Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему:

«Розробка вебзастосунку для автоматизованої генерації та аналізу якості резюме з використанням методів обробки природної мови»

Виконав: студент IV курсу
групи ЗПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Ковальський В.А.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Романюк О.В.
(прізвище та ініціали)

Рецензент: к.т.н., доц. каф. ЗІ Куперштейн Л.М.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ
д.т.н., проф. Романюк О. Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
«21» березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Ковальському Валентину Анатолійовичу

1. Тема роботи – «Розробка вебзастосунку для автоматизованої генерації та аналізу якості резюме з використанням методів обробки природної мови».

Керівник роботи: Романюк Оксана Володимирівна, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Термін подання студентом роботи: 2 червня 2025 року.

3. Тип програмного забезпечення – вебзастосунок; тип архітектури – клієнт-серверна; система керування базами даних – MongoDB; середовище розробки – Visual Studio Code; мова програмування – JavaScript, Node.js; метод автоматизованого збору інформації – вебскрапінг; методи обробки природної мови – велика мовна модель (LLM), n-грами; хмарні сервіси – Google Cloud; вхідні дані – текстові дані користувача; вихідні дані – файл резюме, стилізований шаблон для резюме, результат багатофакторного аналізу тексту резюме, результати автоматизованого збору вакансій, результат аналізу релевантності резюме вакансіям.

4. Зміст пояснювальної записки: вступ; аналіз технологій для аналізу тексту та генерації резюме та постановка задач дослідження; розробка методів та алгоритмів роботи програмного продукту; розробка програмного застосунку; тестування вебзастосунку; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу: титульний слайд – автор, науковий керівник бакалаврської кваліфікаційної роботи, тема; актуальність теми; мета, об'єкт та предмет дослідження; постановка задачі дослідження; наукова новизна отриманих результатів; аналогічні вебзастосунки; порівняльний аналіз аналогів; діаграма компонентів вебсистеми; схема даних вебзастосунку; загальний алгоритм роботи системи; метод та блок-схема алгоритму багатофакторного аналізу тексту; метод автоматизованого збору вакансій; алгоритм визначення

релевантності резюме вакансіям; використані технології; функціонал розробленої системи; створення та редагування резюме; сторінка аналізу резюме; апробація та публікація матеріалів БКР; фінальний слайд.

6. Консультанти розділів бакалаврської кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1 – 4	Романюк О. В., к.т.н., доцент кафедри ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз технологій автоматизованої генерації резюме та постановка задач дослідження	25.03.25 – 1.04.25	виконано
2	Розробка моделі, методів та алгоритмів роботи вебзастосунку	2.04.25 – 14.04.25	виконано
3	Розробка програмних модулів вебсистеми	15.04.25 – 30.04.25	виконано
4	Тестування вебзастосунку, розробка інструкції користувача та опис системних вимог	1.05.25 – 15.05.25	виконано
5	Оформлення матеріалів до захисту БКР	16.05.25 – 30.05.25	виконано

Студент _____ **Ковальський В. А.**
(підпис) (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи _____ **Романюк О. В.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.42

Ковальський В. А. Розробка вебзастосунку для автоматизованої генерації та аналізу якості резюме з використанням методів обробки природної мови: бакалаврська кваліфікаційна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – Інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 106 с.

На укр. мові. Бібліогр. : 31 назв; рис. : 30; табл. : 5.

В ході виконання бакалаврської кваліфікаційної роботи було проведено дослідження, що включає аналіз сучасного стану питання технологій для генерації резюме та шляхи покращення даної сфери.

В результаті дослідження було розроблено вебзастосунок для автоматизованої генерації резюме. Було розроблено метод багатофакторного аналізу тексту резюме та метод автоматизованого збору вакансій. Дані методи були реалізовані у вигляді фунціональних алгоритмів та інтегровано у розроблений програмний застосунок.

Вебзастосунок розроблено з використанням мови програмування Node.js, вебінтерфейс реалізовано з використанням стандартних засобів веброзробки HTML/CSS та фреймворку Vue.js. Розробка відбувалась в інтегрованому середовищі розробки Visual Studio Code. Вибір ключових засобів реалізації застосунку було здійснено на основі порівняльного аналізу.

Результати, що були отримані в ході дослідження, можна використати для покращення процесу пошуку роботи, зокрема етапу створення резюме.

Ключові слова: вебзастосунок, обробка природної мови, трансформерна модель, резюме, вакансія, вебскрапінг.

ABSTRACT

Kovalskyi V. A. Development of web application for automatized generation and text quality analysis of resume with natural language processing. Vinnytsia: VNTU, 2025. 106 p.

In Ukrainian language. Bibliographer: 31 titles; fig. : 30; tabl. : 5.

As part of the bachelor's qualification work, a study was conducted that included an analysis of the current state of technologies for resume generation and ways to improve this field.

As a result of the study, a web application for automated resume generation was developed. A method for multifactor analysis of resume text and a method for automated job vacancy collection were proposed. These methods were implemented as functional algorithms and integrated into the developed software application. The web application was developed using the Node.js programming language. The web interface was implemented using standard web development tools (HTML/CSS) and the Vue.js framework. Development was carried out in the integrated development environment Visual Studio Code.The choice of the main implementation tools was based on comparative analysis.

The results obtained in the course of the research can be used to improve the job search process, in particular the stage of resume creation.

Keywords: web application, natural language processing, transformer model, resume, job vacancy, web scraping.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Використання сучасних інформаційних технологій є невід'ємною складовою процесу пошуку роботи. Створення резюме є важливим етапом даного процесу, а його автоматизація надає кандидату чимало переваг.

Сучасні застосунки для побудови резюме надають зручний інтерфейс для заповнення інформації, що дозволяє зменшити кількість роботи в текстових редакторах та зменшити час на створення. Такі застосунки надають готові шаблони з різними стилями, що дозволяє зекономити час на стилізацію власного резюме і обрати один з поширених дизайнів.

Крім того, застосунки для побудови резюме надають інструменти для оперування декількома екземплярами, що може бути корисним, у випадку, якщо кандидат претендує на декілька вакансій. Просунуті застосунки для побудови резюме також дозволяють використовувати штучний інтелект для покращення тексту резюме. Таким чином, автоматизація процесу створення резюме підвищує ефективність пошуку роботи.

Проте часом креативний дизайн і наповненість резюме відходять на другий план. Поширеним є явище, коли роботодавці використовують програми для автоматизованого аналізу вмісту резюме для попереднього відбору кандидатів. В такому випадку визначним показником якості резюме є змістовна наповненість тексту, відсутність друкарських та стилістичних помилок, наявність ключових слів та релевантність посаді, на яку претендує кандидат. В процесі створення резюме необхідно завжди орієнтуватись на актуальні вакансії, що можна виконувати збираючи інформацію про вакансії на певну посаду з популярних сайтів для пошуку роботи [1].

Саме тому є актуальною розробка вебзастосунку для автоматизованої генерації та аналізу якості резюме з використанням методів обробки природної мови (NLP — Natural Language Processing).

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно з планом виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення якості резюме під час його автоматизованої генерації у вебзастосунку за рахунок використання методів обробки природної мови, що дозволить кандидатам, що шукають роботу підвищити шанси бути поміченими роботодавцями та знайти роботу.

В ході виконання бакалаврської кваліфікаційної роботи необхідно виконати такі **завдання:**

- проаналізувати сучасний стан питання технологій для генерації резюме, аналізу тексту та аналогічних програмних застосунків;
- розробити модель та загальний алгоритм роботи вебзастосунку;
- розробити метод багатофакторного аналізу тексту резюме;
- розробити метод автоматизованого збору вакансій та алгоритм визначення релевантності резюме вакансіям;
- розробити програмні модулі вебзастосунку для генерації резюме;
- провести тестування розробленого вебзастосунку;
- розробити інструкцію користувача та описати системні вимоги до вебзастосунку.

Об’єктом дослідження є процес автоматизованої генерації та аналізу якості резюме з використанням методів обробки природної мови.

Предметом дослідження є методи і засоби реалізації вебзастосунку для автоматизованої генерації та аналізу якості резюме з використанням методів обробки природної мови.

Методи дослідження. У процесі дослідження використовувались наступні методи: метод порівняльного аналізу для оцінки функціональних характеристик аналогічних програмних застосунків та обґрунтування вибору засобів реалізації системи; методи моделювання із застосуванням UML-діаграм для опису моделі та загального алгоритму роботи системи; теорія алгоритмів для розробки та

опису алгоритмів роботи вебзастосунку; методи обробки природної мови для виконання багатофакторного аналізу тексту резюме та визначення рівня релевантності резюме вакансіям; методи збору та обробки даних з відкритих джерел для автоматизованого збору вакансій; метод ручного та приймального тестування для перевірки відповідності розробленого вебзастосунку встановленим вимогам; комп'ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Новизна отриманих результатів.

1. Запропоновано новий метод багатофакторного аналізу тексту резюме, особливістю якого є здійснення аналізу граматики, стилістики, змістовності, визначення ключових слів тексту резюме із застосуванням методів обробки природної мови, включно із застосуванням великих мовних моделей та статичних методів, що дозволило підвищити ймовірність успішного проходження відбору в ході перевірки системами автоматизованого аналізу резюме, що активно використовуються роботодавцями.

2. Удосконалено метод автоматизованого збору вакансій, який на відміну від відомих, ґрунтується на застосуванні методу вебскрапінгу за пошуковим запитом користувача, що дозволило підвищити рівень інформованості кандидата щодо поточного стану ринку праці та створити підґрунтя для подальшого аналізу інформації та прийняття раціональних рішень у процесі формування професійного профілю кандидата.

Практична цінність отриманих результатів полягає у тому, що на основі отриманих в бакалаврській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби для автоматизованої генерації та аналізу якості резюме з використанням методів обробки природної мови.

Особистий внесок здобувача. Усі результати, подані в бакалаврській кваліфікаційній роботі, були отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати: порівняльний аналіз аналогічних програмних засобів, таблиця порівняння функціональних

характеристик аналогічних застосунків [1]; аналіз трансформерних мовних моделей [2].

Апробація матеріалів бакалаврської кваліфікаційної роботи. Основні положення бакалаврської кваліфікаційної роботи доповідалися та обговорювалися на Всеукраїнських конференціях:

- LIV Всеукраїнській науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії (Вінниця, 2025);

- XXV Всеукраїнській науково-технічній конференції молодих вчених, аспірантів та студентів «Стан, досягнення і перспективи інформаційних систем і технологій» (Одеса, 2025).

Публікації. За темою дослідження опубліковано 2 наукові праці у збірниках матеріалів конференцій [1-2].

Структура та обсяг БКР. БКР Складається із вступу, чотирьох розділів, висновків, списку використаних джерел і додатків, у які винесено технічне завдання, протокол перевірки на наявність текстових запозичень, лістинг програмного забезпечення й ілюстративний матеріал до захисту роботи.

1 АНАЛІЗ ТЕХНОЛОГІЙ ДЛЯ АНАЛІЗУ ТЕКСТУ ТА ГЕНЕРАЦІЇ РЕЗЮМЕ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану технологій автоматизованої генерації резюме

Аналіз стану технологій генерації резюме та аналізу тексту з використанням методів обробки природної мови дозволить виявити недоліки існуючих підходів та знайти власний підхід, що дозволить вирішити проблеми існуючих рішень.

Більшість сучасних застосунків для побудови резюме ставлять за мету зменшити час користувача на розробку дизайну резюме та структурування змістових блоків за рахунок надання різноманітних за стилями та розміткою шаблонів. Гарний дизайн резюме є важливим, адже дозволяє виділитись. Проте зі збільшенням компаній, кількістю вакансій та кандидатів час на перегляд кожного резюме зменшується. Тому досить поширеним є рішення використання систем для автоматизованого аналізу резюме або ATS (Applicant Tracking System – система відслідковування кандидатів), що дозволяють роботодавцю отримати ключову інформацію про кожного кандидата або навіть здійснювати пошук кандидатів за параметрами [1].

За даними звіту про використання ATS у 2024 році, розміщеному сервісом Jobscan, 98.4% серед 500 найбільших компаній-роботодавців у США використовує ATS [3]. Даний показник демонструє важливість врахування ATS оптимізації резюме і приділення уваги стилістично правильній побудові резюме. А за даними опитування, проведеними Гарвардською школою бізнесу, близько 90% роботодавців використовують системи управління наймом з ATS для фільтрування чи ранжування кандидатів як потенційно «середніх» та «хороших» в залежності від вимог до конкретної вакансії . Даний підхід підвищує ефективність пошуку кандидата за необхідними навичками, проте закономірно призводить до ситуації, в якій більшість кандидатів не будуть розглянуті взагалі [4]. Адже серед нерозглянутих кандидатів можуть бути дійсно компетентні

спеціалісти, які можуть приділити недостаню увагу процесу створення власного резюме.

За даних умов при створенні резюме доцільним є аналіз саме текстової наповненості. Використання повноцінної генерації тексту нейромережами може поставити під сумнів компетентність кандидата, адже сучасні сервіси дозволяють визначати частку тексту, що створено за допомогою штучного інтелекту. В той же час аналіз тексту за допомогою методів обробки природної мови дозволить виявити помилки різного характеру в процесі створення резюме.

Сучасні платформи для пошуку роботи містять сотні вакансій від різноманітних компаній на велику кількість посад. Зазвичай в процесі пошуку роботи виконується ручний перегляд вакансій для актуалізації власних навичок в залежності від їх затребуваності на ринку праці та формування привабливого професійного профілю кандидата.

Використання автоматизованого пошуку та аналізу вакансій з популярних ресурсів для пошуку роботи дозволить кандидату звернути увагу на необхідні навички, досвід, освіту тощо. Наявність у вебзастосунку вбудованих інструментів для збору вакансій за посадою, яка цікавить кандидата, значно підвищить якість процесу створення резюме.

Отже, аналіз сучасного стану технологій для автоматизованої генерації резюме підтверджує важливість створення нового рішення із використанням інноваційних методів, таких як аналіз тексту з використанням методів обробки природної мови та автоматизований збір актуальних вакансій з ресурсів для пошуку роботи. Враховуючи специфіку сучасного стану процесу найму на роботу, що передбачає використання систем ATS для аналізу кандидатів, та необхідність актуалізації власних навичок відповідно до ринку праці, є доцільною розробка вебзастосунку для автоматизованої генерації та аналізу резюме з методами обробки природної мови.

1.2 Порівняльний аналіз вебзастосунків для генерації резюме

Для визначення актуальності розробки вебзастосунку для автоматизованої генерації та аналізу якості резюме з використанням методів обробки природної мови необхідно проаналізувати існуючі аналогічні програмні застосунки. Наразі існує велика кількість аналогічних програмних застосунків.

Для порівняння було обрано наступні вебсервіси, що визначаються своєю популярністю і зручністю використання серед подібних систем:

- FlowCV;
- Canva;
- CVMaker.

FlowCV – це онлайн-сервіс, що дозволяє будувати резюме з попередньо налаштованими секціями та можливістю зберігати та експортувати резюме у файловий формат (див. рисунок 1.1). Основна ідея даного вебзастосунку це просте і швидке заповнення резюме з обмеженою кастомізацією.

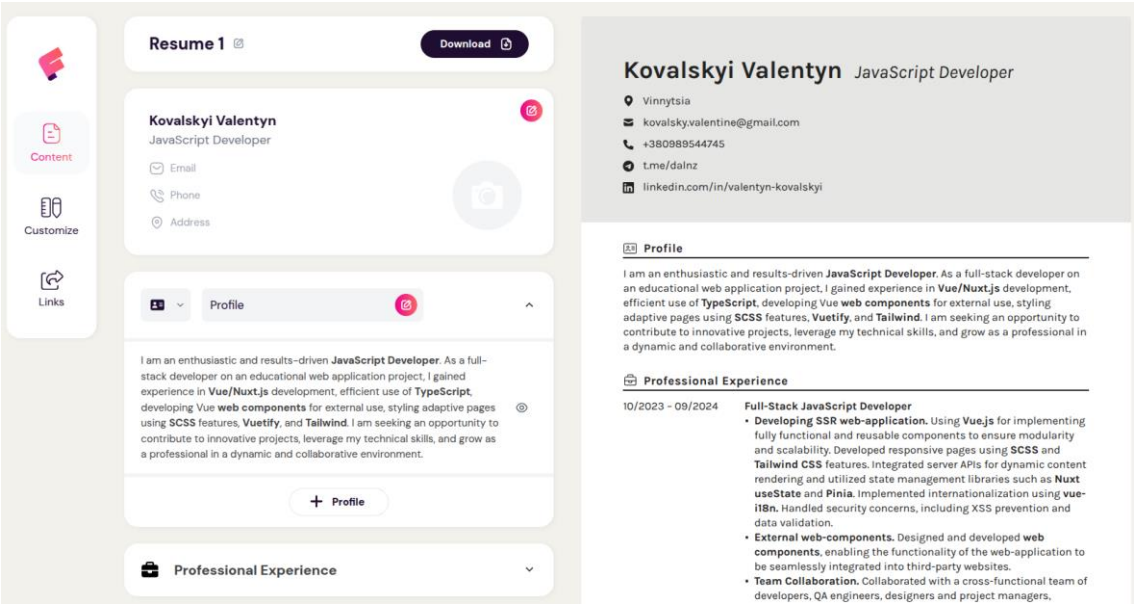

Рисунок 1.1 – Приклад роботи вебзастосунку FlowCV

Платна версія застосунку дозволяє використовувати інструменти ШІ для роботи з текстом та використовувати вбудований в сервіс відслідковування подання резюме на вакансії. Інструменти штучного інтелекту використовуються

для перевірки граматики, скорочення або подовження речень та повноцінної генерації тексту за запитом користувача [5].

Вебзастосунок Canva є універсальним дизайнерським інструментом, який містить стилізовані шаблони для резюме. Даний сервіс не є класичним застосунком для побудови резюме, проте є хорошим варіантом для виконання цього завдання, адже надає інструменти для гнучкої стилізації і повного редагування шаблонів. Застосунок дозволяє працювати над одним файлом декільком учасникам одночасно, що може бути корисним при консультації щодо оформлення резюме зі спеціалістом [6].

Недоліками застосунку є відсутність структурування шаблонів на окремі секції, адже кожний шаблон є по суті набором графічних елементів, які можуть вільно стилізуватись, та відсутність можливості динамічно виконувати зміну шаблону. Приклад роботи вебзастосунку Canva зображено на рисунку 1.2.

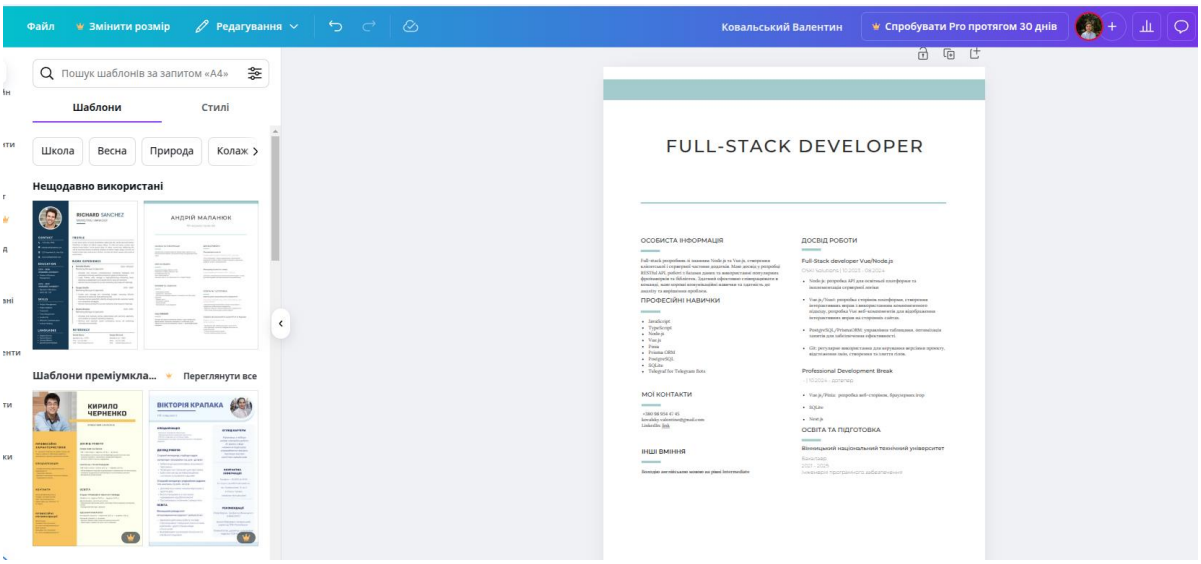

Рисунок 1.2 – Приклад роботи вебзастосунку Canva

CVMaker – це мінімалістичний вебсервіс для створення резюме [7]. Даний застосунок націлений на швидку розробку резюме на основі одного з популярних шаблонів з мінімальними налаштуваннями (див. рисунок 1.3). До переваг застосунку можна віднести швидкість заповнення резюме та наявність спеціалізованих секцій під специфічні професії. Застосунок передбачає

покрокове заповнення резюме за секціями, серед яких особисті дані, досвід, освіта, навички тощо. Також застосунок дозволяє додавати власні секції.

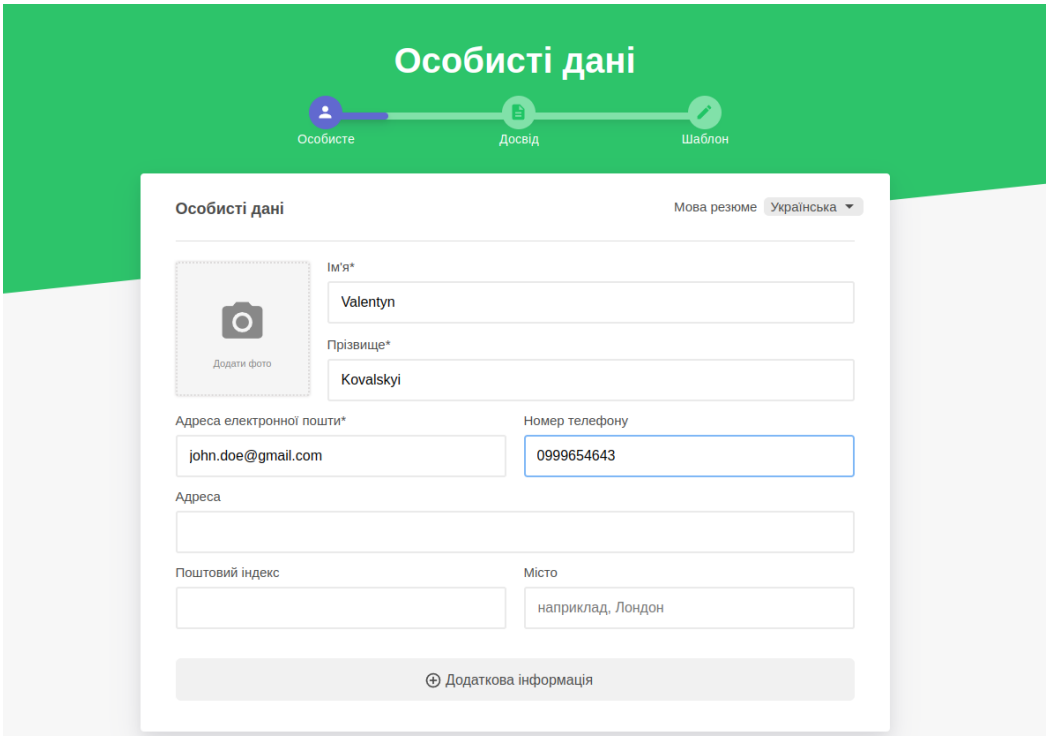

Рисунок 1.3 – Приклад роботи вебзастосунку CVMaker

До недоліків даного вебзастосунку можна віднести відсутність складних налаштувань для зміни структури резюме шаблонів та відсутність аналізу наповнення резюме.

Також вебзастосунок не дозволяє створення облікового запису та використання функцій у безкоштовному варіанті, що могло б бути корисним для демонстраційних цілей. Такий підхід може бути відштовхуючим для звичайних користувачів, які ймовірно спробують знайти інший безкоштовний аналогічний застосунок.

За результатами порівняльного аналізу аналогічних програмних застосунків з вебзастосунком, що буде розроблятись, було складено таблицю, що демонструє відмінність у функціональних можливостях розглянутих застосунків та власною розробкою (див. таблицю 1.1).

Таблиця 1.1 – Порівняльна характеристика аналогів

Функціональні можливості	FlowCV	Canva	CVMaker	Власна розробка
Гнучка стилізація елементів	0	1	1	0
Спільна робота над одним резюме	0	1	0	0
Використання штучного інтелекту для покращення вмісту	1	0	0	1
Динамічна зміна шаблонів	1	0	1	1
Аналіз змістовності тексту резюме	1	0	0	1
Аналіз релевантності резюме актуальним вакансіям	0	0	0	1
Сумарний коефіцієнт	3/6	2/6	2/6	4/6

За результатами порівняння функціональних можливостей власної розробки та аналогічних програмних застосунків було зроблено висновок про доцільність розробки вебзастосунку для автоматизованої генерації та аналізу резюме з використанням методів обробки природної мови. Розроблюваний вебзастосунок дозволить усунути недоліки існуючих підходів та надати користувачу додаткові функціональні можливості.

Власна розробка містить ряд недоліків, такі як, недостатньо гнучка стилізація шаблонів та відсутність спільної роботи над резюме кількома користувачами. Проте розроблюваний вебзастосунок містить ряд переваг, які виділяють застосунок серед інших. Застосунок, що буде розроблятись дозволить користувачу створювати резюме за одним з шаблонів і використовувати методи обробки природної мови для аналізу текстової наповненості резюме та визначати релевантність резюме вакансіям на посаду, на яку претендує користувач.

Отже, результати порівняльного аналізу підтверджують доцільність розробки власного вебзастосунку для автоматизованої генерації резюме.

1.3 Аналіз методів розв’язання задачі

Розробка програмних модулів вебзастосунку для автоматизованої генерації та аналізу якості резюме з використанням методів обробки природної мови передбачає комплексний підхід щодо використання методів аналізу тексту та отримання інформації про вакансії. Існує ряд методів обробки природної мови, які варто розглянути.

Статичні методи аналізу мають високу швидкодію. До таких методів можна віднести:

- TF-IDF;
- використання n-грам;
- векторне представлення слів.

TF-IDF (Term Frequency – Inverse Document Frequency – частота термінів – обернена частота документів) – статичний метод визначення важливості кожного слова в конкретному документі та відносно всієї колекції документів. Зазвичай використовується для визначення ключових слів у масиві текстів та у пошукових системах [8].

Використання n-грам – статичний метод, що передбачає виокремлення та групування суміжних сутностей у тексті для подальшої обробки. Зазвичай це слова, проте в залежності від ступеня деталізації це можуть бути також окремі склади або символи. Значення n визначає порядок n-грам. Поширеним є використання значення n = 2, що відповідає біграмам. Даний статичний метод є цінним, адже дозволяє зберігати контекст та семантику слів. Зазвичай використовується при семантичному порівнянні текстів, в процесі тренування мовних моделей, при виконанні завдань передбачення тексту або розпізнавання мовлення тощо [9].

Векторне представлення слів передбачає перетворення слів у вектори. Близькі за значенням слова мають схожі векторні координати. Даний метод використовується при класифікації текстів та у рекомендаційних системах, наприклад в онлайн магазинах [10].

Статичні методи мають високу швидкодію в порівнянні зі складнішими методами і не вимагають потужного обладнання для обробки інформації та добре працюють для класифікації текстів. Проте дані методи не дуже добре враховують контекст, стилістику тексту, тому їх використання може бути недостатнім для багатофакторного аналізу тексту в даній предметній області.

Останнім часом провідну роль у галузі комплексного аналізу тексту відіграють трансформерні моделі. Дані моделі базуються на використанні трансформерів – нейронних мереж, що використовують механізм уваги і дозволяють моделі концентрувати увагу на найбільш важливих частинах вхідного тексту. Даний механізм дозволяє враховувати контекст слів і обробляти навіть довгі тексти. До найбільш поширених трансформерних моделей відносять GPT та BERT [2].

GPT (Generative Pre-trained Transformer – генеративний попередньо тренований трансформер) – тип генеративної мовної моделі, що використовує за основу архітектуру трансформера з однобічним аналізом [11]. Таким чином аналіз вхідного тексту виконується лише зліва направо. Такі моделі перед використанням проходять навчання на великій кількості даних, в результаті чого модель отримує здатність відтворювати людську мову. За рахунок цього дана модель відома своєю здатністю створювати, перефразовувати текст, продовжувати думку, перекладати текст, виправляти граматичні помилки і виконувати інші завдання за правильно сформульованим запитом користувача. Дана мовна модель є гнучкою для виконання завдань пов’язаних з обробкою тексту, аналізу семантичної подібності текстів, визначення ключових слів тощо.

Одним з найбільш просунутих методів обробки природної мови можна вважати BERT. BERT (Bidirectional Encoders Representations from Transformers – Двонаправлені кодовані представлення з трансформерів) – трансформерна мовна модель, що була розроблена компанією Google в 2018 році. Принцип роботи моделі полягає у двобічному аналізі контексту слова, що означає враховування попереднього та наступного слова. За рахунок двобічного аналізу модель враховує більше контексту. BERT та її варіації використовуються для

класифікації текстів, виявлення сутностей, ключових слів, визначення схожості текстів або відповідей на запитання [12]. Для кожного типу завдань дана модель повинна бути додатково донавчена, що робить її менш гнучкою. Також мовні моделі даного типу є негенеративними і не можуть генерувати текст, що може обмежити потенційні функціональні можливості.

