Міністерство освіти і науки України
Рівненський державний гуманітарний університет
Кафедра інформаційних технологій та моделювання

Кваліфікаційна робота
за освітнім ступенем «магістр»
на тему: «Веб-сервіс для аналізу вакансій»

Виконав: магістрант 2 курсу
групи М-КН-21
спеціальності 122 «Комп'ютерні науки»
Андрощук Максим Валерійович
Керівник: к.т.н., доцент Бабич С.М.

Рівне – 2023

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці веб-сервісу для аналізу вакансій.

Об’єктом дослідження є процес аналізу вакансій та його технічна реалізація як важливий етап при пошуку роботи.

Метою роботи є створення веб-сервісу для аналізу вакансій, що допомагатиме користувачам аналізувати вакансії під час пошуку роботи.

Основними результатами кваліфікаційної роботи виступає аналіз існуючих рішень, вимог до програмно забезпечення, засобів розробки та реалізований програмний продукт у вигляді веб-сервісу для аналізу вакансій.

Кваліфікаційна робота складається зі вступу, трьох розділів, загальних висновків, списку із 37 використаних джерел та додатку із лістингом програмного коду.

У *вступі* зазначені актуальність, об’єкт, предмет дослідження, мета роботи та поставлені завдання.

У *першому розділі* «Аналіз предметної області» розглядається стан ринку онлайн рекрутингу, детально описується сутність вакансії, аналізуються існуючі рішення та вимоги до програмного забезпечення.

У *другому розділі* «Засоби розробки» порівнюються сучасні інструменти для розробки програмних продуктів та на основі порівняння проводиться вибір засобів розробки для кожного модуля системи.

У *третьому розділі* «Опис програмної реалізації» описується обрана архітектура програмного продукту, детально розглядається реалізація кожного програмного модуля системи, описується внутрішня організація компонентів. Проведено тестування системи, описано тестові випадки та створено посібник користувача для роботи з веб-сервісом.

У *висновках* викладено результати проведеного дослідження.

Ключові слова: вакансія, аналіз вакансії, шахрайська вакансія, дошка оголошень, опис вакансії, веб-сервіс, онлайн рекрутинг.

ABSTRACT

The qualification work is devoted to the development of a web service for job vacancy analysis.

The object of research is the process of analyzing vacancies and its technical implementation as an important stage in the job search.

The purpose of the work is to create a web service for analyzing vacancies that will help users analyze vacancies when looking for a job.

The main results of the qualification work are the analysis of existing solutions, software requirements, development tools, and the implemented software product in the form of a web service for job vacancy analysis.

The qualification work consists of an introduction, three chapters, general conclusions, a list of 37 references and an appendix with a listing of the program code.

The introduction describes the relevance, object, subject of the study, purpose, and objectives of the work.

The first chapter «Analysis of the subject area» examines the state of the online recruiting market, describes the essence of the vacancy in detail, existing solutions and software requirements are analyzed.

The second section «Development Tools» compares modern tools for software development and, based on the comparison, selects development tools for each module of the system.

The third section «Description of the software implementation» describes the selected software product architecture, considers in detail the implementation of each software module of the system, and describes the internal organization of the components. The system was tested, test cases were described and a user manual for working with the web service was created.

The results of the study are presented in the conclusions.

Keywords: job, job analysis, fraudulent job, job board, job description, web service, online recruiting.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення

СУБД – система управління базами даних

БД – база даних

API – інтерфейс програми

REST – архітектурний стиль для розробки мережевих програм

SMTP – протокол для передачі електронних листів по мережі

DI – шаблон проектування у програмуванні, який дозволяє зробити компоненти програми менш залежними один від одного

JSON – формат обміну даними

TSL – криптографічний протокол, що забезпечує безпеку передачі даних у мережі Інтернет

SSL – технологія, що забезпечує зашифроване з'єднання між веб-сервером і веб-браузером.

TLS

CSS – мова стилю сторінок

HTML – мова розмітки сторінок

SQL – мова програмування, що використовується для маніпуляцій із даними в реляційних базах даних.

ВСТУП

Робота – невід’ємна складова життя кожної людини. А власне пошук роботи – це важливий, складний та довгий процес. Коли йдеться про пошук роботи сьогодні, насамперед мається на увазі пошук через онлайн-ресурси як соціальні та професійні мережі, дошки оголошень та агрегатори вакансій, де будь-хто за лічені секунди може знайти тисячі вакансій за заданими параметрами пошуку. Для вибору конкретних вакансій з набору даних кандидатам можуть допомогти інструменти для аналізу вакансій.

Саме тому автоматизований аналіз опису вакансій, пошуку переваг, отримання порад щодо проходження співбесід та резюме, визначення підозрілих вакансій, які є потенційно шахрайськими.

Актуальність вибору теми полягає в активному розвитку цифрових технологій та їхньому впливу на процес пошуку роботи та аналізу вакансій. Використання цифрових ресурсів при пошуку роботи, має велике значення, оскільки це дозволяє зробити процес швидшим, ефективнішим та більш привабливим для користувача.

Об’єктом дослідження є процес аналізу вакансій та його технічна реалізація як важливий етап при пошуку роботи.

Предметом дослідження є технології та інструменти розробки веб-сервісів.

Метою роботи є створення веб-сервісу для аналізу вакансій, що допомагатиме користувачам аналізувати вакансій під час пошуку роботи.

Для досягнення вищезгаданої мети були поставлені такі ***завдання:***

- пошук інформації щодо історії та стану ринку онлайн-рекрутингу, проблеми, що пов'язані із пошуком роботи;
- проаналізувати готові рішення, що пропонують функціонал аналізу вакансій;

- проаналізувати сучасні технології та інструменти для програмної розробки веб-сервісу;
- створити архітектуру розроблюваного веб-сервісу на основі проведеного аналізу;
- створити програмну реалізацію функціональності веб-сервісу.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Постановка задачі

У сучасному світі швидкість та доступність інформації відіграють ключову роль у багатьох аспектах життя, включаючи пошук роботи.

Рекрутинг за останні десятиліття еволюціонував від пошуку оголошень про роботу в газетах та журналах, походу на біржу праці та на ярмарки вакансій, надсилання резюме поштою та відвідин офісів компаній особисто до швидкого доступу до сотень тисяч вакансій на одному веб-ресурсів з інструментами для фільтрації оголошень за бажаними параметрам як наприклад розмір компенсації, індустрія, назва компанії, конкретна технологія чи навіть відстань від офісу до місця проживання в кілометрах або хвилинах та багатьма іншими критеріями. Крім цього, у минулому пошук роботи здебільшого був обмежений місцевим ринком праці або рекомендаціями через особисті контакти, натомість сьогодні ринок став глобальним, де професіонали можуть шукати роботу в будь-якій точці світу. Підтвердженням цього є дослідження компанії Jobvite, Яке виявило, що найпопулярнішим інструментом для пошуку роботи є саме онлайн дошки оголошень, 59% опитаних заявили, що використовували їх при пошуку роботи, 39% використовують соціальні медіа по типу Facebook та Linkedin, 33% відповіли, що використовують кар’єрні сайти компаній [36].

Рис. 1.1. Найпопулярніші інструменти для пошуку роботи

Однак, незважаючи на зростання доступності вакансій, виникає нова проблема – це кількість оголошень, що призводить до перевантаження інформацією. Шукачі роботи стикаються з величезною кількістю пропозицій, серед яких потрібно обрати найбільш релевантні та не потрапити на недобросовісних роботодавців або шахраїв. Це вимагає часу та зусиль для аналізу вакансій, що може стати проблемою на шляху до працевлаштування. Саме тут і виникає необхідність в інструменті, який би забезпечив глибше розуміння ринку праці та допоміг би шукачам роботи з пошуком, аналізуючи історичні дані та описи вакансій.

Веб-сервіс, що розробляється призначено для аналізу вакансій та отримання додаткової інформації, пов’язаної з вакансією.

Ключовою задачею є розробка веб-сервісу для аналізу вакансій, що включатиме в себе визначення індустрії та переваг вакансії на основі опису за допомогою технології NLP та інтеграції із сервісом OpenAI API, пошук сайту компанії та визначення «підозрілих» вакансій, надання рекомендацій кандидату.

Для виконання поставленого завдання, буде потрібно вирішити набір наступних взаємопов’язаних задач:

- виконати аналіз предметної області;
- пошук та обробка набору даних необхідного для побудови моделі;
- побудувати математичну модель для визначення підозрілих вакансій;
- вибір та розробка архітектури компонентів та їх взаємодії;
- створення backend частини та API для неї;
- проектування бази даних для зберігання інформації;
- дослідження та інтеграція сервісів Jooble API, Google API, OpenAI API;
- створення інтерфейсу користувача (frontend частини);
- тестування програмних модулів;
- розгортання програмного забезпечення.

1.2. Огляд сучасного стану онлайн рекрутингу та трендів

Провідну роль у процесі трансформації рекрутингу взяли на себе цифрові платформи та соціальні мережі. Такі як Monster, CareerBuilder, Indeed та Linkedin, які задали тренд і змінили спосіб, яким люди шукають роботу, дозволяючи їм створювати профілі, завантажувати резюме та застосовувати фільтри для пошуку специфічних вакансій. Зі свого боку, роботодавці отримали можливість публікувати вакансії та використовувати алгоритми для відбору кандидатів, що відповідають вимогам конкретних ролей.

Завдяки соціальним мережам, особливо LinkedIn, рекрутинг став більш взаємопов'язаним і соціальним. Професійні мережі надають кандидатам можливість будувати особистий бренд, розвивати зв'язки та

отримувати рекомендації. Водночас, роботодавці використовують соціальні мережі для пошуку талантів, особливо в тих сферах, де є нестача кваліфікованих працівників.

Мобільний рекрутинг теж став важливим елементом сучасного ринку праці. Багато кандидатів використовують смартфони для пошуку роботи, що змушує роботодавців адаптувати свої вакансії та процеси подачі заявок для мобільних користувачів. Відповідність мобільним стандартам та забезпечення зручності подачі заявок через телефон стає критичним для залучення кандидатів.

Технології штучного інтелекту та машинного навчання також активно використовуються в рекрутингу, дозволяючи роботодавцям автоматизувати процеси відбору та аналізувати великі обсяги даних для пошуку найкращих кандидатів. Системи штучного інтелекту можуть сканувати резюме та профілі, визначаючи не тільки відповідність ключовим словам, але й оцінюючи більш специфічні характеристики, такі як культурна відповідність та потенційна продуктивність працівників. Це дозволяє роботодавцям пришвидшити процес найму та підвищити його ефективність.

Розвиток аналітики даних дозволяє роботодавцям та шукачам роботи приймати більш обґрунтовані рішення. Роботодавці можуть аналізувати тренди найму, ефективність рекламних кампаній та результативність різних каналів пошуку кандидатів. З іншого боку, кандидати можуть використовувати аналітику для визначення навичок, які затребувані на ринку, та розробляти стратегії пошуку роботи на основі даних про вакансії та тенденції в працевлаштуванні. Цифрові платформи також можуть аналізувати та надавати пошукачеві більш детальну інформацію щодо зарплати та прогнозувати суму компенсації для вакансій, де не вказано зарплату. Яскравим прикладом є сайт djinni.com [31], який постійно надає інформацію про стан ринку і динаміку зарплат в

ІТ-сфері, що допомагає кандидатам краще розуміти поточний рівень компенсації.

Незважаючи на всі переваги, онлайн рекрутинг також має свої виклики. Проблема перевантаження інформацією, як для шукачів роботи, так і для роботодавців, залишається актуальною. Існує ризик упередженості в алгоритмах ШІ, що може призводити до нерівності у наймі. Крім того, збільшення кількості платформ може призвести до фрагментації ринку, що ускладнює взаємодію між кандидатами та роботодавцями. Важливими є також питання конфіденційності та захисту даних у світі, де особиста інформація легко доступна через інтернет. Тому, з огляду на ці фактори, ця дипломна робота покликана розробити веб-сервіс, який би використовував сучасні технології для розв'язання існуючих проблем у сфері пошуку роботи, зокрема проблему з перевантаженням великою кількістю інформації та аналізом вакансій.

Також з розвитком онлайн рекрутингу почали з’являтись шахрайські схеми – це вид обману, коли недобросовісні організації або особи використовують фальшиві пропозиції роботи для виманювання грошей, отримання доступу до банківських карт або іншої персональної інформації. Люди, які перебувають у пошуку роботи, є вразливою групою, оскільки доволі часто, коли кандидати бачать «привабливу» пропозицію роботи, де їм гарантують виплати кожного тижня і існує опція віддаленої роботи, у пошукачів не виникає критичних запитань і підозр. Особливо це актуально сьогодні в Україні, де багато людей через війну втратили роботу, через що перебувають у скрутному становищі. Такі кандидати – це найбажаніша ціль шахраїв [11].

Згідно даних асоціації Better Business Bureau в США та Канаді кожного року жертвами шахрайських вакансій стають близько 14 мільйонів людей, втрати яких становлять 2 мільярди доларів США і ця цифра невпинно зростає кожного року [7].

Рисунок нижче показує статус людей, які стали жертвами шахраїв – у 53% жертвами стають саме безробітні. Найчастіше люди стикаються з шахрайством при пошуку роботи на повний робочий день, що становить 50% усіх зареєстрованих випадків. Найпопулярнішим фактором, який змусив людину звернути увагу на шахрайську вакансію є гнучкість та можливість працювати вдома (53%), другий чинник – висока заробітна плата (23%) [6].

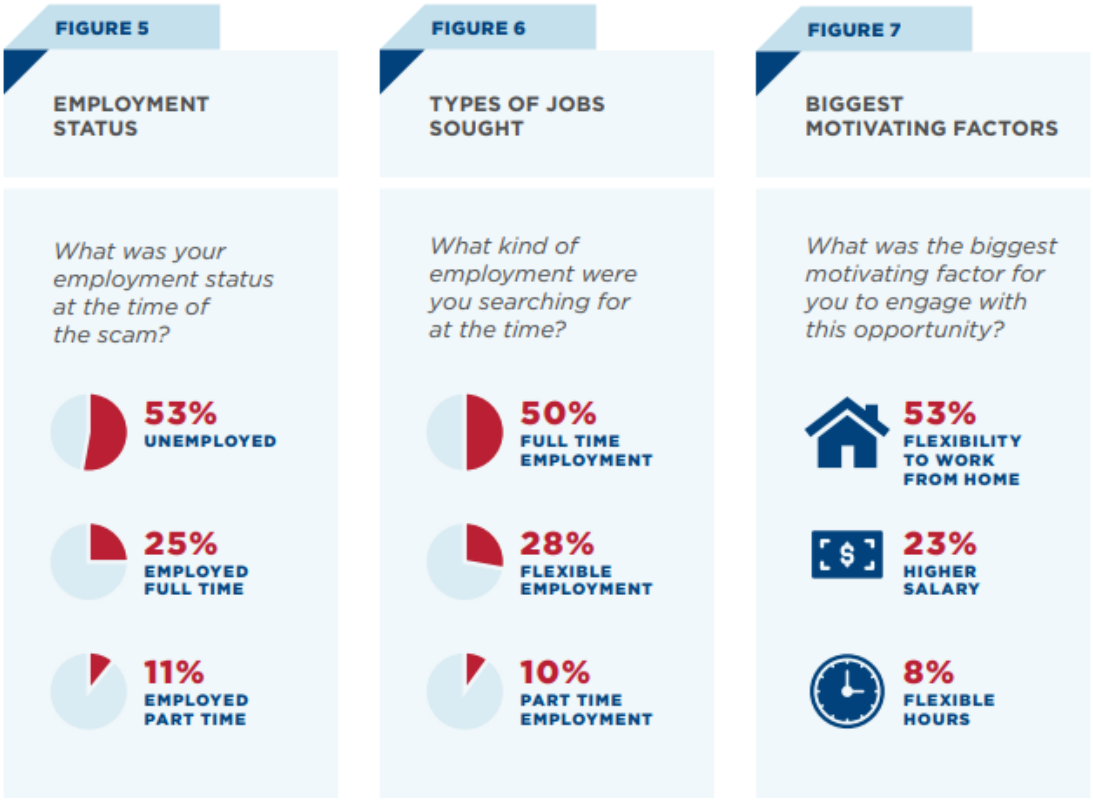

Рис. 1.2. Статистика на основі зареєстрованих випадків шахрайства

Існують різноманітні схеми, якими користуються шахраї, розглянемо декілька з них:

- оплата за навчання або сертифікацію. Так звані «роботодавці» пропонують кандидатам пройти навчання, яке є умовою для працевлаштування. За таке навчання вони просять у кандидатів передоплату у вигляді кількох сотень або тисяч

гривень. Після оплати зазвичай «навчання» та обіцяне працевлаштування не відбувається;

- виманювання банківської інформації. Інша популярна схеми, коли «роботодавці» можуть просити у кандидатів надіслати їм банківську інформацію від кредитної картки для налаштування надходження зарплати. На жаль не усі люди знають, що такого робити не можна і що це може призвести до несанкціонованого доступу до банківського рахунку. За такою ж схемою аферисти можуть просити надіслати їм персональні дані, наприклад фото паспорту, щоб оформити людину на роботу, після чого починають її шантажувати;
- попередня оплата за матеріали. Вакансії по типу «Збір ручок на дому» – це яскравий приклад цієї схеми. Після того як кандидат проявляє інтерес до такої вакансії, шахраї просять у нього внести оплату за матеріали для роботи, гарантуючи, що це лише початкові вкладення. Після таких переказів на картки шахраїв вони зазвичай зникають;
- робота за кордоном. Ця схема включає в себе обіцянку працевлаштування кандидата за кордоном. Оскільки це доволі довгий процес через необхідність оформлення документів та дозволів для роботи за кордоном. Шахраї просять оплату за надані послуги заздалегідь після чого зникають [17].

Як пошукач може зрозуміти, що перед ним фальшива вакансія? Перш за все потрібно зрозуміти мотиви шахраїв. Їх мета – це зацікавити та охопити своїм оголошенням якомога ширшу аудиторію. Цього вони досягають завдяки дуже загальним оголошенням, наприклад «Менеджер з продажу», «Робота з дому», «Оператор онлайн чату», «Робота без досвіду, висока оплата». Мета таких заголовків – зацікавити кандидата і змусити його зробити клік на вакансію. Далі пошукач бачить опис оголошення, яке

зазвичай дуже розмите, без конкретних обов’язків, використовуються загальні фрази як «пунктуальність», «володіння ПК», «якісне виконання завдань», «відповідальність». До того ж зазвичай ці вакансії містять нереалістичні обіцянки та завищену заробітну плату з виплатами кілька разів на місяць [1].

Саме по цих шаблонам можна визначити, що вакансія є підозрілою. Також дуже часто кандидати не ставлять під сумнів вакансії з таким змістом і цим користуються шахраї, «спокушаючи» людей, які шукають роботу, привабливими умовами. Але, навіть відгукнувшись на таку вакансію, у кандидата ще є час та можливість зрозуміти, що вакансія – фальшивка. Пошукачу варто звернути увагу на тон та манеру спілкування роботодавця, якщо роботодавець квапиться та наголошує на тому, що у кандидата є обмежений час для прийняття рішення. Також варто спробувати пошукати інформацію про компанію, погуглити ім’я працедавця та номер телефону. Шахраї намагаються швидко змінювати цю інформацію, але це може допомогти знайти відгуки або інформацію про роботодавця.

1.3. Що таке вакансія

Основні складові вакансії – це власне заголовок вакансії, опис, регіон (це може бути місто, область, країна або позначка про можливість працювати віддалено) та назва компанії. Ці параметри є невід’ємними складовими вакансії, до важливих параметрів також можна віднести заробітну плату, але часто компанії не розкривають ці дані. Згідно з дослідження Indeed за останні роки цей показник в США зріс з 18.4% до 43.7% [20], але це досі менше половини всіх оголошень.

Найбільш інформативною частиною оголошення є її опис. Саме в описі вакансії роботодавець може розповісти більше про специфіку роботи, про вимоги до кандидата та очікування, переваги роботи в даній

компанії, вигоди та пільги на додаток до заробітної плати, описати культуру та досягнення компанії.Боротьба за увагу кандидатів часто призводить до певної одноманітності в оголошенням, тут можна виділити наступні проблеми:

1. Стандартизація описів вакансій: Багато компаній використовують шаблони для створення описів вакансій, що сприяє їх стандартизації. Це може бути ефективним для роботодавців, оскільки спрощує процес створення оголошень, але також може призводити до того, що всі описи виглядають однаково.
2. SEO-оптимізація: Роботодавці часто використовують певні ключові слова, щоб їх оголошення легше знаходилися через пошукові системи та платформи для пошуку роботи. Це може призвести до використання одних і тих же фраз і термінів.
3. Вимоги до робочих навичок: Оскільки багато професій мають стандартні набори навичок та кваліфікацій, описи вакансій часто повторюють ці вимоги, що робить їх схожими одна на одну.

Через що пошукачеві часом важко обрати серед одноманітних вакансій якусь одну і це призводить до наступних проблем.

1. Через схожість описів вакансій пошукачам важко виділити ті, які найбільш відповідають їхнім унікальним навичкам, досвіду та інтересам.
2. Розуміння реальних очікувань. Стандартизовані описи можуть не давати чіткого уявлення про реальні обов'язки та вимоги, що ускладнює для кандидатів розуміння реальних очікувань та вимог до них.
3. Відсутність індивідуального підходу. Пошукачі можуть відчувати, що оголошення не відображають унікальної культури та цінностей компанії, що робить важчим з'ясування, чи буде компанія добрим місцем для роботи.

4. Перевантаження ключовими словами Деякі оголошення можуть бути перевантажені ключовими словами та жаргоном, що ускладнює розуміння суті вакансії та може відлякувати потенційних кандидатів.

Крім цього, згідно досліджень Glassdoor 86% пошукачів шукають в Інтернеті інформацію про компанію та рейтинги, читають відгуки [14]. Це також можна пов’язати з недостатньої інформативністю та одноміністю описів вакансій.

Це ще раз підкреслює важливість таких параметрів вакансії як опис та назва компанії. Аналізу цих полів буде надано суттєве значення при розробці програмного забезпечення для дипломної роботи.

