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

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна технологія пошуку роботи. Частина 2. Надання рекомендацій»

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

Стаднік Е. 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.

м. Вінниця - 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти ІІ-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

Стадніку Ельдару Васильовичу
(прізвище, ім'я, по батькові)

1. Тема роботи: «Інформаційна технологія пошуку роботи. Частина 2. Надання рекомендацій»
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: кількість пропонованих на даний час вакансій – 200; час від подачі заявки до отримання результату прогнозування не перевищує 1 хвилину; мінімальна; загальна база даних вакансій не менше 1000; інтуїтивно зрозумілий інтерфейс користувача; мова програмування – об'єктно-орієнтована.
4. Зміст текстової частини: вступ, обґрунтування доцільності розробки інформаційної технології пошуку роботи; моделювання інформаційної технології пошуку роботи; обґрунтування вибору програмних засобів для реалізації інформаційної технології; економічна частина, висновки, список використаних джерел, додатки.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): uml-діаграма послідовності взаємодії модулів інформаційної технології для реєстрації або входу в систему; uml-діаграма послідовності взаємодії модулів інформаційної технології для реєстрації або входу в систему; структура інформаційної технології надання рекомендацій для пошуку роботи; схема алгоритму роботи інформаційної технології надання рекомендацій для пошуку роботи; серверна частина інформаційної технології у swagger; загальний вигляд інтерфейсного вікна авторизації користувача; інтерфейс для тестування арі рекомендацій у swagger; інтерфейс перегляду вакансії та подібних пропозицій; інтерфейс перегляду вакансії та подібних пропозицій.

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	24.10.24-04.11.24	Розділ 4
5	Апробація результатів дослідження	01.10.24	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 _____ (підпис) Стаднік Е. В. (прізвище та ініціали)

Керівник роботи _____ (підпис) Арсенюк І. Р. (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.42

Стадік Е. В. Інформаційна технологія пошуку роботи. Частина 2. Надання рекомендацій. Магістерська кваліфікаційна робота зі спеціальності 122 Комп'ютерні науки, освітня програма – Системи штучного інтелекту. Вінниця: ВНТУ, 2024. 148 с.

Укр. мовою. Бібліогр.: 34 назв; рис.: 13; табл.: 8.

Дана магістерська кваліфікаційна робота присвячена розробці інформаційної технології пошуку роботи, а саме експертної системи для надання рекомендацій щодо вибору вакансій та серверної частини інформаційної технології. Було проведено аналіз сучасного стану розвитку технологій пошуку роботи. Аналіз програм-аналогів дозволив виявити їх переваги та недоліки. Здійснено проектування експертної системи для пошуку роботи, в ході якого було обґрунтовано вибір продукційної моделі подання знань та проведено структуризацію знань. В ході програмної реалізації спроектованої експертної системи було розроблено базу знань в середовищі CLIPS. Виконано програмну реалізацію серверної частини з використанням технологій .NET Core та MS SQL Server.

В роботі приведені результати досліджень сучасного стану ринку праці та технологій пошуку роботи, а також аналіз існуючих програм-аналогів. Розроблена експертна система дозволяє користувачу отримати персоналізовані рекомендації щодо вибору вакансій, враховуючи його індивідуальні потреби та побажання. Серверна частина забезпечує обробку даних та взаємодію з базою знань.

Ілюстративна частина складається з 14 плакатів із результатами моделювання.

Ключові слова: машинне навчання; персоналізовані рекомендації; автоматизований пошук роботи.

ABSTRACT

Stadik E.V. Information technology of job search. Part 2. Providing recommendations. Master's qualification work in specialty 122 Computer Science, educational program - Artificial Intelligence Systems. Vinnytsia: VNTU, 2024. 148 p.

In English. Bibliogr.: 34 titles; fig.: 13; tables: 8.

This master's qualification work is devoted to the development of information technology for job search, namely an expert system for providing recommendations on the choice of vacancies and the server part of information technology. An analysis of the current state of development of job search technologies was conducted. The analysis of analogous programs allowed to identify their advantages and disadvantages. The design of an expert system for job search was carried out, during which the choice of a product model of knowledge representation was justified and knowledge was structured. During the software implementation of the designed expert system, a knowledge base was developed in the CLIPS environment. The software implementation of the server part was performed using .NET Core and MS SQL Server technologies.

The paper presents the results of research on the current state of the labor market and job search technologies, as well as an analysis of existing analogous programs. The developed expert system allows the user to receive personalized recommendations on the choice of vacancies, taking into account his individual needs and wishes. The server part provides data processing and interaction with the knowledge base.

The illustrative part consists of 14 posters with modeling results.

Keywords: machine learning; personalized recommendations; automated job search.

ЗМІСТ

ВСТУП

Актуальність теми. З моменту інтенсивного впровадження інформаційних технологій та комп'ютерів в життя людей, актуальність ефективного пошуку роботи стрімко зросла у суспільстві. Це обумовлено різноманітними чинниками.

Процес впровадження інформаційних технологій та цифровізація на всіх рівнях життєдіяльності викликали значні зміни на ринку праці. Традиційні підходи пошуку роботи стають менш ефективними, а вимоги роботодавців постійно змінюються. Внаслідок досліджень цих проблем вчені знайшли кореляцію між використанням сучасних технологій пошуку роботи та успішним працевлаштуванням. Таким чином, спостерігається стрімке зростання інтересу до проблеми ефективного пошуку роботи з використанням інформаційних технологій.

Актуальність розробки інформаційної технології пошуку роботи очевидна, оскільки сучасне суспільство відчуває потребу в забезпеченні ефективного працевлаштування через прості та доступні рішення в умовах сучасного ринку праці. Такими рішеннями є веб-застосунки та мобільні застосунки для пошуку роботи, які надають індивідуальні рекомендації щодо вакансій та можливість відслідковувати власні показники та результати пошуку. Користувачі отримують зручність, самостійність та організованість під час пошуку роботи без дороговартісних персональних консультацій чи рекрутерів.

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

Мета і завдання дослідження. Метою дослідження є розширення функціональних можливостей інформаційної технології пошуку роботи застосуванням нечіткої логіки та експертних систем.

Для досягнення даної мети, необхідно сформувати та вирішити наступні завдання:

- здійснити аналіз сучасного стану розвитку технологій пошуку роботи;
- здійснити аналіз моделей подання знань та обґрунтування їх вибору для системи рекомендацій з пошуку роботи;
- провести структуризацію знань для створення системи рекомендацій з пошуку роботи;
- виконати програмну реалізацію експертної системи для надання рекомендацій з пошуку роботи;
- здійснити проектування серверної частини інформаційної технології пошуку роботи;
- виконати програмну реалізацію серверної частини інформаційної технології пошуку роботи.

Об’єктом дослідження є процес надання рекомендацій при пошуку роботи.

Предметом дослідження є програмні засоби надання рекомендацій при пошуку роботи.

Методи дослідження. У роботі використані такі методи наукових досліджень: аналіз аналогічних технологій та програмних рішень для пошуку роботи, що дозволяє визначити більшість проблем та труднощів, які необхідно вирішити в даній роботі. Методи та моделі подання знань, теорія побудови експертних систем, методи інженерії знань, методи та підходи до розробки веб-орієнтованих програмних застосунків, методи об'єктно-орієнтованого програмування.

Наукова новизна одержаних результатів. Запропоновано інформаційну технологію пошуку роботи, що відрізняється від існуючих застосуванням інформаційної моделі із комбінованим застосуванням нечіткої логіки та експертних систем, що забезпечило розширення функціональних можливостей.

Практичне значення одержаних результатів полягає в тому, що визначено основні етапи роботи інформаційної технології пошуку роботи, здійснено проектування інформаційної технології пошуку роботи, а також на основі проведених досліджень здійснено програмну реалізацію серверної частини зазначеної інформаційної технології.

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджується строгістю постановки задач, коректним застосуванням математичних методів під час доведення наукових положень, строгим виведенням аналітичних співвідношень, порівнянням результатів з відомими та збіжністю результатів математичного моделювання з результатами, що отримані під час впровадження розроблених програмних засобів.

Особистий внесок магістранта. Усі результати, наведені у магістерській кваліфікаційній роботі, отримані самостійно. У роботах, опублікованих у співавторстві, автору належать такі результати:

- інтеграція розробленої експертної системи для пошуку роботи до веб-застосунку за допомогою бібліотеки CLIPSCLRWrapper.dll;

- порівняльний аналіз ефективності роботи нечіткої інтелектуальної системи з використанням узагальненої структури та із застосуванням дерева нечіткого логічного виведення для надання рекомендацій з 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 2. Надання рекомендацій».

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ПОШУКУ РОБОТИ

1.1 Аналіз предметної області технологій пошуку роботи

Аналізуючи сучасний стан розвитку технологій пошуку роботи, можна спостерігати значний прогрес та інновації в цій галузі. Спостерігається стрімке зростання ринку застосунків та онлайн-платформ для пошуку роботи. Такі дані свідчать не тільки про економічну привабливість цієї сфери, а й про переосмислення звичок пошуку роботи та працевлаштування. Завдяки сучасним технологіям, пошук роботи стає більш зручним, ефективним та персоналізованим.

Одним із головних трендів у розвитку технологій пошуку роботи є використання штучного інтелекту та машинного навчання. Алгоритми аналізують великі обсяги даних про вакансії та резюме, щоб підібрати найбільш релевантні пропозиції для кандидатів. Це дозволяє зекономити час та зусилля як для роботодавців, так і для пошукачів.

Також все більшої популярності набувають мобільні застосунки та онлайн-платформи для пошуку роботи. Вони надають користувачам зручний доступ до великої бази вакансій, дозволяють створювати та редагувати резюме, відстежувати статус заявок та отримувати сповіщення про нові пропозиції. Такі застосунки стають універсальним інструментом для пошуку роботи, доступним на різних пристроях, і дозволяють кандидатам бути більш самостійними та організованими у своєму пошуку.

Крім того, використання соціальних мереж для пошуку роботи також стає все більш поширеним. Роботодавці активно використовують соціальні мережі для розміщення вакансій та пошуку потенційних кандидатів. LinkedIn, наприклад, став провідною платформою для професійного спілкування та пошуку роботи.

Отже, застосування технологій у пошуку роботи відкриває нові можливості для кандидатів та роботодавців. Вони підвищують ефективність пошуку, надають

персоналізовані рекомендації та сприяють швидкому та успішному працевлаштуванню. Мобільні застосунки, штучний інтелект та соціальні мережі є лише кількома прикладами інновацій, що розширюють можливості пошуку роботи. Враховуючи швидкий розвиток технологій, цей напрямок продовжуватиме зростати та еволюціонувати, сприяючи покращенню процесів пошуку роботи та досягненню високих результатів у працевлаштуванні.

Крім того, слід зазначити, що зростання популярності технологій пошуку роботи відбувається не тільки серед професійних рекрутерів, але й серед звичайних людей. Більшість людей сьогодні мають доступ до смартфонів, планшетів та інших пристроїв, що відкриває їм можливість використовувати ці технології для пошуку роботи та кар'єрного розвитку. Це сприяє більшій мобільності робочої сили та відкриває нові можливості для працевлаштування.

1.2 Аналіз відомих програмних рішень та програм-аналогів

На ринку працевлаштування існує велика кількість застосунків та онлайн-платформ, які дозволяють користувачам шукати вакансії. Основною метою таких застосунків зазвичай є спрощення процесу пошуку роботи та надання користувачам зручних інструментів для пошуку вакансій, які відповідають їхнім потребам та кваліфікації.

LinkedIn є однією з найпопулярніших платформ для професійного спілкування та пошуку роботи [2]. Ця платформа дозволяє користувачам створювати професійні профілі, встановлювати контакти з іншими фахівцями, приєднуватися до професійних груп та шукати вакансії. LinkedIn також пропонує різноманітні інструменти для роботодавців, такі як розміщення вакансій, пошук кандидатів та аналітика ринку праці. Загальний вигляд інтерфейсного вікна сайту продемонстровано на рисунку 1.1.

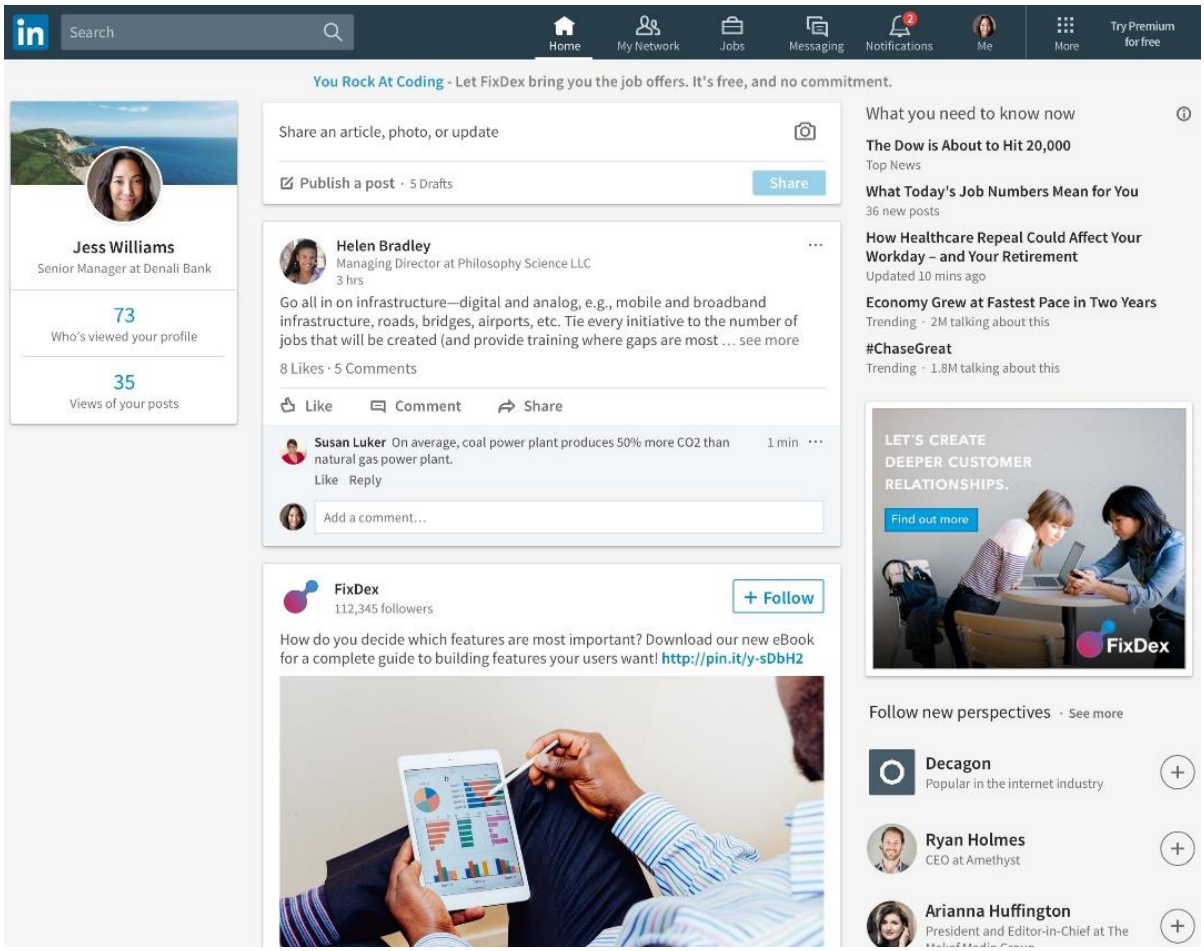

Рисунок 1.1 – Загальний вигляд інтерфейсного вікна сайту Linkedin

Indeed – це ще одна популярна платформа для пошуку роботи, яка агрегує вакансії з різних джерел, включаючи сайти компаній, рекрутингові агентства та інші сайти з працевлаштування [3]. Indeed пропонує користувачам простий та зручний інтерфейс для пошуку вакансій за ключовими словами, місцезнаходженням, галуззю та іншими параметрами. Користувачі можуть також створювати резюме та відстежувати статус своїх заявок. Загальний вигляд інтерфейсного вікна сайту продемонстровано на рисунку 1.2.

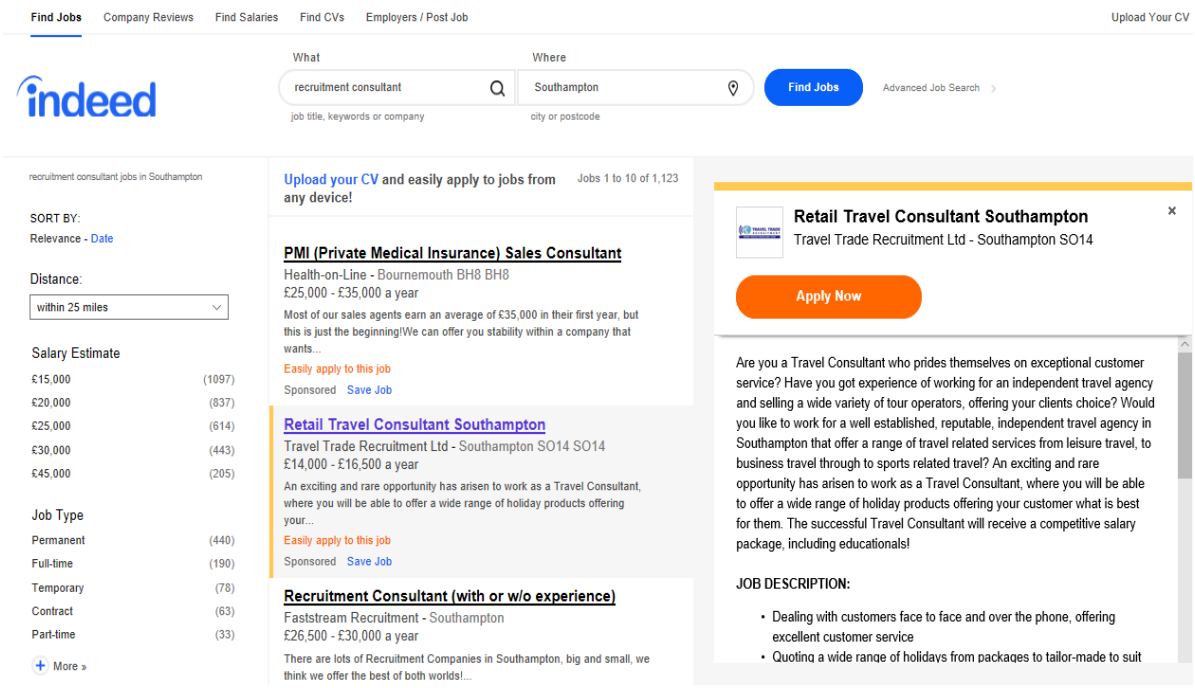

Рисунок 1.2 – Загальний вигляд інтерфейсного вікна сайту Indeed

Work.ua — це провідна українська онлайн-платформа для пошуку роботи, яка забезпечує користувачів доступом до великої кількості вакансій у різних галузях та регіонах України. Завдяки широкому вибору пропозицій, користувачі можуть знайти роботу як для початківців, так і для досвідчених фахівців.

Окрім базової функції пошуку вакансій, Work.ua надає користувачам додаткові можливості для розвитку кар'єри. На платформі розміщено корисні статті, рекомендації та поради, які стосуються створення резюме, проходження співбесід та побудови успішної кар'єри. Ці матеріали допомагають користувачам підвищувати свою конкурентоспроможність на ринку праці та досягати кар'єрних цілей [4]. Загальний вигляд інтерфейсного вікна сайту продемонстровано на рисунку 1.3.

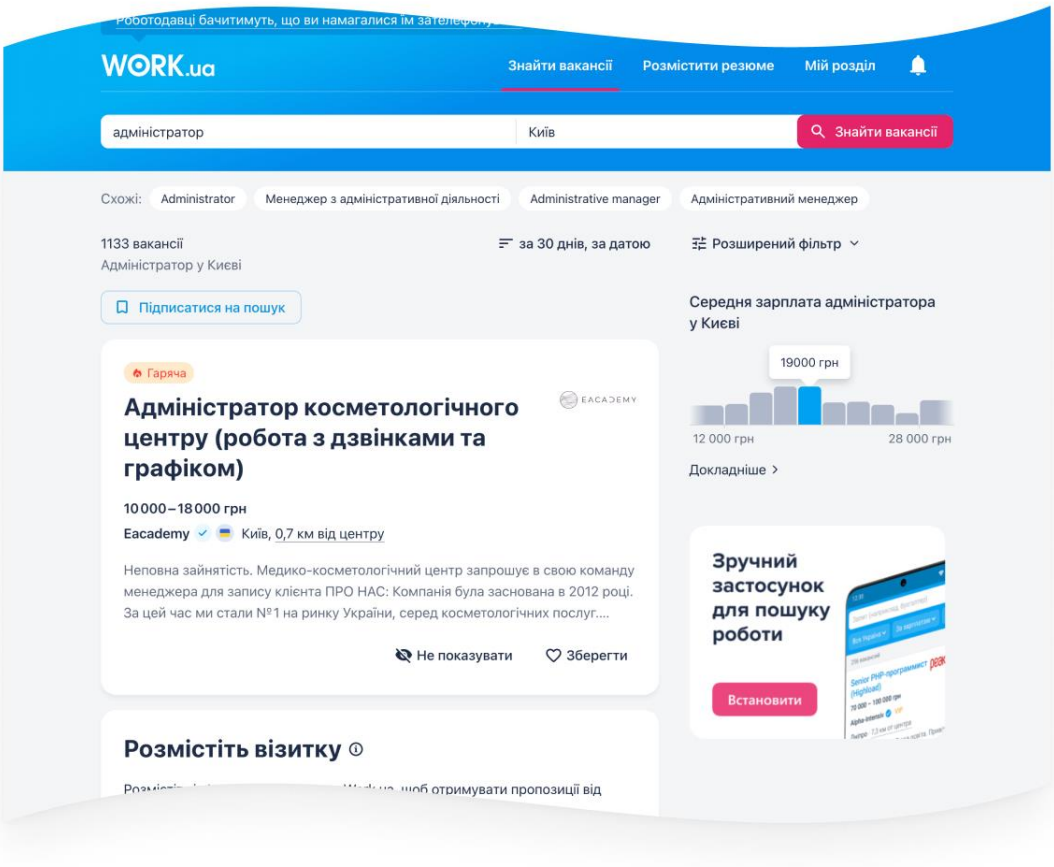

Рисунок 1.3 – Загальний вигляд інтерфейсного вікна сайту Work.ua

Robota.ua – це одна з найбільших українських онлайн-платформ для пошуку роботи, яка надає користувачам доступ до великої кількості вакансій у різних галузях [5]. Платформа пропонує зручний інтерфейс для пошуку роботи за регіоном, сферою діяльності та рівнем досвіду. Роботодавці мають змогу публікувати вакансії та переглядати резюме кандидатів. Robota.ua також надає корисні рекомендації та поради для претендентів, що допомагає ефективніше шукати роботу. Загальний вигляд інтерфейсного вікна сайту продемонстровано на рисунку 1.4

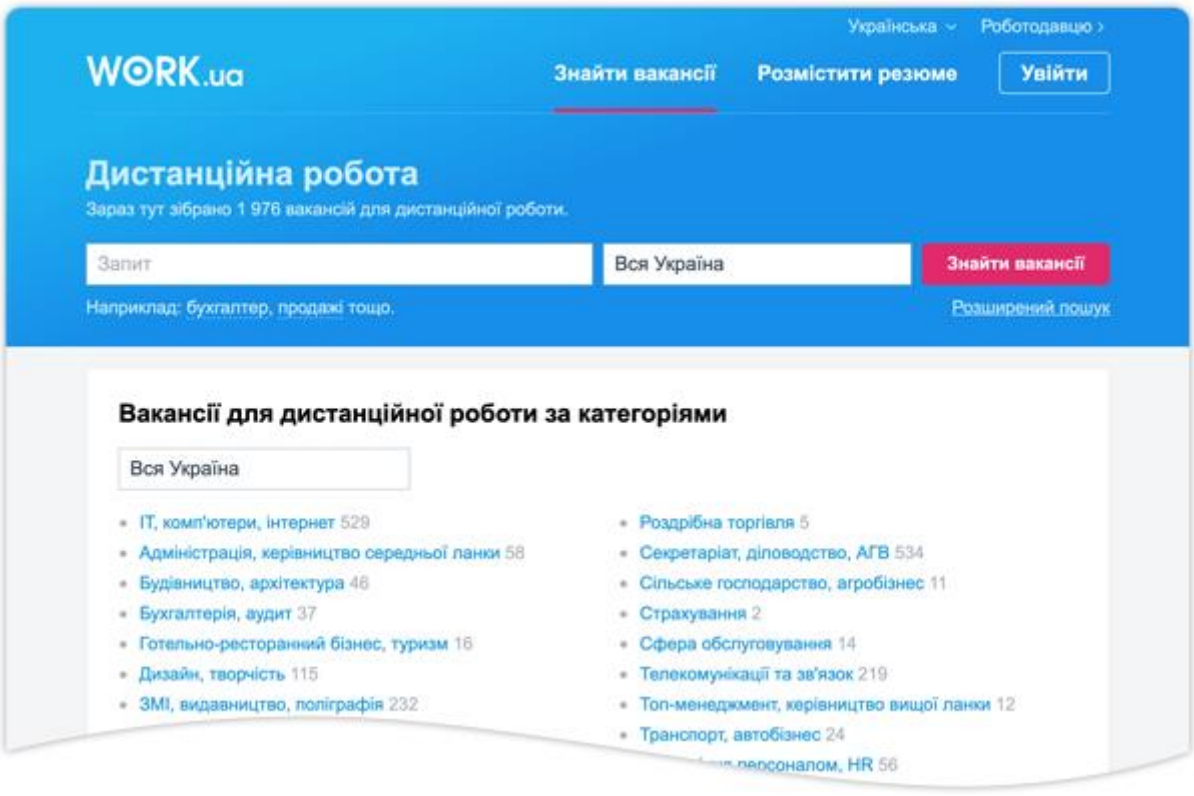

Рисунок 1.4 – Загальний вигляд інтерфейсного вікна сайту Robota.ua

Кожна з цих платформ має свої сильні та слабкі сторони, що впливають на зручність та ефективність їх використання. LinkedIn вважається однією з найбільш потужних і популярних платформ для професійного спілкування, налагодження ділових контактів і пошуку роботи. Вона дозволяє користувачам створювати детальні професійні профілі, приєднуватися до професійних спільнот і знаходити вакансії, що відповідають їхнім навичкам і кар'єрним амбіціям. Однак LinkedIn може бути менш корисною для пошуку роботи у спеціалізованих галузях або регіонах, де інші ресурси можуть бути більш релевантними. Також платформа переважно орієнтована на міжнародні ринки, що може не завжди бути оптимальним для локальних пошуків.

Indeed, на відміну від LinkedIn, агрегує вакансії з різних джерел, таких як корпоративні вебсайти, рекрутингові агентства та інші платформи. Це надає користувачам доступ до найбільшої бази вакансій. Однак через такий широкий

охоплення якість цих вакансій може значно варіюватися, і деякі оголошення можуть бути неактуальними або містити неповну інформацію.

Robota.ua є однією з найпопулярніших платформ для пошуку роботи в Україні. Вона пропонує користувачам зручний інтерфейс для пошуку вакансій за різними критеріями: за регіоном, спеціальністю, рівнем досвіду тощо. Robota.ua також забезпечує роботодавців можливістю публікувати вакансії та переглядати резюме кандидатів. Окрім того, платформа надає корисні поради для пошукачів роботи, допомагаючи їм краще підготуватися до співбесід та поліпшити свої шанси на працевлаштування. Проте Robota.ua переважно орієнтована на український ринок праці і не має такого широкого міжнародного охоплення, як LinkedIn чи Indeed, що може бути недоліком для тих, хто шукає роботу за кордоном.

Work.ua, також зосереджена на українському ринку, надає великий вибір вакансій у різних галузях і регіонах країни. Це робить її чудовим вибором для місцевого працевлаштування. Однак, аналогічно до Robota.ua, платформа не має міжнародної бази вакансій, що робить її менш привабливою для кандидатів, які розглядають можливість працювати за кордоном.

Незважаючи на різноманіття можливостей, які пропонують ці платформи, існує ряд загальних недоліків. Наприклад, багато з них не враховують усі важливі фактори, що впливають на успішний пошук роботи. Одним з таких факторів є особисті уподобання кандидата стосовно корпоративної культури або можливостей для професійного розвитку [6]. Багато платформ також не забезпечують гнучкість в урахуванні таких особливостей. Крім того, деякі веб-застосунки можуть бути недостатньо адаптовані для використання на мобільних пристроях, що може обмежувати їх доступність для тих, хто шукає роботу з телефону або планшета.

1.3 Аналіз об'єкту проектування технології пошуку роботи

Основною метою розробки інформаційної технології пошуку роботи є надання кваліфікованих рекомендацій і готових рішень для пошуку вакансій, які відповідають цілям та потребам користувача. Така технологія створює альтернативу дорогим послугам рекрутингових агенцій та кар'єрних консультантів.

