Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
магістр
(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ РЕКРУТИНГУ ТА ВІДБОРУ КАНДИДАТІВ ДО КОМАНДИ РОЗРОБНИКІВ ІТ ПРОЄКТІВ
RESEARCH AND DEVELOPMENT OF A SOFTWARE COMPONENT FOR IT PROJECT RECRUTING AUTOMATION AND CANDIDATE SELECTION

Виконав(ла): студент(ка) 2 курсу, групи КНТ-124м
Спеціальності 121 Інженерія програмного забезпечення
(код і найменування спеціальності)
Освітня програма (спеціалізація)
Інженерія програмного забезпечення

ПІСКУН О.О.
(ПРІЗВИЩЕ та ініціали)
Керівник ТАБУНЩИК Г.В.
(ПРІЗВИЩЕ та ініціали)
Рецензент ШИТІКОВА О.В.
(ПРІЗВИЩЕ та ініціали)

2025

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____ Сергій СУББОТІН
“____” _____2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ПІСКУНА Олексія Олександровича
(ПРІЗВИЩЕ, ім’я, по батькові)

1. Тема проєкту (роботи) Дослідження та розробка програмного забезпечення для автоматизації рекрутингу та відбору кандидатів до команди розробників IT проєктів. Research and Development of a Software Component for IT Project Recruting Automation and Candidate Selection
керівник проєкту (роботи) к.т.н., професор, ТАБУНЩИК Галина Володимирівна,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім’я, по батькові)
затверджені наказом закладу вищої освіти від “ 30 ” вересня 2025 року № 447
2. Строк подання студентом проєкту (роботи) 08 грудня 2025 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Дослідження та аналіз предметної області. 2. Матеріали і методи. 3. Проєктування програмної системи. 4. Основні рішення щодо реалізації програмного забезпечення. 5. Експлуатація, тестування та експериментальне дослідження застосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	ТАБУНЩИК Г.В., професор		
Нормоконтролер	БЄЛОВА А.В., асистент		

7. Дата видачі завдання “ 30 ” вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів	4-5 тижні	Розділ 2
	вирішення задачі.		
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7-8 тижні	Розділ 4
6	Тестування та експериментальне дослідження	9 тиждень	Розділ 5
	програмного забезпечення.		
7	Оформлення пояснювальної записки та документів до неї.	10-11 тижні	Додатки
8	Нормоконтроль та рецензування.	12 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка) _____ Олексій ПІСКУН
(підпис) (І’мя ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Галина ТАБУНЩИК
(підпис) (І’мя ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра: 123 с., 14 табл., 62 рис., 2 дод., 35 джерел.

ВЕБ-ЗАСТОСУНОК, РЕКРУТИНГ, JAVA, SPRING BOOT, АЛГОРИТМ РАНЖУВАННЯ, АНАЛІЗ БІЗНЕС-ПРОЦЕСІВ.

Об’єкт дослідження – процес інженерії програмного забезпечення для автоматизації рекрутингу та формування проєктних команд в IT-компаніях.

Предмет дослідження – методи та програмні засоби інтелектуального підбору кандидатів на основі зваженого оцінювання навичок.

Мета роботи – мінімізація часових витрат та зменшення трудомісткості процесів найму шляхом розробки спеціалізованої інформаційної системи, що забезпечує автоматизований пошук, ранжування та розподіл кандидатів за проєктами.

Матеріали, методи та технічні засоби: об’єктно-орієнтоване програмування, мова моделювання UML, мова програмування Java, фреймворк Spring Boot (MVC, Security, Data), система керування базами даних PostgreSQL, технології веб-інтерфейсів (Thymeleaf, HTML, CSS).

Результати. Створено веб-застосунок для HR-менеджерів та кандидатів, який реалізує ведення бази кандидатів, налаштування вакансій з урахуванням пріоритетності (ваги) навичок та автоматичний розрахунок рейтингу відповідності (Match Score), що дозволяє миттєво знаходити найбільш релевантних фахівців.

Висновки. Розроблено програмну систему, яка успішно вирішує задачу автоматизації рутинних операцій рекрутингу, забезпечує об’єктивність оцінювання кандидатів за математичним алгоритмом та значно прискорює процес формування команд розробників.

Галузь використання – кадрові відділи (HR) IT-компаній, рекрутингові агентства, департаменти управління персоналом.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 123 pages, 14 tables, 62 figures, 2 appendixes, 35 sources.

WEB APPLICATION, RECRUITMENT, JAVA, SPRING BOOT, RANKING ALGORITHM, BUSINESS PROCESS ANALYSIS.

Object of research – the software engineering process for automating recruitment and forming project teams in IT companies.

Subject of research – methods and software tools for intelligent candidate selection based on weighted skill assessment.

Goal of the work – minimization of time expenditure and reduction of labor intensity in hiring processes by developing a specialized information system that ensures automated search, ranking, and distribution of candidates across projects.

Materials, methods, and technical means: object-oriented programming, UML modeling language, Java programming language, Spring Boot framework (MVC, Security, Data), PostgreSQL database management system, web interface technologies (Thymeleaf, HTML, CSS).

Results. A web application for HR managers and candidates has been created, which implements candidate database management, vacancy configuration taking into account skill priority (weight), and automatic calculation of the compliance rating (Match Score), allowing for the instant identification of the most relevant specialists.

Conclusions. A software system has been developed that successfully solves the task of automating routine recruitment operations, ensures the objectivity of candidate assessment using a mathematical algorithm, and significantly accelerates the process of forming development teams.

Field of application – HR departments of IT companies, recruitment agencies, personnel management departments.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

ATS – Applicant Tracking System;

CRUD – Create, Read, Update, Delete;

CSS – Cascading Style Sheets;

HTML – HyperText Markup Language;

HTTP – HyperText Transfer Protocol;

HR – Human Resources;

IDE – Integrated Development Environment;

JAR – Java ARchive;

JDK – Java Development Kit;

JRE – Java Runtime Environment;

JVM – Java Virtual Machine;

MVC – Model-View-Controller;

NLP – Natural Language Processing;

REST – Representational State Transfer;

SQL – Structured Query Language;

UI – User Interface;

UML – Unified Modeling Language;

UX – User Experience;

ІС – інформаційна система;

ПЗ – програмне забезпечення;

СКБД – система керування базами даних.

ВСТУП

У сучасному цифровому світі сфера інформаційних технологій залишається однією з найбільш динамічних галузей глобальної та української економіки. Стрімка цифровізація бізнесу та перехід до індустрії 4.0 провокують постійне зростання попиту на кваліфікованих ІТ-фахівців. Однак, паралельно зі збільшенням кількості вакансій та проектів, ускладнюється і процес залучення талантів.

Традиційні підходи до рекрутингу, що базуються на ручній обробці резюме та суб'єктивній оцінці, стають неефективними в умовах висококонкурентного ринку та дефіциту часу [1]. Ситуація додатково ускладнюється глобальним переходом на віддалений формат роботи та розподілені команди, особливо для України, в умовах воєнного стану. Рекрутери змушені опрацьовувати величезні масиви даних, враховуючи технічні навички (hard skills) та специфічні вимоги проєкту.

За таких умов критично важливим стає впровадження спеціалізованих інформаційних систем, здатних автоматизувати рутинні операції, систематизувати дані про кандидатів та надавати інструменти для об'єктивного відбору. Саме тому темою дипломного проєкту стало дослідження спеціалізованої інформаційної системи для автоматизації рекрутингових процесів в ІТ-сфері [2].

Метою роботи стала мінімізація часових витрат та зменшення трудомісткості процесів рекрутингу та формування команд розробників ІТ-компанії шляхом створення спеціалізованого програмного забезпечення, що забезпечує підбір кандидатів, їх оцінку відповідно до вимог проєктів.

Програмний продукт проектується як веб-додаток зі зручним графічним інтерфейсом, архітектура якого базується на взаємодії серверної частини та реляційної бази даних для надійного збереження та швидкої обробки масивів даних.

1 ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність розробки програмного рішення

В умовах переходу до цифрової економіки людський капітал стає ключовим активом будь-якої технологічної компанії. Індустрія розробки програмного забезпечення (IT) характеризується високою динамікою змін технологічних стеків, стрімким зростанням кількості проєктів та жорсткою конкуренцією за кваліфікованих фахівців. За таких умов традиційні підходи до управління персоналом, що базуються на ручній обробці даних, стають стримуючим фактором для розвитку бізнесу.

Актуальність розробки спеціалізованого програмного забезпечення зумовлена низкою факторів, які можна класифікувати як операційні, економічні та технологічні.

Операційні проблеми існуючих підходів. Незважаючи на високий рівень цифровізації самої IT-галузі, процеси найму в багатьох компаніях малого та середнього бізнесу в Україні залишаються недостатньо автоматизованими. Значна частина рекрутерів продовжує використовувати неспеціалізовані інструменти, такі як електронна пошта та електронні таблиці (Excel, Google Sheets). Такий підхід має суттєві недоліки:

– фрагментація даних. Резюме надходять з різних каналів (LinkedIn, Djinni, Work.ua) і накопичуються у поштових скриньках, що призводить до дублювання профілів та втрати історії комунікації;

– низька швидкість обробки. Необхідність ручного перенесення даних з файлів резюме до зведених таблиць забирає значну частину робочого часу, який доцільніше витратити на проведення співбесід та оцінку м’яких навичок (soft skills) кандидата;

– відсутність командної взаємодії. При використанні локальних файлів виникають конфлікти версій та проблеми зі спільним доступом до бази кандидатів для кількох менеджерів одночасно.

Економічний аспект та стан ринку. Світова практика демонструє активне використання систем класу HCM (Human Capital Management) та ATS (Applicant Tracking Systems) [3]. Однак, імплементація таких рішень часто є фінансово обтяжливою для українського середнього бізнесу через високу вартість ліцензій та складності інтеграції. Більшість доступних на ринку готових рішень є універсальними і не враховують специфіку проєктної розробки, де критично важливим є не просто факт найму, а формування збалансованої команди з урахуванням грейдів (рівнів кваліфікації) та сумісності технологічного стеку.

Головна мета сучасного рекрутингу трансформується з простого «закриття вакансій» у стратегічне завдання найму співробітників, які здатні забезпечити стабільний розвиток компанії. Автоматизація дозволяє змістити фокус уваги HR-спеціаліста з рутинних операцій на аналітичну діяльність. Програмне забезпечення повинно виступати фільтром, що відсіює нерелевантних кандидатів ще на етапі первинного скринінгу, що значно підвищує ймовірність підбору ефективного персоналу.

Специфіка IT-сфери вимагає врахування великої кількості параметрів при відборі. На відміну від інших галузей, тут критично важливою є деталізація навичок (наприклад, не просто «Java», а «Java + Spring Boot + Microservices»). Людині складно утримувати в пам’яті матрицю компетенцій усіх кандидатів та об’єктивно зіставляти їх з вимогами вакансії. Програмна автоматизація дозволяє вирішити цю проблему шляхом застосування алгоритмів чіткого співставлення (matching), що мінімізує вплив людського фактору та знижує ризик помилкового найму.

Крім того, поширення віддаленого формату роботи вимагає створення єдиного інформаційного простору, де зберігається вся історія взаємодії з кандидатом, результати технічних співбесід та статус розгляду. Це забезпечує прозорість процесів та дозволяє швидко повертатися до «резервних» кандидатів при відкритті нових проєктів.

Таким чином, розробка власного програмного продукту є своєчасною та економічно обґрунтованою відповіддю на поточні виклики ринку праці. Це дозволить не лише автоматизувати рутинні операції та структурувати базу знань компанії, але й значно прискорити процес формування проектних команд, що є критичним фактором конкурентоспроможності в сучасній IT-індустрії.

1.2 Огляд існуючих рішень проблеми

Для обґрунтування доцільності створення власної інформаційної системи необхідно проаналізувати поточний стан ринку програмного забезпечення у сфері управління персоналом (HRM) та автоматизації рекрутингу (ATS - Applicant Tracking Systems). Для детального аналізу було обрано три поширені системи, що використовуються в IT-індустрії: Workday HCM (корпоративний стандарт), Greenhouse (популярна хмарна ATS) та CleverStaff (спеціалізоване рішення для рекрутерів).

"Workday HCM" – це комплексна хмарна система управління людським капіталом, яка використовується великими міжнародними корпораціями. Вона охоплює всі аспекти життєвого циклу співробітника: від найму та адаптації до нарахування зарплати та управління талантами [4]. У контексті рекрутингу Workday дозволяє створювати заявки на вакансії, налаштовувати маршрути погодження та зберігати дані про кандидатів (рис. 1.1).

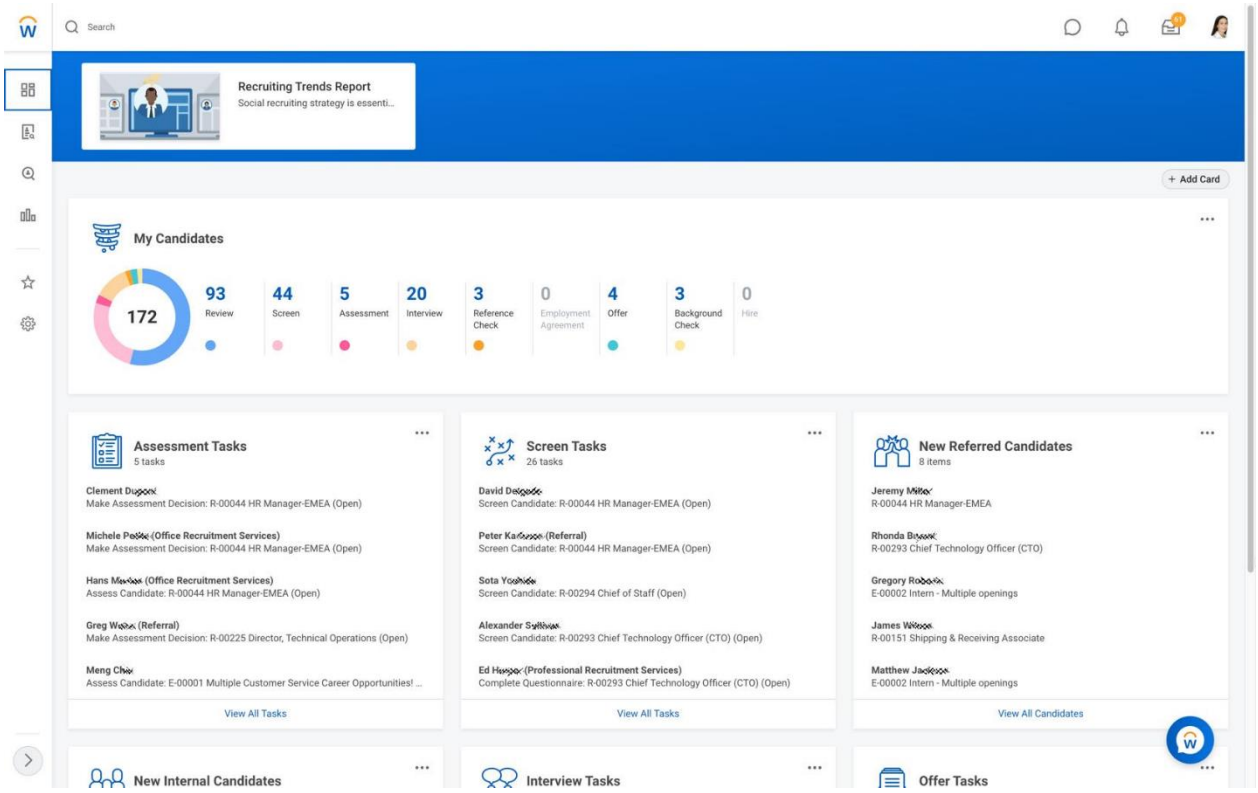


Рисунок 1.1 – Інтерфейс системи "Workday HCM"

Функціональні можливості системи включають:

– повну інтеграцію кадрових процесів з фінансами та плалнуванням;

– потужний модуль аналітики та звітності;

– можливість налаштування складних бізнес-процесів погодження найму;

– високий рівень безпеки даних.

До недоліків даного рішення потрібно віднести:

– надзвичайно високу вартість ліцензії та впровадження, що робить її недоступною для малого та середнього бізнесу;

– складний та перевантажений інтерфейс, який вимагає тривалого навчання персоналу;

– відсутність гнучкості: система орієнтована на стандартизовані корпоративні процеси, а не на специфіку швидкого формування IT-команд;

– складність адаптації під конкретний стек технологій без залучення консультантів.

"Greenhouse" – це спеціалізована платформа для автоматизації рекрутингу, яка фокусується на оптимізації самого процесу найму (Pipeline). Основна ідея системи – структурування етапів співбесід та збір фідбеку від інтерв'юерів для прийняття неупереджених рішень (рис. 1.2) [5].

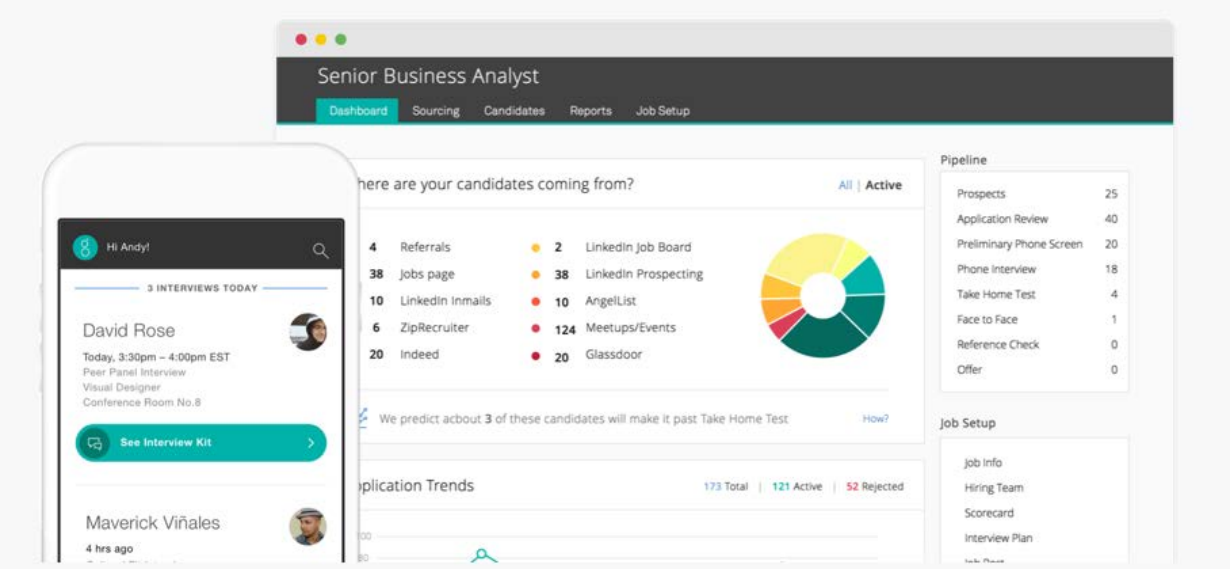


Рисунок 1.2 – Застосунок "Greenhouse"

До позитивних сторін платформи можна віднести наступне:

– зручний та інтуїтивно зрозумілий інтерфейс для рекрутерів;

– автоматизована розсилка листів кандидатам та планування співбесід;

– наявність інструментів для боротьби з упередженістю при наймі (Scorecards);

– широкі можливості інтеграції з іншими сервісами (Slack, Zoom, Gmail).

До недоліків даного ресурсу потрібно віднести:

– фокус на процесі, а не на базі знань, система не має зручних інструментів для пошуку серед кандидатів, у тому числі у складі команди, за складними критеріями навичок;

– відсутність функціоналу для формування команд: система працює в розрізі "Одна вакансія – Один найм", а не в розрізі "Проєкт – Команда";

– висока вартість підписки, що залежить від кількості співробітників компанії.

"CleverStaff" — це професійне програмне забезпечення для автоматизації рекрутингу, розроблене в Україні. Система відома своїми можливостями парсингу (автоматичного розпізнавання) резюме та інтеграцією з локальними сайтами пошуку роботи (рис. 1.3) [6].

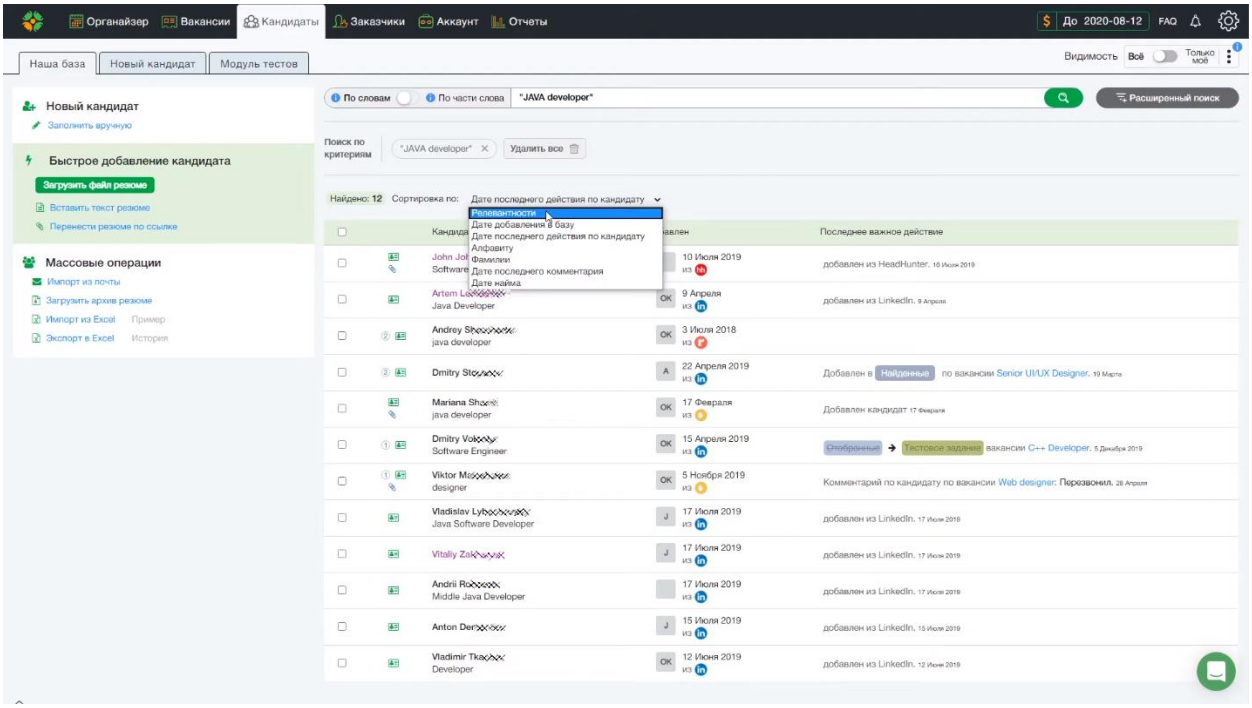


Рисунок 1.3 – Система "CleverStaff"

До плюсів можна віднести:

– автоматичне збереження резюме з LinkedIn та роботних сайтів в один клік;

– візуалізацію воронки рекрутингу у вигляді Kanban-дошки;

– наявність історії спілкування з кандидатом;

– порівняно доступну цінову політику для українського ринку.

До мінусів можна віднести:

– SaaS-модель розповсюдження: дані зберігаються на хмарних серверах провайдера, що може суперечити політикам безпеки деяких IT-компаній;

– обмежені можливості фільтрації за рівнем володіння конкретними технологіями (грейдами);

– відсутність вбудованого алгоритму для автоматичного підбору (метчингу) кандидатів під вимоги проєкту на основі вагових коефіцієнтів навичок.

1.3 Порівняльний аналіз з аналогами та обґрунтування розробки власного застосунку

Для визначення доцільності розробки власної системи автоматизації рекрутингу було проведено порівняльний аналіз функціональних можливостей розглянутих аналогів та проєктованої системи. Головним критерієм порівняння стала здатність системи вирішувати специфічні завдання IT-сектору: деталізований облік технічних навичок та формування команд. Результати аналізу наведено в табл. 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів та розроблюваної системи

Критерій порівняння	Workday HCM	Greenhouse	CleverStaff	Власна розробка (System)
1	2	3	4	5
Спеціалізація	Корпоративний сектор (General)	Процес найму (Pipeline)	Автоматизація рекрутингу	IT-рекрутинг та формування команд
Пошук за навичками (Hard Skills)	Базовий	За ключовими словами	За тегами	Алгоритмічний (за ID та грейдом)
Модуль "Проєктна команда"	Ні	Ні	Ні	Так (розподіл по проєктах)

Кінець таблиці 1.1

1	2	3	4	5
Облік рівня навичок (1-5)	Текстовий опис	Оцінка інтерв'юера	Текстовий опис	Числовий параметр у БД
Можливість кастомізації	Низька (складно)	Середня	Низька	Висока (власний код)
Вартість впровадження	Дуже висока	Висока (підписка)	Середня (підписка)	Низька (Internal tool)
Зручність інтерфейсу (UI/UX)	Низька	Висока	Середня	Висока (адаптована під задачі)

Як видно з табл. 1.1, існуючі комерційні рішення мають суттєві обмеження у контексті поставленої мети:

– універсальність замість спеціалізації. Більшість систем (Greenhouse, Workday) орієнтовані на найм будь-якого персоналу (бухгалтерів, менеджерів, водіїв), тому вони не мають глибокої деталізації технічних скілів, яка критично необхідна в IT (наприклад, різниця між Java 8 та Java 17, або рівень володіння Docker);

– відсутність проєктного підходу. Жоден із розглянутих аналогів не має вбудованого модуля для розподілу кандидатів по конкретних командах (Teams) з відображенням опису проєкту. Вони працюють за принципом "Закрити вакансію", тоді як наша мета — "Сформувати команду";

– алгоритми пошуку. Існуючі системи здебільшого шукають за текстовим входженням (Full-text search) у резюме. Розроблювана система використовує реляційну модель зв'язків «Кандидат — Навичка — Рівень», що дозволяє виконувати точний пошук (наприклад, знайти всіх Middle Java розробників, які знають SQL не нижче рівня 4);

– економічна доцільність. Використання готових SaaS-рішень вимагає постійних щомісячних витрат, тоді як власна розробка є одноразовою інвестицією, що дозволяє адаптувати продукт під унікальні бізнес-процеси компанії без залежності від вендора.

Проведений аналіз показав, що на ринку відсутній доступний програмний продукт, який би поєднував функції бази даних кандидатів з детальним урахуванням технічного стеку та інструментарієм для комплектації проектних команд. Розробка власної системи дозволить заповнити цю прогалину, надаючи компанії ефективний інструмент для швидкого та якісного відбору IT-фахівців.

1.4 Постановка задачі дослідження

На основі проведеного аналізу предметної області та існуючих рішень можна сформулювати основну задачу дипломного проєкту. Вона полягає у створенні інформаційної системи, що дозволяє автоматизувати процеси формування проєктних команд в IT-компанії.

Необхідно розробити програмний компонент, який вирішує проблему багатокритеріального вибору кандидатів на вакантні посади. Система повинна оперувати такими сутностями як "Кандидат", "Навичка (Skill)", "Рівень кваліфікації (Grade)" та "Проєктна роль".

Вимоги до функціональності системи:

– облік кандидатів. Система повинна забезпечувати зберігання розширених профілів кандидатів, включаючи історію їхнього досвіду, контактні дані та деталізований перелік технічних навичок з оцінкою рівня володіння (від 1 до 5);

– управління вакансіями. HR-менеджер повинен мати можливість створювати опис вакансії, вказуючи не лише текстовий опис, а й набір обов'язкових навичок та їх пріоритетність (вагу);

– алгоритмічний підбір. Система повинна реалізовувати алгоритм ранжування кандидатів, який автоматично розраховує відсоток відповідності кандидата вимогам конкретної вакансії, враховуючи як наявність необхідних навичок, так і їх рівень;

– формування команд. Реалізація механізму групування відібраних кандидатів у проєктні команди (Teams) з можливістю перегляду складу команди та ролей учасників;

– рольова модель доступу. Забезпечення розділення прав доступу для ролей "Адміністратор/HR" (повний доступ до бази) та "Кандидат" (доступ лише до власного профілю та відкритих вакансій);

– вимоги до технологічної реалізації. Система повинна бути реалізована як веб-додаток з використанням клієнт-серверної архітектури. Серверна частина повинна забезпечувати надійність бізнес-логіки та безпеку даних, а клієнтська — надавати зручний та інтуїтивно зрозумілий інтерфейс. База даних повинна бути реляційною та нормалізованою для забезпечення цілісності даних та швидкості виконання пошукових запитів.

Вирішення поставленої задачі дозволить мінімізувати суб'єктивність при відборі персоналу та скоротити час на формування ефективних команд розробників.

1.5 Висновки за розділом 1

У першому розділі проведено комплексне дослідження предметної області рекрутингу в IT-сфері та проаналізовано існуючі підходи до формування проектних команд.

Встановлено актуальність теми. Визначено, що в умовах динамічного розвитку ринку та поширення віддаленої роботи, традиційні методи ручного відбору персоналу є неефективними. Вони призводять до значних часових витрат та підвищують ризик помилкового найму через людський фактор.

Проаналізовано ринок. Огляд існуючих програмних рішень (Workday, Greenhouse, CleverStaff) показав, що більшість з них орієнтовані на універсальні процеси найму ("воронку"), є фінансово затратними для середнього бізнесу та не мають спеціалізованого функціоналу для деталізованого обліку технічних навичок (грейдів) і автоматичного формування команд.

Виявлено проблему. На ринку відсутній доступний інструмент, який би дозволяв автоматизувати процес підбору («метчингу») кандидатів під конкретний стек технологій проекту з урахуванням вагових коефіцієнтів навичок.

Сформульовано задачу. На основі проведеного аналізу поставлено задачу розробки власної інформаційної системи, яка вирішить виявлені проблеми шляхом автоматизації процесів збору, оцінювання та групування кандидатів.

Отримані результати є підґрунтям для подальшої розробки математичних моделей та архітектури програмного забезпечення в наступних розділах.

2 МАТЕРІАЛИ ТА МЕТОДИ

2.1 Формалізація та моделювання бізнес-процесів рекрутингу

Для побудови ефективної інформаційної системи необхідно формалізувати предметну область та визначити ключові етапи трансформації вхідних даних (резюме, вакансії) у вихідні (сформована команда). Для структурного аналізу процесів доцільно використати методологію функціонального моделювання IDEF0 [7]-[8].

