НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Артем ВОЛОКИТА
(підпис)

“__” _____2026 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

на тему: Веб-система для пошуку та підбору фахівців у сфері інформаційних технологій

Виконала : студентка __4__ курсу, групи _ІМ-21
(шифр групи)

_____Рабійчук Дар’я Олександрівна_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____ас. каф., доктор філософії Шульга. М.В._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) заст. зав. каф. Гончаренко О.О _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент __доц. каф. ІСТ, к.т.н. Галушко Д.О._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2026 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Артем ВОЛОКИТА
(підпис)

“__” _____2026 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Рабійчук Дар’ї Олександрівни

1. Тема проєкту *Веб-система для пошуку та підбору фахівців у сфері інформаційних технологій*
керівник проєкту Шульга Максим Володимирович, асистент кафедри, доктор філософії,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 3 червня 2026 року №2055-с

2. Термін здачі студентом закінченого проєкту *02 червня 2026 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Огляд існуючих рішень для пошуку фахівців у сфері інформаційних технологій.

Розділ 2. Огляд технологій для проектування веб-системи.

Розділ 3. Деталі розробки веб-системи.

Розділ 4. Огляд та аналіз розробленої системи.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) компоненти системи(структурна схема), діаграма класів (функціональна схема), алгоритм дій системи (принципова схема).

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Гончаренко. О.О.		

7. Дата видачі завдання «25» лютого 2026 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	26.02.2026–05.03.2026	
2.	*Вивчення та аналіз завдання*	05.03.2026–12.03.2026	
3.	*Розробка архітектури та загальної структури системи*	12.03.2026–02.04.2026	
4.	*Розробка структур окремих підсистем*	02.04.2026–16.04.2026	
5.	*Програмна реалізація системи*	16.04.2026–30.04.2026	
6.	*Оформлення пояснювальної записки*	30.04.2026–07.05.2026	
7.	*Захист програмного продукту*	14.05.2026	
8.	*Передзахист*	23.05.2026	
9.	*Захист*	16.06.2026	

Студент-дипломник _____ Дар'я РАБІЙЧУК
(підпис)

Керівник проєкту _____ Максим ШУЛЬГА
(підпис)

АНОТАЦІЯ

У даній роботі проведено огляд існуючих веб-систем для пошуку та підбору фахівців у сфері інформаційних технологій, визначено їхні недоліки. Розроблено архітектуру системи та реалізовано клієнтську і серверну частини. Запропоновано гібридний алгоритм пошуку, що поєднує модель ранжування BM25 та зважену багатокритеріальну оцінку, а також гібридну рекомендаційну систему. Веб-систему створено з використанням TypeScript, Node.js, NestJS, React, PostgreSQL та Elasticsearch. Систему призначено для застосування в компаніях, рекрутингових агенціях та кадрових службах.

Ключові слова: *веб-система, пошук фахівців, підбір фахівців, рекомендаційна система, BM25, NestJS, React, PostgreSQL, Elasticsearch.*

ANNOTATION

This work analyzes existing web systems for searching and selecting specialists in the field of information technology and identifies their drawbacks. The system architecture was designed and the client and server components were implemented. A hybrid search algorithm combining the BM25 ranking model with weighted multi-criteria scoring was proposed, along with a hybrid recommendation system. The web system was built using TypeScript, Node.js, NestJS, React, PostgreSQL, and Elasticsearch. It is intended for use at companies, recruitment agencies, and human resources departments.

Keywords: *web system, specialist search, specialist selection, recommendation system, BM25, NestJS, React, PostgreSQL, Elasticsearch.*

Довідки	Формат	Позначення	Найменування	Кільк. листів	№ екземпляра	Додаток
	А4	ІАЛЦ.467200.001 ОА	ВЕБ-СИСТЕМА ДЛЯ ПОШУКУ ТА ПІДБОРУ ФАХІВЦІВ У СФЕРІ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ Опис альбому	1		
	А4	ІАЛЦ.467200.002 ТЗ	ВЕБ-СИСТЕМА ДЛЯ ПОШУКУ ТА ПІДБОРУ ФАХІВЦІВ У СФЕРІ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ Технічне завдання	4		
	А4	ІАЛЦ.467200.003 ПЗ	ВЕБ-СИСТЕМА ДЛЯ ПОШУКУ ТА ПІДБОРУ ФАХІВЦІВ У СФЕРІ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ Пояснювальна записка	85		
	А4	ІАЛЦ.467200.004 Д1	ВЕБ-СИСТЕМА ДЛЯ ПОШУКУ ТА ПІДБОРУ ФАХІВЦІВ У СФЕРІ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ Принципова схема	2		
	А4	ІАЛЦ.467200.005 Д2	ВЕБ-СИСТЕМА ДЛЯ ПОШУКУ ТА ПІДБОРУ ФАХІВЦІВ У СФЕРІ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ Функціональна схема	2		
	А4	ІАЛЦ.467200.006 Д3	ВЕБ-СИСТЕМА ДЛЯ ПОШУКУ ТА ПІДБОРУ ФАХІВЦІВ У СФЕРІ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ Структурна схема	2		
	А4	ІАЛЦ.467200.007 Д4	ВЕБ-СИСТЕМА ДЛЯ ПОШУКУ ТА ПІДБОРУ ФАХІВЦІВ У СФЕРІ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ Текст програмного коду	7		

Зм.	Лист	№ докум.	Підп.	Дата	ІАЛЦ.467200.001 ОА		
Розробив		Рабійчук Д. О.			Веб-система для пошуку та підбору фахівців у сфері інформаційних технологій Опис альбому	Лит.	Аркуш 1 / Аркушів 1
Перевірив		Шульга М. В.					
Н. контр.		Гончаренко О. О.				НТУУ "КПІ ім. Ігоря Сікорського", ФІОТ ІМ-21	
Затвердив		Волокита А. М.					

ВЕБ-СИСТЕМА ДЛЯ ПОШУКУ ТА ПІДБОРУ ФАХІВЦІВ У СФЕРІ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Технічне завдання

ІА/ЛЦ.467200.002 ТЗ

Київ – 2026 р.

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Це технічне завдання визначає вимоги до програмного забезпечення, призначеного для автоматизації процесу пошуку та підбору фахівців у сфері інформаційних технологій, а також до подальшої підтримки і вдосконалення цього програмного забезпечення. Область застосування: підбір фахівців у сфері інформаційних технологій у компаніях, рекрутингових агенціях, кадрових службах підприємств, а також пошук вакансій фахівцями у сфері інформаційних технологій для працевлаштування.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр інженерії програмного забезпечення», який був затверджений факультетом «Інформатики та обчислювальної техніки» кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою проєкту є створення веб-системи для пошуку та підбору фахівців у сфері інформаційних технологій, яка забезпечить роботодавцям зручний інтерфейс для ефективного підбору кандидатів за заданими критеріями із застосуванням алгоритмів інформаційного пошуку та персоналізованих рекомендацій, а фахівцям у сфері інформаційних технологій — можливість пошуку відповідних вакансій.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проєкту є офіційні документації, публікації та статті в мережі Інтернет на дану тему, науково-технічна література.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має відповідати таким вимогам:

- можливість створення та керування профілями фахівців у сфері інформаційних технологій та роботодавців;
- можливість створення, публікації та керування вакансіями;
- реалізація повнотекстового пошуку фахівців з ранжуванням результатів за релевантністю;
- реалізація персоналізованих рекомендацій фахівців та вакансій.

5.2. Вимоги до програмного забезпечення

- ОС Windows, Mac чи Linux;
- встановлений веб-браузер.

5.3. Вимоги до апаратної частини

- підключення до мережі Інтернет;
- ROM не менше ніж 8 ГБ;
- RAM не менше ніж 8 ГБ.