Вимоги до інформаційної технології пошуку роботи:

- формалізація професійних навичок, досвіду роботи та освіти користувача;
- визначення відповідності навичок користувача до вимог вакансій;
- створення рейтингу вакансій з врахуванням навичок, досвіду роботи, освіти користувача та його побажань щодо зарплати, типу зайнятості та інших параметрів;
- база даних MS SQL Server для збереження інформації про основні сутності інформаційної технології;
- побудова Web API серверної частини за принципами REST для забезпечення продуктивності та масштабованості при взаємодії компонентів інформаційної технології.

Для побудови рекомендаційної системи щодо пошуку роботи необхідно враховувати велику кількість параметрів, що впливають на кінцевий результат. Кожен користувач має унікальний набір характеристик, таких як: вік, стать, досвід роботи, освіта, професійні навички, очікувана зарплата, бажаний тип зайнятості та інші. Врахування цих індивідуальних особливостей дозволяє створити персоналізовані рекомендації, які максимально відповідають потребам та очікуванням користувача.

У зв'язку з цим, розробка універсального алгоритму для підбору вакансій є складним завданням. Існує безліч підходів, але жоден з них не може гарантувати ідеальний результат для кожного користувача. Тому важливо враховувати експертні знання та досвід фахівців у галузі пошуку роботи та працевлаштування.

Поєднання експертних знань з аналізом даних та алгоритмами машинного навчання дозволить створити більш точні та ефективні рекомендації.

Існує кілька типових підходів до пошуку роботи:

- Самостійний пошук: користувач самостійно переглядає вакансії на різних сайтах та платформах, відправляє резюме та проходить співбесіди. Цей підхід вимагає багато часу та зусиль, але дозволяє користувачу мати повний контроль над процесом.
- Звернення до рекрутингових агенцій: користувач доручає пошук роботи професіоналам, які підбирають вакансії та допомагають з працевлаштуванням. Цей підхід може бути дорогим, але зазвичай ефективнішим за самостійний пошук.
- Використання онлайн-платформ та застосунків: користувач використовує спеціалізовані сервіси, які автоматизують та спрощують процес пошуку роботи. Цей підхід є відносно новим, але завдяки своїй зручності та ефективності, він стрімко набирає популярність.

1.4 Висновок до розділу 1

Отже, в першому розділі було виконано всебічний аналіз предметної області технологій пошуку роботи, яка стрімко розвивається та є актуальною через високу економічну привабливість і зростаючий попит на автоматизовані платформи для працевлаштування. Визначено, що розробка спеціалізованих застосунків і онлайн-сервісів є важливим напрямком у цій галузі, оскільки такі інструменти дозволяють значно спростити процес пошуку роботи, автоматизуючи його та підвищуючи ефективність для користувачів.

У рамках виконаного дослідження було здійснено детальний аналіз існуючих аналогів системи, який показав як переваги, так і численні недоліки, які слід враховувати при розробці нової технології. Серед основних недоліків було виявлено обмежену персоналізацію, що означає недостатнє врахування

індивідуальних особливостей і вимог користувачів, що призводить до менш точних рекомендацій. Крім того, багато платформ працюють із застарілими даними про вакансії, що знижує актуальність і якість наданих результатів. Окремо підкреслено, що системи часто мають слабку адаптацію до регіональних особливостей ринку праці, через що вони погано підходять для користувачів, які шукають роботу за межами свого регіону. Ще однією проблемою є складний або незручний інтерфейс, який ускладнює навігацію та використання системи.

Аналіз основних підходів до автоматизованого пошуку роботи дозволив виокремити ключові вимоги до майбутньої інформаційної технології. Зокрема, необхідно забезпечити адаптацію системи до індивідуальних характеристик користувачів, формування рейтингу вакансій за релевантністю, а також зручну інтеграцію з базами даних для оперативного оброблення вакансій і резюме. Також було наголошено на важливості впровадження сучасних технологій для обробки великих даних і машинного навчання.

Таким чином, результатом проведеного аналізу стало визначення цільових аспектів, які потрібно врахувати при створенні ефективного і конкурентоспроможного продукту.

2 МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ПОШУКУ РОБОТИ

2.1 Обґрунтування вибору підходів реалізації інформаційної технології пошуку роботи

Сучасні цифрові технології відкривають нові можливості для автоматизації процесу пошуку роботи, що дозволяє значно підвищити його ефективність та зручність для користувачів. З розвитком онлайн-платформ і розширенням ринку вакансій виникає потреба у створенні систем, здатних не лише надавати доступ до інформації, але й активно допомагати кандидатам у виборі найбільш відповідних робочих місць [7].

Попри існування багатьох цифрових рішень, головною задачею залишається визначення критеріїв, які забезпечать релевантність рекомендацій для конкретного користувача [8]. Важливо враховувати як професійні характеристики кандидатів, так і актуальні тенденції ринку праці, що дозволить забезпечити максимально точне співставлення запитів роботодавців та кваліфікацій кандидатів [9].

1) Одним із ключових підходів для побудови рекомендаційних систем у сфері пошуку роботи є аналіз профілю користувача і вимог до вакансій [10]. Цей підхід допомагає оцінювати відповідність між претендентами та наявними пропозиціями, використовуючи різні фактори: від професійних навичок до культурної сумісності з компанією. Врахування цих факторів дозволяє сформувати більш персоналізовані рекомендації, підвищуючи шанси кандидатів на успішне працевлаштування.

Попит на певні вакансії визначається низкою факторів, включаючи нові тенденції в економіці та зміну законодавчих умов. Рекомендаційні системи, які використовуються на платформах для пошуку роботи, аналізують ці чинники для поліпшення точності підбору відповідних вакансій для користувачів.

2) Оскільки успіх пошуку роботи значною мірою залежить від активності користувачів на ринку праці, одним із підходів покращення рекомендацій є кількісна оцінка цієї активності та аналіз впливу різних факторів на зацікавленість

кандидатів у певних вакансіях. Висновки про рівень інтересу до вакансій можна зробити, аналізуючи дані про те, як часто обговорюються певні професії або вакансії в соціальних мережах і на спеціалізованих платформах для пошуку роботи. Наприклад, одним із таких показників може бути кількість згадок вакансій у соціальних мережах або пошукових запитів у пошукових системах [11-15].

Переваги цього підходу:

- Дозволяє виявляти поточні тренди та популярні вакансії в режимі реального часу;
- Може бути інтегрований з іншими системами для побудови комплексних рекомендацій (наприклад, з системами відгуків чи автоматизованого відбору кандидатів);
- Забезпечує можливість швидкої адаптації до змін на ринку праці, наприклад, при появі нових спеціальностей чи зміні попиту на них.

Серед недоліків:

- Не завжди відображає якість вакансії або відповідність кандидата, оскільки кількість обговорень може бути штучно підвищена (наприклад, через маркетингові кампанії);
- Часто залежить від регіональних особливостей, що може спотворювати результати для міжнародних ринків;
- Дані можуть бути неповними або застарілими, якщо використовуються не всі доступні джерела інформації.

3) Системний підхід до аналізу ринку праці. Системний підхід є важливим підходом для вивчення ринку праці як комплексної системи, що взаємодіє з іншими елементами економіки [16]. Ринок праці функціонує як відкритий, динамічний механізм, де попит і пропозиція на робочі місця постійно змінюються в залежності від ринкових умов. У рамках цього підходу можна виділити ключові елементи, що визначають взаємодію між роботодавцями та кандидатами, а також впливають на ефективність пошуку роботи.

У системі ринку праці можна виокремити два основні компоненти: суб’єкти, які беруть участь у процесі найму (роботодавці та кандидати), і об’єкти управління,

що представляють вакансії та професійні навички. Розвиток ринку праці відбувається через децентралізовану взаємодію цих елементів, які формують єдину інформаційну екосистему.

4) Використання нейронних мереж для покращення рекомендацій. Ще одним ефективним підходом до прогнозування та рекомендацій у сфері пошуку роботи є використання штучних нейронних мереж (ШНМ) [17]. Ці моделі, які засновані на принципах роботи біологічних нейронних мереж, здатні аналізувати великі обсяги даних та виявляти складні закономірності. Концепція нейронних мереж виникла в результаті вивчення процесів, що відбуваються в людському мозку, і сьогодні активно застосовується в різних галузях, зокрема для автоматизації процесів прийняття рішень у сфері працевлаштування.

Штучні нейронні мережі складаються з багатьох пов'язаних простих одиниць, які називаються нейронами. Кожен нейрон обробляє інформацію, отриману від інших нейронів, і може передавати результати обробки далі по мережі. Незважаючи на свою простоту, ці мережі здатні виконувати складні завдання, такі як класифікація резюме, прогнозування успішності кандидатів на основі попередніх даних та адаптація рекомендацій на основі змінюваних умов ринку праці [18].

Завдяки використанню ШНМ, системи рекомендацій можуть стати більш точними та адаптивними, що дозволяє поліпшити шанси кандидатів на отримання роботи, яка найкраще відповідає їхнім навичкам та досвіду.

5) Традиційні методи аналізу та прогнозування в контексті ринку праці.

В аналізі та прогнозуванні ринку праці традиційні математичні моделі займають важливе місце. Ці методи зарекомендували себе у багатьох сферах, таких як прогнозування погоди, обсягів продажу та захворюваності [19]. Основна перевага використання математичних моделей полягає в їх об’єктивності: вони дозволяють уникнути впливу особистих думок і емоцій на процес прогнозування. Завдяки точним даним, які використовуються для навчання моделей, можна виявляти закономірності на ринку праці, що може бути складніше з іншими підходами, які спираються на важко систематизовані дані.

Однак традиційні методи, які грунтуються на припущеннях про стаціонарність, часто не можуть адекватно відображати швидкі зміни та нелінійні процеси, що характеризують сучасний ринок праці. У зв’язку з цим виникає потреба в нових моделях і методах, які здатні швидше реагувати на зміни.

Одним із таких новаторських підходів є використання часових рядів. Часовий ряд — це послідовність даних, які організовані за часом, зазвичай на рівновіддалених інтервалах. Аналіз часових рядів включає в себе виявлення характерних рис, підбір статистичних моделей та прогнозування різних показників. Часові ряди можуть бути стаціонарними, коли не спостерігається тенденція до зміни середнього значення чи дисперсії, або нестаціонарними, якщо такі зміни мають місце.

Характеристики часових рядів вказують на те, що спостереження, які відбуваються в послідовності, можуть бути взаємозалежними, зокрема у випадку близько розташованих даних. Інформативність кожного спостереження також може варіюватися в залежності від часу, адже дані, які віддалені у часі, можуть мати меншу цінність для аналізу. Зростання кількості спостережень не завжди призводить до пропорційного збільшення точності статистичних характеристик; інколи виникнення нових закономірностей може навіть знизити цю точність.

Аналіз часових рядів зазвичай починається з побудови графіка, який ілюструє їх поведінку. Спочатку необхідно перевірити стаціонарність даних. Для цього визначаються числові характеристики, такі як математичне сподівання, дисперсія та коваріація. Якщо часовий ряд є стаціонарним, він має постійні значення цих характеристик протягом часу. У разі, коли ряд виявляється нестаціонарним, важливо виділити його нестаціонарні компоненти. Процес виявлення трендів і інших факторів, що впливають на стаціонарність, може передбачати кілька етапів аналізу залишків, що виникають в результаті розрахунків з вихідного ряду або ж від перетворень.

6) Виділення трендів у аналізі ринку праці. Після перевірки часових рядів на стаціонарність важливо визначити наявність трендів. Тренд у цьому контексті —

це напрямок змін, який відображає загальну тенденцію розвитку ринку праці. Існує кілька типів трендів, які можуть бути важливими для прогнозування.

По-перше, існує тренд середнього, при якому часовий ряд коливається навколо повільно зростаючого або зменшуваного значення. Цей тренд відображає стабільні зміни в середньому показнику, наприклад, зростання попиту на певні професії.

По-друге, тренд дисперсії характеризує зміни амплітуди коливань даних з часом. У ринку праці це може означати, що зростання середнього попиту супроводжується підвищенням нестабільності в попиті на вакансії.

Третім типом є тренд автоковаріації, який вказує на зміну кореляції між поточними та попередніми значеннями показників. Цей аспект допомагає зрозуміти, наскільки вплив одних факторів на інші залишається постійним у часі.

Для перевірки наявності трендів часто використовують параметричні тести, такі як критерії Стьюдента та Фішера. Вони є ефективними лише за умови нормального розподілу даних. Щоб виключити тренд, спочатку потрібно визначити його модель. Якщо існує теоретичне обґрунтування природи тренду, його моделюють відповідно до цієї теорії. У багатьох випадках природа тренду залишається невідомою, і тоді використовують формальну модель, яка апроксимує тренд за допомогою лінійної комбінації поліномів, включаючи вільний член. Параметри трендової моделі визначаються методом найменших квадратів.

Часові ряди також можуть містити сезонні компоненти, які вказують на коливання навколо основного тренду. Для економічних показників сезонні коливання часто зумовлені періодами виробництва та споживання. Аналіз сезонності полягає в порівнянні даних за відповідні періоди, а не за попередні. Для виділення сезонних компонентів використовуються різні підходи, серед яких найпростішим є метод ковзного середнього, а також створення адитивних чи мультиплікативних моделей.

Окрему увагу слід приділити авторегресії, яка базується на регресійній залежності значень ряду від його попередніх значень. Авторегресивні моделі

широко застосовуються для прогнозування як стаціонарних, так і нестаціонарних даних, і мають безліч модифікацій.

Додатковим методом є індуктивний підхід, заснований на моделюванні часового тренду середньорічних значень вакансій у сфері праці. Цей підхід дозволяє виявити загальні закономірності в розвитку ринку праці та формувати рекомендації для кандидатів на основі змін у попиті [20].

Отже, новий підхід у створенні інформаційних систем для підтримки пошуку роботи, порівняно з існуючими, буде зосереджений на інтеграції соціальних факторів і поведінкових аспектів користувачів. Це дозволить значно підвищити точність і персоналізацію рекомендацій. Традиційні системи, як правило, орієнтовані на стандартні критерії, такі як досвід роботи та освіта, але не враховують індивідуальні переваги чи соціальне оточення користувача. Відмінність нового підходу полягає в тому, що він дає змогу аналізувати та враховувати не лише професійні характеристики, але й соціальні та культурні фактори, які мають суттєвий вплив на вибір кар'єрного шляху.

Крім того, застосування новітніх методів, таких як машинне навчання та аналіз великих даних, дозволить покращити процес надання рекомендацій, забезпечуючи більш ефективний і швидкий підбір вакансій. Це сприятиме більш точному відповідно до потреб ринку праці та індивідуальних уподобань користувачів. Завдяки цим перевагам, новий підхід буде здатний значно покращити якість пошуку роботи, надаючи користувачам не тільки актуальні, а й персоналізовані пропозиції.

2.2 Розробка математичної моделі надання рекомендацій

У сучасному світі інформаційні технології займають важливе місце в багатьох сферах, і пошук роботи не є винятком. Створення ефективної системи для пошуку роботи вимагає використання математичних моделей, які допоможуть аналізувати дані, виявляти закономірності та формувати рекомендації для

користувачів. Цей розділ присвячений обґрунтуванню вибору математичної моделі для технології пошуку роботи та надання рекомендацій, включаючи різні підходи та їх формули.

Математичні моделі дозволяють систематизувати інформацію про вакансії, кандидатів і ринок праці в цілому. Вони допомагають виявляти закономірності, що існують у даних, і прогнозувати результати на основі аналізу цих даних. Ключовими компонентами математичних моделей у технології пошуку роботи є:

- Аналіз даних: Збір і обробка великих обсягів даних, включаючи резюме, вакансії, відгуки користувачів і статистику.
- Виявлення закономірностей: Використання алгоритмів для виявлення патернів у даних, таких як популярні професії, навички, що затребувані на ринку, та вподобання користувачів.
- Прогнозування: Створення моделей, які здатні передбачити ймовірність того, що певний кандидат відповідає конкретній вакансії, або ж рекомендації щодо професій.

Однією з основних математичних моделей є фільтрація на основі контенту. Цей підхід ґрунтується на аналізі характеристик вакансій та профілів користувачів, що дозволяє надавати рекомендації на основі схожості між ними. Для реалізації цього підходу можна використовувати різні підходи, такі як TF-IDF (Term Frequency-Inverse Document Frequency) для визначення важливості термінів у вакансіях і профілях.

Формула TF-IDF дозволяє розрахувати значення для кожного терміна у документі:

$$\mathrm{T}F - IDF(t,d) = TF(t,d) * IDF(t), \tag{5.1}$$

де t – термін,

d – документ,

D – корпус документів,

$\mathrm{TF(t, d)}$ − (кількість входжень t в d) / (загальна кількість слів в d),

$\mathrm{IDF(t, D)}$ – (загальна кількість документів / кількість документів, що містять t).

Цей підрахунок дозволяє зрозуміти, які терміни є найбільш значущими для конкретної вакансії або профілю.

Після обчислення TF-IDF для кожного терміна можна використовувати косинусну схожість для оцінки подібності між профілями користувачів і вакансіями:

$$Cosine\ Similarity(A, B) = \frac{A * B}{||A|| * ||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}} \quad (5.2)$$

де A і B — це вектори, що представляють профіль користувача та вакансію відповідно.

Чим ближче значення косинусної схожості до 1, тим більша ймовірність, що користувач підійде для даної вакансії.

Колаборативна фільтрація є ще одним важливим підходом у технології пошуку роботи. Вона ґрунтується на використанні даних про дії інших користувачів для формування рекомендацій. Цей метод передбачає аналіз поведінки користувачів, таких як їх оцінки вакансій або вподобання, що дозволяє створювати персоналізовані рекомендації.

Одна з популярних технік колаборативної фільтрації — матрична факторизація, яка дозволяє виявити приховані фактори, що впливають на вподобання користувачів. Використовуючи сингулярне розкладення (SVD), ми можемо представити матрицю оцінок у вигляді:

$$R \approx U \Sigma V^T \quad (5.3)$$

де R — матриця оцінок,

U — матриця користувачів,

Σ — діагональна матриця сингулярних значень,

V^T — матриця вакансій.

Цей підхід дозволяє заповнити пропуски в даних, оцінюючи ймовірність того, що певний користувач буде зацікавлений у конкретній вакансії.

Гібридні моделі поєднують підходи фільтрації на основі контенту та колаборативної фільтрації. Це дозволяє використовувати переваги обох методів, щоб поліпшити точність рекомендацій. Однією з можливих формул для обчислення гібридної рекомендації може бути:

$$\text{Final Recommendation(i)} = \\ = \alpha \cdot \text{Content Similarity(i)} + \beta \cdot \text{Collaborative Similarity(i)} \quad (5.4)$$

де α і β — це ваги, які можна налаштувати в залежності від специфіки системи.

Цей підхід дозволяє забезпечити більш точні рекомендації, враховуючи індивідуальні вподобання користувачів і загальні тенденції.

Для інформаційної технології пошуку роботи обрані математичні моделі, що поєднують фільтрацію на основі контенту та колаборативну фільтрацію, оскільки вони дозволяють врахувати як характеристики вакансій, так і уподобання користувачів. Ці моделі можуть забезпечити високий рівень точності та адаптивності, що є критично важливим у динамічному середовищі ринку праці.

Використання комбінаційних моделей дозволяє також зменшити недоліки, пов'язані з кожним окремим підходом. Наприклад, фільтрація на основі контенту може мати проблеми з новими вакансіями, які не мають достатньої кількості даних, тоді як колаборативна фільтрація може працювати лише за умови наявності

достатньої кількості користувачів, які оцінюють вакансії. Гібридний підхід вирішує ці проблеми, поєднуючи інформацію з обох джерел.

Таким чином, вибір математичних моделей для технології пошуку роботи та надання рекомендацій базується на їх здатності адаптуватися до потреб користувачів і вимог ринку. Це забезпечить не лише точні рекомендації, але й підвищить загальну ефективність системи, допомагаючи користувачам знайти відповідні вакансії та зробити свій пошук роботи більш продуктивним.

Важливо зазначити, що використання гібридних математичних моделей не тільки покращує точність рекомендацій, але й сприяє створенню адаптивних систем, здатних враховувати зміни на ринку праці та особливості поведінки користувачів у реальному часі. Такий підхід дозволяє оперативно реагувати на нові виклики та тенденції, що виникають у результаті змін в економічній ситуації, технологічного прогресу або ж зміни законодавчих вимог у сфері працевлаштування.

Завдяки можливості комбінування різних джерел даних, таких як історія пошуку, відгуки інших користувачів і аналіз вакансій, система може створювати більш комплексні та персоналізовані рекомендації. Це не тільки підвищує рівень задоволення користувачів, але й сприяє ефективнішому використанню часу, адже система може надати відповідні рекомендації швидше і точніше, скорочуючи час на пошук.

У майбутньому перспективним напрямом розвитку може стати інтеграція моделей машинного навчання та штучного інтелекту для автоматичного аналізу нових вакансій та прогнозування потреб ринку. Це дозволить не тільки оптимізувати процес пошуку роботи, але й надавати більш далекоглядні поради щодо можливостей професійного зростання та розвитку кар'єри. Таким чином, технологія пошуку роботи еволюціонуватиме разом із вимогами ринку праці, забезпечуючи постійний прогрес та відповідність сучасним викликам.

Така система стає не лише інструментом для знаходження роботи, але й потужним механізмом для стратегічного планування кар'єри, що робить її незамінним помічником у динамічному світі праці.

2.3 Розробка UML-діаграми послідовності інформаційної технології пошуку роботи.

Щоб підвищити ефективність розробки та забезпечити глибоке розуміння роботи модуля інформаційної технології надання рекомендацій у процесі пошуку роботи, створимо UML-діаграму для цієї системи. Спершу розглянемо, що являє собою UML-діаграма.

UML (уніфікована мова моделювання) — це інструмент, який використовується для візуалізації функціонування системи. Його застосовують розробники для побудови чіткої концепції архітектури, структури коду, а також для ефективнішої реалізації складних систем. UML-діаграми також мають велике значення у моделюванні робочих та бізнес-процесів, що робить їх корисними в контексті пошуку роботи та надання персоналізованих рекомендацій.

Оскільки програмне забезпечення для надання рекомендацій може включати безліч функцій і значні обсяги коду, це ускладнює його аналіз та оптимізацію. UML-діаграма, створена для цієї системи, дозволяє структурувати інформацію та спростити її розуміння, представляючи ключові елементи в графічній формі. Використання UML-діаграм допомагає стандартизувати представлення моделі системи, що дозволяє чіткіше донести концептуальні ідеї щодо пошуку роботи та формування рекомендацій для користувачів.

Під час розробки програмного забезпечення може виникати безліч функцій та тисячі рядків коду, що ускладнює його аналіз і розуміння. UML-діаграми допомагають спростити це завдання, представляючи інформацію у візуальному форматі, який є більш зручним для сприйняття. Вони використовують стандартизовані підходи для створення моделі системи та відображення ключових ідей.

Окрім самого коду, UML-діаграми чудово підходять для візуалізації зв’язків та ієрархічних структур між компонентами, подібно до дерева рішень або блок-схеми, але орієнтованого саме на специфіку програмного забезпечення.

Існують два основні типи UML-діаграм: структурні та поведінкові:

1. Структурні діаграми візуалізують компоненти, які складають систему, і взаємозв’язок між ними, показуючи статичні аспекти системи.

2. Поведінкові діаграми представляють те, що відбувається всередині системи, включаючи те, як компоненти взаємодіють один з одним та з іншими системами або користувачами.

В рамках розгляду інформаційних технологій пошуку роботи, важливо звернути увагу на використання UML-діаграм для моделювання процесів та взаємодій у системі на етапі надання рекомендацій. Однією з важливих категорій діаграм є діаграми, що описують структуру та поведінку об'єктів у системі. Ці діаграми можуть бути розділені на два основних типи: структурні та поведінкові.

Структурні UML-діаграми, зокрема діаграма класів, є основою для об'єктно-орієнтованого проектування, оскільки вони надають візуальне уявлення про статичну структуру системи. Це дозволяє з'ясувати, як класи і їхні елементи взаємодіють один з одним, що є важливим при розробці програмного забезпечення для рекомендаційних систем у пошуку роботи. Діаграми залежностей допомагають організувати компоненти програмного забезпечення, показуючи, як вони взаємопов'язані між собою. Діаграми об'єктів дають змогу зобразити конкретні екземпляри об'єктів та їхні взаємозв'язки на певний момент часу. Окрім цього, діаграми компонентів описують внутрішню структуру класів, їхні частини та інтерфейси, що сприяє кращому розумінню, як окремі елементи програми взаємодіють у рамках більшої системи.

Що стосується поведінкових діаграм, вони спрямовані на вивчення динамічної частини системи. Діаграми діяльності, наприклад, використовуються для зображення послідовності дій в рамках бізнес-процесів, що дозволяє відслідковувати всі етапи взаємодії користувача з системою. Це є важливим для процесу надання рекомендацій у пошуку роботи, оскільки кожен крок пошуку чи

обробки вакансії має бути чітко зафіксований. Діаграми послідовності показують, як об'єкти взаємодіють з часом, і є ключовими для розуміння того, як функціонує процес рекомендацій в системі. Вони дозволяють уявити, як сервер, база даних і користувач обмінюються інформацією протягом роботи програми.

Діаграми зв'язку також фокусуються на взаємодії об'єктів, але забезпечують ширший погляд на співпрацю між ними, що може бути корисним для опису більш комплексних систем, наприклад, при обробці великої кількості даних у реальному часі. Завдяки цьому підходу можна визначити ключові взаємозв'язки між компонентами та виявити можливі точки оптимізації або вузькі місця в обробці даних. Це особливо важливо для систем, що працюють у режимі реального часу, де затримка в обробці даних може суттєво вплинути на якість роботи всієї системи.

Для опису роботи інформаційної технології, яка забезпечує надання рекомендацій в контексті пошуку роботи, вибрана UML-діаграма послідовності. Це дозволяє наочно відобразити всі етапи взаємодії користувача з системою, починаючи від запиту через веб-клієнт і завершуючи отриманням рекомендацій із бази даних.

Часові діаграми дозволяють враховувати обмеження за часом і продуктивність операцій системи, що важливо для оптимізації рекомендаційних алгоритмів. У рамках цієї діаграми будуть представлені основні компоненти: веб-клієнт, сервер, постачальник даних та база даних, що дозволяє зрозуміти повний цикл дій користувача та обробки даних на кожному етапі. Такий підхід дає змогу чітко уявити всі етапи процесу, що є ключовим для розробки ефективної системи рекомендацій у сфері пошуку роботи. Розроблена UML-діаграма послідовностей зображена на рисунку 2.1 та 2.2.

