Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Факультет комп’ютерних наук

Рекомендовано до захисту:

протокол засідання кафедри

№ ___ від ____.____.2024 р.

Завідувач кафедри _____

_____ _____

(підпис) (прізвище та ініціали)

Пояснювальна записка

до кваліфікаційної роботи бакалавра

на тему «Розробка веб-платформи для працевлаштування співробітників з використанням можливостей LLM моделей»

Захищено на засіданні ЕК № _____

протокол № ____ від ____.____.2024 р.

Оцінка _____ / _____

Голова ЕК

_____ _____

(підпис) (прізвище та ініціали)

Виконав:

студент 4 курсу, групи КС- 41

Спеціальності:

122 Комп’ютерні науки

(шифр і назва спеціальності підготовки)

Бутеньов Матвій Сергійович

(прізвище, ім’я та по батькові, підпис)

Керівник PhD, старший викладач Матвєєв О.М.

(ступень, звання, прізвище та ініціали, підпис)

Рецензент _____

_____

(ступень, звання, прізвище та ініціали, підпис)

Харків – 2025

АНОТАЦІЯ

Кваліфікаційна робота містить: 77 сторінку, 19 рисунків, 2 таблиці, 6 додатків та 25 джерел.

Метою дослідження є створення веб-платформи для працевлаштування співробітників, яка об’єднує управління вакансіями, кандидатами та процесом підбору в єдиному інтерфейсі, із вбудованою підтримкою великих мовних моделей (LLM) для автоматизованого аналізу резюме та рекомендацій.

Об’єктом дослідження є взаємодія користувачів-рекрутерів у контексті інформаційної системи працевлаштування з використанням LLM-моделей з метою підвищити ефективність відбору персоналу та пошуку вакансій.

Предметом дослідження виступають методи проєктування та реалізації ключових компонентів платформи на базі Java-фреймворку Spring Boot і шаблонізатора Thymeleaf, а також способи інтеграції зовнішніх LLM-сервісів для витягнення та аналізу семантичного змісту тексту.

У рамках роботи розроблено веб-додаток, що дозволяє рекрутерам керувати вакансіями та знаходити претендентів. Інструментарій фільтрації пошуку доповнено механізмами LLM-аналізу, що дозволяють не лише відбирати кандидатів за базовими критеріями, а й оцінювати глибинну відповідність їхніх навичок та досвіду потребам вакансії. Кандидати, своєю чергою, можуть застосовувати аналогічний підхід для пошуку найвідповідніших пропозицій, а також завантажувати резюме – система автоматично витягує з тексту ключові компетенції та формує профіль користувача. Весь функціонал реалізовано з використанням Spring Boot, Thymeleaf і реляційної бази даних PostgreSQL. Результати тестування підтвердили надійність роботи сервісів, а також зручність і зрозумілість інтерфейсу для всіх категорій користувачів.

ВЕБ-ПЛАТФОРМА, ПРАЦЕВЛАШТУВАННЯ, LLM, SPRING BOOT, THYMELEAF, POSTGRESQL, СЕМАНТИЧНИЙ ПОШУК, СКОРИНГ, SPRING AI

ABSTRACT

The qualification thesis comprises 77 pages, 19 figures, 2 tables, 6 appendices, and 25 references.

The purpose of the research is to develop a web platform for employee recruitment that integrates vacancy management, candidate tracking, and the recruitment process within a unified interface, with built-in support for large language models (LLMs) to automate resume analysis and recommendations.

The object of the research is the interaction of users and recruiters within an employment information system using LLM models to improve the efficiency of personnel selection and job search.

The subject of the research involves the methods of designing and implementing key components of the platform based on the Java Spring Boot framework and the Thymeleaf templating engine, as well as techniques for integrating external LLM services to extract and analyze the semantic content of text.

As part of the work, a web application was developed that enables recruiters to manage job openings and find candidates. The search filtering toolkit has been enhanced with LLM-based analysis mechanisms, allowing for not only basic criteria filtering but also deep evaluation of candidates' skills and experience in relation to job requirements.

Candidates, in turn, can use a similar approach to search for the most suitable job offers and upload resumes – the system automatically extracts key competencies from the text and generates a user profile.

All functionality is implemented using Spring Boot, Thymeleaf, and the PostgreSQL relational database. Testing results confirmed the reliability of the services, as well as the convenience and clarity of the interface for all categories of users.

WEB-PLATFORM, EMPLOYMENT, LLM, SPRING BOOT, THYMELEAF, POSTGRESQL, SEMANTIC SEARCH, SCORING, SPRING AI

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

LLM – Великі мовні моделі

AI – Artificial Intelligence (Штучний інтелект)

NLP – Natural Language Processing (Обробка природної мови)

NER – Named Entity Recognition (Розпізнавання іменованих сутностей)

POS – Part-Of-Speech (Тегінг частин мови)

GPT – Generative Pre-trained Transformer

BERT – Bidirectional Encoder Representations from Transformers

API – Application Programming Interface (Інтерфейс програмування застосунків)

JSON – JavaScript Object Notation

CV – Curriculum Vitae (Резюме)

PDF – Portable Document Format

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

SQL – Structured Query Language (Мова структурованих запитів)

MVC – Model-View-Controller (Патерн «Модель-Вид-Контролер»)

ORM – Object-Relational Mapping (Об’єктно-реляційне відображення)

TLS – Transport Layer Security (Протокол захищеного рівня передачі)

AES-256 – Advanced Encryption Standard 256-bit (Стандарт AES з ключем 256 біт)

DPIA – Data Protection Impact Assessment (Оцінка впливу захисту даних)

ВСТУП

У сьогоднішньому контексті динамічний ринок праці вимагає від компаній швидкого та ефективного вибору працівників, а від кандидатів - зручні способи самопрезентації та пошуку відповідних вакансій. Традиційні рекрутингові платформи зосереджуються насамперед на зберіганні та сортуванні даних кандидатів та вакансій з суворо визначеними ознаками, не беручи до уваги семантичну глибину професійного досвіду та особисті навички претендентів.

Наразі все більше дослідників і розробників звертають увагу на можливості штучного інтелекту, зокрема моделей LLM, для аналізу тексту та формування рекомендацій у різних галузях, однак застосування таких технологій у сфері HR переважно обмежується пілотними проєктами або використовує зовнішні сервіси без глибокої інтеграції в архітектуру інформаційної системи, що призводить до високих затримок та додаткових витрат.

У цій дипломній роботі обґрунтовано доцільність створення веб-платформи для працевлаштування співробітників, що поєднує управління вакансіями, кандидатами та процесом пошуку в єдиному інтерфейсі з підтримкою LLM-моделей. Метою дослідження є розробка прототипа платформи на базі Spring Boot і шаблонізатора Thymeleaf зі збереженням даних у базі даних PostgreSQL. Об’єктом дослідження виступають взаємодії користувачів із системою працевлаштування, зокрема рекрутерів і кандидатів, за допомогою інтелектуальних сервісів аналізу тексту. Предметом дослідження є методи проєктування та реалізації ключових компонентів платформи, а також способи інтеграції зовнішніх LLM-сервісів для автоматичного вилучення, аналізу семантичного змісту профілів кандидатів та вакансій для подальшого формування рекомендацій.

У ході роботи розроблено веб-додаток, що надає рекрутерам можливість створювати, редагувати та відслідковувати вакансії, користуючись

механізмами LLM-аналізу для фільтрації пошуку та оцінки глибинної відповідності навичок і досвіду кандидатів. Кандидати, у свою чергу, можуть завантажувати резюме, після чого система вилучає з тексту ключові компетенції й автоматично формує їхній профіль з можливістю редагування, а також використовувати аналогічний підхід для пошуку найбільш відповідних пропозицій – такий двосторонній інтелектуальний підхід дозволяє значно скоротити час на первинний огляд резюме та підвищити точність підбору.

Технічна реалізація проєкту базується на Spring Boot, Thymeleaf і PostgreSQL. Це забезпечує швидку й надійну роботу сервісів у різноманітних середовищах. Особлива увага приділялася оформленню зрозумілого користувацького інтерфейсу та оптимізації взаємодії з LLM через асинхронні запити й кешування результатів для мінімізації затримки в виконанні складних операцій. Результати функціонального та навантажувального тестування підтвердили стабільність системи, достатню швидкодію та високу зручність інтерфейсу для обох категорій користувачів. Отже, розроблена веб-платформа є перспективним інструментом інтелектуалізації процесу працевлаштування, що може бути адаптована до потреб різних організацій.

РОЗДІЛ 1. АНАЛІЗ СУЧАСНОГО СТАНУ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Сучасні тенденції в онлайн-рекрутингу

Онлайн-рекрутинг став невід'ємною частиною сучасного ринку праці, радикально змінивши підхід до пошуку роботи та підбору працівників. Його ефективність дедалі більше залежить від здатності компаній адаптуватися до нових інструментів та алгоритмів. Застосування сучасних платформ дозволяє не лише скоротити час пошуку кандидатів, а й значно покращити їх відповідність вимогам завдяки аналітиці, автоматизації та персоналізованим підходам.

Розуміння обмежень та викликів онлайн-рекрутингу є ключовим для створення ефективних систем. Для цього, необхідним стає дослідження та аналіз сучасних трендів та інструментів у цій сфері.

1.1.1 Еволюція онлайн-платформ для пошуку роботи

Протягом останніх років веб-платформи до публікації вакансій та пошуку роботи перейшли від простих дошок оголошень до великих та комплексних систем з безліччю інтеграцій сторонніх сервісів.

Порівняно з 2010-ми роками, коли платформи здебільшого пропонували лише звичайні способи публікацій вакансій та пошук за допомогою ключових слів без рекомендацій чи сортування, лідери ринку станом на 2025 рік, такі як LinkedIn, Glassdoor, Jooble та інші вже інтегрували механізми автоматичної індексації відгуків, впровадили мультипублікації вакансій та програматик-рекламу, що дозволяє швидко охопити аудиторію підходящих кандидатів та тим самим підвищити ефективність закриття позицій [1].

Поява мобільних додатків та зростання ролі соціальних мереж значно поширили можливості платформ для пошуку роботи, як наприклад LinkedIn буквально створив соціальну мережу, що базується на платформі з вакансіями,

впровадивши персоналізовану стрічку рекомендацій на основі профілю та активну комунікацію між користувачами, Indeed реалізував push-сповіщення про нові пропозиції, а Glassdoor – збір анонімних відгуків про компанії, що допомагає здобувачам формувати власні очікування.

Спеціалізовані сервіси як, наприклад, Jooble або Remotive, у 2022–2023 роках демонструють особливо високий темп приросту – це обумовлено тим, що дані сервіси орієнтуються на віддалену роботу та гібридні формати, які стали нормою для глобального ринку праці [2].

Саме ці зміни стали приводом до еволюції платформ та перетворили їх зі звичайних статичних дошок оголошень на динамічні сервіси з великою кількістю сторонніх інтеграцій, що забезпечує повний цикл рекрутингу, відповідно від початкової публікації вакансії, відбору за формальними критеріями до аналітики поведінки користувачів і підготовки прогнозів успішності найму. Це стало можливим, зокрема, завдяки інтеграції штучного інтелекту, масштабуванню інфраструктури платформ та розвитку партнерських програм із HR-технологіями.

1.1.2 Функції та інструменти сучасних рейтингових систем

Як було наведено у попередньому розділі, в сучасних реаліях рекрутингові платформи вже не обмежуються звичайним розміщенням вакансій та пошуком за ключовими словами, натомість, вони пропонують широкий набір функціональності для автоматизації кожного етапу добору персоналу. Так, наприклад, Applicant Tracking Systems (ATS), забезпечують централізоване зберігання та журналювання заявок та взаємодій із кандидатами, що дозволяє відстежувати статус кожного з етапів відгуку та дозволяє уникнути ситуацій, в яких деякі резюме залишаються непоміченими при великому потоці даних [3]. Також, використовуються інструменти для побудови Talent Pool, а CRM-рішення заохочують рекрутерів до підтримування контакту з перспективними кандидатами.

Крім того, платформи активно використовують чат-ботів для первинного скринінгу: вони задають кандидатам уточнювальні питання, автоматично фіксують відповіді та можуть попередньо відсіяти некваліфіковані заявки, звільняючи HR-фахівців від рутини. Інструменти відео-інтерв'ю, онлайн-оцінювання навичок і гейміфіковані тести дають змогу не лише оцінити технічні вміння, а й певні поведінкові характеристики потенціальних робітників ще перед особистою співбесідою з рекрутером. Аналітичні дошки з метриками (time-to-hire, cost-per-hire, source effectiveness), дозволяють в режимі реального часу коригувати стратегії залучення кандидатів та оптимізувати бюджет на рекрутацію [4].

1.1.3 Проблеми онлайн-рекрутації

Незважаючи на різноманітні переваги та плюси інтеграцій, HR-спеціалісти стикаються з певними проблемами, які значно знижують ефективність процесу рекрутації. З таких можна виділити, наприклад, те, що багато систем досі спираються на простий пошук тексту за ключовими словами, який досить часто повертає неякісні результати: наприклад, резюме з некоректними формулюваннями можуть легко „загубитися" і навпаки, резюме, що містять синонімічні вирази – будуть пропущені через недостатню гнучкіть систем аналізу [1].

Іншим прикладом є те, що за високої конкуренції на ринку праці та обмежені можливості індексації контексту, близько 40% отриманих відгуків виявились малорелевантими, що змушує рекрутерів повторно переглядати великі кількості резюме та витрачати час на мануальний первинний скринінг [2].

У висновок можна сказати, що, незважаючи на автоматизацію, велика кількість рекрутингових платформ все ще вимагають ручного відбору, адже алгоритми теж можуть допускати певні помилки, через які час пошуку кандидатів може значно збільшитися. Додатково, такі алгоритми не мають можливості надійно визначити певні нюанси профілю чи галузеві особливості.

1.2 Використання штучного інтелекту в рекрутингу

На сучасному ринку праці швидкість прийняття рішень і точність добору кандидатів мають критичне значення. Зважаючи на це, інтеграція штучного інтелекту в онлайн-рекрутингу стає не просто перевагою, а необхідністю. В умовах величезного обсягу кандидатів та інформації, ШІ дозволяє автоматизувати складні аналітичні процеси: збільшує ефективність рекрутингу, зменшує витрати часу на пошук і мінімізує людський фактор.

1.2.1 Застосування штучного інтелекту у фільтрації кандидатів, генерації рекомендацій

Сучасні AI-системи для пошуку роботи мають змогу здійснювати такі комплексні операції як фільтрація кандидатів, аналіз CV та генерація рекомендацій практично в один клік. Це відбувається так, що спочатку алгоритми машинного навчання сканують велику кількість резюме за ключовими характеристиками (навичками та вимогами), автоматично видляють найбільш релевантні позиції та ранжують їх за степенем відповідності вакансії. Прикладом таких технологій є платформа impress.ai, що здатна за секунди відсортувати пул претендентів, водночас відкидаючи некваліфікованих кандидатів та формуючи список найбільш доцільних заявок для подальшого перегляду рекрутерами [5].

Разом із цим глибинні модулі, що побудовано на основі генеративних моделей типу відомого ChatGPT, здатні аналізувати не тільки сухий текст профілю, а і розуміти супровідні листи та враховувати більш широкий контекст, та навіть оцінювати ймовірну інтеграцію кандидата в команду – на основі цих даних система формуватиме структуру персоналізованих рекомендацій, в цьому автоматичні запрошення на співбесіду, поради щодо поліпшення резюме і т. д., що різко скорочує час первинного скринінгу і дає

змогу сфокусуватися на кандидатах, що мають більші шанси на працевлаштування за компетенціями [6].

1.2.2 Роль NLP у обробці резюме та описів вакансій.

Такі системи як NLP відіграють ключову роль у структуризації та розумінні текстових документів, що відповідно використовується в процесі рекрутингу. Процес відбувається таким чином, що резюме та описи вакансій поступово проходять етап токенцізації, після чого відбуваєтсья виділення сутностей (NER) – імена, посади, дати, навички автоматично формуються у вигляді структурованих полів [7]. Після цього, на основі лематизації та POS-тегінгу відбувається класифікація тексту, наприклад, розподіл набутого досвіду за категоріями.

За допомогою формування векторного уявлення тексту, з`являється можливість семантичного пошуку, який дозволяє враховувати контекст та близькість значень, де, умовно, запит „проєктний менеджмент” знайде не лише точні збіги, а і синоніми та описові формулювання, наприклад: „керування командами”, „управління проектами” [8]. Завдяки цій технології рекрутингові системи мають змогу забезпечувати більш точний пошук, а також знизити кількість false-positive і false-negative результатів пошуку, тим самим підвищуючи релевантність віддачі.

1.2.3 Використання LLM (наприклад, GPT, BERT) у задачах семантичного пошуку, класифікації, скорингу.

Моделі LLM демонструють виняткові можливості для семантичного пошуку, класифікації та скорингу вакансій і резюме зокрема через здатність будувати глибокі контекстні векторні представлення тексту. Такі моделі застосовують підхід перетворення опису вакансії чи резюме в embeddings – багатовимірні вектори, що відображають значення слів та фраз у контексті всього тексту. Такий підхід дозволяє виконувати пошук шляхом порівняння косинусної подібності між векторами, що, в свою чергу, дає можливість

знаходити релевантні результати навіть в випадках, коли використовуються різні формулювання. Для прикладу, модель VacancySBERT показала покращення recall у топ-10 семантично схожих заголовків вакансій на 21,5% порівняно з класичними підходами завдяки використанню навчених embeddings для рекрутингової галузі [9].