6 ЕТАПИ РОЗРОБКИ

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання	Примітки
1	*Затвердження теми роботи*	26.02.2026–05.03.2026	
2	*Вивчення та аналіз завдання*	05.03.2026–12.03.2026	
3	*Розробка архітектури та загальної структури системи*	12.03.2026–02.04.2026	
4	*Розробка структур окремих частин системи*	02.04.2026–16.04.2026	
5	*Програмна реалізація системи*	16.04.2026–30.04.2026	
6	*Виправлення помилок*	30.04.2026–07.05.2026	
7	*Оформлення пояснювальної записки*	07.05.2026–22.05.2026	
8	*Передзахист*	23.05.2026	
9	*Захист*	16.06.2026	

ВЕБ-СИСТЕМА ДЛЯ ПОШУКУ ТА ПІДБОРУ ФАХІВЦІВ У СФЕРІ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Пояснювальна записка

ІА/ЛЦ.467200.003 ПЗ

Київ – 2026 р.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ACID	(Atomicity, Consistency, Isolation, Durability) атомарність, узгодженість, ізольованість, довговічність
API	(Application Programming Interface) прикладний програмний інтерфейс
BM25	(Best Matching 25) ймовірнісна модель ранжування документів
CRUD	(Create, Read, Update, Delete) створення, читання, оновлення, видалення
CSS	(Cascading Style Sheets) каскадні таблиці стилів
DOM	(Document Object Model) об’єктна модель документа
HMAC	(Hash-based Message Authentication Code) код автентифікації повідомлення на основі хешу
HTTP	(HyperText Transfer Protocol) протокол передачі гіпертексту
HTTPS	(HyperText Transfer Protocol Secure) захищений протокол передачі гіпертексту
IT	(Information Technology) інформаційні технології
JSON	(JavaScript Object Notation) текстовий формат обміну даними
JWT	(JSON Web Token) токен на основі JSON
MVCC	(Multi-Version Concurrency Control) контроль конкурентності на основі багатоверсійності
OAuth	(Open Authorization) протокол відкритої авторизації
ORM	(Object-Relational Mapping) об’єктно-реляційне відображення
OWASP	(Open Web Application Security Project) проєкт безпеки відкритих веб-застосунків
PDF	(Portable Document Format) формат переносних документів
REST	(Representational State Transfer) передача стану представлення

SHA	(Secure Hash Algorithm) безпечний алгоритм хешування
SPA	(Single Page Application) односторінковий застосунок
SQL	(Structured Query Language) мова структурованих запитів
TF-IDF	(Term Frequency – Inverse Document Frequency) частота терма — обернена частота документа
TLS	(Transport Layer Security) протокол захисту транспортного рівня
URI	(Uniform Resource Identifier) уніфікований ідентифікатор ресурсу
YAML	(YAML Ain’t Markup Language) мова серіалізації даних
СУБД	система управління базами даних

ВСТУП

Дипломна робота присвячена розробці веб-системи для автоматизованого пошуку і підбору фахівців у сфері інформаційних технологій. Галузь IT залишається однією з найбільш динамічних в економіці України, тому ефективність процесів зіставлення кандидатів і вакансій безпосередньо впливає на швидкість укомплектування команд і витрати компаній на кадрові процеси. Аналіз провідних платформ показує, що жодна з них не поєднує прозорого алгоритмічного ранжування, налаштовуваного багатокритеріального зіставлення та персоналізованих рекомендацій з урахуванням специфіки галузі.

Метою роботи є створення веб-системи, яка вдосконалює процес підбору IT-фахівців за рахунок поєднання повнотекстового ранжування за моделлю BM25 із зваженою багатокритеріальною оцінкою відповідності та гібридної рекомендаційної системи.

Для досягнення поставленої мети у роботі вирішуються такі завдання:

1) проаналізувати провідні платформи підбору IT-фахівців та виявити їхні алгоритмічні обмеження;
2) сформулювати функціональні й нефункціональні вимоги до системи та обрати технологічний стек;
3) розробити архітектуру та програмно реалізувати клієнтську і серверну частини;
4) провести функціональне тестування та оцінити ефективність запропонованих алгоритмів.

Об’єктом дослідження є процес зіставлення кандидатів і вакансій у сфері IT. Предметом дослідження — алгоритми гібридного повнотекстового пошуку із зваженою багатокритеріальною оцінкою та гібридні рекомендаційні алгоритми.

Наукова новизна полягає у запропонованому гібридному алгоритмі

ранжування з можливістю незалежного налаштування ваг п’яти компонентів та у гібридній рекомендаційній схемі з адаптивним коефіцієнтом балансу між контент-базованою і колаборативною складовими.

Практичне значення — створення робочого прототипу системи, придатного для використання у компаніях, рекрутингових агенціях і кадрових службах підприємств. Реалізована система автоматизує найбільш трудомісткі етапи підбору, скорочує час формування списку релевантних кандидатів та забезпечує прозорість критеріїв ранжування.

Пояснювальну записку поділено на чотири розділи: огляд існуючих рішень і формування вимог; огляд технологій та теоретичних основ; деталі реалізації веб-системи; тестування та аналіз розробленого проєкту.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ДЛЯ ПОШУКУ ТА ПІДБОРУ ФАХІВЦІВ У СФЕРІ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1 Актуальність та доцільність розроблення

Сектор інформаційних технологій посідає одне з провідних місць у структурі експортно орієнтованих галузей економіки України. Згідно з результатами дослідження «IT Research Ukraine 2025: From Adaptation to Transformation», проведеного Lviv IT Cluster, обсяг галузі у 2024 році сягнув 7,48 млрд дол. США сукупного обороту і 6,45 млрд дол. США експорту, що становить 41,9 % від загальних обсягів експорту послуг і 12,3 % від експорту товарів та послуг у цілому [1]. На території країни функціонує 2 062 верифіковані IT-компанії, які забезпечують від 644 до 645 тис. прямих і непрямих робочих місць; станом на 2025 рік у галузі задіяно близько 303 тис. фахівців, з яких приблизно 245 тис. постійно проживають і працюють в Україні [1].

Попри відносну стабільність сектору, кадрові процеси у ньому стають дедалі складнішими для обох сторін ринку праці. За щорічним опитуванням ринку праці 2026, опублікованим виданням DOU, 41 % представників компаній характеризують пошук IT-спеціалістів як непростий або складний, тоді як роком раніше такої думки дотримувалися 35 % респондентів [2]. Серед основних причин ускладнення підбору роботодавці називають невідповідність професійного рівня здобувачів очікуванням (61 %), брак релевантного досвіду та невідповідність технологічного стеку. Для самих здобувачів ситуація також непроста: медіанна тривалість одного циклу пошуку нової позиції становить 11 тижнів, середня – 15 тижнів, а кількість поданих заявок під час

одного циклу досягає приблизно 57 вакансій на одного фахівця [2].

Згідно з тим самим джерелом, основними каналами пошуку роботи для IT-фахівців залишаються спеціалізовані електронні платформи: на платформу Djinni припадає близько 28 % успішних працевлаштувань, на LinkedIn – 24 %, на DOU – 19 % [3]. Це свідчить про провідну роль електронних платформ рекрутингу у формуванні ринку IT-праці й одночасно про те, що алгоритмічні та функціональні характеристики таких платформ безпосередньо впливають на швидкість та якість зіставлення кандидатів із вакансіями.

Підбір кадрів у сфері інформаційних технологій відрізняється низкою специфічних рис, які накладають додаткові вимоги на інструменти автоматизації. По-перше, технологічний стек оновлюється швидко, тому опис компетенцій кандидата і вимог вакансії потребує структурованої моделі з рівнями володіння окремими технологіями та можливістю врахування їхньої актуальності. По-друге, всередині галузі співіснує значна кількість окремих напрямів (Frontend, Backend, DevOps, Data Science, Machine Learning Engineering, QA, Mobile та інші), і кожен з них має власний набір ключових технологій та критеріїв оцінки. По-третє, оцінка відповідності кандидата вакансії за своєю природою є багатокритеріальною: одночасно враховуються технологічний стек, рівень кваліфікації, географічна локація, тип зайнятості, очікувана заробітна плата. По-четверте, час як рекрутера, так і кандидата є дефіцитним ресурсом, тож зростає роль автоматизованого ранжування результатів за релевантністю.

Сучасні платформи здебільшого вирішують перелічені задачі лише частково. У типовому сценарії роботодавець користується набором базових фільтрів за технологіями, рівнем і локацією, не маючи інформації про критерії впорядкування результатів і причини появи конкретного кандидата на верхніх позиціях видачі. Алгоритми пошуку зазвичай реалізовані як закри-

ті функції, що позбавляє користувача можливості налаштовувати їхню поведінку під специфіку власного процесу підбору. Окрім того, навіть спеціалізовані ІТ-орієнтовані платформи рідко поєднують у межах одного продукту три ключові механізми сучасного підбору: повнотекстове ранжування за релевантністю на основі ймовірнісних моделей, зважене багатокритеріальне зіставлення з налаштовуваними коефіцієнтами та персоналізовані рекомендації, побудовані з урахуванням як характеристик профілю, так і колективної історії взаємодій користувачів.

Із вищенаведеного випливає, що задача алгоритмічного вдосконалення процесу пошуку та підбору ІТ-фахівців зберігає актуальність і має чітко виражене практичне значення. Доцільність розроблення нової системи зумовлена двома взаємопов'язаними чинниками: обмеженістю функціональності наявних рішень у частині поєднання трьох перелічених механізмів і недостатньою прозорістю використовуваних алгоритмів. Подальший виклад розділу присвячено детальному аналізу окремих платформ цього сегмента ринку та формуванню вимог до програмної реалізації.