1.4. Аналіз існуючих рішень

Розглянуто та проаналізовано низку популярних сайтів з пошуку роботи в Україні, таких як robota.ua, work.ua. djinni.com. До їхніх переваг можна віднести наступне:

- доступ до великого масиву історичних даних по вакансіях та володіння даними, які недоступні пошукачеві, як наприклад кількість відгуків на вакансію, найпопулярніші пошукові запити пошукачів, заробітна плата, історія операцій над вакансією;
- велика аудиторія, яка вимірюється у мільйонах візитів на місяць;
- наявність бюджетів на розробку і можливість впровадження нової функціональності.

Серед недоліків можна виокремити такі:

- відсутність можливості або обмежена можливість безкоштовної публікації оголошень через що вказані сайти не володіють повнотою даних про всі вакансії;

- дані сайти стверджують про наявність механізмів виявлення і блокування підозрілих вакансій, що звісно є перевагою, але зазвичай це мануальна модерація, а у випадку автоматизованих програмних продуктів користувачі не мають доступу до них, тобто не можуть їх застосувати до вакансій, розміщених на інших джерелах;
- відсутність персоналізації та рекомендацій для конкретного пошукача щодо конкретної вакансії.

Такі сайти в першу чергу орієнтовані на масового користувача, які часто не задумуються над проблемою підозрілих/шахрайських вакансій або рекомендаціями щодо проходження співбесіди, оскільки часто для низькокваліфікованих вакансій достатньо телефонного дзвінка.

Окремих інструментів, які надають можливість визначити «підозрілі вакансії» та отримати рекомендації в Інтернеті не виявлено. Зазвичай певну функціональність пропонують саме дошки оголошень, професійні мережі.

1.5. Аналіз вимог до програмного забезпечення

Програмне забезпечення повинно надавати користувачеві доступ до наступних функцій:

- можливість зареєструватись на сайті та підтвердити почту;
- можливість увійти в кабінет та вийти з нього;
- можливість відновити пароль;
- введення інформації про вакансію (поля «Заголовок вакансії», «Опис вакансії», «Назва компанії»);
- отримання результату на основі даних про вакансію чи є вона підозрілою;
- отримання інформації про індустрію введеної вакансії;

- отримання сайту компанії на основі даних з поля «Назва компанії»;
- отримання рекомендацій щодо проходження співбесіди та створення резюме для конкретної вакансії;
- виділення кольором основних переваг, які вказані в описі введеної вакансії;
- можливість ознайомитись та переглянути підозрілі/неякісні вакансії;
- перегляд рекомендованих вакансій на основі заголовку вакансії;
- пошук вакансій на основі ключових слів та регіону.

Веб-сервіс повинен забезпечувати наступні варіанти використання:

Таблиця 1.1 – Варіант використання U-1

Ідентифікатор	U-1
Назва	Реєстрація на сайті та підтвердження електронної пошти
Опис функції	Користувач створює акаунт на сайті, увівши необхідні дані, та підтверджує свою електронну пошту.
Учасники	Неавторизований користувач
Передумови для виконання	Користувач має доступ до веб-сайту і валідну електронну пошту
Результат виконання	Заповнені поля вводу для даних про вакансію
Покроковий сценарій виконання	1. Користувач вибирає опцію «Реєстрація» на веб-сайті. 2. Користувач вводить необхідні дані для

	реєстрації: пароль та електронну адресу. 3. Користувач натискає кнопку «Зареєструватися». 4. Користувач отримує електронний лист із посиланням для підтвердження електронної пошти. 5. Користувач переходить за посиланням у листі для завершення реєстрації.

Таблиця 1.2 – Варіант використання U-2

Ідентифікатор	U-2
Назва	Вхід та вихід з особистого кабінету на сайті
Опис функції	Користувач вводить свої облікові дані для входу в особистий кабінет на сайті та виходить з нього, коли це необхідно.
Учасники	Авторизований користувач, неавторизований користувач
Передумови для виконання	Користувач має зареєстрований акаунт на сайті
Результат виконання	Користувач успішно увійшов у свій кабінет та вийшов з нього
Покроковий сценарій виконання	1. Користувач вибирає опцію «Вхід» на веб-сайті. 2. Користувач вводить свій емейл та пароль. 3. Користувач натискає кнопку «Увійти» для доступу до свого особистого кабінету. 4. Після завершення сесії користувач

	вибирає опцію «Вийти» для виходу з особистого кабінету.

Таблиця 1.3 – Варіант використання U-3

Ідентифікатор	U-3
Назва	Зміна та відновлення пароля
Опис функції	Користувач може змінити існуючий пароль або відновити пароль до свого облікового запису на сайті.
Учасники	Авторизований користувач, неавторизований користувач
Передумови для виконання	Користувач має доступ до свого облікового запису або до електронної пошти, асоційованої з обліковим записом
Результат виконання	Пароль користувача успішно змінено або відновлено
Покроковий сценарій виконання	1. Для відновлення забутого пароля користувач на сторінці входу вибирає опцію «Забули пароль?». 2. Користувач вводить свою електронну пошту, асоційовану з обліковим записом. 3. Користувач натискає кнопку «Відновити пароль» та слідує інструкціям, надісланим на електронну пошту для встановлення нового.

Таблиця 1.4 – Варіант використання U-4

Ідентифікатор	U-4
Назва	Введення інформації про вакансію
Опис функції	Користувач вносить інформацію в поля: назва вакансії, опис вакансії та назва компанії.
Учасники	Авторизований користувач
Передумови для виконання	Браузер відкритий на сторінці веб-сервісу та користувач авторизований
Результат виконання	Поля для внесення інформації про вакансію заповнені
Покроковий сценарій виконання	1. Користувач клікає на поле «Назва вакансії» та вписує необхідні дані. 2. Користувач клікає на поле «Опис вакансії» та вписує необхідні дані. 3. Користувач клікає на поле «Назва компанії» та вписує необхідні дані.

Таблиця 1.5 – Варіант використання U-5

Ідентифікатор	U-5
Назва	Перевірка вакансії на підозрілість
Опис функції	Користувач вводить інформацію про вакансію, і система аналізує ці дані, щоб визначити, чи є вакансія підозрілою.
Учасники	Авторизований користувач

Продовження таблиці 1.5

Передумови для виконання	Користувач ввів інформацію про вакансію, яку потрібно перевірити
Результат виконання	Користувач отримує оцінку підозрілості вакансії на основі введеної інформації
Покроковий сценарій виконання	1. Користувач натискає кнопку «Аналіз» для ініціації аналізу. 2. Система обробляє введені дані, використовуючи алгоритми для визначення підозрілості вакансії. 3. Користувач отримує результат аналізу, який вказує на те, чи вважається вакансія підозрілою.

Таблиця 1.6 – Варіант використання U-6

Ідентифікатор	U-6
Назва	Отримання інформації про індустрію вакансії
Опис функції	Користувач вводить інформацію про вакансію, і система, використовуючи OpenAI API, надає інформацію про індустрію, пов'язану з цією вакансією.
Учасники	Авторизований користувач
Передумови для виконання	Користувач ввів інформацію про вакансію
Результат виконання	Користувач отримує інформацію про індустрію вакансії.

Продовження таблиці 1.6

Покроковий сценарій виконання	1. Користувач натискає кнопку «Аналіз» 2.Система використовує OpenAI API для отримання інформації про відповідну індустрію. 3.Користувач отримує результат у вигляді індустрії.

Таблиця 1.7 – Варіант використання U-7

Ідентифікатор	U-7
Назва	Отримання веб-сайту компанії на основі назви компанії
Опис функції	Користувач вводить інформацію про вакансію, зокрема назву компанії, і система здійснює пошук, щоб надати веб-сайт цієї компанії.
Учасники	Авторизований користувач
Передумови для виконання	Користувач ввів інформацію про вакансію
Результат виконання	Користувач отримує посилання на веб-сайт введеної компанії
Покроковий сценарій виконання	1. Користувач натискає кнопку «Аналіз». 2. Система проводить пошук в Інтернеті за допомогою Google API. 3. Користувач отримує посилання на веб-сайт компанії, який можна відкрити для отримання більш детальної інформації.

Таблиця 1.8 – Варіант використання U-8

Ідентифікатор	U-8
Назва	Отримання рекомендацій для співбесіди та створення резюме
Опис функції	Користувач вводить деталі конкретної вакансії, і система, використовуючи OpenAI API, надає персоналізовані рекомендації щодо проходження співбесіди та створення ефективного резюме для цієї вакансії.
Учасники	Авторизований користувач
Передумови для виконання	Користувач ввів інформацію про вакансію
Результат виконання	Користувач отримує індивідуальні рекомендації для підготовки до співбесіди та створення резюме
Покроковий сценарій виконання	1. Користувач натискає кнопку «Аналіз». 2. Система використовує OpenAI API для аналізу наданих даних і генерування рекомендацій щодо підготовки до співбесіди та створення резюме, яке відповідає вимогам вакансії. 3. Користувач отримує результат у вигляді рекомендацій

Таблиця 1.9 – Варіант використання U-9

Ідентифікатор	U-9
Назва	Виділення кольором переваг у описі вакансії
Опис функції	Користувач вводить текст опису вакансії, і

	система автоматично виділяє кольором ключові переваги (benefits), зазначені в описі.
Учасники	Авторизований користувач
Передумови для виконання	Користувач ввів інформацію про вакансію
Результат виконання	Основні переваги у тексті опису вакансії виділені кольором для легшого сприйняття
Покроковий сценарій виконання	1. Користувач натискає кнопку «Аналіз». 2.Система аналізує текст, виявляє та автоматично виділяє кольором слова або фрази, які описують ключові переваги вакансії (наприклад, гнучкий графік роботи, страхування, корпоративні заходи тощо). 3. Користувач отримує оновлений текст опису, де всі основні переваги виділені кольором.

Таблиця 1.10 – Варіант використання U-10

Ідентифікатор	U-10
Назва	Перегляд списку підозрілих вакансій
Опис функції	Користувач має можливість переглянути та ознайомитися з прикладами підозрілих вакансій, які були використані для побудови моделі виявлення шахрайських пропозицій.
Учасники	Авторизований користувач, неавторизований користувач
Передумови для виконання	Користувач має доступ до веб-сайту

Результат виконання	Відкрита сторінка із списком підозрілих вакансій
Покроковий сценарій виконання	1. Користувач вибирає опцію «Переглянути підозрілі вакансії» на веб-сайті. 2. Система надає доступ до бази даних з прикладами підозрілих вакансій. 3. Користувач бачить список таких вакансій

Таблиця 1.11 – Варіант використання U-11

Ідентифікатор	U-11
Назва	Перегляд рекомендованих вакансій на основі введеної
Опис функції	Користувач вводить інформацію про вакансію і система відображає рекомендовані вакансії, які відповідають введеному заголовку.
Учасники	Авторизований користувач
Передумови для виконання	Користувач коректно заповнив поля про вакансію
Результат виконання	Користувач отримує список рекомендованих вакансій, що відповідають введеній
Покроковий сценарій виконання	1. Користувач натискає кнопку «Аналіз». 2. Система робить запит до Jooble API, щоб знайти та відобразити вакансії зі схожими заголовками. 3. Користувач отримує список вакансій, які відповідають критеріям пошуку, з

	можливістю перегляду детальної інформації про кожну вакансію.

Таблиця 1.12 – Варіант використання U-12

Ідентифікатор	U-12
Назва	Пошук вакансій за ключовими словами та регіоном
Опис функції	Користувач вводить ключові слова та обирає регіон для пошуку вакансій, і система використовуючи Jooble API для пошуку і відображення таких вакансій
Учасники	Авторизований користувач, неавторизований користувач
Передумови для виконання	Браузер відкритий на сторінці веб-сервісу

Продовження таблиці 1.12

Результат виконання	Користувач отримує список вакансій, які відповідають заданим критеріям пошуку
Покроковий сценарій виконання	1. Користувач вводить ключові слова (наприклад, «маркетинг», «розробник») у поле пошуку. 2. Користувач вибирає регіон (наприклад, «Київ», «Львів»). 3. Користувач натискає кнопку «Шукати вакансії». 4. Система використовує Jooble API для здійснення пошуку вакансій на основі введених критеріїв.

	5. Користувач отримує список вакансій, що відповідають його запиту, з можливістю перегляду деталей кожної вакансії.

Описані вище варіанти використання можна представити за допомогою UML діаграми варіантів використання (англ. Use case diagram) [15, с.67-68].

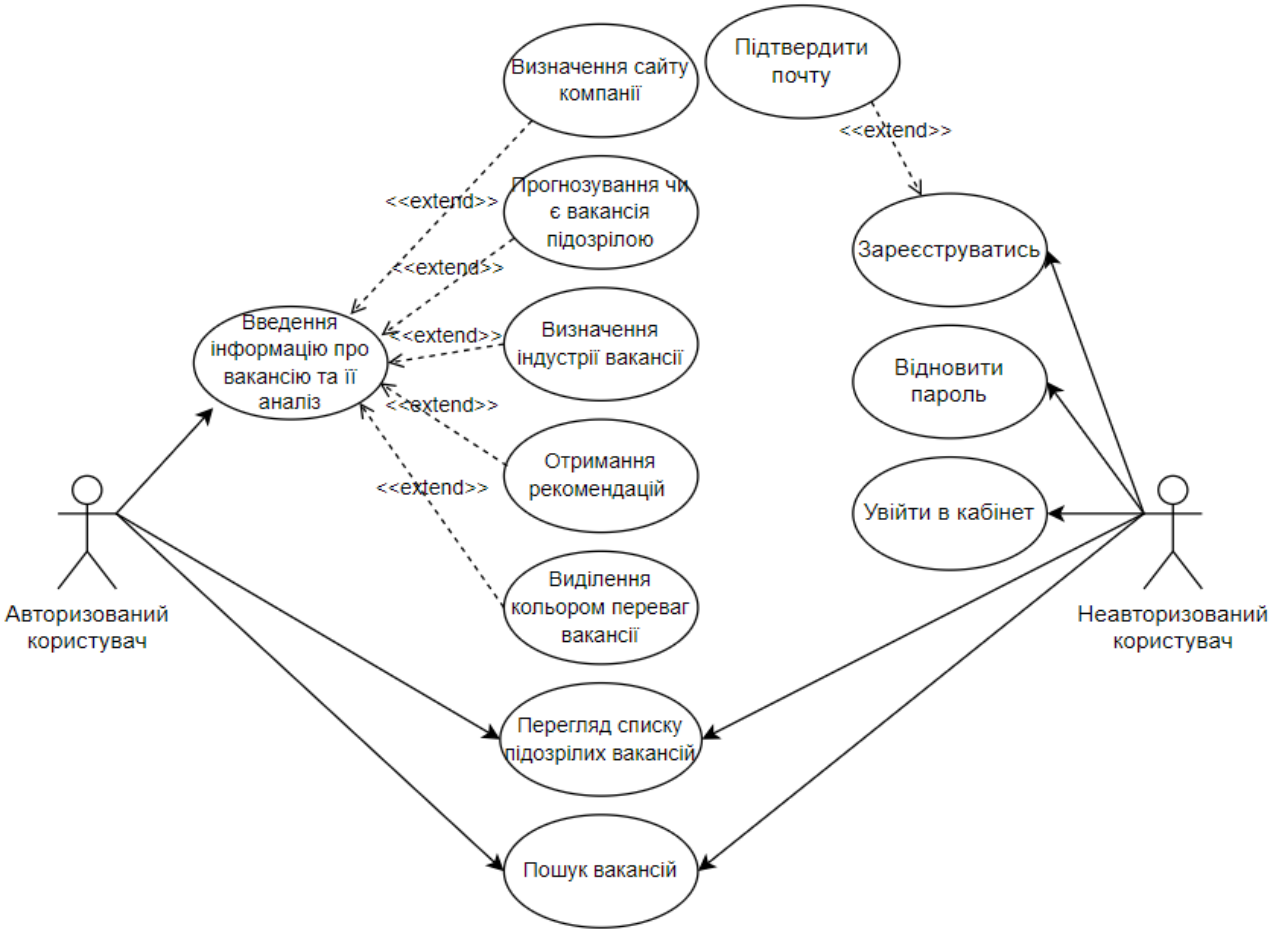

Рис. 1.3. Діаграма варіантів використання

РОЗДІЛ 2
ЗАСОБИ РОЗРОБКИ

Проаналізовано велику кількість мов програмування та СУБД. Головним критерієм вибору засобів є швидкодія, гнучкість, простота в реалізації, що грає важливу роль для розширення системи в майбутньому.

Важливим чинником також є сумісність і взаємодія модулів системи, оскільки система складається з 4 модулів.

2.1. Модуль аналізу вакансій

Мовою програмування для даного компонента системи обрано Python. Оскільки ця мова програмування вирізняється своєю високою гнучкістю, що робить цю мову частим вибором серед інженерів для різноманітних задач, у тому числі обробка та аналіз даних, розробка веб-додатків, автоматизація процесів, машинне навчання та робота зі штучним інтелектом [5].

До основних переваг Python для аналізу описів вакансій можна віднести [3]:

Легкість читання та написання. Python часто порівнюють з псевдокодом, тому що його синтаксис дозволяє розробникам виражати концепції без зайвого кодування, що спрощує підтримку та колаборацію в командах [21, с.3-4].

Велика екосистема: Python підтримується обширною екосистемою модулів та бібліотек, які охоплюють практично будь-яку можливу потребу в аналізі даних. Це означає, що для багатьох задач вже існують готові рішення, які можна легко інтегрувати та використовувати.

Широка підтримка спільноти: Наявність великої кількості документації, форумів та ресурсів для навчання робить процес вирішення проблем більш швидким і ефективним.

Міжплатформенна сумісність: Python може використовуватися на різних операційних системах, що забезпечує гнучкість при розгортанні додатків та систем.

Легкість інтеграції: Python може легко інтегруватися з іншими мовами програмування та системами, що дає можливість використовувати найкращі інструменти для різних задач.

Крім цього, як зазначено вище, Python містить велику кількість модулів та бібліотек. Багато із них будуть використані під час проектування системи. До прикладу бібліотека Pandas пропонує структури даних DataFrame, які спрощують маніпуляцію та візуалізацію великих наборів даних. Для аналізу описів вакансій DataFrame може використовуватися для очищення, трансформації та агрегування інформації, що допомагає швидко без написання зайвого коду виявляти ключові тенденції та патерни [25].

Ще один модуль – це Scikit-learn, який є набором інструментів, який об'єднує простоту використання з потужними методами машинного навчання. Він підтримує велику кількість стандартних алгоритмів класифікації, регресії, кластеризації, що можна застосовувати для розпізнавання шаблонів у описах вакансій та прогнозування тенденцій [12, с.204-205]. Саме ця бібліотека містить математичні моделі, які будуть застосовані для розпізнавання підозрілих вакансій [32].

Бібліотеки NLTK та spaCy використовуються для обробки природної мови: NLTK є першою серед бібліотек NLP для Python і надає інструменти для статистичного аналізу, класифікації та розмітки тексту. SpaCy, з іншого боку, є більш сучасною та швидкою бібліотекою, яка оптимізована для великих обсягів тексту та забезпечує високу точність та

продуктивність в розпізнаванні елементів природної мови [34]. Ці бібліотеки стануть в нагоді для аналізу описів вакансій.

Pymorphy в свою чергу застосовують для морфологічного аналізу: Pymorphy2 є потужним інструментом для аналізу текстів. Він дозволяє виконувати нормалізацію словоформ і може бути використаний для розробки більш точних аналітичних моделей, які враховують морфологічні особливості української мови, що є незамінним при аналізі україномовних вакансій [27].

Окрім цього відомі корпорації також використовують Python в своїх програмних рішеннях. Яскравим прикладом є Google, де Python є однією з трьох основних мов програмування, які використовуються в Google. Він використовується для багатьох платформ та систем, включаючи YouTube та систему управління версіями Google. Facebook використовує Python для різноманітних інфраструктурних компонентів, а також для аналізу даних. Spotify та Netflix використовує Python для своїх сервісів рекомендацій та аналізу даних [37].

2.2. Серверна частина

Для реалізації серверної частини обрано технологію .NET 6.0, яка є передостанньою ітерацією платформи, що забезпечує покращену продуктивність, зменшену використання пам'яті та підтримку сучасних патернів розробки, що є критично важливим для сучасних веб-додатків. Окрім того дана версія LTS (long term support) від Microsoft, що також є перевагою [24]. Також NET 6.0 забезпечує підтримку крос-платформенного розроблення, що дозволяє розгортати веб-додатки на Windows, Linux та macOS без змін у коді.

Одна з вимог до API – це підтримка архітектури REST, оскільки це забезпечує максимальну гнучкість та уніфікацію, це дозволяє легко

взаємодіяти з різними клієнтами, включаючи браузери, мобільні додатки та інші сервери. Ключовим принципом REST є єдність інтерфейсу (Uniform Interface). Інтерфейс між клієнтом і сервером повинен бути стандартизованим. Це означає, що для доступу та управління ресурсами використовуються стандартні HTTP методи (GET, POST, PUT, DELETE тощо) [22, с. 23-25] [23]. У свою чергу .NET 6.0 пропонує розробнику фреймворк ASP.NET Core Web API, який спеціально призначений для створення високопродуктивних веб-API, що підтримують RESTful принципи. Він надає багатий набір функціональностей для маршрутизації, серіалізації, валідації та забезпечення безпеки.

Також дана технологія містить фреймворк Entity Framework Core, який дозволяє працювати із базами даних за допомогою об’єктів .NET, що пришвидшує розробку. Також завдяки даному фреймворку зникає необхідність написання SQL запитів, які можна замінити на LINQ запити (Language Integrated Query) [18]. Це пришвидшує розробку і створення бази даних, оскільки інженер можна виконувати всі налаштування в одному середовищі розробки [19].

Варто відзначити, що .NET 6.0 має вдосконалені можливості для асинхронного програмування, використовуючи асинхронні патерни та конструкції мови, такі як async та await. Це дозволяє покращити продуктивність додатків, особливо у сценаріях з великим обсягом вхідних/вихідних операцій або при роботі з мережевими запитами.

Сьогодні безпека – є дуже важливим фактором, а технології .NET пропонують бібліотеку ASP.NET Core Identity, що дозволяє без зайвих зусиль, використовуючи найкращі практики, налаштувати систему управління користувачами та аутентифікації. Також у контексті безпеки, .NET 6.0 надає розширені можливості для шифрування даних і безпечного обміну інформацією, включаючи підтримку останніх стандартів TLS для захищених мережевих з'єднань. Це дозволяє забезпечити високий рівень

захисту даних користувачів та забезпечити відповідність сучасним вимогам до кібербезпеки.

