subsequently adjusts them to align with the original document. The authors leverage the Python docx module for document manipulation, employ the difflib library to compare documents and pinpoint differences, and use regex to identify and handle specific patterns. This collaborative effort, led by Dr. Venkatesan V, showcases a systematic approach to document layout comparison and adjustment, highlighting the team's collective expertise in document analysis and manipulation.

In his work [3], Kirill Chuvilin introduces an innovative approach to LaTeX document processing, leveraging parse trees to represent document structures. The methodology involves training a machine learning algorithm on a carefully curated dataset of LaTeX documents, manually corrected by experts to include various errors such as typos, grammatical issues, and formatting errors. The algorithm, informed by the parse tree representation, adeptly identifies and rectifies these errors, offering a systematic and machine learning-driven strategy to enhance the precision and quality of LaTeX documents. This work contributes significantly to the field of document processing and error correction within the LaTeX framework.

In [4], Oliveira presents an approach employing two algorithms to automate document page layout. The first algorithm operates under the assumption that previously defined rectangular items are to be placed freely on the document. It calculates the minimum

bounding box for each item, facilitating their placement. The second algorithm addresses the placement of free-form items on pages divided into columns. This is achieved by determining the optimal number of columns for the page. Oliveira's approach offers a systematic and algorithmic solution to automate document page layout, contributing to advancements in document design and formatting.

Nathan Hurst presents a comprehensive survey in [5], exploring various existing approaches for automating page layout. The survey delves into different layout modes, including Coordinate, Flow, Grid, VH-Box, Guillotine, and Box, each of which governs how elements are positioned within documents. Additionally, the survey discusses approaches like Column-driven layout, Cell-driven layout, and Minimal configurations, which play crucial roles in selecting appropriate page layouts to tackle issues related to both micro and macro typography. Hurst's work provides valuable insights into the diverse methods available for automating page layout, contributing to the broader understanding of document design and formatting.

Nasser introduces a novel approach in [6] leveraging the Django framework, this method involves submitting a document as input. The document's contents are stored in a MySQL database and later processed to generate a new document with accurate formatting. Nasser's work stands out for its innovative use of web-based automation, incorporating Django and database management to streamline the word formatting process effectively.

Gange delves into the optimization of table layouts in [7], offering three distinctive approaches for determining the optimal minimum height layout. These methodologies include the AI-based A* algorithm, Constraint Programming Approach 1, and Constraint Programming Approach 2 with Lazy Clause Generation. The choice of the most effective method is contingent upon the unique characteristics of the tables in question, culminating in the adoption of a hybrid CP/SAT approach. Gange's contribution provides valuable insights and a diverse set of methodologies for enhancing table formatting.

Bilauca explores distinct approaches in [8] employing mixed-integer programming (MIP) and constraint programming (CP). The MIP model, while a more general and potent approach, may pose computational challenges, especially for large tables. In contrast, the CP model, while less general, often proves more efficient in solving real-world table layout problems. Through comprehensive evaluations on various real-world tasks, the authors demonstrate the system's capability to generate output comparable to human-generated results. Bilauca's work underscores the effectiveness of combining MIP and CP approaches for achieving optimal table layouts.

In [9], John Bateman provides a comprehensive exploration of Rhetorical Structure Theory (RST), a framework designed to elucidate the organization of text and the relationships existing between its constituent parts. RST operates on the fundamental premise that texts comprise two essential units: elementary discourse units (EDUs) and rhetorical relations. EDUs, representing the smallest analyzable units of text within the RST framework, serve as the building blocks for understanding textual structure. Concurrently, the theory focuses on delineating rhetorical relations, which encapsulate the intricate relationships that bind these EDUs. By elucidating the nuanced interplay between text elements, Bateman's work significantly contributes to our understanding of how textual content is organized and interconnected within the framework of Rhetorical Structure Theory.

In [10], Bolshakov introduces an inventive approach that employs a mathematical cohesion comparator function to evaluate each line within a document. The ensuing cohesion scores are systematically compared, and paragraph segmentation is determined based on a precomputed threshold. The decision to split paragraphs hinges on assessing the disparity in cohesion scores between two lines against the established threshold. This threshold is meticulously derived through an in-depth analysis of a database comprising collocations and semantic rules. Bolshakov's methodology, grounded in mathematical cohesion analysis and semantic rules, offers an analytical and data-driven approach to enhance paragraph segmentation, contributing to the evolution of document organization and readability.

TableNet, developed by Shubham Singh [11], introduces an innovative deep learning model tailored for comprehensive table detection and structured data extraction from scanned document images. Utilizing a two-stream encoder-decoder architecture, the model's encoder extracts features from input images, while the decoder generates two vital output masks—one for the table region and another for individual columns within the table. Addressing the challenges posed by unstructured document images containing tables, TableNet goes beyond traditional table detection. It intricately tackles the task of extracting information from rows and columns within the detected tables. The model's holistic approach, emphasizing both accurate table identification and nuanced structure recognition, responds to the increasing demand for efficient information extraction from diverse documents captured through mobile phones and scanners.

In [12], Widiastuti addresses the formidable task of named entity recognition and classification in historical documents, a challenge amplified by document variety, quality variations, and the scarcity of labeled data. Both rule-based and machine learning approaches have been proposed to tackle these challenges, with recent strides in deep learning and domain-

specific systems offering promising avenues for advancement. The digitization of historical documents has ushered in a new era of accessibility, but it also presents challenges in efficiently mining information from this vast repository. In the abstract, Widiastuti underscores the significance of developing technologies to search, retrieve, and explore information within this "big data of the past." The focus on named entity recognition (NER) systems in the survey aligns with the demands of humanities scholars, highlighting the challenges posed by diverse, historical, and noisy inputs. The survey not only inventories existing resources and outlines past approaches but also outlines key priorities for future developments in this crucial area of historical document analysis.

In [13], Stefan Schweter introduces FLERT, a pioneering method that enriches Named Entity Recognition (NER) models by incorporating document-level features through a Convolutional Neural Network (CNN). FLERT systematically extracts two types of document-level features: global features, encapsulating the overall characteristics of the document, and local features, capturing relationships between named entities within the document. These features are seamlessly integrated into a standard NER model, contributing to enhanced performance. In the broader context of NER approaches traditionally focusing on sentence-level information, FLERT distinguishes itself by leveraging transformer-based models to naturally capture document-level features. The paper conducts a comparative evaluation within the standard NER architectures of "fine-tuning" and "feature-based LSTM-CRF," exploring different hyperparameters for document-level features. Valuable insights from experiments lead to recommendations on effectively modeling document context. The approach is integrated into the Flair framework for reproducibility, presenting new state-of-the-art scores on several CoNLL-03 benchmark datasets and reaffirming the effectiveness of FLERT in advancing document-level feature incorporation for improved NER performance.

In [14], Ming Zhou and their team introduce LayoutLM, a pioneering pre-training framework for Document Image Understanding (DIU) that adeptly predicts both text and layout information in scanned document images. Through extensive pre-training on a large-scale dataset encompassing various tasks such as text recognition, layout analysis, and mask prediction, LayoutLM achieves a holistic understanding of scanned documents. Unlike previous NLP-focused pre-training models that predominantly emphasize text-level manipulation, LayoutLM uniquely incorporates layout and style information crucial for document image understanding. This innovative approach allows LayoutLM to excel in real-world tasks, including information extraction from scanned documents. Leveraging image features to integrate visual information into the model, LayoutLM represents a breakthrough

as the first framework to jointly learn text and layout in a unified manner for document-level pre-training. The versatility of LayoutLM is demonstrated through state-of-the-art results in diverse downstream tasks, such as form understanding, receipt understanding, and document image classification. The code and pre-trained models are made publicly accessible, enhancing accessibility and fostering further advancements in the field.

