НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Вебзастосунок для автоматичного підбору вакансій на основі резюме та адаптації резюме за допомогою нейромереж для ІТ-галузі. API машинного навчання

Виконав студент IV курсу, групи ІП-11
(шифр групи)

Сідак Кирил Ігорович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к.т.н., доц., Ліщук К. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант доцент, к.т.н., доц., Ліщук К. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доц. каф.ІСТ, к.т.н., доц., Писаренко А. В. _____
(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 – 2025

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2025 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Сідаку Кирилу Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Вебзастосунок для автоматичного підбору вакансій на основі резюме та адаптації резюме за допомогою нейромереж для ІТ-галузі. API машинного навчання

керівник проєкту Ліщук Катерина Ігорівна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «23» травня 2025 р. №1705-с

2. Термін подання студентом проєкту «16» червня 2025 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі.
2) Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних.
3) Математичне забезпечення: змістовна та математична постановки задачі, обґрунтування та опис методу розв'язання.
4) Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів.
5) Технологічний розділ: керівництво користувача, методика випробувань програмного продукту.

5. Перелік графічного матеріалу

1) Схема структурна компонентів програмного забезпечення _____

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «15» березня 2025 року _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	15.03.2025	
2	Аналіз існуючих методів розв’язання задачі	25.03.2025	
3	Постановка та формалізація задачі	05.04.2025	
4	Розробка інформаційного забезпечення	15.04.2025	
5	Алгоритмізація задачі	20.04.2025	
6	Обґрунтування вибору використаних технічних засобів	25.04.2025	
7	Розробка програмного забезпечення	15.05.2025	
8	Налагодження програми	20.05.2025	
9	Виконання графічних документів	26.05.2025	
10	Оформлення пояснювальної записки	28.05.2025	
11	Подання ДП на попередній захист	04.06.2025	
12	Подання ДП рецензенту	10.06.2025	
13	Подання ДП на основний захист	16.06.2025	

Студент _____ Кирил СІДАК

(підпис) (ініціали, прізвище)

Керівник _____ Катерина ЛІЩУК

(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з трьох розділів, містить 13 таблиць, 6 рисунків та 23 джерела – загалом 29 сторінок.

Дипломний проєкт присвячений розробці вебзастосунку для підбору вакансій на основі резюме та адаптації резюме.

Мета: створення вебзастосунку, котрий допомагає пошукачам роботи підбирати найбільш релевантні вакансії в IT-галузі та адаптувати зміст резюме користувача від вимоги обраних вакансій, що підвищує точність процесу працевлаштування для обох сторін на ринку праці.

У першому розділі наведено опис функціональних вимог, матрицю трасування визначених функціональних вимог. В результаті визначено основні задачі, котрі потребують розробки.

У другому розділі наведено архітектуру API сервісу машинного навчання, обґрунтування основних засобів розробки та використаних бібліотек, наведено алгоритмічну складову основних розроблених методів.

У третьому розділі наведено результати аналізу якості та тестування API сервісу машинного навчання.

Програмне забезпечення впроваджено на хостингу Render.

КЛЮЧОВІ СЛОВА: ВАКАНСІЇ, РЕЗЮМЕ, НЕЙРОМЕРЕЖІ, BERT, FASTAPI.

ABSTRACT

The explanatory note of the diploma project consists of three sections, contains 13 tables, 6 figures and 23 sources – in total 29 pages.

The diploma project is dedicated to the development of a web application for job search based on resumes and resume adaptation.

The purpose of the diploma project is to create a web application that helps job seekers find the most relevant vacancies in the IT industry and adapt the content of the user's resume to the requirements of the selected vacancies, which increases the accuracy of the employment process for both parties in the labour market.

The first section provides a description of functional requirements and a traceability matrix for the defined functional requirements. As a result, the main tasks that need to be developed are identified.

The second section presents the architecture of the machine learning service API, the rationale for the main development tools and libraries used, and the algorithmic component of the main methods developed.

The third section presents the results of the quality analysis and testing of the machine learning API service.

The software is hosted on Render.

KEYWORDS: VACANCIES, RESUME, NEURAL NETWORKS, BERT, FASTAPI.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-1124.045440.02.81	Пояснювальна записка	29	
3	А4	КПІ.ІП-1124.045440.03.12	Текст програми	15	
4	А4	КПІ.ІП-1124.045440.04.51	Програма та методика тестування	6	
5	А3	КПІ.ІП-1124.045440.06.99	Графічний матеріал	1	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-1124.045440.00.90		
Розроб.		Сідак К.І.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Ліщук К.І.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-11	
Н.контр.		Ліщук К.І.					
Затв.		Жаріков Е.В.					

Пояснювальна записка
до дипломного проєкту

на тему: Вебзастосунок для автоматичного підбору вакансій на основі резюме та адаптації резюме за допомогою нейромереж для ІТ-галузі. API машинного навчання

КПІ.ІП-1124.045440.02.81

Київ – 2025

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE – Integrated Development Environment – інтегроване середовище розробки.

API – Application programming interface, прикладний програмний Інтерфейс.

IT – Інформаційні технології.

ВСТУП

Роль IT галузі у сучасному світі постійно збільшується, причому сама галузь стає все більш конкурентною, тому навіть для досвідчених спеціалістів пошук роботи є досить тривалим та складним процесом. Зважаючи на такий стан речей, швидкий автоматизований пошук релевантних вакансій та можливість отримати рекомендації для адаптації резюме під конкретну вакансію дозволяє значно спростити та пришвидшити процес пошуку бажаної роботи в даній галузі.

Дипломний проєкт присвячено розробці вебзастосунку для пошуку вакансій пошукачами роботи та отримання рекомендацій до адаптації резюме. Метою дипломного проєкту є створення вебзастосунку, котрий допомагає пошукачам роботи підбирати найбільш релевантні вакансії в IT-галузі та адаптувати зміст резюме користувача від вимоги обраних вакансій, що підвищує точність процесу працевлаштування для обох сторін на ринку праці

У рамках індивідуальної частини роботи дипломного проєкту виконано навчання нейромережі для класифікації резюме за категорією роботи з попереднім збором даних, розроблено алгоритми пошуку релевантних вакансій, надання рекомендацій по адаптації резюме під вакансію з використанням нейромереж, API сервіс машинного навчання, що виконує обробку резюме, вакансій та реалізує дані алгоритми.

1 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Розроблення функціональних вимог

В таблиці 1.1 наведено загальну модель вимог до програмного забезпечення, а в таблиці 1.2 наведений опис функціональних вимог до програмного забезпечення. Матриця трасування вимог наведена в таблиці 1.3.

Таблиця 1.1 – Загальна модель вимог

№	Назва	ID Вимоги	Пріоритети	Ризики
1	Обробка резюме	FR-1	Високий	Високий
1.1	Класифікація рензюме за категорією	FR-2	Високий	Високий
2	Обробка вакансій	FR-3	Високий	Середній
3	Визначення релевантності між резюме та вакансіями	FR-4	Високий	Високий
4	Адаптація резюме до кокнретної вакансії	FR-5	Високий	Середній

Таблиця 1.2 – Перелік функціональних вимог

Назва	Опис
FR-1	Обробка резюме. Після завантаження резюме повинна виконуватись автоматична попередня обробка резюме, зокрема переклад тексту резюме на англійську мову, якщо він вже написаний іншою мовою, та очищення його від інформації та символів, які не є релевантними для підбору вакансій або надання рекомендацій по адаптації до конкретної вакансії.

Продовження таблиці 1.2

FR-2	Класифікація резюме за категорією. Для виконання підбору релевантних вакансій для конкретного резюме система повинна виконати класифікацію резюме на основі тексту резюме відповідно до попередньо визначених категорій, які визначають сферу діяльності, до якої відноситься релевантні посади для даного резюме.
FR-3	Обробка вакансій. Система має дозволяти виконати попередню обробку текстів усіх нових вакансій з отриманням векторних представлень текстів вакансій та обчислити метрику релевантності між резюме кожного користувача та кожною з нових вакансій відповідної категорії.
FR-4	Визначення релевантності між резюме та вакансіями. Система має обчислювати відповідність резюме кожній з вакансій, які належать до тієї ж категорії, що й дане резюме, тобто метрику подібності тексту резюме з текстом кожної вакансії на основі попередньо обчислених векторних представлень текстів.
FR-5	Адаптація резюме до конкретної вакансії. Коли користувач завантажив резюме, система має проаналізувати відповідність резюме вакансії на основі семантичної подібності ключових слів у вакансії та резюме й повернути унікальні ключові слова у вакансії та значення метрики відповідності резюме вакансії.

Таблиця 1.3 – Матриця трасування функціональних вимог

	FR-1	FR-2	FR-3	FR-4	FR-5
UC-01					
UC-02					
UC-03					

Продовження таблиці 1.3

UC-04	+	+			
UC-05				+	
UC-06					+
UC-07			+		

1.2 Постановка завдання на розробку програмного забезпечення

Метою дипломного проєкту є створення вебзастосунку, котрий допомагає пошукачам роботи підбирати найбільш релевантні вакансії в IT-галузі та адаптувати зміст резюме користувача від вимоги обраних вакансій, що підвищує точність процесу працевлаштування для обох сторін на ринку праці.

Для досягнення поставленої мети потрібно вирішити наступні задачі:

- проектування та розробка API машинного навчання;
- розробка нейромережі для класифікації резюме;
- розробка методу автоматизованого пошуку релевантних вакансій;
- розробка методу для адаптації резюме;
- оцінка якості навченої моделі для класифікації резюме.

