НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

на тему: Програмний інтерфейс тематичного моделювання тексту
Виконав : студент 4 курсу, групи ІВ-91
(шифр групи)

_____Гришин Олексій Сергійович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____ асистент Кочура Ю.П._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) ст. викладач, Виноградов Ю.М._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2023 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“ Комп’ютерні системи та мережі ”

спеціальність 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2023 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Гришина Олексія Сергійовича

1. Тема проєкту *Програмний інтерфейс тематичного моделювання тексту* керівник проєкту Кочура Ю.П. асистент,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

2. Термін здачі студентом закінченого проєкту *07 червня 2023 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Літературний огляд проблеми
Розділ 2. Набір даних та ознаки для навчання.
Розділ 3. Реалізація програмного інтерфейсу.
Розділ 4. Експерименти та результати.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, функціональна схема (діаграма даних системи), алгоритм роботи системи.

6. Консультанти проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Виноградов Ю.М.		

7. Дата видачі завдання "30" листопада 2022 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	29.11.2022-30.11.2022	
2.	*Вивчення та аналіз сучасного стану проблеми. Опис сильних та слабких сторін існуючих підходів для тематичного моделювання*	01.05.2023-02.05.2023	
3.	*Вибір програмних засобів реалізації інтерфейсу тематичного моделювання тексту*	03.05.2023-04.05.2023	
4.	*Пошук навчальних даних та вибір методів для тематичного моделювання тексту*	05.05.2023-06.05.2023	
5.	*Реалізація програмного інтерфейсу*	07.05.2023-09.05.2023	
6,	*Проведення та опис експериментів, представлення та аналіз отриманих результатів*	10.04.2023-12.05.2023	
7.	*Оформлення пояснювальної записки*	13.04.2023-18.05.2023	
8.	*Передзахист*	01.06.2023-09.06.2023	
9.	*Захист*	21.06.2023	

Студент-дипломник _____ Олексій ГРИШИН
(підпис)

Керівник проєкту _____ Юрій КОЧУРА
(підпис)

Анотація

Дипломний проект на тему "Програмний інтерфейс тематичного моделювання тексту" присвячений створенню програмного інтерфейсу (API), який дозволить розбити текстові дані на тематичні кластери за допомогою алгоритмів машинного навчання. Мотивацією цього проекту слугує користь від обробки інформації як дозволить вирішувати проблеми в сферах пов’язаних з генерацією та аналізом великої кількості текстів (статті, реклама, інтерв’ю, дослідження).

Для вирішення проблеми в основі проекту використовується алгоритм латентного розміщення Діріхле. Він був обраний бо є найбільш оптимальним та відповідно популярним підходом для таких задач. Експерименти проводились на наборі статей журнала «Reuters» з використанням тестового та тренувального наборів даних розміру 5484 та 12528 текстів відповідно.

Ключові слова: *Python, FastAPI, SKlearn, nltk, LDA*

Annotation

A diploma project on the topic of "API-based text topic modeling" aims to create a program interface (API) that enables the categorization of text data into thematic clusters using machine learning algorithms. The motivation behind this project is the benefit of processing information to solve problems in areas related to generating and analyzing a large amount of texts (articles, advertisements, interviews, research).

To address this issue, the project uses the Dirichlet Latent Allocation algorithm, which was chosen because it is the most optimal and popular approach for such tasks. Experiments were conducted on the "Reuters" journal article dataset using test and training data sets of sizes 5484 and 12528 texts, respectively.

Keywords: *Python, FastAPI, SKlearn, nltk, LDA*

Поз.	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпляра
			Документація загальна		
			розроблена по новому		
1	А4	ІАЛЦ.467200.001 ОА	Опис альбому	1	
2	А4	ІАЛЦ.467200.002 ТЗ	Програмний інтерфейс	3	
			тематичного		
			моделювання тексту		
			Технічне завдання		
3	А4	ІАЛЦ.467200.003 ПЗ	Програмний інтерфейс	59	
			тематичного		
			моделювання тексту		
			Пояснювальна записка		
4	А4	ІАЛЦ.467200.004 Д1	Структурна схема	1	
5	А4	ІАЛЦ.467200.005 Д2	Функціональна схема	1	
6	А4	ІАЛЦ.467200.007 Д3	Алгоритм роботи	1	
7	А4	ІАЛЦ.467200.007 Д4	Текст програмного коду		

					ІАЛЦ.467200.001 ОА
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*	
Розроб.		*Гришин О.С.*			***Програмний інтерфейс тематичного моделювання тексту*** ***Опис альбому***
Перевір.		*Кочура Ю.П.*			

Літ.	*Арк.*	*Акрушів*
	1	

КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-91

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Програмний інтерфейс тематичного моделювання тексту*»

Київ – 2023 р.

ЗМІСТ

					ІАЛЦ.467200.002 ТЗ		
		№ докум.	Підпис	Дата			
Розробив		Гришин О.С			*Система управління робочими процесами на підприємстві* *Технічне завдання*	Літ.	Аркуш 1 / Аркушів 3
Перевірив		Кочура Ю.П.					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-91	
Н. Контр.		Виноградов Ю.М.					
Затвердив							

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Данне технічне завдання поширюється на розробку й підтримку системи тематичного моделювання тексту, а також подальшу експлуатацію та покращення розробленої системи.

Системи тематичного моделювання тексту знаходять широке застосування для автоматичної ідентифікації тематичних ключових слів та кластерів документів для покращення пошуку в інформаційних системах. Системи тематичного моделювання можуть допомогти виявити та проаналізувати специфічні інтереси та потреби користувачів, щоб оптимізувати рекламні кампанії та персоналізовані пропозиції.

Системи тематичного моделювання можуть допомогти аналізувати та класифікувати новини та інші засоби масової інформації в реальному часі, що дозволяє користувачам швидко та ефективно отримувати актуальну інформацію.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп’ютерної інженерії», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут імені Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є проектування та розробка системи тематичного моделювання тексту.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного проекту є форуми, офіційна документація, статті та література в мережі Інтернет на дану тему.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

- Простий і зрозумілий програмний інтерфейс.
- Підтримка документації Swagger.
- Підтримка авторизації та аутентифікації.

5.2. Вимоги до програмного забезпечення

- ОС Windows 10, Mac чи Linux.
- Браузер

5.3. Вимоги до апаратної частини

- ЦП Dual core from Intel or AMD at 2.8 GHz
- 4 GB RAM
- Відеокарта NVIDIA GeForce 8600/9600GT, ATI/AMD Radeon HD2600/3600

6. ЕТАПИ РОЗРОБКИ

Найменування етапів дипломного проєкту	Терміни
Затвердження теми проєкту	30.11.2022
Вивчення та аналіз сучасного стану проблеми. Опис сильних та слабких сторін існуючих підходів для тематичного моделювання	02.05.2023
Вибір програмних засобів реалізації інтерфейсу тематичного моделювання тексту	04.05.2023
Пошук навчальних даних та вибір методів для тематичного моделювання тексту	06.05.2023
Реалізація програмного інтерфейсу	09.05.2023
Проведення та опис експериментів, представлення та аналіз отриманих результатів	12.05.2023
Оформлення пояснювальної записки	18.05.2023

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Програмний інтерфейс тематичного моделювання тексту*»

Київ – 2023 р.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

LDA	(latent Dirichlet allocation) Латентне розміщення Діріхле
NLP	(natural language processing) Обробка природньої мови
NMF	(Non-negative Matrix Factorization) Невід'ємне матричне розкладання
BoW	(bag of words) Мішок слів
TF-IDF	(Term Frequency-Inverse Document Frequency) частота слова - обернена частота документа

ВСТУП

В сучасному світі кількість доступної інформації надзвичайно велика, і її обробка та аналіз стає все складнішою задачею. Системи тематичного моделювання тексту дозволяють автоматизувати цей процес, що забезпечує ефективнішу роботу з великим обсягом текстової інформації.

Основні причини розробки цього проекту, а отже і мотивація до його розробки полягає у тому, що велика кількість підприємств та наукових установ потребують аналізу великого обсягу текстових даних, таких як наукові публікації, звіти, новини та інші документи. Системи тематичного моделювання дозволяють автоматично класифікувати ці дані за темами та забезпечують ефективний інструмент аналізу та пошуку інформації. Наприклад, в науковій галузі система може допомогти в обробці великої кількості наукових статей, аналізувати публікації та сприяти пошуку нових наукових досліджень.

У сфері медицини система може бути використана для аналізу медичних записів та допомогти в розробці нових лікарських препаратів. В журналістській галузі система може допомогти аналізувати новини та статті, виявляти тенденції та проблеми в різних галузях.

Для вирішення проблеми в основі проекту використовується алгоритм латентного розміщення Діріхле. Вхідними даними нашого алгоритму є CSV файл що надходить до модулів обробки тексту через програмний інтерфейс. В вхідному файлі повинна построково зберігатися текстова інформація, яку ми хочему розбити на теми.

Потім ми використали алгоритми попередньої обробки тексту для очищення вхідних даних, і подали очищенні дані до модулю що використовує латентне розміщення Діріхле, щоб вивести розподіл текстів по тематиками.

Як результат отримаємо навчену модель для подальшої обробки нових текстів з тестового набору, а також візуалізація розбиття текстів по тематикам.

Отже, у сучасному світі обробка та аналіз великого обсягу інформації стає складною задачею. Системи тематичного моделювання тексту надають ефективний інструмент для автоматизації цього процесу. Проект, що базується на алгоритмі латентного розміщення Діріхле, має на меті розробити систему тематичного моделювання для класифікації текстових даних за темами.

Цей проект має великий потенціал застосування у різних галузях. Наприклад, в науковій сфері він може допомагати аналізувати наукові статті і сприяти пошуку нових досліджень. У медицині система може використовуватися для аналізу медичних записів та допомоги в розробці нових лікарських препаратів. Журналістам вона може надавати засоби для аналізу новин та виявлення тенденцій і проблем.

РОЗДІЛ 1
ЛІТЕРАТУРНИЙ ОГЛЯД ПРОБЛЕМИ ТА АНАЛІЗ СУЧАСНИХ ПІДХОДІВ ДО ТЕМАТИЧНОГО МОДЕЛЮВАННЯ ТЕКСТУ

1.1 Аналіз вимог до систем що використовують тематичне моделювання тексту

Велика кількість підприємств та наукових установ потребують аналізу великого обсягу текстових даних, таких як наукові публікації, звіти, новини та інші документи. Системи тематичного моделювання дозволяють автоматично класифікувати ці дані за темами та забезпечують ефективний інструмент аналізу та пошуку інформації.

Основною задачею проекту є розробка програмного інтерфейсу (API) для тематичного моделювання тексту, що допоможе автоматично класифікувати документи за темами та забезпечити зручний інтерфейс для взаємодії з користувачем. Перш за все система повинна використовувати методи машинного навчання для знаходження тематичних зв'язків в тексті.

Система повинна забезпечувати обробку тексту, включаючи вилучення стоп-слів, лематизацію, токенізацію та інші операції, що забезпечують ефективність тематичного моделювання.

Очевидно що система повинна здійснювати кластеризацію текстів на основі їхніх тематичних зв'язків, для цього система повинна використовувати різні алгоритми для тематичного моделювання, наприклад, LDA (Latent Dirichlet Allocation) або NMF (Non-negative Matrix Factorization).

Отже можемо коротко задокументувати основні структурні елементи нашої системи:

1. Машинне навчання
2. Попередні обробники тексту
3. Кластеризатор тексту

4. Алгоритми
5. API

Для комунікації з користувачем система повинна мати добре задокументоване API для інтеграції з іншими програмами та інтерфейсами користувача.

1.2 Аналіз наявних алгоритмів що підходять для вирішення проблеми

Машинне навчання

Існує безліч методів та алгоритмів машинного навчання, які можуть бути використані для тематичного моделювання тексту. Використання машинного навчання дозволяє забезпечити високу точність та ефективність розпізнавання тематики тексту. [1]

Машинне навчання - це підхід до розробки програм, в яких комп'ютер самостійно навчається на основі даних. Застосування машинного навчання для тематичного моделювання тексту - це один з найбільш популярних методів для аналізу великих обсягів даних. Тематичне моделювання використовується для автоматичної категоризації текстових даних на основі статистичного аналізу.

Основним підходом для виконання цієї задачі є методи без учителя (unsupervised learning). В методах навчання без учителя в нас немає попередньо відмічених даних (наприклад як це зазвичай може бути позитивний та негативний контент в сентиментальному аналізу), тому мета алгоритмів навчання без учителя полягає в пошуку певних закономірностей в даних. Навчання без учителя вирішує в основному задачі кластеризації різного контенту, а також пошуку певних асоціативних правил.

Використання методів без учителя дозволяє зменшити вплив певних апріорних знань про об'єкти аналізу, а також дозволяє більш точно інтерпретувати результати аналізу. [2]

Найпопулярнішим методом без учителя для тематичного моделювання тексту є LDA. LDA використовується для виявлення тем, які згруповані разом в документах. Цей метод працює шляхом розкладання документів на окремі слова та приховані теми. Задача полягає в тому, щоб знайти набір тем та їхні відношення до кожного документу.

Також, машинне навчання може використовуватись для оптимізації процесу тематичного моделювання. Це може бути досягнуто шляхом використання автоматичної підбору оптимальних гіперпараметрів моделі, що є завданням оптимізації.

Обробка тексту є ключовою компонентою систем тематичного моделювання. Вона забезпечує перетворення тексту з формату слова на числовий вектор, який можна використовувати для побудови моделі тематичного моделювання.

Одним з основних методів обробки тексту є побудова векторів слів, що дозволяє перетворити слова в числа, які можна подавати на вхід моделі машинного навчання. Цей метод полягає в побудові вектора, який представляє кожне слово в тексті, і забезпечує збереження інформації про контекст, в якому слово використовується в тексті. Існує кілька методів побудови векторів слів, таких як Bag-of-Words, TF-IDF та Word2Vec.

Bag-of-Words (BoW) - це простий метод побудови векторів слів, в якому кожне слово в тексті розглядається як окремий термін, а вектор слів складається зі збільшення значень відповідних елементів, в залежності від того, скільки разів кожне слово з'являється у тексті.

TF-IDF (Term Frequency-Inverse Document Frequency) є більш складним методом побудови векторів слів, який дозволяє врахувати інформацію про частоту зустрічання слів в тексті та їх вагу в тексті. Цей метод приділяє меншу вагу словам, які зустрічаються в багатьох документах, і більшу вагу словам, які зустрічаються в конкретному документі.

Word2Vec є високорівневим методом побудови векторів слів, який дозволяє ураховувати контекст, в якому використовується кожне слово. Цей метод навчає нейромережу, яка прогнозує ймовірність того, що слово з'явиться в контексті інших слів, що знаходяться в документі.

Іншим методом попередньої обробки тексту який часто використовують є стемінг та лемматизація (більше комплексна форма стемінгу). Стемінг є однією з технік обробки тексту, яка використовується для зменшення форм слова до їх основних (стемів) за допомогою відкидання закінчень і афіксів. Наприклад, слова "книга", "книгами", "книгарня" після стемінгу будуть мати спільний стем "книг".

Стемінг є важливою складовою в задачах обробки природної мови, таких як тематичне моделювання тексту. Використання стемінгу дозволяє зменшити кількість унікальних слів у тексті, що полегшує розрахунок статистичних характеристик, таких як частоти вживання слів і збігів у вхідних даних.

Існує декілька алгоритмів стемінгу, які можуть бути використані для різних мов. Наприклад, для англійської мови часто використовують алгоритми Портера та Snowball (сніжний шарик).

Важливо зазначити, що стемінг може мати деякі недоліки. Через те що він зменшує слова до їх стемів, він може призводити до втрати частини інформації про слово. Наприклад, слова "бібліотека" та "бібліотечний" після стемінгу матимуть спільний стем "бібліотек", інформація про те, що слово "бібліотечний" відноситься до слова "бібліотека" буде втрачена.

Тому, стемінг варто використовувати з розумінням та у відповідних випадках.

Інший випадок це лематизація – більш складна і досконаліша форма приведення слів до роднієї форми. Лематизація є процесом обробки тексту, що використовується для зведення слів до їхніх нормалізованих форм, які

називаються лемами. Лема - це форма слова, яка відповідає його базовій формі, або іншим словам тієї ж самої основи.

Однією з основних відмінностей між стемінгом та лематизацією є те, що стемінг зазвичай використовує прості правила для відсікання закінчень слів, тоді як лематизація використовує більш складні алгоритми, щоб визначити кореневу форму слова.

Наприклад, у слова "бігти", "біг", "бігла", "бігло" та "бігали" лемою буде "бігти". Важливо враховувати, що лематизація не зводить слова до стовпчика нормалізації, як це робить стемінг. Лематизація може враховувати контекст слова та змінні форми, що забезпечує більш точні результати.

Лематизація є корисним інструментом в обробці природньої мови, оскільки вона дозволяє зменшити розмірність тексту, зведення слів до однієї леми. Вона також може бути використана в тематичному моделюванні, щоб знизити кількість слів в документі та покращити якість кластеризації.

Проте, варто зазначити, що лематизація може бути вимогливою до ресурсів та обчислювальної потужності, оскільки вона використовує складні алгоритми та довгий список лексичних ресурсів для визначення лем.

1.3 Аналіз існуючих систем що використовують в собі тематичне моделювання тексту

Google News

Google News: використовує тематичне моделювання для категоризації новинних статей за темами.

Рисунок 1.1 – Логотип Google News

Google News є одним з найбільш популярних новинних агрегаторів у світі, який забезпечує користувачам останні новини з різних джерел та з різних країн. Це можливо завдяки використанню технологій машинного навчання та тематичного моделювання для збирання, фільтрації та групування новин. [3]

Однією з ключових функцій Google News є тематичний аналіз новинних статей. Система Google News використовує тематичне моделювання для групування новин за темами. Вона аналізує новини з різних джерел та призначає їм тематичну категорію на основі змісту статті. Це дозволяє користувачам швидко знайти новини на певну тему та отримувати релевантну інформацію з різних джерел.