Задля класифікації LLM моделей, вони можуть бути налагоджені на різні категорії вакансій або типи навичок, наприклад, GPT-4 здатен без додаткового донавчання розпізнавати, чи резюме відповідає посаді виходячи лише з наданих прикладів у prompt, що сильно скорочує потребу у витратах на вдосконалення існуючих моделей під конкретні кейси. Додатковою важливою рисою є те, що LLM моделі здатні створювати пояснення до своїх рішень, що не лише підвищує довіру користувачів, а і дає можливість генерувати рекомендації до покращення резюме.

Скоринг кандидатів в рекрутингу за допомогою таких моделей передбачає оцінку ступеня відповідності між вакансією та резюме основуючись на комплексному аналізі всіх доступних даних, зокрема: досвіду, навичок, освіти та інших аспектів. Використання великих моделей для embeddings (таких, як SGPT) показує, що при відповідному налаштуванні моделі, при семантичному пошуку, вони можуть значно випереджати інші підходи за точністю навіть використовуючи менші обчислювальні ресурси, що робить можливим гнучке створення та налаштування алгоритмів для скорингу під конкретні бізнес-кейси [10].

1.2.4 Приклади комерційних рішень із вбудованим AI

В сучасних комерційних рішеннях охоплюється весь процес рекрутингу починаючи від скринінгу до глибинної оцінки потенціалу кандидата. Одним з таких прикладів є HireVue, що поєднує відеоінтерв`ю з аналізом невербальних сигналів, тобто платформа здатна автоматично розпізнавати міміку та інтонацію, що дозволяє визначити „employability score” на підставі оцінки патернів мовлення, темпу та вирази обличчя під час бесіди. За даними HireVue,

використання їх інструментів значно скорочує час співбесіди та підвищує якість прогнозування успішності нових співробітників за допомогою ідентифікації ключових навичок комунікації [11].

Pymetrics (Harver) використовує підхід „гейміфікації”, тобто використовує серію онлайн-ігор з ціллю оцінки когнітивних та емоційних рис кандидата, після чого зібрані дані порівнюються з профілями успішних співробітників організації. Це дозволяє мінімізувати упередженість та підвищити різноманітність підбору кандидатів [12].

Ще одним прикладом є Rezi, що здебільшого фокусується конкретно на резюме, даючи інструменти для аналізу сумісності з ATS-системою та оптимізації даних в реальному часі, таким чином, система автоматично генерує балансовані за ключовими словами формулювання добираючи синоніми та структуру, максимізуючи видимість у пошукових алгоритмах.

Rezi фокусується на самому резюме, надаючи інструменти для аналізу ATS-сумісності та оптимізації контенту в реальному часі. Система автоматично генерує балансовані за ключовими словами формулювання, добираючи синоніми й структуру, щоб максимізувати видимість у пошукових алгоритмах роботодавців [13].

Разом ці рішення ілюструють, як вбудований AI допомагає не тільки прискорити процес рекрутингу, так і зробити його більш точним і справедливим, водночас ставлячи нові завдання щодо прозорості алгоритмів і етичного застосування даних.

1.3 Огляд існуючих технологічних рішень та платформ

Для розробки ефективної веб-платформи для працевлаштування співробітників, в першу чергу необхідно було дослідти сучасні технологічні рішення, що застосовуються в системах онлайн рекрутингу для обробки, інтелектуального пошуку та аналізу кандидатських даних на основі традиційних технологій і великих мовних моделей. Метою було проаналізувати сильні та слабкі сторони кожного з рішень, а також оцінити

виклики, що виникають при їх впровадженні. Особлива увагу було приділено технічним та організаційним аспектам інтеграції, таким як складність обслуговування, витрати на інфраструктуру та відповідність нормативним вимогам.

1.3.1 Порівняння архітектур

Рішення, що побудовані на поєднанні реляційних або документних баз даних із пошуковими движками по типу Elasticsearch є добре відпрацьованими та зрілими технологіями. В таких системах дані зберігаються в структурованому вигляді, що забезпечує високий рівень продуктивності та цілісності. Elasticsearch індексує текстові поля за допомогою аналізаторів, дозволяючи дуже швидко виконувати запити та агрегації за різними метаданими, однак для досягнення саме семантичного розуміння тексту, необхідно вручну налаштовувати словники, правила обробки слів та періодично оновлювати оновлювати аналізатори під нові терміни та доменні вимоги, що безпосередньо створює додаткові витрати та ускладнює процес підтримки.

В свою чергу підходи орієнтовані на LLM моделі кардинально змінюють цю парадигму – замість попереднього індексування та жорстких правил використовуються нейронні мережі, що здатні „розуміти” великий контекст та значення слів у ньому. При запиті система передає текст безпосередньо до моделі, отримуючи векторні представлення, після чого виконуючи семантичний пошук за допомогою порівняння косинусних відстаней. Такий підхід значно знижує потребу мануального тюнінгу аналізаторів та словників, але в свою чергу теж вимагає значно більших обчислювальних ресурсів, а також масштабованої інфраструктури, щоб забезпечити обробку великої кількості запитів у реальному часі.

Крім цього, класичні архітектури відрізняються тим, що обслуговування має досить низьку вартість та легко інтегруються в існуючі стеки, порівняно з LLM-системами, що потребують додаткових і складних інструментів

оркестрації (Kubernetes), моніторингу витрат на генерацію відповідей та опрацювання черг запитів (RabbitMQ, Kafka). В деяких сценаріях застосовують гібридний підхід з первинним фільтром за допомогою Elasticsearch та подальшим аналізом LLM, що дозволяє зменшити навантаження на систему.

1.3.2 Використання API-сервісів

Щоб уникнути проблем з розгортанням LLM-системи, розробники все частіше звертаються до існуючих сторонніх API-сервісів, що дозволяють зосередитися на бізнес-логіці замість витрат на власну інфраструктуру та донавчання моделей. Прикладами таких сервісів є, зокрема, OpenAI API [15] або GroqCloud [16]. Дані сервіси надають доступ до різноманітних open-source та комерційних LLM-моделей через REST-інтерфейс. Більшість мов програмування дозволяють налаштувати інтеграцію з такими сервісами за допомогою бібліотек, виступаючих в ролі HTTP-клієнтів (наприклад, Spring AI [16]), що дає розробникам ще більшу абстракцію над комплексними системами LLM. Попри велику кількість переваг такого підходу, зі збільшенням потреб програмного забезпечення зростатимуть кошти використання, можливі певні обмеження на кількість токенів контексту, а також rate limiting [17].

Таким чином, інтеграція з подібними API-сервісами стає доволі простою задачею, що дозволяє зосередитися на використанні моделей з ціллю покращення користувацького досвіду без необхідності організації власної інфраструктури, хоча водночас вимагає ретельного управління лімітами запитів, контролю витрат та забезпечення відповідності політикам безпеки даних.

1.3.3 Проблеми інтеграції

Передача чутливих персональних даних таких як резюме кандидату до сторонніх хмарних LLM сервісів підпадає під суворі вимоги GDPR – зокрема, будь-яке надсилання даних має відбуватися з використанням шифрування при

відправці (TLS) та на диску (AES-256). Також має бути передбачено можливість видаляти персональні дані за запитом [18]. Крім цього, необхідно проводити операцію DPIA щоб оцінити ризики обробки та документувати всі етапи передачі та зберігання інформації.

Великі мовні моделі за своєю природою є «чорними скриньками»: навіть досвідчені інженери не завжди можуть чітко пояснити, чому система відхилила чи підвищила рейтинг певного резюме. Це призводить до невдоволення HR-команд і загрожує юридичними ризиками, якщо випадково алгоритм виявиться упередженим. Для подолання цих проблем застосовують XAI-інструменти, що генерують локальні пояснення рішення моделі, але недоліком такого підходу є те, що вони збільшують складність архітектури, та не завжди коректно відображають внутрішні процеси [19].

Навіть за поточними трендами зниження цін, обробка великих обсягів заявок через LLM моделі залишається досить затратною. Згідно з оцінками, щомісячна вартість inference для великої кількості заявок на базі GPT-4 може досягати кількох тисяч доларів [20]. Власна інфраструктура (GPU-кластери) вимагає капіталовкладень у TensorRT-оптимізації, охолодження та енергоспоживання. Для покращення ситуації компанії вимушені впроваджувати кешування, батчинг запитів та комбінувати дешевші локальні моделі для попереднього фільтрування задля зниження загальних витрат.

1.4 Мотивація до створення нового рішення

Незважаючи на стрімкий розвиток штучного інтелекту, багато існуючих систем залишаються неточними, недостатньо гнучкими або непрозорими для користувача, що знижує їх ефективність й релевантність добору кандидатів та рівень довіри. Усвідомлення цих обмежень стало першим кроком до пошуку рішень, які могли б подолати наявні бар’єри. Для цього було необхідно дослідити сучасні підходи, визначити їхні недоліки та сформувати концепцію платформи, здатної поєднати у собі найкращі рішення.

1.4.1 Виявлені обмеження існуючих підходів

Велика кількість AI-рішень можуть демонструвати значні помилки при скринінгу резюме та оцінці кандидатів особливо представників маргіналізованих груп, як, наприклад, дискримінації можуть піддаватися кандидати з великими перервами у кар'єрі, або особи, що не є носіями англійської мови, що призводить до непропорційного відхилення кваліфікованих претендентів. Це здебільшого відбувається через те, що більшість моделей навчені на англомовних або глобальних даних і слабо справляються зі специфікою локальних ринків праці.

Таким чином, у галузі локалізації AI часто помиляється через мовні нюанси та галузеві терміни, які можуть вимагати ручної корекції після машинного перекладу або семантичного аналізу [21]. Це ускладнює застосування тих самих інструментів для, скажімо, польського, українського чи індійського ринків без суттєвої налаштування або донавчання моделі через відсутність контекстних деталей.

AI-системи зазвичай не надають претендентам достатньої інформації про причини відмови чи поради для покращення. Відсутність зворотного зв'язку та людського елементу робить процес безособовим і демотивує кандидатів. Відсутність порад щодо форматування резюме, рекомендацій щодо підвищення релевантності чи навіть простого пояснення, чому заявку відхилено, створює негативний досвід взаємодії з платформою [22].

1.4.2 Переваги запропонованого підходу

Замість того, щоб покладатися лише на «чорний ящик» мовної моделі або виключно на класичні пошукові індекси, гібридний підхід спочатку відбирає вузький набір кандидатів за допомогою структурованих фільтрів (наприклад, по роках досвіду, локації, галузі), а потім проводить глибокий семантичний аналіз їхніх резюме та мотиваційних листів через LLM. Це дозволяє поєднати швидкість і передбачуваність звичних запитів зі здатністю

великих моделей „розуміти” контекст і нюанси. Таким чином, значно покращується точність підбору, скорочується кількість false-positive і false-negative результатів, водночас знижується ймовірність пропустити релевантних кандидатів.

Завдяки можливості формувати embedding-представлення вакансій і резюме, платформа може обчислювати міри семантичної близькості між описом позиції та профілем кандидата. Це дозволяє генерувати не просто список «найбільш підходящих» за жорсткими правилами, а персоналізовані рекомендації, які враховують як формальні вимоги (строкові дані, сертифікати), так і неформальні сигнали (відтінки досвіду, схожість проєктів).

1.4.3 Актуальність і доцільність розробки нової платформи з підтримкою LLM

У сучасних умовах розвитку ринку праці автоматизація процесу добору персоналу перестала бути додатковою опцією – вона стала критичною вимогою для успішних платформ.

Традиційні рішення, що базуються виключно на ключових словах і жорстко заданих правилах, не здатні вчасно реагувати на появу нових професій чи незвичайні поєднання компетенцій у кандидатів. Використання моделей LLM дозволяє майже миттєво адаптуватися до змін у термінології й галузевих стандартах без необхідності ручного оновлення словників чи перенавчання класичних моделей.

Крім цього, глибинний LLM-аналіз дає змогу забезпечити більш глибоке розуміння контексту резюме та описів вакансій, що відкриває можливості для персоналізованого зворотного зв’язку – кандидати можуть отримувати не лише рішення, а і конкретні рекомендації щодо поліпшення структури чи формулювань у своїх документах. Для роботодавців така платформа стає ефективним інструментом оптимізації часу закриття вакансії та підвищення якості найму за рахунок гнучкого налаштування критеріїв відбору і більш точного підбору відповідних фахівців.

1.4.4 Постановка задачі

Відповідно до перерахованих уваг, завдання дипломної роботи формулюються таким чином:

1. З’ясувати та систематизувати вимоги ринку праці й проаналізувати існуючі підходи до рекрутингу на базі LLM, щоб виявити їхні сильні й слабкі сторони.
2. Розробити архітектуру гібридної платформи, де класичні системи зберігання та пошуку даних інтегруються з модулем семантичного аналізу на основі LLM, причому особливу увагу приділити механізмам передачі даних і масштабованості.
3. Реалізувати прототип із використанням зовнішнього API для LLM та традиційних баз даних, забезпечивши можливість завантаження резюме, фільтрації за базовими характеристиками і глибокої оцінки релевантності.
4. Провести експериментальні випробування з вимірюванням точності відбору та продуктивності системи, а також узагальнення отриманих результатів у вигляді рекомендацій для подальшого промислового впровадження.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПЛАТФОРМИ ДЛЯ ПОШУКУ РОБОТИ З ВИКОРИСТАННЯМ LLM

2.1 Вибір архітектурного підходу та технологій

Сучасна розробка веб-платформ вимагає інтеграції різноманітних технологічних рішень, які підтримують гнучкість і надійність системи. Використання перевірених фреймворків і засобів спрощує процес створення та подальшої супровідної роботи над проектом. При правильному поєднанні інструментів можна скоротити часові витрати на налаштування середовища і зосередитися на реалізації основних функцій. Крім того, продумана архітектура дозволяє легко масштабувати систему з мінімальними змінами в кодовій базі. Також варто заздалегідь визначити, в який спосіб відбуватиметься інтеграція з LLM та які методи моделі будуть використані.

2.1.1 Обґрунтування вибору стеку технологій

Для проектування та імлпементації веб-платформи для пошуку роботи було прийнято рішення використати перевірені, сучасні технології, що забезпечують високу масштабованість, зручність використання, активну підтримку зі сторони розробників та безпеку.

В якості основи було обрано Java-фреймворк Spring Framework, а детальніше – Spring Boot. Ця технологія має широкі функціональні можливості та автоматичну конфігурацію додаткових модулів (Spring Security, Spring Data JPA, Spring AI). Набір інструментів дозволив сфокусуватися на розробці бізнес-логіки, оскільки більшість нефункціональних аспектів майже повністю покриваються фреймворком.

Для реалізації масштабованої архітектури на базі шаблону Model-View-Controller застосовується Spring MVC – модуль фреймворку, що забезпечує чітке розподілення між бізнес-логікою і представленнями. Даний підхід

полегшує масштабування та підтримку коду, а також, оскільки цей патерн побудований на абстракціях, спрощує тестування окремих елементів системи.

Окремо слід зазначити модуль Spring AI, що використовується для легкої інтеграції з зовнішніми LLM API сервісами, впроваджуючи абстракцію в якості http-клієнту з допоміжними методами, що спрощують використання. Детальніше про інтеграцію з LLM у підрозділі 2.5.

У якості системи управління базами даних використовується PostgreSQL — надійна об'єктно-реляційна DBMS з відкритим вихідним кодом. Оскільки в проекті використовується Spring Data JPA що впроваджує абстрактну генерацію запитів (ORM) – обрана модель бази даних може бути легко замінена без великих наслідків.

Задля реалізації механізмів аутентифікації, авторизації та базової безпеки додатку, використано модуль Spring Security. Даний модуль забезпечує захист веб-застосунку від типових загроз, дозволяє гнучко налаштовувати доступ та підтримує інтеграцію з різними схемами аутентифікації. В моєму випадку було використано механізм сесій, що автоматично забезпечено у цьому модулі.

Thymeleaf обрано як шаблонізатор для динамічної генерації HTML-сторінок на стороні сервера – нативна інтеграція з Spring дозволяє ефективно відображати динамічні дані у вигляді HTML-шаблонів без додаткових залежностей, що відповідає концепції простого серверного рендерингу, характерного для монолітних веб-застосунків. Додатково, щоб покращити зовнішній вигляд представлень, було використано фреймворк Bootstrap, що дозволяє швидко створювати адаптивні та візуально привабливі інтерфейси.

2.1.2 Принципи модульності, розширюваності та масштабованості

Оскільки серверна частина застосунку реалізована на основі шаблону MVC, що дозволяє розділити бізнес-логіку, презентаційний шар та обробку запитів, це сприяє структурованості коду, зменшенню зв'язності між компонентами та забезпечує повну можливість незалежного тестування та

розвитку окремих модулів. Таким чином, для інкапсуляції виконання логіки, використовуються інтерфейси сервісів замість конкретних імплементацій, що дає змогу легко замінювати або розширювати бізнес-логіку.

Гнучкість архітектури також забезпечується за рахунок використання принципу Dependency Inversion – такий підхід дозволяє незалежно змінювати окремі частини проекту, наприклад, впроваджувати альтернативні реалізації сервісів.