Дана технологія також пропонує широкі можливості моніторингу та логування, які є важливими для підтримки та оптимізації веб-додатків. .NET 6.0 підтримує інтеграцію з різноманітними системами моніторингу та логування, такими як ELK Stack (Elasticsearch, Logstash, Kibana) або Prometheus і Grafana, що дозволяє розробникам отримувати детальну інформацію про стан додатку та оперативно реагувати на будь-які проблеми. Що може бути дуже важливим фактором, якщо програмний продукт буде масштабуватись.

Для тестування та документування API обрано технологію Swagger UI. Він дозволяє користувачам взаємодіяти з API, виконуючи запити та отримуючи результат. Дуже легкий та зручний у використанні, а найголовніше – не потребує окремої розробки, оскільки є вбудованим в ASP .NET Core WEB API, завдяки чому всі зміни одразу з’являються в документації [28].

2.3. Інтерфейс користувача

Для реалізації інтерфейсу користувача обрано фреймворк Angular. Це повноцінне рішення, яке яке надає розробникам не тільки бібліотеку для створення інтерфейсу, а й набір інструментів для роботи з формами, маршрутизацією, запитами до сервера. Angular використовує компонентний підхід, що полегшує повторне використання коду і робить архітектуру додатків більш зрозумілою і організованою [4].

Angular використовує TypeScript, який надає статичну типізацію та об'єктно-орієнтовані можливості, забезпечуючи легше сприйняття коду та зменшуючи ймовірність помилок. Фреймворк підходить для розробки як великих, так і малих додатків, забезпечуючи необхідну інфраструктуру для масштабування проектів.

Angular має потужну систему внутрішньої залежності (dependency injection), що сприяє створенню модульних та легко тестованих компонентів. Це важливо для розробки великомасштабних додатків, де потреба у високій ступені повторного використання коду та легкості обслуговування є важливими факторами.

Також, Angular підтримує двостороннє зв'язування даних (two-way data binding), що дозволяє автоматично синхронізувати дані між моделлю та представленням. Це спрощує процес розробки, оскільки розробникам не потрібно писати додатковий код для ручного управління DOM та оновлення інтерфейсу користувача.

Головною альтернативою є React, на даний момент ці дві технології є найпопулярнішими серед веб-розробників. Для написання дипломної роботи обрано саме Angular через наявність знань та досвіду написання програм на цьому фреймворку та тому що React представляє собою бібліотеку, а не фреймворк через що поріг входження для опанування технології є нижчим, але для реалізації деяких функцій потрібно також опанувати додаткові JavaScript фреймворки. В свою чергу Angular є повноцінним інструментом із усіма необхідними складовими для розробки [30].

Для розробки власне інтерфейсу обрано фреймворк Bootstrap, який є одним з найпопулярніших фронтенд фреймворків для розробки веб-інтерфейсів. Ось кілька переваг, які можна виокремити: [35, с.101]

- Bootstrap містить уже готові CSS-стилі та компоненти, які можна відразу використовувати як готові шаблони у своєму застосунку. Це в свою чергу зменшує час на розробку та кількість написаного коду;
- Bootstrap підтримує адаптивний дизайн, що гарантує, що компоненти адаптуються та коректно працюють на різних пристроях і розмірах екранів, від мобільних телефонів, планшетів, годинників до великих моніторів;

- Bootstrap пропонує набір стандартів для веб-дизайну, що допомагає забезпечити однаковий вигляд усього веб-сайту;
- Завдяки широкій популярності Bootstrap має активну спільноту, що допомагає швидко знаходити відповіді на запитання та ресурси для навчання. Зокрема Bootstrap має детальну документацію, що зменшує поріг входу в дану технологію;
- Bootstrap легко інтегрується з іншими фронтенд бібліотеками та фреймворками, зокрема Angular, React, Vue.js.

Однак важливо зазначити, що Bootstrap також має деякі недоліки. Складні та масштабні проекти часто вимагають індивідуального дизайну, який не можна реалізувати виключною з допомогою функціональності представленої у фреймворку Bootstrap [8].

2.4. База даних

Веб-сервіс повинен зберігати дані про користувачів, історичні дані про вакансії, пошукові запити користувачів, результати аналізу. Для цього потрібно реалізувати збереження даних в СУБД. Вибір був між двома технологіями: MS SQL Server та PostgreSQL. Дані СУБД мають деякі суттєві відмінності, до прикладу MS SQL Server це продукт компанії Microsoft, який пропонується як комерційний продукт з підтримкою та додатковими сервісами та включає в себе покращені можливості для бізнес-аналітики та звітності, такі як SQL Server Reporting Services (SSRS) та SQL Server Integration Services (SSIS). Він пропонує високу продуктивність та широкі можливості у проектах великого масштабу. Він такоє має власну реалізацію SQL, T-SQL, яка містить деякі пропрієтарні розширення.

У свою чергу PostgreSQL має відкритий код, підтримується спільнотою та є безкоштовним для використання.Широко відомий своєю

високою відповідністю стандартам SQL та підтримкою розширених функцій, таких як складні запити, відсутні в багатьох інших СУБД. Включає в себе потужні можливості для роботи з великими обсягами даних та складними транзакціями, а також підтримку JSON та інших NoSQL-функцій. Вважається однією з найкращих СУБД з точки зору дотримання стандартів.

Зважаючи на це, зокрема на відсутність необхідності купівлі ліцензії, прийнято рішення використовувати PostgreSQL.

Також Entity Framework Core надає можливість легкої інтеграції із PostgreSQL, що є перевагою оскільки EF Core надає можливість створення міграцій – це механізм, який дозволяє розробникам керувати змінами в базі даних за допомогою коду. Вони використовуються для версіонування схеми бази даних шляхом зберігання набору файлів з кодом, які відображають зміни, що були зроблені в моделі дани.

РОЗДІЛ 3
ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1. Опис основних бізнес-процесів

Нижче представлено схему основних бізнес-процесів розроблюваної системи. До ключових бізнес-процесів відносяться найважливіші варіанти використання, що описані на Use Case діаграмі на рисунку 1.3. Бізнес-процеси описані за допомогою нотації BPMN, що допомагає у складанні блок-схеми для опису покрокового виконання бізнес-процесу. Це допомагає визначити послідовність дій та зрозуміти взаємодії програмних модулів системи [10].

Нижче зображено процес реєстрації. Користувач вводить пароль та емейл та натискає на кнопку реєстрації, після чого генерується запит на клієнтську частину звідки на сервер. Сервер перевіряє введені дані та генерує відповідь, якщо вони не відповідають вимогам, якщо дані коректні, то сервер створює токен на підтвердження емейлу та відправляє запит на SMTP сервер для відправки листа користувачеві. Користувач отримує емейл, переходить по посиланню, після чого сервер отримує повідомлення про це та зберігає дані про користувача та повертає відповідь про успішну реєстрацію на фронтенд.

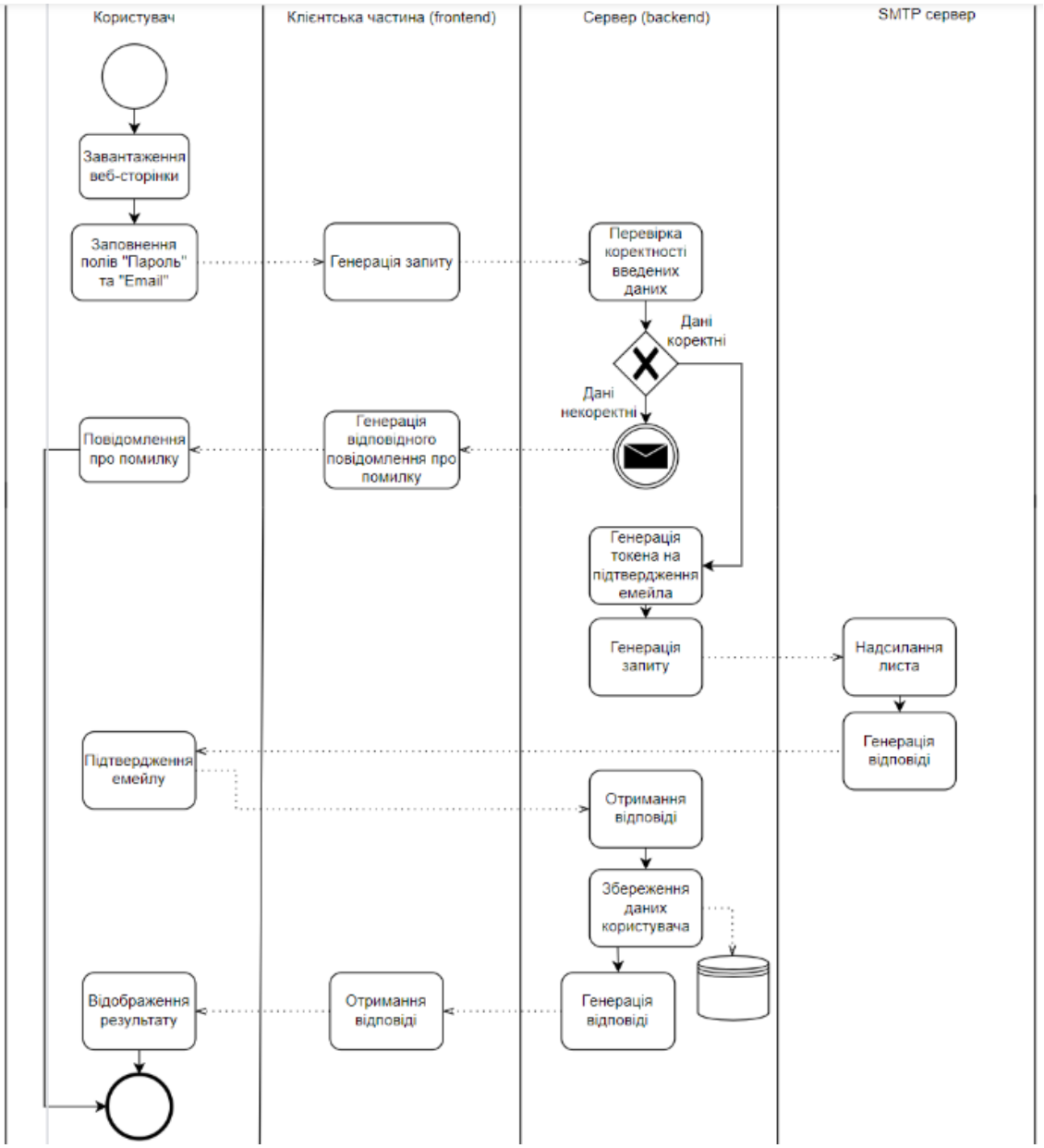

Рис. 3.1. Схема бізнес-процесу реєстрації

Нижче наведеному схему бізнес-процесу визначення індустрії та отримання рекомендацій. Ці два варіанти використання можна об’єднати в одну схему, оскільки єдина відмінність між ними – це різний тип запиту до OpenAI API.

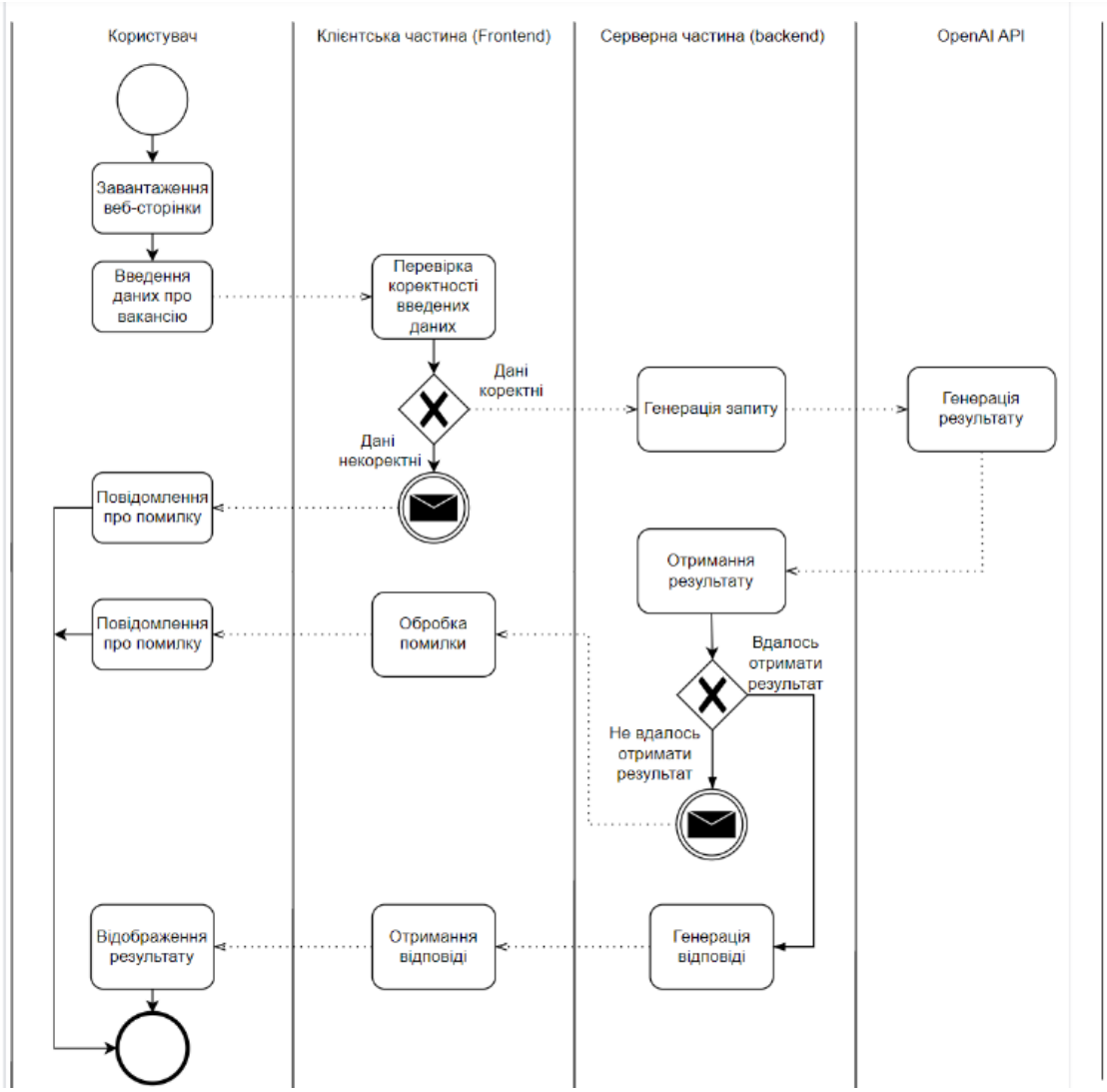

Рис. 3.2. Схема бізнес-процесу визначення індустрії та отримання рекомендацій за допомогою OpenAI API

Користувач заповнює інформацію про вакансію та натискаю на кнопку «Аналіз». Серверна частина виконує запит до OpenAI API з налаштованим prompt і у відповідь отримує результат, який повертає користувачеві. API повертає рекомендації щодо створення резюме та щодо поводження на співбесіді. Окремим запитом також API повертає індустрію вакансії.

На рисунку 3.3. зображено схему бізнес-процесу визначення переваг вакансії.

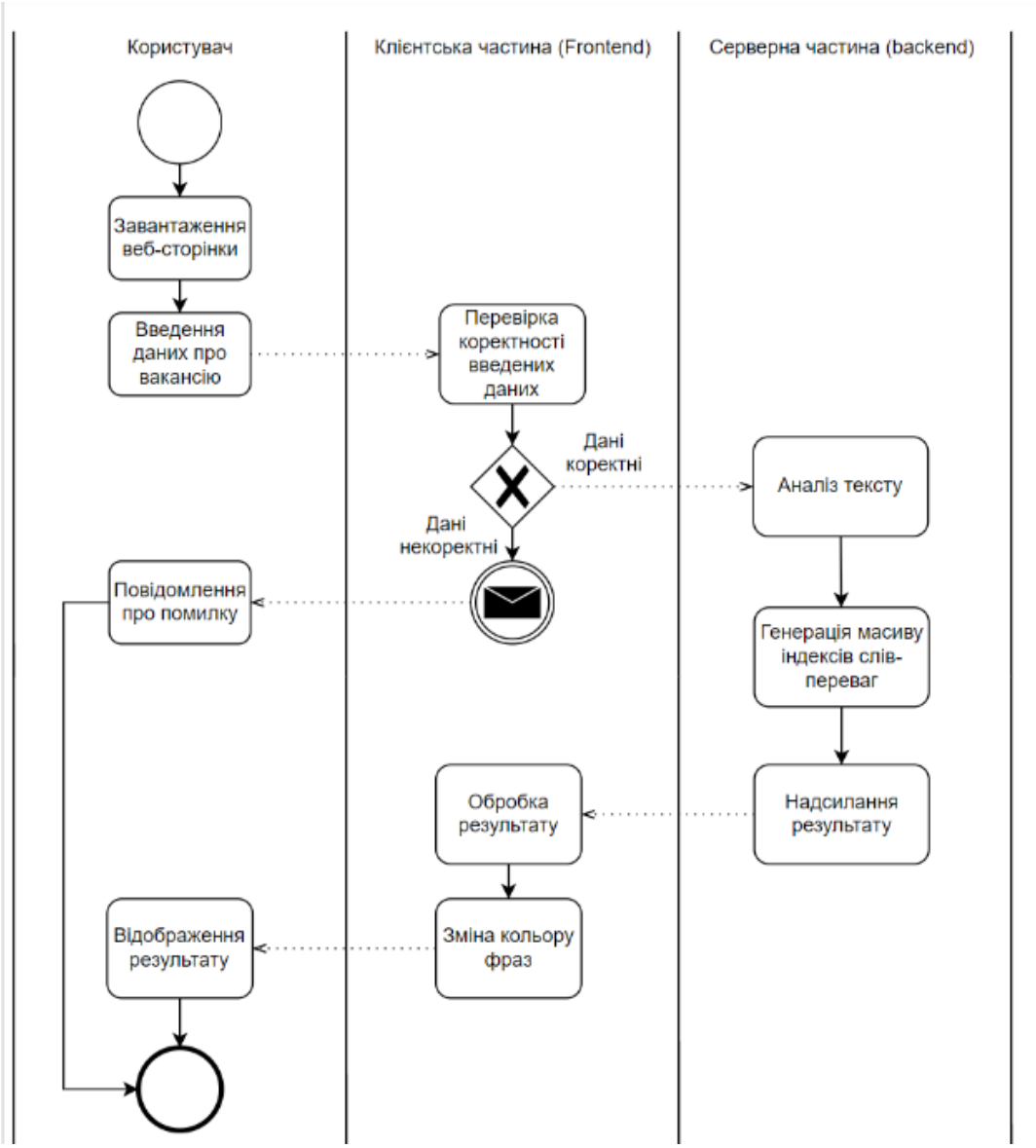

Рис. 3.3. Схема бізнес-процесу визначення переваг вакансії

Користувацька та frontend частина – аналогічна попереднім бізнес-процесам, різниця полягає в алгоритмі, який виконується на серверній частині. В тексті вакансії алгоритм шукає фрази, які визначені наперед на зберігаються в масиві та повертає індекси значень на фронтенд, де відбувається зміна кольору для даних фраз.

На рисунку 3.4. зображено схему бізнес-процесу зміни паролю. Користувач натискає на кнопку на сторінці входу «Забув пароль», вводить свій емейл, який використовувався під час реєстрації. Серверна частина

обробляє запит та створює токен для скидання пароля та створюється callback посилання, яке надсилається через SMTP сервер на почту, сервер прослуховує дане посилання і реагує на його відкриття. Після чого відкривається сторінка зміни паролю, користувач вводить новий пароль, на сервері відбувається валідація, якщо вона успішна, то зміни зберігаються в базу даних і користувач отримує повідомлення.

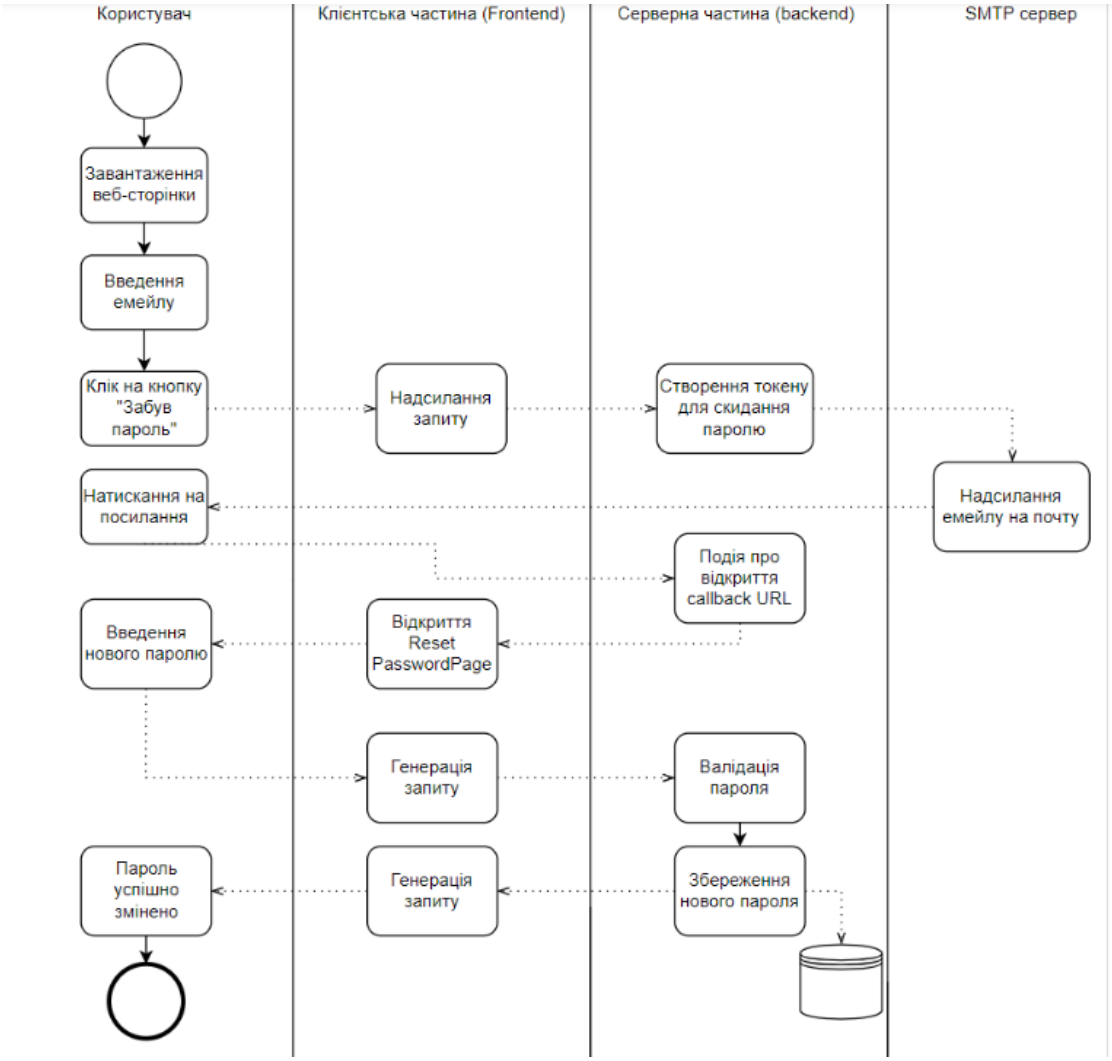

Рис. 3.4. Схема бізнес-процесу зміни пароля

Нижче зображено послідовну схему бізнес-процесу перегляду прикладів підозрілих вакансій. Користувач завантажує відповідну веб-сторінку після чого через клієнтську частину сервер отримує запит і

генерує список вакансій з бази даних, які повертає на клієнтську частину, де формується відображення для користувача. Відображається 100 вакансій за раз, для відображення наступних 100, користувач натискає на кнопку «Ще», після чого виконуються майже аналогічні дії, лише з урахуванням уже відображених вакансій.

