Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна технологія пошуку роботи. Частина 1. Аналіз даних та нечітка логіка»

Виконав: студент 2-го курсу, групи 2КН-23м,
спеціальності 122 – «Комп'ютерні науки»

Стасишен Д. В.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН

Арсенюк І. Р.
(прізвище та ініціали)

«05» 12 2024 р.

Опонент: к.т.н., доцент каф. АІІТ

Барабан М. В.
(прізвище та ініціали)

«05» 12 2024 р.

Допущено до захисту

Завідувач кафедри КН

д.т.н., проф. Яровий А. А.
(прізвище та ініціали)

«06» 12 2024 р.

Вінниця - 2024 року

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти II-й (магістерський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо-професійна програма – Системи штучного інтелекту

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
д.т.н., проф. Яровий А.А.
(підпис)
«11» 10 2024 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Стасишеному Денис Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема роботи: «Інформаційна технологія пошуку роботи. Частина 1. Аналіз даних та нечітка логіка»
Керівник роботи к.т.н., доц., доц. каф. КН, Арсенюк І. Р.
затверджені наказом вищого навчального закладу від «17» 09 2024 року №310
2. Строк подання студентом роботи 11.11 2024 року
3. Вихідні дані до роботи: можливість обробки та аналізу даних для пошуку роботи за останні 6 місяців; час від подачі заявки на пошук вакансії до отримання результатів − не більше 30 секунд; кількість підтримуваних критеріїв для пошуку роботи – не менше 6; мова програмування − об'єктно-орієнтована.
4. Зміст текстової частини: вступ, обґрунтування доцільності розробки інформаційної технології пошуку роботи, моделювання інформаційної технології пошуку роботи, програмна реалізації інформаційної технології пошуку роботи, економічна частина, висновки, список використаних джерел, додатки.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): uml-діаграма послідовності взаємодії модулів інформаційної технології пошуку роботи;структура інформаційної технології пошуку роботи з використанням аналізу даних та нечіткої логіки;схема системи клієнт-сервер;схема роботи аналізу даних та побудова системи оцінки на основі нечіткої логіки;схема послідновності при взаємодії користувача з модулем оцінки кандидатів;загальний вигляд інтерфейсного вікна реєстрації користувача;загальний вигляд вікна головної сторінки;загальний вигляд вікна сторінки з рекомендованими вакансіями;загальний вигляд вікна сторінки з детальною інформацією про вакансію;загальний вигляд вікна сторінки для створення та завантаження резюме;загальний вигляд вікна сторінки адміністративної панелі для управління списком прав користувачів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	виконання прийняв
1–3	Арсенюк І.Р., к.т.н., доц. каф. КН		
4	Адлер О.О., к.т.н., доц. каф. ЕПВМ		

7. Дата видачі завдання 02.09, 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Обґрунтування доцільності розробки	02.09.24-04.09.24	Розділ 1
2	Моделювання інформаційної технології пошуку роботи	05.09.24-22.09.24	Розділ 2
3	Програмна реалізація інформаційної технології пошуку роботи	23.09.24-23.10.24	Розділ 3
4	Підготовка економічної частини	24.10.24-01.11.24	Розділ 4
5	Апробація результатів дослідження	02.11.24-04.11.24	тези доповіді, авторське право на твір
6	Оформлення пояснювальної записки, графічного матеріалу та презентації	05.11.24-11.11.24	Пояснювальна записка, графічний матеріал, презентація

Студент _____ Стасишен Д.
(підпис) (прізвище та ініціа

Керівник роботи _____ Арсенюк І.
(підпис) (прізвище та ініціа

АНОТАЦІЯ

УДК 004.42

Стасишен Д. В. Інформаційна технологія пошуку роботи. Частина 1. Аналіз даних та нечітка логіка. Магістерська кваліфікаційна робота зі спеціальності 122 Комп’ютерні науки, освітня програма – Системи штучного інтелекту. Вінниця: ВНТУ, 2024. 151 с.

Укр. мовою. Бібліогр.: 28 назв; рис.: 16; табл.: 10.

Дана магістерська кваліфікаційна робота присвячена розробці інформаційної технології пошуку роботи, а саме нечіткої інтелектуальної системи для аналізу вакансій та надання рекомендацій щодо їх вибору, а також клієнтської частини інформаційної технології. Проведено аналіз сучасного стану розвитку методів та методологій пошуку роботи. Здійснюється проектування нечіткої інтелектуальної системи, в ході якого буде запропоновано використовувати дерево нечіткого логічного виведення. Планується розробка логіко-лінгвістичної моделі та бази знань нечіткої інтелектуальної системи.

Буде виконано програмну реалізацію клієнтської частини інформаційної технології за допомогою середовища розробки Visual Studio Code, мови програмування TypeScript та фреймворку React. В майбутньому буде проведено тестування для підтвердження підвищення якості надання рекомендацій щодо вибору вакансій у порівнянні з найближчими програмами-аналогами.

Ілюстративна частина складається з 11 плакатів із результатами моделювання.

Ключові слова: пошук роботи, нечітка логіка, дерево нечіткого логічного виведення, вакансії, надання рекомендацій.

ABSTRACT

Stasyshen D. V. Information technology of job search. Part 1: Data analysis and fuzzy logic. Master's Thesis in Specialty 122 Computer Science, Educational Program - Artificial Intelligence Systems. Vinnytsia: VNTU, 2024. 151 p.

In English. Bibliography: 28 titles; illustrations: 16; tables: 10

This master's thesis is devoted to the development of information technology for job search, namely a fuzzy intelligent system for analyzing vacancies and providing recommendations for their selection, as well as the client part of the information technology. The current state of development of job search methods and methodologies is analyzed. The design of a fuzzy intelligent system is underway, during which it will be proposed to use a fuzzy inference tree. It is planned to develop a logical and linguistic model and knowledge base of the fuzzy intelligent system.

The software implementation of the client side of the information technology will be performed using the Visual Studio Code development environment, the TypeScript programming language, and the React framework. In the future, testing will be carried out to confirm the improvement of the quality of providing recommendations for the selection of vacancies in comparison with the nearest analogous programs.

The illustrative part consists of 11 posters with modeling results.

Keywords: job search, fuzzy logic, fuzzy inference tree, vacancies, recommendation.

ЗМІСТ

ВСТУП

Актуальність теми. За останні десятиліття проблема ефективного пошуку роботи стала надзвичайно актуальною у суспільстві. Причиною стрімкого росту зацікавленості цією проблемою є кардинальна зміна ринку праці та способів працевлаштування у сучасному світі. Інтенсивний, всеосяжний процес впровадження цифрових технологій, автоматизації та глобалізації на всіх рівнях економічної діяльності спричинив розвиток нових форм зайнятості та методів пошуку роботи, не притаманних людям раніше. При дослідженні вищевказаних тенденцій, значна частина вчених почала відзначати всебічний вплив інформаційних технологій на процес пошуку роботи та працевлаштування.

Розробка інформаційної технології пошуку роботи є актуальною, оскільки у суспільстві сьогодні є нагальна потреба забезпечення ефективного працевлаштування в умовах динамічного ринку праці, за допомогою простих та доступних рішень. Надання рекомендацій щодо пошуку вакансій, складання резюме або підготовки до співбесіди є складною та нетривіальною задачею. Для розв'язання цієї проблеми доцільним є застосування теорії та методів штучного інтелекту, зокрема використання машинного навчання для аналізу даних ринку праці та підбору вакансій.

Використання методів штучного інтелекту дозволить проводити відносно швидкий аналіз великих обсягів даних про вакансії та резюме з допустимим ступенем точності, значно спростить процес розробки інформаційної технології та дозволить працювати з даними в умовах невизначеності, постійних змін на ринку праці, чи в ситуації, коли побудувати чіткий алгоритм підбору вакансій не є можливим.

Зв'язок роботи з науковими програмами, планами, темами. Магістерська кваліфікаційна робота виконана відповідно до напрямку наукових досліджень кафедри комп’ютерних наук Вінницького національного технічного університету 22 К1 «Розробка прикладних інтелектуальних інформаційних технологій та систем».

Мета і завдання дослідження. Метою магістерської кваліфікаційної роботи є розробка інформаційної технології пошуку роботи. Це дозволить створити зручний клієнтський додаток, який надаватиме персоналізовані рекомендації щодо пошуку вакансій та складання резюме, враховуючи особливості користувача та поточний стан ринку праці. Додаток буде інтуїтивно зрозумілим та зможе конкурувати з іншими системами-аналогами на ринку.

Для досягнення поставленої мети потрібно розв'язати такі завдання:

- Провести аналіз існуючих систем пошуку роботи та обґрунтувати доцільність розробки нової інформаційної технології.
- Розробити математичну модель для надання рекомендацій з пошуку роботи, враховуючи різні параметри (освіта, досвід роботи, навички, вимоги роботодавців тощо).
- Розглянути методи та технології, які можуть бути використані для надання персоналізованих рекомендацій щодо пошуку вакансій та складання резюме.
- Спроектувати архітектуру та структуру програмного модуля для надання рекомендацій з пошуку роботи.
- Проаналізувати та обґрунтувати вибір інструментів для розробки.
- Виконати програмну реалізацію інформаційної технології пошуку роботи.
- Провести тестування розробленої програми та проаналізувати отримані результати.
- Економічно обґрунтувати доцільність розробки інформаційної технології пошуку роботи.

Об’єктом дослідження є – процес аналізу даних та нечіткої логіки при пошуку роботи.

Предметом дослідження є програмні аналізу даних та нечіткої логіки при пошуку роботи.

Методи дослідження. У роботі використані такі методи наукових досліджень: аналіз аналогічних технологій та програмних рішень, що дозволяє визначити більшість проблем та труднощів, які необхідно вирішити в даній роботі. Методи та підходи до розробки систем нечіткого логічного виведення, методи та підходи до розробки веб-орієнтованих програмних застосунків, методи об'єктно-орієнтованого програмування.

Наукова новизна одержаних результатів полягає в наступному: розроблено інформаційну модель пошуку роботи, що відрізняється від існуючих застосуванням методів машинного навчання при проведенні аналізу даних ринку прац.

Практичне значення одержаних результатів полягає у наступному:

1. Визначено основні етапи роботи модулів аналізу даних та надання рекомендацій щодо пошуку вакансій і складання резюме.

2. Розроблено структуру інформаційної технології пошуку роботи, яка включає модулі збору та аналізу даних про вакансії, оцінки резюме, персоналізованих рекомендацій та інтерфейсу користувача.

3. На основі проведених досліджень здійснено програмну реалізацію клієнтської частини інформаційної технології пошуку роботи.

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджується строгістю постановки задач, коректним застосуванням математичних методів під час доведення наукових положень, строгим виведенням аналітичних співвідношень, тестуванням програмної реалізації інформаційної технології пошуку роботи.

Особистий внесок магістранта. Результати магістерської кваліфікаційної роботи отримані самостійно. В публікації у співавторстві здобувачу належить дослідження особливостей розробки інформаційної технології пошуку роботи.

Апробація результатів роботи. Результати досліджень було апробовано на LII науково-технічній конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2024) м. Вінниці у 2024; р [1].

Публікації. За результатами магістерської кваліфікаційної роботи опубліковано тези доповідей на науково-технічних конференціях [1] та подано заявку на реєстрацію авторського права на твір у формі комп’ютерної програми – «Інформаційна технологія пошуку роботи. Частина 1. Аналіз даних та нечітка логіка».

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ПОШУКУ РОБОТИ

1.1 Аналіз предметної області пошуку роботи

Пошук роботи можна визначити як систематичну послідовність дій і заходів, спрямованих на пошук, аналіз та отримання бажаного місця працевлаштування. Цей процес має на меті знаходження роботи, яка відповідає професійним навичкам, досвіду та кар'єрним цілям особи.

Основними причинами збільшення запиту в суспільстві на ефективні методи пошуку роботи є:

− Динамічний ринок праці. У сучасному суспільстві ринок праці швидко змінюється під впливом технологічних інновацій, глобалізації та економічних факторів. Це призводить до необхідності постійного оновлення навичок пошуку роботи та адаптації до нових вимог роботодавців.

− Конкуренція та високі вимоги. Зростаюча конкуренція на ринку праці вимагає від кандидатів більш ефективних стратегій пошуку роботи, включаючи створення привабливого резюме, розвиток навичок проходження співбесід та активне нетворкінг.

− Технологічні зміни. Цифровізація процесу пошуку роботи через онлайн-платформи, соціальні мережі та спеціалізовані додатки змінила способи взаємодії між роботодавцями та кандидатами. Це вимагає від шукачів роботи вміння ефективно використовувати ці інструменти.

− Кар'єрний розвиток та самореалізація. Сучасні працівники все частіше шукають не просто роботу, а можливості для професійного росту та самореалізації. Це призводить до необхідності більш ретельного підходу до вибору роботи та аналізу потенційних роботодавців.

− Економічна нестабільність. Глобальні економічні виклики, такі як пандемії або фінансові кризи, можуть призводити до масових звільнень та зміни структури ринку праці, що підвищує важливість ефективних стратегій пошуку роботи.

Крім того, люди стали більш обізнаними про важливість ефективного пошуку роботи для професійного успіху та фінансової стабільності. Зростання доступності інформації та освітніх ресурсів з питань кар'єрного розвитку підвищило усвідомлення необхідності стратегічного підходу до пошуку роботи. Це призвело до збільшення попиту на інструменти та технології, які можуть оптимізувати процес пошуку вакансій, підготовки резюме та проходження співбесід. Більшість сайтів надає можливість пошуку роботи за різними категоріями, як показано на рисунку 1.1.

Рисунок 1.1 – Загальний вигляд вікна пошуку роботи за категоріями

Загалом, ефективні методи пошуку роботи стали актуальними сьогодні через спільну потребу забезпечення професійної реалізації, фінансової стабільності та кар'єрного розвитку в умовах сучасного ринку праці.

Процес пошуку роботи представляє собою систематичну та організовану діяльність, спрямовану на пошук, аналіз та отримання бажаного місця працевлаштування. Його основна мета – досягнення оптимальних результатів у професійній сфері, таких як отримання бажаної посади, покращення умов праці, підвищення заробітної плати та розвиток кар'єри.

Ефективні методи пошуку роботи:

- позитивно впливають на професійний розвиток і якість життя
- знижують рівень стресу, пов'язаного з безробіттям або незадовільною роботою
- підвищують шанси на отримання бажаної посади
- сприяють розширенню професійних зв'язків та нетворкінгу

Процес пошуку роботи базується на принципах ринку праці, психології, маркетингу та інших галузей. Для ефективності цього процесу важливо враховувати індивідуальні особливості кандидата, а також використовувати інноваційні методи та сучасні технології.

Організація процесу пошуку роботи вимагає використання різноманітних підходів, принципів, методів та методологій, які дозволяють досягти оптимальних результатів у пошуку та отриманні бажаного місця працевлаштування. Розглянемо основні підходи та принципи , які є ключовими в організації ефективного процесу пошуку роботи.

Індивідуалізація та персоніфікація стратегій пошуку роботи. Підходи, спрямовані на індивідуалізацію та персоніфікацію стратегій пошуку роботи, фокусуються на унікальних потребах і можливостях кожного окремого кандидата. Ці підходи стали ключовими в сучасному процесі пошуку роботи, оскільки вони сприяють досягненню кращих результатів і підвищенню ефективності працевлаштування.

Індивідуалізація стратегій пошуку роботи передбачає створення планів, які враховують індивідуальні особливості, навички, досвід та кар'єрні цілі кожного кандидата. Персоніфікація стратегій пошуку роботи ще більше звужує фокус, орієнтуючись на конкретні сильні сторони, особисті якості та унікальні професійні досягнення людини. Цей підхід може включати в себе врахування індивідуальних характеристик, які виходять за рамки стандартних резюме та методів пошуку роботи.

Індивідуалізація та персоніфікація дозволяють максимально використовувати потенціал кожного кандидата та зробити процес пошуку роботи більш ефективним.

Циклічність процесу пошуку роботи. Циклічність процесу пошуку роботи є важливою концепцією в кар'єрному розвитку. Вона передбачає, що процес пошуку роботи розглядається як послідовність циклів або фаз, які регулярно повторюються з метою досягнення покращених результатів та підтримання оптимальної професійної форми. Цей підхід дозволяє краще регулювати зусилля, зменшувати ризик вигорання та забезпечувати систематичний прогрес у пошуку роботи.

1.2 Аналіз існуючих систем-аналогів

Один з найпопулярніших сайтів для пошуку роботи – "Indeed" . Компанія пропонує широкий спектр вакансій та інструментів для пошуку роботи на сайті. Indeed спеціалізується на ринку праці, тому точність підбору вакансій є досить високою.

Переваги Indeed:

- Великий обсяг вакансій з різних джерел
- Зручний інтерфейс та потужний пошуковий механізм
- Можливість створення резюме безпосередньо на сайті

Недоліки Indeed:

- Можлива наявність застарілих вакансій
- Обмежені можливості для networking

Indeed дає можливість шукати роботу як на повний, так і на неповний робочий день, а також віддалену роботу. Методи, використані для підбору вакансій, базуються на складних алгоритмах та машинному навчанні, що дозволяє застосовувати елементи нечіткої логіки при аналізі резюме та вакансій [2]. Загальний вигляд інтерфейсного вікна сайту продемонстровано на рис. 1.2 та рис. 1.3.

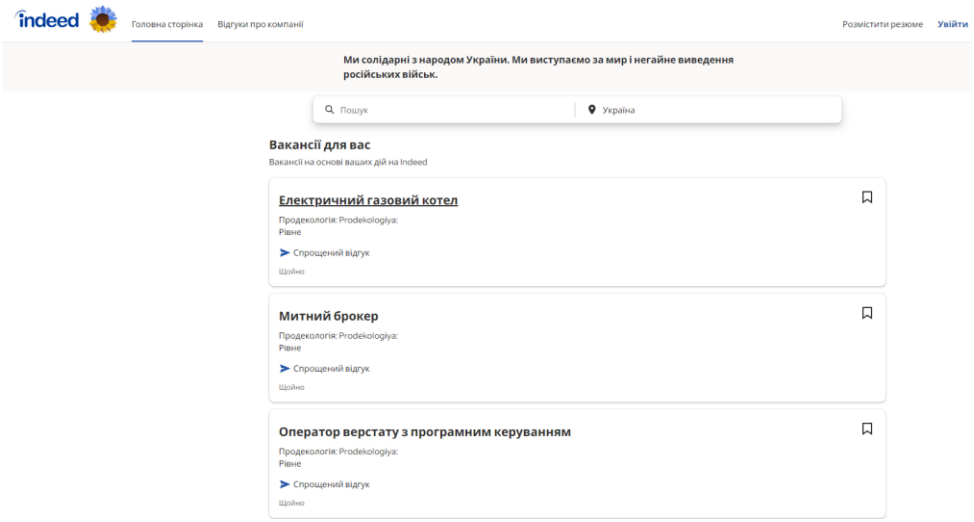

Рисунок 1.2 – Загальний вигляд інтерфейсного вікна сайту з вакансіями Indeed

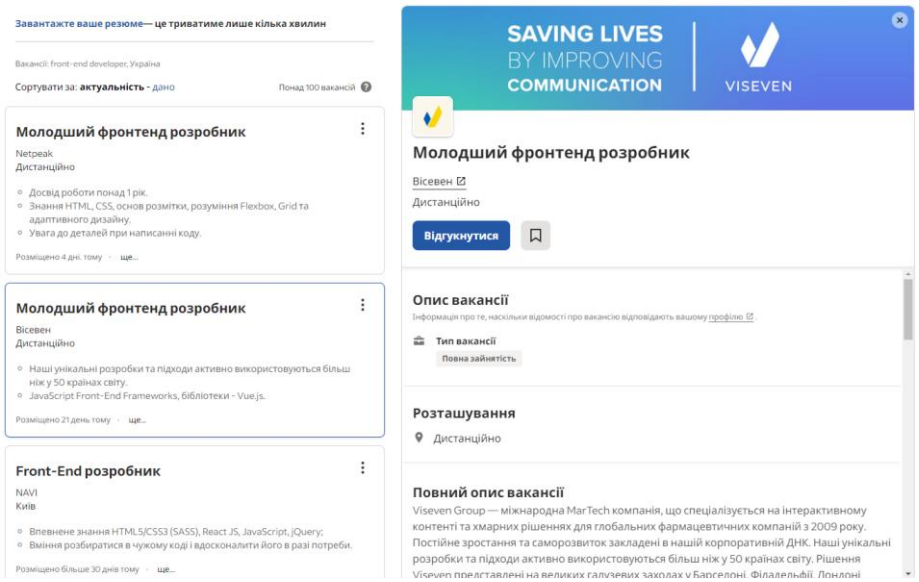

Рисунок 1.3 – Загальний вигляд інтерфейсного вікна сайту Indeed з детальним описом вакансії

Ще один сайт для пошуку роботи – "LinkedIn". Компанія пропонує не лише пошук вакансій, але й можливість створення професійного профілю на сайті.

Переваги LinkedIn:

- Потужні можливості для професійного нетворкінгу

- Інтеграція соціальної мережі та платформи для пошуку роботи
- Можливість отримувати рекомендації від колег та керівників
- Розширений функціонал для рекрутерів та здобувачів

Недоліки LinkedIn:

- Деякі розширені функції доступні лише в платній версії
- Можлива перенасиченість інформацією в стрічці новин
- Необхідність постійного оновлення профілю для ефективного пошуку

LinkedIn дає можливість пошуку роботи, використовуючи як технічний аналіз даних, так і соціальні зв'язки. Алгоритми LinkedIn застосовують методи нечіткої логіки для аналізу профілів користувачів та підбору релевантних вакансій [3]. Загальний вигляд інтерфейсного вікна сайту продемонстровано на рис. 1.4

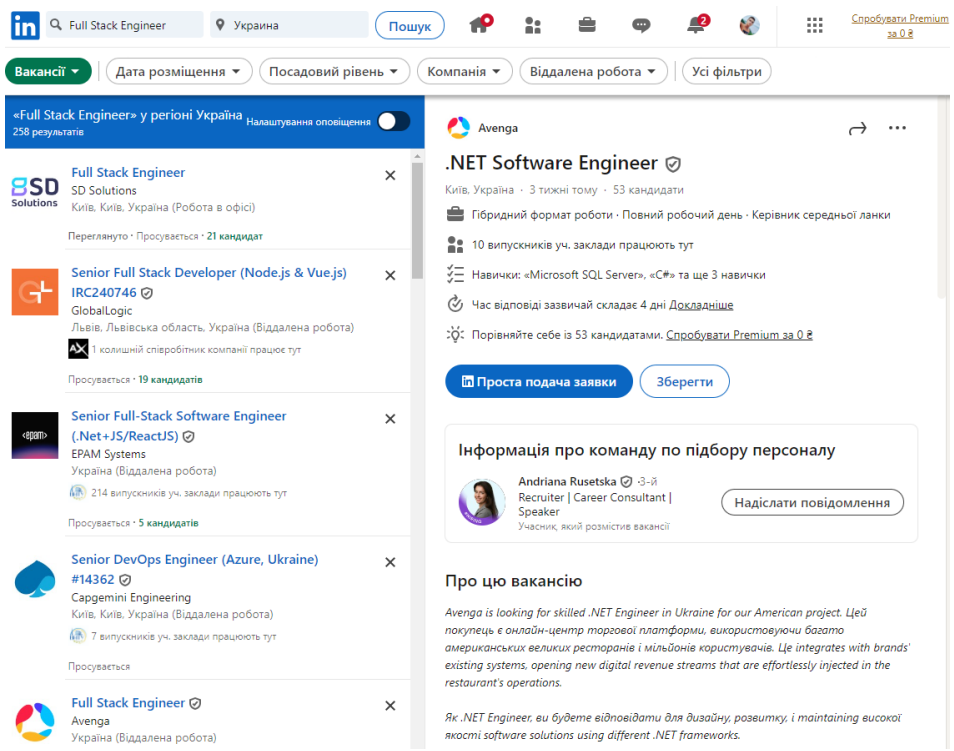

Рисунок 1.4 – Загальний вигляд інтерфейсного вікна сайту зі списком вакансій та детальний опис однієї з вакансій LinkedIn

Третім сайтом для пошуку роботи є Glassdoor. На сайті використовуються різні методи аналізу даних та нечіткої логіки для підбору вакансій та оцінки компаній.

Переваги Glassdoor:

- Унікальна інформація про зарплати та умови праці в компаніях
- Детальні відгуки про роботодавців від колишніх та поточних співробітників
- Можливість порівняння компаній та зарплат у різних галузях
- Корисні інсайти для підготовки до співбесіди

Недоліки Glassdoor:

- Можлива суб'єктивність відгуків про компанії
- Необхідність залишати власний відгук для доступу до повної інформації
- Менша кількість вакансій порівняно з іншими платформами

Glassdoor враховує такі фактори як: відгуки співробітників, рейтинг компаній, середній рівень заробітної плати тощо. Це дозволяє створити більш повну картину потенційного місця роботи. Алгоритми сайту використовують методи нечіткої логіки для аналізу текстових відгуків та формування загальної оцінки компанії [4].

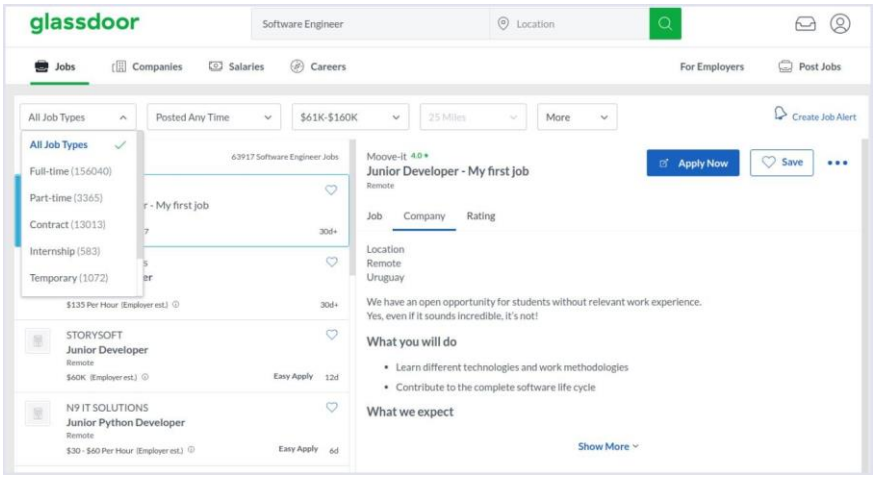

Рисунок 1.5 – Загальний вигляд інтерфейсного вікна сайту з вакансіями Glassdoor

Кожна з цих платформ використовує складні алгоритми аналізу даних та елементи нечіткої логіки для забезпечення найбільш релевантних результатів пошуку роботи. Ці технології дозволяють враховувати не лише точні співпадіння ключових слів, але й семантично близькі поняття, що підвищує ефективність пошуку як для здобувачів, так і для роботодавців.

Враховуючи усі недоліки, що було вказано, а також швидкий розвиток технологій аналізу даних та штучного інтелекту, є доцільним розробити нову, більш досконалу систему пошуку роботи. Така система могла б подолати обмеження існуючих платформ та запропонувати користувачам більш ефективні інструменти для пошуку роботи та розвитку кар'єри.

1.3 Аналіз відомих методів та методологій пошуку роботи

Станом на сьогодні, у людей, які шукають роботу або планують змінити місце роботи, існує чотири найпоширеніших підходи до організації власного процесу пошуку роботи:

Інтуїтивний підхід. Базується на підборі вакансій та відправленні резюме емпіричним шляхом. Зазвичай не має системності та чітко визначеної цілі. Людина відгукується на вакансії, які їй сподобалися, тоді, коли їй зручно.

Шаблонний підхід. Базується на виборі готових, вже кимось складених резюме та супровідних листів. Вибір шаблону виконується за принципом найбільшого рівня відповідності готового шаблону з цілями людини.

Підхід з використанням програмних продуктів та застосунків для пошуку роботи. Якісно відрізняється від двох попередніх підходів, оскільки при побудові подібних резюме використовуються особисті дані та показники людей, для яких це резюме буде створюватися, що робить його більш індивідуально налаштованим.

Консультативний підхід. Базується на делегуванні роботи з пошуку роботи профільним спеціалістам у галузі працевлаштування, тобто рекрутерам, кар'єрним консультантам тощо.

Потрібно зауважити, що часто люди комбінують використання одразу декількох підходів одночасно, з метою мінімізувати їх недоліки та максимізувати переваги. Порівняємо і визначимо переваги та недоліки кожного з підходів.

Основні переваги та недоліки інтуїтивного підходу наведено в таблиці 1.1.

Таблиця 1.1 – Основні переваги та недоліки інтуїтивного підходу

Переваги	Недоліки
- простий та зрозумілий у використанні;	- низька ефективність подібного підходу;
- не вимагає володіти знаннями у галузі працевлаштування;	- може дати результат тільки людині з мінімальним досвідом роботи або людині, чиї навички є дуже затребуваними на ринку праці;
- не потребує додаткових фінансових витрат;	- може призвести до тривалого та неефективного процесу пошуку роботи;
- на початковому етапі викликає більш позитивні враження та відданість процесу пошуку роботи.	- за час використання подібного підходу людина може напрацювати деструктивні навички (наприклад, неправильне складання резюме або невміння проходити співбесіди), які буде досить складно скоригувати в подальшому.

Визначимо основні переваги та недоліки шаблонного підходу та наведемо їх у таблиці 1.2.

Таблиця 1.2 – Основні переваги та недоліки шаблонного підходу

Переваги	Недоліки
- зрозумілий у використанні, оскільки потребує тільки безпосереднього використання вже складеного резюме та супровідного листа;	- необхідність зважено підходити до вибору готового шаблону резюме та супровідного листа;
- доступність шаблонів резюме та супровідних листів;	- необхідність перевіряти актуальність шаблонів резюме та супровідних листів щодо сучасних вимог роботодавців;
- значний обсяг накопиченої інформації про результати використання різних готових шаблонів резюме та супровідних листів;	- шаблони резюме та супровідних листів складно підлаштувати та адаптувати до індивідуальних потреб людини;
- широкий вибір місць та ресурсів для пошуку роботи;	- відсутність індивідуального підходу до особливостей досвіду та навичок людини.
- готові шаблони резюме та супровідних листів, які створені компетентними експертами у галузі працевлаштування, є ефективними у більшості випадків та задовольняють базові запити значної кількості людей.	- використання шаблонних фраз та кліше може зробити резюме та супровідний лист менш привабливими для роботодавців, оскільки вони можуть здаватися шаблонними та нецікавими. Роботодавці часто шукають унікальних кандидатів, які можуть виділитися з натовпу, тому індивідуальний підхід до складання резюме та супровідного листа може бути більш ефективним.

1.4 Висновок до розділу 1

У першому розділі було проведено аналіз сучасного стану розвитку методів та методологій пошуку роботи. Виділено основні підходи та принципи, які є ключовими в організації ефективного процесу пошуку роботи. Це, у свою чергу,

дозволило визначити основні складові процесу пошуку роботи: підготовка резюме та супровідного листа, безпосередній пошук вакансій та співбесіда.

Встановлено, що індивідуальний підхід до складання резюме та супровідного листа дозволяє максимально використати потенціал кожної людини та зробити їх більш привабливими для роботодавців. Циклічність дозволяє досягти прогресу та уникнути вигоряння, оскільки вона передбачає чергування фаз активного пошуку з фазами відпочинку та аналізу. З'ясовано, що принцип поступового підвищення складності є основою для досягнення та покращення результатів у пошуку роботи, а також допомагає уникнути стагнації чи вигоряння. Також встановлено, що використання різних методів пошуку роботи (наприклад, онлайн-платформи, кар'єрні сайти, соціальні мережі) дозволяє охопити ширшу аудиторію роботодавців та збільшити шанси на успіх.

Завдяки проведеному аналізу сучасного стану розвитку методів і методологій пошуку роботи вдалося систематизувати та накопичити достатньо інформації, необхідної для проектування нечіткої інтелектуальної системи.

2 МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

2.1 Обгрунтування вибору методів реалізації інформаційної технології

Останнім часом увага, яку приділяють науковці темі пошуку роботи та аналізу ринку праці, зросла в рази. Незважаючи на велику кількість досліджень, основною задачею дослідників є визначення системи критеріїв та показників, які впливають на ефективність пошуку роботи, загальний глобальний огляд трендів ринку праці, індикатори успішного працевлаштування та зв'язки між різними секторами економіки [5].

Зараз немає чіткої методології, яка б змогла в повній мірі описати процес пошуку роботи в розрізі моделювання та прогнозування. Основним завданням науковців, які займаються дослідженнями в цій сфері, є визначення системи критеріїв та показників, які чинять вплив на успішність працевлаштування, проте цього замало.

Розглянемо фундаментальний метод моделювання та прогнозування у сфері пошуку роботи. Фундаментальний аналіз – це метод оцінки перспектив працевлаштування, спроба виміряти справжню цінність кандидата на ринку праці, вивчаючи відповідні економічні, освітні та інші якісні та кількісні фактори. Фундаментальні аналітики вивчають все те, що може вплинути на успішність пошуку роботи, включаючи макроекономічні чинники, такі як загальна економічна ситуація та галузева кон'юнктура [6].