Доступ до даних здійснюється через Spring Data JPA – це дає незалежність від конкретної DBMS завдяки використанню абстракцій над базою даних. Крім того, Spring Data JPA автоматизує багато рутинних CRUD-операцій, знижуючи ризик помилок і підвищуючи продуктивність розробки.

2.1.3 Підхід до реалізації LLM-інтеграцій

У рамках цього проєкту інтеграція великих мовних моделей здійснюється виключно через зовнішній API-сервіс, що узгоджується з вимогою швидкого прототипування та мінімізації операційних витрат. Такий підхід в рамках цієї роботи був більш доцільним, оскільки він дозволяє сильно пришвидшити початковий етап розробки та зосередитися на безпосередньо на її меті. Окрім цього, використання зовнішнього API-сервісу дозволяє повністю делегувати відповідальність за підтримку та відмовостійкість безпосередньо до провайдеру сервісу.

Однак такий варіант інтеграції у більших маштабах має й певні недоліки. Такий спосіб інтеграції сприяє тому, що система повністю залежатиме від доступності та якості обслуговування зовнішнього провайдера: будь-який простій чи регламентні роботи на його боці миттєво позначаються на працездатності нашої платформи та переривають робочі потоки користувачів. Дуже вірогідною проблемою теж є те, що тарифи на API-запити формуються за кількістю оброблених токенів або запитів, тому масштаби використання моделі безпосередньо корелюють із зростанням операційних витрат, що здебільшого складно передбачити.

Найбільшим недоліком є те, що передача даних сторонньому сервісу завжди містить ризики конфіденційності: навіть за шифрованого каналу та дотримання GDPR-вимог ми не вправі контролювати, як провайдер зберігає, аналізує та потенційно використовує отриману інформацію, тому такий підхід вимагатиме більш ретельного огляду при комерційному використанні.

2.1.4 Використання inference в LLM

У рамках реалізації інтеграції великих мовних моделей ключовим етапом було використання саме процесу inference, тобто безпосереднього генерування відповідей зі збереженою попередньо натренованою моделлю. Математично inference можна розглядати як функцію

$$f(x;\theta) \qquad (2.1.4.1)$$

де x – це вхідний текст (промпт);

θ – набір фіксованих ваг моделі, отриманих під час тренування.

Під час inference модель здійснює послідовність перетворень векторів вхідних токенів: спочатку кожне слово чи підслово переводиться у вектор простору ознак, після чого ці вектори проходять через низку шарів трансформера, де на кожному кроці обчислюються контекстуальні представлення через механізм самоуваги.

З математичної точки зору у шарі самоуваги кожна позиція вхідного вектора h_i породжує запит Q_i, ключі K_j та значення V_j для всіх позицій j, і потім обчислюється зважена сума всіх V_j. Ваги визначаються через softmax від скалярних добутків $\langle Q_i, K_j \rangle$. У результаті кожен токен набуває оновленого векторного представлення, яке враховує контекст усього речення. Після проходження крізь декілька таких шарів і шарів позиційного кодування модель отримує остаточне розподілення ймовірностей для наступного токена:

$$P(y_t \mid y_{<t}, x;\ \theta) = softmax(Wh_t + b) \qquad (2.1.4.2)$$

де y_t – це токен, який генерується на кроці t;

$y_{<t}$ – усі токени до цього моменту;

h_t – його контекстуальне представлення;

W – матриця ваг проєкції у словниковий простір;

b – вектор зсуву (bias);

x – це вхідний текст (промпт);

θ – набір фіксованих ваг моделі, отриманих під час тренування.

Таким чином, з математичної точки зору inference є застосуванням фіксованої функції f до конкретного вхідного тексту, де вектори h ймовірнісно перетворюються за допомогою шарів самоуваги та афінних перетворень, а потім через функцію softmax отримується готовий текстовий відгук. Цей етап не пов'язаний із модифікацією ваг θ всі налаштування моделі вже завершені під час тренування, що дозволяє зосередитися виключно на оперативному виклику API і швидкій генерації результатів. Тому, з огляду на пріоритети проєкту щодо швидкого аналізу та мінімізації витрат, саме inference через зовнішній API-сервіс виявився найдоцільнішим рішенням.

2.2. Архітектура системи та модель проєкту

Архітектура проєкту була розроблена з метою забезпечення чіткої організації компонентів системи та ефективної взаємодії між ними. Вона дозволяє досягти високої підтримуваності, гнучкості та масштабованості, що є критично важливими для подальшого розвитку платформи. Такий підхід дає змогу не лише розмежувати відповідальність між різними рівнями, а й забезпечити надійну інтеграцію зовнішніх сервісів та безпечну роботу з даними.

2.2.1 Загальна схема компонентів

У межах цього підрозділу представлено загальну структурну схему компонентів програмної системи, яка ілюструє основні функціональні блоки,

їхні ролі та взаємодію між собою. Архітектура розробленої платформи побудована відповідно до принципів багаторівневого розділення відповідальностей, що сприяє її підтримуваності, масштабованості та можливості подальшого розширення.

На рисунку 2.2.1.1 зображено загальну блок-схему компонентів, що включає презентаційний рівень (інтерфейс користувача), сервісний рівень (бізнес-логіка), шар доступу до даних, модуль інтеграції з LLM через API, а також взаємодію з базою даних.

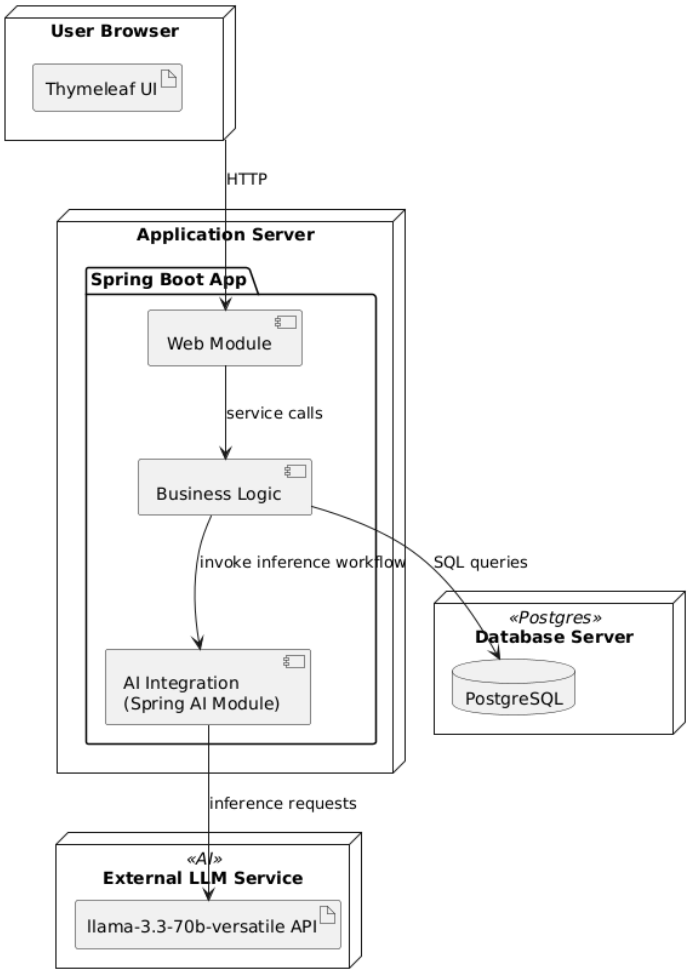

Рисунок 2.2.1.1 – Загальна блок-схема компонентів системи

2.2.2 Детальний опис рівнів архітектури

Архітектура створеної системи реалізована з урахуванням принципів багаторівневого розподілення відповідальностей, тим самим даючи основу до масштабованості та зручності у підтримці.

1. Рівень презентації. На даному рівні відбувається взаємодія з кінцевим користувачем за допомогою HTML-шаблонів інтегрованих з Thymeleaf, що дозволяє легко створювати сторінки на стороні серверу. За обробку запитів відповідальні Spring контролери, що окрім самої обробки теж відповідають за авторизацію користувачів базуючись на поточній сесії. З точки зору проектування, даний рівень не виконує жодної бізнес-логіки: лише приймає запити та повертає структуровані відповіді, в той час як основна функціональність енкапсулюється на сервісному рівні. Додатково, контролери комунікують з сервісами лише через інтерфейси, тобто для рівню контролерів конкретна імплементація сервісів невідома, що теж покращує масштабованість та понижує пов’язання між рівнями.

2. Сервісний рівень. Цей рівень реалізує бізнес-логіку системи – тобто тут зосереджено основні операції пов’язані з застосуванням правил предметної області, координацією викликів між компонентами та в цілому обробкою даних. Як і було зазначено раніше, сервіси мають певні інтерфейси, що дозволяє легко заміняти імплементації без змін у коді. На цьому рівні також відбувається інтеграція з зовнішніми сервісами, зокрема LLM API. Ця інтеграція відбувається за рахунок модулю Spring AI, що надає абстрактний інтерфейс для комунікації з LLM сервісами за допомогою HTTP запитів.

3. Рівень доступу до даних.Цей рівень відповідає за збереження та отримання даних з реляційної бази даних (в даному випадку – PostgreSQL). Доступ до даних реалізується за допомогою Spring Data JPA, що забезпечує абстракцію над SQL-запитами та дозволяє уникнути прямого використання SQL у коді, хоча за потреби дозволяє використати зручний діалект SQL оснований на Java сутностях – JPQL. Такий підхід також сприяє незалежності

бізнес-логіки від конкретної реалізації бази даних, а також підвищує зручність підтримки проєкту.

2.2.3 Організація авторизації та аутентифікації користувачів

Для забезпечення контролю доступу до системи використано модуль **S**pring Security, що надає потужні механізми аутентифікації та авторизації користувачів.

Аутентифікація реалізується на основі стандартного механізму сесій, що впроваджує Spring Security. Після успішного входу користувача система створює сесію, в межах якої зберігається інформація про аутентифікованого користувача.

Авторизація у системі реалізована на основі ролей користувачів, які також зберігаються у базі даних та асоціюються з кожним обліковим записом, а також та перевірки доступів до даних. На рівні контролерів використовуються анотації, такі як @PreAuthorize, що дозволяє обмежувати доступ до окремих методів залежно від ролі користувача або перевіряючи власність над певними даними (наприклад, чи може рекрутер переглядати заявки на вакансії). Таким чином, система гарантує, що лише уповноважені користувачі можуть виконувати певні дії або отримувати доступ до чутливої інформації.

2.2.4. Функціональні вимоги та сценарії використання

Даний підрозділ присвячений деталізованому огляду конкретних функціональних можливостей розроблюваної платформи, що реалізуються для основних акторів (Candidate та Recruiter).

Для обох акторів система реалізує можливість реєстрації та аутентифікації.

Кандидат, в свою чергу, має можливість редагувати профіль, що включає оновлення інформації про навички, попередній досвід і освіту, а також завантажувати резюме. Однією з важливих деталей є автоматичне заповнення

полів профілю на основі LLM-аналізу завантаженого CV: система надсилає текст резюме на обробку, отримує структурований результат у форматі JSON і підставляє його у відповідні поля.

Далі кандидат може шукати вакансії стандартним фільтром за параметрами (локація, категорія, рівень досвіду), або на основі заповненого профілю за допомогою пошуку через LLM, який повертає ті оголошення, що найбільше відповідають його профільному опису. Для обраних вакансій кандидат має можливість подати заявку, а також зберігати цікаві позиції в «обране» для подальшого перегляду.

Для рекрутера перелік задач включає управління вакансіями, перегляд надісланих кандидатами заявок з можливістю залишати текстовий фідбек і призначати статуси. Окремою функцією є пошук кандидатів за ключовими навичками та скоринг на основі LLM-модуля: рекрутер вказує потрібну вакансію, система генерує список найвідповідніших профілів із додатковим обґрунтуванням від моделі.

Важливо виділити що LLM-скорінг відбувається за допомогою одного алгоритму для обох сторін – тобто кандидат, шукаючи вакансію, отримуватиме ту ж інформацію про свій рейтинг зі своєї перспективи, так і рекрутер, передивляючись кандидатів під певні вакансії, бачитиме той самий звіт, що було згенеровано LLM. Це дозволяє уникнути розбіжностей аналізу та оптимізувати систему, оскільки ці операції є досить важкими. На рисунку 2.2.4.1 наведено use-case діаграму функціональності системи.

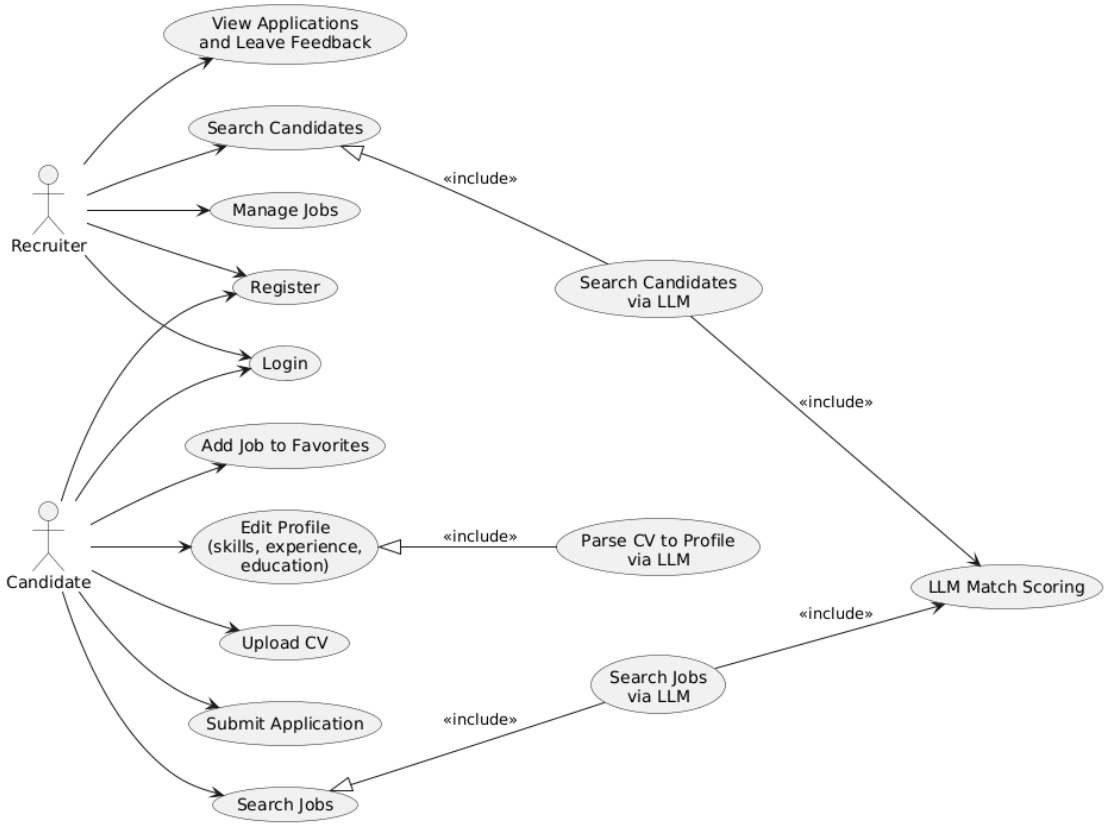

Рисунок 2.2.4.1 – Use-case діаграма системи

2.3 Сутності проєкту

В рамках створеної платформи основними доменними сутностями є User, Candidate, Recruiter, Company, Job, Skill, JobApplication, JobCandidateMatch, CandidateProfile, Education та JobExperience.

Усі вище зазначені моделі успадковують загальний суперклас BaseEntity, що містить поля id, createdAt й updatedAt, забезпечуючи єдину схему ідентифікації та відстеження часу створення й останнього оновлення записів.

Сутність User є базовою для всіх зареєстрованих у системі облікових записів: вона містить електронну пошту, ім’я, прізвище та хешований пароль. Кожен Candidate успадковує ідентификатор користувача (у полі id класу BaseEntity) та додає посилання на файл CV. Аналогічно, Recruiter успадковує від User та містить посилання на компанію, у якій він працює.

Company зберігає основні відомості про роботодавця — назву, опис та часові позначки створення й оновлення запису. Схема зв’язків передбачає, що

кожен Job належить саме до однієї Company та одного Recruiter, а отже, вакансії коректно асоціюються зі своїм джерелом.

Для опису професійних навичок використовується сутність Skill, а зв’язок «багато-до-багатьох» між Job і Skill реалізовано через допоміжну таблицю job_skills. Такий же механізм зв’язку застосовано для відображення навичок у профілі кандидата через таблицю candidate_profile_skills.

Процес подання заявки моделюється сутністю JobApplication, яка містить статус та зворотний зв’язок. Для кожної заявки зберігаються посилання на відповідного Candidate та Job. Результати матчингу кандидатів і вакансій фіксуються в JobCandidateMatch: окрім числового match_score, цей об’єкт може містити текстове обґрунтування, сформоване LLM.

Коли кандидат створює обліковий запис, йому автоматично асоціюється CandidateProfile, котрий зв’язує основні відомості про кандидата з додатковими блоками: Education (освітні записи) та JobExperience (попередні місця роботи). Обидві сутності містять часові рамки, опис, назву установи чи компанії, поле спеціальності або посаду, а також посилання на Candidate та відповідний CandidateProfile, що дозволяє формувати повноцінний кар’єрний портрет користувача.

