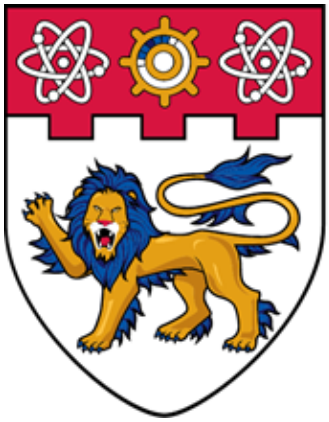


NANYANG TECHNOLOGICAL UNIVERSITY

SINGAPORE

NANYANG TECHNOLOGICAL UNIVERSITY

A Web Interface for Aspect-based Sentiment Analysis through EDU-level Attentions

by

Jeremy Teo Tze Png

(U1921446L)

Supervisor

Associate Professor Sun Aixin

Examiner

Mr Oh Hong Lye

Acknowledgements

I would like to take this momentous opportunity to express my deepest gratitude and utmost appreciation to Dr Sun Aixin (Associate Professor at the School of Computer Engineering, Nanyang Technological University, Singapore) for providing me with the opportunity to be part of this exceptional project.

A/Prof Sun Aixin has contributed significantly to the evolution of the ideas and implementations throughout the project, and the success of the EDU-Web Interface would not have been possible without his guidance, support, and expertise. I am truly grateful for his unwavering dedication and invaluable mentorship throughout the development process.

In addition, I would also like to extend my sincere thanks to Lin Ting, a PhD student who has been tremendously helpful and supportive during the course of this project. Lin Ting's assistance, insights, and expertise have greatly contributed to the overall success of the project.

I am truly grateful for the contributions and support from both A/Prof Sun Aixin and Lin Ting, as their guidance, encouragement, and motivation have been instrumental in shaping the success of this project.

Table of Contents

Table of Figures

1. Introduction

1.1. Abstract

In linguistics, a sentence can have multiple aspects which may each carry their individual sentiments. When the aspects in these sentences carry differing sentiment polarities, the traditional sentiment analysis model's accuracy is often adversely affected. Often, such sentences that carry multiple aspects are expressed through multiple clauses with each clause carrying a unitary sentiment toward a single aspect. In this work, we aim to introduce a web interface that supports considering clause boundaries, which are formally known as elementary discourse units (EDUs), in sentences. This allows our system to leverage an EDU-Attention model, based on the Attention Mechanism which identifies the sentiment of each clause in the sentence. Our system enables users to observe the overall sentiments toward various identified aspects from a group of texts, specifically reviews (Tang, Qin, & Liu, 2016).

1.1. Background and Motivation

In the domain of linguistics, a language can be characterized as an established system of rules and conventions utilized by individuals for the communication and dissemination of information in their daily life. As research in Linguistics and Artificial Intelligence continues to progress, the intersection of the two disciplines gave rise to the development of Natural Language Processing (NLP). NLP constitutes a prominent sub-discipline within artificial intelligence, encompassing the investigation and manipulation of human language by computational algorithms. The primary objective of NLP is to facilitate the understanding and processing of natural language text segments by such algorithms, ultimately enabling them to accomplish designated tasks (Jurafsky & Martin, 2022).

One particular task within the realm of NLP is Sentiment Analysis. Often referred to as opinion mining, Sentiment Analysis constitutes an expanding subfield in NLP, primarily focusing on the examination of individuals' opinions, sentiments, evaluations, attitudes, and emotions concerning entities and their attributes as expressed in predominantly unstructured textual data. The information gleaned from individuals' sentiments or opinions proves to be invaluable for obtaining insights into customer sentiments, brand perception, and market trends (Pang & Lee, 2008). In today's internet age, personal opinions are ubiquitous, with a substantial volume of reviews and comments accessible on diverse websites, including IMDb.com, Amazon.com, and Yelp.com. Concurrently, with the proliferation of social networking sites, such as social media platforms and blogs, the volume of comments and reviews related to everyday activities has experienced exponential growth. Consequently, the significance of Sentiment Analysis in offering stakeholders a more profound understanding of their customers (Tang, Qin, & Liu, 2016) has increasingly become advantageous for corporations, governments, and individuals who base their decisions on these opinions. The opinions embedded within these texts are employed for purposes such as assessing the reputation of films, enhancing products, and offering recommendations for personal items, including books or restaurants.

Over the past decade, researchers within the sphere of Sentiment Analysis have dedicated their efforts to quantifying dynamic sentiments present in unstructured text utilizing various NLP

methodologies and techniques (Dave, Lawrence, & Pennock, 2003). One specific subdomain of Sentiment Analysis that has garnered recent interest is Aspect-Based Sentiment Analysis (ABSA). ABSA entails the identification and extraction of opinions pertaining to particular aspects or features of an entity (e.g. evaluations of a restaurant). This approach proves especially beneficial when analysing comments and reviews on social networking sites, as these often exhibit varying sentiment polarities for distinct aspects within a single sentence.

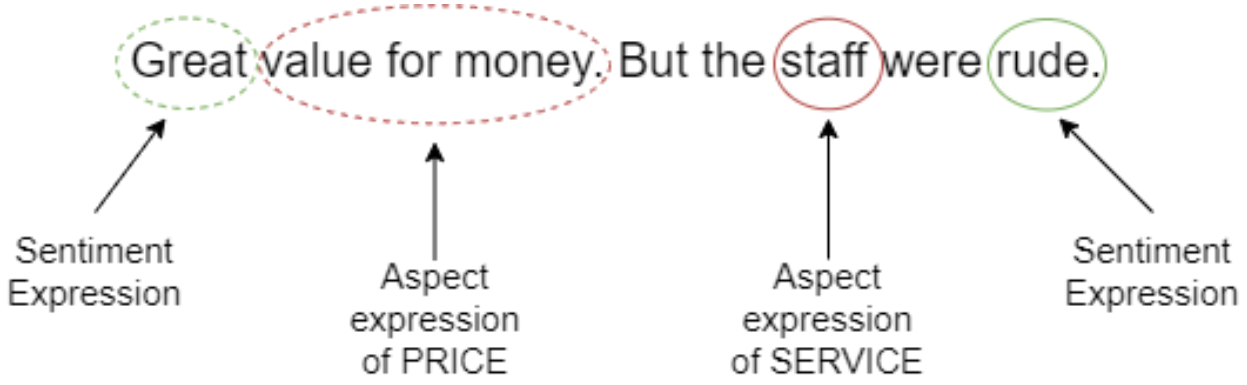


Figure 1: An example of aspect-based sentiment analysis (ABSA).

For example, Figure 1 illustrates a sample restaurant review that discusses two distinct target aspects and their corresponding sentiment expressions. In this instance, one aspect identified in the review is PRICE, accompanied by a sentiment expression commenting on the restaurant's item pricing. The other aspect, SERVICE, is associated with a sentiment expression addressing the quality of service delivered by the restaurant staff. It is noteworthy that these two aspects convey divergent sentiments toward distinct facets of the restaurant experience, with a positive sentiment toward the PRICE aspect and a negative sentiment toward the SERVICE aspect. This scenario exemplifies a typical review of goods or services commonly found online today. Generally, users evaluate various components of a product or service before making any decisions. As a result, ABSA provides a more accurate comprehension of user expectations compared to document or sentence-level sentiment analysis (Mowlaei, Saniee Abadeh, & Keshavarz, 2020).

Traditionally, ABSA has been tackled by partitioning the NLP task into smaller subtasks, which encompass initially identifying aspects within a sentence and subsequently predicting the sentiment polarities of the recognized aspects. In this study, however, our focus will be solely on the latter subtask, specifically the prediction of sentiment polarities for pre-identified aspects,

also referred to as Aspect Category Sentiment Analysis (ACSA). In ACSA, the aspects are supplied, and the objective is to predict the corresponding sentiment for each provided aspect.

As such, the academic paper upon which this project is founded—Aspect-based Sentiment Analysis through EDU-level Attentions—identified that sentences containing multiple aspects and sentiments can also be classified as comprising multiple elementary discourse units (EDUs). Each EDU represents a clause-like grammatical unit that typically conveys a unitary sentiment toward a single aspect. Therefore, the paper proposed an innovative approach, consisting of an EDU-Attention model, to address the ACSA task of predicting the aspect-specific representation of EDUs within a sentence.

The introduction of this ground-breaking EDU-Attention model has underscored the necessity for a web interface capable of showcasing the research findings. Such an interface would provide users with visibility into the different EDUs of a single sentence or review, as well as facilitate interactive exploration of entire datasets comprising sentences and their respective sentiments toward specific aspects.

1.2. Project Objective

In this final year project, the student will be involved in developing a demo system with a web interface and the relevant APIs related to visualising Aspect-based Sentiment Analysis through an Elementary Discourse Units Attention Model. The purpose of this project is to allow users to have a better understanding of the inner working of an EDU-level attention model in learning aspect-specific representation of EDUs in a sentence through a web visualisation. The web visualisation boasts to provide three key functionalities:

1. Visibility of sentiments toward the different aspects: Provide a holistic view of all the sentiments toward the various identified aspects in a dashboard derived from the EDU-level attention model.
2. The interactivity of the dashboard

3. Identification of EDUs in a sentence and the corresponding sentiments:

1.3. Limitations

This project has some limitations. Firstly, the project focuses solely on analyzing restaurant reviews as the dataset and model used are specific to this data type. Nevertheless, this project can be expanded in the future to accommodate different data types. Secondly, more features that were proposed in the design phase have yet to be implemented in the interface, including extractive summaries, abstractive summarization and real-time review website integration. Finally, this web interface may not be friendly to mobile phone users as it is currently optimized for web access. These features may be further explored in the future to integrate the developed APIs with a mobile interface.

1.4. Project Scope

Hence, the score of this FYP is as follows:

1. Conduct FYP Project planning to achieve the milestones of this project.
2. Perform requirement elicitation for this project.
3. Develop the backend and frontend for the web interface.

2. Project Schedule

2.1. Work Breakdown

This section aims to provide a clear understanding of the various tasks/activities, their descriptions, the estimated efforts and their dependencies to facilitate effective project management and resource allocation.

	Work (Activity/Task)	**Description**	**Effort Estimation (Man-days)**	**Dependencies**
1.1	Requirement Elicitation	Identify the project stakeholders, which include the Final Year Project (FYP) Supervisor and the Doctoral Student and collect requirements from them through meetings.	8	NIL
1.2	Requirements Analysis	Analyze the collected requirements and prioritize them, taking into account project constraints and considerations to ensure a well-balanced and feasible development plan.	8	1.1
1.3	Requirements Documentation	Create requirement specification documentation, including OpenAPI Specifications, to comprehensively outline the project's requirements and API details.	4	1.2

2.1	Backend Architecture Design	Design the backend server architecture, encompassing the different microservices and their corresponding API structures to ensure a well-developed system.	10	1.3
2.2	Backend Server Implementation	Develop the backend server following the proposed backend architecture design, ensuring a robust and efficient implementation that aligns with the project requirements.	35	2.1
3.1	Frontend Design	Design frontend interface, including mockups, wireframes, and UI/UX components.	15	1.3
3.2	Frontend Interface Implementation	Integrate the frontend application with the backend server and create the necessary React components in accordance with the design, ensuring seamless communication and a cohesive user experience.	30	3.1
4.1	Requirement Amendments	Evaluate and revise the requirements based on feedback from stakeholders and any project constraints,	15	2.2, 3.2

		ensuring that the project remains aligned with its objectives and adapts to any changes in scope or priorities.		
4.2	Feature Refinement	Fine-tune and refine features by taking into account user feedback and addressing any technical limitations, ensuring that the end product is not only user-friendly but also performs efficiently.	20	4.1
5.1	Meetings	Arrange meetings with the FYP supervisor and the Doctoral Student to seek their guidance, clarify any uncertainties, and ensure the project remains on track and aligned with their expectations.	10	All Tasks
6.1	Interim Report Writing	Write and review an interim report that outlines the project's progress to date, providing insights into the accomplishments, challenges faced, and the direction moving forward.	3	1.3, 2.2, 3.2
6.2	Final Report Writing and submission	Write and review the final report, incorporating a comprehensive account of	21	1.3, 2.2, 3.2, 5.1, 6.1

		the project's research, implementations, considerations, and lessons learned, providing a thorough reflection on the entire development process.		

2.2. Risk Management

In this section, a comprehensive risk management table is introduced to systematically identify and address potential risks associated with the development of the project. The table assigns scores to the risks based on their probability, impact, and risk level while also offering mitigation strategies to be employed if a risk materializes. By serving as an essential instrument for risk monitoring and control throughout the development lifecycle, this risk management table contributes to a more robust and proactive project management approach.

Risk Description	**Risk Mitigation**	**Probability**	**Impact**	**Risk Level**
Technical Difficulties or limitations	Initially explore online tutorials for solutions; consult FYP Supervisor or Doctoral Student if the issue persists.	Medium	High	Medium
Unanticipated dependencies between tasks	Maintain a comprehensive work breakdown structure and update dependencies as required.	Medium	Medium	Medium
Integration difficulties between frontend and backend	Employ consistent APIs, adhere to best practices, and consistently test integration throughout development.	Medium	High	Medium

Inaccurate requirement elicitation or updates in requirements	Conduct additional requirement-gathering meetings with stakeholders and adapt to evolving requirements.	Medium	High	Medium
Unforeseen personal issues or health problems	Develop contingency plans, communicate with the supervisor, and seek support as necessary.	Low	High	Low

3. Literature Review

In this section, we explore various technology stack alternatives for the development of the aspect-based sentiment analysis web interface, encompassing both front-end and back-end development. Choosing the right technologies is a critical step in the development process, as it directly influences the final product's performance, scalability, maintainability, and usability. We will conduct a literature review to compare different technologies, assessing each one's advantages and drawbacks. The technology considerations will cover aspects such as programming languages, frameworks, libraries, and other pertinent tools for developing and deploying the web interface.

3.1. Technology Stack Consideration for Front-end

In this section, we will evaluate three prominent JavaScript frameworks and libraries employed in the development of web applications: React, Vue, and Angular. These technologies have gained considerable popularity and have demonstrated their capabilities as effective tools for creating scalable, high-performance, and maintainable web applications. To determine the best-suited option for the aspect-based sentiment analysis web interface project, we will evaluate each technology based on factors such as its history, community, and development (Daityari, 2023).

Angular

Angular, developed by Google, was first released in 2010 as AngularJS, making it the oldest among the three. It is a TypeScript-based JavaScript framework. In 2016, Angular 2 was released, dropping the “JS” from the original name, and is now referred to as just Angular. Angular is the most mature of the frameworks, backed by many contributors and a complete package for front-end development. However, the learning curve is steep and concepts of development in Angular may put off new developers. Angular is a good choice for companies with large teams and developers who already use TypeScript.