Попит на певні професії формується ринковим шляхом. Чим більший попит на певну спеціальність, тим вище шанси на працевлаштування та рівень заробітної плати. Попит, у свою чергу, залежить від тих переваг, які пропонує професія. Якщо завтра певна технологія стане надзвичайно популярною, то попит на фахівців у цій галузі "злетить до небес". Попит формується на тлі новин, нових технологій, анонсів компаній. Популярність певних навичок, освітні тренди та позитивні

новини про певні галузі також є ознакою швидкого зростання попиту на відповідних фахівців. Чим більше людей знає про перспективність певної професії, тим більше людей захочуть інвестувати у відповідну освіту або перекваліфікацію [7].

Оскільки успішність пошуку роботи залежить від попиту на ринку праці, то одним зі способів прогнозування тенденцій є кількісна оцінка цього попиту та подальший аналіз впливу попиту на зарплати та умови праці. Висновки про попит і популярність тих чи інших професій можна зробити, використовуючи дані про те, як часто ці теми обговорюються в інтернеті, наприклад, один з таких показників – кількість постів зі згадуванням відповідних професій у соціальній мережі LinkedIn.

З переваг цього показника можна виділити такі:

- є кількісним;
- технічно доступний для отримання чи обчислення;
- просто використовувати у математичних моделях.

Серед недоліків можна виділити такі:

- доступ до деяких даних може бути платним;
- доступ може бути обмежений по сумарній кількості отриманої інформації та швидкості отримання.

Ще один метод дослідження ринку праці – системний підхід. Такий метод визначає ринок праці як багаторівневу систему, яка інтегрується з різними сферами економіки та впливає на розвиток інших суб'єктів господарювання. В рамках цього підходу ринок праці розглядається як частина соціально-економічної системи, яка має свої специфічні закони розвитку та функціонування. Ринок праці є відкритою, складною, стохастичною, динамічною та керованою системою, яка постійно еволюціонує під впливом зовнішніх та внутрішніх факторів. У цієї системи є суб'єкт та об'єкт керування: суб'єктами виступають державні органи, підприємства та інші інституції, які здійснюють регулювання ринку праці, а об'єктом – безпосередньо сам ринок праці, що включає в себе всі механізми забезпечення зайнятості, найму та соціального захисту населення.

Системний підхід дозволяє враховувати не тільки внутрішні аспекти функціонування ринку праці, але й його взаємодію з іншими соціально-економічними системами, такими як ринок капіталу, товарів та послуг. Це дозволяє досліджувати ринок праці у контексті його ролі у загальному економічному розвитку країни. Важливим аспектом системного підходу є те, що він передбачає наявність певної структури, яка забезпечує керування процесами на ринку праці. Суб'єктами керування є державні та недержавні інституції, які формують стратегії розвитку ринку праці та приймають відповідні рішення щодо регулювання зайнятості, соціального захисту і умов праці.

З іншого боку, об'єкт керування – це сам ринок праці, що складається з набору елементів, таких як зайнятість, заробітна плата, професійна підготовка, попит і пропозиція робочої сили. Сукупність цих елементів забезпечує функціонування ринку праці як єдиної системи. Таким чином, системний підхід дозволяє глибше зрозуміти складність ринку праці, його взаємозв'язки з іншими сферами економіки та визначити шляхи його оптимального розвитку [8].

Наступним способом прогнозування тенденцій на ринку праці є використання нейронних мереж. Штучні нейронні мережі (ANN) як математичні моделі та їх програмні чи апаратні реалізації засновані на принципі організації та функціонування біологічних нейронних мереж. Ця концепція виникла під час вивчення процесів, що відбуваються в мозку, та спроб моделювати ці процеси. Суть нейронних мереж полягає у здатності адаптуватися до нових умов і навчатися на основі попереднього досвіду, що робить їх ефективними для розв’язання складних задач, пов’язаних із прогнозуванням.

Після розробки алгоритмів навчання отримані моделі використовуються у практичних цілях: для прогнозування тенденцій на ринку праці, розпізнавання навичок у резюме, автоматизації підбору персоналу та інших завдань, що вимагають обробки великої кількості даних. ANN використовуються для виявлення прихованих закономірностей у даних, що робить їх корисними в ситуаціях, коли традиційні методи аналізу ринку праці виявляються недостатньо ефективними.

ANN – це система пов'язаних та взаємодіючих простих процесорів (штучних нейронів). Такі процесори зазвичай досить прості, особливо в порівнянні з процесорами, що використовуються в персональних комп'ютерах. Кожен процесор у такій мережі обробляє лише періодично одержувані ним сигнали та періодично надсилає сигнали іншим процесорам. Однак, завдяки їхній здатності працювати у великих мережах із керованою взаємодією, ці прості процесори можуть колективно виконувати досить складні завдання, такі як аналіз великих обсягів даних про ринок праці, виявлення закономірностей, які неможливо помітити вручну, та прогнозування майбутніх трендів на ринку праці. Такі прогнози можуть бути надзвичайно корисними для державних та приватних організацій, що займаються регулюванням ринку праці та прийняттям стратегічних рішень [9].

Існують традиційні методи для аналізу та прогнозування ситуації на ринку праці з використанням математичних моделей. Такі методи використовуються, наприклад, у прогнозуванні попиту на певні професії, рівня заробітних плат, тенденцій працевлаштування тощо. Математичні моделі дозволяють підійти до процесу аналізу ринку праці більш об'єктивно, відкинувши емоції чи особисті переконання. Ще однією перевагою є точність, оскільки математичні моделі використовують точні дані для навчання і виявлення закономірностей, в той час як традиційні методи аналізу ринку праці часто базуються на даних, які важко систематизувати.

Проте класичні методи, які базуються на припущеннях стаціонарності математичних моделей, не відображають швидко змінні, нелінійні процеси сучасного ринку праці. На сучасному етапі з'являється все більше якісно нових моделей та методів їхнього дослідження. Одними з таких методів є аналіз на основі часових рядів та застосування нечіткої логіки.

Часовий ряд у контексті ринку праці - це послідовність даних, проіндексованих в хронологічному порядку, наприклад, щомісячні дані про рівень безробіття або кількість вакансій у певній галузі. Основними завданнями аналізу часових рядів є визначення характерних особливостей ринку праці, підбір статистичних моделей, прогнозування показників зайнятості та безробіття.

Аналіз часового ряду даних про ринок праці починається з побудови і вивчення його графіку. Спочатку потрібно перевірити часовий ряд на стаціонарність. Якщо часовий ряд нестаціонарний, то перш за все потрібно виділити нестаціонарну складову ряду, яка може відображати, наприклад, сезонні коливання попиту на робочу силу.

Наступним етапом після перевірки на стаціонарність є виділення тренду. Тренд у контексті ринку праці - це зміна, що визначає загальний напрям розвитку, основну тенденцію часових рядів, наприклад, зростання попиту на IT-спеціалістів або зниження попиту на некваліфіковану робочу силу.

Нечітка логіка може бути застосована для аналізу даних про ринок праці, особливо коли йдеться про якісні характеристики, які важко виразити точними числовими значеннями. Наприклад, оцінка "високої кваліфікації" працівника може бути нечіткою і залежати від багатьох факторів.

Використання нечіткої логіки в інформаційній технології пошуку роботи дозволяє врахувати неточності та невизначеності, притаманні процесу пошуку роботи та підбору персоналу. Це може включати оцінку відповідності кандидата вимогам вакансії, аналіз soft skills, прогнозування успішності кандидата на певній посаді тощо [10].

Отже, як висновок до теми інформаційної технології пошуку роботи, очевидно, що ця тема є актуальною та вимагає комплексного підходу. Зараз існує декілька способів вирішення такої задачі, але вони часто представлені у вигляді комерційних рішень і недостатньо повно розкривають деталі реалізації. Також було виявлено, що соціальні фактори мають вагомий вплив на ринок праці, проте вони слабо проаналізовані математично. Тобто проблема створення ефективної інформаційної технології пошуку роботи з використанням аналізу даних та нечіткої логіки залишається відкритою та актуальною для вивчення.

Таблиця 2.1 – Переваги та недоліки методологій пошуку роботи

Метод	Переваги	Недоліки
Фундаментальний аналіз	Комплексний підхід до оцінки перспектив працевлаштування; враховує макроекономічні чинники; дозволяє оцінити справжню цінність кандидата	Складність в обробці великої кількості факторів; може бути суб'єктивним; вимагає глибоких знань економіки та ринку праці
Аналіз попиту на основі соціальних мереж	Кількісний показник; технічно доступний для отримання; просто використовувати у математичних моделях	Доступ до деяких даних може бути платним; обмеження по кількості отриманої інформації та швидкості отримання; може не відображати реальний попит на ринку праці
Системний підхід	Розглядає ринок праці як частину більшої системи; враховує взаємозв'язки між різними елементами; дозволяє аналізувати вплив на інші сектори економіки	Складність моделювання через велику кількість змінних; може бути занадто абстрактним для практичного застосування; вимагає великого обсягу даних
Нейронні мережі (ANN)	Здатність обробляти великі обсяги даних; можливість виявляти складні нелінійні залежності: адаптивність до змін на ринку праці	Вимагають великого обсягу даних для навчання; можуть бути чутливими до якості вхідних даних
Аналіз часових рядів	Дозволяє виявляти тренди та сезонні коливання; можливість прогнозування на основі історичних даних; добре підходить для аналізу кількісних показників ринку праці	Може не враховувати раптові зміни на ринку; вимагає стабільності умов для точних прогнозів; обмежена здатність враховувати якісні фактори
Нечітка логіка	Дозволяє працювати з нечіткими поняттями та якісними характеристиками; може враховувати експертні знання; підходить для оцінки soft skills та інших складно вимірюваних параметрів	Складність у визначенні правил та функцій приналежності; може бути суб'єктивною при встановленні параметрів; обмежена точність у порівнянні з чіткими математичними моделями

2.2 Розробка математичної моделі аналізу даних та нечіткої логіки пошуку роботи

У сучасних умовах динамічного розвитку ринку праці, математичні моделі стають невід'ємною частиною ефективного аналізу даних, що стосуються пошуку роботи. Відзначаючи потребу у створенні більш точних та адаптивних систем, які здатні враховувати як об'єктивні, так і суб'єктивні фактори, розроблено модель, що поєднує в собі методи аналізу даних і нечіткої логіки. Ця модель дозволяє не лише аналізувати історичні дані, але й прогнозувати майбутні тенденції, зменшуючи ризик помилкових рішень в умовах невизначеності [11].

Вихідні дані та основні параметри моделі

Наша математична модель використовує вхідні дані, які можуть бути отримані з різних джерел: резюме кандидатів, вакансії, вимоги роботодавців, статистичні показники ринку праці, а також інформацію з соціальних мереж та професійних платформ. Нехай:

R_i — рейтинг кандидата i, що обчислюється на основі досвіду, навичок, освіти та інших критеріїв;

V_j — релевантність вакансії j, що визначається відповідністю вимог вакансії кваліфікації кандидата;

C_k — фактори, що впливають на процес пошуку роботи, включаючи економічні показники, соціальні мережі, рівень конкуренції на ринку, а також ринкові тенденції.

Рейтинги та релевантність вакансії можна виразити через формули:

$$R_i = \sum_{n=1}^{N} \mathrm{w}_n \cdot s_{i,n}, \tag{2.1}$$

де w_n вага n-го критерію (освіта, досвід, навички),

$s_{i,n}$ — оцінка кандидата за цим критерієм.

Це рівняння дозволяє створити комплексну оцінку, враховуючи різні аспекти професійного профілю кандидата [12].

Релевантність вакансії можна визначити за допомогою схожості вимог роботодавця та профілю кандидата:

$$V_i = \frac{\sum_{m=1}^{M} u_m \cdot f_{j,m}}{\sum_{m=1}^{M} u_m} \tag{2.2}$$

де u_m - вага m-го критерію вакансії,

$f_{j,m}$ — оцінка відповідності кандидата вимогам.

Це дозволяє ранжувати вакансії та виділяти найперспективніші пропозиції.

Ринок праці характеризується високим рівнем невизначеності, що є причиною використання нечіткої логіки. Нечітка логіка дозволяє моделювати ситуації, коли дані можуть бути не точними або неповними, що часто має місце у випадках пошуку роботи. У нашій моделі нечітка логіка використовується для врахування таких факторів, як "середній рівень зарплат", "конкурентоспроможність кандидата", "привабливість вакансії" та "степінь ризику" [13].

Для побудови нечіткої логіки використовуємо функції належності. Нехай функція належності для фактора зарплати виглядає так:

$$\mu_{salary}(x) = \begin{cases} 0, & x \le a \\ \dfrac{x-a}{b-a}, & a < x < b \\ 1, & x \ge b \end{cases} \tag{2.3}$$

де x — розмір зарплати,

a — мінімальна допустима зарплата,

b — максимальна бажана зарплата.

Це означає, що якщо зарплата кандидата нижча за a, то його шанси на отримання роботи зменшуються до нуля, а якщо вона вища за b, то шанси зростають до максимуму.

Аналогічні функції належності можна побудувати для інших критеріїв, таких як кваліфікація кандидата або популярність вакансії, зокрема, для факторів, що стосуються специфічних навичок, досвіду, а також загального попиту на ринку праці. На основі нечітких входів (наприклад, зарплата, рівень конкуренції) та встановлених правил можна здійснити нечітке виведення для оцінки ймовірності успішного працевлаштування кандидата. Правила типу "Якщо зарплата висока і кваліфікація кандидата висока, то ймовірність успішного працевлаштування висока" можуть бути формалізовані через такі нечіткі логічні імплікації:

$$\mu_{salary}(x) \cdot \mu_{qualification}(y) \rightarrow \mu_{success}(z) \tag{2.4}$$

Ці правила можуть бути адаптовані та розширені з урахуванням специфіки кожного окремого ринку праці, зокрема, регіональних особливостей або галузевих вимог.

Після отримання нечітких значень для факторів, необхідно виконати дефазифікацію, тобто перетворення нечіткого значення у чітке. Для цього можна застосувати метод центру ваги:

$$z^* = \frac{\int_z \cdot \mu_{success}(z)dz}{\int_z \mu_{success}(z)dz} \tag{2.5}$$

Цей результат z^* дає нам чітке значення, яке можна трактувати як ймовірність успішного працевлаштування кандидата. Чим вище значення z^*, тим більша ймовірність отримання роботи.

Описана математична модель може бути використана як інструмент для автоматизованих систем пошуку роботи. Системи, побудовані на базі цієї моделі, можуть забезпечувати не лише підбір вакансій на основі об'єктивних даних, але й використовувати нечітку логіку для врахування суб'єктивних факторів, таких як очікування кандидатів або тенденції на ринку. Наприклад, система може пропонувати вакансії, які не є ідеальними з точки зору кваліфікації, але які можуть бути цікавими для кандидатів через інші фактори, такі як можливості кар'єрного зростання або переваги компанії .

Запропонована математична модель пошуку роботи дозволяє комплексно аналізувати дані та враховувати нечіткі фактори, що значно підвищує точність прогнозування і якість рекомендацій для кандидатів. Поєднання аналізу даних і нечіткої логіки дає можливість створити систему, яка враховує як кількісні, так і якісні характеристики ринку праці. Подібний підхід може сприяти не лише підвищенню ефективності пошуку роботи, але й адаптації кандидатів до змінюваних умов ринку, що є важливим фактором у сучасному світі.

В подальшому дослідженні слід розглянути можливості інтеграції машинного навчання для покращення алгоритмів прогнозування та аналізу. Це дозволить зібрати більше даних про тенденції на ринку праці та адаптувати модель під специфічні потреби користувачів, що значно підвищить її ефективність та функціональність. Таким чином, математична модель стане важливим інструментом не лише для окремих кандидатів, але й для роботодавців, які шукають оптимальні рішення у підборі кадрів.

2.3 Розробка UML-діаграми послідовності інформаційної технології пошуку роботи

Для кращого розуміння та оптимізації розробки інформаційної технології для пошуку роботи, створимо UML-діаграму, яка детально відобразить процес роботи системи, включно з усіма ключовими етапами і взаємодіями між компонентами.

Розглянемо, для чого призначені UML-діаграми та як саме вони допомагають у процесі створення інформаційних систем. UML, або уніфікована мова моделювання, є стандартизованою мовою, що дозволяє візуально уявити структуру і динаміку процесів у системі, що є особливо важливим для проєктування складних систем, таких як системи рекомендацій чи пошуку роботи. UML є надзвичайно корисною, коли необхідно побудувати загальну концепцію системи, спроєктувати архітектуру на високому рівні або ж для представлення ідей на початкових етапах розробки.

UML-діаграми дають можливість представити систему графічно, щоб полегшити розуміння логіки її роботи. Інженери програмного забезпечення використовують UML для моделювання структурних і поведінкових аспектів системи. Вони створюють діаграми для відображення зв’язків між компонентами, що складають програму, та їхніх взаємодій, включно з обробкою даних, зберіганням та поверненням результатів. Це забезпечує більш системний підхід до розробки, коли система відображається як набір об’єктів, які комунікують між собою для виконання загального завдання. UML-діаграми допомагають краще розуміти систему як на рівні взаємодії між класами, так і на рівні бізнес-процесів, які відповідають за пошук, аналіз і зберігання даних про роботу.

Структурні діаграми UML фокусуються на статичних зв’язках між компонентами. Зокрема, UML-діаграма класів є однією з найбільш вживаних для візуалізації систем з об’єктно-орієнтованою архітектурою, що дозволяє детально побачити структуру класів, їхні атрибути та методи, а також зв’язки між ними. Це особливо важливо для такої інформаційної технології, як пошук роботи, де необхідно відобразити, як класи взаємодіють один з одним для отримання необхідної інформації, наприклад, інформації про вакансії чи профілі кандидатів. Діаграма залежностей корисна для організації взаємозв’язків між різними модулями, що дає можливість побачити, як один модуль залежить від іншого, та забезпечує більш структурований підхід до розробки.

Поведінкові UML-діаграми дозволяють відстежити динамічні процеси в системі. Для інформаційної технології пошуку роботи важливими є діаграми

діяльності та діаграми послідовностей. Діаграма діяльності є подібною до блок-схеми і може відобразити алгоритм або послідовність дій для процесів, таких як пошук вакансій чи аналіз резюме. Такий підхід є зручним для моделювання бізнес-процесів та послідовності етапів, через які проходять користувачі, починаючи від авторизації і закінчуючи переглядом результатів пошуку вакансій. Це допомагає оптимізувати кроки процесу для зменшення часу обробки запитів та підвищення ефективності.

Основний акцент у проєктуванні інформаційної технології для пошуку роботи зосереджено на UML-діаграмі послідовностей, яка дозволяє детально представити процеси взаємодії між основними елементами системи та послідовність обміну повідомленнями між ними. Ця діаграма дозволяє моделювати взаємодію користувача з системою, показуючи, як різні компоненти, зокрема веб-клієнт, сервер і база даних, обмінюються повідомленнями для виконання запиту. Наприклад, коли користувач шукає вакансію, сервер надсилає запит до бази даних, виконується відповідний пошук, а результати повертаються користувачеві через веб-інтерфейс. Це дозволяє побачити, як різні компоненти співпрацюють між собою для досягнення мети.

Крім того, для реалізації процесу пошуку роботи важливо відобразити і роль об’єкта користувача. Об’єкт користувача є одним з основних елементів, оскільки саме він ініціює всі запити і взаємодіє з інтерфейсом системи. Взаємодія починається з входу до системи, де користувач вводить свої облікові дані, і продовжується на кожному наступному етапі роботи системи.

На рисунку 2.1 представлена розроблена UML-діаграма послідовностей, яка зображує процес взаємодії користувача з системою пошуку роботи. Вона демонструє, як система переходить від етапу до етапу, виконуючи обробку даних та повертаючи користувачеві результати.

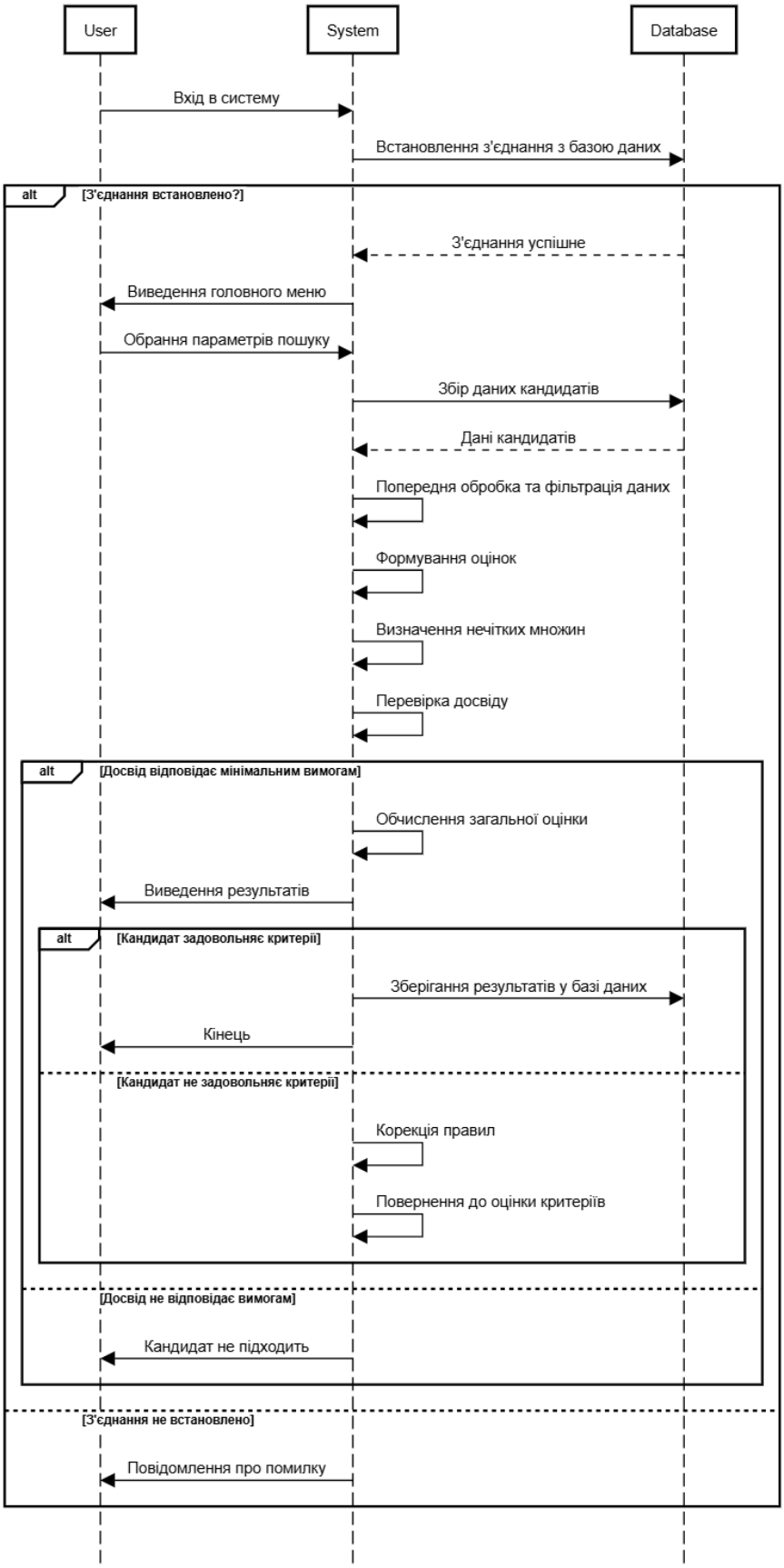

Рисунок 2.1 – UML-діаграма послідовності взаємодії модулів інформаційної технології анілізу даних та нечіткої логіки для пошуку роботи

2.4 Розробка структури інформаційної технології розпізнавання емоцій людини.

Розроблена інформаційна технологія пошуку роботи призначена для ефективного відбору кандидатів, спираючись на сучасні методи аналізу даних та нечіткої логіки. Процес починається з етапу збору інформації про кандидатів, де до уваги беруться такі параметри, як досвід роботи, професійні навички, рівень освіти та інші значущі характеристики, які можуть вплинути на їх відповідність вимогам вакансії. Зібрані дані необхідно ретельно підготувати для подальшого використання в системі, що включає очищення від непотрібної або неточної інформації, а також попередню фільтрацію. Такий підхід дозволяє уникнути помилок та забезпечити точність під час формування оцінок.

На наступному етапі система формує початкові оцінки кандидатів на основі заздалегідь визначених критеріїв, що враховують усі отримані дані. Цей процес дозволяє швидко оцінити рівень відповідності кожного кандидата, виходячи з базових вимог до посади. Для більш гнучкого аналізу система застосовує методи нечіткої логіки, які дозволяють обробляти дані з певним ступенем невизначеності. На основі нечіткої логіки формуються нечіткі множини, що описують такі важливі параметри, як досвід роботи, професійні навички та рівень зарплатних очікувань. Це допомагає створити більш адаптивний підхід до відбору, де враховуються навіть невеликі відхилення у вимогах, дозволяючи точніше оцінювати кандидатів.

Після визначення нечітких множин система виконує перевірку відповідності досвіду кандидата мінімальним вимогам вакансії. Якщо досвід відповідає умовам, процес переходить до етапу обчислення загальної оцінки. Під час цього етапу враховуються всі попередні оцінки та параметри, що дозволяє отримати повну картину щодо відповідності кандидата. Загальна оцінка є інтеграцією всіх раніше зібраних і оброблених даних, що робить її ключовим показником для кінцевої оцінки.

Результати оцінки виводяться для кінцевого користувача, що дозволяє переглянути рівень відповідності кожного кандидата. Крім того, система виконує

додаткову перевірку на відповідність усім критеріям, щоб забезпечити, що кандидат дійсно відповідає заданим вимогам. Якщо кандидат задовольняє всім вимогам, його профіль та оцінка зберігаються в базі даних для подальшого використання або рекомендацій. У разі невідповідності, система надає можливість скоригувати правила оцінки або повернутись до попередніх етапів, що підвищує точність та адаптивність технології до різних типів вакансій.

Таким чином, розроблена інформаційна технологія пошуку роботи дозволяє не тільки автоматизувати процес відбору кандидатів, але й робить його більш ефективним і точним, враховуючи складні аспекти людського досвіду та навичок за допомогою нечіткої логіки.

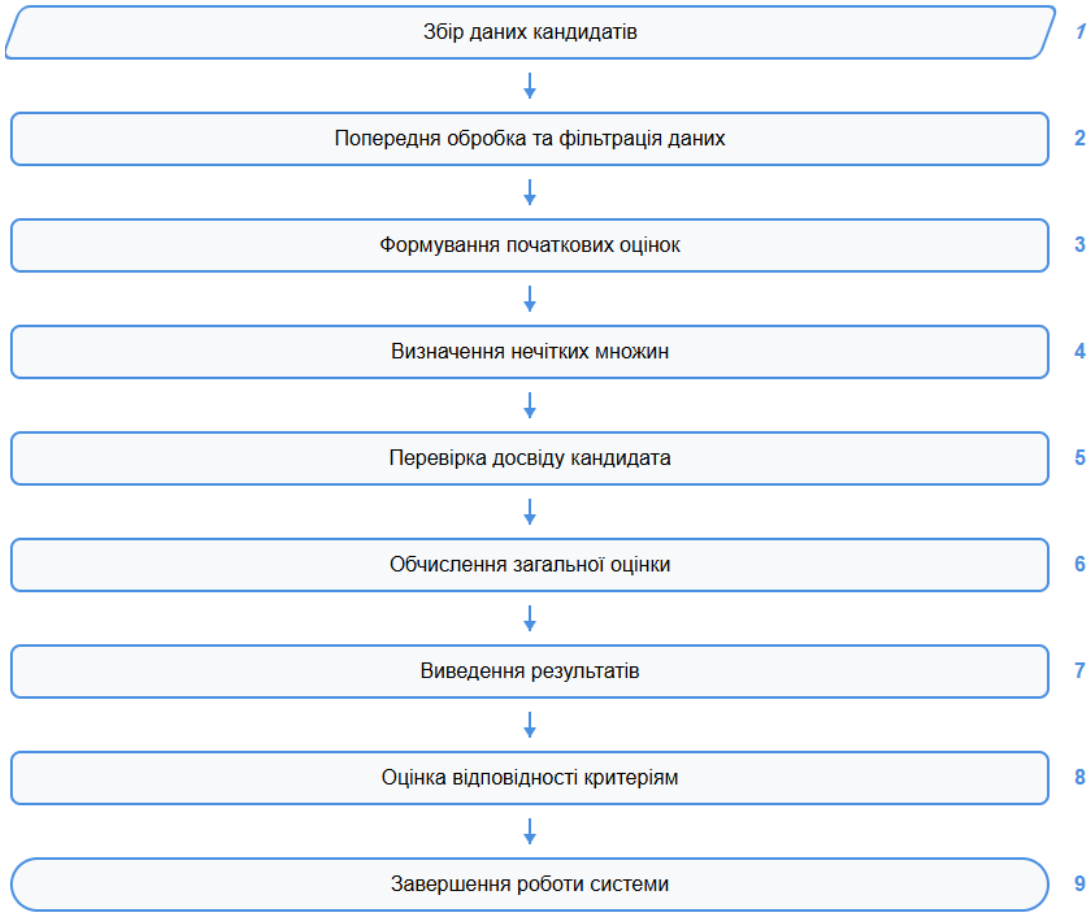

Рисунок 2.2 – Структура інформаційної технології пошуку роботи з використанням аналізу даних та нечіткої логіки

2.5 Висновок до розділу 2

У цьому розділі виконано комплексне моделювання інформаційної технології, спрямованої на полегшення процесу пошуку роботи та підвищення точності отриманих результатів. Здійснено обґрунтований вибір методів, що підходять для ефективної реалізації системи, враховуючи особливості обробки великих обсягів даних, необхідних для пошуку вакансій та підбору кандидатів. У цьому контексті було запропоновано математичну модель, яка включає методи аналізу даних і нечіткої логіки. Вони дозволяють розширити можливості алгоритмів пошуку, надаючи систему гнучкості для обробки різних типів запитів та врахування критеріїв, що не завжди піддаються точному визначенню. Завдяки цьому система забезпечує адаптацію до змінних вимог та унікальних запитів користувачів, підвищуючи релевантність результатів.

Також було розроблено UML-діаграму послідовності, яка відображає основні етапи та динаміку процесів інформаційної технології на кожному кроці взаємодії користувача із системою. Діаграма послідовності дає можливість візуально відтворити весь цикл роботи системи, включаючи обмін повідомленнями між її компонентами та користувачем. Це дозволяє чітко бачити функціональні взаємозв'язки між основними модулями, такими як веб-клієнт, сервер, база даних, і користувач, що забезпечує логічну узгодженість усіх процесів у системі.

Таким чином, моделювання інформаційної технології пошуку роботи охопило не лише технічні аспекти, але й адаптацію алгоритмів під потреби кінцевого користувача. Виконані кроки створюють основу для ефективної розробки системи, здатної обробляти інформацію гнучко, точно та адаптовано до специфічних запитів, що підвищує зручність і якість роботи з технологією.

3 ОБГРУНТУВАННЯ ВИБОРУ ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

3.1 Обгрунтування вибору архітектури інформаційної технології пошуку роботи.

Архітектура інформаційної технології пошуку роботи - це структура системи, яка відображає взаємодію між її компонентами, їх властивості та логічні частини, за які відповідає кожен елемент. Проектування архітектури є критичним етапом розробки такої технології. При створенні системи, орієнтуючись на поставлене технічне завдання, необхідно розробити оптимальну архітектуру, враховуючи ефективність роботи майбутньої системи, зв'язки основних елементів, а також передбачити можливість масштабування [14].

Інформаційна технологія пошуку роботи, що використовує аналіз даних та нечітку логіку, базується на клієнт-серверній архітектурі з додатковим аналітичним модулем. Розглянемо основні компоненти:

Сервер: Це потужна обчислювальна машина, призначена для збереження та обробки інформації про вакансії, резюме, тренди ринку праці, а також для передачі цих даних між користувачами системи.

Клієнт: Відповідає за взаємодію з користувачем через інтерфейс. Клієнтська частина збирає інформацію від шукачів роботи та роботодавців, за необхідності обробляє її та відправляє на сервер.

База даних: Зберігає всю необхідну інформацію, включаючи дані про вакансії, резюме, історичні дані про ринок праці, результати аналізу тощо. Структуру клієнт-серверної архітектури зображено на рисунку 3.1.

Аналітичний модуль: Ключовий компонент системи, який відповідає за аналіз даних та застосування методів нечіткої логіки. Цей модуль виконує такі функції:

- Аналіз часових рядів для виявлення трендів на ринку праці

- Застосування нечіткої логіки для оцінки відповідності кандидатів вакансіям

- Прогнозування попиту на певні професії та навички

API: Забезпечує взаємодію між різними компонентами системи, а також можливість інтеграції з зовнішніми сервісами (наприклад, з job-порталами або соціальними мережами).