Висновки до розділу

У першому розділі пояснювальної записки визначено основі функціональні вимоги до індивідуальної частини дипломного проєкту, які також було відображено на матриці трасування з метою зіставлення вимог з відповідними варіантами використання для більшої наочності.

За результатами даного розділу сформульовано постановку завдання на розробку індивідуальної частини дипломного проєкту.

2 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Архітектура програмного забезпечення

Компоненти API сервісу машинного навчання є наступними:

– компонент резюме: відповідає за класифікацію резюме за категорією та попередню обробку тексту резюме, створення векторного представлення для даного резюме й збереження категорії, обробленого тексту та вектору в базі даних;

– компонент вакансій: відповідає за попередню обробку текстів нових вакансій, обчислення векторного представлення для кожної вакансії, збереження результатів в базі даних та повернення оцінки релевантності (значення метрики подібності) для резюме усіх користувачів та кожної із заданих вакансій відповідної категорії;

– компонент адаптації: відповідає за визначення ключових слів (навичок), які присутні у тексті вакансії, але відсутні у тексті резюме та обчислення відповідності на основі семантичної подібності ключових слів резюме й вакансії;

– компонент моделей: відповідає за завантаження моделей машинного навчання з Hugging Face Hub, а саме нейромереж для класифікації ключових слів, класифікації резюме та створення векторних представлень текстів;

– компонент обробки тексту: відповідає за очищення тексту від зайвої контактної інформації, такої як номер телефону та електронна пошта, та переклад тексту англійською через зовнішній сервіс.

На рисунку 2.1 наведено діаграму компонентів C4:

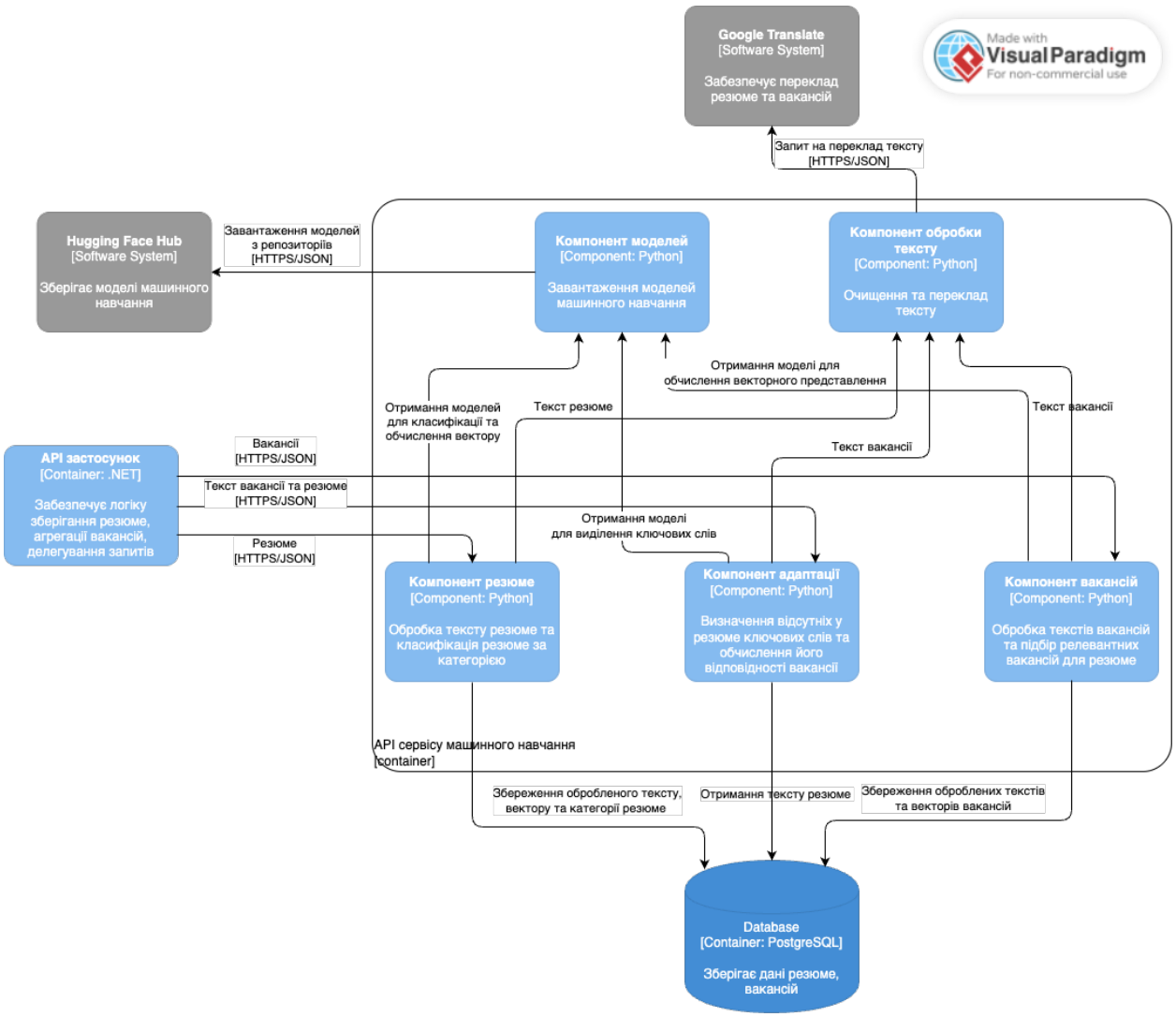


Рисунок 2.1 - Діаграма компонентів C4

2.2 Архітектурні рішення та обґрунтування вибору засобів розробки

В якості мови для розробки API сервісу машинного навчання, яке реалізовує задачі по обробці резюме, вакансій, підбору релевантних вакансій та адаптацію резюме, було обрано Python [1]. Дане рішення зумовлене тим, що дана мова має зручні бібліотеки для роботи з нейромережами (створення, навчання, донавчання та зручна інтеграція) та для машинного навчання в цілому. Крім того, дана мова має фреймворки для створення легких та високопродуктивних API, зокрема FastAPI.

У даному API наявна інтеграція з декількома зовнішніми сервісами, а саме з Hugging Face Hub [2] та Google Translate [3]. Усі моделі нейромереж, використані в API (як сторонні, так і попередньо натреновані), зберігаються на

Hugging Face Hub та завантажуються звідти через HTTPS-протокол у форматі JSON для роботи з ними. Переклад як резюме, так і вакансій англійською мовою здійснюється за допомогою бібліотеки translators [4], яка надсилає HTTP-запити, що імітують поведінку браузера, до вебінтерфейсу Google Translate або ж іншого заданого сервісу для перекладу. Таким чином, дана бібліотека дозволяє перекладати велику кількість текстів безкоштовно. Серед доступних сервісів перекладу було обрано саме Google Translate через його підтримку української мови.

Для створення даного API було обрано фреймворк FastAPI [5]. Також розглядались фреймворки Flask та Django. Django [6] хоч і надає вбудовану адміністративну панель, надає ORM та автентифікацію “з коробки”, проте його використання в даному випадку є недоцільним через надмірну складність, яка підходить для повноцінних вебзастосунків, але не для легких та високопродуктивних API. Flask [7] у свою чергу є мінімалістничним, гнучким та має велику кількість плагінів, проте не має повноцінної підтримки асинхронної моделі, що є важливим для оброки великої кількості одночасних запитів. FastAPI довзоляє досягти вищої продуктивності завдяки асинхронності. Крім того, даний фреймворк автоматично формує OpenAPI документацію, підтримує зручну інтеграцію з Pydantic для валідації запитів та більш простим за Djano з точки зору структурування коду для API.

Для роботи з нейромережами було обрано бібліотеки PyTorch та Transformers. Біблотека Transformers [8] від Hugging Face дозволяє завантажувати попередньо навчені моделі, зокрема трансформери, з Hugging Face Hub та здійснювати донавчання цих моделей. Крім PyTorch, було розглянуто також бібліотеку TensorFlow [9], є краще оптимізованою для продуктивності у великих системах, проте PyTorch [10] краще підтримується у спільності Hugging Face, є більше гнучким для реалізації донавчання моделей, тобто для fine-tuning та підтримується найновішою версією Python 3.13 на даний момент.

Серед IDE для Python найбільш популярними є PyCharm та Visual Studio Code [11], тому було розглянуто саме ці два IDE. Visual Studio Code [12] є легким редактором та містить велику кількість розширень та плагінів для різних мов розробки, зокрема Python. PyCharm [13] забезпечує розширену підтримку Python, має зручні засоби та надає вбудовані засоби для рефакторингу й аналізу коду та тестування. Хоча Visual Studio Code є більш швидким та легким редактором, було обрано PyCharm через його вбудовані засоби, зазначені вище, що є ключовим для зручної роботи зі складним Python проєктом із великою кількістю залежностей.

2.3 Конструювання програмного забезпечення

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 2.1.

Таблиця 2.1 – Опис утиліт, бібліотек та стороннього програмного забезпечення

№ п/п	Назва утиліти	Опис застосування
1	PyCharm	Головне середовище розробки API сервісу машинного навчання дипломного проєкту.
2	Google Colab	Хмарне середовище, яке надає доступ до потужних GPU/TPU для виконання Python-коду та навчання моделей.
3	FastAPI	Фреймворк для побудови високопродуктивного API сервісу машинного навчання.
4	PyTorch	Бібліотека для створення, навчання та донавчання нейромереж.
5	Transformers	Бібліотека для роботи з попередньо навченими моделями трансформерів.
6	SQLAlchemy	ORM бібліотека для роботи з реляційною базою даних.

