VIETNAM NATIONAL UNIVERSITY OF HOCHIMINH CITY

THE INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

A SYSTEM FOR THESIS REPORT MANAGEMENT AND EVALUATION WITH EVENT-DRIVEN ARCHITECTURE

By

Ung Thu Hà

A thesis submitted to the School of Computer Science and Engineering in partial fulfillment of the requirements for the degree of
Bachelor of Information Technology/Computer Science/Computer Engineering

Ho Chi Minh City, Vietnam
Year 2023

A SYSTEM FOR THESIS REPORT MANAGEMENT AND EVALUATION WITH EVENT-DRIVEN ARCHITECTURE

APPROVED BY:

_____ ,

Nguyen Van A, Ph.D, Chair (*Example*)

(Type Committee names beneath lines)

_____

(*Typed Committee name here*)

_____

(*Typed Committee name here*)

_____

(*Typed Committee name here*)

_____

(*Typed Committee name here*)

THESIS COMMITTEE

(Whichever applies)

1

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude to Dr. Tran Thanh Tung, my advisor. I am thankful to have been given the opportunity to carry out this research under his guidance. Dr. Tung has been incredibly helpful and provided me with valuable advice which I am deeply appreciative of.

I am grateful for the School of Computer Science and Engineering at International University for giving me the essential prerequisite knowledge to conduct this research with confidence. It has been a great experience being enrolled in the Computer Science program and studying under knowledgeable instructors.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

ABSTRACT

At the end of each semester, instructors are tasked with grading a large number of thesis reports. Flaws in many of the submissions cause them to spend more time grading which increases the workload. This can be prevented by giving students feedback on those mistakes so that they can revise before the final submission. However, International University does not use any software that analyzes thesis reports. There are also no solutions on the market that can generate feedback based on a predefined template. Therefore, this study develops a system that allows such tasks to be performed. By implementing document processing techniques with an event-driven architecture, the final product is a thesis mangement system that can generate reports on various aspects of a thesis. The system is scalable with room for more comprehensive feedback services to be added in the future. In combination with an intuitive user interface, students and instructors can receive insights and visualized analysis from these reports. Results showed that the feedback generated is descriptive and relatively accurate. They encourage students to revise their thesis and ultimately reduce workload on instructors.

CHAPTER 1
INTRODUCTION

1.1.   Motivation

There are a large number of thesis reports which an instructor has to examine and grade each semester. This puts a large workload on them due to the lengthiness and complexity of each document. Many are incoherent due to grammatical or layout errors, thus requiring more effort and time to be graded. However, most students are not aware of flaws in their work. A feedback mechanism can be implemented to inform them and help improve the quality of their thesis.

The thesis submission process happens at the end of every semester. During this stage, graduating students submit their reports for grading. These reports are the written products of more than half a year of research. Thus, they are often long and elaborate with different complex topics. As a result, grading them requires a great amount of time and mental effort.

For many undergraduate students, the thesis is the first academic paper they have to write. Therefore, it is common that mistakes such as semantic errors, disorganized structure or incorrect formatting happen. These flaws, depending on the severity, can make the grading process become more time-consuming for instructors.

A study by Chaparro B. et. al (2004) showed that reading comprehension is reduced when participants read text with no margin and sub-optimal leading, i.e., space between each line. Another research indicated that different fonts affect reading speed up to 35% when compare between the most and least legible typefaces (Wallace S. et. al, 2022). As a consequence, the time required to grade one thesis with those characteristics can be much longer than grading one that is well-structured and correctly formatted.

Furthermore, students are often not aware of these mistakes in their submissions. A solution would be to use a software that automates document analysis. By generating feedback on multiple aspects, students are informed of the shortcomings in their reports. This allows them to revise according to how they see fit.

Additionally, when students receive sufficient feedback, a higher quality report can be produced. Lipnevich A. A. and Smith J. K. (2009) suggested that elaborate feedback, unaccompanied by grade or praise, is strongly related to a student's improvement in their work. Whether it was computer generated or given by their instructor did not significantly impact the results. Feedback that provides solutions to overcome issues in addition to detailed analysis has consistently been found to be the most effective in improving student's performance.

Therefore, a thesis analysis system could provide great benefits, not just for students but also instructors. Students can revise several times, improve the quality of their thesis and submit the best version. Coherent and well-written reports ease the grading process for instructors because they are comprehensible. Furthermore, feedback generated can be used as checkpoints to help instructors make informed decisions.

Currently, International University does not use any software that provides feedback on thesis submissions. There are also no solutions on the market that could analyze documents based on criteria such as checking chapter titles, page count and ensuring correct font. This provides an opportunity for implementing a software that could perform such tasks all at the same time.

The above-mentioned services can be developed using document processing techniques. Document processing is a field that handles and transforms text documents. It has wide-ranging applications in areas like content summarization, sentiment analysis and document classification. Document processing also involves the use of other fields such as optical character recognition (OCR) and natural language processing (NLP). Its complexity ranges from simple pattern matching to using AI for intelligent detection.

Having multiple analysis tasks, the potential software would benefit from using an event-driven architecture with microservices. First, by developing them as loosely coupled services, these tasks can be independently deployed and scaled when necessary. It also ensures that changes made in one service don't affect others. Second, an event-driven architecture allows components to asynchronously communicate through events they are subscribed to. Thus, combining these two approachs creates an efficient way to manage services.

In conclusion, using a software that provides detailed analysis and generates comprehensive feedback could significantly benefit both students and instructors. It encourages students to revise and improve their thesis, producing a high quality report for their final submission. Additionally,

the feedback is accessible to instructors which provides insight and helps them make informed decisions on the thesis. As a result, grading requires less effort and instructors spend less time on assessment. By using an event-driven architecture with microservices, a responsive and scalable thesis mangement system can be built.

1.2. Problem Statement

Instructors have to grade dozens of thesis reports every semester. Poorly written or unstructured documents require more time to assess because they are less intelligible. Multiplied by the total number of submissions, this causes a large amount of mental workload to be put on instructors. Many students are not aware of those shortcomings due to limited time or overlooking the guidelines. This can be prevented if they receive feedback that inform them on what to improve. A higher quality report can be produced when students revise sufficiently, thus eases the grading process for instructors. However, there is no software available that could analyze thesis documents based on all criteria predefined by the university. This creates an opportunity for the development of an application that can generate feedback on various aspects of a report based on the guidelines provided. This study builds a thesis management system using event-driven architecture with document analysis microservices. Combined with a user interface, students and instructors can receive more insights with visualized feedback.

1.3. Objectives

The ultimate goal of this research is to build an efficient, responsive and accurate thesis checking system that reduces grading time for instructors and improves quality of thesis for students. In order to achieve this, a few objectives are set to assess the progress:

(1) Evaluate the pain points that students and instructors experience during the thesis submission process by conducting interviews with peers and advisor.
(2) Identify where students often make mistakes in a thesis report.
(3) Define the services that analyze those aspects and compare them against the template provided by International University with the use of document processing techniques.

(4) Develop a robust event-driven microservices system to handle thesis submissions and generate reports.

(5) Ensure that the system is flexible and scalable for more services to be added in the future while not compromising performance.

1.4. Scope and Assumption

This study develops a thesis management system for the students and instructors of the Department of Computer Science and Engineering in International University. Therefore, its performance will be evaluated based on the number of reports submitted in this Department only. In particular, there are fewer than 50 submissions per semester. However, the system allows students to submit multiple times before the deadline. This increases the expected number to more than 100 per semester, with at least 10 reports submitted per day.

In order for the system to be functional, assumptions have to be made regarding initial requirements and available resources. They establish the context within which this study is conducted and provide transparency to readers. Interpretations of the system's functionalities should be made on the basis of the given assumptions.

First, users of the system are predefined in the database with their unique IDs and passwords. This eliminates the need for registration. Defining user accounts beforehand is also realistic and consistent with the application in use by International University.

Second, students are allowed to submit their reports multiple times before the deadline as opposed to only once or at most twice in reality. This is to encourage revision so that students can ensure their final submission is of the best quality.

Reports must be submitted in the PDF format due to its compatibility with many devices which ensures the document always appears the same. Additionally, PDFs can sometimes be more compact and of a smaller size than the original document.

Finally, this system is developed using the software and hardware resources provided by the Department. Therefore, the ability to scale will be limited by the server's processing power and its memory. The server uses one CPU core, with 83GB of available disk space and 8.1GB of available memory.

Figure 1. *Available disk space on provided server*

Figure 2. *Available memory on provided server*

Figure 3. *Number of CPU cores in provided server*

1.5. Structure of Thesis

This thesis follows the traditional structure of an academic research paper. According to the template provided by International University, the following sections are included: Abstract, Introduction, Related work and Comparison, Methodology, Implementation and Results, Discussion and Evaluation, Conclusion and Future work, References.