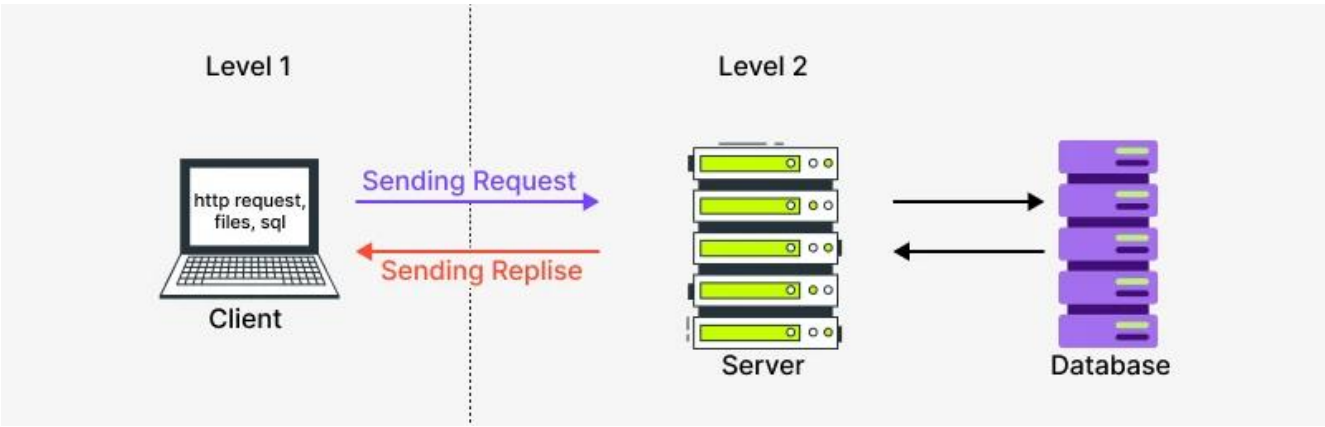

Рисунок 3.1 – Схема системи клієнт-сервер

Принцип роботи такої архітектури полягає в наступному:

- Клієнт збирає інформацію від користувача (наприклад, резюме або опис вакансії).
- Ці дані передаються на сервер за допомогою захищеного HTTP протоколу.
- Сервер зберігає отриману інформацію в базі даних та передає її аналітичному модулю.
- Аналітичний модуль обробляє дані, застосовуючи методи аналізу часових рядів та нечіткої логіки.
- Результати аналізу зберігаються в базі даних та можуть бути використані для подальших рекомендацій.
- Сервер формує відповідь на запит клієнта, використовуючи результати аналізу.

- Клієнт отримує відповідь та відображає результати користувачу.

Така архітектура дозволяє ефективно обробляти великі обсяги даних, застосовувати складні алгоритми аналізу та забезпечувати швидку взаємодію з користувачами. Крім того, вона передбачає можливість масштабування системи за рахунок додавання нових серверів або розширення аналітичного модуля [15].

3.2 Розробка схеми алгоритму аналізу даних та нечіткої логіки для пошуку роботи

Для забезпечення ефективності інформаційної технології пошуку роботи, необхідно розробити алгоритми, які допоможуть обробляти великі обсяги даних про кандидатів та здійснювати їхню оцінку з використанням методів нечіткої логіки. Такі алгоритми відіграють важливу роль у процесі підбору кадрів, оскільки вони дозволяють автоматизувати обробку інформації, зменшити ймовірність людської помилки та забезпечити більш об'єктивний підхід до оцінки кандидатів. Системи, що використовують нечітку логіку, здатні працювати з невизначеністю та неточностями, які часто виникають у реальних даних, таким чином підвищуючи якість прийняття рішень [16].

Алгоритми включають основні етапи збору, обробки та аналізу інформації, що дозволяє системі надавати більш точні результати відповідності кандидатів певним критеріям. У цьому розділі представлені два основні алгоритми: перший алгоритм обробки даних кандидатів та побудови системи оцінки, а також другий алгоритм з варіативними умовами для гнучкої роботи з оцінкою кандидатів на основі нечітких правил [17].

Графічне подання алгоритму аналізу даних та побудови системи оцінки на основі нечіткої логіки наведено на рисунку 3.3.

Опишемо наведений алгоритм аналізу даних користувачів та побудови системи оцінки кандидатів:

1. Збір даних кандидатів: на першому кроці відбувається збір необхідних даних про кандидатів, включаючи інформацію про їхній досвід роботи, навички, освіту тощо.

2. Попередня обробка та фільтрація даних: зібрані дані проходять етап попередньої обробки, в ході якого виконується фільтрація та очищення інформації для подальшого аналізу.

3. Формування оцінок: на основі оброблених даних формуються початкові оцінки кандидатів відповідно до заздалегідь визначених критеріїв.

4. Визначення нечітких множин: визначаються нечіткі множини для ключових параметрів, таких як досвід роботи, навички та зарплатні очікування, які будуть використані у наступних етапах.

5. Перевірка досвіду: оцінюється, чи відповідає досвід роботи кандидата мінімальним вимогам. Якщо так, процес переходить до наступного кроку; якщо ні, виводиться результат, що кандидат не підходить.

6. Обчислення загальної оцінки: на цьому кроці здійснюється обчислення загальної оцінки відповідності кандидата, з урахуванням всіх попередніх етапів.

7. Виведення результатів: після обчислення загальної оцінки результати виводяться для подальшого використання у процесі підбору кадрів.

8. Оцінка задоволення критеріями: на цьому етапі проводиться оцінка, чи задовольняє кандидат визначені критерії. Якщо задовольняє, завершується алгоритм; якщо ні, повертається до етапу корекції правил.

9. Завершення роботи: завершується робота алгоритму, результати зберігаються у базі даних для подальшого використання.

Другий алгоритм описує взаємодію користувача з системою для отримання результатів оцінки кандидатів на підставі зібраних даних та оброблених через нечітку логіку (рис. 3.4).

Алгоритм взаємодії користувача складається з таких кроків:

1. Вхід в систему: процес починається з того, що користувач виконує вхід у систему, вводячи свої облікові дані.

2. Встановлення з'єднання з базою даних: після успішного входу система намагається встановити з'єднання з базою даних для подальшої роботи.

3. З'єднання встановлено?: на цьому етапі система перевіряє, чи вдалося встановити з'єднання з базою даних.

a) Якщо з'єднання встановлено, алгоритм переходить до наступного кроку.

b) Якщо з'єднання не встановлено, система виводить повідомлення про помилку та завершує процес.

4. Виведення головного меню: при успішному встановленні з'єднання система виводить головне меню, де користувач може вибрати подальші дії.

5. Обрання параметрів пошуку: користувач обирає необхідні параметри для пошуку даних, які його цікавлять.

6. Аналіз та розрахунок: система проводить аналіз вибраних параметрів і виконує необхідні розрахунки для отримання результатів.

7. Підготовка результатів: після проведення розрахунків система готує результати для виведення.

8. Виведення результатів: останній етап передбачає виведення підготовлених результатів для користувача.

9. Кінець: алгоритм завершується, і користувач може вирішити, чи продовжити роботу у системі або вийти з неї.

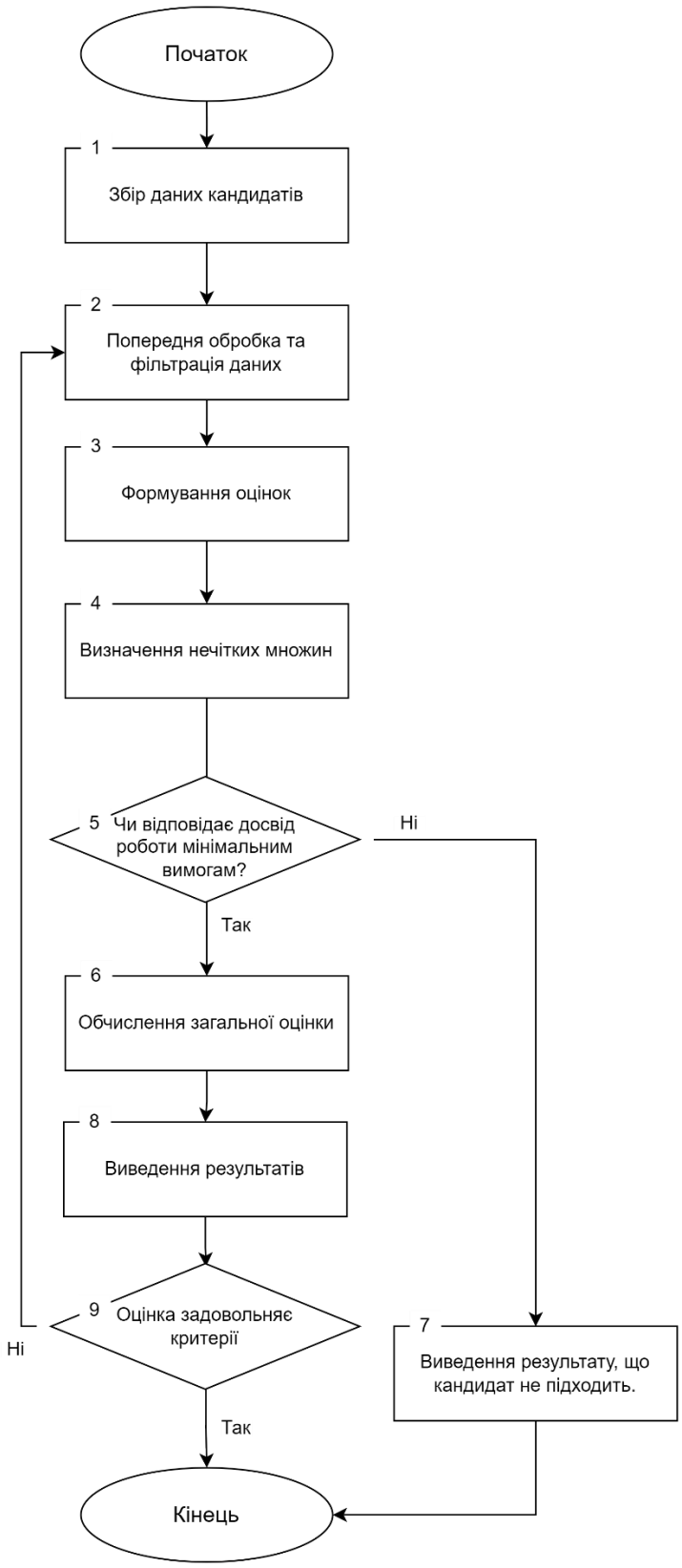

Рисунок 3.3 – Схема роботи аналізу даних та побудова системи оцінки на основі нечіткої логіки

Розглянемо схемупослідовності при взаємодії користувача з модулем оцінки (рис. 3.4).

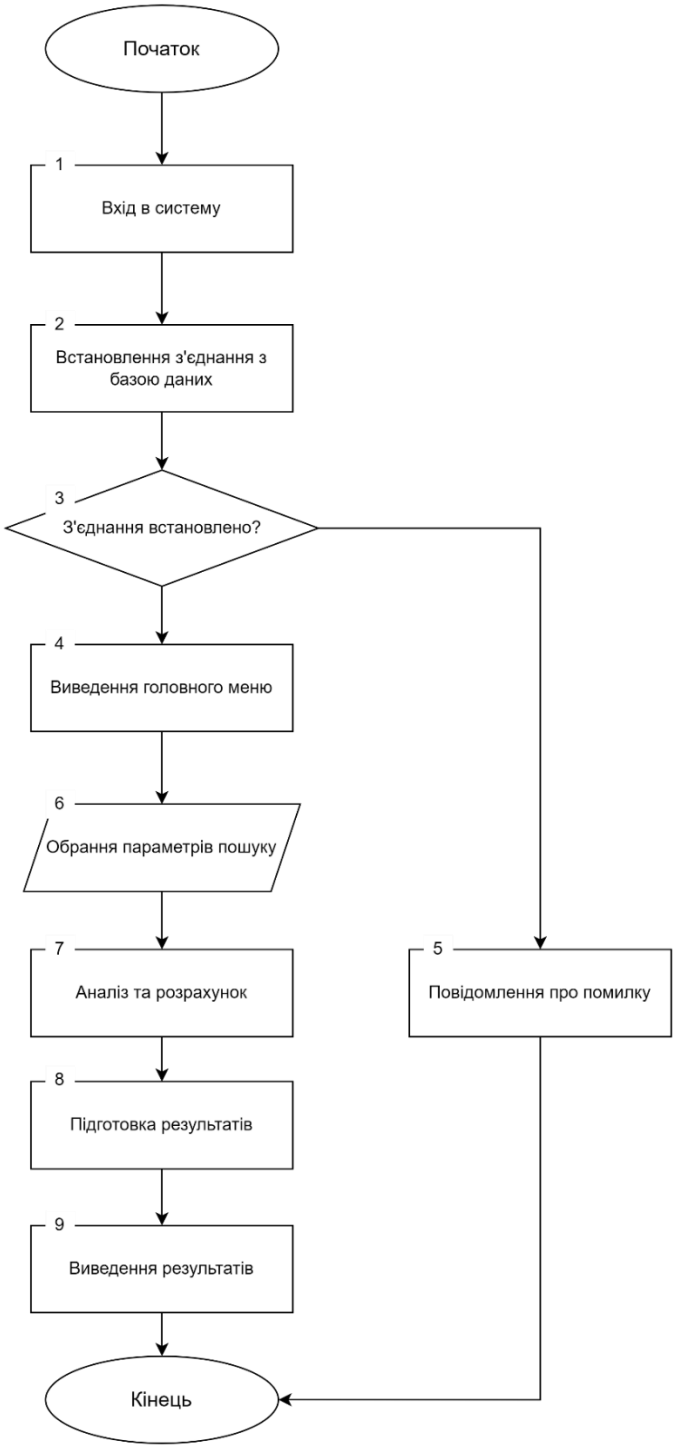

Рисунок 3.4 – Схема послідновності при взаємодії користувача з модулем оцінки кандидатів

3.3 Обгрунтування вибору мови програмування

Вибір мови програмування є важливим етапом у процесі розробки програмного забезпечення. Мова програмування визначає підхід до вирішення задач, структурування даних, інтеграцію з іншими технологіями та навіть продуктивність кінцевого рішення. У сучасному світі існує велика кількість мов програмування, кожна з яких має свої переваги та недоліки, а також специфічні сфери застосування. Цей розділ присвячений огляду різних мов програмування, їх характеристикам, а також обґрунтуванню вибору JavaScript та React для розробки інформаційної технології пошуку роботи.

Мови програмування зазвичай класифікуються за різними критеріями, такими як рівень абстракції (високорівневі та низькорівневі мови), спосіб інтерпретації або компіляції (інтерпретовані чи компільовані), а також за парадигмами програмування (об'єктно-орієнтоване програмування, функціональне програмування тощо). Нижче розглянемо кілька найбільш популярних мов програмування, їх особливості, переваги та недоліки.

Python — це мова програмування загального призначення, яка є однією з найбільш популярних та затребуваних мов у сучасному світі технологій. Вона була розроблена в 1991 році Гвідо ван Россумом як засіб для спрощення процесу програмування та надання можливості писати код, який легко читати та розуміти. Основна мета Python полягала в тому, щоб створити мову з простим синтаксисом, яка при цьому зберігатиме високу функціональність і гнучкість для вирішення широкого кола завдань [18].

Однією з найбільших переваг Python є його високоуровневий підхід до програмування, що означає, що програмісту не потрібно працювати з низькорівневими аспектами, такими як управління пам'яттю. Python дозволяє зосередитися на розв'язанні проблем, а не на технічних деталях, завдяки чому він широко використовується як початківцями, так і досвідченими розробниками.

Переваги Python:

- Простота та читабельність: Код на Python легко читати і писати, що робить його чудовим вибором для початківців.
- Велика кількість бібліотек та фреймворків: Python підтримує багато фреймворків для різних завдань, таких як Django та Flask для веб-розробки, NumPy та TensorFlow для аналізу даних і машинного навчання.
- Кросплатформеність: Python працює на багатьох операційних системах, таких як Windows, Linux і macOS.

Недоліки Python:

- Продуктивність: Через інтерпретований характер, Python повільніший у порівнянні з компільованими мовами, такими як C++ або Java.
- Не підходить для мобільних додатків: Python не є оптимальним вибором для розробки мобільних застосунків [19].

Java — це одна з найвідоміших і найбільш використовуваних об'єктно-орієнтованих мов програмування, яка була розроблена компанією Sun Microsystems у 1995 році. Після придбання Sun Microsystems компанією Oracle в 2010 році, Java отримала новий імпульс до розвитку та стала ще більш впливовою в індустрії розробки програмного забезпечення. Java була створена з ідеєю забезпечити легкість у написанні коду, що може працювати на різних платформах без необхідності адаптації для кожної з них. Це стало можливим завдяки концепції "Пиши один раз, запускай будь-де" (Write Once, Run Anywhere), що є однією з основних переваг Java. Така кросплатформенність досягається за рахунок віртуальної машини Java (JVM), яка дозволяє запускати Java-код на різних операційних системах.

Основна ідея Java полягає в тому, щоб програмний код був універсальним та незалежним від апаратної платформи. Це стало можливо завдяки тому, що Java-код спочатку компілюється в байт-код, який потім виконується на JVM. Це дозволяє програмам на Java працювати на будь-яких пристроях, де встановлена JVM, що робить Java дуже популярною мовою для кросплатформних рішень.

Переваги Java:

- Кросплатформеність: Завдяки JVM (Java Virtual Machine), програми на Java можуть виконуватись на різних платформах без змін у коді.
- Безпека: Java надає високий рівень безпеки, що робить її ідеальною для розробки фінансових та корпоративних застосунків.
- Широка спільнота: Велика кількість розробників і наявність великої кількості інструментів та бібліотек полегшує процес розробки.

Недоліки Java:

- Відносно складний синтаксис: У порівнянні з мовами, такими як Python або JavaScript, синтаксис Java вимагає більше коду для виконання простих завдань.
- Вимоги до ресурсів: Програми на Java можуть споживати більше пам'яті та процесорних ресурсів [20].

C++ — це мова програмування, яка була створена в 1983 році данським вченим Б'ярне Страуструпом як розширення мови C, що на той момент вже була досить популярною завдяки своїй ефективності і низькорівневому доступу до апаратних ресурсів комп'ютера. Одним із головних завдань при розробці C++ було збереження ефективності та продуктивності мови C, при цьому додаючи нові концепції, такі як об'єктно-орієнтоване програмування, яке дозволяє організовувати код у вигляді об'єктів, що представляють реальні сутності та їх поведінку.

C++ швидко стала однією з найпотужніших і найпоширеніших мов програмування, особливо для розробки програмного забезпечення, яке вимагає високої продуктивності та ефективного використання ресурсів комп'ютера. Це пояснюється тим, що C++ дозволяє розробникам отримувати майже прямий доступ до апаратури, керувати пам'яттю вручну і уникати зайвих витрат на абстракції, які можуть уповільнювати виконання програм.

Переваги C++:

- Висока продуктивність: Програми на C++ мають високу швидкість виконання завдяки компіляції та управлінню пам'яттю.

- Контроль над системними ресурсами: Можливість прямого управління пам'яттю робить C++ чудовою мовою для низькорівневих систем.

Недоліки C++:

- Складність: Синтаксис C++ є одним із найскладніших, що може ускладнити його вивчення.
- Ручне управління пам'яттю: Хоча це надає гнучкість, але також може призвести до помилок, таких як витік пам'яті [21].

JavaScript — це мова програмування, яка була створена у 1995 році Бренданом Айком для того, щоб забезпечити динамічну взаємодію користувачів із веб-сторінками. Спочатку JavaScript використовувалася як клієнтська мова програмування для покращення користувацького досвіду на веб-сторінках, дозволяючи додавати анімацію, валідацію форм, реагувати на події користувача в реальному часі. Однак з того часу JavaScript еволюціонувала у потужний та універсальний інструмент, який використовується не лише для фронтенд-розробки, але й для серверної розробки, мобільних додатків, десктопних програм та навіть роботи з інтернетом речей (IoT).

З плином часу JavaScript стала однією з найпопулярніших і найширше використовуваних мов програмування у світі, завдяки її високій гнучкості, кросплатформеності та можливостям інтеграції з іншими мовами. Основною перевагою JavaScript є її здатність виконуватися безпосередньо в браузері користувача, що дозволяє створювати інтерактивні веб-додатки з миттєвою реакцією на дії користувача. Це означає, що динамічний контент можна генерувати і оновлювати без необхідності перезавантаження всієї сторінки, що значно підвищує зручність користування веб-сайтами.

Переваги JavaScript:

- Широке застосування у веб-розробці: JavaScript є невід'ємною частиною фронтенд-розробки та широко використовується для створення інтерактивних веб-сайтів.

- Кросплатформеність: JavaScript працює у всіх сучасних браузерах без потреби в додаткових інструментах або встановленнях.
- Велика кількість фреймворків і бібліотек: Існують популярні фреймворки, такі як React, Angular, Vue, які значно спрощують розробку складних веб-додатків.
- Швидкість розробки: Завдяки фреймворкам і великій кількості бібліотек JavaScript дозволяє швидко створювати веб-застосунки та інтерактивні елементи.

Недоліки JavaScript:

- Безпека: Оскільки JavaScript виконується безпосередньо в браузері, це відкриває можливість для різних типів атак, таких як XSS (Cross-Site Scripting).
- Недоліки продуктивності: Хоча JavaScript досить швидкий для більшості веб-завдань, у порівнянні з компільованими мовами він має нижчу продуктивність [22].

PHP (Hypertext Preprocessor) — це серверна мова програмування, створена для генерації динамічних веб-сторінок та розробки веб-додатків. Вона є однією з найпопулярніших мов для веб-розробки, особливо для серверної частини сайтів. Від моменту свого створення у 1994 році Рамусом Лердорфом, PHP значно еволюціонувала, перетворившись на потужний інструмент для побудови сучасних веб-додатків, здатних обробляти велику кількість запитів та взаємодіяти з різними базами даних. PHP активно використовується у розробці таких популярних систем управління контентом (CMS), як WordPress, Drupal, Joomla. Близько 78% всіх веб-сайтів, які використовують серверні мови програмування, використовують PHP. Це робить мову одним з ключових гравців у сфері веб-розробки.

PHP забезпечує інтеграцію з багатьма популярними базами даних, такими як MySQL, PostgreSQL, Oracle, SQLite, що дозволяє створювати надійні та масштабовані рішення. Наприклад, через простоту інтеграції з MySQL, PHP часто використовується для побудови баз даних для веб-сайтів, що зберігають інформацію про користувачів, замовлення, товари і т.д. Крім цього, PHP підтримує роботу з різними протоколами, такими як HTTP, FTP, SMTP, IMAP, що дозволяє легко створювати багатофункціональні додатки.

Переваги PHP:

- Простота використання: PHP відомий своїм легким для вивчення синтаксисом, що робить його популярним вибором для початківців.
- Підтримка баз даних: PHP легко інтегрується з багатьма базами даних, такими як MySQL та PostgreSQL.
- Велика кількість фреймворків: Існують потужні фреймворки, такі як Laravel, які значно спрощують розробку на PHP.

Недоліки PHP:

- Проблеми з безпекою: Раніші версії PHP мали багато проблем із безпекою, хоча в останніх версіях було зроблено значні покращення.
- Швидкість виконання: PHP не є найшвидшою серверною мовою в порівнянні з іншими, такими як Node.js [23].

Після аналізу різних мов програмування можна дійти висновку, що для розробки веб-додатків на основі інформаційної технології пошуку роботи JavaScript є найкращим вибором. Це обумовлено кількома факторами:

- Широке застосування у фронтенд-розробці: Оскільки веб-сайт потребує інтерактивного та динамічного інтерфейсу, JavaScript у поєднанні з React дозволяє створювати зручні та швидкі інтерфейси.
- Кросплатформеність: JavaScript працює на будь-якій платформі, що робить його універсальним інструментом для веб-розробки.
- Підтримка React: React, як один із найпопулярніших фреймворків, спрощує розробку компонентів та керування станом додатка, що особливо важливо для складних веб-додатків.
- Велика спільнота: JavaScript має величезну спільноту розробників, що забезпечує підтримку та доступ до численних ресурсів, таких як документація, бібліотеки та інструменти.

Таким чином, JavaScript у поєднанні з React є оптимальним вибором для розробки сучасної веб-технології пошуку роботи [24].

3.4 Обґрунтування вибору середовища розробки

При розробці інформаційної технології для аналізу даних та використання нечіткої логіки у процесі пошуку роботи важливим є вибір оптимального середовища для розробки, яке забезпечує ефективність, зручність та високу продуктивність. Існує багато інтегрованих середовищ розробки (IDE), кожне з яких пропонує свої переваги та недоліки. У цій секції детально розглянемо популярні середовища розробки, їх можливості та обґрунтуємо вибір Visual Studio Code (VS Code) для реалізації цього проєкту на базі Node.js та React.

Інтегроване середовище розробки (IDE) зазвичай включає текстовий редактор, компілятор або інтерпретатор, відладчик і засоби автоматизації збірки. Однак сучасні IDE значно розширюють ці можливості та пропонують інтеграцію з системами контролю версій, інструменти для управління залежностями, підтримку юніт-тестування, а також численні додаткові інструменти для ефективного управління проєктами. Популярні IDE, такі як Visual Studio, PyCharm, Eclipse, Atom, Sublime Text, і WebStorm, мають свої особливості та сфери використання, тому важливо порівняти їх, щоб зрозуміти, чому Visual Studio Code є найбільш оптимальним вибором для проєкту.

Visual Studio — потужне IDE від Microsoft, призначене для розробки широкого спектра застосунків, включаючи десктопні, мобільні та веб-застосунки. Воно пропонує повну інтеграцію з .NET-платформою, що робить його відмінним вибором для розробки на C# та інших мовах, пов'язаних із технологіями Microsoft. Visual Studio включає численні інструменти для налаштування інтерфейсу користувача, відладки, роботи з базами даних та інтеграції з Azure. Однак через свою вагомість і орієнтацію на більш складні проєкти Visual Studio часто є "важким" рішенням для легких або середніх проєктів, зокрема для веб-розробки на JavaScript, Node.js або React.

WebStorm від JetBrains — потужне середовище для розробки JavaScript, яке підтримує всі основні фреймворки та бібліотеки, включаючи React, Angular і Vue.js.

WebStorm забезпечує глибоку інтеграцію з Node.js і має чудову підтримку TypeScript, що є великою перевагою для сучасних веб-застосунків. Середовище надає ефективні інструменти для навігації по коду, рефакторингу, налагодження та роботи з системами контролю версій. Проте WebStorm є комерційним продуктом, що означає необхідність ліцензії, яка може бути недоступною для студентів чи невеликих проєктів з обмеженим бюджетом.

PyCharm також розроблений компанією JetBrains і в першу чергу орієнтований на Python-розробку. Він пропонує широкий спектр можливостей для аналітики коду, юніт-тестування та роботи з веб-фреймворками Django та Flask. PyCharm ідеально підходить для проєктів, що передбачають інтенсивну обробку даних або розробку програмного забезпечення на Python. Однак для JavaScript і Node.js цей вибір буде не найкращим, оскільки PyCharm не забезпечує глибокої інтеграції з екосистемою JavaScript і має обмежену підтримку сучасних веб-фреймворків.

Eclipse — відоме середовище для розробки на Java, яке підтримує багато інших мов за рахунок плагінів. Воно відрізняється потужністю і широким функціоналом, включаючи роботу з базами даних, управління залежностями, інтеграцію з системами контролю версій. Однак Eclipse важко адаптувати під легковажні фреймворки на кшталт React і Node.js, і воно може бути занадто складним для проєктів, що передбачають переважно фронтенд-розробку.

Sublime Text і Atom — легкі текстові редактори, які підтримують великий вибір плагінів для розширення функціональності. Вони підходять для швидкого редагування коду і мають зручний інтерфейс. Проте вони не мають такого ж широкого набору інструментів, як класичні IDE, зокрема в них відсутні вбудовані засоби для комплексного налагодження чи розгорнутої підтримки інтеграції з базами даних. Atom особливо підходить для розробки на JavaScript, оскільки він побудований на Electron, що дозволяє легко налаштувати середовище для роботи з Node.js. Однак Sublime Text і Atom поступаються іншим IDE за функціональністю і гнучкістю для великих проєктів.

Visual Studio Code (VS Code), у свою чергу, є оптимальним вибором для цього проєкту з кількох причин. По-перше, VS Code забезпечує потужну підтримку екосистеми JavaScript, включаючи Node.js і фреймворк React, які є основними технологіями для реалізації інформаційної системи пошуку роботи. По-друге, VS Code безкоштовний, що робить його доступним для розробників будь-якого рівня. Він підтримує великий вибір розширень, що дозволяє налаштувати середовище під конкретні потреби проєкту. Наприклад, за допомогою розширень можна додати підтримку TypeScript, GraphQL, REST-клієнтів та інших інструментів, що є корисними для повноцінної веб-розробки.

Серед функцій VS Code, які є особливо корисними для даного проєкту, можна виділити інтеграцію з Git, зручний інтерфейс для навігації по коду, підтримку для налагодження Node.js, а також вбудовану можливість для тестування. Крім того, можливість працювати з Docker безпосередньо у VS Code дозволяє легко налаштувати середовище для розгортання і тестування, що є важливим для сучасних веб-застосунків. Завдяки легкості інтерфейсу та гнучкості налаштувань VS Code є зручним як для великих проєктів, так і для швидкої прототипізації, що дозволяє розробнику залишатися продуктивним на всіх етапах розробки.

Отже, вибір Visual Studio Code для цього проєкту був зумовлений його відповідністю вимогам веб-розробки, наявністю розширень для всіх необхідних технологій та відсутністю обмежень у використанні. Завдяки широкій підтримці JavaScript і Node.js, інтеграції з Git та можливості гнучкого налаштування VS Code є ідеальним середовищем для реалізації інформаційної системи пошуку роботи з використанням нечіткої логіки [25].

3.5 Реалізація програмного модуля інформаційної технології пошуку роботи

У розробці інформаційної технології для пошуку роботи використовуються React для фронтенду та NestJS для бекенду, що дозволяє ефективно управляти запитами та обробляти їх на сервері. Для організації роботи з базою даних створюється кілька моделей, кожна з яких відповідає за певну частину даних, а також сервіси для взаємодії з цими даними. Розглянемо детальніше, як це виглядає в коді.

Першим кроком є створення моделей, що представляють основні сутності системи. Модель User зберігає інформацію про користувачів. Вона містить основні атрибути, такі як email, ім’я та пароль. Це дозволяє ефективно керувати користувачами в системі:

```
export type UserDocument = HydratedDocument<User>;
@Schema({ timestamps: true })
export class User {
    @Prop()
    name: string;
    @Prop({ required: true })
    email: string;
    @Prop({ required: true })
    password: string;
    @Prop()
    age: number;
    @Prop()
    gender: string;
    @Prop()
    address: string;
    @Prop({ type: Object })
    company: {
        _id: mongoose.Schema.Types.ObjectId;
        name: string;
```

```
    };
    @Prop({ type: mongoose.Schema.Types.ObjectId, ref: Role.name })
    role: mongoose.Schema.Types.ObjectId;
    @Prop()
    refreshToken: string;
    @Prop({ type: Object })
    createdBy: {
        _id: mongoose.Schema.Types.ObjectId;
        email: string;
    };
    @Prop({ type: Object })
    updatedBy: {
        _id: mongoose.Schema.Types.ObjectId;
        email: string;
    };
    @Prop({ type: Object })
    deletedBy: {
        _id: mongoose.Schema.Types.ObjectId;
        email: string;
    };
    @Prop()
    createdAt: Date;
    @Prop()
    updatedAt: Date;
    @Prop()
    isDeleted: boolean;
    @Prop()
    deletedAt: Date;
}
export const UserSchema = SchemaFactory.createForClass(User);
```

Модель Job відповідає за вакансії, включаючи їх опис, вимоги та тип роботи. Вона важлива для представлення вакансій, які користувачі можуть переглядати та на які подавати заявки:

```
export type JobDocument = HydratedDocument<Job>;

@Schema({ timestamps: true })
export class Job {
	@Prop()
	name: string;
	@Prop({ type: Object })
	company: {
		_id: mongoose.Schema.Types.ObjectId;
		name: string;
		logo: string;
	};
	@Prop()
	salary: number;
	@Prop()
	endDate: Date;
	@Prop()
	isActive: boolean;
	@Prop({ type: Object })
	createdBy: {
		_id: mongoose.Schema.Types.ObjectId;
		email: string;
	};
	@Prop({ type: Object })
	updatedBy: {
		_id: mongoose.Schema.Types.ObjectId;
		email: string;
	};
	@Prop({ type: Object })
	deletedBy: {
		_id: mongoose.Schema.Types.ObjectId;
		email: string;
	};
	@Prop()
	deletedAt: Date;
```

```
}
export const JobSchema = SchemaFactory.createForClass(Job);
```

Далі ми переходимо до створення сервісів, які дозволяють працювати з базою даних. Ці сервіси виконують запити на отримання вакансій, додавання нових вакансій та обробку заявок. Це критичний елемент системи, оскільки вся логіка обробки даних повинна бути централізована в цих класах.

Метод findAll відповідає за отримання списку вакансій з пагінацією та додатковими параметрами для фільтрації, сортування та заповнення зв'язків (populating). Спочатку, за допомогою бібліотеки aqp, розбирається рядок запиту (qs), щоб отримати об'єкти фільтрації, сортування та популяції. Потім він обчислює кількість записів на сторінці та загальну кількість сторінок. Для цього виконується запит до бази даних, де застосовуються фільтрація, пагінація, сортування та популяція (заповнення зв'язків). Результати запиту повертаються разом з метаданими, що містять поточну сторінку, кількість елементів на сторінці, загальну кількість сторінок та загальну кількість вакансій.