React

React, developed by Facebook, was initially released in 2013. It is widely used in Facebook's products (Facebook, Instagram, and WhatsApp). React has gained widespread acceptance and has a huge number of contributions from the community. The job market for React is promising, and the future for this framework looks bright. React offers the ability to integrate with other frameworks seamlessly, providing flexibility. It is a good choice for someone getting started with front-end JavaScript frameworks, startups, and developers who like flexibility.

Vue

Vue, also known as Vue.js, is the youngest member of the group, developed by ex-Google employee Evan You in 2014. Over the last several years, Vue has seen a substantial shift in popularity, even though it doesn't have the backing of a large company. Vue has gained popularity among users and is valued compared to React. However, the number of contributors for Vue is lower than for Angular and React. Vue should be your choice if you prefer simplicity but also like flexibility. Vue has been chosen as the primary front-end JavaScript framework by several Chinese giants like Alibaba and Baidu.

A comparison is done in the form of a table to identify the most suitable framework (Gairola, 2023).

Criteria	Angular	React	Vue
Availability of Learning Resources	High: Official documentation, community-driven guides, video tutorials, and popular online courses.	High: Extensive official documentation, community-created tutorials, popular online courses, and blogs.	Moderate: Official documentation, growing community resources, online courses, and tutorials are available.
Popularity	High: Active community, large contributor base, and 1.7 million repositories dependent on Angular.	Very High: Most popular on GitHub, huge contributor base, and 5.7 million dependent repositories.	High: Rapidly growing popularity, strong community, and 167,000 dependent repositories combined.
Core Features	Complete Framework:	Component Library:	Flexible Framework:

	Offers full-fledged solutions, including dependency injection, two-way data binding, and form validation.	Focuses on reusable components, one-way data flow, and a virtual DOM.	Adaptable to different use cases, easy to learn, and has a virtual DOM.
Usability	Moderate: Steeper learning curve, especially for beginners, due to its complexity and TypeScript usage.	High: Beginner-friendly, with a more straightforward approach to component-based development.	High: Simple syntax, easy-to-understand component structure, and beginner-friendly documentation.
Ease of Integration with Other Libraries	Moderate: Can integrate with other libraries but may require additional configuration and setup.	High: Easily integrates with various third-party libraries and tools, providing flexibility in development.	High: Designed to be flexible and easily integrates with other libraries, making it adaptable to various use cases.
Future Support	High: Actively developed by Google, with regular updates, long-term support, and a large contributor base.	High: Backed by Facebook, with active development, regular updates, and a large number of contributors.	High: Driven by a dedicated open-source community, with active development, updates, and support.

3.2. Technology Stack Consideration for Back-end

In this section, we assess three notable Python-based frameworks commonly employed in web application development: Django, Flask, and FastAPI. We have opted for a Python-based backend web framework to ensure seamless integration with the EDU-Attention models, which are based on the academic paper that inspired this project—Aspect-based Sentiment Analysis through EDU-level Attentions. We will evaluate the advantages and disadvantages of each framework in this section.

Django

Django, created by Adrian Holovaty and Simon Willison in 2003, is a high-level Python web framework following the Model-Template-View (MTV) pattern. Designed to meet the fast-moving deadlines and complex requirements of modern development teams, Django aims to streamline the development process, enabling developers to create and launch web applications quickly. Its straightforwardness, performance, and open-source nature make it popular among tech giants like YouTube and Instagram. Django excels in rapid development with built-in tools and libraries, reusable code, and a well-organized structure, along with strong community support. However, it can be monolithic and less flexible for smaller projects, has slower performance compared to microframeworks, and has a steeper learning curve compared to Flask or FastAPI. It is an excellent choice for developers working on complex projects, large teams, and those requiring a full-stack solution with built-in tools and features (Podoba, 2022).

Flask

Flask, a Python-based microframework, is widely employed for web application development and has grown even more popular than Django. Initially created by Armin Ronacher as an April Fool's joke, Flask has evolved into a powerful and flexible microframework that does not rely on external libraries for tasks. Its lightweight design and inherent scalability make it ideal for small projects, single-page applications, and developers valuing flexibility and control over their code. Flask is known for its lightweight and flexible nature, customizability, simplicity, and shallow learning curve, and it also boasts a large ecosystem of extensions and libraries. However, Flask may not be ideal for large projects, as it requires additional effort to scale and lacks built-in tools compared to Django, necessitating manual configuration. Its community and support are also smaller compared to Django. Flask is a great choice for startups, developers getting started with web frameworks, and those who enjoy experimenting with various libraries and extensions (Verma, 2021).

FastAPI

FastAPI, a fast, high-performance web framework for building APIs with Python 3.6+, is among the fastest Python frameworks available. Developed to optimize the development experience,

FastAPI enables teams to write simple code for effective APIs. Supporting asynchronous code, it features excellent documentation and differs significantly from Django and Flask. FastAPI is well-known for its speed and efficiency, with support for asynchronous programming, automatic validation, and documentation generation. It is easy to learn and use, with comprehensive documentation. However, FastAPI primarily focuses on API development, making it less suitable for general web development. It is also relatively new and less mature compared to Django and Flask, with a smaller community and fewer resources. FastAPI is a fantastic choice for developers building modern APIs, those looking to leverage the benefits of asynchronous programming, and developers who appreciate a standards-based approach (Sandy, 2021).

A comparison is done in the form of a table to identify the most suitable framework.

Criteria	Angular	React	Vue
Availability of Learning Resources	High: Extensive documentation, large community, numerous online tutorials, courses, and forums	High: Comprehensive documentation, active community, multiple online tutorials and courses	Moderate: Detailed documentation, growing community, an increasing number of tutorials and resources
Popularity	High: Highly popular, used by major tech companies like YouTube and Instagram	High: Gained more popularity than Django, preferred for small projects and single-page applications	Moderate: Rising popularity, especially for API development
Core Features	High: MTV pattern, built-in ORM, admin interface, form handling, authentication, and reusable apps	Moderate: Microframework, lightweight, flexible, extensible through libraries and extensions	Moderate: High-performance, asynchronous programming, automatic validation, and documentation generation
Usability	High: Simplifies the	High: Easy to learn and use,	High: Easy to learn and use,

	development process, suitable for large-scale applications	offers control over code and customizability	excels in modern API development
Ease of Integration with Other Libraries	Moderate: Good integration with other Python libraries and extensions, though less flexible compared to Flask	High: Excellent integration with other Python libraries and extensions, highly customizable	Moderate: Supports integration with other Python libraries, though less mature compared to Django and Flask
Future Support	High: Strong future support due to its large and active community, as well as industry adoption	High: Continued growth in popularity and adoption, ensuring future support and development	Moderate: As popularity grows, future support is expected to increase, driven by community and industry demand

3.3. Technology Choice for Front-end

In this section, we discuss the rationale behind selecting React as the most suitable choice for the front-end development of our aspect-based sentiment analysis web interface project. React, a JavaScript library created by Facebook has become a preferred solution for building user interfaces due to its numerous advantages and widespread adoption.

A primary reason for choosing React is its component-based architecture. React enables developers to construct complex user interfaces by creating small, isolated, and reusable pieces of code known as “components” This modular approach promotes efficient development, simplified maintenance, and enhanced code reusability. Components can be combined in various ways to develop dynamic and interactive UIs while ensuring each component is easily testable and maintainable.

Another significant advantage of React is its performance. React employs a virtual DOM, minimizing direct interactions with the actual DOM. This results in faster updates and improved rendering performance, guaranteeing a seamless user experience even in large-scale applications.

Moreover, React's popularity has fostered a vast ecosystem of libraries and tools that can be seamlessly integrated to extend its functionality. Developers can utilize existing solutions to address common issues and concentrate on the project's unique aspects.

In addition to technical benefits, the choice of React for this project is influenced by the developers' familiarity with the framework. React's learning curve is relatively gentle, and an extensive array of learning resources, including tutorials, courses, and forums, helps developers quickly master the library. This familiarity allows for accelerated development and ensures the developer can effectively troubleshoot and resolve any issues that may arise during the project.

In conclusion, React's component-based architecture, high performance, comprehensive ecosystem, and the developers' familiarity with the framework make it the ideal choice for this project's front-end development. By leveraging React's capabilities, the developer can create a

robust and scalable user interface that fulfils the project's requirements and delivers a smooth user experience.

3.4. Technology Choice for Back-end

After a comprehensive evaluation of various back-end frameworks, FastAPI emerges as the most suitable choice for the back-end development of this project, particularly given its microservices architecture.

One of the primary motivations behind selecting FastAPI is its exceptional performance capabilities. Designed to maximize development efficiency, FastAPI is among the fastest Python frameworks currently available. Its support for asynchronous programming allows a single developer to create highly concurrent and resource-efficient back-end services, which translates into a responsive and scalable application.

In this project, the adoption of a microservices architecture is anticipated, which further accentuates the importance of FastAPI's high performance and speed. Within this approach, microservices are structured as modular, independently deployable components that interact with one another. Consequently, FastAPI's efficiency and support for asynchronous programming play a critical role in fostering the development of performant, low-latency microservices. This, in turn, contributes to a highly responsive and resilient system that effectively meets the project's requirements and objectives.

Another significant advantage of employing FastAPI is its innate ability to perform automatic validation and generate documentation. The framework offers built-in validation for incoming and outgoing data, which streamlines error management and promotes data consistency. Additionally, FastAPI automatically produces documentation for APIs, making it easier for the developer to maintain and update API documentation as the project progresses.

Moreover, FastAPI's comprehensive documentation and ease of use facilitate a swift learning process, enabling the developer to rapidly gain expertise in the framework. This familiarity with

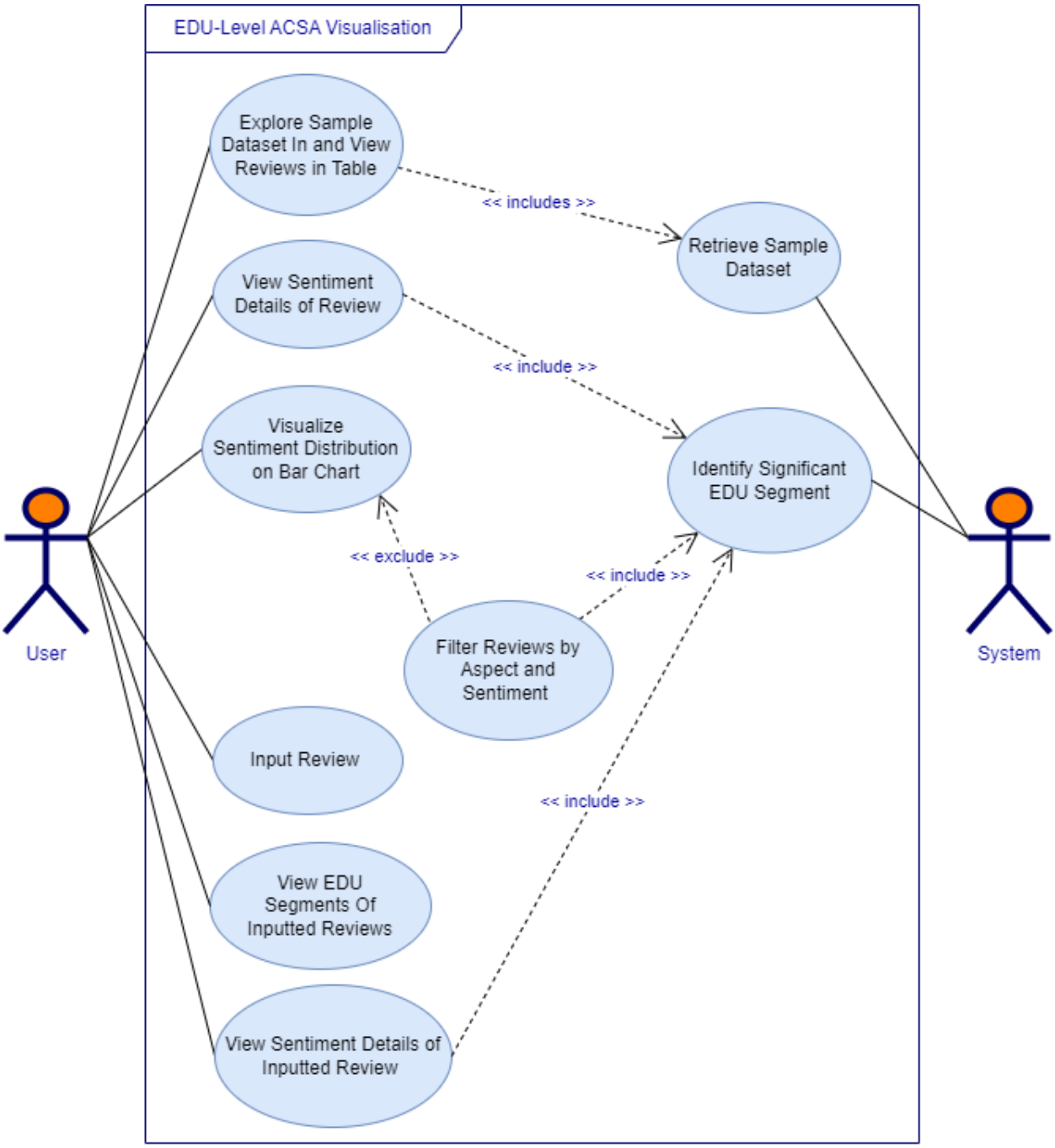
FastAPI will empower the developer to expedite the development and address any issues efficiently.

A crucial aspect of this project involves the seamless integration of the existing Sentiment Analysis and Segmentation Python codebase. FastAPI's inherent flexibility and compatibility with various Python libraries render it the ideal choice for merging these codebases. By utilizing FastAPI, the developer can ensure that back-end services are well-integrated with the Sentiment Analysis and Segmentation components, culminating in a cohesive and efficient solution.

In summary, FastAPI's exceptional performance, support for asynchronous programming, automatic validation and documentation, ease of use, and seamless integration capabilities make it the optimal choice for this project's back-end development within a microservices architecture.