In [15], Widiastuti introduces a modular and scalable Document Image Extraction System designed to transform document images into text, enabling efficient storage, management, and information retrieval. Comprising components such as document image acquisition, pre-processing, text extraction, feature extraction, classification, and extraction, the system employs a modular design and diverse text extraction methods to address the challenges inherent in document image extraction. Through meticulous evaluation, the system demonstrates high accuracy in extracting text from a range of document types. The accompanying abstract outlines the system's overarching design, emphasizing its applicability to specific cases like decision letters, certificates, and assignments. It details the sequential flow of processes, from data input and scanning to character classification, normalization, and extraction. Notably, the system adapts to the uniqueness of each document, with a focused emphasis on character recognition tailored to individual document features. The abstract underscores the need for additional features to select the document type for extraction, acknowledging the importance of a nuanced approach based on each document's distinct characteristics.

Table 2.1 Representation of Summary of Existing Approaches

Ref. No	Methodology	Drawbacks	Advantages
[1]	It is a classification technique which uses a decision tree algorithm to classify elements of document there by applying the predefined style to the document.	It only checks the errors in documents, but it doesn’t edit the document automatically.	Utilizes machine learning algorithms, specifically gradient boosting on decision trees improving performance.
[2]	This approach is based on the usage of docx, regex and difflib library's present in	Lack of template customization options and does not predict	Provides user friendly experience and reducing the risk

	python which are used to classify every element as either correct or wrong.	mathematical equations.	of document rejection and improving overall document quality.
[3]	This method uses groups with a tree pattern and group rules to identify potential errors.	Can wrongly identify a correct statement to avoid such errors a large number of computational resources are required	It uses Zhang-Shasha algorithm to Correct a wide range of errors including syntax errors, style errors, and formatting errors.
[4]	The first approach uses a bounding box method. The second approach is a places item in free form	The algorithms are highly dependent on the input quality of the document.The approach doesn't provide good results on tables and image data.	Recursively divides the document into pages. Places the document items on the pages according to a set of heuristics there by pruning entire traversal
[5]	This survey explains about techniques such as Column-driven layout, Cell-driven layout selection methods	It does not use any machine learning approach to automate document formatting it uses the tree structure of the document to edit formatting	It provides accurate formatting as this approach modifies the tree structure of the document.
[6]	Implemented using Django framework and MySQL Database.	It only provides template for a specific student template.	Interactive web interface making it convenient for users.
[7]	It uses A* algorithm approach Constraint Programming Approach 1 Constraint Programming Approach 2 with Lazy Clause Generation	Drawback of the paper is that it does not address the problem of nested tables. Nested	This heuristic used in this approach is based on the area of the cell content and is more

		tables are tables that are embedded inside other tables which can be used to create complex layouts	accurate than the previously used heuristics.
[8]	It used two different approaches: MIP (Mixed Integer Programming) and CP (Constraint Programming)	Automatic table layout algorithms can sometimes generate tables that are not accurate or consistent. This is especially true for tables with complex layouts or tables that contain a lot of data.	It generates tables in a variety of different formats, such as HTML, PDF, and CSV.
[9]	It uses NLP techniques such as chunking parts of speech tagging and semantic role labelling and statistical model Hidden Markov Models (HMM's)	The system is not able to generate diagrams that are interactive and dynamic.	Generates a variety of different types of output including news articles, scientific papers, and technical documentation.
[10]	It is achieved in following steps: Quantitative evaluation of text cohesion, Smoothing and Normalizing the cohesion function, Splitting text into paragraphs.	Precision value is low as compared to other experts.	The cohesion function is constructed basing on close co-occurring at words pairs contained in a large database collection.
[11]	TableNet is a two-stream encoder-decoder architecture that extracts features from the input image and generates two output masks, one for the table region and one for the column region.	TableNet may not be able to accurately detect and extract tabular data from complex tables, such as tables with nested	TableNet is a deep learning model, which means that it is able to learn complex patterns from the data. This makes it more robust

		headers, merged cells, or irregular structures.	to noise and variations in the appearance of tables in scanned document images.
[12]	NERC in historical documents uses both rule-based and machine learning approaches, with recent advances in deep learning and domain-specific systems.	NERC models require a large amount of labelled data to train effectively. This data can be difficult and expensive to collect, especially for historical documents.	NERC models can be trained to achieve high accuracy in identifying and classifying named entities in historical documents. This is important for many applications, such as digital humanities research and historical archiving.
[13]	FLERT: Document-Level Features for Named Entity Recognition, FLERT allows NER models to capture document-level information that is not available to traditional NER models.	FLERT requires a large amount of labelled data to train effectively. This data can be difficult and expensive to collect, especially for historical documents and low-resource languages.	FLERT has been shown to improve the accuracy of NER models on a variety of datasets. This is because FLERT allows NER models to capture document-level information that is not available to traditional NER models.
[14]	LayoutLM is pre-trained on a large-scale dataset of scanned document image. Once the model is pre-trained, it can be fine-tuned on a variety of DIU tasks	If the pre-training data is biased, the model will be biased as well. This can lead to errors when the model is	It is pre-trained on a large-scale dataset of scanned document images.

		used on real-world data.	
[15]	It Contains 6 components: Document image acquisition, Pre-processing, Text extraction, Feature extraction, Extraction	The system requires a large amount of training data to be trained effectively. This data can be difficult and expensive to collect.	The system is able to extract text from a variety of document types with high accuracy. This is important for applications where the extracted information needs to be reliable.

CHAPTER-3

Proposed Method

3.1 Problem Statement & Objectives of the Project

Problem Statement: Transforming Document Formatting

In the fast-paced world of academia and research, scholars often grapple with the intricacies of document formatting. Whether it's adhering to specific style guidelines, ensuring uniformity across sections, or grappling with the nuances of various formatting styles, the process can be time-consuming and error-prone.

The existing tools and platforms fall short in providing a seamless solution to this persistent challenge. Manual formatting consumes valuable time that could be better utilized for research and content creation. Inconsistencies in formatting may lead to a lack of professionalism in academic and research documents, affecting the overall quality and impact of scholarly work.

The need for an efficient, automated document formatting tool is evident. This tool must not only align with the rigorous standards of academic formatting but also be intuitive, user-friendly, and adaptable to diverse formatting requirements. Addressing this need will empower researchers, students, and academics to focus more on the substance of their work rather than getting bogged down by formatting complexities.

The challenge lies in creating a comprehensive solution that not only automates formatting tasks but also anticipates user needs, offers flexibility, and evolves with the dynamic landscape of document standards. The aim is to bridge the gap, making document formatting a seamless and efficient aspect of the scholarly writing process.

Objective of the Project

To Implement a python-based word formatting tool which automates manual editing to save time and effort.

The approach used is to classify the elements of the document.

Applying a pre-defined style to the categorized elements through manipulation of the Document Tree Structure.