Розглянувши відомі підходи аналізу тексту для реалізації функцій вебзастосунку для автоматизованої генерації та аналізу резюме з використанням методів обробки природної мови, було вирішено використовувати для аналізу тексту модель типу GPT або іншу сумісну з GPT за функціональними характеристиками трансформерну мовну модель. Попереднє тренування подібних моделей на великих обсягах даних дозволить використовувати її для декількох різних за суттю завдань одночасно з можливістю масштабування без додаткового тренування. В той час як моделі типу BERT необхідно провести донавчання для виконання нового типу завдань. Також генеративні можливості GPT можуть бути використанні в розроблюваних модулях для коригування тексту, виправлення граматичних помилок тощо. Також суттєвою перевагою використання GPT є наявність публічних API для різних моделей. Моделі типу BERT в свою чергу частіше використовуються локально і вимагають потужного устаткування для ефективної роботи без затримок [2]. Також вирішено допустити застосування статичних методів обробки природної мови для попереднього аналізу тексту або виконання окремих завдань в ході аналізу.

Іншим аспектом розв’язання поставлених завдань є автоматизований збір даних про вакансії з сайтів для пошуку роботи.

Використання офіційного API сайтів для пошуку роботи є найстабільнішим способом отримання інформації з даних ресурсів. Даний спосіб дозволяє отримати структуровані дані, які ресурс надає добровільно або за певну плату. Проте кількість даних для отримання може бути обмеженою власниками ресурсу. Крім того, можуть виникати обмеження за кількістю звернень до API.

Деякі вебсайти для пошуку роботи дозволяють виконувати експорт даних у форматі таблиць CSV. Таким чином можна отримувати структурований набір

інформації у вигляді файлу, після чого програмно опрацьовувати дані. Недоліками даного способу отримання даних є обмеження експортованих даних, що може бути встановлене самим вебресурсом, та можливість відсутності функції експорту.

Популярним підходом автоматизованого збору даних з мережі Інтернет в сфері програмного забезпечення є вебскрапінг. Вебскрапінг (Web scraping – веб збирання) – процес автоматизованого збору контенту зі звичайних вебсторінок мережі Інтернет у структуровані дані [13]. Даний підхід дозволяє отримати доступ до даних, які можна отримати просто перейшовши на вебсайт для пошуку роботи.

Розглянувши можливі методи збору даних для реалізації функціональності збору актуальних вакансій у розроблюваному ПЗ було вирішено обрати вебскрапінг. Перевагою даного методу є можливість гнучко налаштувати процес збору даних і структурувати їх за власними потребами. Даний спосіб доцільний також у випадку, якщо ресурс не має власного API або функції експорту даних.

Отже, за результатами аналізу методів розв’язання завдань для розробки вебзастосунку для генерації резюме було вирішено залучити підхід з використанням моделі типу GPT або сумісних мовних моделей та застосовувати статичні методи обробки тексту. Для автоматизованого збору даних буде використано метод вебскрапінгу.

1.4 Постановка задачі дослідження

Проаналізувавши переваги та недоліки відомих аналогічних програмних засобів для генерації резюме, було визначено ряд задач, які необхідно виконати для успішної розробки програмного застосунку:

- проаналізувати сучасний стан питання технологій для генерації резюме, аналізу тексту та аналогічних програмних застосунків;
- розробити модель та загальний алгоритм роботи вебзастосунку;
- розробити метод багатофакторного аналізу тексту резюме;

- розробити метод автоматизованого збору вакансій та алгоритм визначення релевантності резюме вакансіям;

- розробити програмні модулі вебзастосунку для генерації резюме;

- провести тестування розробленого вебзастосунку;

- розробити інструкцію користувача та описати системні вимоги до вебзастосунку.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У розділі було розглянуто стан сучасних технологій та підходів до реалізації застосунків для генерації резюме. Було визначено загальні недоліки таких застосунків та обґрунтовано доцільність розробки застосунку, який враховує необхідність аналізу текстової наповненості резюме та вбудований пошук та порівняння резюме з вакансіями.

Було розглянуто відомі аналогічні програмні застосунки для генерації резюме, а саме FlowCV, Canva та CVMaker. За результатами порівняльного аналізу функціональних можливостей було зроблено висновок про доцільність розробки власного застосунку для генерації резюме.

Також було здійснено попередній аналіз методів розв’язання задач дослідження. Було розглянуто методи обробки природної мови, та методи автоматизованого збору інформації в мережі Інтернет.

За результатами аналізу аналогічних застосунків та визначеними шляхами покращення існуючих рішень було сформульовано задачі які необхідно виконати для розробки вебзастосунку для автоматизованої генерації та аналізу якості резюме з використанням методів обробки природної мови.

2 РОЗРОБКА МЕТОДІВ ТА АЛГОРИТМІВ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

2.1 Аналіз вхідних та вихідних даних програмного застосунку

Для реалізації вебзастосунку для автоматизованої генерації та аналізу якості резюме з методами обробки природної мови необхідно визначити вхідні та вихідні дані. Як правило, джерелом вхідних даних програмного застосунку для генерації резюме є користувач, який взаємодіє з вебінтерфейсом.

В ході використання застосунку під час генерації резюме користувач вводить текстові дані, що програмно структуруються в об’єкт резюме, який розділений на логічні секції, такі як контактні дані, професійний досвід, освіта тощо. Дані структуруються в об’єкт для полегшеної подальшої обробки системою. Крім цього користувач забезпечує реєстраційні дані у вигляді електронної пошти та паролю, що використовуються для реєстрації та автентифікації користувача в системі.

В ході аналізу резюме користувач вводить назву посади, на яку він претендує для подальшої обробки системою. Також користувач може обрати бажаний сайт для пошуку роботи зі списку.

Вхідні дані зберігаються в базі даних та обробляються в процесі роботи вебзастосунку. Вищезазначених вхідних даних достатньо для виконання задач застосунку.

До вихідних даних в першу чергу варто віднести результат генерації резюме у вигляді PDF-документу або HTML-файлу. На головній сторінці авторизований користувач може бачити список створених ним резюме та попередній перегляд кожного екземпляру у вигляді зменшеного відображення файлу. В ході заповнення резюме користувач бачить попередній перегляд файлу у вебінтерфейсі, дані в якому змінюються динамічно.

Вебзастосунок також забезпечує користувача результатами аналізу резюме на сторінці аналізу. До результатів аналізу входить перелік визначених ключових

слів, граматичних помилок, стилістичних помилок, виділення малозмістовних речень та змістовних повторень.

У випадку, якщо користувач виконує пошук вакансій за посадою, користувач отримує відсоток схожості з кожною вакансією та значення середньої релевантності резюме у відсотках. Крім цього, користувач у вебінтерфейсі може переглянути посилання на вакансії, що система отримала в ході автоматизованого збору даних.

2.2 Розробка моделі програмного застосунку

Розробка моделі системи є важливим етапом досліджень, що передують безпосередньо процесу програмної реалізації. Моделлю системи є абстрактне уявлення про загальну структуру програмного застосунку, компоненти та взаємоз'вязки між ними. В процесі розробки моделі можна визначити важливі структурні елементи, підібрати найбільш ефективний тип архітектури програмного забезпечення та розглянути типи даних, з якими програмний застосунок буде взаємодіяти.

Для розробки вебзастосунку для генерації резюме було вирішено використати класичну клієнт-серверну архітектуру [14]. Дана архітектура передбачає взаємодію між двома логічними частинами, клієнтом та сервером. Клієнтом вебзастосунку зазвичай є вебсторінка, яку користувач відкрив у веббраузері. Клієнт передбачає обробку дій користувача, відображення інформації та є посередником між людиною та бізнес-логікою системи. Серверна частина відповідає за обробку та збереження даних та взаємодію з іншими зовнішніми компонентами, що беруть участь у формуванні бізнес-логіки, такими як бази даних, сервіси, інші застосунки тощо. В основі зв'язку між клієнтом та сервером лежить взаємодія через мережу за допомогою протоколів HTTP/HTTPS, сокетів, GraphQL тощо.

Для представлення архітектури вебзастосунку для генерації резюме було розроблено UML-діаграму компонентів системи (див. рисунок 2.1).

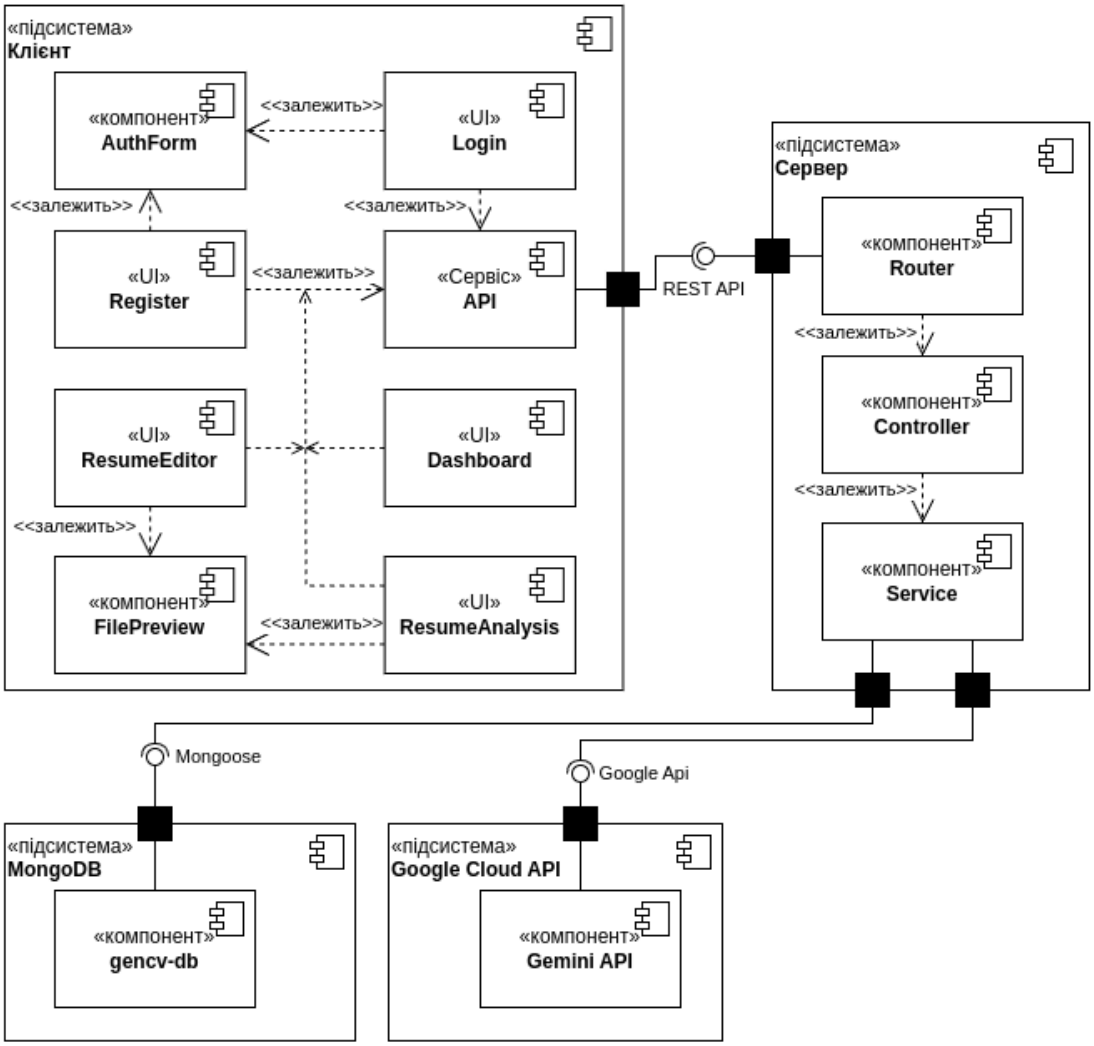

Рисунок 2.1 – Діаграма компонентів

Система розділена на декілька підсистем, що взаємодіють між собою за допомогою інтерфейсів. Підсистема клієнт містить компоненти UI, які є вебсторінками, на які користувач може перейти. Також на діаграмі відзначено залежності сторінок від важливих структурних компонентів, таких як AuthForm та FilePreview. Усі вебсторінки залежать від логічного компоненту API. Даний компонент відповідає за надсилання та обробку запитів до серверу та вимагає від серверу інтерфейс REST API.

Підсистему сервер представлено у вигляді узагальнюючих компонентів задля уникнення надмірної деталізації. Router – компонент, що описує маршрути, до яких може бути здійснено запит та формує інтерфейс REST API. Controller –

тип компонентів, що обробляють запити до серверу. Service – компоненти, які містять конкретну функціональність та формують бізнес-логіку застосунку.

Сервіси серверу вимагають специфічні інтерфейси для взаємодії з базою даних та сторонніми застосунками. В даному випадку сервер комунікує з базою даних MongoDB за допомогою бібліотеки Mongoose, та з хмарним сервісом Google Cloud за допомогою бібліотеки Google API. Підсистема MongoDB містить лише один компонент, базу даних вебзастосунку gencv-db. Підсистема Google Cloud містить компонент Gemini API, що дає доступ до трансформерних мовних моделей.

База даних є важливим елементом вебзастосунку для генерації резюме. Адже необхідно централізовано зберігати інформацію про користувачів системи, екземпляри їх резюме та шаблони для резюме. Для розроблюваного вебзастосунку буде використано документо-орієнтовану систему керування базами даних MongoDB.

Дана СКБД надає гнучкість структури даних, що може змінюватись в ході оновлень застосунку в майбутньому та підтримує природні вкладення об’єктів в документи, що зменшує результуючу кількість запитів. Такий нереляційний підхід організації даних у вигляді документів особливо ефективний, коли існує вкладеність об’єктів, наприклад секції у резюме або підсекції в секцій тощо. Гнучкість структури документів в межах бази даних також є доречною в умовах, де користувач має можливість створювати резюме з довільною структурою. MongoDB дозволяє легко інтегрувати базу даних із серверним застосунком Node.js та забезпечує зручну взаємодію з даними у JSON форматі, який використовується і при взаємодії між клієнтом та сервером [15].

Для розробки вебзастосунку для генерації резюме необхідно визначити схему типів даних, якими буде маніпулювати система. Для представлення логічної моделі даних системи і зв’язків між ними дані було представлено у вигляді UML діаграми класів. Схему бази даних застосунку у вигляді UML-діаграми класів зображено на рисунку 2.2.

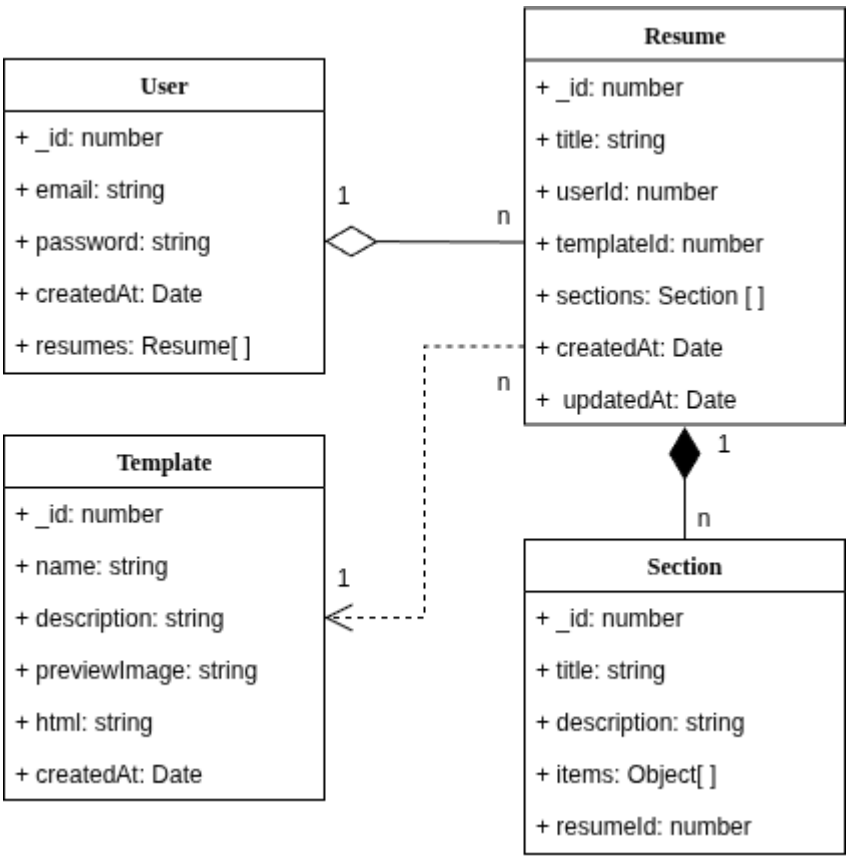

Рисунок 2.2 – Діаграма класів

Зображені на діаграмі класи демонструють рівень абстракції між документами в базі даних та об’єктами в програмному коді. Усі вказані в діаграмі поля присутні як в базі даних, так і в коді безпосередньо. Зв’язки між класами дозволяють уявити логічну структуру впорядкування даних.

Кожний користувач User містить інформацію, необхідну для автентифікації користувача та має зв’язок «один до багатьох» з класом Resume. Між класами встановлений зв’язок агрегації, який означає, що резюме є частиною користувача, проте дані сутності можуть мати різний час існування. Наприклад, якщо акаунт користувача буде видалено, архівний варіант резюме зберігатиметься в базі даних ще деякий час.

Кожне резюме містить декілька секцій, про що говорить зв’язок «один до багатьох» з класом Section. Між класами встановлено логічний зв’язок композиція, який повідомляє про неможливість існування Section поза межами резюме, адже це не має сенсу. Також Resume має залежність від класу Template,

тому що резюме не містить шаблон для резюме безпосередньо, а лише використовує його при відображенні та генерації файлу резюме. Один шаблон може використовуватись багатьма резюме, про що свідчить зв’язок «один до багатьох».

Отже, було спроектовано модель та схему бази даних вебзастосунку та описано їх у вигляді UML-діаграм компонентів та класів відповідно.

2.3 Розробка загального алгоритму роботи програмного застосунку

Для загального уявлення про функціональні можливості вебзастосунку та процес взаємодії користувача з системою доцільно розробити загальний алгоритм роботи застосунку. Загальний алгоритм роботи програмного застосунку описує поведінкову складову роботи, а саме послідновність дій, що відбуваються при взаємодії користувача із системою та стани, у яких може перебувати застосунок в процесі роботи.

Розробка та опис загального алгоритму роботи дозволяє виявити додаткові важливі з точки зору UX (User Experience – досвід користувача) функціональні та нефункціональні вимоги до системи, вимоги до засобів та методів розробки, особливостей проектування та реалізації тощо. Одним з методів опису даного алгоритму є UML-діаграма діяльності.

Діаграма діяльності UML важливий тип так званих поведінкових діаграм, що описують динамічні аспекти системи. Діаграма діяльності описує як певні дії координуються для забезпечення певної функціональної поведінки. Поширеним є використання даного типу діаграм для моделювання та опису того, як набір варіантів використання формує фукціональні можливості програмного застосунку [16].

Для опису загального алгоритму роботи вебзастосунку було розроблено UML-діаграму діяльності (див. рисунок 2.3). Для уникнення зайвої деталізації було вирішено описати варіанти використання, що доступні автентифікованому та авторизованому користувачу.

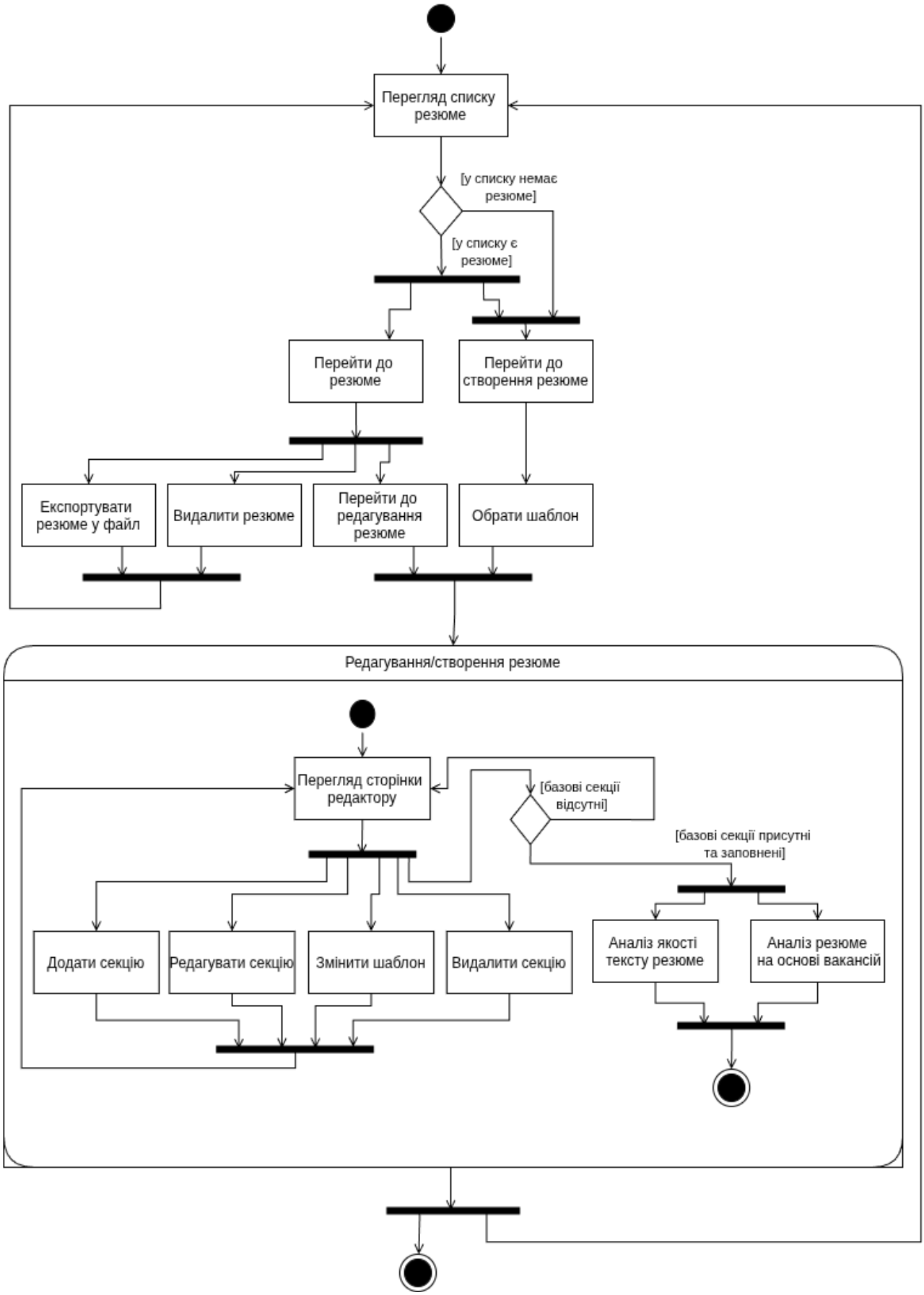

Рисунок 2.3 – Діаграма діяльності

Розроблена діаграма описує дії, які користувач може робити із вже створеними екземплярами резюме та дії, що передують створенню нового. Користувач може видалити, експортувати резюме у файл або перейти до редагування. Після обрання шаблону для резюме при створенні або виборі опції редагування для існуючого резюме відбувається перехід у комплексну дію «Редагування/створення резюме». В ході даної дії користувач може додавати, редагувати та видаляти секції резюме та змінювати їх порядок та стилізований шаблон. Користувач може перейти до аналізу якості тексту або аналізу резюме за вакансіями. Після виходу з комплексної дії, користувач може повернутись до сторінки перегляду списку резюме або завершити використання вебзастосунку.

2.4 Розробка методу багатофакторного аналізу тексту

Однією з ключових функціональних можливостей вебзастосунку для генерації резюме є багатофакторний аналіз тексту резюме. Тому необхідно детальніше розглянути метод багатофакторного аналізу тексту. Особливістю даного методу є те, що в ході аналізу виконується ряд послідовних незалежних аналітичних процесів, що призначені визначити проблемність частини тексту з точки зору певного аспекту та аналіз всього тексту в цілому для виявлення характеристик у загальному контексті. В результаті виконання методу буде сформовано звіт, що містить виявлені проблеми у тексті та рекомендовані виправлення.

До методу багатофакторного аналізу тексту входять такі складові:

- визначення ключових слів, мови, стилю тексту.
- визначення граматичних помилок;
- визначення стилістичних помилок;
- виявлення змістовних повторень;
- виявлення малозмістовних речень.

Метод багатофакторного аналізу тексту дозволить користувачу вебзастосунку виявити приховані проблеми у своєму резюме, виявити важливі характеристики, такі як стиль, змістовність, ключові слова та підвищити його

відповідність вимогам автоматизованих систем перевірки. Варто розглянути алгоритм, що реалізує даний метод.

Для роботи алгоритму на вхід надходить об’єкт резюме, що містить масив секцій, таких як контактні дані, досвід, освіта тощо. На виході алгоритму формується об’єкт-звіт з результатами складових аналізу тексту, після чого даний об’єкт обробляється на клієнт частині, відображаючи знайдені помилки, можливі виправлення тощо. Розроблений алгоритм методу багатофакторного аналізу тексту подано у вигляді блок-схеми на рисунку 2.4.

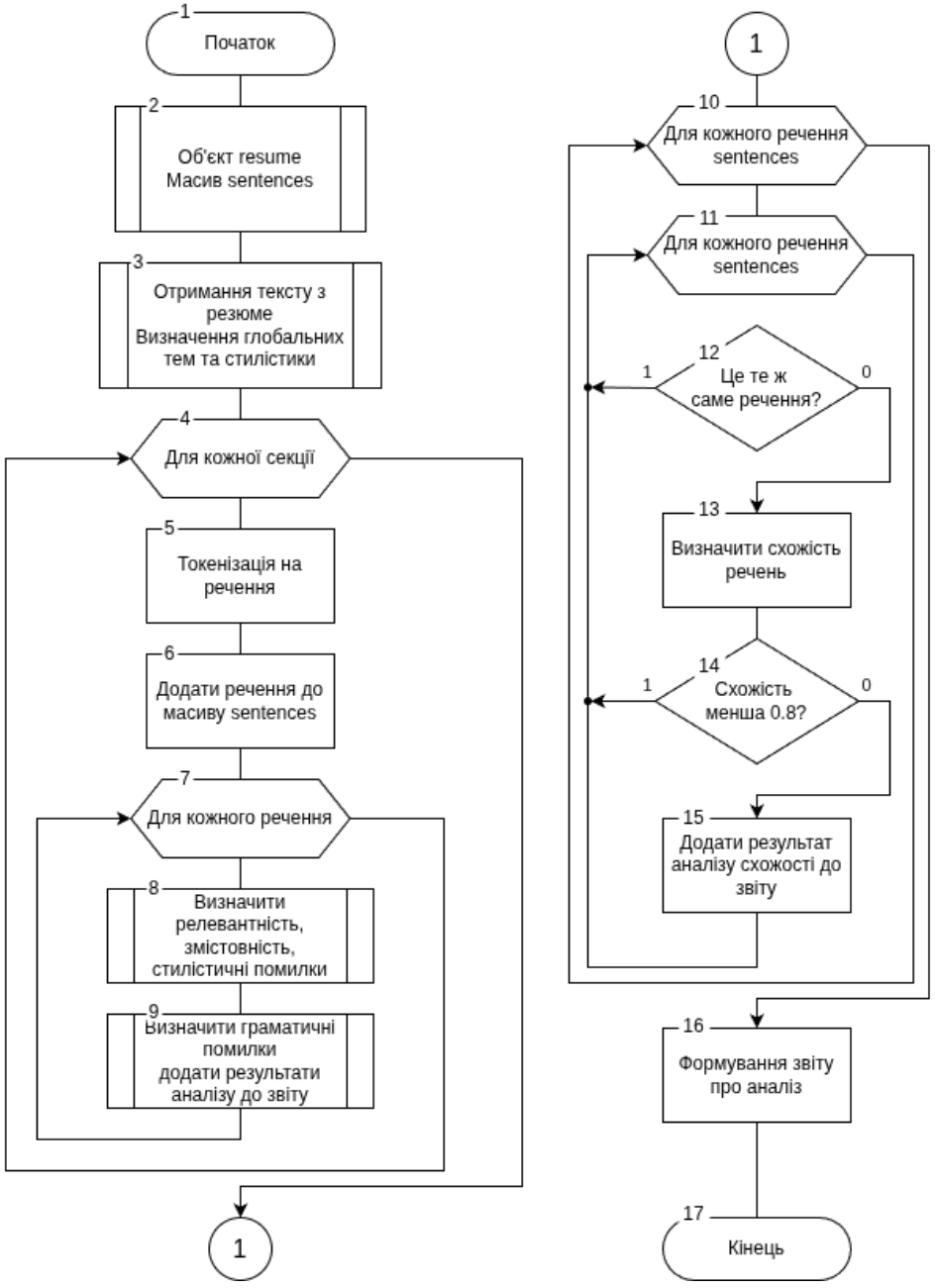

Рисунок 2.4 – Блок-схема алгоритму методу багатофакторного аналізу

Представлено пояснення кожного кроку даного алгоритму:

Крок 1. Початок алгоритму.

Крок 2. На вході в алгоритм визначається об’єкт резюме, та масив sentences, призначений для збору усіх речень резюме та подальшої їх обробки.

Крок 3. Отримання цілісного тексту резюме та формування запиту до мовної моделі на визначення глобальних тем та загального стилістичного забарвлення резюме.

Крок 4. Цикл для обробки кожної секції об’єкту резюме.