Google News використовує алгоритм LDA, який є одним з найбільш поширених алгоритмів тематичного моделювання. LDA аналізує текстові документи та виділяє тематичні кластери, які відображають різні теми, що містяться в тексті. Кожен кластер відповідає конкретній темі, а слова, які входять до кластеру, відображають ключові слова, пов'язані з цією темою.

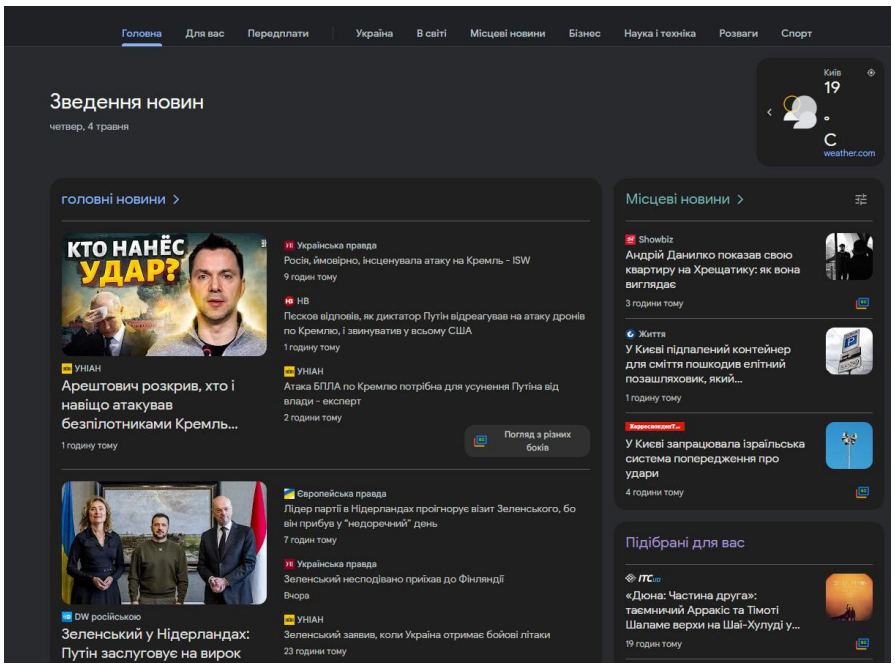


Рисунок 1.2 – Інтерфейс Google News [3]

За допомогою технологій тематичного моделювання Google News не тільки групує новини за темами, але також виконує інші завдання, такі як автоматичний підбір заголовків для статей, відстеження новин за певними ключовими словами та темами, а також виявлення тенденцій та змін в новинах.

BuzzSumo

BuzzSumo - це компанія, яка надає інструменти для аналізу соціальних медіа та веб-контенту. Один з їх основних інструментів - це функція тематичного моделювання, яка дозволяє користувачам знайти ключові теми та тренди, які популярні серед користувачів соціальних медіа. [4]

BuzzSumo використовує тематичне моделювання, щоб аналізувати текстові дані з мільйонів соціальних медіа публікацій та статей з Інтернету. Цей аналіз допомагає виявляти ключові теми та тренди, які популярні серед користувачів соціальних медіа. Наприклад, якщо користувач вводить ключове слово в інструмент BuzzSumo, він може отримати перелік найпопулярніших публікацій за цією темою. Це дає користувачам можливість дізнатися, які теми

найбільш популярні, щоб вони могли створювати вміст, який відповідає запитам своєї аудиторії.

Для того, щоб провести тематичний аналіз, BuzzSumo використовує алгоритм Latent Dirichlet Allocation (LDA). Цей алгоритм дозволяє виявляти складові тематичного моделювання та розподіляти слова в тексті між ними. В результаті аналізу BuzzSumo можна отримати інформацію про найбільш популярні теми та тренди в соціальних медіа.

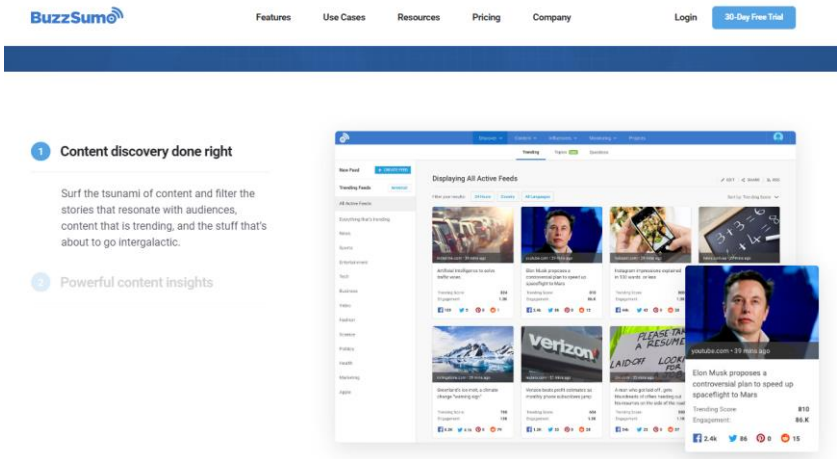


Рисунок 1.3 – Приклад розбиття на тематики сервісом buzzsumo

Таким чином, BuzzSumo використовує тематичне моделювання, щоб допомогти користувачам виявляти ключові теми та тренди, які популярні серед користувачів соціальних медіа. Ця інформація може бути корисною для створення ефективної маркетингової кампанії.

Thomson Reuters

Thomson Reuters - це міжнародна компанія, яка надає різноманітні послуги у сфері фінансів, юриспруденції, наукових досліджень та інформаційних технологій. Компанія використовує технології тематичного моделювання для аналізу даних та отримання корисної інформації. [5]

Одним з прикладів використання тематичного моделювання в Thomson Reuters є їх платформа для аналізу новин News Analytics. Ця платформа

використовує технології машинного навчання та тематичного моделювання для аналізу тисяч новинних статей з усього світу та отримання цінної інформації про ринки та інші галузі.

Thomson Reuters використовує алгоритми тематичного моделювання, щоб автоматично визначити тематику та зміст кожної новинної статті. Аналізуючи дані, платформа News Analytics може зрозуміти, які новини є найбільш важливими для ринків та інших галузей, які тренди є актуальними та які події можуть вплинути на ринки.

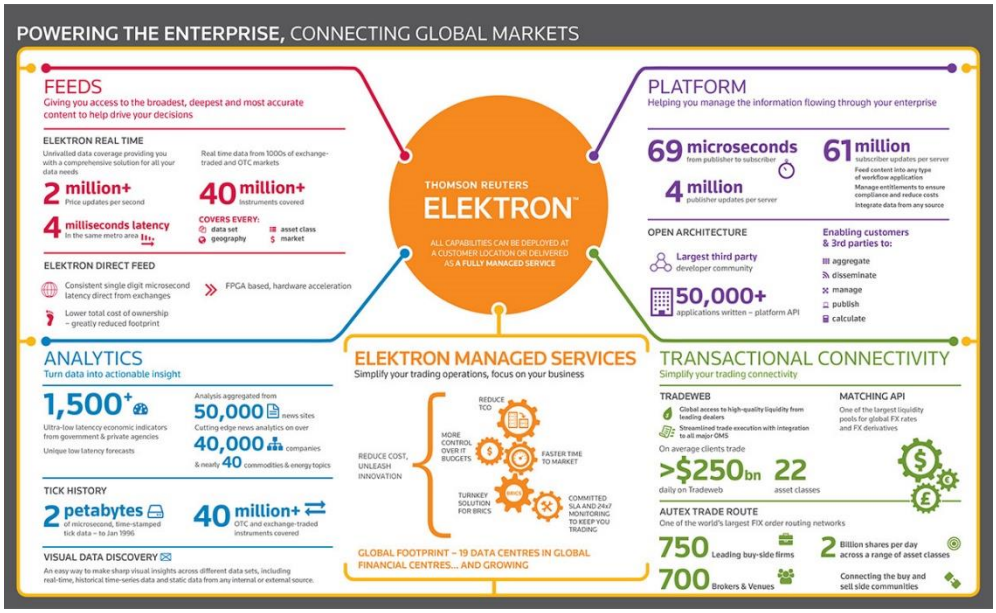


Рисунок 1.4 – сервіс аналізу ринку від Thomson Reuters [6]

Ці дані можуть бути корисні для різних груп користувачів, включаючи фінансових аналітиків, трейдерів та інші особи, які мають інтерес до фінансових ринків та інших галузей. Наприклад, фінансові аналітики можуть використовувати дані з платформи News Analytics, щоб побачити, які новини можуть вплинути на ціни акцій певних компаній чи інших фінансових інструментів.

Таким чином, Thomson Reuters використовує технології тематичного моделювання, щоб допомогти своїм користувачам отримати цінну інформацію з тисяч джерел торгової інформації.

ВИСНОВОК ДО 1 РОЗДІЛУ

Перший розділ дипломного проєкту присвячений аналітичному огляду сучасних систем тематичного аналізу тексту. В цьому розділі були досліджені вимоги, які ставляться перед подібними системами. Було розглянуто основні алгоритми та підходи до обробки тексту та навчання моделей.

Аналізуючи сучасні підходи до тематичного моделювання тексту, можна зробити висновок про значний потенціал цієї галузі та її застосування у багатьох сферах життя. Системи тематичного аналізу тексту дозволяють ефективно класифікувати та розбивати текстову інформацію за темами, що допомагає в роботі з великим обсягом даних.

Застосування таких систем може бути корисним у багатьох галузях. Наприклад, в науковій сфері системи тематичного моделювання тексту можуть допомагати в обробці наукових статей та сприяти пошуку нових наукових досліджень. У медицині такі системи можуть використовуватись для аналізу медичних записів та підтримки розробки нових лікарських препаратів. Журналісти можуть використовувати системи тематичного аналізу тексту для аналізу новин та статей, виявлення тенденцій та проблем в різних галузях.

Таким чином, перший розділ дипломного проєкту пропонує огляд сучасних систем тематичного аналізу тексту, висвітлює основні алгоритми та підходи до обробки тексту та навчання моделей і підкреслює значення цієї галузі в різних сферах життя.

РОЗДІЛ 2
ВИБІР АРХІТЕКТУРИ ТА ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ СИСТЕМИ ТЕМАТИЧНОГО МОДЕЛЮВАННЯ ТЕКСТУ

Згідно з Технічним завданням на дипломний проект для розробки дипломного проекту була обрана мова програмування Python. Подальший розгляд обраних технологій буде спиратися саме на цю мову.

Дослідження різних архітектур та програмних засобів реалізації системи тематичного моделювання тексту є важливим етапом у розробці такої системи. Це дає можливість визначити найбільш оптимальний варіант для певної задачі, а також дозволяє порівняти різні підходи та знайти найбільш ефективний з них.

Наприклад, у випадку вибору технологій розробки серверної частини, можливість порівняти швидкість роботи різних серверних фреймворків та вибрати той, який найбільш відповідає вимогам до продуктивності та масштабованості системи.

У випадку вибору технологій NLP, можливість порівняти різні алгоритми лематизації та стемінгу тексту та вибрати той, який дає найкращі результати для конкретної задачі. Також можна порівняти різні алгоритми тематичного моделювання та вибрати той, який найбільш точно відображає сутність текстів та дає найбільш точні прогнози.

Крім того, дослідження різних архітектур та програмних засобів дозволяє визначити найкращу комбінацію засобів для конкретної задачі та підвищити ефективність роботи системи.

Отже, дослідження різних архітектур та програмних засобів реалізації системи тематичного моделювання тексту є важливим етапом у розробці такої системи, що дозволяє визначити найбільш оптимальний варіант та підвищити ефективність її роботи.

2.1 Вибір технологій розробки серверної частини

Для розробки API був обраний мікрофреймворк FastAPI для мови програмування Python. Також для валідації вхідних запитів була використана бібліотека Pydantic. Для документування програмного інтерфейсу я використовував веб-документацію Swagger. В якості тестового веб-серверу на етапі розробки було використано uvicorn з можливістю подальшої міграції на Nginx.

FastAPI

FastAPI - це фреймворк для створення високопродуктивних веб-додатків та API, написаний на мові програмування Python. Він забезпечує швидку та просту розробку додатків, що працюють з HTTP API. FastAPI заснований на сучасних технологіях Python, таких як ASGI (асинхронний сервер Python), типи даних, які забезпечують перевірку типів під час компіляції, та високоуровневий синтаксис, який дозволяє швидко створювати потужні веб-додатки. (FastAPI, n.d.)

Одним з основних переваг FastAPI є швидкість та продуктивність, яка досягається завдяки асинхронності та використанню високопродуктивних бібліотек, таких як Pydantic та uvicorn. Це робить FastAPI ідеальним вибором для великих проектів, які мають високі вимоги до продуктивності та масштабованості.

FastAPI дозволяє швидко створювати програмний інтерфейс для системи тематичного моделювання тексту завдяки своїй високій швидкості та простоті використання. Основні переваги FastAPI для написання програмного інтерфейсу для системи тематичного моделювання тексту включають:

Асинхронність: FastAPI підтримує асинхронний код, що дозволяє програмному інтерфейсу обробляти більше запитів на одному потоці.

Валідація даних: FastAPI використовує бібліотеку Pydantic для перевірки та валідації даних, що передаються через API.

Swagger UI: FastAPI генерує автоматичну документацію API з Swagger UI, що дозволяє легко перевіряти та тестувати програмний інтерфейс.

Для початку, розглянемо основні переваги використання FastAPI для написання програмного інтерфейсу для системи тематичного моделювання тексту:

Швидкість та продуктивність: FastAPI побудований на основі Starlette та Pydantic, що дозволяє досягти дуже високої швидкості відповіді на запити. Використання асинхронного підходу дозволяє одночасно обслуговувати багато запитів із невеликими затримками.

Мінімальний код: FastAPI дозволяє знизити кількість необхідного коду, завдяки вбудованому автоматичному документуванню API, розширеним схемам Pydantic та генеруванню OpenAPI та Swagger-документації.

Простота використання: FastAPI використовує декларативний підхід, який дозволяє описувати функції-обробники запитів, використовуючи анотації типів даних та параметрів. Це дозволяє знизити кількість необхідного коду, збільшити його читабельність та зробити його більш ефективним.

Інтеграція з іншими інструментами: FastAPI інтегрується з багатьма іншими інструментами, такими як SQLalchemy для роботи з базами даних, Celery для роботи з асинхронними задачами, Pytest для тестування та багатьма іншими.

Щодо використання FastAPI для написання програмного інтерфейсу для системи тематичного моделювання тексту, можна створити набір ендпоінтів, що дозволять взаємодіяти зі системою та отримувати результати аналізу текстів. [7]

Pydantic

Pydantic - це бібліотека Python, яка дозволяє описувати структури даних (схеми) та перевіряти їх на валідність. Це робить процес обробки даних більш простим та ефективним, особливо в випадках, коли вхідні дані можуть бути неточні або неочікувані.

У системі тематичного моделювання тексту Pydantic може бути корисним на кількох рівнях:

Валідація вхідних даних: При розробці програмного інтерфейсу з використанням FastAPI, Pydantic може бути використаний для валідації вхідних даних, таких як розмір датасету, конфігурація алгоритму тематичного моделювання, та інших параметрів. Це дозволяє запобігти некоректній обробці даних, а також допомагає відловити помилки на ранніх етапах розробки.

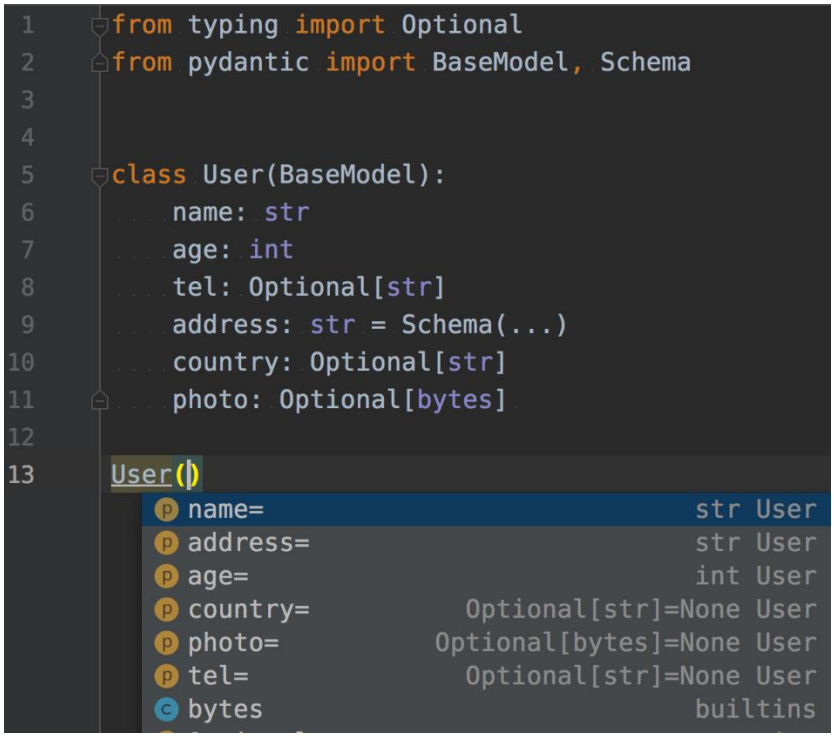


Рисунок 2.1 - Приклад моделі Pydantic

Типізація даних: Pydantic дозволяє створювати схеми даних, що дозволяє описувати типи даних та їх структуру. Це допомагає покращити читабельність

та зрозумілість коду, а також дозволяє зберігати дані в певному форматі, що забезпечує більш високу стійкість до помилок.