- Abstract:

  Abstract provides a short but concise summary of this thesis. It compresses the full content into an informative paragraph with the aim of getting key findings across to readers. Abstract also helps prepare them for the analyses of the forthcoming chapters.

- Introduction:

  This chapter presents the background that motivates the study and raises a problem statement. It also includes information regarding the scope, objectives and context within which the study is conducted.

- Related work and Comparison:

  Relevant applications will be analyzed in this chapter. Specifically, their technology and implementation. An overview will be given on the techniques used in this study and how they compare to those applications.

- Methodology:

  This chapter defines the design strategy utilized to construct the system. This includes explanation for the chosen design as well as the tools used. Additionally, diagrams are provided to help readers visualize use cases, requirements and system architecture.

- Implementation and Results:

  Code structure and setup are included to show how the system is implemented. Different use cases are performed and its results are recorded in the form of pictures or videos.

- Discussion and Evaluation:

  Discussion interprets the results obtained and identifies whether the use cases performed are successful. Evaluation analyzes the results, the system's strengths and weakenesses.

- Conclusion and Future work:

  This last chapter reiterates the motivation defined in the Introduction section. Key interpretations are identified and their relevance in the real world is made clear. Limitations leave room for the discussion of future work, specifically what could be added, removed and improved.

- References:

  Sources that have been referenced during the writing of this research are listed here. Despite not being considered as one of the main chapters, this section is arguably one of the most important because it is where we acknowledge the contribution to those sources.

CHAPTER 2
RELATED WORK & COMPARISON

2.1. Related work

This chapter looks into the technology of Grammarly, ChatGPT and Turnitin. These are applications that have similar functionalities to the system in this study and can be used for academic purposes. They scan textual content, transform them and generate analysis on a variety of aspects.

2.1.1. Grammarly

Grammarly is a popular AI software that provides grammatical review. Its services are customizable and versatile, assisting users in perfecting their writing. It is a trusted and widely used application, not only for individuals but also enterprises. Grammarly's success can be attributed to its adoption of advanced AI and NLP technology which gives its services great accuracy.

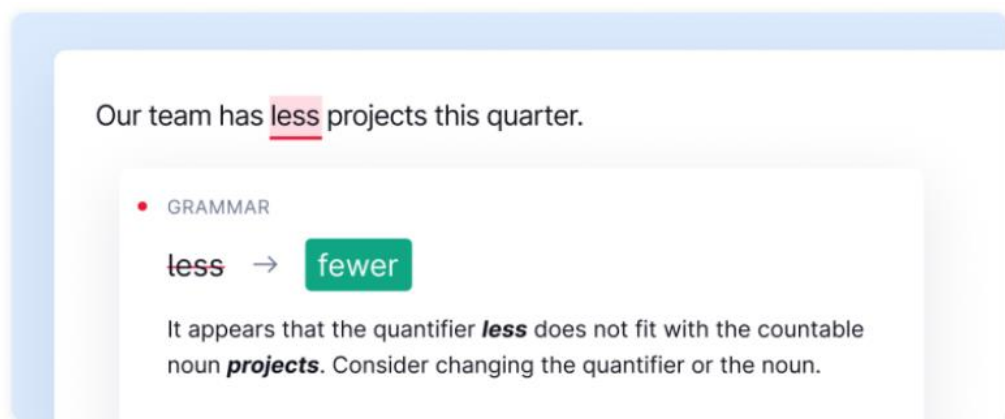**Figure 4.** *Grammarly*

Figure 5. *Grammarly grammar check*

Grammarly offers many tools to users. Its most popular feature, the grammar checker, reviews the inputted text's grammar, spelling, tone and punctuation. From tenses to parts of

speech, this software can pinpoint most errors in English. Additionally, it not only provides the solution but also the explanation behind it.
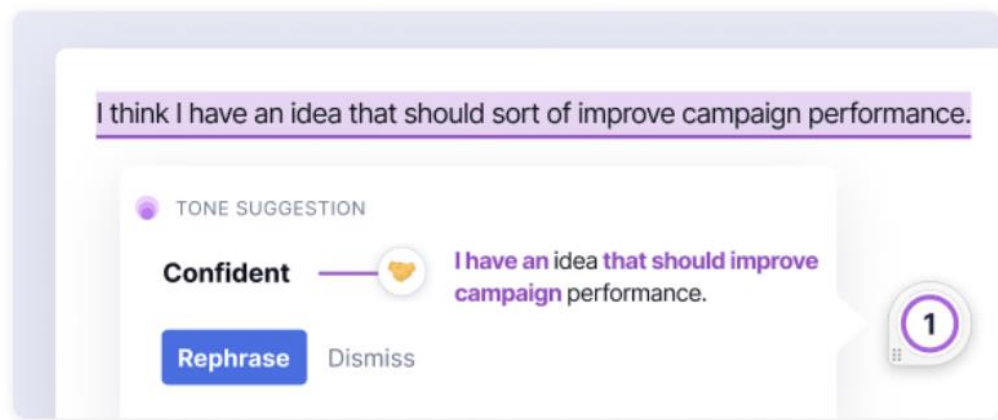
Figure 6. *Grammarly tone suggestion*

When a user submits input, the entire syntax structure of the text gets processed. The software scans individual words, punctuations and forms relationships between them all within a few miliseconds. Part-of-speech tagging, dependency parsing and semantic analysis are just a few of the NLP techniques used by the service to process user input.

Then, the software applies AI algorithms to analyze linguistic choices. These algorithms have been trained on an extensive amount of data sets. Thus, they can recognize textual patterns, derive context and understand English grammatical complexity.

Grammarly's algorithms are becoming increasingly more effective and accurate by applying Machine Learning. Users are provided with the options to accept or reject each suggestion. The results of these interactions are recorded and processed into a feedback loop that allows the system to adapt and improve.

By employing a team of expert scientists in the fields of data science and linguistics, Grammarly ensures that their technology is continuously evolving. Furthermore, advancements in AI and NLP research are being made yearly. This provides opportunities for the team to consistently incorporate and improve their system.

2.1.2. ChatGPT

ChatGPT is an AI chatbot developed by OpenAI. It is programmed to engage in conversations with users and generate detailed responses. ChatGPT is trained on a vast amount

of data and receives consistent fine-tuning from user feedback. Its capabilities are wide-ranging and often used for academic purposes.

Figure 7. *OpenAI*

The most popular function of ChatGPT is its interactive conversational ability. ChatGPT can engage in discussions of different topics and adjust its answers based on user input. It can assume the tasks of providing information, content generation and giving advice. Details and clarity of the prompt also affects quality of the answer.
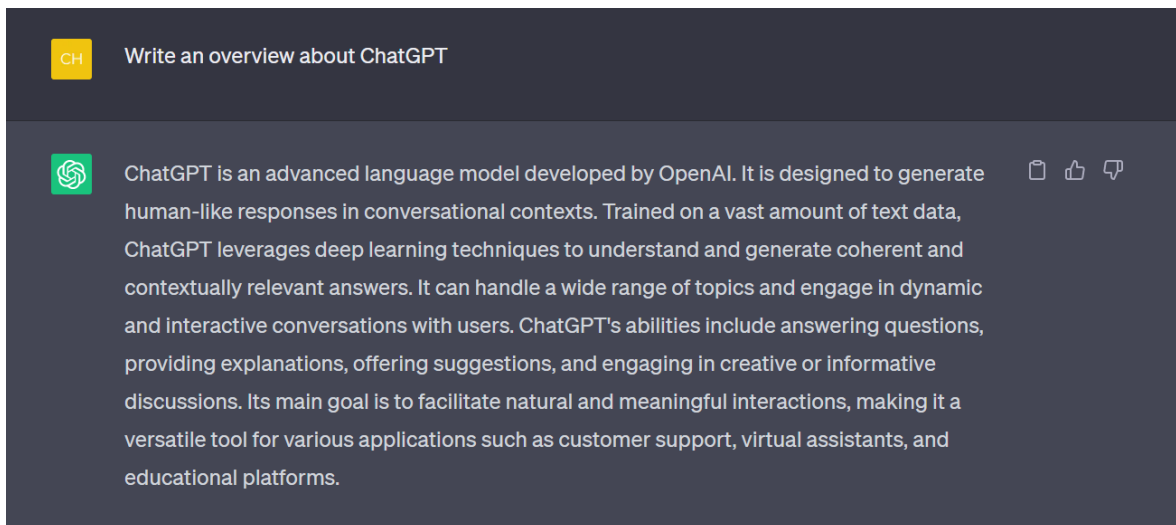
Figure 8. *A question and answer example using ChatGPT*

ChatGPT uses generative pre-trained transformers (GPT), a large language model, to perform various prompt-based tasks. It is a neural network that consists of trillions of parameters and requires great computing power. GPT is designed to be a general purpose deep learning model that excels at a wide range of areas. As the name suggests, it is developed using the techniques of pre-training and fine-tuning.