2.1.1 Аналіз поточної ситуації (Модель «AS-IS»)

На початковому етапі було проаналізовано існуючий бізнес-процес підбору персоналу без використання спеціалізованого ПЗ. Контекстна діаграма процесу зображена на рис. 2.1. Вхідними даними є потреба у нових співробітниках та критерії до кандидата, а вихідними — сформована команда розробників. Управлінням виступають правила та процедури компанії, а механізмами — рекрутер та HR-менеджер.

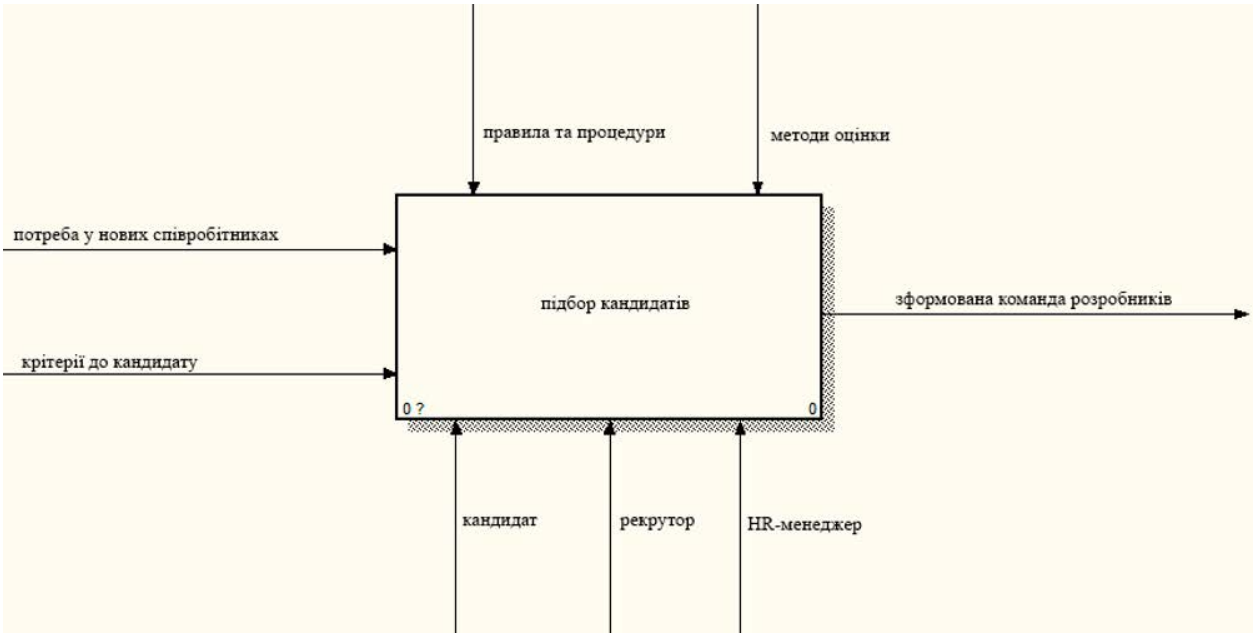


Рисунок 2.1 – Контекстна діаграма «Підбор кандидатів»

Декомпозиція головного процесу (рис. 2.2) дозволяє виділити наступні недоліки ручного керування:

– часові затримки, етап «Збір резюме» та «Відбір та аналіз» виконується вручну з використанням електронної пошти та Excel, що призводить до накопичення неструктурованих даних;

– людський фактор, на етапі «Відбір та аналіз резюме» рекрутер суб'єктивно оцінює відповідність навичок, що може призвести до відсіювання релевантних кандидатів;

– відсутність єдиного сховища, інформація про результати співбесід часто втрачається або не систематизується.

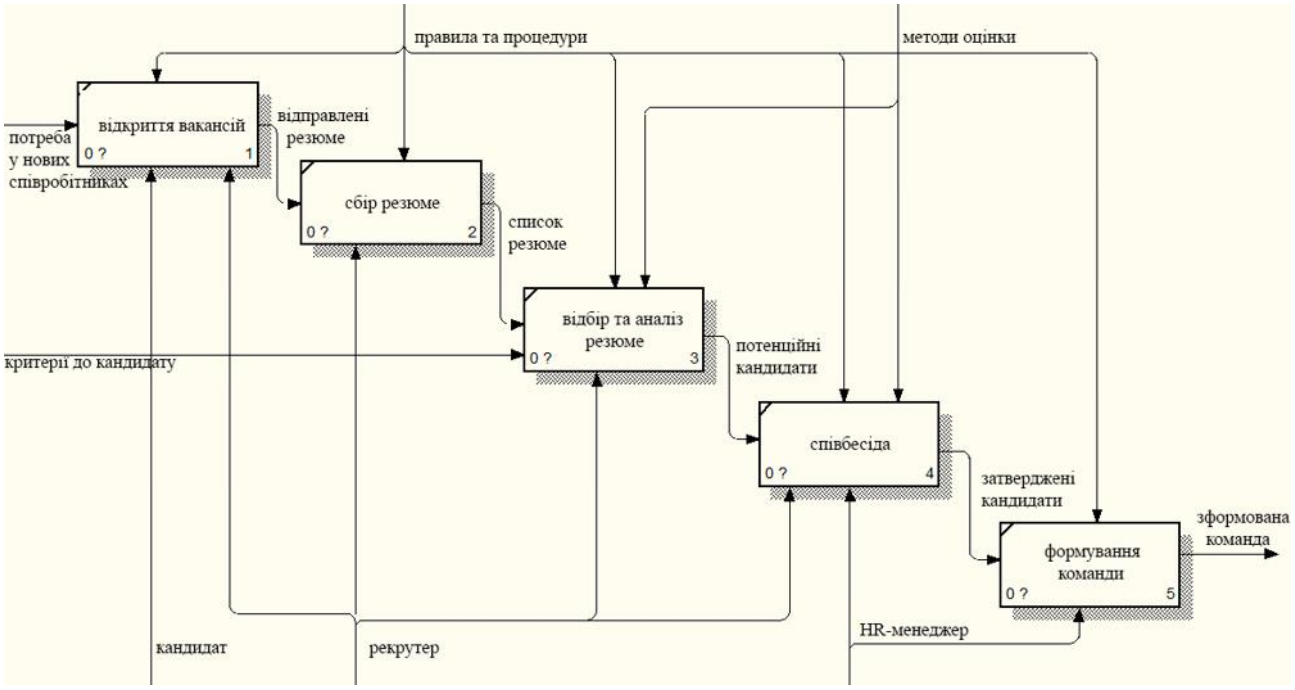


Рисунок 2.2 – Декомпозована діаграма «Підбор кандидатів»

2.1.2 Проєктування удосконаленого процесу (Модель «TO-BE»)

Впровадження розроблюваної інформаційної системи дозволяє реорганізувати бізнес-процеси (Business Process Reengineering) [9]. На рис. 2.3 наведено контекстну діаграму моделі «TO-BE», де в якості механізму додається «Інформаційна система».

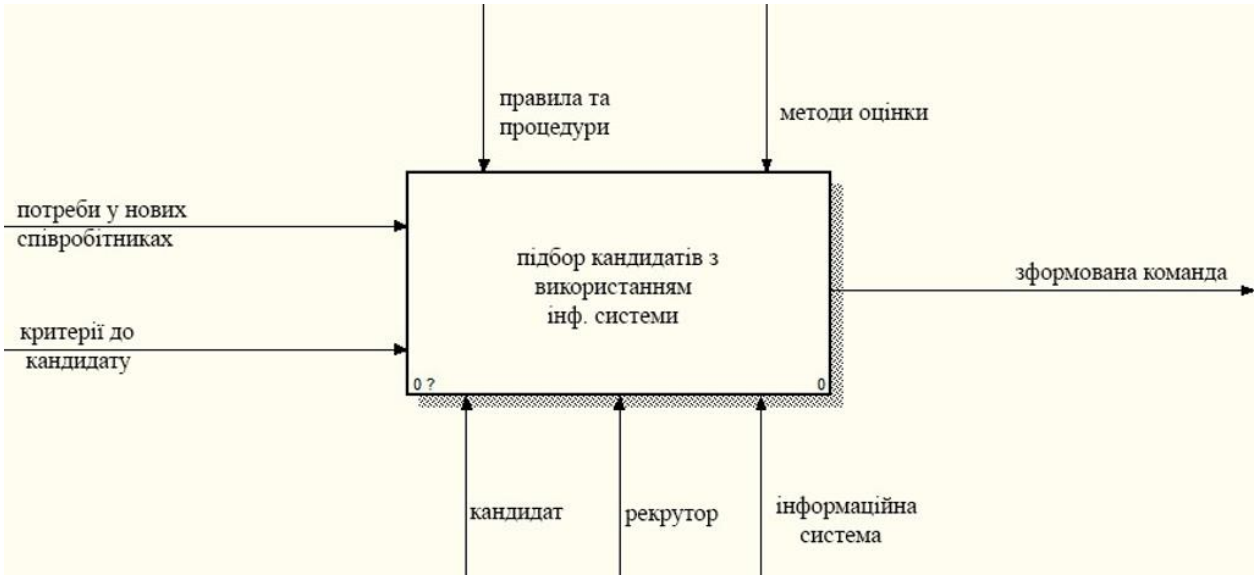


Рисунок 2.3 – Змінена контекстна діаграма «Підбор кандидатів»

Декомпозиція удосконаленого процесу (рис. 2.4) демонструє наступні зміни:

– автоматизація збору даних, процес «Заповнення резюме» тепер виконується кандидатом через веб-інтерфейс системи, що забезпечує надходження вже структурованих даних у базу;

– алгоритмічний аналіз, етап «Оцінка та аналіз резюме» виконується системою автоматично на основі розроблених математичних моделей (див. п. 2.2), що виключає необхідність ручного перегляду сотень нерелевантних анкет;

– прискорення прийняття рішень, рекрутер отримує вже ранжований список кандидатів, що дозволяє зосередитися на етапі «Співбесіда» та «Формування команди».

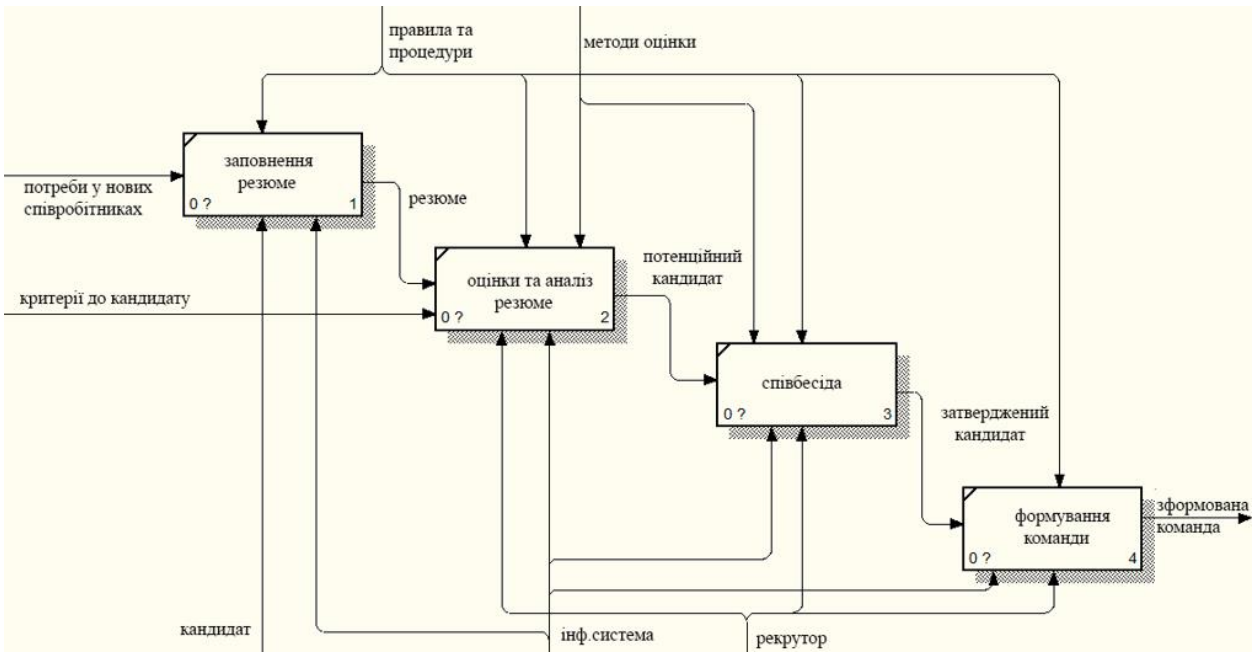


Рисунок 2.4 – Змінена декомпозована діаграма «Підбір кандидатів»

Таким чином, перехід від моделі «AS-IS» до моделі «TO-BE» дозволяє виключити рутинні операції сортування даних та передати функцію первинного скринінгу програмному забезпеченню.

2.2 Математична модель оцінювання відповідності кандидата

Головною проблемою існуючих систем рекрутингу є бінарний підхід до пошуку, який оперує логікою «навичка є» або «навички немає». У рамках даної роботи пропонується використовувати зважену модель багатокритеріального оцінювання, яка враховує не лише факт наявності навички, але й рівень володіння нею та пріоритетність (вагу) для конкретного проєкту [10].

Формалізуємо задачу. Нехай.

$C = \{c_1, c_2, \dots, c_m\}$ – множина кандидатів у базі даних.

$S = \{s_1, s_2, \dots, s_n\}$ – множина всіх можливих технічних навичок (Java, SQL, Docker тощо).

V – вакансія, що характеризується вектором вимог.

Вимоги вакансії V можна представити у виді множини трійок за формулою (2.1):

$$R_v = \{(s_i, w_i, l_{req}) \mid s_i \in S\}, \quad (2.1)$$

де s_i – ідентифікатор навички;

$w_i \in (0, 1]$ – ваговий коефіцієнт (важливість) навички для проєкту;

$l_{req} \in \{1..5\}$ – мінімально необхідний рівень володіння (грейд).

Кожен кандидат s_i має набір компетенцій, де l_{act} – фактичний рівень володіння навичкою s_i.

Цільова функція рейтингу (Match Score) для кандидата c_j відносно вакансії V розраховується як відношення набраної суми балів до максимально можливої суми балів ідеалного кандидата.Спочатку визначимо коефіцієнт відповідності k_i для кожної окремої навички за формулою (2.2):

$$k_i = \begin{cases} 1, \text{якщо } l_{act} \geq l_{req} \\ \frac{l_{act}}{l_{req}}, \text{якщо } 0 < l_{act} < l_{req}, \\ 0, \quad \text{якщо } l_{act} = 0 \end{cases} \quad (2.2)$$

Загальний рейтинг відповідності у відсотках визначається за формулою середньозваженого значення (2.3):

$$Rating(c_j, V) = \frac{\sum_{i=1}^{N}(k_i \cdot w_i)}{\sum_{i=1}^{N} w_i} \cdot 100\%, \quad (2.3)$$

де N – кількість вимог у вакансії.

Такий підхід дозволяє реалізувати "м'який" пошук. Наприклад, якщо вакансія вимагає "Java (вага 1.0)" та "Docker (вага 0.3)", кандидат, який ідеально знає Java, але погано знає Docker, отримає високий бал (близько 85-

90%) і потрапить у топ списку. У класичних булевих системах такий кандидат міг бути відсіяний через невідповідність фільтру "Docker".

Запропонована модель наближає автоматизований пошук до логіки прийняття рішень людиною-експертом.

2.3 Розробка алгоритму пошуку та ранжування

Для програмної реалізації математичної моделі (п. 2.2) розроблено алгоритм фільтрації та сортування бази даних кандидатів. Враховуючи потенційно великий обсяг вхідних даних, алгоритм розділено на два етапи: попередня фільтрація на рівні СКБД та точний розрахунок на рівні сервера застосунків (Application Server).

Груба фільтрація (Hard Filters). На цьому етапі відсіюються кандидати, які не відповідають критичним критеріям, що не підлягають зважуванню (наприклад, статус "В пошуку"). Це виконується за допомогою SQL-запиту, що дозволяє зменшити вибірку даних для подальшої обробки в оперативній пам'яті [11].

Обчислення метрики відповідності (Scoring). Отриманий список кандидатів обробляється алгоритмом, блок-схема якого наведена на рис. 2.5.

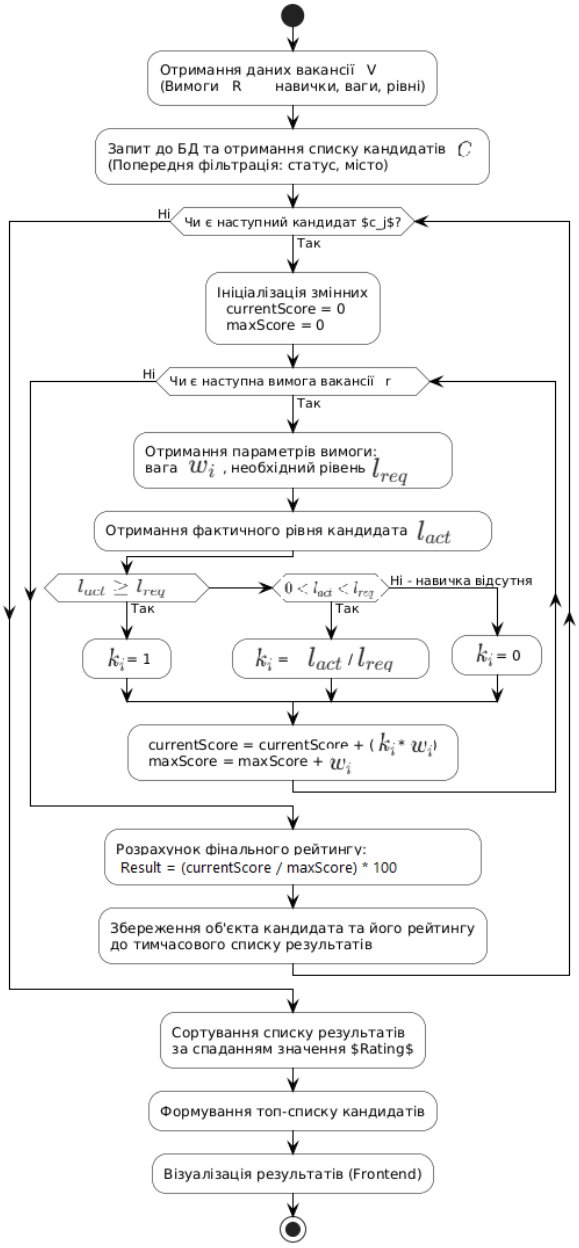


Рисунок 2.5 – Блок-схема алгоритму підбору кандидатів

Оцінка складності алгоритму, нехай *M* – кількість кандидатів, а *N* – середня кількість вимог у вакансії. Складність обчислення рейтингу для одного кандидата становить *O(N).* Загальна часова складність алгоритму становить *O(M • N)* для розрахунку та *O(M log M)* для сортування результатів. Оскільки *N* є малою константою (зазвичай вакансія містить 5-15 навичок), алгоритм має лінійну складність відносно кількості кандидатів, що забезпечує високу швидкодію системи.

2.4 Висновки за розділом 2

У другому розділі виконано формалізацію та моделювання процесів автоматизації рекрутингу.

Проведено реінжиніринг бізнес-процесів, побудовано функціональні моделі, які наочно демонструють, що впровадження системи дозволяє виключити етап ручної обробки неструктурованих резюме та змістити фокус роботи HR-менеджера на прийняття рішень.

Розроблено математичну модель, запропоновано метод зваженого оцінювання відповідності кандидата, який, на відміну від бінарних фільтрів, враховує не лише наявність навичок, а й рівень володіння ними та їх пріоритетність для конкретного проєкту. Це дозволяє підвищити точність ранжування кандидатів.

Розроблено алгоритм пошуку, що включає два етапи фільтрації (грубу на рівні БД та точну на рівні сервера), що забезпечує лінійну складність обчислень та високу швидкодію системи при роботі з великими масивами даних.

Результати цього розділу є теоретичним базисом для програмної реалізації системи, яка буде розглянута в наступному розділі.

3 ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 Обґрунтування вибору мови програмування

Сучасна індустрія розробки корпоративного програмного забезпечення (Enterprise Development) характеризується широким вибором інструментарію. При проєктуванні архітектури системи рекрутингу, яка передбачає складну бізнес-логіку (алгоритми метчингу), роботу з реляційними даними та високі вимоги до безпеки, критично важливим етапом є вибір базової мови програмування.

На сьогоднішній день для реалізації серверної частини (Backend) веб-додатків найчастіше розглядаються три основні технологічні стеки: Java, C# (.NET) та Python. Для обґрунтованого вибору необхідно проаналізувати їх характеристики в контексті поставленої задачі.

Java, ця мова протягом багатьох років залишається "золотим стандартом" для розробки надійних серверних систем. Це об'єктно-орієнтована мова зі суворою статичною типізацією, що виконується на віртуальній машині (JVM) [12].

Перевагами э сувора типізація, яка дозволяє виявляти більшість помилок ще на етапі компіляції, що критично для підтримки великої кодової бази. Екосистема Java має найпотужніші інструменти для роботи з базами даних (JPA/Hibernate) та безпекою (Spring Security). Крім того, Java забезпечує відмінну багатопотоковість, що важливо для одночасної обробки запитів від багатьох рекрутерів.

Недоліком виступає дещо більший обсяг коду порівняно з Python, проте сучасні версії (Java 17+) та використання бібліотеки Lombok нівелюють цю проблему [13].

C# (платформа .NET) – це головний конкурент Java від компанії Microsoft. Має схожий синтаксис та можливості.

Особливостями C# є те, що він пропонує чудову інтеграцію з продуктами Microsoft (Azure, Windows Server). Однак, в середовищі відкритих технологій (Linux-сервери, Docker-контейнери) історично Java мала кращу підтримку, хоча .NET Core наразі вирішує ці питання. Для даного проєкту використання C# вимагало б зміни всієї інфраструктури розробки.

Python – це інтерпретована мова з динамічною типізацією, яка набула популярності завдяки простоті та використанню в Data Science. Python дозволяє писати код дуже швидко. Проте динамічна типізація стає проблемою при зростанні проєкту – рефакторинг коду стає небезпечним, оскільки типи перевіряються лише під час виконання (Runtime). Також швидкість виконання складних математичних алгоритмів (метчинг кандидатів) на чистому Python суттєво поступається компільованим мовам (Java/C#).

Для наочного порівняння характеристики розглянутих мов зведено у табл. 3.1.

Таблиця 3.1 – Порівняльна характеристика мов програмування

Характеристика	Java	C# (.NET)	Python
Типізація	Сувора статична	Сувора статична	Динамічна
Швидкодія	Висока (JIT компіляція)	Висока	Середня/Низька
Екосистема Enterprise	Дуже розвинена (Spring)	Розвинена (ASP.NET)	Помірна (Django/FastAPI)
Багатопотоковість	Відмінна	Відмінна	Обмежена (GIL)
Складність підтримки	Низька (завдяки типам)	Низька	Висока (у великих проєктах)
Кросплатформність	Повна (JVM)	Повна (.NET Core)	Повна (Інтерпретатор)

На основі проведеного аналізу для реалізації системи обрано мову Java. Цей вибір обумовлений необхідністю побудови надійної архітектури зі суворою типізацією даних (грейди, навички, ваги) та наявністю потужного фреймворку Spring, який значно прискорює розробку.

3.2 Обґрунтування вибору фреймворку веб-розробки

Вибір "чистої" мови програмування є недостатнім для ефективної розробки. Сучасні веб-додатки будуються на базі фреймворків, які беруть на себе рутинні задачі: обробку HTTP-запитів, з'єднання з БД, управління залежностями. Розглянемо найпопулярніші рішення у світі Java: Spring Boot, Jakarta EE (раніше Java EE) та Play Framework.

Spring Boot – це еволюція екосистеми Spring, яка фокусується на принципі «Convention over Configuration» (угода важливіша конфігурації) [14].

Перевагою є вбудований сервер (Tomcat), автоматичне налаштування залежностей та потужний механізм ін’єкції залежностей (IoC Container). Модуль Spring Data JPA дозволяє працювати з базою даних через інтерфейси, взагалі не пишучи SQL-запитів для стандартних операцій. Spring Security забезпечує готовий механізм авторизації та захисту від атак (CSRF, XSS) [15].

Недоліком є високий поріг входження для новачків через велику кількість "магії" (автоматичних процесів, прихованих від розробника).

Jakarta EE (Java EE) – це класичний стандарт корпоративної розробки. Вимагає окремого встановлення та налаштування важковагового сервера застосунків (наприклад, WildFly або GlassFish). Цикл розробки та розгортання (deploy) значно повільніший, ніж у Spring Boot. Технологія вважається застарілою для нових проєктів, де важлива гнучкість та мікросервісна архітектура.

Play Framework – це фреймворк, орієнтований на асинхронну обробку даних. Забезпечує високу продуктивність, але має меншу спільноту та складнішу інтеграцію зі стандартними Java-бібліотеками порівняно зі Spring.

Порівняння фрейморків наведено у табл. 3.2.

Таблиця 3.2 – Порівняльний аналіз Java-фреймворків

Критерій	Spring Boot	Jakarta EE	Play Framework
Швидкість розробки	Дуже висока	Низька	Середня
Конфігурація	Мінімальна (анотації)	Складна (XML)	Середня
Робота з БД	Spring Data (дуже зручно)	JPA / EJB	Ebean / JPA
Вбудований сервер	Так (Tomcat/Jetty)	Ні (потрібен зовнішній)	Так (Netty)
Популярність (Community)	Дуже висока	Спадаюча	Низька

Для розробки обрано Spring Boot. Це дозволить зосередитися на бізнес-логіці (алгоритмі підбору), а не на налаштуванні серверів. Використання Spring Data JPA та Spring Security є критичним для реалізації функціоналу системи в стислі терміни.

3.3 Вибір системи управління базами даних (СКБД)

Розробка інформаційної системи рекрутингу передбачає роботу зі значними масивами структурованої інформації: профілями користувачів, описами вакансій, довідниками навичок та історією взаємодії. Ефективність роботи програмного забезпечення, швидкість пошуку кандидатів та цілісність даних безпосередньо залежать від правильно обраної Системи Управління Базами Даних (СКБД) та спроєктованої схеми даних.

На етапі проєктування архітектури необхідно визначитись із класом бази даних. На сьогоднішній день домінують два основні підходи: реляційний (SQL) та нереляційний (NoSQL).

Нереляційні бази даних (NoSQL) зберігають дані у вигляді документів (JSON) або пар ключ-значення. Вони забезпечують високу швидкість запису та гнучкість схеми (schema-less), що дозволяє змінювати структуру профілю кандидата "на льоту". Недоліком для проєкту є відсутність жорстких зв'язків між сутностями та слабка підтримка складних транзакцій. У системі рекрутингу критично важливим є зв'язок «Кандидат – Навички – Вакансії». Реалізація алгоритму метчингу (співставлення) на NoSQL вимагала б складних обчислень на стороні серверу додатку, оскільки NoSQL бази погано справляються з операціями з'єднання (JOIN).

Реляційні бази даних (SQL). Дані зберігаються у вигляді таблиць із чітко визначеними типами полів та зв'язками між ними. Забезпечують цілісність даних (ACID) та потужну мову запитів SQL. Представники: PostgreSQL, MySQL, Oracle.

Перевагою для проєкту є те , що реляційна модель ідеально описує предметну область, де сутності чітко структуровані. Використання зовнішніх ключів (Foreign Keys) гарантує, що у системі не з'явиться "навички", яка не прив'язана до жодного кандидата, або "вакансії" без опису вимог. Тож для реалізації дипломного проєкту обрано реляційну модель, оскільки пріоритетом є цілісність даних та можливість виконання складних аналітичних запитів для фільтрації кандидатів.

MySQL – найпопулярніша у світі Open Source база даних.

До переваг відносяться простота встановлення, висока швидкість читання простих даних, величезна спільнота.

Недоліками ж є обмежена підтримка складних типів запитів (наприклад, Common Table Expressions у старих версіях), менш сувора перевірка стандартів SQL. У складних сценаріях, коли потрібно робити

вибірку за багатьма критеріями (фільтр кандидатів), оптимізатор запитів MySQL може працювати менш ефективно, ніж у конкурентів.

Microsoft SQL Server – потужна комерційна СКБД, відмінна інтеграція з продуктами Microsoft, потужні інструменти адміністрування.

Орієнтована на платформу Windows, є платною для комерційного використання (окрім Express версії), що не відповідає концепції розробки доступного ПЗ.

PostgreSQL – об'єктно-реляційна СКБД з відкритим кодом. Вважається "найбільш просунутою" серед безкоштовних баз даних. Підтримує складні типи даних (масиви, JSONB), повнотекстовий пошук, геометричні дані. Має дуже надійну систему транзакцій та багатоверсійність (MVCC), що дозволяє багатьом HR-менеджерам працювати з базою одночасно без блокувань [16].

Порівняльна характеристика наведена у таблиці 3.3.

Таблиця 3.3 – Порівняльний аналіз Java-фреймворків

Критерій	MySQL	PostgreSQL	MS SQL Server
Ліцензія	GPL (Безкоштовна)	BSD (Безкоштовна)	Комерційна
Відповідність SQL	Часткова	Повна (ANSI SQL)	Повна
Складні JOIN запити	Добре	Відмінно	Відмінно
Типи даних	Стандартні	Розширені (JSON, Array, UUID)	Стандартні + XML
Контроль цілісності	Залежить від двигуна	Дуже високий	Дуже високий

На основі проведеного аналізу для реалізації системи обрано PostgreSQL. Окрім загальних переваг, вибір зумовлений специфічними потребами системи підбору персоналу:

– робота зі зв'язками «Багато-до-багатьох». У системі реалізовано зв'язок між кандидатами та навичками через проміжну таблицю candidate_skills з додатковим полем proficiency_level. PostgreSQL надзвичайно ефективно обробляє такі зв'язки при побудові рейтингів, використовуючи оптимізовані алгоритми з'єднання (Hash Join, Merge Join);

– обмеження цілісності (CHECK Constraints). PostgreSQL дозволяє задавати складні правила валідації на рівні таблиць. Наприклад, ми використовуємо обмеження CHECK (proficiency_level >= 1 AND proficiency_level <= 5), що гарантує, що в базу не потраплять некоректні дані про рівень навичок, навіть якщо помилка станеться у програмному коді;

– масштабованість. Архітектура PostgreSQL дозволяє легко додавати нові функції та розширювати базу без зупинки роботи;

– сумісність з Java, драйвер JDBC для PostgreSQL та діалект Hibernate є високооптимізованими, що забезпечує швидку взаємодію між сервером додатку (Spring Boot) та сховищем даних.