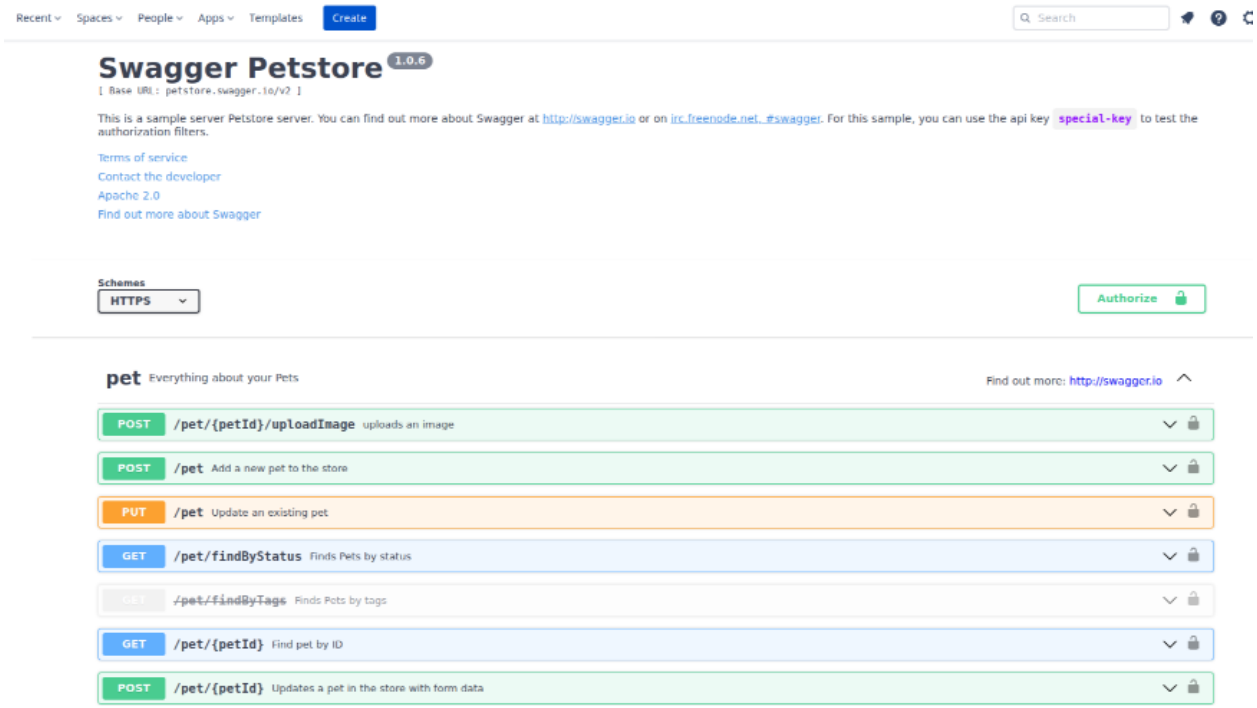


Рисунок 2.2 - Приклад інтерфейсу документації [8]

Документація: Pydantic також має підтримку для генерації автоматичної документації за допомогою стандарту OpenAPI. Це дозволяє автоматично створювати документацію на основі опису вхідних та вихідних даних, що є особливо корисним в великих проектах з багатьма ендпоінтами та різними схемами даних. [8]

Uvicorn

Uvicorn є високопродуктивним ASGI-сервером, який може бути використаний для запуску програмного інтерфейсу, побудованого на FastAPI. Він підтримує протокол HTTP/1.1 та HTTP/2, які є стандартними протоколами для взаємодії між веб-клієнтами та серверами.

Один з основних переваг uvicorn є його висока продуктивність та можливість обробки багатьох запитів одночасно, що дозволяє швидко відповідати на запити користувачів та зменшувати час відповіді програмного

інтерфейсу. Він підтримує асинхронні фреймворки та є оптимізованим для роботи з Python.

У випадку системи тематичного моделювання тексту, uvicorn може бути корисним для запуску програмного інтерфейсу, який надає користувачам можливість виконувати запити на отримання тематичної інформації з текстових даних. Він дозволяє багатокористувацький доступ до системи та може обслуговувати багато запитів одночасно, що є важливим для високонавантажених сервісів.

Крім того, uvicorn має декілька корисних функцій, таких як логування та моніторинг використання ресурсів, які дозволяють зрозуміти, як програмний інтерфейс використовується користувачами та як можна оптимізувати його роботу. Також він має підтримку HTTP/2, що дозволяє прискорити передачу даних між сервером та клієнтом. (Uvicorn , n.d.)

У загальному, uvicorn є потужним сервером, який дозволяє запускати високопродуктивні програмні інтерфейси на основі FastAPI та обслуговувати багато запитів одночасно. [9]

2.2 Вибір технологій NLP

Перш за все варто поетапно розглянути алгоритм за яким відбувається тематичне моделювання тексту.

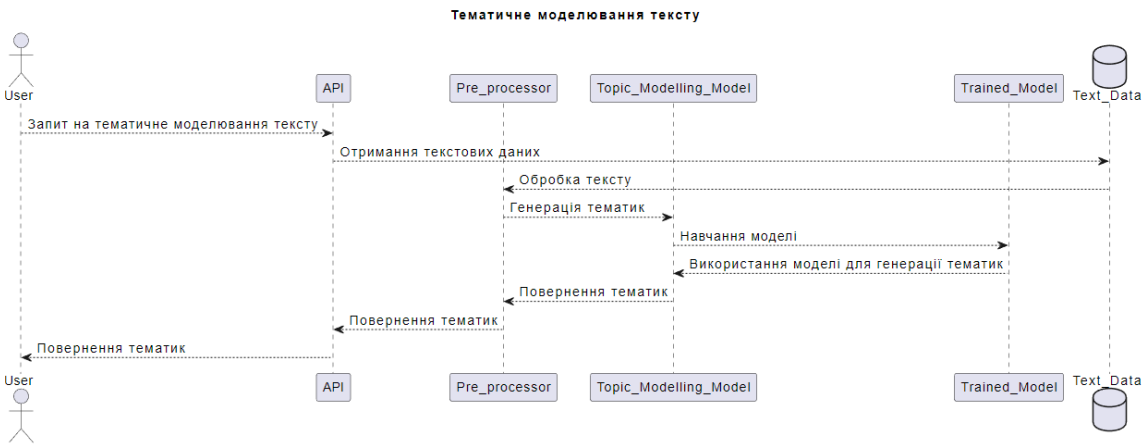


Рисунок 2.3 – алгоритм тематичне моделювання

Тематичне моделювання тексту - це процес аналізу текстових даних з метою визначення тем, які є присутніми в даних. Для побудови тематичних моделей зазвичай використовуються наступні кроки:

1. Отримання даних через програмний інтерфейс: для побудови тематичних моделей потрібні великі обсяги текстових даних, тому першим кроком є отримання даних з різних джерел. Зазвичай це може бути виконано за допомогою програмного інтерфейсу (API). Власне так ми і зробили в цьому проекті.
2. Очистка даних: після отримання даних, їх потрібно очистити від малозначимих частин мови, стоп-слів, а також виконати лематизацію та стемінг для зменшення кількості варіантів написання слів. Це допомагає зменшити розмір словника та поліпшити точність аналізу.
3. Знаходження триграм та біграм: триграми та біграми є комбінаціями слів, які часто зустрічаються разом в тексті. Це може допомогти виявити зв'язки між словами та збільшити точність моделі.
4. Створення словника id2word: для побудови моделі необхідно мати словник, який містить унікальні слова, що зустрічаються в тексті. Словник id2word призначений для використання в бібліотеці gensim для побудови моделі LDA. (gensim, n.d.)
5. Побудова LDA моделі: після попередньої підготовки даних, можна розпочати побудову тематичної моделі. Найпоширенішим методом побудови тематичних моделей є LDA (Latent Dirichlet Allocation), який дозволяє розподілити кожен документ на декілька тем, використовуючи статист
6. Візуалізація результатів: результати тематичного моделювання можна візуалізувати за допомогою різноманітних графіків та діаграм, що дозволяє зрозуміти, які теми домінують у тексті, як вони пов'язані між

собою та які слова є ключовими для кожної теми. Для візуалізації результатів можна використовувати бібліотеки, такі як pyLDAvis.

Для розробки системи тематичного моделювання тексту можна використовувати різноманітні технології та бібліотеки з області обробки природньої мови (Natural Language Processing, NLP).

Однією з основних бібліотек NLP є NLTK (Natural Language Toolkit). Вона містить різноманітні інструменти для обробки тексту, такі як стемінг, лематизація, токенізація та інші. NLTK також містить набір корпусів тексту, які можна використовувати для тренування моделей NLP.

Ще однією популярною бібліотекою для NLP є SpaCy. (spacy, n.d.) Вона надає інструменти для розпізнавання сутностей, токенізації, лематизації та інші. SpaCy також має вбудовані моделі для розпізнавання частин мови та залежностей між словами.

Для векторизації тексту можна використовувати бібліотеку Gensim. Вона містить інструменти для побудови тематичних моделей на основі моделі багатовимірного розподілу (Latent Dirichlet Allocation, LDA). Gensim також містить інструменти для побудови векторних представлень тексту за допомогою алгоритму Doc2Vec.

Також можна використовувати бібліотеку Scikit-learn для векторизації тексту та класифікації текстів за темами. Scikit-learn містить інструменти для побудови моделей машинного навчання, таких як класифікатори на основі навчання з учителем (наприклад, Random Forest) або без учителя (наприклад, кластеризація).

2.3 Алгоритм LDA

Latent Dirichlet Allocation (LDA) - це генеративна модель тематичного моделювання тексту, яка використовується для виявлення тем у текстових

документах. Алгоритм LDA був запропонований Девідом Блеєм та його колегами в 2003 році. [10]

Модель LDA базується на припущенні, що кожен документ можна розглядати як комбінацію декількох тем, а кожна тема може мати різні ймовірності появи кількох слів. У своїй основі LDA є багаторівнева ймовірнісна модель, яка пояснює, як документи генеруються з певних тем.

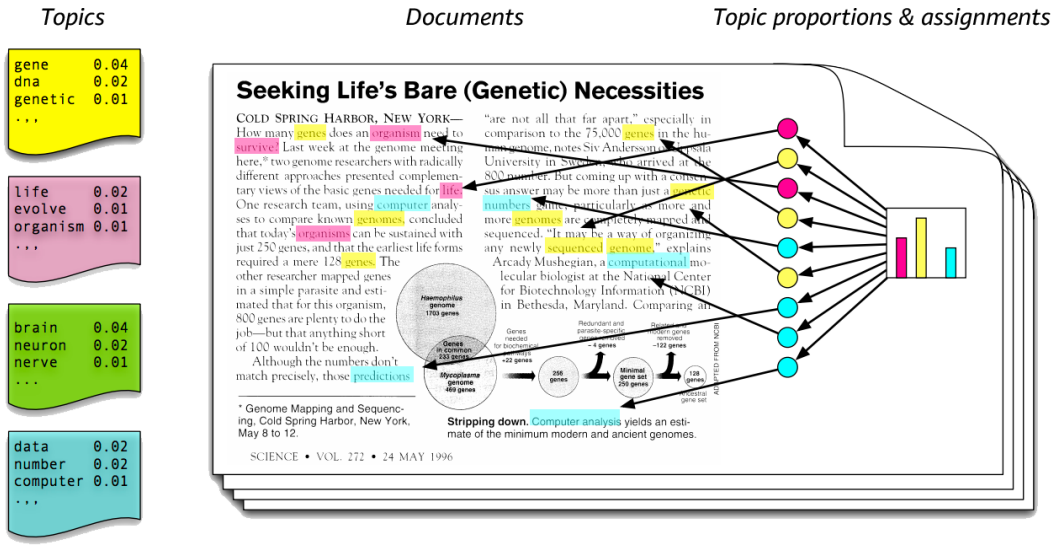


Рисунок 2.3 - Ілюстрація розкидання ключових слів на теми [11]

Алгоритм LDA складається з наступних етапів:

- Вибір кількості тем для моделювання.
- Запис документів у векторний формат, де кожне слово у документі представлене як індекс словника.
- Вибір початкових значень для матриць тем та слів.
- Ітераційний процес, який оновлює матриці тем та слів на кожній ітерації, з метою зменшення функції витрат, яка визначає наскільки добре модель підходить до даних.
- Отримання остаточних матриць тем та слів, які використовуються для отримання тематичного профілю кожного документу.

На кожній ітерації алгоритм LDA оновлює дві матриці: матрицю тем та матрицю слів. Матриця тем містить тематичний профіль кожного документу, а

матриця слів містить ймовірності появи кожного слова в кожній темі. Значення цих матриць залежать одне від одного: якщо тема містить багато певних слів, то ймовірність, що документ буде містити цю тему, збільшується.

Рисунок 2.4 - Матриці тем та слів (термінів) [12]

Оцінювання якості тематичної моделі в LDA зазвичай виконується за допомогою метрик, таких як кошторисний коефіцієнт зв'язності (coherence score) та кошторисний коефіцієнт перплексії (perplexity score).

Кошторисний коефіцієнт зв'язності використовується для оцінювання того, наскільки добре терміни в темах зв'язані між собою. Для цього він обчислюється як сума логарифмів ймовірностей спільно зустрічання двох слів в текстах, що належать до однієї теми. Зазвичай, цей показник є числом від 0 до 1, де значення близько до 1 означає більшу зв'язність між термінами теми.

Кошторисний коефіцієнт перплексії використовується для оцінювання того, наскільки добре модель передбачає наступне слово в тексті. Для цього він обчислюється як експонента середнього логарифму правдоподібності (логарифмічного відношення ентропії) тексту, що вважається нормалізованим за довжиною тексту. Значення цього показника зазвичай є більше за 1, де менше значення означає кращу передбачувану здатність моделі.

Хоча LDA є одним з найпопулярніших алгоритмів для тематичного моделювання тексту, він має певні обмеження [13]. Наприклад, він не дозволяє враховувати залежності між документами, які включаються до моделі. Також, він не може уникнути проблеми перекладу, коли слова мають різні значення в залежності від контексту. Хоча подібне можна уникнути якщо правильно підійти до етапу препроцесінгу тексту.

Одним з найбільших обмежень LDA є необхідність вручну задавати кількість тем. Якщо це число занадто мале, то LDA може не виявити всі теми в тексті. Якщо число тем занадто велике, то може виникнути проблема перенавчання і теми можуть перекриватися, що зменшує якість результатів.

Ще одним недоліком є те, що LDA не враховує семантичну залежність між словами в тексті, тому два слова, які семантично близькі, можуть потрапити в різні теми, що зменшує точність результуючих тематичних моделей. [14]

Також, LDA має певні обмеження щодо типу текстів, які може обробляти. Наприклад, якщо текст містить дуже багато незнайомих слів або неправильні форми слів, то результати тематичного моделювання можуть бути неточними.

Отже, при застосуванні LDA до текстових даних варто враховувати його обмеження та недоліки. Важливо визначити оптимальну кількість тем для конкретного датасету, ретельно очистити дані перед застосуванням алгоритму та враховувати можливість залежності між словами в тексті.

Проте, не зважаючи на ці обмеження, LDA є одним з найкращих алгоритмів тематичного моделювання тексту і продовжує застосовуватися в багатьох дослідженнях та комерційних проектах.

Програмна реалізація алгоритму LDA існує у вигляді відповідного модулю бібліотеки gensim. У бібліотеці gensim реалізовано LDA алгоритм, який використовує EM-алгоритм для вирішення задачі тематичного моделювання.

Першим кроком у використанні LDA в gensim є побудова словника. Для цього необхідно створити список унікальних слів, що зустрічаються в

документах, і присвоїти кожному слову унікальний ідентифікатор. Використовуючи цей словник, gensim може перетворити кожен документ у вектор, де кожен елемент відповідає кількості входжень слова зі словника у даний документ.

Для побудови моделі LDA в gensim необхідно визначити кількість тем, які ви хочете витягнути з даних. Можна використовувати евристики для підбору оптимального значення, наприклад, такі як перевірка перплексії на тестових даних або методи зведення моделі до двох тем і порівняння результатів.

Після визначення кількості тем, можна створити модель LDA. Gensim підтримує різні параметри моделі, такі як кількість ітерацій, альфа та бета параметри, що впливають на процес навчання моделі. Після навчання моделі можна отримати теми та їхній склад за допомогою методу get_topic_terms().

Gensim також підтримує можливість підбору оптимальних параметрів моделі LDA за допомогою пакету gensim.models.ldamodel.CoherenceModel. Цей пакет дозволяє вибрати оптимальну кількість тем та оптимальні значення параметрів моделі LDA за допомогою метрик якості. [15]

2.4 Остаточний вибір технології для проекту

Ми проаналізували різні технології та бібліотеки NLP, а також розглянули можливі варіанти архітектури та програмних засобів для реалізації нашої системи.

Перш за все варто зазначити що для розробки системи ми обрали мову програмування Python.

Python є однією з найбільш популярних мов програмування в галузі науки про дані та машинного навчання. Це пов'язано з тим, що Python має багато бібліотек та інструментів, що дозволяють зручно та ефективно працювати з даними, включаючи NLP-задачі. Наприклад, в Python є такі популярні

бібліотеки, як Gensim, NLTK, spaCy, TextBlob, які дозволяють здійснювати різноманітні операції з текстом, включаючи тематичне моделювання.

Крім того, Python має простий та зрозумілий синтаксис, що дозволяє швидко та зручно писати код. Він також має велику спільноту користувачів, яка активно розвиває нові бібліотеки та інструменти для різних галузей науки та технологій.

За допомогою бібліотек NLTK, Gensim та SpaCy ми зможемо здійснити препроцесінг тексту, а саме: очистку даних, видалення стоп-слів, лематизацію та стемінг, що допоможе покращити якість аналізу тексту. Крім того, для побудови тематичної моделі, ми будемо використовувати бібліотеку Gensim, яка має реалізацію алгоритмів LSA, LDA та інших моделей. [16]

У виборі технологій для розробки серверної частини ми обрали FastAPI та Uvicorn. FastAPI є потужним та швидким фреймворком, який дозволяє створювати високопродуктивні REST API, а Uvicorn - це сервер, який використовує високопродуктивний ASGI-протокол, що дозволяє швидко обробляти багато запитів.

Отже, наша система тематичного моделювання тексту буде побудована на базі сучасних технологій та бібліотек NLP, що дозволить досягти високої точності та ефективності аналізу тексту. Крім того, використання FastAPI та Uvicorn дозволить реалізувати швидкий та масштабований сервер для обробки запитів.