Крок 5. Виконання токенізації, тобто розбиття на окремі семантичні одиниці, секції на речення.

Крок 6. Кожне речення додається у масив sentences.

Крок 7. Цикл для обробки кожного речення поточної секції.

Крок 8. Визначення релевантності та змістовності даного речення в контексті визначених раніше глобальних тем. Пошук стилістичних помилок та формування рекомендованого виправлення.

Крок 9. Пошук граматичних помилок та формування рекомендованого виправлення. Додавання результатів аналізу речення до результату аналізу.

Крок 10. Цикл для обробки кожного речення в масиві sentences.

Крок 11. Вкладений цикл зіставлення кожного речення в масиві sentences один з одним.

Крок 12. Перевірка чи це одне і те ж речення. Якщо одне і те ж, відбувається перехід до наступної ітерації циклу.

Крок 13. Якщо це різні речення, відбувається визначення схожості речень.

Крок 14. Перевірка чи схожість менша значення 0,8. Якщо менша, то відбувається перехід до наступної ітерації циклу.

Крок 15. Додавання інформації про схожі речення до звіту про аналіз.

Крок 16. Формування результуючого об’єкту звіту.

Крок 17. Кінець алгоритму.

Отже, було розроблено метод багатофакторного аналізу тексту резюме за визначеними ознаками. Даний метод може бути використаний під час генерації

резюме для покращення вмісту та для визначення слабких місць в наповненні, наприклад недостатня кількість ключових слів, друкарські помилки, стилістичні помилки тощо. Було розроблено та описано алгоритм, що реалізує даний метод.

2.5 Розробка методу автоматизованого збору вакансій та алгоритму визначенння релевантності резюме

Однією з ключових функціональних можливостей вебзастосунку для генерації резюме є метод автоматизованого збору вакансій.

Ключовою особливістю даного методу є автоматизований збір вакансій з використанням вебскрапінгу за назвою посади або іншим пошуковим запитом користувача, що може включати також назву компанії, навичок тощо. В залежності від досконалості пошукової системи обраного ресурсу для пошуку роботи здійснюється автоматизований збір найпопулярніших серед користувачів вакансій.

Метод автоматизованого збору вакансій дозволить підвищити поінформованість кандидата про актуальні тенденції ринку праці та підвищить якість прийняття рішень у процесі формування професійного профілю

Специфічність використання вебскрапінгу, як методу автоматизованого збору інформації, передбачає неуніверсальність одного і того ж алгоритму для збору вакансій з декількох ресурсів. Така особливість зумовлена тим, що при автоматизованому зборі виконується обробка DOM (Document Object Model – об’єктна модель документа) з різних ресурсів, що мають різну структуру організації даних на вебсторінці [17].

Реалізацію методу автоматизованого збору вакансій з ресурсу для пошуку роботи Djinni [18] описано нижче у вигляді блок-схеми алгоритму. Алгоритм включає використання headless («безголовного») браузера для маніпуляції сторінками. Це такий браузер, що відкривається програмно на сервері і керується лише за допомогою програмного коду.

Розроблений алгоритм автоматизованого збору вакансій з ресурсу Djinni подано у вигляді блок-схеми на рисунку 2.5.

Рисунок 2.5 – Блок-схема алгоритму автоматизованого збору вакансій з ресурсу Djinni

Нижче детально розглянуто кроки даного алгоритму:

Крок 1. Початок.

Крок 2. На вхід до функції, що реалізує алгоритм надходить значення пошукового запиту користувача та кількість вакансій, які необхідно отримати.

Крок 3. Виконання нормалізації пошукового запиту, тобто створення коректного URL для переходу у браузері.

Крок 4. Перехід на сторінку у браузері.

Крок 5. Очікування завантаження сторінки, адже це асинхронна дія, що потребує часу на виконання.

Крок 6. Отримання контейнерів DOM, що містять опис вакансій. В даному випадку це контейнери з класом «.js-original-text».

Крок 7. Цикл для обробки перших контейнерів. У циклі оброляється кількість контейнерів, що дорівнює встановленій на початку кількості необхідних вакансій.

Крок 8. За допомогою DOM API отримується текстовий опис вакансії у контейнері та додається до результату виконання функції.

Крок 9. Кінець.

Даний алгоритм варіюється в залежності від ресурсу для збору інформації.

Формування комплексної оцінки релевантності резюме користувача вказаній посаді відбувається на основі визначення рівня релевантності кожній вакансії та визначенні середнього арифметичного значення. Розроблений алгоритм визначення релевантності резюме вакансіям подано у вигляді блок-схеми на рисунку 2.6.

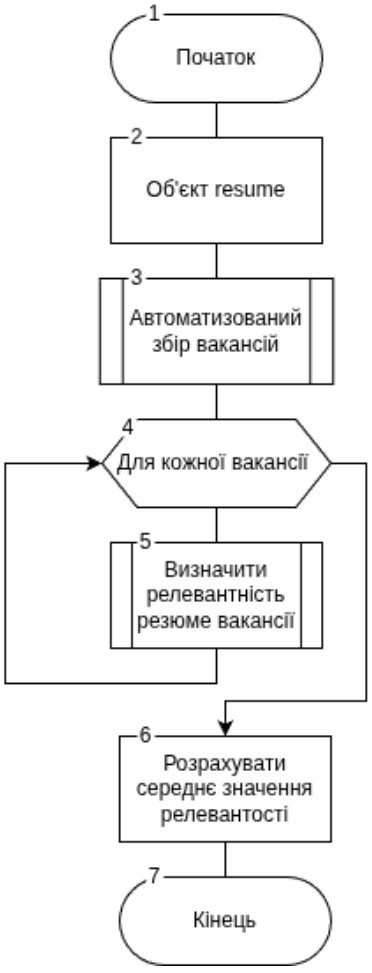

Рисунок 2.6 – Блок-схема алгоритму визначення релевантності резюме вакансіям

Доцільно розглянути кроки алгоритму детальніше:

Крок 1. Початок.

Крок 2. На вхід до алгоритму надходить об’єкт резюме користувача.

Крок 3. Виконання автоматизованого збору вакансій.

Крок 4. Цикл для обробки кожної вакансії.

Крок 5. Визначення релевантності резюме даній вакансії. Виконується на основі використання мовної моделі, адже вона враховує контекст і може цілісно порівняти відповідність двох текстів. Релеватність зберігається у вигляді відсоткового значення.

Крок 6. Після виходу з циклу розраховується середнє значення релевантності резюме знайденим вакансіям.

Крок 7. Кінець.

Варто зауважити, що автоматизований збір вакансій було подано як зумовлений процес, деталі якого блок-схема не пояснює.

Отже, було розроблено метод автоматизованого збору вакансій. Даний метод дозволить підтримувати актуальний рівень знань про ринок праці. Було розроблено алгоритм, що реалізує даний метод для збору вакансій з ресурсу Djinni. Було розроблено описано алгоритм визначення релевантності резюме вакансіям.

2.6 Висновки

В розділі було проаналізовано можливі вхідні та вихідні дані розроблюваного вебзастосунку. Було описано модель застосунку та розроблено структурні UML-діаграми компоненнтів та класів. Було описано та подано у вигляді UML-діаграми загальний алгоритм застосунку. Було розроблено метод багатофакторного аналізу тексту та описано його реалізацію у вигляді блок-схеми алгоритму. Було розроблено метод автоматизованого збору вакансій та описано його реалізацію для ресурсу Djinni у вигляді блок-схеми алгоритму. Також було розроблено та описано алгоритм визначення релевантності резюме вакансіям.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Перед початком процесу програмної реалізації програмного застосунку варто приділити увагу вибору засобам розробки. З огляду на постановку задач бакалаврської кваліфікаційної роботи та результати теоретичних досліджень, що були проведені у попередньому розділі, необхідно ґрунтовно підійти до вибору середовища розробки для вебзастосунку.

Інтегроване середовище розробки (IDE) – програмний засіб, який використовується для розробки програмних застосунків та зазвичай має підтримку розширених можливостей. Також IDE мають підтримку тестування, розгортання тощо, що дозволяє використовувати дані засоби в ході всього життєвого циклу розробки ПЗ.

Доцільно розглянути декілька популярних середовищ для розробки вебзастосунків:

- Visual Studio Code;
- JetBrains WebStorm;
- Sublime Text.

Visual Studio Code – безкоштовне середовище для розробки, підтримуване компанією Microsoft. Даний редактор відносно легкий та має велику кількість плагінів, використання яких може покращити досвід розробки та повноцінно замінити аналогічні платні IDE [19].

JetBrains WebStorm – потужний варіант IDE для веброзробки. Має багато вбудованих інструментів для розробки, підтримку фреймворків, тестування та аналізу коду. Проте даний редактор займає багато пам’яті на відміну від аналогів і професійна версія даного редактору потребує підписки [20].

Sublime Text – мінімалістичний текстовий редактор, який може функціонувати як повноцінна IDE при використанні плагінів. Даний варіант має високу швидкодію та не займає багато місця на жорсткому диску, проте потребує

час на налаштування та встановлення розширень. Деякі веброзробники використовують Sublime Text як повноцінне середовище розробки [21].

Результати порівняння характеристик розглянутих середовищ розробки наведено у таблиці 3.1.

Таблиця 3.1 – Порівняння середовищ розробки

Характеристика	VS Code	WebStorm	Sublime Text
Швидкодія	+	-	+
Розумне автодоповнення	+	+	+/-
Розширюваність	+	+/-	+/-
Інструменти відлагодження	+	+	-
Інструменти тестування	+/-	+	-
Результат	4,5/5	3,5/5	2/5

За результатами порівняльного аналізу було прийнято рішення використовувати для програмної реалізації вебзастосунку для генерації резюме середовище Visual Studio Code. Даний варіант має високу швидкодію, займає небагато місця в пам’яті комп’ютера та дозволяє виконувати реалізацію та суміжні етапи життєвого циклу програмного забезпечення.

Іншим важливим аспектом реалізації програмного застосунку є вибір мови програмування. Доцільно розглянути популярні варіанти мов програмування для реалізаціїї бізнес-логіки вебзастосунку, тобто серверу.

Node.js – фреймворк та рушій для запуску застосунків на JavaScript, в основі роботи якого лежить event loop (цикл подій). Так, як основою мови є JavaScript – типізація динамічна, проте легка інтеграція TypeScript дозволяє впровадити повноцінну типізацію для застосунку. Підтримує асинхронність, має багато вбудованих модулів для роботи з файловою системою, безпекою та побудовою серверних застосунків. Використання екосистеми npm дозволяє зручно керувати застосунками та використовувати бібліотеки, що доступні, як для серверних додатків так і для браузерного виконання [22].

Python – інтерпретована мова програмування, що працює в одному потоці з підтримкою багатопотоковості з додатковими бібліотеками. Мова має динамічну типізацію, велику екосистему бібліотек, особливо в сфері машинного навчання, автоматизації та data science. Використовується для серверної розробки, проте менш ефективна для розробки високонавантажених систем [23].

PHP – інтерпретована мова програмування, головною метою якої є веброзробка. Мова має динамічну типізацію та інструменти для легкого розгортання. Проте мова слабко підтримує асинхронність, типізацію та підтримку сучасних архітектур [24].

Результати порівняння характеристик серверних мов програмування наведено у таблиці 3.2.

Таблиця 3.2 – Порівняння серверних мов програмування

Характеристика	Node.js	Python	PHP
Швидкодія	+	-	+
Асинхронність, багатопоточність	+	+/-	-
Екосистема для веброзробки	+	-	+/-
Зручність для фронтенд-розробників	+	+/-	-
Підтримка модульності та тестування	+	+	+/-
Результат	5/5	2/5	2/5

За результатами порівняльного аналізу зроблено висновок про доцільність розробки серверу вебзастосунку на мові програмування Node.js. Дана мова дозволяє поєднати зручність розробки клієнт частини на сервері, використовувати інструменти, доступні в екосистемі npm, що надає велику кількість користувацьких бібліотек та додати до проекту статичну типізацію, що підвищить якість коду та зменшить кількість ймовірних помилок. Для розробки клієнт-частини вебзастосунку для генерації резюме буде використано фреймворк Vue.js та класичні засоби веброзробки HTML/CSS.

3.2 Розробка серверної частини вебзастосунку для генерації резюме

З огляду на результати теоретичних досліджень, проведених в ході виконання бакалаврської кваліфікаційної роботи, необхідно використати клієнт-серверну архітектуру для реалізації вебзастосунку для генерації резюме. Для розробки серверної частини застосунку було використано фреймворк Express.js, який дозволяє розробляти масштабовані REST API, та часто використовується для розробки систем [25]. Даний фреймворк є надійним фундаментом для роботи Node.js серверу, та використовується як основа для створення більш просунутих фреймворків.

Для зручності підтримки та масштабування серверної частини вебзастосунку, структуру проекту було розділено на окремі модулі:

- routes – містить маршрутизатори для координації запитів;
- controllers – містить логіку обробки запитів;
- middleware – містить проміжні функції обробки;
- services – містить основну бізнес-логіку, що використовується контролерами.

Основою для розробки REST API є визначення маршрутів або ендпойнтів, до яких клієнт частина або навіть інші застосунки, підключені до мережі зможуть робити запити. Для ефективного оперування запитами до застосунку використовується спеціальний об’єкт Router, інакше кажучи, маршрутизатор. Так, як вебзастосунок передбачає модуль реєстрації та логіну, було створено публічний маршрутизатор та приватний. Запити, що надходять до публічного можуть виконуватись без певних передумов, в той час, як приватні маршрути потребують виконання авторизації користувача при кожному запиті.

Механізм автентифікації та авторизації заснований на використанні JWT (JSON Web Token). Даний механізм дозволяє зберігати дані у вигляді зашифрованого рядку. Рядок з корисними даними, наприклад інформацією про користувача, шифрується секретним ключем на сервері, після чого може бути розшифрованим лише з використанням того ж самого ключа. Таким чином дані неможливо підробити без початкового ключа, проте з’являється необхідність

обмежувати час життя рядка, так як його можна перевикористати, якщо зловмисник заволодіє ним [26].

Блок-схема алгоритму механізму автентифікації користувача зображено на рисунку 3.1.

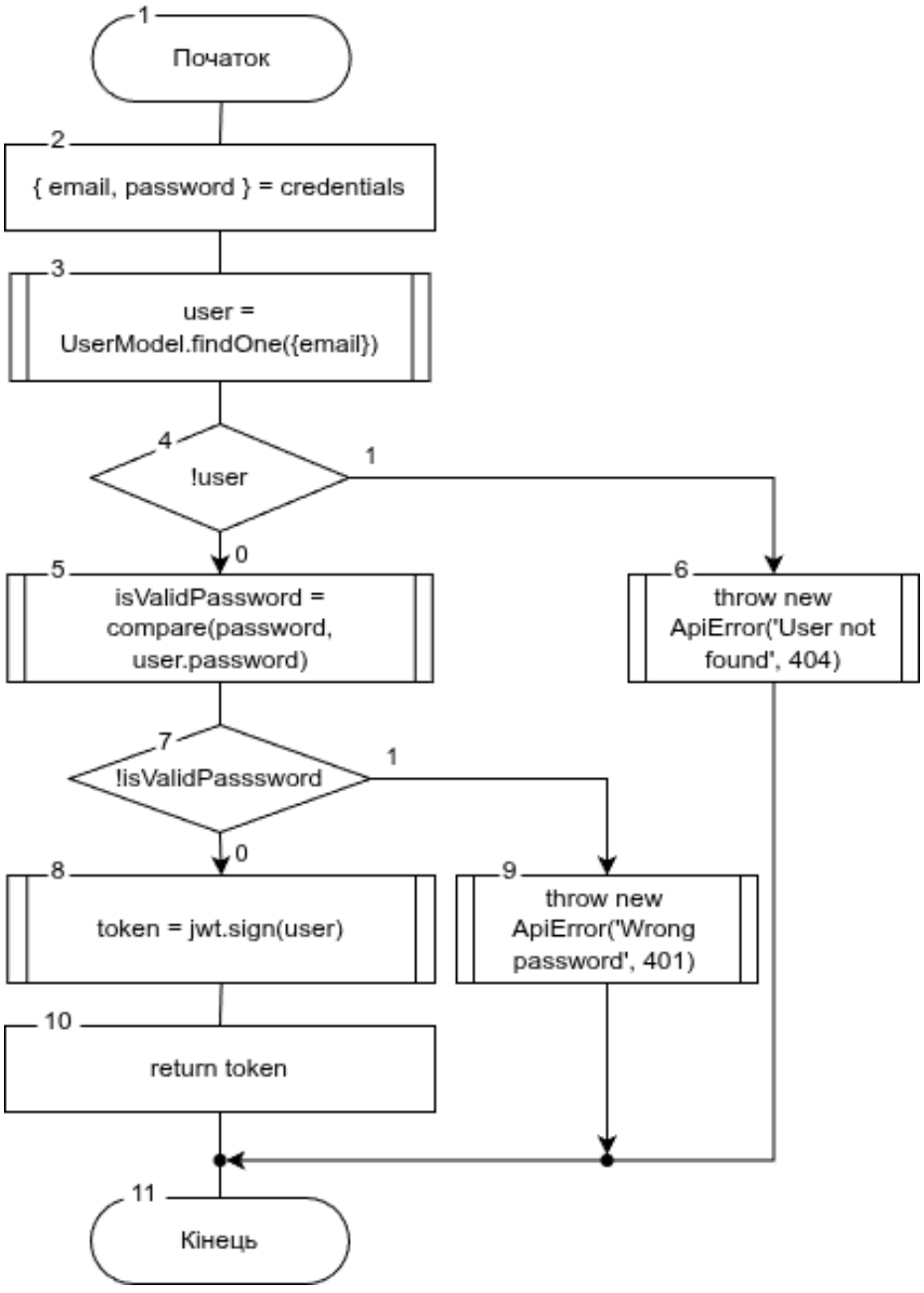

Рисунок 3.1 – Блок-схема алгоритму автентифікації користувача

На вхід функція отримує реєстраційні дані користувача, тобто електронну пошту та пароль. Якщо користувача було знайдено у базі даних користувачів,

відбувається порівняння паролю з хешем паролю користувача у базі даних. У разі, якщо користувач існує та пароль правильний, відбувається створення JWT токену з даними користувача та даний токен повертається користувачу. Після автентифікації клієнт автоматично додає отриманий токен до заголовків запитів, що потребують авторизації.

Для реалізації механізму авторизації було розроблено спеціальну проміжну функцію, що виконується при усіх запитах до приватного маршрутизатора сервера. Проміжною функцією є по суті всі обробники, які викликаються при запиті до сервера. Зазвичай проміжні функції приймають три аргументи. Призначення даних аргументів описано у прикладі нижче.

Функція авторизації отримує на вхід аргументи у вигляді об’єкту запиту користувача req, об’єкт res, що використовується сервером для надсилання відповіді на запит, та функцію next, що при виклику передає керування потоком наступній функції обробки запиту. На початку виконується отримання значення заголовку Authorization запиту, що повинен містити JWT токен користувача. Якщо заголовок відсутній і токена у клієнта користувача немає, авторизація повертає помилку про авторизацію і не дозволяє виконувати авторизовані дії далі. Якщо токен присутній, за допомогою секретного ключа шифрування токен верифікується і до подальшої обробки запитів додається інформація про користувача user, що була зашифрована в токені. В разі помилки верифікації також повертається помилка про авторизацію.

Даний підхід дозволяє зменшити кількість запитів щодо інформації про користувача та забезпечити перевірку кожного запиту, що надходить до серверу. В наступних сервісних функціях, що викликаються шляхом виклику функції next об’єкт req уже буде мати поле user, що містить інформацію про користувача.

Блок-схему алгоритму проміжної функції авторизації користувача зображено на рисунку 3.2.

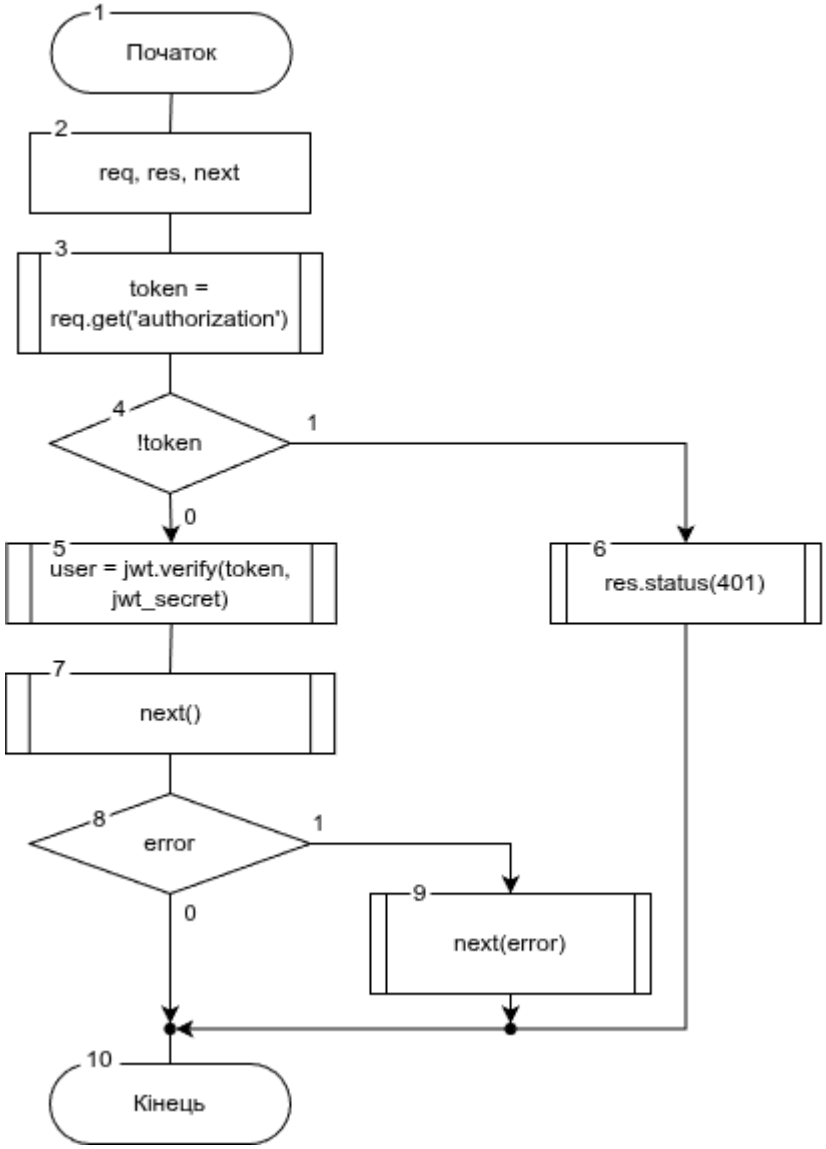

Рисунок 3.2 – Блок-схема алгоритму проміжної функції авторизації

Іншим важливим аспектом якісного REST API є зрозуміла відповідь серверу на усі запити, навіть ті, що завершились помилкою з тих чи інших причин. Для швидкої ідентифікації статусу запиту в протоколі HTTP/HTTPS використовуються спеціальні коди з трьох цифр. До прикладу, успішні запити починаються з цифри 2, статус, що сигналізує про помилку на сервері починається з цифри 5 тощо.

Для універсальної обробки помилок використовується спеціальна проміжна функція, що має четвертий аргумент об’єкт error, що містить інформацію про помилку. Таким чином, при розробці функцій-контролерів, що відповідають за обробку запитів, для надсилання коректної відповіді на запит

достатньо лиш створити помилку з її описом та статусом помилки. Повний програмний код серверного застосунку наведено у додатку В.

Отже, було описано модульну структуру серверної частини вебзастосунку для генерації резюме, описано розроблені засоби безпеки доступу до використання вебзастосунку, що полягає у застосуванні JWT для авторизації користувачів. Також описано реалізацію глобальної обробки помилок, що виникають на сервері.

3.3 Розробка модуля багатофакторного аналізу тексту

В ході проведення теоретичних досліджень за темою бакалаврської кваліфікаційної роботи було розроблено алгоритм багатофакторного аналізу тексту резюме. Для реалізації окремих складових аналізу було використано сервіс Gemini API, який входить до комплекту хмарних інструментів платформи Google Cloud. Даний сервіс надає доступ до використання різних моделей, кожна з яких виконує специфічний спектр завдань, таких як обробка тексту, зображень, відео тощо. Для точкових завдань пов’язаних з аналізом тексту було прийнято рішення використовувати сучасну та швидку модель Gemini 2.0 Flash-Lite. Дана модель є оптимізованим варіантом моделі Gemini 2.0 Flash, що пропонує обробку різних типів даних. Оптимізована модель працює швидше та з меншими часовими затримками [27].

Для досягнення максимальної формалізації та структурування даних обробки тексту для виконання конкретних завдань формується спеціальний запит до моделі (prompt), в якому вказується завдання, яке необхідно виконати та у якому вигляді необхідно надати відповідь. Відповідь моделі надається у форматі JSON, після чого може бути перетворена на об’єкт у програмному коді за допомогою функції JSON.parse мови Node.js. Таким чином можна уникнути генерації неструктурованої текстової інформації, яка непридатна для обробки програмою.

Першим етапом аналізу є визначення глобального контексту резюме, мови, якою воно написане та стилістику. Орієнтуючись на глобальні теми в

подальшому буде здійснено визначення ключових слів, наприклад навичок кандидата, назви посад тощо. Визначення мови та стилістики дозволить в ході аналізу окремих речень здійснити аналіз наявності друкарських чи граматичних помилок в правильному контексті та визначити стилістичні помилки.

Блок-схема алгоритму визначення глобального контексту резюме зображено на рисунку 3.3.

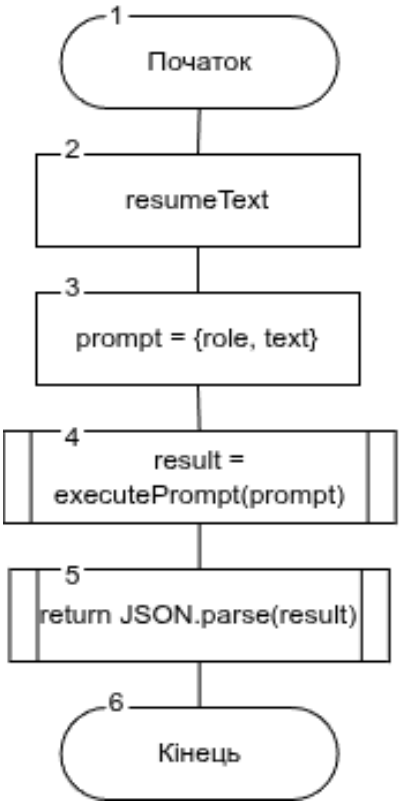

Рисунок 3.3 – Визначення глобального контексту резюме

Для уникнення ризику втрати контексту та концентрації на конкретному реченні подальша обробка резюме проводиться для кожного речення. Для цього відбувається токенізація тексту резюме на речення. Токенізація відбувається з нативними засобами мови Node.js, за допомогою регулярного виразу, що розділяє речення за розділовими знаками. Блок-схему алгоритму токенізації тексту на речення зображено на рисунку 3.4.

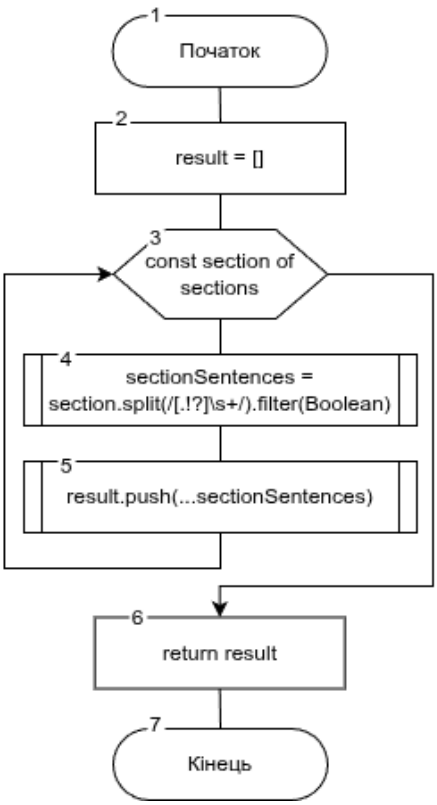

Рисунок 3.4 – Токенізація резюме на речення

В ході аналізу кожного речення відбувається перевірка на друкарські та граматичні помилки. До функції перевірки на помилки додається інформація про визначену раніше мову резюме, яка додається до запиту в якості допоміжного контексту. Функція повертає об’єкт з масивом знайдених помилок та рекомендованих виправлень. Структура промпту до мовної моделі, що використано у функції для перевірки тексту на граматичні та друкарські помилки зображено на рисунку 3.5.

```
        text: `Check the following sentence for grammar issues. Sentence language code: ${lang}

Return ONLY JSON:
{
  "hasErrors": true|false,
  "issues": [
    { "error": string, "suggestion": "string|null"}, ...
  ],
}

Sentence:
"${sentence}"`,
      }]
    }
```

Рисунок 3.5 – Промпт для перевірки тексту на друкарські помилки

За аналогічною структурою виконується визначення ключових слів у реченні та перевірка речення на релевантність глобальному контексту, інформативність та стилістику.

Заключним етапом в ході аналізу тексту є пошук змістовних дублювань у реченнях. Для цього виконується порівняння речень між собою за допомогою модуля string-similarity. Даний модуль використовує статичні методи обробки природної мови, а саме порівняння наборів біграм двох рядків на основі коефіцієнту Соренсена [28].

Діаграма послідовності виконання функції, що реалізує метод багатофакторного аналізу тексту, зображено на рисунку 3.6.