1.2 Огляд та приклади існуючих рішень

1.2.1 Djinni

Платформа Djinni [4] є українським майданчиком пошуку роботи у сфері інформаційних технологій, концептуальною особливістю якого є модель анонімного контакту. Здобувач створює анонімний профіль, де вказує цільову позицію, рівень кваліфікації, очікуваний рівень оплати праці та технологічний стек; роботодавець має доступ лише до цих параметрів, а персональні дані кандидата відкриваються тільки після встановлення прямого контакту через систему обміну повідомленнями. Платформа орієнтована виключно на ІТ-сегмент, що визначає галузеву специфіку довідників технологій

та переліку спеціалізацій.

До сильних сторін платформи можна віднести наступні характеристики:

– чітка спеціалізація на сфері ІТ, що забезпечує відповідні структури довідників та фільтрів;
– анонімна модель взаємодії, яка послаблює упередженість на початковому етапі добору;
– структура профілю здобувача з обов'язковими полями стосовно рівня кваліфікації, очікуваного рівня оплати та володіння англійською мовою;
– вбудована система обміну повідомленнями для прямого спілкування сторін.

Водночас з позиції механізмів пошуку й ранжування платформа має істотні обмеження:

– пошук кандидатів реалізовано як комбінацію незалежних фільтрів за фіксованими значеннями (технологія, рівень, локація, очікувана оплата) без побудови інтегральної оцінки відповідності;
– основним критерієм впорядкування результатів виступає давність оновлення профілю, а не релевантність до вимог конкретної вакансії;
– відсутні персоналізовані рекомендації, які пропонували б фахівців під окрему вакансію або, навпаки, вакансії для конкретного кандидата на основі історії взаємодій;
– роботодавець не має можливості варіювати ваги окремих критеріїв оцінки, наприклад надавати пріоритет збігу технологічного стеку перед географічною локацією.

1.2.2 DOU.ua

Платформа DOU.ua [5] є інформаційно-аналітичним порталом і спільнотою українських IT-спеціалістів; разом з аналітичними публікаціями, оглядами заробітних плат і рейтингами роботодавців на ньому розміщено дошку вакансій. Платформа виступає одним з найбільших джерел даних про український IT-ринок і традиційно використовується як майданчик публікації вакансій та пошуку нової роботи.

Серед переваг DOU.ua у контексті задачі підбору IT-фахівців можна виокремити:

- високий рівень довіри спільноти, що формує значну активну аудиторію;
- інтеграцію дошки вакансій з аналітичними публікаціями про ринок праці, оглядами заробітних плат та рейтингами роботодавців;
- можливість публікувати вакансії в різних форматах, зокрема орієнтованих на початківців.

З позиції алгоритмічного підбору фахівців DOU.ua має такі обмеження:

- платформа функціонує переважно як дошка вакансій з фільтрами за категоріями, а не як повноцінна система активного підбору кандидатів роботодавцем;
- повнотекстовий пошук вакансій базується на простому збігу ключових слів без застосування ймовірнісних моделей ранжування на кшталт BM25;
- інструментарій для роботодавця обмежено публікацією вакансії та переглядом надісланих відгуків, без можливості активного пошуку кандидатів за структурованими критеріями;
- рекомендаційні механізми відсутні, через що платформа не пропо-

нує роботодавцю фахівців, подібних до тих, з якими він уже взаємодіяв раніше.

1.2.3 LinkedIn

LinkedIn [6] являє собою глобальну професійну соціальну мережу, що поєднує функції побудови персонального професійного профілю, обміну професійним контентом і пошуку роботи. На відміну від спеціалізованих IT-платформ, LinkedIn охоплює широкий спектр галузей, що зумовлює одночасно значну користувацьку базу та певні обмеження для вузькопрофільних завдань підбору кадрів у сфері інформаційних технологій.

Сильні сторони LinkedIn для задачі рекрутингу можна охарактеризувати так:

- велика глобальна аудиторія, що дає змогу шукати кандидатів з різних географічних регіонів;
- розгалужені механізми побудови соціального графу, які використовуються в алгоритмах рекомендацій, зокрема при формуванні переліку вакансій на основі дій подібних користувачів;
- спеціалізовані інструменти для рекрутерів (LinkedIn Recruiter), що надають розширений набір фільтрів та можливість прямого звернення до кандидатів.

Зі специфіки задачі підбору фахівців на українському IT-ринку випливає низка недоліків:

- брак галузевої спеціалізації, через що довідники навичок слабо структуровані, не вимагають зазначення рівня володіння технологією й допускають вільне введення, що породжує дублікати (одна технологія може фігурувати як «React», «React.js» та «ReactJS» одночасно);
- закритість алгоритмів ранжування та формування рекомендацій,

що не дозволяє користувачу зрозуміти причини конкретного впорядкування видачі;

– повний доступ до інструментів активного пошуку кандидатів (LinkedIn Recruiter) надається лише на платних тарифах, вартість яких часто є істотним бар'єром для невеликих компаній і агенцій;
– фокус продукту на побудові професійної мережі, а не на ефективному формальному зіставленні кандидата і вакансії за технічними критеріями.

1.2.4 Robota.ua

Robota.ua [7] є універсальною українською платформою пошуку роботи, яка охоплює всі галузі економіки. Платформа має значну загальну аудиторію та широкий каталог вакансій; водночас її універсальність створює природні обмеження для задачі підбору саме IT-фахівців.

До переваг платформи можна віднести:

– значну загальну користувацьку базу, орієнтовану на український ринок праці;
– зручний інтерфейс публікації вакансій та подання резюме;
– базові інструменти фільтрації за галуззю, регіоном, типом зайнятості та рівнем оплати.

З позиції IT-рекрутингу обмеженнями платформи виступають:

– відсутність галузевої спеціалізації, що відображено у структурі профілів і вакансій: технологічний стек, рівень володіння інструментами та поділ на спеціалізації (Frontend, Backend, DevOps тощо) не представлені як категорії першого рівня;
– повнотекстовий пошук базується на простому збігу ключових слів без використання алгоритмів ранжування за релевантністю;
– зважені багатокритеріальні моделі оцінки відповідності кандидата

вакансії не реалізовано;

– рекомендаційні функції мають загальний характер і не враховують специфіки IT-підбору, зокрема рівнів володіння окремими технологіями.

1.3 Аналіз функціональності та вимог до системи

1.3.1 Порівняльна характеристика розглянутих рішень

Узагальнення результатів огляду подано у вигляді порівняльної таблиці 1.1. Для оцінювання обрано ті характеристики, що безпосередньо впливають на ефективність пошуку та підбору IT-фахівців: галузева спеціалізація, структурованість довідника технологій з рівнями володіння, наявність ранжування за релевантністю на основі BM25, реалізація багатокритеріальної зваженої оцінки відповідності та можливості налаштування її коефіцієнтів, наявність двосторонніх рекомендацій, прозорість алгоритмів і рівень безкоштовного доступу до інструментів пошуку кандидатів. В останньому стовпці наведено очікувані значення тих самих характеристик для системи, що проєктується у цій роботі.

Таблиця 1.1 — Порівняльна характеристика платформ пошуку та підбору IT-фахівців

Критерій	Djinni	DOU.ua	LinkedIn	Robota.ua	Запропонована система
Спеціалізація на IT-галузі	так	так	ні	ні	так

Кінець таблиці 1.1

Критерій	Djinni	DOU.ua	LinkedIn	Robota.ua	Запропонована система
Структурований довідник технологій з рівнями володіння	частково	ні	ні	ні	так
Повнотекстовий пошук з BM25-ранжуванням	ні	ні	закритий алгоритм	ні	так
Багатокритеріальна зважена оцінка відповідності	ні	ні	частково	ні	так
Налаштовувані ваги критеріїв пошуку	ні	ні	ні	ні	так
Рекомендації фахівців роботодавцю	ні	ні	так	частково	так
Рекомендації вакансій фахівцю	частково	ні	так	частково	так
Прозорість алгоритмів ранжування	ні	не визначено	ні	ні	так
Безкоштовний доступ до інструментів пошуку кандидатів	частково	частково	ні	так	так

Дані таблиці 1.1 підтверджують, що жодна з охарактеризованих платформ одночасно не поєднує три ключові механізми сучасного підбору кан-

дидатів: повнотекстове ранжування за ймовірнісною моделлю BM25, зважене багатокритеріальне оцінювання відповідності з налаштовуваними коефіцієнтами та гібридні рекомендаційні алгоритми. Там, де частина зазначених механізмів присутня (LinkedIn), вони функціонують як закриті від користувача функції без можливості налаштування. Спеціалізовані українські IT-платформи (Djinni), навпаки, пропонують структуровану модель даних, проте обмежуються базовою фільтрацією без алгоритмічного ранжування. Універсальні рекрутингові платформи (DOU.ua, Robota.ua) не забезпечують ані галузевої моделі даних, ані сучасних алгоритмів підбору.