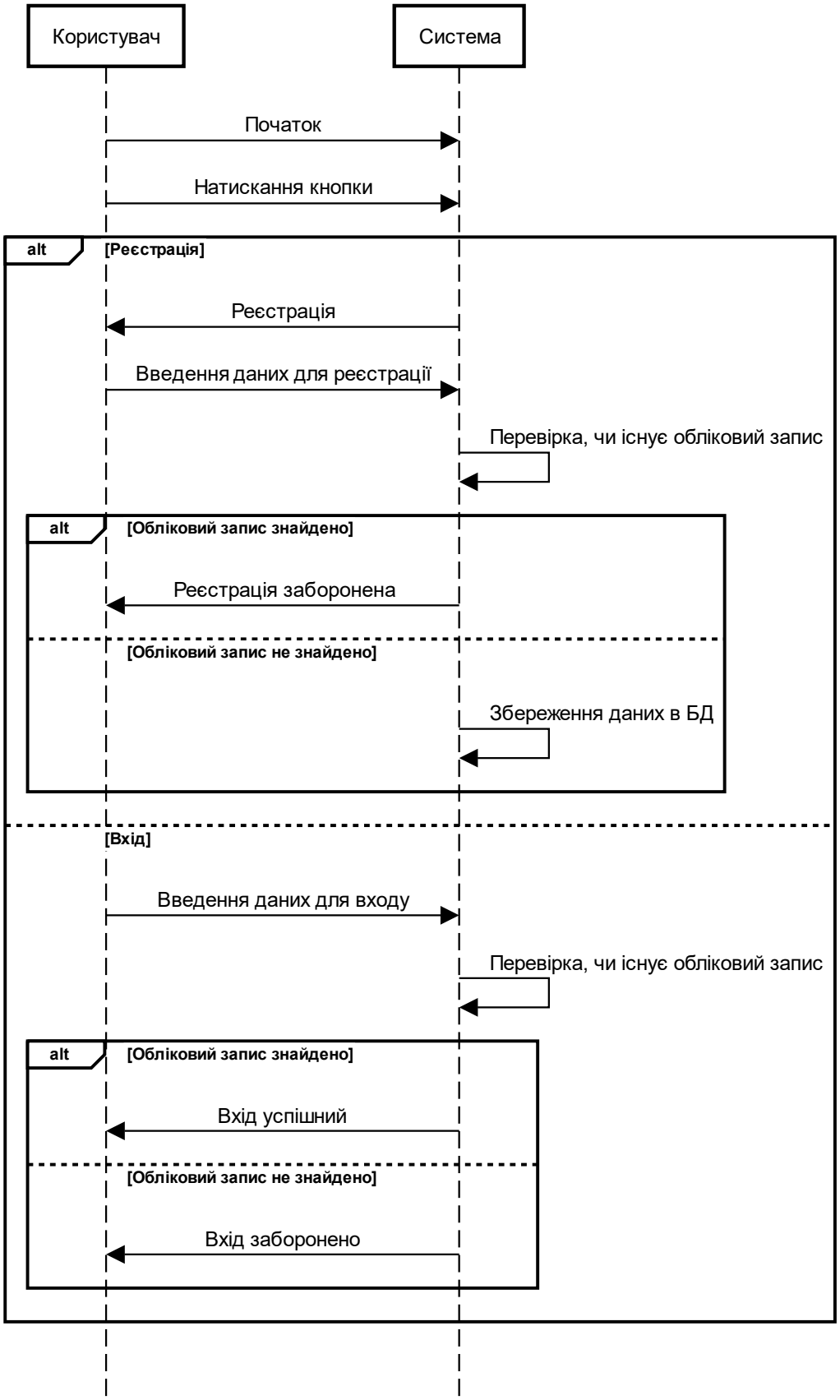

Рисунок 2.1 – UML-діаграма послідовності взаємодії модулів інформаційної технології для реєстрації або входу в систему

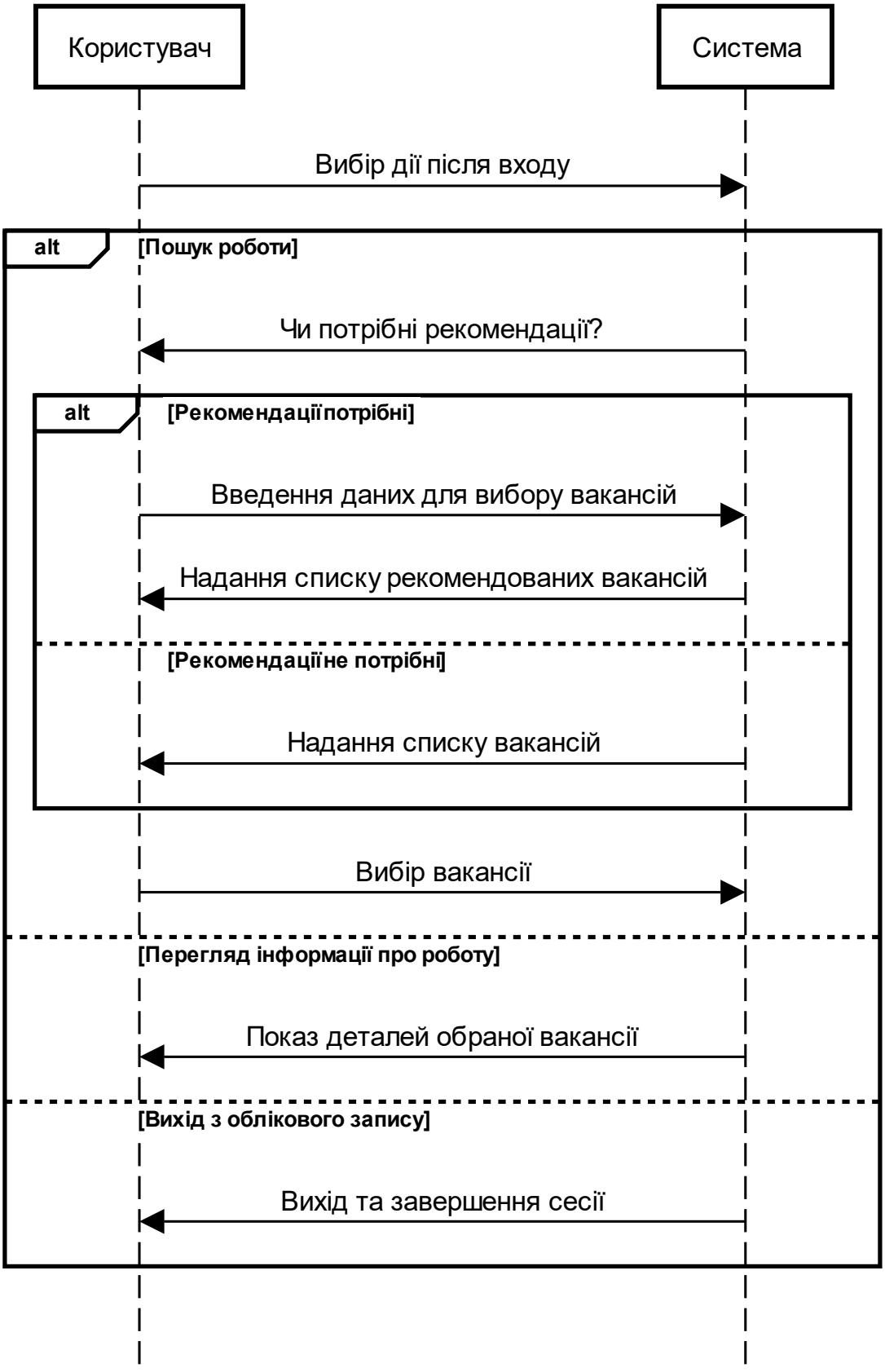

Рисунок 2.2 – UML-діаграма послідовності взаємодії з модулем надання рекомендацій

2.4 Розробка структури інформаційної технології для надання рекомендацій щодо пошуку роботи

На вхід системи надходить інформація про кандидатів, включаючи їхній досвід роботи, професійні навички, бажані умови працевлаштування, а також дані про актуальні вакансії. Для забезпечення високої якості рекомендацій важливо, щоб отримані дані були детальними й актуальними.

Першим етапом є стандартизація вхідних даних. Досвід, навички та інші особливості кожного кандидата приводяться до уніфікованого формату, який відповідає вимогам нейронної мережі. Для кожного значення проводиться масштабування, щоб привести їх до діапазону, прийнятного для алгоритма.

Після підготовки даних, інформація надходить до нейронної мережі, яка аналізує профіль користувача та співставляє його з вимогами роботодавців. На етапі обробки здійснюється відбір найбільш релевантних вакансій, орієнтуючись на подібність навичок, відповідність досвіду, а також особисті вподобання кандидата щодо умов роботи. Для цього мережа застосовує класифікаційні алгоритми, які допомагають структурувати дані та виділяти пріоритетні вакансії.

Після класифікації мережа формує рекомендації, враховуючи актуальні ринкові тенденції та популярність вакансій у певному регіоні або галузі. На основі цього генерується список рекомендацій, який подається користувачеві у зручному форматі з можливістю сортування та фільтрації.

Завершальним етапом є інтерактивний інтерфейс для перегляда і взаємодії з отриманими рекомендаціями, що дає користувачеві можливість обирати вакансії, які найбільше відповідають його очікуванням.

Рисунок 2.3 – Структура інформаційної технології надання рекомендацій для пошуку роботи

2.5 Висновок до розділу 2

Інформаційні технології для пошуку роботи на сьогодні відіграють значну роль у прискоренні та спрощенні процесу працевлаштування. Обґрунтування вибору підходів реалізації технології показує важливість комплексного підходу до аналізу ринку праці, професійних профілів кандидатів та запитів роботодавців. Підходи, такі як профілювання користувача, системний аналіз, нейронні мережі, традиційні підходи математичного моделювання та аналіз часових рядів, доповнюють одне одного та забезпечують всебічний огляд і можливість налаштування рекомендацій з урахуванням індивідуальних особливостей кандидатів і тенденцій ринку праці.

Кожен із цих підходів має свої сильні сторони. Аналіз профілю користувача дозволяє забезпечити персоналізацію, яка сприяє ефективному співставленню кандидатів із вакансіями, підвищуючи ймовірність успішного працевлаштування. Використання системного підходу дозволяє врахувати взаємодію між різними елементами ринку праці, що є важливим для створення більш узагальнених моделей, здатних відображати динамічні зміни. Нейронні мережі, завдяки своїм можливостям аналізувати великі обсяги даних, здатні виявляти глибокі закономірності та адаптуватися до змінюваних умов ринку, що підвищує точність рекомендацій.

Водночас, застосування традиційних математичних моделей і підходів аналізу часових рядів забезпечує високу об’єктивність і надійність, але потребує постійного оновлення даних для ефективного прогнозування. Аналіз часових рядів дозволяє виділяти сезонні та трендові компоненти, що дає змогу точніше оцінювати поточні ринкові тенденції й адаптувати систему до їхніх змін.

Таким чином, моделювання інформаційної технології для пошуку роботи є багатовимірним завданням, що поєднує сучасні підходи та інструменти. Комплексне використання цих підходів дозволяє створити ефективні системи, здатні не лише надавати релевантні рекомендації кандидатам, але й оперативно реагувати на нові виклики ринку.

3 ОБГРУНТУВАННЯ ВИБОРУ ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

3.1 Обгрунтування вибору архітектури інформаційної технології пошуку роботи

Правильне проєктування архітектури системи — це основа для її ефективної роботи. Важливо забезпечити не лише надійну та масштабовану взаємодію між компонентами, але й захист даних користувачів, особливо у контексті персональних рекомендацій під час пошуку роботи [21].

Обрана архітектура для системи надання рекомендацій під час пошуку роботи базується на клієнт-серверній моделі [22]. У цій архітектурі клієнт (веб-додаток або мобільний додаток) взаємодіє з сервером, який обробляє запити та керує базою даних. Ключовим аспектом є безпечна передача даних між клієнтом і сервером, оскільки система працює з чутливою інформацією, як-от особисті дані користувачів, їх професійні профілі та резюме.

Для забезпечення захищеного обміну інформацією між клієнтом і сервером використовується HTTPS (Hypertext Transfer Protocol Secure) — це розширена версія протоколу HTTP, що забезпечує шифрування даних за допомогою протоколу SSL/TLS (Secure Sockets Layer/Transport Layer Security) [23]. HTTPS дозволяє захистити дані від перехоплення або модифікації під час передачі через мережу Інтернет.

Обмін даними через HTTPS включає кілька важливих етапів:

1. Встановлення з'єднання: Коли клієнт ініціює запит до сервера через HTTPS, першим кроком є встановлення захищеного з'єднання. Для цього використовується процес "рукостискання" TLS (TLS handshake), який включає кілька кроків обміну даними між клієнтом і сервером.

2. Обмін сертифікатами: Сервер надає клієнту свій цифровий сертифікат, який підтверджує, що сервер є надійним та автентичним. Цей сертифікат

підписаний довіреним центром сертифікації (CA), і клієнт перевіряє його справжність. Сертифікати містять відкритий ключ сервера, який використовується для шифрування даних.

3. Генерація ключів: Після того, як клієнт перевірив сертифікат сервера, обидві сторони (клієнт і сервер) узгоджують сесіонні ключі шифрування. Це робиться через обмін зашифрованими повідомленнями, що використовують асиметричне шифрування з відкритим і закритим ключем. Далі використовується симетричне шифрування для більш ефективної передачі даних під час сесії.

4. Шифрування даних: Усі дані, які передаються між клієнтом і сервером, шифруються за допомогою симетричного ключа, що гарантує, що будь-яка інформація, яка передається через Інтернет, буде захищена від перехоплення.

5. Захист від підробок: Крім шифрування даних, HTTPS забезпечує захист від спроб зміни або підробки даних під час їх передачі. Якщо хтось спробує змінити передані дані, з'єднання буде автоматично закрито, що робить атаку неефективною.

Використання HTTPS забезпечує кілька важливих переваг для інформаційної системи пошуку роботи:

- Конфіденційність: Персональні дані користувачів, зокрема резюме, історія пошуку роботи та професійні профілі, будуть зашифровані під час передачі, що запобігає їх перехопленню сторонніми особами.

- Інтегритет даних: Шифрування також гарантує, що передані дані не можуть бути змінені під час передачі без виявлення. Це важливо для забезпечення цілісності інформації про вакансії та кандидатів.

- Автентифікація: Завдяки перевірці сертифіката серверу, користувачі можуть бути впевнені, що вони взаємодіють з офіційною системою пошуку роботи, а не з фальшивим ресурсом.

Клієнтська частина системи відповідає за збирання інформації від користувача, включаючи пошукові запити, дані профілю та резюме. Ця інформація передається на сервер через захищені запити HTTPS. Сервер обробляє ці дані, виконує пошук відповідних вакансій або аналізує профіль користувача для надання

рекомендацій. Після обробки сервер надсилає результати на клієнтську частину, де користувач може переглянути вакансії або отримати персоналізовані пропозиції.

Крім безпеки, важливим аспектом клієнт-серверної архітектури є можливість обробки великої кількості запитів одночасно. Сервери можуть обробляти паралельні запити від різних користувачів завдяки розподіленій архітектурі. При зростанні кількості користувачів система може бути легко масштабована шляхом додавання додаткових серверів, що забезпечує стійкість до великих навантажень.

3.2 Розробка схеми алгоритму інформаційної технології надання рекомендацій для пошуку роботи

Для реалізації інформаційної технології надання рекомендацій для пошуку роботи було розроблено схему алгоритму її роботи. Ця схема відображає основні етапи взаємодії користувача з системою та логіку обробки даних для надання персоналізованих рекомендацій.

Опишемо наведений алгоритм надання рекомендацій для пошуку роботи (рис. 4.1).

1. Користувач починає процес (початок).
2. Натискає кнопку для взаємодії з системою.
2. Система перевіряє, яку кнопку натиснув користувач: реєстрація чи вхід. Якщо обрано реєстрацію, перехід до кроку 4; якщо вхід – перехід до кроку 9.
3. Якщо обрано реєстрацію, відбувається процес реєстрації (перехід до кроку 5).
4. Після реєстрації користувач вводить дані для створення облікового запису (перехід до кроку 6).
5. Система перевіряє, чи вже існує обліковий запис (перехід до кроку 7).
6. Якщо запис знайдено, користувач не може повторно зареєструватися (перехід до кроку 3). Якщо запис не знайдено, перехід до кроку 8.

7. Якщо облікового запису немає, дані реєстрації зберігаються в базі даних (БД) (перехід до кроку 3).

8. Якщо обрана кнопка входу, користувач вводить дані для входу (перехід до кроку 10).

9. Вводяться дані для входу (перехід до кроку 11).

10. Система перевіряє, чи існує обліковий запис з введеними даними. Якщо запис знайдено, перехід до кроку 12, якщо ні – перехід до кроку 3.

11. Якщо обліковий запис знайдено, користувач потрапляє на головну сторінку (перехід до кроку 13).

12. Користувач обирає, яку дію виконати: пошук роботи (перехід до кроку 14), перегляд інформації про роботу (перехід до кроку 20) або вихід з облікового запису (перехід до кроку 22).

13. Якщо обрано пошук роботи, система продовжує роботу в цьому напрямку (перехід до кроку 15).

14. Система запитує, чи необхідно надати рекомендації для пошуку роботи. Якщо так, перехід до кроку 17, якщо ні – перехід до кроку 16.

15. Система видає список вакансій з бази даних (перехід до кроку 19).

16. Користувач вводить додаткові дані для вибору відповідних вакансій (перехід до кроку 18).

17. Система надає список рекомендованих вакансій (перехід до кроку 19).

18. Користувач обирає бажану вакансію зі списку (перехід до кроку 13).

19. Якщо обрано перегляд інформації про роботу, система надає деталі обраної вакансії (перехід до кроку 21).

20. Виводиться інформація про вибрану вакансію (перехід до кроку 13).

21. Якщо обрано вихід з облікового запису, користувач виходить із системи (перехід до кроку 23).

22. Кінець процесу.

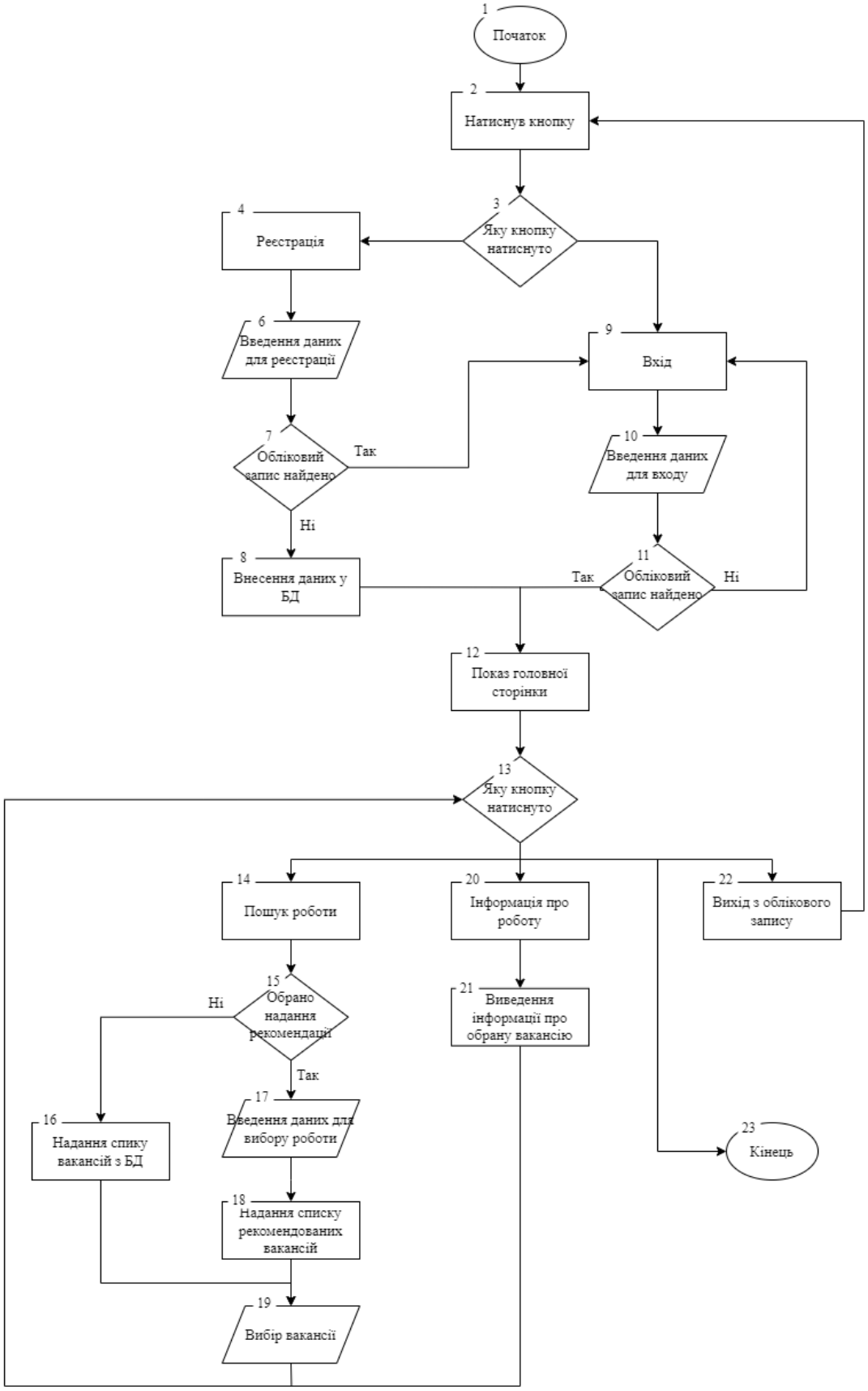

Рисунок 3.1 – Схема алгоритму роботи інформаційної технології надання рекомендацій для пошуку роботи

3.3 Обгрунтування вибору мови програмування

Вибір мови програмування є важливим етапом під час проєктування будь-якої інформаційної системи. Кожна мова має свої сильні сторони та слабкості, і вибір залежить від багатьох факторів, таких як продуктивність, масштабованість, безпека, гнучкість, підтримка, екосистема та відповідність конкретним вимогам проєкту. Для створення інформаційної технології пошуку роботи необхідно вибрати таку мову програмування, яка забезпечить швидку обробку даних, безпечне передавання інформації, а також зручність у подальшому обслуговуванні та розширенні системи.

У цьому розділі буде детально проаналізовано мови програмування, які можуть бути використані для реалізації системи, та обґрунтовано вибір TypeScript та NestJS як основних інструментів для розробки інформаційної технології пошуку роботи.

JavaScript — це одна з найпоширеніших мов програмування, яка спочатку використовувалась для створення інтерактивних елементів на веб-сторінках. З появою платформи Node.js JavaScript став основою для серверної розробки, що дозволило розробникам використовувати одну мову як для клієнтської, так і для серверної частини додатків [24].

Переваги JavaScript:

23. Популярність і підтримка. JavaScript — це мова програмування з величезною екосистемою. Існує безліч бібліотек і фреймворків, таких як React, Angular, Vue.js для клієнтської частини і Node.js для серверної.

24. Використання на клієнті і сервері. Завдяки Node.js JavaScript можна використовувати для обох частин додатка, що спрощує розробку та знижує поріг входження для нових розробників.

25. Асинхронна обробка. JavaScript чудово працює з асинхронними операціями, що дозволяє ефективно обробляти паралельні запити, що є важливим для систем із великим навантаженням.

Недоліки JavaScript:

1. Відсутність статичної типізації. Найбільшою слабкістю JavaScript є відсутність статичної типізації. Це може призвести до помилок під час виконання, особливо в великих системах, де важливо контролювати типи даних.

2. Ускладнена підтримка великих проєктів. Через динамічну природу мови підтримка великих і складних систем може стати викликом. Проблеми виникають через неточне використання типів або через складність відстеження залежностей між компонентами системи.

3. Безпека. Через свою гнучкість JavaScript схильний до певних вразливостей, таких як XSS (міжсайтовий скриптинг). Хоча існують бібліотеки та методи для боротьби з цими проблемами, необхідність додаткових заходів безпеки збільшує складність.

JavaScript є потужним інструментом для розробки веб-додатків, але його відсутність статичної типізації може стати проблемою в масштабних і складних системах, таких як інформаційна технологія пошуку роботи.

TypeScript — це надбудова над JavaScript, яка додає статичну типізацію та інші сучасні функції. Він був створений для усунення багатьох недоліків JavaScript, забезпечуючи кращу підтримку великих систем і складних проєктів [25].

Переваги TypeScript:

1. Статична типізація. Однією з ключових переваг TypeScript є статична типізація, яка дозволяє виявляти помилки під час компіляції, а не під час виконання. Це значно покращує стабільність системи та знижує кількість багів. Для інформаційної технології пошуку роботи, де дані користувачів і вакансій постійно змінюються, можливість контролювати типи є критично важливою.

2. Покращена підтримка великих проєктів. Завдяки суворій типізації та потужним інструментам для організації коду (інтерфейси, типи, модулі), TypeScript полегшує підтримку й масштабування великих систем. Це дає змогу створювати складні системи, розбивати їх на модулі, і підтримувати їх у довгостроковій перспективі.

3. Розширена підтримка екосистеми JavaScript. TypeScript повністю сумісний з JavaScript, що означає можливість використовувати всі бібліотеки JavaScript. Це забезпечує доступ до величезної екосистеми, включаючи популярні фреймворки для фронтенду (React, Angular) та бекенду (Node.js, NestJS).

4. Краща продуктивність розробки. Завдяки інструментам для автоматичної перевірки коду та функціям автозаповнення, TypeScript допомагає розробникам писати код швидше та з меншою кількістю помилок. Це особливо важливо в системах, де важлива висока продуктивність, як у випадку системи пошуку роботи, де велика кількість користувачів постійно взаємодіє з системою.

5. Масштабованість і безпека. Типізація дозволяє створювати безпечніші системи, де всі можливі сценарії взаємодії з даними передбачені заздалегідь. Для системи, яка працює з персональними даними та надає рекомендації, це є критично важливим аспектом.

Недоліки TypeScript:

1. Час на компіляцію. Оскільки TypeScript потребує компіляції в JavaScript, це додає певний час у процесі розробки. Однак цей недолік компенсується перевагами стабільного та захищеного коду.

2. Крива навчання. Для розробників, які не мають досвіду з типізацією, перехід з JavaScript на TypeScript може потребувати певного часу на навчання. Проте з часом TypeScript полегшує підтримку проєктів і знижує кількість помилок.

TypeScript є оптимальним вибором для інформаційної технології пошуку роботи завдяки своїй статичній типізації, що забезпечує безпеку, гнучкість та ефективність під час роботи з великими системами. Це дозволяє краще контролювати код, забезпечуючи надійність і можливість масштабування.

Python — це мова програмування, яка стала популярною завдяки своїй простоті та широким можливостям. Python використовується для різних завдань — від наукових досліджень і автоматизації до веб-розробки. Завдяки фреймворкам, таким як Django та Flask, Python також поширений у серверній розробці [26].

Переваги Python:

1. Простота і зручність. Python має простий і зрозумілий синтаксис, що робить його легким для вивчення та використання. Це дозволяє швидко почати розробку і створювати прототипи нових функцій. У системі пошуку роботи Python міг би використовуватися для швидкого створення алгоритмів обробки даних або інструментів аналізу вакансій.

2. Широка підтримка наукових бібліотек. Python має величезну кількість бібліотек для наукових обчислень, машинного навчання та аналізу даних, таких як NumPy, Pandas, TensorFlow тощо. Це робить його хорошим вибором для реалізації складних алгоритмів аналізу даних або побудови рекомендаційних систем.

3. Активна спільнота. Python має дуже активну спільноту розробників, що забезпечує швидке вирішення проблем і постійне оновлення інструментів.

Недоліки Python:

1. Продуктивність. Python є інтерпретованою мовою, що робить його менш продуктивним у порівнянні з мовами, такими як TypeScript або Java. Це може стати проблемою для великих систем із високим навантаженням, де швидкість обробки запитів має вирішальне значення.

2. Обмеженість для масштабних веб-додатків. Незважаючи на наявність фреймворків для веб-розробки, Python зазвичай не вважається оптимальним вибором для створення великих масштабованих серверних систем. Він не забезпечує тієї ж продуктивності та стабільності, що TypeScript або Java для обробки великої кількості запитів.

3. Глобальна інтерпретація. Через глобальну інтерпретацію Python складно ефективно працювати в багатопоточних середовищах, що також може бути недоліком для системи з великою кількістю користувачів.

Python є чудовим вибором для розробки алгоритмів аналізу даних або прототипування, але в контексті побудови великої і масштабован ої інформаційної системи пошуку роботи, Python може зіткнутися з обмеженнями продуктивності та масштабованості.

C++ — це мова програмування загального призначення, яка є розширенням мови C і підтримує об'єктно-орієнтоване програмування. C++ широко

використовується для створення системного програмного забезпечення, ігор, драйверів, а також для розробки програм, що вимагають високої продуктивності [27].

Переваги C++:

1. Продуктивність. C++ є компільованою мовою, що дозволяє отримувати високу швидкість виконання програм. Це робить її ідеальною для систем, які вимагають інтенсивних обчислень, таких як ігри або системи, що працюють з великими обсягами даних.

2. Контроль над ресурсами. C++ надає розробникам великий контроль над системними ресурсами, такими як пам'ять і процесорний час. Це важливо для програм, що вимагають оптимізації продуктивності.

3. Об'єктно-орієнтоване програмування. C++ підтримує об'єктно-орієнтоване програмування, що дозволяє структурувати код і повторно використовувати його. Це може бути корисно для створення модульних систем.

Недоліки C++:

1. Складність синтаксису. C++ має складніший синтаксис у порівнянні з іншими мовами, такими як Python або JavaScript. Це може ускладнити навчання та підвищити ймовірність помилок у коді.

2. Управління пам’яттю. C++ вимагає ручного управління пам’яттю, що може призвести до проблем, таких як витоки пам’яті або помилки сегментації. Це ускладнює розробку і може збільшити час, витрачений на налагодження.

3. Не так добре підходить для веб-розробки. C++ не є традиційним вибором для веб-розробки, що може ускладнити інтеграцію з популярними фреймворками для веб-додатків.

C# — це мова програмування, розроблена компанією Microsoft, яка є частиною платформи .NET. C# є об'єктно-орієнтованою мовою, що дозволяє створювати різноманітні додатки, включаючи веб-додатки, настільні програми та ігри [28].

Переваги C#:

1. Висока продуктивність. C# компілюється в байт-код, що виконується в середовищі .NET, що забезпечує високу швидкість виконання. Це дозволяє розробляти масштабовані й ефективні серверні додатки.

2. Розвинена екосистема. C# має потужну екосистему, що включає фреймворки, такі як ASP.NET для веб-розробки. Це дозволяє швидко створювати надійні веб-додатки з багатим функціоналом.

3. Сучасні можливості. C# підтримує сучасні можливості програмування, такі як асинхронне програмування, LINQ (Language Integrated Query) для запитів до даних та потужні інструменти для обробки помилок.

Недоліки C#:

1. Платформна залежність. Хоча C# вже підтримується на різних платформах через .NET Core, більшість застосунків традиційно прив'язані до екосистеми Windows, що може обмежити вибір хостингу та розгортання.

2. Час навчання. C# може бути складним для новачків, особливо через свою об'єктно-орієнтовану природу. Це може вимагати більше часу на освоєння для тих, хто звик до більш простих мов, таких як Python.

3. Вартість. Хоча C# сам по собі безкоштовний, використання комерційних продуктів Microsoft (наприклад, Visual Studio) може бути дорогим, що робить його менш привабливим для стартапів або невеликих проектів.