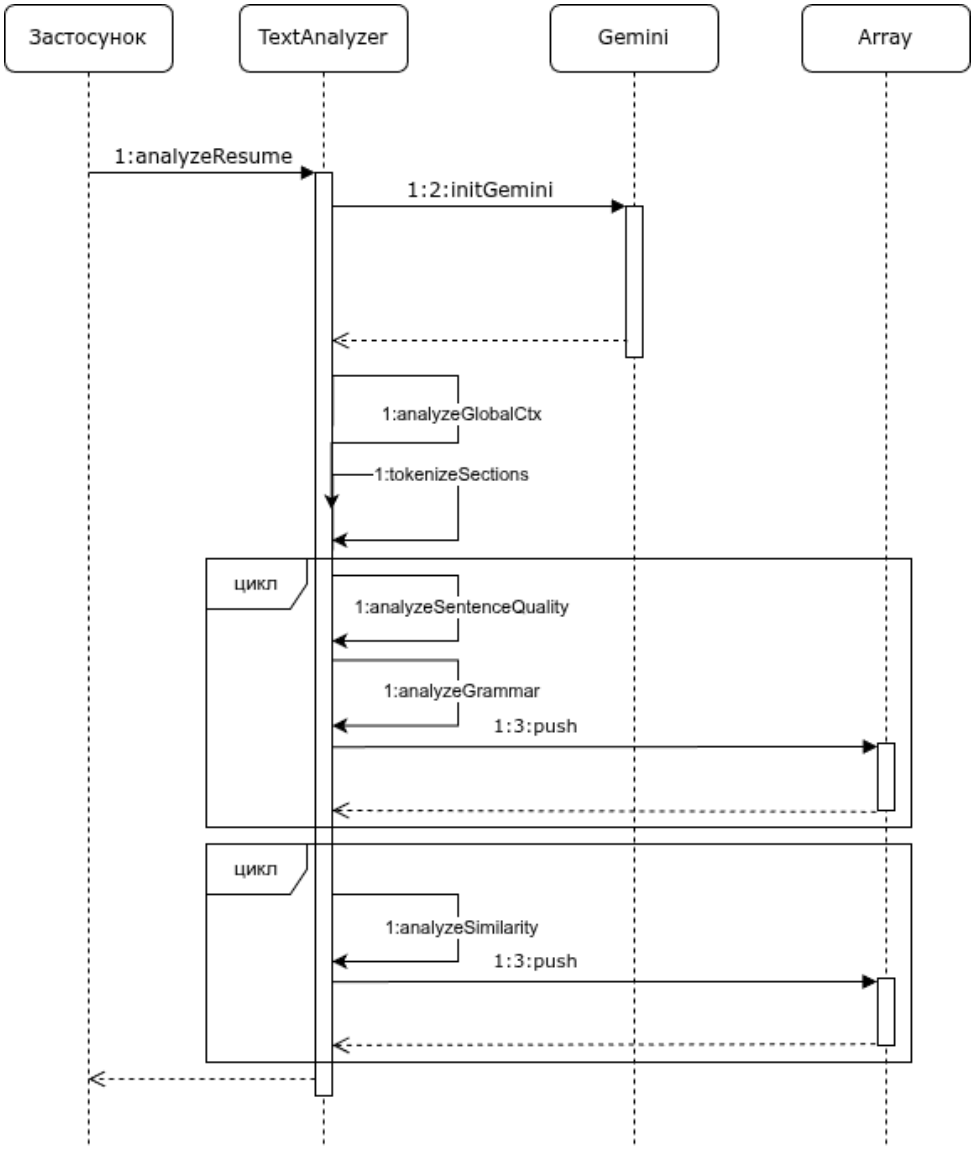

Рисунок 3.6 – Діаграма послідовності функції багатофакторного аналізу тексту

Діаграма послідовності UML дозволяє побачити участь окремих об’єктів, взаємозв’язки між ними та час виконання окремих завдань. Повний лістинг модуля багатофакторного аналізу резюме наведено у додатку В.

3.4 Розробка модуля автоматизованого збору вакансій

Як було зазначено в попередніх розділах, для автоматизованого збору вакансій буде використано технологію вебскрапінгу. Можливості сучасних інструментів для вебскрапінгу варіюються і можуть включати як аналіз статичних сторінок, так і повноцінну інтерактивну взаємодію зі сторінкою. Для розробки модуля вебскрапінгу вакансій було використано бібліотеку Puppeteer. Бібліотека надає інструментарій для запуску віртуального браузера, за допомогою якого можна переходити на сторінки в мережі Інтернет та повноцінно взаємодіяти з ними [29]. Дана особливість може бути корисною у випадку, якщо необхідно виконати дію сортування вакансій на певному сайті або здійснити пошук за допомогою введення даних в рядок тощо.

На початку алгоритму автоматизованого збору здійснюється створення нової сторінки у браузері. В залежності від того, який ресурс для пошуку вакансій вказав користувач на клієнті викликається специфічна функція для пошуку на ресурсі.

Як було зазначено в ході розробки алгоритму автоматизованого збору вакансій, способи збору інформації для кожного ресурсу з вакансіями різняться в залежності від структури організації даних на сторінці вебсайту. Доцільно розглянути нижче алгоритми збору інформації для декількох ресурсів.

Вебскрапінг вакансій з сайту для пошуку роботи Djinni включає формування параметру, що включає в себе пошукові ключові слова в рядку URL сторінки. Після завантаження сторінки за допомогою вбудованих засобів JavaScript для роботи зі сторінкою, а саме функції document.querySelector, збираються об’єкти, що містять повний опис вакансій за класом js-original-text. За вказаним у функції лімітом вакансій обирається певна кількість перших

об’єктів з отриманого масиву та ці об’єкти перетворюються на рядки з описом вакансій, після чого результат виконання функції повертається.

В даному випадку сторінка не містить взаємодії з інтерактивними елементами, так як статична сторінка з повною інформацією формується на серверній частині, а елементи можуть бути наявними в DOM, проте приховані інструментами CSS. Діаграма послідовності виконання функції збору вакансій з сайту Djinni зображено на рисунку 3.7.

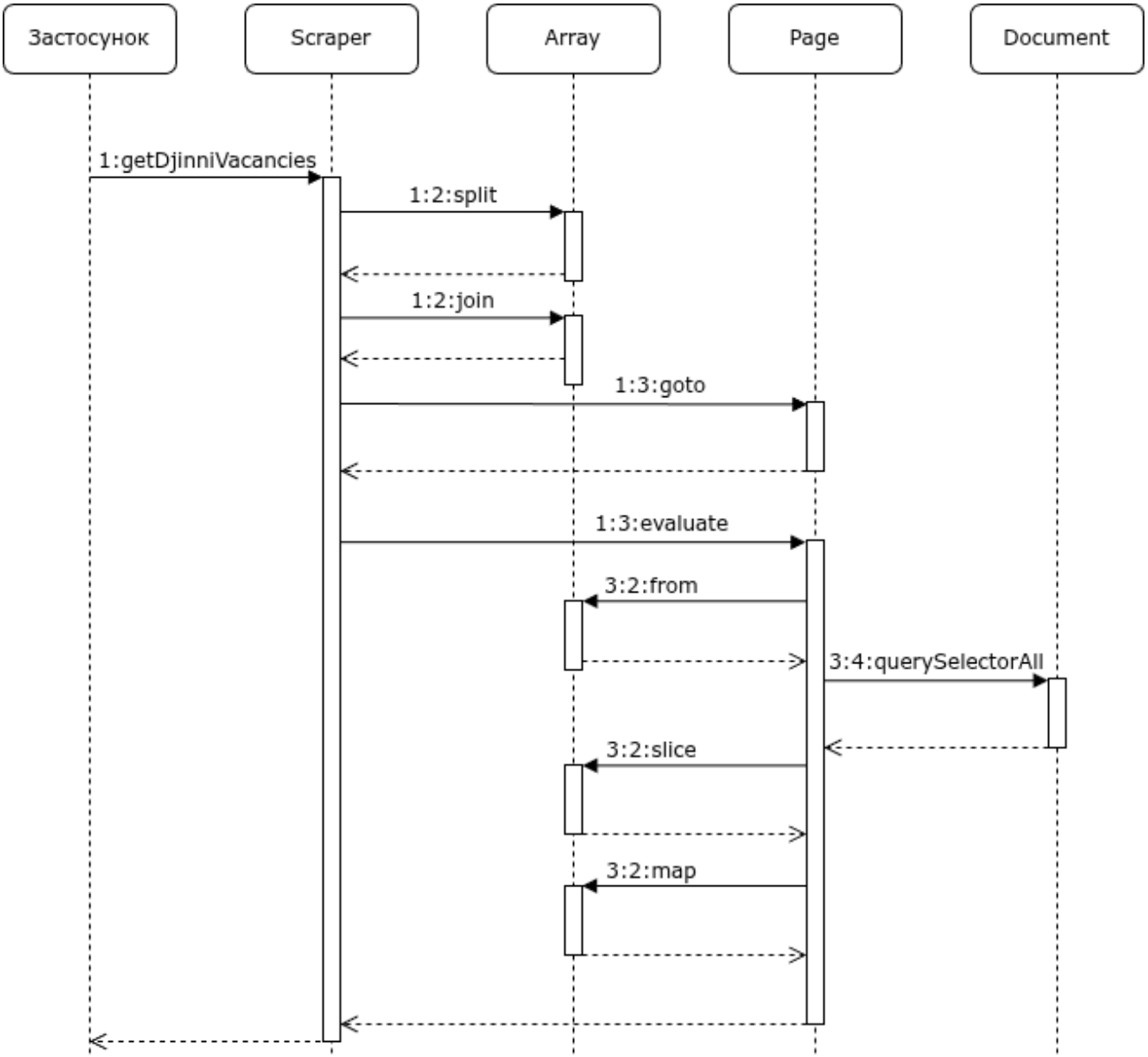

Рисунок 3.7 – Діаграма послідовності функції збору вакансій з сайту Djinni

Іншим прикладом реалізації вебскрапінгу вакансій є функція збору вакансій з відомої соціальної мережі LinkedIn. В даному випадку при пошуку повні описи вакансій підтягуються асинхронно, тільки після того, як користувач натиснув на картку з назвою і коротким описом. Це трішки ускладнює алгоритм і додає необхідність у циклі, в якому відбувається натискання на картку з назвою вакансії та очікування завантаження та відображення її повного опису в окремому контейнері. Блок-схема алгоритму автоматизованого збору вакансій з ресурсу LinkedIn зображено на рисунку 3.8.

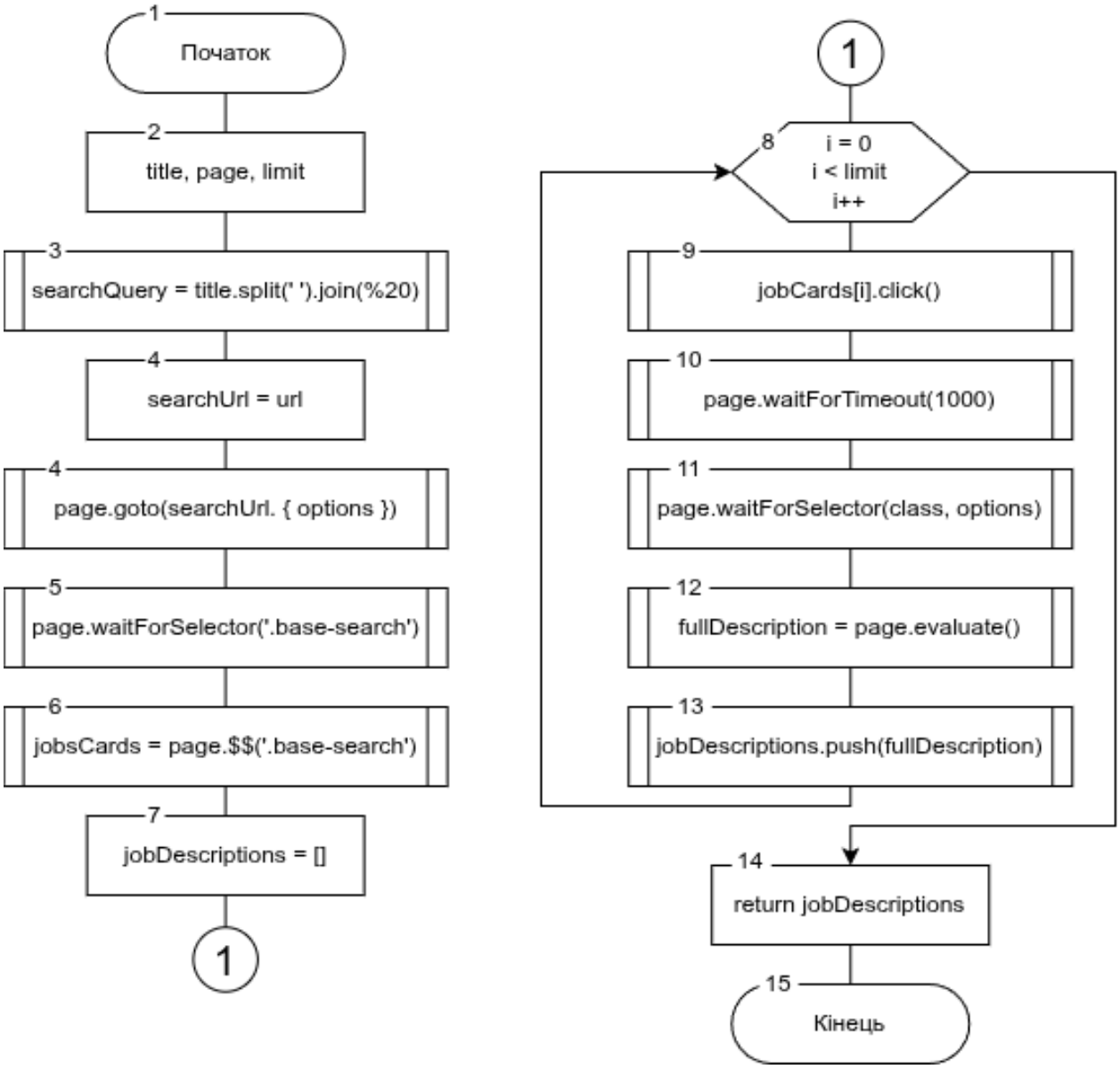

Рисунок 3.8 – Блок-схема алгоритму автоматизованого збору вакансій з LinkedIn

Повний лістинг модулю автоматизованого збору вакансій наведено у додатку В.

3.5 Висновки

У розділі було проведено варіантний аналіз та обґрунтування вибору засобів для реалізації програмного забезпечення. За результатами аналізу було зроблено висновок про доцільність використання Visual Studio Code, як інтегрованого середовища для розробки та мову програмування Node.js для розробки серверної частини вебзастосунку та бізнес-логіки. Для розробки клієнтської частини застосунку було використано стандартні засоби розробки вебсторінок JavaScript, HTML, CSS в поєднанні з фреймворком Vue.js. Було описано аспекти реалізації серверної частини вебзастосунку, а саме принцип автентифікації та авторизації та глобальну обробку помилок. Було описано процес реалізації алгоритму багатофакторного аналізу тексту резюме та алгоритми автоматизованого збору вакансій з різних ресурсів для пошуку роботи.

4 ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

4.1 Аналіз методів тестування програмного забезпечення

Для перевірки успішності та рівня якості реалізації розробленого вебзастосунку необхідно провести його тестування. Тестування – це процес перевірки якості прогамного забезпечення на відповідність встановленим вимогам.

За способом виконання виділяють ручне та автоматизоване тестування. Ручне тестування виконується людиною, яка перевіряє систему, взаємодіючи з нею як користувач. Даний підхід дозволяє наблизитись до досвіду справжнього користувача та виявити неочікувані дефекти, виявлення яких неможливо або ресурсозатратно автоматизувати, наприклад, візуальні дефекти. В той же час ручне тестування вимагає більше часу та врахування людського фактору, що передбачає можливість упустити дефекти системи [30].

Автоматизоване тестування передбачає розробку спеціальних програмних засобів для тестування застосунку. Даний метод дозволяє витратити час один раз на розробку якісних тестів, після чого вони можуть виконуватись безліч разів без залучення людини. При цьому таке тестування стає невід’ємною частиною в ході розгортання та оновлення програмного забезпечення, адже тести можуть виконуватись щоразу при розгортанні і повідомляти про нові дефекти в функціональності. Даний метод дійсно ефективний при розробці масшабованих систем, коли треба зменшити необхідність у ручному тестуванні, що потребує чимало часу, проте написати якісні скрипти для тестування може бути важким завданням. Також автоматизоване тестування може бути неадаптованим до тестування інтерфейсу та візуальних дефектів.

За рівнем абстракції тестування можна розділити на такі типи:

- модульне;
- інтеграційне;
- системне;
- приймальне.

Модульне тестування призначене перевірити якість роботи окремих модулів чи функцій системи. Даний тип тестування дозволяє забезпечити якість на фундаментальному рівні на ранніх стадіях розробки і перевірити окремі компоненти системи до її ускладнення.

Інтеграційне тестування призначене перевірити взаємодію між компонентами системи та виявити дефекти взаємодії між ними. Даний тип тестування доцільний для тестування складних систем, що містять окремі самостійні компоненти, наприклад, вебзастосунки з мікросервісною архітектурою, клієнт-серверні вебзастосунки тощо.

Системне тестування передбачає перевірку всього застосунку як цілісної системи. В даному випадку доцільна перевірка якості системи у відповідності до поставлених вимог. Саме на цьому рівні доцільна оцінка відповідності нефункціональним вимогам, таким як продуктивність, надійність, безпека тощо.

Останнім рівнем абстракції тестування є приймальне тестування, що призначене оцінити систему з точки зору кінцевого користувача. Даний рівень включає аспекти системного тестування, дозволяє виявити непередбачувані дефекти та додатково оцінити якість досвіду користувача, що використовує систему.

Для тестування вебзастосунку для генерації резюме було вирішено використовувати ручне тестування на приймальному рівні. Використання методів обробки природної мови, а зокрема мовних моделей додає специфічність вихідних даних системи, яку недоцільно тестувати автоматизовано. В той же час ручне тестування дозволяє оцінити якість верстки вебінтерфейсу, час відгуку, виявити затримки при роботі тощо.

4.2 Тестування вебзастосунку

Для тестування окремих варіантів використання системи було описано кроки, які виконує користувач при взаємодії з системою та очікуваний відгук системи на дії користувача.

Необхідно перевірити правильність валідації форм реєстрації та автентифікації в застосунок. Для цього було здійснено перехід до форми реєстрації користувача та здійснено спробу зареєструвати користувача не заповнюючи обов’язкові поля. Результатом даної дії є відображення помилок про те, що поля є обов’язковими (див. рисунок 4.1).

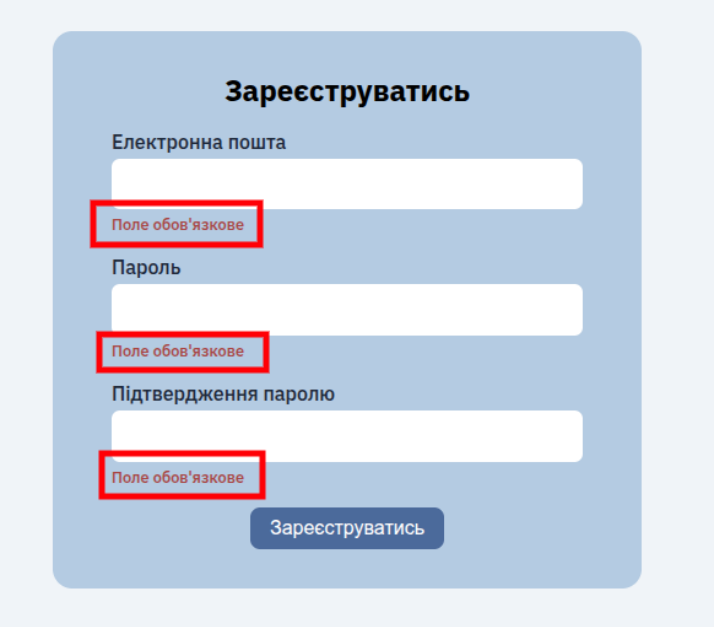

Рисунок 4.1 – Валідація пустих полів форми реєстрації

У випадку, якщо ввести значення, яке не є коректною електронною поштою, у відповідне поле, буде виведено помилку про некоректне значення. Так само система вимагає надійний пароль для користувача, що містить велику літеру, малу літеру та одне число та містить мінімум 8 символів. Поле підтвердження паролю показує помилку, якщо значення другого та третього полів не співпадають. Також при виведенні помилок можливі ситуації, коли помилка містить довгий текст і не може бути відображена повністю. В даному випадку передбачено виведення повного тексту помилки при наведенні на неї курсору миші користувача, що покращує досвід користувача і враховує можливі візуальні недоліки. Помилки про валідацію електронної пошти, паролю, підтвердження паролю та виведення повного тексту помилки при наведенні на неї курсором зображено на рисунку 4.2.

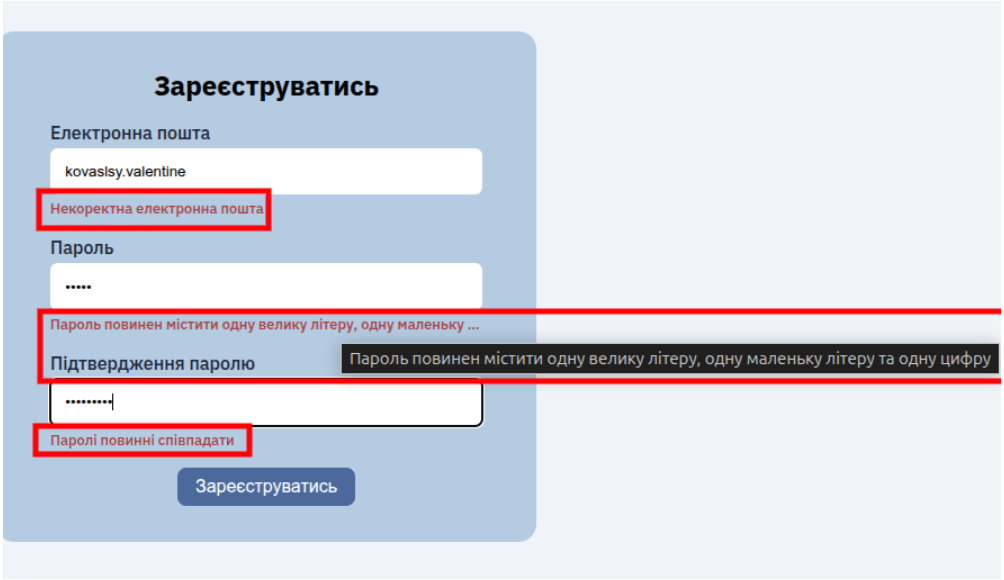

Рисунок 4.2 – Валідація некоректних значень полів форми реєстрації

При введені коректних даних форма надсилається без помилок і створюється новий користувач.

В ході тестування форми автентифікації користувача було виявлено, що форми валідуються правильно і повідомляють про некоректний формат електронної пошти користувача або паролю. При цьому при надсиланні форми з коректними даними, якщо пароль чи електронна пошта не співпадають, виводиться помилка про невідповідні дані до акаунту (див. рисунок 4.3).

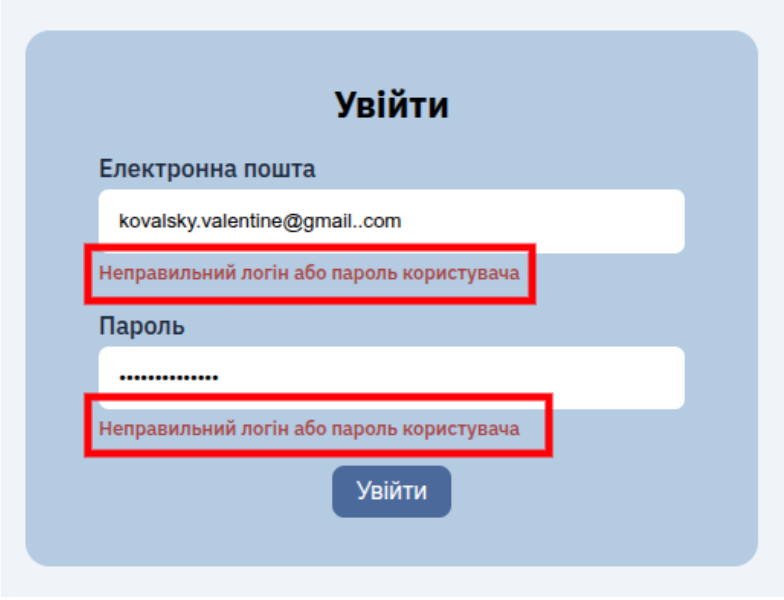

Рисунок 4.3 – Виведення помилок про невірні дані користувача

Якщо дані до акаунту коректні, виконується вхід до акаунту користувача і користувач вважається авторизованим.

Іншим важливим аспектом є перевірка функціональності заповнення та редагування резюме користувача та аналізу тексту. Для тестування було обрано шаблон для резюме та заповнено секції резюме. При заповненні секції зміни у попередньому перегляді екрану відображаються динамічно і при зміні тексту форми документ змінюється відповідно.

Динамічне відображення документу резюме при зміні тексту секції зображено на рисунку 4.4.

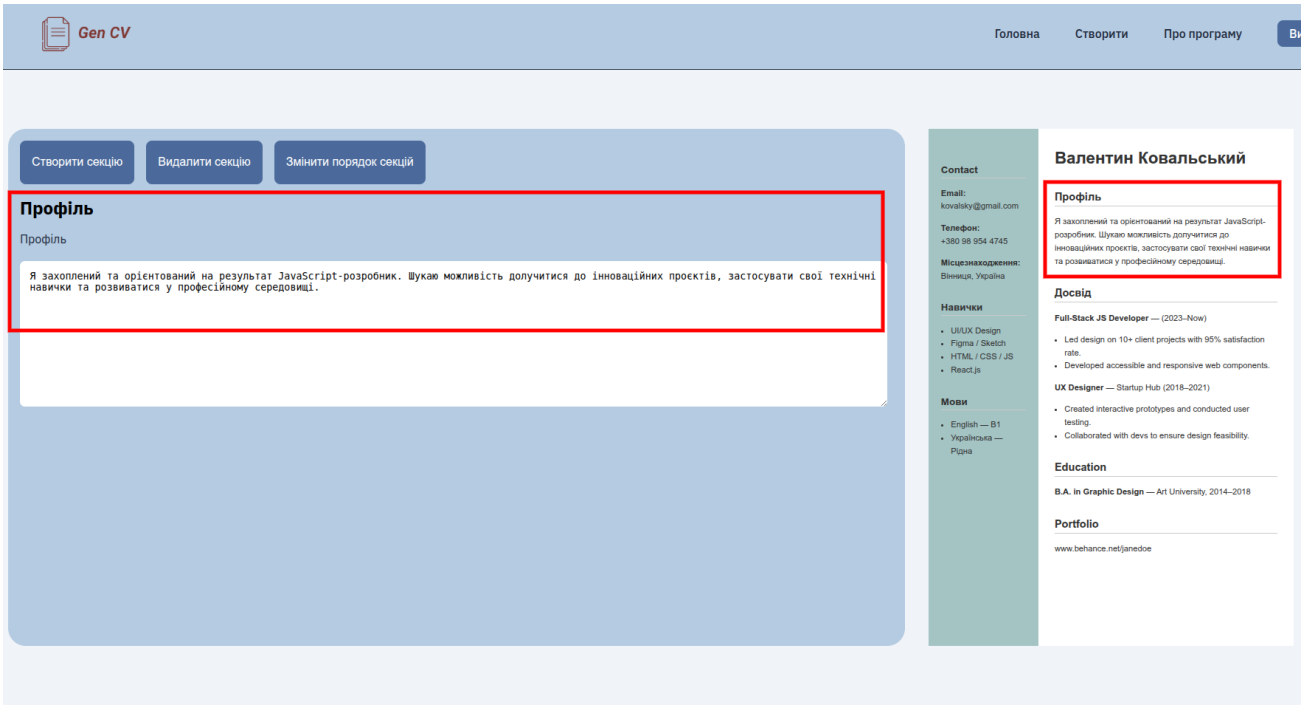

Рисунок 4.4 – Динамічне відображення змін у тексті

В ході подальшого тестування було здійснено перевірку створення секції резюме, видалення секції, зміну порядку секцій, збереження резюме у файл та аналіз резюме. В результаті тестування не було виявлено дефектів функціональності, верстки вебінтерфейсу та нефункціональних характеристик, таких як безпека, продуктивність та надійність системи.

4.3 Розробка інструкції користувача

Користувач має змогу створити та автентифікуватись в свій особистий акаунт у вебзастосунку. Користувач повинен вказати електронну пошту та надійний пароль для створення облікового запису. Після автентифікації користувач переходить на головний екран, де він може переглянути список своїх резюме та перейти до створення нового (див. рисунок 4.5).

При наведені курсору на резюме відображається назва файлу та час, коли було створено чи змінено резюме.