Виявлені обмеження наявних рішень підтверджують доцільність розроблення нового програмного забезпечення, у якому три перелічені механізми інтегруються в єдиний робочий процес, а параметри ранжування доступні для прозорого налаштування з боку роботодавця.

1.3.2 Вимоги до реалізації веб-системи

На підставі результатів огляду сформовано перелік функціональних та нефункціональних вимог, які надалі визначають вибір технологічного стеку, структуру архітектури й деталі програмної реалізації. Функціональні вимоги фіксують перелік можливостей, що мають підтримуватися системою; нефункціональні – визначають характеристики якості, яким система має відповідати.

Функціональні вимоги поділено на сім груп.

1. Управління обліковими записами та автентифікація. Система забезпечує реєстрацію користувачів за електронною адресою з обов’язковим підтвердженням, автентифікацію за парою «електронна адреса і пароль», альтернативний вхід через OAuth-провайдера Google, відновлення паролю та автоматичне блокування облікового запису після п’яти послідовних невдалих спроб вхо-

ду.

2. Управління профілями фахівців. Фахівець має змогу створити та редагувати власний профіль, який охоплює персональні дані, технологічний стек з рівнями володіння, обрану спеціалізацію, рівень кваліфікації, досвід роботи, освіту, сертифікати, бажану заробітну плату, тип зайнятості та географічну локацію.
3. Управління профілями роботодавців. Роботодавець створює профіль компанії, що містить назву, логотип, опис, рік заснування, розмір, сферу діяльності та локацію.
4. Управління вакансіями. Роботодавець може створювати, публікувати, редагувати та архівувати вакансії; кожна вакансія описує обов'язкові технології з рівнями володіння, рівень кваліфікації, локацію, тип зайнятості та бюджет.
5. Гібридний пошук фахівців. Система виконує повнотекстовий пошук кандидатів з ранжуванням за релевантністю на основі моделі BM25, доповнений зваженою багатокритеріальною оцінкою відповідності за технологічним стеком, рівнем кваліфікації, локацією та очікуваною заробітною платою.
6. Рекомендаційна система. Система формує персоналізовані рекомендації у двох напрямах: добирає фахівців роботодавцю під конкретну вакансію та пропонує фахівцю релевантні вакансії на основі його профілю й історії взаємодій.
7. Взаємодія між користувачами. Система передбачає подання фахівцем відгуку на вакансію із супровідним листом, керування статусом відгуку з боку роботодавця, а також обмін повідомленнями між фахівцями і роботодавцями.

Нефункціональні вимоги до системи описано у вигляді переліку характеристик якості:

– час відгуку сервера на пошуковий запит не повинен перевищувати 1 с, на типовий CRUD-запит – 500 мс;
– система забезпечує одночасне обслуговування до 1 000 активних користувачів;
– обсяг даних, що підтримується системою, передбачає зберігання до 100 тис. профілів фахівців;
– рівень захищеності відповідає вимогам OWASP Top 10, зокрема щодо хешування паролів, обмеження частоти запитів та валідації вхідних даних;
– клієнтська частина зберігає сумісність із актуальними версіями браузерів Google Chrome, Mozilla Firefox, Microsoft Edge та Apple Safari;
– інтерфейс користувача реалізовано українською мовою.

Сформульовані функціональні та нефункціональні вимоги слугують підставою для технічного завдання на розроблення системи і визначають вибір технологічного стеку, який розглядається у наступному розділі.

ВИСНОВКИ ДО РОЗДІЛУ

У першому розділі схарактеризовано стан задачі пошуку та підбору фахівців у сфері інформаційних технологій з урахуванням специфіки українського IT-ринку. Проведено порівняльний аналіз чотирьох провідних платформ (Djinni, DOU.ua, LinkedIn, Robota.ua) і встановлено, що жодна з них одночасно не поєднує повнотекстового пошуку з BM25-ранжуванням, зваженого багатокритеріального оцінювання відповідності кандидата з налаштовуваними ваговими коефіцієнтами та гібридної рекомендаційної системи. На підставі виявлених обмежень сформульовано перелік функціональних і нефункціональних вимог до програмного забезпечення, що проєктується. Сформовані вимоги визначають вибір технологічного стеку та архітектурних рішень, які розглядаються у наступних розділах пояснювальної записки.

РОЗДІЛ 2
ОГЛЯД ТЕХНОЛОГІЙ ДЛЯ ПРОЕКТУВАННЯ ВЕБ-СИСТЕМИ

2.1 Загальні принципи побудови веб-застосунків

2.1.1 Клієнт-серверна архітектура

Клієнт-серверна архітектура є моделлю розподіленої обробки даних, у якій обчислювальна логіка розподілена між двома функціонально відокремленими сторонами. Клієнтська сторона виступає ініціатором запитів та відповідає за побудову інтерфейсу взаємодії з користувачем. Серверна сторона приймає вхідні запити, виконує їхню обробку відповідно до бізнес-логіки та формує відповіді. Обидві сторони виконуються в незалежних адресних просторах і обмінюються даними виключно через мережеві протоколи передачі.

Базовим транспортним протоколом для веб-застосунків виступає HTTP, специфікація якого визначена у RFC 9110 [8]. Залежно від версії протокол має текстове (HTTP/1.0, HTTP/1.1) або бінарне (HTTP/2, HTTP/3) представлення; всі версії побудовані за моделлю «запит–відповідь» без збереження стану взаємодії між послідовними запитами. Кожен запит охоплює метод (GET, POST, PUT, PATCH, DELETE та інші), цільовий ресурс у вигляді URI, набір заголовків та опціональне тіло; кожна відповідь містить статус-код, заголовки та опціональне тіло.

Для організації прикладного програмного інтерфейсу між клієнтом і сервером поширене застосування архітектурного стилю REST (Representational State Transfer), запропонованого Р. Філдінгом у 2000 році [9]. Стиль REST задає такі обмеження на спосіб взаємодії компонентів:

– клієнт-серверне розділення відповідальності між сторонами взає-

модії;

- відсутність збереження стану сесії на сервері, через що кожен запит несе всю інформацію, необхідну для його обробки;
- можливість кешування відповідей сервера на боці клієнта або проміжних вузлів передачі;
- уніфікований інтерфейс на основі ресурсів, які ідентифікуються за URI та маніпулюються стандартними методами HTTP;
- шарувата організація, у якій клієнт не зобов'язаний знати, обробляє його запит безпосередньо цільовий сервер чи проміжний шар.

Дотримання перелічених обмежень створює передумови для горизонтального масштабування веб-системи: будь-який екземпляр сервера здатний обслуговувати довільний запит будь-якого клієнта без потреби звертатися до внутрішнього стану інших екземплярів, що дає змогу безпечно розгортати кілька серверних процесів за балансувальником навантаження.

2.1.2 Основні компоненти веб-системи

Сучасну веб-систему доцільно розглядати як багатокомпонентний розподілений програмний комплекс, у якому виокремлюють такі типові складові:

- клієнтський застосунок, що виконується у середовищі браузера та забезпечує побудову інтерфейсу, обробку взаємодій з користувачем і обмін даними із сервером через мережевий протокол;
- серверний застосунок, що реалізує бізнес-логіку, надає прикладний програмний інтерфейс і координує взаємодію з підсистемами зберігання та обробки даних;
- систему управління базами даних, призначену для надійного довготривалого зберігання структурованих даних з підтримкою транзакційного доступу;

– кеш-сервер, який зберігає у пам’яті результати обчислень або проміжні дані з метою зменшити навантаження на основну СУБД та скоротити час відгуку;
– пошуковий рушій, що підтримує спеціалізовані індексні структури для повнотекстового пошуку та ранжування результатів за релевантністю;
– сервіс зберігання файлів, призначений для розміщення об’єктів великого розміру, які недоцільно зберігати у реляційній базі (зображення, документи, мультимедіа);
– зворотний проксі-сервер, що приймає клієнтські запити, маршрутизує їх до відповідних внутрішніх сервісів і одночасно виконує функції завершення TLS, балансування навантаження та обмеження частоти запитів.

Спосіб розподілу функціональності між наведеними компонентами визначальним чином впливає на масштабованість, відмовостійкість та зручність супроводу веб-системи. У наступних підрозділах розглянуто конкретні технології, що використано для реалізації кожної з перелічених складових.

2.2 Огляд технологій для реалізації веб-системи

2.2.1 Технології серверної частини

Серверна частина веб-системи відповідає за реалізацію бізнес-логіки, надання прикладного програмного інтерфейсу та взаємодію з базою даних і зовнішніми сервісами. Технологічний стек серверної частини охоплює мову програмування, середовище виконання, прикладний фреймворк і допоміжні бібліотеки.