3.2 Architecture Diagram

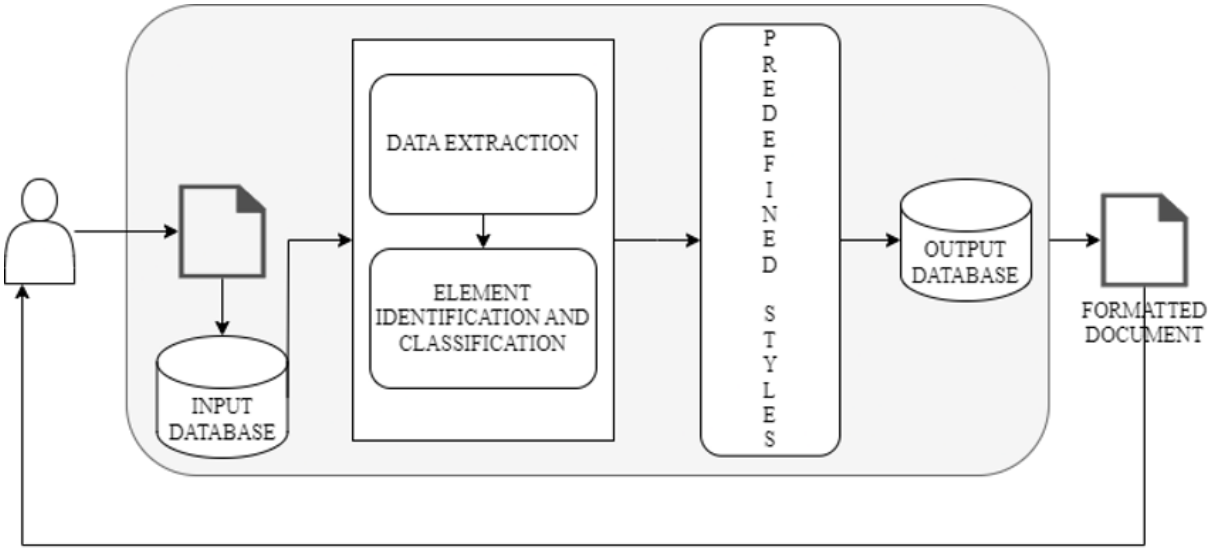

Figure 3.1 Architecture Diagram

As illustrate in fig 3.1, the architectural design of the Automated Word Formatter organises a smooth interaction between users and the underlying system. Users initiate the process by uploading Microsoft Word documents through the platform's intuitive interface. These documents find their repository in a NoSQL MongoDB database, as a document is unstructured type of data making it a perfect choice to select a NoSQL database, additionally its adaptability and scalability, ensuring efficient document storage and retrieval are the key features provided by MongoDB. Once the document is retrieved from MongoDB, the system engages in a data extraction process facilitated by the python-docx module. This step involves parsing and capturing both textual content and structural elements embedded within the document. Subsequently, an element identification phase takes place to discern between headings and non-headings, a pivotal aspect for accurate formatting.

Following the data is extracted Machine learning comes into play with the application of an XGBoost classifier for element classification which involves classifying the elements as either headers, non-headers, tables or images. This dynamic approach ensures the differentiation of document components, paving the way for the implementation of predefined formatting styles. Each element is formatted with precision in accordance with established research paper standards. Then each element is now analysed using the regex module to capture its style once these style attributes are captured, the system automatically updates the style of each element accordingly. This ensures a precise adjustment of the document's appearance, aligning it with the predefined formatting standards. The use of regex adds a sophisticated layer

to the formatting process, making it adaptive to diverse document styles and enhancing the overall accuracy of the project.

The correctly formatted document is then seamlessly reintegrated into the MongoDB database. This harmonious cycle not only ensures the user's ability to access the formatted document at their convenience but also sets the stage for the generation of a personalized download link.The provision of a download link is a key feature, directly connecting users to their formatted documents stored in the MongoDB database. This link acts as a bridge, enabling users to effortlessly retrieve their documents with a simple click. The end result is an efficient, user-centric process that aligns with the platform's commitment to delivering correct and aesthetically pleasing document formatting.

3.3 Software and Hardware Requirements

These generalized software and hardware requirements need to satisfied in order to build this project.

Frontend Development	**Software:** Text Editor (e.g., Visual Studio Code, Atom, Sublime Text) and a Web Browser (e.g., Google Chrome, Mozilla Firefox) **Hardware:** Computer with a minimum of 4GB RAM and High-resolution display.
Backend Development	**Software:** Python: Install the latest version of Python >=3.8.0, Install the Django web framework. **Hardware:** Computer with a minimum of 8GB RAM and Adequate free storage >= 10gb to store project documents and database.
Database	**Software:** MongoDB Compass and MongoDB Atlas account to access the cloud-based MongoDB. **Hardware:** free storage >= 1gb to store project documents that are fetched from the database.

3.4 Modules and its Description

3.4.1 User Module

Registration and Authentication:

User Registration: This involves providing a mechanism for users to create accounts within the system. During registration, users typically input essential information such as their username, email address, and password.

Authentication: Once registered, users need a secure way to log into their accounts. Authentication ensures that only authorized users can access the system. This process often involves verifying a user's identity through credentials (e.g., username and password) and employing secure authentication protocols to prevent unauthorized

User Profile Management

Profile Updates: Users should have the ability to manage and update their profiles. This includes modifying personal information like names, email addresses, and passwords. A user-friendly interface is essential to make this process straightforward.

Security Measures: Robust security measures, such as password encryption and multi-factor authentication, are implemented to safeguard user information.

File Upload and Download

File Upload: This feature allows users to upload Word documents or other specified file types to the system. Proper validation and security checks are necessary to prevent malicious uploads and ensure the integrity of the uploaded files.

Download Functionality: Users is able to download formatted documents or files from the system. Access controls and permissions should be in place to regulate which users can download specific content.

3.4.2 Input and Processing Module

File Upload Handling

Receiving Files: This involves creating a mechanism to receive uploaded Word documents from users. The system should have a secure and reliable file upload component that can handle various file sizes and formats.

Validation: Before processing the uploaded files, it is crucial to implement validation checks to ensure that the files are of the expected type (Word documents, in this case) and do not contain any malicious content. This helps in maintaining the integrity of the system and preventing potential security vulnerabilities.

Storing in Database: Once the file is validated it is stored in the remote cloud-based MongoDB database a NoSQL database which supports un-structed data like the documents.

Data Extraction

Text Extraction: Once a Word document is uploaded, the system needs to extract relevant data from it. This includes extracting the text content, which may involve parsing paragraphs, headings, and other textual elements within the document.

Document Structure: In addition to text, extracting the document structure is important. This could involve identifying sections, headers, footers, tables, and other structural elements within the Word document. Understanding the document's structure is essential for further processing and analysis.

Data Preprocessing

Cleaning Data: Extracted data may contain unwanted elements such as extra spaces, formatting artifacts, or non-textual characters. Data preprocessing involves cleaning and sanitizing the extracted information to ensure that it is accurate and consistent.

Formatting: Ensuring data consistency by standardizing formats is vital. This may involve converting extracted data into a common format or structure that is compatible with the system's requirements.

Quality Assurance: Implementing quality checks during data preprocessing helps identify and handle any anomalies or errors in the extracted data. This ensures that the processed data is reliable and suitable for downstream applications.

3.4.3 Predefined Style Application Module

Layout Adjustment:

Compliance with Standards: One of the primary functions of this module is to ensure that the layout of the document adheres to predefined standards, especially in the context of research

papers. This involves adjusting elements such as margins, spacing, indentation, and page formatting to meet the specified requirements.

Template Application: In cases where a specific document template or layout is required, the module may apply or adjust the document structure to conform to the predefined template. This ensures consistency in the visual presentation of documents within a given context.

Style Consistency:

Textual Styles: Applying consistent styles to textual elements throughout the document is essential. This includes aspects such as font type, size, and color. The module may use regular expressions or other pattern-matching techniques to identify and modify text based on predefined style rules.

Headings and Subheadings: Ensuring consistent styling for headings and subheadings is crucial for document readability and professional presentation. The module can automatically detect and apply styles to these structural elements based on predefined criteria.

Citations and References: In academic or research documents, the module may handle the consistent application of citation styles (e.g., APA, MLA) and formatting of references. This ensures that citations and reference lists follow the prescribed rules and guidelines.

Dynamic Styling Rules:

Configurability: The module may allow for configurability to accommodate different style requirements based on user preferences or specific document types. This flexibility enables users to customize the application of styles to suit their needs.