ВИСНОВОК ДО 2 РОЗДІЛУ

Дослідження різних архітектур та програмних засобів для реалізації системи тематичного моделювання тексту є важливим етапом при розробці такої системи. Перевірка ефективності різних підходів дозволяє вибрати найбільш оптимальний варіант для конкретної задачі.

Кожна задача має свої особливості і може потребувати різних підходів до її вирішення. Дослідження дозволяє визначити, які засоби і архітектури найкраще підходять для реалізації конкретної задачі з тематичного моделювання тексту.

Різні архітектури мають свої переваги та недоліки. Дослідження дозволяє порівняти ефективність різних архітектур і вибрати найкращий варіант для конкретної задачі.

Кожен програмний засіб має свої обмеження та може бути підходящим для вирішення певної задачі. Дослідження дозволяє порівняти ефективність різних програмних засобів та вибрати найоптимальніший варіант для реалізації конкретної системи тематичного моделювання тексту.

Дослідження дозволяє виявити проблемні місця та можливості оптимізації процесів роботи системи тематичного моделювання тексту.

Як результат ми обрали такий стек технологій:

- Мова програмування Python
- За допомогою бібліотек NLTK, Gensim та SpaCy ми зможемо здійснити препроцесінг тексту
- У виборі технологій для розробки серверної частини ми обрали FastAPI та Uvicorn.
- Для побудови тематичної моделі, ми будемо використовувати бібліотеку Gensim
- В якості основного алгоритму беремо LDA

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ПРОГРАМНОГО ІНТЕРФЕЙСУ

Після того як було визначено функціонал програми, який має дозволяти отримувати текстові дані через програмний інтерфейс, проводити очистку даних, знаходити біграми та триграми, створювати словник id2word та будувати модель LDA можемо перейти власне розробки самої програми. Для реалізації програми використовувалися мова програмування Python, фреймворк FastAPI для створення серверної частини, бібліотеки Pydantic та uvicorn для підтримки інтерфейсу API, а також бібліотека Gensim для реалізації моделювання тематик. Далі буде описана структура проекту та описаний код програми з коментарями.

3.1 Опис структури проекту та призначення основних складових

По опису проекту та визначеному функціоналу який ми розглянули раніше стає зрозуміло що система має в собі дві складові – API та певний класифікатор (кластеризатор) що відповідає за власне обробку тексту.

Як результат маємо пакети /api та /classifier в директорії з сирцевим кодом (див. рис. 3.1). Пакет /api в свою чергу складається з маршрутів /routers та додактових модулей в /utils. Модуль DataBodies.py відповідає за визначення моделей запитів що може приймати програмний інтерфейс. Модуль Exceptions.py описає можливі помилки що можуть виниктути під час роботи системи. TextPreProcessor.py відповідає за попередню обробку тексту.

В директорії /routers знаходяться модулі що відповідають за маршутизацію запитів. Є два окремих роутера під аналіз тексту та під створення нової моделі.

В модулі Dependency додатково знаходиться реалізація авторизації та аутентифікації користувачів які потім як залежність підв’язується до кожного rout. Модуль main.py є основним модулем програмного інтерфейсу з якого і

відбувається підняття серверу, підключення маршутирізації, підключення middleware.

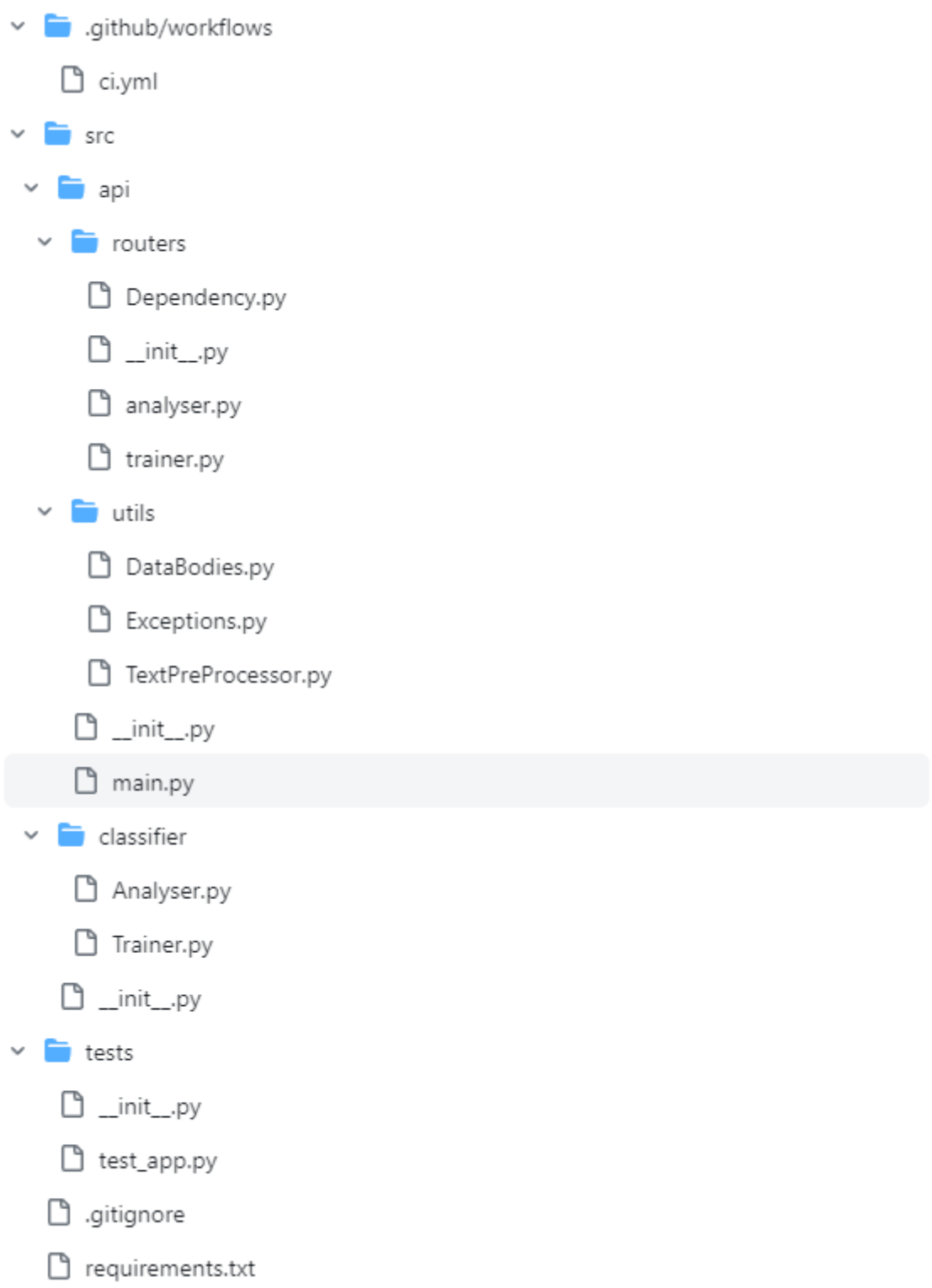


Рисунок 3.1 – Структура проекту

Middleware - це програмний компонент, що забезпечує обробку запитів перед тим, як вони досягнуть основного додатку. Його можна використовувати для різних цілей, наприклад:

- обробки запитів та відповідей у стандартному форматі
- логування подій
- аутентифікації та авторизації користувачів
- обробки помилок та винятків

Використання middleware дозволяє розділити логіку додатку на окремі компоненти та забезпечити більш гнучку та масштабовану архітектуру. У багатьох веб-фреймворках, в тому числі у FastAPI, підтримується використання middleware.

В пакеті класифікатор є Analyser та Trainer яку відповідають за або використання старої моделі, або створення нової моделі.

3.2 Опис основних складових проекту

routers.analyser.py

Спершу розглянемо основні ендпоінти нашого програмного інтерфейсу. Цей код (див. рис. 3.2) створює маршрути для API, які взаємодіють з модулем Analyser. Аналізатор відповідає за кластеризацію текстових даних.

Код починається з імпортування потрібних модулів. Основним модулем є fastapi, який використовується для створення маршрутів. Далі імпортується функція get_current_active_user з модуля Dependency, яка відповідає за авторизацію користувачів. Також імпортується клас Analyser з модуля Analyser.

Далі створюється маршрутизатор з префіксом /analyser, який буде відповідати за обробку запитів, пов'язаних з аналізатором. У залежності вказується функція get_current_active_user, яка використовується для авторизації користувачів зі значенням scopes=["User"].

Наступні дві функції обробляють запити, які надійшли на маршрутизатор. Перша функція get_topic_for_text приймає тіло запиту від клієнта, яке містить модель та текст для аналізу. Ця функція створює об'єкт Analyser з переданою моделлю, викликає метод define_text для кластеризації тексту та повертає прогнозований кластер у вигляді рядка.

```python
# -*- coding: utf-8 -*-

from fastapi import APIRouter, Security

from src.api.routers.Dependency import get_current_active_user
from src.api.utils.DataBodies import AnalyserBody
from src.classifier.Analyser import Analyser

router = APIRouter(
    prefix="/analyser",
    responses={404: {"description": "Not found"}},
    dependencies=[Security(get_current_active_user, scopes=["User"])]
)


@router.post('/topic')
def get_topic_for_text(body: AnalyserBody):
    analyser = Analyser(body.model_id)
    prediction = analyser.define_text(body.text)
    return str(prediction)


@router.post('/lda_visualization')
def get_visualization(body: AnalyserBody):
    analyser = Analyser(body.model_id)
    model = analyser.get_model()
    return model
```

Рисунок 3.2 – Модуль routers.analyser.py

Друга функція get_visualization приймає тіло запиту від клієнта, яке містить модель для відображення візуалізації. Ця функція створює об'єкт Analyser з переданою моделлю, викликає метод get_model, який повертає модель для візуалізації. Далі ця модель повертається у відповідь на запит.

router.trainer.py

```python
# -*- coding: utf-8 -*-

import uuid

import pandas as pd
from fastapi import APIRouter, BackgroundTasks, UploadFile, File, Depends, Security

from src.api.routers.Dependency import get_current_active_user
from src.api.utils.DataBodies import Meta
from src.classifier.Trainer import Trainer

router = APIRouter(
    prefix="/trainer",
    responses={404: {"description": "Not found"}},
    dependencies=[Security(get_current_active_user, scopes=["User"])]
)


@router.post('/train_model/')
async def train_model(background_tasks: BackgroundTasks,
                      file: UploadFile = File(...),
                      meta: Meta = Depends()):
    # todo chose column
    df = pd.read_csv(file.file, delimiter=meta.delimiter)
    data = df[meta.column].tolist()
    random_id = uuid.uuid4()
    trainer = Trainer(num_topics=meta.topics_num, stop_words=meta.stop_words)
    background_tasks.add_task(trainer.process, data, random_id)
    # return id on which we can call for a model
    return random_id
```

Рисунок 3.3 – Модуль router.trainer.py

Код trainer.py містить API роутер для навчання моделі. Роутер має префікс /trainer та є захищеним, тому його можуть використовувати лише користувачі з правами "User".

У роутері є один ендпойнт /train_model/, який приймає файли з даними та параметри метаданих, таких як delimiter, column, topics_num та stop_words.

Після отримання файлу, він зчитується у форматі CSV та зберігається як pandas.DataFrame. З DataFrame витягується колонка, що вказана у параметрах метаданих, та перетворюється у список.

Далі генерується випадковий ідентифікатор та створюється об'єкт Trainer з параметрами num_topics та stop_words. Задача Trainer оброблюється як фонова задача, і її виконання додається до background_tasks. Не очікуючи обробки задачі користувачеві повертається ідентифікатор моделі щоб в подальшому він міг її отримати.

BackgroundTasks в FastAPI - це механізм, який дозволяє виконувати довгі процеси асинхронно, без блокування основного потоку виконання.

Іноді API-запити потребують виконання довгих процесів, наприклад, обробка великої кількості даних, відправка повідомлень на email або маніпуляції з файлами, що можуть зайняти багато часу. Якщо ці процеси виконувати в основному потоці виконання, це може призвести до блокування потоку і зниження продуктивності всього додатку.

BackgroundTasks дозволяє відокремити довгі процеси від основного потоку виконання, тобто відправити їх в окремий потік або робочий процес, щоб не блокувати основний потік.

Фактично, коли запит приходить на сервер, асинхронна функція додає задачу в чергу фонових процесів, і відповідь на запит повертається відразу ж, не чекаючи завершення довгого процесу. Це дозволяє більш швидко обслуговувати запити та збільшити продуктивність додатку.

У FastAPI BackgroundTasks забезпечуються об'єктом BackgroundTasks, який можна ін'єктувати в функцію-обробник запиту за допомогою ключового слова background_tasks. Далі, функція-обробник може додати процес в чергу фонових задач, використовуючи метод add_task().

BackgroundTasks також можуть використовуватись в сполученні з бібліотеками, які підтримують асинхронність, такими як asyncio, Celery або RQ.

data bodies

```python
# -*- coding: utf-8 -*-
from typing import Optional, List

from pydantic import BaseModel, conint

PATTERN = r'\p{IsCyrillic}'

DATE_FORMAT = "%Y-%m-%d"
DATE_FORMAT_TIME = "%Y-%m-%d %H:%M:%S"


class Subscribe(BaseModel):
    user_id: int


class Meta(BaseModel):
    header: bool = True
    delimiter: str = ","
    column: Optional[str] = 'text'
    topics_num: conint(ge=5, le=20) = 10
    stop_words: Optional[List[str]]


class AnalyserBody(BaseModel):
    model_id: str
    text: Optional[str]


class Token(BaseModel):
    access_token: str
    token_type: str


class TokenData(BaseModel):
    username: Optional[str] = None


class User(BaseModel):
    username: str
    email: Optional[str] = None
    full_name: Optional[str] = None
    disabled: Optional[bool] = None


class UserInDB(User):
    hashed_password: str
```

Рисунок 3.4 – Моделі запитів та елементів системи

Для всіх запитів до програмного інтерфейсу були визначені основні моделі формату вхідних даних (див. рис. 3.4). Цей код містить описи декількох моделей Pydantic, які використовуються у фреймворку FastAPI.

BaseModel є базовим класом, який використовується для визначення моделей, що передаються між сервером та клієнтом. Кожна модель містить властивості, які можуть бути обов'язковими або необов'язковими.

Subscribe - модель, яка містить обов'язкову властивість user_id з цілим числом, яка вказує на ID користувача, який хоче підписатися на щось.

Meta - модель, яка містить властивості для настройки параметрів аналізу даних. Наприклад, delimiter - роздільник у файлі, column - колонка, на якій має проводитися аналіз, topics_num - кількість тем, які необхідно визначити під час аналізу, та stop_words - список слів, які слід ігнорувати під час аналізу.

AnalyserBody - модель, яка містить обов'язкову властивість model_id з рядком та необов'язкову властивість text з рядком, яка використовується для визначення теми тексту.

Token - модель, яка містить обов'язкову властивість access_token з рядком та необов'язкову властивість token_type з рядком, яка використовується для авторизації користувача.

TokenData - модель, яка містить необов'язкову властивість username з рядком, яка використовується для авторизації користувача.

User - модель, яка містить обов'язкову властивість username з рядком та необов'язкові властивості email, full_name та disabled, які використовуються для визначення користувача.

UserInDB - модель, яка містить властивості користувача та хешований пароль. Вона використовується для зберігання даних користувача.

Pydantic моделі використовуються в FastAPI для валідації запитів, що приходять на сервер. Вони дозволяють задати структуру запиту, тобто вказати, які поля мають бути в запиті та якого вони типу.

Pydantic моделі дозволяють задати такі параметри для кожного поля запиту:

- тип даних
- опціональність поля (чи є воно обов'язковим до заповнення)
- значення за замовчуванням
- обмеження (наприклад, максимальне значення числового поля)
- опис поля

Також, Pydantic автоматично генерує документацію для кожного запиту на основі моделей, що спрощує розробку та підтримку API. Використання Pydantic моделей для запитів дозволяє писати більш безпечний та стабільний код, оскільки вони допомагають забезпечити коректність даних, що передаються на сервер.

TextPreProcessor

Даний код (див. рис. 3.5) містить клас TextPreProcessor, який відповідає за попередню обробку тексту. Об'єкт цього класу отримує список текстів у вигляді стрічок та повертає список списків оброблених токенів (слів), які можуть використовуватися для подальшої обробки тексту, наприклад, для побудови тематичної моделі.

Клас містить наступні методи:

- __init__(self, stop_words=None, tags=None): конструктор, де stop_words - список стоп-слів, які необхідно виключити з тексту, та tags - список тегів частин мови, які використовуються для відбору слів у тексті.
- process(self, texts: Union[List[str], str]) -> list[list]: метод, який приймає список текстів у вигляді стрічок або один текст, виконує попередню обробку тексту (токенізацію, відбір слів за тегами

частин мови, відкидання стоп-слів, лематизацію та стемінг) та повертає список списків токенів.