PHP — це мова програмування, спеціально розроблена для веб-розробки. Вона використовується для створення динамічних веб-сайтів і веб-додатків. PHP є однією з найпопулярніших мов для серверної розробки, завдяки її простоті та швидкості [29].

Переваги PHP:

1. Простота використання. PHP має простий синтаксис, що робить його легким для вивчення. Це дозволяє швидко створювати прототипи веб-додатків, що особливо корисно для стартапів і малих підприємств.

2. Велика спільнота та екосистема. PHP має величезну спільноту розробників і безліч бібліотек і фреймворків, таких як Laravel і Symfony, що дозволяє швидко реалізовувати різні функції.

3. Широке використання. PHP використовується на багатьох веб-сайтах і платформах, включаючи WordPress, що робить його надійним вибором для веб-розробки.

Недоліки PHP:

1. Обмежена продуктивність. PHP може не забезпечити таку ж продуктивність, як інші мови, такі як Java або C#. Це може бути проблемою для великих систем, де продуктивність є критично важливою.

2. Старі фреймворки. Хоча PHP має багато фреймворків, деякі з них можуть бути застарілими або менш гнучкими в порівнянні з сучасними рішеннями, такими як NestJS або ASP.NET.

3. Безпека. PHP часто піддається критиці за свою вразливість до атак, таких як SQL-ін'єкції або XSS. Розробники повинні дотримуватися суворих стандартів безпеки, щоб запобігти цим загрозам.

При розробці інформаційної системи пошуку роботи вибір фреймворку є критично важливим етапом, оскільки він впливає на архітектуру програми, швидкість розробки, продуктивність системи та можливості її подальшого розвитку. У цьому розділі розглянемо декілька популярних фреймворків, які можуть бути використані для створення таких систем, зокрема NestJS, Express.js, Django, Ruby on Rails та Flask.

NestJS — це прогресивний фреймворк для створення серверних додатків, який використовує TypeScript і Node.js. NestJS орієнтований на модульність, що дозволяє розробникам структурувати код у логічні компоненти [30].

Основні переваги NestJS:

1. Модульна архітектура: NestJS дозволяє легко організувати код, розбиваючи його на модулі. Це забезпечує зручність у підтримці та масштабуванні додатків, що особливо важливо для складних систем.

2. Інжекція залежностей: NestJS підтримує інжекцію залежностей, що спрощує управління компонентами та їх взаємодію, дозволяючи створювати чистий та тестований код.

3. Підтримка RESTful і GraphQL API: Фреймворк дозволяє легко створювати як RESTful API, так і GraphQL, що робить його гнучким для інтеграцій з клієнтською частиною системи.

4. Потужна система аутентифікації: NestJS забезпечує можливості для інтеграції різних механізмів аутентифікації, що критично важливо для систем, що працюють із чутливими даними.

Недоліки NestJS:

1. Крива навчання: Для новачків, які не мають досвіду роботи з TypeScript або Node.js, може знадобитися деякий час для освоєння концепцій і структур.

2. Відносна складність конфігурації: NestJS має високо конфігуровану структуру, яка може здатися складною для тих, хто звик до мінімалістичних фреймворків, таких як Express.js. Важливо налаштувати багато аспектів вручну, що збільшує час розгортання проєктів.

3. Надлишковість функцій: Для невеликих або простих проєктів NestJS може бути надмірно потужним інструментом. Використання всіх модулів і компонентів фреймворку може бути непотрібним для базових завдань, що призводить до збільшення складності без очевидних переваг.

Express.js — це легкий фреймворк для Node.js, який надає базові функції для створення веб-додатків і API. Він є мінімалістичним, що робить його простим у використанні [31].

Переваги Express.js:

1. Простота та легкість у використанні: Express має простий синтаксис, що дозволяє швидко створювати додатки без значних зусиль.

2. Гнучкість: Розробники можуть додавати лише ті функції, які їм потрібні, що робить Express дуже гнучким для різних проєктів.

3. Велика спільнота: Оскільки Express є популярним фреймворком, розробники можуть знайти безліч ресурсів і підтримки.

Недоліки Express.js:

1. Відсутність строгої структури: У великих проєктах це може призвести до безладності в коді і ускладнити його підтримку.

2. Можливість повторного кодування: Через легкість використання можуть виникати проблеми з повторним використанням коду.

Django — це високорівневий фреймворк для веб-розробки на Python, що спрощує створення складних веб-додатків. Django слідує принципам "конвенція над конфігурацією", що дозволяє швидко розробляти проекти [32].

Переваги Django:

1. Швидкість розробки: Django надає безліч вбудованих функцій, які зменшують час на реалізацію проектів.

2. Потужна ORM: Система об'єктно-реляційного відображення (ORM) спрощує роботу з базами даних, що дозволяє швидко виконувати запити.

3. Безпека: Django включає в себе вбудовані механізми захисту від загроз, таких як CSRF і XSS.

Недоліки Django:

1. Продуктивність: Django може бути менш продуктивним у порівнянні з Node.js, оскільки є інтерпретованою мовою.

2. Крива навчання: Для новачків може знадобитися час на вивчення специфіки фреймворка і Python.

Ruby on Rails — це фреймворк для веб-розробки на Ruby, який підходить для швидкого створення веб-додатків. Він акцентує увагу на продуктивності та простоті [33].

Переваги Ruby on Rails:

1. Швидка розробка: Rails дозволяє швидко створювати прототипи завдяки вбудованим інструментам.

2. Зручність: Синтаксис Ruby є простим і зрозумілим, що полегшує навчання.

3. Конвенція над конфігурацією: Rails використовує ці принципи, що дозволяє зменшити обсяг коду, який потрібно писати.

Недоліки Ruby on Rails:

1. Продуктивність: Rails може бути менш швидким у порівнянні з Node.js.

2. Проблеми зі скейлінгом: Для великих систем можуть виникати складнощі з масштабуванням.

Таблиця 3.1 – Порівняння мов програмування для розробки системи пошуку роботи

Критерії	Javascript	TypeScript	Python	C++	C#	PHP
Типізація	Динамічна	Статична	Динамічна	Динамічна	Статична	Динамічна
Продуктивність	Середня	Вища через типізацію	Низька (для великих навантажень)	Висока	Висока	Середня
Масштабованість	Обмежена	Висока	Обмежена (для великих систем)	Висока	Висока	Середня
Легкість вивчення	Висока (для базових задач)	Середня (вимагає досвіду з JavaScript)	Висока	Низька	Середня	Висока
Безпека	Середня (залежить від розробника)	Вища (через типізацію)	Низька (потрібна додаткова увага)	Висока (якщо правильно використовувати)	Висока	Низька
Простота розробки	Легка (особливо для маленьких проектів)	Середня (вимагає додаткових знань)	Висока	Середня	Середня	Висока
Підтримка великих проектів	Обмежена	Висока	Обмежена	Висока	Висока	Обмежена
Підтримка фреймворків	Розвинена (React, Angular, Node.js)	Повна підтримка фреймворків JavaScript	Розвинена (Django, Flask, FastAPI)	Обмежена для веб-розробки	Розвинена (ASP.NET)	Розвинена (Laravel, Symfony)
Популярність	Висока	Висока	Висока	Середня	Висока	Висока

Отже, при виборі мови програмування та фреймворку для інформаційної системи пошуку роботи, TypeScript у поєднанні з NestJS виявляються оптимальним рішенням. TypeScript забезпечує статичну типізацію, що дозволяє виявляти

помилки на етапі компіляції та підвищує надійність системи. Це особливо важливо, оскільки система обробляє чутливі дані користувачів, такі як резюме та професійні профілі.

NestJS, як прогресивний фреймворк, пропонує потужні інструменти для розробки серверних додатків, включаючи модульність, інжекцію залежностей і підтримку RESTful і GraphQL API. Ці можливості дозволяють організувати код у логічні компоненти, що спрощує його підтримку і розширення, а також забезпечує гнучкість у реалізації нових функцій.

Порівняння з іншими популярними фреймворками, такими як Express.js, Django, Ruby on Rails і Flask, показало, що хоча вони мають свої переваги, жоден з них не забезпечує такої ж масштабованості, продуктивності та безпеки, як TypeScript та NestJS разом.

3.4 Обґрунтування вибору фреймворку для розробки інформаційної технології пошуку роботи

Фреймворк — це програмна платформа, що забезпечує базову структуру для розробки програмного забезпечення. Він надає інструменти, бібліотеки та стандартизовані процеси, що дозволяють програмістам розробляти додатки з меншими зусиллями, зосереджуючись на логіці, а не на інфраструктурі.

Розробка за допомогою фреймворку зазвичай включає наступні компоненти:

- Базові бібліотеки для обробки даних і побудови інтерфейсів.
- Набір стандартних шаблонів і модулів.
- Інструменти для тестування, налагодження та розгортання.
- Підтримка інтеграції з іншими сервісами та API.

Розробка інформаційної технології для пошуку роботи та надання рекомендацій потребує вибору найефективнішого фреймворку, який буде відповідати вимогам щодо масштабованості, продуктивності, гнучкості та інтеграції з іншими сервісами. Оскільки мова програмування для розробки обрана

TypeScript і середовище має бути орієнтованим на серверні додатки, розглянемо декілька популярних фреймворків для створення таких систем, зокрема NestJS, Express**,** Koa та Fastify, і порівняємо їх за кількома критеріями.

NestJS — це фреймворк для створення серверних додатків на Node.js, який використовує TypeScript за замовчуванням. Він побудований на основі Express і має модульну архітектуру, що дозволяє зручно розподіляти логіку та компоненти додатку. NestJS підтримує мікросервісну архітектуру, а також має вбудовану підтримку для роботи з базами даних, зовнішніми API та бібліотеками для роботи з чергами повідомлень, що є важливим для складних систем, як-от система рекомендацій для пошуку роботи.

Переваги**:**

- Модульна архітектура для масштабованих проектів.
- Підтримка TypeScript з самого початку.
- Підтримка мікросервісів та інтеграція з різними технологіями.
- Зручний інструментарій для розробки та тестування.
- Підтримка WebSocket, GraphQL і REST API для інтеграцій.

Недоліки:

- Вищий рівень абстракції, який може бути складним для новачків.
- Може бути важчим для невеликих проектів.

Express — це найпопулярніший фреймворк для створення серверних додатків на Node.js. Він простий і мінімалістичний, забезпечує максимальну гнучкість і дозволяє швидко створювати сервери, проте вимагає більше ручного налаштування для реалізації складних архітектур.

Переваги:

- Легкий і швидкий у використанні.
- Широко підтримується і має велику кількість плагінів.
- Простота інтеграції з іншими бібліотеками і фреймворками.

Недоліки:

- Відсутність вбудованої підтримки для мікросервісів і складних архітектур.

- Не має модульної структури за замовчуванням, що ускладнює масштабування великих проектів.

Koa — це ще один фреймворк від розробників Express, але з більшим акцентом на сучасні технології та проміси. Він більш мінімалістичний, надає більшу гнучкість, але вимагає більше зусиль для налаштування необхідної функціональності.

Переваги:

- Вища гнучкість і контроль над логікою додатка.
- Сучасний підхід до обробки запитів за допомогою асинхронних функцій.
- Легкість у розширенні та інтеграції з іншими бібліотеками.

Недоліки:

- Вимагає більше часу на налаштування, особливо для складних проектів.
- Може бути складним для новачків через відсутність багато вбудованих функцій.

Fastify — це високопродуктивний фреймворк для Node.js, орієнтований на швидкість і ефективність. Він забезпечує високу продуктивність для обробки HTTP-запитів і має вбудовану підтримку для плагінів і мікросервісів.

Переваги:

- Найвища продуктивність серед конкурентів.
- Підтримка плагінів і мікросервісної архітектури.
- Вбудована підтримка для валідації запитів і відповіді.
- Легкий у налаштуванні і масштабуванні.

Недоліки:

- Менш популярний за Express і NestJS, тому має меншу кількість готових рішень і плагінів.
- Може бути занадто швидким для невеликих проектів, де продуктивність не є критичною.

Після ретельного порівняння основних фреймворків для розробки серверних додатків на Node.js, таких як NestJS, Express, Koa та Fastify, можна

зробити висновок, що для розробки інформаційної технології пошуку роботи та надання рекомендацій найкращим вибором є NestJS.

Основною перевагою NestJS є його модульна архітектура, яка дозволяє чітко розподіляти логіку додатку на окремі частини, що спрощує масштабування та підтримку системи в майбутньому. Це є важливим аспектом для проекту, який передбачає обробку великих обсягів даних, надання рекомендацій та інтеграцію з іншими зовнішніми сервісами. Крім того, NestJS поєднує зручність використання TypeScript, що забезпечує високий рівень безпеки коду та полегшує його підтримку.

У порівнянні з іншими фреймворками, такими як Express та Koa, NestJS пропонує більш розширену інфраструктуру для розробки складних систем. Хоча Express є найбільш простим і гнучким, він не надає вбудованих рішень для масштабування або організації складних архітектур, що потребує додаткових зусиль розробника для налаштування таких можливостей. Koa, в свою чергу, надає більш сучасний підхід до обробки запитів, але також вимагає більше часу на налаштування, що робить його менш підходящим для великих і складних проектів.

Fastify, хоча і має найвищу продуктивність серед конкурентів, більше орієнтований на швидкість обробки запитів і може бути занадто швидким для невеликих проектів, де продуктивність не є першочерговим завданням. Його менш широка популярність і обмежена кількість плагінів може стати бар'єром при розробці складних систем.

Таким чином, NestJS є найкращим вибором для даної задачі завдяки своїй здатності підтримувати масштабовані та складні архітектури, а також наявності вбудованих рішень для інтеграції з базами даних, зовнішніми API та іншими сервісами. Крім того, його популярність серед розробників і велика спільнота гарантують стабільність та постійну підтримку фреймворку. Враховуючи всі ці фактори, NestJS ідеально підходить для створення надійної та ефективної технології пошуку роботи та надання рекомендацій, яка зможе ефективно масштабуватися з ростом обсягів даних і користувачів.

3.5 Реалізація програмного модуля інформаційної технології

Для початку роботи програми необхідно провести авторизацію користувача. Для цього використовується функція login, яка забезпечує процес аутентифікації та генерацію токенів доступу для користувача. Спочатку з об'єкта користувача отримуються основні дані, такі як ідентифікатор _id, ім'я name, електронна пошта email, роль role та дозволи permissions. Ці дані використовуються для формування об'єкта payload, який включає метадані: суб'єкта токена sub та видавця токена iss.

Далі генерується refresh токен за допомогою методу createRefreshToken, який потім оновлюється в базі даних через метод updateUserToken. Це дозволяє зберегти новий токен для користувача. Окрім того, refresh токен передається клієнту у вигляді cookie для забезпечення сесії, з параметром httpOnly, що обмежує доступ до токена через JavaScript. Термін дії цього cookie залежить від налаштування JWT_REFRESH_EXPIRE, яке визначається в конфігураційних файлах системи.

Після цього генерується access токен, який дає користувачу доступ до захищених ресурсів. Цей токен створюється за допомогою методу jwtService.sign, використовуючи попередньо сформований payload.

Функція повертає об'єкт, який містить access токен і дані про користувача, включаючи його ідентифікатор, ім'я, електронну пошту, роль та дозволи.

```
async login(user: IUser, response: Response) {
  const { _id, name, email, role, permissions } = user;
  const payload = {
    sub: 'token login',
    iss: 'from server',
    _id,
    name,
    email,
    role,
  };

  const refresh_token = this.createRefreshToken(payload);
```

```
    await this.usersService.updateUserToken(refresh_token, _id);
    response.cookie('refresh_token', refresh_token, {
      httpOnly: true,
      maxAge: ms(this.configService.get<string>('JWT_REFRESH_EXPIRE')),
    });
    return {
      access_token: this.jwtService.sign(payload),
      user: {
        _id,
        name,
        email,
        role,
        permissions,
      },
    };
  }
```

Після авторизації користувача для підтримки сесії необхідно обробити оновлення токенів. Функція processNewToken займається перевіркою та оновленням refresh токена. Це дозволяє користувачам отримувати нові токени без необхідності повторної авторизації через введення пароля.

Функція приймає два параметри: refreshToken — сам refresh токен, що передається користувачем, та об'єкт response, через який надсилаються оновлені cookie.

У першу чергу, функція перевіряє правильність refresh токена за допомогою методу verify з використанням секретного ключа, який зберігається в конфігурації. Якщо токен є дійсним, то виконується пошук користувача в базі даних за допомогою сервісу usersService.findUserByToken.

Якщо користувач знайдений, функція генерує новий refresh токен із оновленим payload, що включає ідентифікатор користувача, ім'я, електронну пошту та роль. Потім цей новий refresh токен оновлюється в базі даних за допомогою методу updateUserToken.

Додатково функція отримує роль користувача і викликає сервіс rolesService.findOne для отримання даних про роль, зокрема прав користувача (permissions).

Після цього функція очищає старий refresh токен у cookie за допомогою response.clearCookie('refresh_token') і встановлює новий refresh токен з параметрами httpOnly та терміном дії, що визначається в конфігураціях.

Функція надає механізм для підтримки сесії користувача, гарантує безпечну передачу токенів та забезпечує оновлення прав доступу, необхідних для подальшої роботи користувача з програмою.

```
processNewToken = async (refreshToken: string, response: Response) => {
  try {
    this.jwtService.verify(refreshToken, {
      secret: this.configService.get<string>('JWT_REFRESH_TOKEN_SECRET'),
    });
    let user = await this.usersService.findUserByToken(refreshToken);
    if (user) {
      const { _id, name, email, role } = user;
      const payload = {
        sub: 'token refresh',
        iss: 'from server',
        _id,
        name,
        email,
        role,
      };

      const refresh_token = this.createRefreshToken(payload);
      await this.usersService.updateUserToken(refresh_token, _id.toString());
      const userRole = user.role as unknown as { _id: string; name: string };
      const temp = await this.rolesService.findOne(userRole._id);
      response.clearCookie('refresh_token');
      response.cookie('refresh_token', refresh_token, {
        httpOnly: true,
```

```
            maxAge: ms(this.configService.get<string>('JWT_REFRESH_EXPIRE')),
          });

          return {
            access_token: this.jwtService.sign(payload),
            user: {
              _id,
              name,
              email,
              role,
              permissions: temp?.permissions ?? [],
            },
          };
        } else {
          throw new BadRequestException(`Refresh token не дійсний. Будь ласка, увійдіть знову.`);
        }
      } catch (error) {
        throw new BadRequestException(`Refresh token не дійсний. Будь ласка, увійдіть знову.`);
      }
    };
```

Далі, після того, як користувач пройшов авторизацію та отримав нові токени для доступу до системи, йому можуть бути надані персоналізовані рекомендації вакансій, що найбільше відповідають його навичкам. Для цього використовується спеціальний сервіс, що аналізує навички користувача та надає відповідні вакансії через нейронну мережу.

Метод getRecommendations працює наступним чином:

- Нормалізація даних: Навички користувача перетворюються у числові значення, що дозволяє передавати їх у вигляді тензорів до нейронної мережі. Для цього обчислюється довжина кожного рядка навички, що використовується як вхідне значення.

- Побудова та тренування моделі нейронної мережі: Для прогнозування підходящих вакансій створюється модель нейронної мережі, яка навчається на основі популярності вакансій. Навчання моделі відбувається через 200 епох.

- Прогнозування рекомендацій: Модель прогнозує співпадіння навичок користувача з навичками вакансій і на основі цього генерує рекомендації, де кожна вакансія має свій відсоток співпадіння та рейтинг популярності.

- Результат: Після прогнозування користувач отримує список вакансій з відповідними характеристиками, включаючи популяристь вакансії, навички, що необхідні для неї, і відсоток співпадіння з навичками користувача.

```
@Injectable()
export class RecommendationsService {
  private jobList = [
    { title: 'Frontend Developer', skills: ['JavaScript', 'React', 'HTML', 'CSS'], popularity: 4.5 },
    { title: 'Backend Developer', skills: ['Node.js', 'NestJS', 'MongoDB', 'SQL'], popularity: 4.2 },
    { title: 'Full Stack Developer', skills: ['React', 'Node.js', 'MongoDB'], popularity: 4.8 },
    { title: 'Data Scientist', skills: ['Python', 'Machine Learning', 'Data Analysis'], popularity: 5.0
},
  ];
  async getRecommendations(userSkills: string[]) {
    const inputTensor = tf.tensor2d(userSkills.map(skill => skill.length), [userSkills.length, 1]);
    const outputTensor = tf.tensor2d(this.jobList.map(job => job.popularity),
[this.jobList.length, 1]);
    const model = tf.sequential();
    model.add(tf.layers.dense({ inputShape: [ 1 ], units: 50, activation: 'relu' }));
    model.add(tf.layers.dense({ units: 1, activation: 'sigmoid' }));
    model.compile({ optimizer: 'adam', loss: 'meanSquaredError' });
    await model.fit(inputTensor, outputTensor, { epochs: 200 });
    const predictions = model.predict(inputTensor) as tf.Tensor;
    const predictedValues = Array.from(predictions.dataSync());
    return this.jobList.map((job, index) => ({
      title: job.title,
      popularity: job.popularity,
      matchPercentage: (predictedValues[index] * 100).toFixed(2),
```

```
      requiredSkills: job.skills,
    }));
  }
}
```

Для ефективного керування даними про компанії, нам необхідно створити схему для зберігання цієї інформації в базі даних. Відповідно до цього, ми створюємо Company схему, яка містить основні властивості компанії, а також забезпечує можливість відстеження, хто створив, оновив або видалив даний запис.

Структура схеми Company

1. name: Назва компанії.
2. address: Адреса компанії.
3. description: Опис компанії.
4. logo: Логотип компанії (може містити URL або базовані на Base64 дані).
5. createdBy: Інформація про користувача, який створив запис.
6. updatedBy: Інформація про користувача, який останнім редагував запис.
7. deletedBy: Інформація про користувача, який видалив запис (якщо застосовується).
8. createdAt: Дата створення запису.
9. updatedAt: Дата останнього оновлення.
10. isDeleted: Статус видаленості (логічне видалення).
11. deletedAt: Дата видалення запису.

Код схеми для MongoDB:

```
@Schema({ timestamps: true })
export class Company {
        @Prop()
        name: string;
        @Prop()
        address: string;
        @Prop()
        description: string;
```

```
    @Prop()
    logo: 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;
}
```

Сервіс для управління компаніями. У цьому сервісі реалізовано кілька основних функцій:

1. create – створення нової компанії.
2. findAll – отримання списку компаній з підтримкою пагінації та фільтрації.
3. findOne – пошук однієї компанії за її ID.
4. update – оновлення даних компанії.

5. remove – м'яке видалення компанії з бази даних.

Сервіс використовує SoftDeleteModel, що дозволяє обробляти видалення компаній без їхнього фізичного видалення. Це особливо корисно для збереження історичних даних та запобігання втраті важливої інформації.

Метод create дозволяє створювати нові записи в базі даних, при цьому зберігається інформація про того, хто створив запис. Це важливо для контролю за діями користувачів у системі. Сервіс приймає DTO (Data Transfer Object) для створення компанії та автоматично додає дані про користувача, що створює запис.

```
create(createCompanyDto: CreateCompanyDto, user: IUser) {
return this.companyModel.create({
        ...createCompanyDto,
        createdBy: {
                _id: user._id,
                email: user.email,
        },
});
```

Метод findAll дозволяє отримати список компаній з пагінацією та фільтрацією, використовуючи популярну бібліотеку aqp для обробки запитів. Тут реалізовано розрахунок кількості сторінок на основі загальної кількості компаній та переданого ліміту на сторінку.

```
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.companyModel.find(filter)).length;
const totalPages = Math.ceil(totalItems / defaultLimit);
const result = await this.companyModel
        .find(filter)
        .skip(offset)
```

```
            .limit(defaultLimit)
            .sort(sort as any)
            .populate(population)
            .exec();
    return {
            meta: {
                    current: currentPage, // поточна сторінка
                    pageSize: limit, // кількість елементів на сторінці
                    pages: totalPages, // кількість сторінок
                    total: totalItems, // загальна кількість записів
            },
            result, // результат запиту
    };
```

Метод findOne дозволяє знайти компанію за її ID. Якщо ID некоректне, викидається виключення BadRequestException.

```
async findOne(id: string) {
if (!mongoose.Types.ObjectId.isValid(id)) {
        throw new BadRequestException(`not found company with id=${id}`);
}
return await this.companyModel.findById(id);
}
```

Метод update оновлює дані компанії за її ID. При цьому також зберігається інформація про того, хто виконав оновлення.

```
async update(id: string, updateCompanyDto: UpdateCompanyDto, user: IUser) {
return await this.companyModel.updateOne(
        { _id: id },
        {
                ...updateCompanyDto,
                updatedBy: {
                        _id: user._id,
                        email: user.email,
                },
```

```
        },
);
```

Метод remove реалізує м'яке видалення компанії. Після того, як компанія позначена як видалена, її статус змінюється на isDeleted: true, а також зберігається інформація про того, хто виконав видалення.

```
async remove(id: string, user: IUser) {
await this.companyModel.updateOne(
        { _id: id },
        {deletedBy: {_id: user._id, email: user.email, },},
);
return this.companyModel.softDelete({
        _id: id,
});
```

Сервіс ResumesService забезпечує управління резюме користувачів у системі. Це включає можливості для створення, оновлення, видалення та пошуку резюме з різними параметрами фільтрації та сортування. Сервіс також підтримує м'яке видалення резюме, зберігаючи дані про користувача, який видалив запис, та історію змін статусу резюме.

Основні можливості:

- Створення резюме: дозволяє користувачам додавати нові резюме до системи, зберігаючи відповідну інформацію, таку як URL, компанія, робота та статус.

- Пошук резюме: дозволяє знаходити резюме користувачів з можливістю фільтрації, сортування та пагінації, що допомагає ефективно обробляти великий обсяг даних.

- Історія змін статусу: кожне резюме містить історію змін статусу, що дозволяє відслідковувати всі дії, пов'язані з його обробкою.

- Оновлення статусу резюме: дає можливість оновлювати статус резюме (наприклад, "PENDING", "APPROVED", "REJECTED") і зберігати інформацію про зміни.

- М'яке видалення: при видаленні резюме не відбувається фізичного видалення з бази даних, що дозволяє зберігати інформацію для подальшого аналізу або відновлення.

Використовувані технології:

- MongoDB та Mongoose для зберігання даних.
- Soft delete plugin для підтримки м'якого видалення.
- API Query Parameters (AQP) для фільтрації та сортування даних.
- DTO (Data Transfer Object) для валідації вхідних даних.

Цей сервіс є частиною більшої системи управління кар'єрними можливостями та забезпечує зручний інтерфейс для управління резюме, що дозволяє компаніям та кандидатам ефективно взаємодіяти.

Сервіс ResumesService містить кілька важливих методів для роботи з резюме користувачів. Кожен з цих методів реалізує окрему функціональність, що дозволяє ефективно управляти резюме в системі.

Метод create дозволяє створити нове резюме для користувача. Він приймає об'єкт createUserCvDto, який містить необхідні дані для створення резюме, такі як URL, ID компанії та ID вакансії. Крім того, цей метод автоматично додає інформацію про користувача, який створює резюме (email та ID). Резюме отримує статус PENDING і зберігається разом з історією змін статусу, де зазначається, що резюме в даний момент має статус "Очікує".

```
async create(createUserCvDto: CreateUserCvDto, user: IUser) {
  const { url, companyId, jobId } = createUserCvDto;
  const { email, _id } = user;

  const newCV = await this.resumeModel.create({
    url,
    companyId,
    email,
```

```
    jobId,
    userId: _id,
    status: 'PENDING',
    createdBy: { _id, email },
    history: [
      {
        status: 'PENDING',
        updatedAt: new Date(),
        updatedBy: {
          _id: user._id,
          email: user.email,
        },
      },
    ],
  });
  return {
    _id: newCV?._id,
    createdAt: newCV?.createdAt,
  };
}
```

Метод update оновлює статус конкретного резюме. Він приймає ID резюме, новий статус та дані користувача, який оновлює резюме. Окрім оновлення статусу, цей метод додає запис до історії змін, фіксуючи новий статус разом з інформацією про те, хто та коли оновив резюме. Якщо ID резюме некоректне, метод викидає виключення BadRequestException.

```
async update(_id: string, status: string, user: IUser) {
  if (!mongoose.Types.ObjectId.isValid(_id)) {
    throw new BadRequestException('не знайдено резюме');
  }
  const updated = await this.resumeModel.updateOne(
    { _id },
    {
      status,
```

```
      updatedBy: {
        _id: user._id,
        email: user.email,
      },
      $push: {
        history: {
          status: status,
          updatedAt: new Date(),
          updatedBy: {
            _id: user._id,
            email: user.email,
          },
        },
      },
    },
  );
  return updated;
}
```

Отже, в цьому розділі було розглянуто основні аспекти роботи з сервісами в рамках проекту, починаючи з авторизації і до управління резюме користувачів через сервіс ResumesService.

Загалом, підхід, використаний в проекті, орієнтований на забезпечення високої ефективності роботи з даними та безпеку системи. Авторизація, реалізована через JWT, і гнучкий сервіс для роботи з резюме дозволяють створювати потужні та безпечні додатки для користувачів і адміністраторів.