7	Pandas	Бібліотека для обробки та аналізу табличних даних.

Продовження таблиці 2.1

8	Scikit-learn	Бібліотека для машинного навчання, яка надає інструменти для попередньої оброки даних та створення моделей машинного навчання.
9	Translators	Бібліотека для безкоштовного автоматичного перекладу текстів без використання API.
10	LangDetect	Бібліотека для автоматичного визначення мови тексту.
11	Beautiful Soup	Бібліотека для парсингу HTML та XML-документів.

2.3.1 Розробка методу пошуку релевантних вакансій

Оскільки у застосунку пошук релевантних вакансій здійснюється повністю автоматично та виключно на основі тексту резюме, то для такого пошуку потрібно певним чином порівняти текст резюме з текстами вакансій, тобто для резюме та кожної вакансії постає задача отримати оцінку семантичної подібності двома текстовими документами. З метою збільшення швидкості та точності пошуку було прийнято рішення визначити категорії посад в рамках IT сфери та відразу після завантаження резюме попередньо класифікувати його й призначати йому певну категорію серед заданих. При скрейпінгу вакансій також можна з відповідних сайтів отримати категорію для кожної вакансії, тому для вакансій достатньо лише задати мапінг категорій із визначених сайтом до визначених у застосунку. Таким чином, процес підбору релевантних вакансій можна оптимізувати шляхом звуження простору пошуку лише до вакансій тієї ж категорії, що й резюме.

Оскільки не було знайдено у відкритому доступі навчених моделей для класифікації резюме IT сектору по категоріям виключно на основі їхнього тексту, то було прийнято рішення використати попередньо навчену модель BERT, додати шар для багатокласової класифікації та виконати донавчання

моделі на розміченому текстовому корпусі резюме. Єдиним набором даних у відкритому доступі, що містить тексти резюме в IT сфері та відповідні категорії є Djinni Recruitment Dataset [14]. Після аналізу предметної області було визначено 12 категорій та виконано мапінг категорій датасету до визначених. Хоча даний датасет містить 210250 унікальних резюме [15], проте суттєва частина резюме звідти стосуються посад, які максимум опосередковано можуть відноситись до IT сфери, а серед тих, що є релевантними до IT сфери, не для всіх категорій наявні резюме в достатній кількості. Крім того, даний датасет обмежений лише резюме з Djinni, тобто це здебільшого резюме виключно українців. Для збільшення датасету в цілому та зменшення дисбалансу між категоріями було прийнято рішення додатково зібрати дані шляхом вебскрейпінгу нових резюме з Djinni та резюме із сайту пошуку роботи PostJob [16], який містить резюме з різних країн. Для кожної категорії був сформований відповідний пошуковий запит, який шукає резюме по наявності певних ключових слів саме у назві резюме.

Наступним було виконано очищено текстів резюме від зайвої контактної інформації, HTML та спеціальних символів і залишено лише резюме англійською мовою. У результаті було отримано датасет, що містить 191202 унікальних резюме. Отриманий датасет було розділено на навчальний та валідаційний набір у відношенні 90% до 10% відповідно з однаковою пропорцією класів в обох наборах. На навчальному датасеті було донавчено модель BERT для багатокласової класифікації за 2 епохи та оцінено на валідаційному. Таким чином, після завантаження резюме відбувається очищення тексту, його переклад англійською за потреби та автоматичне визначення категорії з використанням навченої моделі.

Для кожної вакансії та резюме після їхнього завантаження виконується обчислення векторного представлення на основі очищеного тексту за допомогою попередньо навченої моделі типу Sentence Transformer, а саме all-mpnet-base-v2 [17], яка повертає 768-вимірний вектор для заданого тексту, що враховує семантику, контекст та порядок слів. Таким чином, маючи вже

попередньо обчислені векторні представлення та визначені категорії, алгоритм пошуку релевантних вакансій спрощується до обчислення косинусної подібності, тобто косинусу кута між вектором резюме та вектором кожної з вакансій відповідної категорії та сортування вакансій за спаданням значення цієї метрики. Косинусна подібність між двома векторами ω_1 та ω_2 обчислюється за наступною формулою:

$$cos = \frac{\omega_1 \omega_2}{\|\omega_1\| \cdot \|\omega_2\|} \quad (2.1)$$

Оскільки моделі Sentence Transformer повертають достатньо змістовні векторні представлення, то дана метрика подібності є достатньою для визначення схожості між текстовими документами та при цьому забезпечує швидкість пошуку [18].

2.3.2 Розробка методу надання рекомендацій по адаптації резюме до вакансії

Одним із основних шляхів до адаптації резюме під вакансію є додавання релевантних ключових слів вакансії в резюме. Ці ключові слова здебільшого повинні бути навичками, причому частіше технічними у випадку IT сфери, тобто певні технології. Таким чином, у рамках дипломного проєкту було розроблено метод для визначення ключових слів (навичок) у вакансії, які відсутні в резюме.

Для визначення навичок, зазначених у вакансії, було обрано модель архітектури BERT для задачі класифікації токенів, а саме модель ihk/skillner [19], яка донавчена на 4112 реченнях оголошень про роботу на основі моделі JobBERT [20], яка в свою чергу навчена на 3.2 мільйонах речень оголошень про роботу [21]. Влучність даної моделі на валідаційному наборі становить приблизно 56%, а повнота – приблизно 68% [19]. Це означає, що модель знаходить більшість навичок, наявних у тексті, проте серед токенів, які класифіковані як навички, часто наявні токени, які насправді не є навичками. Для покращення влучності за допомогою моделі KeyBERT [22] визначаються найбільш релевантні ключові слова (юніграми та біграми) у вакансії, а

результуючі навички визначаються як перетин множини навичок та множини отриманих ключових слів. Оскільки модель ihk/skillner класифікує саме токени, а не цілі слова, то даний підхід гарантує, що отримані навички будуть дійсними релевантними ключовими словами, наявними у тексті. Аналогічно за допомогою моделі KeyBERT визначаються найбільш релевантні ключові слова для резюме. Відсутні в резюме навички визначаються як різниця множини навичок вакансії та множини ключових слів у резюме. Варто зауважити, що для резюме немає потреби окремо визначати саме навички, адже модель ihk/skillner може пропустити певні навички, а кількість ключових слів є суттєво більшою, тому ймовірність пропустити певні навички є значно меншою.

Обмеженням моделей архітектури BERT є те, що на вхід вони можуть приймати максимум 512 токенів, що часто є меншим за опис вакансії. Більше того, інколи у вакансіях навички знаходяться в кінці опису. Зважаючи на це, було модифіковано алгоритм як визначення навичок, так і ключових слів шляхом попереднього розбиття тексту на фрагменти розміром до 512 токенів, визначення ключових слів або навичок у кожному фрагменті та об’єднанні результатів у множину.

Крім того, з метою пришвидшення алгоритму ключові слова для резюме визначаються відразу при його завантаженні та зберігаються у базі даних. Отже, при кожному запиті на адаптацію ключові слова для резюме просто завантажуються із бази даних.

Оцінка подібності ключових слів резюме та вакансії – це косинусна подібність між векторним представленням ключових слів резюме та векторним представленням ключових слів вакансії. Векторне представлення обчислюється за допомогою моделі моделі all-mpnet-base-v2.

Висновки до розділу

У другому розділі пояснювальної записки було представлено за допомогою діаграми компонентів у нотації C4 та описано архітектуру

індивідуальної частини дипломного проєкту, наведено відповідні архітектурні рішення та обґрунтовано виборів засобів розробки. Крім того, було описано утиліти, стороннє програмне забезпечення й бібліотеки, які використовуються при розробці програмного забезпечення, та описано алгоритм пошуку релевантних вакансій для резюме й метод надання рекомендацій по адаптації резюме до вакансії.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Аналіз якості програмного забезпечення було здійснено з використанням сервісу Codacy [23], який підтримує мову програмування Python, на якій була здійснена розробка API сервісу машинного навчання та відповідних моделей машинного навчання, та дозволяє обрати мови програмування або розмітки, які треба аналізувати. Для виконання аналізу було виконано авторизацію через GitHub та надано доступ до відповідного репозиторію з дипломним проєктом. Оскільки даний репозиторій містить не тільки файли з кодом для API сервісу машинного навчання, але й інші файли, то було обрано лише файли з розширенням .py для аналізу (рисунок 3.1).