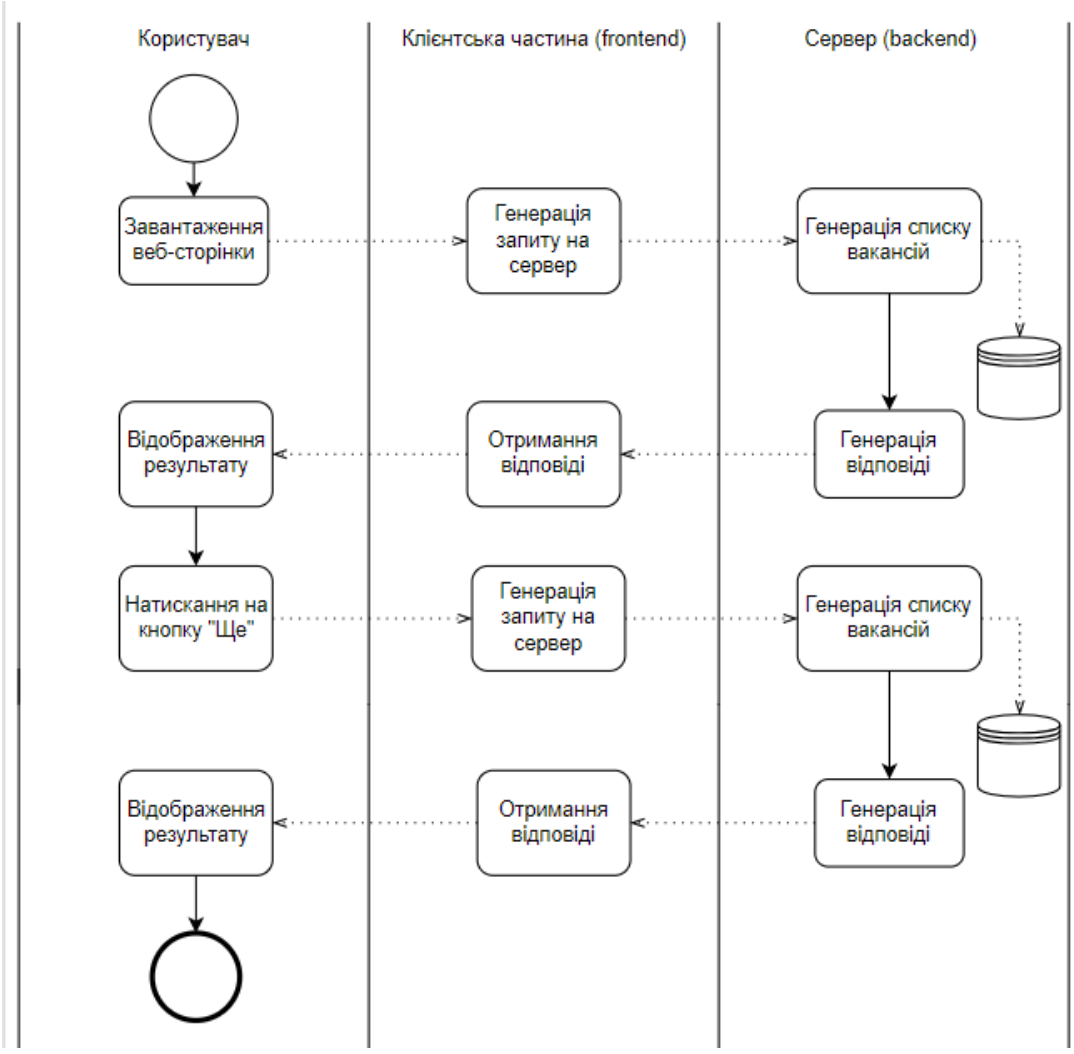

Рис. 3.5. Схема бізнес-процесу перегляду прикладів підозрілих вакансій

Вище описано основні бізнес-процеси веб-сервісу, інші варіанти використання дуже схожі своєю послідовністю кроків до бізнес-процесів, які зображені вище.

3.2. Архітектура програмного продукту

Важливим етапом розробки програмного забезпечення є вибір архітектури ПЗ. Існує багато підходів до розробки та проектування, можна виділити наступні архітектури: однорівнева(монолітна), мікросервісна, подійно-орієнтована, N-рівнева архітектура [2]. Для розробки веб-сервісу обрано трирівневу архітектуру, яка є однією з найпопулярніших. Це має як свої переваги та недоліки. До переваг можна віднести модульність, що дозволяє розділити програмний продукт на три рівні (презентаційний, бізнес-логіка застосунку та база даних), що дозволяє розробникам змінювати будь-який із трьох модулів без впливу на інші та масштабувати їх, також це дозволяє обирати різні технології для кожного рівня архітектури. Це в свою чергу також спрощує обслуговування, тестування, пошук та виправлення помилок і підтримку модулей завдяки розділенню відповідальності [29 с.15-20].

У свою чергу модульність також є недоліком, оскільки необхідна розробка та впровадження кожного рівня, що є більш складним підходом в порівнянні з альтернативами. Модульність також впливає на продуктивність через необхідність постійно «спілкуватись» та взаємодіяти рівням між собою, на цю взаємодію також негативно може впливати нестабільна мережа, що може відобразитися на стабільності та доступності програмного забезпечення.

Як зазначено вище, трирівнева архітектура включає три основні шари, які зображені на рисунку 3.6.:

- Презентаційний Рівень (User Interface Layer). Даний рівень відповідає за взаємодію з кінцевим користувачем. Він забезпечує інтерфейс через який користувачі можуть взаємодіяти з програмою. Презентаційний рівень може надавати інтерфейс у вигляді веб-сторінок, настільних застосунків, мобільних додатків тощо. Основна

мета цього рівня – трансформувати дані та результати виконання команд у формі, яка буде зрозумілою для кінцевого користувача.

- Рівень Бізнес-Логіки (Business Logic Layer). Цей шар керує бізнес-правилами та обробкою даних, які генеруються на презентаційному рівні. Він є відповідальним за виконання команд користувачів, роботу з даними та обчисленнями, які необхідні для реалізації функцій програми. Рівень бізнес-логіки також може виконувати валідацію даних, виконання бізнес-правил, виконання обчислень та інші завдання.
- Рівень Даних (Data Access Layer) відповідає за зберігання та управління даними. Це може бути реалізовано у різні способи як наприклад у вигляді бази даних, файлових систем, віддалені веб-сервіси або інші засоби зберігання. Він відокремлює бізнес-логіку від деталей реалізації зберігання даних, що дозволяє змінювати схеми зберігання без впливу на інші частини продукту.

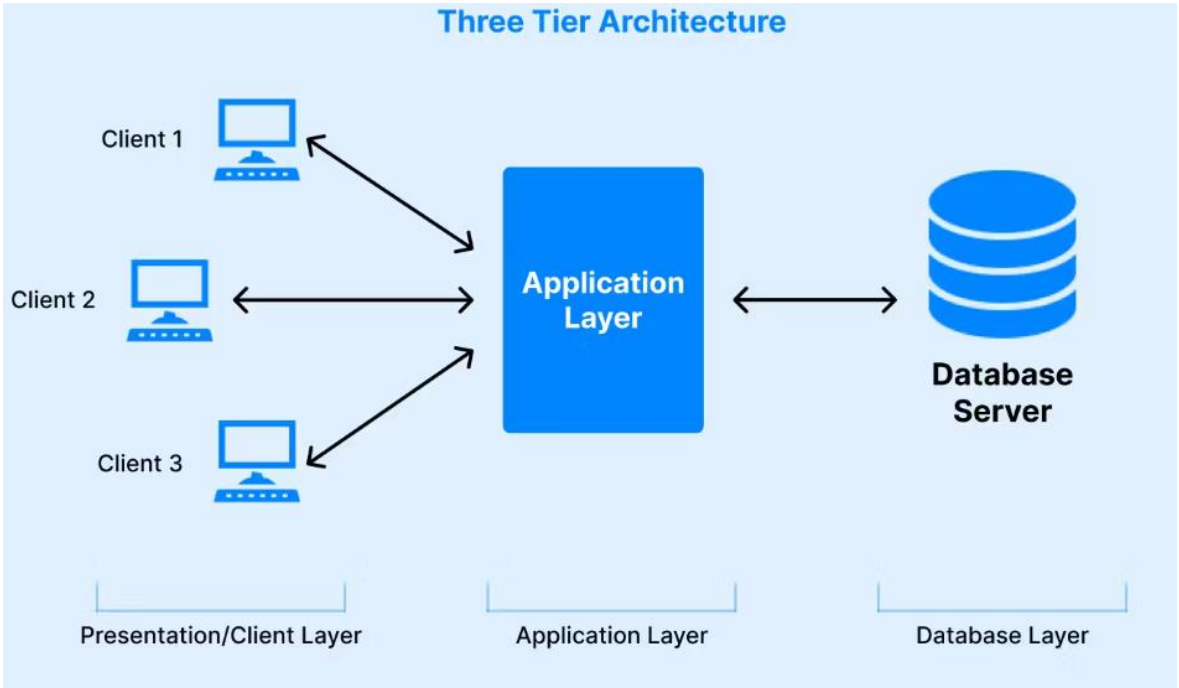

Рис. 3.6. Загальний випадок вигляду трирівневої архітектури [33]

Важливим аспектом є взаємодія та комунікація між рівнями. Взаємодія між рівнем відображення та рівнем бізнес-логіки відбуватиметься через HTTPS, що є розширенням від Hypertext Transfer Protocol (HTTP), але який вважається більш безпечним та загальноприйнятим стандартом взаємодії. Головною його перевагою є те, що всі дані шифруються за допомогою TSL або SSL також даний протокол гарантує конфіденційність та цілісність передачі даних. Даний протокол також застосовуватиметься для взаємодії між сервісом аналізу вакансій та рівнем бізнес-логіки.

Взаємодія між рівнем бізнес-логіки та рівнем даних відбуватиметься за допомогою Entity Framework Core, який є ORM фреймворком, що дозволяє працювати з базами даних без необхідності написання SQL коду. Це досягається завдяки можливості працювати із сутностями таблиць у вигляді класів об’єктно-орієнтованого програмування. Використання EF Core спрощує взаємодію з базою даних, усуваючи необхідність концентруватись на написанні SQL-коду та управлінні з'єднанням з базою даних.

На рисунку 3.7 власне зображено модулі системи та тип взаємодії між ними. Вирішено виділити аналіз вакансій в окремий сервіс, оскільки він може незалежно використовуватись як API в майбутньому, також це дозволяє застосувати технології, які краще підходять для аналізу та обробки великих масивів даних.

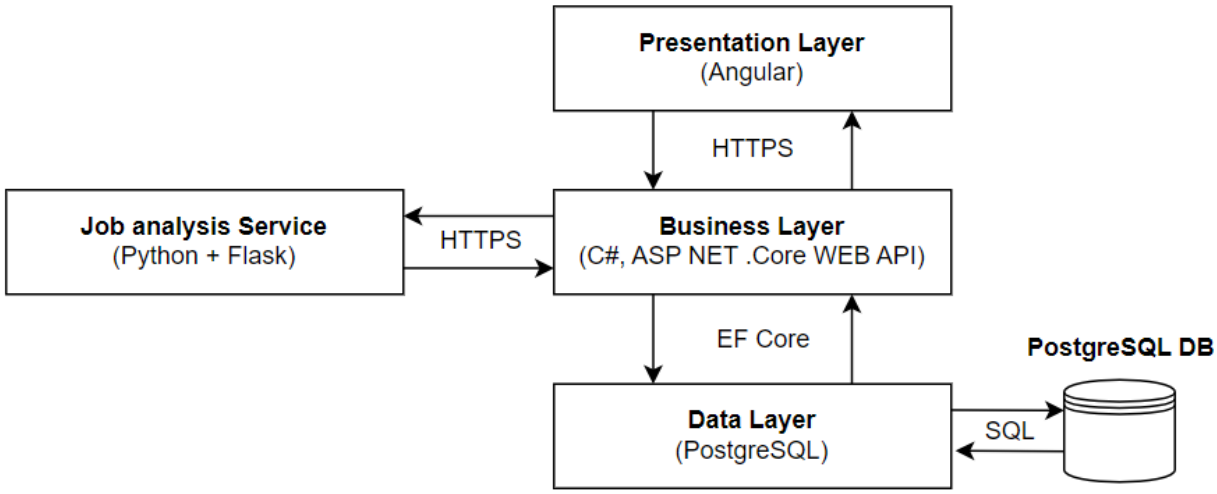

Рис 3.7. Схема рівнів розроблюваного програмного продукту та взаємодії між ними

Підсумовуючи, трирівнева архітектура забезпечує баланс між модульністю, гнучкістю та масштабованістю, однак вимагає ретельного планування та експертизи для ефективної реалізації.

3.3. Програмна реалізація сервісу аналізу вакансій

Даний сервіс має наступну структуру, що зображена на рисунку 3.8

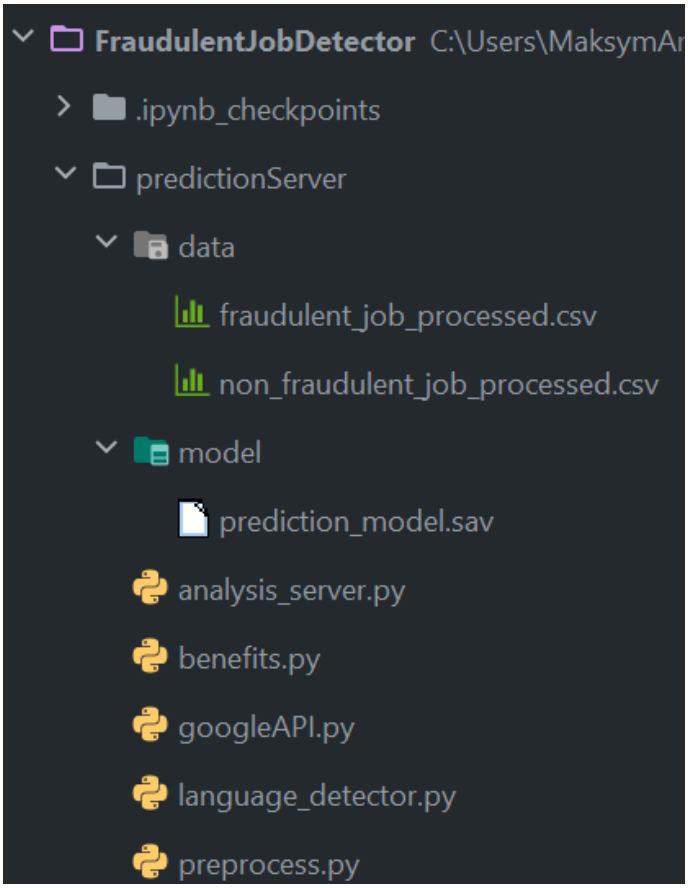

Рис. 3.8. Структура проекту аналізу вакансій у середовищі PyCharm

Для реалізації функції прогнозування потрібно створити модель «prediction_model.sav», перед цим обробивши дані та зберігши їх у файли в папці «data». Перш за все для цього потрібно отримати розмічений набір даних, який міститиме вакансії, що помічені як підозрілі, тобто потенційно шахрайські або ні. Дана інформація зберігається у двох файлах «fraudulent_job.csv» – 400000 записів та «non_fraudulent_job.csv» – 800000 записів.

Спочатку потрібно ці дані обробити, це відбувається за допомогою JupyterNotebook, що прискорює час необхідний на аналіз та фільтрацію даних. Після видалення дублікатів в файлах «fraudulent_job.csv», «non_fraudulent_job.csv» залишається близько 200000 та 520000 записів відповідно. Далі ці дані в «сирому» вигляді записуються в базу в таблицю JobsRaw.

Наступною проблемою є мова тексту. Для кожної мови необхідно будувати окрему математичну модель, оскільки кожна мова має свої унікальні властивості, форми та фрази. Оскільки застосунок розробляється для користувачів в Україні, вирішено виокремити лише вакансії українською мовою. Для цього використано бібліотеку «langdetect», яка визначає мову тексту, даний функціонал реалізовано окремою функцією у файлі «language_detector.py» Після застосування даної функції до описів вакансій отримано наступний розподіл за мовами:

- російська мова – 50% вакансій у наборі даних
- українська мова – 40% вакансій
- англійська мова – 7% вакансій
- інші мови – 3%

Наступним кроком є обробка тексту вакансій, для цього використовуються підходи видалення стоп-слів, стемінг, лематизація, токенізація. Токенізація – це розділення тексту на окремі слова, так звані токени. Для цього використовується бібліотека nltk. Лематизація – це зведення слів до їх базової форми (виділення леми слова). Для цього використовується pymorphy2. Стемінг – це видалення суфіксів зі слів для отримання їх кореневої форми. Ця функція реалізована за допомогою класу UkrainianStemmer, який поширюється на GitHub з MIT ліцензією [13]. Це реалізовано у файлі «preprocess.py» у вигляді функції preprocess, яка повертає масив токенів для кожної вакансії.

Ще одна проблема, яка потребує вирішення – незбалансована кількість даних в наборах. 208000 записів зі значенням IsFraudulent = 0 та 80000 записів зі значенням IsFraudulent = 1. Це вирішується завдяки методу oversampling – це техніка, що використовується для вирішення проблеми незбалансованих даних, коли кількість зразків одного класу менша порівняно з іншими класами. Це особливо актуально в класифікаційних задачах, де незбалансованість класів може призвести до

поганої продуктивності моделі, особливо для менш представлених класів. Основні цілі та переваги використання oversampling:

- збільшуючи кількість зразків менш представлених класів, oversampling сприяє більш збалансованому розподілу класів, що може покращити продуктивність моделі;
- моделі, навчені на збалансованих даних, зазвичай краще визначають характеристики менш представлених класів, що може підвищити точність класифікації;
- коли деякі класи переважають, моделі можуть вивчити упередженість на користь цих класів. Oversampling допомагає зменшити таку упередженість.

Для цього використовується метод простого повторення, тобто копіювання існуючих зразків з менш представлених класів. Далі відбувається розбиття набору на дані для навчання та тестування у співвідношенні 4 до 1. Наступним кроком текстові дані перетворюються у числові вектори за допомогою функції CountVectorizer(). Дана функція створює «мішок слів» (bag of words), де кожен унікальний токен з текстового набору даних представлений як окрема функція, а значення функції – кількість повторень даного токена в тексті. Для прогнозування було обрано метод логістичної регресії, який показав найкращу точність. Далі ця модель була збережена у файл із розширенням «.sav».

«analysis_server.py» – це основний файл сервісу, який оголошує методи API:

- POST /predict отримує в тілі запиту інформацію про вакансію, далі завантажує модель для прогнозування із файлу «prediction_model.sav», визначає мову, якщо мова українська, то функція передає опис вакансії в модель і отримує результат прогнозування або повертає помилку;

- POST /detect-benefits отримує опис вакансії та викликає функцію get_benefits() із файлу «benefits.py», яка обробляє текст та шукає в описі вакансії фрази, що відповідають за переваги вакансій. Даний масив переваг визначений наперед завдяки аналізу вакансій та пошуку переваг в Інтернеті. Даний метод повертає індекси переваг;
- POST /detect-industry метод, що генерує запит до OpenAI із підказкою «На основі наступного опису роботи визнач індустрію вакансії: {job_description}»;
- POST /get-advice даний метод також генерує запит до OpenAI, де описується, що відповідь на даний запит має бути у вигляді порад для пошукача на що звернути увагу при складанні резюме та при проходженні співбесіди.