Метод findOne знаходить конкретну вакансію за її ідентифікатором. Спочатку перевіряється, чи є ідентифікатор валідним об'єктом MongoDB (через mongoose.Types.ObjectId.isValid). Якщо ідентифікатор не валідний, повертається повідомлення про помилку. Якщо він валідний, виконується пошук вакансії за допомогою методу findById.

Метод update відповідає за оновлення вакансії за її ідентифікатором. Він приймає дані для оновлення, а також інформацію про користувача, який виконує оновлення (для запису, хто саме оновив вакансію). Використовується метод updateOne для оновлення конкретної вакансії в базі даних. Після оновлення повертається результат цієї операції.

Метод remove відповідає за видалення вакансії. Спочатку перевіряється, чи є ідентифікатор валідним. Якщо ні, повертається повідомлення про помилку. Якщо ідентифікатор валідний, проводиться оновлення вакансії з вказівкою, хто саме видалив вакансію, після чого вакансія м'яко видаляється за допомогою методу

softDelete. Цей метод не видаляє запис з бази даних, а просто позначає його як видалений.

Ці методи разом з контролером дозволяють керувати вакансіями через API з можливістю фільтрації, сортування, популяції зв'язків, а також зручними операціями для оновлення та видалення вакансій.

Сервіс для вакансій виглядає так:

```
async findAll(currentPage: number, limit: number, qs: string) {
        const { filter, sort, population } = aqp(qs);
        delete filter.current;
        delete filter.pageSize;
        let offset = (+currentPage - 1) * +limit;
        let defaultLimit = +limit ? +limit : 10;
        const totalItems = (await this.jobModel.find(filter)).length;
        const totalPages = Math.ceil(totalItems / defaultLimit);
        const result = await this.jobModel
            .find(filter)
            .skip(offset)
            .limit(defaultLimit)
            .sort(sort as any)
            .populate(population)
            .exec();
        return {
            meta: {
                current: currentPage,
                pageSize: limit,
                pages: totalPages,
                total: totalItems,
            },
            result,
        };
    }
    async findOne(id: string) {
        if (!mongoose.Types.ObjectId.isValid(id)) return `не знайдено роботи`;
```

```
        return await this.jobModel.findById(id);
    }
    async update(_id: string, updateJobDto: UpdateJobDto, user: IUser) {
        const updated = await this.jobModel.updateOne(
            { _id },
            {
                ...updateJobDto,
                updatedBy: {
                    _id: user._id,
                    email: user.email,
                },
            },
        );
        return updated;
    }
    async remove(_id: string, user: IUser) {
        if (!mongoose.Types.ObjectId.isValid(_id)) return `не знайдено роботи`;

        await this.jobModel.updateOne(
            { _id },
            {
                deletedBy: {
                    _id: user._id,
                    email: user.email,
                },
            },
        );
        return this.jobModel.softDelete({
            _id,
        });
    }
}
```

Наступним етапом є реалізація контролерів, які обробляють HTTP-запити. У цьому випадку контролери приймають запити від користувачів та направляють їх

до відповідних сервісів для обробки. Цей код визначає контролер JobsController для роботи з вакансіями в NestJS. Контролер реалізує стандартні CRUD операції для роботи з вакансіями: створення, отримання, оновлення та видалення.

У конструкторі контролера ініціалізується сервіс JobsService, який містить логіку для обробки запитів. Для кожної операції контролер викликає відповідний метод сервісу, передаючи необхідні дані.

Метод create відповідає за створення нової вакансії. Він приймає дані вакансії через DTO (CreateJobDto) та інформацію про користувача, який ініціює запит, і передає їх у сервіс для подальшої обробки.

Метод findAll використовується для отримання списку вакансій з пагінацією. Він приймає параметри сторінки та кількості елементів на сторінці через запит і передає їх до сервісу разом з додатковими параметрами запиту для пошуку вакансій.

Метод findOne повертає конкретну вакансію за її ідентифікатором. Ідентифікатор вакансії передається як параметр шляху.

Метод update дозволяє оновити вакансію за її ідентифікатором. Він приймає ідентифікатор вакансії, нові дані через DTO (UpdateJobDto) і інформацію про користувача, який ініціює оновлення, передаючи їх до сервісу для виконання операції.

Метод remove відповідає за видалення вакансії за її ідентифікатором. Як і в попередньому методі, передається ідентифікатор вакансії та інформація про користувача, який виконує операцію видалення.

Кожен з методів контролера використовує декоратори для надання метаданих, таких як шляхи запитів, HTTP методи (POST, GET, PATCH, DELETE), а також спеціальні повідомлення для відповіді.

Контролер для вакансій виглядає так:

```
@ApiTags('jobs')
@Controller('jobs')
export class JobsController {
        constructor(private readonly jobsService: JobsService) {}
```

```
@Post()
@ResponseMessage('Створіть нову роботу')
create(@Body() createJobDto: CreateJobDto, @User() user: IUser) {
	return this.jobsService.create(createJobDto, user);
}
@Get()
@Public()
@ResponseMessage('Отримати роботу з Pagition')
findAll(
	@Query('current') currentPage: string,
	@Query('pageSize') limit: string,
	@Query() qs: string,
) {
	return this.jobsService.findAll(+currentPage, +limit, qs);
}
@Get(':id')
@Public()
@ResponseMessage('Отримайте роботу за ідентифікатором')
findOne(@Param('id') id: string) {
	return this.jobsService.findOne(id);
}

@Patch(':id')
@ResponseMessage('Оновіть роботу')
update(
	@Param('id') id: string,
	@Body() updateJobDto: UpdateJobDto,
	@User() user: IUser,
) {
	return this.jobsService.update(id, updateJobDto, user);
}

@Delete(':id')
@ResponseMessage('Видалити роботу')
remove(@Param('id') id: string, @User() user: IUser) {
```

```
            return this.jobsService.remove(id, user);
        }
}
```

У процесі пошуку вакансій для кандидатів важливо забезпечити ефективний механізм пошуку, який враховує не лише точні відповідності запитів, але й дозволяє знайти вакансії, які можуть бути релевантними, навіть якщо запит не є абсолютно точним. Одним із способів реалізації такого пошуку є використання fuzzy search — технології, яка дозволяє знаходити близькі збіги навіть у випадках, коли користувач вводить не зовсім точний запит. Це може бути корисно, наприклад, коли кандидат шукає вакансії, але вводить деякі орфографічні помилки або не використовує точні терміни. Використання бібліотек, як-от fuzzy-search, дозволяє реалізувати пошук, що враховує різні варіації запиту, порівнюючи їх з даними вакансій, зберігаючи їх у базі даних.

Для цього підключаємо бібліотеку fuzzy-search, яка дозволяє здійснювати пошук за різними критеріями, такими як назва вакансії або її опис. Вакансії отримуються з бази даних за допомогою SQL-запитів, що забезпечує актуальність і точність даних. Цей підхід допомагає знайти найбільш релевантні вакансії навіть при неточному або неповному введенні користувачем.

Крім того, застосування машинного навчання, зокрема нейронних мереж, дозволяє вдосконалити пошукову систему, створюючи алгоритми, які адаптуються до нових даних. Наприклад, для прогнозування сумісності кандидата з вакансією можна використовувати нейронну мережу. Прогнозування може базуватися на різних параметрах, таких як навички кандидата і вимоги вакансії. Нейронні мережі можуть бути навчені на основі історичних даних, де зберігаються взаємозв'язки між кандидатами та вакансіями, а потім використовуватися для передбачення найбільш ймовірних варіантів.

Приклад використання бібліотеки fuzzy-search для пошуку схожих вакансій:

```
import { FuzzySearch } from 'fuzzy-search';
import { Pool } from 'pg'; // Використовуємо pg для роботи з PostgreSQL
import dotenv from 'dotenv';
```

```
dotenv.config();
// Налаштування підключення до бази даних
const pool = new Pool({
  user: process.env.DB_USER,
  host: process.env.DB_HOST,
  database: process.env.DB_NAME,
  password: process.env.DB_PASSWORD,
  port: 5432,
});
async function getJobListings(): Promise<any[]> {
  const result = await pool.query('SELECT id, title, description FROM job_listings');
  return result.rows;
}
async function searchJobs(query: string) {
  try {
    const jobs = await getJobListings();
    const searcher = new FuzzySearch(jobs, ['title', 'description']);
    const results = searcher.search(query);
    console.log(`Знайдено вакансій: ${results.length}`);
    results.forEach((job, index) => {
      console.log(`${index + 1}. ${job.title} - ${job.description}`);
    });
  } catch (error) {
    console.error('Помилка при пошуку вакансій:', error);
  }
}
```

У цьому контексті використання бібліотеки, як-от neurojs, дозволяє створити модель, яка вчиться на основі вхідних даних, таких як кількість навичок кандидата і річний досвід роботи. Коли кандидат намагається знайти вакансію, система використовує ці дані для прогнозування того, наскільки кандидат підходить для кожної конкретної вакансії. Наприклад, якщо кандидат має кілька навичок, які збігаються з вимогами вакансії, система може присвоїти високу ймовірність сумісності, що дозволяє порекомендувати цю вакансію кандидату.

Реалізація цієї функціональності включає кілька етапів. Спочатку потрібно отримати необхідні дані, такі як навички кандидата і вимоги вакансії, з бази даних, після чого ці дані обробляються та передаються до моделі для прогнозування. Нейронна мережа може працювати з відносними значеннями, наприклад, співвідношенням кількості навичок кандидата до кількості необхідних навичок у вакансії, або порівнянням досвіду роботи кандидата з досвідом, який вимагається для вакансії.

Ці технології не лише спрощують процес пошуку вакансій для кандидатів, але й дозволяють зробити цей процес більш точним і персоналізованим. Застосування fuzzy search і нейронних мереж може значно покращити ефективність пошуку вакансій та допомогти кандидатам знаходити найкращі варіанти. Це також забезпечує велику гнучкість системи, оскільки вона може адаптуватися до нових даних і умов. Завдяки такому підходу можна досягти значного покращення якості та швидкості пошуку, а також підвищити рівень задоволення користувачів.

У загальному, використання таких методів, як fuzzy search і нейронні мережі, дозволяє реалізувати складніші алгоритми для пошуку вакансій і прогнозування сумісності кандидатів. Це не тільки підвищує точність результатів, але й дозволяє створити інтелектуальну систему, яка може адаптуватися до змінних умов та потреб користувачів, що в свою чергу підвищує конкурентоспроможність і ефективність процесу підбору кадрів.

Приклад використання neurojs для прогнозування сумісності кандидатів та вакансій:

```
import { NeuralNetwork } from 'neurojs';
import { Pool } from 'pg';
import dotenv from 'dotenv';
dotenv.config();
// Налаштування підключення до бази даних
const pool = new Pool({
  user: process.env.DB_USER,
  host: process.env.DB_HOST,
  database: process.env.DB_NAME,
```

```
  password: process.env.DB_PASSWORD,
  port: 5432,
});
async function getCandidateData(candidateId: number): Promise<any> {
  const result = await pool.query('SELECT skills, experience_years FROM candidates WHERE id = $1', [candidateId]);
  return result.rows[0];
}
// Створення та навчання нейронної мережі
const net = new NeuralNetwork();
net.train([
  { input: [0, 0], output: [0] },
  { input: [1, 1], output: [1] },
  { input: [0, 1], output: [0] },
  { input: [1, 0], output: [1] },
]);
// Функція для прогнозування сумісності кандидата з вакансією
async function predictJobFit(candidateId: number, jobId: number) {
  try {
    const candidate = await getCandidateData(candidateId);
    const job = await pool.query('SELECT required_skills, experience_years FROM job_listings WHERE id = $1', [jobId]);
    const candidateSkills = candidate.skills.split(',').length;
    const jobSkills = job.rows[0].required_skills.split(',').length;

    const input = [candidateSkills / jobSkills, candidate.experience_years / job.rows[0].experience_years];
    const prediction = net.run(input);
    const fit = prediction[0] > 0.5 ? 'Висока сумісність' : 'Низька сумісність';
    console.log(`Прогноз для кандидата #${candidateId} і вакансії #${jobId}: ${fit}`);
  } catch (error) {
    console.error('Помилка при прогнозуванні:', error);
  }
}
predictJobFit(1, 3);
```

3.6 Тестування інформаційної технології пошуку роботи

Тестування є надзвичайно важливим етапом у розробці будь-якого нового програмного продукту. На цьому етапі потрібно впевнитись, що програма виконує всі необхідні функції, працює без збоїв і помилок та відповідає вимогам, які до неї ставляться. Поряд із перевіркою технічних аспектів, тестування дозволяє також оцінити зручність користування та загальну якість інтерфейсу, що забезпечує позитивний досвід користувача. Основною метою тестування є виявлення всіх можливих проблем і недоліків в роботі програми до того, як вона буде передана кінцевим користувачам. Це дає можливість своєчасно виявити дефекти, усунути їх і запобігти виникненню серйозних проблем у майбутньому. Таким чином, тестування не лише допомагає удосконалити продукт, а й значно знижує ймовірність виникнення неприємностей після його запуску.

У процесі тестування було проведено понад 100 тестових запусків додатку, що дозволило ретельно перевірити всі функціональні можливості програмного забезпечення та оцінити його загальну ефективність. Це включало перевірку усіх основних функцій, взаємодію з користувачем, а також стабільність роботи при різних сценаріях використання. Для початку роботи з інформаційними технологіями користувачеві необхідно мати доступ до інтернет-браузера. Важливим аспектом є забезпечення стабільного підключення до мережі Інтернет, що може бути здійснене через мобільну мережу або за допомогою Wi-Fi з'єднання. Відповідно, для того, щоб програма працювала без збоїв і мала можливість обробляти запити користувачів, стабільне з’єднання є критичним фактором. На рисунку 3.5 представлено вікно реєстрації користувача, яке є важливим етапом входу до системи і дозволяє користувачеві отримати доступ до необхідних функцій додатку.

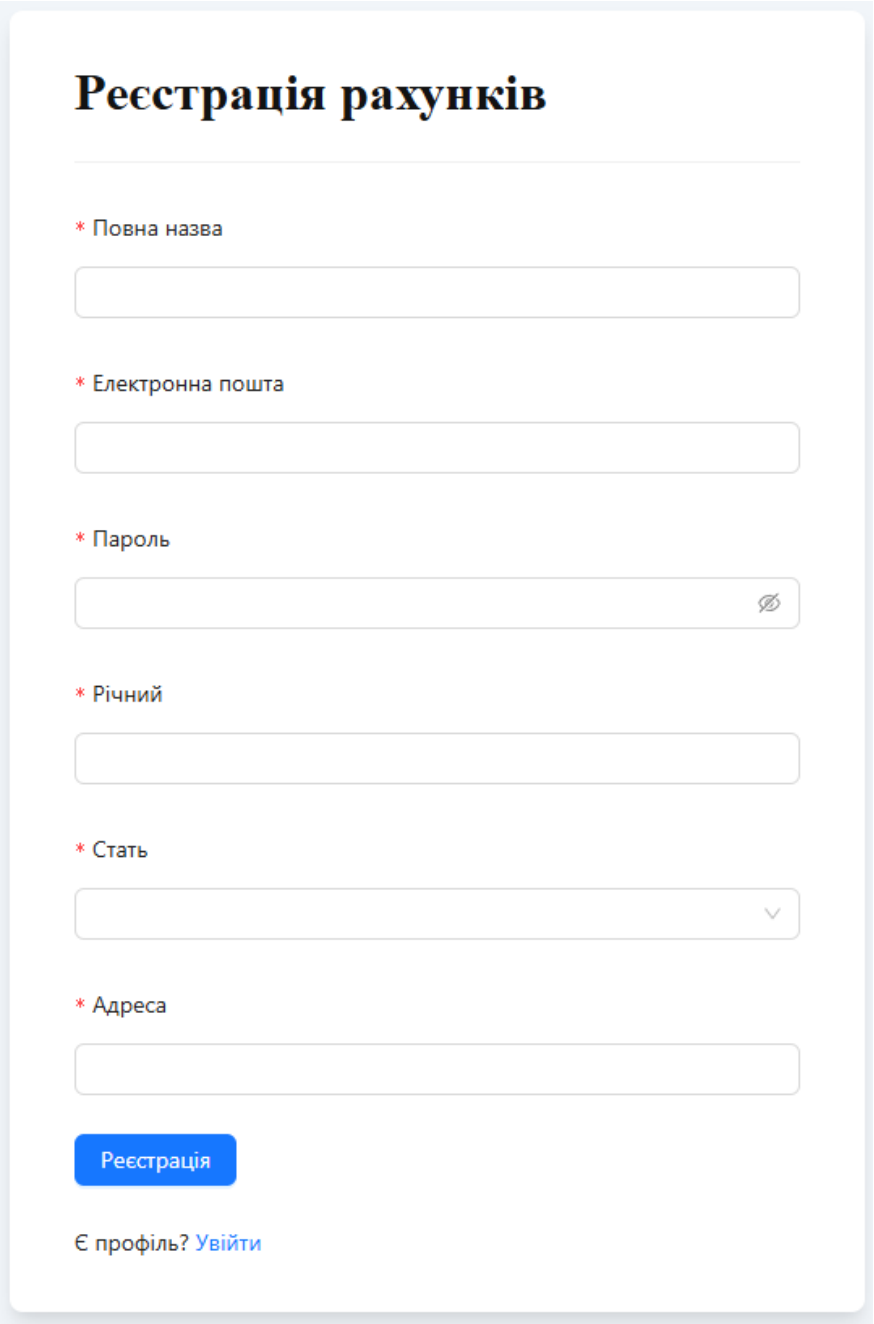

Рисунок 3.5 – Загальний вигляд інтерфейсного вікна реєстрації користувача

Після успішної авторизації користувач потрапляє на головну сторінку сайту, яка вітає його простим і зрозумілим екраном привітання. У верхній частині сторінки розташована панель навігації, що дозволяє легко переміщатися між розділами платформи. Користувачеві доступні основні функції, необхідні для ефективного пошуку роботи: перегляд вакансій, пошук за різними параметрами,

розділ про компанії, можливість створення резюме та інші ресурси, призначені для успішного працевлаштування.

Головна сторінка надає загальну інформацію про можливості платформи, зокрема розділи з описом категорій вакансій, гнучкими графіками роботи та варіантами віддаленої співпраці, що робить сайт зручним для різних типів користувачів — від початківців до професіоналів. У вітальному вікні також є коротка інформація про те, як розпочати пошук роботи, що спрощує початкове використання системи для нових користувачів.

Навігація у верхній частині екрану надає доступ до різних розділів сайту. Розділ "Вакансії" дозволяє користувачу переглядати всі доступні вакансії, а також пропонує фільтрацію за різними критеріями, такими як розташування, сфера діяльності, вимоги до кваліфікації, тип зайнятості тощо. Завдяки цьому користувач може швидко знайти вакансії, які найбільше відповідають його інтересам і кваліфікації. Функція пошуку вакансій забезпечена інтуїтивно зрозумілим інтерфейсом, що полегшує процес пошуку і дозволяє налаштувати критерії під особисті потреби.

Окрім основного розділу з вакансіями, у верхній панелі навігації доступні й інші розділи, такі як "Про компанії", де можна ознайомитися з інформацією про роботодавців, "Розробник резюме", що допомагає користувачам створити професійне резюме, "Кандидати", де розміщені профілі інших шукачів роботи, що дає змогу порівняти досвід та кваліфікацію. Також є розділ "Блог", де публікуються статті, корисні поради та новини щодо працевлаштування, допомагаючи користувачам бути в курсі сучасних тенденцій на ринку праці.

Завдяки структурованій навігації, користувач може швидко знайти потрібний розділ і отримати необхідну інформацію. Платформа побудована таким чином, щоб забезпечити максимальну зручність користувачам, допомогти їм легко орієнтуватися у функціях сайту та швидко приступити до пошуку відповідних вакансій. На головній сторінці також є можливість підписатися на оновлення нових вакансій, отримувати персональні рекомендації та налаштовувати повідомлення про роботу, що підходить під вказані параметри.

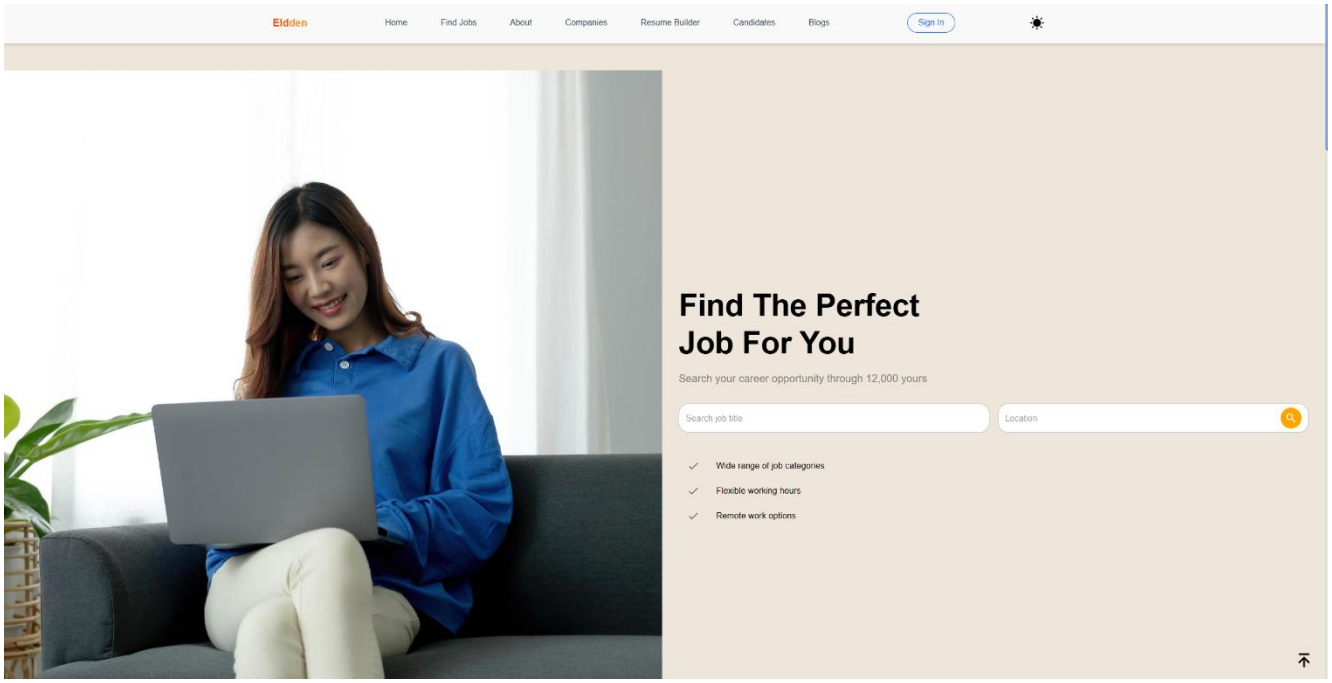

Рисунок 3.6 – Загальний вигляд вікна головної сторінки

Переходячи до розділу "Рекомендовані вакансії", користувач потрапляє на сторінку, що демонструє список вакансій, підібраних спеціально для нього на основі профілю та вказаних параметрів – рисунок 3.7. На цій сторінці користувач може скористатися розширеним пошуком і налаштуванням фільтрів, щоб уточнити вибір вакансій.

Інтерфейс забезпечує комфортний огляд з різними варіантами для налаштування пошуку. Користувач може змінювати тип вакансії, обирати рівень досвіду, та застосовувати інші критерії для пошуку саме тих позицій, що йому підходять. Завдяки адаптивному дизайну сайту користувач швидко знайде вакансії, які відповідають його вимогам.

Натиснувши на одну з вакансій зі списку, користувач потрапляє на детальну сторінку цієї вакансії, де йому буде надано всю необхідну інформацію – рисунок 3.8. На сторінці вакансії відображається основна інформація про позицію, зокрема назва вакансії, місцезнаходження, заробітна плата, кількість заявок та кількість відкритих місць. Також є розділ з детальним описом обов’язків, вимог до кандидата та умов праці.

Користувач може ознайомитися з вимогами до кваліфікації, які необхідні для цієї вакансії, а також з інформацією про компанію, що пропонує роботу. Зацікавившись вакансією, користувач має можливість натиснути на кнопку "Подати заявку", що автоматично перенаправить його на форму подачі заявки або зв’яжеться з роботодавцем. Такий інтуїтивний інтерфейс дозволяє користувачам швидко й ефективно знаходити необхідну інформацію і подавати заявку на відповідні вакансії.

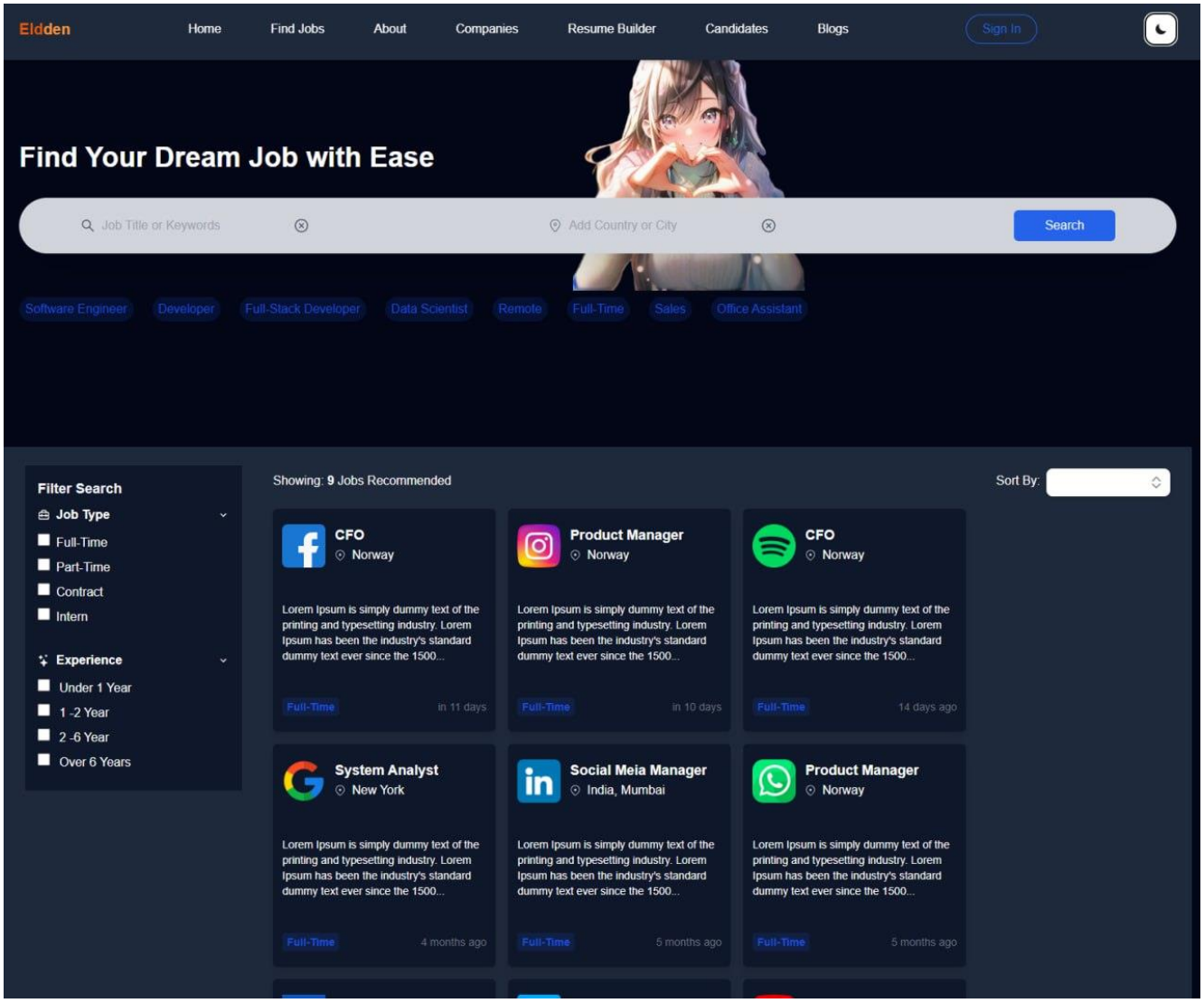

Рисунок 3.7 – Загальний вигляд вікна сторінки з рекомендованими вакансіями

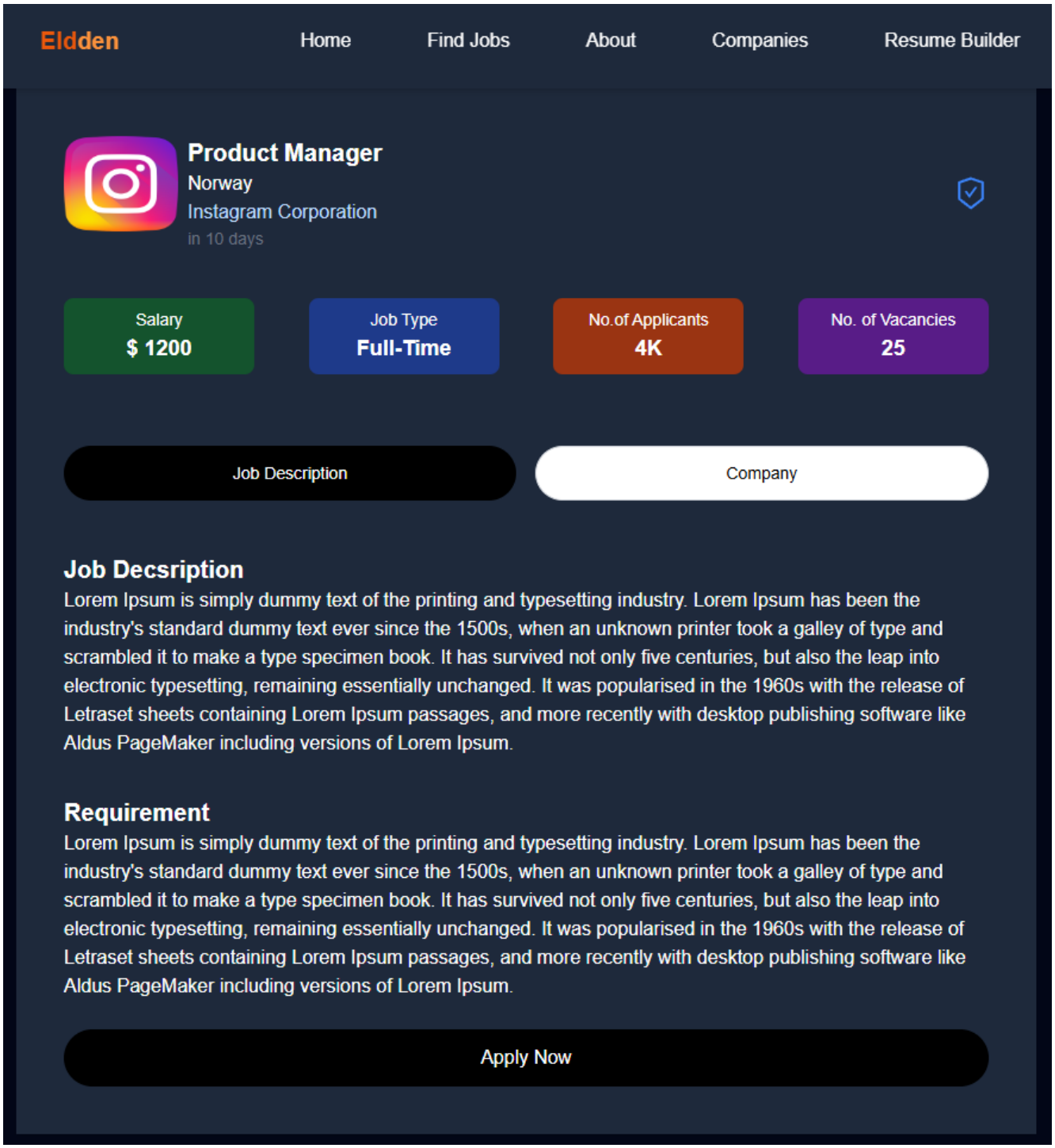

Рисунок 3.8 – Загальний вигляд вікна сторінки з детальною інформацією про вакансію

На сторінці створення резюме – рисунок 3.9, користувачі отримують доступ до інтерактивного Markdown-редактора, який дозволяє легко форматувати текст і створювати професійно оформлені документи. Markdown-редактор – це потужний

інструмент, який дозволяє не тільки швидко додавати текст, але й застосовувати до нього різні стилі, як-от жирний шрифт, курсив, заголовки різних рівнів, списки, цитати та інше. Такий підхід допомагає структурувати інформацію і зробити її більш читабельною та привабливою для потенційного роботодавця.

Редактор оснащений зручними підказками та кнопками для швидкого застосування основних елементів форматування. Наприклад, користувач може додавати секції для опису досвіду, навичок або особистих досягнень, легко створювати заголовки для кожного розділу, структурувати текст за допомогою списків чи підкреслювати важливу інформацію. Markdown-редактор також підтримує функцію попереднього перегляду, що дозволяє миттєво побачити, як буде виглядати резюме після експорту. Це особливо корисно для тих, хто хоче переконатися, що резюме має професійний та акуратний вигляд перед завантаженням.