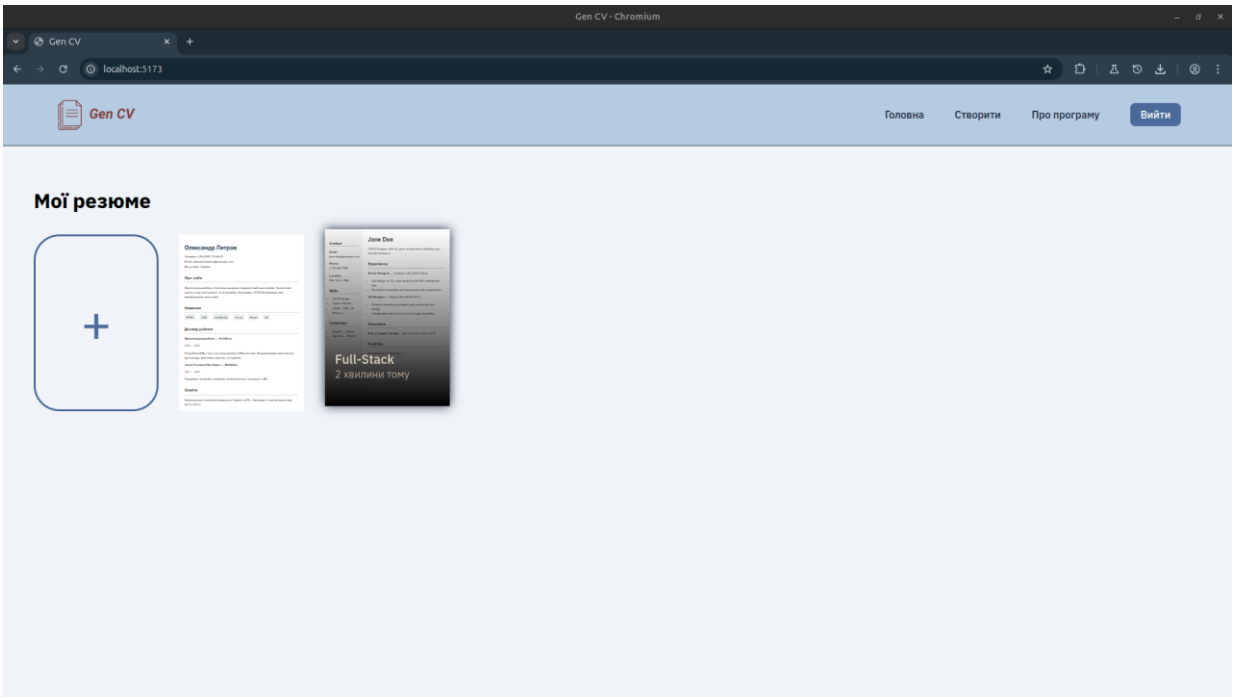

Рисунок 4.5 – Головний екран вебзастосунку

При переході до створення нового резюме за кнопкою у списку або кнопкою «Створити» на навігаційній панелі відкривається сторінка із шаблонами для резюме (див. рисунок 4.6).

Шаблони визначають розташування основних елементів на сторінці файлу резюме та мають різне стилістичне оформлення. Після створення резюме користувач зможе змінити шаблон в подальшому. При цьому усі текстові секції будуть збережені і розміщені відповідно до нового шаблону.

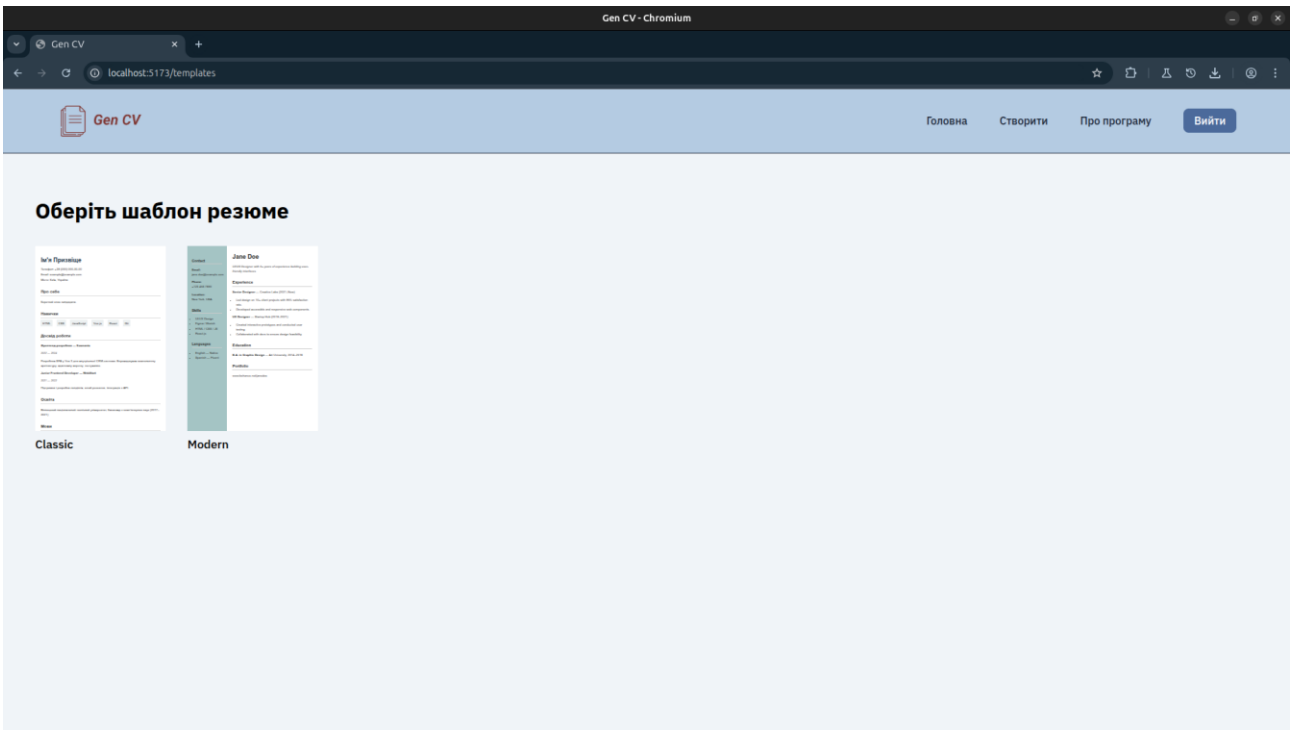

Рисунок 4.6 – Сторінка з шаблонами для резюме

При натисканні на шаблон відкривається попередній перегляд шаблону. Користувач може продовжити створення резюме з шаблоном або закрити попередній перегляд (див.рисунок 4.7).

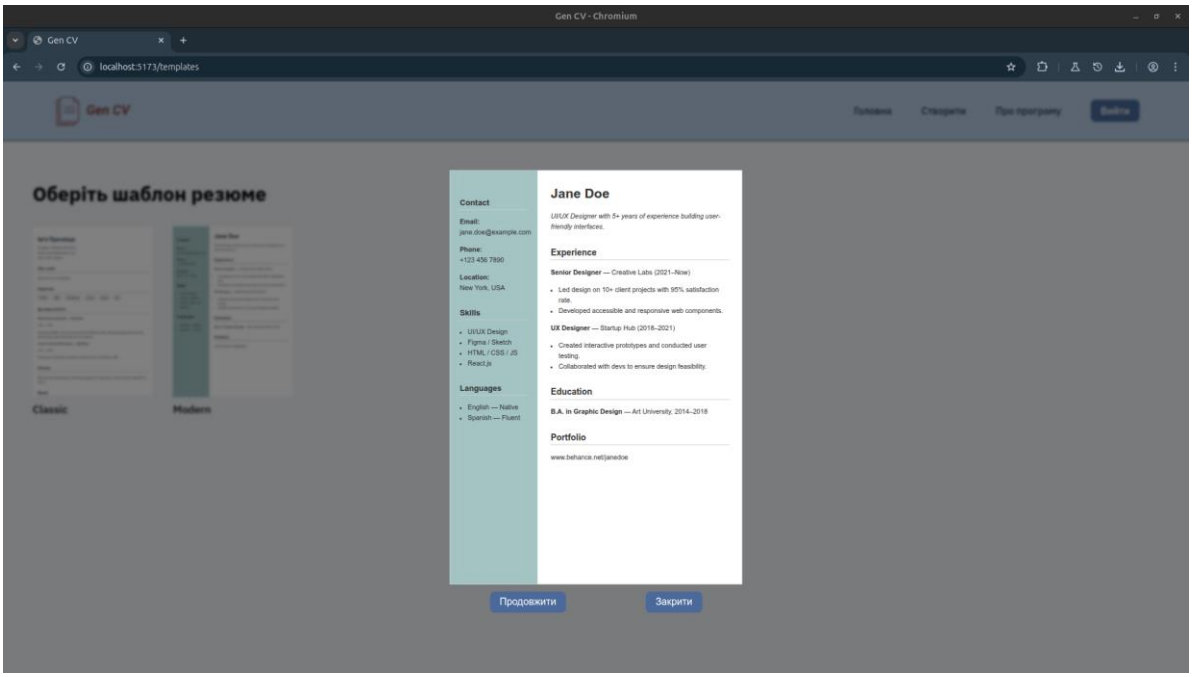

Рисунок 4.7 – Попередній перегляд шаблону

Перейшовши до створення або редагування резюме користувач бачить сторінку, що слугує редактором. Сторінка складається з елементів управління, що дозволяють змінювати текст резюме, додавати, видаляти секції або змінювати їх порядок. Також в правій частині екрану розміщено попередній перегляд файлу резюме, вигляд якого змінюється динамічно.

Для збереження та подальшого аналізу необхідно мати заповненими такі базові секції, як особиста інформація, контакти та профіль кандидата. Користувач може додати та змінити порядок секцій у резюме у модальному вікні за допомогою перетягування назв секцій. Модальне вікно для зміни порядку секцій зображено на рисунку 4.8.

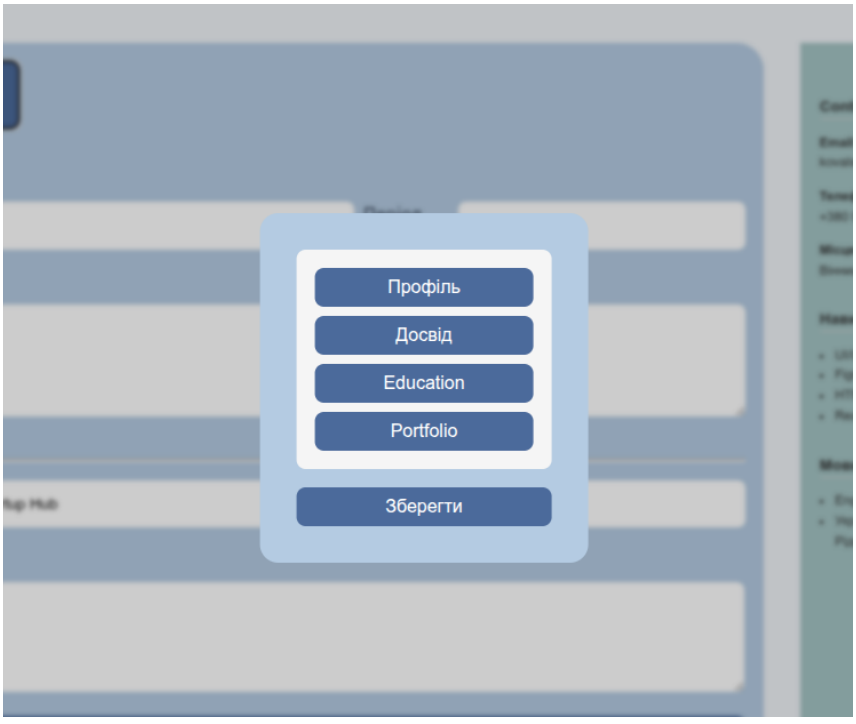

Рисунок 4.8 – Модальне вікно зміна порядку секцій

Також користувач має можливість створити нову секцію, що може бути корисним для демонстрації додаткової інформації або досягнень, які є цінними в певних професіях. Передбачено можливість видалення секцій.

При заповненні секцій, що містять декілька однотипних записів, користувач має можливість додавати та видаляти такі записи. Приклад заповнення секції «Досвід» зображено на рисунку 4.9.

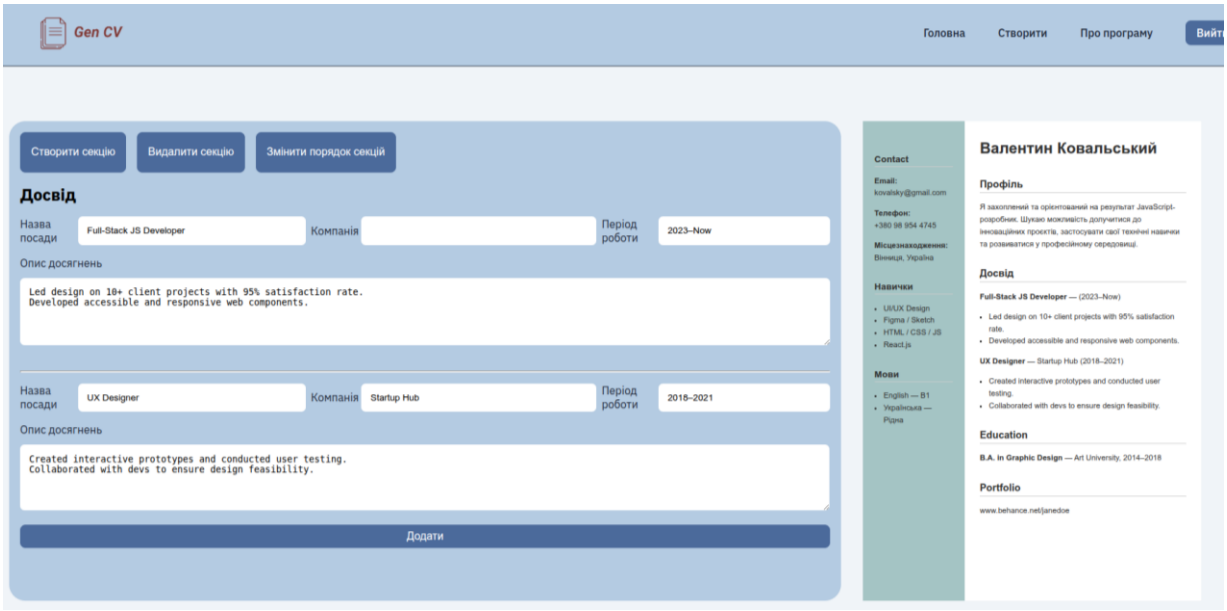

Рисунок 4.9 – Заповнення секції про досвід кандидата

Опісля заповнення резюме користувач може зберегти резюме, експортувати у файл або перейти до подальшого аналізу. Вигляд сторінки перед початком аналізу зображено на рисунку 4.10.

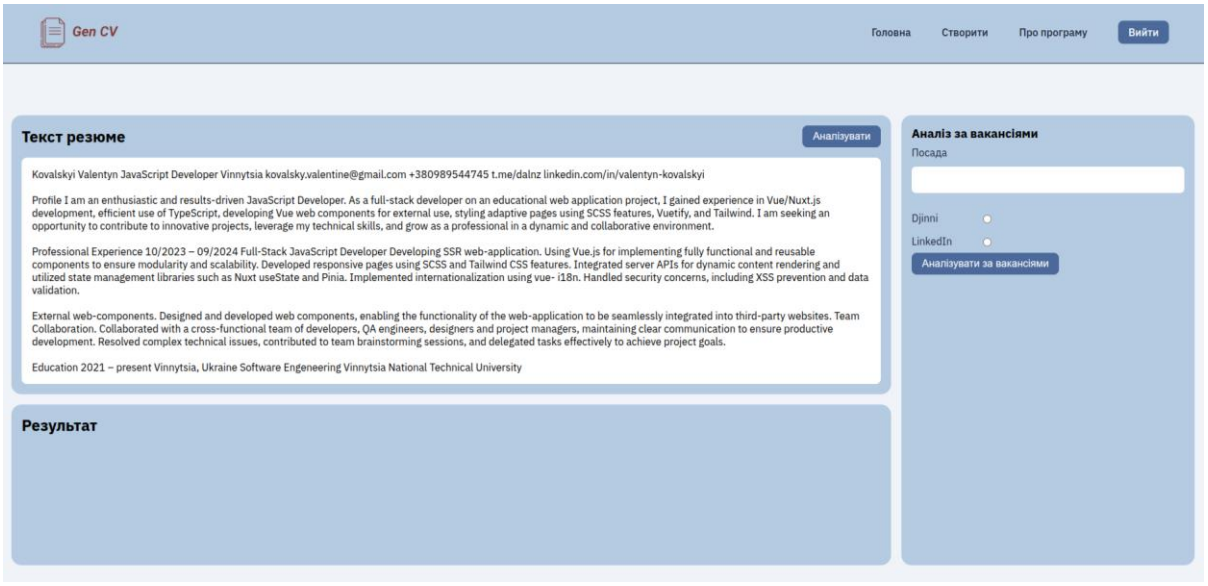

Рисунок 4.10 – Сторінка аналізу перед початком аналізу

Після натиснення на кнопку «Аналізувати» запускається алгоритм багатофакторного аналізу, що був описаний у попередніх розділах. Орієнтовно

через 20-40 секунд відображаються результати багатофакторного аналізу. Аналіз текстю резюме включає визначення мови, якою написане резюме, загальної стилістики, ключових слів, виявлення друкарських та стилістичних помилок, та визначення малозмістовних речень або логічних повторень.

Сторінка з відображенням результатів багатофакторного аналізу зображено на рисунку 4.11.

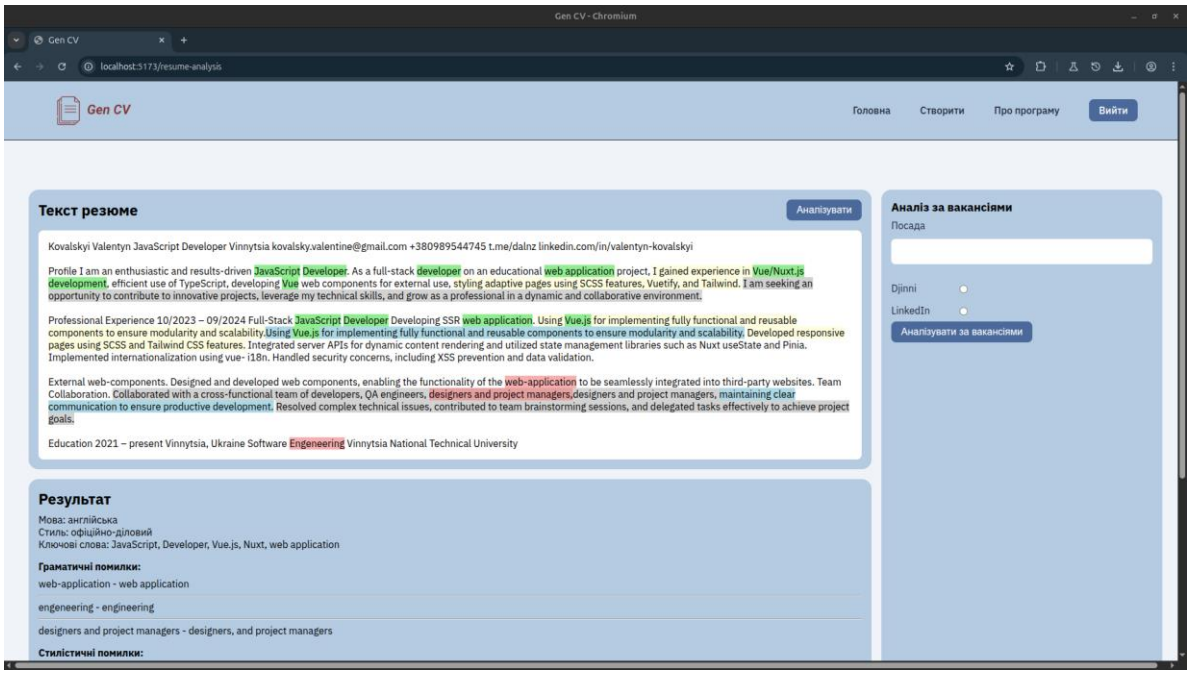

Рисунок 4.11 – Сторінка з відображенням результатів аналізу

Результати аналізу представлені візуальним виділенням частин тексту за допомогою кольорів та текстового опису аналізу в нижній частині сторінки. Зеленим у тексті виділено ключові слова, жовтим виділено повторюваний зміст, сірим виділено малозмістовні речення, червоним виділено граматичні та друкарські помилки, синім виділено стилістичні помилки. Оскільки візуальний аналіз може виглядати громіздким, користувач має можливість переглянути текстовий аналіз, в якому наведено знайдені помилки та запропоновано ймовірні покращення до тексту. Також при наведенні курсора миші на виділення можна одразу ж побачити ймовірне виправлення або пораду щодо покращення фрагменту тексту (див. рисунок 4.12).

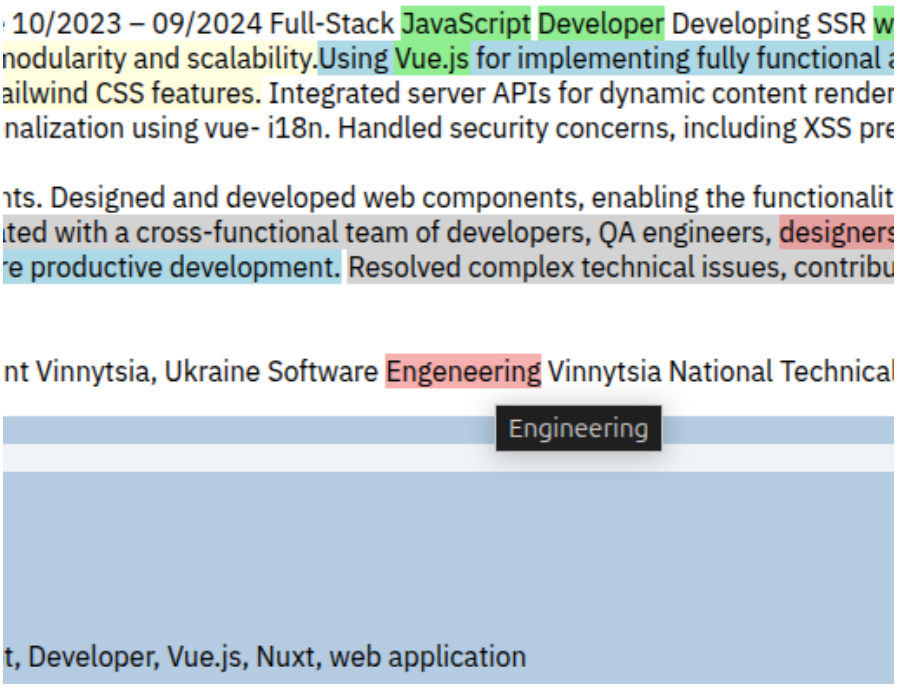

Рисунок 4.12 – Приклад відображення виправлення при наведенні на помилку

Іншим важливим аспектом аналізу тексту є аналіз релевантності резюме актуальним вакансіям. Для цього у правій частині екрану користувач може ввести назву посади, на яку він претендує, обрати один з доступних ресурсів для пошуку роботи і здійснити такий аналіз. При натисканні кнопки «Аналізувати за вакансіями» буде здійснено автоматизований збір вакансій за посадою та визначено середній рівень релевантності резюме вказаній посаді.

В результаті аналізу резюме за вакансіями буде відображено назви знайдених вакансій, при натисканні на які можна перейти на ресурс для пошуку роботи для повного перегляду опису вакансії. Кожній знайденій вакансії відповідає визначений рівень релевантності у відсотках. В залежності від рівня релевантності колір змінюється від зеленого, коли релевантність висока, до червоного, коли релевантність низька.

Також в результаті аналізу виводиться середній рівень релевантності вакансіям, знайденим за пошуковим запитом користувача. Результат виконання аналізу за актуальними вакансіями зображено на рисунку 4.13.

Рисунок 4.13 – Результати аналізу релевантності резюме вакансіям

Таким чином, було описано принцип використання вебзастосунку для автоматизованої генерації та аналізу якості тексту резюме з використанням методів обробки природної мови.

4.4 Системні вимоги

Важливим аспектом використання розробленого вебзастосунку є формалізація системних вимог, які необхідні для забезпечення його роботи. Для клієнт-серверного застосунку необхідно окреслити вимоги для апаратного забезпечення, на якому буде розгорнуто сервер, та вимоги до пристроїв, на яких буде відбуватись взаємодія з клієнт-частиною.

Так як сервер працює на легкому JavaScript фреймворку Express.js, застосунок не потребуватиме великих ресурсів.

Системні вимоги для розгортання серверу розробленого вебзастосунку наведено у таблиці 4.1.

Таблиця 4.1 – Системні вимоги до серверу

Операційна система	Linux (Ubuntu 20.04+)
Оперативна пам’ять	512 Мегабайтів
Накопичувач	16 Гігабайтів
Головний процесор	2 ядра
Додаткові вимоги	Node.js(18+), Yarn, MongoDB Server

Вимоги до клієнтської частини вебзастосунку включають вимоги до оперативної пам’яті девайсу користувача та до браузера, так як вебзастосунок може використовуватись на довільній операційній системі, що має браузер. Системні вимоги для запуску вебзастосунку наведено у таблиці 4.2.

Таблиця 4.2 – Системні вимоги до клієнта

Браузер	Chrome, Firefox, Edge останньої версії
Оперативна пам’ять	4 Гігабайти
Доступ до мережі Інтернет	1Мегабіт/секунда або більше
Додаткові вимоги	Увімкнений JavaScript у браузері

Отже, було описано системні вимоги до апаратного забезпечення, на якому буде розгорнуто серверний застосунок та буде запускатись клієнтська частина вебзастосунку.

4.5 Висновки

У розділі було проаналізовано підходи до тестування програмного забезпечення. Було проведено ручне приймальне тестування вебзастосунку для генерації резюме, розроблено інструкцію користувача та визначено системні вимоги до вебзастосунку.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі було розроблено вебзастосунок для автоматизованої генерації та аналізу якості резюме з використанням методів обробки природної мови. Для реалізації вебзастосунку було використано інтегроване середовище розробки Visual Studio Code.

Пояснювальну записку до бакалаврської кваліфікаційної роботи оформлено згідно методичних вказівок [31].

В ході виконання роботи було розглянуто сучасний стан технологій для автоматизованої генерації резюме та доведено доцільність виконання дослідження. Було здійснено порівняльний аналіз аналогічних програмних засобів, який підтвердує необхідність розробки вебзастосунку для генерації резюме. Також було здійснено аналіз методів розв’язання задачі, який дозволив прийняти рішення щодо обрання підходу до аналізу тексту та автоматизованого збору інформації.

Було проаналізовано можливі вхідні та вихідні дані розроблюваного вебзастосунку. Було описано модель застосунку та розроблено структурні UML-діаграми компоненнтів та класів. Було описано та подано у вигляді UML-діаграми загальний алгоритм застосунку. Було розроблено метод багатофакторного аналізу тексту та описано його реалізацію у вигляді блок-схеми алгоритму. Було розроблено метод автоматизованого збору вакансій та описано його реалізацію у вигляді блок-схеми алгоритму. Також було розроблено та описано алгоритм визначення релевантності резюме вакансіям.

В ході виконання конструктивного етапу дослідження було здійснено варіантний аналіз засобів розробки вебзастосунку. На основі проведених порівняльних аналізів середовищ розробки та мов програмування було прийнято рішення про доцільність використання Visual Studio Code в якості середовища розробки та мови програмування Node.js для розробки серверу та бізнес-логіки застосунку. Для розробки вебінтерфейсу було використано стандартні засоби веброзробки HTML/CSS/JS та фреймворк Vue.js. Було описано аспекти розробки

серверної частини вебзастосунку та реалізації основних функціональних алгоритмів. Реалізовані функції було описано у вигляді блок-схем алгоритмів та UML-діаграм послідовності. В результаті дослідження було розроблено повністю функціональний вебзастосунок, що виконує поставлені завдання.

На завершальному етапі дослідження було здійснено вибір методів тестування застосунку та здійснено ручне приймальне тестування, що підтверджує повну працездатність застосунку та відповідність технічному завданню. Було розроблено інструкцію користувача та описано системні вимоги для розгортання серверу вебзастосунку та використання клієнтської частини.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ковальський В. А., Романюк О. В. Розробка вебзастосунку для автоматизованої генерації та аналізу якості резюме з використанням методів обробки природної мови // Матеріали LIV науково-технічної конференції підрозділів Вінницького національного технічного університету – 2025: збірник доповідей Всеукраїнської науково-технічної конференції, 24–27 березня, 2025 р. Вінниця – Вінниця: ВНТУ, 2025 – 303–306 с.

2. Ковальський В. А., Романюк О. В. Застосування трансформерних моделей для обробки природної мови у вебзастосунку для генерації резюме // Стан, досягнення і перспективи інформаційних систем і технологій: матеріали XXV Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів. Одеса, 17–18 квітня 2025 р. – Одеса: Видавництво ОНТУ, 2025 – 116–117 с.

3. What is an Applicant Tracking System (ATS)?: вебсайт. URL: https://www.jobscan.co/blog/8-things-you-need-to-know-about-applicant-tracking-systems/ (дата звернення: 26.03.2025).

4. Fuller, J., Raman, M., Sage-Gavin, E., Hines, K., та інші Hidden Workers: Untapped Talent: звіт. – Бостон, США: Harvard Business School Project, Accenture, 2021 – 74 с.

5. FlowCV: вебсайт. URL: https://flowcv.com/ (дата звернення: 27.03.2025).

6. Canva: вебсайт. URL: https://www.canva.com (дата звернення: 27.03.2025).

7. CVMaker: вебсайт. URL: https://www.cvmaker.com.ua/ (дата звернення: 27.03.2025).

8. Understanding TF-IDF (Term Frequency-Inverse Document Frequency): вебсайт. URL: https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/ (дата звернення: 30.03.2025).