Загалом, така модель доменних сутностей забезпечує чітку типізацію ролей (User – Candidate/Recruiter), гнучке описування як вакансій, так і профілів кандидатів, а також зручний механізм оцінки їх відповідності один одному.

Для структурованого представлення відношень між сутностями було створено ER-діаграму, що наведено на рисунку 2.3.1:

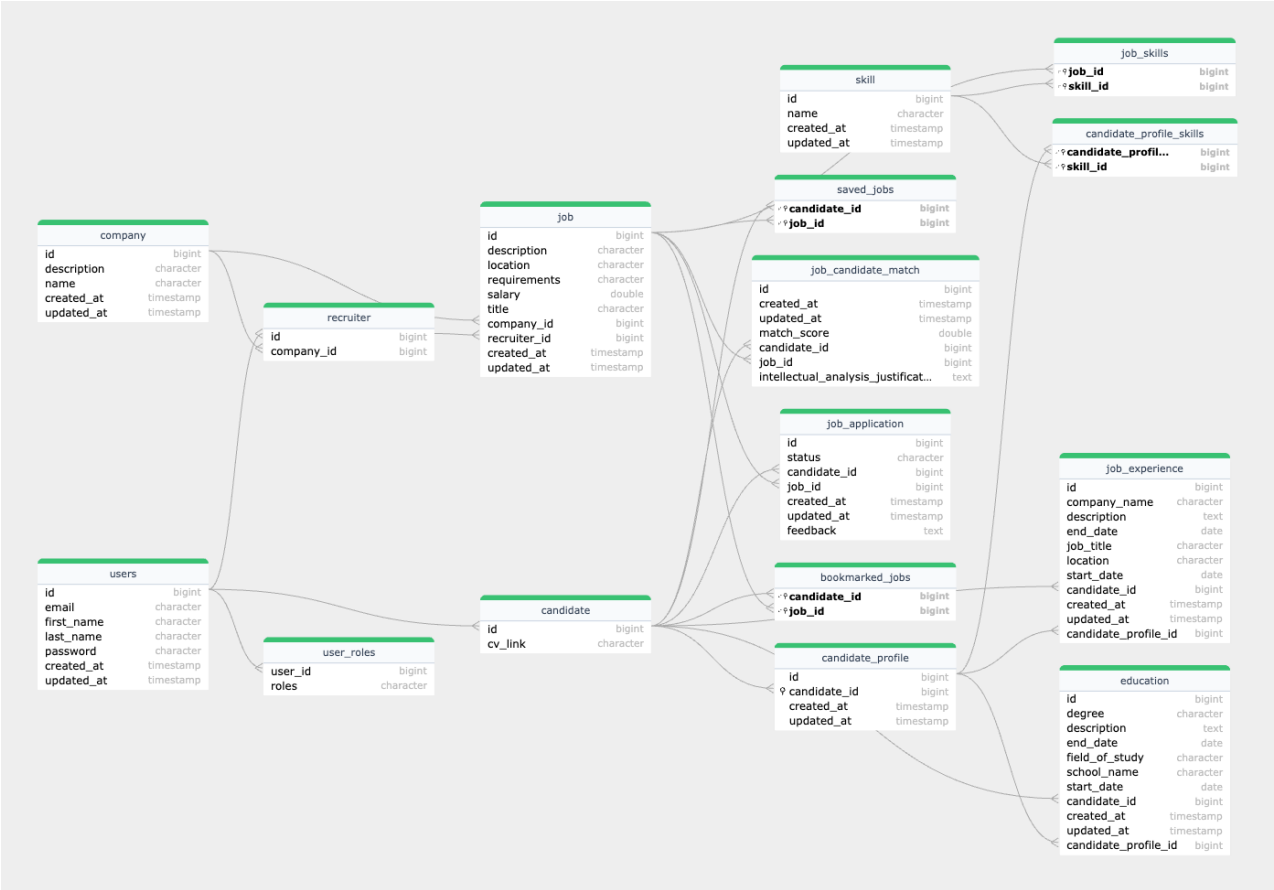

Рисунок 2.3.1 – ER-діаграма сутностей проекту

2.4 Інтеграція з LLM

В даному підрозділі розглянуто вибір провайдеру, конкретної LLM, застосування модулю Spring AI, створення промптів та оптимізацію процесів.

2.4.1 Вибір провайдера LLM API та моделі

Головними критеріями при виборі провайдеру були: легка інтеграція з фреймворком Spring, відсутність початкових витрат при невеликих обсягах обробки та доступність потрібних моделей.

Найбільш популярним провайдером LLM API є OpenAI, що пропонує найбільш просунуті на ринку моделі (GPT-4, GPT-4 Turbo тощо), які демонструють вищу якість генерації тексту та глибший контекстний аналіз, а також цей сервіс має дуже розвинену екосистему та SLA гарантії прямо від

розробника моделей, однак цей сервіс не надає можливості безкоштовного використання API, що сприяло пошуку інших рішень.

Рішенням, що задовільнило вимоги цієї роботи, було GroqCloud – сервіс надає безкоштовний тарифний план, що цілком покриває обсяг запитів на етапі прототипування та тестування. Крім того, GroqCloud пропонує доступ до кількох архітектур LLM – від легковагових для швидких попередніх оцінок до більш потужних для глибинних генерацій тексту й аналізу. Також варто зазначити, що GroqCloud реалізує REST API в стилі OpenAI, що дозволяє без змін інтегрувати цей сервіс за допомогою Spring AI, тому за усіма критеріями було прийнято рішення на користь цього провайдеру.

2.4.2 Вибір моделі LLM

При виборі LLM, ключовими критеріями стали архітектурні та функціональні властивості, що безпосередньо відповідають вимогам платформи, тобто глибина та точність розуміння тексту, здатність працювати з довгими контекстами, а також з нефункціональної точки зору швидкодія та ціна використання. Для прийняття рішення я звернувся до досліджень, що порівнюють сучасні LLM між собою за певними критеріями, зокрема дослідження від Vellum [25]. Дане дослідження показує, що llama-3.3-70b-verstaile хоч і поступається в точності таким важким та дорогим моделям, як GPT-4o або Llama 405b, але при цьому порівняно до інших рішень показує кращі результати, тому я вирішив зупинитися якраз на цій моделі.

2.4.3 Інтеграція за допомогою Spring AI

В свою чергу інтеграція за допомогою Spring AI вимагала мінімальних налаштувань – достатньо було вказати базовий URL та ім’я моделі, а механізми авторизації, кешування та десеріалізації даних були реалізовані ООТВ, що суттєво пришвидшило вихід MVP і дозволило зосередитися на бізнес-логіці без витрат часу на оптимізацію комунікації з API.

Spring автоматично створює bean класу OpenAiChatModel, що представляє собою HTTP-клієнт, надаючи зручний інтерфейс для взаємодії з моделлю у вигляді Java-методів. Цей клас дає можливість як висилати звичайний prompt у вигляді тексту, так і об’єкт, що містить певні налаштування до запиту, такі як temperature або maxTokens.

2.4.4 Створення промптів

При створенні промптів я керувався трьома головними засадами: чітке визначення ролі системи, деталізація завдання та строгий опис очікуваного формату відповіді.

Наприклад, для аналізу CV та щоб витягнути з тексту резюме структуровані дані про освіту, досвід і навички, я формував запит так, щоб модель одразу розуміла, що вона повинна повернути вичерпний JSON конкретного формату, що спрощувало подальшу обробку. Вигляд промпту наведено нижче на рисунку 2.4.4.1:

```
Please extract the following information from the CV text. If any of the information is not available, please leave the field empty.
If there are no job experiences or education, leave the corresponding arrays empty.
No matter what is in the CV text, please strictly respond with the following JSON format:
{
  "jobExperiences": [
    {
      "jobTitle": "string",
      "companyName": "string",
      "location": "string",
      "startDate": "YYYY-MM-DD",
      "endDate": "YYYY-MM-DD",
      "description": "string"
    }
  ],
  "educations": [
    {
      "degree": "string",
      "schoolName": "string",
      "fieldOfStudy": "string",
      "startDate": "YYYY-MM-DD",
      "endDate": "YYYY-MM-DD",
      "description": "string"
    }
  ],
  "skills": [
    "skill1",
    "skill2",
    "skill3"
  ]
}
CV text: {{cvText}}
```

Рисунок 2.4.4.1 – Промпт до моделі для аналізу резюме

Дана структура промпту запевнює консистентність відповіді від моделі, а також саме використання LLM для аналізу CV сприяє більш точному аналізу та пришвидшенню процесу заповнення профілю користувача.

Наступним прикладом промпту був запит для скорингу пар кандидат-вакансія, що використовувався для семантичного пошуку кандидатів або вакансій. Тут я застосував аналогічний підхід, однак змінив вхідні дані та очікувану структуру відповіді. Промпт наведено нижче на рисунку 2.4.4.2:

```
You are an expert in recruitment and job matching.
Analyze the provided candidate profile and compare it to the job description.
Evaluate the relevance of the candidate's work experience, education, and skills to the job requirements.
Assign an experience match score on a scale from 0 to 1, where 0 represents minimal relevance and 1 represents a highly suitable match.
The score should vary based on the degree of alignment, rather than being strictly 0 or 1.
Justify your score with a concise, structured explanation.
Avoid including the score itself in the justification.
Format the output as JSON with the following fields:
{
  "experience_match": float,
  "justification": string
}
Candidate JSON: {{candidate}}
Job JSON: {{job}}
```

Рисунок 2.4.4.2 – Промпт до моделі для скорингу пар кандидат-вакансія

Важливою деталлю цього промпту є більш детальний опис ролі моделі – LLM має розуміння своєї ролі (рекрутер), розуміє в який спосіб має проводити аналіз та в цілому розуміє повний контекст проблеми.

2.5 Реалізація алгоритму скорингу та семантичний пошук

За скоринг пар кандидат-вакансія в системі відповідає пакет „matching”. Цей пакет містить певні класи, що забезпечують відповідну логіку для оцінки відповідності кандидатів до вакансій. Класи з цього пакету використовуються у пакетах „candidates” та „jobs” при виконанні семантичного пошуку. В цьому підрозділі буде розглянуто аспекти цих пакетів, що відповідають за пошук, більш детально.

2.5.1 Алгоритмічне попереднє фільтрування за навичками

Сутності Candidate та Job містять посилання на об’єкти типу Skill, що визначає навички, якими володіє кандидат, та які потрібні на вакансію. За допомогою цих даних при семантичному пошуку ми маємо змогу спочатку відсортувати кандидатів або вакансії за відповідністю навичок, що в подальшому дає змогу оптимізувати процес LLM-аналізу, оскільки це дозволяє робити аналіз лише для обмеженої кількості найбільш релевантних пар.

Дане фільтрування відбувається на рівні бази даних, тобто за допомогою SQL-запиту. Рішення робити це через SQL було прийнято через те, що в такий спосіб, за великою кількістю записів на базі даних, можна досягнути найшвидшого часу виконання, порівняно до потенціального сортування на рівні Java-коду, де спочатку усі об’єкти з бази даних мали б пройти десеріалізацію, що значно б сповільнило виконання цього процесу. На рисунку 2.5.1.1 наведено метод репозиторію з пакету „candidates” відповідальний за фільтрацію:

```
@Query(value = """
        SELECT u.*, c.*
        FROM users u
        JOIN candidate c ON c.id = u.id
        LEFT JOIN candidate_profile cp ON cp.candidate_id = c.id
        LEFT JOIN candidate_profile_skills cps ON cps.candidate_profile_id = cp.id
            AND cps.skill_id IN (:skillIds)
        GROUP BY u.id, c.id
        ORDER BY COUNT(cps.skill_id) DESC
        """,
        countQuery = """
                SELECT COUNT(DISTINCT u.id)
                FROM users u
                JOIN candidate c ON c.id = u.id
                LEFT JOIN candidate_profile cp ON cp.candidate_id = c.id
                LEFT JOIN candidate_profile_skills cps ON cps.candidate_profile_id = cp.id
                    AND cps.skill_id IN (:skillIds)
                """,
        nativeQuery = true
)
Page<Candidate> findByMatchingSkills(
        @Param("skillIds") Set<Long> skillIds,
        Pageable pageable
);
```

Рисунок 2.5.1.1 – Метод репозиторію CandidateRepository для фільтрації кандидатів

Аналогічний спосіб фільтрації застосовано для пошуку вакансій відносно кандидатів.

2.5.2 Семантичний скоринг за допомогою LLM

Як було зазначено у попередньому пункті, до знайдених кандидатів за допомогою алгоритмічної фільтрації застосовується семантичний аналіз за допомогою LLM.

Цей процес відбувається так, що профіль кандидату та вакансія серіалізуються в JSON формат, відправляються до сервісу відповідального за аналіз (LlmIntellectualJobCandidateMatchService), де, в свою чергу, відбувається комунікація з LLM API за допомогою промпту, наведеного на рисунку 2.4.4.2 у розділі 2.4.4 – в результаті операції, повертається DTO JobCandidateMatchDto об’єкт з полями expierenceMatch (float) та justification (String).

Пізніше experienceMatch використовується для обчислення зваженої суми LLM-результату та алгоритмічної оцінки на кількість співпадінь за навичками. Цей алгоритм може розширюватися та змінюватися за потреби:

$$match\_score = skill\_match \times 0.4 + experience\_match \times 0.6 \qquad (2.5.2.1)$$

де skill_match – відповідність навичок (алгоритмічне порівняння);
experience_match – результат семантичного LLM аналізу.

2.5.3 Кешування результатів та механізм інвалідації

Після обчислення кожного JobCandidateMatch запис з фінальним matchScore та justification зберігається в базі. Це пришвидшує наступні запити як від кандидатів (пошук вакансій), так і від рекрутерів (пошук кандидатів).

У разі зміни ключових характеристик (оновлення профілю кандидата або редагування вакансії) всі відповідні кешовані записи видаляються. Валідація відбувається в сервісному шарі одразу при змінах відповідно в

профілі кандидату або описі вакансій, що забезпечує завчасне оновлення даних та очистку неактуальної інформації.

2.6 Парсинг резюме за допомогою LLM

Для автоматизованого вилучення структурованих даних з резюме в форматі PDF використовується сервіс DefaultLlmCvProcessingService, який реалізує інтерфейс LlmCvProcessingService. Цей компонент керується трьома ключовими етапами: підготовка промпту, видобуток тексту з документа та виклик LLM-моделі для отримання JSON-відповіді.

На початку методу завантажується шаблон промпту з ресурсів проєкту (llm-prompts/cv-parse-prompt.txt). Цей шаблон містить інструкції для моделі щодо того, які блоки необхідно витягти з резюме (досвід, освіту, навички) та у якому форматі повернути результат. Після завантаження шаблону сервіс викликає метод, що за допомогою Apache PDFBox перетворює вміст файлу за заданим шляхом на одиничний рядок тексту.

Після відправлення запиту, отримана відповідь перетворюється в об’єкт LlmCvProcessingDto за допомогою Jackson ObjectMapper.

Завдяки такому підходу, сервіс гарантує, що будь-яке резюме, яке зберігається у файловій системі чи хмарному сховищі й доступне за посиланням, може бути автоматично проаналізоване та перетворене в чітку структуру полів (досвіду, освіти та навичок) без необхідності ручного введення. Варто зазначити, що знайдені навички перевіряються на присутність у базі даних і лише потім зберігаються – у поточній імплементації це дозволяє запевнити, що користувачі не можуть додавати нових навичок до системи.

2.7 Виклики, проблеми та шляхи подолання

У процесі розробки та впровадження платформи з використанням LLM виникає низка технічних та етичних викликів, які вимагають уважного аналізу. Визначення таких проблем дозволяє чітко зрозуміти обмеження технології,

передбачити критичні сценарії та знайти оптимальні способи забезпечення стабільної, безпечної й ефективної роботи системи.

2.7.1 Продуктивність та масштабованість LLM-запитів

Однією з найбільших проблем при інтеграції великих мовних моделей є забезпечення прийнятного часу відповіді та здатності системи обробляти значні обсяги паралельних запитів без деградації сервісу.

У нашій архітектурі кожен семантичний запит до LLM (чи то заповнення профілю із CV, чи скоринг кандидат–вакансія) здійснюється через зовнішній GroqCloud API, що з одного боку знімає з нас обов’язок керувати інфраструктурою GPU, але з іншого – накладає обмеження на число одночасних викликів та час очікування відповіді.

Особливо ці обмеження відчувалися помітними при завантаженні сторінок, що використовують семантичний скоринг – умовно 10 операцій підрахування відповідності відбувалися по черзі, що могло займати близько 10 секунд, що не є припустимим у сучасній веб-розробці.

Для того, щоб уникнути цієї проблеми або пом’якшити час виконання, мною було розроблено підхід, при якому запити на розрахунок відповідності одразу для великої кількості пар відбуваються у паралельному режимі, тобто замість того, щоб виконувати всі 10 операції аналізу послідовно, одночасно відправляється 10 запитів.

Паралелізація запитів реалізована через поєднання CompletableFuture, пулу потоків Spring-Annotation @Async("llmExecutor") і структури inFlight, яка гарантує, що на одну й ту ж пару «вакансія–кандидат» не буде запущено кілька одночасних обчислень.