Клас використовує пакет nltk для токенізації та позначення частин мови, а також для видалення стоп-слів та застосування стеммера. У методі process використовуються наступні кроки обробки тексту:

1. Токенизація тексту за допомогою word_tokenize з пакету nltk.
2. Визначення частин мови за допомогою pos_tag з пакету nltk.
3. Відбір слів за тегами частин мови, які визначені при створенні об'єкта класу.
4. Лематизація слів за допомогою WordNetLemmatizer з пакету nltk.
5. Стемінг слів за допомогою SnowballStemmer з пакету nltk.
6. Видалення стоп-слів з тексту та залишення тільки слів довжиною більше 3 символів.
7. Повернення списку списків оброблених токенів.

```python
from typing import List, Union

import nltk
from nltk import word_tokenize, SnowballStemmer, pos_tag
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords

nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')


class TextPreProcessor:
    stop_words = stopwords.words('english')
    tags = []

    def __init__(self, stop_words=None, tags=None):
        self.lemmatizer = WordNetLemmatizer()
        if tags is None:
            self.tags = ['NN', 'JJ']
        else:
            self.tags = tags

        if stop_words is not None:
            self.stop_words.extend(stop_words)

    def process(self, texts: Union[List[str], str]) -> list[list]:
        if type(texts) is str:
            texts = [texts]

        stemmer = SnowballStemmer('english')
        result = []
        for doc in texts:
            # Токенизация
            try:
                words = word_tokenize(doc.lower())
            except AttributeError:
                continue
            # Определение частей речи
            tagged_words = pos_tag(words)
            # Оставляем только существительные и прилагательные
            filtered_words = [word for word, tag in tagged_words if tag in self.tags]
            # Удаление стоп-слов
            lemmatized_tokens = [self.lemmatizer.lemmatize(token) for token in filtered_words]
            # Применение стеммера
            stemmed_words = [stemmer.stem(word) for word in lemmatized_tokens]
            filtered_sw_words = [word for word in stemmed_words if
                                 word not in self.stop_words and word.isalpha and len(word) > 3]

            result.append(filtered_sw_words)
        return result
```

Рисунок 3.5 – Модуль який відповідає за попередню обробку тексту

Trainer.py

```python
class Trainer:
    """
    Gives functionality to train new LDA models
    """
    corpus = None
    id2word = None
    num_topics = None
    lda_model = None

    def __init__(self, stop_words=None, tags=None, num_topics=10):
        self.pre_processor = TextPreProcessor(stop_words, tags)
        self.num_topics = num_topics

    @staticmethod
    def make_trigrams(data_words):
        # Build the bigram and trigram models
        bigram = gensim.models.Phrases(data_words, min_count=5, threshold=100)  # higher threshold fewer phrases.
        trigram = gensim.models.Phrases(bigram[data_words], threshold=100)
        # Faster way to get a sentence clubbed as a trigram/bigram
        bigram_mod = gensim.models.phrases.Phraser(bigram)
        trigram_mod = gensim.models.phrases.Phraser(trigram)

        def trigrams(texts):
            return [trigram_mod[bigram_mod[doc]] for doc in texts]

        # Form Bigrams
        data_words_trigrams = trigrams(data_words)
        return data_words_trigrams

    def save_model(self, model_id):
        ld_avis_data_filepath = os.path.join('warehouse/ldavis_prepared_' + str(model_id))
        # save LdaModel
        self.lda_model.save(ld_avis_data_filepath+'_model')
        # save Dictionary
        self.id2word.save(ld_avis_data_filepath+'_dictionary')

        lda_prepared = gensimvis.prepare(self.lda_model, self.corpus, self.id2word)
        pyLDAvis.save_html(lda_prepared, ld_avis_data_filepath+'.html')

    def build_model(self, model_id: str):
        # Build LDA model
        self.lda_model = gensim.models.LdaModel(corpus=self.corpus,
                                                id2word=self.id2word,
                                                num_topics=self.num_topics,
                                                random_state=100,
                                                update_every=1,
                                                chunksize=100,
                                                passes=10,
                                                alpha='auto',
                                                per_word_topics=True
                                                )

        self.save_model(model_id)

    def process(self, data, model_id):
        data_words = self.pre_processor.process(data)
        data_words_trigrams = self.make_trigrams(data_words)
        self.id2word = corpora.Dictionary(data_words_trigrams)
        # Term Document Frequency
        self.corpus = [self.id2word.doc2bow(text) for text in data_words_trigrams]

        self.build_model(model_id)
```

Рисунок 3.6 - Модуль навчання моделей

Модуль Trainer (див. рис. 3.7) відповідає за створення моделей LDA. Клас Trainer містить наступні методи:

- make_trigrams(data_words) - Цей метод створює біграми та триграми зі словесних даних. Він використовує gensim.models.Phrases для створення біграм та триграм, а потім gensim.models.phrases.Phraser для їх перетворення в список.
- save_model(self, model_id) - Цей метод зберігає навчену модель LDA разом зі словником та візуалізацією, які можна відобразити за допомогою PyLDAvis. Модель та словник зберігаються у вигляді файлів з іменами, що містять ідентифікатор моделі.
- build_model(self, model_id: str) - Цей метод створює та навчає модель LDA на основі корпусу та словника, що були побудовані раніше. Модель зберігається за допомогою методу save_model.
- process(self, data, model_id) - Цей метод обробляє нові дані за допомогою TextPreProcessor та будує на них нову модель LDA з використанням build_model. Кожен раз, коли викликається цей метод, старі дані зберігаються.

```
def build_model(self, model_id: str):
    # Build LDA model
    self.lda_model = gensim.models.LdaModel(corpus=self.corpus,
                                            id2word=self.id2word,
                                            num_topics=self.num_topics,
                                            random_state=100,
                                            update_every=1,
                                            chunksize=100,
                                            passes=10,
                                            alpha='auto',
                                            per_word_topics=True
                                            )

    self.save_model(model_id)
```

Рисунок 3.7 – Метод побудови моделей

В методі build_model (див. рис. 3.7) ми використовуємо декілька параметрів для навчання моделі:

- **corpus**: це список списків, який містить Bag-of-Words представлення документів. Кожен документ представлений як список кортежів (term ID, term frequency).
- **id2word**: це словник, який містить відображення ідентифікаторів слів в їхні текстові представлення.
- **num_topics**: це число тем, які необхідно знайти в текстових документах.
- **random_state**: це параметр випадкового генератора, який використовується в LDA моделі.
- **update_every**: це кількість документів, що обробляються, перш ніж здійснюється оновлення моделі.
- **chunksize**: це кількість документів, яка обробляється в одному пакеті (chunk), який використовується для оновлення моделі.
- **passes**: це кількість ітерацій алгоритму LDA, які необхідно виконати для знаходження оптимальних параметрів моделі.
- **alpha**: це параметр, який відповідає за розподіл тем в документах. Якщо встановлено значення "auto", то alpha визначається автоматично.
- **per_word_topics**: це параметр, який вказує, чи потрібно повернути список тем та їхні ваги для кожного слова в документі. Якщо значення True, то метод **lda_model.get_document_topics** буде повертати список тем та їх ваг для кожного слова в документі, які використовуються для обчислення тематичного профілю документа. Якщо False, то метод буде повертати лише загальну вагу теми в документі.

Term ID (ідентифікатор терміна) - це унікальний числовий ідентифікатор, який призначається кожному терміну (слову) в словнику. Ідентифікатори термінів використовуються для створення Bag of Words моделі, яка використовується для аналізу тексту. У Bag of Words моделі, кожен документ

відображається у векторному представленні, де кожна позиція вектора відповідає ідентифікатору терміна, а значення в позиції відображає кількість разів, коли цей термін з'являється у документі.

Term frequency (частота термінів) - це кількість разів, коли термін (слово) з'являється в документі. Наприклад, якщо слово "кіт" з'являється 3 рази в документі, то його term frequency дорівнює 3. Term frequency може використовуватись для створення Bag of Words моделі, яка відображає документ у векторному представленні, де значення в позиції відображає term frequency терміна. [17]

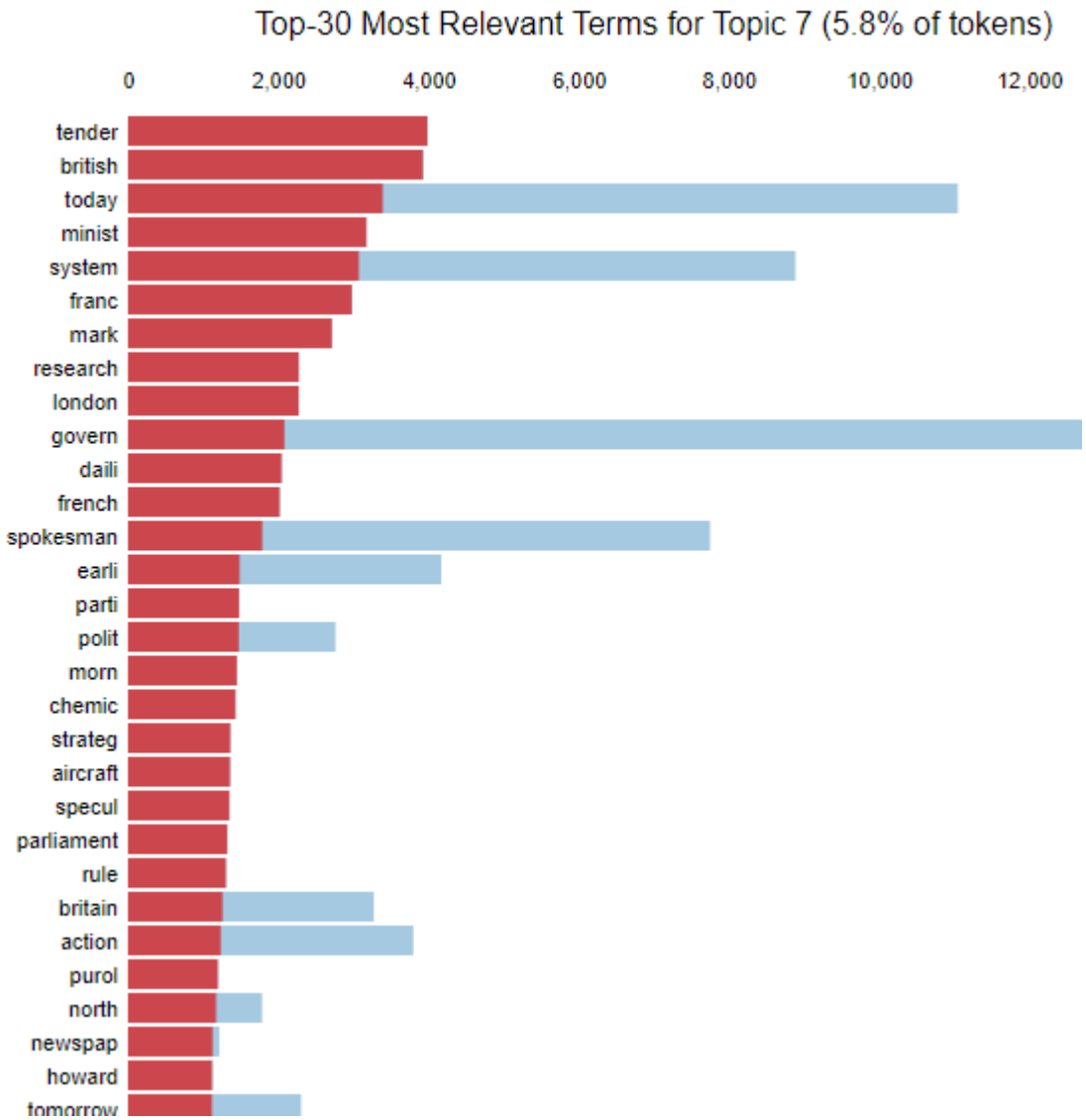


Рисунок 3.8 – приклад Term frequency [18]

3.3 Опис додаткових складових проекту

Крім основних складових проекту що відповідають за основний функціонал системи тематичного моделювання тексту також існує декілька додаткових складових – тести, налаштування CI/CD, список бібліотек.

ci.yml

```yaml
on:
  push:
jobs:
  build:
    runs-on: ubuntu-latest
    if: github.ref_type == 'branch'
    steps:
    - name: Checkout code
      uses: actions/checkout@v2
    - name: Setup Python
      uses: actions/setup-python@v2
      with:
        python-version: '3.x'
    - name: Install dependencies
      run: pip install -r requirements.txt
    - name: Run tests
      run: pytest
    - name: Notify on failure
      if: failure()
      run: echo "Tests failed."
```

Рисунок 3.9 – Конфігурація автоматичних тестів

Цей код визначає конфігурацію дій для виконання автоматичного тестування під час здійснення push-операції в гілці. Конфігурація включає наступні кроки:

1. Checkout code: Завантажує код з відповідної гілки.
2. Setup Python: Налаштовує середовище Python версії 3.

3. Install dependencies: Встановлює залежності, які описані у файлі "requirements.txt" відповідно до налаштувань віртуального середовища Python.
4. Run tests: Запускає автоматичні тести за допомогою бібліотеки pytest.
5. Notify on failure: Якщо тести не пройдені успішно, виводиться повідомлення "Tests failed."

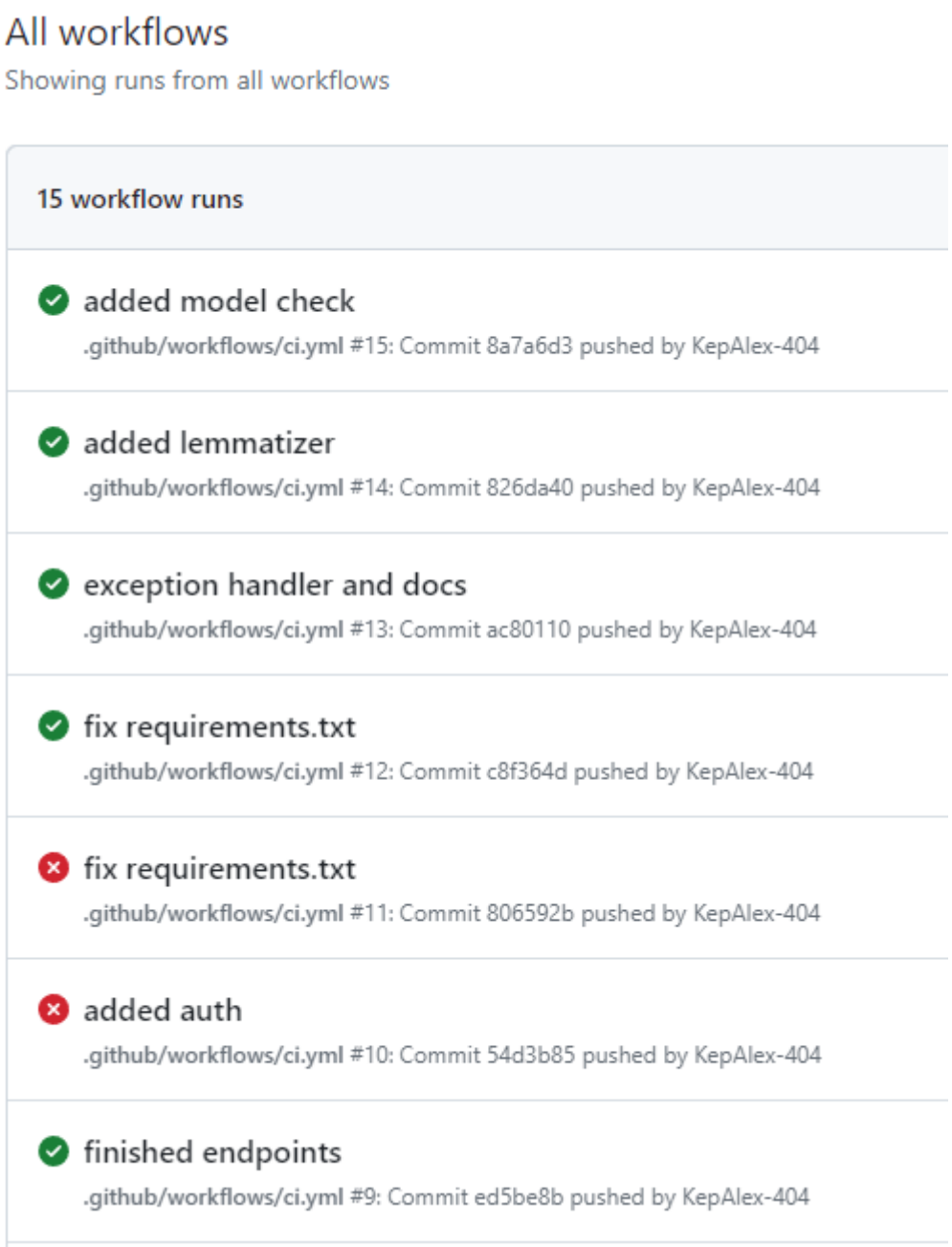


Рисунок 3.10 – Вигляд пайплайнів з тестами на гітхабі

Test_app.py

Для автоматичного тестування системи під час розробки та деплою були розроблені автотести (див. рис. 3.11). Перші три рядки викликають імпорт необхідних бібліотек, тобто TestClient для тестування додатку, завантаження додатку з головного файлу та Dependency відповідно.

У наступному рядку створюється функція override_dependency, яка замінює залежність користувача на заздалегідь заданий об'єкт користувача, для забезпечення стабільності тестів.

Далі визначаються тестові змінні для типу контенту форми і для екземпляра клієнта TestClient. Затім слідують два тестові методи test_openapi_schema та test_login_for_access_token.

У методі test_openapi_schema перевіряється статус-код відповіді на запит до шляху /openapi.json, який повинен бути 200, щоб довести, що структура API OpenAPI документа вірна.

У методі test_login_for_access_token створюється підрядок з бібліотекою mock.patch, для того, щоб замінити функцію аутентифікації на функцію, яка повертає фіксований об'єкт користувача.