Після того, як резюме повністю готове, користувач має можливість одним кліком завантажити його у форматі PDF. Такий формат є універсальним та ідеально підходить для пересилання та подачі на вакансії, зберігаючи всю структуру та форматування. Завдяки Markdown-редактору і підтримці експорту, користувачі можуть створити якісне резюме швидко і без потреби звертатися до сторонніх програм, зберігаючи всі важливі дані локально на своєму комп'ютері.

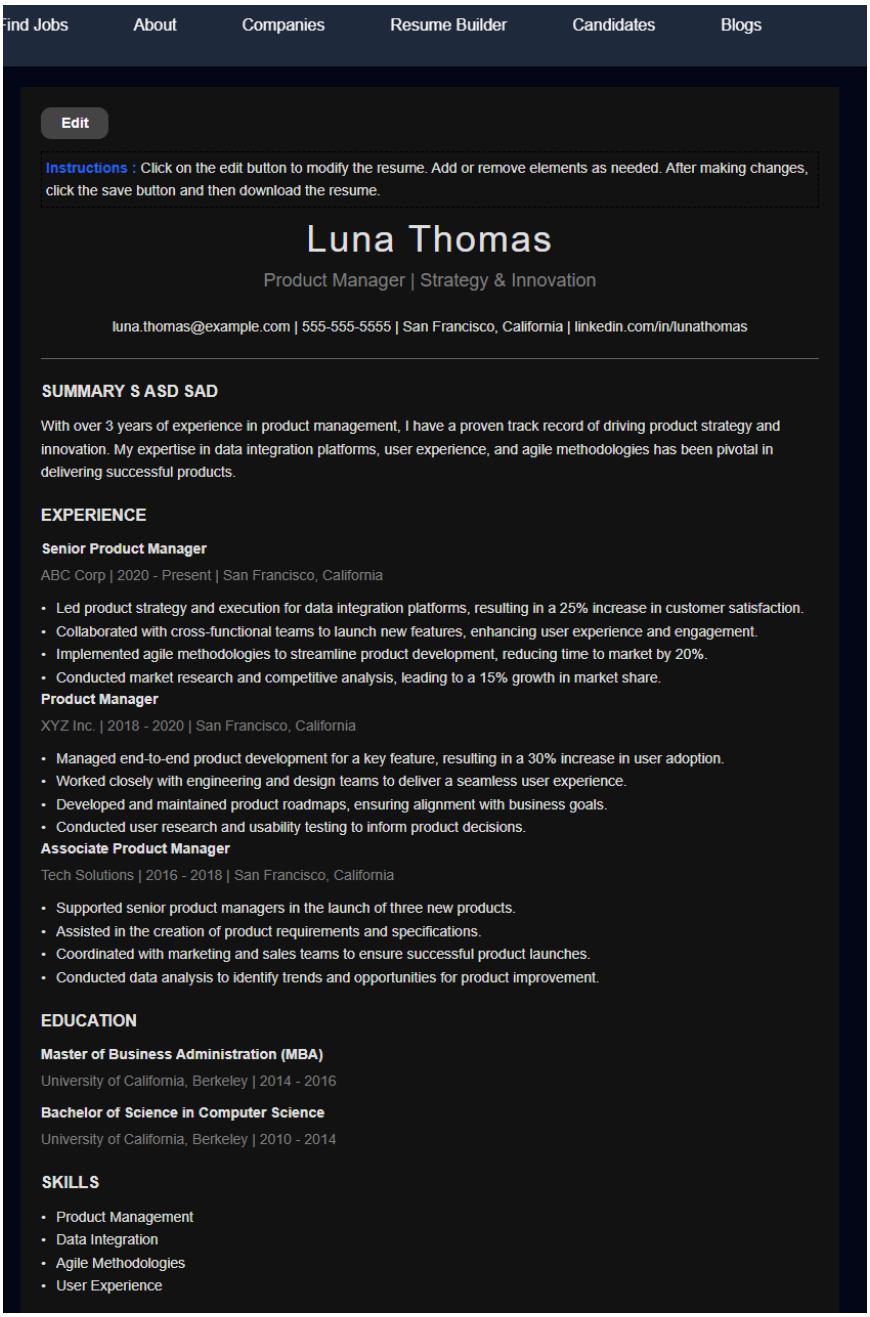

Рисунок 3.9 – Загальний вигляд вікна сторінки для створення та завантаження резюме

На адміністративній панелі сайту передбачена спеціальна сторінка для налаштування прав доступу користувачів – рисунок 3.10. Переходячи до цієї сторінки, адміністратор отримує доступ до інтуїтивного інтерфейсу, який дозволяє легко керувати списком дозволів. Тут можна переглядати наявних користувачів, їхні ролі, а також змінювати рівень доступу для кожного з них залежно від їхньої

ролі або специфічних вимог до роботи.

Крім того, якщо виникає потреба надати спеціальний доступ окремому користувачу, не змінюючи його ролі, адміністративна панель також пропонує можливість створювати індивідуальні дозволи. Це дозволяє гнучко адаптувати систему прав доступу під конкретні завдання та потреби. Усі зміни зберігаються автоматично, і адміністратор може бути впевнений, що вони негайно вступлять у силу, надаючи або обмежуючи доступ користувачам відповідно до нових налаштувань.

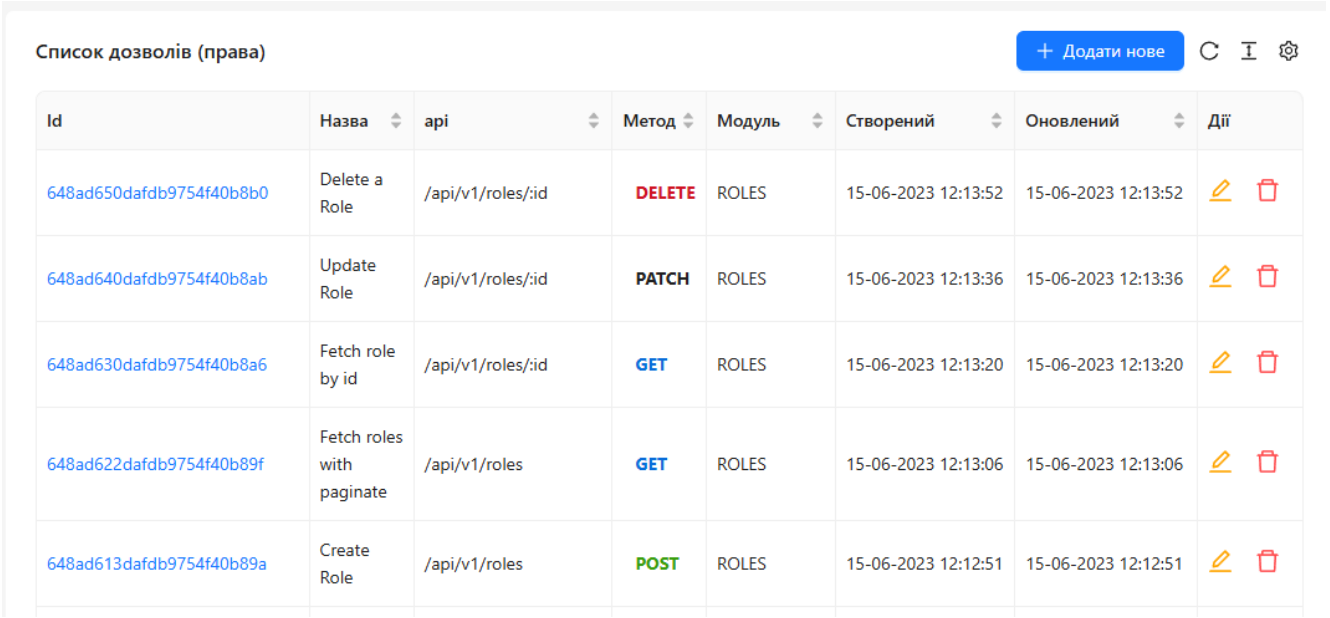

Рисунок 3.10 – Загальний вигляд вікна сторінки адміністративної панелі для управління списком прав користувачів

Розроблений програмний модуль успішно пройшов тестування та відповідає всім заданим вимогам.

В таблиці 3.1 подано результати тестування розробленого програмного продукту та аналогів.

Таблиця 3.1 – Результати тестування розробленого програмного продукту та аналогів

Критерій	Ваш продукт	Indeed	LinkedIn	Glassdoor
Використання нечіткої логіки для персоналізованих рекомендацій	+	-	-	-
Відсутність реклами	+	-	-	-
Локалізація для українського ринку	+	-	+	-
Можливість збереження та аналізу історії пошуків	+	-	+	-
Інтеграція з освітніми платформами (курси, тренінги)	-	-	+	-
Надання інформації про середню зарплату за позицією	-	+	+	+

Проаналізувавши функціональні можливості розробленого програмного продукту, можна зробити висновок, що він має суттєві переваги порівняно з аналогами, такими як Indeed, LinkedIn та Glassdoor. З таблиці видно, що розробка перевершує аналоги за низкою ключових показників. Зокрема, продукт використовує нечітку логіку для персоналізованих рекомендацій, що є унікальною функцією серед представлених систем і забезпечує точніше налаштування рекомендацій відповідно до потреб користувача. Крім того, повна відсутність реклами сприяє комфортнішому користувацькому досвіду, що вигідно вирізняє продукт на фоні конкурентів.

Особливо важливою є підтримка української локалізації, яка робить систему доступною для широкої аудиторії в Україні, чого не забезпечують більшість аналогів. Також продукт пропонує можливість збереження та аналізу історії пошуків, що покращує ефективність використання та дозволяє оптимізувати пошукові процеси. У той же час певні аспекти, такі як інтеграція з освітніми платформами або надання інформації про середню зарплату за позицією, відсутні, що може стати напрямком для майбутніх удосконалень.

Загалом, розроблений програмний продукт демонструє високий рівень інноваційності та адаптації до потреб локального ринку. Його розширений

функціонал забезпечує конкурентні переваги та сприяє підвищенню задоволеності користувачів.

3.7 Висновок до розділу 3

У третьому розділі магістерської роботи обґрунтовано вибір архітектури інформаційної технології для надання рекомендацій щодо пошуку роботи. Для реалізації цієї технології було обрано клієнт-серверну архітектуру, оскільки вона підтримує багатокористувацький доступ, забезпечує цілісність та безпеку даних, оптимізована для роботи з великими обсягами даних і дозволяє оновлювати систему без змін на стороні клієнта. Розроблено схему алгоритму роботи програмного модуля та детально описано його етапи.

Було проведено аналіз мов програмування для створення як клієнтської, так і серверної частини системи, зокрема React та Node.js для клієнта, а також Python та інші технології для серверної частини. Визначено їх особливості, переваги та недоліки в контексті реалізації пошукових алгоритмів і обробки даних. На основі цього аналізу для клієнтської частини обрано React, що дозволяє ефективно створювати динамічні інтерфейси, а для серверної частини — Node.js, що гарантує високу продуктивність при обробці великих масивів даних.

Описано процес реалізації основних компонентів інформаційної технології, включаючи аналіз даних та застосування нечіткої логіки для генерації рекомендацій. Особливу увагу приділено взаємодії серверної частини з базами даних, а також тестуванню системи на різних етапах її розробки.

В результаті тестування підтверджено, що основні функції системи пошуку роботи були успішно реалізовані. Внесено кілька важливих доповнень: підтримка багатомовної локалізації, можливість фільтрації та персоналізації рекомендацій для користувачів, а також інтеграція алгоритмів для покращення якості пошукових результатів.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Проведення комерційного та технологічного аудиту інформаційної технології пошуку роботи (аналіз даних та нечітка логіка).

Метою проведення комерційного і технологічного аудиту є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності, тобто під час виконання магістерської кваліфікаційної роботи [26].

Для проведення комерційного та технологічного аудиту залучаємо 3-х незалежних експертів, якими є провідні викладачі випускової або спорідненої кафедри.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу здійснюємо із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, а результати зводимо до таблиці 4.1.

Таблиця 4.1 – Результати оцінювання науково-технічного рівня і комерційного потенціалу інформаційної технології пошуку роботи (аналіз даних та нечітка логіка)

Критерії	Експерти		
	Експерт 1	Експерт 2	Експерт 3
	Бали, виставлені експертами		
Технічна здійсненність концепції	2	2	3
Ринкові переваги (наявність аналогів)	2	1	2
Ринкові переваги (ціна продукту)	3	2	3
Ринкові переваги (технічні властивості)	3	2	3
Ринкові переваги (експлуатаційні витрати)	2	3	2
Ринкові перспективи (розмір ринку)	1	2	3
Ринкові перспективи (конкуренція)	2	2	2

Продовження таблиці 4.1

Критерії	Експерти		
	Експерт 1	Експерт 2	Експерт 3
	Бали, виставлені експертами		
Практична здійсненність (наявність фахівців)	2	3	3
Практична здійсненність (наявність фінансів)	2	3	3
Практична здійсненність (необхідність нових матеріалів)	3	3	3
Практична здійсненність (термін реалізації)	3	2	3
Практична здійсненність (розробка документів)	3	1	3
Сума балів	28	26	33
Середньоарифметична сума балів, СБ	29		

За результатами розрахунків, наведених в таблиці 4.1 робимо висновок про те, що науково-технічний рівень та комерційний потенціал інформаційної технології пошуку роботи (аналіз даних та нечітка логіка) – середній.

4.2 Розрахунок витрат на здійснення науково-дослідної розробки інформаційної технології пошуку роботи (аналіз даних та нечітка логіка)

Витрати на оплату праці. Належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці, також будь-які види грошових і матеріальних доплат, які належать до елемента «Витрати на оплату праці».

Основна заробітна плата дослідників. Витрати на основну заробітну плату дослідників ($З_о$) розраховують відповідно до посадових окладів працівників, за формулою:

$$З_о = \sum_{i=1}^{к} \frac{М_{ni} \cdot t_i}{Т_р}, \quad (4.1)$$

де к – кількість посад дослідників, залучених до процесу дослідження;

$М_{ni}$ – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

$Т_р$ – число робочих днів в місяці; приблизно $Т_р$ = (21…23) дні;

t_i – число робочих днів роботи розробника (дослідника).

Зроблені розрахунки зводимо до таблиці 4.2.

Таблиця 4.2 – Витрати на заробітну плату дослідників

Посада	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник	43000	2048	25	51200
Розробник	32000	1524	21	32000
Консультанти	20000	952	28	26656
Всього:	109856			

Основна заробітна плата робітників. Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт розраховують за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i, \quad (4.2)$$

де $С_i$ – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника на виконання певної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду С і можна визначити за формулою:

$$C_i = \frac{М_м \cdot К_i \cdot К_c}{Т_р \cdot t_{зм}}, \quad (4.3)$$

де $М_м$ – розмір прожиткового мінімуму працездатної особи або мінімальної місячної заробітної плати (залежно від діючого законодавства), у 2024 році $М_м$=8000 грн;

$К_i$ – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду;

$К_c$ – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати;

$Т_р$ – середня кількість робочих днів в місяці, приблизно $Т_р$ = 21…23 дні; $t_{зм}$ – тривалість зміни, год.

Зроблені розрахунки зводимо до таблиці 4.3.

Таблиця 4.3 – Витрати на заробітну плату робітників

Найменування робіт	Трудомісткість, н-год.	Розряд роботи	Погодинна тарифна ставка	Тариф. коеф.	Величина, грн.
Аналіз предметної області	24	6	65,9	1,45	1582
Моделювання інформаційної технології	99	5	61,8	1,36	6118
Створення основної структури проєкту	28	5	61,8	1,36	1730

Продовження таблиці 4.3

Найменування робіт	Трудомісткість, н-год.	Розряд роботи	Погодинна тарифна ставка	Тариф. коеф.	Величина, грн.
Створення та наповнення бази даних	65	6	65,9	1,45	4284
Тестування інформаційної технології	61	5	61,8	1,36	3770
Всього					17484

Додаткова заробітна плата. Додаткова заробітна плата $З_д$ всіх розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховується як (10…12)% від суми основної заробітної плати всіх розробників та робітників, тобто:

$$З_д = 0{,}11 \cdot (З_о + З_р) = 0{,}11 \cdot (109856 + 17484) = 14007 \text{ грн.} \quad (4.4)$$

Відрахування на соціальні заходи. Нарахування на заробітну плату $Н_{зп}$ розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою:

$$Н_{зп} = \beta \cdot (З_о + З_р + З_д) = \quad (4.5)$$
$$= 0{,}22 \cdot (109856 + 17484 + 14007) = 31096 \text{ грн.}$$

де $З_о$ – основна заробітна плата розробників, грн.;

$З_р$ – основна заробітна плата робітників, грн.;

$З_д$ – додаткова заробітна плата всіх розробників та робітників, грн.;

β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % (приймаємо для 1-го класу професійності ризику 22%).

Програмне забезпечення. До балансової вартості програмного забезпечення входять витрати на його інсталяцію, тому ці витрати беруться додатково в розмірі

10…12% від вартості програмного забезпечення. Балансову вартість програмного забезпечення розраховують за формулою:

$$В_{прг} = \sum_{1}^{к} Ц_{іпрг} \cdot С_{прг.і} \cdot К_і, \quad (4.6)$$

де $Ц_{іпрг}$ – ціна придбання програмного забезпечення і-го виду, грн.;

$С_{прг.і}$ – кількість одиниць програмного забезпечення відповідного виду, шт.;

$К_і$ – коефіцієнт, що враховує інсталяцію, налагодження програмного забезпечення, $К_і$ = (1,1…1,12);

к – кількість видів програмного забезпечення.

Зроблені розрахунки зводимо до таблиці 4.4.

Таблиця 4.4 – Витрати на придбання програмного забезпечення

Найменування програмного забезпечення	Ціна за одиницю, грн.	Витрачено	Вартість програмного забезпечення, грн.
Heroku	219	1	219
JetBrains WebStorm	290	1	290
Всього, з врахуванням коефіцієнта інсталяції та налагодження			560

Амортизація обладнання. Амортизація обладнання, комп'ютерів та приміщень, які використовувались під час (чи для) виконання даного етапу роботи.

У спрощеному вигляді амортизаційні відрахування А в цілому бути розраховані за формулою:

$$А = \frac{Ц_б}{Т_в} \cdot \frac{t}{12}, \quad (4.7)$$

де $Ц_б$ – загальна балансова вартість всього обладнання, комп'ютерів, приміщень тощо, що використовувались для виконання даного етапу роботи, грн.;

t – термін використання основного фонду, місяці;

$Т_в$ – термін корисного використання основного фонду, роки.

Зроблені розрахунки зводимо до таблиці 4.5.

Таблиця 4.5 – Амортизаційні відрахування за видами основних фондів

Найменування	Балансова вартість, грн.	Строк корисного використання, років	Термін використання, місяців	Сума амортизації, грн.
Ноутбук	25000	2	1,6	1667
Монітор	4000	2	1,6	267
Стіл	3500	5	1,6	93
Крісло	4000	5	1,6	107
Мишка	1000	2	1,6	67
Клавіатура	1200	2	1,6	80
Разом	2281			

Витрати на електроенергію для науково-виробничих цілей. Витрати на силову електроенергію $В_е$. Витрати на електроенергію зображені у таблиці 4.6.

Таблиця 4.6 – Витрати на електроенергію

Найменування обладнання	Потужність, кВт	Тривалість годин роботи
Ноутбук	0,15	265
Освітлення	0,031	140

Якщо ця стаття має суттєве значення для виконання даного етапу роботи, розраховуються за формулою:

$$\text{Ве} = \sum \frac{W_i \cdot t_i \cdot \text{Ц}_\text{е} \cdot \text{К}_\text{впі}}{\text{ККД}} = \frac{0{,}15 \cdot 265 \cdot 7{,}5 \cdot 0{,}95}{0{,}98} + \frac{0{,}031 \cdot 140 \cdot 7{,}5 \cdot 0{,}95}{0{,}98} = 321 \text{ грн.}, \quad (4.8)$$

W_i – встановлена потужність обладнання, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год.;

Це – вартість 1 кВт електроенергії, грн.;

$К_{впі}$ – коефіцієнт використання потужності; ККД – коефіцієнт корисної дії обладнання.

Інші витрати. До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$\text{Ів} = \left(З_\text{о} + З_\text{р}\right) \cdot \frac{\text{Н}_\text{ів}}{100\%} = (109856 + 17484) \cdot \frac{95}{100} = 120973 \text{ грн.}, \quad (4.9)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати».

Накладні (загальновиробничі) витрати. До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$\text{Внзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%} = (109856 + 17484) \cdot \frac{110}{100} = 140074 \text{ грн.}, \quad (4.10)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

Витрати на проведення науково-дослідної роботи. Витрати на проведення науково-дослідної роботи розраховуються як сума всіх попередніх статей витрат за формулою:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + \quad (4.11)$$
$$+ I_в + В_{нзв} =$$
$$= 109856 + 17484 + 14007 + 31096 + 560 +$$
$$2281 + 321 + 120973 + 140074 = 436652 \text{ грн.}$$

Загальні витрати. Загальні витрати ЗВ на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} = \frac{436652}{0{,}9} = 485169 \text{ грн.}, \quad (4.12)$$

де η – коефіцієнт, що характеризує етап виконання науково-дослідної роботи. Оскільки, якщо науково-технічна розробка знаходиться на стадії технічного проектування, то η=0,2.

4.3 Розрахунок економічної ефективності науково-технічної розробки інформаційної технології пошуку роботи (аналіз даних та нечітка логіка) за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

В даному випадку відбувається розробка засобу, тому основу майбутнього економічного ефекту буде формувати: ΔN – збільшення кількості споживачів, яким надається відповідна інформаційна послуга в аналізовані періоди часу; N – кількість споживачів, яким надавалась відповідна інформаційна послуга у році до впровадження результатів нової науково-технічної розробки; $Ц_б$ – вартість послуги у році до впровадження інформаційної системи; $\pm\Delta Ц_о$ – зміна вартості послуги (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу [27, 28].

Можливе збільшення чистого прибутку у потенційного інвестора ΔПі для кожного із років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховується за формулою:

$$\Delta П = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N_i)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\vartheta}{100}\right), \quad (4.13)$$

де ±ΔЦ – зміна основного якісного показника від впровадження результатів науково-технічної розробки в аналізованому році. Зазвичай, таким показником може бути зміна ціни реалізації одиниці нової розробки в аналізованому році (відносно року до впровадження цієї розробки);

$\pm\Delta Ц_о$ може мати як додатне, так і від'ємне значення (від'ємне – при зниженні ціни відносно року до впровадження цієї розробки, додатне – при зростанні ціни);

N – основний кількісний показник, який визначає величину попиту на аналогічні чи подібні розробки у році до впровадження результатів нової науково-технічної розробки;

$Ц_о$ – основний якісний показник, який визначає ціну реалізації нової науково-технічної розробки в аналізованому році;

$Ц_б$ – основний якісний показник, який визначає ціну реалізації існуючої (базової) науково-технічної розробки у році до впровадження результатів;

ΔN – зміна основного кількісного показника від впровадження результатів науково-технічної розробки в аналізованому році. Зазвичай таким показником може бути зростання попиту на науково-технічну розробку в аналізованому році (відносно року до впровадження цієї розробки);

λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2024 році ставка податку на додану вартість становить 20%, а коефіцієнт $\lambda = 0{,}8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту (послуги). Рекомендується брати $\rho = 0{,}2...0{,}5$; ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2024 році $\vartheta = 18\%$.

Очікуваний термін життєвого циклу розробки 1 рік, тому:

$$\Delta П = ((6500 - 5900) \cdot 8000 - (6500 - 8000) \cdot 650) \cdot 0{,}8333 \cdot 0{,}5 \cdot \left(1 - \frac{18}{100}\right) = 1894200 \text{ грн.} \quad (4.14)$$

Далі розраховують приведену вартість збільшення всіх чистих прибутків ПП, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t} = \frac{1894200}{(1+0{,}1)^1} = 1722000 \text{ грн.}, \quad (4.15)$$

де ΔП – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн.;

Т – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки (приймаємо Т=1 рік);

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,05…0,15;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

Далі розраховують величину початкових інвестицій PV , які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} \cdot ЗВ = 3 \cdot 485169 = 1455507 \text{ грн.} \quad (4.16)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай kінв=2...5, але може бути і більшим;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

Тоді абсолютний економічний ефект $Е_{абс}$ або чистий приведений дохід для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$\text{Е}_{\text{абс}} = \text{ПП} - PV = 1722000 - 1455507 = 266493 \text{ грн.}, \quad (4.17)$$

де ПП – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, грн.;

PV – теперішня вартість початкових інвестицій, грн.

Оскільки $\text{Е}_{\text{абс}}>0$, то можемо припустити про потенційну зацікавленість інвесторів у розробці.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність Ев або показник внутрішньої норми дохідності вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Внутрішня економічна дохідність інвестицій Ев, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки, розраховується за формулою:

$$\text{Е}_{\text{в}} = \sqrt[\text{Т}_{\text{ж}}]{1 + \frac{\text{Еабс}}{PV}} - 1 = \sqrt[1]{1 + \frac{266493}{1455507}} - 1 = 0{,}8, \quad (4.18)$$

де $\text{Т}_{\text{ж}}$ – життєвий цикл розробки, роки.

Визначимо бар'єрну ставку дисконтування $\tau_{\text{мін}}$, тобто мінімальну внутрішню економічну дохідність інвестицій, нижче якої кошти у впровадження науково-технічної розробки та її комерціалізацію вкладатися не будуть.

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{\text{мін}}$ визначається за формулою:

$$\tau_{\text{мін}} = d + f = 0{,}12 + 0{,}05 = 0{,}17, \quad (4.19)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках. В 2024 році в Україні d =0,9...0,12;

f – показник, що характеризує ризикованість вкладення інвестицій; зазвичай величина f =0,05...0,5, але може бути і значно вищою.

Оскільки $E_в$=0,8>$\tau_{мін}$=0,17, то потенційний інвестор може бути зацікавлений у фінансуванні впровадження науково-технічної розробки та виведенні її на ринок, тобто в її комерціалізації.

Далі розраховуємо період окупності інвестицій $T_о$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки інформаційної технології пошуку роботи (аналіз даних та нечітка логіка):

$$T_о = \frac{1}{E_в} = \frac{1}{0,8} = 1,25 \text{ року.} \quad (4.20)$$

Оскільки $T_о$<1…3-х років, то це свідчить про комерційну привабливість науково-технічної розробки інформаційної технології пошуку роботи (аналіз даних та нечітка логіка) і може спонукати потенційного інвестора профінансувати впровадження цієї розробки та виведення її на ринок.

4.4 Висновок до розділу 4

Згідно з проведеними розрахунками, комерційний потенціал розробки під назвою "Інформаційна технологія пошуку роботи. Частина 1. Аналіз даних та нечітка логіка" оцінено як середній, що свідчить про перспективи подальшого розвитку та комерціалізації. Загальна середня оцінка становить 29 балів за результатами експертного оцінювання. Це дозволяє зробити висновок, що розробка відповідає сучасним стандартам науково-технічного рівня та має базові передумови для виходу на ринок.

Загальні витрати на реалізацію та завершення науково-дослідної роботи становлять 436652 грн, що демонструє адекватний рівень інвестицій для виконання такого проєкту.

Розрахований абсолютний економічний ефект від комерціалізації розробки є позитивним, що створює сприятливі умови для потенційних інвесторів. Це означає, що інвестори зможуть отримати додатковий чистий прибуток, впроваджуючи результати цієї науково-технічної діяльності. Важливим показником є внутрішня економічна дохідність інвестицій, яка становить 0,8 і суттєво перевищує бар'єрну ставку дисконтування на рівні 0,17. Цей факт свідчить про привабливість проєкту з точки зору економічної доцільності, оскільки інвестиції матимуть значний запас прибутковості порівняно з альтернативними варіантами вкладень.

Ще одним ключовим фактором є окупність інвестицій, яка становить трохи більше 1 року. Це виключно сприятливий показник, який підкреслює швидке повернення вкладених коштів та мінімізацію ризиків для інвесторів. Такий короткий термін окупності робить проєкт особливо цікавим для тих, хто шукає можливості швидкої віддачі від інвестицій у високотехнологічні розробки.

Таким чином, результати проведеного аналізу підтверджують значний комерційний потенціал проєкту "Інформаційна технологія пошуку роботи. Частина 1. Аналіз даних та нечітка логіка". Успішне впровадження цієї розробки на ринок може забезпечити не лише економічну вигоду для інвесторів, але й сприяти розширенню ринку інноваційних послуг у сфері інформаційних технологій. Інвестори можуть бути впевнені в надійності та перспективності вкладень, розраховуючи на швидке досягнення фінансових результатів.

ВИСНОВКИ

У процесі виконання дослідження було здійснено глибокий аналіз існуючих інформаційних технологій у сфері пошуку роботи, зосереджуючись на їхніх функціональних можливостях, архітектурі та особливостях реалізації. Виявлено ключові переваги та недоліки таких популярних платформ, як LinkedIn, Indeed і Glassdoor, що дозволило визначити напрями вдосконалення для створюваного програмного забезпечення. Розроблений додаток було детально порівняно з вказаними платформами, і в результаті аналізу підтверджено його конкурентні переваги. Серед них — використання нечіткої логіки для персоналізації рекомендацій, підтримка української локалізації та повна відсутність реклами, що значно підвищує зручність і комфорт користувачів. Усі ці фактори дозволяють вважати створену систему більш адаптивною до потреб локального ринку та універсальною для розширення на інші сегменти.

Методи аналізу даних, використані в дослідженні, включали статистичні підходи, алгоритми машинного навчання та нечіткої логіки. Зокрема, нечітка логіка обрана як основний метод для побудови рекомендаційної системи завдяки її здатності ефективно обробляти великі обсяги інформації та враховувати різноманітність критеріїв оцінки. Такий підхід забезпечує високу точність і персоналізацію результатів, що дозволяє задовольнити як потреби користувачів у пошуку роботи, так і вимоги роботодавців до підбору кадрів. Додатково було опрацьовано питання інтеграції даних із зовнішніх баз, що значно розширює функціональність системи.

Розроблений алгоритм функціонування інформаційної технології чітко регламентує кожен етап роботи системи, починаючи від збору даних і завершуючи аналізом, оцінкою та представленням результатів у зручному для користувача форматі. На основі алгоритму було створено математичну модель, яка використовує концепцію нечітких множин для оцінки ключових параметрів. Ця

модель дозволяє враховувати як об'єктивні показники (досвід, навички, рівень освіти), так і суб'єктивні уподобання роботодавців, що значно підвищує якість та інформативність рекомендацій.

Окрім технічної реалізації, велика увага приділена економічному обґрунтуванню доцільності розробки. Було проведено розрахунок витрат, до яких увійшли оплата праці команди розробників, придбання ліцензійного програмного забезпечення, витрати на амортизацію обладнання, електроенергію та інші ресурси. Загальні витрати виявилися оптимальними, що підтверджує фінансову доцільність проєкту. Крім того, оцінено потенційні вигоди від впровадження розробки, зокрема швидкість окупності та перспективність для інвесторів. Результати розрахунків засвідчили високу рентабельність і конкурентоспроможність створеної системи.

У підсумку визначено мету, об’єкт, предмет і задачі подальших досліджень. Розроблена інформаційна технологія є сучасним і перспективним рішенням, яке має значний потенціал для розвитку. Успішне впровадження цієї системи не лише спростить процес пошуку роботи для користувачів, а й забезпечить більш точний і якісний підбір кандидатів для роботодавців. Завдяки інноваційним технологіям, економічній доцільності та локалізованим функціям система стане потужним інструментом у сфері рекрутингу. Вона відкриває нові можливості для інтеграції з іншими сервісами, вдосконалення функціоналу та масштабування на міжнародний ринок.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Стасишен Д. В., Арсенюк І. Р. Обґрунтування доцільності розробки Інформаційної технології пошуку роботи. *Матеріали молодіжної науково-практичної інтернет-конференції студентів аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)» : збірник матеріалів*. [Електронний ресурс]. Вінниця: ВНТУ, 2021. С. 485 – 486. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/viewFile/21548/17999. (дата звернення 04.10.2024).

2 Indeed – сайт з пошуку роботи №1 у світі. Пошук роботи в Україні. База вакансій з тисяч джерел. Швидкий і зручний пошук, розсилка свіжих вакансій. [Електронний ресурс]. URL: https://ua.indeed.com/ (дата звернення 04.10.2024).

3. LinkedIn — соціальна мережа для пошуку і встановлення ділових контактів. [Електронний ресурс]. URL: https://www.linkedin.com/ (дата звернення 04.10.2024).

4. Glassdoor - американський веб-сайт, на якому нинішні та колишні працівники анонімно відгукуються про компанії [Електронний ресурс]. URL: https://www.glassdoor.com/Community/index.htm/ (дата звернення 04.10.2024).

5. Zadeh L. A. Fuzzy Sets. Information and Control, 1965. V. 8(3). P. 338−353.

6. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. Pearson, 2021. 1152 p.

7. Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, 2009. 745 p.