Коли надходить запит на скоринг певної пари, метод getJobMatchScoreAsync спочатку перевіряє, чи вже існує активний або завершений CompletableFuture для цього ключа (jobId–candidateId) у мапі inFlight. Якщо такий є, він повертається клієнту, і відбувається «підсадка» другого запиту до того самого обчислення. Якщо ж пара ще не обробляється,

на пулі llmExecutor запускається асинхронний виклик calculateAndStoreMatchScore, який відразу повертає CompletableFuture. У момент завершення обчислення відповідний запис у inFlight атомарно видаляється (за допомогою whenComplete), роблячи простір для нових запитів у майбутньому – такий підхід дозволяє значно понизити час відкриття сторінки до 2-3 секунд.

Даний підхід теж не є ідеальним рішенням, оскільки відправлення великої кількості запитів одночасно може призвести до проблем з rate limiting. Для ще більшої оптимізації цього процесу, можна було б застосувати батчинг запитів – цей вид оптимізації полягає у об’єднанні декілька скорингових запитів в один, передаючи до LLM не тільки одну пару, а список з кілька. Хоча в рамках нашої роботи перший підхід теж показує досить гарні результати, підхід з батчингом був би більш доцільним за умови, що система розширюватиметься та використовуватиметься в більших масштабах.

2.7.2 Забезпечення конфіденційності персональних даних

При розробці платформи було використано два різних підходи до передачі даних до LLM. Для семантичного скорингу пар вакансія-кандидат, використовуються лише поля, що мають значення в бізнес-контексті, тобто набір навичок, опис досвіду та освіти. В свою чергу аналіз CV з ціллю заповнення профілю вимагає відправлення всього тексту резюме, тому навіть беручи під увагу безпечну передачу даних, поточна реалізація допускає тимчасове тримання на сервері незашифрованого CV до завершення обробки.

Щоб покращити рівень захисту, можна використати підхід анонімізації, де резюме б попередньо аналізувалось системою та вразливі дані були б приховані. В цілому, користувач має бути чітко поінформований про те, як його дані використовуватимуться та підтвердити згоду на обробку своїх даних моделями LLM.

2.7.3 Проблеми при роботі з неточними даними користувачів

В певних випадках, коли опис вакансії або профіль кандидата містять недостатньо детальну інформацію, система не буде в стані коректно аналізувати відповідність та повертатиме нерелевантні результати.

Наприклад, в ситуації коли кандидат зазначить недостатню кількість навичок або зазначить їх некоректно, алгоритм попереднього фільтрування може повертати вакансії, що не відповідатимуть очікуванням кандидата, та операції додаткового LLM-аналізу будуть неефективні. Аналогічно, при незмістовному описі вакансій, LLM модель буде змушена працювати з недостатнім контекстом, в результаті чого користувач отримає список позицій, що не відповідатимуть його очікуванням.

Отже, щоб досягти максимальної точності пошуку й рекомендацій, важливо, щоб інформація заповнювалася в повному обсязі – лише в такому випадку алгоритм семантичного пошуку діятиме так, як цього очікують користувачі системи. Щоб цьому сприяти, до системи можна додати додаткову валідацію, оцінюючу повноту інформації та змушуючу користувачів до розширення опису.

2.8 Висновки до розділу

У підсумку виконання другого розділу показало, як поєднання вибору технологій, архітектурних рішень і алгоритмів підбору створює єдину цілісну систему з підтримкою LLM. Усі ключові компоненти, починаючи від серверної частини до механізмів семантичного аналізу, працюють у взаємодії та забезпечують необхідну продуктивність і точність. У наступних підпунктах буде більш детально представлено огляд досягнутих результатів та перспективних напрямків з урахуванням практичних кейсів і реальних потреб користувачів.

2.8.1 Підсумки реалізованого рішення

Першим кроком розробки було обрано та обґрунтовано стек технологій на основі Spring Boot, PostgreSQL і Thymeleaf. Використання цих технологій забезпечило швидкий старт розробки та автоматичну конфігурацію функціональності потрібної для системи. Наступним етапом було розроблено архітектуру системи, де було чітко розділено систему на рівні: презентаційний, сервісний, доступу до даних, а також описано механізми, що забезпечили авторизацію та аутентифікацію в системі. Додатково, було описано зв’язки між сутностями системи, наведено use-case діаграму для розуміння очікувань від кінцевого результату.

Ключовою частиною реалізації було створення алгоритму для підбору кандидатів та вакансій, що складається з SQL-фільтрування за співпадінням кількості навичок із подальшим виділенням 10 найбільш релевантних пар, що дозволило уникнути навантаження на Java рівні. Після цього, до знайдених пар застосовувався семантичний скоринг за допомогою LLM API, де кожну пару було проаналізовано з використанням заздалегідь створеного промпту та оцінено на співпадіння від 0 до 1. Далі, це значення використовувалось у формулі для розрахунку зваженої оцінки разом із простим співпадінням навичок. Модель також повертала обґрунтування своєї оцінки, після чого все разом зберігалось до бази даних у вигляді моделі JobCandidateMatch для кешування. Завдяки паралелізації вдалося скоротити середній час обробки десятка запитів із понад 10 секунд до 2–3 секунд.

Усі ці складові створили стійку, масштабовану платформу, яка дозволяє швидко адаптуватися до зростання навантаження та вимог бізнесу, одночасно зберігаючи високу точність підбору та захист персональних даних.

В цілому, можна сказати, що програму було успішно реалізовано у відповідності до початкових вимог.

2.8.2 Перспективи розвитку та можливості розширення системи

Одним з ключових напрямків подальшого розвитку платформи може бути відповідно розширення моделі профілю користувача за рахунок додаткових атрибутів та переваг. Наприклад, до існуючих полів можна додати розділи з уподобаннями щодо корпоративної культури, бажаний формат зайнятості (віддалений, гібридний), бажаний рівень заробітної плати та географічні побажання. Такі зміни дозволили б ще точніше формулювати запити та отримувати ще точніші результати як для кандидатів, так і рекрутерів. Додатково, можна було б покращити формулу match_score враховуючи ці деталі.

Ще одним напрямком є вдосконалення механізмів анонімізації та псевдонімізації персональних даних. Доцільно автоматично розпізнавати й приховувати, наприклад, дати народження, імена навчальних закладів або організацій, контактну інформацію, тобто все, що може однозначно ідентифікувати особу.

Для масштабних навантажень слід реалізувати батчинг LLM-запитів, коли декілька пар вакансія-кандидат упаковані в один виклик API. Це може значно знизити кількість HTTP-з’єднань і ризик спрацювання rate limiting. Варто дослідити оптимальні розміри батчів і стратегії черговості запитів, щоб забезпечити баланс між швидкістю та точністю розрахунків.

Крім того, можна інтегрувати систему з внутрішніми та зовнішніми джерелами даних (наприклад, профілі LinkedIn, GitHub), щоб поліпшити якість матчингу, автоматично підтягувати нові навички та досвід кандидата.

РОЗДІЛ 3. РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ ТА РЕАЛІЗАЦІЇ СИСТЕМИ

3.1 Досягнення поставленої мети та вирішення завдань

В результаті проведеної роботи, було розроблено прототип веб-платформи для працевлаштування за допомогою Java-фреймворку Spring Boot з використанням Thymeleaf для реалізації інтерфейсу користувача та базою даних PostgreSQL. Цей результат відповідає меті дипломної роботи, що полягала в створенні єдиного інтерфейсу для управління вакансіями та кандидатами з вбудованою підтримкою LLM сервісів.

Під час реалізації були послідовно вирішені наступні завдання:

1. Аналіз предметної області – проведено аналіз підходів до автоматизації рекрутингу, оцінено потреби користувачів і сформульовано вимоги до платформи.
2. Проєктування архітектури – розроблено багаторівневу архітектуру з виділенням презентаційного, сервісного, рівня доступу до даних і зовнішніх інтеграцій, що забезпечило модульність, розширюваність та простоту підтримки.
3. Реалізація базового функціоналу – створено механізми реєстрації й аутентифікації через Spring Security, CRUD-операції для сутностей, також інтерфейс для редагування профілів і публікації вакансій, пошук вакансій, взаємодію між рекрутером та кандидатом, тощо.
4. Інтеграція з LLM – використано модуль Spring AI для взаємодії із зовнішнім провайдером GroqCloud. Також розроблено спеціальні промпти для автоматичного парсингу CV і скорингу пар «кандидат–вакансія», налаштовано кешування і механізми інвалідації результатів.
5. Оптимізація продуктивності – реалізовано алгоритмічне попереднє фільтрування за навичками на рівні БД, паралелізацію LLM-запитів

із використанням Async механізмів Java та Spring, що скоротило час обробки десятка запитів із понад 10 до 2-3 секунд.

Отже, поставлена мета досягнута у повному обсязі, а сформульовані предметом і об’єктом дослідження завдання було успішно реалізовано. Прототип забезпечує необхідний набір функціональних можливостей, відповідає вимогам продуктивності та безпеки, а отримані результати закладають основу для подальшого вдосконалення та масштабування платформи.

3.2 Оцінка результатів розробки

Реалізоване рішення базується на унікальній стратегії скорингу, що поєднує традиційне порівняння співпадіння навичок із глибинним семантичним аналізом LLM. Цей підхід дозволяє навіть при відсутності повного збігу ключовиї слів дозволяє системі коректно виявляти найбільш релевантні вакансії або кандидатів, враховуючи близькі за значенням формулювання та контекст. Це дає змогу подолати обмеження традиційних рішень, які або опираються лише на текстові збіги, або використовують векторні представлення без урахування формальних вимог.

Універсальність інтеграції забезпечується використанням Spring AI та REST-сумісного API GroqCloud. Завдяки такій абстракції бізнес-логіка не прив’язана до жодного конкретного постачальника LLM та до жодної конкретної моделі. В перспективі це дозволить без змін переключитися на інший сервіс або модель лише змінивши конфігурацію додатку.

Водночас ця архітектура має певні обмеження, як наприклад те, що повна передача даних до зовнішнього API створює залежність від його доступності й тарифів, а також певні труднощі з конфіденційними даними користувачів.

Ще однією проблемою є нечіткість або неповнота вхідних даних. За умови, що профіль кандидата або опис вакансії заповнені поверхово, семантичний модуль оперує недостатнім контекстом, що призводить до

неточних рекомендацій. Як і всі рішення на основі GPT-моделей, наш LLM-шар може успадкувати відмови й упередження, закладені в тренувальних даних, тому для промислового застосування варто використати модель треновану спеціально під рекрутингові цілі.

Попри ці обмеження, комбінація алгоритмічного та семантичного скорингу виявилася практичною та конкурентоспроможною. Вона дозволяє поєднати швидкість і зрозумілість классичних методів із глибиною і гнучкістю LLM-аналізу, забезпечуючи значно вищий рівень релевантності, ніж при використанні кожного підходу окремо, однак варто розуміти, що поточна реалізація потребує більш ретельної оптимізації.

3.3 Демонстрація функціональності системи

В даному підрозділі буде наведено скріншоти та опис результатів виконання системи при найбільш важливих сценаріях використання.

На рисунках 3.3.1-3.3.3 наведено скріншоти сторінок аутентифікації та реєстрації:

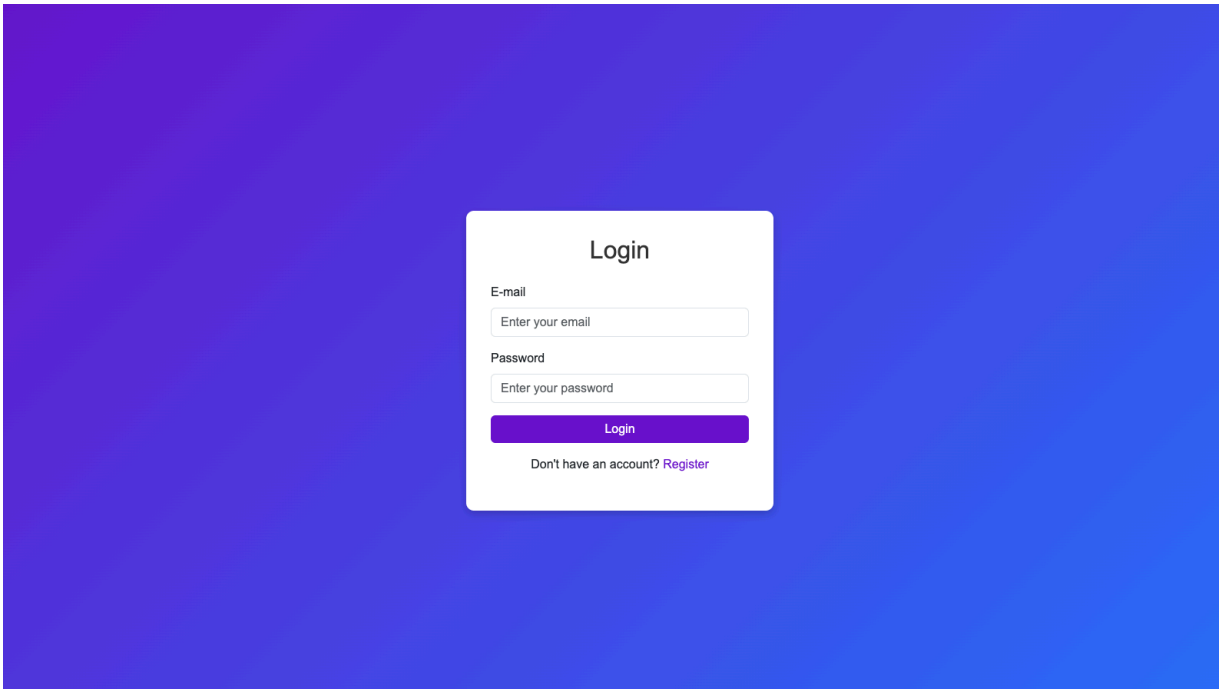

Рисунок 3.3.1 – Сторінка для аутентифікації користувача

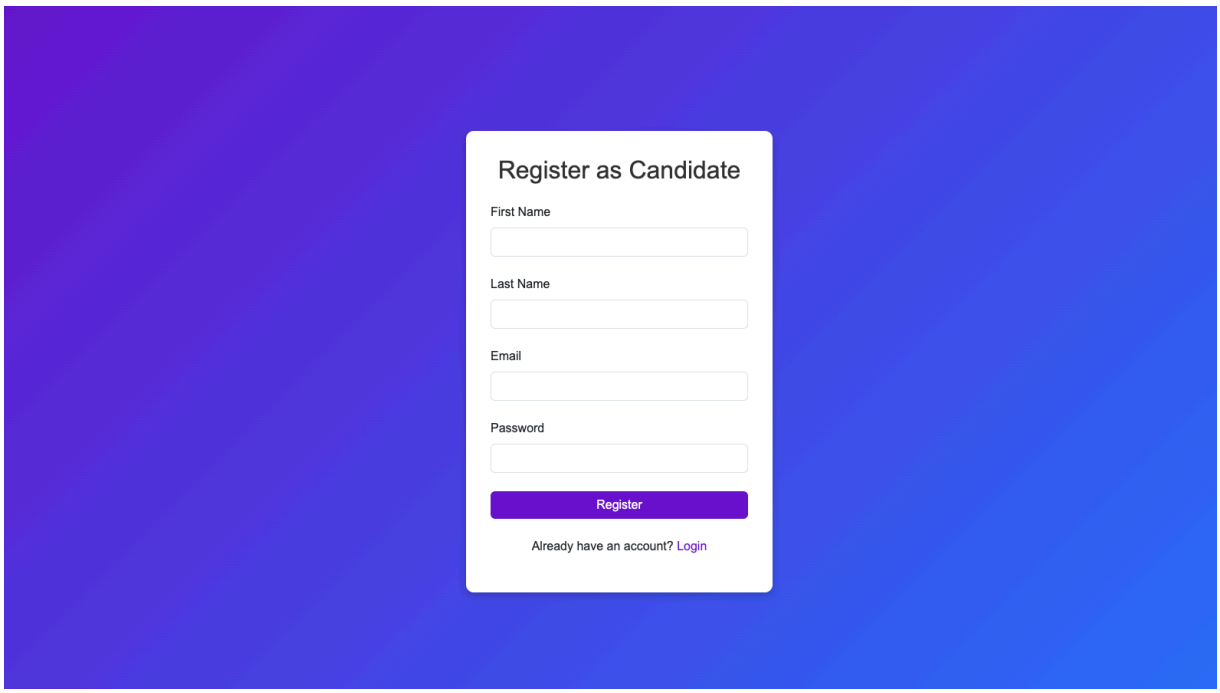

Рисунок 3.3.2 – Сторінка для реєстрації кандидата

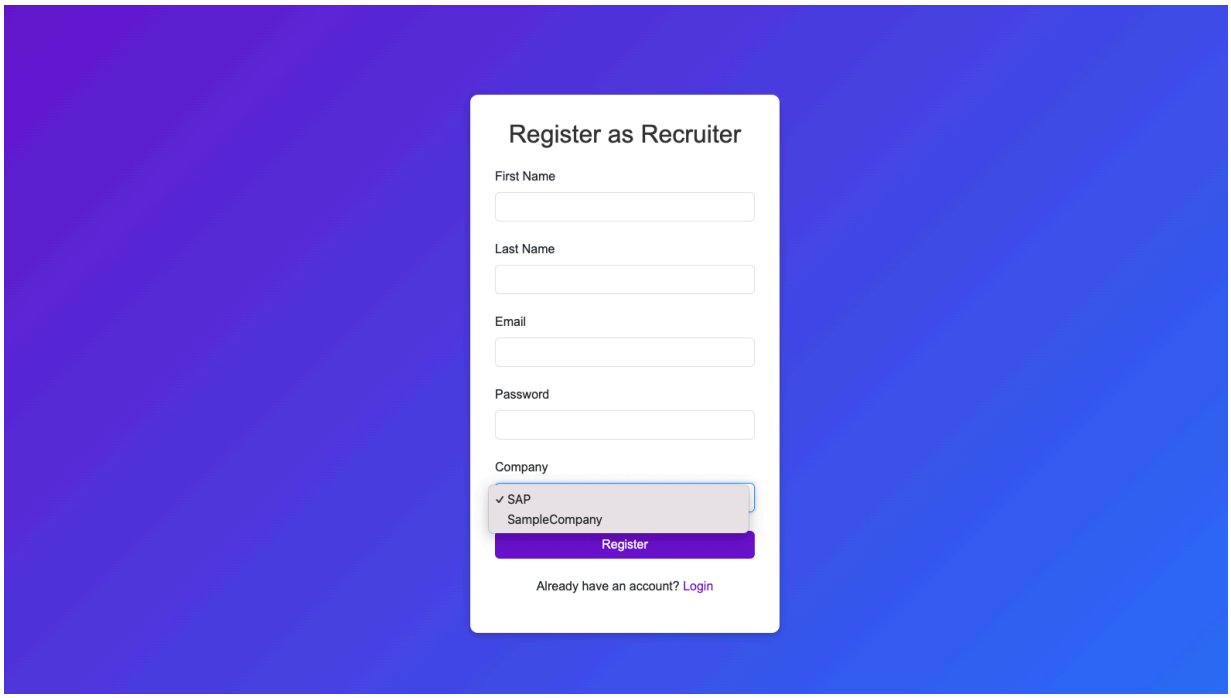

Рисунок 3.3.3 – Сторінка для реєстрації кандидата

На наступному рисунку 3.3.4 наведено dashboard кандидата. На даній сторінці знаходиться вся найбільш корисна інформація для користувача – огляд його профілю, уподобані вакансії, останні подані заявки та деякі рекомендовані вакансії.