Після цього тестуються запити на видачу токена доступу, перевіряється, що відповідь містить три поля access_token, refresh_token, token_type, та що поле token_type дорівнює bearer. Нарешті, перевіряється, що поля access_token та refresh_token є стрічками і не є порожніми.

```python
async def override_dependency():
    return User(**{
        'id': 777,
        'username': 'test',
    })


form_urlencoded = "application/x-www-form-urlencoded"

app.user_middleware.clear()
app.middleware_stack = app.build_middleware_stack()

client = TestClient(app)


def test_openapi_schema():
    response = client.get("/openapi.json")
    assert response.status_code == 200, response.text


def test_login_for_access_token():
    with patch('routers.security.authenticate_user') as perm_mock:
        perm_mock.return_value = User(**{
            'id': 777,
            'username': 'test',
        })
        response = client.post('security/token',
                               data={"username": "johndoe", "password": "secret", "grant_type": "password"},
                               headers={"content-type": form_urlencoded})

        assert response.status_code == 200
        assert all(i in ('access_token', 'refresh_token', 'token_type') for i in response.json().keys())
        assert response.json()['token_type'] == 'bearer'
        assert type(response.json()['access_token']) is str and len(response.json()['access_token'])
        assert type(response.json()['refresh_token']) is str and len(response.json()['refresh_token'])
```

Рисунок 3.11 – Модуль Test_app

3.4 Налаштування тестового середовища для проекту

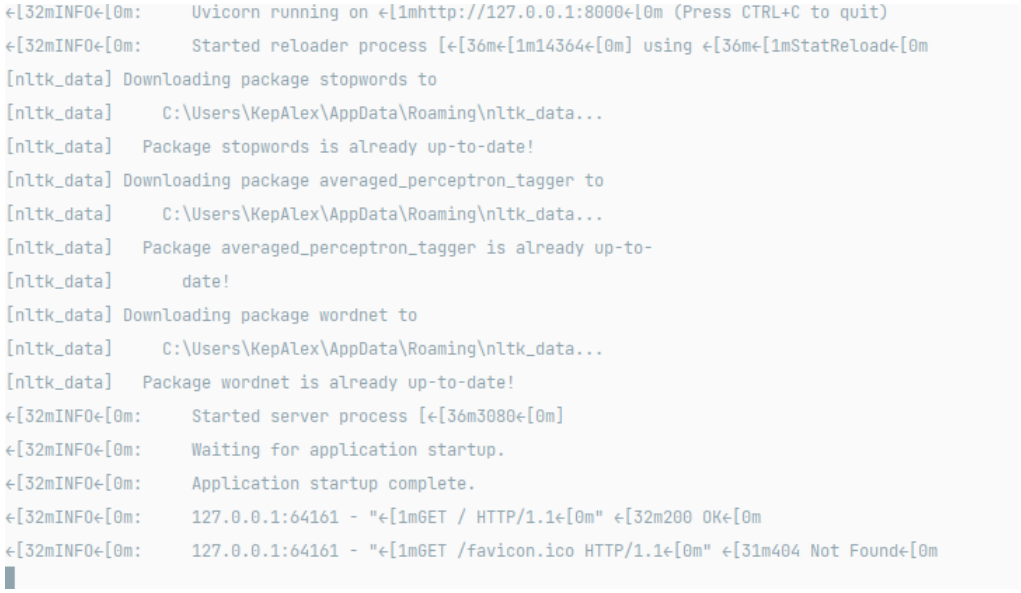


Рисунок 3.12 Результат запуску серверу uvicorn

Для запуску сервера, потрібно запустити команду uvicorn із параметрами шляху до основного модуля FastAPI та обраного вами порту. Наприклад, якщо ваш основний файл називається main.py та ви хочете запустити сервер на порту 8000, то команда для запуску буде наступною:

uvicorn src.api.main:app --port 8000

У цій команді main:app вказує на модуль FastAPI (main.py), та app - на інстанцію класу FastAPI, що створюється в основному файлі, а --port 8000 вказує на порт, на якому буде запущений сервер.

ВИСНОВОК ДО 3 РОЗДІЛУ

У третьому розділі дипломного проєкту були розроблені основні елементи системи тематичного моделювання тексту. Перш за все, у цьому розділі був проведений детальний опис структури проекту, що передбачає розробку системи. Описано призначення та функції основних та додаткових складових проекту, що включають алгоритми, використовувані для створення тематичних моделей та обробки текстів.

У цьому розділі були докладно розглянуті алгоритми, які використовуються у системі для створення тематичних моделей. Це можуть бути алгоритми, такі як латентне розміщення Діріхле (LDA) або інші методи, які дозволяють класифікувати текстову інформацію за темами. Також були описані алгоритми для обробки текстів, що включають у себе завдання очищення даних, лематизації, векторизації та інших операцій, що допомагають підготувати дані для подальшого аналізу.

Окрім того, у третьому розділі був описаний процес налаштування тестового середовища для проекту. Це включає встановлення необхідного програмного забезпечення, налаштування параметрів системи, визначення робочих процесів та створення тестових наборів даних для перевірки працездатності системи.

Таким чином, третій розділ дипломного проєкту пропонує детальний опис основних елементів системи тематичного моделювання тексту, включаючи структуру проекту, функції складових, використовувані алгоритми та процес налаштування тестового середовища.

РОЗДІЛ 4
ЕКСПЕРИМЕНТИ ТА РЕЗУЛЬТАТИ

4.1 Опис програмного інтерфейсу

Програмний інтерфейс умовно поділений на дві складові – ендпоїнти для аналізу тексту за допомогою наявних моделей, та ендпоїнти для створення моделей. Ці складові зовуться роутерами, та кожен з цих роутерів захищений за допомогою JWT токенів за схемою OAuth.

Перший Роутер має префікс /trainer та є захищеним, тому його можуть використовувати лише користувачі з правами "User". У роутері є один ендпойнт /train_model/, який приймає файли з даними та параметри метаданих, таких як delimiter, column, topics_num та stop_words. Під час передачі датасету для навчання варто вказувати колонку з якої буде братися текст.

Далі генерується випадковий ідентифікатор та створюється об'єкт Trainer з параметрами num_topics та stop_words. Задача Trainer оброблюється як фонова задача, і її виконання додається до background_tasks. Коли задача виконається, повертається ідентифікатор, за яким можна отримати модель.

З цим ідентифікатором згодом можна звернутися до роутеру analyser щоб або отримати візуалізацію моделі, або отримати аналіз теми нового окремого тексту за наявною моделлю.

Отриманні дані проходять етап очистки та подаються на створення моделей LDA.

4.2 Демонстрація розробленого застосунку

Так як застосунок являє собою лише програмний інтерфейс, то для наочності використаємо документацію Swagger на наш API.

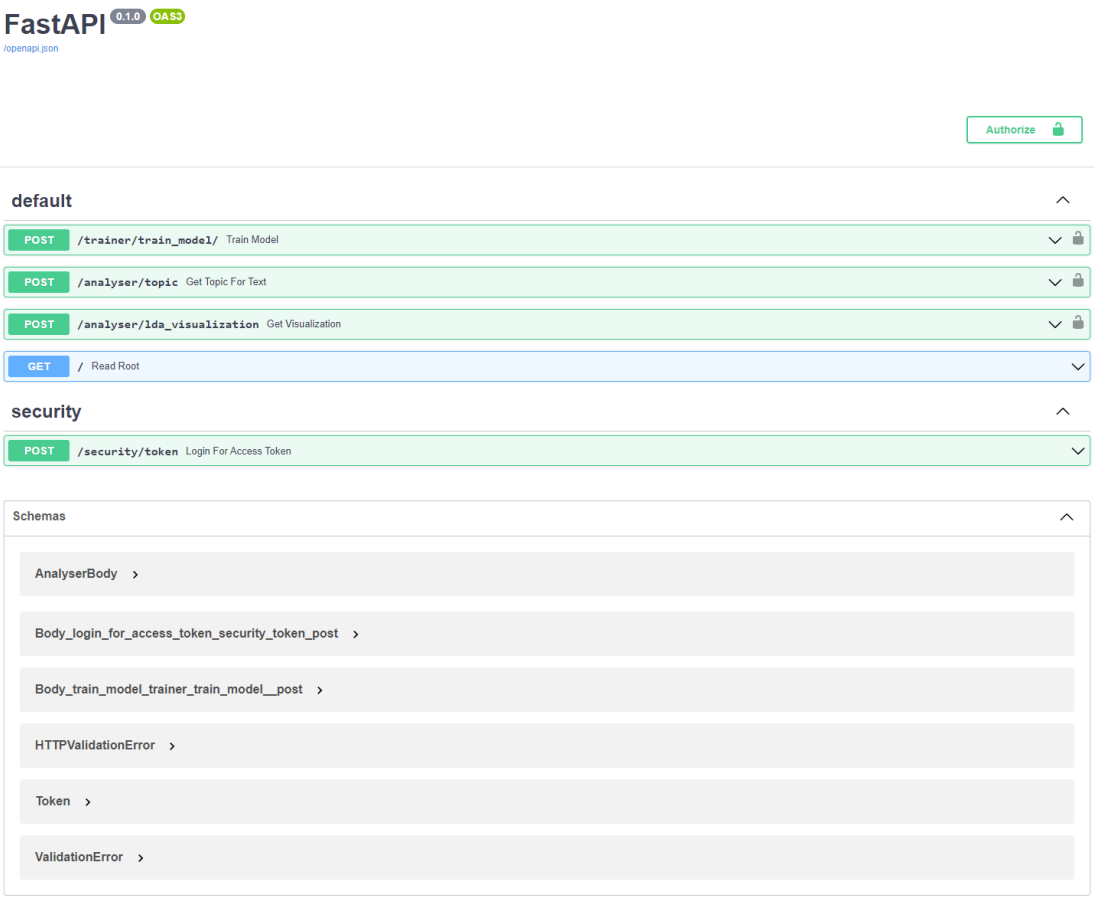


Рисунок 4.1 - Swagger документація застосунку

На сторінці за адресою 127.0.0.1/docs можна знайти опис всіх можливих форматів даних(схем), наявних ендпоїнтів (бачимо розбиття на analyser, trainer, security). Зверху сторінки є кнопка авторизації користувача, при натисканні якої відкривається вікно авторизації (див. рис. 4.2)

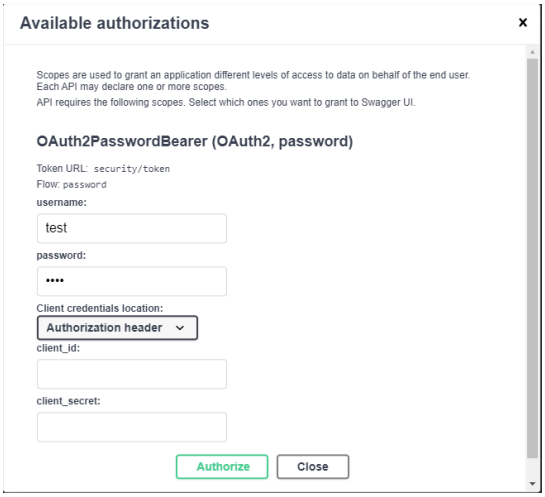


Рисунок 4.2 - Вікно авторизації

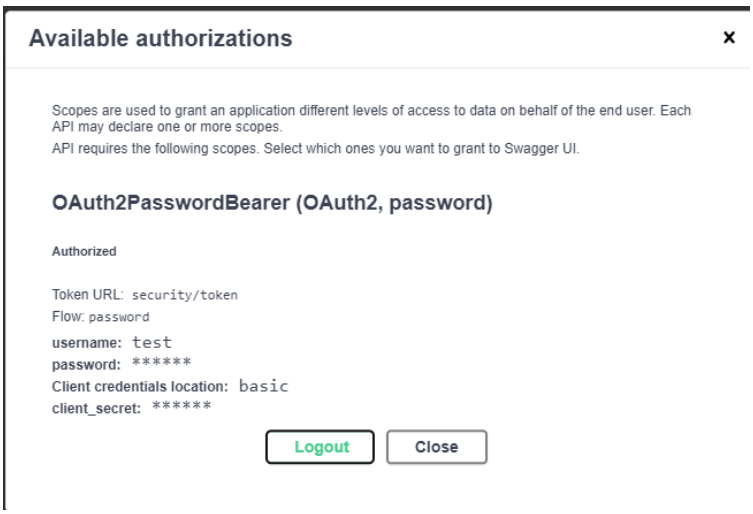


Рисунок 4.3 - Авторизація успішна

Ендпойнт /train_model/, який приймає файли з даними та параметри метаданих, таких як delimiter, column, topics_num та stop_words.

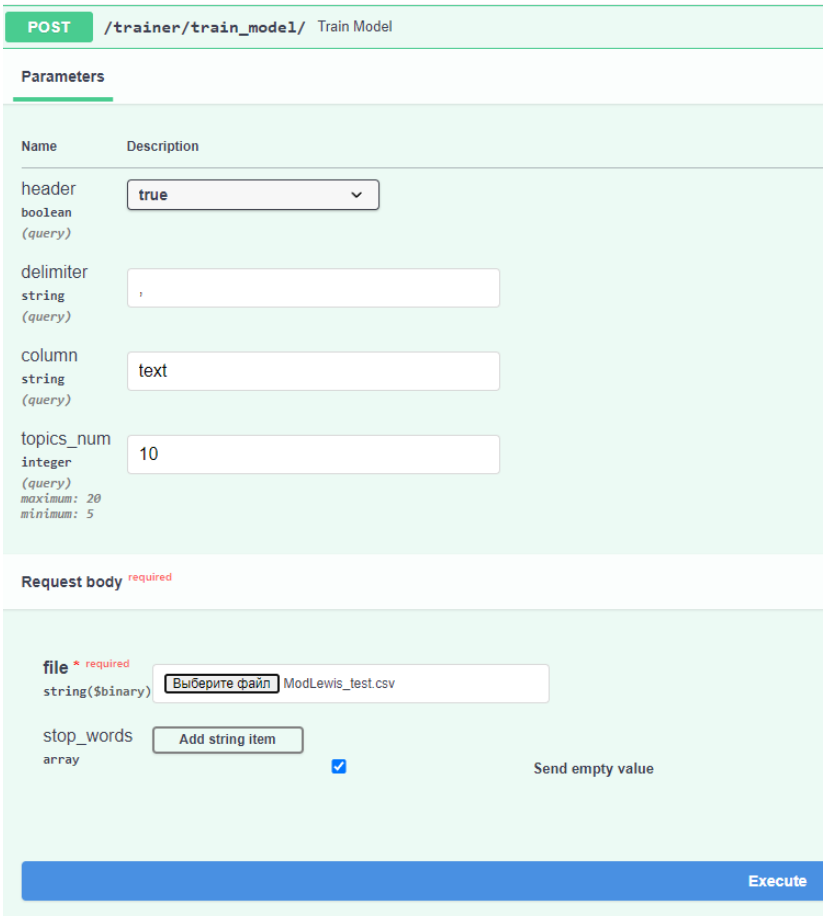


Рисунок 4.4 - Ендпоінт для створення моделі

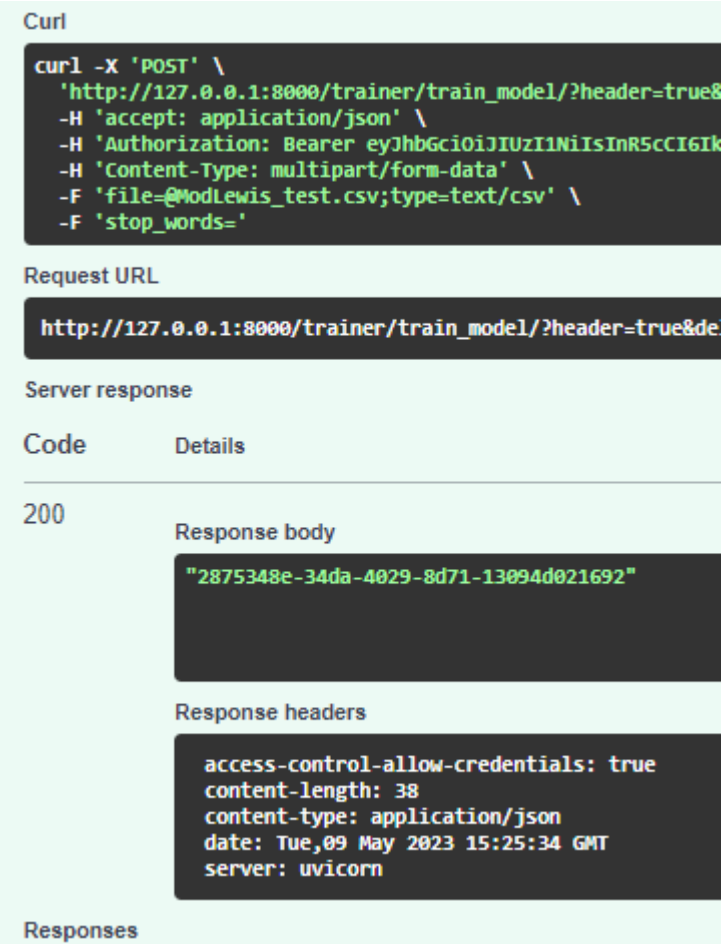


Рисунок 4.5 - Приклад запиту на створення моделі

Ми обрали файл з якого хочему отримати розбивку по темам. Також була зазначена колонка в файлі в якій зберігаються всі тексті. Swagger автоматично прикріплює авторизаційний заголовок до кожного запиту на захищенні ендпоінти. У відповідь ми отримати id моделі який потім можемо використовувати.

Для отримання візуалізації навченої моделі використовуємо ендпоінт lda_visualization. При передачі ідентифікатора моделі ми отримаємо у відповідь html файл що дозволяє відобразити теми, що виявлені в корпусі документів, та їхній зв'язок зі словами в текстах (див. рис. 4.6).