4. Software Requirements

4.1. Use Case Diagram



4.1.1. Use Case Description

Use Case ID:	UC-1
Use Case Name:	Explore Sample Dataset and View Reviews in Table
Created By:	Jeremy Teo
Date Created:	15 December 2022
Actor:	User
Description:	The user explores a sample dataset of restaurant reviews and views them in a table format on the "Explore" page.
Preconditions:	The user has accessed the Explore page of the web interface for aspect-based sentiment analysis of restaurant reviews.
Postconditions	-
Priority:	High
Frequency of Use:	Frequent
Flow of Events:	1. The user selects one of the two available sample datasets. 2. The system retrieves the selected dataset (UC-8: Retrieve Sample Dataset). 3. The system processes the selected dataset and displays the restaurant reviews in a table. 4. The user views the table and can click on a row to see more details about the review and its associated sentiments.
Alternative Flows:	-
Exceptions	EX1: If the system fails to retrieve the selected dataset. The system displays the message "Unable to load the dataset. Please try again."
Includes	UC-8 (Retrieve Reviews Dataset)
Special Requirements:	-
Assumptions:	The pre-uploaded Reviews Dataset contains valid reviews, aspects, and sentiments.
Notes and Issues:	-

Use Case ID:	UC-2
Use Case Name:	View Sentiment Details of Review
Created By:	Jeremy Teo
Date Created:	15 December 2022
Actor:	User
Description:	The user views the sentiment details of a specific review by clicking on the review from a table.
Preconditions:	The user is viewing a table containing a list of reviews on the “Explore” page.
Postconditions	The user can view the sentiment details of the selected review.
Priority:	High
Frequency of Use:	Frequent
Flow of Events:	1. The user clicks on a specific review in the table. 2. The system displays a card component which contains the review text and corresponding aspect-based sentiment present in the review. 3. The user clicks on an aspect-based sentiment button. 4. The system highlights the corresponding EDU segment in the review text, showcasing the sentiment.
Alternative Flows:	-
Exceptions	-
Includes	UC-9 (Identify Significant EDU Segment)
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	UC-3
Use Case Name:	Visualize Sentiment Distribution on Bar Chart
Created By:	Jeremy Teo
Date Created:	15 December 2022
Actor:	User
Description:	The user visualizes the sentiment distribution across all target aspects in a bar chart on the “Explore” page.
Preconditions:	The user is viewing a sample dataset of restaurant reviews on the “Explore” page.
Postconditions	The user can view and interact with the sentiment distribution bar chart.
Priority:	High
Frequency of Use:	Frequent
Flow of Events:	1. The system displays a bar chart with sentiment distribution across all target aspects. 2. Each group of bars represents one of the target aspects, and each bar within the group represents a Positive, Negative, or Neutral sentiment. 3. The user can click on a bar to view the reviews corresponding to that target aspect and sentiment.
Alternative Flows:	-
Exceptions	-
Extends	UC-4 (Filter Reviews by Aspect and Sentiment)
Includes	UC-9 (Identify Significant EDU Segment)
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	UC-4
Use Case Name:	Filter Reviews by Aspect and Sentiment
Created By:	Jeremy Teo
Date Created:	15 December 2022
Actor:	User
Description:	The user filters restaurant reviews by a specific target aspect and sentiment on the "Explore" page after interacting with the sentiment distribution bar chart.
Preconditions:	1. The user is viewing the sentiment distribution bar chart on the "Explore" page. 2. The user has clicked on a bar representing a specific target aspect and sentiment in the chart.
Postconditions	The user views a filtered list of reviews corresponding to the selected target aspect and sentiment, with the corresponding EDU segment highlighted.
Priority:	High
Frequency of Use:	Frequent
Flow of Events:	1. The user clicks on a bar representing a specific target aspect and sentiment in the sentiment distribution bar chart (UC-3). 2. The system filters the reviews based on the selected target aspect and sentiment. 3. The system displays a table with the filtered list of reviews. 4. The table shows the corresponding EDU segment highlighted for each review, which represents the selected target aspect and sentiment.
Alternative Flows:	-
Exceptions	-
Includes	UC-9 (Identify Significant EDU Segment)
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	UC-5
Use Case Name:	Input Review
Created By:	Jeremy Teo
Date Created:	15 December 2022
Actor:	User
Description:	The user inputs a restaurant review on the “Segment” or “Analyze” page for EDU segmentation or aspect-based sentiment analysis, respectively.
Preconditions:	The user is on either the “Segment” or “Analyze” page of the web interface.
Postconditions	The user has successfully submitted a review for processing.
Priority:	High
Frequency of Use:	Frequent
Flow of Events:	1. The user navigates to either the “Segment” or “Analyze” page. 2. The user inputs a restaurant review in the provided text field. 3. The user clicks the “Submit” button to process the review. 4. The system processes the user's input based on the chosen page: a. If the user is on the “Segment” page, the system processes the review for EDU segmentation (UC-6). b. If the user is on the “Analyze” page, the system processes the review for aspect-based sentiment analysis (UC-7).
Alternative Flows:	-
Exceptions	-
Includes	UC-6 (View EDU Segments of Inputted Review) UC-7 (View Sentiment Details of Inputted Review)
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	UC-6
Use Case Name:	View EDU Segments of Inputted Review
Created By:	Jeremy Teo
Date Created:	15 December 2022
Actor:	User
Description:	The user views the EDU segments of the inputted restaurant review on the "Segment" page after submitting the review.
Preconditions:	1. The user has inputted a restaurant review on the "Segment" page (UC-5). 2. The user has clicked the "Submit" button to process the review.
Postconditions	The user views the EDU segments of the inputted review in a card component.
Priority:	High
Frequency of Use:	Frequent
Flow of Events:	1. The system processes the inputted review for EDU segmentation. 2. The system displays a card component containing the EDU segments of the review as a list. 3. The user views the EDU segments of the inputted review in the card component.
Alternative Flows:	-
Exceptions	EX1: If the system fails to process the input review for EDU segmentation. 1. The system displays the message "Unable to process the review. Please try again."
Includes	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	UC-7
Use Case Name:	View Sentiment Details of Inputted Review
Created By:	Jeremy Teo
Date Created:	15 December 2022
Actor:	User
Description:	The user views the aspect-based sentiment details of the inputted restaurant review on the "Analyze" page after submitting the review.
Preconditions:	1. The user has inputted a restaurant review on the "Analyze" page (UC-5). 2. The user has clicked the "Submit" button to process the review.
Postconditions	The user views the aspect-based sentiment details of the inputted review in a card component.
Priority:	High
Frequency of Use:	Frequent
Flow of Events:	1. The system processes the inputted review for aspect-based sentiment analysis. 2. The system displays a card component containing the review text and corresponding aspect-based sentiment present in the review. 3. The user views the aspect-based sentiment details of the inputted review in the card component. 4. The user can click on an aspect-based sentiment button to highlight the corresponding EDU segment in the review text.
Alternative Flows:	-
Exceptions	EX1: If the system fails to process the input review for aspect-based sentiment analysis. 2. The system displays the message "Unable to process the review. Please try again."
Includes	UC-9 (Identify Significant EDU Segment)
Special Requirements:	-
Assumptions:	-

Notes and Issues:	-

Use Case ID:	UC-8
Use Case Name:	Retrieve Sample Dataset
Created By:	Jeremy Teo
Date Created:	15 December 2022
Actor:	System
Description:	The system retrieves a sample dataset of restaurant reviews for the “Explore” page.
Preconditions:	The user is on the “Explore” page and has selected one of the two available sample datasets.
Postconditions	The system has retrieved the selected dataset and made it available for the user to explore.
Priority:	High
Frequency of Use:	Frequent
Flow of Events:	1. The user selects one of the two available sample datasets on the “Explore” page. 2. The system retrieves the selected dataset and makes it available for exploration. 3. The system processes the dataset and displays the restaurant reviews in a table format (UC-1).
Alternative Flows:	-
Exceptions	EX1: If the system fails to retrieve the selected dataset. 1. The system displays the message “Unable to load the dataset. Please try again.”
Includes	-
Special Requirements:	-
Assumptions:	-

Notes and Issues:	-

Use Case ID:	UC-9
Use Case Name:	Identify Significant EDU Segment
Created By:	Jeremy Teo
Date Created:	15 December 2022
Actor:	System
Description:	The system identifies and highlights the significant EDU segments in a restaurant review when the user clicks on the aspect-based sentiment button.
Preconditions:	The user is viewing a card component with aspect-based sentiment details, either on the “Explore” page or the “Analyze” page.
Postconditions	The system highlights the corresponding EDU segment in the review text, showcasing the sentiment.
Priority:	High
Frequency of Use:	Frequent
Flow of Events:	1. The user clicks on an aspect-based sentiment button in the card component. 2. The system identifies the significant EDU segment corresponding to the selected aspect-based sentiment. 3. The system highlights the identified EDU segment in the review text.
Alternative Flows:	-
Exceptions	-
Includes	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

4.2 Functional and Non-Functional Requirements

In this following section, we will be providing a thorough overview of the functional and non-functional requirements for the web interface project. These requirements have been amassed during the requirements elicitation phase, utilizing interviews with various stakeholders to gain insight into their expectations for the web interface. The collection of these requirements is crucial to the project's success.

Functional requirements relate to the software's operational aspects, including the necessary features and functions that must be incorporated to achieve its intended purpose. On the other hand, non-functional requirements encompass the implicit or anticipated attributes the software should display, with a focus on the way it carries out its functions. Our objective in defining these requirements is to lay a robust foundation for the project's success and guarantee that the final product aligns with user needs and expectations.

4.2.1. Functional Requirements

Explore Sample Dataset and View Reviews in Table

1. The system must allow users to select a sample dataset on the Explore page.
    1.1. The system must display a drop-down menu for the user to select a sample dataset on the Explore page.
2. The system must display restaurant reviews from the selected dataset in a table format.
    2.1. The system must allow users to view more details about a review and its associated sentiments by clicking on a row in the table.

View Sentiment Details of Review

3. The system must display a card component containing the review text and corresponding aspect-based sentiment for a specific review when a row is clicked.
    3.1. The system must display an aspect-based sentiment button for the user to click and highlight the corresponding EDU segment in the review text.

Visualize Sentiment Distribution on Bar Chart

4. The system must display a bar chart on the Explore page for the user to view the sentiment distribution across all target aspects.
    4.1. The system must allow users to click on a bar in the chart to view the reviews corresponding to that target aspect and sentiment.

Filter Reviews by Aspect and Sentiment

5. The system must enable filtering of restaurant reviews based on a specific target aspect and sentiment.
    5.1. The system must display a table with the filtered list of reviews, with the corresponding EDU segment highlighted for each review.

Input Review

6. The system must display a text field on the Segment or Analyze page for the user to input a restaurant review.
    6.1. The system must allow users to submit the review for EDU segmentation or aspect-based sentiment analysis, depending on the chosen page.

View EDU Segments of Inputted Review

7. The system must display a card component containing the EDU segments of the inputted review as a numbered list.
    7.1. The system must allow users to view the EDU segments of the inputted review in the card component.

View Sentiment Details of Inputted Review

8. The system must display a card component containing the inputted review text and corresponding aspect-based sentiment.
    8.1. The system must allow users to view the aspect-based sentiment details of the inputted review in the card component.
    8.2. The system must allow users to click on an aspect-based sentiment button to highlight the corresponding EDU segment in the review text.

Retrieve Sample Dataset

9. The system must retrieve the selected sample dataset of restaurant reviews when the user selects a dataset on the Explore page.
    9.1. The system must display the restaurant reviews in a table format for the user to explore.

Identify Significant EDU Segment

10. The system must identify and highlight the significant EDU segments in a restaurant review when the user clicks on the aspect-based sentiment button.
    10.1. The system must display the highlighted EDU segment in the review text.

4.2.2. Non-Functional Requirements

Usability

1. The system must be designed with an intuitive and user-friendly interface, minimizing the learning curve for new users.
2. The system must provide clear instructions and tooltips to help users understand the functionality of each component and feature.
3. The system must use consistent visual design and interaction patterns across all pages and components to ensure a cohesive user experience.
4. The system must offer appropriate feedback to the user, such as visual cues, confirmation messages, and progress indicators, to help users understand the results of their actions.
5. The system must allow users to easily navigate between different pages and sections, with a clear and logical information hierarchy.

Performance

1. The system must load the Explore, Segment, and Analyze pages within 3 seconds for a smooth user experience.
2. The system must process and display EDU segmentation and aspect-based sentiment analysis results within 7 seconds after the user submits a review.

Reliability

1. The system must handle any errors gracefully by displaying appropriate error messages to the user.

Maintainability

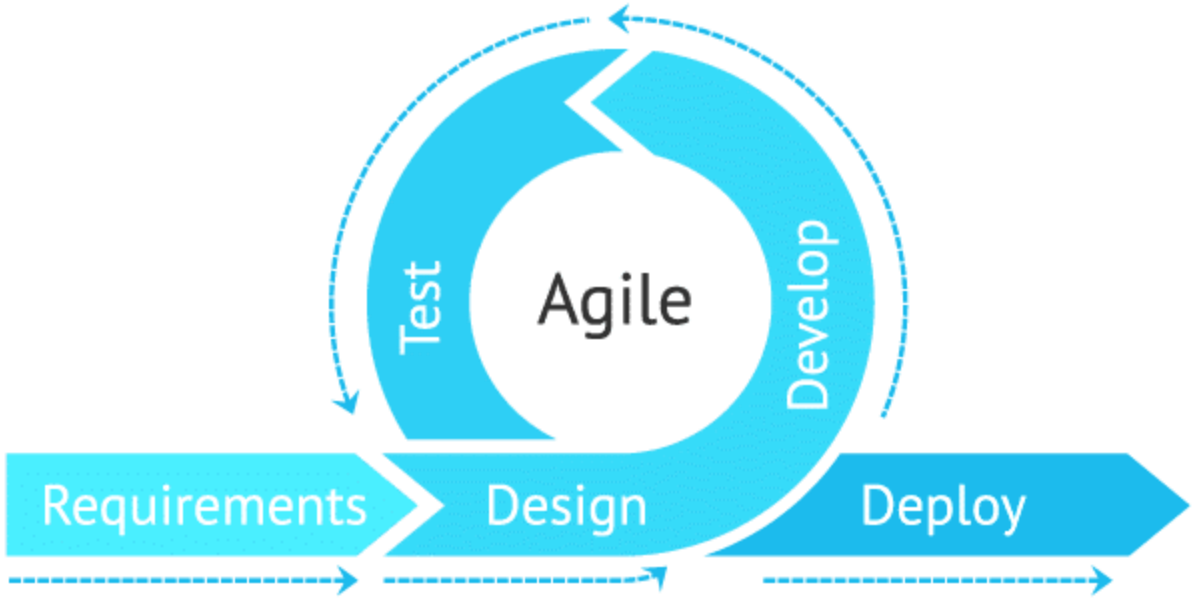
1. The system must exhibit software design traits that show low coupling for maintainability and extensibility.
2. The system must exhibit software design traits that show high cohesion for maintainability and extensibility.
3. The system's code must be well-documented and follow established coding standards and best practices.

4. The system must be compatible with the latest versions of popular web browsers, including Google Chrome, Mozilla Firefox, Microsoft Edge, and Apple Safari.