При розробці структури бази даних було дотримано принципів нормалізації до третьої нормальної форми (3NF) для уникнення надлишковості даних та аномалій при оновленні [17].

Основні рішення при проєктуванні:

– використання сурогатних ключів. Усі таблиці використовують поле id (тип BIGINT з автоінкрементом) як первинний ключ (PRIMARY KEY), що забезпечує швидкість індексації та стабільність зв'язків;

– каскадні операції. Для забезпечення посилальної цілісності використано механізм зовнішніх ключів (FOREIGN KEY) з правилами ON DELETE CASCADE. Наприклад, при видаленні користувача з таблиці users, його профіль у таблиці candidates видаляється автоматично. Це запобігає появі "сміттєвих" даних;

– індексація. Для прискорення пошуку за ключовими полями (email, username) створено унікальні індекси.

Такий підхід до проєктування бази даних створює надійний фундамент для функціонування всієї інформаційної системи.

3.4 Засоби розробки інтерфейсу та середовище (IDE)

Ефективність процесу створення програмного забезпечення та зручність кінцевого продукту залежать від правильно підібраного комплексу інструментальних засобів. Для реалізації системи обрано стек, який забезпечує швидку розробку (Rapid Application Development), легку підтримку коду та стабільність розгортання.

3.4.1 Технологія побудови інтерфейсу користувача (Frontend)

Враховуючи архітектурне рішення будувати систему як єдиний монолітний додаток (Monolithic Architecture) на базі Spring Boot, для реалізації рівня представлення (View) обрано шаблонний рушій Thymeleaf.

Thymeleaf – це сучасний серверний рушій шаблонів Java для веб-середовища та автономних середовищ. Його головною особливістю є концепція "Natural Templates" – шаблони Thymeleaf виглядають як звичайний HTML, що дозволяє коректно відображати їх у браузері навіть без запущеного сервера (як статичні прототипи) [18].

Переваги використання у проєкті:

– серверний рендеринг (SSR). Генерація HTML-сторінок відбувається на стороні сервера. Це спрощує архітектуру, оскільки відпадає необхідність створювати окремий API та клієнтський додаток (як у випадку з SPA). Дані передаються з контролера безпосередньо у вигляд;

– глибока інтеграція зі Spring Boot. Thymeleaf автоматично налаштовується через стартери Spring. Він підтримує Spring Security "з коробки", що дозволяє легко приховувати або показувати елементи інтерфейсу (кнопки, меню) залежно від ролі користувача (HR або Кандидат);

– фрагментація інтерфейсу. Технологія дозволяє виносити повторювані частини коду (шапка сайту, бічне меню, футер) в окремі файли-фрагменти (fragments) і підключати їх однією строкою коду. Це значно спрощує підтримку та зміну дизайну.

3.4.2 Інтегроване середовище розробки (IDE)

Для написання, налагодження та рефакторингу програмного коду використовується IntelliJ IDEA Community Edition від компанії JetBrains. Це середовище є стандартом де-факто для сучасної Java-розробки.

IDE розуміє контекст Spring Framework. Вона автоматично підказує імена бінів, властивості у файлі application.properties та виявляє помилки конфігурації ще до запуску програми.

Вбудований інструмент Database Tools дозволяє підключатися до PostgreSQL безпосередньо з інтерфейсу середовища, виконувати SQL-запити, переглядати структуру таблиць та дані, що критично важливо для тестування алгоритму підбору кандидатів.

Середовище дозволяє безпечно перейменовувати класи, методи та змінні по всьому проєкту, автоматично оновлюючи всі посилання на них.

3.4.3 Система автоматизації збірки та управління залежностями

Для управління життєвим циклом проєкту та бібліотеками обрано Apache Maven. Це інструмент, який використовує декларативний підхід до конфігурації проєкту [19].

Управління залежностями (Dependency Management) – всі зовнішні бібліотеки (Spring Boot Starter Web, Spring Data JPA, PostgreSQL Driver, Lombok) описуються у файлі конфігурації pom.xml. Maven автоматично завантажує необхідні JAR-файли з центрального репозиторію, слідкує за їх версіями та вирішує конфлікти між ними.

Maven диктує чітку структуру тек (src/main/java, src/main/resources, src/test), що робить проєкт зрозумілим для будь-якого Java-розробника та спрощує навігацію. Інструмент надає стандартизовані команди для різних етапів:

– очищення проєкту від попередніх збірок (clean);
– компіляція вихідного коду (compile);
– запуск модульних тестів (test);

збірка готового до розгортання файлу (package).

Використання Maven гарантує відтворюваність збірки (reproducibility), тож проєкт буде однаково компілюватися та запускатися як на локальному комп'ютері розробника, так і на будь-якому іншому сервері.

3.5 Проєктування функціональної моделі системи

Функціональне моделювання призначене для визначення меж системи та специфікації її поведінки з точки зору зовнішніх користувачів. Для візуалізації функціональних вимог використано діаграму варіантів використання (Use Case Diagram) мови UML [20].

3.5.1 Ідентифікація акторів та їх ролей

На основі аналізу бізнес-процесів виділено два типи користувачів (акторів), які взаємодіють із системою:

HR-менеджер (HR Manager) – це основний користувач системи, відповідальний за процеси найму. Його права доступу це повний доступ до модулів управління вакансіями, базою кандидатів та командами. А ключовими задачами є створення вакансій з вагами навичок, запуск алгоритму підбору, формування складу проєктних команд.

Кандидат (Candidate) – користувач, який шукає роботу, його ключовими задачами є реєстрація, заповнення профілю (персональні дані, навички), моніторинг статусу та можливість відгуку на вакансію.

Права доступу: обмежений доступ (тільки власний профіль та перегляд відкритих вакансій).

3.5.2 Перелік функціональних вимог

В результаті дослідження предметної області та аналізу вимог до розроблюваної системи було сформовано наступну систему бізнес-правил, які визначають логіку функціонування програмного забезпечення [21]:

– кожен користувач системи унікально ідентифікується логіном (username) та захищається паролем. Користувачеві присвоюється одна з двох ролей: «HR-менеджер» або «Кандидат», що визначає його рівень доступу до даних;

– сутність «Кандидат» характеризується персональними даними (ім’я, прізвище, вік, місто), контактною інформацією (email, телефон), відомостями про освіту та досвід роботи;

– професійна придатність кандидата визначається набором технічних навичок. Кожна навичка у профілі кандидата обов'язково має оцінку рівня володіння (Grade) за шкалою від 1 до 5 (де 1 — початковий рівень, 5 — експерт);

– кожна вакансія характеризується назвою, текстовим описом проєкту та містом. Вакансія містить динамічний список вимог до навичок;

– кожна вимога вакансії складається з посилання на навичку, мінімально необхідного рівня володіння нею та вагового коефіцієнту (Weight), який визначає пріоритетність цієї навички для розрахунку рейтингу;

– система автоматично розраховує рейтинговий бал кандидата для конкретної вакансії. Рейтинг є динамічною величиною і залежить від збігу навичок кандидата з вимогами вакансії з урахуванням їх ваги;

– кожна команда має унікальну назву та опис проєкту. Кандидат може бути закріплений лише за однією командою одночасно (статус «EMPLOYED») або перебувати у стані пошуку (статус «SEARCHING»);

– користувач із роллю «Кандидат» має право на створення власного профілю, внесення та редагування інформації про свої навички, а також перегляд списку активних вакансій;

– HR-менеджер має повноваження створювати нові вакансії, визначати для них список вимог, встановлювати ваги навичок та редагувати або видаляти існуючі позиції;

– HR-менеджер має право ініціювати алгоритмічний пошук кандидатів та переглядати ранжований список результатів, відсортований за спаданням рейтингу відповідності;

– HR-менеджер має виключне право створювати нові команди, додавати до них кандидатів, які успішно пройшли відбір, та виключати їх зі складу команди (звільняти).

3.5.3 Діаграма варіантів використання

Графічне представлення функціональних вимог та зв'язків між акторами й прецедентами наведено на рисунку 3.1 [22].

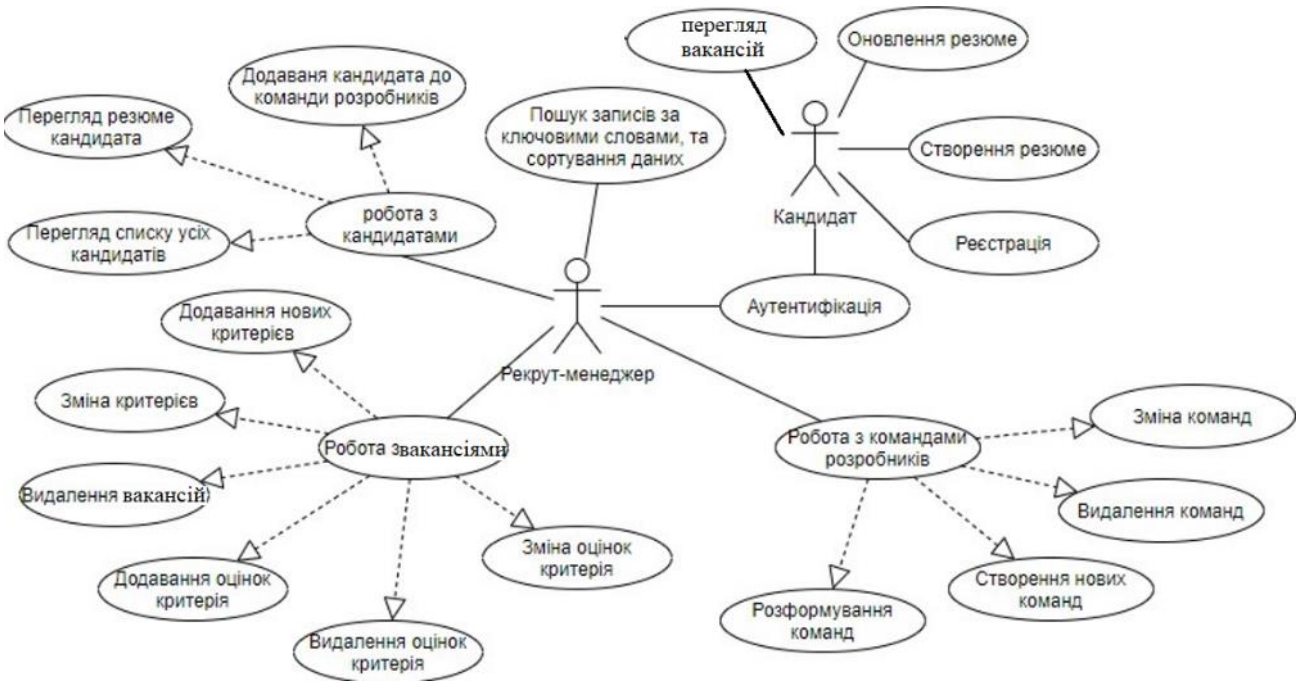


Рисунок 3.1 – Діаграма варіантів використання системи (Use Case Diagram)

3.5.4 Діаграма станів вакансії (Job Position State Diagram)

Для моделювання взаємодії об'єктів у часі та опису життєвого циклу ключових сутностей розроблено діаграму динамічної поведінки.

Діяльність HR-менеджера зосереджена навколо управління вакансіями. Вакансія проходить певні етапи: від створення чернетки до повного укомплектування команди. Діаграма станів сутності «Job Position» наведена на рис. 3.2 [23].

DRAFT (Чернетка), рекрутер створює опис, додає вимоги та налаштовує ваги навичок. У цьому стані вакансія невидима для пошукових алгоритмів.

OPEN (Відкрита), вакансія опублікована і система автоматично відстежує нових кандидатів, що підходять під критерії.

PROCESSING (Активний відбір), рекрутер запустив алгоритм підбору та проводить співбесіди з відібраними кандидатами.

FILLED (Укомплектована), необхідна кількість спеціалістів знайдена, команда сформована.

CANCELLED (Скасована), робота над вакансією припинена (наприклад, проєкт закрито).

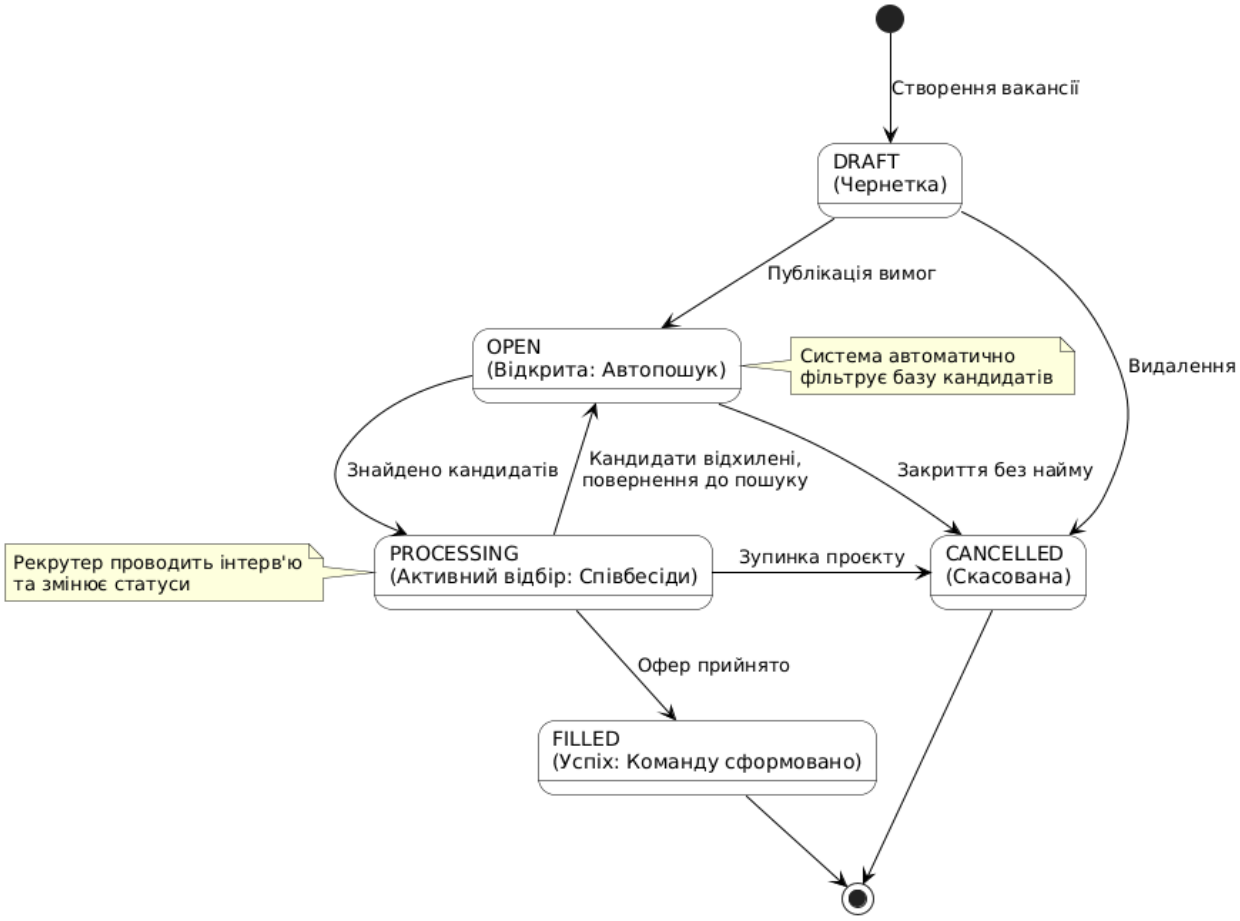


Рисунок 3.2 – Діаграма станів вакансії

3.6 Висновки за розділом 3

У третьому розділі виконано етап системного проєктування програмного комплексу для автоматизації рекрутингу.

Обґрунтовано технологічний стек. Для реалізації системи обрано мову програмування Java та фреймворк Spring Boot, що забезпечує високу продуктивність та надійність серверної частини. В якості сховища даних визначено реляційну СКБД PostgreSQL, яка підтримує необхідну цілісність зв'язків. Для розробки інтерфейсу обрано технологію Thymeleaf.

Формалізовано вимоги. Визначено функціональні межі системи та ролі користувачів (HR-менеджер, Кандидат). Розроблено діаграму варіантів використання (Use Case), яка деталізує можливості кожного актора.

Спроєктовано динаміку системи. Розроблено діаграми станів для сутностей «Кандидат» та «Вакансія», що дозволяє програмно контролювати життєвий цикл найму та уникати логічних помилок у бізнес-процесах.

Прийняті проєктні рішення створюють необхідну базу для переходу до етапу програмної реалізації, опису структур даних та написання коду.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Реалізація структури бази даних

Основою інформаційної системи є база даних під управлінням PostgreSQL. На етапі реалізації було створено фізичну модель даних, яка відповідає 3-й нормальній формі [24]. Структура зв'язків між таблицями зображена на ER-діаграмі (рис. 4.1).

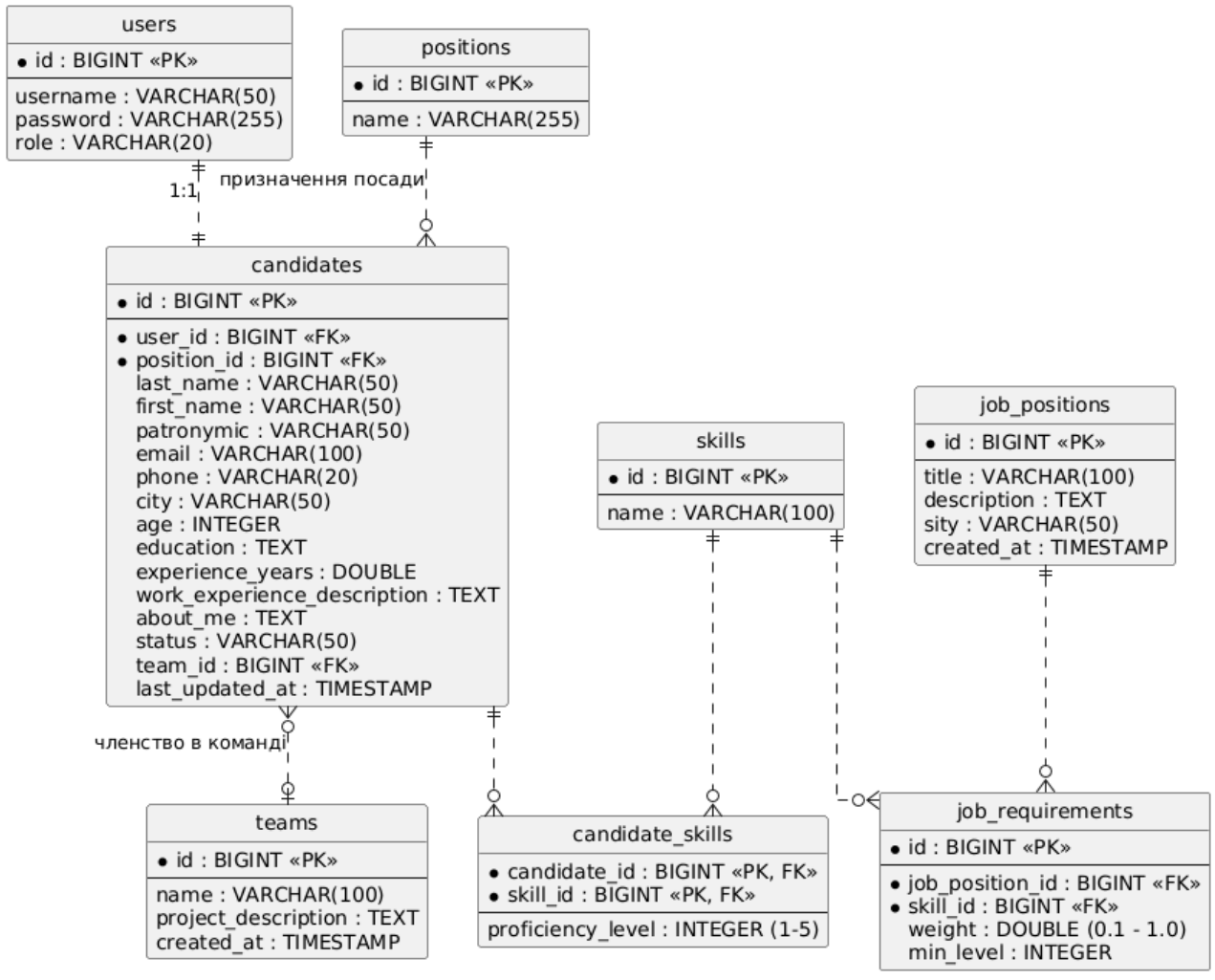


Рисунок 4.1 – Фізична модель бази даних (ER-діаграма)

Таблиця «users» призначена для зберігання облікових даних користувачів системи (як рекрутерів, так і кандидатів). Кожний запис таблиці складається з наступних полів, опис яких наведений в таблиці 4.1.

Таблиця 4.1 – Опис структури таблиці «users»

Ключ	Ім’я поля	Тип даних	Розмір поля	Опис
PK	id	Чисельний	BigInt	AI Унікальний ідентифікатор
	username	Текстовий	50	Логін користувача
	password	Текстовий	255	Хеш паролю
	role	Текстовий	20	Роль (ROLE_HR, ROLE_CANDIDATE)

Таблиця «positions» є довідником посад (професій), які можуть обіймати кандидати. Кожний запис таблиці складається з наступних полів, опис яких наведений в таблиці 4.2.

Таблиця 4.2 – Опис структури таблиці «positions»

Ключ	Ім’я поля	Тип даних	Розмір поля	Опис
PK	id	Чисельний	BigInt	AI Ідентифікатор посади
	name	Текстовий	255	Назва посади (напр. Java Developer)

Таблиця «candidates» призначена для зберігання розширеної особистої та професійної інформації про шукача роботи. Кожний запис таблиці складається з наступних полів, опис яких наведений в таблиці 4.3.

Таблиця 4.3 – Опис структури таблиці «candidates»

Ключ	Ім’я поля	Тип даних	Розмір поля	Опис
PK	id	Чисельний	BigInt	AI Ідентифікатор анкети
FK	user_id	Чисельний	BigInt	Посилання на обліковий запис
FK	position_id	Чисельний	BigInt	Бажана посада

Кінець таблиці 4.3

Ключ	Ім’я поля	Тип даних	Розмір поля	Опис
	last_name	Текстовий	50	Прізвище
	first_name	Текстовий	50	Ім'я
	patronymic	Текстовий	50	По батькові
	email	Текстовий	100	Електронна пошта
	phone	Текстовий	20	Телефон
	city	Текстовий	50	Місто проживання
	age	Чисельний	Integer	Вік
	education	Текстовий	Text	Інформація про освіту
	experience_years	Чисельний	Double	Досвід роботи (років)
	work_experience_descr	Текстовий	Text	Опис попереднього досвіду
	about_me	Текстовий	Text	Додаткова інформація
	status	Текстовий	50	Статус (SEARCHING, HIRED)
FK	team_id	Чисельний	BigInt	Поточна команда (якщо найнятий)
	last_updated_at	Дата/Час	Timestamp	Час останнього оновлення

Таблиця «skills» є довідником технічних навичок. Кожний запис таблиці складається з наступних полів, опис яких наведений в таблиці 4.4.

Таблиця 4.4 – Опис структури таблиці «skills»

Ключ	Ім’я поля	Тип даних	Розмір поля	Опис
PK	id	Чисельний	BigInt	AI Ідентифікатор навички
	name	Текстовий	100	Назва технології (Java, SQL)

Таблиця «candidate_skills» є проміжною таблицею для реалізації зв'язку «багато-до-багатьох» та зберігання рівня володіння навичкою. Кожний запис таблиці складається з наступних полів, опис яких наведений в таблиці 4.5.

Таблиця 4.5 – Опис структури таблиці «candidate_skills»

Ключ	Ім’я поля	Тип даних	Розмір поля	Опис
PK, FK	candidate_id	Чисельний	BigInt	Ідентифікатор кандидата
PK, FK	skill_id	Чисельний	BigInt	Ідентифікатор навички
	proficiency _level	Чисельний	Integer	Рівень володіння (1-5)

Таблиця «job_positions» призначена для зберігання вакансій, створених рекрутером. Кожний запис таблиці складається з наступних полів, опис яких наведений в таблиці 4.6.

Таблиця 4.6 – Опис структури таблиці «job_positions»

Ключ	Ім’я поля	Тип даних	Розмір поля	Опис
PK	id	Чисельний	BigInt	AI Номер вакансії
	title	Текстовий	100	Заголовок вакансії
	description	Текстовий	Text	Опис вимог та умов
	sity	Текстовий	50	Локація (Місто/Remote)
	created_at	Дата/Час	Timestamp	Дата створення

Таблиця «job_requirements» визначає вимоги до конкретної вакансії, включаючи вагу навички для алгоритму підбору. Кожний запис таблиці складається з наступних полів, опис яких наведений в таблиці 4.7.

Таблиця 4.7 – Опис структури таблиці «job_requirements»

Ключ	Ім’я поля	Тип даних	Розмір поля	Опис
PK	id	Чисельний	BigInt	AI Ідентифікатор вимоги
FK	job_position_id	Чисельний	BigInt	Посилання на вакансію
FK	skill_id	Чисельний	BigInt	Посилання на навичку
	weight	Чисельний	Double	Ваговий коефіцієнт (0.1-1.0)
	min_level	Чисельний	Integer	Мінімальний необхідний рівень

Таблиця «teams» призначена для зберігання інформації про сформовані команди розробників. Кожний запис таблиці складається з наступних полів, опис яких наведений в таблиці 4.8.

Таблиця 4.8 – Опис структури таблиці «teams»

Ключ	Ім’я поля	Тип даних	Розмір поля	Опис
PK	id	Чисельний	BigInt	AI Номер команди
	name	Текстовий	100	Назва команди
	project_description	Текстовий	Text	Опис проєкту команди
	created_at	Дата/Час	Timestamp	Дата створення

4.2 Об'єктно-орієнтована модель та архітектура класів

Програмна реалізація системи виконана з використанням об'єктно-орієнтованого підходу. Для взаємодії з реляційною базою даних використано технологію ORM (Object-Relational Mapping) на базі специфікації JPA

(Hibernate) [25]. Це дозволяє представити таблиці бази даних у вигляді класів мови Java (Entity), а зв'язки між таблицями — у вигляді полів об'єктів.

Архітектура класів сутностей (Domain Model) зображена на діаграмі класів (рис. 4.2).

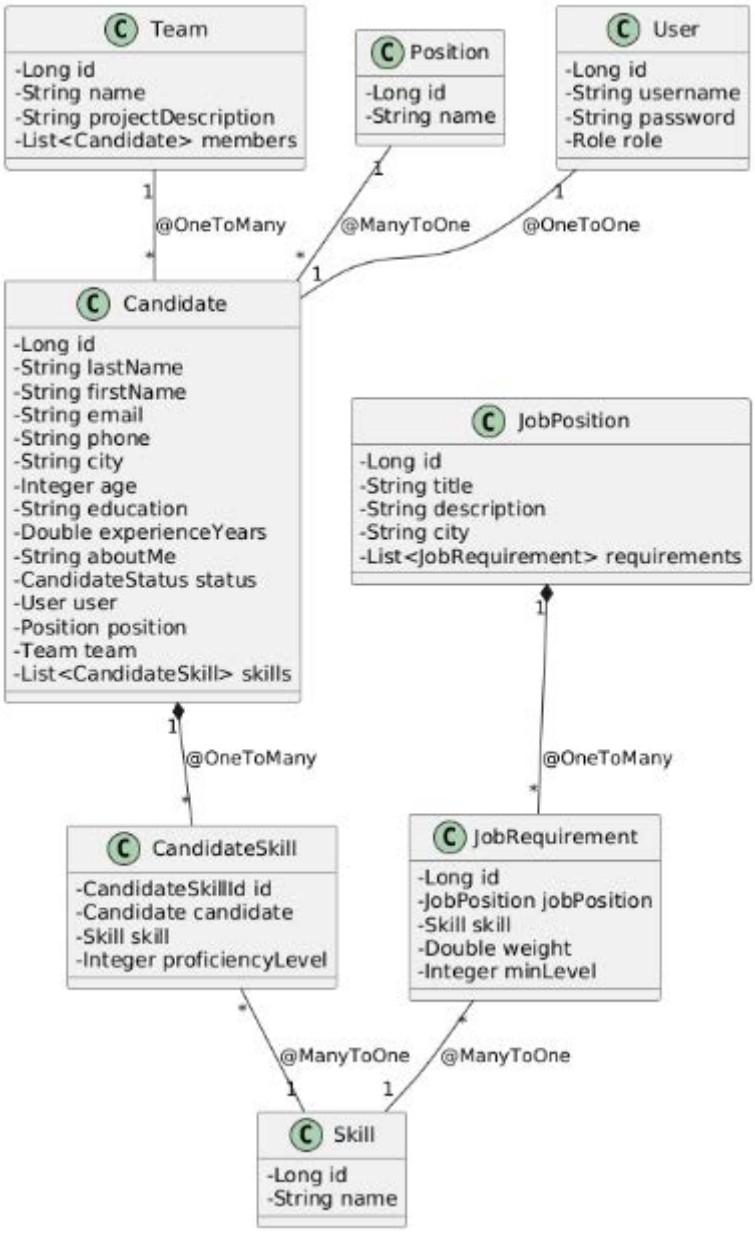


Рисунок 4.2 – Діаграма класів сутностей (Entity Class Diagram)

Основою додатку є шар доступу до даних, який відображає структуру бази даних PostgreSQL, описану в пункті 4.1.

Пакет entity містить Java-класи, анотовані як @Entity. Вони представляють таблиці бази даних.