Pre-training develops the model's ability to predict tokens, such as completing a sentence or filling in a missing word. This may seem like a simple task in the context of advanced deep learning. Nevertheless, it allows the model to learn about textual patterns and relationships, thus forming its knowledge on the grammatical structure and complexity of the language.

Next, it is fine-tuned on more specific tasks such as content summarization, language translation and sentiment analysis. This process enhances the pre-trained model by exposing it to diverse domains, allowing it to adapt and improve performance. Reinforcement learning is another technique that is implemented in which human-generated feedback is provided to the model's responses. These responses are ranked in terms of human preference and a reward is applied accordingly.

By using a large data set that contains trillions of tokens, OpenAI's GPT models have achieved impressive results. This is demonstrated in their ability to produce highly accurate responses on various topics. Thus, the models have not only been implemented as a chatbot but also as content generators, text summarizers, virtual assistants, etc.

Similar to other generative large language models, GPT experiences "hallucinations" now and then. Hallucination in an NLP model is the situation where it produces incorrect, illogical or meaningless content. This may be caused by errors present in the training data set or from biases generated by human during reinforcement learning.

OpenAI's GPT is among the most prominent large language models that have been revolutionizing the AI industry. Its extensive capabilities aid humans in numerous tasks that were once manual. ChatGPT's widespread attention have opened up possibilities for its application in several domains. However, more research is needed to ensure the ethical and responsible use of this powerful language model.

2.1.3. Turnitin

Turnitin is an online plagiarism detection application that is widely used in educational institutions. It assists students in ensuring the integrity of their documents by performing checks against large databases of academic content. Turnitin works by identifying similarities between the submission and other existing sources. The results can be used by students to revise and avoid plagiarism or by educators to determine an appropriate grade.

Figure 9. *Turnitin*

Turnitin's most popular tool is its originality checker. Submitted documents get scanned and compared against content found in its academic databases. Then, a report is generated that highlights any potential similarity and calculates an originality percentage.

The algorithm behind this tool includes two main steps: scanning and matching. The scanning process involves a look-up into content available in its own databases, which are constantly updated with published and private academic sources. Next, matching uses NLP techniques such as fingerprinting, bag of words, etc. to search for similarities between the submitted text and content from the databases.

In addition to its plagiarism detection feature, Turnitin also offers other services that help improve quality of the submitted work. GradeMark allows instructors to provide detailed feedback in many forms such as text, audio and highlighting. Integrated within the tool is an automated grammar and spelling checker to optimize the feedback process. PeerMark is a peer review tool that allows students to evaluate the work of their classmates. Students can give anonymous or known reviews either by writing or answering given questions.
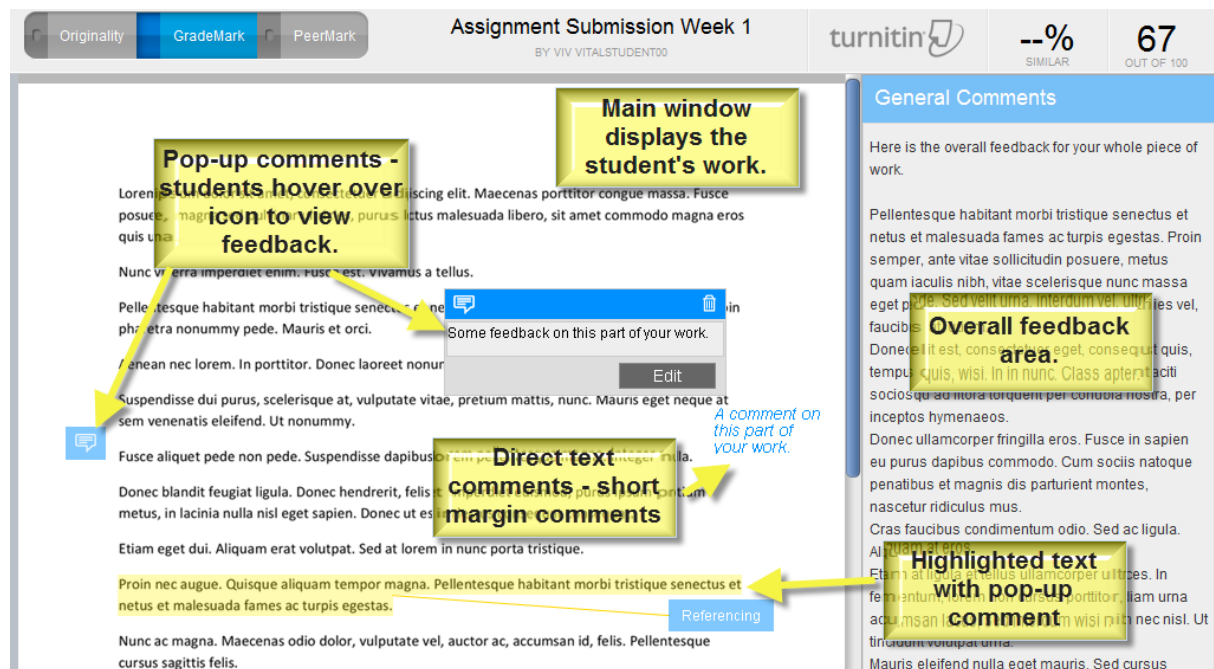

Figure 10. *Turnitin's GradeMark tool*

2.2.   Background

Before comparing this study's system to the applications above, its technology and functionality must first be considered. This thesis management system consists of many services that analyze reports based on different criteria. They scan the document for certain information and run algorithms to process the extracted text. These algorithms utilize document processing techniques to perform various tasks. The generated analysis is stored and accessible in the file system. Option can be selected on the user interface to display the analysis. The system is developed using the microservices and event-driven architecture.

2.2.1. Document processing techniques

Document processing is the manipulation and transformation of text documents. It involves techniques used in optical character recognition (OCR) and natural language processing (NLP). Document processing can be used to automate tasks and extract insights.

This system implements relatively simple NLP algorithms such as pattern matching and chunking. For complex analysis tasks such as keyword extraction, GPT – OpenAI's neural network language model, is used to achieve more accurate results.

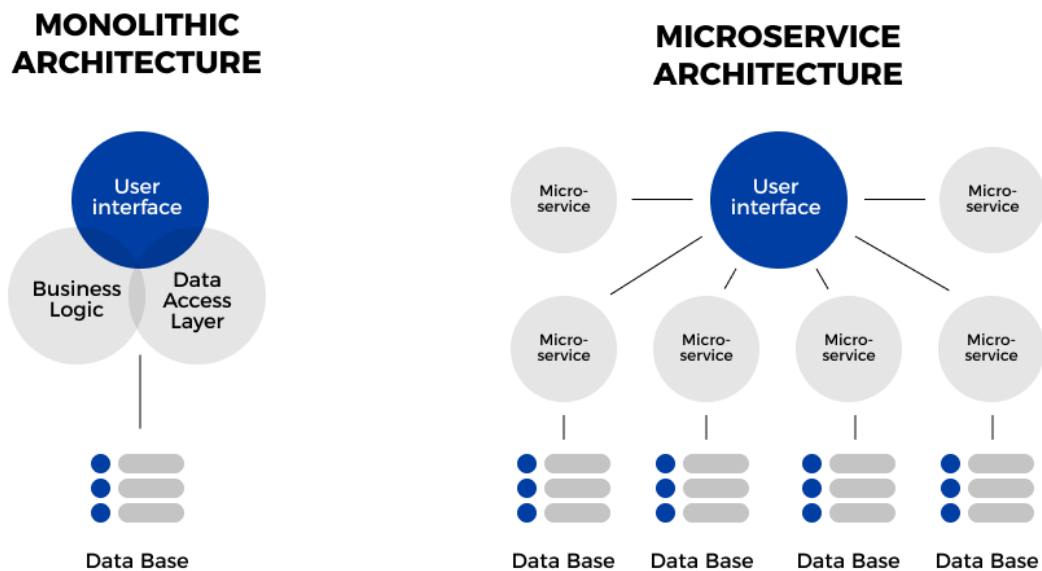### 2.2.2. Microservices architecture

Figure 11. *Monolithic vs. Microservices architecture*

In a traditional monolithic architecture, the application is built as a tight-knit unit of interdependent components. It is straightforward to develop since logic for all services are kept

at the same place and a single database is used. Since all services are kept at the same place, making changes to one requires building and deployment of the entire application.

On the contrary, microservices architecture divides the system into modular components that are loosely coupled. These components do not directly interact with each other and have their own data store. They can be independently deployed and without affecting one another. The microservices architecture is a popularly used architectural style for developing complex and scalable software systems.