TypeScript [10] становить строго типізовану надбудову над мовою

JavaScript, розроблену компанією Microsoft. Розширення додає до JavaScript систему статичних типів, інтерфейси, узагальнення та модифікатори доступу. Усі програми TypeScript перед запуском компілюються до JavaScript, тому зберігається повна сумісність з усіма середовищами виконання JavaScript. Статична типізація на етапі розробки дає змогу виявляти значну частину помилок ще до запуску програми, покращує автодоповнення в інтегрованих середовищах та підвищує самодокументованість коду.

Середовище виконання Node.js [11] забезпечує запуск JavaScript-коду поза браузером. Платформа побудована на віртуальній машині V8, розробленій компанією Google для Chrome, і реалізує неблокуючу однопотокову модель введення-виведення на основі циклу подій. Операції введення-виведення (читання з диска, мережеві операції, запити до баз даних), які традиційно є блокуючими, делегуються спеціалізованому набору системних потоків з поверненням результату через механізм зворотних викликів або проміжних обіцянок. Така модель забезпечує високу пропускну здатність для застосунків з переважанням операцій введення-виведення, до яких належить більшість веб-сервісів.

NestJS [12] є прикладним фреймворком для побудови серверних застосунків на Node.js, написаний мовою TypeScript. Архітектурно фреймворк запозичує підходи Angular: модульну структуру, ін’єкцію залежностей і декоратори для оголошення метаданих. NestJS не пов’язаний жорстко з конкретним HTTP-фреймворком: за замовчуванням використовується Express.js, проте передбачено альтернативу у вигляді Fastify.

Усередині NestJS застосунок організовано як ієрархію модулів. Кожен модуль інкапсулює окрему функціональну область, експортує сервіси та може імпортувати залежні модулі. У межах модуля виокремлюють такі типові компоненти:

– контролери, що приймають HTTP-запити, делегують обробку сер-

вісам і повертають відповіді;

– сервіси, які реалізують бізнес-логіку та не залежать від транспортного протоколу;
– охоронці, відповідальні за рішення про допуск запиту до контролера на основі автентифікації та авторизації;
– перехоплювачі, які додають наскрізну функціональність до обробки запитів (логування, трансформацію відповідей, кешування);
– канали, що виконують валідацію та перетворення вхідних даних.

Ін’єкція залежностей у NestJS реалізована за принципом інверсії керування: компоненти не створюють власні залежності самостійно, а отримують їх з контейнера фреймворку через параметри конструктора. Такий підхід істотно спрощує модульне тестування, оскільки залежності замінюються на тестові підставні об’єкти без змін у тестованому коді.

Prisma ORM [13] становить систему об’єктно-реляційного відображення для TypeScript та Node.js. На відміну від традиційних ORM, де моделі описуються класами мови програмування, Prisma використовує декларативну мову Prisma Schema Language для опису сутностей предметної області та зв’язків між ними. На основі схеми генеруються типобезпечний клієнт доступу до бази даних та SQL-міграції.

Ключовими властивостями Prisma, важливими для цієї роботи, є наступні:

– автоматична генерація TypeScript-типів для всіх сутностей та запитів, що усуває цілий клас помилок невідповідності типу даних на межі коду та СУБД;
– декларативний синтаксис запитів з підтримкою вкладеного включення пов’язаних сутностей, який запобігає проблемі типу «N+1 запитів»;
– версіонована система міграцій з автоматичним відстеженням змін

схеми;

– можливість підключення проміжного програмного забезпечення для реалізації наскрізних поведінкових патернів, наприклад «м’якого» видалення записів.

2.2.2 Технології клієнтської частини

Клієнтська частина веб-системи виконується у браузері користувача та відповідає за побудову інтерфейсу, обробку дій користувача й обмін даними з серверним прикладним інтерфейсом. Сучасні веб-застосунки переважно реалізують за моделлю односторінкового застосунку (Single Page Application, SPA), в якій сторінка завантажується одноразово, а подальша зміна вмісту відбувається за рахунок локального оновлення моделі документа без повного перезавантаження.

React [14] є бібліотекою для побудови користувацьких інтерфейсів, розробленою компанією Meta. В основу бібліотеки покладено компонентну модель, в якій інтерфейс описується як композиція функцій або класів, що повертають декларативне подання деревовидної структури елементів. Це подання є проміжним і відоме як віртуальна модель документа (Virtual DOM). При зміні стану компонента React обчислює різницю між поточним та попереднім поданнями і застосовує до реальної моделі документа лише мінімально необхідні зміни.

Актуальні версії React оперують функціональними компонентами та механізмом хуків, які дають компонентам змогу зберігати стан між циклами рендерингу й реагувати на події життєвого циклу без використання класів. Базовими хуками виступають `useState` (збереження локального стану), `useEffect` (виконання побічних дій після рендерингу), `useMemo` та `useCallback` (оптимізація обчислень і посилань на функції).

Інструмент збірки Vite [15] під час розробки спирається на нативну

підтримку модулів ECMAScript у браузерах, а для виробничої збірки застосовує Rollup. Така архітектура забезпечує практично миттєвий запуск середовища розробки та швидке гаряче оновлення модулів незалежно від розміру проєкту.

Tailwind CSS [16] є CSS-фреймворком, побудованим за принципом utility-first. Замість готових стилізованих блоків, як це прийнято у класичних бібліотеках компонентів, Tailwind пропонує набір атомарних класів-утиліт, що відповідають окремим CSS-властивостям. Стилізація компонента формується як композиція таких класів безпосередньо у розмітці. Підхід дає змогу уникнути написання користувацьких CSS-файлів та забезпечує цілісність дизайн-системи за рахунок обмеженого набору доступних значень відступів, кольорів і параметрів типографіки.

Бібліотека Zustand [17] реалізує управління глобальним станом для застосунків React за рахунок легковагової концепції сховища, у якому містяться значення стану та функції їх зміни, доступні з будь-якого компонента через спеціальний хук. На відміну від Redux, тут не потрібно описувати дії, редюсери чи проміжне програмне забезпечення для асинхронних операцій, унаслідок чого помітно скорочується обсяг шаблонного коду.

TanStack Query [18] є бібліотекою для управління серверним станом у клієнтських застосунках. Вона розв’язує задачі кешування результатів запитів, дедуплікації одночасних звернень, фонового оновлення даних, повторних спроб при помилках мережі та синхронізації стану між компонентами. Серверний стан принципово відрізняється від клієнтського: він може застаріти без відома клієнта, запитуватися паралельно з різних місць і потребувати окремої політики оновлення. TanStack Query надає декларативну модель роботи із серверним станом через хуки `useQuery` та `useMutation`.

2.2.3 Характеристика системи управління базами даних

Для довготривалого зберігання структурованих даних веб-системи використано об’єктно-реляційну СУБД PostgreSQL [19]. СУБД має відкритий вихідний код, підтримує стандарт SQL і надає розширений набір додаткових можливостей.

До ключових характеристик PostgreSQL, які роблять цю СУБД придатною для розв’язуваної задачі, належать такі:

– транзакційна модель ACID, що забезпечує атомарність, узгодженість, ізольованість та довговічність змін даних і гарантує їхню цілісність у разі одночасних транзакцій або збоїв системи;
– підтримка складних типів даних (масивів, діапазонів, JSON та JSONB, перелічувальних і користувацьких композитних типів), що дає змогу моделювати предметну область точніше, ніж дозволяє базовий SQL;
– розширена індексація з використанням типів B-tree, hash, GiST, GIN, BRIN, а також підтримка часткових і виразних індексів; типи GIN та GiST уможливлюють ефективний повнотекстовий пошук засобами самої СУБД;
– підтримка синхронної та асинхронної реплікації, що дозволяє формувати конфігурації з підвищеною доступністю та масштабувати читання за рахунок реплік;
– багатоверсійний контроль конкурентності (MVCC), за якого паралельне читання й запис відбуваються без взаємного блокування читачів і записувачів.

У розроблюваній системі PostgreSQL виступає основним сховищем даних, у якому зберігаються профілі користувачів, вакансії, відгуки, повідомлення, історія переглядів та інші структуровані сутності.

2.2.4 Ключові допоміжні бібліотеки та інструменти

Окрім основних компонентів клієнтської та серверної частин, типова веб-система спирається на низку допоміжних технологій, які вирішують спеціалізовані задачі.

Redis [20] є сховищем даних типу «ключ–значення» з резидентністю в оперативній пам’яті. Окрім простих рядкових значень Redis підтримує структури даних на кшталт списків, множин, упорядкованих множин, хеш-таблиць та потоків. Висока швидкодія (порядок мільйонів операцій на секунду на одному вузлі) досягається завдяки повному розміщенню даних у пам’яті й однопотоковій моделі обробки команд. У цій роботі Redis застосовується для кешування результатів обчислень рекомендаційної підсистеми з обмеженим часом життя ключів.