5. Planning and Design

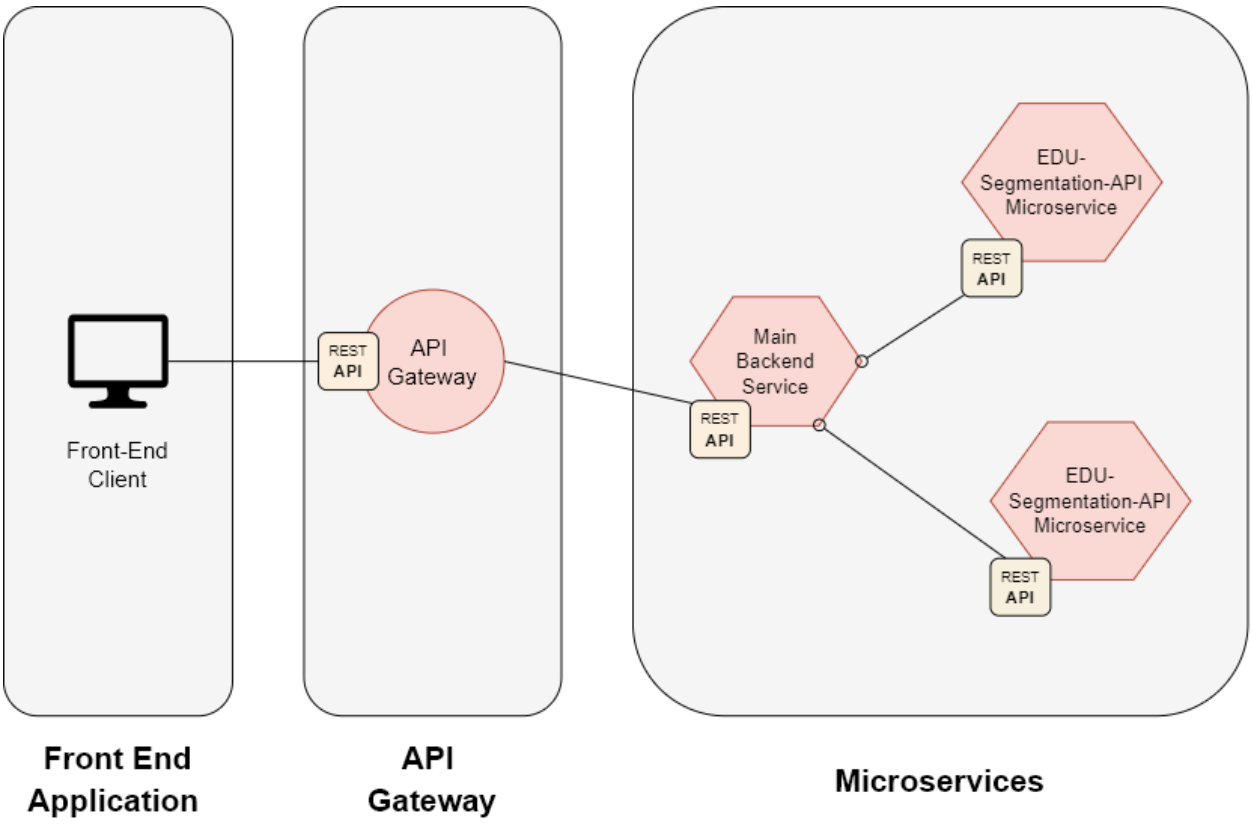
5.1. Project Development Methodology

In this section, we discuss the software development approach employed for the project, which involves building a Web Interface to showcase an aspect-based sentiment analysis model. The software development methodology utilized for this project is the Agile Development Methodology. Adopting this approach, the project is broken down into different stages and consistently involves various stakeholders. These stages include requirements elicitation, design, and software development. At every stage, there is continuous development and iteration, allowing for constant improvement and adaptation to changing requirements.



Adhering to agile software development principles, 2-week duration Sprints are typically implemented. Each Sprint encompasses a full development cycle, including analysis, design, coding, and testing. At the end of each Sprint, working software is available and reviewed by the supervisor during biweekly meetings. Following the review, extensions, changes, or enhancements to the existing working software are carried out in the subsequent Sprint, bringing the software closer to its final version. The Agile Development Methodology is well-suited for projects like this, where requirements and functionalities of the web application are added incrementally, and the ability to adapt to changes is a primary concern.

5.2. System Architecture



The system architecture selected for this project is a microservices architecture, dividing the application into a collection of small, modular, and independently deployable services, each responsible for specific functionality. These services communicate with each other through well-defined interfaces, typically via HTTP/RESTful APIs. In this project, the architecture consists of three main components: Frontend Application, API Gateway, and a set of Microservices including the Main Backend Service, EDU-Segmentation-API, and EDU-Sentiment-API.

The microservices architecture contributes to maintainability, scalability, and reusability, and demonstrates the separation of concerns principle, organizing the codebase and enhancing the system's modularity. Each service encapsulates its logic, focusing on a particular aspect of the application, which improves the system's maintainability and simplifies navigation through the codebase. In this project, the distinct microservices - the segmentation and sentiment services -

require different Python environments due to unique dependencies with other NLP Python packages. A microservices architecture enables these services to operate in their respective environments without conflicts, ensuring smooth and efficient development.

Decomposing the system into loosely coupled services facilitates dependency management and minimizes the impact of changes in one service on others. Services can be reused across projects, as long as they adhere to the same interfaces, enabling efficient development and deployment of new features and applications without reimplementing functionality.

The microservices architecture, combined with FastAPI for backend development and React for frontend development, guarantees a highly maintainable, scalable, and responsive system meeting the project's requirements and objectives. This architecture's modular nature allows seamless integration with the Sentiment Analysis and Segmentation Python codebase, ensuring a cohesive and efficient development process.

In this specific project, restaurant reviews are analyzed and segmented using interconnected services. The Frontend Application serves as the user interface, receiving reviews, sending requests to the backend services, and displaying results. The API Gateway acts as a single entry point for all microservices in the architecture, routing incoming requests to the appropriate microservices based on the request path, simplifying frontend code, and enabling independent management and scaling of microservices.

The Main Backend Service coordinates data flow between the frontend application and other microservices, processing raw review text, segmenting it with the EDU-Segmentation-API, and performing sentiment analysis using the EDU-Sentiment-API. The EDU-Segmentation-API microservice segments the raw review text into sentences, while the EDU-Sentiment-API microservice performs sentiment analysis on the segmented sentences.

By leveraging the microservices architecture and FastAPI and React technologies, this project ensures a highly maintainable, scalable, and efficient system that meets its objectives while

facilitating the analysis and segmentation of restaurant reviews through a series of interconnected, specialized services.

5.3. User Interface Wireframe

This section presents the wireframe designs for the key pages of the EDU-Web Interface, providing a clear visual representation of the interface's layout and functionality before beginning the development process. By showcasing these wireframes, stakeholders can gain a comprehensive understanding of the overall structure, navigation, and user experience, enabling them to provide feedback and suggestions for improvement.

"Explore" Page

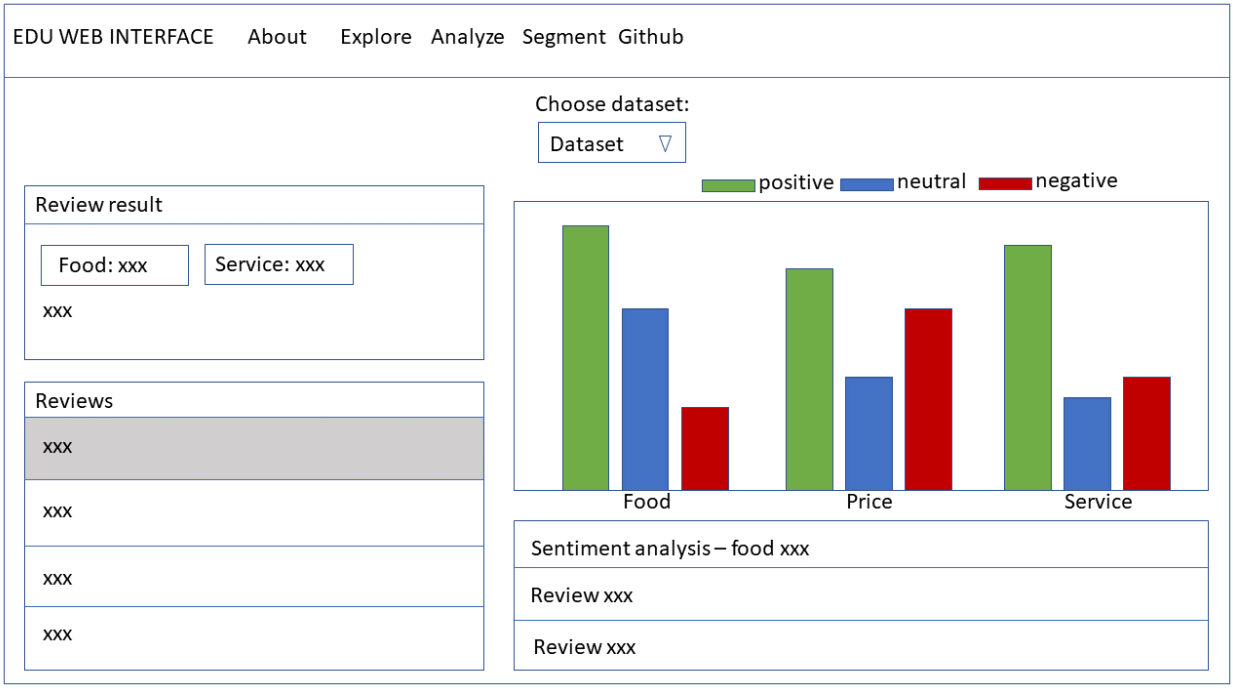


Figure 2: "Explore" Page (Wireframe)

The "Explore" page is designed to allow users to explore a sample dataset of restaurant reviews. They can choose between two different datasets, one containing 25 hard sentences and the other containing another set of reviews. These reviews are displayed in a clickable table, and upon selecting a row, a card component appears, showcasing the specific review and its associated sentiments. In the card component, the aspect-based sentiments will be displayed as clickable

buttons. Upon clicking these buttons, the corresponding EDU segments in the review that contribute to the specific aspect-based sentiment will be highlighted. Additionally, the page also features a chart that displays the distribution of sentiments across five target aspects for the restaurant, with clickable bars to reveal the corresponding reviews and highlighted EDU segments in a table below the chart.

“Segment” Page

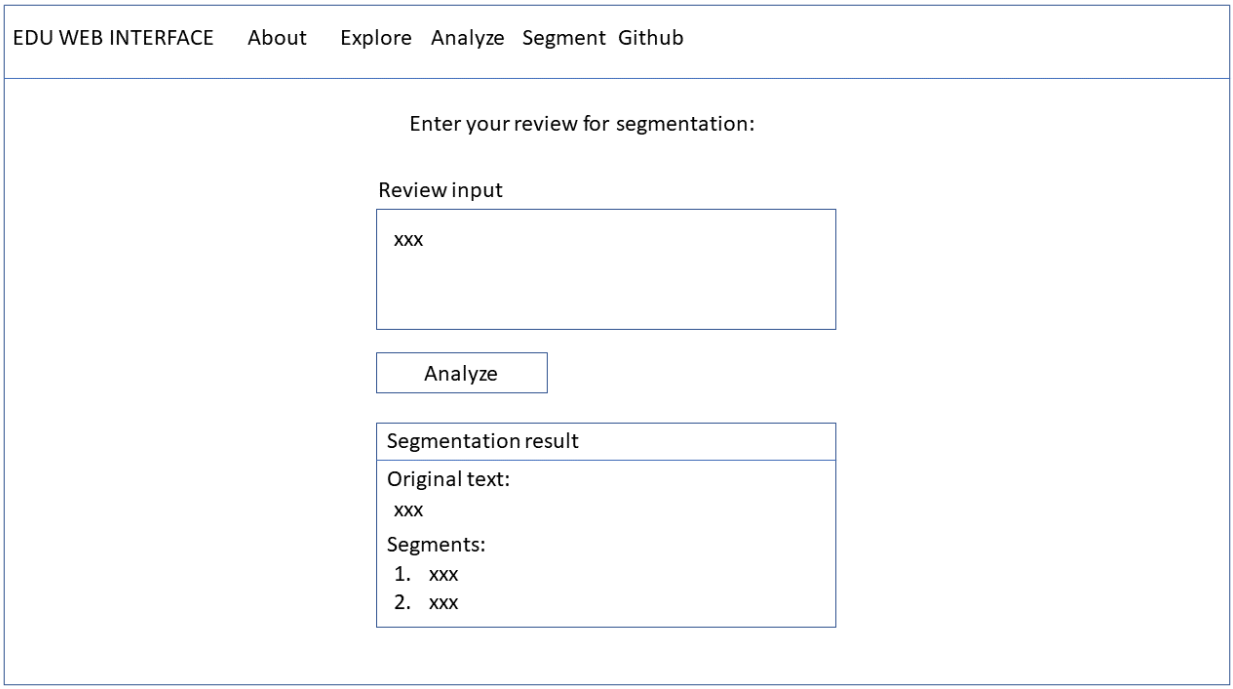


Figure 3: "Segment" Page (Wireframe)

The "Segment" page provides a text field for users to input a review, which will then be segmented according to its EDU segments. The segmented review will be displayed in a card component, with the EDU segments presented as a list below the text.

“Analyze” Page

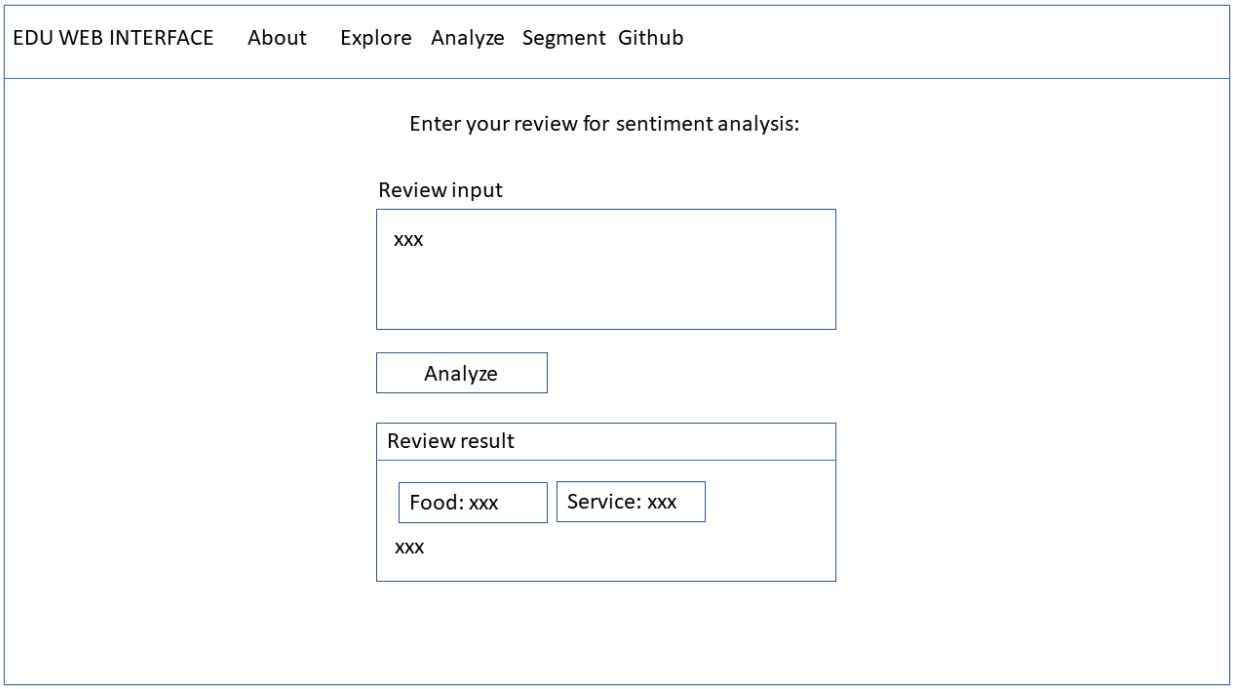


Figure 4: "Analyze" Page (Wireframe)

The "Analyze" page enables users to input a restaurant review for aspect-based sentiment analysis. After the analysis, a card component displays the review alongside the detected aspect-based sentiments, represented as clickable buttons. Upon clicking these buttons, the corresponding EDU segments in the review that contribute to the specific aspect-based sentiment will be highlighted.