3.4. Програмна реалізація серверної частини

На рисунку 3.8. зображена структура проекту, використовуючи інтерфейс IDE Rider

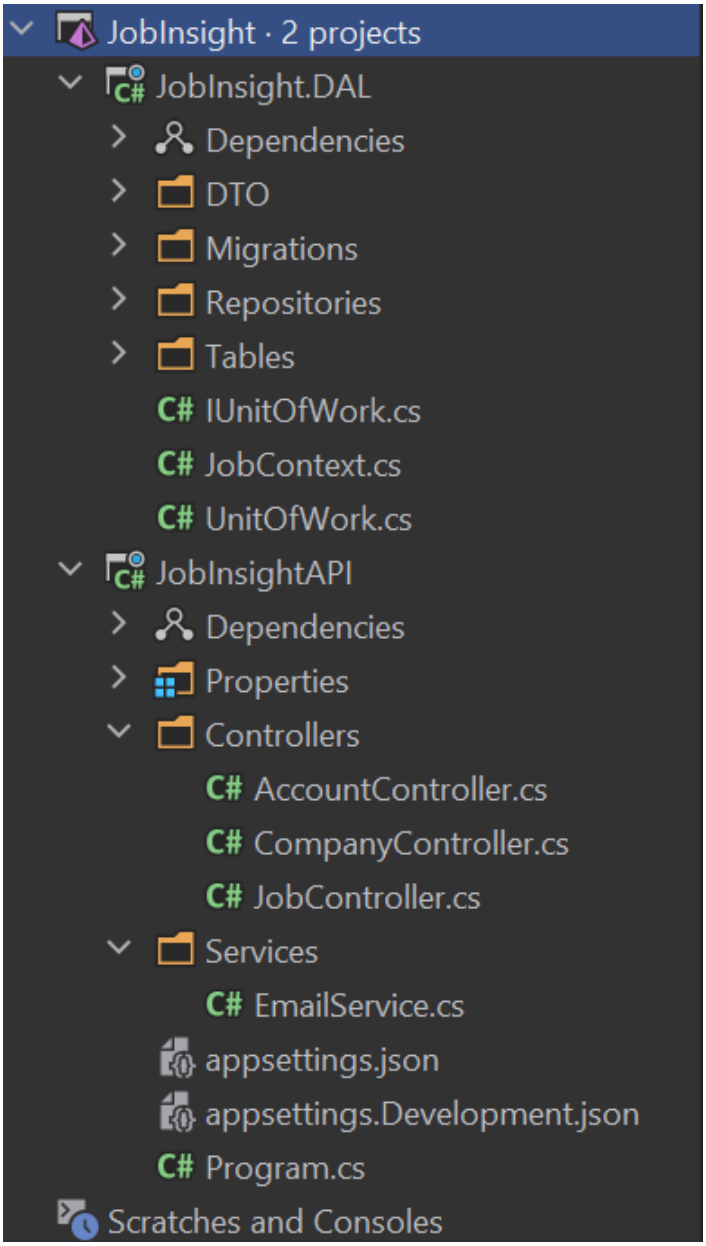

Рис. 3.8. Структура проекту серверної частини

Проєкт складається із двох основних частин: JobInsight.DAL, JobInsightAPI, що виконують функції рівня доступу до даних та рівня бізнес-логіки відповідно.

Основним класом частини JobInsight.DAL є JobContext, який успадковується від IdentityDbContext<IdentityUser>, що є одним із основних інтерфейсів EF Core. JobContext використовується для взаємодії з базою даних, він представляє собою сесію з базою даних, що дозволяє

робити запити до БД. JobContext складається із властивостей типу DbSet, кожен з яких представляє клас, що відповідає сутності, а точніше таблиці у базі даних. Дані класи зберігаються у папці Tables.

Для більш зручної та ефективної взаємодії між рівнем бізнес-логіки та даними реалізовано патерни Unit of Work та Repository. Патерн Repository базується на принципах абстракції даних, що допомагає відокремити логіку доступу до даних від бізнес-логіки додатку та інкапсуляції даних, коли всі запити до БД інкапсульовані всередині Repository, що спрощує модифікацію коду. Клас DbSet можна розглядати як реалізацію патерну Repository, але власна реалізація дозволяє імплементувати додаткову логіку фільтрації, збереження чи агрегації даних.

У свою чергу Unit of Work відповідає за цілісність даних, що досягається керуванням транзакціями. Керування транзакцій передбачає, що зміни в БД відбудуться лише в тому випадку, якщо всі зміни всередині однієї бізнес-операції будуть успішно виконані. Це відбувається завдяки виклику методу DbContext.SaveChange. Це гарантує так звану атомарність операцій. Також Unit of Work виступає єдиною точкою взаємодії з даними, що гарантує, що будь-який ресурс незалежно від часу, матиме доступ до однакової версії даних.

На рисунку 3.9. зображено схеми взаємодії контролерів та джерела даних у випадку використання патернів Unit of Work та Repository та у випадку їх відсутності. На перший погляд можна зробити висновок, що схема без використання даних шаблонів виглядає більш зрозуміло та простіше. Проте зі збільшенням кількості таблиць у базі даних, зі збільшенням кількості тестів та бізнес-логіки, варіант без патернів програє, оскільки підтримувати такий код буде дуже складно і допустити помилку стає легше. І хоча DbSet та DbContext відтворюють принципи

даних патернів, відсутність можливості імплементації власної логіки є мінусом.

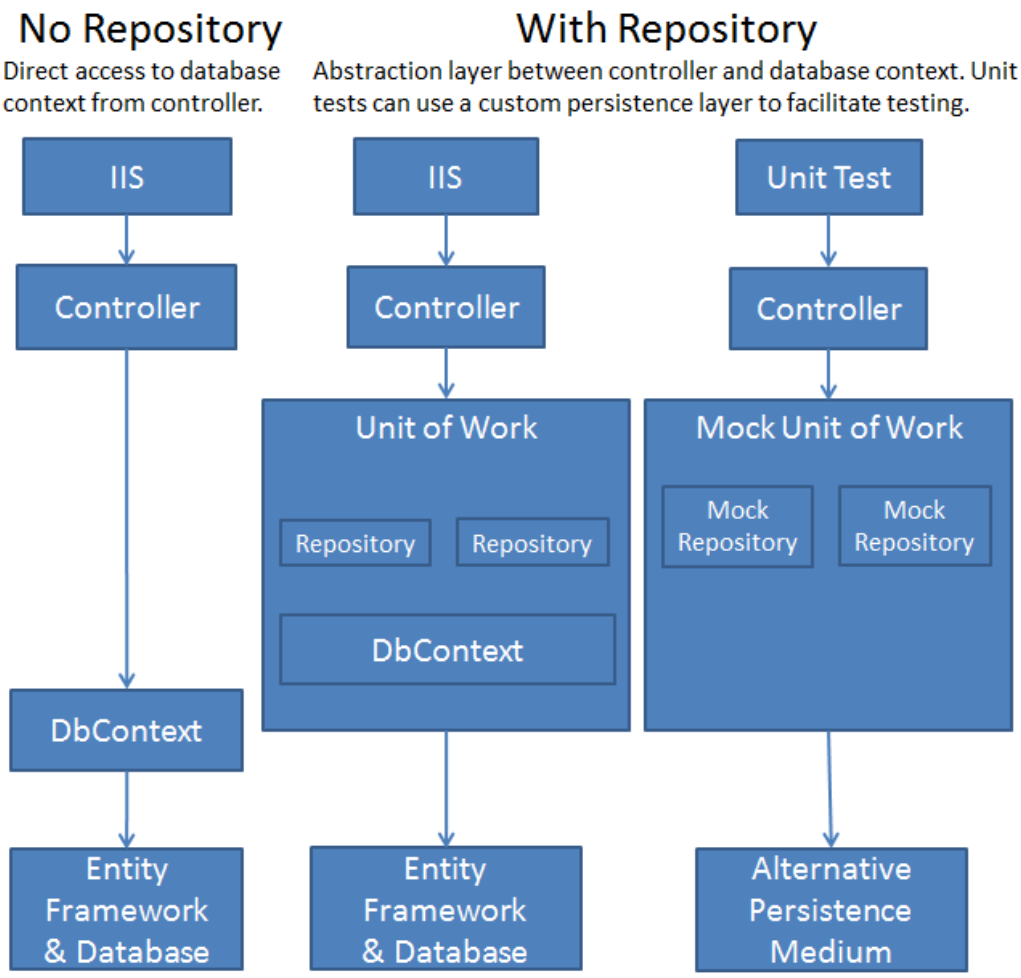

Рис. 3.9. Різні схеми взаємодії між контролером та базою даних [16]

Дані між рівнями передаються за допомогою класів DTO (Data transfer object). Даний тип класів допомагає зменшити зв’язність між частинами системи. Дані об’єкти використовуються виключно для передачі даних між рівнями. Це потрібно для чого, оскільки наприклад дані про вакансію, які отримує рівень бізнес-логіки можуть відрізнятись від даних, які зберігаються в базу. Відповідні класи зберігаються в папці проекту із назвою DTO.

Другою частиною проекту є JobInsightAPI, яка відповідає за роботу контролерів, які містять усю необхідну бізнес-логіку та конфігурацію WebAPI.

API містить 3 контролери: AccountController, CompanyController, JobController.

AccountController відповідає за реєстрацію, авторизацію, зміну пароля та підтвердження емейлу та складається із наступних методів, які зображено на рисунку 3.10.

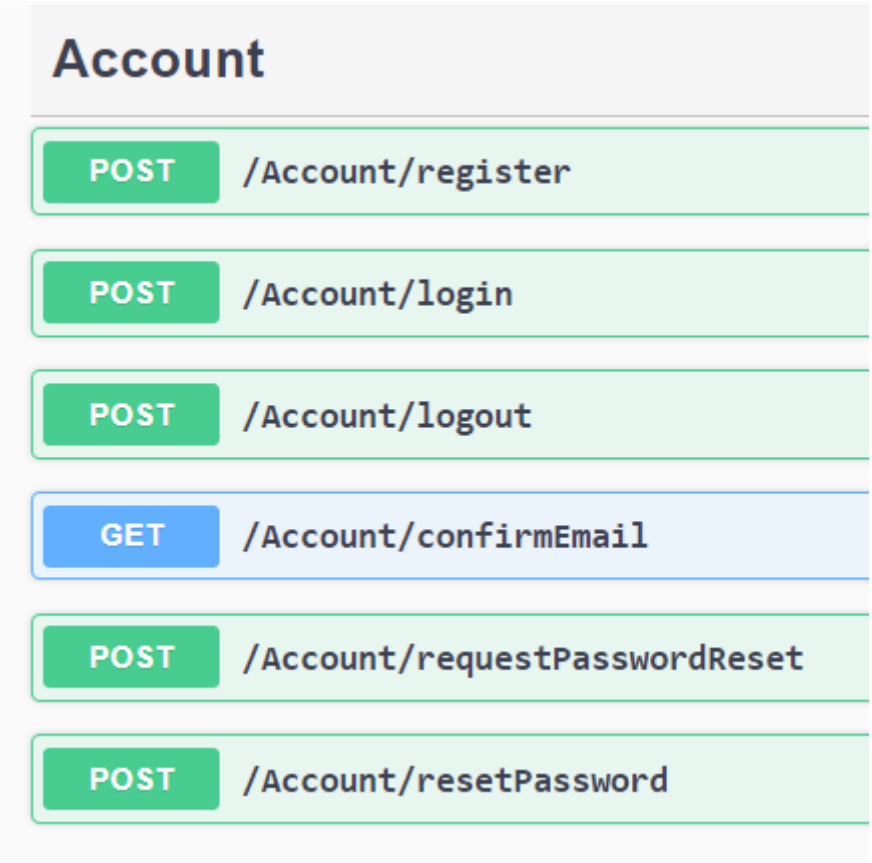

Рис. 3.10. Методи AccountController у середовищі Swagger

- Post /Account/register метод, який відповідає за реєстрацію користувача. З тіла запиту через HTTPS у метод передається емейл та пароль, після чого створюється новий об’єкт сутності IdentityUser, якщо пароль відповідає всім вимогам і такий користувач ще не існує

в базі, то наступним кроком створюється токен для підтвердження паролю. Даний токен передається через Callback посилання, яке надсилається через SMTP сервер користувачеві на його вказану поштову скриньку. У разі натискання на посилання користувач створює запит на метод /Account/confirmEmail;

- Get /Account/confirmEmail метод, що приймає токен та UserId, перевіряє дані параметри на валідність та у разі успішної перевірки підтверджує емейл користувача з відповідним UserId;
- Post /Account/login метод, що відповідає за авторизацію користувача. Метод отримує з тіла запиту емейл та пароль та перевіряє чи даний користувач вже зареєстрований і чи підтверджений у нього емейл, після чого викликається SignInManager, якщо пароль правильний, то далі відбувається процес створення JWT токена;
- Post /Account/logout даний метод з допомогою метода SignOutAsync використовується для виходу користувача, який виконав запит із системи;
- Post /Account/requestPasswordReset один із двох методів, який використовується для зміни пароля. Даний метод отримує емейл користувача з тіла запиту, перевіряє наявність такого користувача в базі, якщо користувач існує, створюється токен на скидання пароля, який передається через Callback посилання. В разі натискання на дане посилання відкриється відповідна сторінка, де користувач зможе ввести новий пароль;
- Post /Account/resetPassword метод, який відповідає власне за зміну пароля. Він приймає з тіла запиту токен, емейл та новий пароль. Якщо передані дані валідні, то метод ResetPasswordAsync класу UserManager.

CompanyController відповідає за роботу із сутністю компанії. На даний момент він містить один метод для пошуку вебсайту компанії.

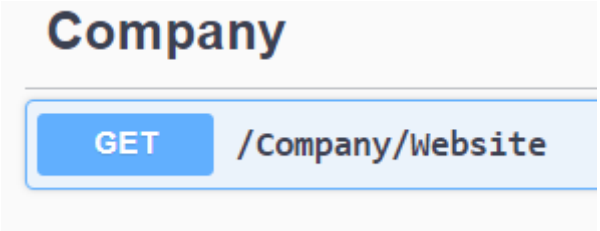

Рис. 3.11. Методи CompanyController у середовищі Swagger

Get /Company/Website метод приймає назву компанії та виконує запит до зовнішнього API, а саме до пошукового API Google. GoogleAPI повертає результат пошукового запиту, посилання на вебсайт компанії отримується з першого результату, оскільки він вважається найбільш релевантним.

JobController відповідає за роботу із сутністю вакансії. Це може бути список підозрілих вакансій, результат прогнозування, пошук вакансій, переваги вакансії, індустрія вакансії, поради на основі даних про вакансію, додавання нової, отримання і видалення по id.

Рис. 3.12. Методи JobController у середовищі Swagger

- Get /Job/ScamExample даний метод використовуючи об’єкт UnitOfWork отримує з бази даних 100 вакансій і повертає їх у форматі JSON;
- Post /Job/Prediction метод приймає дані про вакансію та генерує запит до сервісу аналізу вакансій. Після чого повертає отриману відповідь користувачеві;
- Post /Job/SearchResult приймає запит користувача (регіон та пошуковий запит) після чого генерує запит до стороннього JoobleAPI, яке повертає список вакансій, які відповідають умовам пошуку;

- Post /Job/Benefits виступає у ролі посередника. Приймає параметри у вигляді опису вакансії та генерує запит до сервісу аналізу вакансій, що відповідає за визначення переваг вакансії;
- Post /Job/Indsutry діє за схожим принципом як попередній метод. Він приймає опис вакансії та викликає відповідний метод сервісу аналізу вакансій, який відповідає за визначення індустрії. Після чого генерується відповідь з результатом для користувача;
- Post /Job/Advice також викликає метод сервісу аналізу вакансій для створення списку порад. Відповідає за отримання параметрів, генерацію запиту та надсилання результату;
- Post /Job/Add отримує вакансію та зберігає її в базі даних;
- Post /Job/{id} отримує id вакансії та виконує пошук по id в базі даних;
- Delete /Job/{id} отримує id вакансії та виконує видалення по id в базі даних.

Окремим сервісом в даному проекті реалізований EmailService, який відповідає за надсилання листів користувачам на поштову скриньку. На дани Для цього використовується SMTP сервер від провайдер Brevo.com, який пропонує функцію «Transactional email», що дозволяє надсилати до 300 листів на день безкоштовно [9]. Для цього потрібно зареєструватись на сайті та отримати посилання на smtp сервер, логін та пароль для авторизації.

Файл Program.cs є точкою входу в застосунок, він відповідає за конфігурацію та старт застосунку. В даному файлі відбувається налаштування та реєстрацію Dependency Injection, як зображено на рисунку 3.13. Цей механізм забезпечує, що екземпляри даних служб будуть автоматично надані класам, які цього потребують.

```
builder.Services.AddTransient<IJobRepository, JobRepository>();
builder.Services.AddTransient<IBadJobRepository, BadJobRepository>();
builder.Services.AddTransient<ISearchQueryRepository, SearchQueryRepository>();
builder.Services.AddTransient<ISearchQueryJobRepository, SearchQueryJobRepository>();
builder.Services.AddTransient<ISearchQueryRepository, SearchQueryRepository>();
builder.Services.AddTransient<IUnitOfWork, UnitOfWork>();
builder.Services.AddTransient<IEmailService>(x IServiceProvider =>
    new EmailService(
        smtpHost: builder.Configuration["Brevo:smtpHost"],
        smtpPort: 465,
        mailjetApiKey: builder.Configuration["Brevo:ApiKey"]
        ));
```

Рис. 3.13. Налаштування ін'єкції залежностей проєкту

Крім цього в даному файлі налаштовується підключення до бази даних, додаються контролери, Swagger, налаштовуються правила валідації паролів та JWT.

Також варто звернути увагу на забезпечення безпеки даних. Робота з паролями користувачів несе в собі небезпеку компрометації даних. Саме тому важливою складовою є забезпечення безпеки передачі та зберігання конфіденційної інформації. Можна виокремити два важливих аспекти: передача конфіденційних даних з клієнтської частини на бекенд та збереження цих даних. Безпека під час передачі забезпечується HTTPS, передача в тілі запиту пароля, який використовує HTTPS запит вважається загальноприйнятою практикою. Це гарантує, що усі дані, передані між браузером користувача та сервером, будуть зашифровані. Також важливо є слідкувати, щоб паролі не зберігались в локальному сховищі чи куках на фронтенді. На сервері паролі одразу хешуються з використанням надійного алгоритму. AspNetCore Identity надає можливість використовувати алгоритм PBKDF2 (Password-Based Key Derivation Function 2). Цей алгоритм є стандартом для безпечного зберігання паролів і рекомендується багатьма експертами в області кібербезпеки. Процес хешування за допомогою даного алгоритма складається з наступних

кроків. Спочатку відбувається комбінування пароля та солі. Сіль є унікальною для кожного пароля, що забезпечує, що навіть ідентичні паролі хешуватимуться у різні хеші, що робить даний алгоритм стійким до атак методом райдужних таблиць. Далі об'єднаний пароль та сіль подаються на вхід хеш-функції. Хеш-функція застосовується багато разів (за замовчуванням йдеться про 10000 ітерацій). Цей процес витратний з точки зору обчислень, що робить атаки грубою силою менш ефективними. Після завершення всіх ітерацій отримується кінцевий хеш, який може бути збережений у базі даних [26].

3.5. Програмна реалізація клієнтської частини

Клієнтська частина представлена у вигляді наступної структури, яка зображена на рисунку 3.14.

Головні модулі знаходяться в папках «Components», «Models», «Services» та у файлах «app.component.html», «app.module.ts», «app-routing.module.ts», «Index.html».

Папка «Components» містить компоненти, які є ключовими будівельними блоками. Кожна підпапка містить три файли, а саме .css файл, який відповідає за стилі компонента; .html файл, що відповідає за розмітку блока; та .ts файл, що відповідає за виконання бізнес-логіки компонента, валідацію даних. Нижче детальніше описано призначення кожного компонента.

- Job-Example відповідає за схожих вакансій, це відбувається завдяки виклику SimilarVacanciesService, що виконує запит до методу /Job/SearchResult, у вигляді параметрів передаються заголовок вакансії, яку користувач ввів з метою її аналізу;
- Job-Input відповідає за вигляд форми для введення інформації про вакансію та за валідацію даних та виклик методів відповідних сервісів після кліку на кнопку «Аналіз»;

- Job-Search компонент відповідає за форму для пошуку і відображення вакансій та генерація запиту до серверу для отримання вакансій, які відповідають запиту користувача;
- Job-Serp компонент, який вбудований в компонент JobSearch, відповідає за відображення списку вакансій;
- Login-Page компонент, що містить в собі компоненти форми реєстрації та входу і можливість переключитись між цими опціями;
- Logout потрібен для функції виходу з акаунту;
- Main-Page основна сторінка застосунку, що містить в собі компоненти JobInput та JobSerp, та відображає результати аналізу;
- Nav-Bar відображає навігаційну панель зверху сайту та забезпечує навігацію між сторінками. Також відображає емейл акаунту, якщо користувач авторизований та можливість вийти за допомогою компоненту Logout;
- Prediction-Result відповідає за відображення результату прогнозування чи є вакансія підозрілою;
- Reset-Password окрема сторінка, яка відкривається після натискання на посилання в листі про зміну пароля;
- Sign-In компонент відповідає за форму авторизації та за виклик відповідних запитів для авторизації;
- Sign-Up потрібен для форми реєстрації користувачів.

У папці «Models» зберігаються класи TypeScript, які використовуються для передачі даних. Сюди входять Job, що відповідає за сутність вакансії; JobAnalysis, що містить сутність Job та окремо властивість для результату прогнозування. SearchQuery сутність, що використовується для передачі даних про пошуковий запит користувача.

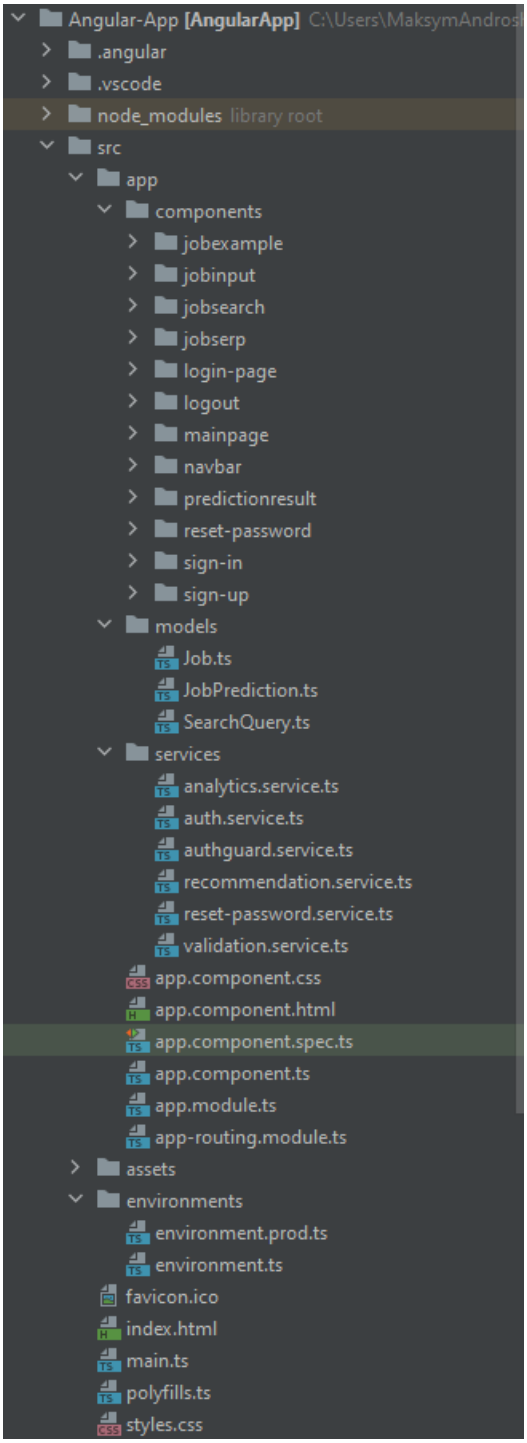

Рис. 3.14. Структура проекту клієнтської частини

У папку «Services» винесені окремі сервіси, які виконують певну бізнес-логіку, яка може використовуватись в різних компонентах.