2.2.3. Event-driven architecture

Event-driven architecture is a software design pattern that focuses on the asynchronous communication between components. An event is considered as a change in state. Components can be subscribed to many events and programmed to react when they happen. This approach is used in systems that require real-time processing. It can be combined with the microservices architecture to provide an efficient inter-service communication mechanism.

To facilitate communication between components, message brokers are used. It is an intermediary that receives and stores messages from the sender, routes them to the correct receivers and ensures their delivery.

Figure 12. *Pub/sub pattern in event-driven architecture*

This system implements publish/subscribe, the most commonly used messaging pattern in event-driven applications. In this pattern, senders and receivers have no information about each other. Instead, the sender publishes messages to specific channels managed by the message broker. Receivers that are subscribed to those channels are guaranteed to receive the messages. This makes the system flexible since receivers can be added or removed without affecting the sender.

2.3.   Comparison

This study's thesis management system and the applications mentioned have the same goal of improving content quality and reducing workload for users. Despite that, their functionalities and implementation are vastly different.

First, Grammarly and Turnitin only provides a few specific services. They also cannot analyze content based on user-defined criteria. ChatGPT has the capabilities to do this. However, since it is a prompt-based software, the quality of its answer depends on the clarity of the prompt. Furthermore, these prompts are limited to 3000 words. With most thesis reports having twice the amount of words, it would be difficult to use ChatGPT for document analysis in this case.

On the other hand, the thesis management system offers many services that could process lengthy documents. The generated analysis persists and can be accessed within the file system. These services can be added and modified with ease due to the implementation of an event-driven microservices architecture.

Second, the applications reviewed can produce quick, highly accurate and comprehensive results. This is because they are large funded companies with many experts in the field and powerful resources. On the contrary, the services developed in this study use simple algorithms. Therefore, the tasks performed are not complex and generated analysis can sometimes be flawed.

In conclusion, each application is suitable for certain projects. They all provide beneficial insight and help improve the quality of a document. In the context of analyzing thesis reports and reducing workload for instructors, the thesis management system is the best solution.

CHAPTER 3
METHODOLOGY

"A chain is only as strong as its weakest link". This saying holds true for many aspects, including the development of a software product. A system should be well-thought-out regardless of its function, scale or complexity. Having a good system design lays the foundation for a successful application. Therefore, it is essential that a great amount of effort and consideration is put into planning and designing this system.

3.1. Requirement analysis

The planning stage involves defining the functional and non-functional requirements. A use case diagram is included to help visualize interactions with the system. Effective planning ensures that the final product is well-aligned with business objectives.

3.1.1. Functional requirements

Functional requirements define the necessary features this system should have. They describe the intended functionalities and user interactions. In short, functional requirements answer the question "What does this system do?".

	Function	Description
1	User authentication	Users can log in or out using their provided credentials. The software only allows access after a user logs in.
2	User authorization	User type determines accessibility to certain feaures. For example, students can submit their thesis but instructors cannot.
3	Thesis submission	Students can use the user interface to input information about their thesis and submit their reports.
4	Resubmission	Students can resubmit for as many times as they want. Previous submissions and their reports are still accessible.

	Function	Description
5	Detect service completion	Completion is detected when the generated analysis is present at the folder location.
6	File storage	Submissions and analysis results get stored in a folder with the student's credentials.
7	File retrieval	Files stored can be read or written depending on the properties set for that file.
8	Data management	Each service has its own database. The types of information stored include user credentials, thesis reports and analysis results. Data can be retrieved using queries.
9	Document analysis	Submissions will be analyzed by services to produce comprehensive reports.
10	Display analysis	User interface displays analysis of the selected aspect to users. This can include charts and graphs for better visualization of the results.
11	Download analysis	Analysis generated on all aspects can be combined into a PDF file that is made downloadable to users.

Table 1. *Functional requirements*

3.1.2. Non-functional requirements

Non-functional requirements define the qualities of this system. They are integral to providing a good user experience and ensuring a successful product. Non-functional requirements answer the question "How does this system perform?".

	Function	Description
1	Performance	The system can handle at least 100 simultaneous inputs. Its response time is acceptable. Each service takes less than 10 seconds to generate analysis.
2	Scalability	New services are easy to add without affecting the performance of others.
3	Reliability	There are error handling mechanisms for all algorithms. Informative error messages are provided and data is backed up frequently.

4	Maintainability	The system is modular which makes it easy for updates or bug fixing. Clean code practices are followed.
5	Usability	User interface is intuitive and appealing. It has a simple but coherent design and easy to navigate.

Table 2. *Non-functional requirements*
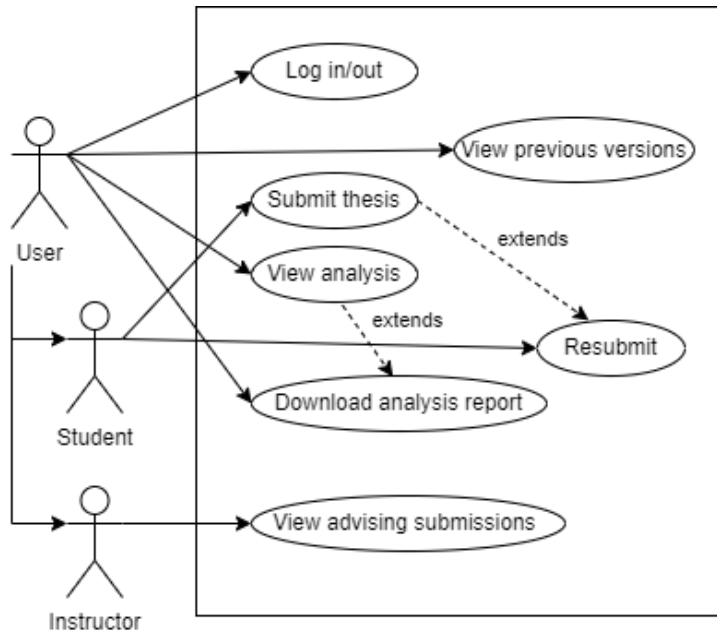
3.1.3. Use case diagram

Figure 13. *Use case diagram*

The main actors of the system are Student and Instructor. They share some actions but also have actions where only they can perform.

Use case	Description
Log in/out	User authenticates with their provided credentials to access the system.
Submit thesis	Student inputs information about their thesis, includes the report and submits it within the deadline.
Resubmit	After submitting, students can resubmit for as many times a they want to.

View previous versions	User can view previous submissions and their analyses if resubmission has been made.
View analysis	User can view analysis generated on a report.
Download analysis report	User can download all analysis for a submission in the form of a PDF file.
View advising submissions	Instructor can view all submissions of which they are the advisor.

Table 3. *Use case description*

3.2. System design

Figure 14 gives a simplified view of the system's architecture. It consists of five main components which are grouped based on their functionality. The client side provides a user interface where inputs are submitted and outputs are displayed. A REST API that receives those requests, processes them and sends messages to the message broker. The message broker routes and delivers messages to the analysis services. These services and REST API have access to the file system.
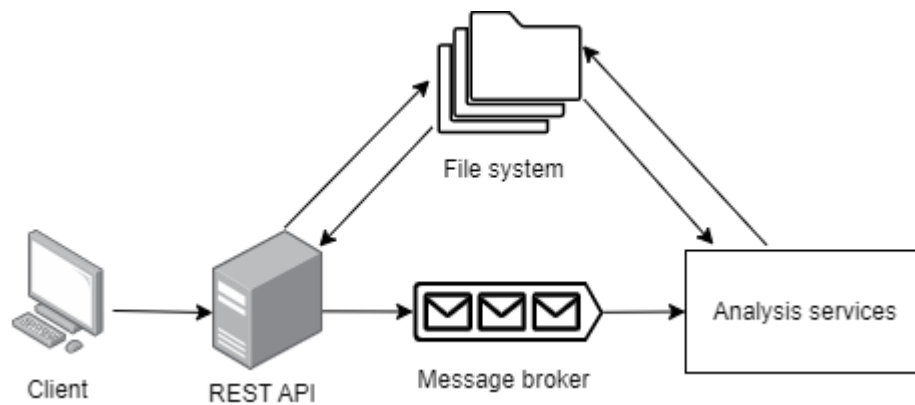

Figure 14. *Simplified system architecture*

3.2.1. Architecture

As given in the functional requirements, the thesis management system should have the following: APIs that can handle user authentication and authorization, analysis services, file system and user interface. Furthermore, these services should be scalable and independent of

26

each other. Therefore, to develop a system with such requirements, the event-driven microservices architecture is applied.

Adopting the microservices architecture solves many potential problems for this system. First, it requires that services are loosely coupled, thus ensuring scalability. This is because each service has its own data store and no direct communication with each other. Therefore, updating, adding and removing a service is unlikely to affect others. Such operations can be performed quickly since only the updated service needs to be redeployed.