6. Implementation

6.1. Backend Development

In this section, we will be providing an overview of the FastAPI backend implementation. Here, we will cover the installation process and environment setup, focusing on package requirements, version constraints, and dependency management. We will delve into data models (schemas) and middleware, detailing their purpose, relationships, and contributions to functionality and security. Routes will be outlined, explaining their roles within the overall architecture, while the testing strategy, including unit and integration tests, will be described to demonstrate the robustness and scalability of our chosen approach. This comprehensive overview will offer insights into the backend architecture and design choices made for this project.

6.1.1. Installation

This project adopts a Microservices architecture, and as a result, the backend server consists of several services developed using different versions of Python. The API Gateway, Main Backend Server, and EDU-Sentiment-API use Python 3.9.13, while the EDU-Segmentation-API employs Python 3.7.16 due to dependency requirements for the text segmentation task.

Windows

To set up the appropriate Python environments for each service on Windows, download and install the required Python versions (3.9.13 and 3.7.16) from https://www.python.org/downloads/windows/, making sure to check the box to add Python to the PATH environment variable during installation. Once the installation is complete, run the ***python --version*** command in the command prompt to ensure that the required versions of Python are successfully installed.

Mac

For Mac users, Python can be installed using Homebrew with the following commands: ***brew install python@3.9*** and ***brew install python@3.7***.

6.1.2. Environment

Here, we will be outlining how the environment was set up for each of the services in the project. As each service has its specific dependencies, it was necessary to create separate environments with the appropriate Python versions and requirements. To achieve this, we created distinct "requirements.txt" files for each service, detailing the necessary packages and their corresponding versions. The details of each of the "requirements.txt" file can be found in the figures below:

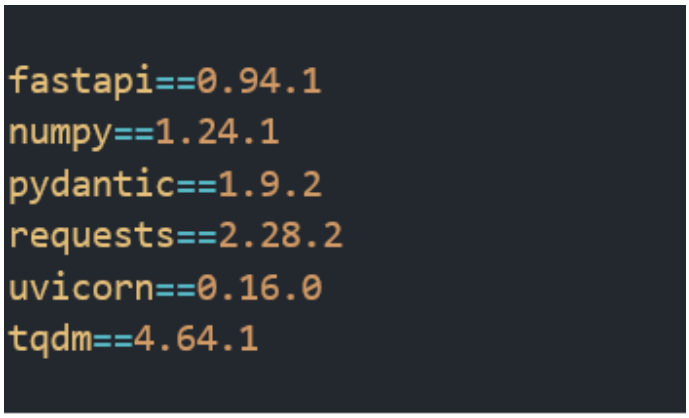


Figure 5: API Gateway and Main Backend Service "requirements.txt"

```
fastapi==0.95.0
numpy==1.24.1
pydantic==1.9.2
pytest==7.1.2
scikit_learn==1.2.1
tensorboard_logger==0.1.0
torch==1.13.1
tqdm==4.64.1
uvicorn==0.16.0
nltk==3.8.1
transformers==4.26.0
```

Figure 6: EDU-Sentiment-API requirements.txt

```
fastapi==0.95.0
nltk==3.8.1
numpy==1.21.6
pydantic==1.10.6
uvicorn==0.21.1
https://download.pytorch.org/whl/cpu/torch-1.2.0%2Bcpu-cp37-cp37m-
manylinux1_x86_64.whl
```

Figure 7: EDU-Segmentation-API requirements.txt

After setting up the correct Python versions, as mentioned in the installation instructions, we proceeded to create virtual environments for each service with the respective "requirements.txt" file. For the API Gateway, Main Backend Service, and EDU-Sentiment-API services, which use Python 3.9.13, the command executed was: ***py -3.9 -m venv venv***. On the other hand, for the EDU-Segmentation-API service, which uses Python 3.7.16, the command executed was: ***py -3.7 -m venv venv***.

The creation of these virtual environments allows for each service to operate without conflicts between their dependencies. This is especially important for the EDU-Sentiment-API and EDU-Segmentation-API services, which have conflicting dependencies in their codebases. Having

created these virtual environments, we installed the required dependencies for each service using the pip install -r requirements.txt command. This command reads the respective requirements.txt file and installs the listed packages and their specific versions into the corresponding virtual environment.

6.1.3. Data Models (Schemas)

Next, we will discuss the data models or schemas used in each service, which are essential for ensuring data consistency and validation throughout the system. To accomplish this, we have used Pydantic, a powerful library that leverages Python type annotations for data validation and settings management. By using Pydantic, we can safely parse JSON data into specific Python objects, making it easier to work with and integrate into the application's logic (Samiullah, 2021).

It is important to note that the API Gateway service does not have JSON validation in place and, therefore, does not use Pydantic data models to parse the incoming JSON data. This is because the primary responsibility of the API Gateway is to route incoming requests to the appropriate microservices without processing or modifying the received data. Moreover, since the microservices receiving the requests already have their validation mechanisms in place, the API Gateway can delegate the validation responsibility, allowing for more focused and tailored validation.

```
class InputText(BaseModel):
    text: str
```

Figure 8: Schema for Main Backend Service

In the Main Backend Service, a Pydantic schema named ***InputText*** is used to ensure data consistency and validation. The InputText schema consists of a single field called text, which is of the type str. This schema is designed to receive and validate the raw review text from the client, ensuring that it is properly formatted as a string before further processing.

```python
class InputQuery(BaseModel):
    query: str

class SegmentResult(BaseModel):
    text: str
    segs: list
```

Figure 9: Schema for EDU-Segmentation-API Service

In the EDU-Segmentation-API, two Pydantic schemas are utilized to maintain data consistency and validation: ***InputQuery*** and ***SegmentResult***.

The ***InputQuery*** schema comprises a single field called ***query***, which is of the type ***str***. This schema is designed to receive and validate the raw review text from the Main Backend Service, ensuring that it is properly formatted as a string before being processed for segmentation.

The ***SegmentResult*** schema consists of two fields: ***text*** and ***segs***. The ***text*** field is of the type ***str*** and represents the original input text. The ***segs*** field is of the type ***list***, and it stores the segmented sentences resulting from the segmentation process. This schema is utilized to return the segmented text as a structured Python object, making it easier for the Main Backend Service to handle the data and subsequently pass it to the EDU-Sentiment-API for sentiment analysis.

```python
class InputData(BaseModel):
    text: str
    segs: list


class AnalysisResult(BaseModel):
    query: str
```

Figure 10: Schema for EDU-Sentiment-API Service

In the EDU-Sentiment-API, Pydantic is utilized to define two schemas that help maintain data consistency and validation: ***InputData*** and ***AnalysisResult***.

The ***InputData*** schema comprises two fields: ***text*** and ***segs***. The ***text*** field is of the type ***str*** and represents the original input text. The ***segs*** field is of the type ***list***, and it stores the segmented sentences provided by the EDU-Segmentation-API. This schema is designed to receive and validate the input data from the Main Backend Service, ensuring proper formatting before performing sentiment analysis on the segmented sentences.

The ***AnalysisResult*** schema consists of a single field called ***query***, which is of the type ***str***. This schema is used to return the sentiment analysis results as a structured Python object. It facilitates the handling of the data by the Main Backend Service, which subsequently returns the results to the frontend application for display.

The integration of Pydantic and well-defined schemas in the Main Backend Service, EDU-Segmentation-API, and EDU-Sentiment-API significantly contributes to maintaining data consistency, validation, and proper formatting throughout the system. The utilization of Pydantic for data validation and settings management establishes a robust and reliable foundation for handling JSON data and converting it into specific Python objects.
As a result, this not only streamlines data processing within each service but also facilitates the integration of each service's distinct application logic. As a result, this strategy enhances the overall efficiency, maintainability, and scalability of the system.

6.1.4. Middleware

In this project, there is a middleware implementation to enable Cross-Origin Resource Sharing (CORS) across all microservices. CORS is a security feature implemented in web browsers that restricts web applications from making requests to a different domain than the one that served the web application. However, in our project, since the frontend and backend services may be hosted on different domains, it is necessary to allow CORS for seamless communication between the services.

To achieve this, we have implemented a middleware using FastAPI's built-in ***CORSMiddleware*** class. The following code snippet demonstrates the configuration used in each service to allow CORS for all origins.

```
# Allow CORS for all origins
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Figure 11: CORSMiddleware Implementation

By adding this middleware, we have configured the services to accept requests from any origin, allowing all methods and headers. This configuration ensures that our frontend application can communicate with the backend services without any CORS-related restrictions, facilitating seamless integration and data exchange between the different components of our system.

6.1.5. Routes

In this section, we will be discussing the implementation of the routes that are used in all the microservices.

6.1.5.1. API Gateway

Here, we will discuss the implementation of the routes in API Gateway service. The API Gateway service has three main routes as shown in the figure below.

```python
@app.get("/api/rest-raw-data/{id}")
def get_raw_data(id: int = Path(..., description="The raw data file ID")):
    url = SERVER_URL + f"rest-raw-data/{id}"
    response = requests.get(url)
    return response.json()


@app.post("/api/analyze-rest-review")
def analyze_rest_review(*, payload: dict):
    url = SERVER_URL + "analyze-rest-review"
    headers = {"Content-Type": "application/json"}
    response = requests.post(url, json=payload, headers=headers)
    return response.json()


@app.post("/api/segment-rest-review")
def segment_rest_review(*, payload: dict):
    url = SERVER_URL + "segment-rest-review"
    headers = {"Content-Type": "application/json"}
    response = requests.post(url, json=payload, headers=headers)
    segs_response = response.json()
    return segs_response
```

Figure 12: API Gateway Routes

/api/rest-raw-data/{id}

This route serves raw data based on the provided ID. The function ***get_raw_data*** receives the ***ID*** as a path parameter and forwards the request to the main backend service. The response from the main backend service is then returned as a JSON object.

/api/analyze-rest-review

This route forwards requests to the main backend service to analyze restaurant reviews. The function ***analyze_rest_review*** receives the payload as a dictionary and sends a POST request to the main backend service with the appropriate URL, headers, and JSON data.

/api/segment-rest-review

This route forwards requests to the main backend service to segment restaurant reviews. The function segment_rest_review receives the payload as a dictionary and sends a POST request to

the main backend service with the appropriate URL, headers, and JSON data. The segmented sentences are returned as a JSON object.

6.1.5.2. Main Backend Service

Here, we will discuss the implementation of the routes in the Main Backend Service. The Main Backend service has three main routes, responsible for serving raw data, analyzing restaurant reviews, and segmenting user reviews.

```python
@app.get("/api/rest-raw-data/{id}")
def get_raw_data(id: int):
    file_paths = {
        1: "C:/Users/jimmy/Documents/GitHub/FYP-App/server/data/rest_han_reg.raw",
        2: "C:/Users/jimmy/Documents/GitHub/FYP-App/server/data/rest_han_reg_v2.raw",
    }

    file_path = file_paths.get(id)
    if file_path:
        json_string = util.convert_to_json_from_path(file_path)
        return json_string
    else:
        raise HTTPException(status_code=404, detail="Raw data file not found")
```

Figure 13: Main Backend Service Route 1

/api/rest-raw-data/{id}

This route serves raw data based on the provided ID. The function ***get_raw_data*** receives the ID as a path parameter, retrieves the corresponding file path from a predefined dictionary, and returns the JSON data from the specified file. If the file is not found, a 404 HTTP exception is raised.

```python
@app.post("/api/analyze-rest-review")
def analyze_text(input_text: InputText = Body(...)):
    seg_url = SEGBOT_URL + "segbot-segment-service"
    payload = {"query": input_text.text}
    headers = {"Content-Type": "application/json"}

    seg_response = requests.post(seg_url, data=json.dumps(payload), headers=headers)

    if seg_response.status_code == 200:
        seg_data = seg_response.json()
        edu_url = EDU_URL + "edu-sentiment-analysis-service"
        edu_payload = seg_data
        edu_response = requests.post(edu_url, data=json.dumps(edu_payload), headers=headers)

        if edu_response.status_code == 200:
            edu_data = edu_response.json()
            result = edu_data
            result_json_string = util.convert_to_json(result)
            return result_json_string
        else:
            raise HTTPException(status_code=edu_response.status_code, detail="EDU analysis API request failed")
    else:
        raise HTTPException(status_code=seg_response.status_code, detail="Segmentation API request failed")
```

Figure 14: Main Backend Service Route 2

/api/analyze-rest-review

This route takes in a user review, segments it, runs it through the EDU classifier and returns the result. The function ***analyze_text*** receives the input text as a Pydantic object and makes requests to the segmentation and sentiment analysis services in sequence.

```python
@app.post("/api/segment-rest-review")
def segment_text(input_text: InputText = Body(...)):
    url = SEGBOT_URL + "segbot-segment-service"
    payload = {"query": input_text.text}
    headers = {"Content-Type": "application/json"}

    response = requests.post(url, data=json.dumps(payload), headers=headers)

    if response.status_code == 200:
        segs_response = response.json()
        return segs_response
    else:
        raise HTTPException(status_code=response.status_code, detail="API request failed")
```

Figure 15: Main Backend Service Route 3

/api/segment-rest-review

This route takes in a user review, segments it, and returns the segments. The function ***segment_text*** receives the input text as a Pydantic object and makes a request to the segmentation service. The segmented sentences are returned as a JSON object.

6.1.5.3. EDU-Segmentation-API Service

Here, we will discuss the implementation of the route in the EDU-Segmentation-API service. The EDU-Segmentation-API service has one main route, responsible for segmenting input text data into their respective EDU segments as shown in the figure below.

```python
@app.post("/api/segbot-segment-service", response_model=SegmentResult)
def segment_text(input_query: InputQuery = Body(...)):
    segment_result = run_segbot.main_input_output(input_query.query)
    return {"text": input_query.query, "segs": segment_result}
```

/api/segbot-segment-service

This route accepts text input, segments it into a sequence of EDUs, and returns the segmented text. The function ***segment_text*** receives the input data as a Pydantic object and processes it using the ***main_input_output*** function from the ***run_segbot*** module. The segmentation model is based on the paper - SegBot: A Generic Neural Text Segmentation Model with Pointer Network (Li, Sun, & Joty, 2018). The segmentation result is returned as a dictionary containing the ***"text"*** and ***"segs"*** keys.

6.1.5.4. EDU-Sentiment-API Service

Here, we will discuss the implementation of the route in the EDU-Sentiment-API service. The EDU-Sentiment-API service has one main route, responsible for performing sentiment analysis on segmented input data.