9. N-grams in NLP: вебсайт. URL: https://bit.ly/3SA8aA0 (дата звернення: 30.03.2025).

10. Vectorization Techniques in NLP: вебсайт. URL: https://bit.ly/3HosaTP (дата звернення: 30.03.2025).

11. What is GPT AI?: вебсайт. URL: https://aws.amazon.com/what-is/gpt/ (дата звернення: 30.03.2025).

12. S. Ravichandiran Getting Started with Google BERT: посібник – Бірмінгем, Велика Британія: Packt Publishing, 2021 – 352 с.

13. What is Web-Scraping and how to use it?: вебсайт. URL: https://www.geeksforgeeks.org/what-is-web-scraping-and-how-to-use-it/ (дата звернення: 30.03.2025).

14. Client-Server Model: вебсайт. URL: https://www.geeksforgeeks.org/client-server-model/ (дата звернення: 5.04.2025).

15. MongoDB: вебсайт. URL: https://www.mongodb.com/ (дата звернення: 6.04.2025).

16. What is Activity Diagram?: вебсайт. URL: https://bit.ly/4jxSYOA (дата звернення: 7.04.2025).

17. DOM. Living Standard: вебсайт. URL: https://dom.spec.whatwg.org/ (дата звернення: 8.04.2025).

18. Анонімний пошук роботи на Djinni: вебсайт. URL: https://djinni.co/ (дата звернення: 9.04.2025).

19. Visual Studio Code: вебсайт. URL: https://code.visualstudio.com/ (дата звернення: 16.04.2025).

20. The JavaScript and TypeScript IDE: вебсайт. URL: https://www.jetbrains.com/webstorm/ (дата звернення: 16.04.2025).

21. Sublime Text: вебсайт. URL: https://www.sublimetext.com/ (дата звернення: 16.04.2025).

22. Node JS: вебсайт. URL: https://nodejs.org/en. (дата звернення: 17.04.2025).

23. Python: вебсайт. URL: https://www.python.org/ (дата звернення: 17.04.2025).

24. PHP: вебсайт. URL: https://www.php.net/ (дата звернення: 17.04.2025).

25. Express. Web framework for Node.js: вебсайт. URL: https://expressjs.com/ (дата звернення: 20.04.2025).

26. JSON Web Token: вебсайт. URL: https://jwt.io/ (дата звернення: 21.04.2025).

27. Gemini Developer API: вебсайт. URL: https://ai.google.dev/gemini-api/docs (дата звернення: 22.04.2025).

28. Dice Coefficient. What is it?: вебсайт. URL: https://bit.ly/4jw1UnJ (дата звернення: 24.04.2025).

29. D. Kondratiuk UI Testing with Puppeteer: посібник – Бірмінгем, Велика Британія: Packt Publishing, 2021 – 298 с.

30. Що таке тестування програмного забезпечення?: вебсайт. URL: https://qalight.ua/baza-znaniy/shho-take-testuvannya-programnogo-zabezpechennya/ (дата звернення: 2.05.2025).

31. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення» / Уклад. : О. Н. Романюк, Г. О. Черноволик. – Вінниця : ВНТУ, 2022. – 51 с.

ДОДАТКИ

67

Додаток А. Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
«21» березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу
«Розробка вебзастосунку для автоматизованої генерації та аналізу якості резюме з використанням методів обробки природної мови»
за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. Романюк О. В.
«21» березня 2025 р.

Виконав:
студент гр. ЗПІ-21б Ковальський В. А.
«21» березня 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка вебзастосунку для автоматизованої генерації та аналізу якості резюме з використанням методів обробки природної мови».

Галузь застосування – цифрові HR-технології, платформи для створення та аналізу резюме, автоматизовані системи кар'єрного консалтингу.

2. Підстава для розробки

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від 20 березня 2025 року про закріплення тем БКР.

3. Мета та призначення розробки

Метою роботи є підвищення якості резюме під час його автоматизованої генерації у вебзастосунку за рахунок використання методів обробки природної мови, що дозволить кандидатам, що шукають роботу підвищити шанси бути поміченими роботодавцями та знайти роботу.

Призначення роботи – розробка та реалізація вебзастосунку для автоматизованої генерації резюме з використанням методів обробки природної мови.

4. Вихідні дані для проведення НДР

Перелік основних джерел, на основі яких буде виконуватись БКР:

1. Gemini Developer API: вебсайт. URL: https://ai.google.dev/gemini-api/docs (дата звернення: 22.04.2025).

2. What is Web-Scraping and how to use it?: вебсайт. URL: https://www.geeksforgeeks.org/what-is-web-scraping-and-how-to-use-it/ (дата звернення: 30.03.2025).

3. D. Kondratiuk UI Testing with Puppeteer: посібник – Бірмінгем, Велика Британія: Packt Publishing, 2021 – 298 с.

4. Client-Server Model: вебсайт. URL: https://www.geeksforgeeks.org/client-server-model/ (дата звернення: 5.04.2025).

5. Технічні вимоги

Тип програмного забезпечення – вебзастосунок; тип архітектури – клієнт-серверна; система керування базами даних – MongoDB; середовище розробки – Visual Studio Code; мова програмування – JavaScript, Node.js; метод автоматизованого збору інформації – вебскрапінг; методи обробки природної мови – велика мовна модель (LLM), n-грами; хмарні сервіси – Google Cloud; вхідні дані – текстові дані користувача; вихідні дані – файл резюме, стилізований шаблон для резюме, результат багатофакторного аналізу тексту резюме, результати автоматизованого збору вакансій, результат аналізу релевантності резюме вакансіям.

6. Конструктивні вимоги

Вебзастосунок та вебклієнт повинні відповідати ергономічним та технічним вимогам.

Текстова та графічна документація повинна відповідати стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

- пояснювальна записка до БКР;
- технічне завдання;
- лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз технологій автоматизованої генерації резюме та постановка задач дослідження	25.03.25 – 1.04.25
2	Розробка методів та алгоритмів роботи вебзастосунку	2.04.25 – 14.04.25
3	Розробка програмних модулів вебсистеми	15.04.25 – 30.04.25
4	Тестування вебзастосунку та розробка інструкції користувача	1.05.25 – 15.05.25
5	Оформлення матеріалів до захисту БКР	16.05.25 – 30.05.25

10. Порядок контролю та прийняття.

Всі етапи бакалаврської кваліфікаційної роботи контролюються науковим керівником згідно плану по виконанню роботи.

Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

Додаток Б. Протокол перевірки БКР на наявність текстових запозичень

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка вебзастосунку для автоматизованої генерації та аналізу якості резюме з використанням методів обробки природної мови

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра програмного забезпечення, ФІТКІ, ЗПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 7,7 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Романюк О. В., доцент кафедри ПЗ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Ковальський В. А.
(підпис) (прізвище, ініціали)

Додаток В. Лістинг програми

main.js

```
import express from "express";
import cors from "cors";
import  dotenv from "dotenv";
import { fileURLToPath} from "url";
import { dirname } from "path";
import morgan from "morgan";
import helmet from "helmet";
import compression from "compression";
import { errorHandler } from "./middleware/errorHandler.middleware.js";
import { publicRouter } from "./routes/public.router.js";
import { protectedRouter } from "./routes/protected.router.js";
import { connectDb } from "./config/database.config.js";
import { initBrowser } from "./services/scraper.service.js";
import { initGemini } from "./services/gpt.service.js";


dotenv.config()
const _filename = fileURLToPath(import.meta.url);
const _dirname = dirname(_filename);
connectDb();
initBrowser();
initGemini();


const app = express()


app.use(cors());
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(morgan("dev"));
app.use(helmet());
app.use(compression());
app.use('/api', publicRouter, protectedRouter);
app.use(errorHandler)


const PORT = process.env.SERVER_PORT || 5000;
app.listen(PORT, () => {
   console.log(`Server running on http://localhost:${PORT}`)
})
```

errorHandler.middleware.js

```
export function errorHandler(error, req, res, next) {
   const statusCode = error.statusCode || 500;
   const message = error.message || "Internal server error";
   res.status(statusCode).json({ error: message });
}
```

Vite.config.ts

```
import { defineConfig } from 'vite'
import vue from '@vitejs/plugin-vue'
import { fileURLToPath, URL } from 'node:url'


// https://vite.dev/config/
export default defineConfig({
  server: {
    proxy: {
      '/api': {
        target: 'http://localhost:5000',
        changeOrigin: true
      }
    }
  },
  plugins: [vue()],
  resolve: {
    alias: {
      '@': fileURLToPath(new URL('./src', import.meta.url)),
      '@components': fileURLToPath(new URL('./src/components', import.meta.url)),
      '@views': fileURLToPath(new URL('./src/views', import.meta.url)),
      '@layouts': fileURLToPath(new URL('./src/layouts', import.meta.url)),
      '@constants': fileURLToPath(new URL('./src/utils/constants', import.meta.url)),
    }
  },
  css: {
    preprocessorOptions: {
      scss: {
        additionalData: '@import "@/assets/styles/variables.scss";'
      }
    }
  }
})
```

textAnalysis.js

```
import { initGemini, executePrompt } from './gemini.js';
import stringSimilarity from 'string-similarity';


export async function analyzeGlobalCtx(resumeText) {
  const prompt = [
    {
      role: 'user',
      parts: [{
        text: `Perform a concise thematic and stylistic analysis of the following resume.


              Return ONLY JSON with:
{
  "themes": [ "string", ... ],
  "style": "string",
  "language": "<language_code>", // example: "en", "uk", "de", "fr"
}
```

```
Resume:
"""${resumeText}"""`,
      }]
    }
  ];
  const result = await executePrompt(prompt);
  return JSON.parse(result.response.text());
}


export function tokenizeSections(sections) {
  result = [];
  for (const section of sections) {
    const sectionSentences = section.split(/[.!?]\s+/).filter(Boolean);
    result.push(...sectionSentences);
  }
  return result;
}


export async function analyzeSentenceQuality(sentence) {
  const prompt = [
    {
      role: 'user',
      parts: [{
        text: `Evaluate the following resume sentence for relevance, informativeness, and style.


Return ONLY JSON:
{
  "relevance": 0-10,
  "informativeness": 0-10,
  "styleScore": 0-10,
  "comment": "string"
}


Sentence:
"${sentence}"`,
      }]
    }
  ];
  const result = await executePrompt(prompt);
  return JSON.parse(result.response.text());
}


export async function analyzeGrammar(sentence, lang="en") {
  const prompt = [
    {
      role: 'user',
      parts: [{
        text: `Check the following sentence for grammar issues. Sentence language code: ${lang}
```

```
Return ONLY JSON:
{
  "hasErrors": true|false,
  "issues": [
    { "error": string, "suggestion": "string|null"}, ...
  ],
}


Sentence:
"${sentence}"`,
      }]
    }
  ];
  const result = await executePrompt(prompt);
  return JSON.parse(result.response.text());
}


export function analyzeSimilarity(sentA, sentB) {
  const similarity = stringSimilarity.compareTwoStrings(sentA, sentB);
  return {
    similarity,
    tooSimilar: similarity >= 0.8
  };
}


export async function analyzeResume(resume) {
  await initGemini();
  const globalCtx = await analyzeGlobalCtx(resume.text);
  const tokenizedSections = tokenizeSections(resume.sections);
  const report = {
    language: globalCtx.language,
    themes: globalCtx.themes,
    style: globalCtx.style,
    sentences: []
  };
  for (const sentence of tokenizedSections) {
    const quality = await analyzeSentenceQuality(sentence);
    const grammar = await analyzeGrammar(sentence, report.language);
    report.sentences.push({
      sentence,
      quality,
      grammar
    });
  }
  for (let i = 0; i < sentences.length; i++) {
    for (let j = i + 1; j < sentences.length; j++) {
      const { similarity, tooSimilar } = analyzeSimilarity(sentences[i], sentences[j]);
      if (tooSimilar) {
        report.sentences.push({
          similarityWarning: {
            sentenceA: sentences[i],
            sentenceB: sentences[j],
            similarity
          }
```

```
        });
      }
    }
  }
  return report;
}
```

scraper.service.js

```
import { launch, Browser} from 'puppeteer';


/**
 *  @type {Browser}
 */
let browser;


export async function initBrowser() {
    try {
        browser = await launch();
    } catch (e) {
        console.error(e);
        throw e;
    }
}


export async function getVacanciesByTitle(title, source, limit=5) {
    try {
        const ctx = await browser.createBrowserContext();
        const page = await ctx.newPage();
        let result;
        if(source === "djinni") {
            result = await getDjinniVacancies(title, page, limit)
        } else if (source === "linkedin") {
            result = await getLinkedinVacancies(title, page, limit)
        }
        return result;
    } catch (e) {
        console.error(e);
    }
}


async function getDjinniVacancies (title, page, limit) {
    const params = title.split(' ').join('%20');
    await page.goto(`https://djinni.co/jobs/?all_keywords=${params}`,
        { waitUntil: 'networkidle2' });
    const vacancyDescriptions = await page.evaluate(() => {
        return Array.from(document.querySelectorAll('.js-original-text'))
        .slice(0, limit)
        .map(el => el.textContent);
    })
    return vacancyDescriptions;
}
```

```
export async function closeBrowser() {
    try {
        await browser.close();
    } catch (e) {
        console.error(e);
        throw e;
    }
}
export async function getLinkedinVacancies(title, page, limit) {
    const searchQuery = title.trim().split(' ').join('%20');
    const searchUrl = `https://www.linkedin.com/jobs/search/?keywords=${searchQuery}`;

    await page.goto(searchUrl, { waitUntil: 'networkidle2' });
    await page.waitForSelector('.base-search-card', { timeout: 10000 });


    const jobCards = await page.$$('.base-search-card');
    const jobDescriptions = [];


    for (let i = 0; i < Math.min(limit, jobCards.length); i++) {
        try {
            await jobCards[i].click();
            await page.waitForTimeout(1000);
            await page.waitForSelector('.jobs-description-content__text',
                { timeout: 5000 });
            const fullDescription = await page.evaluate('.jobs-description-content__text',
                el => el.textContent.trim());
            jobDescriptions.push(fullDescription);
        } catch (e) {
            console.warn(`Cannot parse vacancy ${i}:`, e.message);
        }
    }


    return jobDescriptions;
}
```

vacancyRelevancy.service.js

```
import { executePrompt } from './gemini.js'

export async function evaluateResumeRelevance(resumeText, jobTitle) {


    const languageCode = JSON.parse(langResponse.response.text()).code;
    const resume = {
        text: resumeText,
        language: languageCode,
    };
```

```
    const vacancies = await scrapeVacancies();
    const relevanceResults = [];


    for (const vacancy of vacancies) {
        const relevancePrompt = [
            {
                role: 'user',
                parts: [
                    {
                        text: `You are an expert HR system.
                        Evaluate how relevant the following resume is to the job description below.
                        Provide a numeric score from 0 to 100 and a short structured explanation in English.


                        Return response in this JSON format:
                        {
                            "score": number,
                            "explanation": string
                        }
                        RESUME:
                        ${resume.text}
                        VACANCY:
                        ${vacancy.description}`
                    }
                ]
            }
        ];


        const response = await executePrompt(relevancePrompt);
        const parsed = JSON.parse(response.response.text());


        relevanceResults.push({
            vacancyId: vacancy.id,
            score: parsed.score,
            explanation: parsed.explanation,
        });
    }


    const averageRelevance =
        relevanceResults.reduce((sum, r) => sum + r.score, 0) /
        relevanceResults.length;


    return {
        averageRelevance: averageRelevance.toFixed(2),
        results: relevanceResults,
    };
}
```

gpt.service.js

```
import { GoogleGenAI,  } from '@google/genai'
import { config } from 'dotenv'
```

```
config();


/**
 * @type {GoogleGenAI}
 */
let googleAI;


export async function initGemini() {
    try {
        googleAI = new GoogleGenAI({ apiKey: process.env.GEMINI_API_KEY });
    } catch (e) {
        console.error(e)
        throw e;
    }
}


export async function executePrompt (prompt) {
    try {
        const result = await googleAI.models.generateContent({
            model: 'gemini-2.0-flash-lite',
            contents: prompt,
        })
        return result;
    } catch (e) {
        console.error(e)
        throw e;
    }
}
```

auth.service.js

```
import { compare, hash } from "bcryptjs";
import jwt from 'jsonwebtoken';
import { UserModel } from "../models/user.model.js";
import { ApiError } from "../errors.js/apiError.js";
import { validateEmail, validatePassword } from "./validation.js";


export async function registerUser(userInfo) {
    const { email, password } = userInfo;


    if(!validateEmail(email)) throw new ApiError('Invalid email', 400);
    if(!validatePassword(password)) throw new ApiError('Invalid password, must contain 8 characters, one uppercase letter, on lowercase letter and one digit', 400);


    const passwordHash = await hash(password, 10);
    const isAvailableEmail = await UserModel.findOne({ email }).exec();
    if(isAvailableEmail) throw new ApiError('Email already in use', 409);
```

```
    const user = new UserModel({ email, password: passwordHash });
    await user.save();
    const token = jwt.sign({
        id: user._id,
        email: user.email,
    }, process.env.JWT_SECRET, { expiresIn: '72h' });

    return token;
}


export async function login(credentials) {
    const { email, password } = credentials;
    console.log(credentials)
    const user = await UserModel.findOne({ email }).exec();
    console.log(user);
    if(!user) throw new ApiError('User not found', 404);


    const isValidPassword = await compare(password, user.password);
    if(!isValidPassword) throw new ApiError('Wrong password', 401);


    const token = jwt.sign({
        id: user._id,
        email: user.email,
    }, process.env.JWT_SECRET, { expiresIn: '72h' });


    return token;
}


export async function authenticateToken(token) {
    return jwt.verify(token, process.env.JWT_SECRET);
}
```

auth.controller.js

```
import { ApiError } from "../errors.js/apiError.js";
import { login, registerUser } from "../services/auth.service.js";


export async function registerController (req, res, next){
    const { email, password } = req.body;
    if(email && password) {
        try {
            const token = await registerUser(req.body);
            res.status(200).send({ token });
        } catch (error) {
            next(error)
        }
    } else {
        next(new ApiError("Missing email or password", 400));
    }
```

```
}


export async function loginController (req, res, next) {
   const { email, password } = req.body;
   if(email && password) {
      try {
         const token = await login(req.body);
         res.status(201).json({ token });
      } catch (error) {
         next(error)
      }
   } else {
      next(new ApiError("Missing email or password", 400));
   }
}
```

auth.middleware.js

```
import { authenticateToken } from "../services/auth.service.js";


export async function userAuthenticate (req, res, next) {
   const token = req.get('authorization');


   if(!token) {
      res.status(401).json({ error: "No token" });
   }


   try {
      const user = await authenticateToken(token);
      req.user = user;
      next();
   } catch (error) {
      next(error);
   }
}
```

FilePreview.vue

```
<template>
   <iframe ref="iframe" class="file-preview" scrolling="no">
   </iframe>
</template>
<script setup lang="ts">
import { onMounted, onUnmounted, ref } from 'vue';


interface Props {
   file: string;
}


const props = defineProps<Props>();
const iframe = ref<HTMLIFrameElement | null>(null);
```

```
const resizeObserver = new ResizeObserver((entries) => {
    for (let entry of entries) {
        if (entry.target === iframe.value) {
            const doc = iframe.value?.contentDocument || iframe.value?.contentWindow?.document;
            if (doc) {
                doc.documentElement.style.fontSize = entry.contentRect.height / 1000 + 'rem';
            }
        }
    }
});


onMounted(() => {
    if (iframe.value) {
        const doc = iframe.value.contentDocument || iframe.value.contentWindow?.document;
        if (doc) {
            doc.open();
            doc.write(props.file);
            doc.close();
            doc.documentElement.style.fontSize = iframe.value.clientHeight / 1000 + 'rem';
        }
        resizeObserver.observe(iframe.value);
    }
})
onUnmounted(() => {
    resizeObserver.disconnect();
})
</script>
```

ResumeList.vue

```
<template>
  <div class="resume-list">
    <router-link to="/templates" class="resume-list__create">
      +
    </router-link>
    <ResumeListItem
      v-for="resume in resumes"
      :key="resume.id"
      :content="resume.html"
    />
  </div>
</template>

<script setup lang="ts">
import { ref, onMounted } from 'vue'
import ResumeListItem from './ResumeListItem.vue'

interface Resume {
  id: string
  html: string
}

const resumes = ref<Resume[]>([])
```

```
const fetchResumes = async () => {
  try {
    const response = await fetch('/api/resumes')
    if (!response.ok) throw new Error('Failed to fetch')
    const data = await response.json()
    resumes.value = data
  } catch (error) {
    console.error('Error fetching resumes:', error)
  }
}

onMounted(fetchResumes)
</script>
```

ResumeEditor.vue

```
<template>
  <div class="editor">
    <div class="editor-forms">
      <div class="editor-forms-actions">
        <button class="button" @click="addExperience">{{ $t("Створити секцію") }}</button>
        <button class="button" @click="removeLastExperience" :disabled="experience.length === 0">
          {{ $t("Видалити секцію") }}
        </button>
        <button class="button" @click="overlay = true">
          {{ $t("Змінити порядок секцій") }}
        </button>
      </div>

      <div class="editor-forms-form">
        <h2>Досвід</h2>
        <div v-for="(exp, index) in experience" :key="index" class="experience-section">
          <h3>Секція {{ index + 1 }}</h3>
          <div class="input-row">
            <div class="input">
              <label>Назва посади</label>
              <input v-model="exp.position" class="input__field" />
            </div>
            <div class="input">
              <label>Компанія</label>
              <input v-model="exp.company" class="input__field" />
            </div>
            <div class="input">
              <label>Період</label>
              <input v-model="exp.period" class="input__field" />
            </div>
          </div>
          <div class="input">
            <label>Опис досягнень</label>
            <textarea v-model="exp.description" class="input__field"></textarea>
          </div>
          <hr />
        </div>

        <button class="button" @click="addExperience">Додати секцію</button>
      </div>
```

```
    </div>

    <div class="editor-preview">
      <FilePreview :file="generatedPreview" class="resume-list-item__iframe" />
    </div>

    <div v-if="overlay" class="overlay">
      <div class="overlay-container">
        <draggable v-model="experience" item-key="id">
          <template #item="{ element, index }">
            <div class="drag-item">
              <span>{{ element.position || 'Посада ' + (index + 1) }}</span>
            </div>
          </template>
        </draggable>
        <button class="button" @click="overlay = false">Зберегти порядок</button>
      </div>
    </div>
  </div>
</template>

<script lang="ts" setup>
import { ref, computed, watch } from 'vue';
import FilePreview from '@/components/FilePreview.vue';
import { nanoid } from 'nanoid';
import draggable from 'vuedraggable';

const overlay = ref(false);

interface Experience {
  id: string;
  position: string;
  company: string;
  period: string;
  description: string;
}

const experience = ref<Experience[]>([
  {
    id: nanoid(),
    position: '',
    company: '',
    period: '',
    description: '',
  },
]);

function addExperience() {
  experience.value.push({
    id: nanoid(),
    position: '',
    company: '',
    period: '',
    description: '',
  });
}
```

```
function removeLastExperience() {
  if (experience.value.length > 0) {
    experience.value.pop();
  }
}

const generatedPreview = computed(() => {
  const expSections = experience.value
    .map(
      (e) => `
        <p><strong>${e.position}</strong> — ${e.company} (${e.period})</p>
        <ul><li>${e.description}</li></ul>
      `
    )
    .join('');

  return `
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <title>Резюме</title>
  <style>
    body { font-family: Arial, sans-serif; padding: 2rem; line-height: 1.6; }
    h1 { font-size: 2rem; }
    h2 { margin-top: 2rem; border-bottom: 1px solid #ccc; }
  </style>
</head>
<body>
  <h1>Валентин Ковальський</h1>
  <h2>Досвід</h2>
  ${expSections || '<p>Немає досвіду</p>'}
</body>
</html>
`;
});
</script>

<style scoped>
.editor {
  display: flex;
  gap: 2rem;
  padding: 2rem;
  align-items: flex-start;
}

.editor-forms,
.editor-preview {
  flex: 1;
}

.input-row {
  display: flex;
  gap: 1rem;
  margin-bottom: 1rem;
}
```

```
.input {
  display: flex;
  flex-direction: column;
  flex: 1;
}

.input__field {
  padding: 0.5rem;
  font-size: 1rem;
  border: 1px solid #ccc;
  border-radius: 0.25rem;
}

.button {
  padding: 0.5rem 1rem;
  background-color: #004f7c;
  color: white;
  border: none;
  border-radius: 0.25rem;
  cursor: pointer;
  margin-top: 1rem;
}

.button:disabled {
  background-color: #888;
}

.overlay {
  position: fixed;
  inset: 0;
  background: rgba(0, 0, 0, 0.5);
  display: flex;
  justify-content: center;
  align-items: center;
}

.overlay-container {
  background: #fff;
  padding: 2rem;
  border-radius: 1rem;
  width: 400px;
}

.drag-item {
  padding: 1rem;
  background-color: #eee;
  border: 1px solid #ccc;
  border-radius: 0.25rem;
  margin-bottom: 0.5rem;
  cursor: grab;
}
</style>
```

ResumeAnalysis.vue

```
<template>
  <div class="analysis">
```

```
    <div class="analysis__resume-text">
      <div class="analysis__resume-text__header">
        <h2>Текст резюме</h2>
        <button class="button" @click="analyzeResume">Аналізувати</button>
      </div>
      <p class="analysis__resume-text__text" v-html="highlightedResume"></p>
    </div>

    <div class="analysis__relevancy">
      <h3>Аналіз за вакансіями</h3>
      <div class="input">
        <label class="input__label" for="position">Посада</label>
        <input class="input__field" type="text" id="position" v-model="targetPosition">
        <span class="input__error">{{ errorText }}</span>
      </div>

      <div class="input" v-for="(source, index) in sources" :key="index" style="display: flex; gap: 1rem;">
        <label class="input__label">{{ source }}</label>
        <input class="input__field" type="radio" :value="source" v-model="selectedSource">
      </div>

      <button class="button" @click="analyzeVacancies">Аналізувати за вакансіями</button>
      <hr>
      <h3>Результат</h3>
      <ol>
        <li v-for="(vacancy, index) in analyzedVacancies" :key="index">
          <a :href="vacancy.link" target="_blank">{{ vacancy.title }}</a> —
          <span :style="{ color: vacancy.color, fontWeight: '600' }">{{ vacancy.score }}%</span>
        </li>
      </ol>
      <h3>
        Середній рівень релевантності:
        <span :style="{ color: averageColor, fontWeight: '600' }">{{ averageScore.toFixed(1) }}%</span>
      </h3>
    </div>

    <div class="analysis__comments">
      <h2>Результат</h2>
      <p>Мова: {{ language }}</p>
      <p>Стиль: {{ style }}</p>
      <p>Ключові слова: {{ keywords.join(', ') }}</p>

      <h4>Граматичні помилки:</h4>
      <p v-html="formattedArray(grammarMistakes)"></p>

      <h4>Стилістичні помилки:</h4>
      <p v-html="formattedArray(styleMistakes)"></p>

      <h4>Порожні фрази:</h4>
      <p v-html="formattedArray(emptyPhrases)"></p>

      <h4>Повтори:</h4>
      <p v-html="formattedArray(repetitions)"></p>
    </div>
  </div>
</template>
```

```
<script setup>
import { ref, computed } from 'vue'

const resumeText = ref(`Kovalskyi Valentyn JavaScript Developer Vinnytsia kovalsky.valentine@gmail.com
...`)
const highlightedResume = ref('')

const targetPosition = ref('')
const selectedSource = ref('')
const errorText = ref('\u00A0')

const sources = ['Djinni', 'LinkedIn']

const language = ref('')
const style = ref('')
const keywords = ref([])

const grammarMistakes = ref([])
const styleMistakes = ref([])
const emptyPhrases = ref([])
const repetitions = ref([])

const analyzedVacancies = ref([])

const averageScore = computed(() => {
  if (analyzedVacancies.value.length === 0) return 0
  return analyzedVacancies.value.reduce((sum, v) => sum + v.score, 0) / analyzedVacancies.value.length
})

const averageColor = computed(() => {
  if (averageScore.value >= 80) return 'green'
  if (averageScore.value >= 50) return 'orange'
  return 'darkred'
})

function formattedArray(arr) {
  return arr.map(item => item + '<hr>').join('')
}

function getColorByScore(score) {
  if (score >= 80) return 'green'
  if (score >= 50) return 'orange'
  return 'darkred'
}

async function analyzeResume() {
  try {
    const response = await fetch('/api/analyze-resume', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ text: resumeText.value })
    })

    if (!response.ok) throw new Error('Помилка аналізу резюме')

    const result = await response.json()
```

```
    language.value = result.language
    style.value = result.style
    keywords.value = result.keywords
    grammarMistakes.value = result.grammarMistakes
    styleMistakes.value = result.styleMistakes
    emptyPhrases.value = result.emptyPhrases
    repetitions.value = result.repetitions

    highlightedResume.value = highlightWithColors(resumeText.value, result.highlightedWords)
  } catch (err) {
    console.error(err)
  }
}

async function analyzeVacancies() {
  if (!targetPosition.value || !selectedSource.value) {
    errorText.value = 'Вкажіть посаду та джерело'
    return
  }

  try {
    const response = await fetch('/api/vacancy-relevance', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        position: targetPosition.value,
        source: selectedSource.value,
        resume: resumeText.value
      })
    })

    if (!response.ok) throw new Error('Помилка аналізу вакансій')

    const result = await response.json()

    analyzedVacancies.value = result.vacancies.map(v => ({
      ...v,
      color: getColorByScore(v.score)
    }))
  } catch (err) {
    console.error(err)
  }
}

function highlightWithColors(text, wordMap) {
  let result = text
  for (const [color, words] of Object.entries(wordMap)) {
    words.forEach(word => {
      const regex = new RegExp(`(${word})`, 'gi')
      result = result.replace(regex, `<span class="highlight-${color}">$1</span>`)
    })
  }
  return result.replace(/\n/g, '<br>')
}
</script>

<style scoped lang="scss">
```

```
.analysis {
  display: grid;
  grid-template-columns: 3fr 1fr;
  grid-template-rows: auto auto;
  gap: 1rem;
  padding: 2rem;

  &__resume-text,
  &__relevancy,
  &__comments {
    background-color: #f4f4f4;
    padding: 1rem;
    border-radius: 1rem;
  }

  &__resume-text__text {
    background: white;
    padding: 1rem;
    border-radius: 0.5rem;
    overflow-y: auto;
  }
}

.highlight-green {
  background-color: lightgreen;
}
.highlight-orange {
  background-color: lightyellow;
}
.highlight-darkred {
  background-color: rgba(255, 0, 0, 0.2);
}
</style>
```