Платформа Docker [21] забезпечує контейнеризацію застосунків: програмний продукт пакується разом зі своїми залежностями та середовищем виконання в ізольований контейнер. Контейнер являє собою легковагову альтернативу віртуальній машині, оскільки використовує ядро операційної системи хоста, але має ізольовані простори імен для файлової системи, мережі та процесів. Інструмент Docker Compose дає змогу описати конфігурацію всіх сервісів багатокомпонентного застосунку у вигляді декларативного YAML-файлу. У цій роботі Docker Compose використано для одночасного запуску серверного застосунку, PostgreSQL, Redis і Elasticsearch у локальному середовищі розробки.

Nginx [22] є високопродуктивним веб-сервером і зворотним проксі-сервером. У розроблюваній системі Nginx виконує функції зворотного проксі: приймає вхідні HTTPS-запити, маршрутизує статичні ресурси клієнтського застосунку безпосередньо з файлової системи, а звернення до прикладного інтерфейсу спрямовує до серверного застосунку Node.js.

Хмарний сервіс Firebase Storage [23] забезпечує об’єктне зберігання користувацьких файлів великого розміру. Сервіс надає HTTP-інтерфейс для завантаження, отримання та видалення об’єктів, підтримує контроль доступу на рівні правил безпеки та автоматично масштабує доступний обсяг сховища. У цій роботі сервіс застосовується для зберігання аватарів користувачів, логотипів компаній, документів резюме у форматі PDF та зображень сертифікатів.

2.3 Механізми автентифікації та авторизації

2.3.1 Використання JWT для автентифікації

JSON Web Token (JWT) визначений як відкритий стандарт у RFC 7519 [24]. Документ описує компактний та безпечний для передачі URL-сумісним способом формат, призначений для подання тверджень (claims) між двома сторонами взаємодії. Структура JWT тривимірна; три частини розділено символом крапки:

$$\mathrm{JWT} = \mathrm{Base64Url(Header)} \,.\, \mathrm{Base64Url(Payload)} \,.\, \mathrm{Base64Url(Signature)} \quad (2.1)$$

де Header – заголовок, що містить тип токену та алгоритм підпису;
Payload – корисне навантаження зі списком тверджень про користувача та токен;
Signature – криптографічний підпис, обчислений на основі заголовка, корисного навантаження та секретного ключа.

Обчислення підпису для алгоритму HMAC-SHA256 виконується за формулою:

$$\text{Signature} = \text{HMACSHA256}\big(\text{Base64Url(Header)}\,.\,\text{Base64Url(Payload)},\ K\big) \quad (2.2)$$

де HMACSHA256 – криптографічна функція коду автентифікації повідомлення на основі SHA-256;

K – секретний ключ, відомий лише серверу.

Перевірка JWT на серверному боці зводиться до обчислення підпису для отриманих заголовка і корисного навантаження з використанням секретного ключа з подальшим порівнянням результату з підписом, що міститься в самому токені. Будь-яка модифікація вмісту токену клієнтом призводить до невідповідності підпису й до відхилення запиту.

У межах роботи застосовано схему з двома типами токенів: короткоживучим access-токеном та довгоживучим refresh-токеном. Access-токен має термін дії 15 хвилин, використовується для авторизації запитів до прикладного інтерфейсу й не зберігається на сервері. Refresh-токен має термін дії 7 днів, зберігається в окремій таблиці бази даних і використовується для одержання нового access-токену після завершення терміну дії попереднього. Зберігання refresh-токенів у БД забезпечує можливість примусового відкликання у разі підозри на компрометацію облікового запису.

Для хешування паролів застосовано алгоритм bcrypt [25], який є адаптацією шифру Blowfish для побудови криптографічно стійкої односторонньої функції з налаштовуваною обчислювальною складністю. Алгоритм має вбудований механізм солі: випадкове значення довжиною 128 бітів генерується для кожного пароля та зберігається разом з результатом хешування, що унеможливлює атаки з використанням попередньо обчислених таблиць (rainbow tables). Налаштовуваним параметром bcrypt виступає кількість раундів, яка задає обчислювальну складність функції: збільшення цього пара-

метра на одиницю подвоює час обчислення, що дає змогу адаптуватися до зростання обчислювальних потужностей з часом. У роботі використано значення параметра 12 раундів.

2.3.2 Інтеграція з OAuth 2.0 через Google

OAuth 2.0 є протоколом делегованої авторизації, формально визначеним у RFC 6749 [26]. Протокол передбачає, що клієнтський застосунок отримує обмежений доступ до ресурсів користувача, які зберігаються в іншого постачальника (провайдера), без необхідності отримувати від користувача безпосередньо облікові дані до цього провайдера.

OAuth 2.0 підтримує кілька сценаріїв одержання маркера доступу (grant types). Для веб-застосунків із серверною частиною основним рекомендованим сценарієм є Authorization Code Grant. Послідовність взаємодій у цьому сценарії має такий вигляд:

1. клієнтський застосунок перенаправляє користувача на сторінку авторизації провайдера з параметрами запиту: ідентифікатором клієнта, переліком запитуваних дозволів (scopes) і URI повернення;
2. користувач виконує автентифікацію перед провайдером і надає дозвіл на доступ;
3. провайдер перенаправляє користувача назад на URI клієнта з кодом авторизації;
4. серверна частина клієнтського застосунку обмінює код авторизації на маркер доступу, звертаючись до API провайдера з кодом, ідентифікатором клієнта та секретом клієнта;
5. провайдер повертає маркер доступу, з яким клієнтський застосунок надалі звертається до захищених ресурсів.

У межах роботи OAuth 2.0 застосовується для автентифікації користувачів через Google. Інтеграція виконана з використанням бібліотеки

Passport.js [27], що надає єдиний інтерфейс до різних стратегій автентифікації. При першому вході через Google система отримує електронну адресу користувача з його профілю і створює локальний обліковий запис, прив'язаний до цієї адреси; при наступних входах виконується аналогічне зіставлення за тією самою електронною адресою.

2.4 Методи інформаційного пошуку

2.4.1 Поняття повнотекстового пошуку

Повнотекстовий пошук визначають як підхід, у якому зіставлення відбувається не із заздалегідь заданими метаданими документа, а з повним його текстовим вмістом. Для досягнення прийнятної швидкодії на великих колекціях документів повнотекстовий пошук не може реалізовуватися послідовним скануванням всіх документів; натомість використовується спеціальна індексна структура, яка має назву інвертованого індексу.

Інвертований індекс зіставляє кожному терму (слову або лексикографічній одиниці) перелік документів, у яких цей терм зустрічається, разом з додатковими відомостями: частотою терма в документі, позиціями входжень, ваговими коефіцієнтами тощо. Побудові індексу передують операції попередньої обробки тексту:

- токенізація, тобто розбиття вхідного тексту на окремі лексичні одиниці за заданими роздільниками;
- нормалізація регістра, що передбачає переведення всіх токенів у єдиний регістр для забезпечення регістронечутливого пошуку;
- видалення стоп-слів, тобто виключення з індексу високочастотних службових слів, які не несуть смислового навантаження для пошуку;
- стемінг або лематизація, тобто зведення словоформ до спільної

основи, що дає змогу знаходити документи з різними формами однієї лексеми за одним запитом.

Класичною моделлю оцінювання релевантності документа щодо запиту виступає модель TF–IDF (Term Frequency – Inverse Document Frequency). За цією моделлю вагомість терма t у документі D обчислюється як добуток двох величин: частоти терма в документі та оберненої частоти терма у колекції:

$$\text{TF-IDF}(t, D) = \text{TF}(t, D) \cdot \text{IDF}(t) \tag{2.3}$$

де $\text{TF}(t, D)$ – кількість входжень терма t у документі D (можливо, нормована);

$\text{IDF}(t)$ – обернена частота терма, обчислена на основі загальної кількості документів у колекції та кількості документів, що містять терм.

Попри простоту й інтуїтивну зрозумілість, модель TF–IDF має істотний недолік: вона не передбачає насичення частоти терма та не нормалізує оцінку залежно від довжини документа. Для усунення цих обмежень була запропонована ймовірнісна модель ранжування BM25, що набула широкого застосування у сучасних пошукових системах.

2.4.2 Огляд алгоритму ранжування BM25

Модель BM25 (Best Matching 25) [28] становить ймовірнісну функцію ранжування документів за релевантністю до запиту, запропоновану С. Робертсоном та К. Спарк-Джонс на основі ймовірнісної моделі інформаційного пошуку. Модель є фактичним стандартом у сучасних пошукових системах і використовується як базова функція ранжування в Apache Lucene, Apache Solr та Elasticsearch.