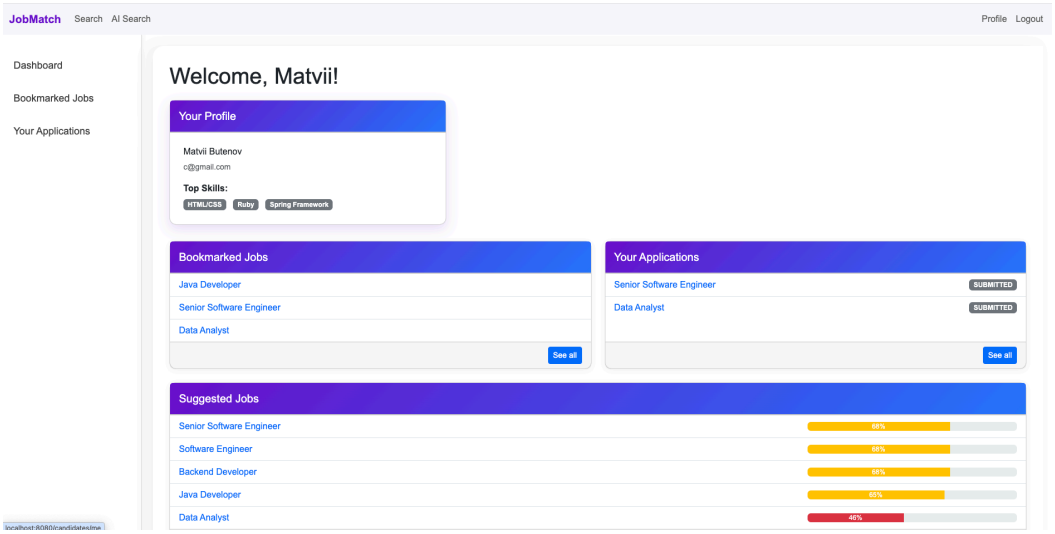

Рисунок 3.3.4 – Сторінка-dashboard кандидата

На наступному рисунку 3.3.5 наведено сторінку для семантичного пошуку вакансій. Окрім семантичного пошуку, на даній сторінці також можна застосувати різні фільтри (в тому числі локація, навички, компанії, бажана компенсація). Вакансії сортуються за релевантністю та біля кожної вакансії позначається відсоток наскільки поточний кандидат відповідає певній вакансії.

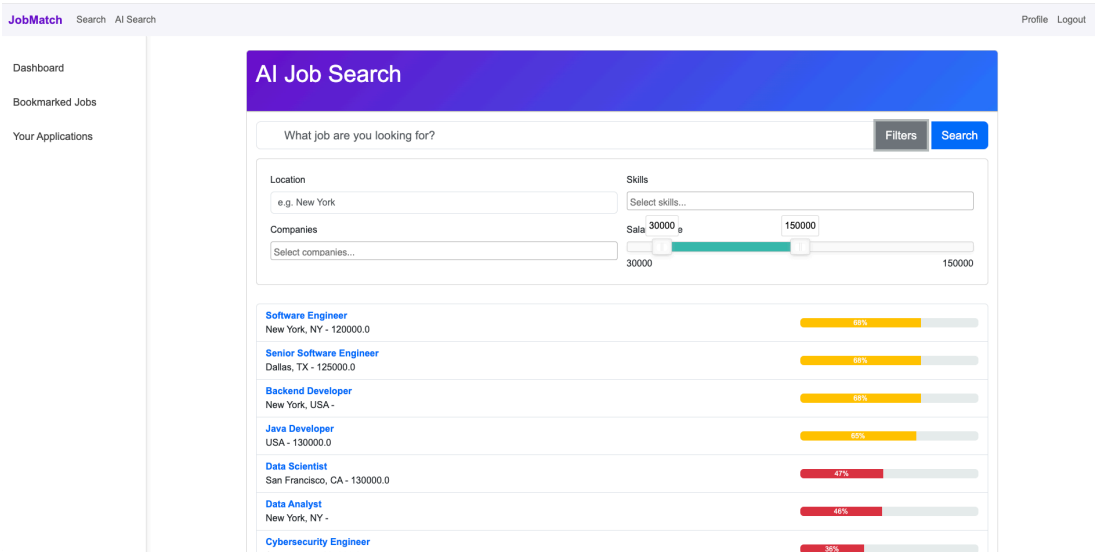

Рисунок 3.3.5 – Сторінка семантичного пошуку вакансій

Разом із оцінкою відповідності, також можна отримати доступ до обґрунтування від моделі. Щоб побачити обґрунтування, достатньо перейти на сторінку вакансії. Скріншот сторінки вакансії наведено на рисунку 3.3.6:

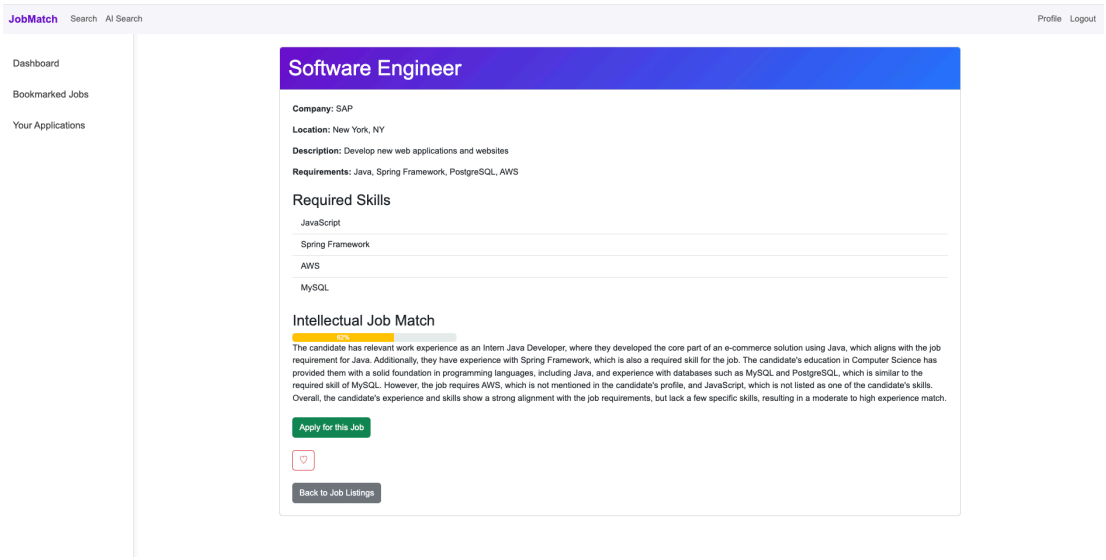

Рисунок 3.3.6 – Сторінка вакансії

Оцінка відповідності, як і генерація обґрунтування, відбуваються за одну операцію. У додатках А та Б наведено відповідно промпт для оцінки відповідності та результат, повернений від моделі.

Наступною важливою функцією веб-платформи є можливість автоматизованого парсингу CV. На рисунку 3.3.7 наведено сторінку для редагування профілю користувача. Щоб скористатися з цієї функції, потрібно завантажити резюме у форматі PDF (весь текст резюме буде відправлено як частину промпту до моделі) та натиснути кнопку „Retrieve data from CV”:

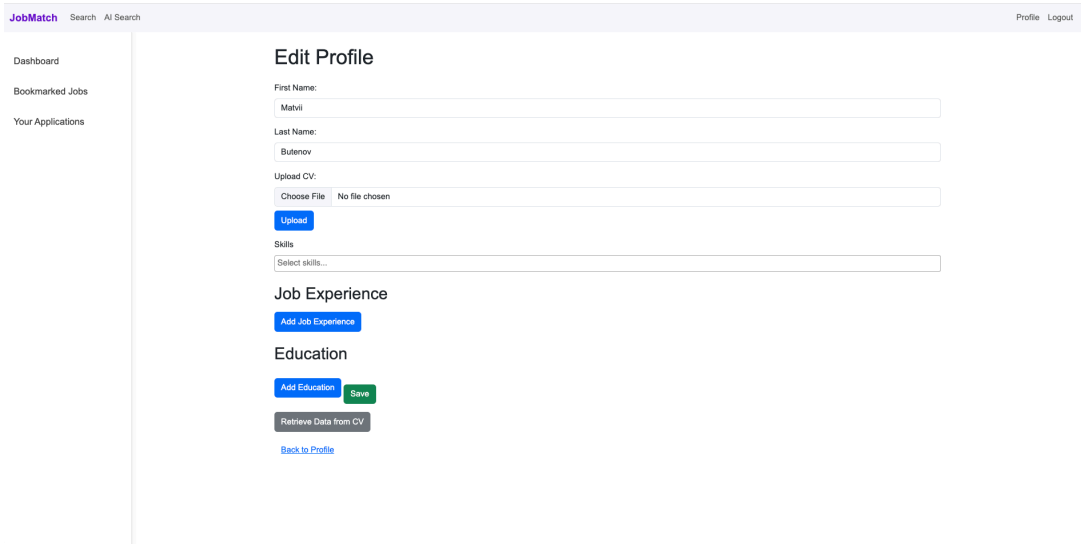

Рисунок 3.3.7 – Сторінка редагування профілю

Після виконання цих операцій, система виконає запит до LLM та заповнить профіль. У додатках В та Г відповідно наведено резюме використане для тестування та відповідь від моделі. На рисунку 3.3.8 показано скріншот профілю користувача після виконання цієї операції.

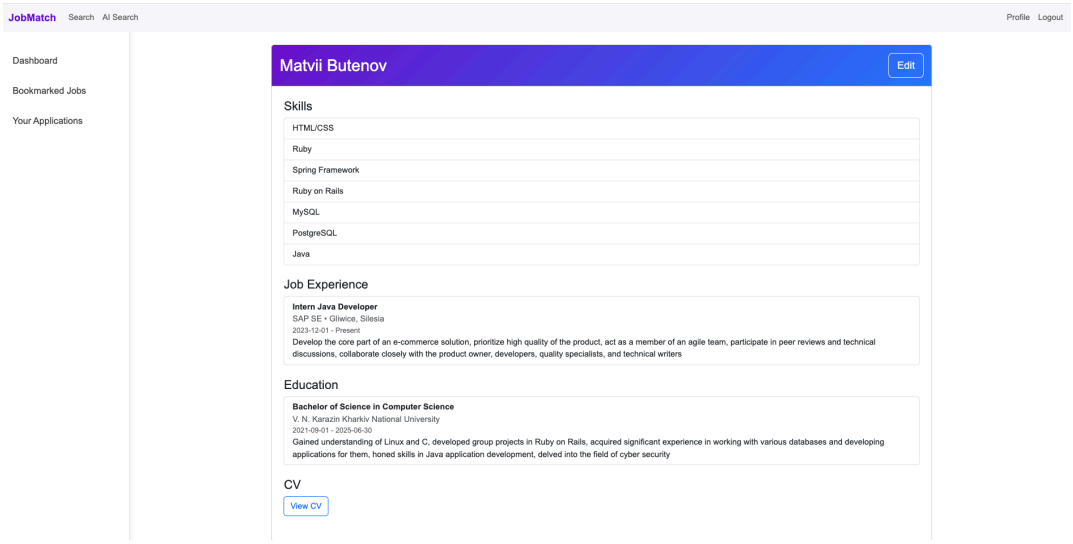

Рисунок 3.3.8 – Сторінка профілю кандидата

Перейдемо до ключових функцій рекрутера. На рисунку 3.3.9 нижче наведено скріншот сторінки dashboard для рекрутера:

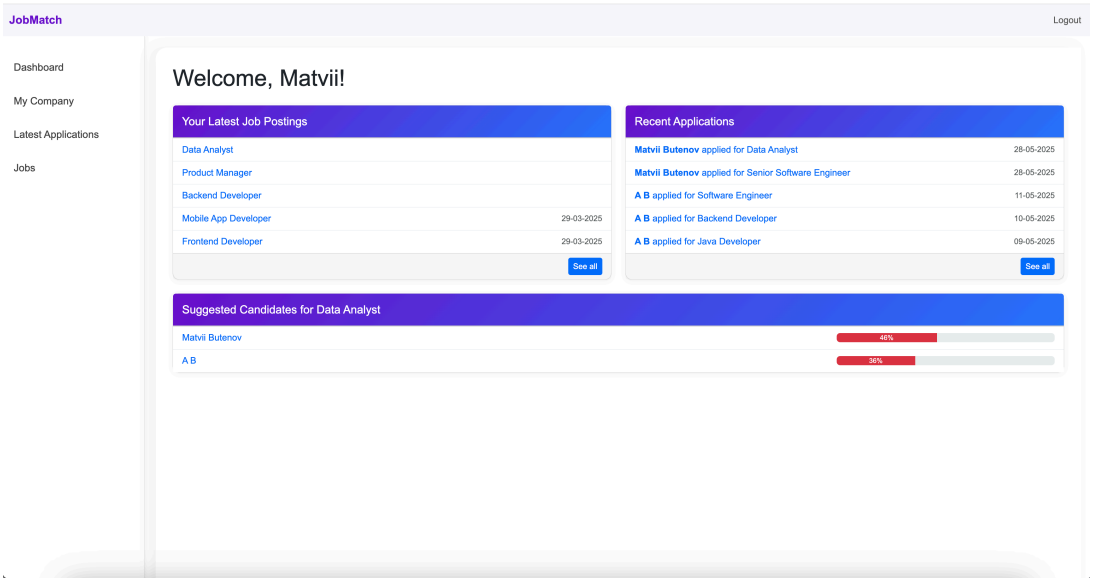

Рисунок 3.3.9 – Сторінка-dashboard рекрутера

Подібно для сторінки-dashboard для кандидата, на цій сторінці наведено найбільш корисні дані для рекрутера – останні вакансії, останні заявки та рекомендовані кандидати для останньої створеної вакансії.