3.6 Тестування інформаційної технології

Для проведення тестування серверної частини інформаційної технології було використано Swagger, як показано на рисунку 3.2. Це інструмент для документації та тестування API, який дозволяє автоматично створювати

документацію розробленого API завдяки опису її структури в форматі YAML або JSON файлу. Для кожного з методів HTTP, Swagger дозволяє описати параметри запитів і формат відповідей. Swagger також дозволяє описувати параметри та формат відповідей для методів POST, PUT і DELETE.

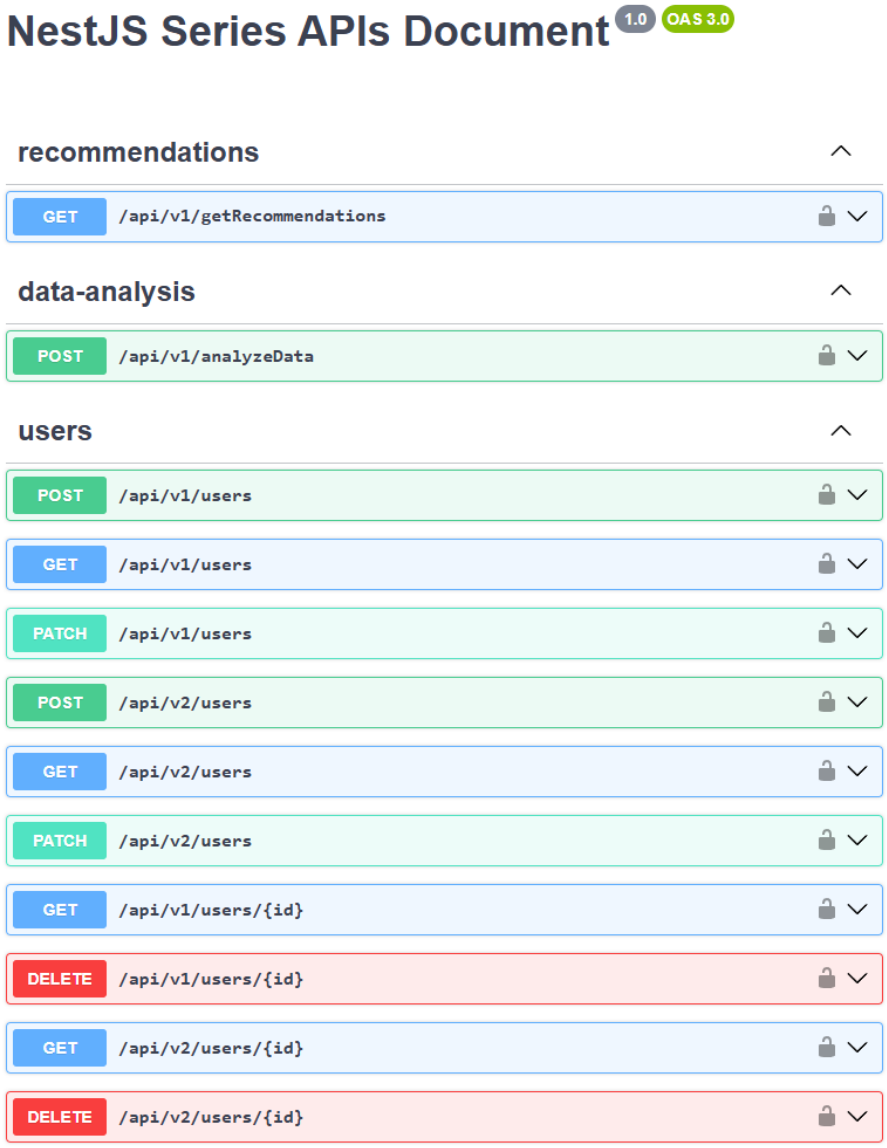

Рисунок 3.2 – Серверна частина інформаційної технології у Swagger

Після переходу на сторінку логіну користувач потрапляє на інтерфейс для введення своїх облікових даних. Сторінка містить два основних поля: для введення логіну та паролю, а також кнопку «Увійти» для авторизації в системі. Користувач

може також вибрати опцію «Забули пароль?» для відновлення доступу, якщо виникли проблеми з уведенням пароля.

При правильному введенні даних система перевіряє їх на сервері, і у разі успішної авторизації, користувач перенаправляється до основного інтерфейсу системи, як показано на рисунку 3.3. Якщо ж введені дані невірні, система відображає відповідне повідомлення про помилку, а користувач може спробувати знову.

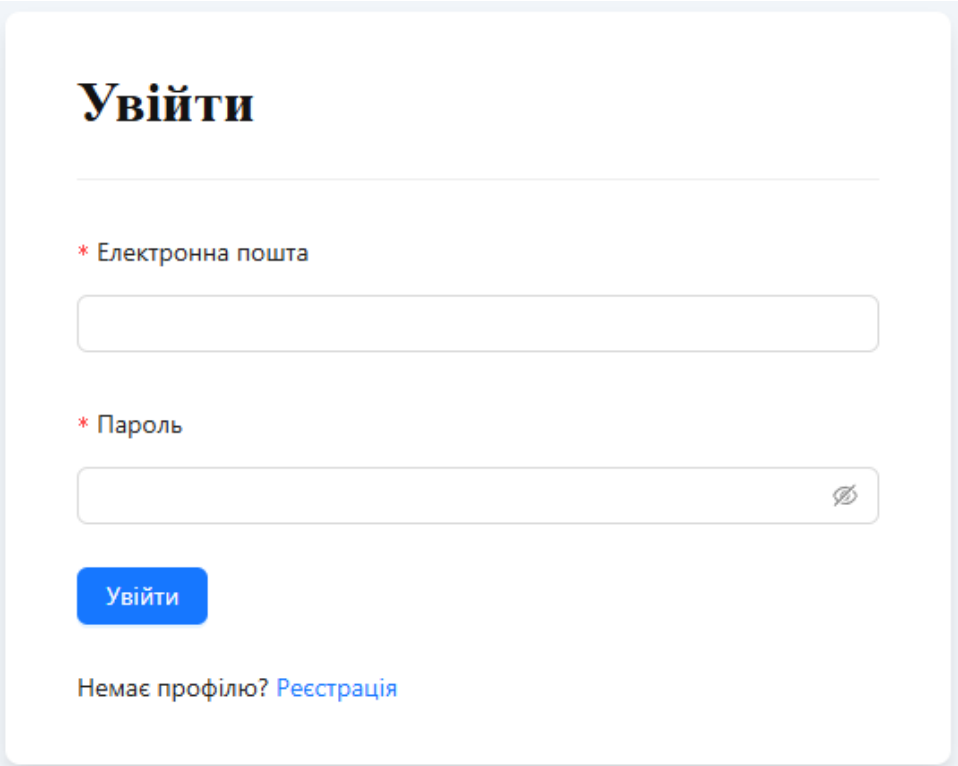

Рисунок 3.3 – Загальний вигляд інтерфейсного вікна авторизації користувача

На рисунку 3.4 продемонстровано інтерфейс для тестування API-запитів, де реалізовано метод GET /api/v1/getRecommendations. Цей метод надає рекомендації для користувача, на основі його вподобань, вказаних через userId. Додатково, запит може включати фільтрацію за категорією (category) та локацією (location), які вказуються як параметри запиту.

При успішному виконанні запиту (статус 200) користувач отримує відповідь у форматі JSON, що містить список рекомендацій. Кожна рекомендація представлена як об’єкт, що включає jobTitle (посаду), companyName (назву компанії) та location (локацію).

Інструмент Swagger надає можливість тестування цього API методом введення необхідних параметрів і перевірки результатів безпосередньо в браузері. Це дозволяє швидко перевірити, як працює система, і переконатись у правильності відповіді, яку вона повертає.

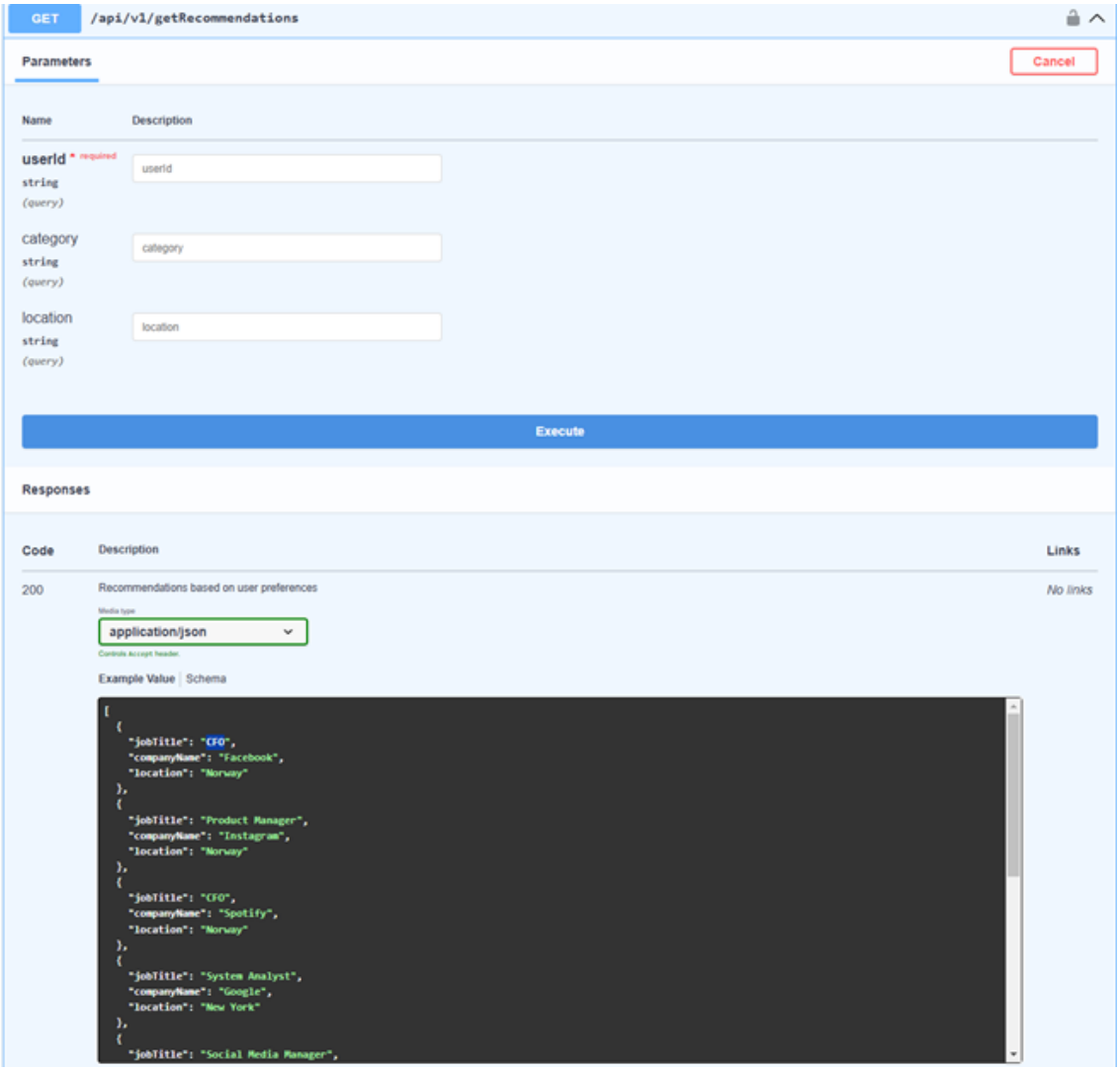

Рисунок 3.4 – Інтерфейс для тестування API рекомендацій у Swagger

На рисунку 3.5 зображено інтерфейс користувача для перегляду детальної інформації про вакансію. Інтерфейс включає основні відомості про вакансію, такі як назва посади (CFO), локація (Норвегія), компанія (Spotify Corporation), дата публікації (14 днів тому), а також дані про зарплату, тип зайнятості, кількість заявок та кількість доступних вакансій. Основний опис вакансії подано в розділі «Job Description», де розміщено загальну інформацію про вимоги та обов’язки.

Крім цього, в інтерфейсі є панель з подібними вакансіями (Similar Job Post), розташована праворуч. Тут користувач може переглянути схожі вакансії, що містять інформацію про посаду, місце розташування, а також дату публікації. Цей блок дозволяє користувачеві легко переходити до інших можливих варіантів працевлаштування.

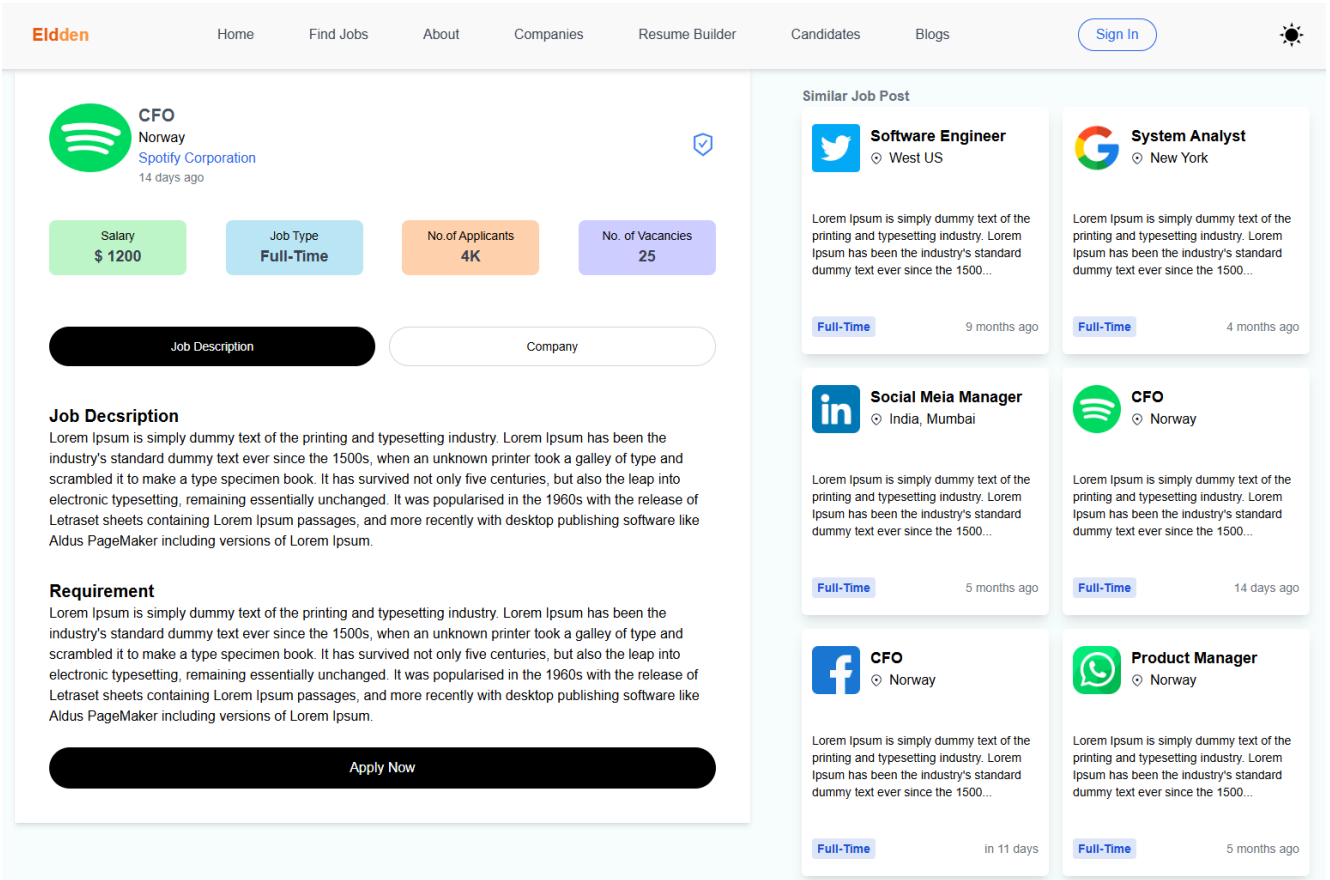

Рисунок 3.5 – Інтерфейс перегляду вакансії та подібних пропозицій

Після переходу до сторінки створення або редагування вакансії ("Upsert Job"), користувач має змогу заповнити ключові параметри роботи. Інтерфейс, який показано на рисунку 3.6, містить такі основні поля:

1. Назва посади – поле для введення назви роботи. Це обов'язкове поле, яке допомагає визначити роль або позицію в компанії.
2. Необхідні навички – випадаючий список для вибору навичок, необхідних для виконання завдань на посаді. Це дозволяє структуровано задати вимоги до потенційного кандидата.
3. Розташування – поле для вибору місця роботи, яке може бути вказано як конкретне місто або регіон.
4. Зарплата – секція для зазначення пропонованої заробітної плати. Можна ввести значення в локальній валюті.
5. Кількість – поле для вказання кількості відкритих вакансій на дану посаду.
6. Рівень – випадаючий список, де можна вибрати рівень кандидата, наприклад, "Junior", "Middle", або "Senior".
7. Належить компанії – поле для вибору компанії, до якої належить ця вакансія. Може бути корисним для великих організацій із кількома підрозділами.
8. Дата початку та кінцева дата – календарі для зазначення дати початку роботи та кінцевої дати, якщо це тимчасова позиція.
9. Статус – перемикач для встановлення статусу вакансії ("ACTIVE" або інше), щоб контролювати її видимість.
10. Опис вакансії – текстове поле з функціями форматування, що дозволяє детально описати обов'язки, вимоги та умови роботи.

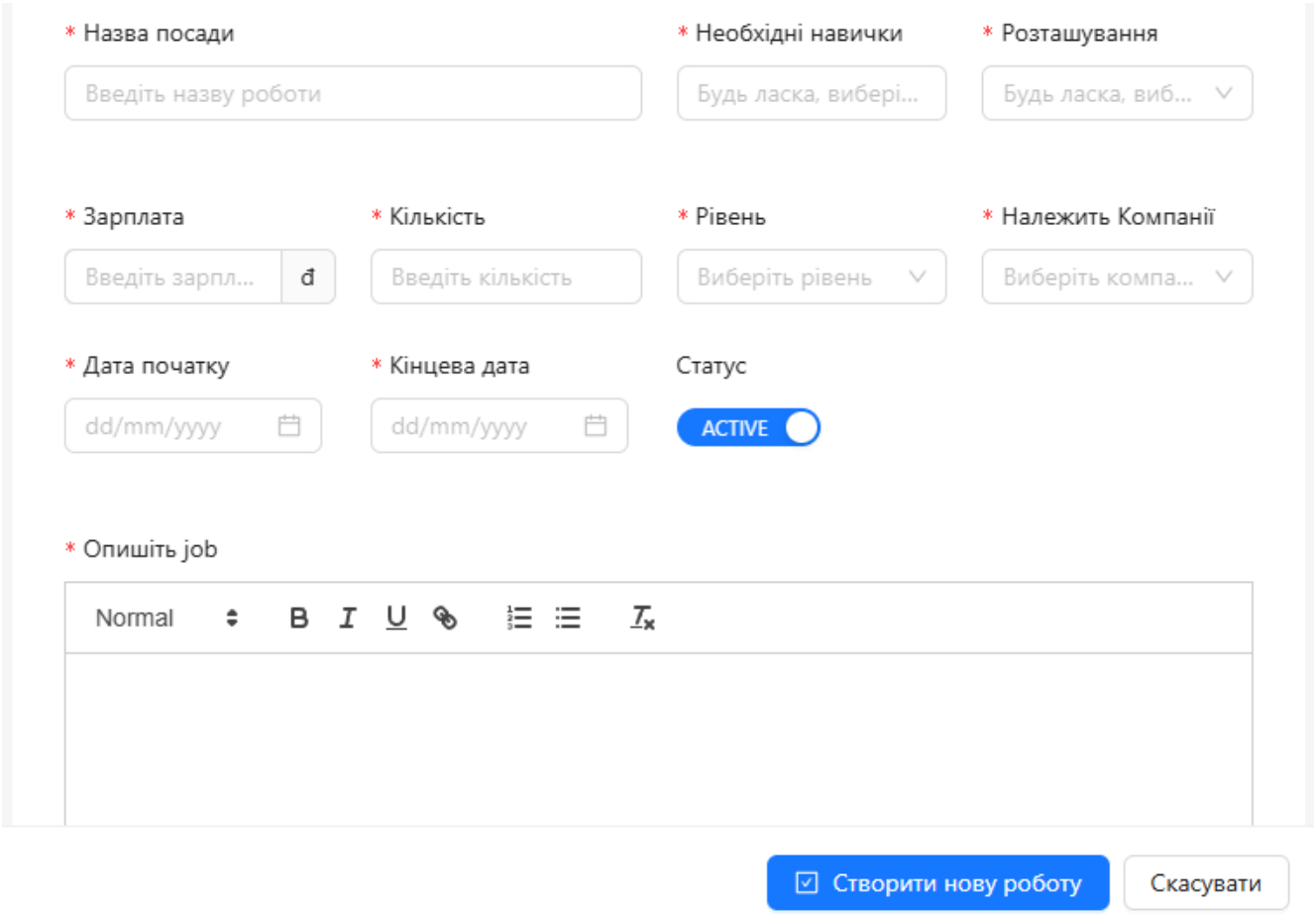

Рисунок 3.6 – Інтерфейс перегляду вакансії та подібних пропозицій

Розроблений програмний модуль надання рекомендацій для догляду за кімнатними рослинами успішно пройшов тестування та відповідає всім заданим вимогам.

В таблиці 3.2 подано результати тестування розробленого програмного продукту та аналогів.

Таблиця 3.2 – Результати тестування розробленого програмного продукту та аналогів

Параметр	Розроблений програмний продукт	LinkedIn	Indeed	Work.ua
Оптимізація інтерфейсу	+	-	+	-
Вартість	Нижча	Вища	Вища	Середня
Використання експертної системи для надання рекомендацій	+	-	-	-
Швидкість пошуку вакансій	Висока	Середня	Середня	Низька
Персоналізовані рекомендації для користувачів	+	+	-	-
Анонімність даних користувачів	+	-	-	-

Розроблений програмний продукт перевершує аналоги за ключовими параметрами, такими як оптимізація інтерфейсу, швидкість пошуку вакансій, використання експертної системи та анонімність даних. Це робить його ефективнішим та зручнішим вибором для користувачів.

3.7 Висновок до розділу 3

У цьому розділі було детально обґрунтовано вибір архітектури, мови програмування та фреймворку для розробки інформаційної технології пошуку роботи. Представлені рішення спрямовані на забезпечення ефективності, безпеки та масштабованості системи.

Обґрунтування вибору архітектури базується на клієнт-серверній моделі, яка забезпечує надійне управління запитами користувачів та обробку інформації. Надійність і безпека передавання даних гарантується використанням HTTPS-протоколу, що шифрує інформацію, захищаючи її від можливих загроз під час передавання через мережу Інтернет. Це особливо важливо у контексті роботи з чутливими даними, включаючи персональні профілі, резюме та іншу інформацію користувачів.

Схема алгоритму, розроблена для реалізації інформаційної технології пошуку роботи, описує ключові етапи взаємодії користувача з системою. Алгоритм включає процеси реєстрації, авторизації, пошуку вакансій та надання персоналізованих рекомендацій. Зокрема, враховано можливі сценарії, зокрема створення нового облікового запису, вхід до системи та пошук роботи, з можливістю уточнення параметрів для отримання більш точних результатів. Логіка алгоритму забезпечує оптимальне використання ресурсів системи та підвищує зручність роботи для користувачів.

Вибір мови програмування TypeScript було здійснено з огляду на її можливості забезпечення статичної типізації, що дає змогу виявляти помилки ще на етапі компіляції, тим самим підвищуючи загальну надійність системи. Це має ключове значення, оскільки система повинна ефективно обробляти та захищати великі обсяги чутливої інформації. Крім того, використання TypeScript сприяє покращенню підтримки коду та його масштабованості.

Фреймворк NestJS обрано завдяки його потужним можливостям для створення серверних додатків. NestJS забезпечує модульну структуру коду, підтримує інжекцію залежностей і має вбудовану підтримку для RESTful і GraphQL API. У порівнянні з іншими фреймворками, такими як Express.js, Django, Ruby on Rails і Flask, саме комбінація TypeScript і NestJS забезпечує необхідний рівень продуктивності, масштабованості та безпеки.

Таким чином, обґрунтований підхід до вибору архітектури, мови програмування та фреймворку дозволяє створити сучасну та надійну технологію пошуку роботи. Впроваджені рішення відповідають усім вимогам проєкту, забезпечуючи високу продуктивність, безпеку передавання даних та можливість подальшого розширення функціоналу. Система здатна ефективно масштабуватися з ростом кількості користувачів та обсягів оброблюваних даних, що робить її перспективною для використання у великих масштабах.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1. Проведення комерційного та технологічного аудиту інформаційної технології пошуку роботи (надання рекомендацій).

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 [34].

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 (надання рекомендацій)

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

Продовження таблиці 4.1

Критерії	Експерти		
	Експерт 1	Експерт 1	Експерт 1
	Бали, виставлені експертами		
Практична здійсненність (необхідність нових матеріалів)	3	3	3
Практична здійсненність (термін реалізації)	3	2	3
Практична здійсненність (розробка документів)	3	1	2
Сума балів	30	29	33
Середньоарифметична сума балів, СБ	30,7		

За результатами розрахунків, наведених в таблиці 1 робимо висновок про те, що науково-технічний рівень та комерційний потенціал інформаційної технології пошуку роботи (надання рекомендацій) – вище середнього.

4.2 Розрахунок витрат на здійснення науково-дослідної роботи розробки інформаційної технології пошуку роботи (надання рекомендацій)

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, 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:

$$С_і = \frac{М_м \cdot К_і \cdot К_с}{Т_р \cdot t_{зм}}, \quad (4.3)$$

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

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

$К_с$ – 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	25	5	61,8	1,36	1545
Моделювання інформаційної технології	104	6	65,9	1,45	6854
Створення основної структури проєкту	26	5	61,8	1,36	1607
Створення та наповнення бази даних	71	6	65,9	1,45	4679
Тестування інформаційної технології	67	5	61,8	1,36	4141
Всього					18826

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:

$$З_д = 0{,}1 \cdot (З_о + З_р) = 0{,}1 \cdot (109856 + 18826) = 12869 \text{ грн.} \quad (4.4)$$

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

$$Н_{зп} = \beta \cdot (З_о + З_р + З_д) = \quad (4.5)$$
$$= 0{,}22 \cdot (109856 + 18826 + 12869) = 31142 \text{ грн.}$$

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.
Docker for Business	999	1	999
VS Code Professional	1845	1	1845
Всього, з врахуванням коефіцієнта інсталяції та налагодження			3125

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.
Комп'ютер	48500	2	1,5	3031
Монітор	7849	2	1,5	491
Стіл	3590	5	1,5	90
Стілець	1349	5	1,5	34
Мишка	1450	2	1,5	91
Клавіатура	1250	2	1,5	78
	3815			

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

Таблиця 4.6 – Витрати на електроенергію

Найменування обладнання	Потужність, кВт	Тривалість годин роботи
Комп'ютер	0,21	290
Освітлення	0,029	125

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

$$\text{Ве} = \sum \frac{W_i \cdot t_i \cdot \text{Ц}_\text{е} \cdot \text{К}_\text{впі}}{\text{ККД}} \tag{4.8}$$

$$= \frac{0{,}21 \cdot 290 \cdot 7{,}5 \cdot 0{,}85}{0{,}98} + \frac{0{,}029 \cdot 125 \cdot 7{,}5 \cdot 0{,}85}{0{,}98}$$

$$= 419 \text{ грн.},$$

де 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:

$$Ів = \left(З_о + З_р\right) \cdot \frac{Н_{ів}}{100\%} = (109856 + 18826) \cdot \frac{85}{100} = 109383 \text{ грн.}, \quad (4.9)$$

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:

$$Внзв = \left(З_о + З_р\right) \cdot \frac{Н_{нзв}}{100\%} = (109856 + 18826) \cdot \frac{150}{100} = 193029 \text{ грн.}, \quad (4.10)$$

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:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + \\ + І_в + В_{нзв} = \\ = 109856 + 18826 + 12869 + 31142 + 3125 + 3815 + 419 + \\ 109383 + 193029 = 482468 \text{ грн.} \quad (4.11)$$

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

$$ЗВ = \frac{В_{заг}}{\eta} = \frac{482468}{0,9} = 536076 \text{ грн.}, \quad (4.12)$$

де η – коефіцієнт, що характеризує етап виконання науково-дослідної роботи.

Оскільки, якщо науково-технічна розробка знаходиться на стадії впровадження, то η=0,9.

4.3 Розрахунок економічної ефективності науково-технічної розробки інформаційної технології пошуку роботи (надання рекомендацій) 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.

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 λ =0,8333;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту (послуги). Рекомендується брати ρ =0,2...0,5; ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2024 році ϑ =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:

$$PV = k_{\text{інв}} \cdot \text{ЗВ} = 3 \cdot 536076 = 1608227 \text{ грн.} \quad (4.16)$$

де $k_{\text{інв}}$ – 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 $\text{Е}_{\text{абс}}$ або чистий приведений дохід для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$\text{Е}_{\text{абс}} = \text{ПП} - PV = 1722000 - 1608227 = 113773 \text{ грн.}, \quad (4.17)$$

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

PV – теперішня вартість початкових інвестицій, грн.