8. Haykin S. Neural Networks: A Comprehensive Foundation. Prentice Hall, 1998. 1104 p.

9. Wooldridge J. M. Econometric Analysis of Cross Section and Panel Data. MIT Press, 2010. 1096 p.

10. Granovetter M. The Strength of Weak Ties. American Journal of Sociology, 1973. V. 78(6). P. 1360−1380.

11. Jang J. S. R., Sun C. T., Mizutani E. Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence [Електронний ресурс]. URL: https://www.sciencedirect.com/book/9780132610667/neuro-fuzzy-and-soft-computing (дата звернення 06.10.2024).

12. Dubois D., Prade H. Fuzzy Sets and Systems: Theory and Applications [Електронний ресурс]. URL: https://www.springer.com/gp/book/9780122227505 (дата звернення 06.10.2024).

13. Bishop C. M. Pattern Recognition and Machine Learning [Електронний ресурс]. URL: https://www.springer.com/gp/book/9780387310732 (дата звернення 06.10.2024).

14. Андрієнко В. М., Семенов А. С. Методика статистичного аналізу економічних часових рядів. Науковий вісник Ужгородського національного університету. Серія: Міжнародні економічні відносини та світове господарство, 2018. Вип. 21(1). С. 5–13.

15. Гур'янова Л. С., Клебанова Т. С., Сергієнко О. А., Прокопович С. В. Економетрика: навчальний посібник для студентів напряму підготовки «Економічна кібернетика» всіх форм навчання. Харків: ХНЕУ ім. С. Кузнеця, 2015. 384 с.

16. Буйницька О. П.Інформаційні технології та технічні засоби навчання. Навч. посіб. – К.: Центр учбової літератури, 2012. – 240 с.

17. Jang J. S. R., Sun C. T., Mizutani E. Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence. Prentice Hall, 1997. 614 p.

18. Яковенко А. В. Основи програмування. Python. Частина 1: навч. посіб. Київ: КПІ ім. І. Сікорського, 2018. 195 с.

19. Python Загальна документація [Електронний ресурс]. URL: https://www.python.org/doc/ (дата звернення 05.10.2024).

20. Java Загальна документація [Електронний ресурс]. URL: https://docs.oracle.com/en/java/javase/11/docs/api/index.html (дата звернення 05.10.2024).

21. C++ Загальна документація [Електронний ресурс]. URL: https://en.cppreference.com/w/cpp (дата звернення 05.10.2024).

22. JavaScript Загальна документація [Електронний ресурс]. URL: https://developer.mozilla.org/uk/docs/Web/JavaScript (дата звернення 30.09.2023).

23. PHP Загальна документація [Електронний ресурс]. URL: https://www.php.net/manual/ru/ (дата звернення 05.10.2024).

24. React Документація [Електронний ресурс]. URL: https://reactjs.org/docs/getting-started.html (дата звернення 05.10.2024).

25. Методичні вказівки до виконання магістерських кваліфікаційних робіт для студентів спеціальності 122 «Комп'ютерні науки» [Електронний ресурс] / уклад.: А. А. Яровий, О. К. Колесницький. Вінниця : ВНТУ, 2023. (PDF, 58 с.)

26. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад.: В. О. Козловський, О. Й. Лесько, В. В. Кавецький. Вінниця : ВНТУ, 2021. 42 с.

27. Як розраховується прибуток [Електронний ресурс]. URL: https://support.joinposter.com/uk/articles/7067856 (дата звернення: 16.10.2024).

28. Собівартість послуг: особливості формування [Електронний ресурс]. URL: https://interbuh.com.ua/ua/documents/ib/10166/138487 (дата звернення: 16.10.2024).

Додаток А (обов'язковий)

Протокол перевірки кваліфікаційної роботи на наявність текстових запозичень

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Інформаційна технологія пошуку роботи. Частина 1. Аналіз даних та нечітка логіка

Тип роботи: магістерська кваліфікаційна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА
(кафедра, факультет)

Показники звіту подібності Turnitin

Оригінальність 93,00% Схожість 7,00%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Ознайомлені з повним звітом подібності, який був згенерований системою Turnitin щодо роботи.

Автор роботи Стасишен Д.В.

Керівник роботи Арсенюк І.Р.

Опис прийнятого рішення

Магістерську кваліфікаційну роботу допущено до захисту

Особа, відповідальна за перевірку Озеранський В.С.

Додаток Б (обов'язковий)

Лістинг програми

```
import ModalCompany from "@/components/admin/company/modal.company";
import DataTable from "@/components/client/data-table";
import { useAppDispatch, useAppSelector } from "@/redux/hooks";
import { fetchCompany } from "@/redux/slice/companySlide";
import { ICompany } from "@/types/backend";
import { DeleteOutlined, EditOutlined, PlusOutlined } from "@ant-design/icons";
import { ActionType, ProColumns } from '@ant-design/pro-components';
import { Button, Popconfirm, Space, message, notification } from "antd";
import { useState, useRef } from 'react';
import dayjs from 'dayjs';
import { callDeleteCompany } from "@/config/api";
import queryString from 'query-string';
import Access from "@/components/share/access";
import { ALL_PERMISSIONS } from "@/config/permissions";
const CompanyPage = () => {
    const [openModal, setOpenModal] = useState<boolean>(false);
    const [dataInit, setDataInit] = useState<ICompany | null>(null);
    const tableRef = useRef<ActionType>();
    const isFetching = useAppSelector(state => state.company.isFetching);
    const meta = useAppSelector(state => state.company.meta);
    const companies = useAppSelector(state => state.company.result);
    const dispatch = useAppDispatch();
    const handleDeleteCompany = async (_id: string | undefined) => {
        if (_id) {
            const res = await callDeleteCompany(_id);
            if (res && res.data) {
                message.success('Видалити компанію успішно');
                reloadTable();
            } else {
                notification.error({
                    message: 'Сталася помилка',
                    description: res.message
                });
            }
        }
    }

    const reloadTable = () => {
        tableRef?.current?.reload();
```

```
}
const columns: ProColumns<ICompany>[] = [
  {
    title: 'STT',
    key: 'index',
    width: 50,
    align: "center",
    render: (text, record, index) => {
      return (
        <>
          {(index + 1) + (meta.current - 1) * (meta.pageSize)}
        </>)
    },
    hideInSearch: true,
  },
  {
    title: 'Id',
    dataIndex: '_id',
    width: 250,
    render: (text, record, index, action) => {
      return (
        <span>
          {record._id}
        </span>
      )
    },
    hideInSearch: true,
  },
  {
    title: 'Назва',
    dataIndex: 'name',
    sorter: true,
  },
  {
    title: 'Адреса',
    dataIndex: 'address',
    sorter: true,
  },

  {
    title: 'Створений',
```

```
    dataIndex: 'createdAt',
    width: 200,
    sorter: true,
    render: (text, record, index, action) => {
      return (
        <>{dayjs(record.createdAt).format('DD-MM-YYYY HH:mm:ss')}</>
      )
    },
    hideInSearch: true,
  },
  {
    title: 'Оновлений',
    dataIndex: 'updatedAt',
    width: 200,
    sorter: true,
    render: (text, record, index, action) => {
      return (
        <>{dayjs(record.updatedAt).format('DD-MM-YYYY HH:mm:ss')}</>
      )
    },
    hideInSearch: true,
  },
  {
    title: 'Дії',
    hideInSearch: true,
    width: 50,
    render: (_value, entity, _index, _action) => (
      <Space>
        <Access
          permission={ALL_PERMISSIONS.COMPANIES.UPDATE}
          hideChildren
        >

          <EditOutlined
            style={{
              fontSize: 20,
              color: '#ffa500',
            }}
            type=""
            onClick={() => {
              setOpenModal(true);
```

```
                    setDataInit(entity);
                  }}
                />
              </Access>
              <Access
                permission={ALL_PERMISSIONS.COMPANIES.DELETE}
                hideChildren
              >
                <Popconfirm
                  placement="leftTop"
                  title={"Підтвердити видалення компанії"}
                  description={"Ви впевнені, що хочете видалити цю компанію?"}
                  onConfirm={() => handleDeleteCompany(entity._id)}
                  okText="Підтверджувати"
                  cancelText="Скасувати"
                >
                  <span style={{ cursor: "pointer", margin: "0 10px" }}>
                    <DeleteOutlined
                      style={{
                        fontSize: 20,
                        color: '#ff4d4f',
                      }}
                    />
                  </span>
                </Popconfirm>
              </Access>
            </Space>
        ),

    },
];
const buildQuery = (params: any, sort: any, filter: any) => {
    const clone = { ...params };
    if (clone.name) clone.name = `/${clone.name}/i`;
    if (clone.address) clone.address = `/${clone.address}/i`;

    let temp = queryString.stringify(clone);

    let sortBy = "";
    if (sort && sort.name) {
        sortBy = sort.name === 'ascend' ? "sort=name" : "sort=-name";
```

```
    }
    if (sort && sort.address) {
        sortBy = sort.address === 'ascend' ? "sort=address" : "sort=-address";
    }
    if (sort && sort.createdAt) {
        sortBy = sort.createdAt === 'ascend' ? "sort=createdAt" : "sort=-createdAt";
    }
    if (sort && sort.updatedAt) {
        sortBy = sort.updatedAt === 'ascend' ? "sort=updatedAt" : "sort=-updatedAt";
    }

    //mặc định sort theo updatedAt
    if (Object.keys(sortBy).length === 0) {
        temp = `${temp}&sort=-updatedAt`;
    } else {
        temp = `${temp}&${sortBy}`;
    }

    return temp;
}

return (
    <div>
        <Access
            permission={ALL_PERMISSIONS.COMPANIES.GET_PAGINATE}
        >
            <DataTable<ICompany>
                actionRef={tableRef}
                headerTitle="Список компаній"
                rowKey="_id"
                loading={isFetching}
                columns={columns}
                dataSource={companies}
                request={async (params, sort, filter): Promise<any> => {
                    const query = buildQuery(params, sort, filter);
                    dispatch(fetchCompany({ query }))
                }}
                scroll={{ x: true }}
                pagination={
                    {
                        current: meta.current,
```

```
                pageSize: meta.pageSize,
                showSizeChanger: true,
                total: meta.total,
                showTotal: (total, range) => { return (<div> {range[0]}-{range[1]} trên {total} rows</div>) }
              }
            }
            rowSelection={false}
            toolBarRender={(_action, _rows): any => {
              return (
                <Access
                  permission={ALL_PERMISSIONS.COMPANIES.CREATE}
                  hideChildren
                >
                  <Button
                    icon={<PlusOutlined />}
                    type="primary"
                    onClick={() => setOpenModal(true)}
                  >
                    Додати нове
                  </Button>
                </Access>
              );
            }}
          />
        </Access>
        <ModalCompany
          openModal={openModal}
          setOpenModal={setOpenModal}
          reloadTable={reloadTable}
          dataInit={dataInit}
          setDataInit={setDataInit}
        />
      </div>
    )
}

export default CompanyPage;
import { Card, Col, Row, Statistic } from "antd";
import CountUp from 'react-countup';

const DashboardPage = () => {
```

```
  const formatter = (value: number | string) => {
    return (
      <CountUp end={Number(value)} separator="," />
    );
  };

  return (
    <Row gutter={[20, 20]}>
      <Col span={24} md={8}>
        <Card title="Назва картки" bordered={false} >
          <Statistic
            title="Активні користувачі"
            value={112893}
            formatter={formatter}
          />

        </Card>
      </Col>
      <Col span={24} md={8}>
        <Card title="Назва картки" bordered={false} >
          <Statistic
            title="Активні користувачі"
            value={112893}
            formatter={formatter}
          />
        </Card>
      </Col>
      <Col span={24} md={8}>
        <Card title="Назва картки" bordered={false} >
          <Statistic
            title="Активні користувачі"
            value={112893}
            formatter={formatter}
          />
        </Card>
      </Col>

    </Row>
  )
}
```

```
export default DashboardPage;
import DataTable from "@/components/client/data-table";
import { useAppDispatch, useAppSelector } from "@/redux/hooks";
import { IJob } from "@/types/backend";
import { DeleteOutlined, EditOutlined, PlusOutlined } from "@ant-design/icons";
import { ActionType, ProColumns, ProFormSelect } from '@ant-design/pro-components';
import { Button, Popconfirm, Select, Space, Tag, message, notification } from "antd";
import { useState, useRef } from 'react';
import dayjs from 'dayjs';
import { callDeleteJob } from "@/config/api";
import queryString from 'query-string';
import { useNavigate } from "react-router-dom";
import { fetchJob } from "@/redux/slice/jobSlide";
import Access from "@/components/share/access";
import { ALL_PERMISSIONS } from "@/config/permissions";
const JobPage = () => {
    const tableRef = useRef<ActionType>();
    const isFetching = useAppSelector(state => state.job.isFetching);
    const meta = useAppSelector(state => state.job.meta);
    const jobs = useAppSelector(state => state.job.result);
    const dispatch = useAppDispatch();
    const navigate = useNavigate();

    const handleDeleteJob = async (_id: string | undefined) => {
        if (_id) {
            const res = await callDeleteJob(_id);
            if (res && res.data) {
                message.success('Видалити роботу успішно');
                reloadTable();
            } else {
                notification.error({
                    message: 'Сталася помилка',
                    description: res.message
                });
            }
        }
    }

    const reloadTable = () => {
        tableRef?.current?.reload();
    }
```

```
const columns: ProColumns<IJob>[] = [
  {
    title: 'STT',
    key: 'index',
    width: 50,
    align: "center",
    render: (text, record, index) => {
      return (
        <>
          {(index + 1) + (meta.current - 1) * (meta.pageSize)}
        </>)
    },
    hideInSearch: true,
  },
  {
    title: 'Назва посади',
    dataIndex: 'name',
    sorter: true,
  },
  {
    title: 'Зарплата',
    dataIndex: 'salary',
    sorter: true,
    render(dom, entity, index, action, schema) {
      const str = "" + entity.salary;
      return <>{str?.replace(/\B(?=(\d{3})+(?!\d))/g, ',')} đ</>
    },
  },
  {
    title: 'Рівень',
    dataIndex: 'level',
    renderFormItem: (item, props, form) => (
      <ProFormSelect
        showSearch
        mode="multiple"
        allowClear
        valueEnum={{
          INTERN: 'INTERN',
          FRESHER: 'FRESHER',
          JUNIOR: 'JUNIOR',
```

```
            MIDDLE: 'MIDDLE',
            SENIOR: 'SENIOR',
          }}
          placeholder="Виберіть рівень"
        />
    ),
},
{
    title: 'Статус',
    dataIndex: 'isActive',
    render(dom, entity, index, action, schema) {
        return <>
            <Tag color={entity.isActive ? "lime" : "red"} >
                {entity.isActive ? "АКТИВНО" : "НЕАКТИВНО"}
            </Tag>
        </>
    },
    hideInSearch: true,
},

{
    title: 'Створений',
    dataIndex: 'createdAt',
    width: 200,
    sorter: true,
    render: (text, record, index, action) => {
        return (
            <>{dayjs(record.createdAt).format('DD-MM-YYYY HH:mm:ss')}</>
        )
    },
    hideInSearch: true,
},
{
    title: 'Оновлений',
    dataIndex: 'updatedAt',
    width: 200,
    sorter: true,
    render: (text, record, index, action) => {
        return (
            <>{dayjs(record.updatedAt).format('DD-MM-YYYY HH:mm:ss')}</>
        )
```

```
    },
    hideInSearch: true,
},
{

    title: 'Дії',
    hideInSearch: true,
    width: 50,
    render: (_value, entity, _index, _action) => (
        <Space>
            <Access
                permission={ALL_PERMISSIONS.JOBS.UPDATE}
                hideChildren
            >
                <EditOutlined
                    style={{
                        fontSize: 20,
                        color: '#ffa500',
                    }}
                    type=""
                    onClick={() => {
                        navigate(`/admin/job/upsert?id=${entity._id}`)
                    }}
                />
            </Access>
            <Access
                permission={ALL_PERMISSIONS.JOBS.DELETE}
                hideChildren
            >
                <Popconfirm
                    placement="leftTop"
                    title={"Підтвердьте роботу видалення"}
                    description={"Ви впевнені, що хочете видалити цю роботу?"}
                    onConfirm={() => handleDeleteJob(entity._id)}
                    okText="Підтверджувати"
                    cancelText="Скасувати"
                >
                    <span style={{ cursor: "pointer", margin: "0 10px" }}>
                        <DeleteOutlined
                            style={{
                                fontSize: 20,
```

```
                            color: '#ff4d4f',
                        }}
                    />
                </span>
            </Popconfirm>
        </Access>
    </Space>
  ),

 },
];

const buildQuery = (params: any, sort: any, filter: any) => {
  const clone = { ...params };
  if (clone.name) clone.name = `/${clone.name}/i`;
  if (clone.salary) clone.salary = `/${clone.salary}/i`;
  if (clone?.level?.length) {
    clone.level = clone.level.join(",");
  }

  let temp = queryString.stringify(clone);

  let sortBy = "";
  if (sort && sort.name) {
    sortBy = sort.name === 'ascend' ? "sort=name" : "sort=-name";
  }
  if (sort && sort.salary) {
    sortBy = sort.salary === 'ascend' ? "sort=salary" : "sort=-salary";
  }
  if (sort && sort.createdAt) {
    sortBy = sort.createdAt === 'ascend' ? "sort=createdAt" : "sort=-createdAt";
  }
  if (sort && sort.updatedAt) {
    sortBy = sort.updatedAt === 'ascend' ? "sort=updatedAt" : "sort=-updatedAt";
  }

  //mặc định sort theo updatedAt
  if (Object.keys(sortBy).length === 0) {
    temp = `${temp}&sort=-updatedAt`;
  } else {
    temp = `${temp}&${sortBy}`;
```

```
    }

    return temp;
}

return (
    <div>
        <Access
            permission={ALL_PERMISSIONS.JOBS.GET_PAGINATE}
        >
            <DataTable<IJob>
                actionRef={tableRef}
                headerTitle="Список роботи"
                rowKey="_id"
                loading={isFetching}
                columns={columns}
                dataSource={jobs}
                request={async (params, sort, filter): Promise<any> => {
                    const query = buildQuery(params, sort, filter);
                    dispatch(fetchJob({ query }))
                }}
                scroll={{ x: true }}
                pagination={
                    {
                        current: meta.current,
                        pageSize: meta.pageSize,
                        showSizeChanger: true,
                        total: meta.total,
                        showTotal: (total, range) => { return (<div> {range[0]}-{range[1]} trên {total} rows</div>) }
                    }
                }
                rowSelection={false}
                toolBarRender={(_action, _rows): any => {
                    return (
                        <Button
                            icon={<PlusOutlined />}
                            type="primary"
                            onClick={() => navigate('upsert')}
                        >
                            Додати нове
                        </Button>
```

```
                    );
                }}
            />
        </Access>
    </div>
  )
}

export default JobPage;
import DataTable from "@/components/client/data-table";
import { useAppDispatch, useAppSelector } from "@/redux/hooks";
import { IPermission } from "@/types/backend";
import { DeleteOutlined, EditOutlined, PlusOutlined } from "@ant-design/icons";
import { ActionType, ProColumns } from '@ant-design/pro-components';
import { Button, Popconfirm, Space, message, notification } from "antd";
import { useState, useRef } from 'react';
import dayjs from 'dayjs';
import { callDeletePermission } from "@/config/api";
import queryString from 'query-string';
import { fetchPermission } from "@/redux/slice/permissionSlide";
import ViewDetailPermission from "@/components/admin/permission/view.permission";
import ModalPermission from "@/components/admin/permission/modal.permission";
import { colorMethod } from "@/config/utils";
import Access from "@/components/share/access";
import { ALL_PERMISSIONS } from "@/config/permissions";

const PermissionPage = () => {
  const [openModal, setOpenModal] = useState<boolean>(false);
  const [dataInit, setDataInit] = useState<IPermission | null>(null);
  const [openViewDetail, setOpenViewDetail] = useState<boolean>(false);

  const tableRef = useRef<ActionType>();

  const isFetching = useAppSelector(state => state.permission.isFetching);
  const meta = useAppSelector(state => state.permission.meta);
  const permissions = useAppSelector(state => state.permission.result);
  const dispatch = useAppDispatch();

  const handleDeletePermission = async (_id: string | undefined) => {
    if (_id) {
      const res = await callDeletePermission(_id);
```

```
        if (res && res.data) {
            message.success('Видалити успіх дозволу');
            reloadTable();
        } else {
            notification.error({
                message: 'Сталася помилка',
                description: res.message
            });
        }
    }
}

const reloadTable = () => {
    tableRef?.current?.reload();
}

const columns: ProColumns<IPermission>[] = [
    {
        title: 'Id',
        dataIndex: '_id',
        width: 250,
        render: (text, record, index, action) => {
            return (
                <a href="#" onClick={() => {
                    setOpenViewDetail(true);
                    setDataInit(record);
                }}>
                    {record._id}
                </a>
            )
        },
        hideInSearch: true,
    },
    {
        title: 'Назва',
        dataIndex: 'name',
        sorter: true,
    },
    {
        title: 'api',
        dataIndex: 'apiPath',
```

```
        sorter: true,
      },
      {
        title: 'Метод',
        dataIndex: 'method',
        sorter: true,
        render(dom, entity, index, action, schema) {
          return (
            <p style={{ paddingLeft: 10, fontWeight: 'bold', marginBottom: 0, color: colorMethod(entity?.method as
string) }}>{entity?.method || ''}</p>
          )
        },
      },
      {
        title: 'Модуль',
        dataIndex: 'module',
        sorter: true,
      },
      {
        title: 'Створений',
        dataIndex: 'createdAt',
        width: 200,
        sorter: true,
        render: (text, record, index, action) => {
          return (
            <>{dayjs(record.createdAt).format('DD-MM-YYYY HH:mm:ss')}</>
          )
        },
        hideInSearch: true,
      },
      {
        title: 'Оновлений',
        dataIndex: 'updatedAt',
        width: 200,
        sorter: true,
        render: (text, record, index, action) => {
          return (
            <>{dayjs(record.updatedAt).format('DD-MM-YYYY HH:mm:ss')}</>
          )
        },
        hideInSearch: true,
```

```
},
{

  title: 'Дії',
  hideInSearch: true,
  width: 50,
  render: (_value, entity, _index, _action) => (
    <Space>
      <Access
        permission={ALL_PERMISSIONS.PERMISSIONS.UPDATE}
        hideChildren
      >
        <EditOutlined
          style={{
            fontSize: 20,
            color: '#ffa500',
          }}
          type=""
          onClick={() => {
            setOpenModal(true);
            setDataInit(entity);
          }}
        />
      </Access>
      <Access
        permission={ALL_PERMISSIONS.PERMISSIONS.DELETE}
        hideChildren
      >
        <Popconfirm
          placement="leftTop"
          title={"Підтвердити видалення дозволу"}
          description={"Ви впевнені, що хочете видалити цей дозвіл?"}
          onConfirm={() => handleDeletePermission(entity._id)}
          okText="Підтверджувати"
          cancelText="Скасувати"
        >
          <span style={{ cursor: "pointer", margin: "0 10px" }}>
            <DeleteOutlined
              style={{
                fontSize: 20,
                color: '#ff4d4f',
```

```
                        }}
                    />
                </span>
            </Popconfirm>
        </Access>
    </Space>
  ),

 },
];

const buildQuery = (params: any, sort: any, filter: any) => {
  const clone = { ...params };
  if (clone.name) clone.name = `/${clone.name}/i`;
  if (clone.apiPath) clone.apiPath = `/${clone.apiPath}/i`;
  if (clone.method) clone.method = `/${clone.method}/i`;
  if (clone.module) clone.module = `/${clone.module}/i`;


  let temp = queryString.stringify(clone);

  let sortBy = "";
  if (sort && sort.name) {
    sortBy = sort.name === 'ascend' ? "sort=name" : "sort=-name";
  }
  if (sort && sort.apiPath) {
    sortBy = sort.apiPath === 'ascend' ? "sort=apiPath" : "sort=-apiPath";
  }
  if (sort && sort.method) {
    sortBy = sort.method === 'ascend' ? "sort=method" : "sort=-method";
  }
  if (sort && sort.module) {
    sortBy = sort.module === 'ascend' ? "sort=module" : "sort=-module";
  }
  if (sort && sort.createdAt) {
    sortBy = sort.createdAt === 'ascend' ? "sort=createdAt" : "sort=-createdAt";
  }
  if (sort && sort.updatedAt) {
    sortBy = sort.updatedAt === 'ascend' ? "sort=updatedAt" : "sort=-updatedAt";
  }
```

```
    //mặc định sort theo updatedAt
    if (Object.keys(sortBy).length === 0) {
      temp = `${temp}&sort=-updatedAt`;
    } else {
      temp = `${temp}&${sortBy}`;
    }

    return temp;
}

return (
    <div>
      <Access
        permission={ALL_PERMISSIONS.PERMISSIONS.GET_PAGINATE}
      >
        <DataTable<IPermission>
          actionRef={tableRef}
          headerTitle="Список дозволів (права)"
          rowKey="_id"
          loading={isFetching}
          columns={columns}
          dataSource={permissions}
          request={async (params, sort, filter): Promise<any> => {
            const query = buildQuery(params, sort, filter);
            dispatch(fetchPermission({ query }))
          }}
          scroll={{ x: true }}
          pagination={
            {
              current: meta.current,
              pageSize: meta.pageSize,
              showSizeChanger: true,
              total: meta.total,
              showTotal: (total, range) => { return (<div> {range[0]}-{range[1]} trên {total} rows</div>) }
            }
          }
          rowSelection={false}
          toolBarRender={(_action, _rows): any => {
            return (
              <Button
                icon={<PlusOutlined />}
```

```
                        type="primary"
                        onClick={() => setOpenModal(true)}
                    >
                        Додати нове
                    </Button>
                );
            }}
        />
    </Access>
    <ModalPermission
        openModal={openModal}
        setOpenModal={setOpenModal}
        reloadTable={reloadTable}
        dataInit={dataInit}
        setDataInit={setDataInit}
    />

    <ViewDetailPermission
        onClose={setOpenViewDetail}
        open={openViewDetail}
        dataInit={dataInit}
        setDataInit={setDataInit}
    />
  </div>
 )
}

export default PermissionPage;
import DataTable from "@/components/client/data-table";
import { useAppDispatch, useAppSelector } from "@/redux/hooks";
import { IRole } from "@/types/backend";
import { DeleteOutlined, EditOutlined, PlusOutlined } from "@ant-design/icons";
import { ActionType, ProColumns } from '@ant-design/pro-components';
import { Button, Popconfirm, Space, Tag, message, notification } from "antd";
import { useState, useRef } from 'react';
import dayjs from 'dayjs';
import { callDeleteRole } from "@/config/api";
import queryString from 'query-string';
import { fetchRole, fetchRoleById } from "@/redux/slice/roleSlide";
import ModalRole from "@/components/admin/role/modal.role";
import { ALL_PERMISSIONS } from "@/config/permissions";
```

```
import Access from "@/components/share/access";

const RolePage = () => {
    const [openModal, setOpenModal] = useState<boolean>(false);
    const tableRef = useRef<ActionType>();
    const isFetching = useAppSelector(state => state.role.isFetching);
    const meta = useAppSelector(state => state.role.meta);
    const roles = useAppSelector(state => state.role.result);
    const dispatch = useAppDispatch();
    const handleDeleteRole = async (_id: string | undefined) => {
        if (_id) {
            const res = await callDeleteRole(_id);
            if (res && res.data) {
                message.success('Видалити роль успішно');
                reloadTable();
            } else {
                notification.error({
                    message: 'Сталася помилка',
                    description: res.message
                });
            }
        }
    }
    const reloadTable = () => {
        tableRef?.current?.reload();
    }
    const columns: ProColumns<IRole>[] = [
        {
            title: 'Id',
            dataIndex: '_id',
            width: 250,
            render: (text, record, index, action) => {
                return (
                    <span>
                        {record._id}
                    </span>
                )
            },
            hideInSearch: true,
        },
        {
```

```
    title: 'Назва',
    dataIndex: 'name',
    sorter: true,
},
{
    title: 'Статус',
    dataIndex: 'isActive',
    render(dom, entity, index, action, schema) {
        return <>
            <Tag color={entity.isActive ? "lime" : "red"} >
                {entity.isActive ? "АКТИВНО" : "НЕАКТИВНО"}
            </Tag>
        </>
    },
    hideInSearch: true,
},
{
    title: 'Створений',
    dataIndex: 'createdAt',
    width: 200,
    sorter: true,
    render: (text, record, index, action) => {
        return (
            <>{dayjs(record.createdAt).format('DD-MM-YYYY HH:mm:ss')}</>
        )
    },
    hideInSearch: true,
},
{
    title: 'Оновлений',
    dataIndex: 'updatedAt',
    width: 200,
    sorter: true,
    render: (text, record, index, action) => {
        return (
            <>{dayjs(record.updatedAt).format('DD-MM-YYYY HH:mm:ss')}</>
        )
    },
    hideInSearch: true,
},
{
```

```
title: 'Дії',
hideInSearch: true,
width: 50,
render: (_value, entity, _index, _action) => (
  <Space>
    <Access
      permission={ALL_PERMISSIONS.ROLES.UPDATE}
      hideChildren
    >
      <EditOutlined
        style={{
          fontSize: 20,
          color: '#ffa500',
        }}
        type=""
        onClick={() => {
          dispatch(fetchRoleById((entity._id) as string))
          setOpenModal(true);
        }}
      />
    </Access>
    <Access
      permission={ALL_PERMISSIONS.ROLES.DELETE}
      hideChildren
    >
      <Popconfirm
        placement="leftTop"
        title={"Підтвердити видалення ролі"}
        description={"Ви впевнені, що хочете видалити цей Галм?"}
        onConfirm={() => handleDeleteRole(entity._id)}
        okText="Підтверджувати"
        cancelText="Скасувати"
      >
        <span style={{ cursor: "pointer", margin: "0 10px" }}>
          <DeleteOutlined
            style={{
              fontSize: 20,
              color: '#ff4d4f',
            }}
          />
        </span>
```

```
                </Popconfirm>
              </Access>
            </Space>
          ),

      },
];
const buildQuery = (params: any, sort: any, filter: any) => {
    const clone = { ...params };
    if (clone.name) clone.name = `/${clone.name}/i`;

    let temp = queryString.stringify(clone);

    let sortBy = "";
    if (sort && sort.name) {
        sortBy = sort.name === 'ascend' ? "sort=name" : "sort=-name";
    }
    if (sort && sort.createdAt) {
        sortBy = sort.createdAt === 'ascend' ? "sort=createdAt" : "sort=-createdAt";
    }
    if (sort && sort.updatedAt) {
        sortBy = sort.updatedAt === 'ascend' ? "sort=updatedAt" : "sort=-updatedAt";
    }
    if (Object.keys(sortBy).length === 0) {
        temp = `${temp}&sort=-updatedAt`;
    } else {
        temp = `${temp}&${sortBy}`;
    }

    return temp;
}
return (
    <div>
        <Access
            permission={ALL_PERMISSIONS.ROLES.GET_PAGINATE}
        >
            <DataTable<IRole>
                actionRef={tableRef}
                headerTitle="Список ролей (роль)"
                rowKey="_id"
                loading={isFetching}
```

```
                columns={columns}
                dataSource={roles}
                request={async (params, sort, filter): Promise<any> => {
                  const query = buildQuery(params, sort, filter);
                  dispatch(fetchRole({ query }))
                }}
                scroll={{ x: true }}
                pagination={
                  {
                    current: meta.current,
                    pageSize: meta.pageSize,
                    showSizeChanger: true,
                    total: meta.total,
                    showTotal: (total, range) => { return (<div> {range[0]}-{range[1]} trên {total} rows</div>) }
                  }
                }
                rowSelection={false}
                toolBarRender={(_action, _rows): any => {
                  return (
                    <Button
                      icon={<PlusOutlined />}
                      type="primary"
                      onClick={() => setOpenModal(true)}
                    >
                      Додати нове
                    </Button>
                  );
                }}
              />
            </Access>
            <ModalRole
              openModal={openModal}
              setOpenModal={setOpenModal}
              reloadTable={reloadTable}
            />
          </div>
        )
      }

export default RolePage; import { Button, Divider, Form, Input, message, notification } from 'antd';
import { Link, useLocation, useNavigate } from 'react-router-dom';
```

```
import { callLogin } from 'config/api';
import { useState, useEffect } from 'react';
import { useDispatch } from 'react-redux';
import { setUserLoginInfo } from '@/redux/slice/accountSlide';
import styles from 'styles/auth.module.scss';
import { useAppSelector } from '@/redux/hooks';

const LoginPage = () => {
    const navigate = useNavigate();
    const [isSubmit, setIsSubmit] = useState(false);
    const dispatch = useDispatch();
    const isAuthenticated = useAppSelector(state => state.account.isAuthenticated);

    let location = useLocation();
    let params = new URLSearchParams(location.search);
    const callback = params?.get("callback");

    useEffect(() => {
        //đã login => redirect to '/'
        if (isAuthenticated) {
            // navigate('/');
            window.location.href = '/';
        }
    }, [])

    const onFinish = async (values: any) => {
        const { username, password } = values;
        setIsSubmit(true);
        const res = await callLogin(username, password);
        setIsSubmit(false);
        if (res?.data) {
            localStorage.setItem('access_token', res.data.access_token);
            dispatch(setUserLoginInfo(res.data.user))
            message.success('Увійдіть у успішний рахунок!');
            window.location.href = callback ? callback : '/';
        } else {
            notification.error({
                message: "Сталася помилка",
                description:
                    res.message && Array.isArray(res.message) ? res.message[0] : res.message,
                duration: 5
```

```
      })
    }
};


return (
  <div className={styles["login-page"]}>
    <main className={styles.main}>
      <div className={styles.container}>
        <section className={styles.wrapper}>
          <div className={styles.heading}>
            <h2 className={`${styles.text} ${styles["text-large"]}`}>Увійти</h2>
            <Divider />

          </div>
          <Form
            name="basic"
            // style={{ maxWidth: 600, margin: '0 auto' }}
            onFinish={onFinish}
            autoComplete="off"
          >
            <Form.Item
              labelCol={{ span: 24 }} //whole column
              label="Електронна пошта"
              name="username"
              rules={[{ required: true, message: 'Електронна пошта не повинна залишатися порожньою!' }]}
            >
              <Input />
            </Form.Item>

            <Form.Item
              labelCol={{ span: 24 }} //whole column
              label="Пароль"
              name="password"
              rules={[{ required: true, message: 'Пароль заборонено залишати порожню!' }]}
            >
              <Input.Password />
            </Form.Item>

            <Form.Item
            // wrapperCol={{ offset: 6, span: 16 }}
```

```
                    >
                        <Button type="primary" htmlType="submit" loading={isSubmit}>
                            Увійти
                        </Button>
                    </Form.Item>
                    <Divider>Або</Divider>
                    <p className="text text-normal">Немає рахунку?
                        <span>
                            <Link to='/register' > Реєстрація </Link>
                        </span>
                    </p>
                </Form>
            </section>
        </div>
    </main>
</div>
)
}

export default LoginPage; import { Button, Divider, Form, Input, Row, Select, message, notification } from 'antd';
import { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import { callRegister } from 'config/api';
import styles from 'styles/auth.module.scss';
import { IUser } from '@/types/backend';
const { Option } = Select;


const RegisterPage = () => {
    const navigate = useNavigate();
    const [isSubmit, setIsSubmit] = useState(false);

    const onFinish = async (values: IUser) => {
        const { name, email, password, age, gender, address } = values;
        setIsSubmit(true);
        const res = await callRegister(name, email, password as string, +age, gender, address);
        setIsSubmit(false);
        if (res?.data?._id) {
            message.success('Успішна реєстрація рахунку!');
            navigate('/login')
        } else {
```

```
        notification.error({
          message: "Сталася помилка",
          description:
            res.message && Array.isArray(res.message) ? res.message[0] : res.message,
          duration: 5
        })
      }
};


return (
  <div className={styles["register-page"]} >

    <main className={styles.main} >
      <div className={styles.container} >
        <section className={styles.wrapper} >
          <div className={styles.heading} >
            <h2 className={`${styles.text} ${styles["text-large"]}`}> Реєстрація рахунків </h2>
            < Divider />
          </div>
          < Form<IUser>
            name="basic"
            // style={{ maxWidth: 600, margin: '0 auto' }}
            onFinish={onFinish}
            autoComplete="off"
          >
            <Form.Item
              labelCol={{ span: 24 }} //whole column
              label="Повна назва"
              name="name"
              rules={[{ required: true, message: 'Повне ім’я не повинно залишатися порожнім!' }]}
            >
              <Input />
            </Form.Item>


            <Form.Item
              labelCol={{ span: 24 }
              } //whole column
              label="Електронна пошта"
              name="email"
```

```
    rules={[{ required: true, message: 'Електронна пошта не повинна залишатися порожньою!' }]}
>
    <Input type='email' />
</Form.Item>

<Form.Item
    labelCol={{ span: 24 }} //whole column
    label="Пароль"
    name="password"
    rules={[{ required: true, message: 'Пароль заборонено залишати порожню!' }]}
>
    <Input.Password />
</Form.Item>
<Form.Item
    labelCol={{ span: 24 }} //whole column
    label="Річний"
    name="age"
    rules={[{ required: true, message: 'Вік не залишається порожнім!' }]}
>
    <Input type='number' />
</Form.Item>


<Form.Item
    labelCol={{ span: 24 }} //whole column
    name="gender"
    label="Стать"
    rules={[{ required: true, message: 'Гендер не повинен залишатися порожнім!' }]}
>
    <Select
        // placeholder="Select a option and change input text above"
        // onChange={onGenderChange}
        allowClear
    >
        <Option value="male">Назва</Option>
        <Option value="female">Жіночий</Option>
        <Option value="other">Інший</Option>
    </Select>
</Form.Item>
```

```
                    <Form.Item
                      labelCol={{ span: 24 }} //whole column
                      label="Адреса"
                      name="address"
                      rules={[{ required: true, message: 'Адреса не повинна залишатися порожньою!' }]}
                    >
                      <Input />
                    </Form.Item>

                    < Form.Item
                    // wrapperCol={{ offset: 6, span: 16 }}
                    >
                      <Button type="primary" htmlType="submit" loading={isSubmit} >
                        Реєстрація
                      </Button>
                    </Form.Item>
                    <Divider> Або </Divider>
                    <p className="text text-normal" > Є рахунок?
                      <span>
                        <Link to='/login' > Увійти </Link>
                      </span>
                    </p>
                  </Form>
                </section>
              </div>
            </main>
          </div>
    )
}

export default RegisterPage; import { useLocation, useNavigate } from "react-router-dom";
import { useState, useEffect } from 'react';
import { ICompany } from "@/types/backend";
import { callFetchCompanyById } from "@/config/api";
import styles from 'styles/client.module.scss';
import parse from 'html-react-parser';
import { Col, Divider, Row, Skeleton } from "antd";
import { EnvironmentOutlined } from "@ant-design/icons";


const ClientCompanyDetailPage = (props: any) => {
```

```
const [companyDetail, setCompanyDetail] = useState<ICompany | null>(null);
const [isLoading, setIsLoading] = useState<boolean>(false);

let location = useLocation();
let params = new URLSearchParams(location.search);
const id = params?.get("id"); // job id

useEffect(() => {
    const init = async () => {
        if (id) {
            setIsLoading(true)
            const res = await callFetchCompanyById(id);
            if (res?.data) {
                setCompanyDetail(res.data)
            }
            setIsLoading(false)
        }
    }
    init();
}, [id]);

return (
    <div className={`${styles["container"]} ${styles["detail-job-section"]}`}>
        {isLoading ?
            <Skeleton />
            :
            <Row gutter={[20, 20]}>
                {companyDetail && companyDetail._id &&
                    <>
                        <Col span={24} md={16}>
                            <div className={styles["header"]}>
                                {companyDetail.name}
                            </div>

                            <div className={styles["location"]}>
                                <EnvironmentOutlined style={{ color: '#58aaab' }} /> {(companyDetail?.address)}
                            </div>

                            <Divider />
                            {parse(companyDetail?.description ?? "")}
                        </Col>
```

```
                    <Col span={24} md={8}>
                        <div className={styles["company"]}>
                            <div>
                                <img
                                    alt="example"

src={`${import.meta.env.VITE_BACKEND_URL}/images/company/${companyDetail?.logo}`}
                                />
                            </div>
                            <div>
                                {companyDetail?.name}
                            </div>
                        </div>
                    </Col>
                </>
            }
        </Row>
    }
  </div>
 )
}
export default ClientCompanyDetailPage; import { Divider } from 'antd';
import styles from 'styles/client.module.scss';
import SearchClient from '@/components/client/search.client';
import JobCard from '@/components/client/card/job.card';
import CompanyCard from '@/components/client/card/company.card';

const HomePage = () => {
 return (
  <div className={`${styles["container"]} ${styles["home-section"]}`}>
   <div className="search-content" style={{ marginTop: 20 }}>
    <SearchClient />
   </div>
   <Divider />
   <CompanyCard />
   <div style={{ margin: 50 }}></div>
   <Divider />
   <JobCard />
  </div>
 )
```

```
}

export default HomePage; import { createAsyncThunk, createSlice } from '@reduxjs/toolkit';
import { callFetchAccount } from '@/config/api';

// First, create the thunk
export const fetchAccount = createAsyncThunk(
    'account/fetchAccount',
    async () => {
        const response = await callFetchAccount();
        return response.data;
    }
)

interface IState {
    isAuthenticated: boolean;
    isLoading: boolean;
    isRefreshToken: boolean;
    errorRefreshToken: string;
    user: {
        _id: string;
        email: string;
        name: string;
        role: {
            _id: string;
            name: string;
        }
        permissions: {
            _id: string;
            name: string;
            apiPath: string;
            method: string;
            module: string;
        }[]
    };
    activeMenu: string;
}

const initialState: IState = {
    isAuthenticated: false,
    isLoading: true,
```

```
    isRefreshToken: false,
    errorRefreshToken: "",
    user: {
        _id: "",
        email: "",
        name: "",
        role: {
            _id: "",
            name: "",
        },
        permissions: [],
    },

    activeMenu: 'home'
};


export const accountSlide = createSlice({
    name: 'account',
    initialState,
    // The `reducers` field lets us define reducers and generate associated actions
    reducers: {
        // Use the PayloadAction type to declare the contents of `action.payload`
        setActiveMenu: (state, action) => {
            state.activeMenu = action.payload;
        },
        setUserLoginInfo: (state, action) => {
            state.isAuthenticated = true;
            state.isLoading = false;
            state.user._id = action?.payload?._id;
            state.user.email = action.payload.email;
            state.user.name = action.payload.name;
            state.user.role = action?.payload?.role;
            state.user.permissions = action?.payload?.permissions;
        },
        setLogoutAction: (state, action) => {
            localStorage.removeItem('access_token');
            state.isAuthenticated = false;
            state.user = {
                _id: "",
                email: "",
```

```
            name: "",
            role: {
                _id: "",
                name: "",
            },
            permissions: [],
        }
    },
    setRefreshTokenAction: (state, action) => {
        state.isRefreshToken = action.payload?.status ?? false;
        state.errorRefreshToken = action.payload?.message ?? "";
    }

},
extraReducers: (builder) => {
    // Add reducers for additional action types here, and handle loading state as needed
    builder.addCase(fetchAccount.pending, (state, action) => {
        if (action.payload) {
            state.isAuthenticated = false;
            state.isLoading = true;
        }
    })

    builder.addCase(fetchAccount.fulfilled, (state, action) => {
        if (action.payload) {
            state.isAuthenticated = true;
            state.isLoading = false;
            state.user._id = action?.payload?.user?._id;
            state.user.email = action.payload.user?.email;
            state.user.name = action.payload.user?.name;
            state.user.role = action?.payload?.user?.role;
            state.user.permissions = action?.payload?.user?.permissions;
        }
    })

    builder.addCase(fetchAccount.rejected, (state, action) => {
        if (action.payload) {
            state.isAuthenticated = false;
            state.isLoading = false;
        }
    })
```

```
    },

});

export const {
    setActiveMenu, setUserLoginInfo, setLogoutAction, setRefreshTokenAction
} = accountSlide.actions;

export default accountSlide.reducer; import { createAsyncThunk, createSlice } from '@reduxjs/toolkit';
import { callFetchCompany } from '@/config/api';
import { ICompany } from '@/types/backend';

interface IState {
    isFetching: boolean;
    meta: {
        current: number;
        pageSize: number;
        pages: number;
        total: number;
    },
    result: ICompany[]
}
// First, create the thunk
export const fetchCompany = createAsyncThunk(
    'company/fetchCompany',
    async ({ query }: { query: string }) => {
        const response = await callFetchCompany(query);
        return response;
    }
)


const initialState: IState = {
    isFetching: true,
    meta: {
        current: 1,
        pageSize: 10,
        pages: 0,
        total: 0
    },
```

```
  result: []
};


export const companySlide = createSlice({
  name: 'company',
  initialState,
  // The `reducers` field lets us define reducers and generate associated actions
  reducers: {
    // Use the PayloadAction type to declare the contents of `action.payload`
    setActiveMenu: (state, action) => {
      // state.activeMenu = action.payload;
    },


  },
  extraReducers: (builder) => {
    // Add reducers for additional action types here, and handle loading state as needed
    builder.addCase(fetchCompany.pending, (state, action) => {
      state.isFetching = true;
      // Add user to the state array
      // state.courseOrder = action.payload;
    })

    builder.addCase(fetchCompany.rejected, (state, action) => {
      state.isFetching = false;
      // Add user to the state array
      // state.courseOrder = action.payload;
    })

    builder.addCase(fetchCompany.fulfilled, (state, action) => {
      if (action.payload && action.payload.data) {
        state.isFetching = false;
        state.meta = action.payload.data.meta;
        state.result = action.payload.data.result;
      }
      // Add user to the state array

      // state.courseOrder = action.payload;
    })
  },
```

```
});

export const {
    setActiveMenu,
} = companySlide.actions;

export default companySlide.reducer; import { createAsyncThunk, createSlice } from '@reduxjs/toolkit';
import { callFetchRole, callFetchRoleById } from '@/config/api';
import { IRole } from '@/types/backend';

interface IState {
    isFetching: boolean;
    meta: {
        current: number;
        pageSize: number;
        pages: number;
        total: number;
    },
    result: IRole[];
    isFetchSingle: boolean;
    singleRole: IRole
}
// First, create the thunk
export const fetchRole = createAsyncThunk(
    'resume/fetchRole',
    async ({ query }: { query: string }) => {
        const response = await callFetchRole(query);
        return response;
    }
)

export const fetchRoleById = createAsyncThunk(
    'resume/fetchRoleById',
    async (id: string) => {
        const response = await callFetchRoleById(id);
        return response;
    }
)
```

```
const initialState: IState = {
    isFetching: true,
    isFetchSingle: true,
    meta: {
        current: 1,
        pageSize: 10,
        pages: 0,
        total: 0
    },
    result: [],
    singleRole: {
        _id: "",
        name: "",
        description: "",
        isActive: false,
        permissions: []
    }
};



export const roleSlide = createSlice({
    name: 'role',
    initialState,
    // The `reducers` field lets us define reducers and generate associated actions
    reducers: {

        resetSingleRole: (state, action) => {
            state.singleRole = {
                _id: "",
                name: "",
                description: "",
                isActive: false,
                permissions: []
            }
        },

    },
    extraReducers: (builder) => {
        // Add reducers for additional action types here, and handle loading state as needed
        builder.addCase(fetchRole.pending, (state, action) => {
            state.isFetching = true;
```

```
    // Add user to the state array
    // state.courseOrder = action.payload;
})

builder.addCase(fetchRole.rejected, (state, action) => {
    state.isFetching = false;
    // Add user to the state array
    // state.courseOrder = action.payload;
})

builder.addCase(fetchRole.fulfilled, (state, action) => {
    if (action.payload && action.payload.data) {
        state.isFetching = false;
        state.meta = action.payload.data.meta;
        state.result = action.payload.data.result;
    }
    // Add user to the state array

    // state.courseOrder = action.payload;
})

builder.addCase(fetchRoleById.pending, (state, action) => {
    state.isFetchSingle = true;
    state.singleRole = {
        _id: "",
        name: "",
        description: "",
        isActive: false,
        permissions: []
    }
    // Add user to the state array
    // state.courseOrder = action.payload;
})

builder.addCase(fetchRoleById.rejected, (state, action) => {
    state.isFetchSingle = false;
    state.singleRole = {
        _id: "",
        name: "",
        description: "",
        isActive: false,
```

```
            permissions: []
        }
        // Add user to the state array
        // state.courseOrder = action.payload;
    })

    builder.addCase(fetchRoleById.fulfilled, (state, action) => {
        if (action.payload && action.payload.data) {
            state.isFetchSingle = false;
            state.singleRole = action.payload.data;
        }
    })
},

});

export const {
    resetSingleRole
} = roleSlide.actions;

export default roleSlide.reducer;
```