```python
@app.post("/api/edu-sentiment-analysis-service", response_model=AnalysisResult)
def edu_sentiment_analysis(input_data: InputData = Body(...)):
    parsed_query_json = server_util.parse_input(input_data)
    raw_analysis_result = store_attention_scores_from_input(parsed_query_json)
    analysis_result = server_util.convert_to_json(data=raw_analysis_result)
    return {"query": analysis_result}
```

Figure 16: EDU-Sentiment-API Service Route

/api/edu-sentiment-analysis-service

This route takes in segmented text data, performs sentiment analysis on the segments, and returns the analysis result. The function ***edu_sentiment_analysis*** receives the input data as a Pydantic object and processes it using the ***parse_input*** function. The parsed data is then passed to the ***store_attention_scores_from_input*** function from the EDU-Attention models to perform the sentiment analysis. The raw analysis result is converted to a JSON object using the ***convert_to_json*** function, and the result is returned as a dictionary containing the "query" key.

6.1.6. API Testing

Here we will be discussing the API testing process that we conducted for the API Gateway. We recognise the importance of API testing to ensure that the services are functioning correctly and that the endpoints provide the expected results. For testing the endpoints, we used Postman, a popular API Testing tool. Postman facilitates sending HTTP requests and analysing the responses, allowing developers to validate the functionality and performance of their APIs easily. For the API Gateway service, we will be testing the three main routes responsible for serving raw data, analysing restaurant reviews, and segmenting user reviews. Examples of these testing are shown in the figures below.

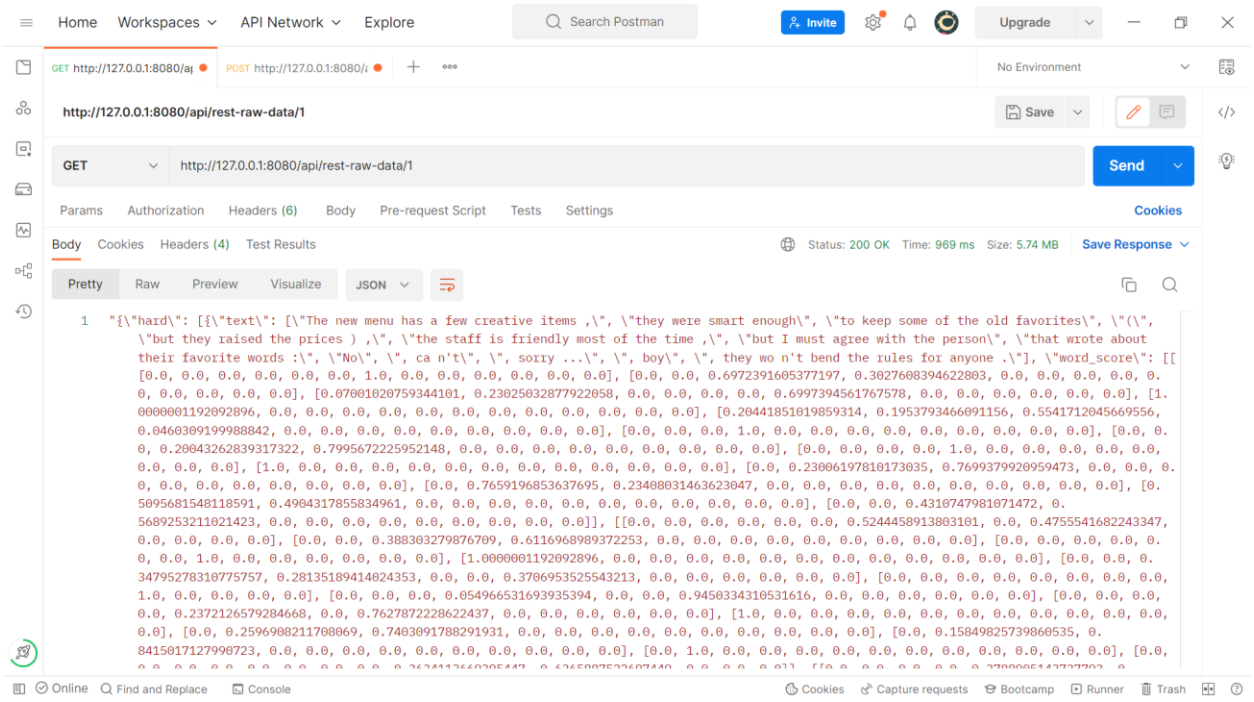


Figure 17: API Testing with Postman for GET /api/rest-raw-data/

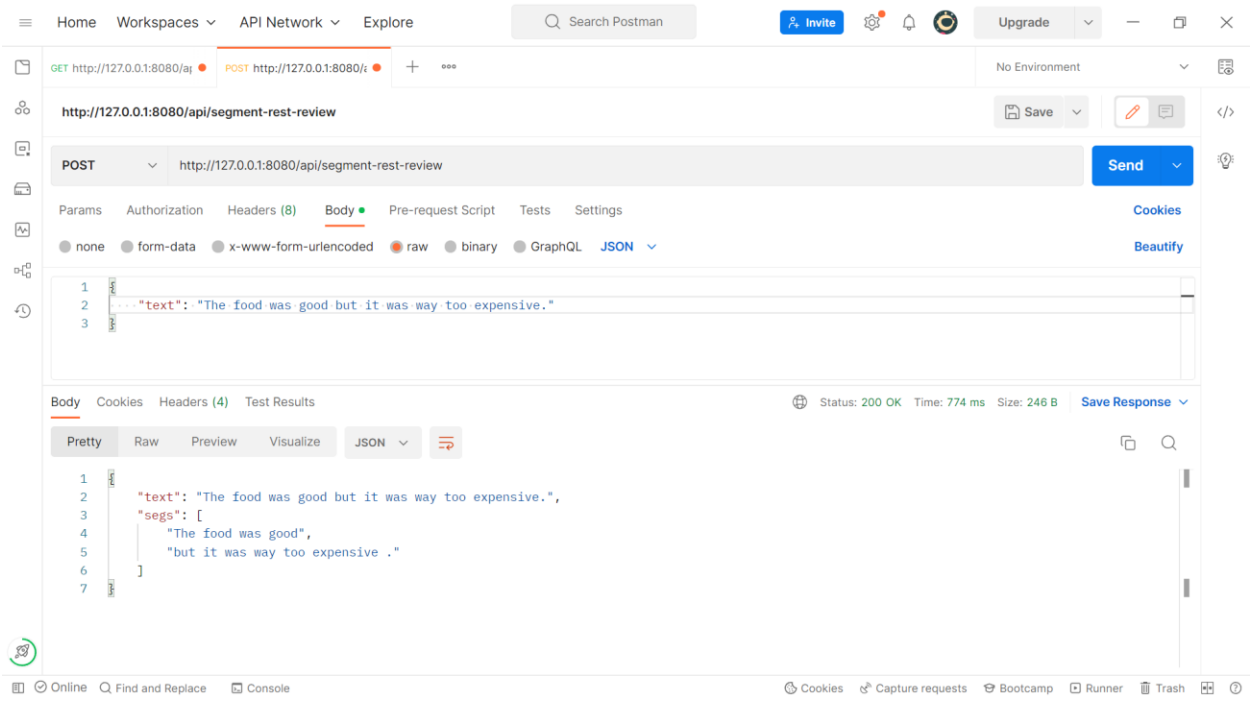


Figure 18: API Testing with Postman for POST /api/segment-rest-review/

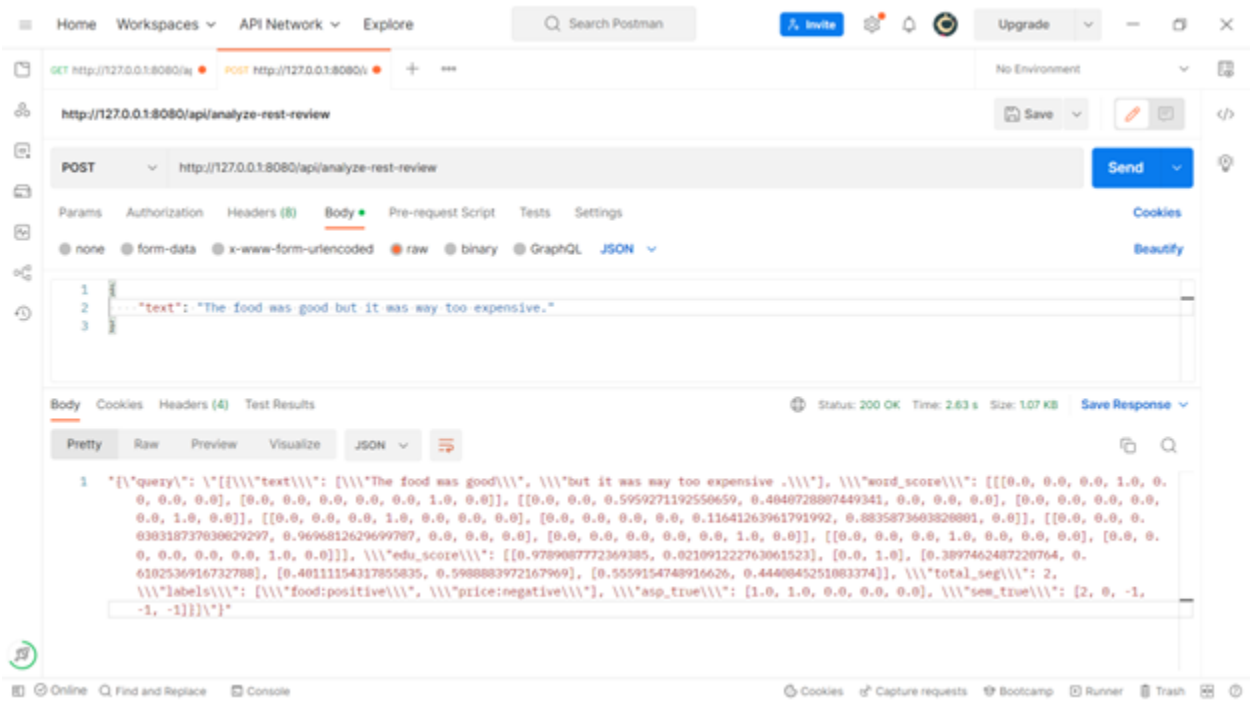


Figure 19: API Testing with Postman for POST /api /analyze-rest-review

6.2. Frontend Development

In this section, we provide an overview of the front-end development implementation for the EDU-Web Interface project. We will cover the installation process for creating a React app and setting up the environment, which includes installing the required libraries. We will also discuss configuring Axios for seamless communication with the back-end services, designing an intuitive user interface for the "Explore," "Analyze," and "Segment" pages, and implementing modular React components used in this web interface. This comprehensive overview offers insights into front-end development, focusing on key technologies and methodologies employed to create this web interface.

6.2.1. Installation and Environment

Given that the front-end framework of choice is React, the command ***npx create -react-app EDU-Web-Interface*** was used to create this React application.

For this project, the technologies and packages used are listed below. These packages are installed using the ***npm install*** command.

Packages

- @emotion/react: 11.10.6
- @emotion/styled: 11.10.6
- @mui/icons-material: 5.11.16
- @mui/material: 5.11.13
- axios: 1.3.2
- bootstrap: 5.2.2
- bootswatch: 5.2.3
- chart.js: 4.2.1
- framer-motion: 10.3.1
- react: 18.2.0
- react-bootstrap: 2.7.1
- react-chartjs-2: 5.2.0
- react-data-table-component: 7.5.3
- react-dom: 18.2.0
- react-highlight-words: 0.20.0
- react-loader-spinner: 5.3.4
- react-router-dom: 6.8.2

- react-scripts: 5.0.1
- styled-components: 5.3.8

6.2.2. HTTP Client Implementation

Axios is a popular JavaScript library that functions as an HTTP client to help facilitate seamless communication between the front-end and back-end services of this web interface. Using Axios will allow us to interact with the API Gateway, which connects us to the other microservices such as the Main Backend Service, EDU-Segmentation-API Service and EDU-Sentiment-API Service. In this project, we implemented this in the **"apiService.js"** component as shown in the figures below.

```
import axios from "axios";

const API_GATEWAY_URL = "http://127.0.0.1:8080/api";
```

Figure 20: Axios Installation in apiService.js

```
const apiService = {
  getResData: async (selectedOption) => {
    try {
      const response = await axios.get(`${API_GATEWAY_URL}/rest-raw-data/1`);
      const jsonResponse = JSON.parse(response.data);

      switch (selectedOption) {
        case "hard":
          return jsonResponse.hard;

        case "test":
          return jsonResponse.test;
        default:
          return null;
      }
    } catch (error) {
      console.error(error);
      return null;
    }
  },
```

Figure 21: getResData in apiService.js

```
postReview: async (inputText) => {
  try {
    const response = await axios.post(
      `${API_GATEWAY_URL}/analyze-rest-review`,
      JSON.stringify({ text: inputText }),
      {
        headers: {
          "Content-Type": "application/json",
        },
      }
    );
    return response.data;
  } catch (error) {
    console.error(error);
    return null;
  }
},
```

Figure 22: postReview in apiService.js