Оскільки $Е_{абс}>0$, 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:

$$Е_в = \sqrt[Т_ж]{1+\frac{Еабс}{PV}} - 1 = \sqrt[1]{1+\frac{113773}{1608227}} - 1 = 0{,}3, \quad (4.18)$$

де $Т_ж$ – життєвий цикл розробки, роки.

Визначимо бар'єрну ставку дисконтування $\tau_{мін}$, 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 $\tau_{мін}$ визначається за формулою:

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

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 $E_в$=0,3>$\tau_{мін}$=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 (надання рекомендацій):

$$T_о = \frac{1}{E_в} = \frac{1}{0,3} = 3,3 \text{ року.} \quad (4.20)$$

Оскільки $T_о$<1…4-х років, то це свідчить про комерційну привабливість науково-технічної розробки інформаційної технології пошуку роботи (надання рекомендацій) і може спонукати потенційного інвестора профінансувати впровадження цієї розробки та виведення її на ринок.

4.4 Висновок до розділу 4

Згідно з проведеними дослідженнями, рівень комерційного потенціалу розробки за темою «Інформаційна технологія пошуку роботи. Частина 2. Надання рекомендацій» становить 30,7 балів. Це свідчить про вище середнього рівня потенціалу, що вказує на можливість успішного виходу на ринок і реалізації проєкту в умовах сучасної конкуренції.

Загальні витрати на проведення науково-дослідної роботи становлять 536076 грн, при цьому абсолютний економічний ефект на рівні 113773 грн демонструє, що розробка може приносити значний фінансовий прибуток за умов її ефективної комерціалізації. Внутрішня економічна дохідність інвестицій становить 0,3, що суттєво перевищує бар’єрну ставку дисконтування (0,17). Це є вагомим фактором

для залучення потенційних інвесторів, адже інвестиції в цю розробку демонструють високу перспективу економічного зростання.

Термін окупності становить 3,3 роки, що є конкурентним показником у порівнянні з аналогічними проєктами. Враховуючи це, розробка має значний потенціал для швидкої інтеграції на ринок та отримання стійких фінансових результатів.

Впровадження цієї розробки не тільки сприятиме підвищенню ефективності ринку праці, але й відкриє нові можливості для користувачів шляхом забезпечення сучасних алгоритмів рекомендацій. Завдяки цьому, продукт може стати затребуваним серед організацій, які займаються підбором персоналу, а також серед окремих користувачів, які шукають роботу.

Крім того, розробка демонструє позитивний соціальний ефект, оскільки дозволяє покращити якість підбору вакансій і кандидатів, зменшити час пошуку роботи та підвищити рівень задоволення користувачів. У сучасних умовах диджиталізації та зростаючої потреби в автоматизованих рішеннях, її впровадження може значно посилити конкурентні позиції компаній, які використовують даний інструмент.

Отже, розробка за темою «Інформаційна технологія пошуку роботи. Частина 2. Надання рекомендацій» є не лише економічно обґрунтованою, але й має великий потенціал впливу на ринок праці. Доцільність її впровадження є очевидною, а перспективи комерційного успіху та соціального ефекту забезпечують підґрунтя для подальшого розвитку і масштабування проєкту.

ВИСНОВКИ

В процесі виконання дослідження було проаналізовано сучасні технології пошуку роботи та існуючі платформи, зокрема, «Indeed», «LinkedIn» і «Work.ua». У розділі проаналізовано переваги та недоліки цих систем для виявлення ключових характеристик, що стали основою при формуванні вимог до розробки нової інформаційної технології. Досліджено, як персоналізовані рекомендації та інтелектуальні системи можуть підвищити ефективність пошуку вакансій, з урахуванням індивідуальних характеристик користувачів, їхніх навичок та досвіду.

Другий розділ було присвячено аналізу методів оцінки кваліфікації користувачів та рекомендаційних систем на основі навичок і компетенцій. Розглянуто та обґрунтовано вибір відповідних методів для розробки рекомендаційної системи, зокрема методів машинного навчання та аналізу даних, що є ефективними для формування персоналізованих порад.

У третьому розділі обґрунтовано вибір програмних засобів для реалізації платформи. Обрані інструменти дозволяють оптимізувати роботу платформи та забезпечити її стабільність і масштабованість для великої кількості користувачів. Розроблений алгоритм функціонування платформи враховує навички, досвід користувача та ринкові тенденції для надання актуальних рекомендацій, що сприяє ефективності взаємодії з кінцевими користувачами.

Розроблений програмний продукт було порівняно з існуючими платформами за ключовими параметрами: оптимізація інтерфейсу, вартість, використання експертної системи для надання рекомендацій, швидкість пошуку вакансій, персоналізовані рекомендації для користувачів та анонімність даних. Аналіз показав, що розробка перевершує аналоги за більшістю критеріїв.

Зокрема, оптимізація інтерфейсу та швидкість пошуку вакансій є значно кращими, що забезпечує комфортне користування та оперативний доступ до потрібної інформації. Крім того, впровадження експертної системи дозволяє надавати персоналізовані рекомендації, що значно підвищує ефективність пошуку.

Анонімність даних користувачів є ще однією важливою перевагою, яка сприяє зростанню довіри до продукту.

Таким чином, розроблений програмний продукт демонструє значну перевагу над конкурентами, поєднуючи високу функціональність, доступність і зручність для користувачів, що робить його перспективним рішенням на ринку платформ для пошуку роботи.

Аналіз економічної ефективності розробки «Інформаційна технологія пошуку роботи. Частина 2. Надання рекомендацій» показав, що її комерційний потенціал становить 30,7 балів, що свідчить про високі перспективи виходу на ринок. Загальні витрати становлять 536076 грн, а абсолютний економічний ефект у 113773 грн демонструє можливість отримання значного прибутку. Внутрішня економічна дохідність (0,3) перевищує бар'єрну ставку дисконтування, а термін окупності проєкту — 3,3 роки.

Розробка має значний соціальний вплив, оскільки сприяє підвищенню ефективності пошуку роботи, зменшує час на підбір вакансій і кандидатів та підвищує задоволення користувачів. Це рішення відповідає сучасним тенденціям і може стати важливим інструментом для компаній і користувачів, підвищуючи їх конкурентоспроможність.

Визначено мету, об'єкт, предмет та задачі подальшого дослідження для вдосконалення системи та покращення якості рекомендацій.

Таким чином, результати дослідження підтвердили доцільність розробки інформаційної технології для пошуку роботи з рекомендаційним функціоналом. Представлена система відповідає сучасним вимогам ринку праці та може ефективно забезпечувати користувачів актуальними рекомендаціями, враховуючи їхні особисті потреби і тенденції ринку. Це робить технологію не лише корисною, але й перспективною для подальшого розвитку у сфері автоматизованого пошуку роботи.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Стаднік Е. В., Арсенюк І. Р. ДОСЛІДЖЕННЯ ТА АНАЛІЗ АЛГОРИТМІВ МАШИННОГО НАВЧАННЯ ДЛЯ ПЕРСОНАЛІЗОВАНИХ РЕКОМЕНДАЦІЙ 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]. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/viewFile/21558/17841. (дата звернення 04.09.2024).

2. LinkedIn. Офіційний сайт професійної мережі [Електронний ресурс]. URL: https://www.linkedin.com/ (дата звернення 07.10.2024).

3. Indeed. Пошук роботи та оголошення про вакансії [Електронний ресурс]. URL: https://www.indeed.com/ (дата звернення 07.10.2024).

4. Work.ua. Пошук роботи в Україні [Електронний ресурс]. URL: https://www.work.ua/ (дата звернення 07.10.2024).

5. Robota.ua. Пошук вакансій в Україні [Електронний ресурс]. URL: https://www.robota.ua/ (дата звернення 07.10.2024).

6. UNIAN. Рейтинг сайтів з роботою в Україні [Електронний ресурс]. URL: https://www.unian.ua/economics/other/reyting-saytiv-z-robotoyu-v-ukrajini-12253731.html (дата звернення 07.10.2024).

7. LinkedIn Engineering Blog. Методи аналізу профілю користувача та вакансій [Електронний ресурс]. URL: https://engineering.linkedin.com/blog (дата звернення 07.10.2024).

8. The Future of Work Report, World Economic Forum [Електронний ресурс]. URL: https://www.weforum.org/reports/future-of-jobs-report (дата звернення 07.10.2024).

9. Молчанова Е., Солодковський Ю. Глобальна сервісна природа сучасних криптовалют. Міжнародна економічна політика. 2014. № 1. С. 60 – 79.

10. Indeed Hiring Lab. Інструменти для оцінки професійної відповідності кандидатів [Електронний ресурс]. URL: https://www.hiringlab.org/research/candidate-fit (дата звернення 07.10.2024).

11. Glassdoor. Тренди найму в режимі реального часу [Електронний ресурс]. URL: https://www.glassdoor.com/research/job-trends (дата звернення 07.10.2024).

12. LinkedIn. Інтеграція даних для оптимізації процесу відбору [Електронний ресурс]. URL: https://www.linkedin.com/business/talent/integrations (дата звернення 07.10.2024).

13. Коваленко І. Я., Федоренко С. І. Аналіз впливу маркетингових стратегій на ринок праці. Економічний огляд. 2022. № 5. С. 22-30.

14. Global Skills Gap Report. Вплив регіональних факторів на ринок праці [Електронний ресурс]. URL: https://www.globalskillsgap.com/report (дата звернення 07.10.2024).

15. OECD. Використання даних для аналізу ринку праці [Електронний ресурс]. URL: https://www.oecd.org/employment/data-analysis.htm (дата звернення 07.10.2024).

16. Бутенко Л. С., Чередниченко В. О. Системний підхід до аналізу ринку праці: концептуальні засади. Економіка та управління. 2023. № 3. С. 15-22.

17. Коваленко О. С. Штучні нейронні мережі: нові горизонти застосування в економіці. Економічні дослідження. 2022. № 2. С. 10-18.

18. Черненко Н. А. Аналіз даних за допомогою штучних нейронних мереж: підходи та перспективи. Науковий журнал. 2023. № 1. С. 88-95.

19. Чумак В. А. Математичні моделі в аналізі ринку праці: сучасні тенденції. Науковий вісник. 2022. № 3. С. 25-32.

20. Бондар В. П., Лисенко О. А. Аналіз і прогнозування ринку праці: сучасні підходи. Економічні науки. 2023. Т. 15. С. 45-51.

21. 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.).

22. Клієнт-серверна архітектура [Електронний ресурс]. URL: https://uk.wikipedia.org/wiki/Клієнт-серверна_архітектура (дата звернення 07.10.2024).

23. HTTPS: що таке протокол безпеки [Електронний ресурс]. URL: https://www.techtarget.com/searchsoftwarequality/definition/HTTPS (дата звернення 07.10.2024).

24. JavaScript [Електронний ресурс]. URL: https://developer.mozilla.org/uk/docs/Web/JavaScript (дата звернення 07.10.2024).

25. TypeScript [Електронний ресурс]. URL: https://www.typescriptlang.org/ (дата звернення 07.10.2024).

26. Python [Електронний ресурс]. URL: https://www.python.org/ (дата звернення 07.10.2024).

27. C++ [Електронний ресурс]. URL: https://www.iso.org/standard/77821.html (дата звернення 07.10.2024).

28. C# [Електронний ресурс]. URL: https://docs.microsoft.com/en-us/dotnet/csharp/ (дата звернення 07.10.2024).

29. PHP [Електронний ресурс]. URL: https://www.php.net/ (дата звернення 07.10.2024).

30. NestJS [Електронний ресурс]. URL: https://nestjs.com/ (дата звернення 07.10.2024).

31. Express.js [Електронний ресурс]. URL: https://expressjs.com/ (дата звернення 07.10.2024).

32. Django [Електронний ресурс]. URL: https://www.djangoproject.com/ (дата звернення 07.10.2024).

33. Ruby on Rails [Електронний ресурс]. URL: https://rubyonrails.org/ (дата звернення 07.10.2024).

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

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 2. Надання рекомендацій

Тип роботи: магістерська кваліфікаційна робота
(БДР, МКР)

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

Показники звіту подібності Turnitin

Оригінальність 96,00% Схожість 4,00%

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 Стаднік Е.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 { HttpException, HttpStatus, Injectable } from '@nestjs/common';
import {
	MulterModuleOptions,
	MulterOptionsFactory,
} from '@nestjs/platform-express';
import fs from 'fs';
import { diskStorage } from 'multer';
import path, { join } from 'path';

@Injectable()
export class MulterConfigService implements MulterOptionsFactory {
	getRootPath = () => {
		return process.cwd();
	};

	ensureExists(targetDirectory: string) {
		fs.mkdir(targetDirectory, { recursive: true }, (error) => {
			if (!error) {
				console.log(
					'Directory successfully created, or it already exists.',
				);
				return;
			}
			switch (error.code) {
				case 'EEXIST':
					// Error:
					// Requested location already exists, but it's not a directory.
					break;
				case 'ENOTDIR':
					// Error:
					// The parent hierarchy contains a file with the same name as the dir
					// you're trying to create.
					break;
				default:
					// Some other error like permission denied.
					console.error(error);
					break;
			}
		});
```

```
}

createMulterOptions(): MulterModuleOptions {
        return {
                storage: diskStorage({
                        destination: (req, file, cb) => {
                                const folder = req?.headers?.folder_type ?? 'default';
                                this.ensureExists(`public/images/${folder}`);
                                cb(
                                        null,
                                        join(this.getRootPath(), `public/images/${folder}`),
                                );
                        },
                        filename: (req, file, cb) => {
                                //get image extension
                                let extName = path.extname(file.originalname);

                                //get image's name (without extension)
                                let baseName = path.basename(file.originalname, extName);

                                let finalName = `${baseName}-${Date.now()}${extName}`;
                                cb(null, finalName);
                        },
                }),
                fileFilter: (req, file, cb) => {
                        const allowedFileTypes = [
                                'jpg',
                                'jpeg',
                                'png',
                                'gif',
                                'pdf',
                                'doc',
                                'docx',
                        ];
                        const fileExtension = file.originalname
                                .split('.')
                                .pop()
                                .toLowerCase();
                        const isValidFileType =
                                allowedFileTypes.includes(fileExtension);
```

```
                                        if (!isValidFileType) {
                                                cb(
                                                        new HttpException(
                                                                'Invalid file type',
                                                                HttpStatus.UNPROCESSABLE_ENTITY,
                                                        ),
                                                        null,
                                                );
                                        } else cb(null, true);
                                },
                                limits: {
                                        fileSize: 1024 * 1024 * 1, // 1MB
                                },
                        };
                }
}
import {
        Controller,
        Get,
        Post,
        Body,
        Patch,
        Param,
        Delete,
        UseInterceptors,
        UploadedFile,
        ParseFilePipeBuilder,
        HttpStatus,
        UseFilters,
} from '@nestjs/common';
import { FilesService } from './files.service';
import { CreateFileDto } from './dto/create-file.dto';
import { UpdateFileDto } from './dto/update-file.dto';
import { FileInterceptor } from '@nestjs/platform-express';
import { Public, ResponseMessage } from 'src/decorator/customize';
import { ApiTags } from '@nestjs/swagger';
import { HttpExceptionFilter } from 'src/core/http-exception.filter';

@ApiTags('files')
@Controller('files')
export class FilesController {
```

```
	constructor(private readonly filesService: FilesService) {}

	@Public()
	@Post('upload')
	@ResponseMessage('Upload Single File')
	@UseInterceptors(FileInterceptor('fileUpload'))
	@UseFilters(new HttpExceptionFilter())
	uploadFile(@UploadedFile() file: Express.Multer.File) {
		return {
			fileName: file.filename,
		};
	}

	@Get()
	findAll() {
		return this.filesService.findAll();
	}

	@Get(':id')
	findOne(@Param('id') id: string) {
		return this.filesService.findOne(+id);
	}

	@Patch(':id')
	update(@Param('id') id: string, @Body() updateFileDto: UpdateFileDto) {
		return this.filesService.update(+id, updateFileDto);
	}

	@Delete(':id')
	remove(@Param('id') id: string) {
		return this.filesService.remove(+id);
	}
} import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
import mongoose, { HydratedDocument } from 'mongoose';

export type JobDocument = HydratedDocument<Job>;

@Schema({ timestamps: true })
export class Job {
	@Prop()
	name: string;
```

```
@Prop()
skills: string[];

@Prop({ type: Object })
company: {
        _id: mongoose.Schema.Types.ObjectId;
        name: string;
        logo: string;
};

@Prop()
location: string;

@Prop()
salary: number;

@Prop()
quantity: number;

@Prop()
level: string;

@Prop()
description: string;

@Prop()
startDate: Date;

@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()
	createdAt: Date;

	@Prop()
	updatedAt: Date;

	@Prop()
	isDeleted: boolean;

	@Prop()
	deletedAt: Date;
}

export const JobSchema = SchemaFactory.createForClass(Job) import { Controller, Get } from '@nestjs/common';
import { MailService } from './mail.service';
import { Public, ResponseMessage } from 'src/decorator/customize';
import { MailerService } from '@nestjs-modules/mailer';
import { SoftDeleteModel } from 'soft-delete-plugin-mongoose';
import {
	Subscriber,
	SubscriberDocument,
} from 'src/subscribers/schemas/subscriber.schema';
import { Job, JobDocument } from 'src/jobs/schemas/job.schema';
import { InjectModel } from '@nestjs/mongoose';
import { Cron, CronExpression } from '@nestjs/schedule';
import { ApiTags } from '@nestjs/swagger';

@ApiTags('mail')
@Controller('mail')
```

```
export class MailController {
	constructor(
		private readonly mailService: MailService,
		private mailerService: MailerService,

		@InjectModel(Subscriber.name)
		private subscriberModel: SoftDeleteModel<SubscriberDocument>,

		@InjectModel(Job.name)
		private jobModel: SoftDeleteModel<JobDocument>,
	) {}

	@Cron(CronExpression.EVERY_30_SECONDS)
	testCron() {}

	@Get()
	@Public()
	@ResponseMessage('Тестовий електронний лист')
	@Cron('0 10 0 * * 0') // 0.10' am every sunday
	async handleTestEmail() {
		const subscribers = await this.subscriberModel.find({});
		for (const subs of subscribers) {
			const subsSkills = subs.skills;
			const jobWithMatchingSkills = await this.jobModel.find({
				skills: { $in: subsSkills },
			});
			if (jobWithMatchingSkills?.length) {
				const jobs = jobWithMatchingSkills.map((item) => {
					return {
						name: item.name,
						company: item.company.name,
						salary:
							`${item.salary}`.replace(
								/\B(?=(\d{3})+(?!\d))/g,
								',',
							) + ' đ',
						skills: item.skills,
					};
				});

				await this.mailerService.sendMail({
```

```
                                        to: 'harypham97@gmail.com',
                                        from: '"Команда підтримки" <support@example.com>', // override default
from
                                        subject: 'Ласкаво просимо до приємного додатка!Підтвердьте свій
електронний лист',
                                        template: 'new-job',
                                        context: {
                                                receiver: subs.name,
                                                jobs: jobs,
                                        },
                                });
                        }
                }
        }
} import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
import mongoose, { HydratedDocument } from 'mongoose';
import { Company } from 'src/companies/schemas/company.schema';
import { Job } from 'src/jobs/schemas/job.schema';

export type PermissionDocument = HydratedDocument<Permission>;

@Schema({ timestamps: true })
export class Permission {
        @Prop()
        name: string;

        @Prop()
        apiPath: string;

        @Prop()
        method: string;

        @Prop()
        module: 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 PermissionSchema = SchemaFactory.createForClass(Permission); import { Prop, Schema, SchemaFactory }
from '@nestjs/mongoose';
import mongoose, { HydratedDocument } from 'mongoose';
import { Company } from 'src/companies/schemas/company.schema';
import { Job } from 'src/jobs/schemas/job.schema';

export type ResumeDocument = HydratedDocument<Resume>;

@Schema({ timestamps: true })
export class Resume {
    @Prop()
    email: string;

    @Prop()
    userId: mongoose.Schema.Types.ObjectId;
```

```
@Prop()
url: string;

@Prop()
status: string;

@Prop({ type: mongoose.Schema.Types.ObjectId, ref: Company.name })
companyId: mongoose.Schema.Types.ObjectId;

@Prop({ type: mongoose.Schema.Types.ObjectId, ref: Job.name })
jobId: mongoose.Schema.Types.ObjectId;

@Prop({ type: mongoose.Schema.Types.Array })
history: {
        status: string;
        updatedAt: Date;
        updatedBy: {
                _id: mongoose.Schema.Types.ObjectId;
                email: 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 ResumeSchema = SchemaFactory.createForClass(Resume); import { Prop, Schema, SchemaFactory } from
'@nestjs/mongoose';
import mongoose, { HydratedDocument } from 'mongoose';
import { Permission } from 'src/permissions/schemas/permission.schema';

export type RoleDocument = HydratedDocument<Role>;

@Schema({ timestamps: true })
export class Role {
    @Prop()
    name: string;

    @Prop()
    description: string;

    @Prop()
    isActive: boolean;

    @Prop({ type: [mongoose.Schema.Types.ObjectId], ref: Permission.name })
    permissions: Permission[];

    @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 RoleSchema = SchemaFactory.createForClass(Role); import { Prop, Schema, SchemaFactory } from
'@nestjs/mongoose';
import mongoose, { HydratedDocument } from 'mongoose';

export type SubscriberDocument = HydratedDocument<Subscriber>;

@Schema({ timestamps: true })
export class Subscriber {
    @Prop({ required: true })
    email: string;

    @Prop()
    name: string;

    @Prop()
    skills: 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 SubscriberSchema = SchemaFactory.createForClass(Subscriber); import { Prop, Schema, SchemaFactory }
from '@nestjs/mongoose';
import mongoose, { HydratedDocument } from 'mongoose';
import { Role } from 'src/roles/schemas/role.schema';

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); import { Prop, Schema, SchemaFactory } from
'@nestjs/mongoose';
import mongoose, { HydratedDocument } from 'mongoose';

export type CompanyDocument = HydratedDocument<Company>;

@Schema({ timestamps: true })
export class Company {
    @Prop()
    name: string;

    @Prop()
    address: string;

    @Prop()
    description: string;

    @Prop()
    logo: 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 CompanySchema = SchemaFactory.createForClass(Company); import {
    ExecutionContext,
    ForbiddenException,
    Injectable,
    UnauthorizedException,
} from '@nestjs/common';
import { Reflector } from '@nestjs/core';
import { AuthGuard } from '@nestjs/passport';
import { Request } from 'express';
import { IS_PUBLIC_KEY, IS_PUBLIC_PERMISSION } from 'src/decorator/customize';

@Injectable()
export class JwtAuthGuard extends AuthGuard('jwt') {
```

```
constructor(private reflector: Reflector) {
	super();
}

canActivate(context: ExecutionContext) {
	const isPublic = this.reflector.getAllAndOverride<boolean>(
		IS_PUBLIC_KEY,
		[context.getHandler(), context.getClass()],
	);
	if (isPublic) {
		return true;
	}
	return super.canActivate(context);
}

handleRequest(err, user, info, context: ExecutionContext) {
	const request: Request = context.switchToHttp().getRequest();

	const isSkipPermission = this.reflector.getAllAndOverride<boolean>(
		IS_PUBLIC_PERMISSION,
		[context.getHandler(), context.getClass()],
	);

	// You can throw an exception based on either "info" or "err" arguments
	if (err || !user) {
		throw (
			err ||
			new UnauthorizedException(
				'Token không hợp lệ or không có token ở Bearer Token ở Header request!',
			)
		);
	}

	//check permissions
	const targetMethod = request.method;
	const targetEndpoint = request.route?.path as string;

	const permissions = user?.permissions ?? [];
	let isExist = permissions.find(
		(permission) =>
			targetMethod === permission.method &&
```

```
                        targetEndpoint === permission.apiPath,
                    );
                    if (targetEndpoint.startsWith('/api/v1/auth')) isExist = true;
                    if (!isExist && !isSkipPermission) {
                        throw new ForbiddenException(
                            'Bạn không có quyền để truy cập endpoint này!',
                        );
                    }

                    return user;
                }
} import { ExtractJwt, Strategy } from 'passport-jwt';
import { PassportStrategy } from '@nestjs/passport';
import { Injectable } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import { IUser } from 'src/users/users.interface';
import { RolesService } from 'src/roles/roles.service';

@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {
  constructor(
    private configService: ConfigService,
    private rolesService: RolesService
  ) {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      ignoreExpiration: false,
      secretOrKey: configService.get<string>("JWT_ACCESS_TOKEN_SECRET"),
    });
  }

  async validate(payload: IUser) {
    const { _id, name, email, role } = payload;
    // cần gán thêm permissions vào req.user
    const userRole = role as unknown as { _id: string; name: string }
    const temp = (await this.rolesService.findOne(userRole._id)).toObject();

    //req.user
    return {
      _id,
      name,
```

```
            email,
            role,
            permissions: temp?.permissions ?? []
        };
    }

}
import {
            Controller,
            Post,
            UseGuards,
            Req,
            Body,
            Res,
            Get,
} from '@nestjs/common';
import { AuthService } from './auth.service';
import { Public, ResponseMessage, User } from 'src/decorator/customize';
import { LocalAuthGuard } from './local-auth.guard';
import { RegisterUserDto, UserLoginDto } from 'src/users/dto/create-user.dto';
import { Request, Response } from 'express';
import { IUser } from 'src/users/users.interface';
import { RolesService } from 'src/roles/roles.service';
import { ThrottlerGuard, Throttle } from '@nestjs/throttler';
import { ApiTags, ApiBody } from '@nestjs/swagger';

@ApiTags('auth')
@Controller('auth')
export class AuthController {
            constructor(
                        private authService: AuthService,
                        private rolesService: RolesService,
            ) {}

            @Public()
            @UseGuards(LocalAuthGuard)
            @UseGuards(ThrottlerGuard)
            @Throttle(5, 60)
            @ApiBody({ type: UserLoginDto })
            @Post('/login')
            @ResponseMessage('User Login')
```

```
    handleLogin(@Req() req, @Res({ passthrough: true }) response: Response) {
        return this.authService.login(req.user, response);
    }

    @Public()
    @ResponseMessage('Register a new user')
    @Post('/register')
    handleRegister(@Body() registerUserDto: RegisterUserDto) {
        return this.authService.register(registerUserDto);
    }

    @ResponseMessage('Get user information')
    @Get('/account')
    async handleGetAccount(@User() user: IUser) {
        const temp = (await this.rolesService.findOne(user.role._id)) as any;
        user.permissions = temp.permissions;
        return { user };
    }

    @Public()
    @ResponseMessage('Get User by refresh token')
    @Get('/refresh')
    handleRefreshToken(
        @Req() request: Request,
        @Res({ passthrough: true }) response: Response,
    ) {
        const refreshToken = request.cookies['refresh_token'];
        return this.authService.processNewToken(refreshToken, response);
    }

    @ResponseMessage('Logout User')
    @Post('/logout')
    handleLogout(
        @Res({ passthrough: true }) response: Response,
        @User() user: IUser,
    ) {
        return this.authService.logout(response, user);
    }
}
import { BadRequestException, Injectable } from '@nestjs/common';
import { UsersService } from 'src/users/users.service';
```

```
import { JwtService } from '@nestjs/jwt';
import { IUser } from 'src/users/users.interface';
import { RegisterUserDto } from 'src/users/dto/create-user.dto';
import { ConfigService } from '@nestjs/config';
import ms from 'ms';
import { Response } from 'express';
import { RolesService } from 'src/roles/roles.service';

@Injectable()
export class AuthService {
	constructor(
		private usersService: UsersService,
		private jwtService: JwtService,
		private configService: ConfigService,
		private rolesService: RolesService,
	) {}

	//ussername/ pass là 2 tham số thư viện passport nó ném về
	async validateUser(username: string, pass: string): Promise<any> {
		const user = await this.usersService.findOneByUsername(username);
		if (user) {
			const isValid = this.usersService.isValidPassword(
				pass,
				user.password,
			);
			if (isValid === true) {
				const userRole = user.role as unknown as {
					_id: string;
					name: string;
				};
				const temp = await this.rolesService.findOne(userRole._id);

				const objUser = {
					...user.toObject(),
					permissions: temp?.permissions ?? [],
				};

				return objUser;
			}
		}
```

```
        return null;
}

async login(user: IUser, response: Response) {
        const { _id, name, email, role, permissions } = user;
        const payload = {
                sub: 'token login',
                iss: 'from server',
                _id,
                name,
                email,
                role,
        };

        const refresh_token = this.createRefreshToken(payload);

        //update user with refresh token
        await this.usersService.updateUserToken(refresh_token, _id);