style.scss

```
@import "./themes.scss";
@import "./mixins.scss";
@import "./components.scss";


*, *::before, *::after {
    box-sizing: border-box;
}


body {
    margin: 0;
    background-color: var(--bg-color);
    min-width: fit-content;
}


p, h1, h2, h3, h4, h5, h6 {
    margin: 0;
}
```

```
button {
    appearance: none;
    border: none;
    font-size: 1rem;
}


#app {
    height: 100vh;
    width: 100vw;
}


.auth-form {
    height: 100%;
    display: flex;
    width: 100%;
    align-items: center;
    justify-content: center;
    &__header {
        align-self: center;
    }
    &__form {
        width: 25%;
        height: fit-content;
        font-weight: 500;
        background-color: var(--container-bg-color);
        padding: 2em 3em;
        display: flex;
        flex-direction: column;
        align-items: center;
        border-radius: 1em;
        gap: 1em;
    }
    &__input {
        display: flex;
        flex-direction: column;
        gap: 0.25em;
        width: 100%;
        &__label {
            color: var(--container-font-color);
        }
        &__field {
            padding: 1em;
            border: none;
            border-radius: 0.5em;
        }
        &__error {
            color: var(--error-color);
            font-size: 0.8em;
            @include text-ellipsis;
        }
    }
}
```

```
.input {
   display: flex;
   flex-direction: column;
   gap: 0.25em;
   width: 100%;
   &__label {
      color: var(--container-font-color);
   }
   &__field {
      padding: 1em;
      border: none;
      border-radius: 0.5em;
   }
   &__error {
      color: var(--error-color);
      font-size: 0.8em;
      @include text-ellipsis;
   }
}



.button {
   display: inline-block;
   background-color: var(--btn-bg-color);
   color: var(--btn-font-color);
   text-decoration: none;
   font-weight: 500;
   padding: 0.5em 1em;
   border-radius: .5em;
   transition: all .2s ease-in-out;
   &:hover {
      filter: brightness(1.1);
      transform: scale(1.03);
   }
}



.dashboard {
   height: 100%;
   width: 100%;
   display: flex;
   flex-direction: column;
   align-items: start;
   padding: 2em 3em;
   &__title {
      padding: 1em 0;
   }
}


.resume-list {
   display: flex;
   flex-direction: row;
```

```
    align-self: start;
    flex-wrap: wrap;
    height: 100%;
    width: 100%;
    gap: 2em;
& > * {
        height: 40%;
    }
    &-item {
        position: relative;
        transition: all 0.2s ease-in-out;
        &__iframe {
            height: 100%;
            aspect-ratio: 1/1.414;
            border: none;
            pointer-events: none;
            overflow: hidden;
        }
        &:hover {
            filter: drop-shadow(0 0 0.5em var(--container-font-color));
            transform: translateY(-0.5em);
            cursor: pointer;
        }
        &__info {
            position: absolute;
            top: 0;
            bottom: 0;
            height: 100%;
            width: 100%;
            opacity: 0;
            display: flex;
            flex-direction: column;
            justify-content: end;
            transition: all 0.2s ease-in-out;
            padding: 2.5em 1em;
            & > * {
                filter: invert(1)
            }
            &:hover {
                opacity: 1;
                background: #000000;
                background: -webkit-linear-gradient(0deg,rgba(0, 0, 0, 1) 0%, rgba(0, 0, 0, 0.44) 49%, rgba(255,
255, 255, 0) 100%);
                background: -moz-linear-gradient(0deg,rgba(0, 0, 0, 1) 0%, rgba(0, 0, 0, 0.44) 49%, rgba(255, 255,
255, 0) 100%);
                background: linear-gradient(0deg,rgba(0, 0, 0, 1) 0%, rgba(0, 0, 0, 0.44) 49%, rgba(255, 255, 255,
0) 100%);
                filter: progid:DXImageTransform.Microsoft.gradient(
                startColorstr="#000000",
                endColorstr="#FFFFFF",
                GradientType=0
                );
            }
            &__title {
                font-size: 1.25rem;
                font-weight: 500;
                color: var(--primary-font-color);
```

```
            @include text-ellipsis;
        }
        &__last-edit {
            font-weight: 400;
            color: var(--secondary-font-color);
        }


    }
  }
  &__create {
    aspect-ratio: 1/1.414;
    border: 3px solid var(--btn-bg-color);
    color: var(--btn-bg-color);
    border-radius: 0.5em;
    display: flex;
    align-items: center;
    justify-content: center;
    text-decoration: none;
    font-size: 5rem;
    transition: all 0.2s ease-in-out;
    &:hover {
      filter: drop-shadow(0 0 0.5em var(--container-font-color));
      transform: translateY(-0.1em);
      cursor: pointer;
    }
  }
}


.header {
  height: 10vh;
  width: 100vw;
  display: flex;
  justify-content: space-between;
  align-items: center;
  gap: 2em;
  padding: 1rem clamp(1rem, 5vw, 5rem);
  background-color: var(--container-bg-color);
  color: var(--container-font-color);
  border-bottom: 1px solid var(--container-font-color);


  &__logo {
    display: flex;
    justify-content: start;
    flex-basis: 25%;
    height: 100%;
    width: 100%;
    background: url("logo.png") left center no-repeat;
    background-size: contain;
    transition: filter .1s ease-in-out;
    &:hover {
      cursor: pointer;
      filter: brightness(0)
    }
  }
```

```
    &__nav {
        display: flex;
        justify-content: end;
        align-items: center;
        flex-basis: 50%;
        flex-grow: 1;
        a {
            position: relative;
            text-decoration: none;
            color: var(--container-font-color);
            font-weight: 500;
            padding: 0.5em 1em;
            border-radius: .5em;
            transition: all .2s ease-in-out;
            &:hover {
                background-color: rgb(0,0,0, 0.1);
            }
        }
        & ul {
            list-style: none;
            display: flex;
            align-items: center;
            gap: 1em;
            padding: 0;
        }
    }
    &__auth {
        display: flex;
        justify-content: space-evenly;
        align-items: center;
        gap: 1em;
    }
}
```

Додаток Г. Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ АВТОМАТИЗОВАНОЇ ГЕНЕРАЦІЇ ТА АНАЛІЗУ ЯКОСТІ РЕЗЮМЕ З ВИКОРИСТАННЯМ МЕТОДІВ ОБРОБКИ ПРИРОДНОЇ МОВИ

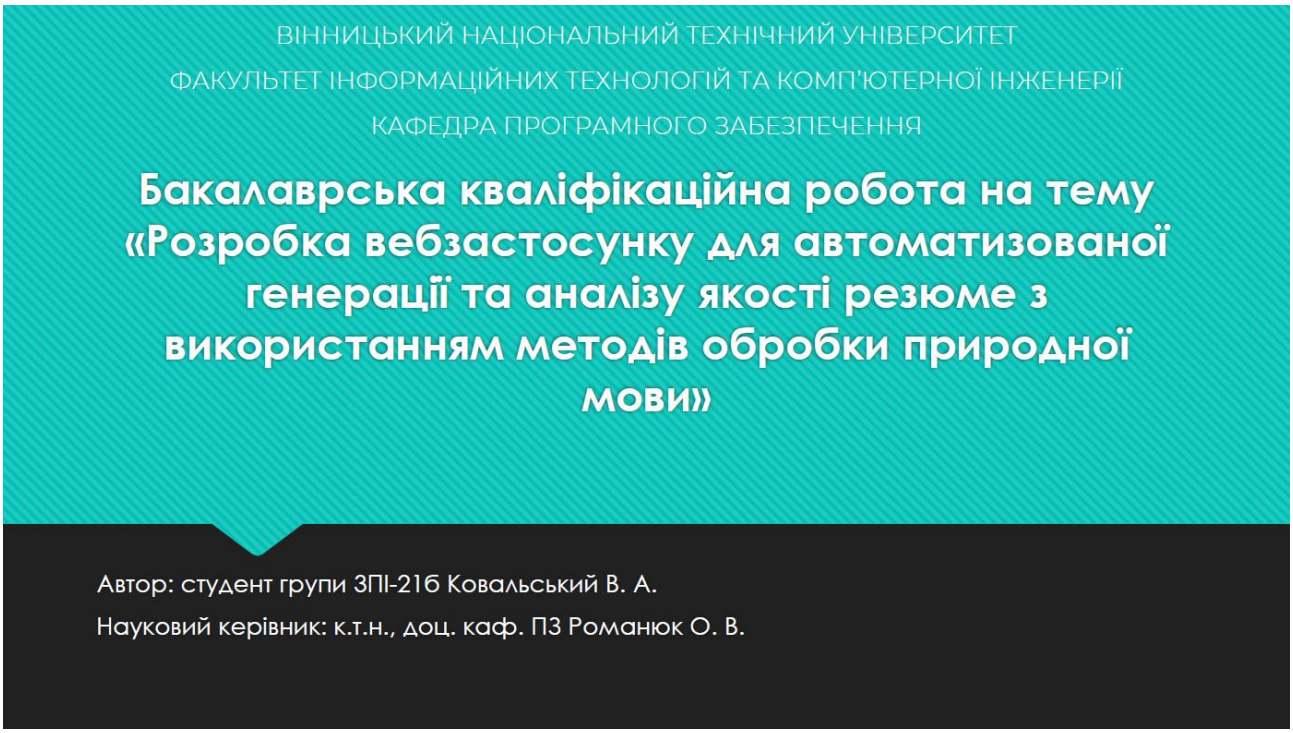

Рисунок Г.1 – Титульний слайд

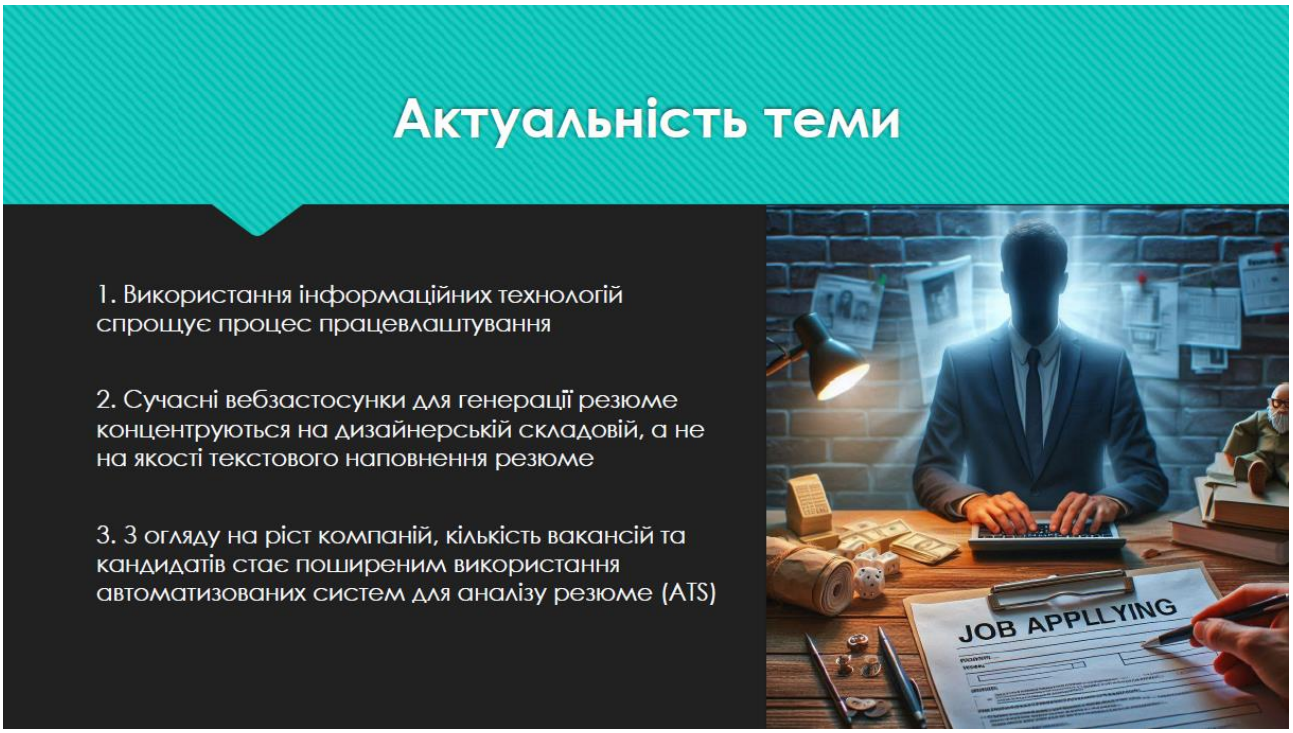

Рисунок Г.2 – Актуальність теми

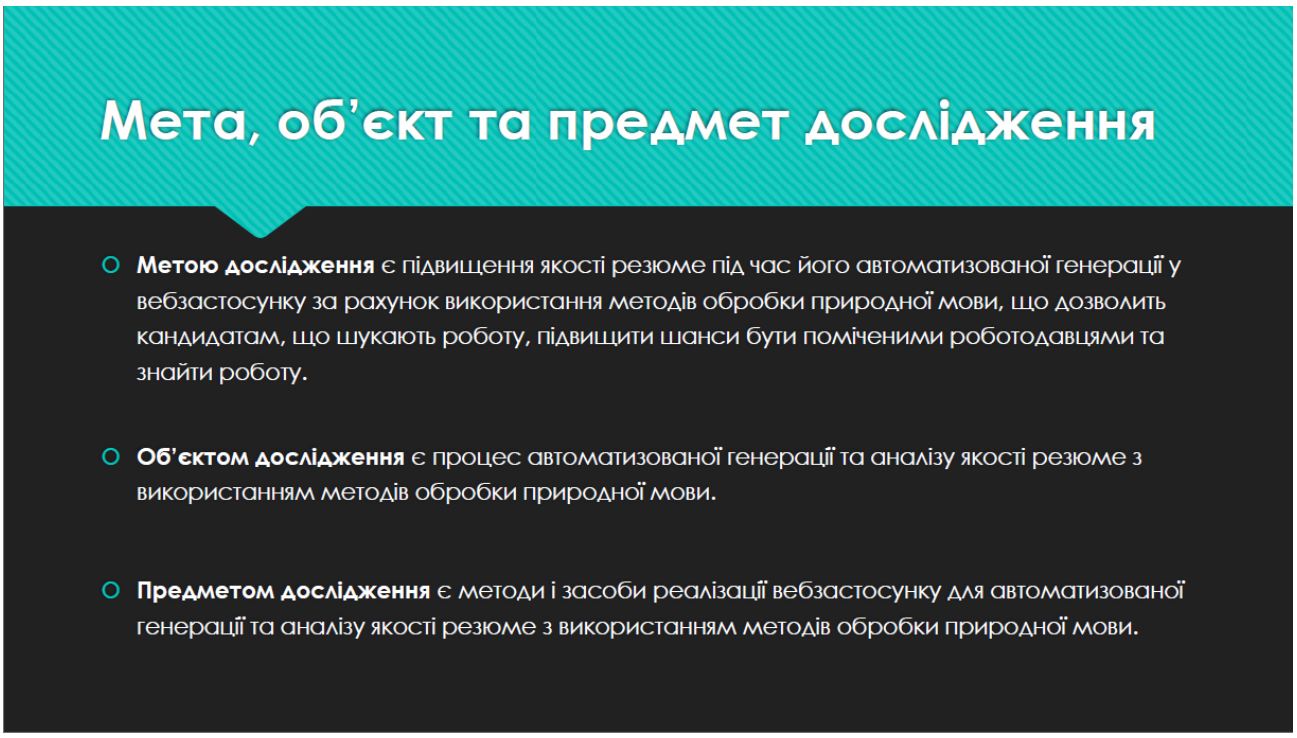

Рисунок Г.3 – Мета, об’єкт та предмет дослідження

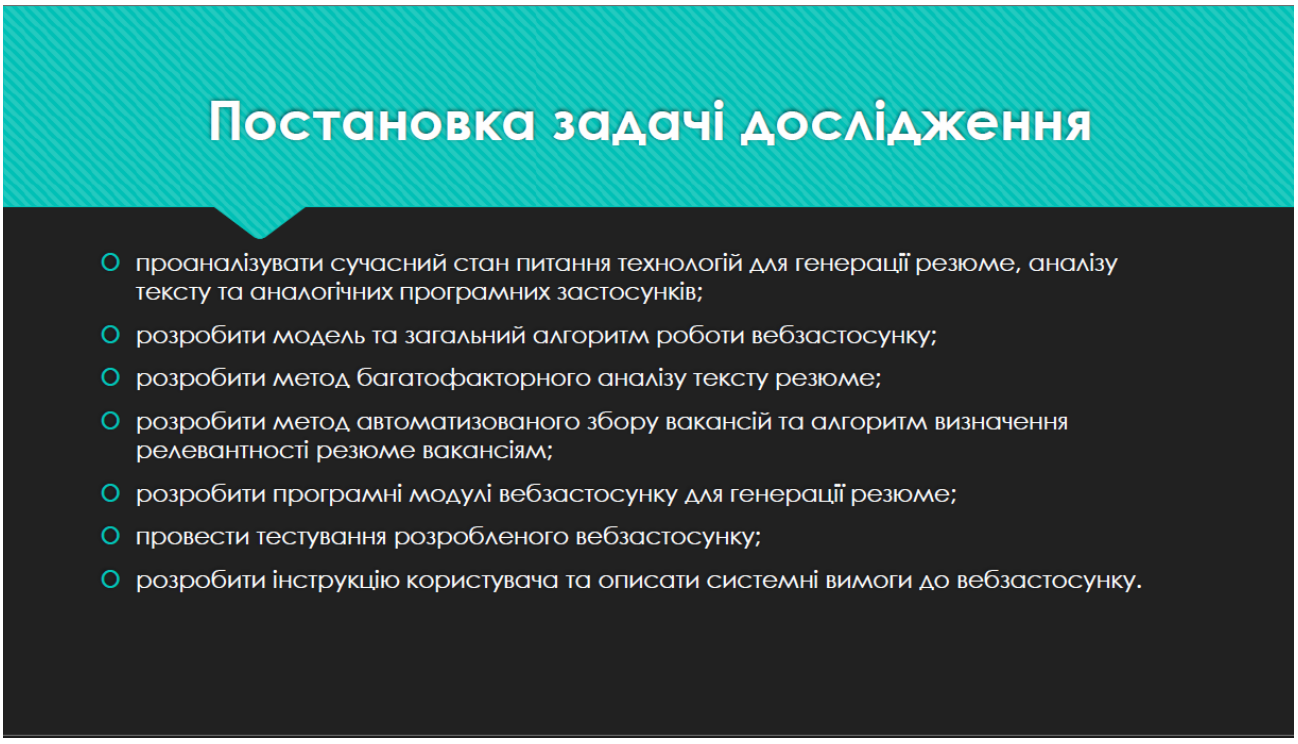

Рисунок Г.4 – Постановка задачі дослідження

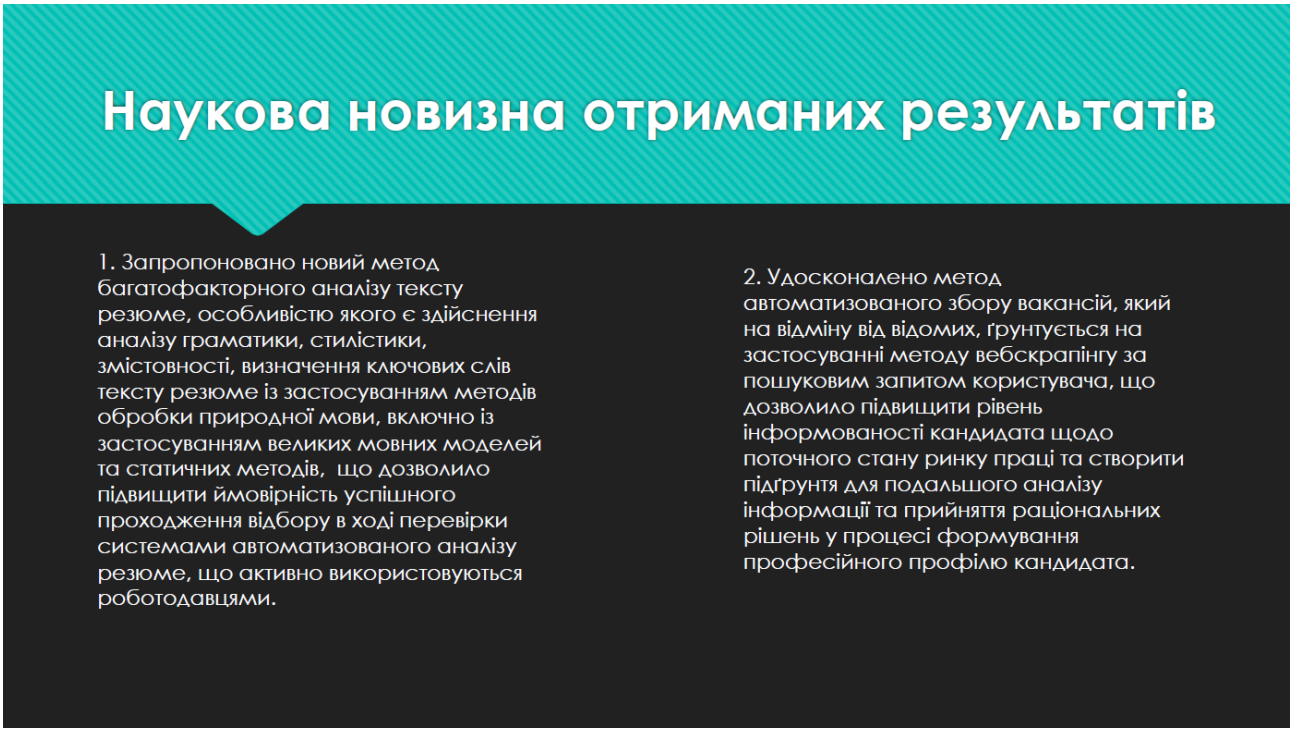

Рисунок Г.5 – Наукова новизна отриманих результатів

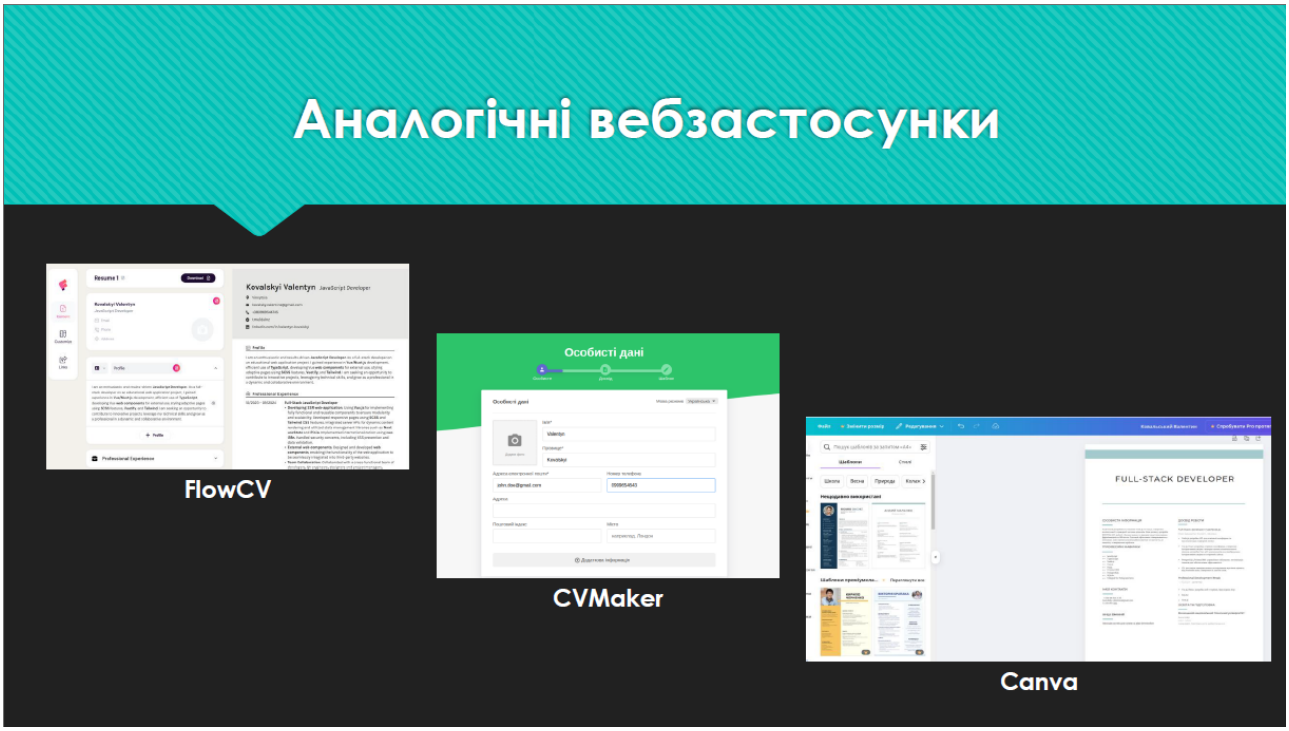

Рисунок Г.6 – Аналогічні вебзастосунки

Порівняльний аналіз аналогів

Функціональні можливості	FlowCV	Canva	CVMaker	Власна розробка
Гнучка стилізація елементів	0	1	1	0
Спільна робота над одним резюме	0	1	0	0
Використання штучного інтелекту для покращення вмісту	1	0	0	1
Динамічна зміна шаблонів	1	0	1	1
Аналіз змістовності тексту резюме	1	0	0	1
Аналіз релевантності резюме актуальним вакансіям	0	0	0	1
Сумарний коефіцієнт	3/6	2/6	2/6	4/6

Рисунок Г.7 – Порівняльний аналіз аналогів

Діаграма компонентів вебсистеми

В основі моделі системи лежить клієнт-серверна архітектура. Система розділена на декілька підсистем, що взаємодіють між собою за допомогою інтерфейсів.