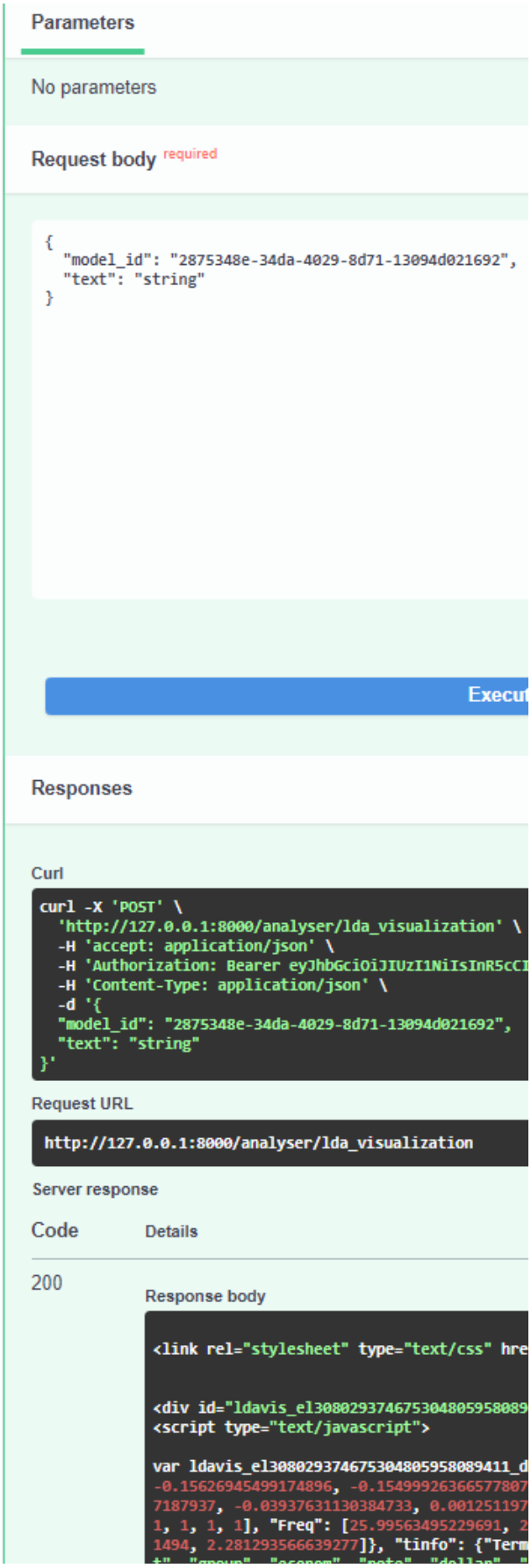


Рисунок 4.6 - Запит на отримання візуалізації

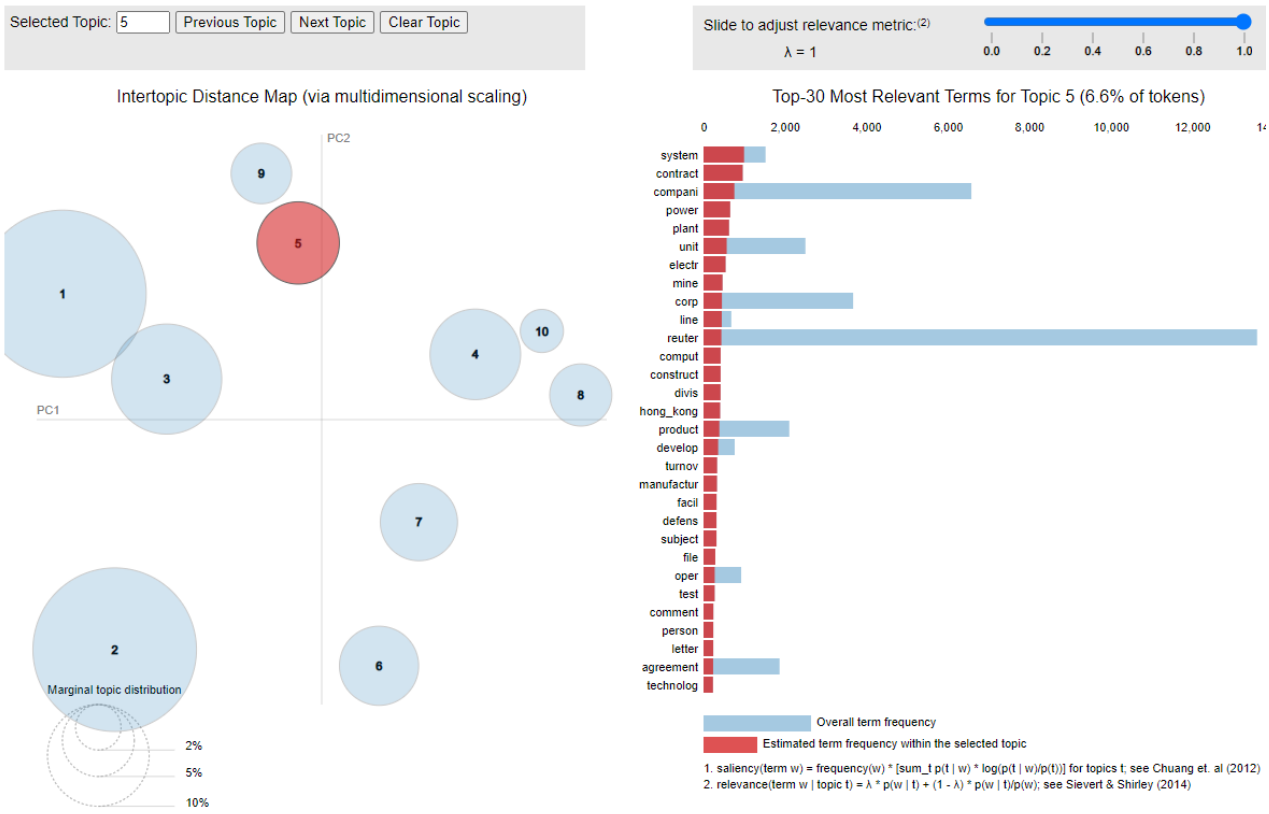


Рисунок 4.7 - Візуалізація розбиття тем

PyLDAvis відповідає за розбиття корпусу документів на теми та відображення цих тем у вигляді кругових діаграм та діаграм розсіювання, що дозволяє зрозуміти, які теми переважають у корпусі та як вони пов'язані між собою. Також PyLDAvis дозволяє відобразити список найбільш імовірних слів для кожної теми та дозволяє фільтрувати ці слова за різними критеріями, такими як значення ваги терміну (term weight) або значення різниці між частотою вживання терміну в даній темі та його загальної частоти в корпусі.

Крім того, PyLDAvis дозволяє відобразити залежність між темами та документами в корпусі. За допомогою інтерактивних інструментів, можна відобразити текст кожного документа, який відноситься до відповідної теми, або відобразити список документів, які відносяться до вибраної теми.

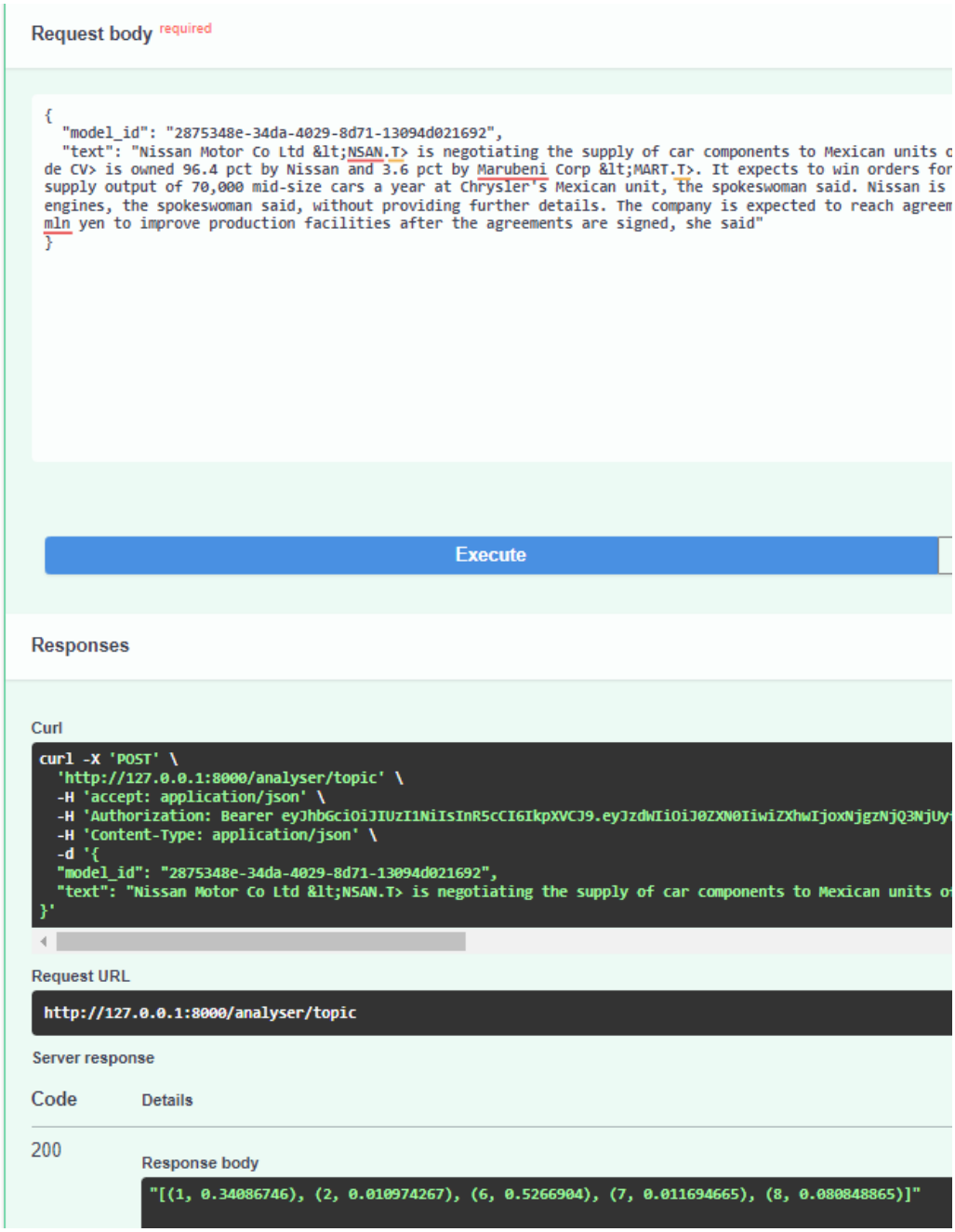


Рисунок 4.8 - Запит на аналіз окремого тексту

Для аналізу окремо взятого тексту та його відношення в рамках тієї чи іншої моделі можна використати ендпоінт /topic. Ми подаємо ідентифікатор моделі та власне сам текст, а у відповідь отримаємо розбиття вірогідності приналежності тексту до тієї чи іншої теми.

ВИСНОВОК ДО 4 РОЗДІЛУ

Результатом виконання четвертого розділу дипломного проекту стала розроблена система програмного інтерфейсу тематичного моделювання тексту. В рамках дослідження було проведено аналіз та розглянуто існуючі ендпоїнти системи, які дозволяють працювати з моделями тематичного моделювання. Також був розглянутий алгоритм роботи з системою, який полягає у створенні тематичної моделі на основі вхідних даних, та відображенні результатів моделювання у зручному форматі.

Описані результати моделювання демонструють ефективність тематичного моделювання тексту у побудові моделей, які дозволяють отримувати корисну інформацію з великої кількості даних. Зокрема, було проведено дослідження ефективності розробленої системи на прикладі аналізу текстових документів різного типу та структури. Отримані результати свідчать про успішність використання системи для вирішення різноманітних завдань, пов'язаних з аналізом тексту.

Підсумовуючи, можна зазначити, що розроблена система програмного інтерфейсу тематичного моделювання тексту є потужним інструментом для аналізу та обробки великого обсягу даних. Вона дозволяє здійснювати ефективну роботу з текстом та отримувати корисну інформацію для різноманітних завдань.

ВИСНОВКИ

В результаті дипломного проєкту була розроблена система програмного інтерфейсу тематичного моделювання тексту.

Розділ 1 роботи присвячений аналізу сучасних підходів до тематичного моделювання тексту, вимог до систем, які використовують цей підхід, аналізу наявних алгоритмів та існуючих систем.

У розділі 2 було розглянуто вибір архітектури та програмних засобів реалізації системи, включаючи вибір технологій розробки серверної частини та NLP, а також опис алгоритму LDA.

В розділі 3 описано розроблення основних елементів системи, включаючи структуру проекту та призначення основних складових. Були описані основні складові проекту, такі як routers.analyser.py, router.trainer.py, data bodies, TextPreProcessor, Trainer.py, а також додаткові складові проекту, такі як ci.yml та Test_app.py.

Розділ 4 містить опис розробленої системи тематичного моделювання тексту, включаючи опис програмного інтерфейсу та демонстрацію розробленого застосунку. Загальний висновок полягає у тому, що розроблена система програмного інтерфейсу тематичного моделювання тексту відповідає вимогам та є результатом аналізу наявних підходів та систем, використовує вибрані технології NLP та алгоритм LDA, має структуру проекту та призначення основних та додаткових складових.

Результатом є опис програмного інтерфейсу та демонстрація розробленого застосунку, який дозволяє здійснювати тематичне моделювання тексту.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Імовірнісні моделі: від наївного Байєса до LDA [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/companies/surfingbird/arвticles/228249/.
2. Lee H. Unsupervised feature learning for audio classification [Електронний ресурс] / Honglak Lee – Режим доступу до ресурсу: http://www.robotics.stanford.edu/~ang/papers/nips09-AudioConvolutionalDBN.pdf.
3. google.com [Електронний ресурс] – Режим доступу до ресурсу: https://news.google.com/home.
4. buzzsumo [Електронний ресурс] – Режим доступу до ресурсу: https://buzzsumo.com/
5. Thomson Reuters [Електронний ресурс] – Режим доступу до ресурсу: https://www.thomsonreuters.com/en.html.
6. Elektron [Електронний ресурс] – Режим доступу до ресурсу: https://3.bp.blogspot.com/-cYG2IJigEPI/Vfbt9kv5IHI/AAAAAAAARsM/iCyvY1NByyk/s1600/elektron-infographic-large.jpg
7. FastAPI [Електронний ресурс] – Режим доступу до ресурсу: https://fastapi.tiangolo.com/
8. Pydantic [Електронний ресурс] – Режим доступу до ресурсу: docs.pydantic.dev
9. Uvicorn [Електронний ресурс] – Режим доступу до ресурсу: https://www.uvicorn.org.
10. A Beginner’s Guide to Latent Dirichlet Allocation [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/latent-dirichlet-allocation-lda-9d1cd064ffa2

11. Ілюстрація розкидання ключових слів на теми [Електронний ресурс] – Режим доступу до ресурсу: https://miro.medium.com/v2/resize:fit:1400/0*UtPk76NZQo2Y30Vy.png.
12. ryanong.co.uk [Електронний ресурс] – Режим доступу до ресурсу: https://ryanong.co.uk/wp-content/uploads/2020/08/fig2-4.png.
13. Topic Modeling and Latent Dirichlet Allocation [Електронний ресурс] – Режим доступу до ресурсу: https://www.analyticsvidhya.com/blog/2021/06/part-2-topic-modeling-and-latent-dirichlet-allocation-lda-using-gensim-and-sklearn.
14. A Beginner’s Guide to Latent Dirichlet Allocation [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/latent-dirichlet-allocation-lda-9d1cd064ffa2
15. gensim [Електронний ресурс] – Режим доступу до ресурсу: https://radimrehurek.com/gensim.
16. spacy [Електронний ресурс] – Режим доступу до ресурсу: https://spacy.io/.
17. A Beginner’s Guide to Latent Dirichlet Allocation [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/latent-dirichlet-allocation-lda-9d1cd064ffa2
18. Ілюстрація розкидання ключових слів на теми [Електронний ресурс] – Режим доступу до ресурсу: https://miro.medium.com/v2/resize:fit:1400/0*UtPk76NZQo2Y30Vy.png

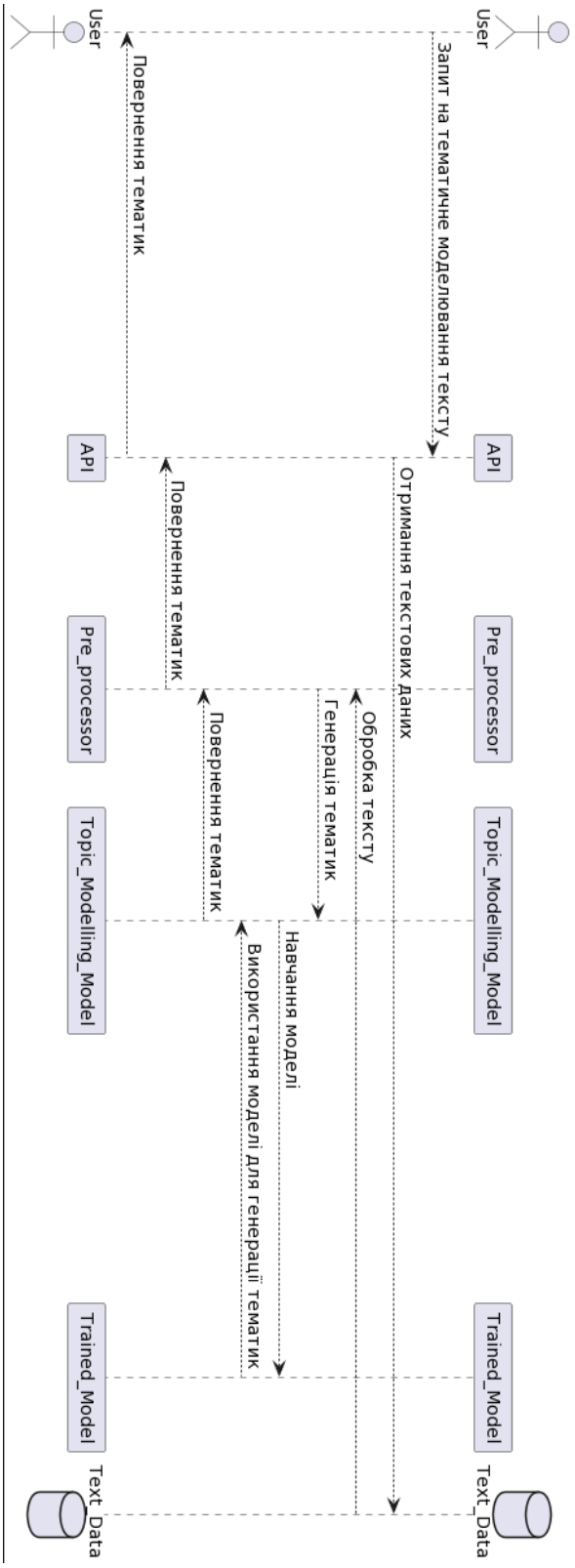
ДОДАТОК А

Програмний інтерфейс тематичного моделювання тексту

Структурна схема системи

ІАЛЦ.467200.004 Д1

Київ – 2023 р



					ІАЛЦ.467200.004 Д1	
		№ докум.	Підпис	Дата		
Розробив		Гришин О.С.			***Програмний інтерфейс тематичного моделювання тексту*** *Структурна схема системи*	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Кочура Ю.П.				
Н. Контр.		Виноградов Ю.М.				КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-91
Затвердив						