```
  segmentReview: async (inputText) => {
    try {
      const response = await axios.post(
        `${API_GATEWAY_URL}/segment-rest-review`,
        JSON.stringify({ text: inputText }),
        {
          headers: {
            "Content-Type": "application/json",
          },
        }
      );
      return response.data;
    } catch (error) {
      console.error(error);
      return null;
    }
  },
};
export default apiService;
```

Figure 23: segmentReview in apiService.js

The ***apiService*** object contains three methods - ***getResData***, ***postReview***, and ***segmentReview*** - that each makes API calls to different endpoints. The ***getResData*** method accepts a ***selectedOption*** parameter and makes a GET request to the API_GATEWAY_URL endpoint. Depending on the value of the ***selectedOption*** value, it will either return the “hard” or “test” data. The ***postReview*** and ***segmentReview*** method takes an ***inputText*** parameter and sends a POST request to the ***/analyze-rest-review*** and `/segment-rest-review` endpoint respectively. It passes the input text as a JSON string in the request body and sets the 'Content-Type' header to 'application/json'.

In the case of any errors during the API call, the error is logged to the console and returns a null value to ensure that the web interface does not break in the event of any erroneous input. This Axios-based HTTP client implementation simplifies the process of making API requests and handling responses, contributing to the overall maintainability and readability of the front-end code.

The ***apiService*** object contains three methods: ***getResData***, ***postReview***, and ***segmentReview***. Each method makes API calls to different endpoints. The ***getResData*** method accepts a ***selectedOption*** parameter and makes a GET request to the API_GATEWAY_URL endpoint. Depending on the value of ***selectedOption***, it returns either the "hard" or "test" data. The ***postReview*** and ***segmentReview*** methods take an ***inputText*** parameter and send a POST request to the ***/analyze-rest-review*** and ***/segment-rest-review*** endpoints, respectively. The methods send the input text as a JSON string within the request body and set the 'Content-Type' header to 'application/json'.

If any errors occur during the API call, the error is logged to the console, and the method returns a null value to prevent the web interface from breaking. By employing the Axios-based HTTP client implementation, API request execution and response handling are simplified, enhancing the overall maintainability and readability of the front-end code.

6.2.3. User Interface

In this section, we will delve into the implementation of the User Interface (UI) of the EDU-Web Interface for this project. Referencing the wireframes that were introduced earlier, we have designed and built a visually appealing and user-friendly interface that effectively presents the features of the web application which is to give users visibility into the different EDUs of a single sentence or review, as well as facilitate interactive exploration of entire datasets comprising sentences and their respective sentiments toward specific aspects. By focusing on usability, responsiveness, and aesthetics, we aim to create an engaging experience for users while ensuring that the application remains intuitive and easy to navigate.

6.2.3.1. "About" Page

Figure 24: About Page

When a user launches the website, the user will be brought to this "About" page which provides some details of this project which include the relevant information about the academic paper that this project is based on —Aspect-based Sentiment Analysis through EDU-level Attentions. Users can navigate to any of the other pages in the web application by the Navigation Bar at the top.

6.2.3.2. “Explore” Page

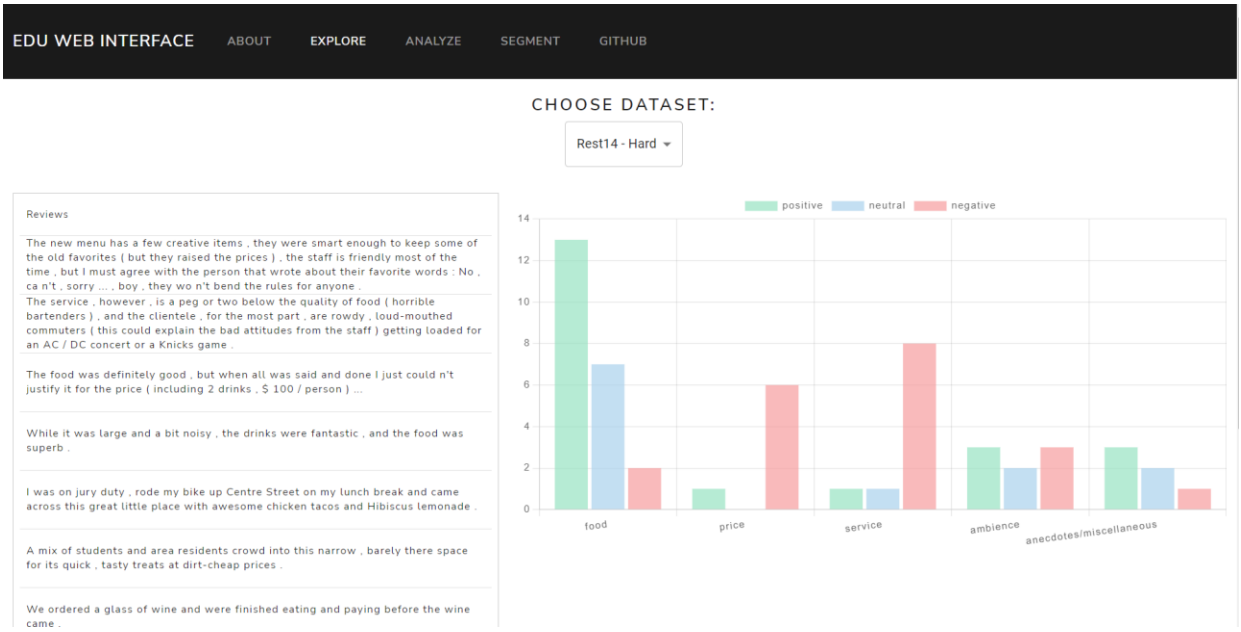


Figure 25: Explore Page

On the "Explore" page, users will find a dropdown menu that allows them to select either the "hard" dataset or the "text" dataset to explore and display. To the left of the page, there is a table containing all the reviews belonging to the chosen dataset. On the right, a chart is displayed, showing the distribution of sentiments toward the five target aspects of a restaurant, in the case of this sample restaurant dataset. This layout enables users to easily explore the dataset while providing a clear visualization of sentiment distribution across the different target aspects.

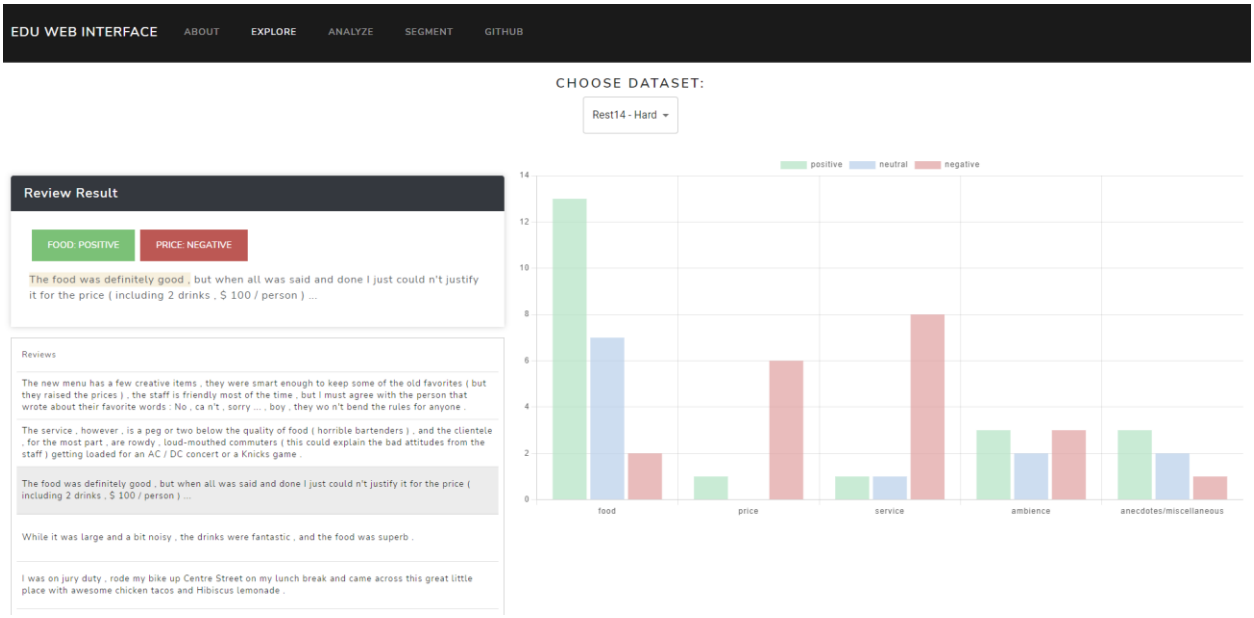


Figure 26: Explore Page - View Sentiment Details

Upon clicking on a review, a card component is displayed that showcases the sentiment details of the selected review. This includes aspect-based sentiments, which are presented as buttons with their corresponding background colours indicating the sentiment type, such as positive, negative, or neutral. When users click on these buttons, the corresponding EDU segment in the review text is highlighted, providing a clear visual connection between the aspect sentiment and its related segment. This intuitive design choice allows users to better understand the analysis results and enhances the overall user experience while interacting with the application.

Figure 27: View Reviews by Aspect and Sentiments - Explore Page

The user interface also features an interactive chart where the bars are clickable. When a user clicks on a specific bar representing a particular aspect-based sentiment, a table appears below the chart displaying the corresponding reviews that contain that sentiment. Within the table, the relevant EDU segments are highlighted, allowing users to quickly identify the specific phrases in the reviews that contribute to the selected sentiment. This feature provides a convenient way for users to delve deeper into the analysis and gain insights into the factors that influence the sentiment of the reviews.

6.2.3.3. “Analyze” Page

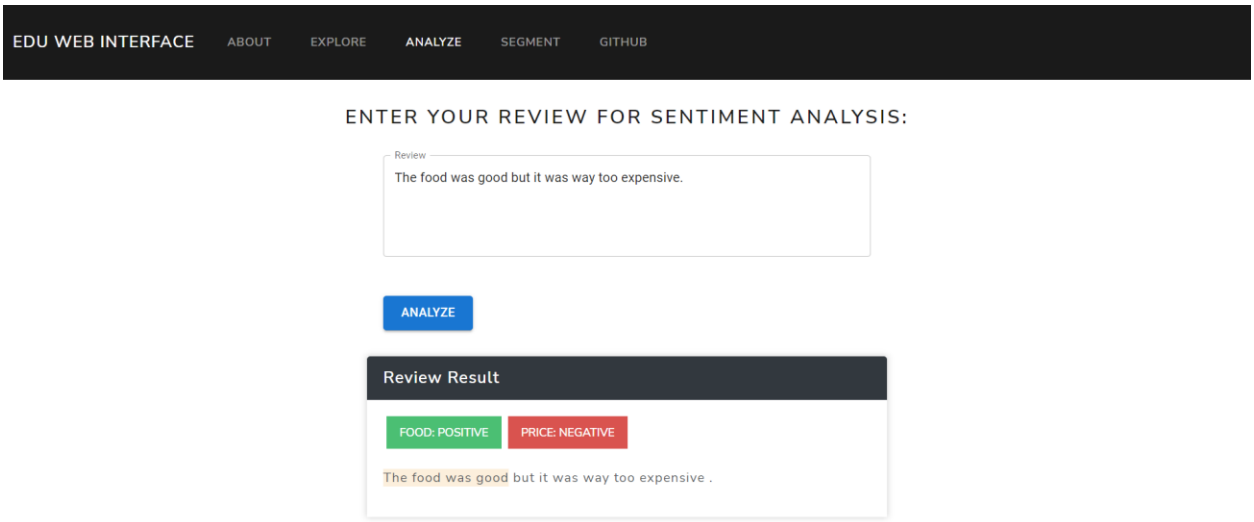


Figure 28: View Sentiment Result - Analyze Page

On the "Analyze" page, users have the opportunity to input a restaurant review of their choice for aspect-based sentiment analysis using a text field provided. After entering the review and clicking the "Analyze" button, a card component appears, displaying the sentiment details of the input review. Similar to the functionality on the "Explore" page, clicking on the buttons within the card component will highlight the respective EDU segment that corresponds to the selected aspect and sentiment. This feature allows users to interactively examine the sentiment analysis results for any restaurant review they choose, enhancing their understanding of the underlying aspects and sentiments.

6.2.3.4. “Segment” Page

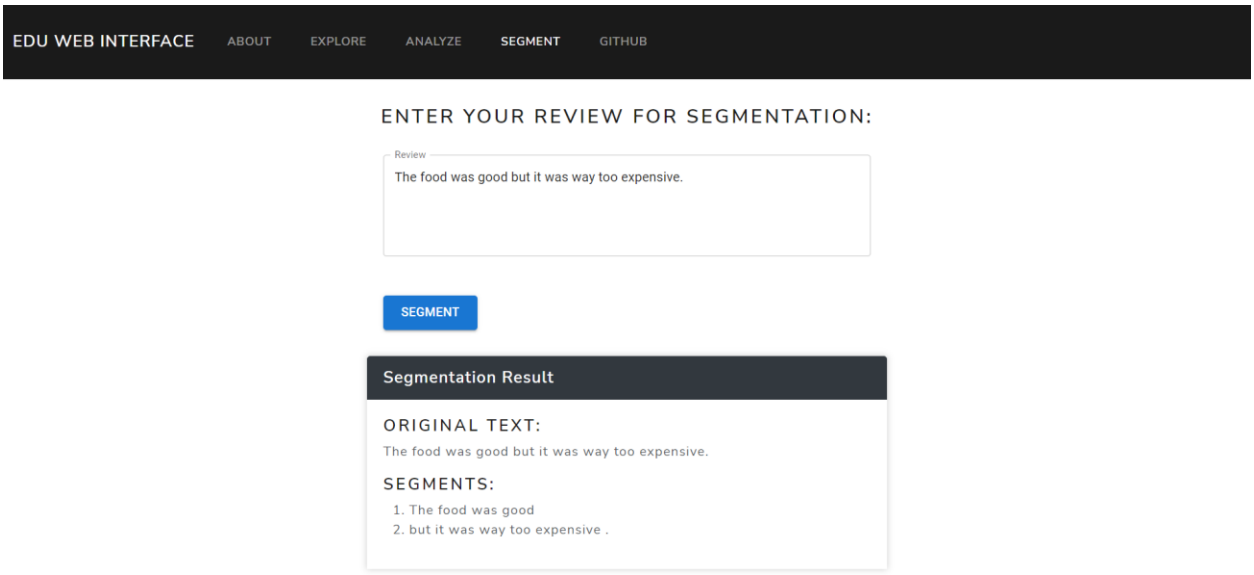


Figure 29: View Segmentation Result - Segment Page

On the "Segment" page, users will find a text field where they can input a review for segmentation. After entering the review and clicking the "Segment" button, the segmentation result is displayed in the form of a card component. This card component is divided into two sections: the first section displays the original text of the input review, while the second section showcases the different EDU segments identified within that particular review in a list format. This feature allows users to understand how the input review has been segmented into various EDU segments, providing insights into the structure and composition of the text.

6.2.4. React Components and Libraries

In React, components serve as the fundamental building blocks, enabling developers to create independent and reusable pieces of code. The reusability of these components guarantees that they can be utilized repeatedly, with more extensive components composed of smaller components.

One of the critical attributes of React components is their ability to maintain a private internal state to preserve data that may change over time. This functionality empowers components to manage their internal state, promoting a more maintainable and modular codebase. Utilizing

components, developers can generate intricate and dynamic user interfaces while maintaining an organized, comprehensible, and extensible codebase.

In this project, we have created 20 components as shown in the figure below.

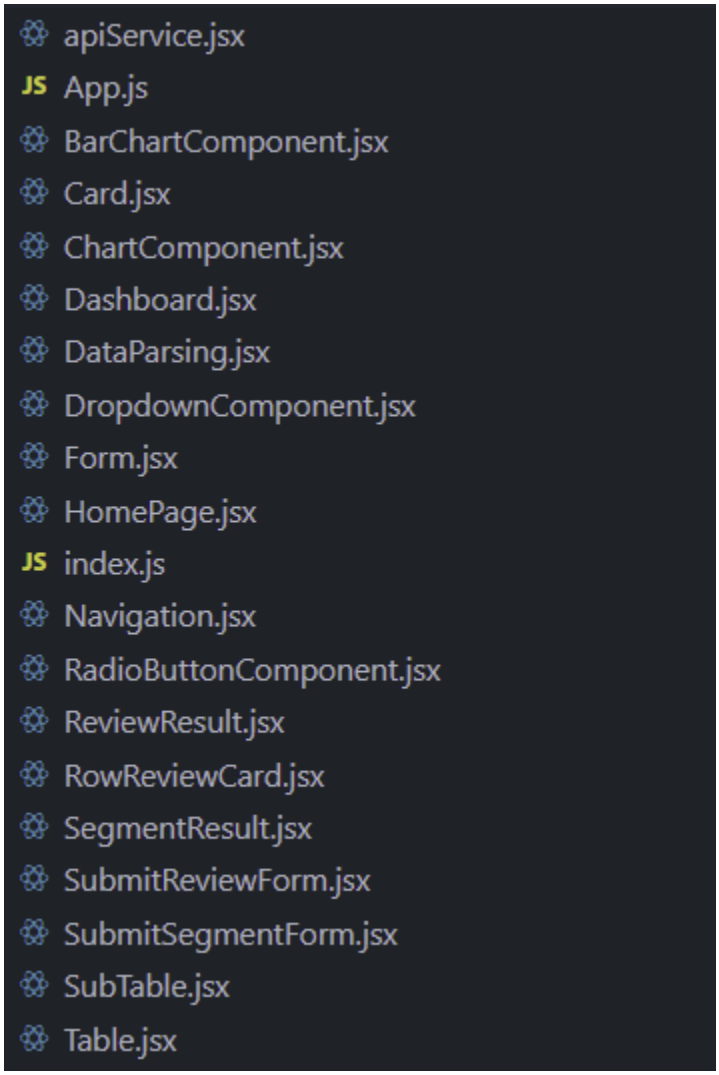


Figure 30: React Components

6.2.4.1. Screenshots of React Components

In the following section, we will showcase the individual components used in this project through a series of figures. These figures will provide a visual representation of each component, demonstrating their roles and interactions within the overall EDU-Web Interface. By examining these components in detail, we can better understand the structure and organization of the project, as well as the importance of each component in delivering a seamless user experience.

1. Navigation Bar Component (***Navigation.jsx***)

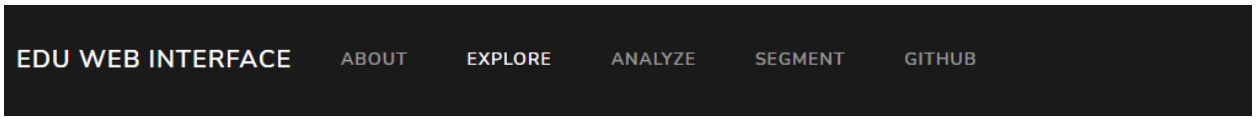
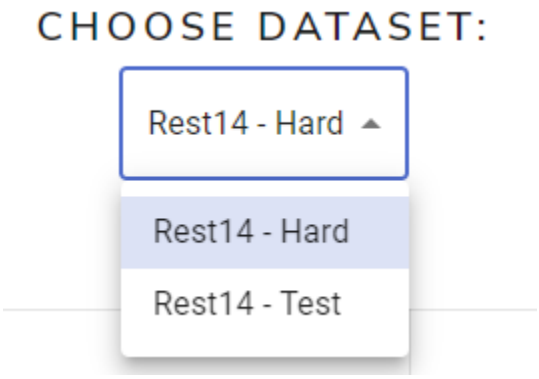