Rule-Based Styling: Implementing a rule-based system allows the module to apply styles dynamically based on a set of predefined rules. This could involve recognizing certain patterns, keywords, or document structures and applying styles accordingly.

Version Control:

Document Versioning: If document version control is part of the system, the module may play a role in maintaining consistency across different versions of a document. This ensures that style changes or updates are applied uniformly throughout the document history.

3.4.4 Output Module

Formatted Document Generation:

Content Integration: The module integrates with other components, such as the Input and Processing Module and the Predefined Style Application Module, to gather processed and styled content. This may include text, images, and other media elements.

Document Assembly: Formatted document generation involves assembling the processed content into a cohesive and well-structured document. This includes arranging sections, applying styles, and ensuring that all elements are properly formatted according to the specified standards or styles.

Dynamic Content: In cases where the content is dynamic or user-specific, the module dynamically generates documents tailored to individual preferences or parameters.

File Download Handling:

User Interaction: The module provides a user-friendly mechanism for users to download the generated documents. This could involve the presentation of a download button or link within the user interface.

File Format Considerations: Depending on the system requirements, the Output Module may support various file formats. In this case, it specifically generates and handles Word documents. The module ensures that the generated document is in the appropriate format, compatible with common word processing software.

Download Permissions: Implementing download permissions and access controls is essential to regulate which users can download specific documents. This helps maintain security and confidentiality, especially in environments where document access needs to be restricted.

3.5 Requirements Engineering

3.5.1 Functional

File Upload: Users should be able to easily upload Word documents through the web interface.

Automatic Formatting: The system must automatically format the uploaded documents according to research paper standards.

Download Options: The application should offer users the option to download the formatted document in different formats (e.g., Word, PDF).

3.5.2 Non-Functional

Security and Privacy: Users' uploaded documents must be handled securely to maintain data privacy and prevent unauthorized access.

Cross-Browser Compatibility: The web application should work seamlessly across various web browsers (Chrome, Firefox, Safari, etc.) to accommodate user preferences.

Performance: Users expect quick and efficient processing of their documents without significant delays.

Clear Error Messages: In case of errors, users should receive clear and user-friendly error messages to understand and address issues.

3.6 Analysis and Design through UML

This analysis contains Class Diagram, Sequence Diagram, Use case diagram and the activity diagram of Automated Word Formatter.

3.6.1 Class Diagram

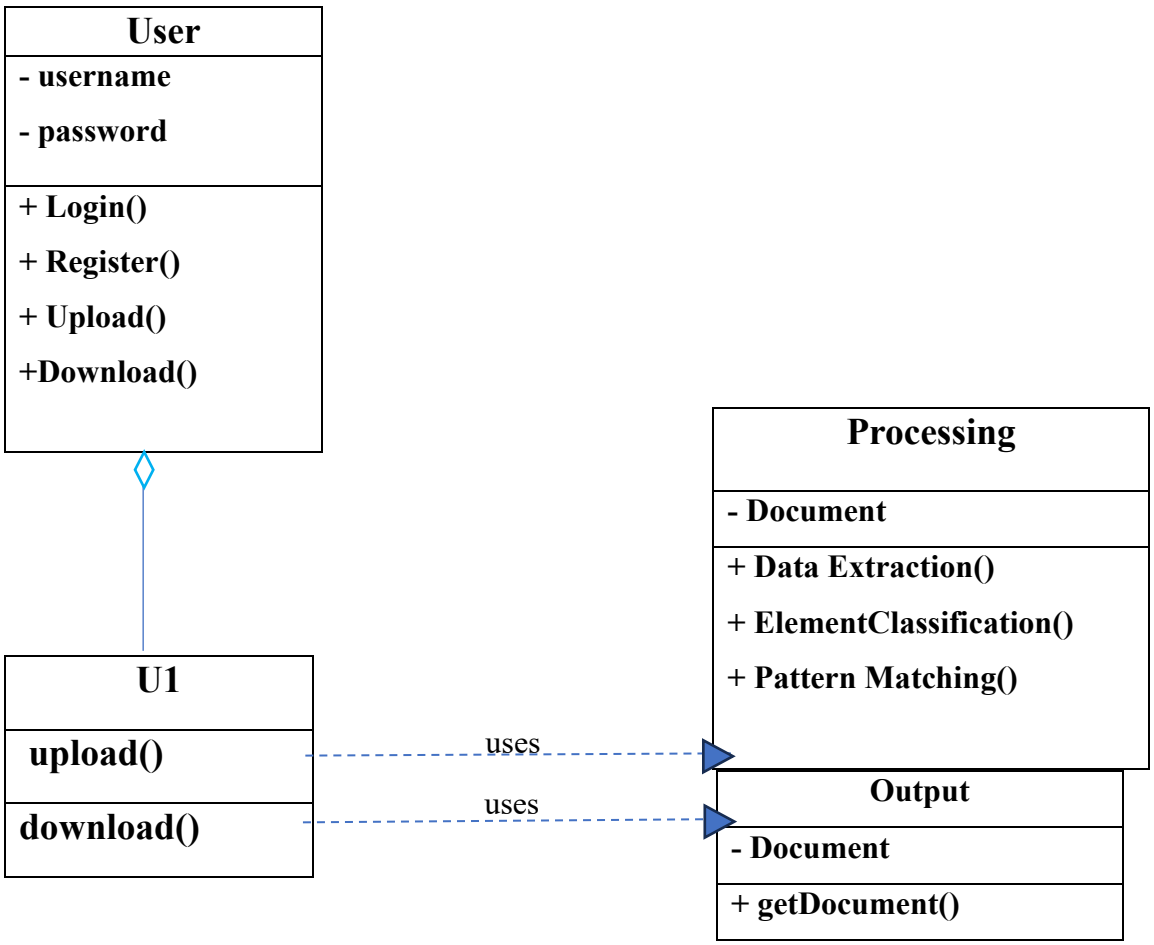

Figure 3.2 Class Diagram of Automatic word formatter

Illustrated in fig 3.2, The presented class diagram outlines three primary classes: User, Processing, and Output. In this depiction, u1 is an object instantiated from the User class, which utilizes both the Processing and Output modules. The User class encapsulates private variables for username and password, along with methods for login, registration, document upload, and download. On the Processing side, there is a private document variable, and the class hosts methods like Data Extraction, Elements Classification, and Pattern Matching, executed in a sequential manner. The Output class facilitates the user in downloading the final processed document. This design encapsulates a streamlined process where users interact by uploading documents, triggering a sequence of processing steps, and culminating in the availability of the refined document for download.