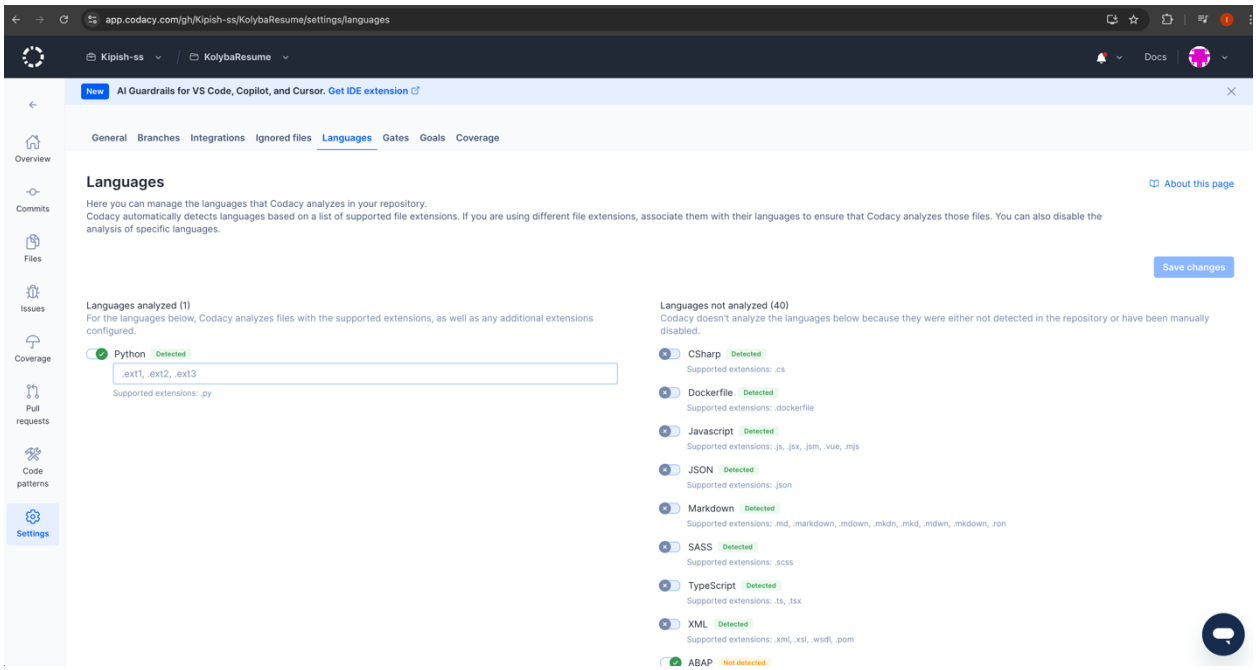


Рисунок 3.1 – Фільтрування по розширенню файлів для аналізу

При аналізі даний сервіс використовує наступні метрики: цикломатична складність та дублювання коду. Цикломатична складність враховує складність методів або функцій, які наявні у проєкті. Метрика дублювання враховує фрагменти коду, що дублюються в різних файлах репозиторію. У результаті аналізу якості програмного забезпечення було отримано оцінку A, яка є найвищою у даному сервісі (рисунок 3.2).

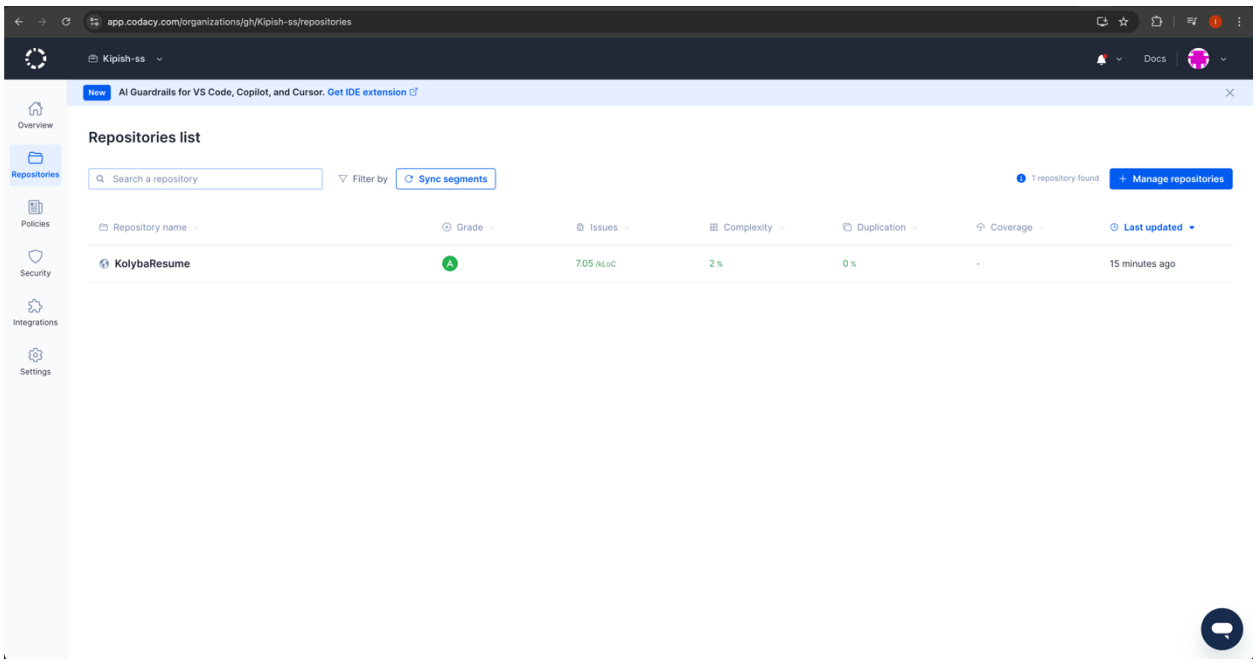


Рисунок 3.2 – Результат аналізу якості програмного забезпечення

Метрика складності становить 2%, що є абсолютно допустимим значенням. Codacy також дозволяє переглянути проблеми, виявлені у коді програмного забезпечення більш детально (рисунок 3.3).

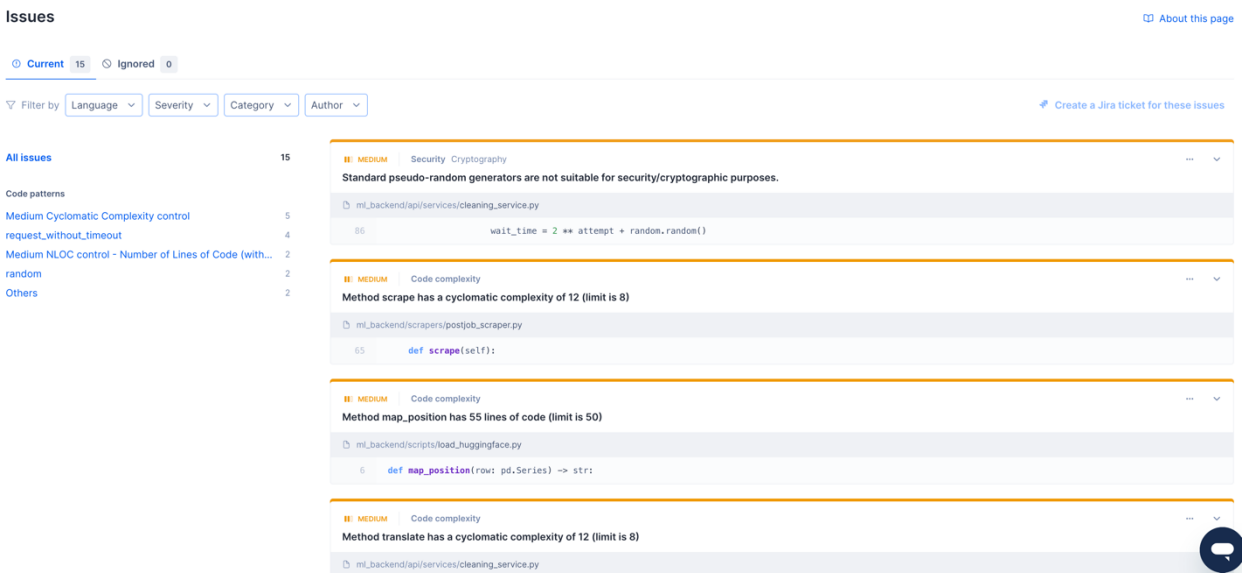


Рисунок 3.3 – Проблеми, виявлені у коді ПЗ

З рисунку 3.3 можна побачити, що сумарно виявлено лише 15 проблем, які не є серйозними та пов’язані здебільшого або з кількістю рядків коду на функцію чи метод, або з цикломатичною складністю функції чи методу, причому лише частина з них стосується коду API, решта – коду для збору та попередньої обробки даних для навчання моделей.

Отже, в результаті аналізу якості програмного забезпечення за допомогою сервісу Codacy було отримано найвищу загальну оцінку та близькі до ідеальних значення метрик.

Крім того, важливою частиною програмного забезпечення, яка відповідає за виконання ключових функціональних вимог є навчена модель для класифікації резюме, тому оцінка її якості є також важливою частиною оцінкою якості програмного забезпечення. Оскільки датасет містить 12 категорій та є незбалансованим (рисунок 3.4), то бажано використати одну метрику, яка враховує точність для кожної з 12 категорій, причому підходить для випадку незбалансованого датасету.

Category	
product/project manager	36310
qa engineer	26849
marketing	22763
sales manager	17977
hr	16150
data	15273
web	14284
ux	10622
business analyst	10003
mobile	9513
devops/cloud	6116
customer support	5342

Рисунок 3.4 – Розподіл класів у датасеті

Таким чином, зважаючи на ці умови, було обрано метрику $Macro\ F1$, яка однаково враховує усі класи незалежно від їхнього розміру та обчислюється за наступною формулою:

$$Macro\ F1 = \frac{1}{C}\sum_{i=1}^{C} F1_i \tag{3.1}$$

Тут C – це кількість класів, а $F1_i$ – значення метрики $F1$ для класу i, яка обчислюється наступним чином:

$$F1_i = \frac{2 \cdot Precision_i \cdot Recall_i}{Precision_i + Recall_i} \tag{3.2}$$

$$Precision_i = \frac{TP_i}{TP_i + FP_i} \tag{3.3}$$

$$Recall_i = \frac{TP_i}{TP_i + FN_i} \tag{3.4}$$

TP_i – це кількість коректно класифікованих резюме, які належать до класу i, FP_i – кількість резюме, які модель віднесла до класу i, проте вони належать до іншого класу, а FN_i – кількість резюме, які модель віднесла до іншого класу, проте вони належать до класу i. У результаті навчання моделі після другої епохи було отримано значення цієї метрики 0.9089 на валідаційному наборі (рисунок 3.5), що є досить високим показником для задачі класифікації текстів.