Пакет repository забезпечує абстракцію над базою даних [26]. Інтерфейси (наприклад, CandidateRepository, JobRequirementRepository) розширюють JpaRepository, що дозволяє виконувати CRUD-операції та складні пошукові запити без написання SQL-коду вручну (рис. 4.3).

Рисунок 4.3 – Структура шару даних (Entity та Repository)

Обробка HTTP-запитів від клієнта здійснюється контролерами, розташованими в пакеті controller. Архітектура контролерів чітко розділена за функціональним призначенням.

AuthController відповідає за реєстрацію та авторизацію користувачів.

Група HrController (HrController, HrJobController, HrMatchController, HrTeamController) реалізує функціонал для рекрутера – створення вакансій, запуск пошуку, управління командами.

Група CandidateController забезпечує функціонал для шукача роботи, а саме редагування профілю та навичок, взаємодія з вакансіями.

Такий поділ дозволяє легко налаштовувати права доступу для різних URL-адрес (рис. 4.4).

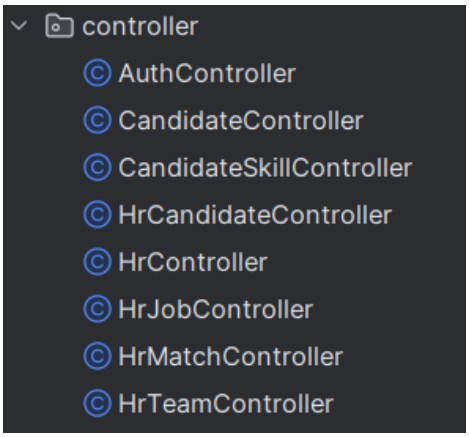


Рисунок 4.4 – Структура веб-контролерів

Вся бізнес-логіка додатку інкапсульована у сервісному шарі. Це запобігає дублюванню коду в контролерах.

Пакет service містить клас MatchingService. Саме в цьому класі реалізовано математичний алгоритм зваженого пошуку (розроблений у Розділі 2), який розраховує рейтинг відповідності кандидата вимогам вакансії.

Пакет dto (Data Transfer Object) містить клас CandidateMatchDTO. Він використовується для передачі даних про кандидата разом із розрахованим рейтингом (score) на веб-інтерфейс, відокремлюючи внутрішню модель бази даних від представлення користувачеві (рис. 4.5).

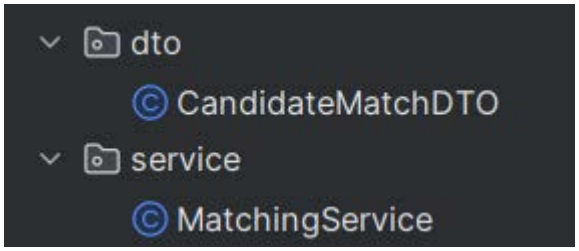


Рисунок 4.5 – Структура шару бізнес-логіки

Для захисту інформаційної системи реалізовано модуль безпеки на базі Spring Security. SecurityConfig визначає правила доступу (хто має доступ до

/hr/**, а хто до /candidate/**), налаштовує форму входу та шифрування паролів (BCrypt) [27].

CustomUserDetailsService реалізує завантаження користувачів з нашої таблиці users для механізму аутентифікації Spring.

Файл application.properties містить налаштування підключення до бази даних (URL, логін, пароль) та конфігурацію діалекту Hibernate (рис. 4.6).

Рисунок 4.6 – Компоненти конфігурації та безпеки

Інтерфейс користувача реалізовано за допомогою технології серверного рендерингу Thymeleaf. HTML-шаблони розташовані в каталозі templates і структуровані за модулями. Це забезпечує чітке розділення інтерфейсу для різних ролей користувачів (рис. 4.7) [28].

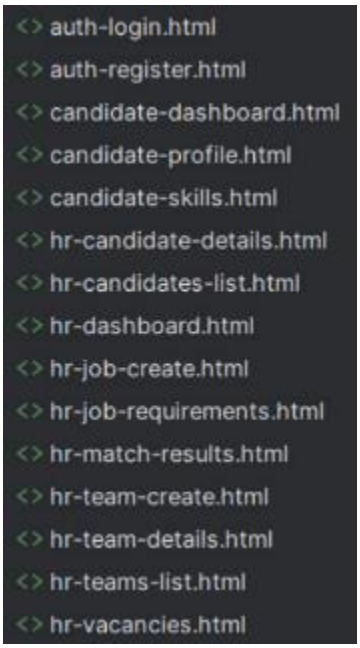


Рисунок 4.7 – Структура шаблонів інтерфейсу користувача

4.3 Програмна реалізація алгоритмів роботи

Ключовим компонентом розробленої системи є модуль інтелектуального підбору (Matching Module). Його реалізація базується на взаємодії компонентів сервісного шару та виконанні алгоритму зваженого оцінювання, математична модель якого обґрунтована в другому розділі.

Процес пошуку ініціюється HTTP-запитом від клієнта. Реалізація цього сценарію побудована на взаємодії контролера HrMatchController та сервісу MatchingService. Динаміка передачі повідомлень між об'єктами зображена на діаграмі послідовності (рис. 4.8).

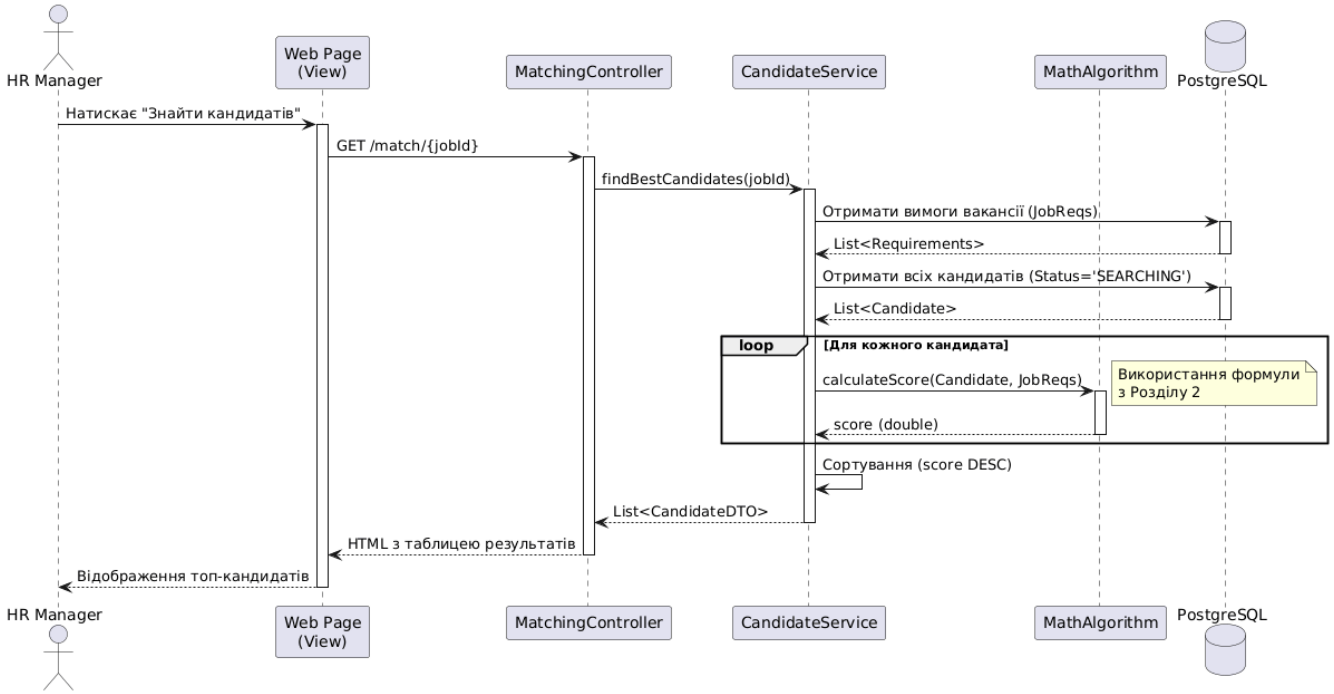


Рисунок 4.8 – Діаграма послідовності процесу підбору кандидатів

Замість прямого використання програмного коду, логіку роботи методу calculateScore() доцільно представити у вигляді блок-схеми алгоритму. Це дозволяє абстрагуватися від синтаксису мови Java та зосередитися на бізнес-правилах.

Алгоритм виконує порівняння вектора навичок кандидата з вектором вимог вакансії. Блок-схема алгоритму наведена на рисунку 4.9.

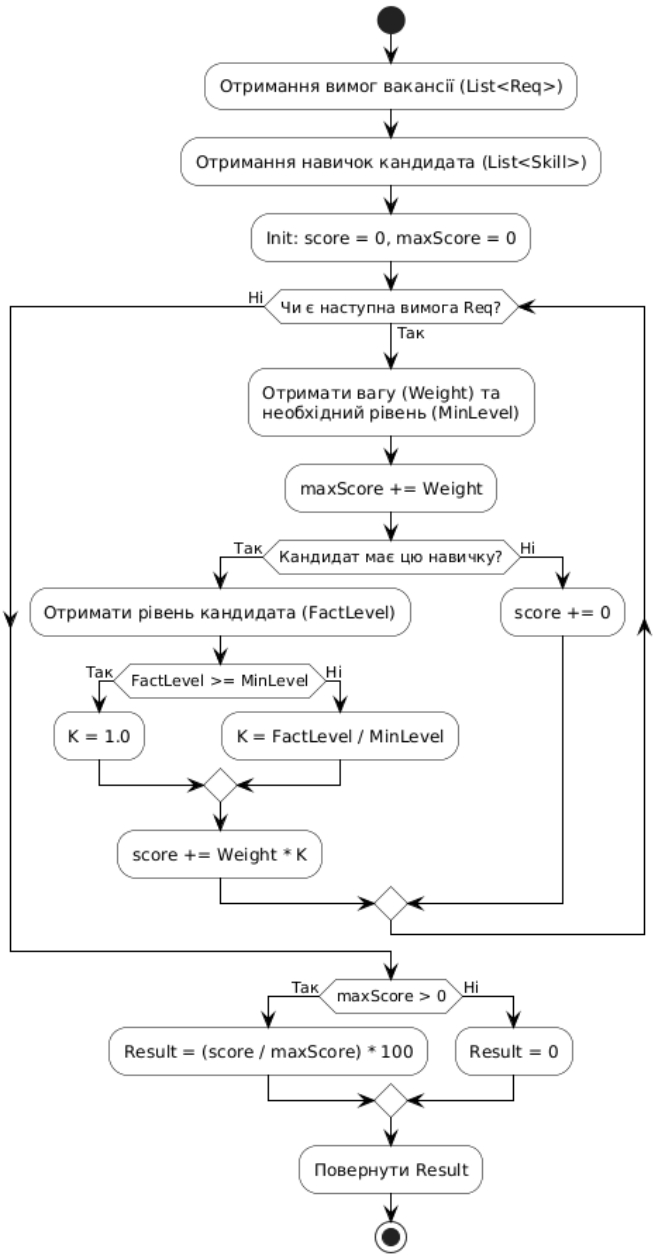


Рисунок 4.9 – Блок-схема алгоритму розрахунку рейтингу відповідності

4.4 Програмно-апаратна архітектура

Для візуалізації фізичного розміщення програмних компонентів на апаратних вузлах розроблено діаграму розгортання (Deployment Diagram) [29]. Система побудована за триланковою архітектурою «Клієнт-Сервер».

Схема розгортання зображена на рисунку 4.10.

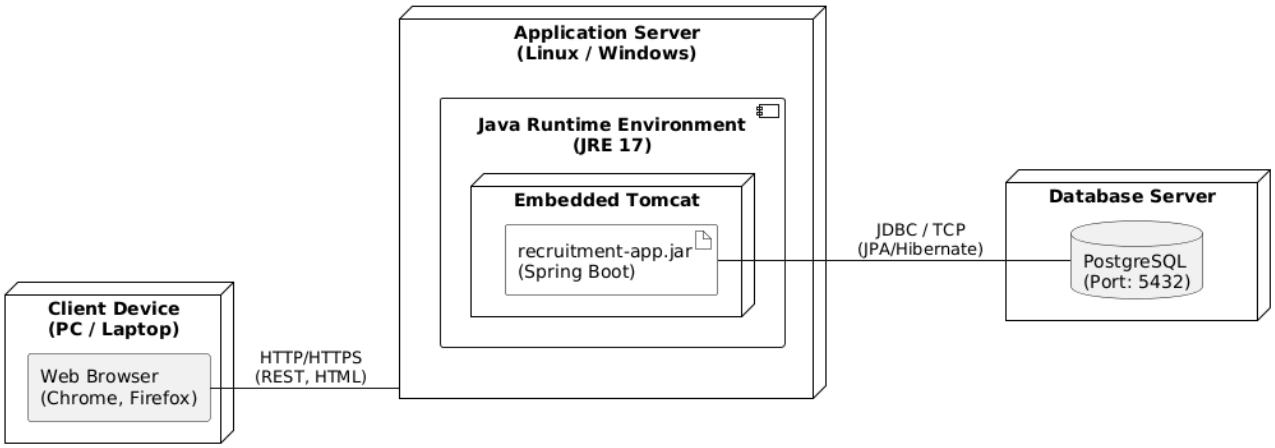


Рисунок 4.10 – Діаграма розгортання

Система побудована за класичною триланковою архітектурою «Клієнт-Сервер». Клієнтська частина представлена веб-браузером на пристрої користувача, який взаємодіє з сервером додатку за протоколом HTTP/HTTPS. Серверна логіка виконується в середовищі Java Runtime Environment (JRE) під управлінням вбудованого контейнера сервлетів Apache Tomcat, де розгорнуто скомпільований артефакт системи (recruitment-system.jar) [30]. Третім рівнем архітектури є сервер бази даних під управлінням СКБД PostgreSQL, взаємодія з яким здійснюється через протокол JDBC по стандартному порту.

4.4 Висновки за розділом 4

У четвертому розділі виконано програмну реалізацію інформаційної системи автоматизації рекрутингу.

Реалізовано структуру бази даних. Створено фізичну модель даних у СКБД PostgreSQL, яка складається з 8 таблиць і забезпечує зберігання профілів, вакансій та навичок із дотриманням цілісності даних.

Розроблено структуру проєкту на базі Spring Boot з чітким розділенням на шари (Entity, Repository, Service, Controller), що спрощує підтримку коду.

Програмно реалізовано математичну модель зваженого оцінювання кандидатів у класі MatchingService, що дозволяє автоматично ранжувати резюме за релевантністю.

Створено зручний веб-інтерфейс користувача на базі Thymeleaf, який забезпечує виконання всіх функціональних вимог для ролей HR-менеджера та Кандидата.

Система повністю готова до подальшого етапу тестування.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЗАСТОСУНКУ

5.1 Опис програми

Розроблений програмний продукт призначений для комплексної автоматизації та інтелектуалізації процесів пошуку, відбору та найму IT-фахівців. Його функціональні можливості спрямовані на мінімізацію рутинної роботи рекрутерів, підвищення об'єктивності оцінювання кандидатів та прискорення закриття вакансій завдяки використанню алгоритмічних методів ранжування.

Система реалізує повний цикл рекрутингу від створення вакансії з детальними технічними вимогами до формування фінальних проєктних команд. Ключовою особливістю програмного засобу є впровадження модуля автоматизованого підбору (Matching), який, на відміну від класичного пошуку за ключовими словами, виконує математичний розрахунок рейтингу відповідності кандидата на основі ваги навичок та їх рівня володіння. Це дозволяє нівелювати суб'єктивний фактор при первинному скринінгу резюме.

У базі даних системи організовано централізоване зберігання структурованої інформації про професійні компетенції кандидатів (Hard Skills). Для кожного претендента зберігається історія його навичок з грейдами (оцінками від 1 до 5), досвід роботи, контактні дані та поточний статус зайнятості. Це дає змогу формувати динамічний кадровий резерв та швидко знаходити спеціалістів під нові проєкти без необхідності повторного розміщення оголошень на зовнішніх ресурсах.

Окремий функціональний блок системи забезпечує рольову взаємодію користувачів. Реалізовано особисті кабінети для двох типів користувачів:

– HR-менеджер отримує інструменти для управління вакансіями, перегляду автоматично згенерованих рейтингів та адміністрування складу команд;

– кандидат має можливість створювати цифрове резюме, самостійно актуалізувати навички та відстежувати свій статус у системі.

Такий підхід забезпечує прозорість процесу найму та дозволяє компанії приймати обґрунтовані управлінські рішення при формуванні штату розробників.

5.2 Системні вимоги та інструкція з розгортання

Для забезпечення стабільної роботи програмного комплексу та коректного виконання всіх функцій визначено мінімальні та рекомендовані системні вимоги. Оскільки система реалізована як веб-застосунок з клієнт-серверною архітектурою, вимоги поділяються на дві групи: до сторони клієнта та до сторони сервера.

Вимоги до клієнтського робочого місця (Client Side) Клієнтська частина системи не вимагає встановлення спеціалізованого програмного забезпечення та працює через веб-браузер.

Апаратні вимоги:

– процесор із тактовою частотою від 1.6 ГГц (рівня Intel Core i3 / AMD Ryzen 3 або аналогічні мобільні процесори);

– оперативна пам'ять (RAM) не менше 4 Гб (для комфортної роботи браузера);

– дисплей, рекомендована роздільна здатність екрану 1366x768 пікселів або вище.

Програмні вимоги:

– операційна система Windows 10/11, macOS, Linux, Android або iOS;

– веб-браузер Google Chrome (версії 90+), Mozilla Firefox, Microsoft Edge або Safari з підтримкою HTML5, CSS3 та JavaScript (ES6+).

Вимоги до серверного середовища (Server Side) Серверна частина відповідає за обробку бізнес-логіки, роботу з базою даних та хостинг веб-інтерфейсу.

Апаратні вимоги:

– багатоядерний процесор (мінімум 2 ядра) з частотою від 2.0 ГГц;

– оперативна пам'ять (RAM) мінімум 4 Гб (рекомендовано 8 Гб для забезпечення швидкодії JVM та PostgreSQL);

– накопичувач SSD з вільним простором не менше 10 Гб (для файлів додатку, СКБД та логів);

– стабільне підключення до Інтернету або локальної мережі зі швидкістю від 10 Мбіт/с.

Програмні вимоги:

– операційна система Linux (Ubuntu 20.04+, CentOS 8+) або Windows Server 2019+;

– середовище виконання Java JRE (Java Runtime Environment) версії 17 або вище;

– система керування базами даних:PostgreSQL версії 14 або вище.

Процес розгортання системи на сервері складається з наступних етапів:

– встановити СКБД PostgreSQL, якщо вона відсутня на сервері;

– створити нову базу даних (наприклад, з ім'ям recruitment_db) та користувача з повними правами доступу до неї;

– ініціалізувати структуру бази даних. Для цього необхідно виконати SQL-скрипт з дамп-файлу recruitment_db_backup.sql або дозволити Hibernate автоматично створити таблиці при першому запуску (параметр ddl-auto=update);

– налаштування конфігурації Перед запуску перевірити файл налаштувань application.properties, який знаходиться в одній директорії з виконуваним файлом або в папці config/;

– запуск додатку Система постачається у вигляді виконуваного JAR-архіву. Для запуску необхідно відкрити термінал (консоль) та виконати команду java -jar recruitment-system.jar [31];

– експлуатація Після запуску сервера доступ до системи здійснюється через веб-браузер за адресою: http://<IP-адреса-сервера>:<порт>/.

5.3 Сценарій експлуатації програмного забезпечення користувачем

Експлуатація системи здійснюється через веб-інтерфейс і передбачає виконання послідовності дій відповідно до бізнес-процесів найму. Взаємодія з програмним комплексом розділена на рольові сценарії.

5.3.1 Сценарій роботи Кандидата

Сценарій для пошукача роботи орієнтований на створення максимально інформативного профілю для алгоритмів системи.

Кандидат повинен авторизуватись у системі, а у разі якщо це новий користувач заповнює реєстраційну форму, вказуючи персональні дані, і отримує доступ до особистого кабінету. Інтерфейс авторизації знаходиться на рисунку 5.1, а реєстрації на рисунку 5.2.

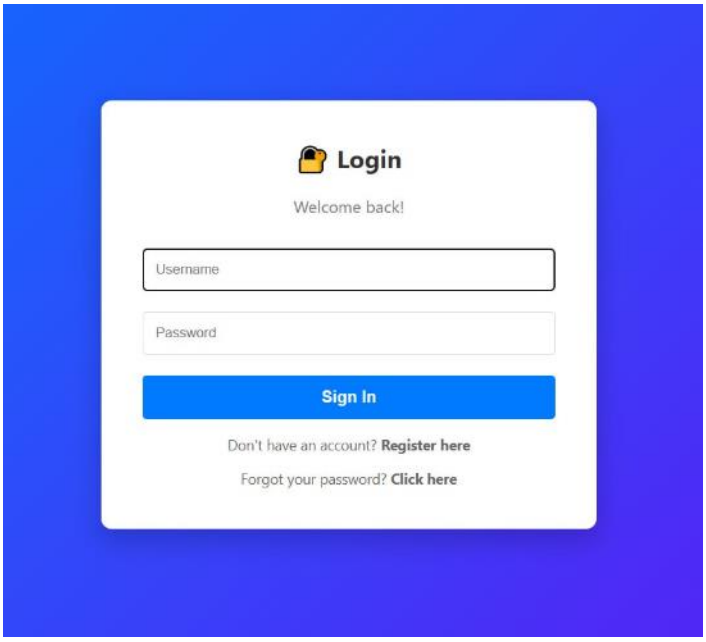


Рисунок 5.1 – Авторизація користувача

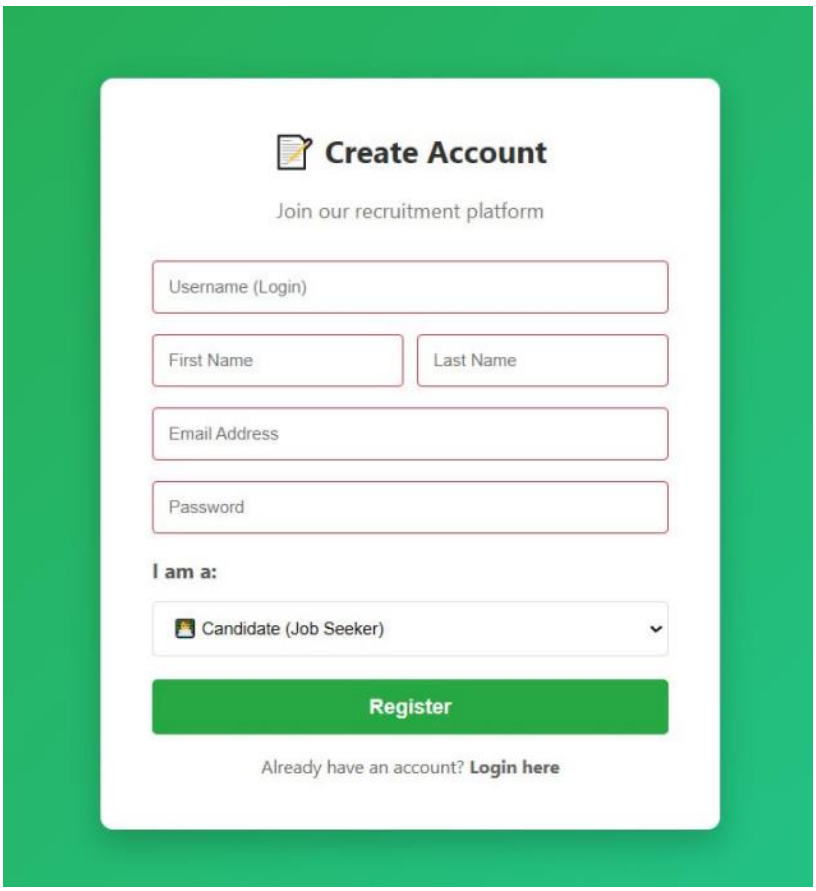


Рисунок 5.2 – Сторінка реєстрації

Після успішної авторизації користувач із роллю «Кандидат» автоматично перенаправляється на головну сторінку особистого кабінету (Dashboard). Інтерфейс сторінки спроєктовано таким чином, щоб надати миттєвий доступ до найважливішої інформації.

У верхній частині розташовано блок поточного статусу, який відображає стан кандидата в системі (наприклад, приналежність до команди або активний пошук) та містить кнопку для швидкого переходу до редагування резюме. Основну частину екрана займає секція «Open Vacancies» (Відкриті вакансії), де у табличному вигляді подано перелік актуальних позицій із зазначенням дати публікації, назви посади та локації. Користувач має можливість відгукнутися на вакансію, натиснувши кнопку «Apply Now».

Зовнішній вигляд головної сторінки кабінету кандидата наведено на рисунку 5.3.

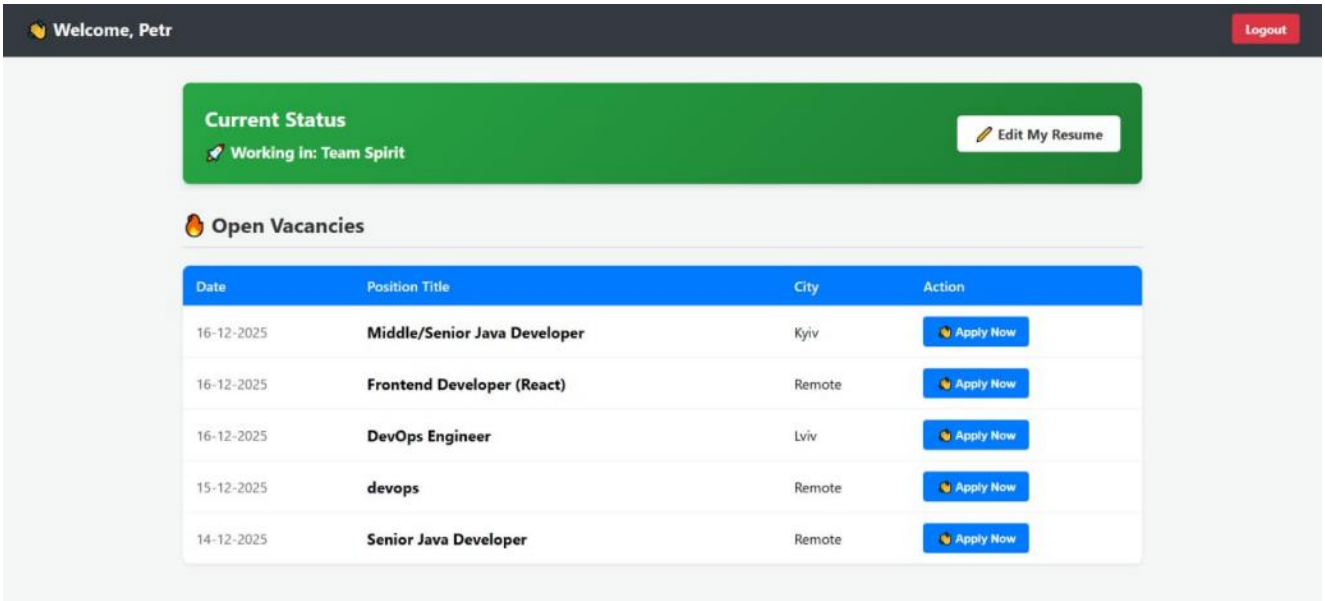


Рисунок 5.3 – Головна сторінка особистого кабінету кандидата

Натиснувши кнопку «Edit My Resume» на головній панелі, користувач потрапляє на сторінку редагування особистих даних. Ця форма є основою для створення цифрового резюме в системі. Інтерфейс дозволяє внести або оновити персональну інформацію (ПІБ, контакти, місто проживання), інтерфейс зображен на рисунку 5.4.

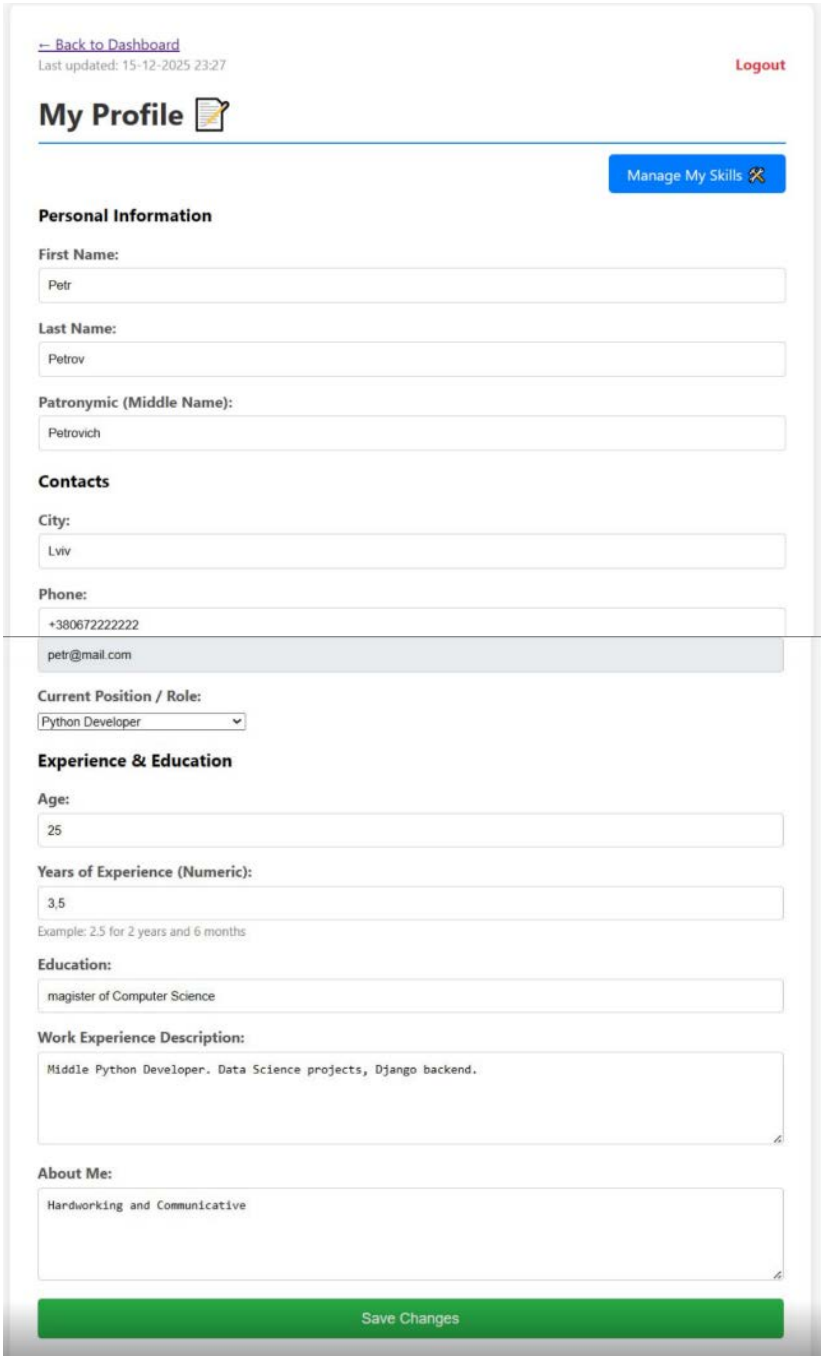


Рисунок 5.4 – Інтерфейс редагування профілю кандидата

Ключовим етапом заповнення профілю є сторінка «My Skills», оскільки саме на основі цих даних розраховується рейтинг відповідності кандидата вимогам вакансії. Інтерфейс розділено на два логічні блоки.