3.6.2 Sequence Diagram

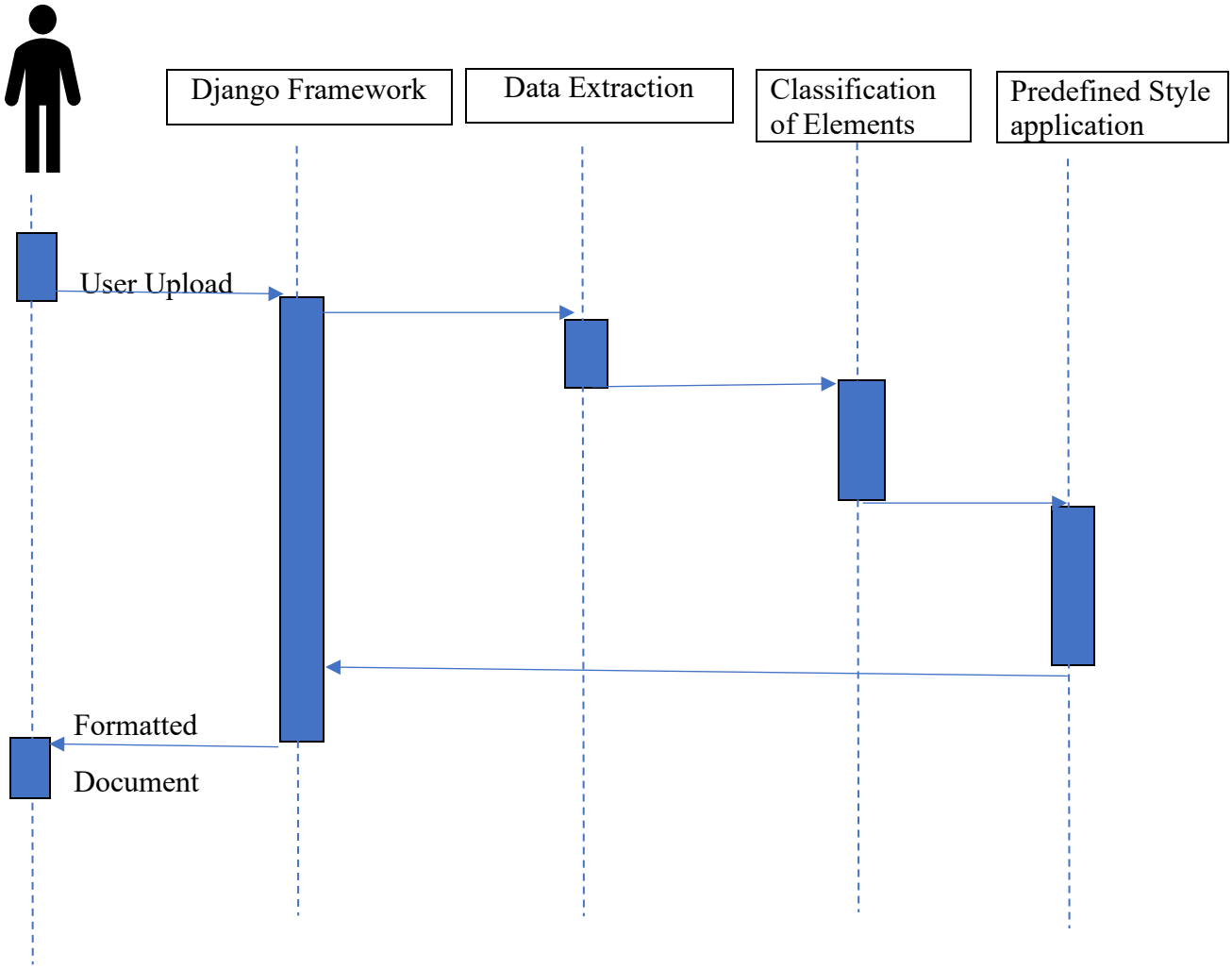

Figure 3.3 Sequence Diagram of Automatic word formatter

Illustrated in fig 3.3, The sequence begins with the "User (upload)" action, where a document is securely uploaded through Django Framework into the MongoDB database. Following this, the "Data Extraction" step utilizes the Python Docx module to extract textual and structural components from the uploaded document. Subsequently, the Classification of Elements employs a machine learning algorithm to categorize elements, setting the stage for the Predefined Style Application. Here, Python's regex module applies formatting styles based on predefined patterns there by improves the overall style and formatting of the document it also sets the layout of the document. The sequence concludes with User (download), providing the user with a professionally formatted document, completing a streamlined and efficient process.

3.6.3 Use case diagram

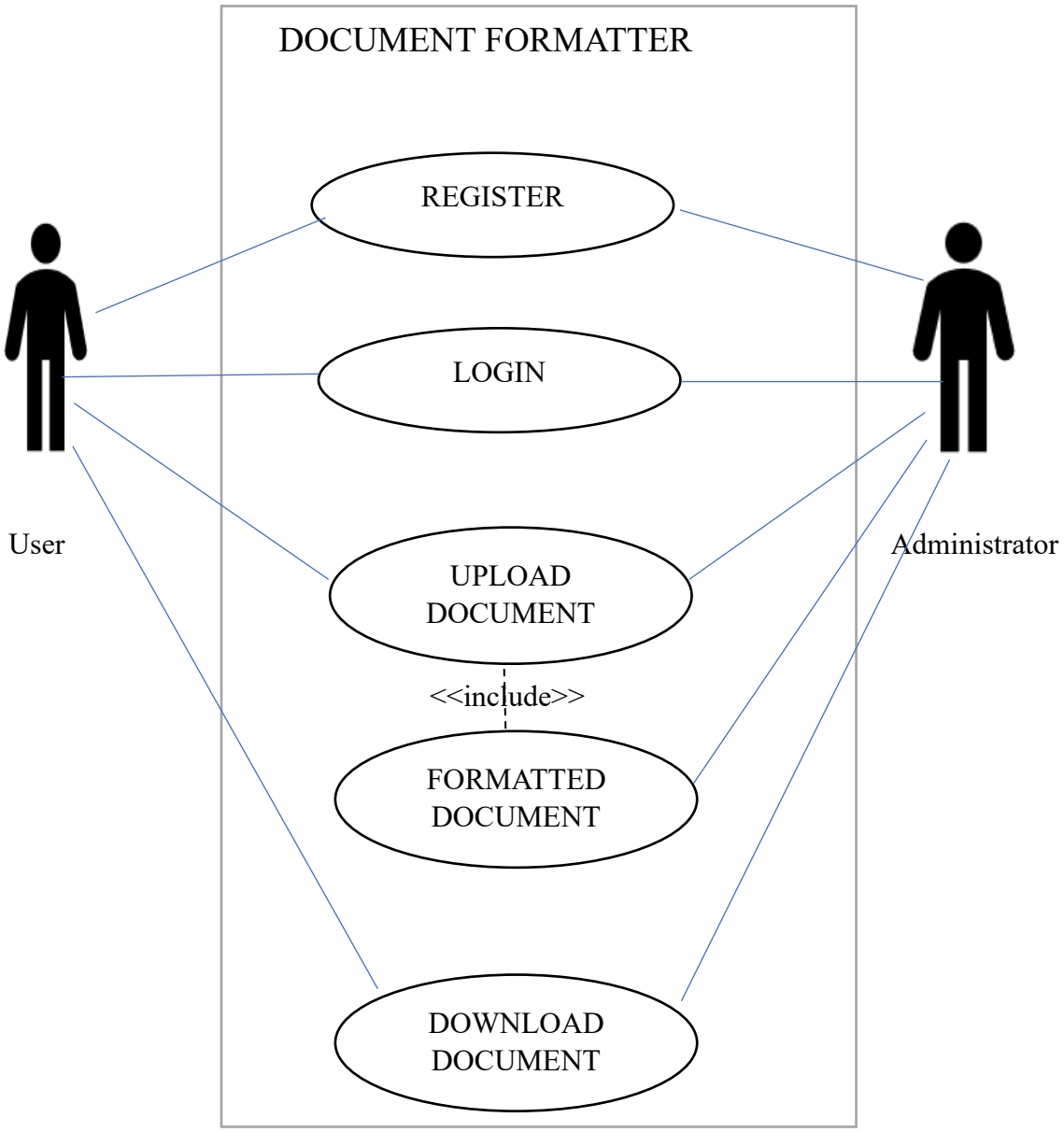

Figure 3.4 use-case diagram of Automatic word formatter

The fig 3.4 illustrates three primary functionalities: "Upload Document," where users upload Word documents; "Format Document," where the system extracts, identifies, and formats document elements, storing them in MongoDB; and "Download Document," allowing users to download the formatted document.

The following is a sequence of events that occur when a user uses the Automated Word Formatter:

The user uploads a Microsoft Word document to the platform.

The system extracts the data from the document.

The system identifies the different elements in the document.

The system formats the elements according to the predefined formatting styles.

The system stores the formatted document in the MongoDB database.

The system generates a download link for the formatted document.