Оцінка релевантності документа D запиту Q, що складається з термів $q_1, q_2, \ldots, q_n$, обчислюється за формулою:

$$\text{score}(D, Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{avgdl}\right)} \qquad (2.4)$$

де $f(q_i, D)$ – частота входження терма q_i у документі D;

$|D|$ – довжина документа D, виміряна у кількості термів;

avgdl – середня довжина документа у колекції;

k_1 – параметр, що контролює ступінь насичення частоти терма (стандартне значення $k_1 = 1{,}2$);

b – параметр нормалізації за довжиною документа (стандартне значення $b = 0{,}75$);

$\text{IDF}(q_i)$ – обернена частота документа для терма q_i.

Обернену частоту документа у BM25 обчислюють за формулою:

$$\text{IDF}(q_i) = \ln\left(\frac{N - n(q_i) + 0{,}5}{n(q_i) + 0{,}5} + 1\right) \qquad (2.5)$$

де N – загальна кількість документів у колекції;

$n(q_i)$ – кількість документів, у яких міститься терм q_i.

Порівняно з TF–IDF модель BM25 містить дві принципові відмінності. По-перше, друга частина формули (2.4) забезпечує насичення за частотою терма: при збільшенні $f(q_i, D)$ її значення асимптотично прямує до k_1+1, тому додаткові входження одного і того самого терма дають дедалі менший приріст оцінки. Така поведінка відповідає інтуїтивному уявленню про те, що поява терма в документі двадцять разів не означає двадцятикратного зростання релевантності у порівнянні з єдиним входженням. По-друге, формула нормалізує оцінку за довжиною документа через параметр b: документи, довші за середню довжину колекції, отримують зменшену оцінку, що зменшує упередженість моделі до довгих документів.

2.4.3 Пошуковий рушій Elasticsearch

Elasticsearch [29] є розподіленим пошуковим рушієм з відкритим вихідним кодом, побудованим на основі бібліотеки Apache Lucene. Рушій надає високорівневий HTTP-інтерфейс для індексації документів, виконання повнотекстових запитів, аналітичних агрегацій та керування кластером. Дані в Elasticsearch організовано в індекси, які поділяються на шарди (shards), що дає змогу розподіляти індекс між кількома вузлами кластера; для забезпечення доступності кожен шард може мати одну або більше реплік.

Базовою функцією ранжування в Elasticsearch є BM25, реалізована в Apache Lucene. Параметри k_1 та b можна налаштовувати на рівні окремого поля індексу. Для документів, які містять кілька текстових полів (наприклад, профіль фахівця із заголовком, біографією та переліком технологій), Elasticsearch підтримує багатопольовий пошук з можливістю задання вагових коефіцієнтів для кожного поля.

Для побудови складних функцій ранжування, що поєднують текстову релевантність із додатковими критеріями, в Elasticsearch передбачено механізм `function_score`. Цей механізм дає змогу обчислити підсумкову оцінку документа як комбінацію базової BM25-оцінки та значень довільних функцій від полів документа. Підтримуються режими комбінації `multiply`, `sum`, `avg`, `max`, `min` та `replace`. Це створює технічну основу для реалізації гібридного алгоритму пошуку фахівців, у якому BM25-релевантність текстового запиту об'єднується із зваженою багатокритеріальною оцінкою відповідності за технологічним стеком, рівнем кваліфікації, локацією та бюджетом.

2.5 Методи побудови рекомендаційних систем

2.5.1 Контент-базована фільтрація

Контент-базована фільтрація формує рекомендації на основі схожості властивостей (контенту) об'єктів. Об'єкти подаються у вигляді векторів ознак у деякому багатовимірному просторі; рекомендованими вважаються ті об'єкти, які виявляються найближчими до профілю інтересів користувача в цьому просторі.

У задачі підбору фахівців профіль фахівця та опис вакансії можна подати як вектори у просторі технологій:

$$S = (s_1, s_2, \ldots, s_n), \quad V = (v_1, v_2, \ldots, v_n) \tag{2.6}$$

де S – вектор технологій фахівця;

V – вектор технологій вакансії;

n – загальна кількість унікальних технологій у довіднику;

s_i – рівень володіння фахівцем технологією i, виражений цілим числом від нуля (технологією не володіє) до чотирьох (експертний рівень);

v_i – потрібний для вакансії рівень технології i (нуль, якщо технологія не входить до вимог).

Найпоширенішою мірою близькості двох векторів у контент-базованій фільтрації виступає косинусна подібність. Її обчислюють для пари ненульових векторів за формулою:

$$\cos(S, V) = \frac{S \cdot V}{\|S\| \cdot \|V\|} = \frac{\sum\limits_i s_i v_i}{\sqrt{\sum\limits_i s_i^2} \cdot \sqrt{\sum\limits_i v_i^2}} \tag{2.7}$$

де $S \cdot V$ – скалярний добуток векторів;

$\|S\|$ та $\|V\|$ – евклідові норми відповідних векторів.

Косинусна подібність набуває значень у відрізку $[-1, 1]$; для векторів з невід'ємними компонентами, до яких належать вектори у просторі рівнів володіння технологіями, значення обмежено відрізком $[0, 1]$. Значення, близьке до одиниці, відповідає високій подібності векторів, а нульове – їхній ортогональності, тобто відсутності спільних технологій. Принциповою перевагою косинусної подібності порівняно з евклідовою відстанню є нечутливість до абсолютних значень компонентів: дві пропорційні шкали навичок одного й того самого напряму спеціалізації будуть визнані максимально подібними незалежно від загального рівня кваліфікації фахівця.

Контент-базована фільтрація має два істотні обмеження. По-перше, вона не виходить за межі ознак, явно заданих у моделі: якщо вакансію описано через перелік технологій, рекомендації ґрунтуватимуться лише на цих технологіях, без урахування інших характеристик, які можуть бути значущими для роботодавця. По-друге, такий підхід не використовує колективного досвіду інших користувачів системи: якщо двоє роботодавців переглядають подібні профілі, цей сигнал не впливає на рекомендації, доки самі профілі не виявляться подібними за вмістом.

2.5.2 Колаборативна фільтрація

Колаборативна фільтрація формує рекомендації, спираючись на аналіз поведінки множини користувачів. Базова ідея підходу полягає у тому, що користувачі, які раніше виявляли подібні вподобання, з високою ймовірністю виявлять схожі вподобання і у майбутньому.

У колаборативній фільтрації виокремлюють два основні підходи. У підході user-based для побудови рекомендацій конкретному користувачу u спочатку знаходять користувачів з найбільш подібною до u історією взаємодій, після чого рекомендують об'єкти, які подобалися цим подібним користувачам, але ще не оцінювалися користувачем u. У підході item-based обчи-

слюють подібність між парами об'єктів на основі того, які користувачі з ними взаємодіяли, після чого для користувача u рекомендують об'єкти, подібні до тих, які вже були йому цікаві. На практиці підхід item-based отримав більше поширення завдяки відносній стабільності матриці подібності об'єктів у часі та можливості її попереднього обчислення.

У задачі підбору фахівців сигналами для колаборативної фільтрації слугує імпліцитний (непрямий) зворотний зв'язок, а саме історія переглядів профілів фахівців роботодавцями та історія переглядів вакансій фахівцями. На відміну від експліцитного зворотного зв'язку (оцінок, рейтингів), імпліцитний не вимагає додаткових дій з боку користувача, проте має невизначену семантику: сам факт перегляду профілю не обов'язково означає високий рівень зацікавленості. У роботі [30] запропоновано формальну модель імпліцитного зворотного зв'язку, у якій кількість взаємодій інтерпретується не як оцінка, а як рівень довіри до позитивної інтерпретації.

Item-based колаборативна фільтрація у задачі підбору фахівців реалізується таким чином: для пари фахівців обчислюється міра спільності множин роботодавців, які переглядали їхні профілі. Нехай V_A та V_B – множини роботодавців, що переглядали профілі двох фахівців A та B. Подібність двох фахівців обчислюється за коефіцієнтом Жаккара:

$$\text{sim}_{\text{collab}}(A, B) = \frac{|V_A \cap V_B|}{|V_A \cup V_B|} \tag{2.8}$$

де $|V_A \cap V_B|$ – кількість роботодавців, які переглянули профілі обох фахівців; $|V_A \cup V_B|$ – кількість роботодавців, які переглянули принаймні один з двох профілів.