Second, the microservices architecture allows for different tech stack to be implemented easily. The services in this system perform diverse tasks, some may benefit from using existing libraries in another language. By leveraging the strengths of different tools and technology when developing, performance can be optimized.

Since services cannot communicate directly to ensure loose coupling, another mechanism must be in use to allow for the transfer of information. Therefore, the event-driven pattern is implemented to provide an efficient yet indirect mean of messaging, the message broker.
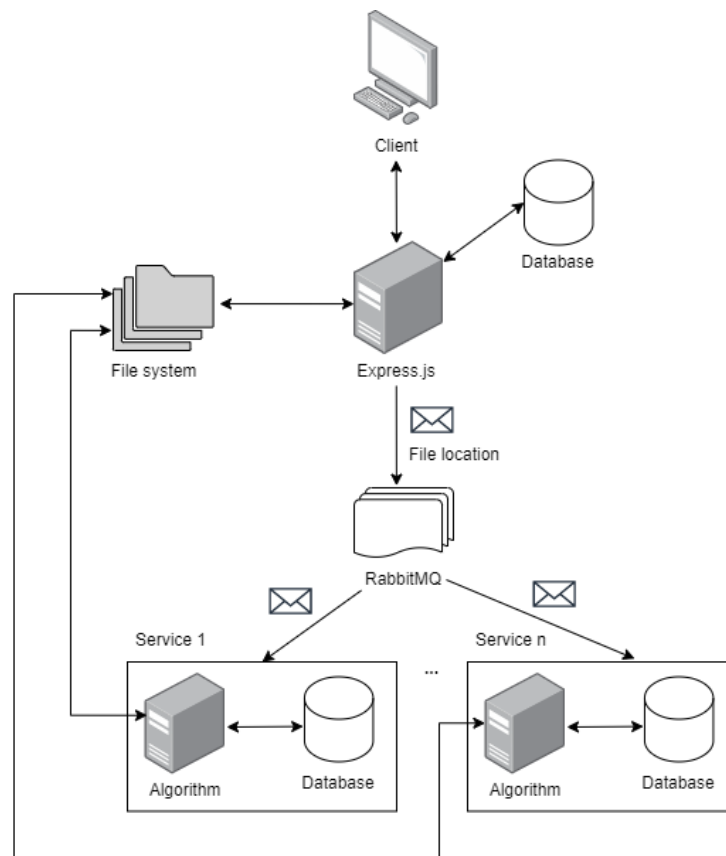

Figure 15. *The system's event-driven microservices architecture*
27

The functional and non-functional requirements of this system can be met when combining these two approaches. Figure 15 gives the result, a thesis management system with independent and modular services that communicate with each other through a message broker, RabbitMQ. These services can access the file system to retrieve thesis reports and generate analyses. The REST API, Express.js, is able to retrieve them and display to the client. All of these operations happen asynchronously and without knowledge of one another.

3.2.2. Database design

Data management is necessary in every system, whether simple or complex. Thus, designing a well-structured database lays the foundation for a robust application. Applying an appropriate database model ensures the system's functionality and optimizes performance.

A database model defines how data is organized. Relational, document and key-value are a few of the most popularly used models. In this system, there are only a few number of relations. However, they are all associated with each other in some form of relationship. According to the functional requirements, queries will often be made that span many tables. For example, when an instructor wants to view all versions of thesis submission given the student's name. Therefore, to accommodate complex queries, the relational database model is used for this system.

There are two main database structures used in this application. One for the REST APIs (Express.js) and one for the analysis services.
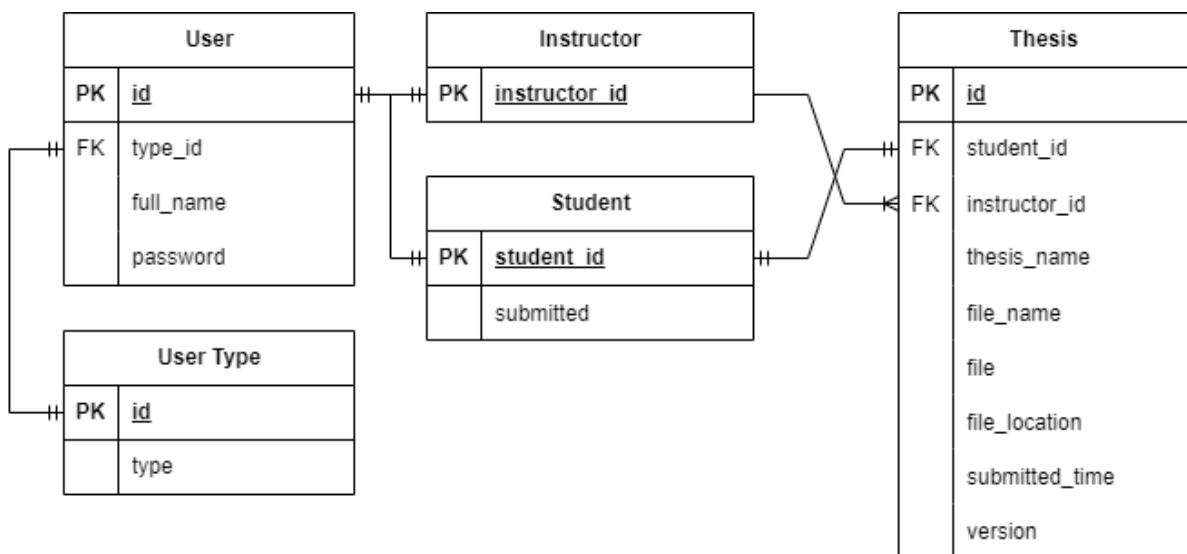

Figure 16. *Express.js database*

28

The first includes the relations User, User Types, Student, Instructor, and Thesis. The User relation contains all necessary information about the user of the system. There are two types of users, defined in the User Type table: Student and Instructor. Each of these types will have access to certain features of the application. The Thesis table contains all relevant information about a student's thesis. When a student resubmits a new document, the version number will be incremented and thus, it will considered as a new tuple.

The second layout only has one relation which is Output. This table records information about the generated analysis from each service. A service type attribute is not necessary since this is included in the file_location attribute and can be obtained via processing the string.

Figure 17. *Analysis service database*

3.2.3. User interface design

A well-designed user interface can greatly enhance user experience and improve usability. Thus, this system provides a simple but intuitive web application by implementing responsive design, visual cues and feedback messages in combination with contrasting colors and purposeful layouts.

Figure 18. *Button changes color on hover*

Figure 19. *Concise and informative feedback messages*

Figure 20. *Functional UI on different devices*

3.3. Tools

One of the challenging parts of designing a system is choosing the appropriate tools and tech stack to implement. The tools listed below are the ones that maximize benefits for this system and minimize trade-offs.

3.3.1. Express.js

Express.js is a Node.js backend framework that is suitable for a wide range of applications. Its asynchronous and non-blocking feature makes it ideal for handling high concurrency and heavy load, such as multiple simultaneous file submissions. It is used in this application to develop REST APIs that perform the tasks of authentication, file upload, file retrieval and polling for service's completion.

Figure 21. *Express.js*

3.3.2. React.js

The client side of this application is developed using React. React is a JavaScript library that allows for building interactive and appealing UI with reusable components. React was chosen because of its extensive open-source ecosystem. Furthermore, the React community is active and constantly updating and improving packages.
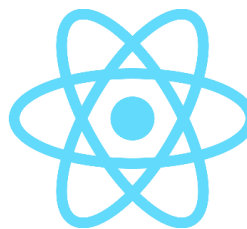

Figure 22. *React*

3.3.3. PostgreSQL

PostgreSQL is the relational database management system used for this application. It supports a large range of data types, from arrays to JSON. PostgreSQL is a suitable choice for developing a robust thesis management system because it allows for concurrent inserts and updates while ensuring they are ACID compliant.

31

Figure 23. *PostgreSQL*

3.3.4. RabbitMQ

RabbitMQ is the choice of message broker for facilitating asynchronous inter-service communication in this system. RabbitMQ allows message persistence to disk which ensures its recovery in the event of system failure. It also guarantees message delivery to the correct recipient regardless of its availability. RabbitMQ was chosen over Apache Kafka and other alternatives due to its easier setup and configuration. Furthermore, RabbitMQ excels in traditional messaging where strict ordering and delivery of messages are required.

Figure 24. *RabbitMQ*

3.3.5. Python libraries

```python
import os
import openai

openai.api_key = os.getenv("OPENAI_API_KEY")

response = openai.Completion.create(
  model="text-davinci-003",
  prompt="Correct this to standard English:\n\nShe no went to the market.",
  temperature=0,
  max_tokens=60,
  top_p=1.0,
  frequency_penalty=0.0,
  presence_penalty=0.0
)
```

Figure 25. *OpenAI's grammar correction API example in Python*

The services that analyze thesis documents are written using Python because of its extensive amount of libraries for NLP and PDF processing. pdfplumber, Sumy and OpenAI are a few of the libraries used. Python was also chosen because of its clean syntax and readibility which is especially beneficial when writing complex algorithms.