The system sends the download link to the user.

The user clicks on the download link and downloads the formatted document.

3.6.4 Activity Diagram

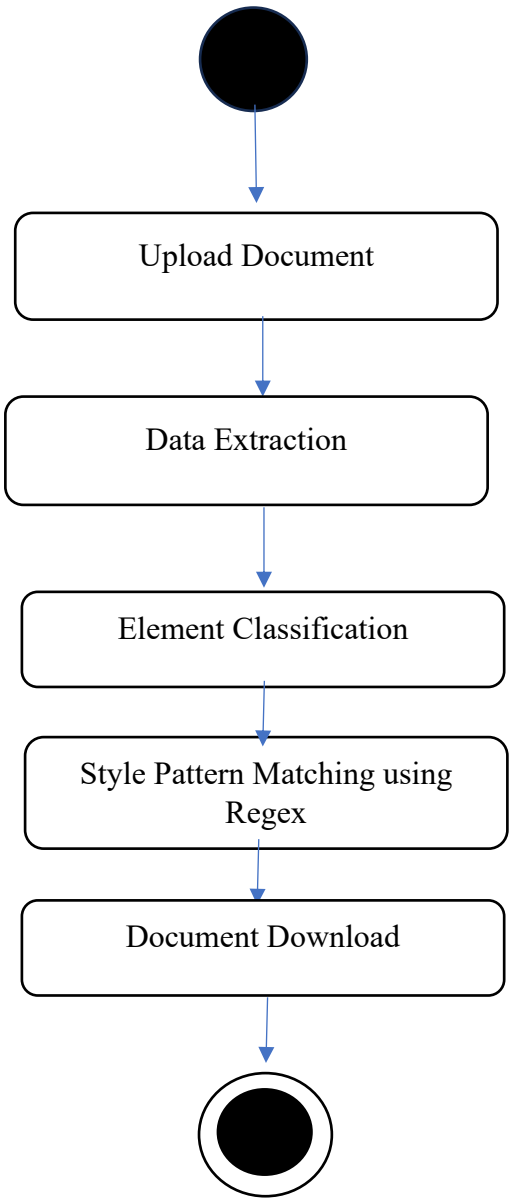

Figure 3.5 Activity Diagram of Automatic word formatter

The fig 3.5 illustrates a structured sequence of actions in the document processing system. The process initiates with the "Upload Document" activity, where a user inputs a document into the

system. Subsequently, the system engages in "Data Extraction," extracting relevant data from the uploaded document. Following this, the "Element Classification" activity categorizes different elements within the document. The process proceeds to "Style Pattern Matching using Regex," where predefined formatting styles are applied based on identified patterns. Finally, the system concludes with the "Document Download" activity, allowing the user to retrieve the processed document. This sequential flow ensures a systematic and efficient progression of tasks, from document input to the application of formatting styles, ultimately providing the user with a downloadable, refined document.

CHAPTER 4

Results and Discussions

4.1 Experimental Results

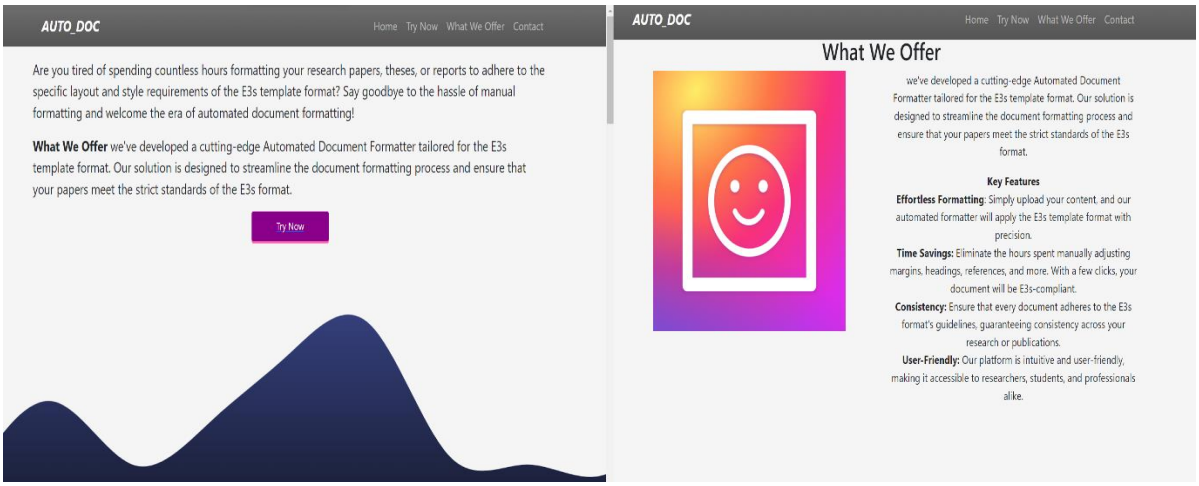

Figure 4.1 User Interface

The project's landing page as shown in fig 4.1, is designed with a user-centric approach, featuring three key components. Firstly, a prominent section outlines the core offerings of the platform, succinctly detailing the automated formatting service according to research paper standards. This ensures that users quickly grasp the primary functionality of the Automated Word Formatter. Secondly, a user-friendly feedback form is seamlessly integrated, allowing users to provide valuable input and suggestions. This element fosters a sense of engagement and inclusivity, encouraging users to share their experiences and contribute to the platform's continuous improvement. Lastly, the landing page incorporates a streamlined upload and download option for Word documents. This intuitive feature empowers users to effortlessly navigate the platform, upload their documents for formatting, and conveniently download the formatted results. Together, these components create a well-rounded landing page that prioritizes clarity, user interaction, and efficient document processing.

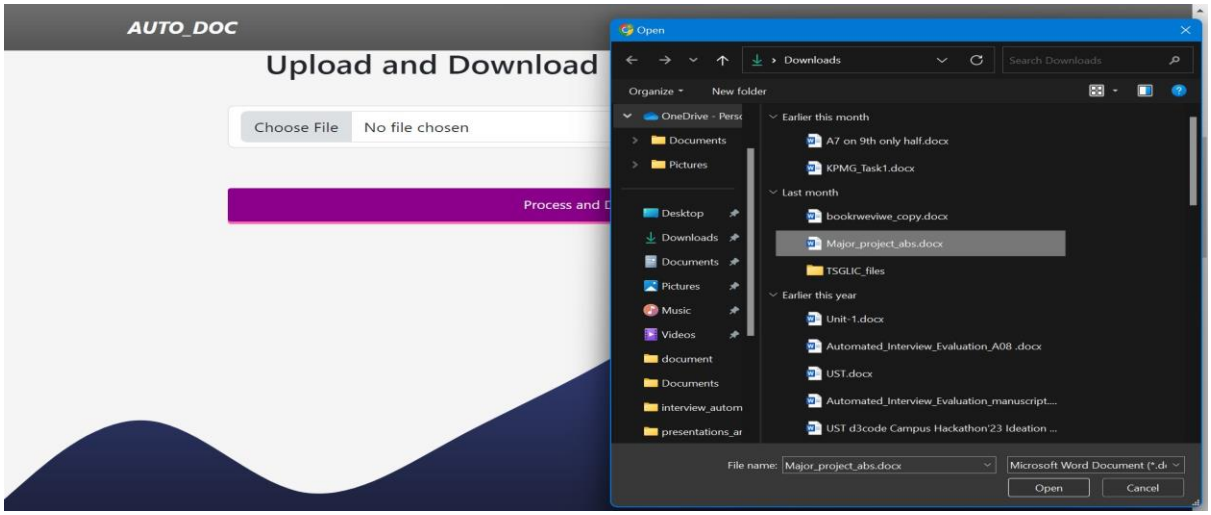

Figure 4.2 Uploading Document

Illustrated in fig 4.2 the user engagement process begins with a clear and user-friendly interface. Users initiate document upload by interacting with the "Choose File" button, which then outputs a window to select the document from the file manger once the file is selected the user can reselect the file again or proceed by clicking the process and download button provided.