Додаток В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

«ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ НАДАННЯ РЕКОМЕНДАЦІЙ ДЛЯ ДОГЛЯДУ ЗА КІМНАТНИМИ РОСЛИНАМИ»

Виконав: студент 2-го курсу, групи 2КН-23м
спеціальності 122 – «Комп`ютерні науки»
_____ Стасишен Д. В.
(прізвище та ініціали)

Керівник: д.т.н., доц. каф. КН
_____ Арсенюк І. Р.
(прізвище та ініціали)
«_05_» _____12_____2024 р.

Вінниця ВНТУ - 2024 рік

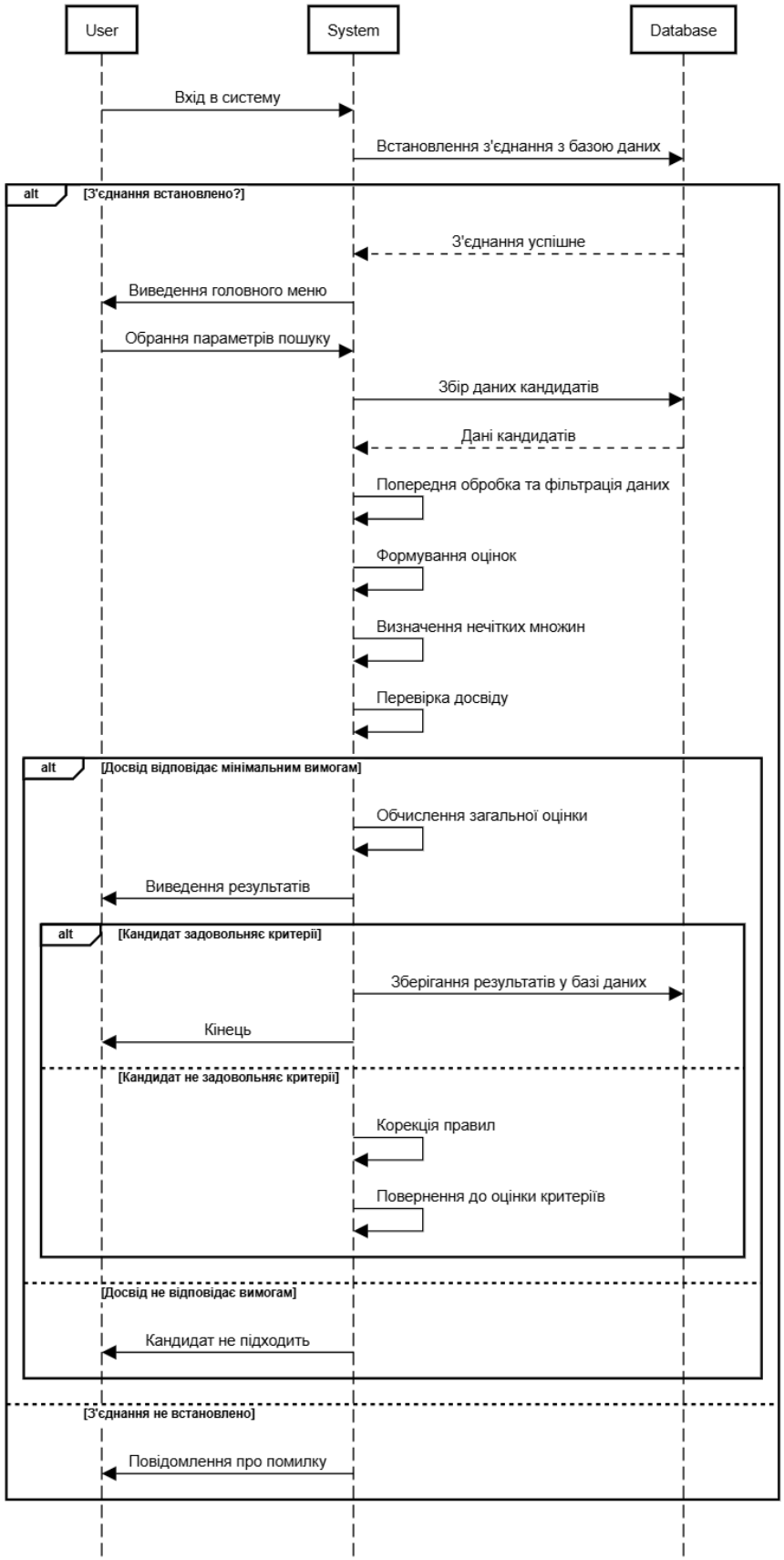

Рисунок В.1 – UML-діаграма послідовності взаємодії модулів інформаційної технології пошуку роботи

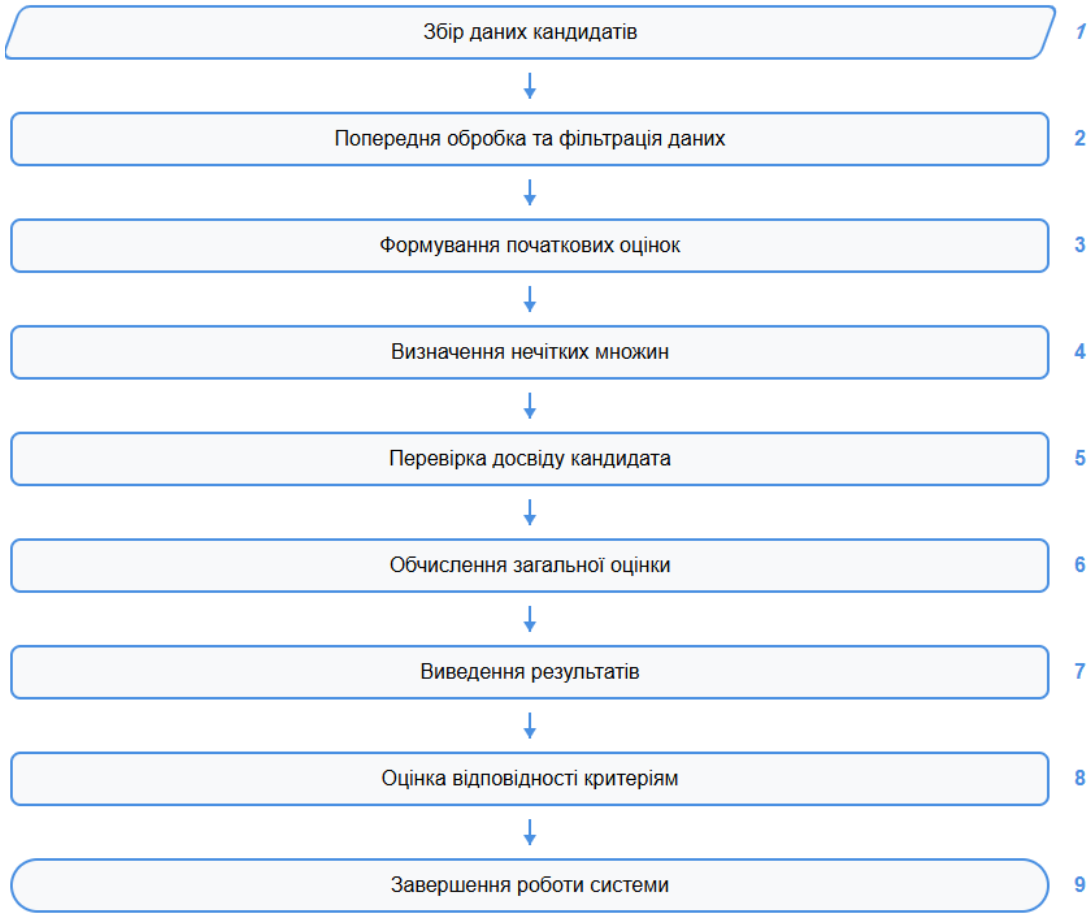

Рисунок В.2 – Структура інформаційної технології пошуку роботи з використанням аналізу даних та нечіткої логіки

Рисунок В.3 – Схема системи клієнт-сервер

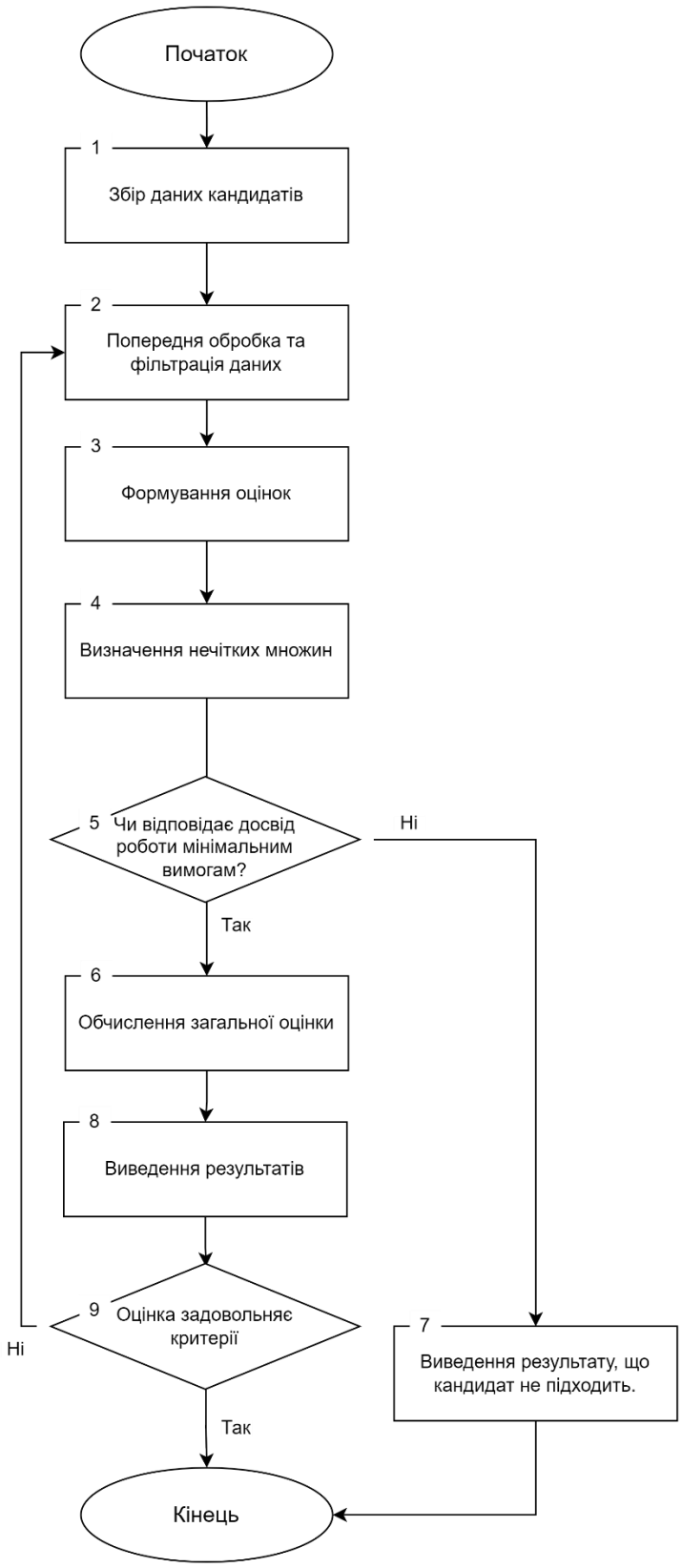

Рисунок В.4 – Схема роботи аналізу даних та побудова системи оцінки на основі нечіткої логіки

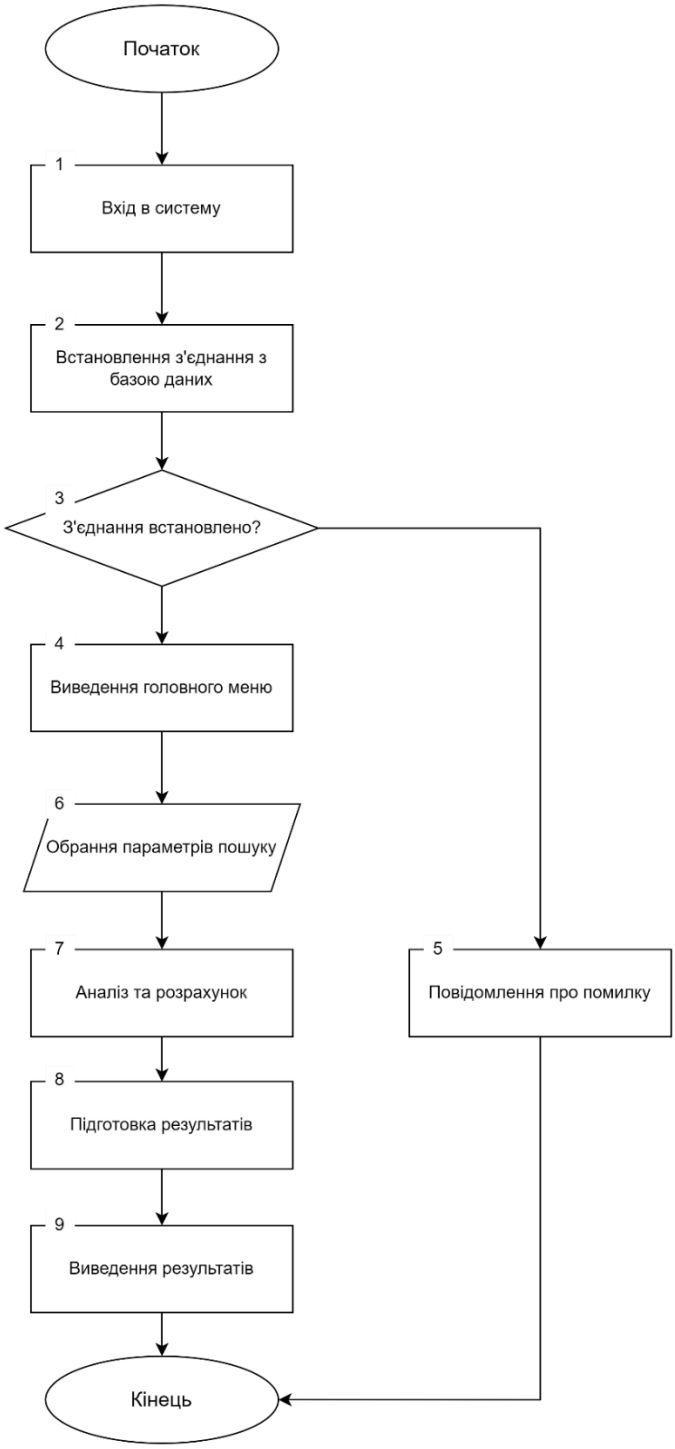

Рисунок В.5 – Схема послідновності при взаємодії користувача з модулем оцінки кандидатів