Перший блок дозволяє встановити рівень володіння англійською мовою, що є універсальною вимогою для більшості IT-вакансій. Другий блок призначений для додавання технічних навичок (Hard Skills). Щоб уникнути дублювання та помилок у назвах технологій, вибір навички здійснюється з

випадаючого списку, який підтягується з довідника бази даних. Користувач самостійно оцінює свій рівень експертизи за шкалою від 1 (Beginner) до 5 (Expert).

У нижній частині сторінки формується динамічна таблиця «Current Technical Skills». Вона візуалізує поточний набір компетенцій кандидата за допомогою зіркового рейтингу та надає можливість видалити помилково додані записи.

Зовнішній вигляд сторінки управління навичками наведено на рисунку 5.5.

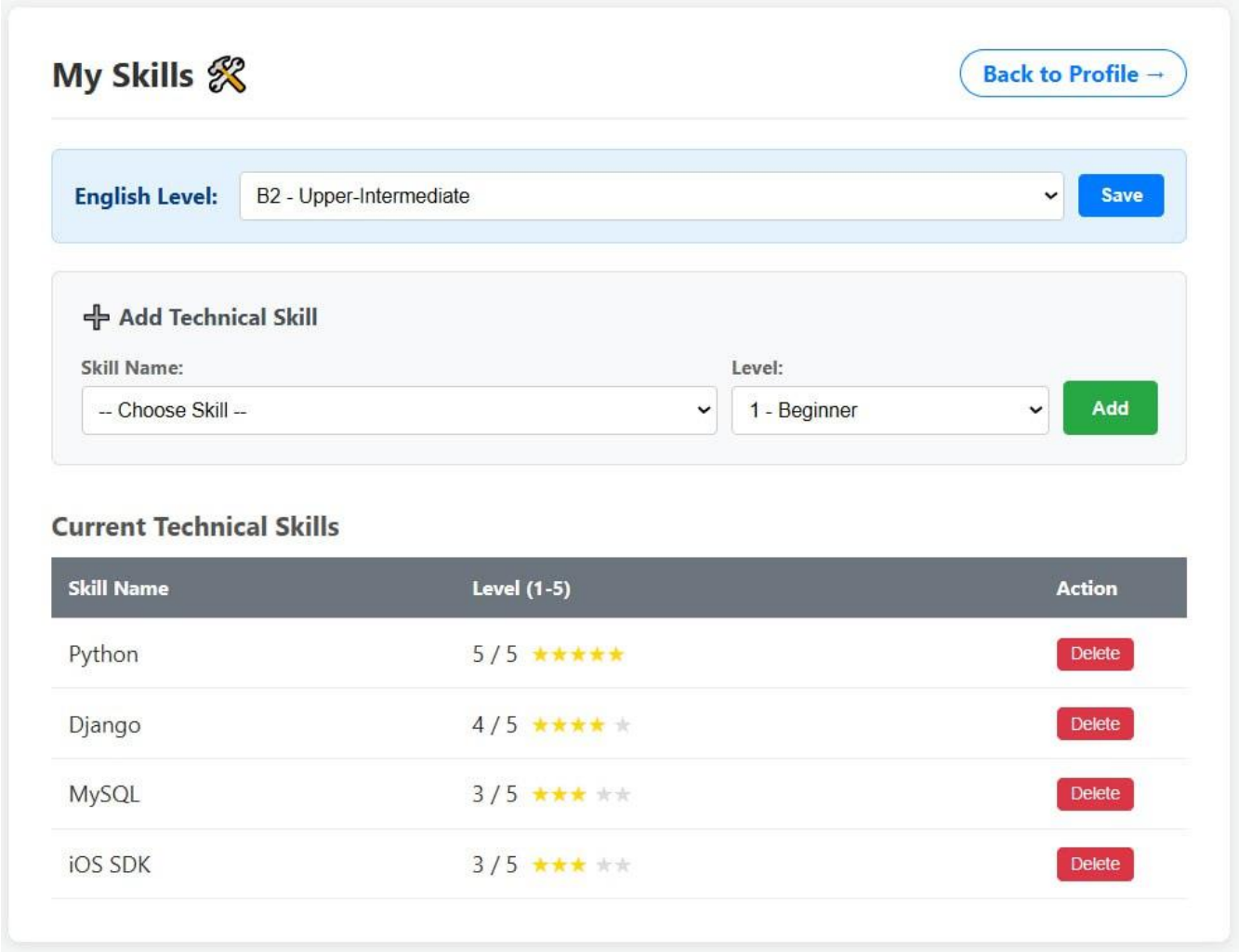


Рисунок 5.5 – Інтерфейс управління навичками кандидата

5.3.2 Сценарій роботи HR-менеджера

Після авторизації в системі під обліковим записом рекрутера користувач потрапляє на головну сторінку управління вакансіями (HR Dashboard). Цей екран є відправною точкою для всіх процесів найму.

Центральним елементом інтерфейсу є таблиця «Active Job Positions», яка містить перелік усіх відкритих вакансій компанії із зазначенням дати створення, назви посади та локації. Для зручності роботи з великою кількістю даних реалізовано функцію сортування за датою (Newest/Oldest).

Зовнішній вигляд панелі управління наведено на рисунку 5.6.

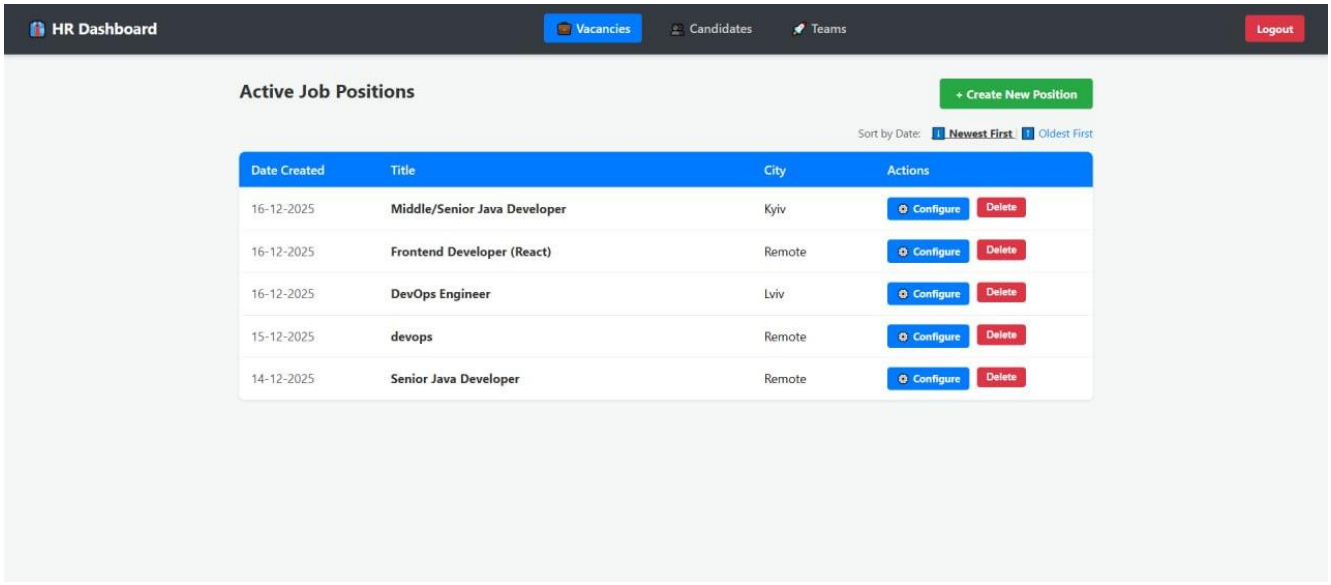


Рисунок 5.6 – Головна панель управління вакансіями (HR Dashboard)

Процес додавання нової позиції розпочинається з заповнення базової інформації. Натискання кнопки створення на головній панелі відкриває відповідну форму. Форма створення нової вакансії зображена на рисунку 5.7.

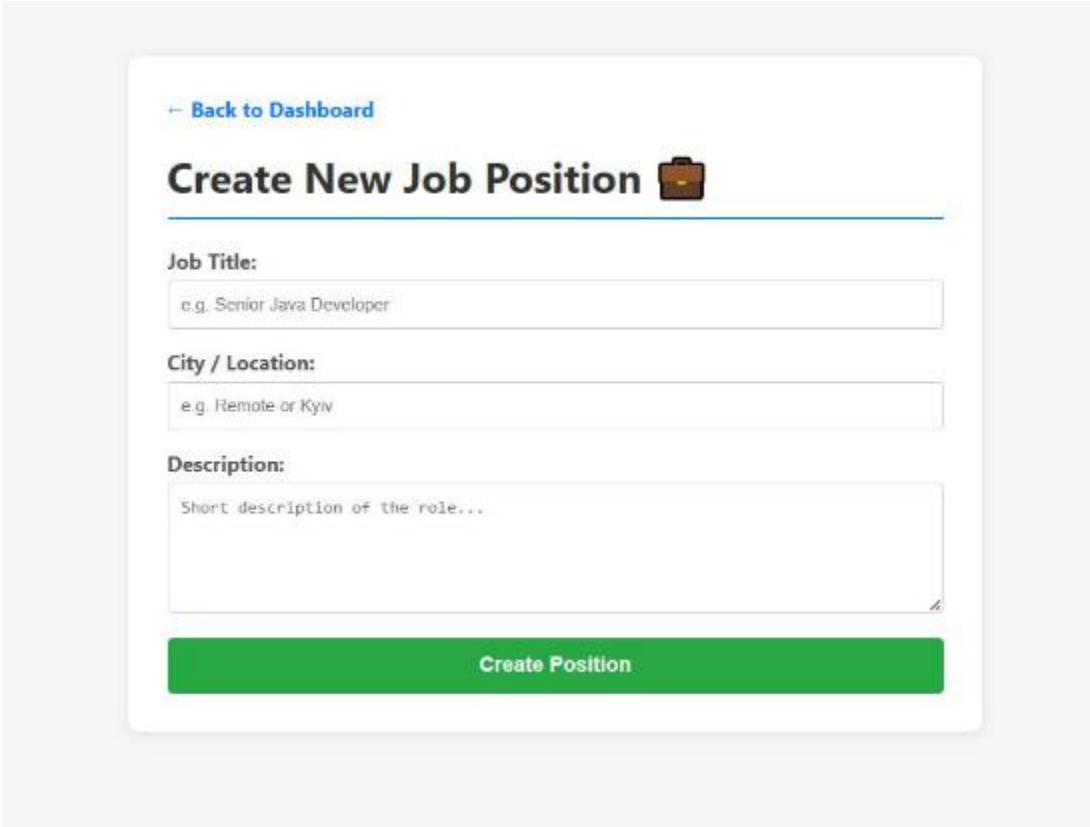


Рисунок 5.7 – Форма створення нової вакансії

Після створення вакансії HR-менеджер переходить до етапу конфігурації профілю вимог. Ця сторінка є інтерфейсом для введення параметрів математичної моделі, описаної у другому розділі дипломної роботи. У нижній частині екрана розташована форма «Add New Requirement», яка дозволяє додати нову технічну вимогу. Для кожного критерію (навички) рекрутер задає два ключові параметри ваги та мінімальний рівень володіння. Інтерфейс налаштування вимог до вакансії наведено на рисунку 5.8.

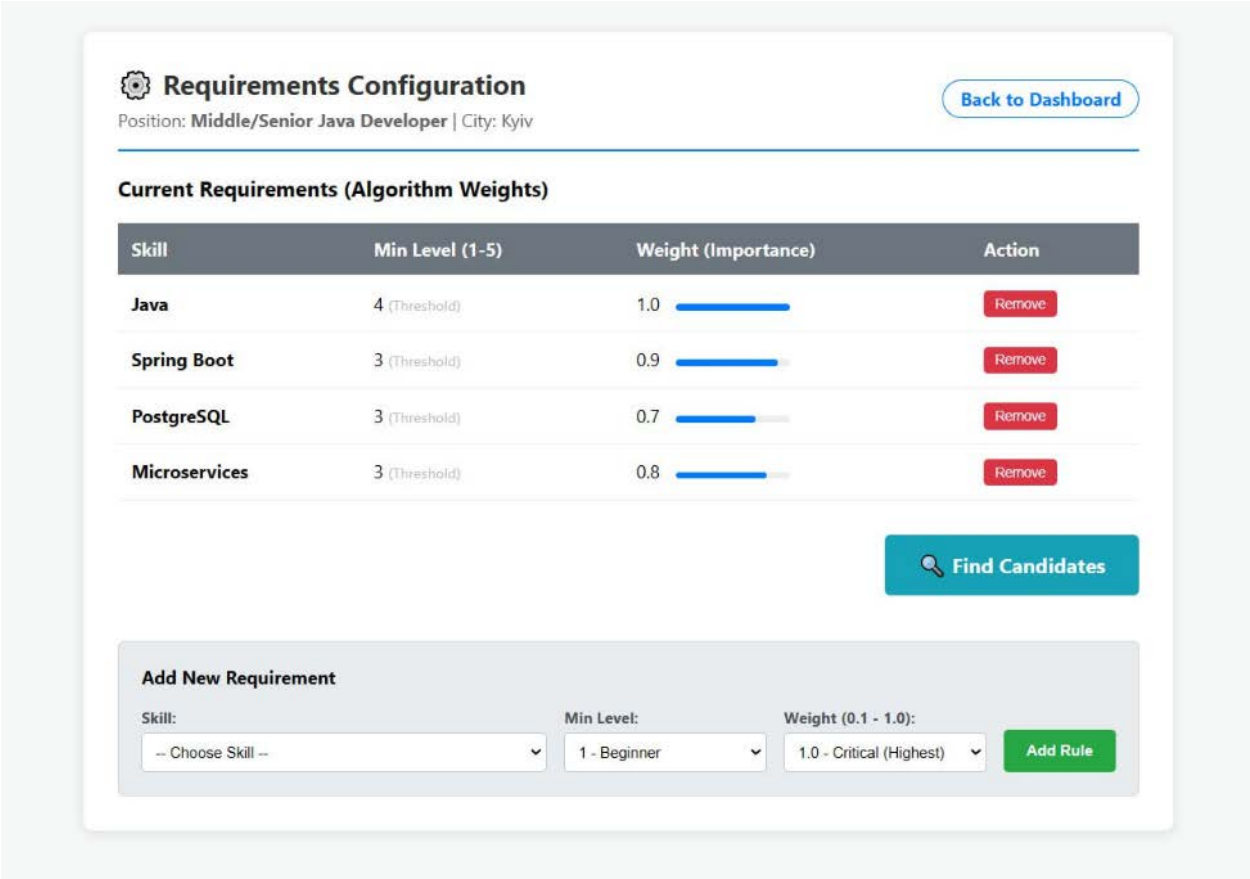


Рисунок 5.8 – Інтерфейс конфігурації вимог та ваг алгоритму

Після натискання кнопки пошуку система виконує обчислення рейтингу для кожного кандидата в базі даних та генерує сторінку «AI Match Results». Цей екран демонструє кінцевий результат роботи розробленого програмного забезпечення.

Список кандидатів автоматично відсортований за спаданням показника Match Score. Цей показник є відсотковим відображенням того, наскільки навички кандидата відповідають профілю вакансії з урахуванням їх ваги. Для зручності сприйняття реалізовано кольорову індикацію.

Зовнішній вигляд сторінки результатів інтелектуального пошуку наведено на рисунку 5.9.

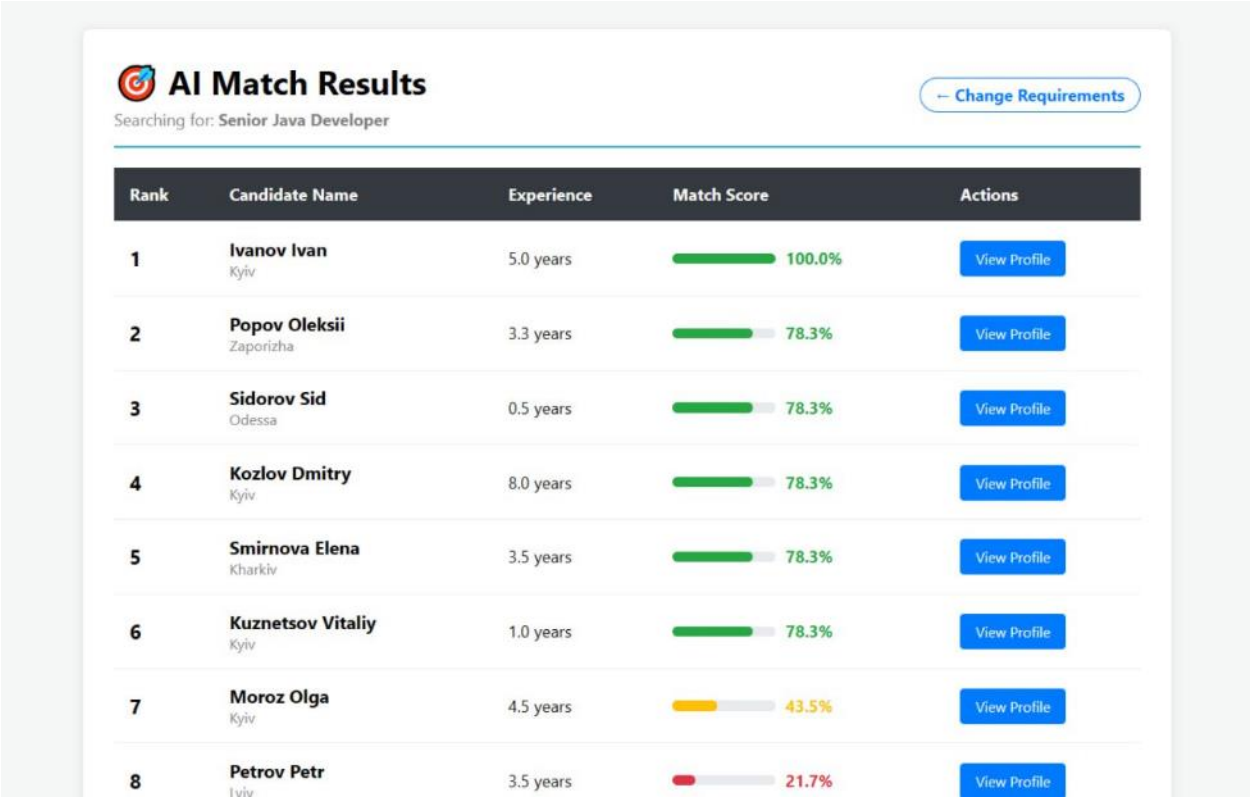


Рисунок 5.9 – Результати роботи алгоритму інтелектуального підбору

Окрім автоматизованого підбору під конкретну вакансію, HR-менеджер має повний доступ до загального реєстру талантів через вкладку «Candidates». Цей модуль призначений для оперативного моніторингу кадрового резерву компанії.

Для ефективної роботи з великими масивами даних інтерфейс оснащено потужною системою фільтрації та сортування, яка дозволяє знайти потрібного спеціаліста за лічені секунди.

Пошук за ключовими словами – текстове поле дозволяє миттєво знайти кандидата за ім'ям або прізвищем.

Фільтрація за навичками – випадаючий список «Skill» дозволяє зробити вибірку кандидатів за конкретною технологією (наприклад, показати всіх, хто знає «Java» або «Docker»), ігноруючи інші параметри.

Динамічне сортування – система дозволяє впорядковувати список за статусом (Active/Employed), позицією, містом, досвідом або датою оновлення профілю.

Інтерфейс загальної бази кандидатів із інструментами пошуку зображено на рисунку 5.10.

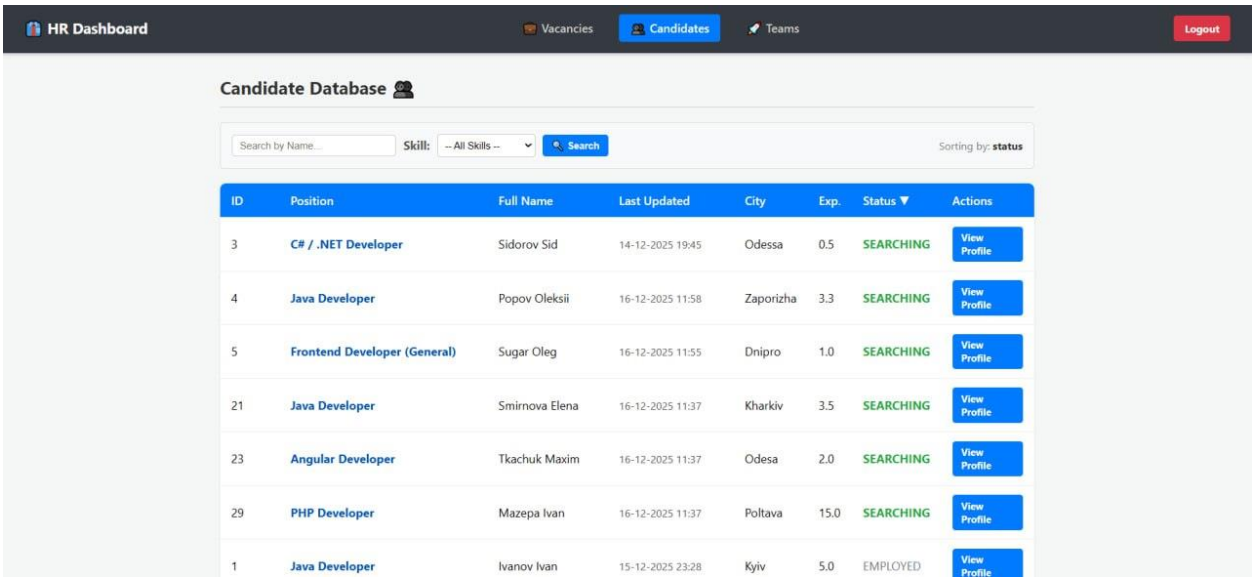


Рисунок 5.10 – База кандидатів із засобами фільтрації та пошуку

Клікнувши на кандидата у загальному списку або у результатах пошуку, рекрутер переходить до сторінки детального перегляду профілю. Цей екран консолідує всю наявну інформацію: контактні дані, історію освіти, детальний опис попереднього досвіду роботи та верифікований список технічних навичок.

Права панель «Actions» надає інструменти для зміни життєвого циклу кандидата в компанії:

– управління статусом, система дозволяє переводити кандидата зі стану пошуку (SEARCHING) у стан найму (EMPLOYED);

– розподіл по командах, HR-менеджер може призначити співробітника у конкретну проєктну команду (наприклад, «Alpha Team») через випадаючий список. Це автоматично оновлює структуру команди в базі даних;

– звільнення, кнопка «Fire / Dismiss» дозволяє припинити співпрацю, після чого статус кандидата оновлюється, а його профіль архівується або повертається у загальний пошук.

Інтерфейс управління статусом та командним розподілом кандидата наведено на рисунку 5.11.

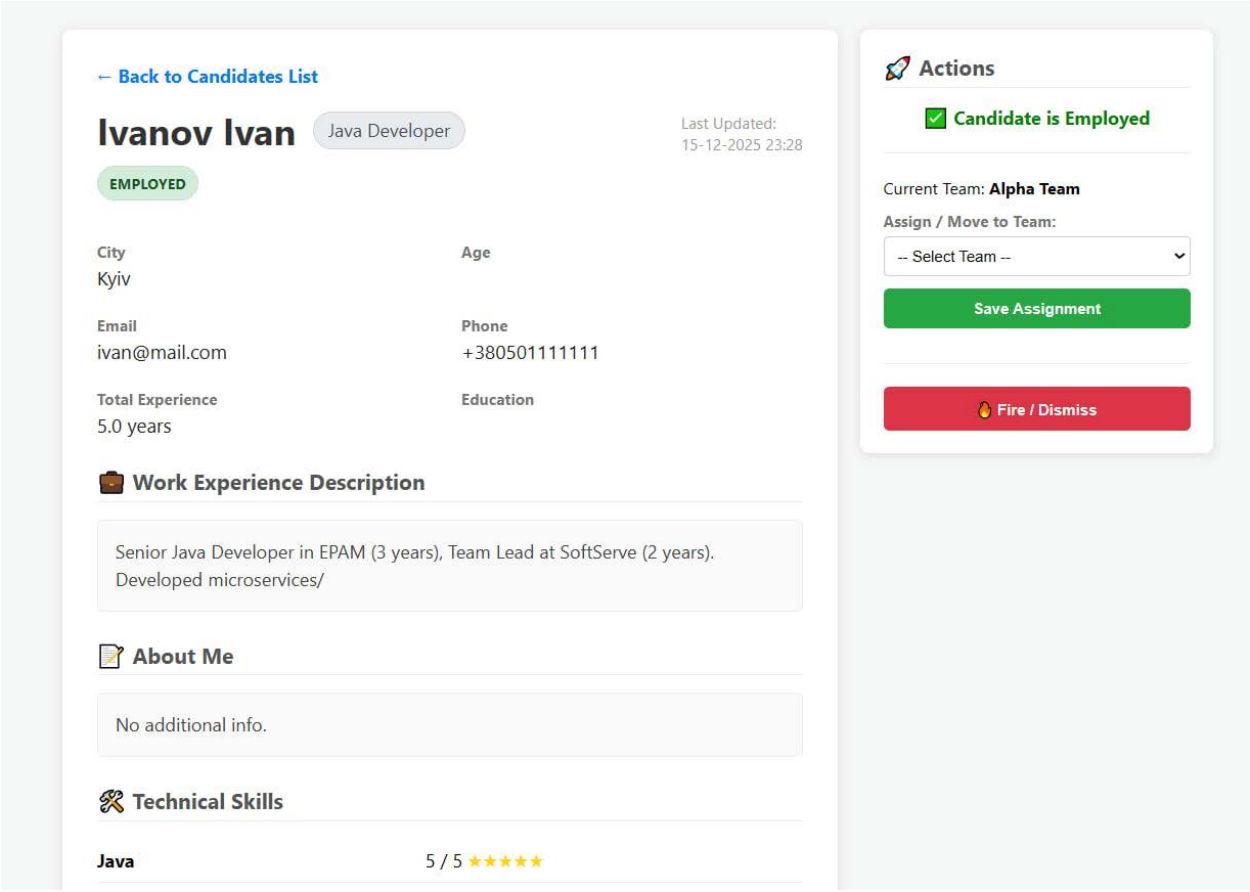


Рисунок 5.11 – Інтерфейс детального перегляду та управління статусом кандидата

Вкладка «Teams» надає централізований доступ до управління організаційною структурою компанії. Цей модуль дозволяє HR-менеджеру бачити загальну картину зайнятості та розподілу ресурсів. У таблиці відображається перелік активних команд, дата їх створення та короткий опис проєкту, над яким вони працюють. Інтерфейс списку активних проєктних команд зображено на рисунку 5.12.

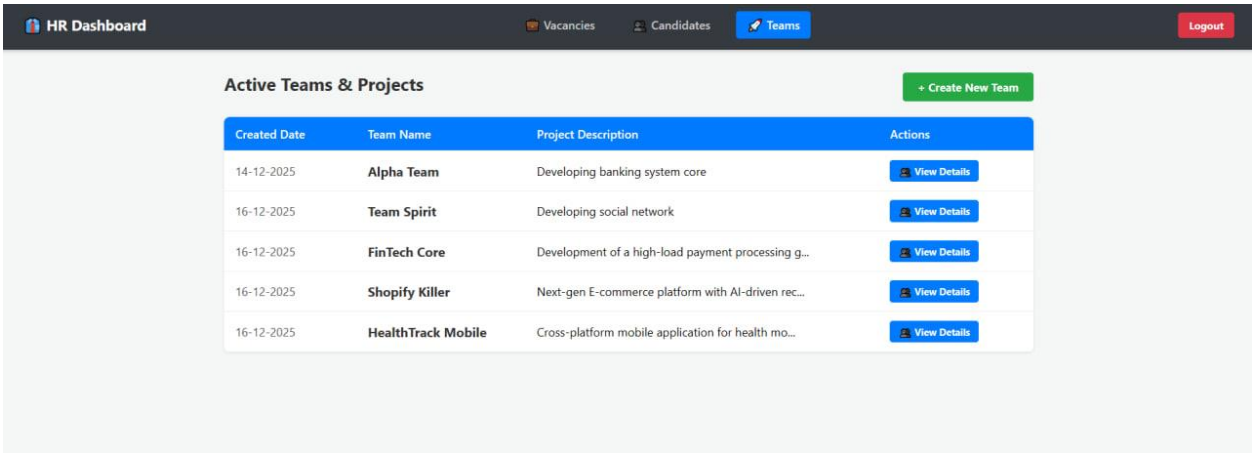


Рисунок 5.12 – Панель управління командами та проєктами

Для розширення організаційної структури компанії передбачено функціонал додавання нових команд. Натискання кнопки «Create New Team» відкриває модальне вікно або окрему сторінку з формою створення. Після підтвердження введення нова сутність записується в базу даних і стає доступною у випадаючих списках для призначення співробітників.

Інтерфейс створення нової команди наведено на рисунку 5.13.