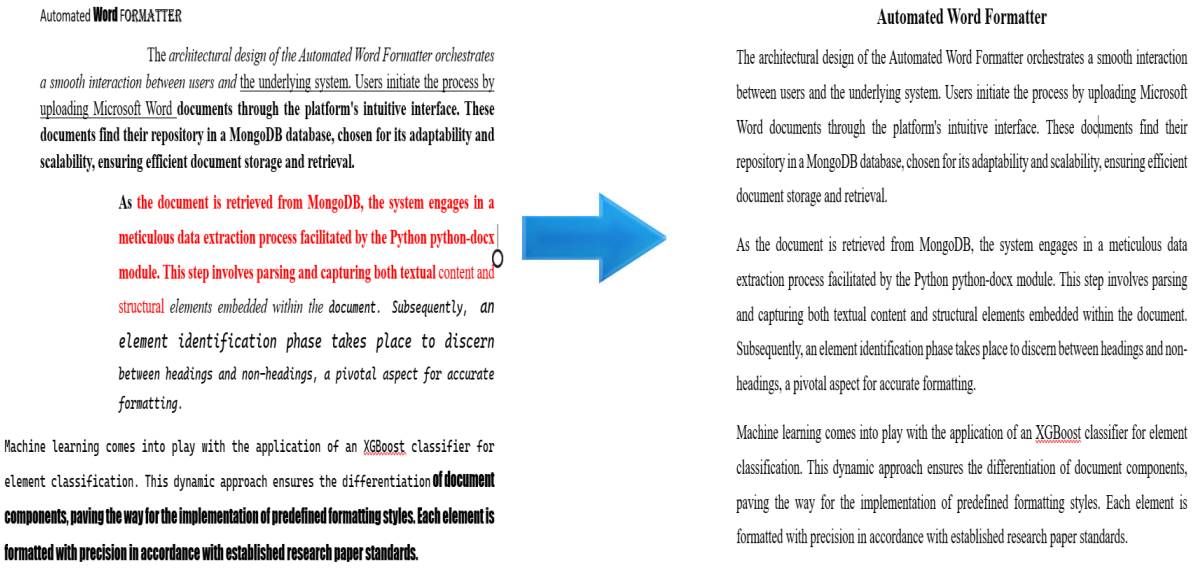

Figure 4.3 Formatted Document

Upon successful document upload, the formatting process commences, as visually represented in Figure 4.3. This pivotal stage involves a systematic approach to document enhancement. The system initiates by collecting essential data from the uploaded document, delving into its textual content and structural elements. Subsequently, an intricate element identification process takes place, distinguishing between headings and non-headings. This

critical step lays the foundation for the application of predefined styles tailored to meet the specific requirements of the user. The user-centric approach ensures that the formatting aligns seamlessly with their preferences and conforms to established standards, contributing to a refined and polished document. This structured workflow underscores the system's commitment to delivering accurate and customized formatting, enhancing the overall user experience.

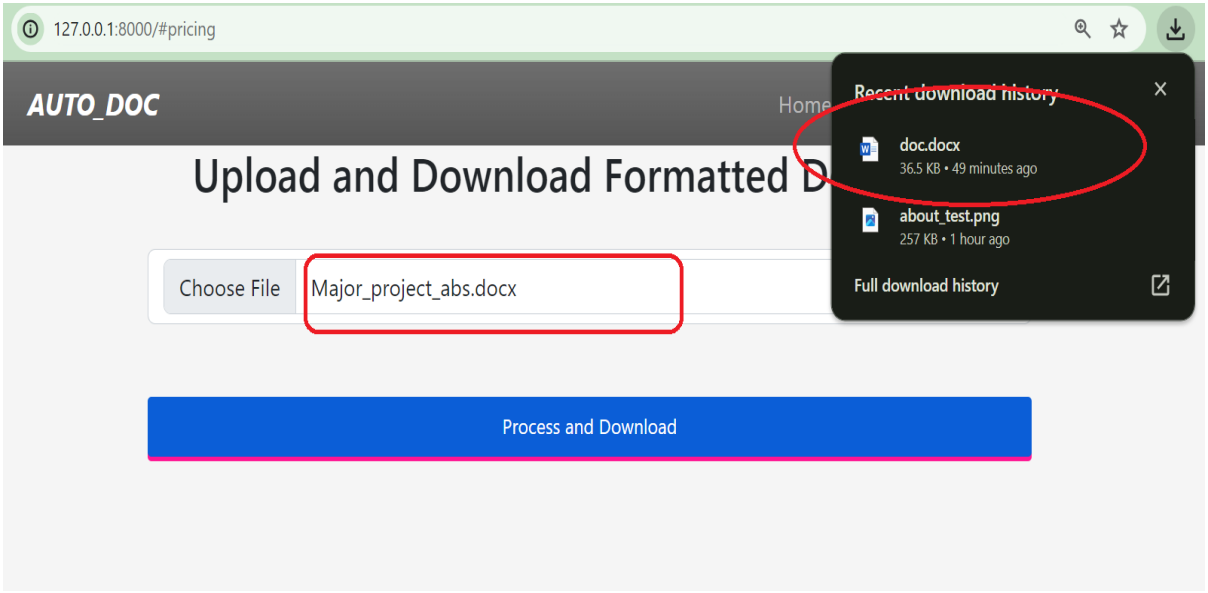

Figure 4.4 Downloading the Document

As depicted in fig 4.4, once the user uploads the required document the name of the uploaded document is displayed to the user, he can change the document if it is not the intended one after this step once the user clicks on the process and download button the document formatting process starts in the backend once the formatting is done the document gets automatically downloaded and is displayed to the user in the downloads section.

4.2 Discussion

The proposed method for automated word formatting holds significant advantages in streamlining the process of document formatting, particularly in the context of research papers. One of its key benefits lies in the efficiency and time-saving it brings to the table. By automating the formatting process, the method eliminates the need for manual adjustments, allowing authors to focus more on content creation rather than intricate formatting details.The automated approach ensures a high level of consistency in formatting throughout the document. This is crucial for maintaining a professional and polished appearance, a challenging feat when relying solely on manual formatting. Adherence to established research paper standards is

another advantage. The method is designed to meet the specific requirements of academic or industry guidelines, enhancing the chances of acceptance and publication of research work.

The user-friendly interface of the web-based platform is a noteworthy feature. Authors can effortlessly upload their documents and receive correctly formatted outputs, making the process accessible to individuals with varying levels of technical expertise. Furthermore, the automation significantly reduces the likelihood of human errors in formatting, particularly important in documents with intricate style guidelines and specific layout requirements.

Flexibility and customization are also key strengths of the proposed method. While automating standard formatting, users may have the flexibility to incorporate their preferred styles or make adjustments as needed. This strikes a balance between automation and user control. The scalability of the automated method is crucial for handling a large volume of documents efficiently, making it suitable for academic institutions, publishers, or any entity dealing with numerous documents. Additionally, the proposed method may incorporate version control features, allowing users to track changes made during the formatting process. This enhances collaboration and provides a mechanism for reviewing and reverting modifications if necessary. In conclusion, the automated word formatting method offers a transformative solution, combining efficiency, consistency, adherence to standards, and user-friendly features to streamline the document preparation process for researchers, authors, and institutions alike.