        //set refresh_token as cookies
        response.cookie('refresh_token', refresh_token, {
                httpOnly: true,
                maxAge: ms(this.configService.get<string>('JWT_REFRESH_EXPIRE')),
        });

        return {
                access_token: this.jwtService.sign(payload),
                user: {
                        _id,
                        name,
                        email,
                        role,
                        permissions,
                },
        };
}

async register(user: RegisterUserDto) {
        let newUser = await this.usersService.register(user);

        return {
```

```
                _id: newUser?._id,
                createdAt: newUser?.createdAt,
        };
}

createRefreshToken = (payload: any) => {
        const refresh_token = this.jwtService.sign(payload, {
                secret: this.configService.get<string>('JWT_REFRESH_TOKEN_SECRET'),
                expiresIn: 10000,
        });
        return refresh_token;
};

processNewToken = async (refreshToken: string, response: Response) => {
        try {
                this.jwtService.verify(refreshToken, {
                        secret: this.configService.get<string>(
                                'JWT_REFRESH_TOKEN_SECRET',
                        ),
                });
                let user = await this.usersService.findUserByToken(refreshToken);
                if (user) {
                        const { _id, name, email, role } = user;
                        const payload = {
                                sub: 'token refresh',
                                iss: 'from server',
                                _id,
                                name,
                                email,
                                role,
                        };

                        const refresh_token = this.createRefreshToken(payload);

                        //update user with refresh token
                        await this.usersService.updateUserToken(
                                refresh_token,
                                _id.toString(),
                        );

                        //fetch user's role
```

```
                const userRole = user.role as unknown as {
                    _id: string;
                    name: string;
                };
                const temp = await this.rolesService.findOne(userRole._id);

                //set refresh_token as cookies
                response.clearCookie('refresh_token');

                response.cookie('refresh_token', refresh_token, {
                    httpOnly: true,
                    maxAge: ms(
                        this.configService.get<string>('JWT_REFRESH_EXPIRE'),
                    ),
                });

                return {
                    access_token: this.jwtService.sign(payload),
                    user: {
                        _id,
                        name,
                        email,
                        role,
                        permissions: temp?.permissions ?? [],
                    },
                };
            } else {
                throw new BadRequestException(
                    `Refresh token không hợp lệ. Vui lòng login.`,
                );
            }
        } catch (error) {
            throw new BadRequestException(
                `Refresh token không hợp lệ. Vui lòng login.`,
            );
        }
};

logout = async (response: Response, user: IUser) => {
        await this.usersService.updateUserToken('', user._id);
        response.clearCookie('refresh_token');
```

```
                return 'ok';
        };
}import { Injectable } from '@nestjs/common';
import { CreateJobDto } from './dto/create-job.dto';
import { UpdateJobDto } from './dto/update-job.dto';
import { SoftDeleteModel } from 'soft-delete-plugin-mongoose';
import { JobDocument, Job } from './schemas/job.schema';
import { InjectModel } from '@nestjs/mongoose';
import { IUser } from 'src/users/users.interface';
import mongoose from 'mongoose';
import aqp from 'api-query-params';

@Injectable()
export class JobsService {
        constructor(
                @InjectModel(Job.name)
                private jobModel: SoftDeleteModel<JobDocument>,
        ) {}

        async create(createJobDto: CreateJobDto, user: IUser) {
                const {
                        name,
                        skills,
                        company,
                        salary,
                        quantity,
                        level,
                        description,
                        startDate,
                        endDate,
                        isActive,
                        location,
                } = createJobDto;

                let newJob = await this.jobModel.create({
                        name,
                        skills,
                        company,
                        salary,
                        quantity,
                        level,
```

```
                description,
                startDate,
                endDate,
                isActive,
                location,
                createdBy: {
                        _id: user._id,
                        email: user.email,
                },
        });

        return {
                _id: newJob?._id,
                createdAt: newJob?.createdAt,
        };
}

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,
        });
```

```
        }
}
import { BadRequestException, Injectable } from '@nestjs/common';
import { CreatePermissionDto } from './dto/create-permission.dto';
import { UpdatePermissionDto } from './dto/update-permission.dto';
import { IUser } from 'src/users/users.interface';
import { InjectModel } from '@nestjs/mongoose';
import { SoftDeleteModel } from 'soft-delete-plugin-mongoose';
import { Permission, PermissionDocument } from './schemas/permission.schema';
import aqp from 'api-query-params';
import mongoose from 'mongoose';

@Injectable()
export class PermissionsService {
        constructor(
                @InjectModel(Permission.name)
                private permissionModel: SoftDeleteModel<PermissionDocument>,
        ) {}

        async create(createPermissionDto: CreatePermissionDto, user: IUser) {
                const { name, apiPath, method, module } = createPermissionDto;

                const isExist = await this.permissionModel.findOne({ apiPath, method });
                if (isExist) {
                        throw new BadRequestException(
                                `Дозвіл на apiPath=${apiPath} , method=${method} існував!`,
                        );
                }

                const newPermission = await this.permissionModel.create({
                        name,
                        apiPath,
                        method,
                        module,
                        createdBy: {
                                _id: user._id,
                                email: user.email,
                        },
                });

                return {
```

```
                _id: newPermission?._id,
                createdAt: newPermission?.createdAt,
        };
}

async findAll(currentPage: number, limit: number, qs: string) {
        const { filter, sort, population, projection } = aqp(qs);
        delete filter.current;
        delete filter.pageSize;

        let offset = (+currentPage - 1) * +limit;
        let defaultLimit = +limit ? +limit : 10;

        const totalItems = (await this.permissionModel.find(filter)).length;
        const totalPages = Math.ceil(totalItems / defaultLimit);

        const result = await this.permissionModel
                .find(filter)
                .skip(offset)
                .limit(defaultLimit)
                .sort(sort as any)
                .populate(population)
                .select(projection as any)
                .exec();

        return {
                meta: {
                        current: currentPage,
                        pageSize: limit,
                        pages: totalPages,
                        total: totalItems,
                },
                result,
        };
}

async findOne(id: string) {
        if (!mongoose.Types.ObjectId.isValid(id)) {
                throw new BadRequestException('не знайдено дозволу');
        }
```

```
        return await this.permissionModel.findById(id);
}

async update(
        _id: string,
        updatePermissionDto: UpdatePermissionDto,
        user: IUser,
) {
        if (!mongoose.Types.ObjectId.isValid(_id)) {
                throw new BadRequestException('не знайдено дозволу');
        }
        const { module, method, apiPath, name } = updatePermissionDto;

        const updated = await this.permissionModel.updateOne(
                { _id },
                {
                        module,
                        method,
                        apiPath,
                        name,
                        updatedBy: {
                                _id: user._id,
                                email: user.email,
                        },
                },
        );
        return updated;
}

async remove(id: string, user: IUser) {
        await this.permissionModel.updateOne(
                { _id: id },
                {
                        deletedBy: {
                                _id: user._id,
                                email: user.email,
                        },
                },
        );
        return this.permissionModel.softDelete({
                _id: id,
```

```
                });
        }
}
import { BadRequestException, Injectable } from '@nestjs/common';
import { CreateRoleDto } from './dto/create-role.dto';
import { UpdateRoleDto } from './dto/update-role.dto';
import { Role, RoleDocument } from './schemas/role.schema';
import { InjectModel } from '@nestjs/mongoose';
import { SoftDeleteModel } from 'soft-delete-plugin-mongoose';
import { IUser } from 'src/users/users.interface';
import aqp from 'api-query-params';
import mongoose from 'mongoose';
import { ADMIN_ROLE } from 'src/databases/sample';

@Injectable()
export class RolesService {
        constructor(
                @InjectModel(Role.name)
                private roleModel: SoftDeleteModel<RoleDocument>,
        ) {}

        async create(createRoleDto: CreateRoleDto, user: IUser) {
                const { name, description, isActive, permissions } = createRoleDto;

                const isExist = await this.roleModel.findOne({ name });
                if (isExist) {
                        throw new BadRequestException(
                                `Role với name="${name}" đã tồn tại!`,
                        );
                }

                const newRole = await this.roleModel.create({
                        name,
                        description,
                        isActive,
                        permissions,
                        createdBy: {
                                _id: user._id,
                                email: user.email,
                        },
                });
```

```
        return {
            _id: newRole?._id,
            createdAt: newRole?.createdAt,
        };
}

async findAll(currentPage: number, limit: number, qs: string) {
        const { filter, sort, population, projection } = aqp(qs);
        delete filter.current;
        delete filter.pageSize;

        let offset = (+currentPage - 1) * +limit;
        let defaultLimit = +limit ? +limit : 10;

        const totalItems = (await this.roleModel.find(filter)).length;
        const totalPages = Math.ceil(totalItems / defaultLimit);

        const result = await this.roleModel
            .find(filter)
            .skip(offset)
            .limit(defaultLimit)
            .sort(sort as any)
            .populate(population)
            .select(projection as any)
            .exec();

        return {
            meta: {
                current: currentPage, //trang hiện tại
                pageSize: limit, //số lượng bản ghi đã lấy
                pages: totalPages, //tổng số trang với điều kiện query
                total: totalItems, // tổng số phần tử (số bản ghi)
            },
            result, //kết quả query
        };
}

async findOne(id: string) {
        if (!mongoose.Types.ObjectId.isValid(id)) {
            throw new BadRequestException('not found role');
```

```
        }

        return (await this.roleModel.findById(id)).populate({
                path: 'permissions',
                select: { _id: 1, apiPath: 1, name: 1, method: 1, module: 1 },
        });
}

async update(_id: string, updateRoleDto: UpdateRoleDto, user: IUser) {
        if (!mongoose.Types.ObjectId.isValid(_id)) {
                throw new BadRequestException('not found role');
        }

        const { name, description, isActive, permissions } = updateRoleDto;

        // const isExist = await this.roleModel.findOne({ name });
        // if (isExist) {
        //   throw new BadRequestException(`Role với name=${name} đã tồn tại!`)
        // }

        const updated = await this.roleModel.updateOne(
                { _id },
                {
                        name,
                        description,
                        isActive,
                        permissions,
                        updatedBy: {
                                _id: user._id,
                                email: user.email,
                        },
                },
        );

        return updated;
}

async remove(id: string, user: IUser) {
        const foundRole = await this.roleModel.findById(id);
        if (foundRole.name === ADMIN_ROLE) {
                throw new BadRequestException('Không thể xóa role ADMIN');
```

```
                }
                await this.roleModel.updateOne(
                        { _id: id },
                        {
                                deletedBy: {
                                        _id: user._id,
                                        email: user.email,
                                },
                        },
                );
                return this.roleModel.softDelete({
                        _id: id,
                });
        }
}
import { BadRequestException, Injectable } from '@nestjs/common';
import { CreateUserDto, RegisterUserDto } from './dto/create-user.dto';
import { UpdateUserDto } from './dto/update-user.dto';
import { InjectModel } from '@nestjs/mongoose';
import { User as UserM, UserDocument } from './schemas/user.schema';
import mongoose, { Model } from 'mongoose';
import { genSaltSync, hashSync, compareSync } from 'bcryptjs';
import { SoftDeleteModel } from 'soft-delete-plugin-mongoose';
import { IUser } from './users.interface';
import { User } from 'src/decorator/customize';
import aqp from 'api-query-params';
import { USER_ROLE } from 'src/databases/sample';
import { Role, RoleDocument } from 'src/roles/schemas/role.schema';

@Injectable()
export class UsersService {
        constructor(
                @InjectModel(UserM.name)
                private userModel: SoftDeleteModel<UserDocument>,

                @InjectModel(Role.name)
                private roleModel: SoftDeleteModel<RoleDocument>,
        ) {}

        getHashPassword = (password: string) => {
                const salt = genSaltSync(10);
```

```
        const hash = hashSync(password, salt);
        return hash;
};

async create(createUserDto: CreateUserDto, @User() user: IUser) {
        const { name, email, password, age, gender, address, role, company } =
                createUserDto;

        //add logic check email
        const isExist = await this.userModel.findOne({ email });
        if (isExist) {
                throw new BadRequestException(
                        `Email: ${email} đã tồn tại trên hệ thống. Vui lòng sử dụng email khác.`,
                );
        }

        const hashPassword = this.getHashPassword(password);

        let newUser = await this.userModel.create({
                name,
                email,
                password: hashPassword,
                age,
                gender,
                address,
                role,
                company,
                createdBy: {
                        _id: user._id,
                        email: user.email,
                },
        });
        return newUser;
}

async register(user: RegisterUserDto) {
        const { name, email, password, age, gender, address } = user;
        //add logic check email
        const isExist = await this.userModel.findOne({ email });
        if (isExist) {
                throw new BadRequestException(
```

```
                    `Email: ${email} đã tồn tại trên hệ thống. Vui lòng sử dụng email khác.`,
                );
        }

        //fetch user role
        const userRole = await this.roleModel.findOne({ name: USER_ROLE });

        const hashPassword = this.getHashPassword(password);
        let newRegister = await this.userModel.create({
                name,
                email,
                password: hashPassword,
                age,
                gender,
                address,
                role: userRole?._id,
        });
        return newRegister;
}

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.userModel.find(filter)).length;
        const totalPages = Math.ceil(totalItems / defaultLimit);

        const result = await this.userModel
                .find(filter)
                .skip(offset)
                .limit(defaultLimit)
                .sort(sort as any)
                .select('-password')
                .populate(population)
                .exec();

        return {
```

```
                meta: {
                        current: currentPage, //trang hiện tại
                        pageSize: limit, //số lượng bản ghi đã lấy
                        pages: totalPages, //tổng số trang với điều kiện query
                        total: totalItems, // tổng số phần tử (số bản ghi)
                },
                result, //kết quả query
        };
}

async findOne(id: string) {
        if (!mongoose.Types.ObjectId.isValid(id)) return `not found user`;

        return await this.userModel
                .findOne({
                        _id: id,
                })
                .select('-password')
                .populate({ path: 'role', select: { name: 1, _id: 1 } });

        //exclude >< include
}

findOneByUsername(username: string) {
        return this.userModel
                .findOne({
                        email: username,
                })
                .populate({
                        path: 'role',
                        select: { name: 1 },
                });
}

isValidPassword(password: string, hash: string) {
        return compareSync(password, hash);
}

async update(updateUserDto: UpdateUserDto, user: IUser) {
        const updated = await this.userModel.updateOne(
                { _id: updateUserDto._id },
```

```
                {
                        ...updateUserDto,
                        updatedBy: {
                                _id: user._id,
                                email: user.email,
                        },
                },
        );
        return updated;
}

async remove(id: string, user: IUser) {
        if (!mongoose.Types.ObjectId.isValid(id)) return `not found user`;

        const foundUser = await this.userModel.findById(id);
        if (foundUser && foundUser.email === 'admin@gmail.com') {
                throw new BadRequestException(
                        'Không thể xóa tài khoản admin@gmail.com',
                );
        }

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

updateUserToken = async (refreshToken: string, _id: string) => {
        return await this.userModel.updateOne({ _id }, { refreshToken });
};

findUserByToken = async (refreshToken: string) => {
        return await this.userModel.findOne({ refreshToken }).populate({
```

```
                path: 'role',
                select: { name: 1 },
            });
        };
}
import { BadRequestException, Injectable } from '@nestjs/common';
import { CreateSubscriberDto } from './dto/create-subscriber.dto';
import { UpdateSubscriberDto } from './dto/update-subscriber.dto';
import { IUser } from 'src/users/users.interface';
import { InjectModel } from '@nestjs/mongoose';
import { Subscriber, SubscriberDocument } from './schemas/subscriber.schema';
import { SoftDeleteModel } from 'soft-delete-plugin-mongoose';
import aqp from 'api-query-params';
import mongoose from 'mongoose';

@Injectable()
export class SubscribersService {
    constructor(
        @InjectModel(Subscriber.name)
        private subscriberModel: SoftDeleteModel<SubscriberDocument>,
    ) {}

    async create(createSubscriberDto: CreateSubscriberDto, user: IUser) {
        const { name, email, skills } = createSubscriberDto;
        const isExist = await this.subscriberModel.findOne({ email });
        if (isExist) {
            throw new BadRequestException(
                `Email: ${email} đã tồn tại trên hệ thống. Vui lòng sử dụng email khác.`,
            );
        }