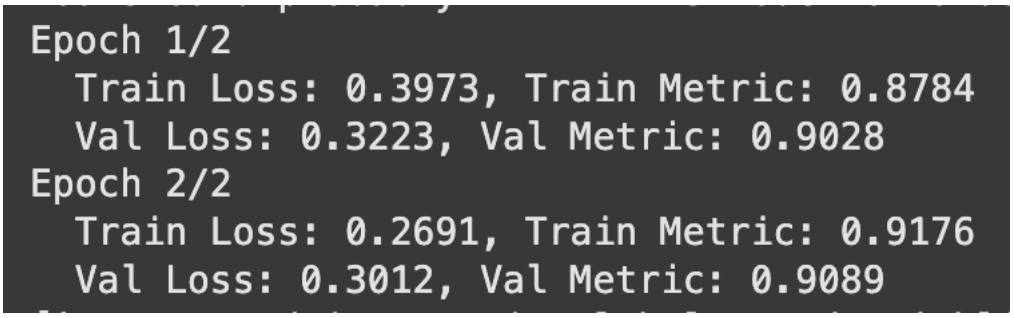


Рисунок 3.5 – Значення метрики на навчальному та валідаційному наборах

3.2 Опис процесів тестування

Тестування виконується згідно документу «Програма та методика тестування».

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 3.1 – 3.4.

Таблиця 3.1 – Тест 1.1 Запит на обробку резюме з існуючим у базі даних ID

Початковий стан системи	Запущений API сервісу машинного навчання, база даних доступна
Вхідні дані	Існуючий ID резюме у базі даних
Опис проведення тесту	Надсилається POST запит на ендпоїнт /resume з існуючим ID резюме у базі даних.
Очікуваний результат	Очищено та перекладено текст резюме, визначена категорія резюме, створено векторне представлення тексту резюме, визначено ключові слова у резюме, результати збережені у відповідних полях бази даних для резюме із заданим ID.
Фактичний результат	Збігається з очікуваним.

Таблиця 3.2 – Тест 1.2 Запит на обробку резюме з неіснуючим у базі даних ID

Початковий стан системи	Запущений API сервісу машинного навчання, база даних доступна
Вхідні дані	Неіснуючий у базі даних ID резюме
Опис проведення тесту	Надсилається PUT запит на ендпоїнт /resume з неіснуючим у базі даних ID резюме.
Очікуваний результат	Повертається статус код 404 з повідомленням про те, що резюме із заданим ID не знайдено.
Фактичний результат	Збігається з очікуваним.

Таблиця 3.3 – Тест 1.3 Запит на обробку непорожнього масиву ID вакансій

Початковий стан системи	Запущений API сервісу машинного навчання, база даних доступна
Вхідні дані	Непорожній масив ID вакансій
Опис проведення тесту	Надсилається POST запит на ендпоїнт /vacancies з масивом ID вакансій.
Очікуваний результат	Очищено та перекладено вакансії, створено векторні представлення для тексту кожної вакансії, результати збережено у базі даних у відповідних полях для вакансій із заданими ID. Повертається масив об’єктів з полями ID користувача, ID релевантної вакансії до його резюме та оцінка релевантності.
Фактичний результат	Збігається з очікуваним.

Таблиця 3.4 – Тест 1.4 Запит на обробку порожнього масиву ID вакансій

Початковий стан системи	Запущений API сервісу машинного навчання, база даних доступна
Вхідні дані	Порожній масив ID вакансій
Опис проведення тесту	Надсилається POST запит на ендпоїнт /vacancies з порожнім масивом ID вакансій.
Очікуваний результат	Повертається статус код 400 з повідомленням про те, що масив ID вакансій є порожнім.
Фактичний результат	Збігається з очікуваним.

Таблиця 3.5 – Тест 1.5 Запит на пошук релевантних вакансій по існуючому ID резюме у базі даних

Початковий стан системи	Запущений API сервісу машинного навчання, база даних доступна
Вхідні дані	Існуючий ID резюме у базі даних
Опис проведення тесту	Надсилається GET запит на ендпоїнт /vacancies/score з існуючим у базі даних ID резюме.
Очікуваний результат	Повертається масив об’єктів з полями ID релевантної вакансії та оцінка релевантності.
Фактичний результат	Збігається з очікуваним.

Таблиця 3.6 – Тест 1.6 Запит на пошук релевантних вакансій по неіснуючому ID резюме у базі даних

Початковий стан системи	Запущений API сервісу машинного навчання, база даних доступна
Вхідні дані	Неіснуючий у базі даних ID резюме
Опис проведення тесту	Надсилається GET запит на ендпоїнт /vacancies/score з неіснуючим у базі даних ID резюме.
Очікуваний результат	Повертається статус код 404 з повідомленням про те, що резюме із заданим ID не знайдено.
Фактичний результат	Збігається з очікуваним.

Таблиця 3.7 – Тест 1.7 Запит на надання рекомендацій по тексту вакансії для резюме з існуючим у базі даних ID

Початковий стан системи	Запущений API сервісу машинного навчання, база даних доступна
Вхідні дані	Текст вакансії, існуючий у базі даних ID резюме

Продовження таблиці 3.7

Опис проведення тесту	Надсилається POST запит на ендпоїнт /adaptation з існуючим у базі даних ID резюме та текстом вакансії.
Очікуваний результат	Повертається об’єкт з наступними полями: масив відсутніх ключових слів у вакансії та значення семантичної подібності ключових слів.
Фактичний результат	Збігається з очікуваним.

Таблиця 3.8 – Тест 1.8 Запит на надання рекомендацій по тексту вакансії для резюме з неіснуючим у базі даних ID

Початковий стан системи	Запущений API сервісу машинного навчання, база даних доступна
Вхідні дані	Текст вакансії, неіснуючий у базі даних ID резюме
Опис проведення тесту	Надсилається POST запит на ендпоїнт /adaptation з неіснуючим у базі даних ID резюме та текстом вакансії.
Очікуваний результат	Повертається статус код 404 з повідомленням про те, що резюме із заданим ID не знайдено.
Фактичний результат	Збігається з очікуваним.

Таблиця 3.9 – Тест 1.9 Запит на надання рекомендацій по порожньому тексту вакансії для резюме з існуючим у базі даних ID

Початковий стан системи	Запущений API сервісу машинного навчання, база даних доступна
Вхідні дані	Текст вакансії, існуючий у базі даних ID резюме
Опис проведення тесту	Надсилається POST запит на ендпоїнт /adaptation з існуючим у базі даних ID резюме та порожнім текстом вакансії.

Продовження таблиці 3.9

Очікуваний результат	Повертається статус код 404 з повідомленням про те, що текст вакансії не може бути порожнім.
Фактичний результат	Збігається з очікуваним.

Висновки до розділу

У третьому розділі пояснювальної записку було виконано аналіз якості програмного забезпечення за допомогою сервісу Codacy й моделі класифікації з використанням метрики $Macro\,F1$ для оцінки точності та проведено мануальне тестування API сервісу машинного навчання й описано відповідні тести.

ВИСНОВКИ

У рамках індивідуальної частини дипломного проєкту було розроблено нейромережу для класифікації резюме за категорією роботи, метод пошуку релевантних вакансій, метод надання рекомендацій для адаптації резюме та API сервісу машинного навчання, що реалізує зазначені методи та виконує обробку резюме й вакансій.

В першому розділі наведено опис функціональних вимог, матрицю трасування визначених функціональних вимог. В результаті визначено основні задачі, котрі потребують розробки.

В другому розділі наведено архітектуру API сервісу машинного навчання, обґрунтування основних засобів розробки та використаних бібліотек, наведено алгоритмічну складову основних розроблених методів: пошуку релевантних вакансій та адаптації резюме до вакансії.

В третьому розділі наведено результати аналізу якості та тестування API сервісу машинного навчання.

Мету дипломного проєкту - пришвидшення та автоматизація пошуку релевантних вакансій та надання користувачу рекомендацій по адаптації резюме до конкретної вакансії, було досягнуто за рахунок навчання власної нейромережі, використанні попередньо навчених нейромереж та збереженні у базі даних результатів попередніх обчислень для резюме та вакансії, що використовуються в алгоритмі пошуку та адаптації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Python documentation. URL: https://www.python.org/ (дата звернення 16.05.2025)

2) Hugging Face Hub documentation. URL: https://huggingface.co/docs/hub/en/index (дата звернення 16.05.2025)

3) Google Translate. URL: https://translate.google.com/ (дата звернення 16.05.2025)

4) Translators documentation. URL: https://pypi.org/project/translators/ (дата звернення 16.05.2025)

5) FastAPI documentation. URL: https://fastapi.tiangolo.com/ (дата звернення 16.05.2025)

6) Django documentation. URL: https://docs.djangoproject.com/en/5.2/ (дата звернення 16.05.2025)

7) Flask documentation. URL: https://flask.palletsprojects.com/en/stable/ (дата звернення 16.05.2025)

8) Transformers documentation. URL: https://pypi.org/project/transformers/ (дата звернення 16.05.2025)

9) TensorFlow. URL: https://www.tensorflow.org/ (дата звернення 16.05.2025)

10) PyTorch. URL: https://pytorch.org/ (дата звернення 16.05.2025)

11) Результати опитування Python розробників за 2023 рік. URL: https://lp.jetbrains.com/python-developers-survey-2023/ (дата звернення 16.05.2025)

12) Visual Studio Code. URL: https://code.visualstudio.com/ (дата звернення 16.05.2025)

13) PyCharm. URL: https://www.jetbrains.com/pycharm/ (дата звернення 16.05.2025)