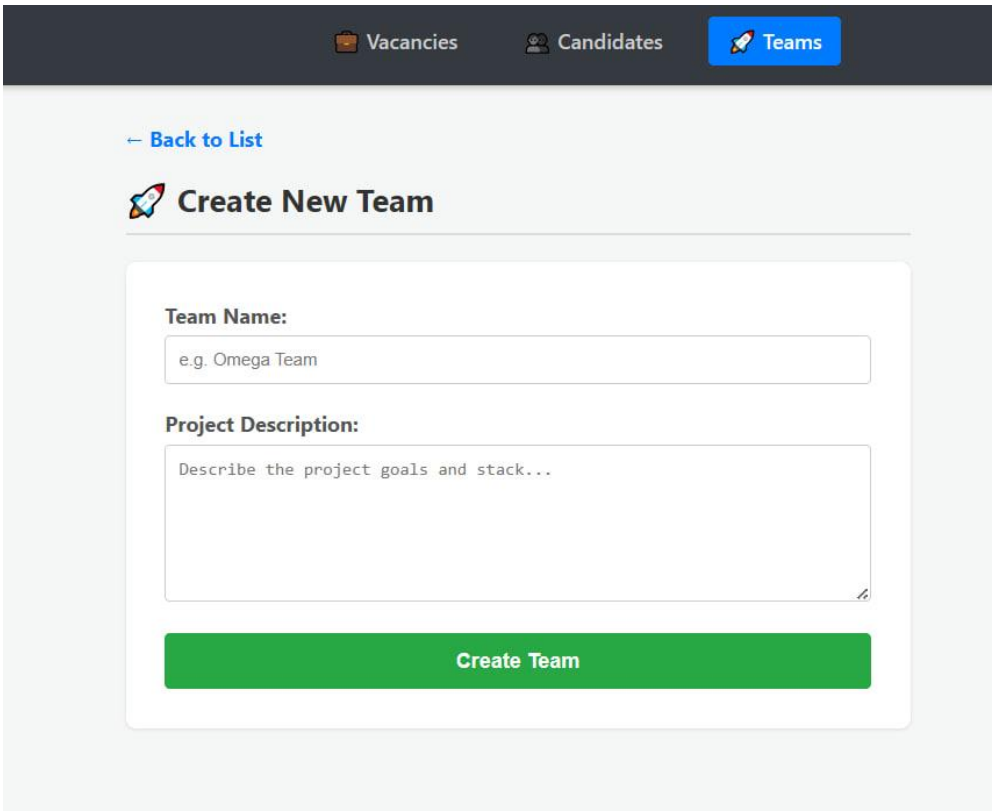


Рисунок 5.13 – Форма створення нової проєктної команди

При виборі конкретної команди зі списку відкривається сторінка детальної інформації. Цей інтерфейс призначений для оперативного управління складом робочої групи.

У верхній частині сторінки відображається назва команди та опис проєкту, що дозволяє менеджеру тримати в фокусі бізнес-цілі групи. Основну частину екрана займає таблиця «Team Members», яка містить перелік усіх призначених фахівців із зазначенням їхніх позицій (Java Developer, QA, PM тощо) та контактних даних.

Інтерфейс адміністрування складу проєктної команди наведено на рисунку 5.14.

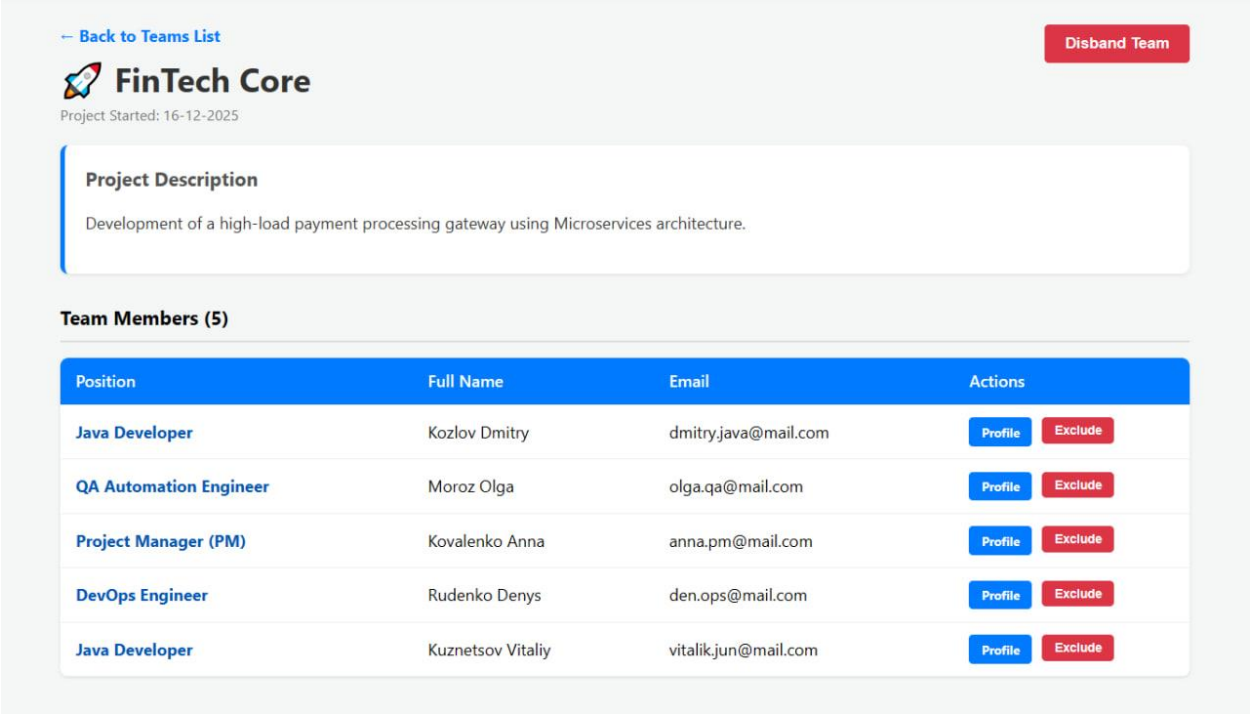


Рисунок 5.14 – Детальний склад команди та інструменти управління

Важливим аспектом проєктування інтерфейсу користувача є захист від випадкових деструктивних дій, що можуть призвести до втрати важливої інформації. Для забезпечення цілісності даних у системі реалізовано механізм подвійного підтвердження операцій (Confirmation Modals).

При спробі виконати будь-яку незворотну дію — таку як розформування команди, видалення вакансії, очищення списку навичок або

звільнення співробітника — система блокує інтерфейс модальним вікном. Воно містить попередження про наслідки та вимагає явного підтвердження намірів користувача.

Приклад реалізації такого механізму захисту («Danger Zone») при розформуванні команди наведено на рисунку 5.15.

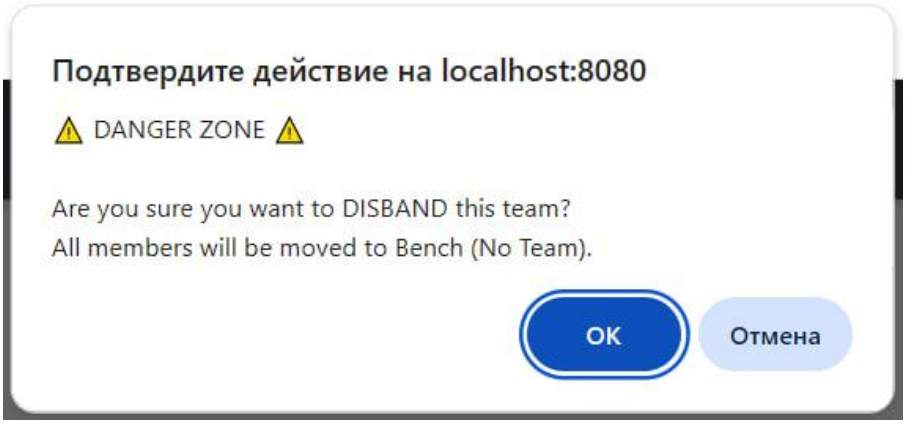


Рисунок 5.15 – Модальне вікно підтвердження критичної операції

5.4 Тестування програмного продукту

Метою етапу тестування є комплексна перевірка відповідності реалізованого функціоналу вимогам, сформульованим у технічному завданні, а також підтвердження стабільності роботи алгоритмів підбору. Особлива увага приділялася верифікації коректності математичних розрахунків у сервісному шарі додатку та забезпеченню цілісності даних при виконанні CRUD-операцій.

Для перевірки працездатності системи було застосовано метод функціонального тестування (Black Box Testing) [32]. Цей підхід передбачає перевірку реакції системи на вхідні дані без аналізу внутрішньої структури коду. Для забезпечення повноти покриття функціоналу було розроблено набір контрольних прикладів (тест-кейсів), що включають стандартні сценарії використання, перевірку обробки помилок та граничних значень [33]. Результати виконання тестів наведено у таблиці 5.1 [34].

Таблиця 5.1 – Протокол функціонального тестування системи

№	Сценарій тестування (Test Case)	Очікуваний результат	Фактичний результат	Статус
1	Авторизація: Вхід із невірним паролем	Система відмовляє у доступі, виводиться повідомлення про помилку	Доступ заборонено, повідомлення відображено	Пройдено
2	Реєстрація: Створення профілю кандидата	Дані зберігаються в БД, відбувається автоматичний вхід у кабінет	Профіль створено, перенаправлення успішне	Пройдено
3	Валідація: Створення вакансії без обов'язкових полів	Система блокує збереження, поля підсвічуються червоним	Вакансія не створена, валідація спрацювала	Пройдено
4	Matching Алгоритм: Розрахунок повної відповідності	Кандидат, що має всі навички з необхідним рівнем, отримує 100% Score	Кандидат отримав 100% рейтингу	Пройдено
5	Matching Алгоритм: Розрахунок часткової відповідності	Кандидат із рівнем навички нижче необхідного отримує зменшений бал	Рейтинг розраховано коректно (менше 100%)	Пройдено
6	Фільтрація: Пошук за навичкою	У списку відображаються лише кандидати, що володіють обраною технологією	Список відфільтровано коректно	Пройдено
7	Управління командами: Додавання учасника	Статус кандидата змінюється на "EMPLOYED", він з'являється у складі команди	Учасник доданий до списку команди, статус оновлено	Пройдено
8	Адміністрування: Видалення вакансії	Вакансія зникає зі списку доступних для кандидатів	Запис видалено з бази даних	Пройдено

5.5 Експериментальне дослідження

Для оцінки практичної цінності розробленого програмного забезпечення було проведено експериментальне дослідження, метою якого є порівняння часових витрат на підбір персоналу при традиційному (ручному) підході та при використанні розробленої автоматизованої системи.

В якості об'єкта дослідження обрано процес ранжування бази кандидатів під вимоги конкретної вакансії.

Умови експерименту:

– вхідні дані, вакансія «Senior Java Developer» з 5 вимогами (Java, Spring, SQL, Docker, AWS;

– обсяг вибірки, тестування проводилося на наборах даних з 10, 50, 100 та 500 кандидатів;

– сценарій А (Ручний режим), HR-менеджер відкриває кожне резюме, аналізує навички, порівнює їх з вимогами та суб'єктивно оцінює відповідність. Середній час обробки одного резюме прийнято за 3 хвилини (180 секунд);

– сценарій Б (Автоматизований режим), використовується розроблений алгоритм MatchingService, який автоматично обраховує рейтинг та сортує список. Час вимірюється системним таймером.

Результати вимірювань часових витрат для обох сценаріїв наведено в таблиці 5.2.

Таблиця 5.2 – Результати експериментального дослідження

Кількість кандидатів	Час обробки вручну (хв)	Час роботи алгоритму (мс)	Прискорення процесу (разів)
10	30 хв	~15 мс	120000
50	150 хв (2.5 год)	~45 мс	200000
100	300 хв (5 год)	~80 мс	225000
500	1500 хв (25 год)	~320 мс	280000

Як видно з отриманих даних, залежність часу виконання операції від кількості кандидатів у ручному режимі є лінійною і критично зростає зі збільшенням бази даних. Обробка 500 резюме вручну зайняла б у рекрутера понад 3 робочі дні (25 годин чистого часу).

Натомість розроблена система виконує ту саму задачу менш ніж за 1 секунду (320 мс для 500 записів). Це свідчить про те, що алгоритмічна складність реалізованого рішення дозволяє миттєво обробляти великі масиви даних [35].

Тож впровадження розробленої системи дозволяє скоротити час на етап первинного відбору (Screening) практично до нуля, звільняючи HR-спеціаліста для проведення співбесід та прийняття рішень. Економічна ефективність від впровадження системи є високою.

5.6 Висновки за розділом 5

У п’ятому розділі виконано комплексний аналіз експлуатаційних характеристик розробленої системи автоматизації рекрутингу та проведено перевірку її працездатності.

Було сформульовано системні вимоги. Визначено мінімально необхідні параметри апаратного та програмного забезпечення для серверної та клієнтської частин.

Описано процес експлуатації. Розроблено детальні сценарії використання системи для ролей HR-менеджера та Кандидата. Наведено покроковий опис ключових бізнес-процесів.

Було проведено функціональне тестування основних модулів системи методом «чорної скриньки». А експериментальне дослідження показало значну перевагу автоматизованого підходу над ручним.

Таким чином, розроблена інформаційна система повністю відповідає поставленим вимогам, пройшла етап верифікації та готова до впровадження у промислову експлуатацію.

ВИСНОВКИ

В умовах динамічного розвитку ринку інформаційних технологій конкуренція за кваліфіковані кадри стає одним із ключових викликів для IT-компаній. Ефективність формування проєктних команд напряму впливає на успішність реалізації комерційних проєктів. Проте традиційні методи рекрутингу, що базуються на ручному опрацюванні резюме, характеризуються значними часовими витратами та високим впливом людського фактору. Тому будь-яка компанія повинна бути зацікавленою у розробці спеціалізованих інформаційних систем, здатних автоматизувати та інтелектуалізувати ці процеси.

У дипломній роботі вирішено задачу мінімізації часових витрат та зменшенні трудомісткості процесів рекрутингу шляхом створення програмного забезпечення для підбору та оцінки кандидатів, тож мета роботи була досягнута в повному обсязі.

В рамках роботи було проведено системний аналіз предметної області, досліджено бізнес-процеси найму в IT-сфері та виявлено недоліки існуючих рішень. Встановлено, що ключовою проблемою є складність об'єктивного ранжування кандидатів за сукупністю технічних навичок (Hard Skills). Це обґрунтувало необхідність створення власної системи з гнучким алгоритмом зваженого пошуку відповідності (Matching Algorithm).

В результаті виконання роботи було створено веб-застосунок на базі технологій Java Enterprise (Spring Boot). Використання архітектурного патерну MVC забезпечило модульність коду та легкість його супроводу. Реалізовано зручний інтерфейс користувача за допомогою Thymeleaf, який надає інструменти для створення вакансій з налаштуванням ваг, автоматичного пошуку та адміністрування команд.

Практична цінність роботи полягає у створенні повноцінного програмного продукту, готового до впровадження в IT-компаніях. Система надає можливість не лише автоматизувати рутинні операції, а й забезпечити

відбор якісного персоналу у короткий проміжок часу завдяки об'єктивній математичній оцінці компетенцій, що в кінцевому підсумку сприяє формуванню більш ефективних команд розробників.

Перспективи подальшого розвитку системи вбачаються у впровадженні засобів штучного інтелекту для автоматичної обробки файлів резюме та переході до мікросервісної архітектури для забезпечення масштабування у великих корпораціях.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Armstrong, M. Handbook of Human Resource Management Practice / M. Armstrong, S. Taylor. – 15th edition. – London: Kogan Page, 2020. – 880 p.

2. Strohmeier S. Digital Human Resource Management: A Conceptual Clarification / S. Strohmeier // German Journal of Human Resource Management. – 2020. – Vol. 34. – Issue 3. – 345 p.

3. Laakso J. Recruitment Process Automation: Benefits and Challenges / J. Laakso // Journal of Human Resources Management. – 2021 – Vol. 15, No. 3. – P. 45–58.

4. Workday Human Capital Management. Official Website [Electronic resource]. – Access mode: https://www.workday.com/en-us/products/human-capital-management/overview.html (дата звернення: 05.10.2025).

5. Greenhouse Software. Hiring Software & ATS for Enterprises [Electronic resource]. – Access mode: https://www.greenhouse.io/ (дата звернення: 06.10.2025).

6. CleverStaff – професійний софт для рекрутингу. Офіційний сайт [Електрон. ресурс]. – Режим доступу: https://cleverstaff.net/ua/ (дата звернення: 06.10.2025).

7. Davenport T. H. Process Innovation: Reengineering Work through Information Technology / T. H. Davenport. – Harvard Business School Press, 1993. – 336 p.

8. Integration Definition for Function Modeling (IDEF0). Draft Federal Information Processing Standards Publication, 1993. – 183 p.

9. Грищук Ю. В. Моделювання бізнес-процесів : навчальний посібник / Ю. В. Грищук. – Львів : Видавництво Львівської політехніки, 2022. – 210 с.

10. Saaty T. L. Decision Making for Leaders: The Analytic Hierarchy Process for Decisions in a Complex World 3rd ed. / T. L. Saaty. – Pittsburgh : RWS Publications, 2013. – 320 p.

11. Cormen T. H. Introduction to Algorithms. 4th ed / [T. H. Cormen, C. E. Leiserson, R. L. Rivest et al.]. – MIT Press, 2022. – 1312 p.

12. Bloch J. Effective Java. 3rd ed / J. Bloch. – Addison-Wesley Professional, 2018. – 416 p.

13. Project Lombok Features [Electronic resource]. – Access mode: https://projectlombok.org/features (дата звернення: 8.11.2025).

14. Spring Boot Reference Documentation [Electronic resource]. – Access mode: https://docs.spring.io/spring-boot/docs/current/reference/html/ (дата звернення: 11.10.2025).

15. Walls C. Spring in Action. 6th ed. / C. Walls. – Manning Publications, 2022. – 520 p.

16. PostgreSQL 14.5 Documentation [Electronic resource]. – Access mode: https://www.postgresql.org/docs/14/index.html (дата звернення: 11.10.2025).

17. Garcia-Molina H. Database Systems: The Complete Book. 2nd ed. / H. Garcia-Molina, J. D Ullman., J. Widom – Pearson, 2008. – 1248 p.

18. Thymeleaf Documentation [Electronic resource]. – Access mode: https://www.thymeleaf.org/documentation.html (дата звернення: 14.10.2025).

19. Maven – Introduction to the Standard Directory Layout [Electronic resource]. – Access mode: https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html (дата звернення: 14.10.2025).

20. Booch G. The Unified Modeling Language User Guide. 2nd ed. / G. Booch, J. Rumbaugh, I. Jacobson. – Addison-Wesley Professional, 2005. – 496 p.

21. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) : ISO/IEC 25010:2011. – [Effective from 2011-03-01]. – Geneve: ISO, 2011. – 26 p.

22. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design. 3rd ed. / C. Larman. – Prentice Hall, 2004. – 736 p.

23. Rumbaugh J. The Unified Modeling Language Reference Manual. 2nd

ed. / J. Rumbaugh, I. Jacobson, G. Booch. – Pearson, 2004. – 721 p.

24. Date C. J. An Introduction to Database Systems. 8th ed. / C. J. Date. – Pearson, 2003.– 1328 p.

25. Bauer C. Java Persistence with Hibernate. 2nd ed. / C. Bauer, G. King, G. Gregory. – Manning Publications, 2015. – 608 p.

26. Fowler M. Patterns of Enterprise Application Architecture / M. Fowler. – Addison-Wesley Professional, 2002. – 560 p.

27. Spring Security Reference [Electronic resource]. – Access mode: https://docs.spring.io/spring-security/reference/index.html (дата звернення: 25.10.2025).

28. MDN Web Docs: HTML & CSS [Electronic resource]. – Access mode: https://developer.mozilla.org/en-US/ (дата звернення: 25.10.2025).

29. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design / R. C. Martin. – Prentice Hall, 2017. – 432 p.

30. Apache Tomcat 10 Documentation [Electronic resource]. – Access mode: https://tomcat.apache.org/tomcat-10.0-doc/index.html
(дата звернення: 15.11.2025).

31. Heckler M. Spring Boot: Up and Running: Building Cloud Native Java and Kotlin Applications / M. Heckler. – O'Reilly Media, 2021. – 320 p.

32. Myers G. J. The Art of Software Testing. 3rd ed. / G. J. Myers, C. Sandler, T. Badgett. – Wiley, 2011. – 240 p.

33. Beizer B. Software Testing Techniques. 2nd ed. / B. Beizer. – Van Nostrand Reinhold, 1990. – 550 p.

34. JUnit 5 User Guide [Electronic resource]. – Access mode: https://junit.org/junit5/docs/current/user-guide/ (дата звернення: 19.11.2025).

35. Sommerville I. Software Engineering. 10th ed. / I. Sommerville. – Pearson, 2015. – 816 p.

ДОДАТОК А

Текст програми

А.1 Текст файлу Candidate.java

```
package com.diploma.recruitment.entity;

import jakarta.persistence.*;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

@Entity
@Table(name = "candidates")
@Data
@NoArgsConstructor
public class Candidate {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @OneToOne
    @JoinColumn(name = "user_id", referencedColumnName = "id")
    private User user;

    @Column(name = "last_name", nullable = false)
    private String lastName;

    @Column(name = "first_name", nullable = false)
    private String firstName;

    private String patronymic;

    @Column(nullable = false)
    private String email;

    private String phone;
    private String city;
    private Integer age;
    private String education;

    @Column(name = "experience_years")
    private Double experienceYears;
```

```
    @Column(name = "work_experience_description") // Вказуємо точне ім'я колонки в БД
    private String workExperienceDescription;

    @Column(name = "about_me")
    private String aboutMe;

    private String status;

    @Column(name = "last_updated_at")
    private LocalDateTime lastUpdatedAt;

    @ManyToOne
    @JoinColumn(name = "team_id") // Зв'язок із таблицею teams
    private Team team;

    @ManyToOne
    @JoinColumn(name = "position_id") // Зв'язок з таблицею positions
    private Position position;

    // Зв'язок: Один кандидат -> Багато навичок
    // mappedBy = "candidate" вказує на поле 'candidate' у класі CandidateSkill
    @OneToMany(mappedBy = "candidate", cascade = CascadeType.ALL, orphanRemoval = true)
    @ToString.Exclude
    private List<CandidateSkill> candidateSkills = new ArrayList<>();
}
```

А.2 Текст файлу CandidateSkill.java

```
package com.diploma.recruitment.entity;

import jakarta.persistence.*;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Table(name = "candidate_skills")
@Data
@NoArgsConstructor
public class CandidateSkill {

    @EmbeddedId
```

```
    private CandidateSkillKey id;

    @ManyToOne
    @MapsId("candidateId") // Зв'язує поле ключа candidateId з об'єктом
Candidate
    @JoinColumn(name = "candidate_id")
    private Candidate candidate;

    @ManyToOne
    @MapsId("skillId") // Зв'язує поле ключа skillId з об'єктом Skill
    @JoinColumn(name = "skill_id")
    private Skill skill;

    @Column(name = "proficiency_level", nullable = false)
    private Integer proficiencyLevel;
}
```

А.3 Текст файлу CandidateSkillKey.java

```
package com.diploma.recruitment.entity;

import jakarta.persistence.Column;
import jakarta.persistence.Embeddable;
import lombok.Data;
import lombok.EqualsAndHashCode;
import java.io.Serializable;

@Embeddable
@Data
@EqualsAndHashCode
public class CandidateSkillKey implements Serializable {

    @Column(name = "candidate_id")
    private Long candidateId;

    @Column(name = "skill_id")
    private Long skillId;
}
```

А.4 Текст файлу JobPosition.java

```
package com.diploma.recruitment.entity;

import jakarta.persistence.*;
import lombok.Data;
```

```
import lombok.NoArgsConstructor;
import java.time.LocalDateTime;

@Entity
@Table(name = "job_positions")
@Data
@NoArgsConstructor
public class JobPosition {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String title;

    private String description;

    @Column(name = "sity") // Мапінг: в Java "city", в БД колонка "sity"
    private String city;

    @Column(name = "created_at")
    private LocalDateTime createdAt = LocalDateTime.now();
}
```

А.5 Текст файлу JobRequirement.java

```
package com.diploma.recruitment.entity;

import jakarta.persistence.*;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Table(name = "job_requirements")
@Data
@NoArgsConstructor
public class JobRequirement {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "job_position_id", nullable = false)
```

```
    private JobPosition jobPosition;

    @ManyToOne
    @JoinColumn(name = "skill_id", nullable = false)
    private Skill skill;

    @Column(nullable = false)
    private Double weight; // Той самий коефіцієнт (0.1 - 1.0)

    @Column(name = "min_level")
    private Integer minLevel; // Мінімальний поріг (1-5)
}
```

А.6 Текст файлу Position.java

```
package com.diploma.recruitment.entity;

import jakarta.persistence.*;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Table(name = "positions")
@Data
@NoArgsConstructor
public class Position {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String name; // Наприклад: "Java Developer", "QA Engineer"
}
```

А.7 Текст файлу Skill.java

```
package com.diploma.recruitment.entity;

import jakarta.persistence.*;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Table(name = "skills")
```

```
@Data
@NoArgsConstructor
public class Skill {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String name;
}
```

А.8 Текст файлу Team.java

```
package com.diploma.recruitment.entity;

import jakarta.persistence.*;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.time.LocalDateTime;

@Entity
@Table(name = "teams")
@Data
@NoArgsConstructor
public class Team {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String name;

    @Column(name = "project_description")
    private String projectDescription;


    @Column(name = "created_at")
    private LocalDateTime createdAt = LocalDateTime.now(); //
Автоматически ставим дату при создании объекта
}
```

А.9 Текст файлу User.java

```
package com.diploma.recruitment.entity;

import jakarta.persistence.*;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Table(name = "users") // Вказуємо, що це таблиця "users"
@NoArgsConstructor // Lombok: порожній конструктор (потрібний для Hibernate)
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String username;

    @Column(nullable = false)
    private String password;

    @Column(nullable = false)
    private String role; // Зберігаємо як рядок: "ROLE_HR" або "ROLE_CANDIDATE"
}
```

А.9 Текст файлу CandidateRepository.java

```
package com.diploma.recruitment.repository;

import com.diploma.recruitment.entity.Candidate;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;
import java.util.Optional;
import org.springframework.data.domain.Sort;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

public interface CandidateRepository extends JpaRepository<Candidate, Long> {
    // Знайти кандидата з логіну пов'язаного User
    Optional<Candidate> findByUserUsername(String username);
```

```
    // Знайти кандидатів, які мають конкретний Skill ID
    // Ми робимо JOIN з таблицею candidateSkills та перевіряємо ID навички
    @Query("SELECT c FROM Candidate c JOIN c.candidateSkills cs WHERE cs.skill.id = :skillId")
    List<Candidate> findBySkillId(@Param("skillId") Long skillId, Sort sort);

    // Знайти всіх за ID команди
    List<Candidate> findByTeamId(Long teamId);

    @Query("SELECT DISTINCT c FROM Candidate c " +
           "LEFT JOIN c.candidateSkills cs " +
           "WHERE (:skillId IS NULL OR cs.skill.id = :skillId) " +
           "AND (:name IS NULL OR c.firstName ILIKE :name OR c.lastName ILIKE :name)")
    List<Candidate> search(@Param("name") String name, @Param("skillId") Long skillId, Sort sort);
}
```

А.10 Текст файлу AuthController.java

```
package com.diploma.recruitment.controller;

import com.diploma.recruitment.entity.Candidate;
import com.diploma.recruitment.entity.User;
import com.diploma.recruitment.repository.CandidateRepository;
import com.diploma.recruitment.repository.UserRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

import java.util.regex.Pattern;

@Controller
@RequestMapping("/auth")
@RequiredArgsConstructor
public class AuthController {

    private final UserRepository userRepository;
    private final CandidateRepository candidateRepository;
```

```
        private final PasswordEncoder passwordEncoder;

        // перeірка
        private static final String USERNAME_REGEX = "^[a-zA-Z0-
9_]{3,20}$";
        // Проста перевірка email
        private static final String EMAIL_REGEX = "^[A-Za-z0-9+_.-
]+@(.+)$";

        @GetMapping("/login")
        public String loginPage() {
           return "auth-login";
        }

        @GetMapping("/register")
        public String registerPage() {
           return "auth-register";
        }

        @PostMapping("/register")
        public String registerUser(@RequestParam String username,
                                   @RequestParam String password,
                                   @RequestParam String firstName,  // Нове поле
                                   @RequestParam String lastName,   // Нове поле
                                   @RequestParam String email,      // Нове поле
                                   @RequestParam String role) {

           // 1. ВАЛІДАЦІЯ ДАНИХ
           // Перевірка на пустоту
           if (username.isBlank() || password.isBlank() || firstName.isBlank() ||
lastName.isBlank() || email.isBlank()) {
              return "redirect:/auth/register?error=missing_fields";
           }

           // Перевірка Username
           if (!Pattern.matches(USERNAME_REGEX, username)) {
              return "redirect:/auth/register?error=invalid_username";
           }

           // Перевірка Email
           if (!Pattern.matches(EMAIL_REGEX, email)) {
              return "redirect:/auth/register?error=invalid_email";
           }

           // Перевірка: чи зайнятий логін
```

```
            if (userRepository.findByUsername(username).isPresent()) {
                return "redirect:/auth/register?error=exists";
            }

            // 2. СТВОРЕННЯ КОРИСТУВАЧА (User)
            User user = new User();
            user.setUsername(username);
            user.setPassword(passwordEncoder.encode(password));
            user.setRole("ROLE_" + role);

            userRepository.save(user);

            // 3. ЯКЩО ЦЕ КАНДИДАТ -> Створюємо анкету із заповненими
даними
            if ("CANDIDATE".equals(role)) {
                Candidate candidate = new Candidate();
                candidate.setUser(user);

                // Записываем данные из формы в анкету
                candidate.setFirstName(firstName);
                candidate.setLastName(lastName);
                candidate.setEmail(email);

                candidate.setStatus("SEARCHING"); // Статус по умолчанию

                candidateRepository.save(candidate);
            }
            // Якщо це HR, ми просто створюємо User-а (у HR немає окремої
таблиці профілю у нашій схемі)

            return "redirect:/auth/login?success";
        }
    }
```