3.3.6.   Docker

This system uses Docker, a platform that allows for containerization, to automate the packaging and deployment of services. Docker lets an application run consistently across environments by encapsulating it along with its dependencies, libraries and configurations. Docker provides Docker Hub, a registry for sharing and distributing container images. This allows for quick deployment of an application by pushing its image to the registry, pulling it into the server and running the container.
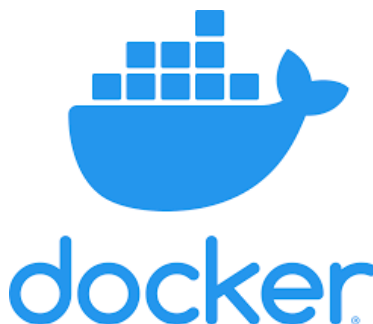

Figure 26. *Docker*

CHAPTER 4

IMPLEMENTATION AND RESULTS

4.1. Implementation

4.1.1. Overview

Each tool mentioned in the previous chapter is provided in the table below. The libraries and packages used are also included with their descriptions.

Type	Tool	Description
Event-driven	RabbitMQ	Message broker
Database	PostgreSQL	RDBMS
Containerization	Docker Desktop	Docker GUI
REST API	Express.js	Backend framework
	Node.js	JavaScript runtime environment
	amqplib	AMQP implementation for Node.js
	Sequelize	Node.js ORM
Document processing	Apache Tika	PDF text extraction
	pdfplumber, PyPDF2	PDF format detection
	OpenAI	AI model
	sumy	Text summarization
	pika	AMQP implementation for Python
Frontend	React	Frontend library
	SCSS	Preprocessor scripting language
	React Context API	State management

	react-router-dom	Routing package
	D3.js	JavaScript library

Table 4. *Tech stack*

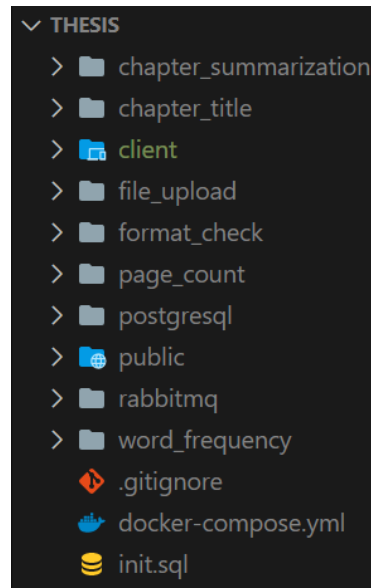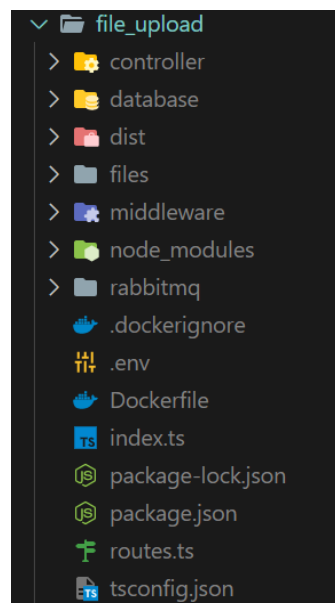**Figure 27.** *Project's folder structure*

Figure 28. *Express.js folder structure*

REST APIs are implemented using the Express.js framework and its model-view-controller architecture. Model is an abstraction of a relation in the database. It is defined as a

35

Sequelize instance and stored in the "database" folder. Views are the responses sent to the client side. Controllers are where the main logic is located, stored in the "controller" folder. Additionally, the middleware pattern is implemented to validate and preprocess oncoming requests before passing them on to controllers. Thesis reports submitted by students are stored in "files".

```typescript
const uploadThesis = async (req: Request, res: Response) => {
    if (!req.file) {
        res.status(400).send("Missing file.");
    }
    else {
        const instructor: Model | null = await User.findOne({where: {full_name: req.body.advisor_name}});
        const student: Model | null = await Student.findOne({where: {student_id: req.body.student_id.toUpperCase()}});

        const data: {[key: string]: any} = {
            id: res.locals.id,
            student_id: req.body.student_id.toUpperCase(),
            instructor_id: instructor?.dataValues.id,
            thesis_name: req.body.thesis_name,
            file_name: res.locals.file_name,
            file_location: res.locals.file_location,
            submitted_time: (new Date()).toString(),
            version: res.locals.version
        }

        const instance = Thesis.create(data as any).then(async response => {
            const producer: Producer = new Producer();
            await producer.publishMessage(res.locals.file_location + "/" + res.locals.file_name);

            if (student?.dataValues.has_submitted) {
                res.status(200).send(data.id);
            }
            else {
                student?.set({has_submitted: true});
```

Figure 29. *Upload thesis controller*

4.1.3. Document processing

All document processing services follow the same template to promote scalability.

> 📁 chapter_summarization
> 📁 chapter_title
> 📁 client
> 📁 file_upload
> 📁 format_check
> 📁 page_count
> 📁 postgresql
> 🌐 public
> 📁 rabbitmq
> 📁 word_frequency

Figure 30. *Document processing services (highlighted)*

36

Each service consists of consumer.py – an event-driven consumer configuration file, database.py – a database configuration file, main.py – the point where program excution begins, processor.py – where the analysis algorithm is located, requirements.txt – a text file that defines the imported packages, template.txt – the guidelines to be compared, Dockerfile – a text file that contains commands to build an image of the service and output – the folder where generated analysis is stored.
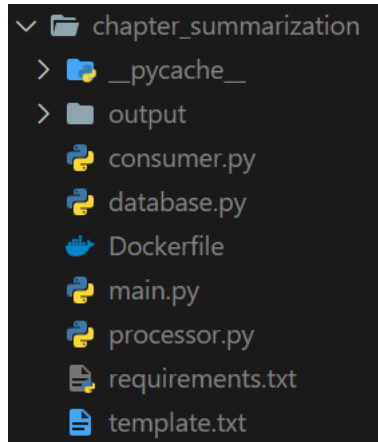

Figure 31. *Folder structure of a document processing service*

```python
def split_into_chunks(text, max=4000):
    chunks = [[]]
    chunk_total_words = 0

    nlp = spacy.load("en_core_web_sm")

    for sentence in nlp(text).sents:
        chunk_total_words += len(sentence.text.split(" "))

        if chunk_total_words > 10000:
            chunks.append([])
            chunk_total_words = len(sentence.text.split(" "))

        chunks[len(chunks)-1].append(sentence.text)

    return chunks
```

Figure 32. *Chunk splitting algorithm for chapter summarization service*

4.1.4. Frontend

The frontend folder consists of the following subfolders. "asset" stores the images used such as International Unversity logo and backgrounds. Components stored in "components" are used to develop "pages". "router" includes the client-side routing algorithm that directs

users and displays the correct web pages. "routes" define PrivateRoute and ProtectedRoute to limit access to different user type.

```
const handleClickOutside = (e) => {
    if (active && dropdown.current && !dropdown.current.contains(e.target)) {
        setActive(false);
    }
}

const handleSubmitAgain = () => {
    setHasSubmitted(false);
}

const handleViewReport = () => {
    axios.post(process.env.REACT_APP_BACKEND_HOST + "get-all-submissions", {student_id: user.user_id
        headers: {
            "Content-Type": "application/json"
        }
    }).then(response => {
        const submissions = response.data;
        const newestSubmission = submissions[submissions.length - 1];
        navigate(`/report/${newestSubmission.id}`);
    }).catch(error => {
        console.log(error.message);
    });
}

useEffect(() => {
    if (localStorage.has_submitted === "true") {
        setHasSubmitted(true);
    }
```
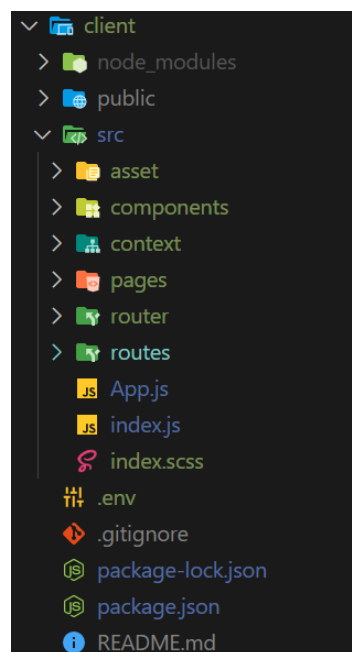
Figure 33. *Student homepage*

```
∨ 📁 client
  > 📁 node_modules
  > 📁 public
  ∨ 📁 src
      > 📁 asset
      > 📁 components
      > 📁 context
      > 📁 pages
      > 📁 router
      > 📁 routes
        JS App.js
        JS index.js
        S index.scss
    🎚 .env
    ◆ .gitignore
    JS package-lock.json
    JS package.json
    ℹ README.md
```