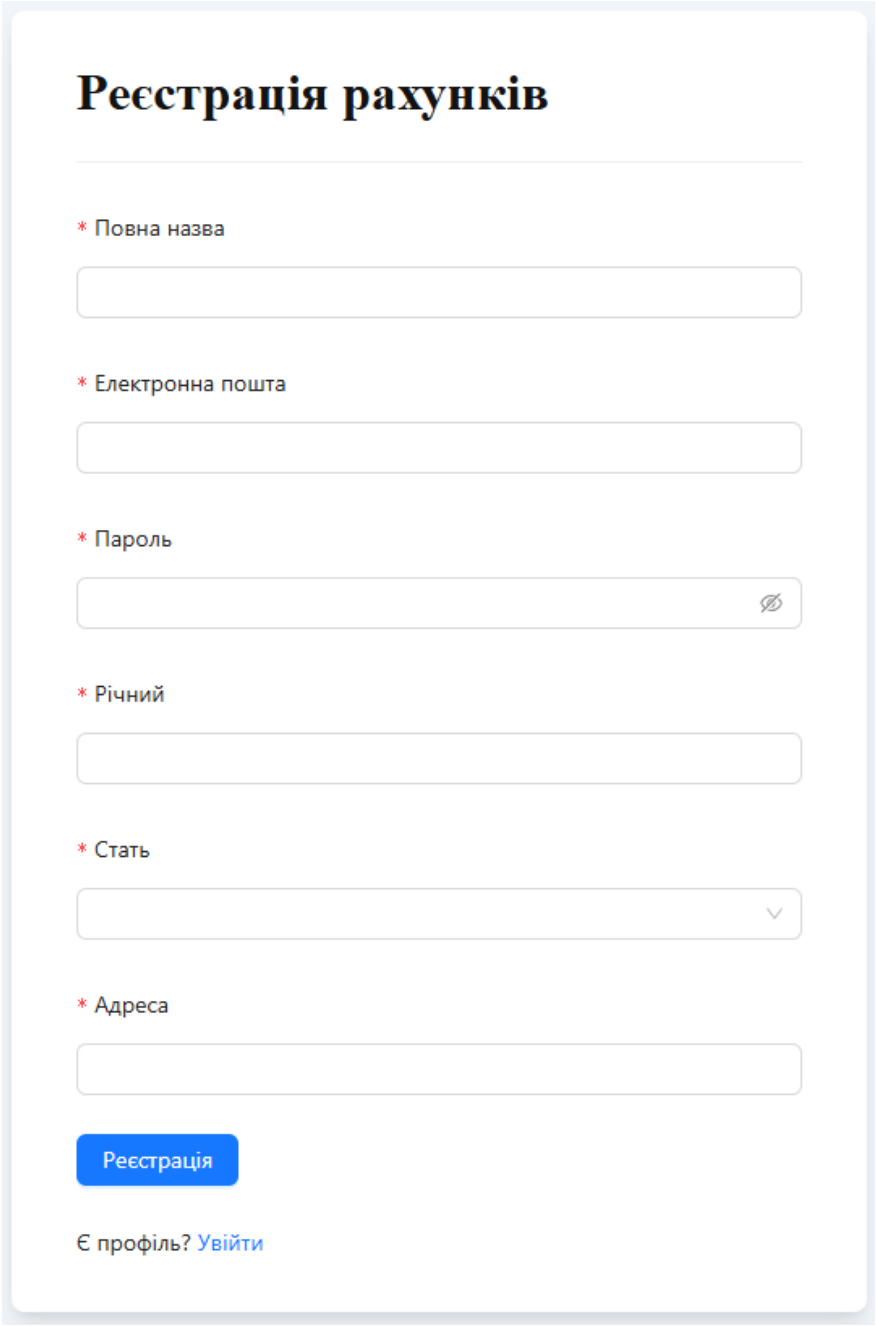

Рисунок В.6 – Загальний вигляд інтерфейсного вікна реєстрації користувача

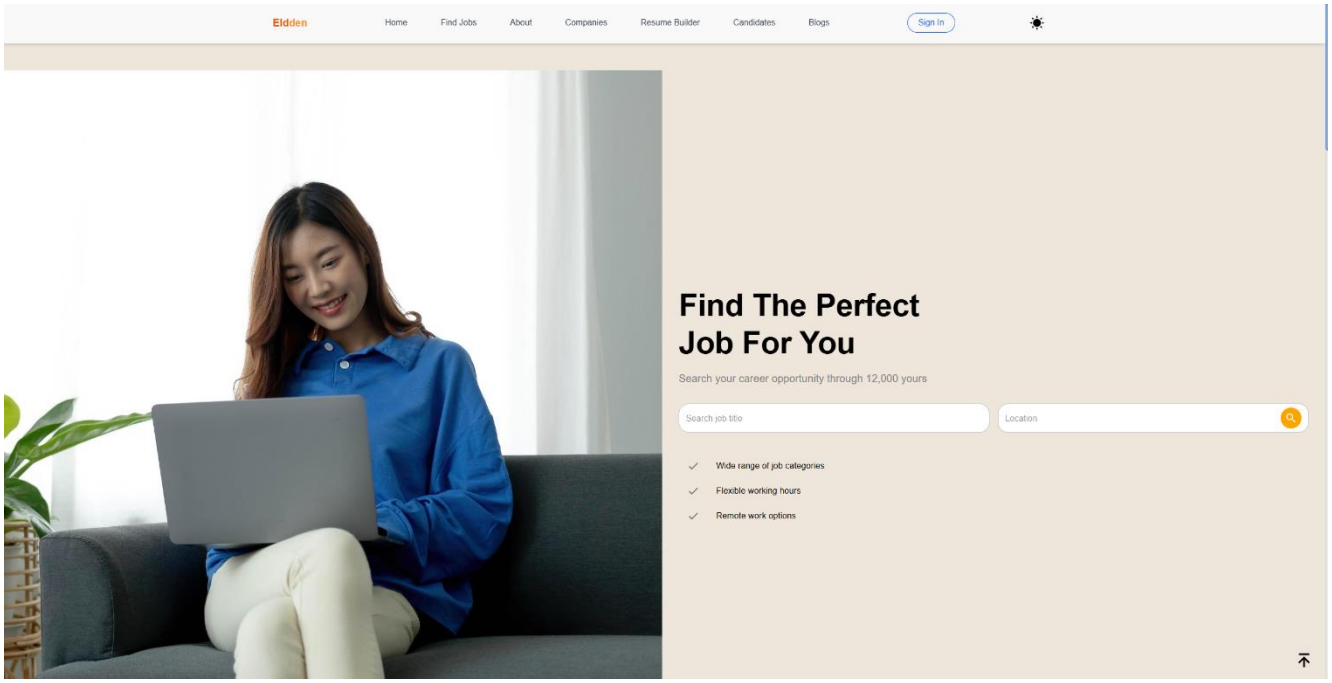

Рисунок В.7 – Загальний вигляд вікна головної сторінки

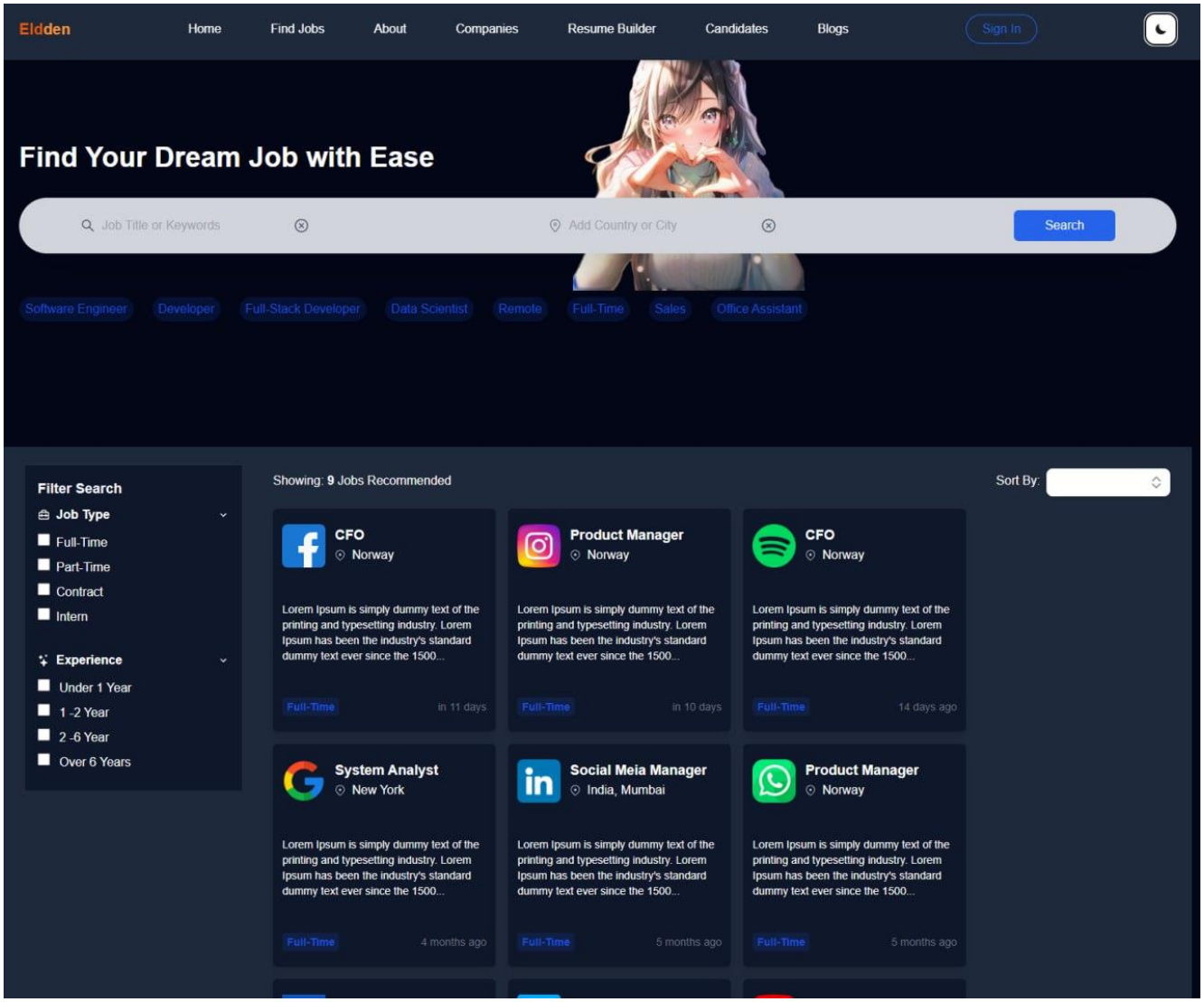

Рисунок В.8 – Загальний вигляд вікна сторінки з рекомендованими вакансіями

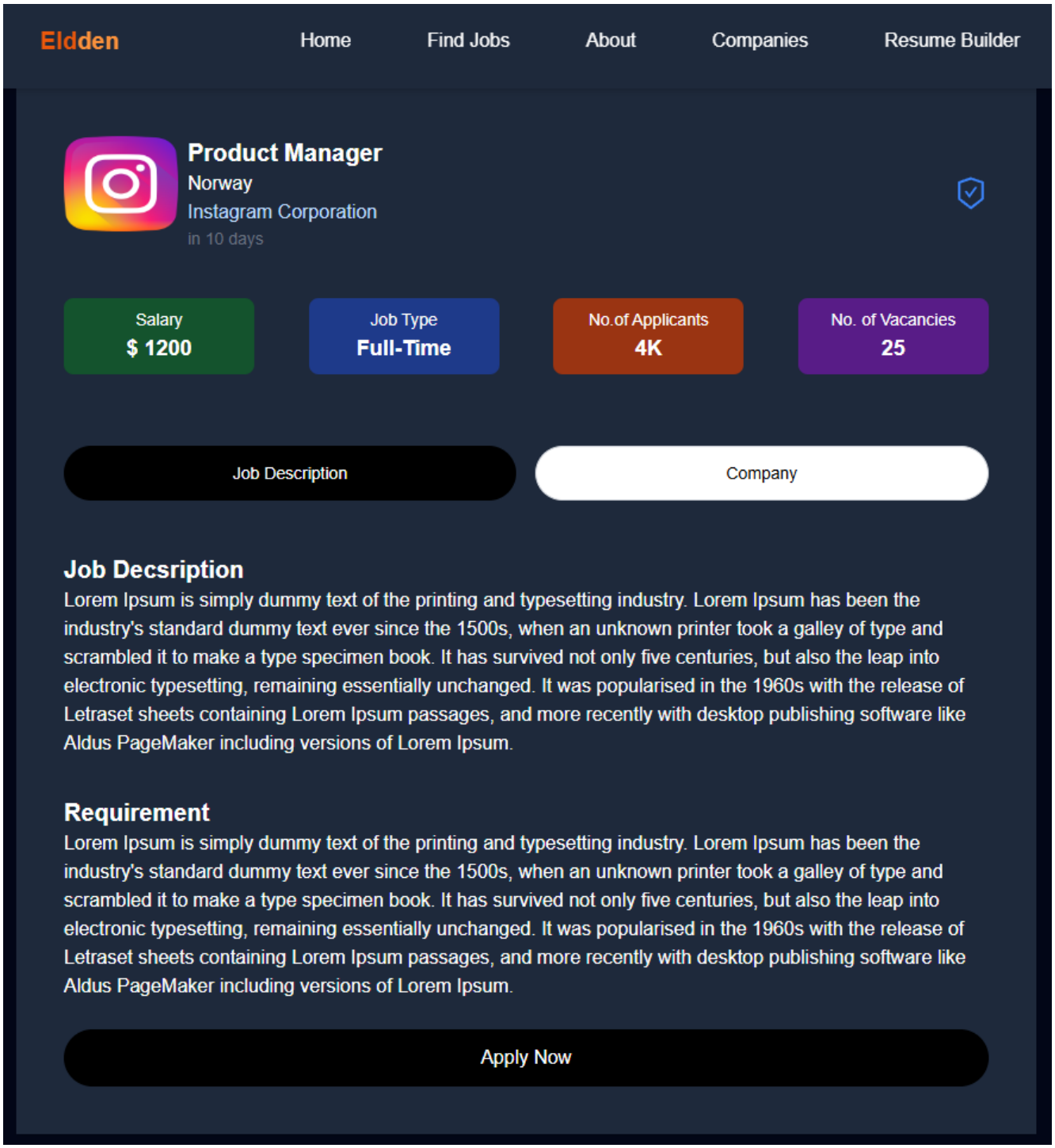

Рисунок В.9 – Загальний вигляд вікна сторінки з детальною інформацією про вакансію

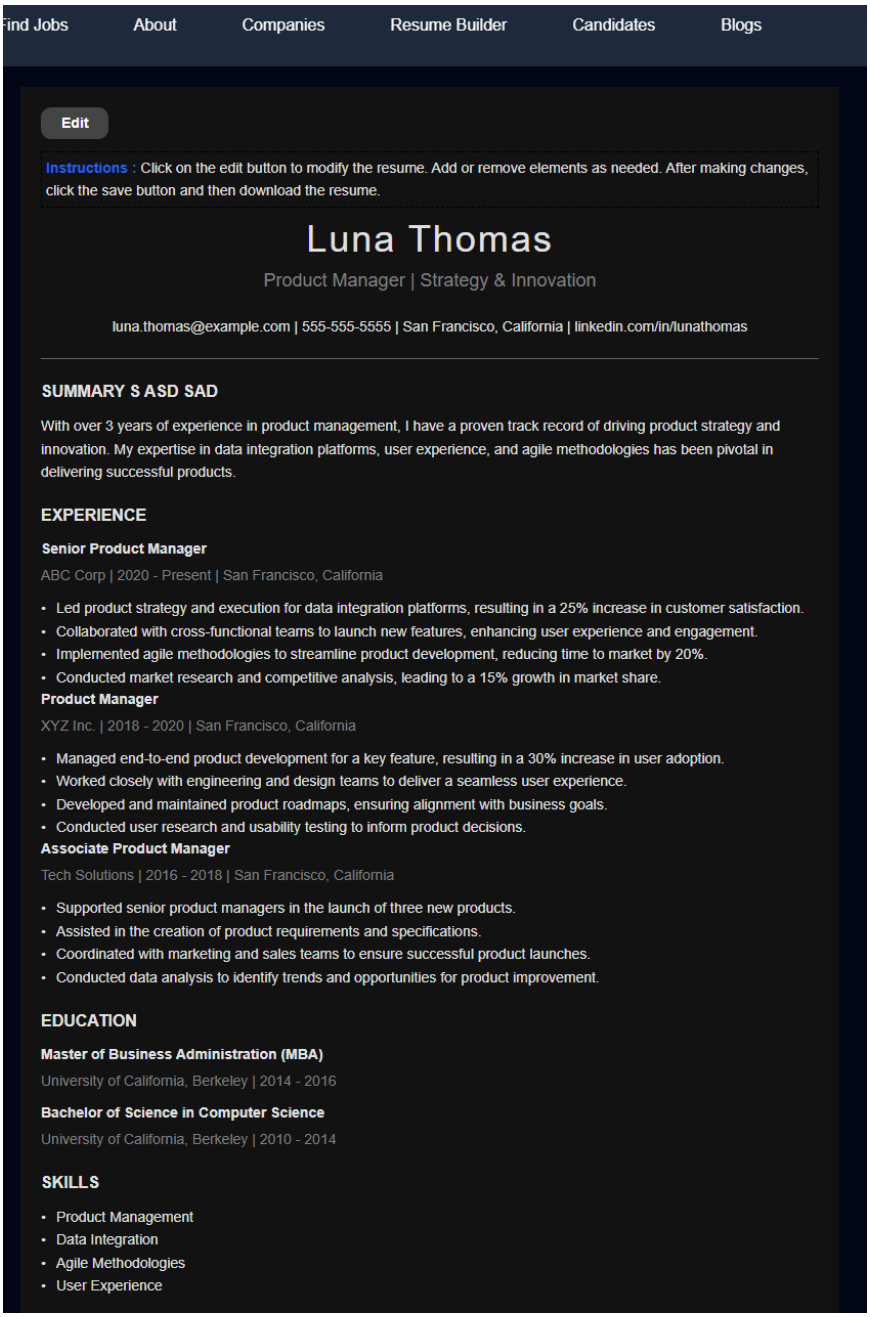

Рисунок В.10 – Загальний вигляд вікна сторінки для створення та завантаження резюме

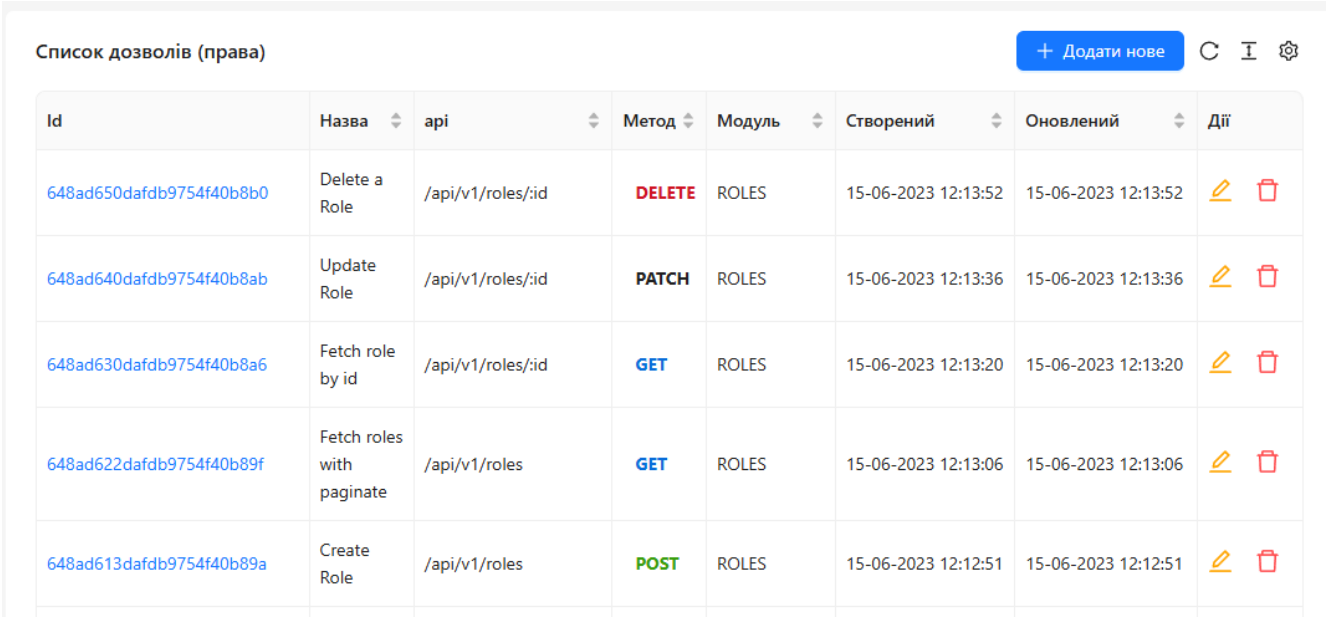

Рисунок В.11 – Загальний вигляд вікна сторінки адміністративної панелі для управління списком прав користувачів

Додаток Г (довідниковий)

Інструкція користувача

Для початку роботи з інформаційною технологією пошуку роботи необхідно мати доступ до інтернет-браузера та стабільне підключення до Інтернету через мобільну мережу або Wi-Fi (рис. Г.1).

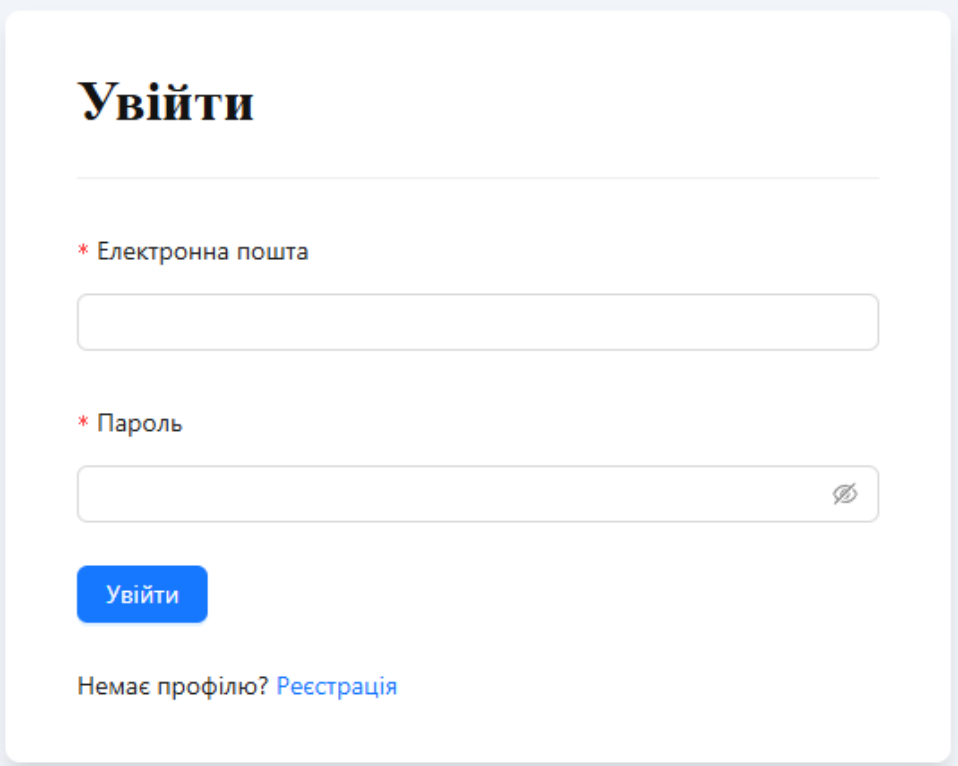

Рисунок Г.1 – Загальний вигляд інтерфейсного вікна авторизації користувача

Після успішної авторизації користувач потрапляє на головне меню системи, яке містить усю необхідну інформацію. У цьому вікні відображаються персоналізовані рекомендації щодо вакансій, які система визначає як найкраще відповідні для користувача. Кожна вакансія представлена окремим блоком, що містить основну інформацію про неї. Для перегляду деталей користувач може натиснути кнопку “Детальніше”. На рисунку Г.2 зображено вигляд блоку з вакансіями.

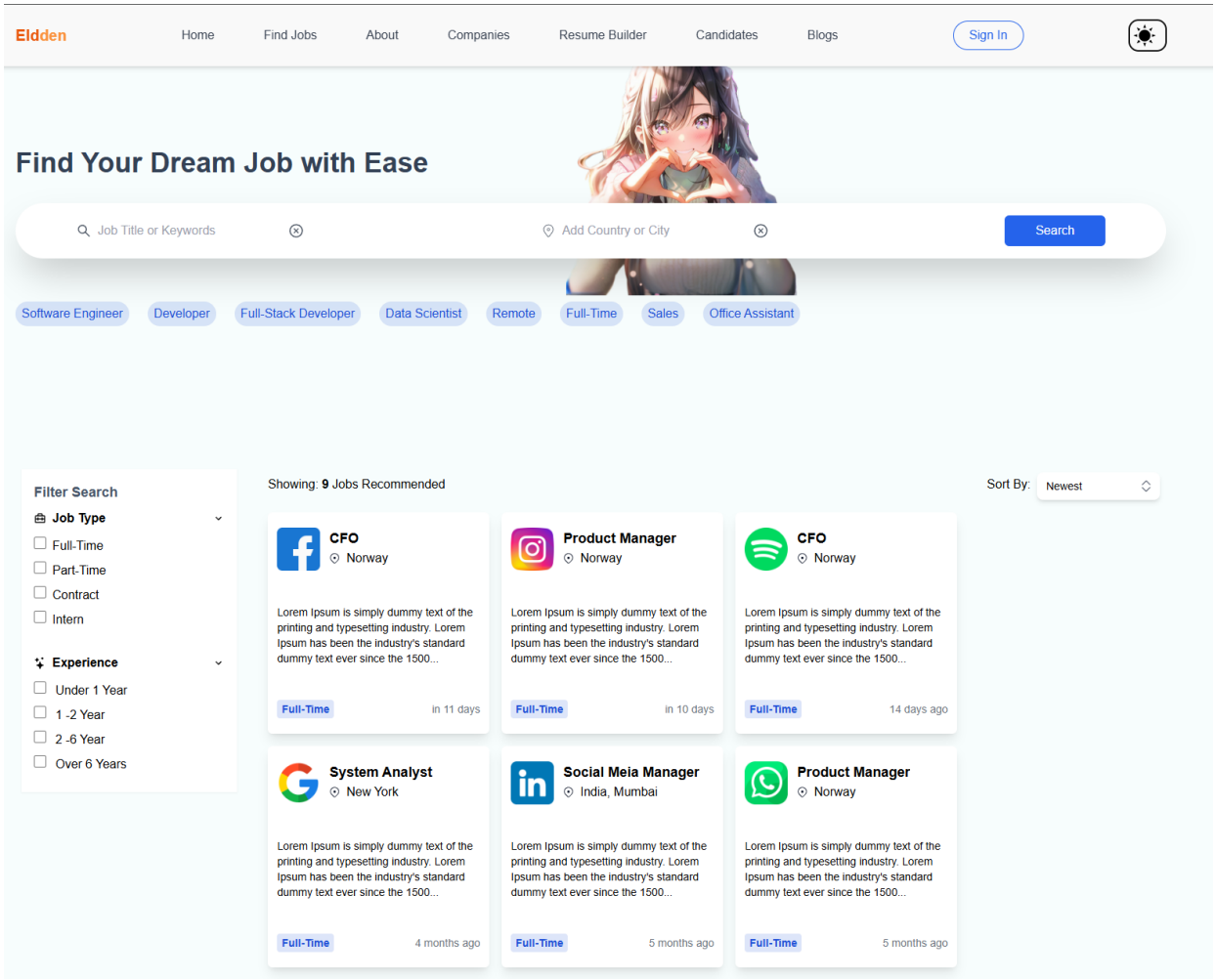

Рисунок Г.2 – Загальний вигляд блоку з вакансіями

При натисканні на рекомендовану вакансію відкривається вікно з деталями, що надає користувачу повну інформацію про обрану позицію. Користувач може переглянути деталі вакансії, її вимоги, а також основні характеристики, такі як зарплата, місцезнаходження, тип зайнятості та інше. Під інформацією про вакансію користувач також побачить список схожих вакансій, які можуть бути релевантними. Вигляд вікна з інформацією про вакансію показано на рисунку Г.3, а вигляд списку схожих вакансій – на рисунку Г.4.

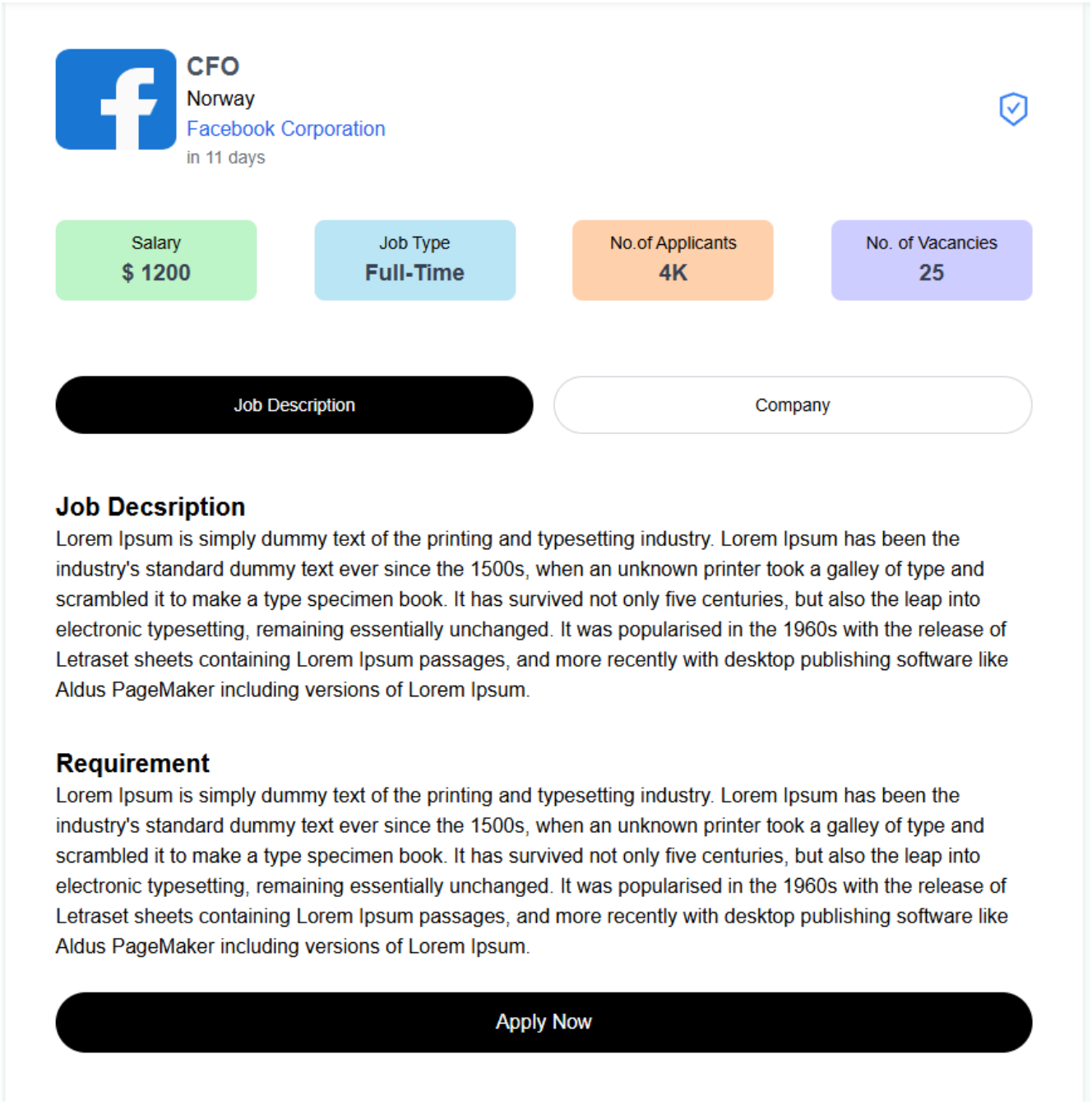

Рисунок Г.3 – Вигляд вікна інформації про вакансію

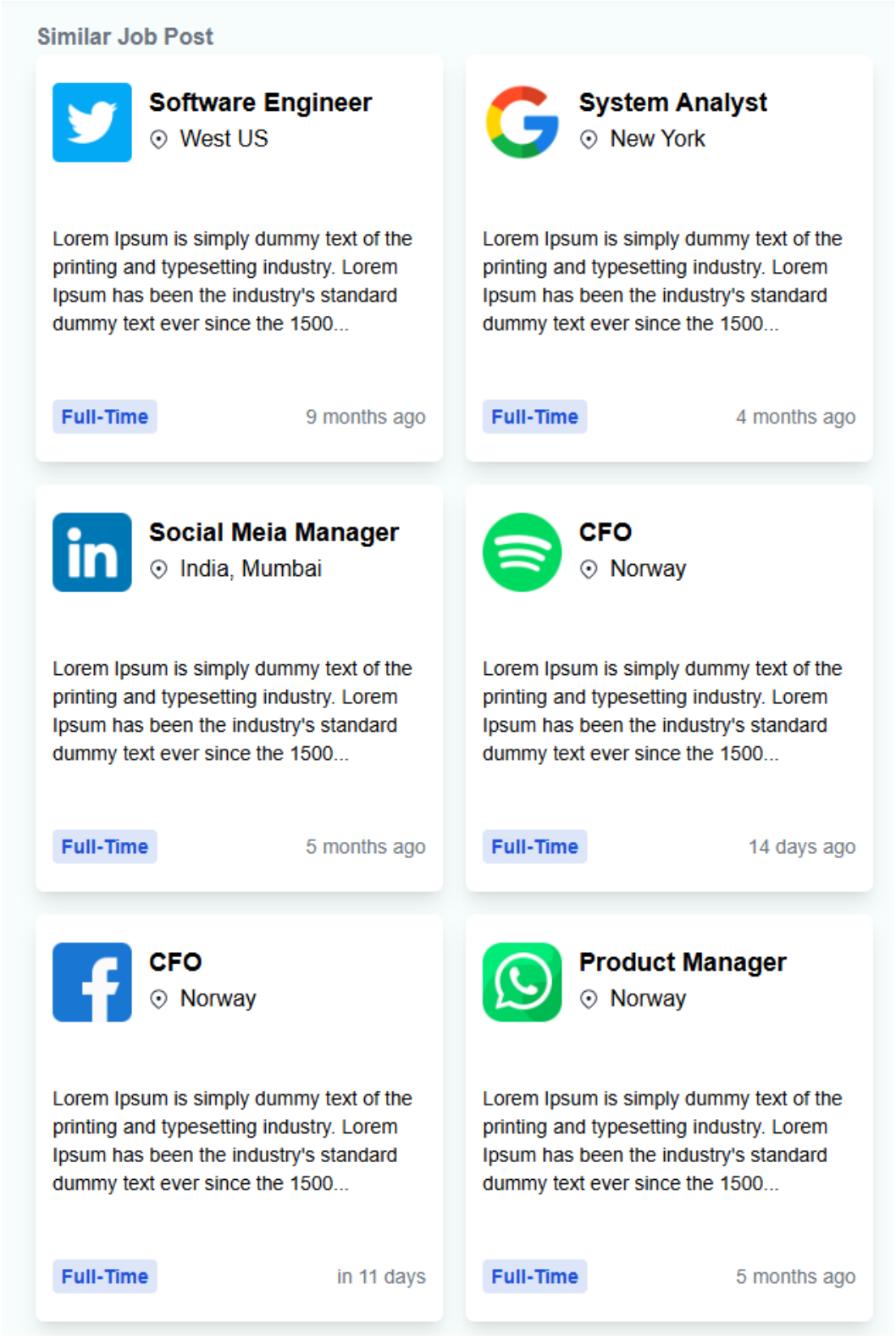

Рисунок Г.4 – Вигляд списку схожих вакансій