- Сервіс auth відповідає за реєстрацію, автентифікацію, збереження JWT токенів. Він використовує методи контролера Account.
- AuthGuard сервіс відповідає за авторизацію, тобто перевірку чи має користувач доступ до певних ресурсів застосунку. Він працює у взаємодії з сервісом auth, а саме його методом isAuthenticated.
- Analytics сервіс, що реалізує взаємодію з сервером, а саме контролером Job, що відповідає за обробку даних, які стосуються вакансії.
- SimilarVacancies сервіс виконує роль створення запитів для отримання списку вакансій на пошуковий запит користувача.
- Reset-Password допомагає із організацією запитів щодо відновлення пароля.
- InputValidation сервіс містить в собі логіку валідації полів різних форм застосунку.

Така практика відділення загальної бізнес-логіки окремо від компонентів є загальноприйнятою і дозволяє перевикористати вже реалізовані методи та зменшує час, необхідний для внесення змін в програмний код. Відсутність таких сервісів змусила б дублювати логіку в кожному компоненті системи, що значно впливає на швидкість розробки зі зростанням кількості програмного коду.

Файл app-routing.module.ts відповідає за конфігурацію маршрутів веб-застосунку, що зображено на рисунку 3.15.

```
const routes: Routes = [
  {
    path: "login", component: LoginPageComponent
  },
  {
    path: "", component: PageComponent,  canActivate: [AuthGuardService]
  },
  {
    path: "job", component: JobSearchComponent,
  },
  {
    path: "example", component: JobExampleComponent,
  },
  {
    path: 'reset-password', component: ResetPasswordComponent
  }
]
```

Рис. 3.15. Маршрутизація застосунку

На рисунку зображено шлях та компонент, який виступає головним при переході по даному маршруту. Властивість canActivate забезпечує необхідність авторизації для доступу до маршруту. Це дозволяє забезпечити обов'язкову авторизацію для доступу до головної сторінки.

Файл app.module.ts визначає, які компоненти, сервіси та директиви будуть доступні в межах застосунку. Це дозволяє визначати, які частини коду повинні бути використані та як вони взаємодіють між собою.

3.6. Структура бази даних

База даних має наступні таблиці:

- BadVacancies зберігає вакансії для яких не вдалось провести аналіз;
- Vacancies зберігає дані про вакансії, які попередньо оброблено. Сюди також зберігаються вакансії, які вводить користувач при аналізі;

- VacanciesRaw зберігає початковий розмічений набір даних, що представляє собою набір вакансій, на основі яких створено модель для прогнозування після їх обробки;
- UserSearchQueries зберігає пошукові запити користувачів, які були зроблені в застосунку;
- UserSearchQueryJobs зберігає вакансії, які були отримані із стороннього API як відповідь на пошуковий запит користувача. В подальшому їх можна використати для покращення алгоритмів аналізу вакансій;
- UserSession зберігає інформацію про сесії користувачів;
- AnalysisResults дана таблиця містить інформацію про результати аналізу вакансій та тип аналізу (пошук індустрії, надання порад, прогнозування чи вакансія підозріла).

Наступні таблиці створені за допомогою AspNetCore Identity:

- AspNetRoleClaims ця таблиця дозволяє призначати певні пари «ключ-значення» безпосередньо до ролей, які потім наслідуються всіма користувачами цих ролей;
- AspNetRole – це таблиця для зберігання ролей. Вона використовується для визначення різних ролей у системі, наприклад «адміністратор», «користувач» тощо, та присвоєння цих ролей користувачам;
- AspNetUserClaims – таблиця для зберігання пар «ключ-значення», які містять специфічну інформацію про користувача, наприклад, його повне ім'я або дату народження;
- AspNetUserLogins ця таблиця дозволяє користувачам використовувати кілька методів входу для одного акаунта.
- AspNetUserRoles – ця таблиця пов’язує користувачів та ролі. Вона визначає, які ролі має кожен користувач;

- AspNetUsers зберігає основну інформацію про користувачів, включно з їхніми іменами, електронними адресами, хешами паролів, та іншою інформацією, яка ідентифікує кожного користувача в системі;
- AspNetUserTokens. Тут зберігаються токени безпеки, пов'язані з користувачами, наприклад, токен для скидання пароля або токени підтвердження електронної пошти.

3.7. Тестування

Розрізняють кілька підходів до тестування як тестування чорного ящика, білого ящика та сірого ящика. Тестування чорного ящика передбачає, що тестувальник не знає внутрішньої структури системи, цей тип тестування фокусується на тому, що програма робить, але не як вона це робить. Це дозволяє виявити невідповідності між очікуваною та реальною поведінкою системи, але цим способом неможливо виявити внутрішні проблеми коду. В свою чергу метод білого ящика передбачає тестування власне програмних модулів, іншими словами тестування відбувається всередині системи, що допомагає виявити внутрішні помилки. Метод сірого ящика – це комбінаціях цих двох підходів. Для тестування обрано метод чорного ящика.

Таблиця 3.1 – Опис тестового випадку

Назва тесту	Перевірка функціоналу реєстрації з коректними даними.
Покрокова схема тесту	1. Відкрити сайт у браузері 2. Натиснути на кнопку «Sign up now» 3. Ввести емейл та пароль 4. Натиснути на кнопку «Sign up»

	5. Перейти на поштову скриньку та натиснути на кнопку підтвердження емейлу.
Очікуваний результат	Користувач успішно зареєстрований в системі. Тепер користувач може увійти на веб-сайт, використовуючи дані для входу, що були вказані під час реєстрації.
Отриманий результат	Користувач успішно зареєстрований в системі. Тепер користувач може увійти на веб-сайт, використовуючи дані для входу, що були вказані під час реєстрації.

Таблиця 3.2 – Опис тестового випадку

Назва тесту	Перевірка функціоналу реєстрації при веденні некоректного емейлу
Покрокова схема тесту	1. Відкрити сайт у браузері 2. Натиснути на кнопку «Sign up now» 3. Ввести некоректний емейл, наприклад без символу «@» 4. Натиснути на кнопку «Sign up»

Продовження таблиці 3.2

Очікуваний результат	Користувач отримує повідомлення про некоректно введений емейл
Отриманий результат	Користувач отримує повідомлення про некоректно введений емейл

Таблиця 3.3 – Опис тестового випадку

Назва тесту	Перевірка функціоналу реєстрації з порожніми полями

Покрокова схема тесту	1. Відкрити сайт у браузері 2. Натиснути на кнопку «Sign up now» 3. Натиснути на кнопку «Sign up»
Очікуваний результат	Користувач отримує повідомлення про те, що поля є порожніми і необхідно їх заповнити
Отриманий результат	Користувач отримує повідомлення про те, що поля є порожніми і необхідно їх заповнити

Таблиця 3.4 – Опис тестового випадку

Назва тесту	Перевірка функціоналу входу при введенні коректних даних
Покрокова схема тесту	1. Відкрити сайт у браузері 2. Ввести емейл та пароль, що використовувались при реєстрації 3. Натиснути на кнопку «Sign in»
Очікуваний результат	Користувач отримує повідомлення успішну авторизацію

Продовження таблиці 3.4

Отриманий результат	Користувач отримує повідомлення успішну авторизацію

Таблиця 3.5 – Опис тестового випадку

Назва тесту	Перевірка функціоналу входу при введенні з неправильного пароля
Покрокова схема тесту	1. Відкрити сайт у браузері 2. Ввести емейл та неправильний пароль 3. Натиснути на кнопку «Sign in»
Очікуваний результат	Користувач отримує повідомлення про те, що

	пароль неправильний
Отриманий результат	Користувач отримує повідомлення про те, що пароль неправильний

Таблиця 3.6 – Опис тестового випадку

Назва тесту	Перевірка функціоналу виходу з акаунту
Покрокова схема тесту	1. Відкрити сайт у браузері 2. Зайти в існуючий акаунт 3. Натиснути в правому верхньому куті на кнопку «Logout»
Очікуваний результат	Користувач не матиме доступу до ресурсів, що потребують авторизації
Отриманий результат	Користувач не матиме доступу до ресурсів, що потребують авторизації

Таблиця 3.7 – Опис тестового випадку

Назва тесту	Перевірка функціоналу відновлення пароля
Покрокова схема тесту	1. Відкрити сайт у браузері 2. Ввести емейл 3. Натиснути на кнопку «Forgot password?» 4. Перейти по посиланню для відновлення пароля, який відправлений на поштову скриньку 5. Ввести новий пароль на сторінці, яка відкриється після натискання на посилання
Очікуваний результат	Пароль для входу в акаунт успішно змінено.
Отриманий результат	Пароль для входу в акаунт успішно змінено.

Таблиця 3.8 – Опис тестового випадку

Назва тесту	Перевірка функціоналу аналізу вакансій
Покрокова схема тесту	1. Авторизуватись в системі 2. Заповнити поля «Опис», «Назва компанії», «Заголовок» 3. Натиснути на кнопку «Аналіз»
Очікуваний результат	Користувач отримує результат аналізу, що включає в себе пошук вебсайту, визначення переваг та індустрії вакансії, прогнозування чи вакансія є підозрілою та список порад.
Отриманий результат	Користувач отримує результат аналізу, що включає в себе пошук вебсайту, визначення переваг та індустрії вакансії, прогнозування чи вакансія є підозрілою та список порад.

Таблиця 3.9 – Опис тестового випадку

Назва тесту	Перевірка функціоналу аналізу вакансій при відправці порожньої форми
Покрокова схема тесту	1. Авторизуватись в системі 2. Натиснути на кнопку «Аналіз»
Очікуваний результат	Користувач отримує повідомлення про те, що поля є обов’язковими для заповнення
Отриманий результат	Користувач отримує повідомлення про те, що поля є обов’язковими для заповнення

Таблиця 3.10 – Опис тестового випадку

Назва тесту	Перевірка функціоналу аналізу вакансій при введенні опису вакансії англійською мовою

Покрокова схема тесту	1. Авторизуватись в системі 2. Заповнити поля «Опис» ввівши англійський опис вакансії, «Назва компанії», «Заголовок» 3. Натиснути на кнопку «Аналіз»
Очікуваний результат	Користувач отримує повідомлення про те, що в описі вакансії розпізнано англійський текст і що система працює лише для україномовних вакансій

Продовження таблиці 3.9

Отриманий результат	Користувач отримує повідомлення про те, що в описі вакансії розпізнано англійський текст і що система працює лише для україномовних вакансій

Таблиця 3.10 – Опис тестового випадку

Назва тесту	Перевірка функціоналу рекомендацій вакансії
Покрокова схема тесту	1. Авторизуватись в системі 2. Заповнити поля «Опис», «Назва компанії», «Заголовок» 3. Натиснути на кнопку «Аналіз»
Очікуваний результат	Користувач отримує в правій частині екрану рекомендовані вакансії з ресурсу Jooble, які мають схожий заголовок із введеною вакансією
Отриманий результат	Користувач отримує в правій частині екрану рекомендовані вакансії з ресурсу Jooble, які мають схожий заголовок із введеною вакансією

3.8. Робота із застосунком

При відкритті застосунку користувач побачить наступний екран, що зображено на рисунку 3.16., який відповідає за логін користувача.

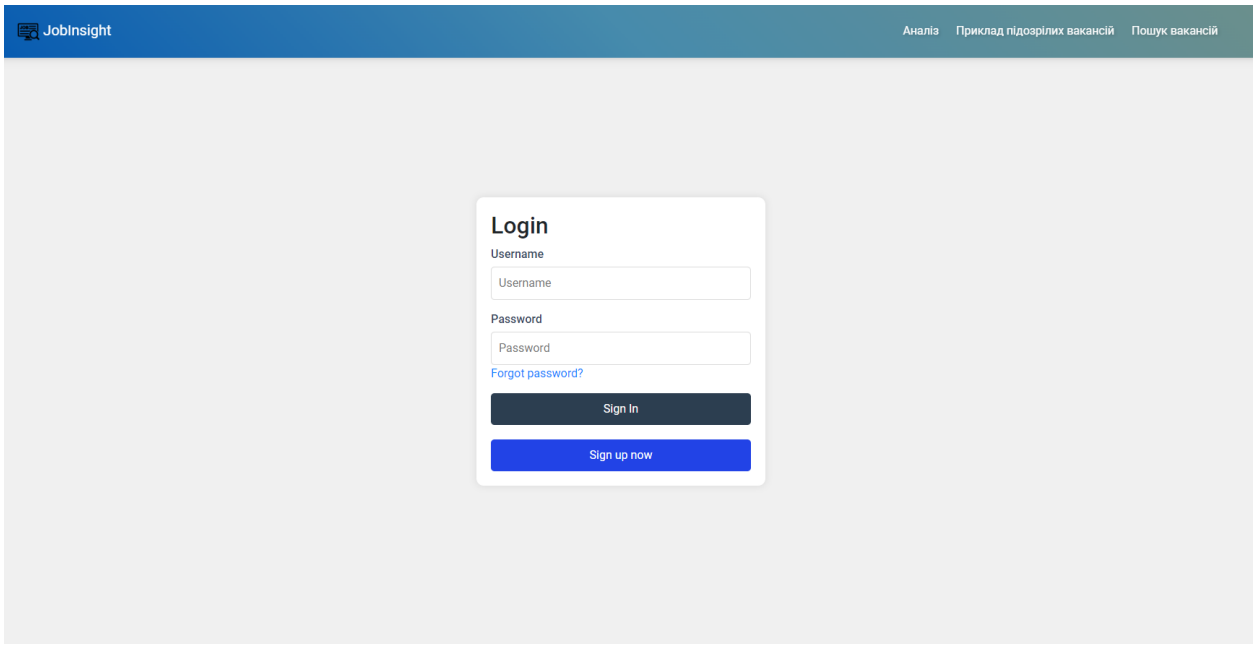

Рис. 3.16. Екран входу в застосунок

Також користувач може натиснути на кнопку «Forgot password?», ввівши свій емейл, це дозволить користувачеві отримати лист про зміну пароля.

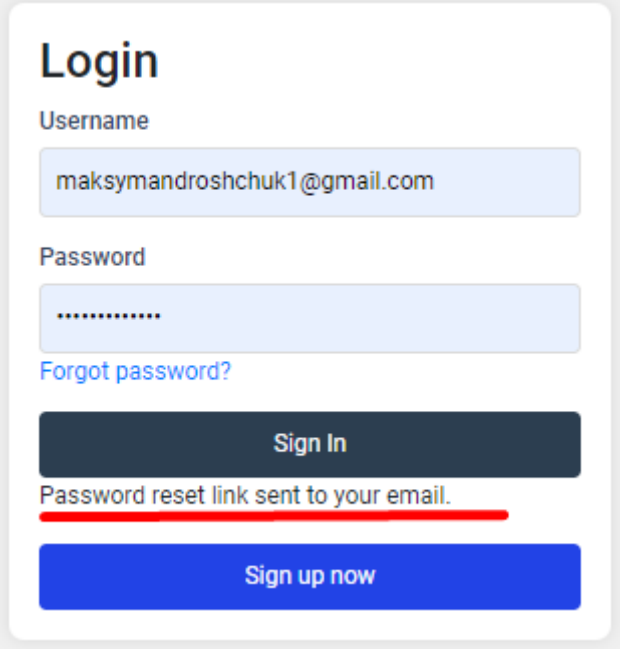

Рис. 3.17. Повідомлення про надсилання посилання для скидання пароля

Лист виглядає наступним чином:

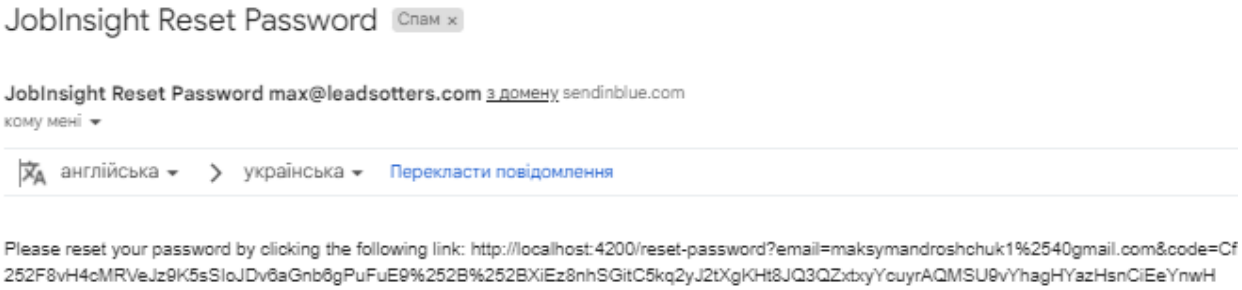

Рис. 3.18. Лист для скидання пароля

Після переходу за посиланням користувач потрапляє на сторінку для введення нового пароля.

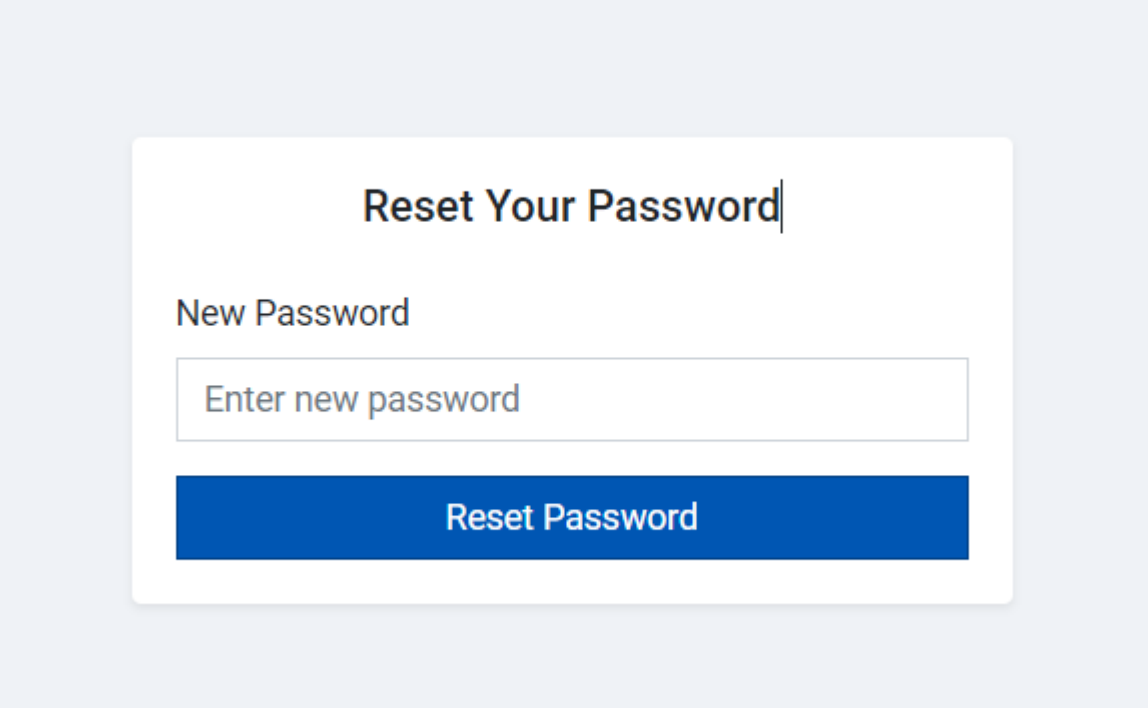

Рис. 3.19. Форма для скидання пароля

Якщо користувач натисне на кнопку «Sign up now», він перейде на сторінку реєстрації. У користувача є можливість побачити введений пароль, поставивши галочку у відповідному полі «Show password».

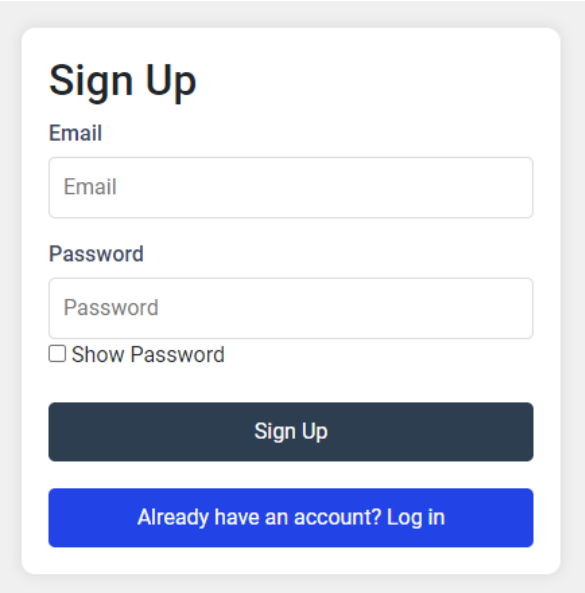

Рис. 3.20. Форма реєстрації

Після кліку на кнопку «Already have an account? Log in» відбудеться повернення на сторінку входу.

Ввівши емейл та пароль, та натиснувши на кнопку «Sign Up» користувач отримає повідомлення, що лист для підтвердження почти надіслано на поштову скриньку.