Figure 34. *React folder structure*

38

4.1.5. Event-driven

Both Express.js and the document processing services use the AMQP implementation of RabbitMQ. RabbitMQ is built into Docker image instance using a Dockerfile.

```
FROM rabbitmq:3-management
RUN apt-get update
RUN apt-get install -y curl
EXPOSE 5672 15672
```

Figure 35. *RabbitMQ Dockerfile*

```typescript
export class Producer {
    channel: Channel;
    exchangeName: string;

    async init() {
        const connection: Connection = await amqp.connect(process.env.RABBITMQ_HOST!);
        this.channel = await connection.createChannel();
        this.exchangeName = process.env.EXCHANGE_NAME!;

        console.log("\nProducer connected to " + this.exchangeName);
    }

    async publishMessage(message: string) {
        if (this.channel === undefined) {
            await this.init();
        }

        this.channel.assertExchange(this.exchangeName, "fanout", {durable: true});
        this.channel.publish(this.exchangeName, "", Buffer.from(message));

        console.log("\nSent message: " + message);
    }
}
```

Figure 36. *Producer class in Express.js using amqplib*

```python
def consume_message(self):
    channel = self.connection.channel()

    channel.exchange_declare(exchange=self.upload_file_exchange, exchange_type="fanout",

    channel.queue_declare(queue=self.upload_file_queue, durable=True)
    channel.queue_declare(queue=self.service_complete_queue, durable=True)

    channel.queue_bind(exchange=self.upload_file_exchange, queue=self.upload_file_queue)

    def callback(ch, method, properties, body): ...

    channel.basic_consume(queue=self.upload_file_queue,
                          auto_ack=True,
                          on_message_callback=callback)
    channel.start_consuming()
```

Figure 37. *Consumer in Python using pika*

4.1.6. Database

The PostgreSQL database is containerized for ease of deployment and testing on different environments.

Figure 38. *PostgreSQL Dockerfile*

Figure 39. *Student model definition in Express.js*

Figure 40. *Populate database with init.sql*

40

4.1.7. Containerization

A Docker compose file is used to define services, their dependencies and environment variables. This file allows for the convenient start and stop of the entire application as a container.

```yaml
services:
  postgresql:
    container_name: "postgresql"
    build: postgresql/.
    image: postgresql
    restart: always
    ports:
      - "5432:5432"
    volumes:
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql
    environment:
      POSTGRES_USER: "postgres"
      PG_USER: "postgres"
      POSTGRES_PASSWORD: "postgres"
      POSTGRES_DB: "thesis_upload"
    healthcheck:
      test: ["CMD-SHELL", "sh -c 'pg_isready -U ${POSTGRES_USER} -d ${POSTGRES_DB}'"]
      interval: 10s
      timeout: 5s
      retries: 3
  file_upload:
    container_name: "file_upload"
    build: file_upload/.
    image: "file_upload"
    restart: always
    ports:
      - "5000:5000"
    depends_on:
      postgresql:
```

Figure 41. *Project's Docker compose file*

4.2. Results

Running the command "Docker compose up" builds all required images. On the local environment, only the database (postgresql) and message broker (rabbitmq) instances are required to be run. For Express.js and frontend, run "npm start" to start the web application. Run "python main.py" for the remaining analysis services. The web application is now functional and can be accessed via the port 3000.

The first page that is displayed is the authentication page which asks for user's credentials.

41

Figure 42. *Log in as a student*

After logging in as a student, user will be redirected to the homepage. Here, if they have never submitted their report before, a thesis submission form will be displayed. The form contains inputs for the thesis title, a dropdown with a list of advisors to select and a file submission mechanism that only accepts PDF files. Furthermore, additional information such as the deadline is also included to remind students.

Figure 43. *Thesis submission form*

A Research on Thesis Report Management and
Evaluation with Event-driven Architecture

by Ung Thu Ha - ITITIU19114

Submitted on 12:30:59 PM 26/05/2023

Download file   Resubmit

Page count

Keywords

Chapter title

Format check
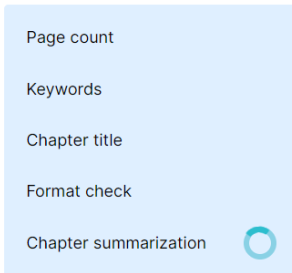
Chapter summarization

Figure 44. *Polling for service completion*

After submitting, the analysis page will appear. Analysis options are given on the left where students can check their completion status. Services that have finished generating a feedback will no longer have a spinner animation present and can be selected.

A Research on Thesis Report Management and
Evaluation with Event-driven Architecture
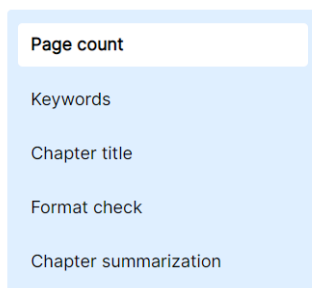
by Ung Thu Ha - ITITIU19114

Submitted on 12:30:59 PM 26/05/2023

Download file   Download report   Resubmit

Page count

Keywords

Chapter title

Format check

Chapter summarization

Page count required for each chapter:

TOTAL: 40

CHAPTER 1: 10

CHAPTER 2: 5

CHAPTER 3: 10

CHAPTER 4: 5

CHAPTER 5: 3

CHAPTER 6: 3

REFERENCES: 1

Figure 45. *Page count report*

Selecting each service gives the result which contains an analysis of the report on that specific aspect. Data visualization is incorporated in some services to provide more insights in the form of graphs and charts.

Results:
Total page count: Not enough
CHAPTER 1: Not enough
CHAPTER 2: Not enough
CHAPTER 3: Not enough
CHAPTER 4: Not enough
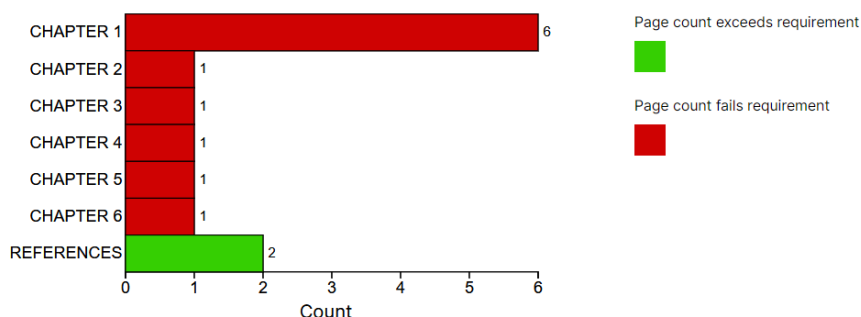CHAPTER 5: Not enough
CHAPTER 6: Not enough
REFERENCES: Exceeded

Figure 46. *Page count analysis*

In another service such as chapter summarization, a coherent dropdown list of each chapter's content is provided.

Page count

Keywords

Chapter title

Format check

Chapter summarization

Chapter 1

CHAPTER 1 INTRODUCTION

Background There are a large number of theses which an instructor has to examine and grade each semester. Some of them may be formatted incorrectly according to the guideline provided by the department, or not well-worded such as the poor use of English grammar and typographical errors. Flaws in the document's layout may range anywhere from incorrect margin or font family to a lack of content numbering and structure. Instructors can get their workload reduced by reading shorter paragraphs provided as a summary by the software.

Problem Statement An average of 50 theses are submitted each semester in each department. Assuming they are supervised equally between instructors, then each instructor would have more or less 10 theses to check. Each document consists of at least 50 pages that detail a specific complex topic. By implementing NLP techniques to process text documents, services included in the software are efficient and produce reports of good accuracy.