На наступному рисунку 3.3.10 наведено сторінку для семантичного пошуку кандидатів відносно вакансії. Ця сторінка працює за тим самим принципом, що сторінка для семантичного пошуку вакансій відносно кандидата:

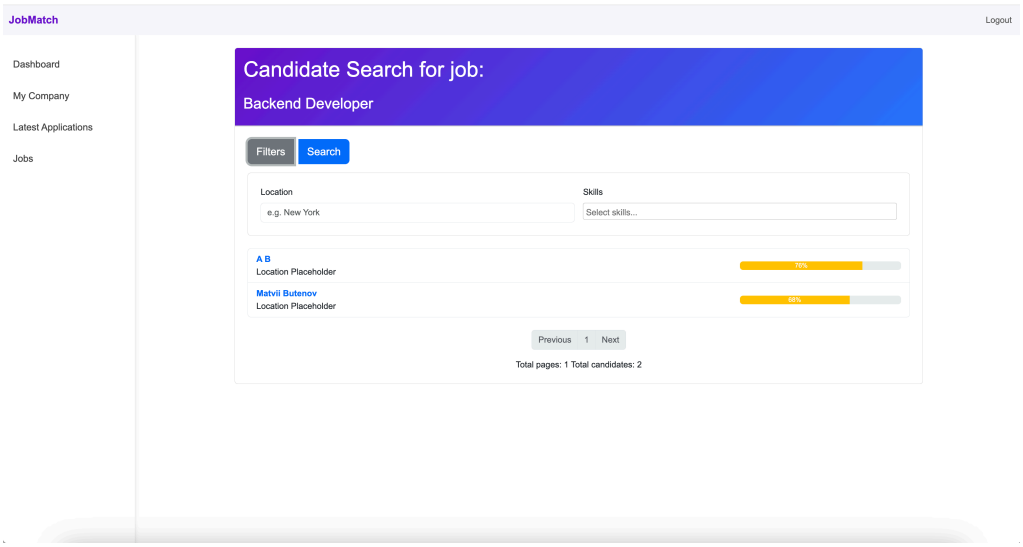

Рисунок 3.3.10 – Сторінка семантичного пошуку кандидатів

Аналогічно, щоб дізнатися обґрунтування оцінки, вистачить зайти на профіль кандидата, обрати потрібну вакансію та натиснути кнопку „Check Match”. Результат цієї операції наведено на рисунку 3.3.11:

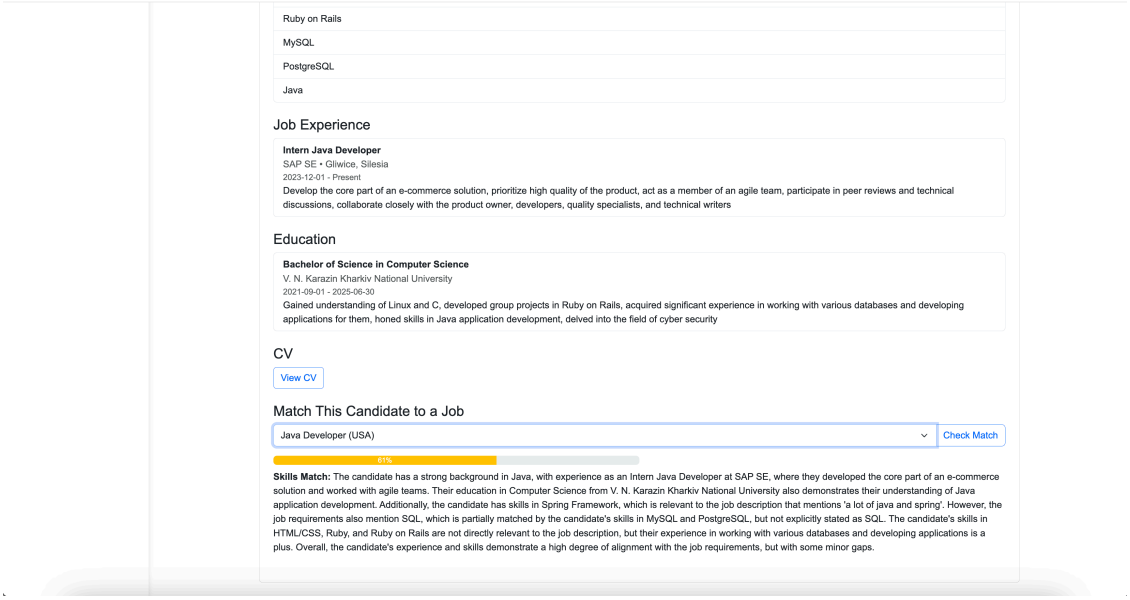

Рисунок 3.3.11 – Профіль кандидата зі сторони рекрутера з оцінкою сумісності відносно обраної вакансії

3.4. Тестування основної функціональності додатку

Для тестування додатку було написано інтеграційні тести за допомогою фреймворку JUnit 5. Через легку інтеграцію з контекстом Spring Boot, фреймворк став найкращим вибором для швидкого створення тестів.

В рамках даного підрозділу ми розглянемо більш детально інтеграційний тест MatchingIntegrationTest (код тесту наведено у додатку Д), що перевіряє сценарій семантичного скорингу відповідності кандидат-вакансія із інтеграцією з LLM. Для цього було створено певний датасет зі сталими вхідними даними (дані наведено у додатку Е). Розглянемо тест покроково:

1) Підготовка тестового оточення

Перед кожним прогоном ми очищаємо всі таблиці (jobCandidateMatchRepository.deleteAll(), jobRepository.deleteAll(),

candidateRepository.deleteAll()), щоб гарантувати відсутність залишків попередніх виконань. Далі створюємо одного кандидата і п’ять вакансій, зберігаючи їх у тестовій базі даних H2 та отримуючи реальні згенеровані ідентифікатори, які потім використовуються у DTO.

2) Формування вхідних DTO

Для кандидата збираємо детальний профіль із п’ятьма навичками, двома позиціями досвіду та двома освітніми записами. Для кожної з п’яти вакансій відповідно створюємо об’єкт із набором вимог і списком необхідних навичок, що відповідають нашим категоріям:

- STRONG1 і STRONG2 (повна та майже повна відповідність),
- PARTIAL1 і PARTIAL2 (часткова відповідність),
- NO_FIT (відсутня відповідність).

3) Перевірка результатів

Після виклику методу та повернення результату перевіряємо, що JobCandidateMatch дійсно збережено у базі. За допомогою switch по категорії вакансії верифікуємо, що отриманий matchScore лежить у попередньо визначених діапазонах:

- STRONG1 > 0.9,
- STRONG2 ≈ 0.75 (0.7–0.8),
- PARTIAL1 ≈ 0.49 (0.4–0.6),
- PARTIAL2 ≈ 0.63 (0.4–0.7),
- NO_FIT < 0.2.

Даний тест виконується 5 разів за допомогою анотації @RepeatedTest.

Це допомагає впевнитися, що результати скорингу є стабільними та більш-менш такі самі для однакових даних. Нижче на рисунку 3.4.1 наведено результат виконання тестів (варто зазначити, що тут застосовується послідовне виконання, тобто час виконання є довший, ніж за допомогою асинхронного виконання):

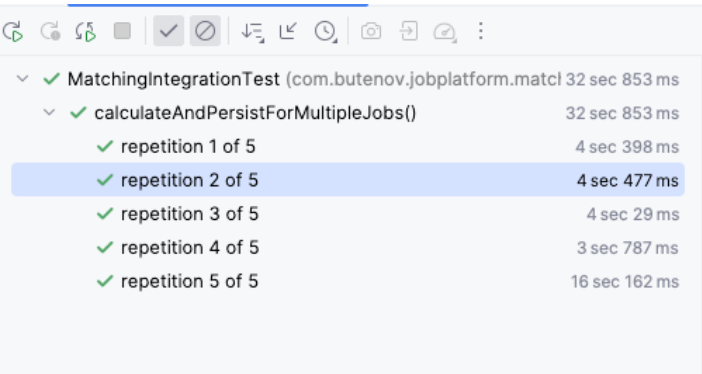

Рисунок 3.4.1 – Результати виконання тестів

За результатами даних тестів можна зробити висновок, що система завжди оцінює пари кандидат-вакансія згідно з очікуваннями. Тобто, кандидат, маючий досвід у розробці на Java та Spring має більше співпадіння з вакансіями, що вимагають навички пов’язані із розробкою на Java (так як вакансії STRONG1 та STRONG2), ніж, наприклад, вакансія NO_FIT, що вимагає навичики у фронтенд розробці.

Також важливо зазначити те, що в кожному з 5 повторів тесту, результати вийшли дуже схожими, що свідчить про високу стабільність результатів семантичної оцінки відповідності. Результати кожного з повторень наведено на таблиці 3.4.1:

Таблиця 3.4.1 – Результати повторень тестів скорингу відповідності

Повторення	**STRONG1**	**STRONG2**	**PARTIAL1**	**PARTIAL2**	**NO_FIT**
1	0.94	0.74667	0.62667	0.53533	0.12
2	0.94	0.74667	0.62667	0.49333	0.12
3	0.94	0.74667	0.62667	0.49333	0.12
4	0.94	0.74667	0.62667	0.49333	0.12
5	0.91	0.74667	0.62667	0.49333	0.12

Незважаючи на стабільність та коректність результатів, наразі складно оцінити, чи такий підхід працює краще ніж з використанням інших підходів, що застосовують існуючі платформи для працевлаштування. Також варто

розуміти, що справжні результати роботи системи матимуть набагато більшу достовірність та вартість для оцінки ефективності, ніж тестові дані.

Підбиваючи підсумок цього підрозділу, можна стверджувати, що для отримання повноцінних результатів, веб-платформа має використовуватись у реальних випадках, з реальними користувачами та вакансіями, але навіть на цьому етапі видно, що система може чітко та стабільно оцінювати пари кандидат-вакансія на відповідність, що і було ціллю цього дослідження.

3.5 Області використання та значущість розробки

Розроблена платформа може потенційно знайти застосування в багатьох корпоративних та галузевих контекстах – поєднання класичного управління вакансіями та кандидатами з інтелектуальним аналізом резюме й описів профілів і вакансій робить її корисною для компаній, що прагнуть централізувати процес рекрутингу. Організації, що мають велике щомісячне надходження заявок, зможуть суттєво скоротити час першого скринінгу за рахунок автоматизованої фільтрації та семантичного скорингу, делегувавши рутинну роботу системі й зосередивши увагу відділу HR на більш складних етапах відбору.

Інтеграція з LLM-моделями також відкриває можливість використання платформи у навчальних закладах або рекрутингових агенціях, де потрібне швидке оцінювання великого числа резюме із формуванням рекомендацій для студентів чи кандидатів. HR у компаніях зможуть додавати власні правила та словники, адаптуючи систему під специфіку ринку праці своїх регіонів або галузей. В цілому, навіть якщо веб-платформа не до кінця реалізує потреби організацій, сам алгоритм скорингу з може бути використаний та застосований у досить ефективний спосіб.

З наукової точки зору, поєднання алгоритмічного фільтра за множинами навичок із семантичним аналізом векторних представлень розширює підходи до інформаційного пошуку в предметній галузі рекрутингу. Запропонований алгоритм може стати основою подальших досліджень в області explainable AI

(XAI), що зосереджується не лише на тому, щоб поєднати кількісні оцінки, але й на тому, щоб надати обґрунтування рішень.

Очевидною є також господарська вартість. Підвищення швидкості та якості підбору персоналу може призвести до економії бюджетів на закриття вакансій та зменшення плинності кадрів. Соціальний ефект полягає у підвищенні доступу кандидатів до рекомендацій, зворотного зв’язку та прозорості процесу відбору, що зміцнює довіру до автоматизованих рекрутингових систем.

Таким чином, реалізована платформа має широку сферу застосування – від внутрішніх HR-систем корпорацій до інтеграції в глобальні системи вакансій, а її науково-практичні результати можуть стати відправною точкою для подальшого розвитку інтеграції AI в рекрутингу.

ВИСНОВКИ

У процесі виконання роботи було створено веб-платформу для працевлаштування співробітників, що об’єднує управління вакансіями та кандидатами в єдиному інтерфейсі на базі Spring Boot та Thymeleaf. Використовуючи гібридний підхід до пошуку кандидатів та вакансій, де спочатку застосовується швидка SQL-фільтрація із подальшим семантичним скорингом за допомогою LLM, вдалося створити алгоритм пошуку, що поєднує швидкість та якість підбору. Також важливим стала автоматизація парсингу резюме з допомогою LLM. Модель продемонструвала свою ефективність у швидкому та точному витяганні ключових даних про освіту, досвід та професійні навички. Це дало можливість користувачам заповнювати профіль базуючись на своєму резюме буквально в одне натискання кнопки. Використання Spring AI-модуля для інтеграції з зовнішнім API моделей забезпечило модульність архітектури та спростило подальші оновлення або заміну моделей.

Результати тестування підтвердили надійність і стабільність системи, а також її зручність для користувачів. Механізм кешування результатів матчингу із автоматичною інвалідацією при зміні даних запобіг дублюванню обчислень і гарантував актуальність інформації без втрати продуктивності.

Серед особливо цінних моментів розробки слід відзначити можливість гнучкої заміни компонентів системи, в тому числі моделей LLM та постачальників, що відкриває шлях до інтеграції як комерційних, так і open-source рішень, а також потенціал для локалізації моделей під різні мови з урахуванням галузевих особливостей.

З огляду на подальший розвиток, доцільно звернути увагу на оптимізацію продуктивності через батчинг запитів до моделей і розширене кешування. Такий підхід дасть змогу знизити затрати на обчислення й масштабувати систему у хмарі з контейнеризацією. Також варто розглянути інтеграцію модулів відео- та текстових чат-ботів для попереднього скринінгу,

підключення зовнішніх HR-систем та соціальних мереж, а також впровадження сучасного фронтенду (наприклад, на React) для покращення UX/UI.

В цілому, реалізоване рішення заклало міцний фундамент для інтелектуалізації процесу підбору персоналу. Запропоновані шляхи вдосконалення дозволять адаптувати платформу до зростаючих потреб ринку з мінімальними додатковими витратами часу та ресурсів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Jobiqo Blog. “5 Online Recruitment Trends for Job Boards to Watch in 2023” // Jobiqo Blog. 2023. URL: https://www.jobiqo.com/blog/5-online-recruitment-trends-for-job-boards-to-watch-in-2023/ (Дата звернення: 02.05.2025)
2. Market.us Scoop. “Online Recruitment Statistics and Facts (2025)” // Market.us Scoop. March 2025. URL: https://scoop.market.us/online-recruitment-statistics/ (Дата звернення: 03.05.2025)
3. Peicheva, M. “DATA ANALYSIS FROM THE APPLICANT TRACKING SYSTEM” // University of National and World Economy. November 2023. URL: https://www.researchgate.net/publication/375517346_DATA_ANALYSIS_FROM_THE_APPLICANT_TRACKING_SYSTEM (Дата звернення: 03.05.2025)
4. Indeed Editorial Team. “12 Modern Recruitment Techniques for Your Hiring Process” // Indeed. February 2025. URL: https://www.indeed.com/career-advice/career-development/modern-recruitment-techniques (Дата звернення: 03.05.2025)
5. impress.ai. “AI-powered Recruiting Software – impress.ai” // impress.ai. 2024. URL: https://www.impress.ai (Дата звернення: 04.05.2025)
6. Bennett, M. “How Generative AI Is Changing the Hiring Process” // Cloudapper.ai. February 2025. URL: https://www.cloudapper.ai/talent-acquisition/how-does-generative-ai-affect-recruitment/ (Дата звернення: 04.05.2025)
7. Alphanumerical Journal. “Resume Screening with Natural Language Processing (NLP)” // Alphanumerical Journal. August 2024. URL: https://alphanumericjournal.com/article/resume-screening-with-natural-language-processing-nlp (Дата звернення: 04.05.2025)

8. Gawande, A. “How Semantic Search is Being Used in AI Recruitment” // Cvviz.com. April 2024. URL: https://cvviz.com/blog/how-semantic-search-used-in-recruitment/ (Дата звернення: 04.05.2025)

9. Bocharova, M., Malakhov, E., & Mezhuyev, V. “VacancySBERT: the approach for representation of titles and skills for semantic similarity search in the recruitment domain” // arXiv. July 31, 2023. c.57. URL: https://arxiv.org/abs/2307.16638 (Дата звернення: 04.05.2025)

10. Muennighoff, N. “SGPT: GPT Sentence Embeddings for Semantic Search” // arXiv. February 17, 2022. . URL: https://arxiv.org/abs/2202.08904 (Дата звернення: 04.05.2025)

11. HireVue. “HireVue AI Hiring Platform Overview” // HireVue. 2024. URL: https://www.hirevue.com/product/ai (Дата звернення: 05.05.2025)

12. Silbiger, S. “The Pymetrics Games - Overview and Practice Guidelines” // Oxford University Careers Service. November 2021. URL: https://www.careers.ox.ac.uk/article/the-pymetrics-games-overview-and-practice-guidelines (Дата звернення: 05.05.2025)

13. Rezi Official Site. “Rezi: AI Resume Builder” // Rezi Official Site. 2024. URL: https://www.rezi.ai/ai-resume-builder (Дата звернення: 05.05.2025)

14. HireVue. “2025 Global Guide to AI in Hiring” // HireVue. 2025. URL: https://www.hirevue.com/resources/report/2025-global-guide-to-ai-in-hiring (Дата звернення: 05.05.2025)

15. OpenAI. “Open AI API” // OpenAI. 2025. URL: https://openai.com/api/ (Дата звернення: 05.05.2025)

16. GroqDocs. “Overview – Groq Cloud” // GroqDocs. 2025. URL: https://console.groq.com (Дата звернення: 05.05.2025)

17. Spring.io. “Overview – Spring AI” // Spring.io. 2025. URL: https://spring.io (Дата звернення: 05.05.2025)

18. IBM. “LLM APIs: Tips for Bridging the Gap” // IBM. 2025. URL: https://www.ibm.com/think/insights/llm-apis (Дата звернення: 06.05.2025)

19. European Data Protection Board. “Opinion 01/2024 on Artificial Intelligence and Data Protection” // European Data Protection Board. March 2024. URL: https://www.edpb.europa.eu/news/news/2024/edpb-opinion-ai-models-gdpr-principles-support-responsible-ai_en (Дата звернення: 06.05.2025)

20. ResearchGate. “Unlocking the Black Box: Explainable Artificial Intelligence (XAI) for Trust and Transparency in AI Systems” // ResearchGate. 2024. URL: https://www.researchgate.net/publication/372042827_Unlocking_the_Black_Box_Explainable_Artificial_Intelligence_XAI_for_Trust_and_Transparency_in_AI_Systems (Дата звернення: 06.05.2025)