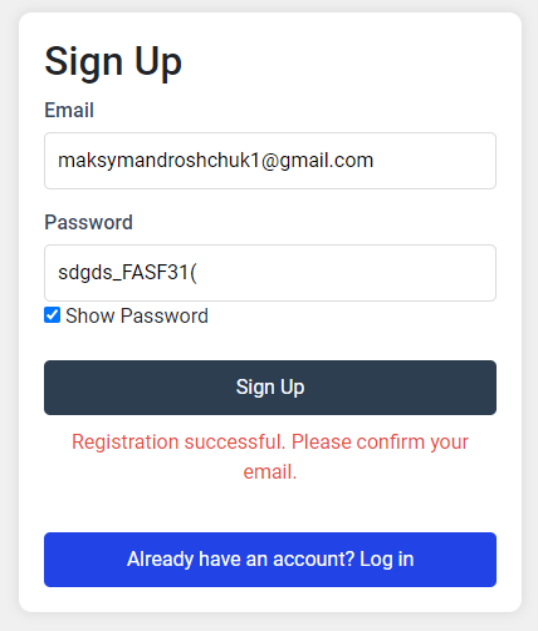

Рис. 3.21. Повідомлення про успішну реєстрацію

На почту прийде наступний лист:

Рис. 3.22. Лист про підтвердження емейлу

Перейшовши по посиланню в листі, користувач перейде на наступну сторінку:

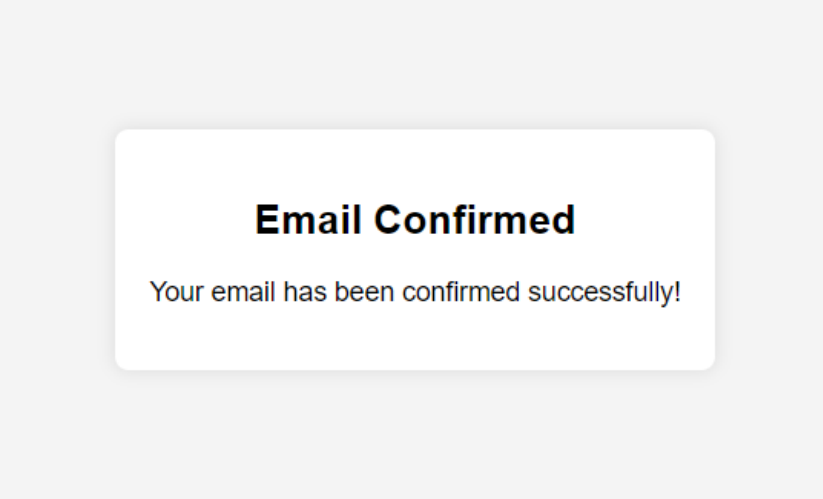

Рис. 3.23. Повідомлення про підтвердження емейлу

Після успішного входу в користувача з’являється його емейл та кнопка виходу на навігаційній панелі.

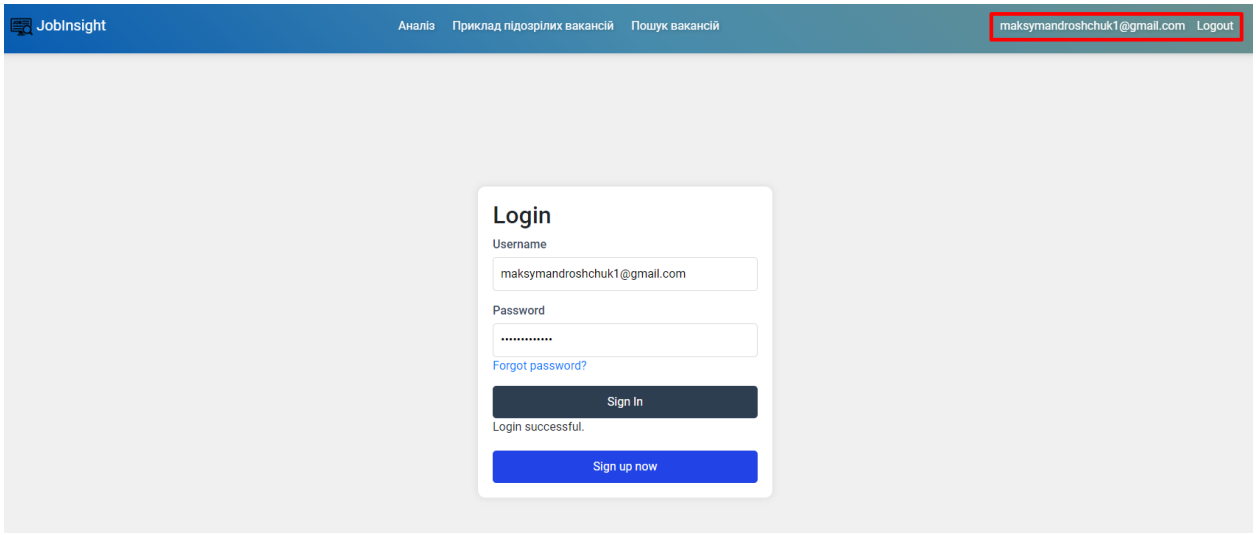

Рис. 3.24. Повідомлення про успішний вхід в акаунт

Після входу користувач отримує доступ до вкладки «Аналіз»

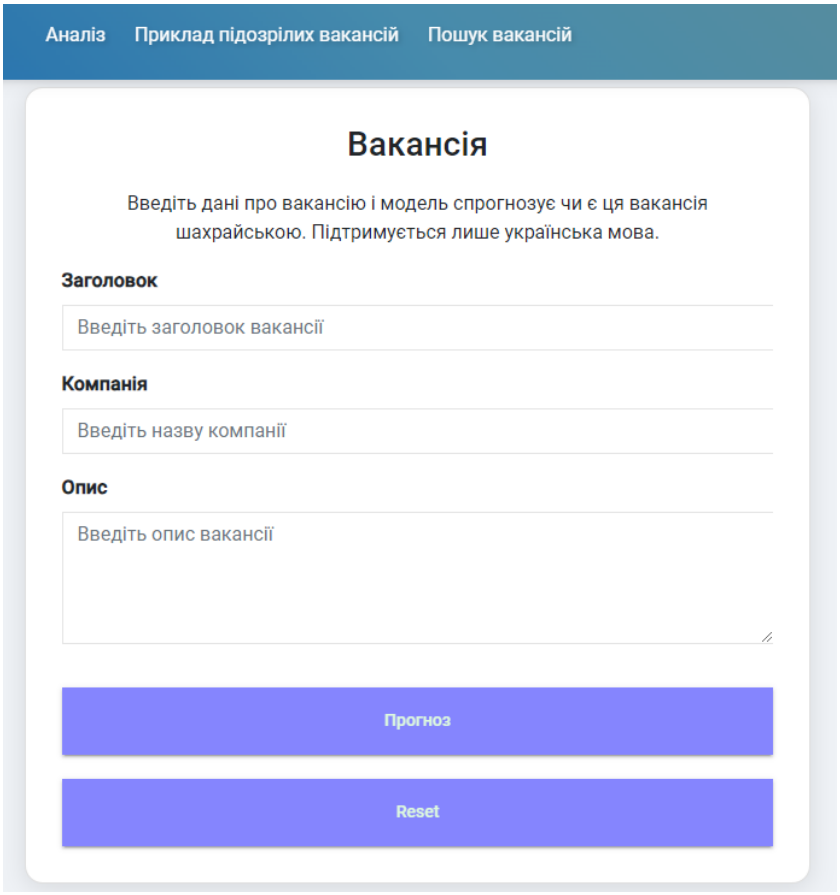

Рис. 3.25. Вкладка «Аналіз»

Після введення вакансії та натискання на кнопку прогноз, користувач отримує результат аналізу як зображено на рисунку нижче. У правій частині блок із рекомендованими вакансіями, які відповідають заголовку.

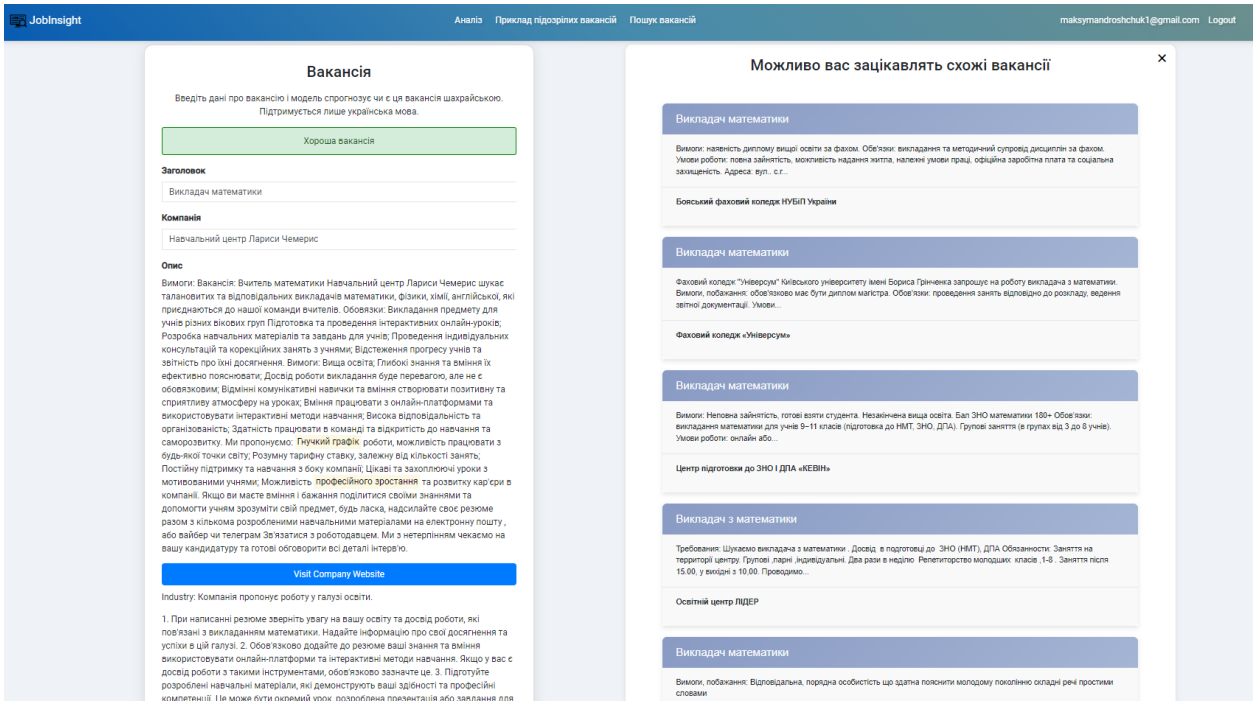

Рис. 3.26. Результат аналізу

Неавторизований користувач має доступ до сторінок «Приклади підозрілих вакансій» та «Пошук вакансій».

Нижче зображено сторінку «Приклади підозрілих вакансій», натиснувши на кнопку «Показати», користувач може ознайомитись зі списком зі 100 таких вакансій.

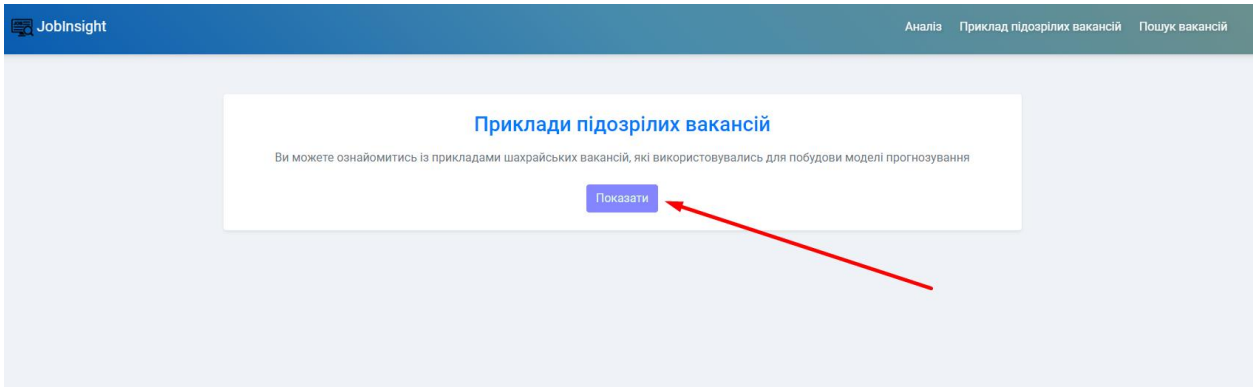

Рис. 3.27. Сторінка «Приклади підозрілих вакансій»

На вкладці «Пошук вакансій» користувач може ввести пошукове слово та регіон у відповідне поле та натиснути на кнопку «Шукати».

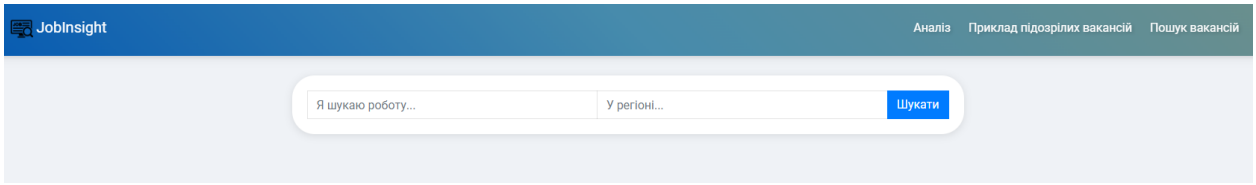

Рис. 3.28. Сторінка «Пошук вакансій»

Після натискання на кнопку «Шукати» користувач отримує результат як зображено на рисунку 3.29. Усі картки є клікабельними та ведуть на сайт, де можна ознайомитись з інформацією детальніше.

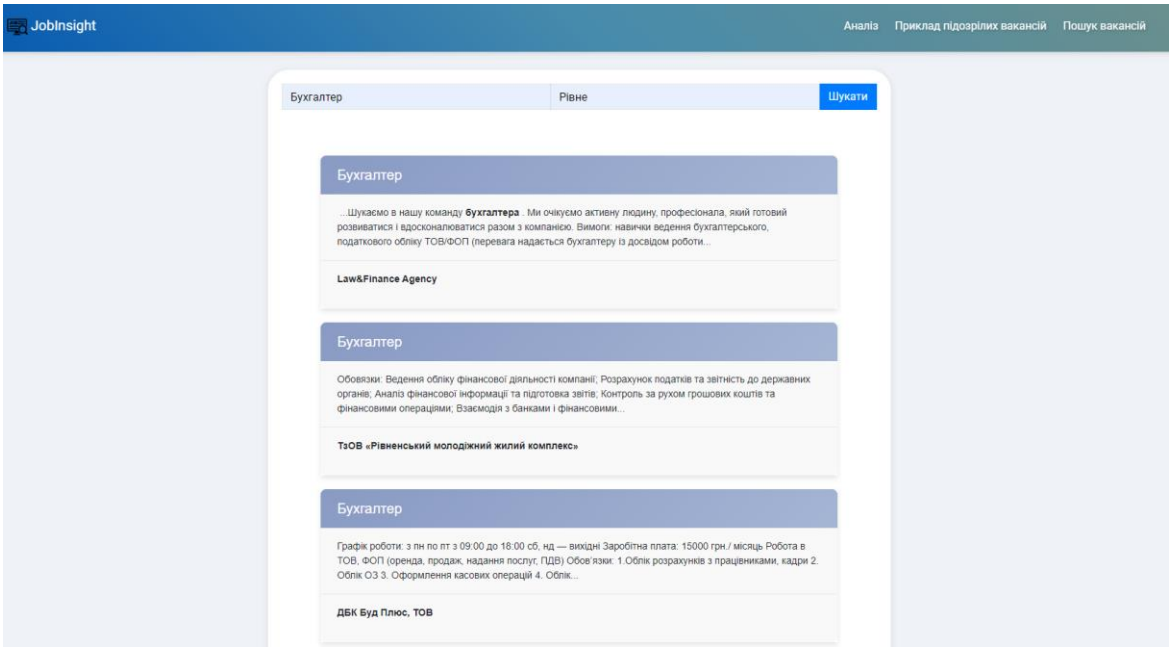

Рис. 3.29. Результат пошуку

Натиснувши на кнопку «Показати ще» користувач зможе побачити більше вакансій.

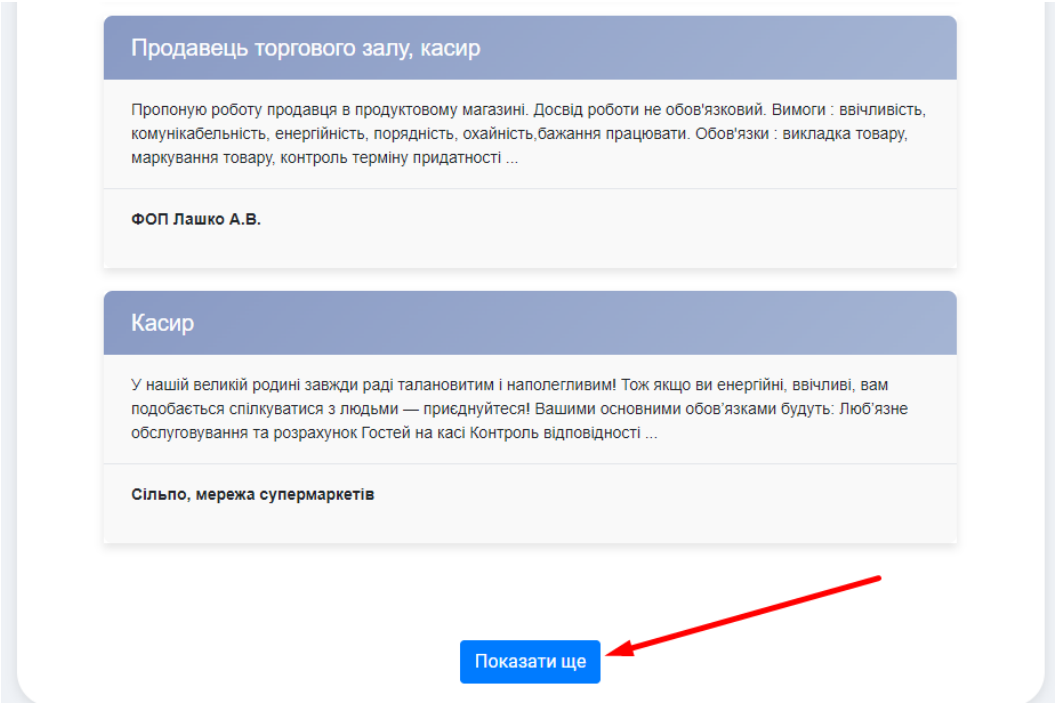

Рис. 3.30. Кнопка «Показати ще»

ВИСНОВКИ

У ході виконання кваліфікаційної роботи проведено аналіз процесу перевірки вакансій та інструментів розробки веб-сервісів. Розроблено веб-сервіс для аналізу вакансій з використанням сучасних технологій, шаблонів та архітектур розробки програмного забезпечення.

У першому розділі поставлено список задач, які необхідно виконати для написання роботи. Також проаналізовано стан та розвиток ринку онлайн-рекрутингу, наявні тренди в цій галузі, існуючі рішення. Детальніше проаналізовано та описано сутність вакансії. Розроблено функції, які реалізовуватиме програмне забезпечення. Створено варіанти використання та описано кожен у табличному вигляді. Також представлено діаграму варіантів використання.

У другому розділі описано та проаналізовано сучасні засоби розробки. Наведено переваги та недоліки кожного із них. На основі проведеного порівняння обрано інструменти розробки для кожного модуля розроблюваного веб-сервісу: для серверної частини обрано мову програмування C# та технологію Asp .Net Core Web API; для модуля аналізу вакансій обрано мову програмування Python та фреймворк Flask; для зберігання даних обрано систему управління базами даних PostgreSQL; для користувацького інтерфейсу обрано мову програмування TypeScript та фреймворк Angular.

У третьому розділі побудовано та описано бізнес-процеси з використанням BPMN діаграм. Описано програмну реалізацію та внутрішню організацію кожного із модулів веб-сервісу. Описано логіку роботи найважливіших функцій та складових системи. Описано таблиці, що використовуються для зберігання даних. Також протестовано найважливішу функціональність за допомогою методу чорного ящика. У

посібнику користувача описано та зображено, як користуватись функціями застосунку.

Завдяки виконанню всіх поставлених завдань, отримано повноцінний програмний продукт, що виконує передбачений цією роботою функціонал в реальних умовах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Поширені вакансії шахраїв - Jooble. Jooble Help Center. URL: https://help.jooble.org/uk/support/solutions/articles/60000961708-Поширені-вакансії-шахраїв (дата звернення: 23.08.2023).
2. 15 types of software architecture pattern. Turing Blog. URL: https://www.turing.com/blog/software-architecture-patterns-types/ (дата звернення: 16.10.2023).
3. Advantages of the python language over other ones | vilmate. Nearshore Software Development Company in Ukraine - VILMATE. URL: https://vilmate.com/blog/python-vs-other-programming-languages/ (дата звернення: 15.09.2023).
4. Angular. URL: https://angular.io/guide/what-is-angular (дата звернення: 11.10.2023).
5. Applications for python. Python.org. URL: https://www.python.org/about/apps/ (дата звернення: 13.09.2023).
6. BBB study: job scams | full study. International Association of Better Business Bureaus. URL: https://www.bbb.org/all/scamstudies/jobscams/jobscamsfullstudy (дата звернення: 10.08.2023).
7. BBB: The Sign of a Better Business | Better Business Bureau®. URL: https://www.bbb.org/content/dam/bbb-institute-(bbbi)/files-to-save/2020-bbb-employmentscams-report.pdf (дата звернення: 12.08.2023).
8. Bootstrap. Bootstrap · The most popular HTML, CSS, and JS library in the world. URL: https://getbootstrap.com/ (дата звернення: 13.10.2023).
9. Brevo. URL: https://app.brevo.com/billing/account/plans/subscriptions (дата звернення: 11.10.2023).

10. Business process model and notation (BPMN). 2011. С. 2–9. URL: https://www.omg.org/spec/BPMN/2.0/PDF (дата звернення: 15.10.2023).
11. Fraudulent job postings. nyu.edu. URL: https://www.nyu.edu/students/student-information-and-resources/career-development-and-jobs/find-a-job-or-internship/fraudulent-job-postings.html (дата звернення: 04.08.2023).
12. Géron A. Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow. 2-ге вид. 2019. С. 204–205.
13. GitHub - amice13/ukr_stemmer: stemmer for ukrainian language in python. GitHub. URL: https://github.com/Amice13/ukr_stemmer (дата звернення: 10.10.2023).
14. Glassdoor review 2023: features, pros & cons. Forbes Advisor. URL: https://www.forbes.com/advisor/business/glassdoor-review/#:~:text=Glassdoor%20data%20from%202021%20reveals,Review%20Management%20and (дата звернення: 27.11.2023).
15. Hamilton K., Miles R. Learning UML 2.0. 2006. С. 67–68.
16. Implementing the repository and unit of work patterns in an ASP.NET MVC application (9 of 10). Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application (дата звернення: 20.10.2023).
17. Job scams. Consumer Advice. URL: https://consumer.ftc.gov/articles/job-scams (дата звернення: 16.08.2023).
18. Language integrated query (LINQ) in C# - C#. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/dotnet/csharp/linq/ (дата звернення: 07.10.2023).