А.11 Текст файлу CandidateController.java

```
package com.diploma.recruitment.controller;

import com.diploma.recruitment.entity.Candidate;
import com.diploma.recruitment.entity.Position;
import com.diploma.recruitment.repository.CandidateRepository;
import com.diploma.recruitment.repository.PositionRepository;
import com.diploma.recruitment.repository.JobPositionRepository;
import org.springframework.data.domain.Sort;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;
```

```
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.security.Principal;
import java.time.LocalDateTime;

@Controller
@RequestMapping("/candidate")
@RequiredArgsConstructor
public class CandidateController {

    private final CandidateRepository candidateRepository;
    private final PositionRepository positionRepository; // 1. Добавили
репозиторий позиций
    private final JobPositionRepository jobPositionRepository;

    // 2. ДАШБОРД КАНДИДАТУ
    @GetMapping("/dashboard")
    public String dashboard(Model model, Principal principal) {
        String username = principal.getName();
        Candidate candidate =
candidateRepository.findByUserUsername(username)
                .orElseThrow(() -> new RuntimeException("Candidate not
found"));

        model.addAttribute("candidate", candidate);

        // список
        model.addAttribute("jobs",
jobPositionRepository.findAll(Sort.by(Sort.Direction.DESC, "createdAt")));

        return "candidate-dashboard";
    }
    // 3. Функція відгуку на вакансію
    @PostMapping("/apply/{jobId}")
    public String applyForJob(@PathVariable Long jobId,
RedirectAttributes redirectAttributes) {

        redirectAttributes.addFlashAttribute("successMessage", "You have
successfully applied for this position! HR will contact you soon.");

        return "redirect:/candidate/dashboard";
    }
```

```
        // 1. Показати сторінку профіля
        @GetMapping("/profile")
        public String profile(Model model, Principal principal) {
            String username = principal.getName();
            Candidate candidate =
candidateRepository.findByUserUsername(username)
                    .orElseThrow(() -> new RuntimeException("Candidate profile
not found for user: " + username));

            model.addAttribute("candidate", candidate);

            // 2. Додали список
            model.addAttribute("allPositions", positionRepository.findAll());

            return "candidate-profile";
        }

        // 2. Зберігаємо зміни
        @PostMapping("/profile/update")
        public String updateProfile(@ModelAttribute Candidate
updatedCandidate,
                                    @RequestParam(required = false) Long positionId,
                                    Principal principal) {

            String username = principal.getName();
            Candidate existingCandidate =
candidateRepository.findByUserUsername(username)
                    .orElseThrow(() -> new RuntimeException("Profile not found"));

            existingCandidate.setFirstName(updatedCandidate.getFirstName());
            existingCandidate.setLastName(updatedCandidate.getLastName());
            existingCandidate.setPatronymic(updatedCandidate.getPatronymic());
            existingCandidate.setPhone(updatedCandidate.getPhone());
            existingCandidate.setCity(updatedCandidate.getCity());
            existingCandidate.setAge(updatedCandidate.getAge());
            existingCandidate.setEducation(updatedCandidate.getEducation());

existingCandidate.setExperienceYears(updatedCandidate.getExperienceYears());

existingCandidate.setWorkExperienceDescription(updatedCandidate.getWorkExpe
rienceDescription());
            existingCandidate.setAboutMe(updatedCandidate.getAboutMe());

            if (positionId != null) {
```

```
                Position position = positionRepository.findById(positionId)
                        .orElseThrow(() -> new RuntimeException("Position not
found"));
                existingCandidate.setPosition(position);
            }

            // оновлюємо дату
            existingCandidate.setLastUpdatedAt(LocalDateTime.now());

            candidateRepository.save(existingCandidate);

            return "redirect:/candidate/profile?success";
        }
    }
```

А.12 Текст файлу CandidateSkillController.java

```
package com.diploma.recruitment.controller;

import com.diploma.recruitment.entity.*;
import com.diploma.recruitment.repository.*;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.security.Principal;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

@Controller
@RequestMapping("/candidate/skills")
@RequiredArgsConstructor
public class CandidateSkillController {

    private final CandidateRepository candidateRepository;
    private final SkillRepository skillRepository;
    private final CandidateSkillRepository candidateSkillRepository;

    @GetMapping
    public String showSkillsPage(Model model, Principal principal) {
        Candidate candidate = getCandidate(principal);
        List<CandidateSkill> mySkills =
candidateSkillRepository.findByCandidate(candidate);
```

```
            // 1. Покуш англійского
            Optional<CandidateSkill> myEnglish = mySkills.stream()
                    .filter(s -> s.getSkill().getName().equalsIgnoreCase("English"))
                    .findFirst();

            // рівень анлійської
            model.addAttribute("englishLevel",
myEnglish.map(CandidateSkill::getProficiencyLevel).orElse(null));

            // 2. Список навичок
            List<CandidateSkill> otherSkills = mySkills.stream()
                    .filter(s -> !s.getSkill().getName().equalsIgnoreCase("English"))
                    .collect(Collectors.toList());
            model.addAttribute("mySkills", otherSkills);

            // 3. Список усіх кандидатів
            List<Skill> allSkills = skillRepository.findAll().stream()
                    .filter(s -> !s.getName().equalsIgnoreCase("English"))
                    .collect(Collectors.toList());
            model.addAttribute("allSkills", allSkills);

            return "candidate-skills";
        }

        // Метод додавання навичок
        @PostMapping("/add")
        public String addSkill(@RequestParam(required = false) Long skillId,
                               @RequestParam(required = false) String skillName, //
Можно передати (для English)
                               @RequestParam Integer level,
                               Principal principal) {
            Candidate candidate = getCandidate(principal);
            Skill skill;

            // Пошук за ім’ям
            if (skillName != null) {
                skill = skillRepository.findByName(skillName)
                        .orElseThrow(() -> new RuntimeException("Skill not found: "
+ skillName));
            } else {
                // або шукаємо за id
                skill = skillRepository.findById(skillId)
                        .orElseThrow(() -> new RuntimeException("Skill ID not
found"));
```

```
            }

            //Зберігаєм
            saveCandidateSkill(candidate, skill, level);

            return "redirect:/candidate/skills";
        }

        @PostMapping("/delete")
        public String deleteSkill(@RequestParam Long skillId, Principal
principal) {
            Candidate candidate = getCandidate(principal);
            CandidateSkillKey key = new CandidateSkillKey();
            key.setCandidateId(candidate.getId());
            key.setSkillId(skillId);
            candidateSkillRepository.deleteById(key);
            return "redirect:/candidate/skills";
        }

        private Candidate getCandidate(Principal principal) {
            return
candidateRepository.findByUserUsername(principal.getName())
                    .orElseThrow(() -> new RuntimeException("Candidate not
found"));
        }

        private void saveCandidateSkill(Candidate candidate, Skill skill, Integer
level) {
            CandidateSkill cs = new CandidateSkill();
            CandidateSkillKey key = new CandidateSkillKey();
            key.setCandidateId(candidate.getId());
            key.setSkillId(skill.getId());

            cs.setId(key);
            cs.setCandidate(candidate);
            cs.setSkill(skill);
            cs.setProficiencyLevel(level);

            candidateSkillRepository.save(cs);
        }
    }
```

А.13 Текст файлу HrCandidateController.java

```
package com.diploma.recruitment.controller;
```

```
import com.diploma.recruitment.entity.Candidate;
import com.diploma.recruitment.entity.Team;
import com.diploma.recruitment.repository.CandidateRepository;
import com.diploma.recruitment.repository.CandidateSkillRepository;
import com.diploma.recruitment.repository.TeamRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

@Controller
@RequestMapping("/hr/candidates")
@RequiredArgsConstructor
public class HrCandidateController {

    private final CandidateRepository candidateRepository;
    private final CandidateSkillRepository candidateSkillRepository;
    private final TeamRepository teamRepository;

    @GetMapping("/{id}")
    public String viewCandidateProfile(@PathVariable Long id, Model
model) {
        Candidate candidate = candidateRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Candidate not
found"));

        model.addAttribute("candidate", candidate);
        model.addAttribute("skills",
candidateSkillRepository.findByCandidate(candidate));
        model.addAttribute("teams", teamRepository.findAll());

        return "hr-candidate-details";
    }

    // Найняти до команди
    @PostMapping("/{id}/hire")
    public String hireToCompany(@PathVariable Long id) {
        Candidate candidate =
candidateRepository.findById(id).orElseThrow();
        candidate.setStatus("EMPLOYED");
        candidateRepository.save(candidate);
        return "redirect:/hr/candidates/" + id;
    }
```

```
        // Назначити команду
        @PostMapping("/{id}/assign-team")
        public String assignTeam(@PathVariable Long id, @RequestParam
Long teamId) {
            Candidate candidate =
candidateRepository.findById(id).orElseThrow();

            if (teamId == -1) {
               // логіка без команди
               candidate.setTeam(null);
            } else {
               Team team = teamRepository.findById(teamId).orElseThrow();
               candidate.setTeam(team);
            }
            candidate.setStatus("EMPLOYED");

            candidateRepository.save(candidate);
            // повертаємо на стрінку профілю
            return "redirect:/hr/candidates/" + id;
        }

        // НОВиЙ МЕТОД: Звільнення
        @PostMapping("/{id}/fire")
        public String fireCandidate(@PathVariable Long id) {
            Candidate candidate =
candidateRepository.findById(id).orElseThrow();

            // оновлення статусу
            candidate.setStatus("SEARCHING");
            // видалення з команди
            candidate.setTeam(null);

            candidateRepository.save(candidate);
            return "redirect:/hr/candidates/" + id;
        }
    }
```

А.14 Текст файлу HrController.java

```
package com.diploma.recruitment.controller;

import com.diploma.recruitment.repository.CandidateRepository;
import com.diploma.recruitment.repository.JobPositionRepository;
import com.diploma.recruitment.repository.TeamRepository;
```

```
import com.diploma.recruitment.repository.SkillRepository; // Не забудь
импорт
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
@RequestMapping("/hr")
@RequiredArgsConstructor
public class HrController {

    private final CandidateRepository candidateRepository;
    private final TeamRepository teamRepository;
    private final JobPositionRepository jobPositionRepository;
    private final SkillRepository skillRepository;

    //ВАКАНСІІ
    @GetMapping("/dashboard")
    public String vacancies(Model model,
                            @RequestParam(required = false, defaultValue = "desc")
String sort) {

        Sort sorting = sort.equals("asc")
                ? Sort.by(Sort.Direction.ASC, "createdAt")
                : Sort.by(Sort.Direction.DESC, "createdAt");

        model.addAttribute("jobs", jobPositionRepository.findAll(sorting));

        model.addAttribute("currentSort", sort);

        return "hr-vacancies";
    }

    // 2. СПИСОК КАНДИДАТІВ
    @GetMapping("/candidates-list")
    public String candidates(Model model,
                             @RequestParam(required = false, defaultValue =
"lastUpdatedAt") String sortField,
                             @RequestParam(required = false, defaultValue = "desc")
String sortDir,
                             @RequestParam(required = false) Long skillId,
```

```
                                @RequestParam(required = false) String searchName) {

            Sort.Direction direction = sortDir.equals("asc") ? Sort.Direction.ASC :
Sort.Direction.DESC;
            Sort sort = Sort.by(direction, sortField);

            String searchPattern = null;
            if (searchName != null && !searchName.isBlank()) {
               searchPattern = "%" + searchName.trim() + "%";
            }

            model.addAttribute("candidates",
candidateRepository.search(searchPattern, skillId, sort));


            model.addAttribute("allSkills", skillRepository.findAll());
            model.addAttribute("selectedSkillId", skillId);
            model.addAttribute("searchName", searchName);

            model.addAttribute("sortField", sortField);
            model.addAttribute("sortDir", sortDir);
            model.addAttribute("reverseSortDir", sortDir.equals("asc") ? "desc" :
"asc");

            return "hr-candidates-list";
         }

         // 3. СПИСОК КОМАНД
         @GetMapping("/teams-list")
         public String teams(Model model) {
            model.addAttribute("teams", teamRepository.findAll());
            return "hr-teams-list";
         }
      }
```

А.15 Текст файлу HrJobController.java

```
package com.diploma.recruitment.controller;

import com.diploma.recruitment.entity.JobPosition;
import com.diploma.recruitment.entity.JobRequirement;
import com.diploma.recruitment.entity.Skill;
import com.diploma.recruitment.repository.JobPositionRepository;
import com.diploma.recruitment.repository.JobRequirementRepository;
import com.diploma.recruitment.repository.SkillRepository;
```

```
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@Controller
@RequestMapping("/hr/jobs")
@RequiredArgsConstructor
public class HrJobController {

    private final JobPositionRepository jobPositionRepository;
    private final SkillRepository skillRepository;
    private final JobRequirementRepository jobRequirementRepository;

    // --- Управління вакансіями ---

    @GetMapping("/create")
    public String createJobForm(Model model) {
        model.addAttribute("job", new JobPosition());
        return "hr-job-create";
    }

    @PostMapping("/create")
    public String createJob(JobPosition jobPosition) {
        jobPositionRepository.save(jobPosition);
        return "redirect:/hr/dashboard";
    }

    @PostMapping("/delete")
    public String deleteJob(@RequestParam Long id) {
        jobPositionRepository.deleteById(id);
        return "redirect:/hr/dashboard";
    }

    // --- Управління вимогами(Алгоритм) ---

    // 1. Сторінка вимог
    @GetMapping("/{id}/requirements")
    public String requirementsPage(@PathVariable Long id, Model model)
{
        JobPosition job = jobPositionRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Job not found"));
```

```
        List<JobRequirement> requirements =
jobRequirementRepository.findByJobPositionId(id);
        List<Skill> allSkills = skillRepository.findAll();

        model.addAttribute("job", job);
        model.addAttribute("requirements", requirements);
        model.addAttribute("allSkills", allSkills);

        return "hr-job-requirements";
    }

    // 2. Додати вимогу
    @PostMapping("/{id}/requirements/add")
    public String addRequirement(@PathVariable Long id,
                                 @RequestParam Long skillId,
                                 @RequestParam Double weight,
                                 @RequestParam Integer minLevel) {
        JobPosition job = jobPositionRepository.findById(id).orElseThrow();
        Skill skill = skillRepository.findById(skillId).orElseThrow();

        JobRequirement req = new JobRequirement();
        req.setJobPosition(job);
        req.setSkill(skill);
        req.setWeight(weight);     // Важность (0.1 - 1.0)
        req.setMinLevel(minLevel); // Минимальный уровень (1-5)

        jobRequirementRepository.save(req);

        return "redirect:/hr/jobs/" + id + "/requirements";
    }

    // 3. Видалити вимогу
    @PostMapping("/{jobId}/requirements/delete")
    public String deleteRequirement(@PathVariable Long jobId,
@RequestParam Long reqId) {
        jobRequirementRepository.deleteById(reqId);
        return "redirect:/hr/jobs/" + jobId + "/requirements";
    }
}
```

А.16 Текст файлу HrMatchController.java

```
package com.diploma.recruitment.controller;

import com.diploma.recruitment.dto.CandidateMatchDTO;
```

```
import com.diploma.recruitment.entity.JobPosition;
import com.diploma.recruitment.repository.JobPositionRepository;
import com.diploma.recruitment.service.MatchingService;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;

import java.util.List;

@Controller
@RequestMapping("/hr/match")
@RequiredArgsConstructor
public class HrMatchController {

    private final MatchingService matchingService;
    private final JobPositionRepository jobPositionRepository;

    @GetMapping("/{jobId}")
    public String findCandidates(@PathVariable Long jobId, Model model)
{
        // 1. Сервіс з алгоритмом
        List<CandidateMatchDTO> matches =
matchingService.findMatchesForJob(jobId);

        // 2. Беремо інфу о вакансіі
        JobPosition job =
jobPositionRepository.findById(jobId).orElseThrow();

        model.addAttribute("matches", matches);
        model.addAttribute("job", job);

        return "hr-match-results";
    }
}
```

А.17 Текст файлу HrTeamController.java

```
package com.diploma.recruitment.controller;

import com.diploma.recruitment.entity.Team;
import com.diploma.recruitment.repository.CandidateRepository;
import com.diploma.recruitment.repository.TeamRepository;
```

```
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

@Controller
@RequestMapping("/hr/teams")
@RequiredArgsConstructor
public class HrTeamController {

    private final TeamRepository teamRepository;
    private final CandidateRepository candidateRepository;

    // Перегляд составу команди
    @GetMapping("/{id}")
    public String viewTeamDetails(@PathVariable Long id, Model model) {
        Team team = teamRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Team not found"));

        model.addAttribute("team", team);

        // список учасників
        model.addAttribute("members",
candidateRepository.findByTeamId(id));

        return "hr-team-details";
    }

    // 1. Форма створення
    @GetMapping("/create")
    public String createTeamForm(Model model) {
        model.addAttribute("team", new Team());
        return "hr-team-create";
    }

    // 2. Зберегти нову команду
    @PostMapping("/create")
    public String createTeam(Team team) {

        teamRepository.save(team);
        return "redirect:/hr/teams-list"; // до списку
    }

    // 3. Видалити з команди
    @PostMapping("/{teamId}/remove-member")
```

```
    public String removeMember(@PathVariable Long teamId,
@RequestParam Long candidateId) {
        var candidate = candidateRepository.findById(candidateId)
            .orElseThrow(() -> new RuntimeException("Candidate not
found"));
        candidate.setTeam(null);

        candidateRepository.save(candidate);

        return "redirect:/hr/teams/" + teamId;
    }

    // 4. Распустить (удалить) команду
    @PostMapping("/delete")
    public String deleteTeam(@RequestParam Long teamId) {
        var members = candidateRepository.findByTeamId(teamId);

        for (var member : members) {
            member.setTeam(null);
            // member.setStatus("EMPLOYED");
            candidateRepository.save(member);
        }

        // Теперь, когда команда пуста, удаляем её
        teamRepository.deleteById(teamId);

        return "redirect:/hr/teams-list";
    }
}
```

А.18 Текст файлу CustomUserDetailsService.java

```
package com.diploma.recruitment.security;

import com.diploma.recruitment.entity.User;
import com.diploma.recruitment.repository.UserRepository;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

@Service
public class CustomUserDetailsService implements UserDetailsService {
```

```
    private final UserRepository userRepository;

    public CustomUserDetailsService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        // 1. Шукаємо користувача в БД
        User user = userRepository.findByUsername(username)
                .orElseThrow(() -> new
UsernameNotFoundException("Користувач не знайден: " + username));

        // 2. Перетворення User в UserDetails ( Spring Security)
        return org.springframework.security.core.userdetails.User.builder()
                .username(user.getUsername())
                .password(user.getPassword()) // Тут лежит зашифрованный
пароль
                .roles(user.getRole().replace("ROLE_", "")) // Spring ждет роль
без префикса "ROLE_" (например, просто "HR")
                .build();
    }
}
```

А.19 Текст файлу SecurityConfig.java

```
package com.diploma.recruitment.security;

import com.diploma.recruitment.security.CustomUserDetailsService;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.annotation.authentication.configuration.Authe
nticationConfiguration;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSec
urity;
```

```
import
org.springframework.security.config.annotation.web.configurers.AbstractHttpConf
igurer;
import
org.springframework.security.crypto.password.NoOpPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.AuthenticationSuccessHandler;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

  private final CustomUserDetailsService userDetailsService;

  public SecurityConfig(CustomUserDetailsService userDetailsService) {
    this.userDetailsService = userDetailsService;
  }

  @Bean
  public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
    http
        .csrf(AbstractHttpConfigurer::disable)

        .authorizeHttpRequests(auth -> auth
            .requestMatchers("/css/**", "/js/**", "/images/**",
"/auth/**").permitAll()

            .requestMatchers("/hr/**").hasRole("HR")

.requestMatchers("/candidate/**").hasRole("CANDIDATE")
            .anyRequest().authenticated()
        )
        .formLogin(form -> form
            .loginPage("/auth/login")
            .loginProcessingUrl("/perform_login")
            .successHandler(myAuthenticationSuccessHandler())
            .permitAll()
        )
        .logout(logout -> logout
            .logoutUrl("/logout")
            .logoutSuccessUrl("/auth/login?logout")
            .permitAll()
```

```
                );

        return http.build();
    }

    @Bean
    public DaoAuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new
DaoAuthenticationProvider(userDetailsService);

        authProvider.setPasswordEncoder(passwordEncoder());
        return authProvider;
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return NoOpPasswordEncoder.getInstance();
    }

    @Bean
    public AuthenticationManager
authenticationManager(AuthenticationConfiguration authConfig) throws
Exception {
        return authConfig.getAuthenticationManager();
    }

    @Bean
    public AuthenticationSuccessHandler
myAuthenticationSuccessHandler() {
        return (request, response, authentication) -> {
            var authorities = authentication.getAuthorities();
            String redirectUrl = "/";

            if (authorities.stream().anyMatch(a ->
a.getAuthority().equals("ROLE_HR"))) {
                redirectUrl = "/hr/dashboard";
            } else if (authorities.stream().anyMatch(a ->
a.getAuthority().equals("ROLE_CANDIDATE"))) {
                redirectUrl = "/candidate/dashboard";
            }

            response.sendRedirect(redirectUrl);
        };
    }
}
```

ДОДАТОК Б

Слайди презентації

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

Кафедра програмних засобів

Спеціальність: 121 «Інженерія програмного забезпечення»

Дипломна кваліфікаційна робота магістра

на тему: Дослідження та розробка програмного забезпечення

для автоматизації рекрутингу та відбору кандидатів до команди розробників ІТ проєктів

Виконав:
студент групи КНТ-124м
Олексій ПІСКУН
Керівник роботи:
к.т.н., професор
Галина ТАБУНЩИК

Запоріжжя 2025

Рисунок Б.1 – Слайд 1

- Об’єкт дослідження – процес інженерії програмного забезпечення для автоматизації рекрутингу та формування проєктних команд в ІТ-компаніях.
- Предмет дослідження – методи та програмні засоби інтелектуального підбору кандидатів на основі зваженого оцінювання навичок.
- Мета роботи – мінімізація часових витрат та зменшення трудомісткості процесів найму шляхом розробки спеціалізованої інформаційної системи, що забезпечує автоматизований пошук, ранжування та розподіл кандидатів за проєктами.
- Галузь використання – кадрові відділи (HR) ІТ-компаній, рекрутингові агентства, департаменти управління персоналом.

Слайд 2/26

Рисунок Б.2 – Слайд 2

Актуальність теми

Незважаючи на високий рівень цифровізації самої IT-галузі, процеси найму в багатьох компаніях малого та середнього бізнесу в Україні залишаються недостатньо автоматизованими. Значна частина рекрутерів продовжує використовувати неспеціалізовані інструменти, такі як електронна пошта та електронні таблиці. Такий підхід має суттєві недоліки:

1) фрагментація даних;

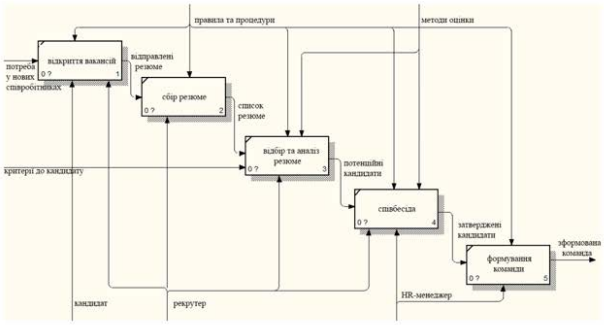
2) низька швидкість обробки;

3) відсутність командної взаємодії.

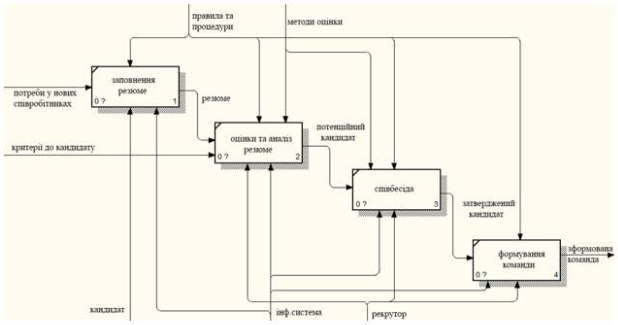
Слайд 3/26

Рисунок Б.3 – Слайд 3

Досліждення бізнес-процесу



Контекстна діаграма моделі «AS-IS»

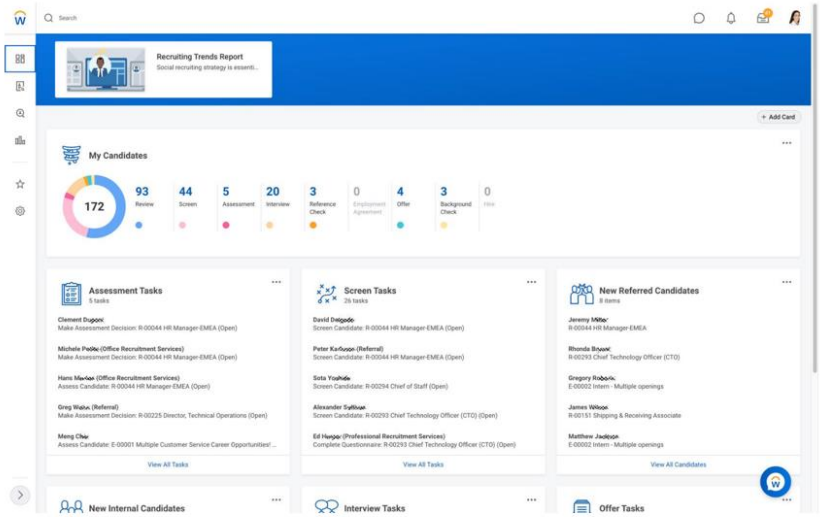


Контекстна діаграма моделі «TO-BE»

Слайд 4/26

Рисунок Б.4 – Слайд 4

Аналіз аналогів

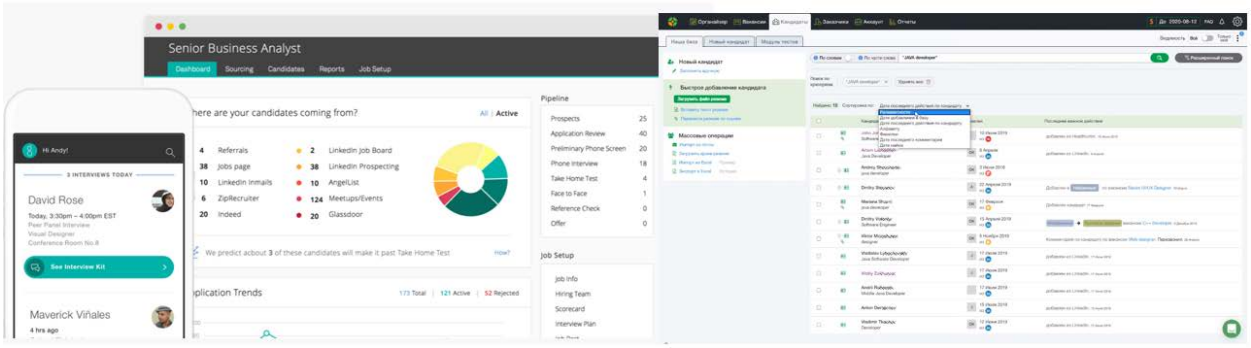


Інтерфейс системи "Workday HCM"

Слайд 5/26

Рисунок Б.5 – Слайд 5

Аналіз аналогів



Застосунок "Greenhouse"

Система "CleverStaff"

Слайд 6/26

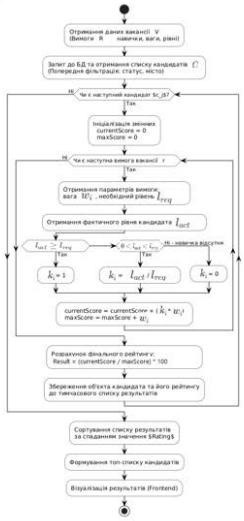
Рисунок Б.6 – Слайд 6

Алгоритмізація підбору кандидатів

$$R_v = \{(s_i, w_i, l_{req}) \mid s_i \in S\},$$

$$k_i = \begin{cases} 1, & \text{якщо } l_{act} \geq l_{req} \\ \frac{l_{act}}{l_{req}}, & \text{якщо } 0 < l_{act} < l_{req} \\ 0, & \text{якщо } l_{act} = 0 \end{cases}$$

$$Rating(c_j, V) = \frac{\sum_{i=1}^{N}(k_i \cdot w_i)}{\sum_{i=1}^{N} w_i} \cdot 100\%$$

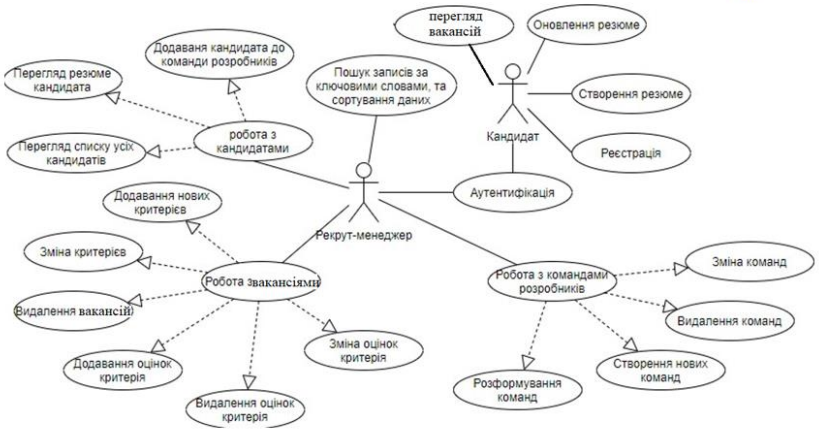


Слайд 7/26

Рисунок Б.7 – Слайд 7

Визначення фунціональних вимог

- HR-менеджер (HR Manager) – це основний користувач системи, відповідальний за процеси найму. Його права доступу це повний доступ до модулів управління вакансіями, базою кандидатів та командами. А ключовими задачами є створення вакансій з вагами навичок, запуск алгоритму підбору, формування складу проєктних команд.
- Кандидат (Candidate) – користувач, який шукає роботу, його ключовими задачами є реєстрація, заповнення профілю (персональні дані, навички), моніторинг статусу та можливість відгуку на вакансію.