21. Terasky. “The Misleading Costs of Private vs Public Inference” // Medium. July 2024. URL: https://medium.com/terasky/the-misleading-costs-of-private-vs-public-inference-be4292729910 (Дата звернення: 06.05.2025)

22. Acclaro Blog. “What are the pros and cons of leveraging AI in localization?” // Acclaro Blog. December 2023. URL: https://www.acclaro.com/blog/pros-and-cons-of-ai-localization/#:~:text=On%20one%20hand%2C%20it%20offers,only%20humans%20can%20fully%20understand. (Дата звернення: 07.05.2025)

23. Tam Recruiting. “Human Element of Recruiting Cannot Be Replaced by Artificial Intelligence” // Tam Recruiting. October 2020. URL: https://tamrecruiting.com/2017-01-18-the-human-element-of-recruiting-cannot-be-replaced-by-artificial-intelligence/ (Дата звернення: 07.05.2025)

24. GroqCloud Documentation. “Llama-3.3-70B-Versatile Documentation” // GroqCloud Documentation. 2025. URL: https://console.groq.com/docs/model/llama-3.3-70b-versatile (Дата звернення: 07.05.2025)

25. Kirkovska, A. “Llama 3.3 70b vs GPT-4o” // Vellum. December 2024. URL: https://www.vellum.ai/blog/llama-3-3-70b-vs-gpt-4o (Дата звернення: 07.05.2025)

ДОДАТОК А. ПРОМПТ ДЛЯ СЕМАНТИЧНОГО СКОРИНГУ ПАРИ „КАНДИДАТ-ВАКАНСІЯ”

You are an expert in recruitment and job matching. Analyze the provided candidate profile and compare it to the job description. Evaluate the relevance of the candidate’s work experience, education, and skills to the job requirements. Assign an experience match score on a scale from 0 to 1, where 0 represents minimal relevance and 1 represents a highly suitable match. The score should vary based on the degree of alignment, rather than being strictly 0 or 1. Justify your score with a concise, structured explanation. Avoid including the score itself in the justification. Format the output as JSON with the following fields: { 'experience_match': float, 'justification': string }." Candidate JSON:
{"jobExperiences":[{"id":74,"createdAt":"2025-05-28T21:51:49.460+00:00","updatedAt":"2025-05-28T21:51:49.460+00:00","jobTitle":"Intern Java Developer","companyName":"SAP SE","location":"Gliwice, Silesia","startDate":"2023-12-01","endDate":null,"description":"Develop the core part of an e-commerce solution, prioritize high quality of the product, act as a member of an agile team, participate in peer reviews and technical discussions, collaborate closely with the product owner, developers, quality specialists, and technical writers"}],"skills":[{"id":17,"createdAt":null,"updatedAt":null,"name":"HTML/CSS"},{"id":9,"createdAt":null,"updatedAt":null,"name":"Ruby"},{"id":28,"createdAt":null,"updatedAt":null,"name":"Spring Framework"},{"id":29,"createdAt":null,"updatedAt":null,"name":"Ruby on Rails"},{"id":51,"createdAt":null,"updatedAt":null,"name":"MySQL"},{"id":52,"createdAt":null,"updatedAt":null,"name":"PostgreSQL"},{"id":1,"createdAt":null,"updatedAt":null,"name":"Java"}],"educations":[{"id":78,"createdAt":"2025-05-28T21:51:49.454+00:00","updatedAt":"2025-05-28T21:51:49.454+00:00","degree":"Bachelor of Science","schoolName":"V. N. Karazin Kharkiv National University","fieldOfStudy":"Computer Science","startDate":"2021-09-01","endDate":"2025-06-30","description":"Gained understanding of Linux and C, developed group projects in Ruby on Rails, acquired significant experience in working with various databases and developing applications for them, honed skills in Java application development, delved into the field of cyber security"}]} Job JSON: {"title":"Software Engineer","description":"Develop new web applications and websites","location":null,"requirements":"Java, Spring Framework, PostgreSQL, AWS","requiredSkills":[{"id":7,"createdAt":null,"updatedAt":null,"name":"JavaScript"},{"id":28,"createdAt":null,"updatedAt":null,"name":"Spring

Framework"},{"id":43,"createdAt":null,"updatedAt":null,"name":"A
WS"},{"id":51,"createdAt":null,"updatedAt":null,"name":"MySQL"}]
}

ДОДАТОК Б. JSON ВІДПОВІДЬ ВІД LLM ДЛЯ ПРОМПТУ З ДОДАТКУ А


```
{
  "experience_match": 0.7,
  "justification": "The candidate has relevant work
experience as an Intern Java Developer, where they developed the
core part of an e-commerce solution using Java, which aligns
with the job requirement for Java. Additionally, they have
experience with Spring Framework, which is also a required skill
for the job. The candidate's education in Computer Science has
provided them with a solid foundation in programming languages,
including Java, and experience with databases such as MySQL and
PostgreSQL, which is similar to the required skill of MySQL.
However, the job requires AWS, which is not mentioned in the
candidate's profile, and JavaScript, which is not listed as one
of the candidate's skills. Overall, the candidate's experience
and skills show a strong alignment with the job requirements,
but lack a few specific skills, resulting in a moderate to high
experience match."
}
```

ДОДАТОК В. РЕЗЮМЕ ВИКОРИСТАНЕ ДЛЯ ТЕСТУВАННЯ LLM ПАРСИНГУ

[redacted]

CONTACT

[redacted]

SKILLS

- Java, Spring Framework, Java EE
- JUnit, Mockito
- SAP Commerce Cloud
- Ruby, Ruby on Rails
- HTML, CSS, Bootstrap 5, JSP, Thymeleaf
- Hibernate, JDBC, Spring JPA
- Maven, Gradle, Ant
- MySQL, PostgreSQL

LANGUAGES

English
Advanced (C1)

Polish
Advanced (C1)

Russian
Native

Ukrainian
Native

CERTIFICATIONS AND LICENSES

Security Champion Bootcamp by SAP - 2024
PRACTICAL JAVA by SoftServe – 2023

[redacted]

Java Developer

PROFESSIONAL SUMMARY

I am a **dedicated** and **passionate Java Developer** with valuable commercial experience gained as an **Intern Java Developer** at **SAP SE**. Over the past **2+ years**, I have actively immersed myself in the **Java ecosystem**, continuously learning and honing my skills. This includes the development of **numerous pet projects** (link) and ongoing participation in remote **Computer Science studies at V. N. Karazin Kharkiv National University**. My time at **SAP SE** has provided me with hands-on experience and deepened my understanding of **Java development** practices within a **professional setting**.

WORK HISTORY

Intern Java Developer 12/2023 - Current
SAP SE - Gliwice, Silesia
I have gained valuable experience working as an **Intern Java Developer** at **SAP SE** on the **SAP Commerce Cloud's** platform project. During my time in this role, I have been engaged in the following tasks:

- Develop the **core** part of an **e-commerce** solution during my work
- Prioritize **high quality** of the product throughout the software development cycle
- Act as a member of an **agile team** within an international organization
- Participate in **peer reviews** and **technical discussions** with the team
- Collaborate closely with the **product owner**, **developers**, **quality specialists**, and **technical writers**

In addition to my general responsibilities, I actively pursued **security studies** by taking bootcamps to **contribute** to the project as a **Security Champion** for our team. I also took the **lead** in the **multi-module migration** to **JUnit5** while keeping **backward compatibility** and **assisted a newcomer** in getting acclimated to the project.

EDUCATION

Bachelor of Science: Computer Science, 09/2021 – 06/2025
V. N. Karazin Kharkiv National University – Kharkiv, Ukraine
During my **remote studies at Karazin University**, I undertook various projects that significantly enhanced and sharpened my programming skills. Specifically, I worked extensively with **C**, **Java**, and **Ruby on Rails** improving my ability to write efficient and functional code.

Key highlights from studies:

- Gained understanding of **Linux** and **C**
- Developed group projects in **Ruby on Rails**, showcasing my ability to work **collaboratively in a team setting** (link #1, link #2)
- Acquired **significant experience** in working with various **databases** and developing applications for them
- Honed my skills in **Java application development** creating functional and efficient code
- Delved into the field of **cyber security** gaining valuable knowledge into **safeguarding** digital systems and networks

Рисунок В.1 – Резюме використане для тестування LLM парсингу

ДОДАТОК Г. JSON ВІДПОВІДЬ ВІД LLM ДЛЯ ПАРСИНГУ РЕЗЮМЕ З ДОДАТКУ В


```
{
  "jobExperiences": [
    {
      "jobTitle": "Intern Java Developer",
      "companyName": "SAP SE",
      "location": "Gliwice, Silesia",
      "startDate": "2023-12-01",
      "endDate": "",
      "description": "Develop the core part of an e-
commerce solution, prioritize high quality of the product, act
as a member of an agile team, participate in peer reviews and
technical discussions, collaborate closely with the product
owner, developers, quality specialists, and technical writers"
    }
  ],
  "educations": [
    {
      "degree": "Bachelor of Science",
      "schoolName": "V. N. Karazin Kharkiv National
University",
      "fieldOfStudy": "Computer Science",
      "startDate": "2021-09-01",
      "endDate": "2025-06-30",
      "description": "Gained understanding of Linux and C,
developed group projects in Ruby on Rails, acquired significant
experience in working with various databases and developing
applications for them, honed skills in Java application
development, delved into the field of cyber security"
    }
  ],
  "skills": [
    "Java",
    "Spring Framework",
    "Java EE",
    "JUnit",
    "Mockito",
    "SAP Commerce Cloud",
    "Ruby",
    "Ruby on Rails",
    "HTML",
    "CSS",
    "Bootstrap 5",
    "JSP",
    "Thymeleaf",
    "Hibernate",
    "JDBC",
    "Spring JPA",
    "Maven",
```

```
    "Gradle",
    "Ant",
    "MySQL",
    "PostgreSQL"
  ]
}
```

ДОДАТОК Д. ВИХІДНИЙ КОД ТЕСТУ MATCHINGINTEGRATIONTEST

```
@SpringBootTest(classes = { JobPlatformApplication.class })
@TestPropertySource(locations = "classpath:application-
test.properties")
@ActiveProfiles("test")
class MatchingIntegrationTest
{

    @Autowired
    private JobRepository jobRepository;

    @Autowired
    private CandidateRepository candidateRepository;

    @Autowired
    private JobCandidateMatchRepository
jobCandidateMatchRepository;

    @Autowired
    private MatchScoreCalculationAsyncExecutor executor;

    private Job job;
    private Candidate candidate;

    @BeforeEach
    void setUp()
    {
       jobCandidateMatchRepository.deleteAll();
       jobRepository.deleteAll();
       candidateRepository.deleteAll();

       job = jobRepository.save(new Job());

       candidate = candidateRepository.save(new Candidate());
    }

    @RepeatedTest(5)
    void calculateAndPersistForMultipleJobs()
    {
       // 1) Persist and categorize five jobs
       enum JobCategory
       {STRONG1, STRONG2, PARTIAL1, PARTIAL2, NO_FIT}
       final Map<JobCategory, Long> jobIds = new
EnumMap<>(JobCategory.class);

       for (final JobCategory category : JobCategory.values())
       {
          Job tempjob = new Job();
```

```
            tempjob = jobRepository.save(tempjob);
            jobIds.put(category, tempjob.getId());
        }

        // 2) Build one rich candidate DTO
        final CandidateMatchingDto candidateDto =
CandidateMatchingDto.builder()

.id(candidate.getId())

.skills(Stream.of("Java", "Spring Boot", "Docker", "SQL",

"Python")

.map(Skill::new)

.toList())

.jobExperiences(List.of(

new JobExperience("Senior Java Developer", "Acme",

"Warsaw",

LocalDate.of(2020, 1, 1),

LocalDate.of(2023, 1, 1),

"Built microservices", null),

new JobExperience("DevOps Engineer", "BuildIt",

"Remote",

LocalDate.of(2019, 1, 1),

LocalDate.of(2020, 1, 1),

"Automated Docker CI/CD", null)

))

.educations(List.of(

new Education("MSc CS", "Warsaw University",

"Software Engineering",

LocalDate.of(2017, 9, 1),

LocalDate.of(2019, 6, 30),

"Thesis on distributed systems", null),
```

```
new Education("BSc IT", "Tech Academy",

"Computer Networks",

LocalDate.of(2013, 9, 1),

LocalDate.of(2017, 6, 30),

"Focused on network security", null)

))

.build();

        // 3) Build job DTOs mapped by category
        final Map<JobCategory, JobMatchingDto> jobDtos = Map.of(
              JobCategory.STRONG1, JobMatchingDto.builder()

.id(jobIds.get(JobCategory.STRONG1))

.title("Java/Spring Boot Backend Engineer")

.requirements("Java, Spring Boot, Docker, SQL")

.requiredSkills(Stream.of("Java", "Spring Boot", "Docker",
"SQL")

.map(Skill::new)

.toList())
                                                    .build(),

              JobCategory.STRONG2, JobMatchingDto.builder()

.id(jobIds.get(JobCategory.STRONG2))
                                                    .title("Backend
Java Developer")

.requirements("Java, Spring Boot, Hibernate")

.requiredSkills(Stream.of("Java", "Spring Boot", "Hibernate")

.map(Skill::new)

.toList())
                                                    .build(),

              JobCategory.PARTIAL1, JobMatchingDto.builder()

.id(jobIds.get(JobCategory.PARTIAL1))
                                                     .title("Data
Analyst")
```

```
.requirements("Python, SQL, Excel")

.requiredSkills(

Stream.of("Python", "SQL", "Excel").map(Skill::new).toList())
                                                    .build(),

            JobCategory.PARTIAL2, JobMatchingDto.builder()

.id(jobIds.get(JobCategory.PARTIAL2))
                                                    .title("DevOps
Engineer")

.requirements("Docker, Jenkins, AWS")

.requiredSkills(

Stream.of("Docker", "Jenkins", "AWS").map(Skill::new).toList())
                                                    .build(),

            JobCategory.NO_FIT, JobMatchingDto.builder()

.id(jobIds.get(JobCategory.NO_FIT))
                                                .title("Frontend
Engineer")

.requirements("Angular, TypeScript, CSS")

.requiredSkills(Stream.of("Angular", "TypeScript", "CSS")

.map(Skill::new)

.toList())
                                                .build()
       );

       for (final var entry : jobDtos.entrySet())
       {
          final JobCategory category = entry.getKey();
          final JobMatchingDto jobDto = entry.getValue();

          final JobCandidateMatch match =
executor.calculateAndStoreMatchScore(jobDto, candidateDto)
                                                     .join();

          final double score = match.getMatchScore();
          System.out.printf("Category %s → score=%.5f%n",
category, score);

          assertTrue(jobCandidateMatchRepository
                        .findByJobAndCandidate(match.getJob(),
match.getCandidate())
```

```
                    .isPresent(),
                "Match not persisted for " + category);

        switch (category)
        {
           case STRONG1 -> assertTrue(score > 0.9,
                 () -> "Expected >0.9 for STRONG1, got " +
score);
           case STRONG2 -> assertTrue(score > 0.7 && score <
0.8,
                 () -> "Expected ~0.75 for STRONG2, got " +
score);
           case PARTIAL1 -> assertTrue(score > 0.4 && score <
0.7,
                 () -> "Expected ~0.49 for PARTIAL1, got " +
score);
           case PARTIAL2 -> assertTrue(score > 0.4 && score <
0.7,
                 () -> "Expected ~0.63 for PARTIAL2, got " +
score);
           case NO_FIT -> assertTrue(score < 0.2,
                 () -> "Expected <0.2 for NO_FIT, got " +
score);
        }
     }
  }


  @Configuration
  static class AsyncTestConfig implements AsyncConfigurer
  {
     @Bean(name = "llmExecutor")
     public ThreadPoolTaskExecutor llmExecutor()
     {
        ThreadPoolTaskExecutor exec = new
ThreadPoolTaskExecutor();
        exec.setCorePoolSize(2);
        exec.afterPropertiesSet();
        return exec;
     }
  }
}
```

ДОДАТОК Е. ДАНІ ДЛЯ ТЕСТУ MATCHINGINTEGRATIONTEST

Кандидат (єдиний профіль для всіх сценаріїв)

Skills:

- Java
- Spring Boot
- Docker
- SQL
- Python

Job Experiences:

1. Senior Java Developer, Acme (Warsaw), 2020-01-01 … 2023-01-01, «Built microservices»
2. DevOps Engineer, BuildIt (Remote), 2019-01-01 … 2020-01-01, «Automated Docker CI/CD»

Educations:

1. MSc Computer Science, Warsaw University, 2017-09-01 … 2019-06-30, «Thesis on distributed systems»
2. BSc Information Technology, Tech Academy, 2013-09-01 … 2017-06-30, «Focused on network security»

Вакансії наведено нижче на таблиці Е.1:

Таблиця Е.1 – Тестові дані вакансій

Категорія	Назва вакансії	Requirements	Required Skills
STRONG1	Java/Spring Boot Backend Engineer	Java, Spring Boot, Docker, SQL	Java, Spring Boot, Docker, SQL
STRONG2	Backend Java Developer	Java, Spring Boot, Hibernate	Java, Spring Boot, Hibernate
PARTIAL1	Data Analyst	Python, SQL, Excel	Python, SQL, Excel
PARTIAL2	DevOps Engineer	Docker, Jenkins, AWS	Docker, Jenkins, AWS
NO_FIT	Frontend Engineer	Angular, TypeScript, CSS	Angular, TypeScript, CSS