14) Introducing the Djinni Recruitment Dataset: A Corpus of Anonymized CVs and Job Postings. URL: https://aclanthology.org/2024.unlp-1.2.pdf (дата звернення 16.05.2025)

15) Djinni Dataset (English CVs part). URL: https://huggingface.co/datasets/lang-uk/recruitment-dataset-candidate-profiles-english (дата звернення 16.05.2025)

16) PostJobFree. URL: https://www.postjobfree.com/ (дата звернення 16.05.2025)

17) All-Mpnet-Base-V2. URL: https://huggingface.co/sentence-transformers/all-mpnet-base-v2 (дата звернення 16.05.2025)

18) Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. URL: https://arxiv.org/pdf/1908.10084 (дата звернення 16.05.2025)

19) ihk/skillner. URL: https://huggingface.co/ihk/skillner (дата звернення 23.05.2025)

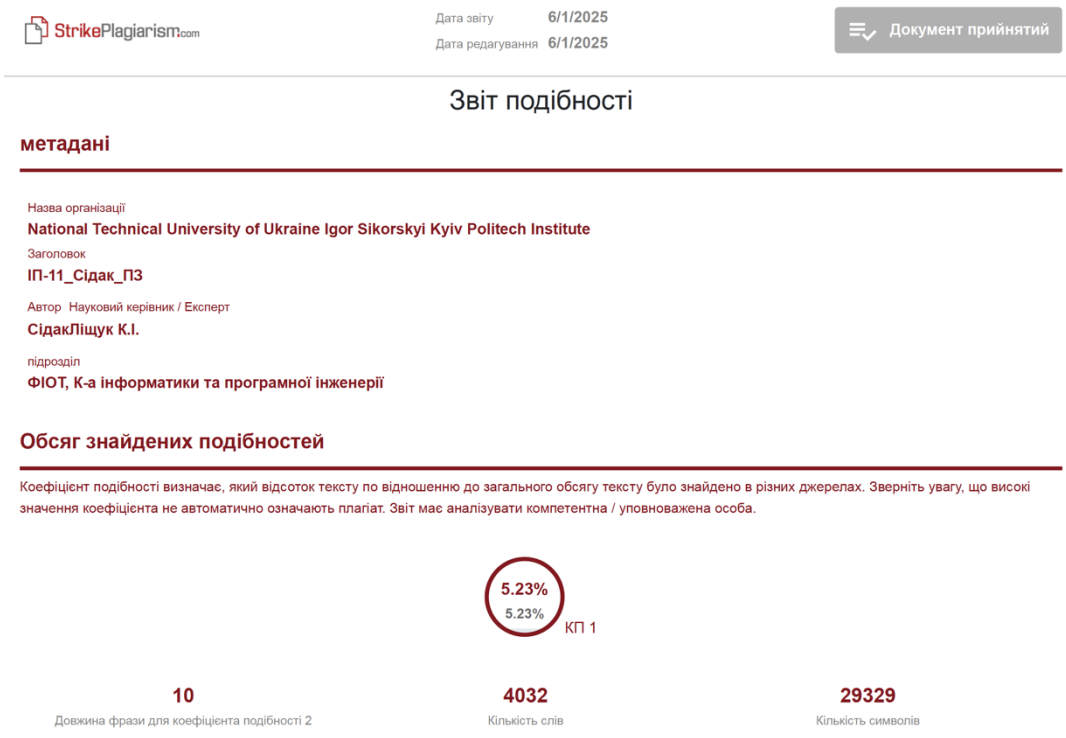
20) JobBERT. URL: https://huggingface.co/jjzha/jobbert-base-cased (дата звернення 23.05.2025)

21) SKILLSPAN: Hard and Soft Skill Extraction from English Job Postings. URL: https://arxiv.org/pdf/2204.12811 (дата звернення 23.05.2025)

22) KeyBERT documentation. URL: https://maartengr.github.io/KeyBERT/ (дата звернення 23.05.2025)

23) Codacy. URL: https://www.codacy.com/ (дата звернення 27.05.2025)

ДОДАТОК А ЗВІТ ПОДІБНОСТІ



StrikePlagiarism.com

Дата звіту 6/1/2025
Дата редагування 6/1/2025

Документ прийнятий

Звіт подібності

метадані

Назва організації
National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute
Заголовок
ІП-11_Сідак_ПЗ
Автор Науковий керівник / Експерт
СідакЛіщук К.І.
підрозділ
ФІОТ, К-а інформатики та програмної інженерії

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

5.23%
5.23%
КП 1

10
Довжина фрази для коефіцієнта подібності 2

4032
Кількість слів

29329
Кількість символів

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2025 р.

Вебзастосунок для автоматичного підбору вакансій на основі резюме та адаптації резюме за допомогою нейромереж для IT-галузі. API машинного навчання

Текст програми

КПІ.ІП-1124.045440.03.12

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Катерина ЛІЩУК

Нормоконтроль:

_____ Катерина ЛІЩУК

Виконавець:

_____ Кирил СІДАК

Київ – 2025

Посилання на репозиторій з повним текстом програмного коду

https://github.com/YuraRiabov/KolybaResume

Файл classification_trainer.py

Реалізація функціональної задачі розробки нейромережі для класифікації резюме

```
import torch
from torch.utils.data import DataLoader
from torch.nn import Module
from torch.optim import Optimizer
from typing import Callable
from torchmetrics import Metric
from transformers import get_linear_schedule_with_warmup
from tqdm import tqdm


class ClassificationTrainer:
    def __init__(
            self,
            train_dataloader: DataLoader,
            val_dataloader: DataLoader,
            model: Module,
            optimizer: Optimizer,
            loss_function: Callable[[torch.Tensor, torch.Tensor], torch.Tensor],
            metric: Metric,
            epochs: int,
            device: torch.device):
        self.train_dataloader = train_dataloader
        self.val_dataloader = val_dataloader
        self.model = model
        self.optimizer = optimizer
        self.loss_function = loss_function
        self.metric = metric
        self.epochs = epochs

        self.device = device
        self.model.to(self.device)
        self.metric.to(self.device)

        num_training_steps = self.epochs * len(self.train_dataloader)
```

```
    self.scheduler = get_linear_schedule_with_warmup(self.optimizer, num_warmup_steps=0,
                                                     num_training_steps=num_training_steps)
    self.history = {'train_loss': [], 'val_loss': [], 'train_metric': [], 'val_metric': []}

def _train_step(self) -> tuple[float, float]:
    self.model.train()
    loss = 0

    for batch in tqdm(self.train_dataloader, desc="Training", leave=False):
        token_ids = batch['input_ids'].to(self.device)
        attention_mask = batch['attention_mask'].to(self.device)
        labels = batch['labels'].to(self.device)

        self.optimizer.zero_grad()
        batch_loss, logits = self.model(input_ids=token_ids, attention_mask=attention_mask, labels=labels,
                                        return_dict=False)

        loss += batch_loss.item()
        batch_loss.backward()
        torch.nn.utils.clip_grad_norm_(self.model.parameters(), 1.0)
        self.optimizer.step()
        self.scheduler.step()

        preds = torch.argmax(logits, dim=1)
        self.metric.update(preds, labels)

    avg_loss = loss / len(self.train_dataloader)
    metric_result = self.metric.compute()

    return avg_loss, metric_result.item()

def _val_step(self) -> tuple[float, float]:
    self.model.eval()
    loss = 0

    with torch.no_grad():
        for batch in tqdm(self.val_dataloader, desc="Validating", leave=False):
            token_ids = batch['input_ids'].to(self.device)
            attention_mask = batch['attention_mask'].to(self.device)
            labels = batch['labels'].to(self.device)

            batch_loss, logits = self.model(input_ids=token_ids, attention_mask=attention_mask, labels=labels,
```

```
                                return_dict=False)

            loss += batch_loss.item()

            preds = torch.argmax(logits, dim=1)
            self.metric.update(preds, labels)

        avg_loss = loss / len(self.val_dataloader)
        metric_result = self.metric.compute()

        return avg_loss, metric_result.item()

    def fine_tune(self) -> None:
        for epoch in range(1, self.epochs + 1):
            self.metric.reset()
            train_loss, train_metric = self._train_step()

            self.metric.reset()
            val_loss, val_metric = self._val_step()

            self.history['train_loss'].append(train_loss)
            self.history['val_loss'].append(val_loss)
            self.history['train_metric'].append(train_metric)
            self.history['val_metric'].append(val_metric)

            print(f'Epoch {epoch}/{self.epochs}')
            print(f'  Train Loss: {train_loss:.4f}, Train Metric: {train_metric:.4f}')
            print(f'  Val Loss: {val_loss:.4f}, Val Metric: {val_metric:.4f}')
```