Діаграма варіантів використання системи (Use Case Diagram)

Слайд 8/26

Рисунок Б.8 – Слайд 8

Діаграма станів вакансії

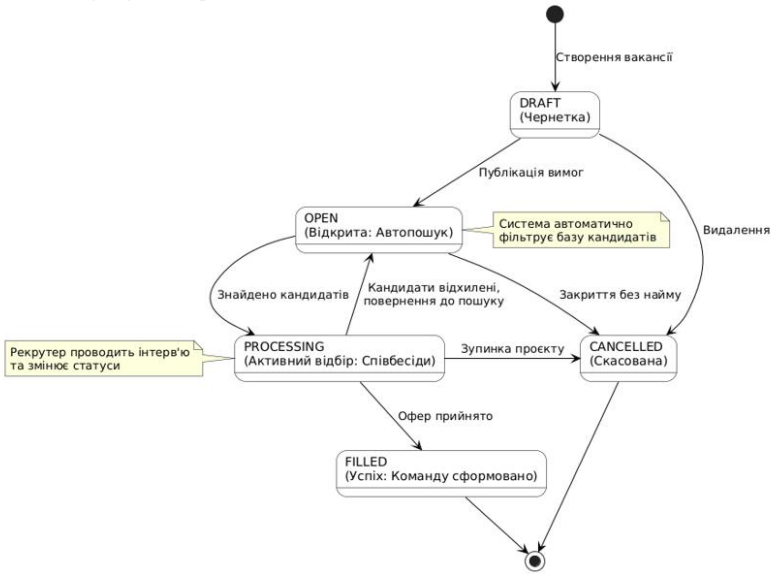


Рисунок Б.9 – Слайд 9

Обраний стек технологій

- Java Enterprise Edition – надійна мова програмування, обрана для реалізації серверної частини та бізнес-логіки системи завдяки її стабільності та суворій типізації.
- Spring Boot – комплексний фреймворк, що забезпечує архітектуру MVC, механізми безпеки (Security) та швидку взаємодію з даними (Data JPA).
- PostgreSQL – потужна реляційна система управління базами даних, що гарантує цілісність зв'язків та високу швидкість виконання складних алгоритмічних запитів.
- Thymeleaf – серверний шаблонізатор Java, який використовується для генерації динамічного та адаптивного веб-інтерфейсу користувача (HTML5).

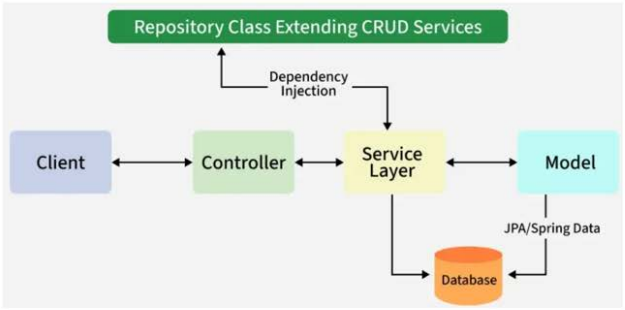


Рисунок Б.10 – Слайд 10

Модель бази даних

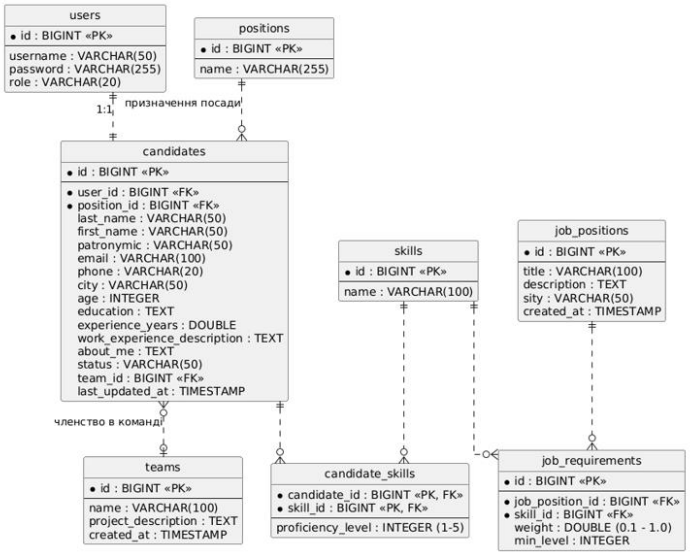


Рисунок Б.11 – Слайд 11

Діаграма класів сутностей та архітектура проєкту

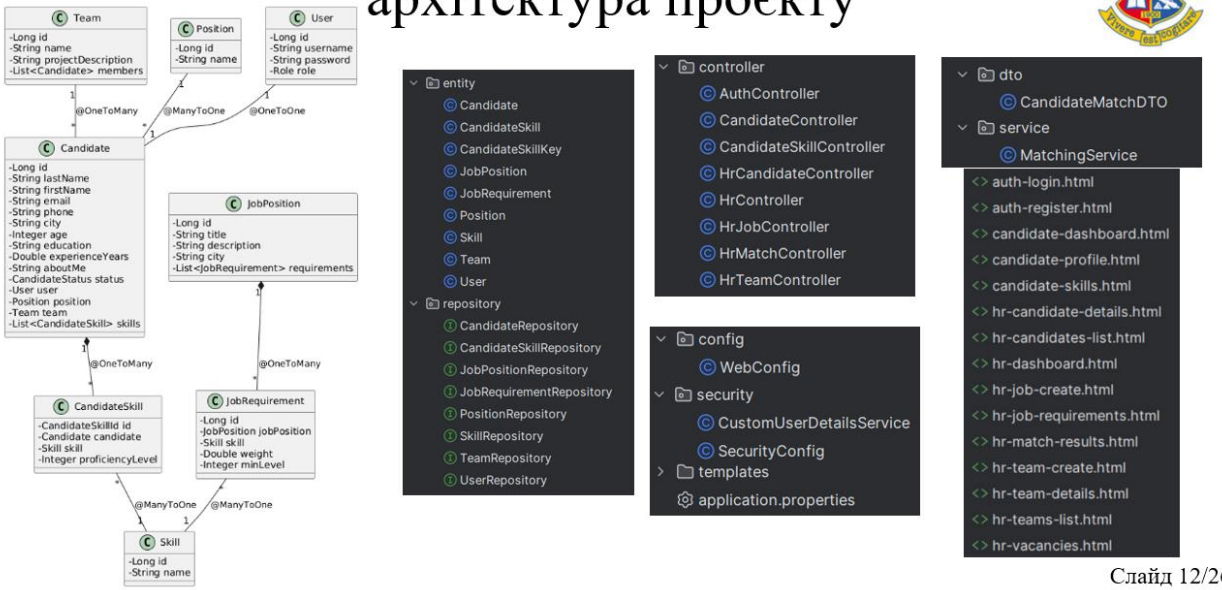
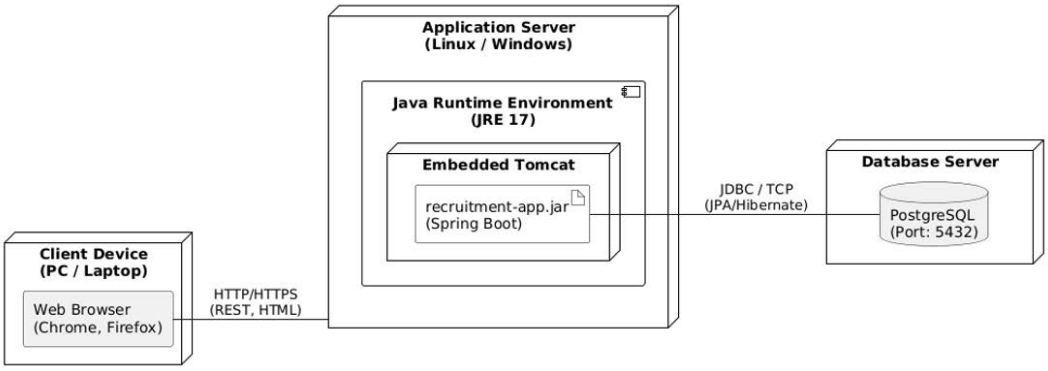


Рисунок Б.12 – Слайд 12

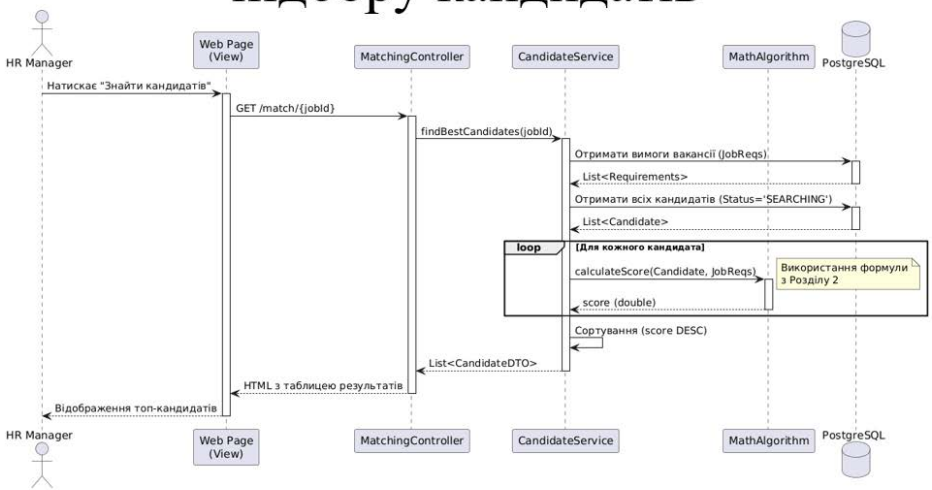
Діаграма розгортання



Слайд 13/26

Рисунок Б.13 – Слайд 13

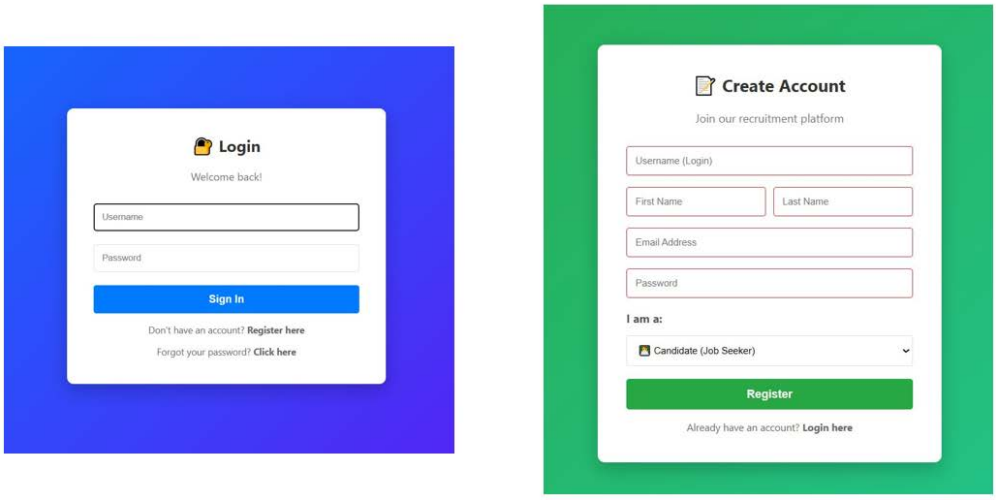
Діаграма послідовності підбору кандидатів



Слайд 14/26

Рисунок Б.14 – Слайд 14

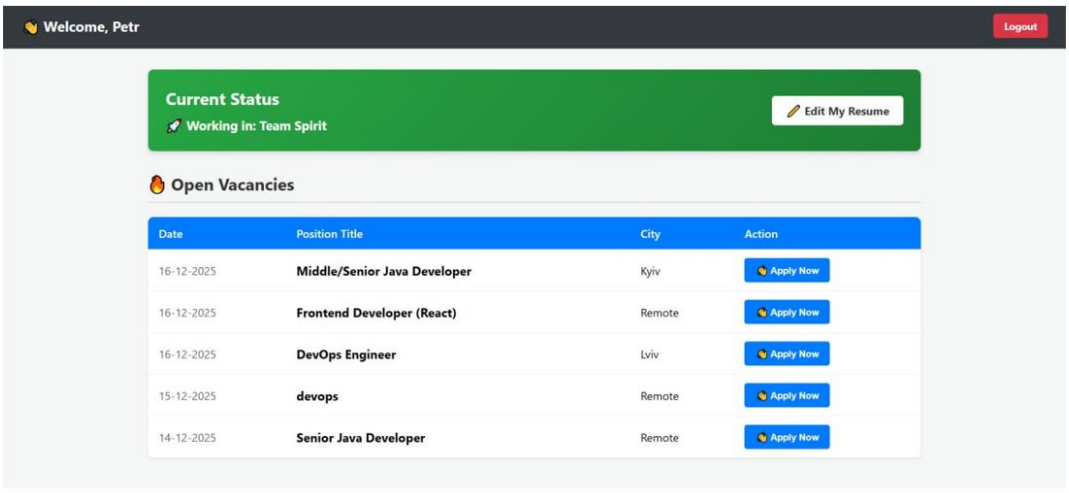
Сторінки авторизації та реєстрації



Слайд 15/26

Рисунок Б.15 – Слайд 15

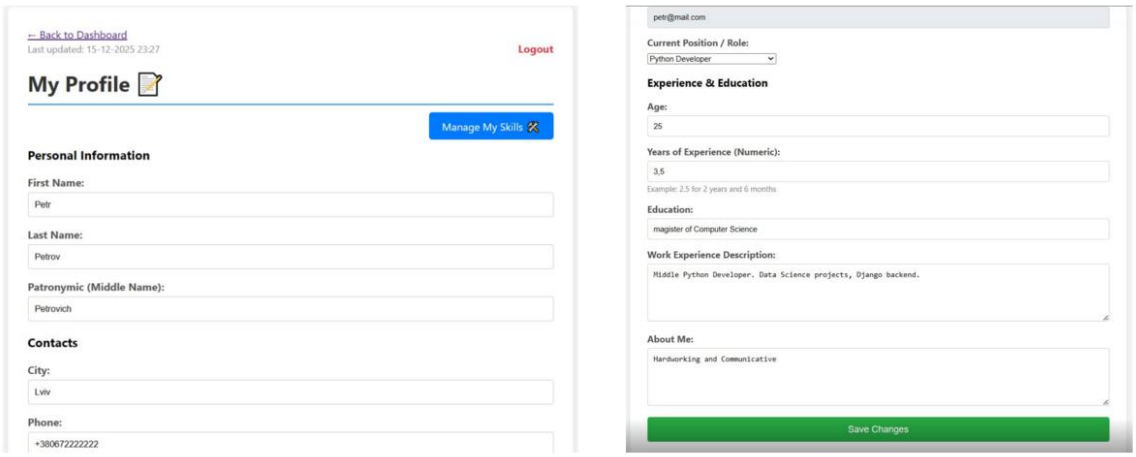
Головне меню користувача



Слайд 16/26

Рисунок Б.16 – Слайд 16

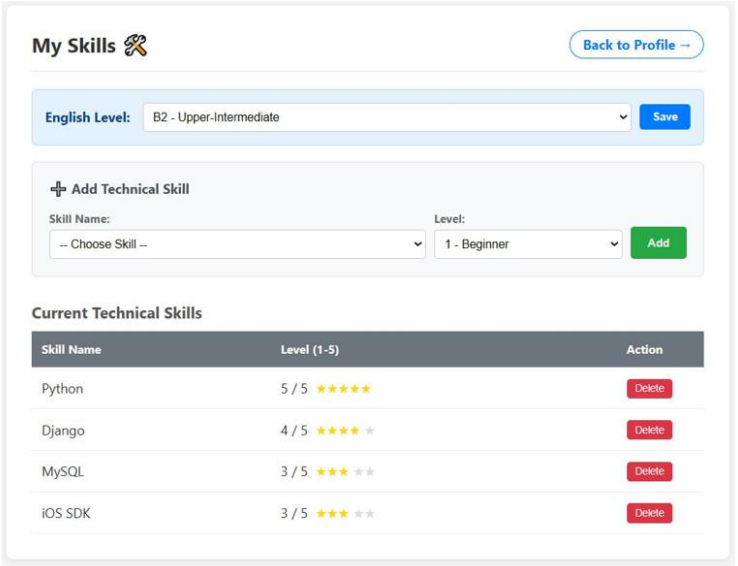
Налаштування профілю (резюме)



Слайд 17/26

Рисунок Б.17 – Слайд 17

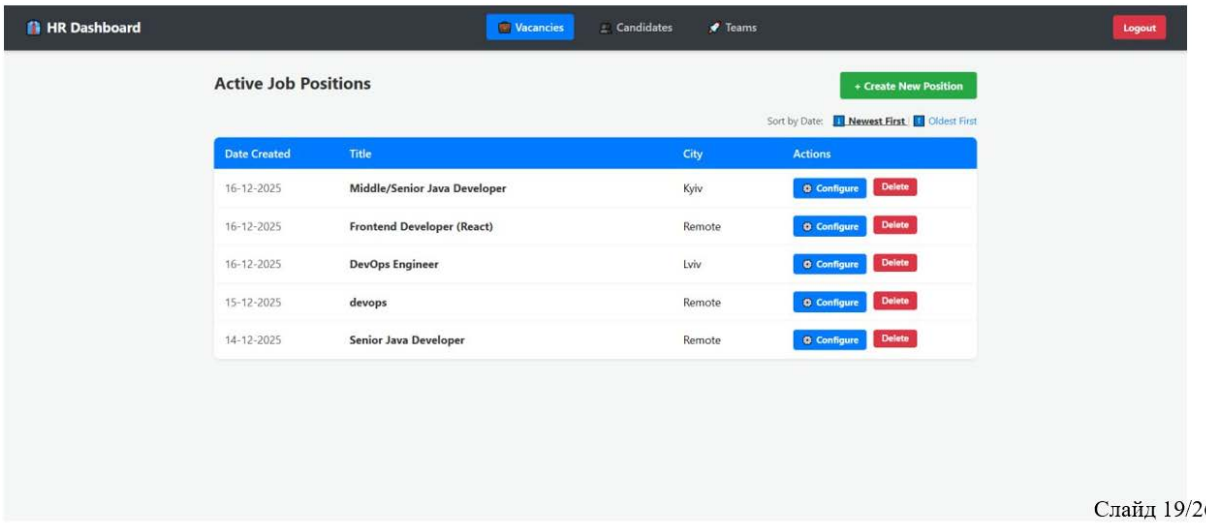
Налаштування профілю (резюме)



Слайд 18/26

Рисунок Б.18 – Слайд 18

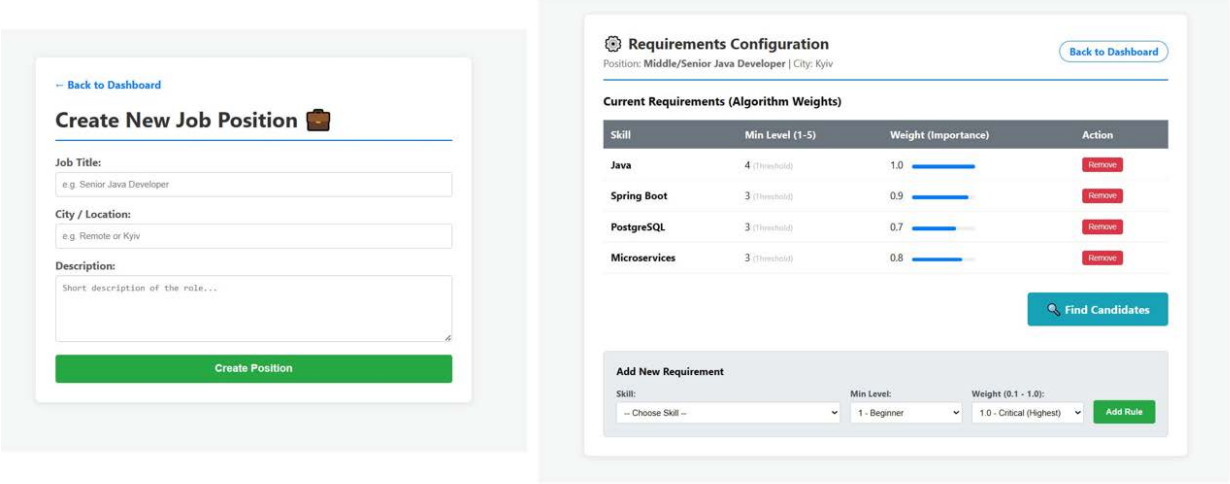
Головне меню рекрутера



Слайд 19/26

Рисунок Б.19 – Слайд 19

Робота с вакансіями



Слайд 20/26

Рисунок Б.20 – Слайд 20

Результат роботи алгоритма підбору

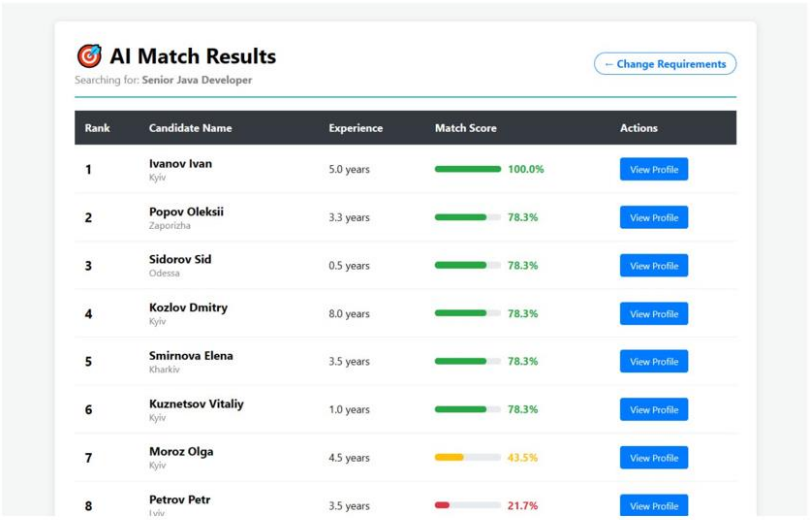


Рисунок Б.21 – Слайд 21

Зручний список кандидатів

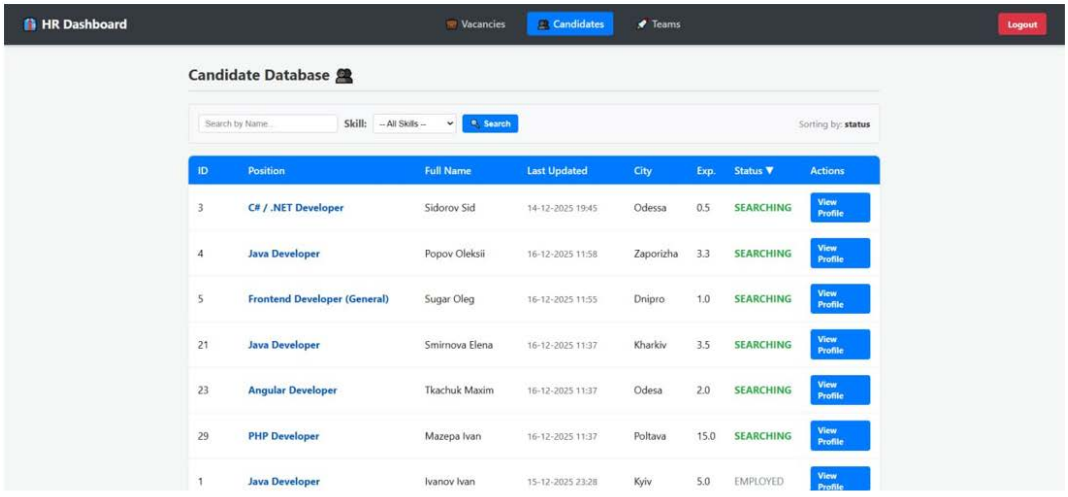
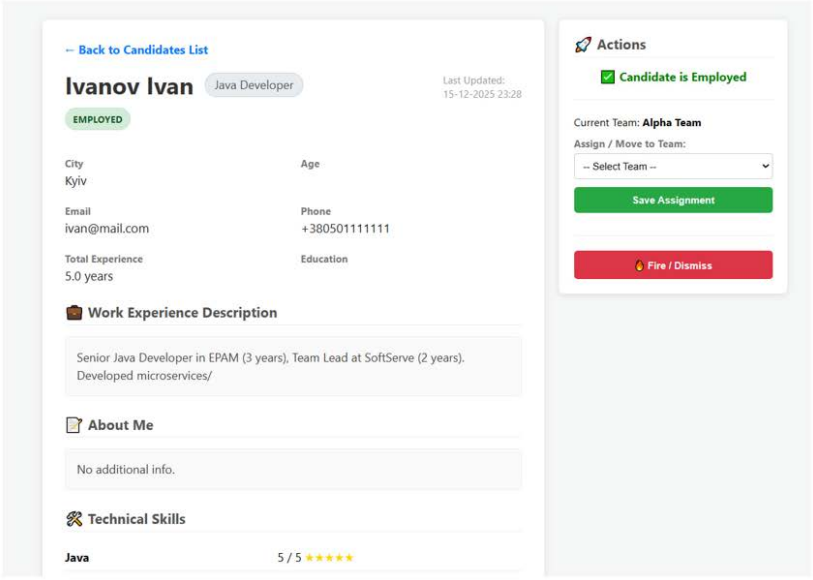


Рисунок Б.22 – Слайд 22

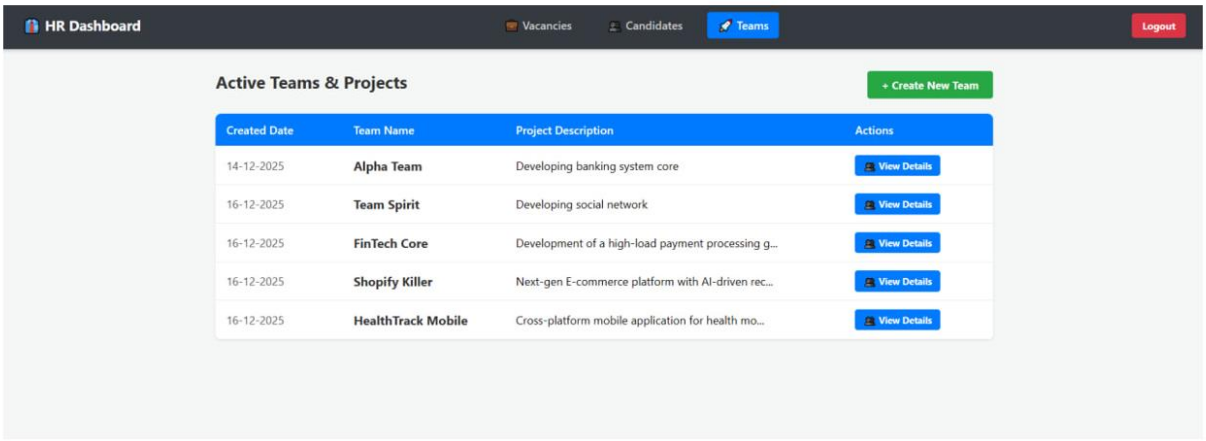
Перегляд профілей кандидатів



Слайд 23/26

Рисунок Б.23 – Слайд 23

Вікно списку команд розробників



Слайд 24/26

Рисунок Б.24 – Слайд 24

Функціонал роботи з командами

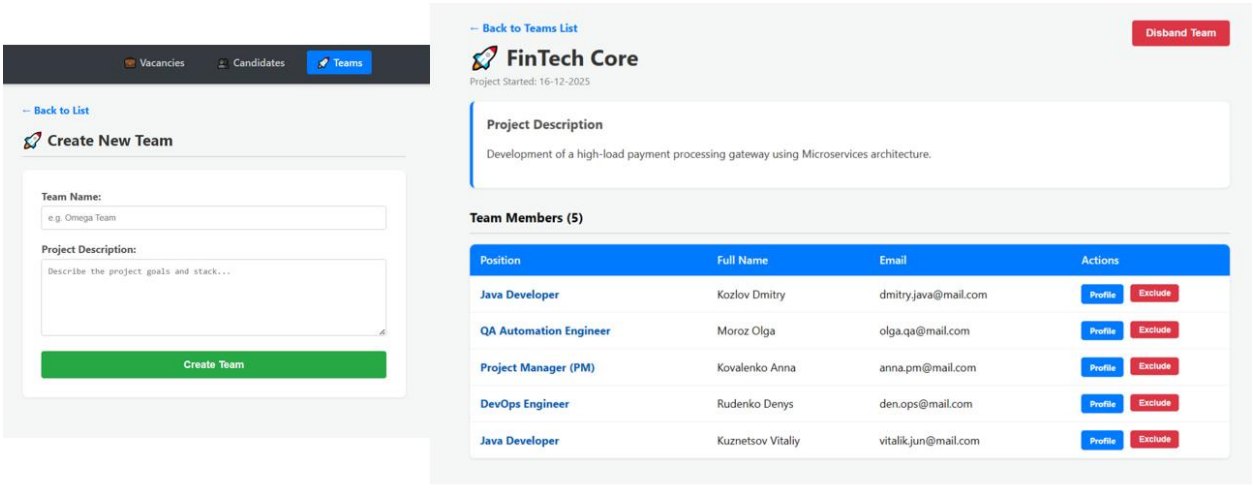


Рисунок Б.25 – Слайд 25

Висновки

- У дипломній роботі вирішено задачу мінімізації часових витрат та зменшенні трудомісткості процесів рекрутингу шляхом створення програмного забезпечення для підбору та оцінки кандидатів, тож мета роботи була досягнута в повному обсязі.
- Практична цінність роботи полягає у створенні повноцінного програмного продукту, готового до впровадження в IT-компаніях. Система надає можливість не лише автоматизувати рутинні операції, а й забезпечити відбор якісного персоналу у короткий проміжок часу завдяки об'єктивній математичній оцінці компетенцій, що в кінцевому підсумку сприяє формуванню більш ефективних команд розробників.
- Перспективи подальшого розвитку системи вбачаються у впровадженні засобів штучного інтелекту для автоматичної обробки файлів резюме та переході до мікросервісної архітектури для забезпечення масштабування у великих корпораціях.

Рисунок Б.26 – Слайд 26