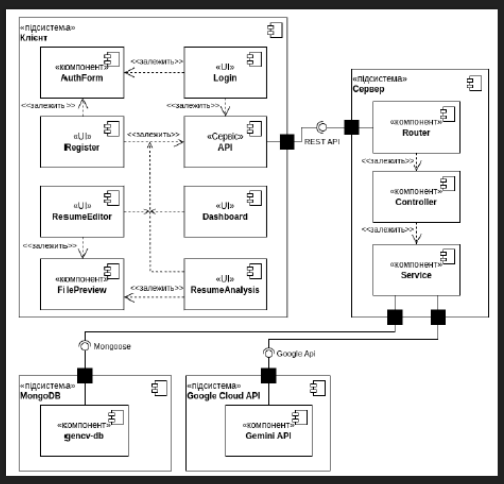

Рисунок Г.8 – Діаграма компонентів вебсистеми

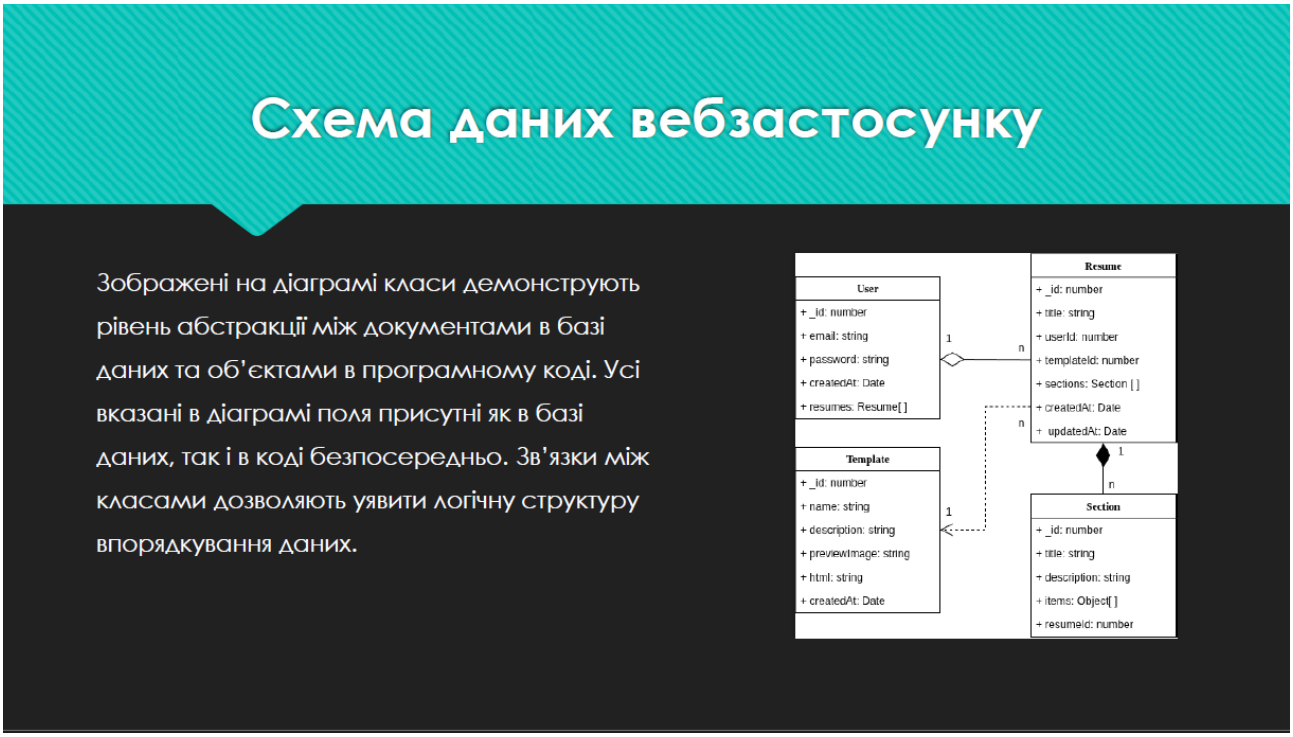

Рисунок Г.9 – Схема даних вебзастосунку

Рисунок Г.10 – Загальний алгоритм роботи системи

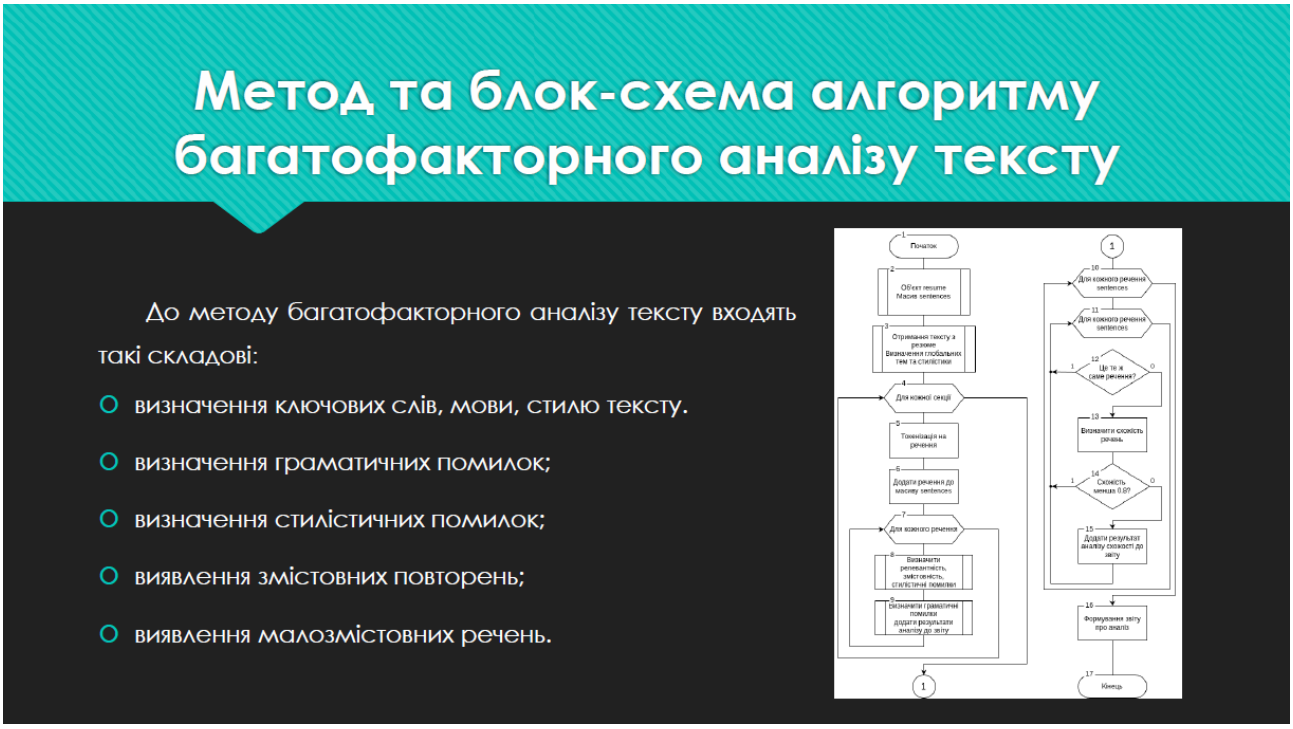

Рисунок Г.11 – Метод та блок-схема алгоритму багатофакторного аналізу тексту

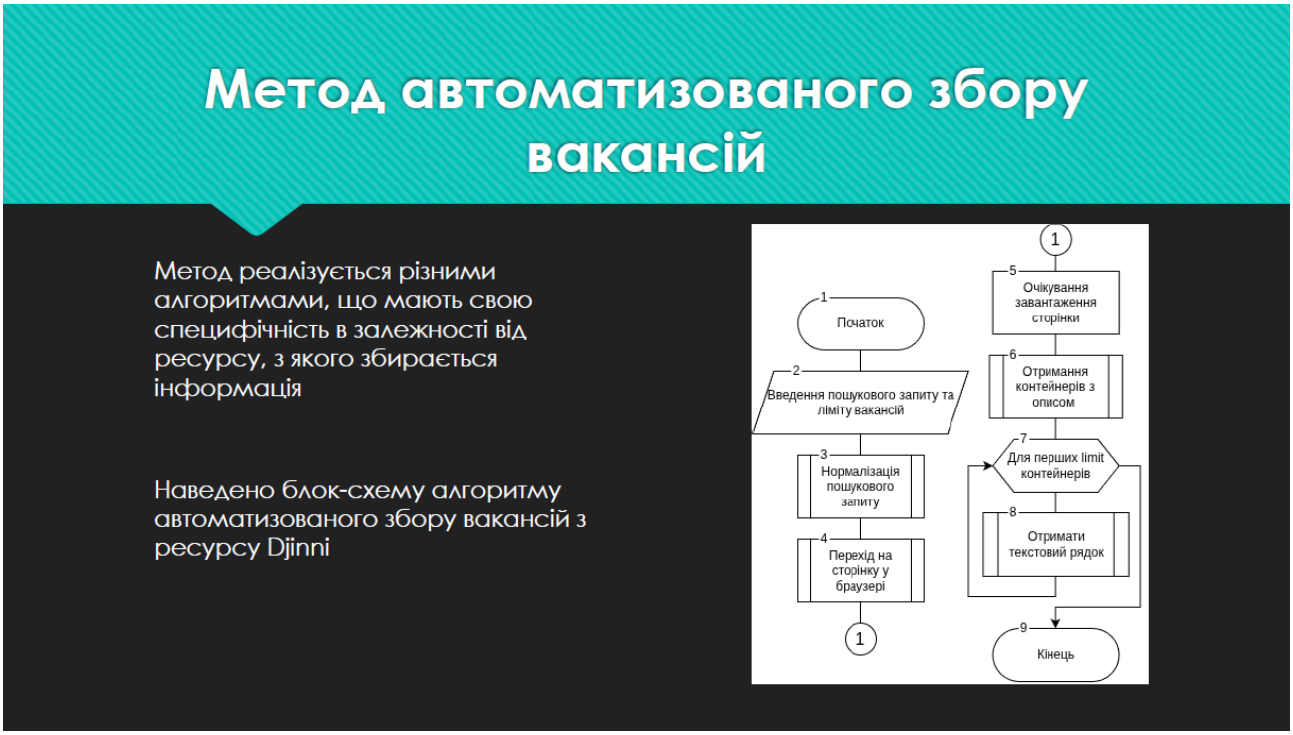

Рисунок Г.12 – Метод автоматизованого збору інформації

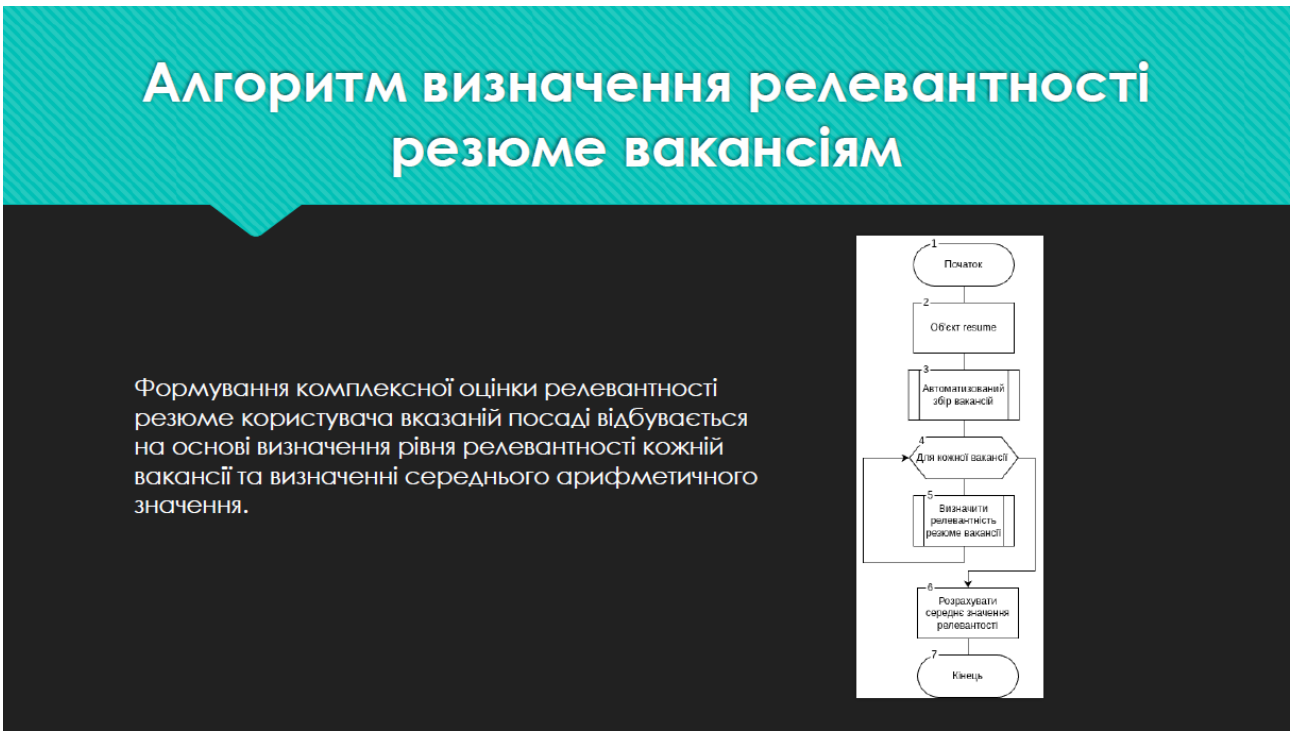

Рисунок Г.13 – Алгоритм визначення релевантності резюме вакансіям

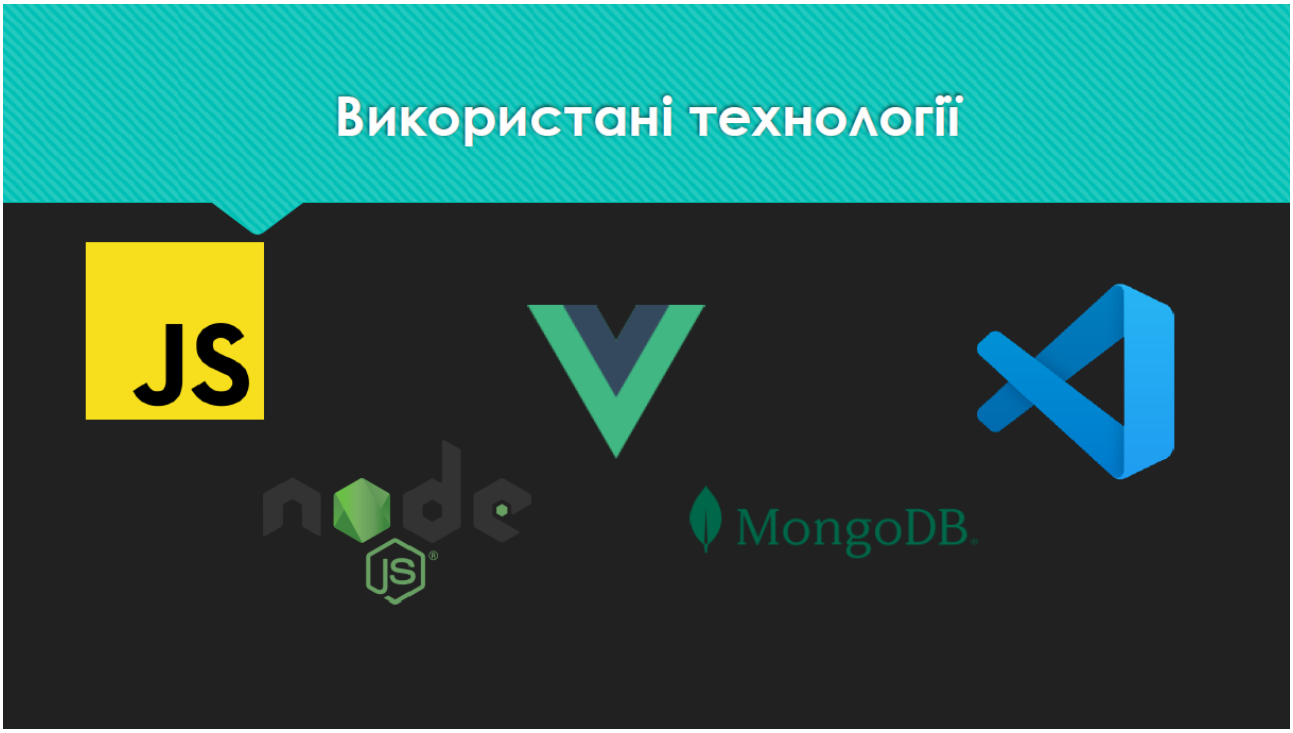

Рисунок Г.14 – Використані технології

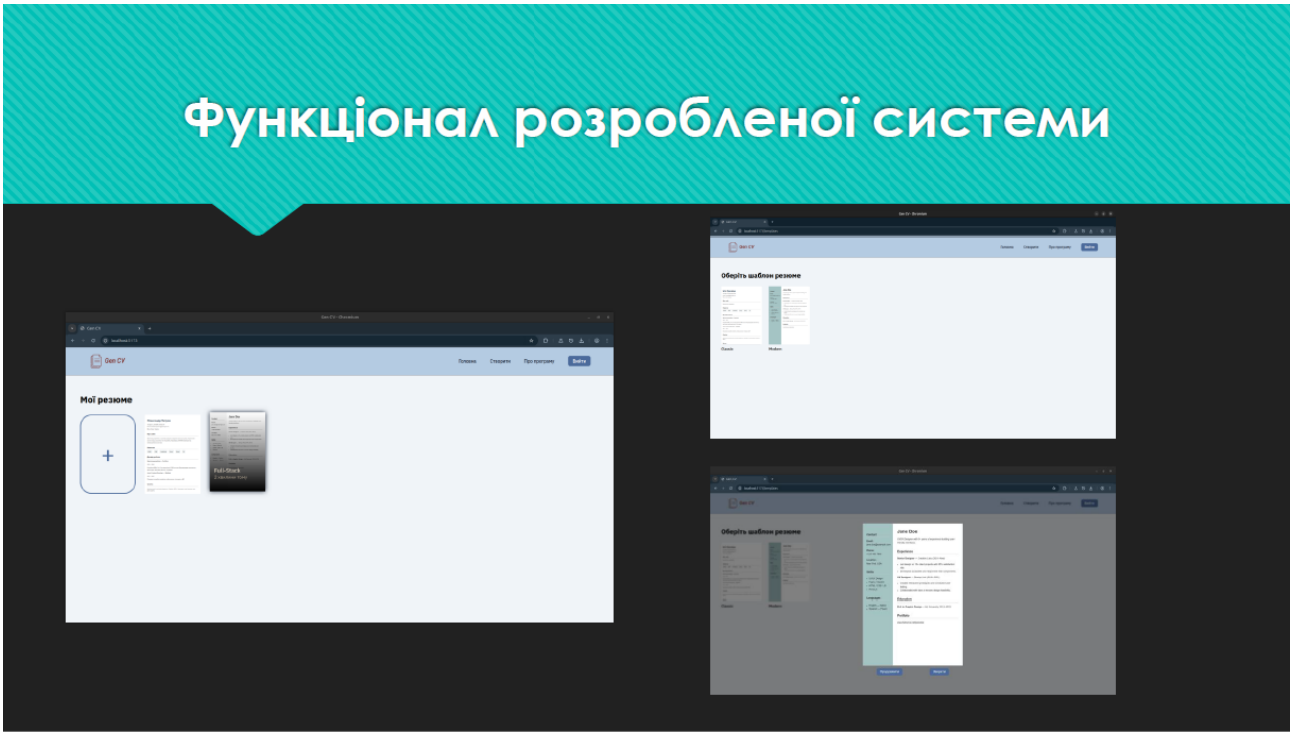

Рисунок Г.15 – Функціонал розробленої системи

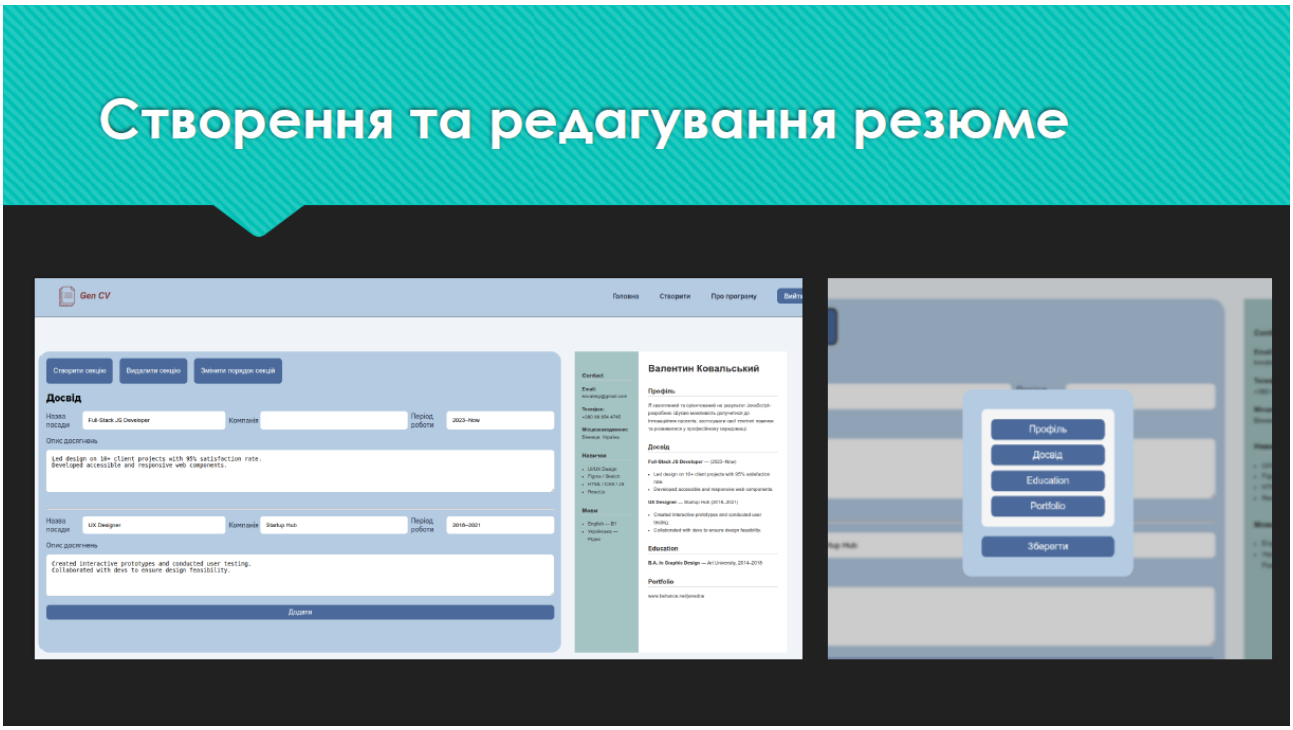

Рисунок Г.16 – Створення та редагування резюме

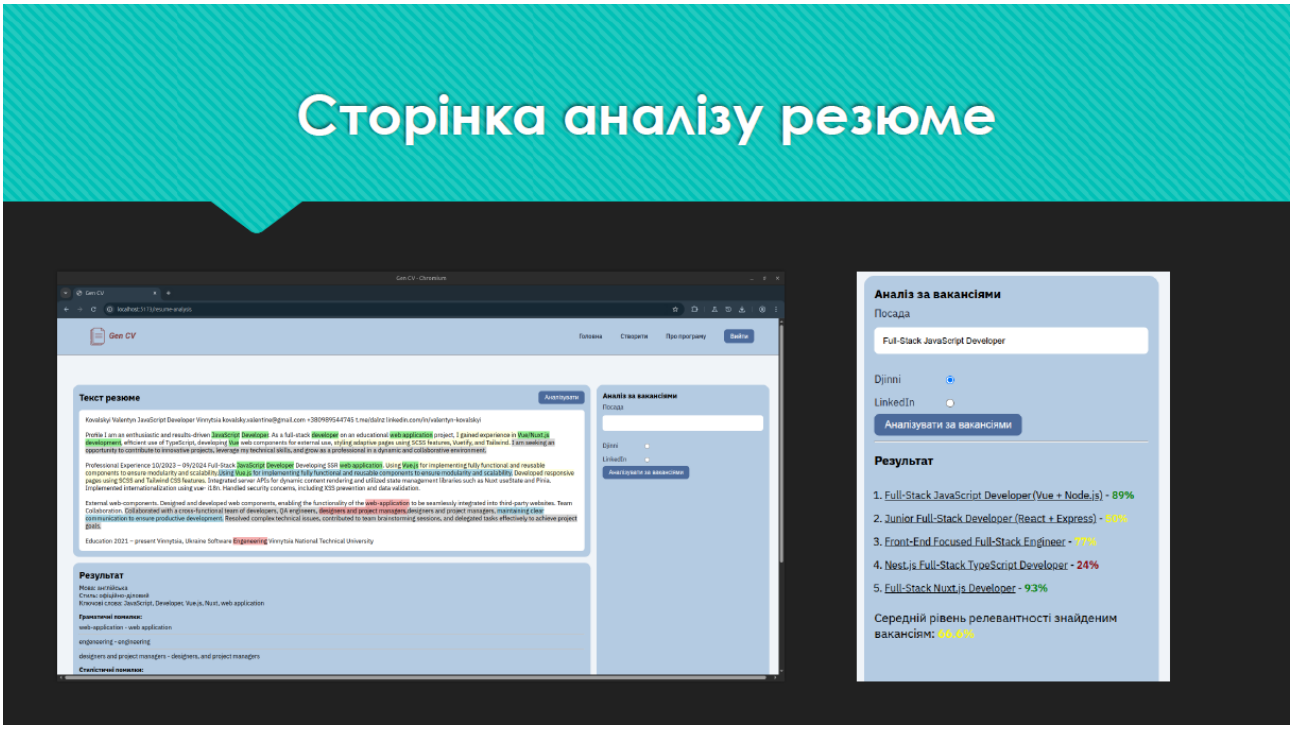

Рисунок Г.17 – Сторінка аналізу резюме

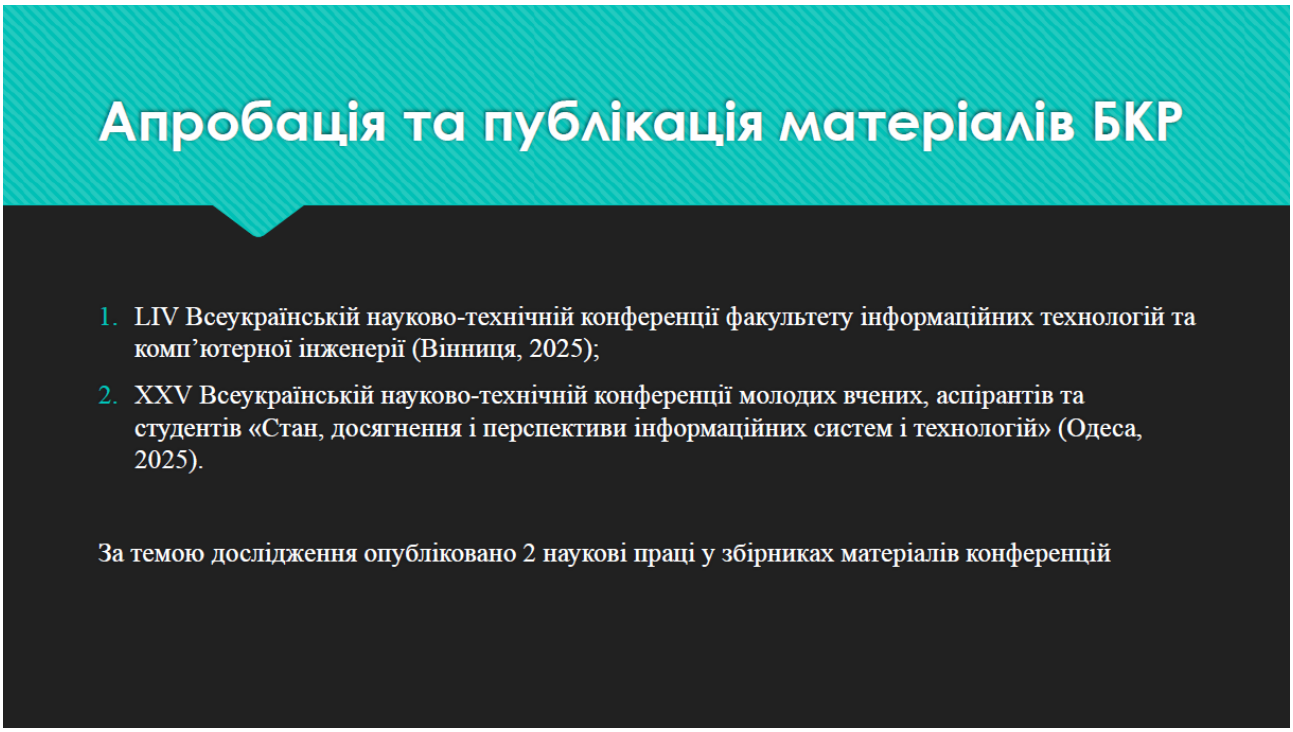

Рисунок Г.18 – Апробація та публікація матеріалів БКР

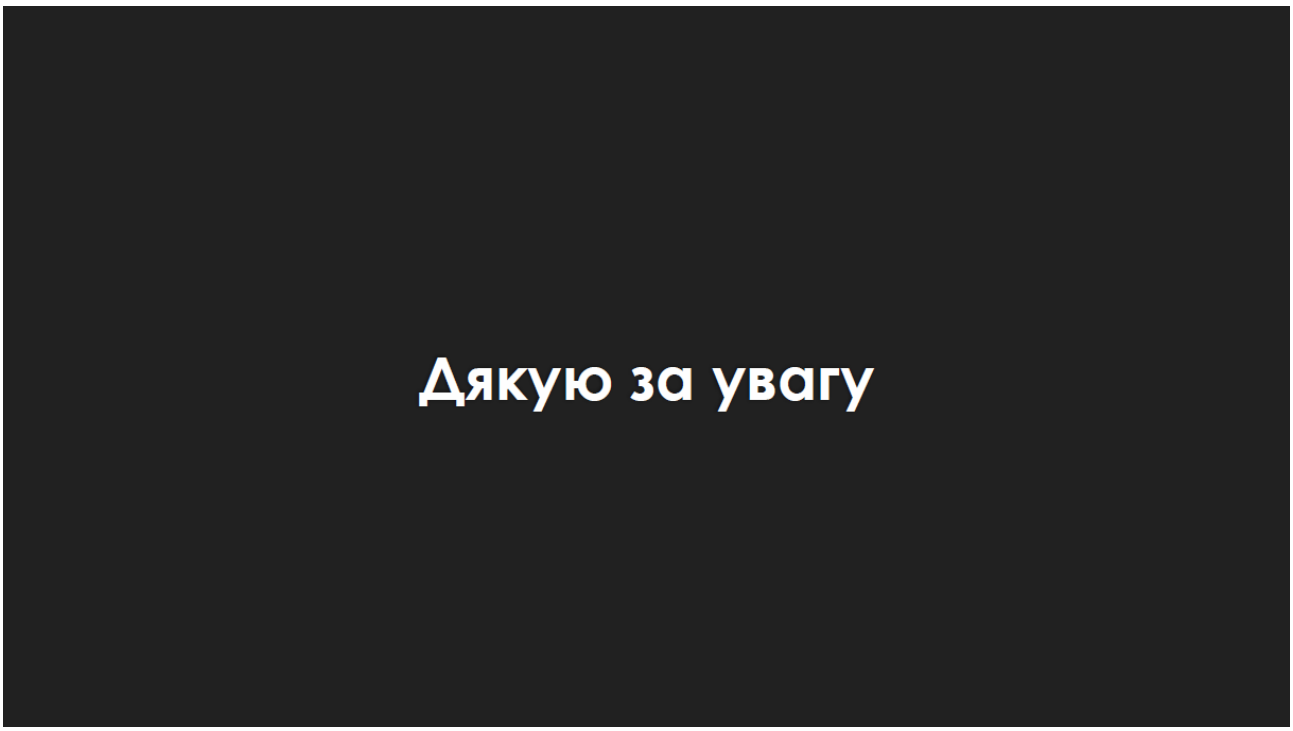

Рисунок Г.19 – Фінальний слайд