Файл train_bert_classifier.py

Реалізація функціональної задачі оцінки якості навченої моделі для класифікації резюме

```
from torch.optim import AdamW
from torch.utils.data import DataLoader
from transformers import BertTokenizer, BertForSequenceClassification
from ml_backend.resume_classifier import ClassificationTrainer, ResumeDataset
from sklearn.preprocessing import LabelEncoder
from torchmetrics import F1Score
from sklearn.model_selection import train_test_split
import pandas as pd
import torch
from joblib import dump
```

```
file_path = "/content/drive/MyDrive/preprocessed_resumes.parquet"

EPOCHS = 2
BATCH_SIZE = 16
LEARNING_RATE = 2e-5
MODEL_NAME = 'bert-base-uncased'
MAX_LENGTH = 512
VAL_SIZE = 0.1
RANDOM_STATE = 42



def load_data(file_path: str) -> tuple[list[str], list[int], LabelEncoder]:
    df = pd.read_parquet(file_path)
    resumes = df['Resume'].tolist()

    encoder = LabelEncoder()
    labels = encoder.fit_transform(df['Category']).tolist()

    return resumes, labels, encoder



resumes, labels, encoder = load_data(file_path)

train_resumes, val_resumes, train_labels, val_labels = train_test_split(
    resumes, labels, test_size=VAL_SIZE, stratify=labels, random_state=42)

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
train_dataset = ResumeDataset(train_resumes, train_labels, tokenizer, max_length=MAX_LENGTH)
val_dataset = ResumeDataset(val_resumes, val_labels, tokenizer, max_length=MAX_LENGTH)
train_dataloader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
val_dataloader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False)

num_classes = len(encoder.classes_)
model = BertForSequenceClassification.from_pretrained(
    'bert-base-uncased',
    num_labels=num_classes)

optimizer = AdamW(model.parameters(), lr=LEARNING_RATE)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
loss_function = torch.nn.CrossEntropyLoss()
metric = F1Score(task='multiclass', num_classes=num_classes, average='macro')
```

```
trainer = ClassificationTrainer(
    train_dataloader=train_dataloader,
    val_dataloader=val_dataloader,
    model=model,
    optimizer=optimizer,
    loss_function=loss_function,
    metric=metric,
    epochs=EPOCHS,
    device=device
)

trainer.fine_tune()

model.save_pretrained("/content/drive/MyDrive/fine_tuned_bert")
tokenizer.save_pretrained("/content/drive/MyDrive/fine_tuned_bert")
dump(encoder, "/content/drive/MyDrive/label_encoder.joblib")
```

Файл vacancy_service.py

Реалізація функціональної задачі розробки методу автоматизованого пошуку релевантних вакансій

```
from sqlalchemy import select
from sqlalchemy.orm import Session
from ml_backend.api.db.models import Vacancy, Resume
from ml_backend.api.models.schemas import VacancyScoreResponse, ResumeVacancyMatch
from ml_backend.api.services.cleaning_service import clean_text, translate
from ml_backend.api.services.model_service import get_embedding_model
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
from collections import defaultdict
import logging

logger = logging.getLogger(__name__)


def process_vacancies(db: Session, vacancy_ids: list[int], batch_size: int = 32) -> list[VacancyScoreResponse]:
    stmt = select(Vacancy).where(Vacancy.Id.in_(vacancy_ids))
    all_vacancies = db.execute(stmt).scalars().all()
    if not all_vacancies:
        return []

    model = get_embedding_model()
```

```
vacancy_vectors = {}
for i in range(0, len(all_vacancies), batch_size):
    batch_vacancies = all_vacancies[i:i + batch_size]
    batch_texts = []
    successful_vacancies = []

    for vacancy in batch_vacancies:
        try:
            text = translate(f"{vacancy.Title} {vacancy.Text}")
            cleaned = clean_text(text)
            vacancy.CleanedText = cleaned
            batch_texts.append(cleaned)
            successful_vacancies.append(vacancy)
        except Exception as e:
            logger.error(f'Error processing vacancy {vacancy.Id}: {str(e)}')

    if not successful_vacancies:
        continue

    try:
        encoded_batch = model.encode(batch_texts)
        for j, vacancy in enumerate(successful_vacancies):
            vector = encoded_batch[j]
            vacancy.Vector = vector.tobytes()
            vacancy_vectors[vacancy.Id] = vector

        db.commit()
        logger.info(f'Processed {len(successful_vacancies)} vacancies')
    except Exception as e:
        db.rollback()
        logger.error(f'Error encoding batch: {str(e)}')

categories = {v.Category for v in all_vacancies if v.Category is not None}
if not categories:
    return []

stmt = select(Resume).where(
    Resume.Category.in_(categories),
    Resume.Vector.is_not(None)
)
matching_resumes = db.execute(stmt).scalars().all()
```

```
    if not matching_resumes:
        return []

    resumes_by_category = defaultdict(list)
    resume_vectors = {}

    for resume in matching_resumes:
        resumes_by_category[resume.Category].append(resume)
        resume_vectors[resume.Id] = np.frombuffer(resume.Vector, dtype=np.float32)

    results = []
    for vacancy in all_vacancies:
        vacancy_category = vacancy.Category
        if vacancy_category not in resumes_by_category:
            continue

        vacancy_vector = vacancy_vectors[vacancy.Id]
        for resume in resumes_by_category[vacancy_category]:
            resume_vector = resume_vectors[resume.Id]
            similarity = model.similarity(vacancy_vector, resume_vector)[0][0]
            score = int(similarity * 100)

            results.append(VacancyScoreResponse(
                user_id=resume.UserId,
                vacancy_id=vacancy.Id,
                score=score
            ))

    return results


def get_matches_for_resume(db: Session, resume: Resume) -> list[ResumeVacancyMatch]:
    if not resume.Vector or not resume.Category:
        return []

    stmt = select(Vacancy).where(
        Vacancy.Category == resume.Category,
        Vacancy.Vector.is_not(None)
    )
    result = db.execute(stmt)
    matching_vacancies = result.scalars().all()
```

```
    if not matching_vacancies:
        return []

    resume_vector = np.frombuffer(resume.Vector, dtype=np.float32)

    results = []
    for vacancy in matching_vacancies:
        vacancy_vector = np.frombuffer(vacancy.Vector, dtype=np.float32)

        similarity = cosine_similarity(
            resume_vector.reshape(1, -1),
            vacancy_vector.reshape(1, -1)
        )[0][0]

        score = int(similarity * 100)

        results.append(
            ResumeVacancyMatch(
                vacancy_id=vacancy.Id,
                score=score
            )
        )
    results.sort(key=lambda x: x.score, reverse=True)

    return results[:200]
```

Файл adaptation_service.py

Реалізація функціональної задачі розробки методу для адаптації резюме

```
import json
from ml_backend.api.services.keyword_service import extract_keywords
from ml_backend.api.services.model_service import get_embedding_model
from ml_backend.api.db.models import Resume
from ml_backend.api.models.schemas import AdaptationResponse
from ml_backend.api.services.cleaning_service import clean_text, translate



def get_keywords_score(resume: Resume, vacancy_text: str, clean: bool) -> AdaptationResponse:
    if clean:
        vacancy_text = translate(vacancy_text)
        vacancy_text = clean_text(vacancy_text)

    resume_keywords = set(json.loads(resume.Keywords))
    vacancy_keywords, vacancy_skills = extract_keywords(vacancy_text, extract_skills=True)
```

```
    missing_keywords = list(vacancy_skills - resume_keywords)
    score = keyword_similarity(resume_keywords, vacancy_keywords)
    return AdaptationResponse(score=score, missing_keywords=missing_keywords)


def keyword_similarity(resume_keywords: set[str], vacancy_keywords: set[str]) -> int:
    resume_key_text = " ".join(resume_keywords)
    vacancy_key_text = " ".join(vacancy_keywords)
    model = get_embedding_model()
    resume_embedding = model.encode(resume_key_text)
    vacancy_embedding = model.encode(vacancy_key_text)

    similarity = model.similarity(resume_embedding, vacancy_embedding)[0][0]

    return int(similarity * 100)
```

Файл keyword_service.py

Реалізація функціональної задачі розробки методу для адаптації резюме

```
from transformers import Pipeline, BertTokenizer
from ml_backend.api.services.model_service import get_keybert_model, get_skills_model


def create_overlapping_chunks(text: str, tokenizer: BertTokenizer, max_length: int = 512, overlap: int = 50) -> list[
    str]:
    full_tokens = tokenizer.encode(text, add_special_tokens=False, truncation=False)
    total_tokens = len(full_tokens)
    effective_max_length = max_length - 2

    if total_tokens <= effective_max_length:
        return [text]

    chunks = []
    start_token = 0

    while start_token < total_tokens:
        end_token = min(start_token + effective_max_length, total_tokens)
        chunk_tokens = full_tokens[start_token:end_token]
        chunk_text = tokenizer.decode(chunk_tokens, skip_special_tokens=True)
        chunks.append(chunk_text)

        if end_token >= total_tokens:
            break
```

```
        start_token = end_token - overlap

    return chunks


def extract_chunk_skills(chunk_text: str, pipe: Pipeline) -> set[str]:
    results = pipe(chunk_text)
    skills = []
    current_skill_words = []
    prev_end = 0
    for result in results:
        entity = result['entity']
        word = result['word']
        start = result['start']
        end = result['end']

        if entity.startswith('B-SKILL') and result['score'] > 0.6:
            if current_skill_words:
                skill = ''.join(current_skill_words).replace('##', '')
                skill = skill.strip()
                if skill and len(skill) > 1:
                    skills.append(skill)
            current_skill_words = [word]

        elif entity.startswith('I-SKILL') and current_skill_words:
            word = word if start == prev_end else ' ' + word
            current_skill_words.append(word)

        else:
            if current_skill_words:
                skill = ''.join(current_skill_words).replace('##', '').strip()
                if skill and len(skill) > 1:
                    skills.append(skill)
                current_skill_words = []

        prev_end = end

    if current_skill_words:
        skill = ''.join(current_skill_words).replace('##', '').strip()
        if skill:
            skills.append(skill)
```

```
    skills = set([skill.lower() for skill in skills])

    return skills


def is_repeated_word(phrase: str) -> bool:
    words = phrase.lower().split()
    return len(words) > 1 and all(word == words[0] for word in words)


def extract_keywords(text: str, extract_skills: bool = False, top_n: int = 100, max_tokens: int = 510,
                     overlap_tokens: int = 50) -> set[str] | tuple[set[str], set[str]]:
    tokenizer, pipe = get_skills_model()
    chunks = create_overlapping_chunks(text, tokenizer, max_tokens, overlap_tokens)
    model = get_keybert_model()

    unigrams = model.extract_keywords(
        chunks,
        keyphrase_ngram_range=(1, 1),
        use_mmr=True,
        diversity=0.7,
        top_n=top_n
    )

    bigrams = model.extract_keywords(
        chunks,
        keyphrase_ngram_range=(2, 2),
        use_mmr=True,
        diversity=0.7,
        top_n=top_n
    )

    all_keywords = unigrams + bigrams
    if all_keywords and isinstance(all_keywords[0], tuple):
        all_keywords = [all_keywords]

    filtered_keywords = set()
    for keywords in all_keywords:
        for keyword, _ in keywords:
            if not is_repeated_word(keyword):
                filtered_keywords.add(keyword.lower())
```

```
    if extract_skills:
        skills = set()
        for chunk in chunks:
            chunk_skills = extract_chunk_skills(chunk, pipe)
            skills = skills.union(chunk_skills)

        skills = filtered_keywords.intersection(skills)
        return filtered_keywords, skills

    return filtered_keywords
```