19. Learn programming languages with books and examples. URL: https://riptutorial.com/Download/linq.pdf (дата звернення: 08.10.2023).
20. Lewis C., Press T. A. Job ads showing salary ranges are 'becoming more of the norm' and helping people seeking work negotiate better pay. Fortune. URL: https://fortune.com/2023/03/20/job-ads-salary-ranges-pay-transparency-laws-a/ (дата звернення: 27.08.2023).
21. Lutz M. Learning python. 4-те вид. 2009. С. 3–5.
22. Massé M. REST API design rulebook. 2012. С. 23–25.
23. Mastering REST architecture – REST architecture details. URL: https://ahmetozlu.medium.com/mastering-rest-architecture-rest-architecture-details-e47ec659f6bc (дата звернення: 05.10.2023).
24. NET and .NET Core official support policy. Microsoft. URL: https://dotnet.microsoft.com/en-us/platform/support/policy/dotnet-core (дата звернення: 04.10.2023).
25. Pandas - python data analysis library. pandas - Python Data Analysis Library. URL: https://pandas.pydata.org/ (дата звернення: 30.09.2023).
26. PBKDF2 Hashing Algorithm. URL: https://nishothan-17.medium.com/pbkdf2-hashing-algorithm-841d5cc9178d (дата звернення: 20.10.2023).
27. Pymorphy2. PyPI. URL: https://pypi.org/project/pymorphy2/ (date of access: 01.10.2023).
28. REST API documentation tool | swagger UI. API Documentation & Design Tools for Teams | Swagger. URL: https://swagger.io/tools/swagger-ui/ (дата звернення: 09.10.2023).
29. Richards M. Software architecture patterns understanding common architectural styles and when to use them. 2-ге вид. 2022. С. 15–21.
30. Romanyuk O. Angular vs react: which one to choose for your app. freeCodeCamp.org. URL: https://www.freecodecamp.org/news/angular-vs-react-what-to-choose-for-your-app-2/ (дата звернення: 11.10.2023).

31. Salary statistics support on djinni. Djinni | Hire talent or find a job: remotely & on your own. URL: https://djinni.co/salaries/support/?utm_campaign=market-report-nov23&utm_source=djinni.co&utm_medium=topbanner (дата звернення: 03.08.2023).
32. scikit-learn: machine learning in Python – scikit-learn 0.16.1 documentation. URL: https://scikit-learn.org/ (дата звернення: 01.10.2023).
33. Sistla S. P. DBMS series part 2: DBMS architecture. Coding Minutes Blog. URL: https://blog.codingminutes.com/dbms-architecture (дата звернення: 19.10.2023).
34. spaCy · Industrial-strength Natural Language Processing in Python. URL: https://spacy.io/ (дата звернення: 01.10.2023).
35. Spurlock J. Bootstrap. 2013. С. 99–103.
36. Streamline Complex Talent Acquisition Activities with Jobvite. URL: https://web.jobvite.com/rs/328-BQS-080/images/2022-12-2022JobSeekerNationInfographic.pdf (дата звернення: 02.08.2023)
37. Top 30 companies that use python for success and profit – jaydevs. JayDevs. URL: https://jaydevs.com/top-companies-that-use-python/ (дата звернення: 03.10.2023).

ДОДАТКИ

Додаток А Лістинг програмного коду

Модуль аналізу вакансій:

```
import flask
import openai

import preprocess
from flask_limiter.util import get_remote_address
import pickle
import language_detector as lang_detect
from benefits import get_benefits
from predictionServer.preprocess import preprocess
from fuzzywuzzy import fuzz
from flask import request, jsonify
from flask_limiter import Limiter
import googleAPI as ga

app = flask.Flask(__name__)
limiter = Limiter(app=app, key_func=get_remote_address)

filename = 'model/prediction_model.sav'
predictor = pickle.load(open(filename, 'rb'))
print('predictor model READY')



def process_input(input_data: Dict[str, str]) -> Dict[str, Any]:
    title = input_data.get('title', '')
    company = input_data.get('company', '')
    description = input_data.get('snippet', '')
    lang_result = lang_detect.detect_language(description)

    return {
        'title': title,
        'company': company,
        'description': description,
        'language': lang_result
    }
```

```
def make_prediction(processed_data: Dict[str, Any]) -> int:
    if processed_data['language'] == 'uk':
        concat_data = f"{processed_data['title']} {processed_data['company']}
{processed_data['description']}"
        preprocess_data = preprocess(concat_data)
        return predictor.predict([preprocess_data])[0]
    return 0

@app.route('/predict', methods=["POST"])
@limiter.limit("100/second; 10000/hour")
def predict():
    try:
        input_data = request.get_json()
        if not input_data:
            raise ValueError("Invalid input: no data provided")

        processed_data = process_input(input_data)
        prediction = make_prediction(processed_data)

        response = {
            'isScam': prediction,
            'language': processed_data['language']
        }
        return jsonify(response)

    except ValueError as ve:
        return jsonify({'error': str(ve)}), 400
    except Exception as e:
        # Ideally log the exception here
        return jsonify({'error': 'An unexpected error occurred'}), 500

@app.route('/detect-benefits', methods=['POST'])
@limiter.limit("50/second; 5000/hour")
def detect_benefits():
    data = flask.request.json
    job_description = data.get('description', "")
    extended_benefits_keywords_uk = get_benefits()

    indices = {}
```

```
    for benefit in extended_benefits_keywords_uk:
        closest_match, match_score = get_best_fuzzy_match(job_description, benefit)
        if match_score >= 85:
            start_index = job_description.find(closest_match)
            if start_index != -1:
                indices[closest_match] = start_index
    response = {"indices": indices}
    return flask.jsonify(response)


def get_best_fuzzy_match(text, pattern):
    words = text.split()
    n = len(pattern.split())
    ngrams = [' '.join(words[i:i+n]) for i in range(len(words)-n+1)]
    scores = [(ngram, fuzz.ratio(ngram, pattern)) for ngram in ngrams]
    best_match = max(scores, key=lambda x: x[1])
    return best_match


@app.route('/detect-industry', methods=['POST'])
@limiter.limit("50/second; 5000/hour")
def detect_industry():
    try:
        data = flask.request.json
        job_description = data.get('Description', "")
        print(job_description)
        if not job_description:
            raise ValueError("Invalid input: 'description' not provided")

        openai.api_key = get_key('openai')
        response = openai.Completion.create(
            engine="gpt-3.5-turbo-instruct",
            max_tokens=500,
            prompt=f'На основі наступного опису роботи визнач індустрію вакансії:
"{job_description}"'
        )

        if response.choices and len(response.choices) > 0:
            industry = response.choices[0].text.strip()
            print(industry)
```

```
            return jsonify({'industry': industry})
        else:
            raise ValueError("OpenAI API returned no results")

    except ValueError as ve:
        return jsonify({'error': str(ve)}), 400
    except Exception as e:
        return jsonify({'error': 'An unexpected error occurred'}), 500


@app.route('/get-advices', methods=['POST'])
@limiter.limit("50/second; 5000/hour")
def get_advices():
    try:
        data = flask.request.json
        job_description = data.get('Description', "")
        print(job_description)
        if not job_description:
            raise ValueError("Invalid input: 'description' not provided")

        openai.api_key = get_key('openai')

        response = openai.Completion.create(
            engine="gpt-3.5-turbo-instruct",
            max_tokens=1500,
            prompt=f'Я хочу пройти співбесіду на вакансію, опис якої я надам нижче.
Дай мені поради на що звернути увагу в моєму резюме та на співбесіді.'
                   f'Опис вакансії: "{job_description}"'
        )

        if response.choices and len(response.choices) > 0:
            print(response.choices[0].text.strip())
            advice = response.choices[0].text.strip()
            return jsonify({'advice': advice})
        else:
            raise ValueError("OpenAI API returned no results")

    except ValueError as ve:
        return jsonify({'error': str(ve)}), 400
    except Exception as e:
```

```
        return jsonify({'error': 'An unexpected error occurred'}), 500


@app.route('/get-website', methods=["POST"])
@limiter.limit("50/second; 5000/hour")
def get_website_endpoint():
    try:
        data = request.get_json()
        if not data:
            raise ValueError("Invalid input: no data provided")

        company_name = data.get('company_name', '')
        print(company_name)
        if not company_name:
            raise ValueError("Invalid input: 'company_name' not provided")

        website, source = ga.get_website(company_name)
        response = {
            'website': website,
            'source': source
        }
        return jsonify(response)
    except ValueError as ve:
        return jsonify({'error': str(ve)}), 400
    except Exception as e:
        return jsonify({'error': 'An unexpected error occurred'}), 500

if __name__ == '__main__':
    HOST = '127.0.0.1'
    PORT = 4000

    app.run(HOST, PORT)
```

Email Service

```
using MailKit.Net.Smtp;
using MimeKit;
using System.Threading.Tasks;

namespace ScamDetectorAPI.Services;
```

```
using MimeKit;
using MailKit.Net.Smtp;
using System.Threading.Tasks;

public interface IEmailService
{
   Task<(bool Success, string ErrorMessage)> SendEmailAsync(string email, string
subject, string message);
   Task<(bool Success, string ErrorMessage)> SendPasswordResetEmailAsync(string
email, string callbackUrl);
   Task<(bool Success, string ErrorMessage)> SendEmailConfirmationAsync(string
email, string message);

}

public class EmailService : IEmailService
{
   private readonly string _smtpHost;
   private readonly int _smtpPort;
   private readonly string _mailjetApiKey;
   private readonly string _mailjetApiSecret;

   public EmailService(string smtpHost, int smtpPort, string mailjetApiKey
      )
   {
      _smtpHost = smtpHost;
      _smtpPort = smtpPort;
      _mailjetApiKey = mailjetApiKey;
   }


   public async Task<(bool Success, string ErrorMessage)> SendEmailAsync(string
email, string subject, string message)
   {
      var emailMessage = new MimeMessage();
      emailMessage.From.Add(new MailboxAddress(subject, _email));
      emailMessage.To.Add(new MailboxAddress("", email));
      emailMessage.Subject = subject;
      emailMessage.Body = new TextPart("html") { Text = message };
```

```
        using (var client = new SmtpClient())
        {
            await client.ConnectAsync(_smtpHost, _smtpHost,
MailKit.Security.SecureSocketOptions.StartTls);
            await client.AuthenticateAsync(_email, _password);
            await client.SendAsync(emailMessage);
            await client.DisconnectAsync(true);
        }

        return (true, "Email sent successfully.");
    }

    public async Task<(bool Success, string ErrorMessage)>
SendPasswordResetEmailAsync(string email, string callbackUrl)
    {
        var subject = "JobInsight Reset Password";
        Console.WriteLine(callbackUrl);
        var message = $"Please reset your password by clicking the following link:
{Uri.EscapeUriString(callbackUrl)}";

        // Use the existing SendEmailAsync method to send the password reset email
        return await SendEmailAsync(email, subject, message);
    }

    public async Task<(bool Success, string ErrorMessage)>
SendEmailConfirmationAsync(string email, string message)
    {
        var subject = "JobInsight email confirmation";
        //var message = $"Please reset your password by clicking the following link:
{Uri.EscapeUriString(callbackUrl)}";

        // Use the existing SendEmailAsync method to send the password reset email
        return await SendEmailAsync(email, subject, message);
    }
}
```

Account Controller

```
using System.IdentityModel.Tokens.Jwt;
```

```
using System.Net;
using System.Security.Claims;
using System.Text;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.IdentityModel.Tokens;
using ScamDetector.DAL.DTO;
using ScamDetectorAPI.Services;
using JwtRegisteredClaimNames =
Microsoft.IdentityModel.JsonWebTokens.JwtRegisteredClaimNames;

[Route("[controller]")]
[ApiController]
public class AccountController : ControllerBase
{
   private readonly UserManager<IdentityUser> _userManager;
   private readonly SignInManager<IdentityUser> _signInManager;
   private readonly IEmailService _emailService;
   private readonly IConfiguration _configuration;  // Add this line

   public AccountController(UserManager<IdentityUser> userManager,
SignInManager<IdentityUser> signInManager, IEmailService emailService,
IConfiguration configuration)
   {
      _userManager = userManager;
      _signInManager = signInManager;
      _emailService = emailService;
      _configuration = configuration;
   }

   [HttpPost("register")]
   public async Task<IActionResult> Register([FromBody] RegisterRequest request)
   {

      var user = new IdentityUser { UserName = request.Email, Email = request.Email
};
      var result = await _userManager.CreateAsync(user, request.Password);


      if (result.Succeeded)
```

```
        {
            // Send an email confirmation link
            var code = await
_userManager.GenerateEmailConfirmationTokenAsync(user);
            var callbackUrl = Url.Action("ConfirmEmail", "Account", new { userId =
user.Id, code = code }, protocol: HttpContext.Request.Scheme);

            var emailResult = await
_emailService.SendEmailConfirmationAsync(request.Email, $"Please confirm your
account by clicking this link: <a href=\'{callbackUrl}\'>Click here</a>
{callbackUrl}");

            if(emailResult.Success)
                return Ok(new { Message = "Registration successful. Please confirm your
email." });
            else
                return BadRequest(new { Message = $"Registration successful, but failed to
send confirmation email. Reason: {emailResult.ErrorMessage}" });
        }
        return BadRequest(result.Errors);
    }

    [HttpPost("login")]
    public async Task<IActionResult> Login([FromBody] RegisterRequest request)
    {
        var user = await _userManager.FindByEmailAsync(request.Email);
        if (user != null && !await _userManager.IsEmailConfirmedAsync(user))
        {
            return BadRequest(new { Message = "You must confirm your email to login."
});
        }

        var result = await _signInManager.PasswordSignInAsync(request.Email,
request.Password, false, false);

        if (result.Succeeded)
        {
            var claims = new[]
            {
                new Claim(JwtRegisteredClaimNames.Sub, user.Email),
```

```
            new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
            // You can add more claims if required
        };

        var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Jwt:Key"]));
        var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

        var token = new JwtSecurityToken(
            issuer: _configuration["Jwt:Issuer"],
            audience: _configuration["Jwt:Audience"],
            claims: claims,
            expires: DateTime.Now.AddHours(1),  // Token lifetime of 3 hours
            signingCredentials: creds);

        return Ok(new
        {
            token = new JwtSecurityTokenHandler().WriteToken(token),
            expiration = token.ValidTo
        });
      }

      return Unauthorized();
    }


    [HttpPost("logout")]
    public async Task<IActionResult> Logout()
    {
      await _signInManager.SignOutAsync();
      return Ok();
    }

    [HttpGet("confirmEmail")]
    public async Task<IActionResult> ConfirmEmail(string userId, string code)
    {
      if (userId == null || code == null)
      {
        return BadRequest();
      }
```

```
var user = await _userManager.FindByIdAsync(userId);
if (user == null)
{
   return NotFound();
}

var result = await _userManager.ConfirmEmailAsync(user, code);
if (result.Succeeded)
{
   var htmlContent = @"
      <!DOCTYPE html>
      <html lang='en'>
      <head>
         <meta charset='UTF-8'>
         <meta name='viewport' content='width=device-width, initial-scale=1.0'>
         <title>Email Confirmation</title>
         <style>
            body {
               font-family: Arial, sans-serif;
               background-color: #f4f4f4;
               display: flex;
               justify-content: center;
               align-items: center;
               height: 100vh;
            }
            .confirmation-box {
               background-color: #fff;
               padding: 20px;
               border-radius: 8px;
               box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
               text-align: center;
            }
         </style>
      </head>
      <body>
         <div class='confirmation-box'>
            <h2>Email Confirmed</h2>
            <p>Your email has been confirmed successfully!</p>
         </div>
```

```
            </body>
            </html>";

        return new ContentResult
        {
            ContentType = "text/html",
            StatusCode = (int)HttpStatusCode.OK,
            Content = htmlContent
        };
    }

    return BadRequest(result.Errors);
}

[HttpPost("requestPasswordReset")]
public async Task<IActionResult> RequestPasswordReset([FromBody]
PasswordResetRequestModel requestModel)
{
    var user = await _userManager.FindByEmailAsync(requestModel.Email);
    if (user == null)
    {
        // Don't reveal that the user does not exist or is not confirmed
        return Ok(new { Message = "If your email address exists in our database, you
will receive a password reset link." });
    }

    var code = await _userManager.GeneratePasswordResetTokenAsync(user);
    var frontendResetPasswordUrl = "http://localhost:4200/reset-password"; // Use
the correct route
    var encodedCode = WebUtility.UrlEncode(code);
    var callbackUrl =
$"{frontendResetPasswordUrl}?email={WebUtility.UrlEncode(requestModel.Email)}
&code={encodedCode}";

    var emailResult = await
_emailService.SendPasswordResetEmailAsync(requestModel.Email, callbackUrl);

    if(emailResult.Success)
        return Ok(new { Message = "Password reset link has been sent to your email
address." });
```

```
        else
            return BadRequest(new { Message = $"Failed to send password reset email.
Reason: {emailResult.ErrorMessage}" });
    }

    [HttpPost("resetPassword")]
    public async Task<IActionResult> ResetPassword([FromBody]
ResetPasswordModel model)
    {
        if (!ModelState.IsValid)
        {
            // Return the validation errors in the model state
            return BadRequest(ModelState);
        }

        var email = WebUtility.UrlDecode(model.Email);
        var code = WebUtility.UrlDecode(model.Code);

        var user = await _userManager.FindByEmailAsync(email);
        if (user == null)
        {
            // Log the error internally and return a generic error message
            return BadRequest(new { Message = "Could not reset password. Please try
again or contact support if the problem persists." });
        }

        var result = await _userManager.ResetPasswordAsync(user, code,
model.NewPassword);
        if (result.Succeeded)
        {
            return Ok(new { Message = "Your password has been reset." });
        }

        // Process the errors to make them more user-friendly
        return BadRequest(result.Errors);
    }


    public class PasswordResetRequestModel
    {
```

```
        public string Email { get; set; }
    }

    public class ResetPasswordModel
    {
        public string Email { get; set; }
        public string Code { get; set; }
        public string NewPassword { get; set; }
    }
}
```

Job Controller

```
using System.Text;
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using Newtonsoft.Json.Serialization;
using OpenAI_API.Completions;
using ScamDetector.DAL;
using ScamDetector.DAL.DTO;

namespace ScamDetectorAPI.Controllers
{
    [Route("[controller]")]
    [ApiController]
    public class JobController : ControllerBase
    {
        private readonly IUnitOfWork _unitOfWork;
        private static IConfiguration _configuration;

        public JobController(IUnitOfWork unitOfWork,
            IConfiguration configuration)
        {
            _unitOfWork = unitOfWork;
            _configuration = configuration;
        }

        [HttpGet("ScamExample")]
        public IEnumerable<Job> Get()
        {
```

```
        var task = _unitOfWork.Jobs.Get(true, 100);
        var jobs = task.Result;
        Console.WriteLine(jobs.Count());
        return jobs;
    }

    [HttpPost("Benefits")]
    public async Task<ActionResult<DetectBenefitsResponseDTO>>
DetectBenefits([FromBody] DetectBenefitsRequestDTO request)
    {
        var settings = new JsonSerializerSettings
        {
            ContractResolver = new CamelCasePropertyNamesContractResolver()
        };

        var client = new HttpClient();

        // Get the Flask API URL from configuration
        var flaskApiUrl =
_configuration.GetSection("PredictionServer").GetSection("benefitsUrl").Value;
        var json = JsonConvert.SerializeObject(request, Formatting.Indented,
settings);

        var httpContent = new StringContent(json, Encoding.UTF8,
"application/json");

        var httpResponse = await client.PostAsync(flaskApiUrl, httpContent);

        if (!httpResponse.IsSuccessStatusCode)
        {
            return BadRequest("Error calling Flask API");
        }

        var jsonResponse = await httpResponse.Content.ReadAsStringAsync();
        Console.WriteLine(jsonResponse);
        var response =
JsonConvert.DeserializeObject<DetectBenefitsResponseDTO>(jsonResponse);

        return Ok(response);
    }
```

```
public class JobDescriptionRequest
{
    public string Description { get; set; }
}


[HttpPost("Industry")]
public async Task<IActionResult>
GetIndustryFromJobDescriptionAsync([FromBody] JobDescriptionRequest request)
{
    var jobDescription = request.Description;
    Console.WriteLine(jobDescription);
    var httpClient = new HttpClient();

    // Your Flask API endpoint
    var flaskApiUrl =
_configuration.GetSection("PredictionServer").GetSection("chatgptUrl").Value;

    // Prepare the JSON payload
    var payload = new JobDescriptionPayload
    {
        Description = jobDescription
    };

    var jsonString = JsonConvert.SerializeObject(payload);
    var content = new StringContent(jsonString, Encoding.UTF8,
"application/json");

    var response = await httpClient.PostAsync(flaskApiUrl, content);

    if (response.IsSuccessStatusCode)
    {
        var responseContent = await response.Content.ReadAsStringAsync();
        var responseObject =
JsonConvert.DeserializeObject<ResponseObject>(responseContent);

        // Handle the response, for this example, return the industry as JSON
        Console.WriteLine(new JsonResult(new { Industry =
responseObject.Industry }));
```

```
            return new JsonResult(new { Industry = responseObject.Industry });
        }
        else
        {
            var errorContent = await response.Content.ReadAsStringAsync();
            var errorResponse =
JsonConvert.DeserializeObject<ErrorResponse>(errorContent);

            // Return the error as JSON with a Bad Request status
            return new BadRequestObjectResult(new { Error = $"Error from API:
{errorResponse.Error}" });
        }
    }

    [HttpPost("Advice")]
    public async Task<IActionResult> GetAdvice([FromBody]
JobDescriptionRequest request)
    {
        // Convert the request to JSON format
        var json = JsonConvert.SerializeObject(request);
        var content = new StringContent(json, Encoding.UTF8, "application/json");

        var flaskApiUrl =
_configuration.GetSection("PredictionServer").GetSection("adviceUrl").Value;

        // Set up HttpClient
        using (HttpClient client = new HttpClient())
        {
            // Make the POST request to the Flask endpoint
            var response = await client.PostAsync(flaskApiUrl, content);

            if (response.IsSuccessStatusCode)
            {
                // If the request was successful, parse the response and return
                var responseString = await response.Content.ReadAsStringAsync();
                var responseObject =
JsonConvert.DeserializeObject<AdviceResponse>(responseString);
                return Ok(responseObject);
            }
            else
```

```
            {
                // Handle the error
                return BadRequest("Error calling Flask API");
            }
        }
    }

    public class AdviceResponse
    {
        public string Advice { get; set; }
    }

    public class JobDescriptionPayload
    {
        public string Description { get; set; }
    }

    public class ResponseObject
    {
        public string Industry { get; set; }
    }

    public class ErrorResponse
    {
        public string Error { get; set; }
    }
}
```