1.3. Scope and Objectives This research studies the performance and effects of the thesis checking software on students and instructors of the Department of Computer Science and Engineering at International University. Therefore, the scope of this project will be limited to fewer than 10 submissions or documents per instructor, conducted over a period of one month. 10

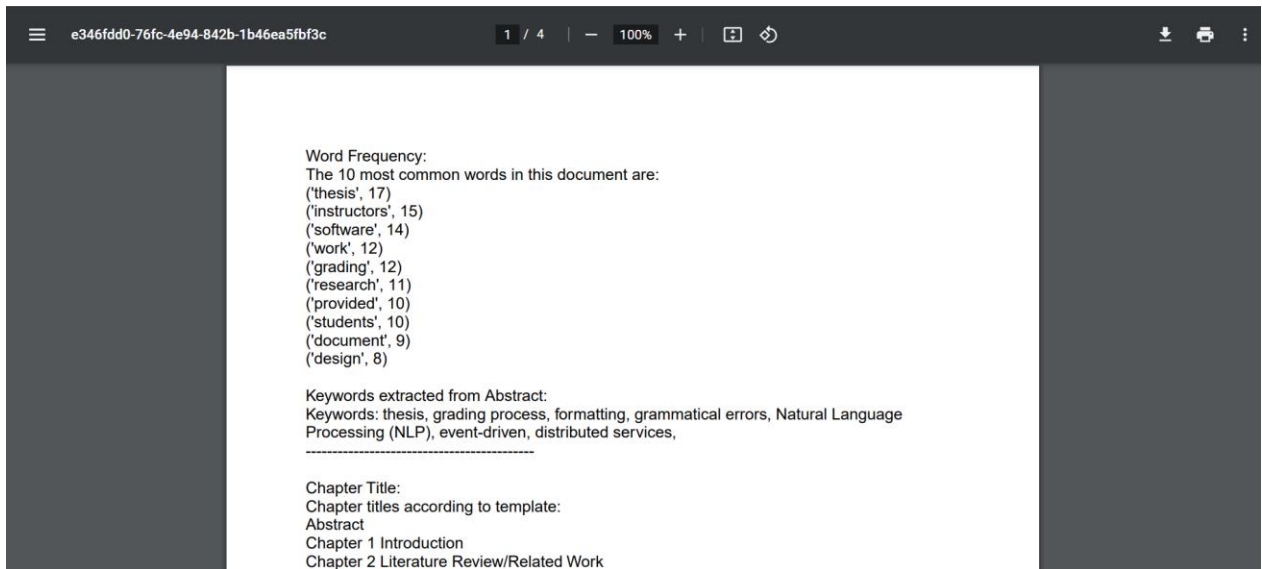**Figure 47.** *Chapter summarization report*

Figure 48. *Getting the full report*

Furthermore, students can download the full report as a PDF file by selecting "Download report" in the analysis page.

Clicking on the school's logo will redirect students back to the homepage. If they have already submitted their report at least once, which in this case, the homepage will no longer display the initial thesis submission form. However, students can still choose "Submit again".

Figure 49. *Homepage when students have already submitted*

After resubmitting, the flow remains the same. Students will be redirected to the analysis page of the newest submission. This time, there will be a version history at the end of the page containing previous versions. These versions are still accessible and downloadable.

A Research on Reconstruction and Visualization of 3D Objects on VR Application

by Ung Thu Ha - ITITIU19114

Submitted on 12:35:53 PM 26/05/2023

Download file   Resubmit

Page count

Keywords

Chapter title

Format check

Chapter summarization

Previous submissions

A Research on Thesis Report Management and Evaluation with Event-driven Architecture

Submitted on 12:30:59 PM 26/05/2023

Figure 50. *After resubmitting, version history appears*

Instructors have a different user flow than students. This is because they use this system to grade reports, not to submit.

International University

Thesis Management Platform

User ID   ititeach001

Password   ••••••••

Sign in

Figure 51. *Log in as an instructor*

After logging in as an instructor, their homepage will display all of the newest submissions of the theses that they supervise. If any of these submissions have previous versions, they are not displayed here but only on the analysis page.

Student Submissions

Deadline: 25/06/2023

2 submissions

A Research on Reconstruction and Visualization of 3D Objects on VR Application

Dinh Bao Duy - ITITIU19107

26/05/2023

A Research on Reconstruction and Visualization of 3D Objects on VR Application

Ung Thu Ha - ITITIU19114

26/05/2023

Figure 52. *List of student submissions by instructor*

CHAPTER 5

DISCUSSION AND EVALUATION

5.1. Discussion

From the implementation section above, the system is shown to have met its basic functional requirements.

This includes authentication, which allows for the separate logging in of different user types. Students and instructors are redirected to different homepages. A thesis submission form is displayed to students whereas instructors are presented with a list of submissions that they advise.

Some report generation services display their information differently. The page count service checks the number of pages each chapter has and compares them to the provided guidelines. To let users better visualize, a chart is included where the number of pages are plotted and color coded according to whether they meet the requirements. On the other hand, the chapter summarization service, which summarizes the content of each chapter, displays its analysis through expandable sections.

Additionally, the version history feature allows both students and instructors to view the previously submitted documents and their reports. This aids students in revising and choosing the best version for their final submission. Instructors also receive insights into what the student has changed based on the differences in the reports.

5.2. Evaluation

The thesis management system performs relatively well under 10 simultaneous inputs. Given that each submission takes at least 5 seconds for all services to complete, the 10th input might need to wait 1 minute before receiving the results. However, in reality, only at most 10 submissions are made per day during the thesis submission process. This still presents an issue that could be improved upon if other Departments were to adopt the system and the number of inputs increase.

Due to limited time, testing has not been conducted sufficiently to assess performance and memory capacity of the system. Depending on the resources available in the host's machine, in this case, 83GB will be the upper limit disk space for the Docker container. The size limit for each thesis document is 20MB. Considering that a student might submit for a total of at least 3 times, the system will then store 65MB of content for each student and their generated analysis. Dividing 83GB by 65MB gives 1277, the maximum number of 3-time submissions per student the system can handle. An average of 30 students submit their thesis each semester in every Department. This accounts to a total of 360 students for the entire university. Therefore, the current file system has enough of the capacity to handle submissions from all Departments.

CHAPTER 6
CONCLUSION AND FUTURE WORK

6.1.   Conclusion

Every semester, instructors are tasked with grading dozens of thesis submissions that are lengthy and complex. Some documents have typographical or layout errors that cause its content to become incoherent. These flaws hinder the grading process as they require more time to examine, thus increasing the already large workload on instructors.

On the other hand, many students are not aware of the mistakes they made in their reports. There is also no feedback mechanism that informs them of those shortcomings. Studies have shown that when students receive sufficient feedback, the quality of their work improves. Furthermore, grading a well-written document would require less mental effort and time for instructors.

A lack of a thesis analysis and feedback service has become the motivation for this study. Thus, it develops a thesis management system that allows for the submission, storing and analysis of thesis documents into comprehensive reports. The ultimate goal of this application is to encourage students to revise their submission and reduce the workload for instructors.

By implementing an event-driven microservices architecture, a scalable and responsive thesis management system is achieved. Each service has their own database to promote loose coupling and access to the file system for storage and retrieval. By applying the same template to all microservices with only a few modifications, ease of scalability is ensured. Additional features are implemented to provide more robust functionalities for the system: compiling all reports into a downloadable PDF, interactive version history and using data visualization to produce charts on the analyses.

However, the system is without shortcomings due to limitations in time, knowledge and resources. First, the testing conducted is insufficient and has not accounted for the scope of 100 to 1000 simultaneous inputs. Considering that this system may be used by other Departments, it is important to guarantee that it at least functions normally and there are no performance bottlenecks. Limited time to research and develop is another factor that reduces the quality of the system. Due

to inefficient time management, some aspects of the application are rushed and some features could not be implemented such as manual feedback from instructors and admin functionality.

In conclusion, the thesis management system meets the basic specified requirements of providing feedback to students and insight to instructors. Reports generated can inform students of flaws they are not aware of, thus helping them revise and improve the quality of their thesis. Instructors can benefit from grading more coherent documents with fewer mistakes. This study introduces many opportunities for more comprehensive services to be implemented. Ultimately, it serves to enhance academic integrity, quality and experience.

6.2.   Future work

To address the limitations mentioned above, additional time will be spent redefining requirements, optimizing services and testing the system.

First, more features can be added to enrich the application. From changes to the user interface design to services that analyze more complex aspects of the thesis. Thus, this requires updating the project's requirements and reassessment of the planning stage.

The current algorithms used in both REST APIs and analysis services, are relatively slow and inefficient. Worker threads in Node.js can be used to improve file upload speed by splitting the file into chunks. Report generation can be optimized by reducing unnecessary imported packages or switching to libraries that provide more efficient APIs. Furthermore, horizontal scaling can be implemented to improve processing speed.

Finally, more testing is needed to ensure the system's functionality, especially with large amounts of concurrent inputs.

REFERENCES

Wästlund E., Norlander T. & Archer T. (2008). The effect of page layout on mental workload: A dual-task experiment.

Hojjati N. & Muniandy B. (2014). The Effects of Font Type and Spacing of Text for Online Readability and Performance.

Lipnevich, A. A., & Smith, J. K. (2009). Effects of differential feedback on students' examination performance. Journal of Experimental Psychology: Applied, 15(4), 319–333.

Dunham S., Lee E. & Persky, A. M. (2020). The Psychology of Following Instructions and Its Implications. American Journal of Pharmaceutical Education August 2020, 84 (8).

APPENDIX

VIETNAM NATIONAL UNIVERSITY OF HOCHIMINH CITY

THE INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

A SYSTEM FOR THESIS REPORT MANAGEMENT AND EVALUATION WITH EVENT-DRIVEN ARCHITECTURE

By

Ung Thu Hà