ДОДАТОК Б

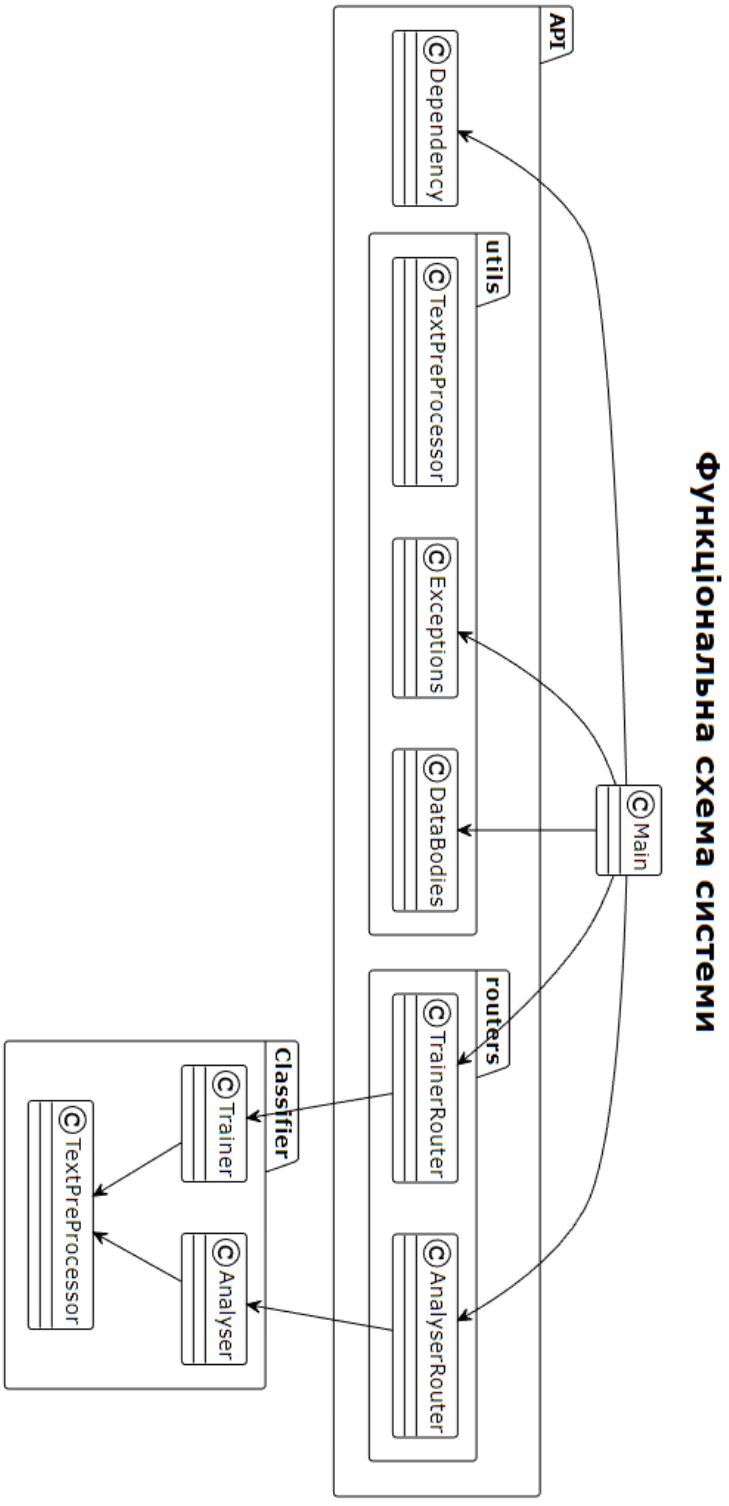
Програмний інтерфейс тематичного моделювання тексту

Функціональна схема

(діаграма класів)

ІАЛЦ.467200.005 Д2

Київ – 2023 р.



Функціональна схема системи

					ІАЛЦ.467200.005 Д2	
		№ докум.	Підпис	Дата		
Розробив		Гришин О.С.			***Програмний інтерфейс тематичного моделювання тексту***	Літ. / Аркуш / Аркушів: 1 / 1
Перевірив		Кочура Ю.П.				
Н. Контр.		Виноградов Ю.М.			*Функціональна схема*	КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-91
Затвердив						

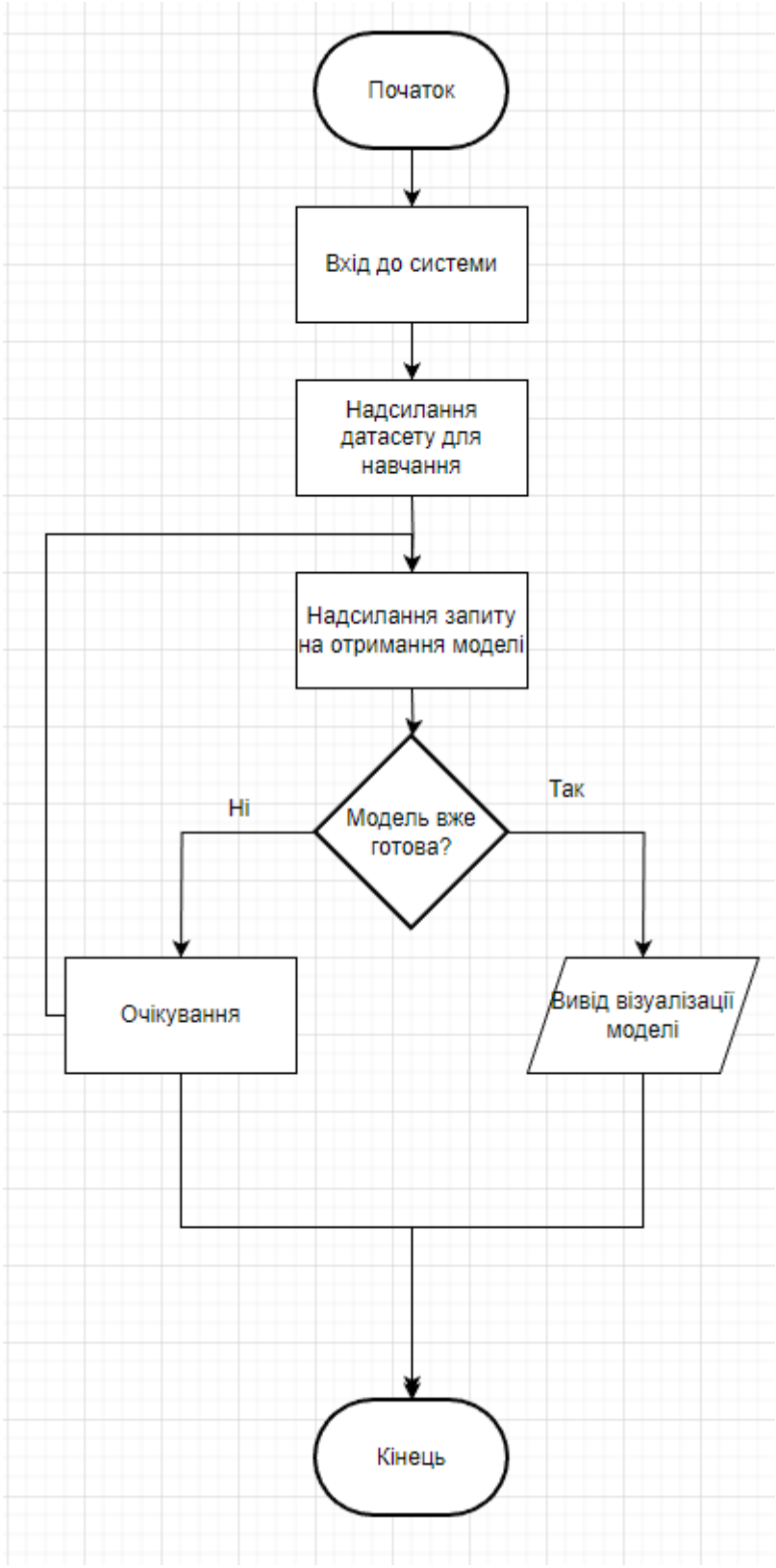
ДОДАТОК В

Програмний інтерфейс тематичного моделювання тексту

Алгоритм роботи системи

ІАЛЦ.467200.006 Д3

Київ – 2023 р.



					ІАЛЦ.467200.006 Д3		
		№ докум.	Підпис	Дата			
Розробив		Гришин О.С.			***Програмний інтерфейс тематичного моделювання тексту***	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Кочура Ю.П.					
Н. Контр.		Виноградов Ю.М.			*Алгоритм роботи системи*	КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-91	
Затвердив							

ДОДАТОК Г

Програмний інтерфейс тематичного моделювання тексту

Текст програмного коду

ІАЛЦ.467200.007 Д4

Київ – 2023 р.

```
# -*- coding: utf-8 -*-
from fastapi import FastAPI, Request
from fastapi.middleware.cors import CORSMiddleware
from fastapi_cache import FastAPICache
from fastapi_cache.backends.inmemory import InMemoryBackend
from starlette import status

from src.api.routers import trainer, analyser, Dependency

app = FastAPI(redoc_url=None)

"""подключаем все роутеры апи"""
app.include_router(trainer.router)
app.include_router(analyser.router)
app.include_router(Dependency.router)

WHLITELISTED_IPS = ['127.0.0.1']

origins = [
    "http://localhost.tiangolo.com",
    "https://localhost.tiangolo.com",
    "http://localhost",
    "http://localhost:8080",
]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)


@app.on_event("startup")
async def startup():
    """инициализация кеша"""
    FastAPICache.init(InMemoryBackend(), prefix="fastapi-cache")


@app.get("/")
async def read_root(request: Request):
    return status.HTTP_200_OK

# -*- coding: utf-8 -*-

from fastapi import APIRouter, Security
from fastapi.responses import FileResponse

from src.api.routers.Dependency import get_current_active_user
from src.api.utils.DataBodies import AnalyserBody
from src.classifier.Analyser import Analyser
```

```
router = APIRouter(
    prefix="/analyser",
    responses={404: {"description": "Not found"}},
    dependencies=[Security(get_current_active_user, scopes=["User"])]
)


@router.post('/topic')
def get_topic_for_text(body: AnalyserBody):
    analyser = Analyser(body.model_id)
    prediction = analyser.define_text(body.text)
    return str(prediction)


@router.post('/lda_visualization')
def get_visualization(body: AnalyserBody):
    analyser = Analyser(body.model_id)
    file_path = analyser.get_visualization_path()
    return FileResponse(file_path, filename="index.html",
media_type="text/html")
```

```
# -*- coding: utf-8 -*-

import uuid

import pandas as pd
from fastapi import APIRouter, BackgroundTasks, UploadFile, File, Depends,
Security

from src.api.routers.Dependency import get_current_active_user
from src.api.utils.DataBodies import Meta
from src.classifier.Trainer import Trainer

router = APIRouter(
    prefix="/trainer",
    responses={404: {"description": "Not found"}},
    dependencies=[Security(get_current_active_user, scopes=["User"])]
)


@router.post('/train_model/')
async def train_model(background_tasks: BackgroundTasks, file: UploadFile =
File(...), meta: Meta = Depends()):
    # todo chose column
    df = pd.read_csv(file.file, delimiter=meta.delimiter)
    data = df[meta.column].tolist()
    random_id = uuid.uuid4()
    trainer = Trainer(num_topics=meta.topics_num)
    background_tasks.add_task(trainer.process, data, random_id)
    # return id on which we can call for a model
    return random_id
```

```
# -*- coding: utf-8 -*-
from typing import Optional, List

from pydantic import BaseModel, conint

PATTERN = r'\p{IsCyrillic}'

DATE_FORMAT = "%Y-%m-%d"
```

```
DATE_FORMAT_TIME = "%Y-%m-%d %H:%M:%S"


class Subscribe(BaseModel):
    user_id: int


class Meta(BaseModel):
    header: bool = True
    delimiter: str = ","
    column: Optional[str] = 'text'
    topics_num: conint(ge=5, le=20) = 10
    stop_words: Optional[List[str]]


class AnalyserBody(BaseModel):
    model_id: str
    text: Optional[str]


class Token(BaseModel):
    access_token: str
    token_type: str


class TokenData(BaseModel):
    username: Optional[str] = None


class User(BaseModel):
    username: str
    email: Optional[str] = None
    full_name: Optional[str] = None
    disabled: Optional[bool] = None


class UserInDB(User):
    hashed_password: str

from fastapi import HTTPException, status

credentials_exception = HTTPException(
    status_code=status.HTTP_401_UNAUTHORIZED,
    detail="Could not validate credentials",
)

empty_body_exception = HTTPException(
    status_code=status.HTTP_400_BAD_REQUEST,
    detail="WRONG JSON body",
)
from typing import List, Union

import nltk
from nltk import word_tokenize, SnowballStemmer, pos_tag
from nltk.corpus import stopwords

nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')


class TextPreProcessor:
```

```
    stop_words = stopwords.words('english')
    tags = []

    def __init__(self, stop_words=None, tags=None):
        if tags is None:
            self.tags = ['NN', 'JJ']
        else:
            self.tags = tags

        if stop_words is not None:
            self.stop_words.extend(stop_words)

    def process(self, texts: Union[List[str], str]) -> list[list]:
        if type(texts) is str:
            texts = [texts]

        stemmer = SnowballStemmer('english')
        result = []
        for doc in texts:
            # Токенизация
            try:
                words = word_tokenize(doc.lower())
            except AttributeError:
                continue
            # Определение частей речи
            tagged_words = pos_tag(words)
            # Оставляем только существительные и прилагательные
            filtered_words = [word for word, tag in tagged_words if tag in
self.tags]
            # Удаление стоп-слов
            filtered_words = [word for word in filtered_words if
                              word not in self.stop_words and word.isalpha and
len(word) > 3]
            # Применение стеммера
            filtered_words = [stemmer.stem(word) for word in filtered_words]
            result.append(filtered_words)
        return result
import os

from gensim.corpora import Dictionary
from gensim.models import LdaModel

from src.api.utils.TextPreProcessor import TextPreProcessor


class Analyser:
    lda_model = None
    dictionary = None

    def __init__(self, model_id):
        self.pre_processor = TextPreProcessor()
        self.model_id = model_id
        self.load_model()

    def get_visualization_path(self):
        return 'warehouse/ldavis_prepared_' + str(self.model_id) + '.html'

    def load_model(self):
        ld_avis_data_filepath = os.path.join('warehouse/ldavis_prepared_' +
str(self.model_id))
        self.lda_model = LdaModel.load(ld_avis_data_filepath + '_model')
```

```
        self.dictionary = Dictionary.load(ld_avis_data_filepath +
'_dictionary')

    def define_text(self, text: str):
        processed_text = self.pre_processor.process(text)[0]
        bow_vector = self.dictionary.doc2bow(processed_text)
        topic_probabilities = self.lda_model.get_document_topics(bow_vector)
        return topic_probabilities
import os

import gensim
import gensim.corpora as corpora
import pyLDAvis
import pyLDAvis.gensim_models as gensimvis
from gensim.models import CoherenceModel

from src.api.utils.TextPreProcessor import TextPreProcessor


class Trainer:
    """
    Gives functionality to train new LDA models
    """
    corpus = None
    id2word = None
    num_topics = None
    lda_model = None

    def __init__(self, stop_words=None, tags=None, num_topics=10):
        self.pre_processor = TextPreProcessor(stop_words, tags)
        self.num_topics = num_topics

    @staticmethod
    def make_trigrams(data_words):
        # Build the bigram and trigram models
        bigram = gensim.models.Phrases(data_words, min_count=5, threshold=100)
# higher threshold fewer phrases.
        trigram = gensim.models.Phrases(bigram[data_words], threshold=100)
        # Faster way to get a sentence clubbed as a trigram/bigram
        bigram_mod = gensim.models.phrases.Phraser(bigram)
        trigram_mod = gensim.models.phrases.Phraser(trigram)

        def trigrams(texts):
            return [trigram_mod[bigram_mod[doc]] for doc in texts]

        # Form Bigrams
        data_words_trigrams = trigrams(data_words)
        return data_words_trigrams

    def save_model(self, model_id):
        ld_avis_data_filepath = os.path.join('warehouse/ldavis_prepared_' +
str(model_id))
        # save LdaModel
        self.lda_model.save(ld_avis_data_filepath+'_model')
        # save Dictionary
        self.id2word.save(ld_avis_data_filepath+'_dictionary')

        lda_prepared = gensimvis.prepare(self.lda_model, self.corpus,
self.id2word)
        pyLDAvis.save_html(lda_prepared, ld_avis_data_filepath+'.html')
```

```
    def build_model(self, model_id: str):
        # Build LDA model
        self.lda_model = gensim.models.LdaModel(corpus=self.corpus,
                                                id2word=self.id2word,
                                                num_topics=self.num_topics,
                                                random_state=100,
                                                update_every=1,
                                                chunksize=100,
                                                passes=10,
                                                alpha='auto',
                                                per_word_topics=True
                                                )

        self.save_model(model_id)

    def process(self, data, model_id):
        data_words = self.pre_processor.process(data)
        data_words_trigrams = self.make_trigrams(data_words)
        self.id2word = corpora.Dictionary(data_words_trigrams)
        # Term Document Frequency
        self.corpus = [self.id2word.doc2bow(text) for text in
data_words_trigrams]

        self.build_model(model_id)
from fastapi.testclient import TestClient

from src.api.main import app

client = TestClient(app)


def test_read_main():
    response = client.get("/")
    assert response.status_code == 200
```

ci.yml

```
on:
  push:
jobs:
  build:
    runs-on: ubuntu-latest
    if: github.ref_type == 'branch'
    steps:
    - name: Checkout code
      uses: actions/checkout@v2
    - name: Setup Python
      uses: actions/setup-python@v2
      with:
        python-version: '3.x'
    - name: Install dependencies
      run: pip install -r requirements.txt
    - name: Run tests
      run: pytest
    - name: Notify on failure
      if: failure()
      run: echo "Tests failed."
```