Файл resume.py

Реалізація функціональної задачі проектування та розробки API машинного навчання

```
from fastapi import APIRouter, Depends, HTTPException, status
from sqlalchemy.orm import Session
from ml_backend.api.models.schemas import ResumeRequest
from ml_backend.api.db.base import get_db
from ml_backend.api.services.resume_service import preprocess_resume_text
from ml_backend.api.db.models import Resume

router = APIRouter()


@router.put("/resume", status_code=status.HTTP_200_OK)
async def process_resume(request: ResumeRequest, db: Session = Depends(get_db)):
    resume = db.get(Resume, request.resume_id)
    if not resume:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail=f"Resume with id {request.resume_id} not found"
        )
    preprocess_resume_text(db, resume)
    return {"status": "success", "message": "Resume vector updated successfully"}
```

Файл vacancies.py

Реалізація функціональної задачі проектування та розробки API машинного навчання

```
from fastapi import APIRouter, Depends, HTTPException, status
from sqlalchemy.orm import Session
from ml_backend.api.db.models import Resume
from ml_backend.api.models.schemas import VacanciesRequest, VacancyScoreResponse, ResumeVacancyMatch
```

```
from ml_backend.api.db.base import get_db
from ml_backend.api.services.vacancy_service import process_vacancies, get_matches_for_resume
import logging

logger = logging.getLogger(__name__)

router = APIRouter()



@router.post("/vacancies", response_model=list[VacancyScoreResponse])
async def process_new_vacancies(request: VacanciesRequest, db: Session = Depends(get_db)):
    if not request.vacancy_ids:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="No vacancy IDs provided"
        )
    results = process_vacancies(db, request.vacancy_ids)

    return results



@router.get("/vacancies/score/{resume_id}", response_model=list[ResumeVacancyMatch])
async def get_vacancy_matches(resume_id: int, db: Session = Depends(get_db)):
    resume = db.get(Resume, resume_id)
    if not resume:
        raise HTTPException(status.HTTP_404_NOT_FOUND, detail=f"Resume with {resume_id} not found")

    matches = get_matches_for_resume(db, resume)

    return matches
```

Файл adaptation.py

Реалізація функціональної задачі проектування та розробки API машинного навчання

```
from fastapi import APIRouter, Depends, HTTPException, status
from sqlalchemy.orm import Session
from ml_backend.api.models.schemas import AdaptationRequest, AdaptationResponse
from ml_backend.api.db.base import get_db
from ml_backend.api.services.adaptation_service import get_keywords_score
from ml_backend.api.db.models import Resume

router = APIRouter()
```

```
@router.post("/adaptation", response_model=AdaptationResponse)
async def get_recommendations(request: AdaptationRequest, db: Session = Depends(get_db)):
    if not request.vacancy_text.strip():
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail=f"Vacancy text cannot be empty"
        )
    resume = db.get(Resume, request.resume_id)
    if not resume:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail=f"Resume with id {request.resume_id} not found"
        )

    result = get_keywords_score(resume, request.vacancy_text, request.clean)

    return result
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

Вебзастосунок для автоматичного підбору вакансій на основі резюме та адаптації резюме за допомогою нейромереж для IT-галузі. API машинного навчання

Програма та методика тестування

КПІ.ІП-1124.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Катерина ЛІЩУК

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Кирил СІДАК

Київ – 2025

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є тестування API машинного навчання Kolyba Resume, а саме коректності обробки резюме та вакансій, пошуку релевантних вакансій і надання рекомендацій по адаптації резюме.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;
– знаходження проблем, помилок і недоліків з метою їх усунення.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- статичне тестування – перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;
- функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;
- мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;

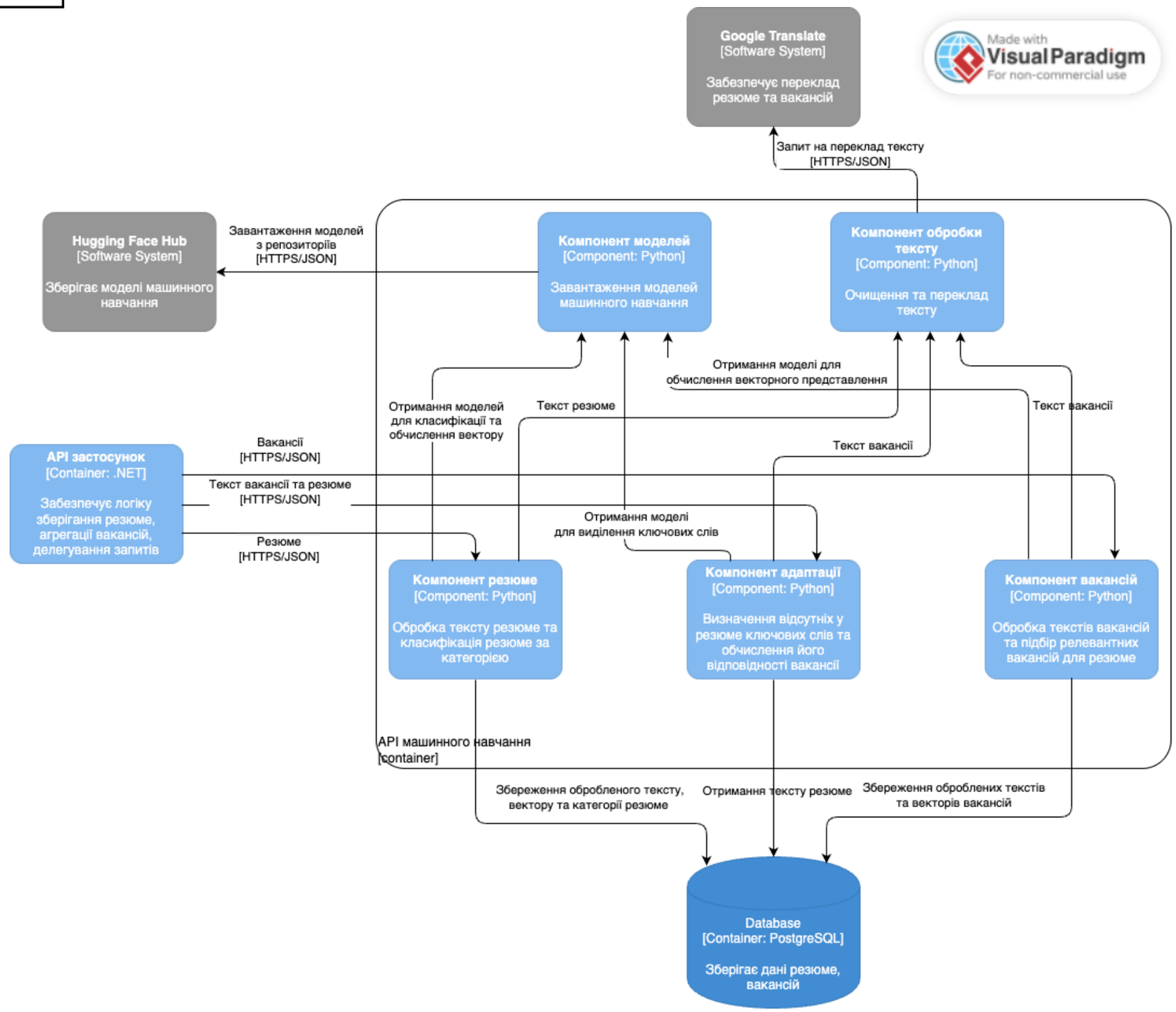
4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Під час проведення тестування будуть використовуватись допоміжний засіб Codacy для статичного аналізу коду.

Порядок проведення тестування буде наступним:

– статичний аналіз коду;
– тестування на виведення повідомлень про помилку, коли це необхідно;
– функціональне тестування на відповідність функціональним вимогам.

КПІ.ІП-1124.045440.06.99.ССМ



					КПІ.ІП-1124.045440.06.99.ССМ
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна компонентів програмного забезпечення
Розробив		Сідак К.І.			Літера: — Маса: — Масштаб: —
Перевірив		Ліщук К.І.			Аркуш 1 / Аркушів 1
Т. контр.					Вебзастосунок для автоматичного підбору вакансій на основі резюме та адаптації резюме за допомогою нейромереж для ІТ-галузі. API машинного навчання
Н. контр.		Ліщук К.І.			КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-11
Затвердив		Жаріков Е.В.			