        let newSubs = await this.subscriberModel.create({
            name,
            email,
            skills,
            createdBy: {
                _id: user._id,
                email: user.email,
            },
        });
```

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»
Стаднік Е. 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

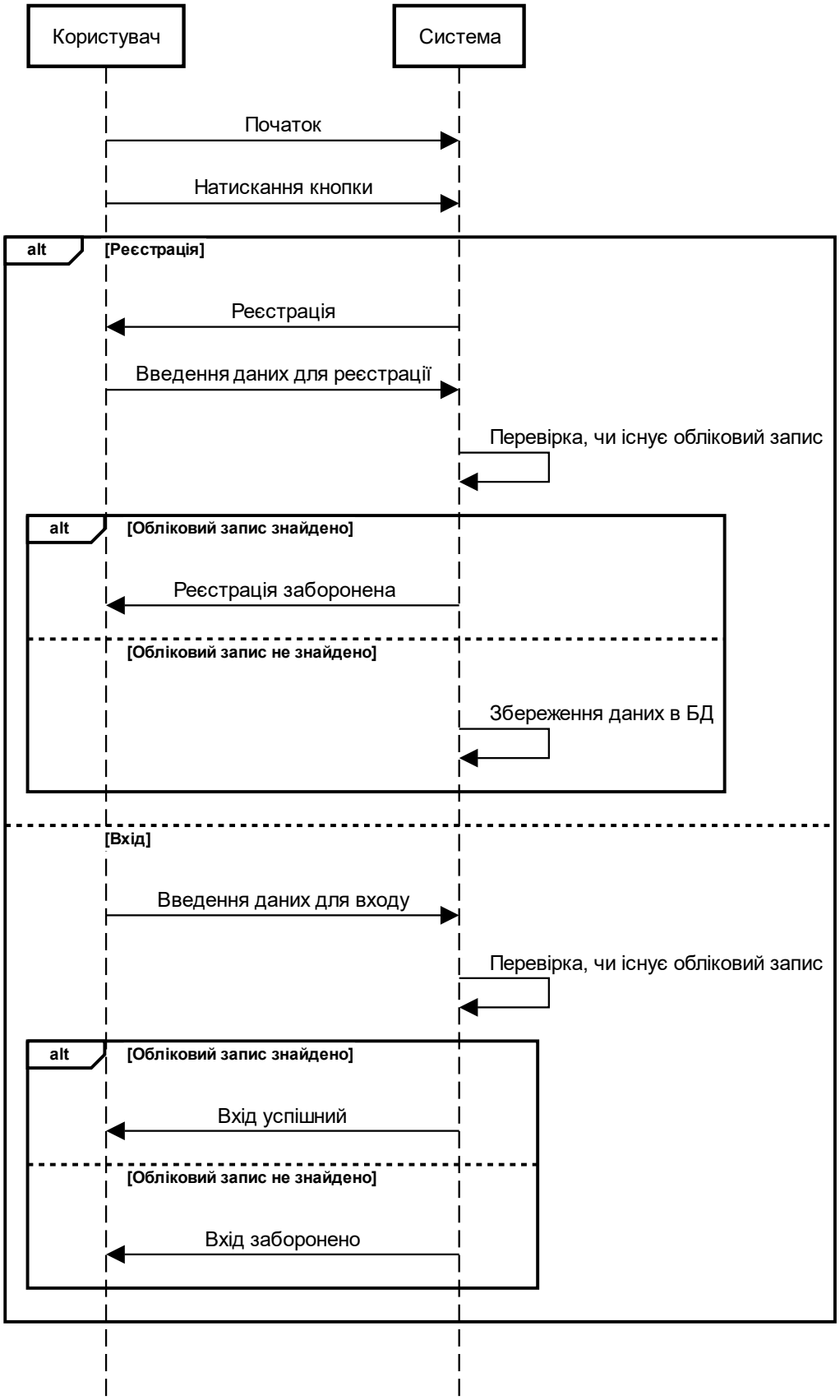

Рисунок В.1 – UML-діаграма послідовності взаємодії модулів інформаційної технології для реєстрації або входу в систему

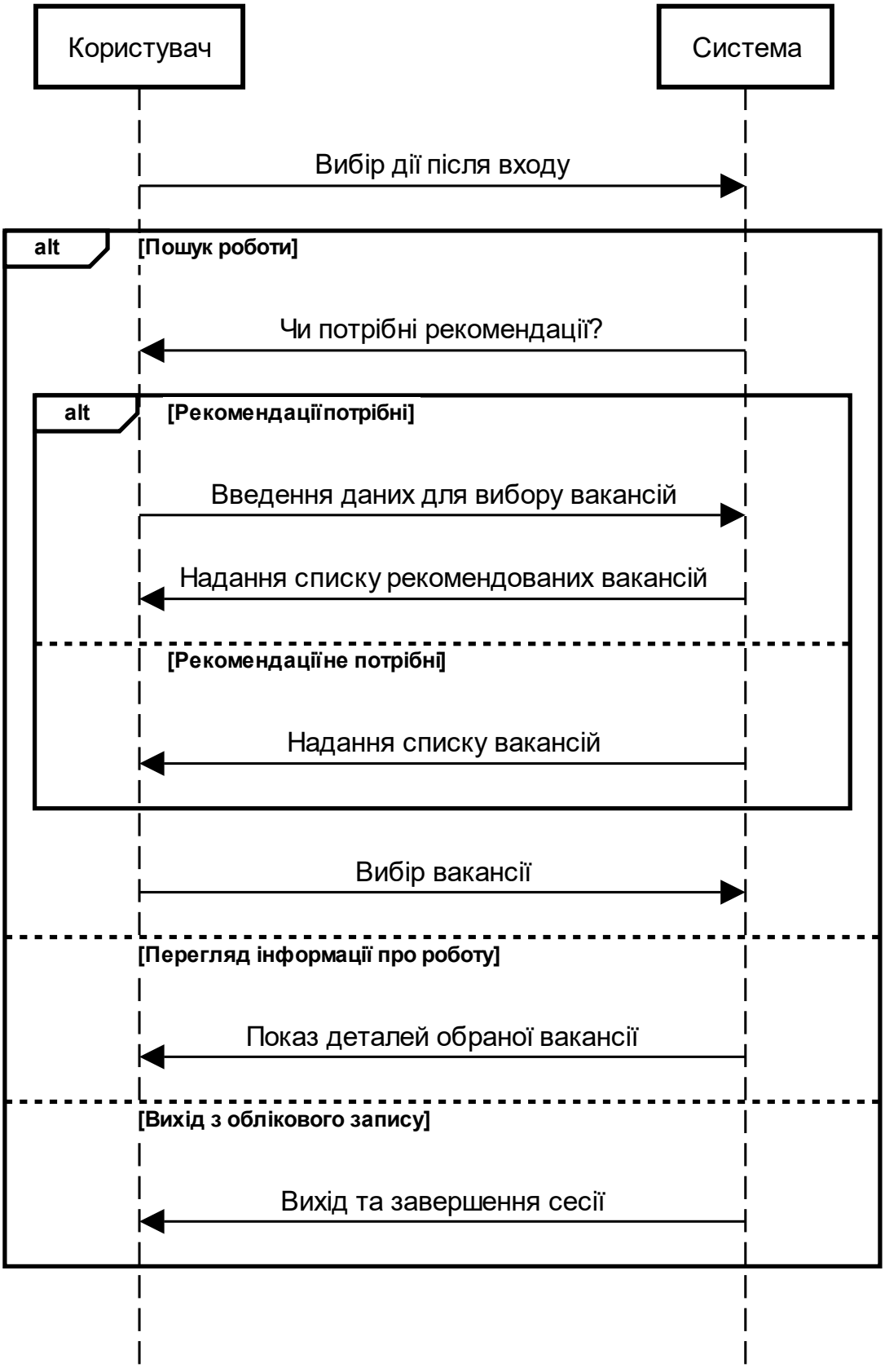

Рисунок В.2 – UML-діаграма послідовності взаємодії з модулем надання рекомендацій

Рисунок В.3 – Структура інформаційної технології надання рекомендацій для пошуку роботи

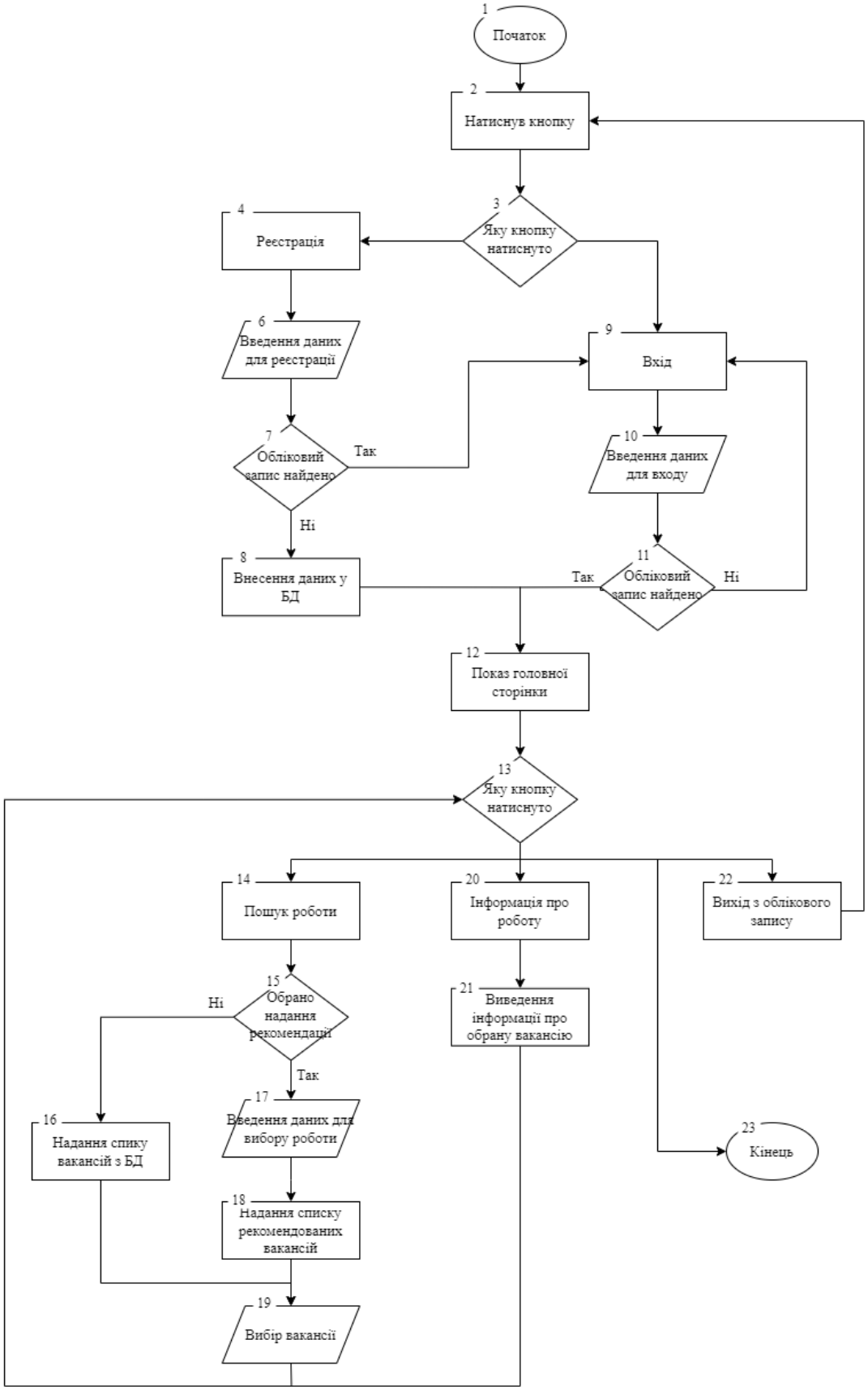

Рисунок В.4 – Схема алгоритму роботи інформаційної технології надання рекомендацій для пошуку роботи

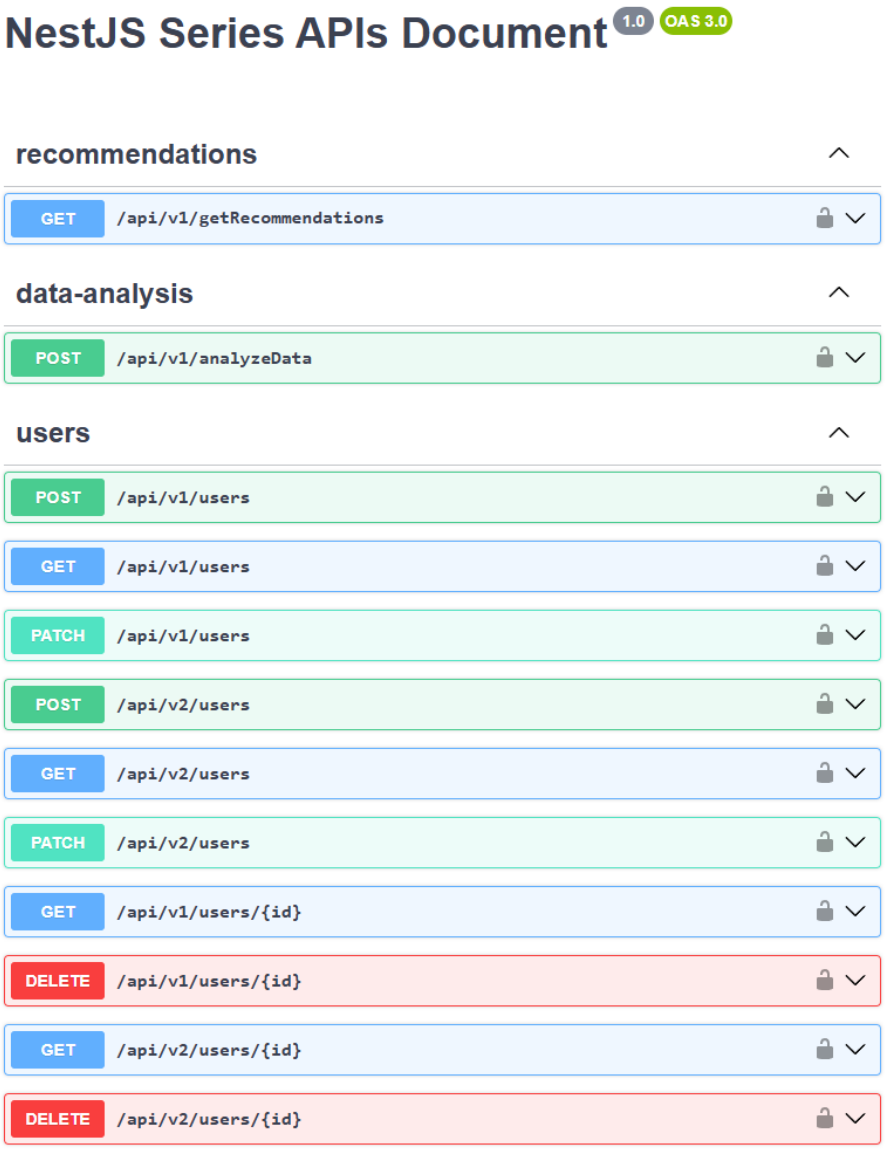

Рисунок В.5 – Серверна частина інформаційної технології у Swagger

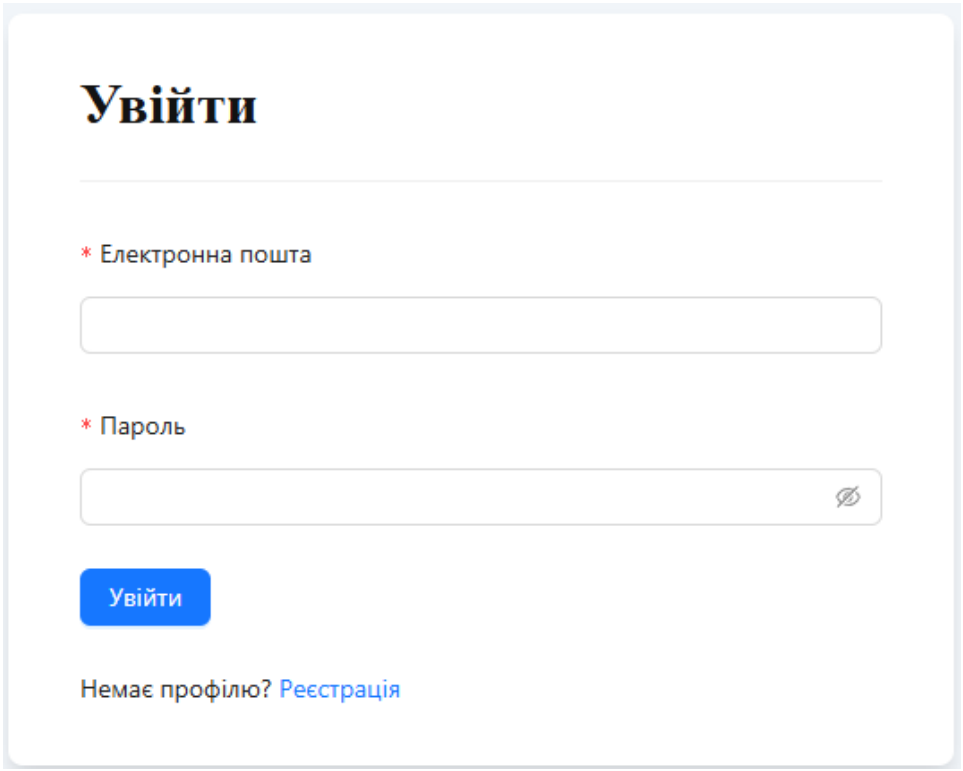

Рисунок В.6 – Загальний вигляд інтерфейсного вікна авторизації користувача

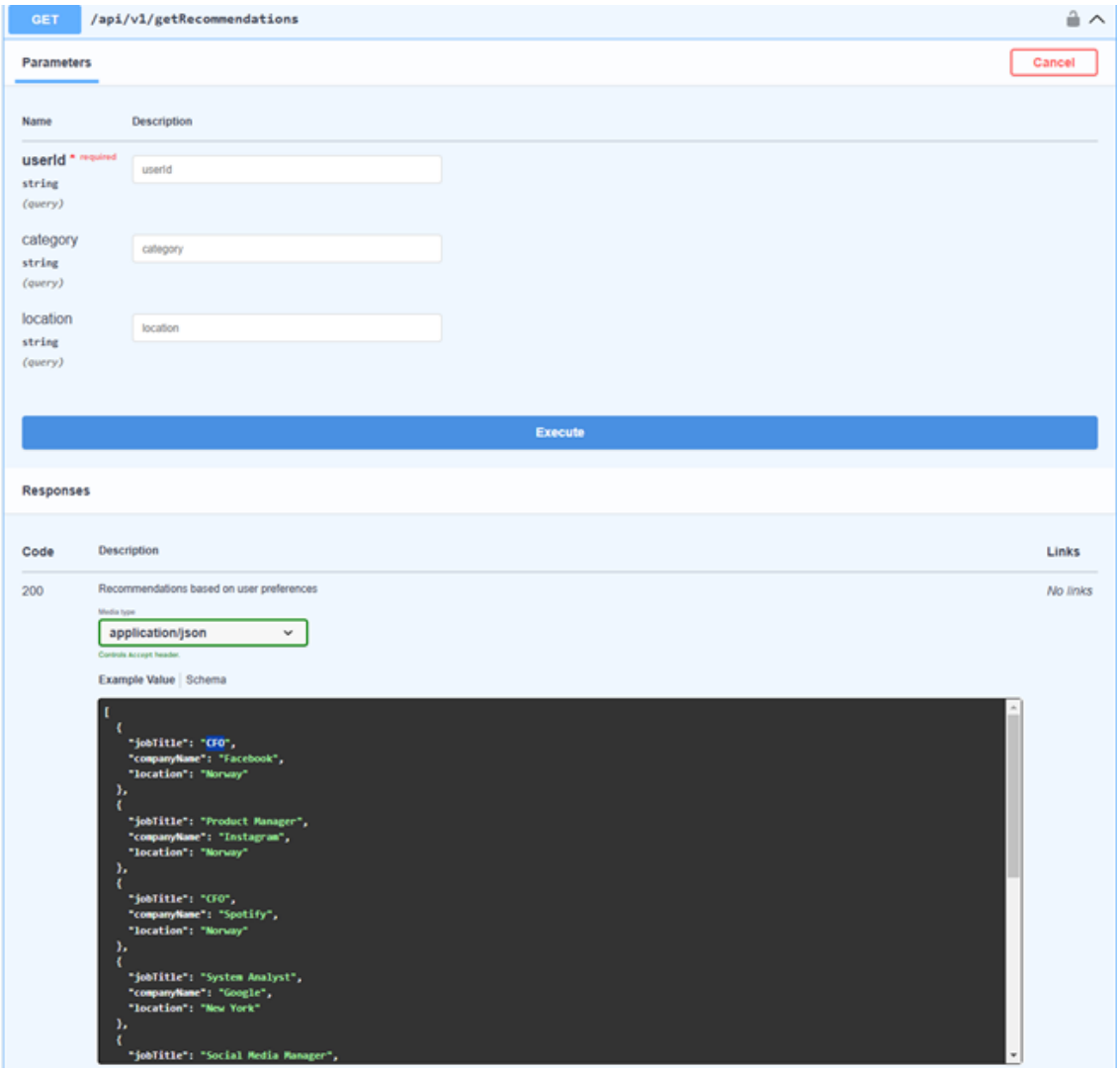

Рисунок В.7 – Інтерфейс для тестування API рекомендацій у Swagger

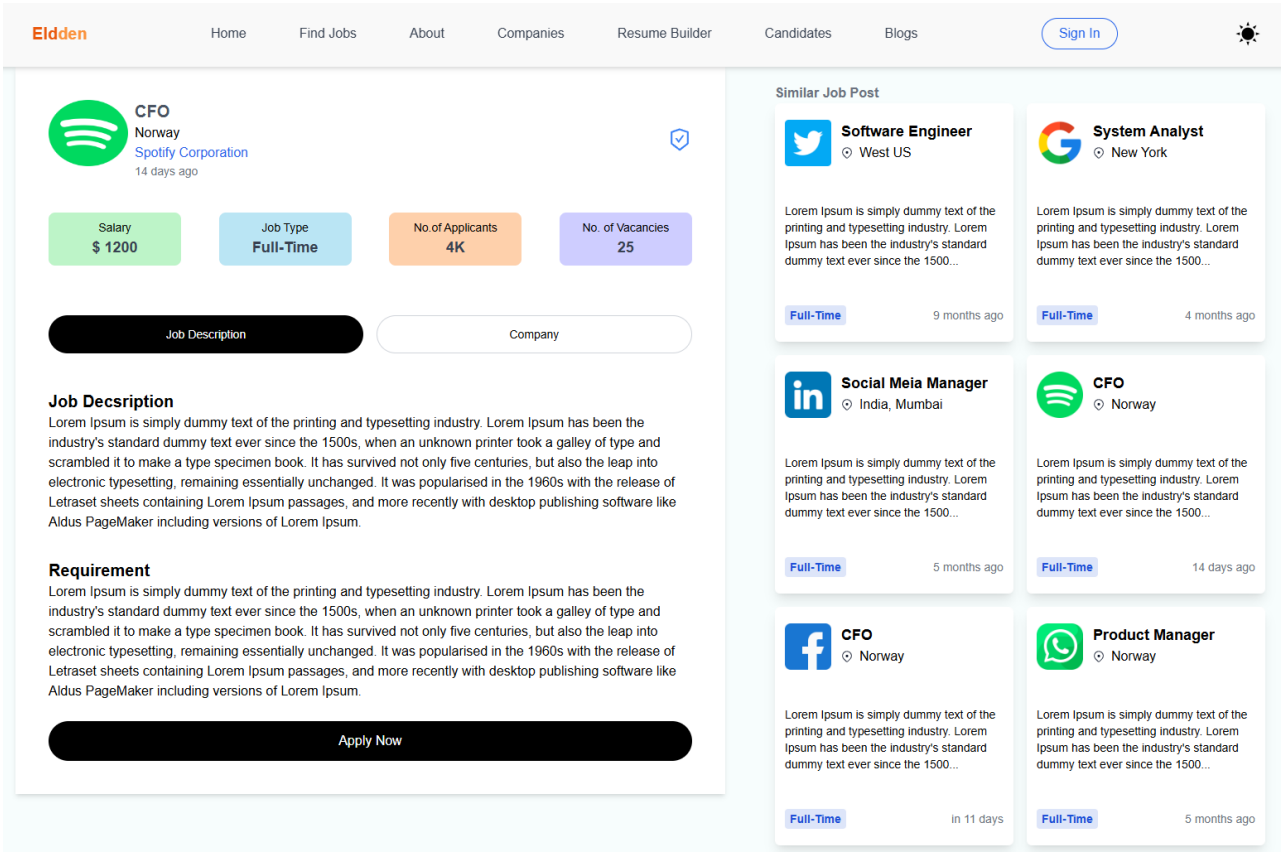

Рисунок В.8 – Інтерфейс перегляду вакансії та подібних пропозицій

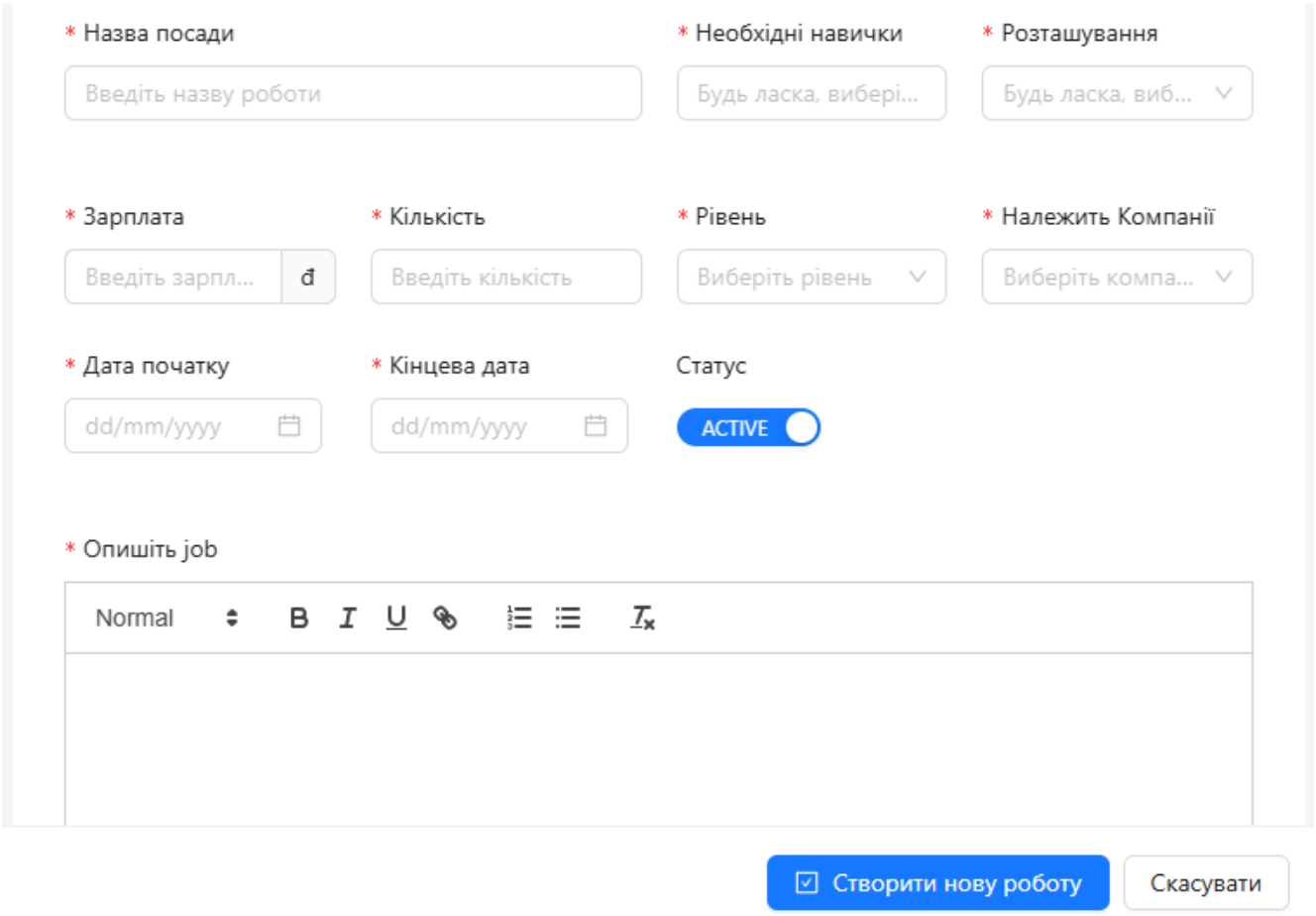

Рисунок В.9 – Інтерфейс перегляду вакансії та подібних пропозицій

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).

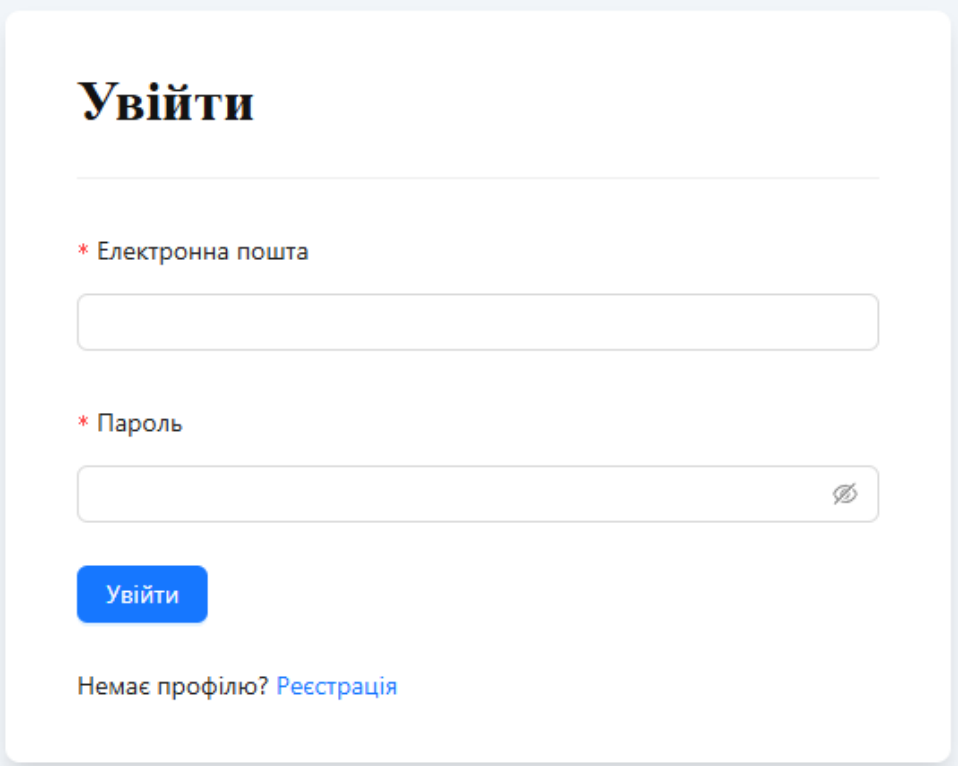

Рисунок Г.1 – Загальний вигляд інтерфейсного вікна авторизації користувача

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.

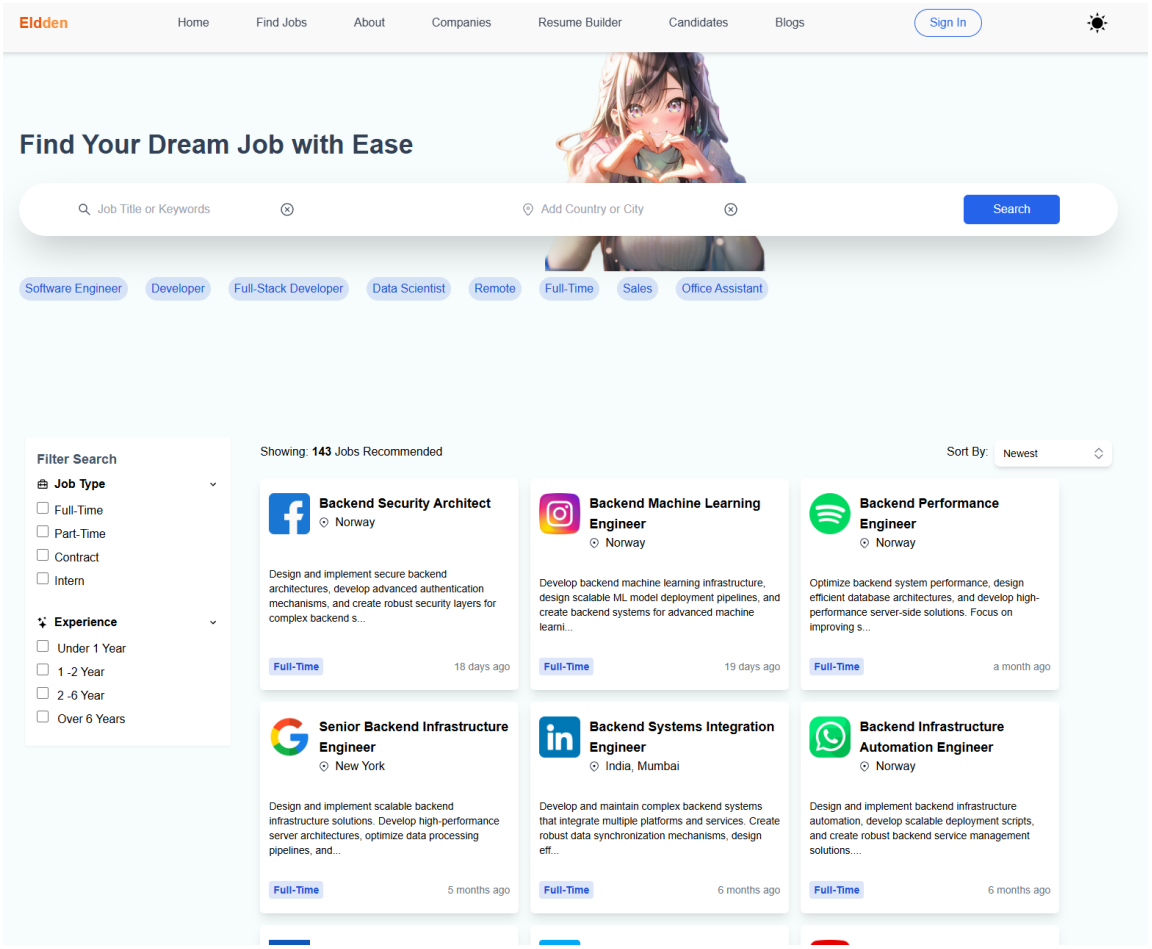

Рисунок Г.2 – Загальний вигляд блоку з вакансіями

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.

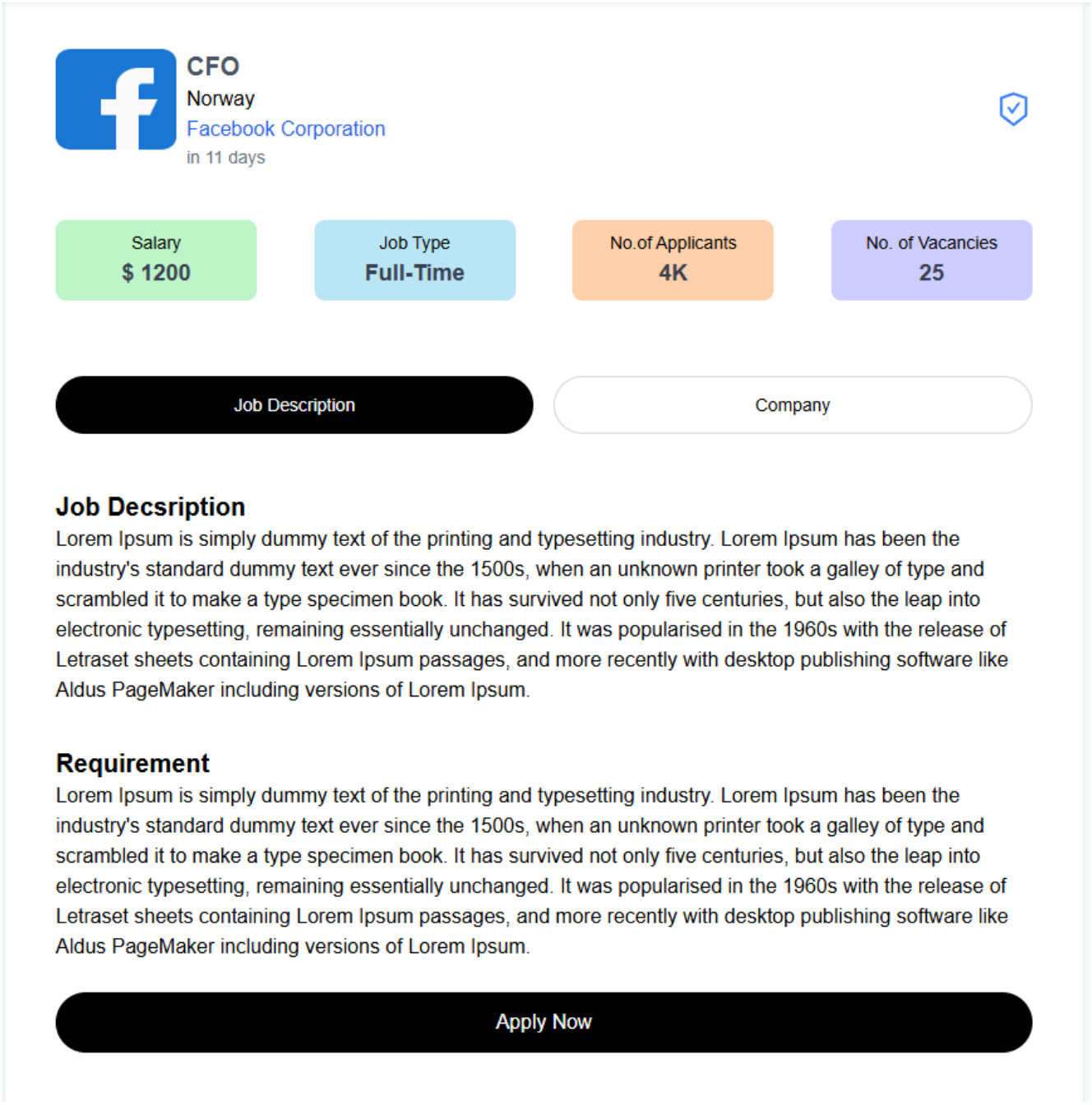

Рисунок Г.3 – Вигляд вікна інформації про вакансію

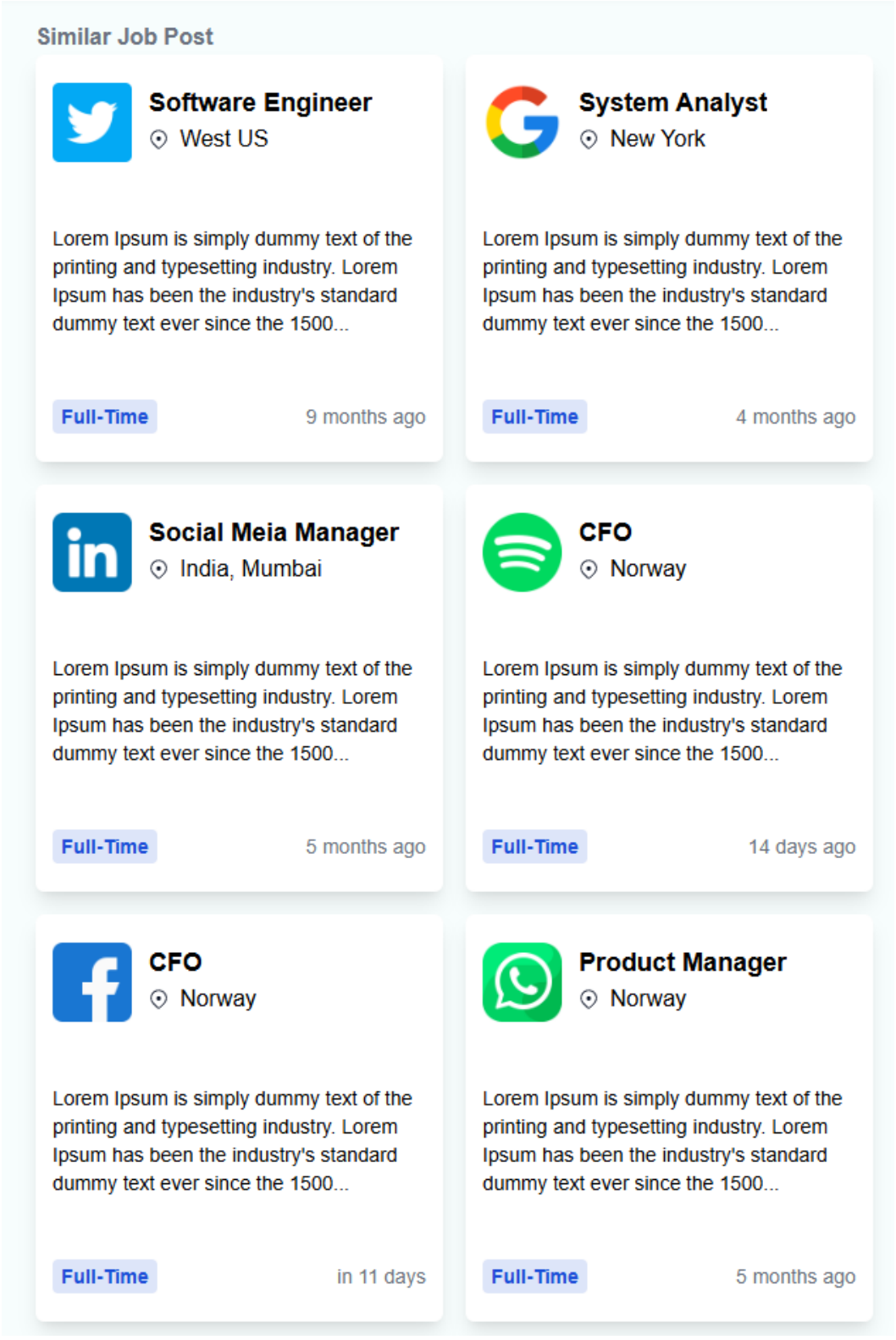

Рисунок Г.4 – Вигляд списку схожих вакансій