Figure 31: Navigation Bar Component

2. Dropdown Menu Component (***DropdownComponent.jsx***)

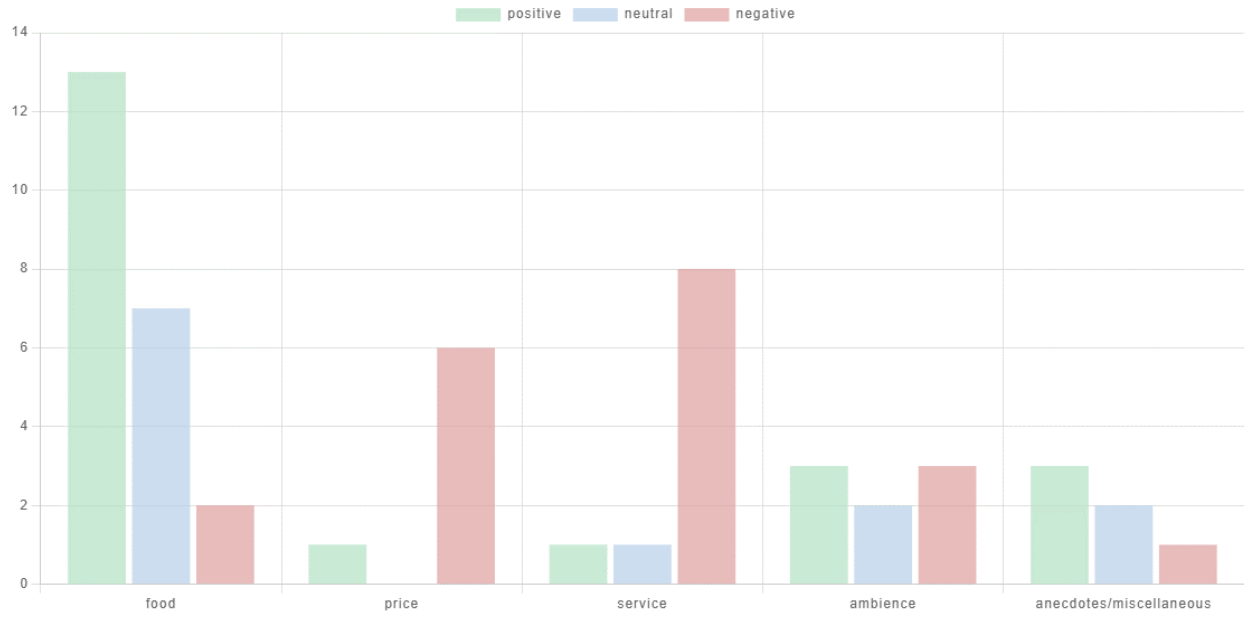


3. Table Component (**Table.jsx**)

Reviews
The new menu has a few creative items , they were smart enough to keep some of the old favorites (but they raised the prices) , the staff is friendly most of the time , but I must agree with the person that wrote about their favorite words : No , ca n't , sorry ... , boy , they wo n't bend the rules for anyone .
The service , however , is a peg or two below the quality of food (horrible bartenders) , and the clientele , for the most part , are rowdy , loud-mouthed commuters (this could explain the bad attitudes from the staff) getting loaded for an AC / DC concert or a Knicks game .
The food was definitely good , but when all was said and done I just could n't justify it for the price (including 2 drinks , $ 100 / person) ...
While it was large and a bit noisy , the drinks were fantastic , and the food was superb .
I was on jury duty , rode my bike up Centre Street on my lunch break and came across this great little place with awesome chicken tacos and Hibiscus lemonade .
A mix of students and area residents crowd into this narrow , barely there space for its quick , tasty treats at dirt-cheap prices .
We ordered a glass of wine and were finished eating and paying before the wine came .
Yes , they 're a bit more expensive then typical , but then again , so is their food .
good place to hang out during the day after shopping or to grab a simple soup or classic french dish over a glass of wine .
While there 's a decent menu , it should n't take ten minutes to get your drinks and 45 for a dessert pizza .

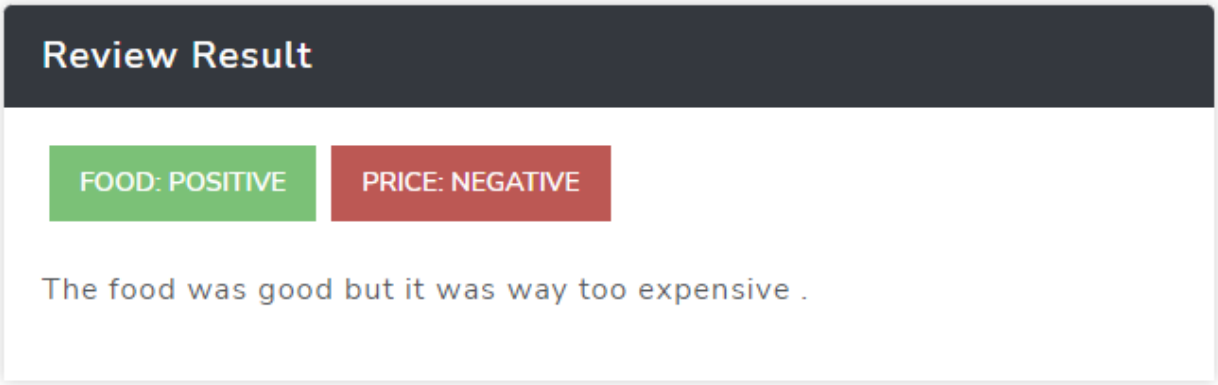
Rows per page: 10 1-10 of 25

4. Bar Chart Component (***BarChartComponent.jsx***)



5. Text Field Component

6. Review Result Component (***ReviewResult.jsx***)



7. Project Difficulties and Learning Outcomes

Throughout the development of the EDU-Web interface, I encountered various challenges and gained valuable insights. In this section, I will discuss the difficulties I faced during the project and the learning outcomes that emerged from overcoming these challenges.

7.1. Project Difficulties

Configuring and managing application logic

One major challenge was configuring and managing the application logic for the EDU-Segmentation-API and EDU-Sentiment-API, based on respective research papers. Understanding their implementations, resolving dependency conflicts, and integrating them with a microservice implementation took considerable time and effort.

Integrating multiple microservices

Managing and integrating various microservices such as API Gateway Service, into a seamless user experience proved to be a challenging task. Ensuring smooth communication between the services and handling errors appropriately required careful planning and implementation.

Designing an intuitive user interface

Creating a user interface that effectively displays complex information while maintaining simplicity and ease of use was another challenge. Balancing aesthetics and functionality required a deep understanding of user needs and an iterative design process.

Adapting to new technologies

The project required me to learn and adapt to new technologies, such as React, Axios, and FastAPI. Gaining proficiency in these tools and applying them effectively took time and effort.

7.2. Learning Outcomes

Problem-solving and critical thinking

Overcoming the project's challenges required creative problem-solving and critical thinking skills. I learned to analyze problems, identify solutions, and make informed decisions throughout the development process.

Technical proficiency

The project provided an opportunity to gain hands-on experience with new technologies and tools, resulting in a deeper understanding of React, Axios, and FastAPI, and improved overall technical proficiency.

User-centric design

A key takeaway from this project was the importance of user-centric design. By focusing on user needs, I created a user interface that effectively met their requirements while maintaining a high level of usability.

8. Future Implementation

In this section, we will explore potential enhancements and extensions for the EDU-Web interface that could be implemented in the future to further improve its functionality and applicability for Aspect-based sentiment analysis.

Extractive Summarisation

A possible future implementation could include integrating an extractive summarization algorithm that identifies and selects the most important sentences or phrases from a review. This would enable users to quickly comprehend the key points of a review without reading the entire text. The algorithm could also generate summary statistics, such as the frequency of specific aspects or sentiments mentioned in the reviews, offering valuable insights for decision-making.

Abstractive Summarisation

Along with extractive summarization, abstractive summarization techniques could be implemented. Abstractive summarization algorithms generate a condensed version of the original text by paraphrasing and restructuring the content, rather than merely extracting important sentences or phrases. This could provide users with a more coherent and concise summary of the reviews, making it easier to understand the overall sentiments of the target aspect present in the dataset.

Integration with Real-time Review Websites

A substantial enhancement to the EDU-Web Interface would be its integration with real-time review websites like Yelp, Google Reviews, or TripAdvisor. This integration would enable users to visualize sentiment analysis of reviews for specific restaurants directly within the EDU-Web Interface application, providing up-to-date information that aids in better decision-making. Additionally, the integration could allow users to receive notifications about new reviews or notable changes in sentiment, empowering them to respond promptly to customer feedback.

9. References

Bootswatch. (n.d.). *Bootswatch*. Retrieved from Bootswatch: https://bootswatch.com/

Daityari, S. (24 March, 2023). *Angular vs react vs Vue: Which framework to choose in 2023*. Retrieved from CodeinWP: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/

Dave, K., Lawrence, S., & Pennock, M. D. (2003). *Proceedings of the twelfth international conference on World Wide Web - WWW '03*, Mining the peanut gallery. doi:10.1145/775152.775226

Gairola, A. (21 March, 2023). *Transform web dev with best frontend frameworks 2023 picks*. Retrieved from Bacancytechnology: https://www.bacancytechnology.com/blog/best-frontend-framework

Jurafsky, D., & Martin, J. H. (2022). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition.* Pearson.

Li, J., Sun, A., & Joty, S. (2018). SegBot: A generic neural text segmentation model with pointer network. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 4166-4172. doi:10.24963/ijcai.2018/579

Lin, T., Sun, A., & Wang, Y. (2022). Aspect-based Sentiment Analysis through EDU-level Attentions. *Advances in Knowledge Discovery and Data Mining*, 156-168.

Material UI. (n.d.). *Material UI*. Retrieved from Material UI: https://mui.com/

Mowlaei, M. E., Saniee Abadeh, M., & Keshavarz, H. (2020). Aspect-based sentiment analysis using adaptive aspect-based lexicons. *Expert Systems with Applications*.

Pang, B., & Lee, L. (01, 2008). Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval, 2*, 1-135. doi:10.1561/1500000011

Podoba, V. (27 July, 2022). *Django vs Flask vs FastAPI for Software Founders*. Retrieved from Softformance: https://www.softformance.com/blog/django-vs-flask/

Postman. (n.d.). *Postman*. Retrieved from Postman: https://www.postman.com/

Pydantic. (n.d.). *Schema*. Retrieved from https://docs.pydantic.dev/usage/schema/

React Data Table Component. (n.d.). Retrieved from https://react-data-table-component.netlify.app/

react-chartjs-2. (n.d.). Retrieved from https://react-chartjs-2.js.org/examples/vertical-bar-chart

Remix Software, Inc. (n.d.). *React Router*. Retrieved from React Router: https://reactrouter.com/en/main

Samiullah, C. (16 July, 2021). *The Ultimate FastAPI Tutorial Part 4 - Pydantic Schemas*. Retrieved from https://christophergs.com/tutorials/ultimate-fastapi-tutorial-pt-4-pydantic-schemas/

Sandy, J. (4 January, 2021). *Choosing between Django, Flask, and FastAPI*. Retrieved from Section: https://www.section.io/engineering-education/choosing-between-django-flask-and-fastapi/

styled-components. (n.d.). Retrieved from https://styled-components.com/

Tang, D., Qin, B., & Liu, T. (2016). Aspect Level Sentiment Classification with Deep Memory Network. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 214-224.

Verma, Y. (25 September, 2021). *Django vs Flask VS fastapi - A comparative guide to python web frameworks*. Retrieved from Analytics India Magazine: https://analyticsindiamag.com/django-vs-flask-vs-fastapi-a-comparative-guide-to-python-web-frameworks/