CHAPTER 5

CONCLUSION AND FUTURE ENHANCEMENTS

5.1 Conclusions

The Automated Word Formatter stands as a transformative solution, rendering document formatting a hassle-free. Users experience seamless formatting magic by effortlessly uploading their documents to the platform. This marks a significant shift, making the once tedious task of formatting documents a breeze. One of the standout features is the platform's commitment to delivering an effortless formatting experience. Users can achieve a polished and professional look for their documents without the need for intricate manual adjustments. The formatter streamlines the process, enabling users to channel their focus on the document's content rather than getting entangled in formatting intricacies. Consistency in style is a paramount achievement facilitated by the Automated Word Formatter. Fonts, alignments, and spacing are standardized, ensuring a cohesive and professional appearance across documents. This level of consistency is valuable for users aiming to maintain a unified style in their written work. A notable advantage of the formatter is its capacity to save time. The manual adjustment of paragraphs and headings becomes a thing of the past. The platform's streamlined process allows users to efficiently allocate their time to more critical aspects of content creation, knowing that formatting complexities are expertly handled. The formatter's adaptability shines through, making it a universal solution for diverse document types. Whether users are crafting a research paper or report the platform readily adapts to varying needs, providing a versatile and comprehensive formatting solution.

The user-friendly interface further enhances the formatter's accessibility. Its intuitiveness accommodates users with varying technical expertise, ensuring a straightforward process from document upload to receiving a beautifully formatted version. This accessibility is key to fostering a positive user experience for individuals with different levels of familiarity with formatting tools. Beyond mere formatting, the Automated Word Formatter contributes significantly to the professional presentation of documents. The platform elevates the visual appeal of written work, adding a refined aesthetic touch that is particularly advantageous for those seeking a polished and sophisticated presentation. Underlying all these benefits is the platform's commitment to conserving document integrity. Leveraging the python-docx module, the formatter ensures that the core content and structural integrity of the document remain

untouched. The focus is solely on enhancing the stylistic elements, providing users with an assurance of document preservation.

5.2 Future Enhancements

These advancements can be addressed ahead in future

Additional Document Formats: Extend support beyond DOCX to include other popular document formats, ensuring a broader user base.

Collaborative Editing: Implement collaborative editing features to enable multiple users to work on a document simultaneously.

Enhanced Error Handling: Improve error handling mechanisms to provide users with clear feedback in case of formatting issues or document errors.

Optimization for Large Documents: Optimize the formatter to efficiently handle and format large documents, ensuring robust performance.

User Accounts and History: Introduce user accounts and a document history feature to allow users to revisit previously formatted documents and track their usage.

Advanced Styling Options: Introduce advanced styling options to allow users to customize formatting according to their specific needs.

Document Editing Platform: Providing the user, a platform to edit the word document in the browser itself.

CHAPTER 6

APPENDICES

A sample code of Django views which handles the process, home and download modules.

```
from docx import Document
from docx.shared import Mm,Pt
from docx.enum.text import WD_ALIGN_PARAGRAPH
from django.core.files.base import ContentFile
import re
import os
# from gridfs import GridFS
from .models import ProcessedFile


def home(request):
    return render(request,"home.html")


def process(request):
    if request.method == "POST":
        file = request.FILES["myFile"]
        existing_doc = Document(file)
        e3s_doc = Document()
        section = e3s_doc.sections[0]
        section.start_type
        section.page_width = Mm(170)
        section.page_height = Mm(250)
        section.left_margin = Mm(20)
        section.right_margin = Mm(20)
```

```
        section.top_margin = Mm(24)
        section.bottom_margin = Mm(16)
        title = False
        abs = r'^Abstract\b.*'
        for paragraph in existing_doc.paragraphs:
            if title == False:
                e3s_paragraph = e3s_doc.add_paragraph((paragraph.text).title())
                e3s_paragraph.paragraph_format.space_before = Mm(22)
                e3s_paragraph.paragraph_format.space_before = Mm(6)
                for run in e3s_paragraph.runs:
                    run.font.name = 'Arial'
                    run.font.size = Pt(16)
                    run.bold = True
                title = True
                e3s_paragraph.alignment =
WD_ALIGN_PARAGRAPH.JUSTIFY
                continue
            # Create a new paragraph in the E3S-formatted document
            e3s_paragraph = e3s_doc.add_paragraph(paragraph.text)

            s = e3s_paragraph.text
            for run in e3s_paragraph.runs:
                run.font.name = 'Times New Roman'

            if re.search(abs,s):
                split_index = s.find("Abstract") + len("Abstract")
                first_half_text = s[:split_index]
                second_half_text = s[split_index:]
                e3s_paragraph.clear()
```

```
            run1 = e3s_paragraph.add_run(first_half_text)
            run1.bold = True
            run1.font.name = 'Arial'
            run2 = e3s_paragraph.add_run(second_half_text)
            run2.font.name = 'Times New Roman'
        e3s_paragraph.alignment = WD_ALIGN_PARAGRAPH.JUSTIFY
        e3s_doc.save('formatted_document.docx')
    return redirect('/download')
  else:
    return HttpResponse("<h1> Some Technical issue </h1>")



def download(request):
  file_path = os.path.join(os.getcwd(),"formatted_document.docx")
  if os.path.exists(file_path):
    with open(file_path, 'rb') as file:
      response = HttpResponse(file.read(),
content_type='application/vnd.openxmlformats-
officedocument.wordprocessingml.document')
      response['Content-Disposition'] = 'inline; filename="doc.docx"'
      return response
  else:
    return render(request, 'error_page.html', {'error_message': 'File not
found'})
```

REFERENCES

[1] Kirill Chuvilin, "Machine Learning Approach to Automated Correction of Latex Documents," in *CONFERENCE OF FRUCT ASSOCIATION*, 2016.

[2] Oliveira, "Two Algorithms for Automatic Document Page Layout," in *Proceeding of the Eighth ACM Symposium*, 2008.

[3] Nathan Hurst, "Review of Automatic Document Formatting," in *Proceedings of the 9th ACM Symposium*, 2009.

[4] Ibrahim Nasser, "Web Application for Generating a Standard Coordinated Documentation for CS Students' Graduation Project in Gaza Universities," *International Journal of Engineering and Information Systems (IJEAIS),* vol. 1, no. 6, 2017.

[5] Graeme Gange, "Optimal Automatic Table Layout," in *Proceedings of the 11th ACM Symposium*, 2011.

[6] Mihai Bilauca, "Automatic Minimal-Height Table Layout," *INFORMS Journal on Computing,* vol. 27, 2015.

[7] John Bateman, "Towards Constructive Text, Diagram, and Layout Generation for Information Presentation," *Computational Linguistics,* vol. 27, 2001.

[8] Igor A. Bolshakov, "Text Segmentation into Paragraphs Based on Local Text Cohesion," *Springer-Verlag,* 2001.

[9] Widiastuti, "Document Image Extraction System Design," in *3rd International Conference on Informatics, Engineering, Science, and Technology (INCITEST 2020)*, 2020.

[10] "TableNet: Deep Learning Model for End-to-end Table Detection and Tabular Data Extraction from Scanned Document Images," in *International Conference on Document Analysis and Recognition (ICDAR)*, 2020.

[11] Maud Ehrmann, "Named Entity Recognition and Classification in Historical Documents: A Survey," *ACM Computing Surveys,* vol. 56, no. 2, 2023.

[12] Stefan Schweter, "FLERT: Document-Level Features for Named Entity Recognition," in *Computation and Language*, 2021.

[13] Yiheng Xu, "LayoutLM: Pre-training of Text and Layout for Document Image Understanding," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

[14] N. Nasyrov, "Automated formatting verification technique of paperwork based on the gradient boosting on decision trees," in *Procedia Computer Science*, 2020.

[15] Dr. Venkatesan, "An Implementation Approach Towards The Automation Of Document Formatting Using Python," *International Journal of Aquatic Science ,* vol. 12, no. 2, 2021.