2.5.3 Гібридні підходи

Гібридні рекомендаційні системи поєднують контент-базовану та колаборативну фільтрації, що дає змогу компенсувати слабкі сторони кожного з підходів окремо. Контент-базована фільтрація успішно працює для нових об'єктів, у яких ще немає історії взаємодій, оскільки спирається на властивості самих об'єктів, проте не використовує сигналів від інших користувачів. Колаборативна фільтрація навпаки виявляє нетривіальні зв'язки на основі колективної поведінки, але страждає від проблеми «холодного старту»: рекомендації не можуть бути сформовані для нових користувачів або нових об'єктів через відсутність історії взаємодій.

Найпростішим і водночас одним з найефективніших способів об'єднання двох підходів є зважена лінійна комбінація. Підсумкова оцінка для пари (користувач u, об'єкт i) обчислюється як зважена сума оцінок, отриманих контент-базованим і колаборативним підходами:

$$\text{score}(u, i) = \alpha \cdot \text{score}_{\text{content}}(u, i) + (1 - \alpha) \cdot \text{score}_{\text{collab}}(u, i) \qquad (2.9)$$

де $\text{score}_{\text{content}}(u, i)$ – оцінка, обчислена контент-базованим підходом (наприклад, косинусна подібність векторів);

$\text{score}_{\text{collab}}(u, i)$ – оцінка, обчислена колаборативним підходом;

$\alpha \in [0, 1]$ – ваговий коефіцієнт балансу між підходами.

Перевагою такої моделі виступає можливість динамічного налаштування коефіцієнта α залежно від поточної ситуації. Для нового користувача, у якого ще немає історії взаємодій із системою, доцільно задати $\alpha = 1$, що відповідає виключно контент-базованим рекомендаціям. У міру накопичення історії взаємодій значення α зменшують, унаслідок чого колаборативна складова починає робити внесок у формування рекомендацій. Такий підхід

дає змогу ефективно вирішувати проблему холодного старту без ускладнення моделі.

Якість рекомендаційних систем оцінюють за низкою стандартних метрик. Найпоширенішими з них є наступні [31]:

– точність на k позиціях (precision@k), яка визначає частку релевантних об'єктів серед перших k позицій рекомендації;
– повнота на k позиціях (recall@k), що відображає частку релевантних об'єктів, які увійшли до перших k позицій, від загальної кількості релевантних об'єктів;
– F-міра на k позиціях ($F_1@k$), обчислювана як гармонійне середнє точності й повноти:

$$F_1@k = \frac{2 \cdot \text{precision}@k \cdot \text{recall}@k}{\text{precision}@k + \text{recall}@k} \tag{2.10}$$

– частота попадань на k позиціях (hit rate@k), що відображає частку користувачів, для яких принаймні один релевантний об'єкт потрапив до перших k позицій рекомендації.

Перелічені метрики застосовуються до оцінювання якості розробленої рекомендаційної системи у розділі 4 пояснювальної записки.

ВИСНОВКИ ДО РОЗДІЛУ

У другому розділі систематизовано теоретичні засади та технологічні рішення, що складають основу для подальшої реалізації програмного забезпечення. Розглянуто принципи побудови сучасних веб-застосунків: клієнт-серверну архітектуру, обмеження архітектурного стилю REST та модель «запит–відповідь» протоколу HTTP як основу горизонтального масштабування серверної частини.

Обґрунтовано вибір технологічного стеку серверної частини, до якого входять мова TypeScript з статичною типізацією, середовище виконання Node.js з неблокуючою моделлю введення-виведення, прикладний фреймворк NestJS з модульною архітектурою та ін’єкцією залежностей, а також система об’єктно-реляційного відображення Prisma. Для клієнтської частини обрано бібліотеку React з функціональними компонентами та хуками, інструмент збірки Vite з підтримкою нативних модулів ECMAScript, CSS-фреймворк Tailwind, легковагову бібліотеку керування глобальним станом Zustand та бібліотеку TanStack Query для роботи з серверним станом. Як основну СУБД обрано PostgreSQL з підтримкою ACID-транзакцій, складних типів даних, розширеної індексації та багатоверсійного контролю конкурентності. Допоміжними технологіями визначено кеш-сервер Redis, платформу контейнеризації Docker з інструментом Docker Compose, зворотний проксі-сервер Nginx та хмарне об’єктне сховище Firebase Storage.

Розглянуто механізми автентифікації та авторизації: формат JSON Web Tokens із тривимірною структурою, схему з двома типами токенів (короткоживучим access-токеном і довгоживучим refresh-токеном) та можливістю примусового відкликання, алгоритм bcrypt з вбудованою сіллю й налаштовуваною обчислювальною складністю, протокол OAuth 2.0 у сценарії Authorization Code Grant з інтеграцією через Google як альтернативний шлях

автентифікації.

Проаналізовано методи інформаційного пошуку: класичну модель TF–IDF та її обмеження, ймовірнісну модель ранжування BM25 з наведенням формул обчислення підсумкової оцінки релевантності та оберненої частоти документа, а також архітектуру пошукового рушія Elasticsearch і механізм `function_score`, що дає змогу будувати складні функції ранжування з поєднанням текстової релевантності та зваженого багатокритеріального оцінювання.

Розглянуто методи побудови рекомендаційних систем: контент-базовану фільтрацію з косинусною мірою подібності векторів навичок, item-based колаборативну фільтрацію на основі коефіцієнта Жаккара та гібридну схему зваженого об'єднання двох підходів з адаптивним коефіцієнтом α, що забезпечує плавний перехід від режиму холодного старту до повноцінного гібридного режиму. Описано стандартні метрики оцінювання якості рекомендаційних систем, що застосовуватимуться у четвертому розділі.

Систематизований у розділі теоретичний та технологічний матеріал виступає підставою для архітектурних рішень і програмної реалізації, опис яких подано у наступному розділі.

РОЗДІЛ 3
ДЕТАЛІ РОЗРОБКИ ВЕБ-СИСТЕМИ

3.1 Архітектура системи

3.1.1 Загальна архітектурна схема веб-системи

Розроблюваний програмний комплекс організовано як розподілений застосунок з виразним розподілом відповідальності між клієнтською та серверною частинами. В основу покладено клієнт-серверну архітектуру з REST API; обмін даними між сторонами відбувається у форматі JSON через захищений транспортний протокол HTTPS.

Кодова база проєкту оформлена як монорепозиторій на основі інструмента Nx версії 20.8.4 і містить три застосунки та дві спільні бібліотеки. Серверну частину розміщено у директорії apps/backend (NestJS), клієнтську – у apps/frontend (React), а наскрізні тести – у apps/backend-e2e. Спільні TypeScript-типи виокремлено у libs/shared-types, а константи з параметрами алгоритмів, зокрема ваговими коефіцієнтами пошуку, зосереджено у libs/shared-constants.

Серверна частина взаємодіє з чотирма зовнішніми підсистемами зберігання та обробки даних: реляційною СУБД PostgreSQL, пошуковим рушієм Elasticsearch 8.19.1, кеш-сервером Redis (через клієнт ioredis 5.11) та хмарним об’єктним сховищем Firebase Storage. Технологічні стеки клієнтської та серверної частин повністю відповідають обґрунтуванню, поданому у підрозділах 2.2.1 та 2.2.2.

Загальна структурна схема системи з компонентами та інформаційними потоками подана у графічних матеріалах (додаток Д1).

3.1.2 Архітектурні особливості клієнтської частини

Клієнтська частина реалізована як односторінковий застосунок (Single Page Application) на основі бібліотеки React 19 з інструментом збірки Vite. Структурно код поділено на маршрутні сторінки (директорія pages/) та перевикористовувані компоненти (директорія components/). Окремий шар api/ інкапсулює взаємодію із серверним прикладним інтерфейсом через HTTP-клієнт Axios.

Керування станом розділено на дві категорії. Глобальний клієнтський стан обробляється бібліотекою Zustand з персистентністю у браузерному локальному сховищі під спеціальним ключем, тоді як серверний стан обслуговується бібліотекою TanStack React Query з налаштовуваним часом застарівання (staleTime) 30 секунд. Детальніший опис організації директорій клієнтської частини подано у підрозділі 3.3.1.

Маршрутизацію реалізовано засобами бібліотеки React Router версії 6 з підтримкою вкладених маршрутів. Перелік маршрутів клієнтського застосунку з зазначенням рольових обмежень доступу наведено у таблиці 3.1.

Таблиця 3.1 — Маршрути клієнтського застосунку

Маршрут	Сторінка	Доступ
/	Публічна головна	Public
/login, /register	Вхід, реєстрація	Public
/vacancies, /vacancies/:id	Вакансії, деталі	SPECIALIST
/recommendations	Рекомендації	SPECIALIST
/applications	Мої заявки	SPECIALIST
/profile, /profile/edit	Профіль фахівця	SPECIALIST
/specialists/:id	Публічний профіль	Authenticated

Кінець таблиці 3.1

Маршрут	Сторінка	Доступ
/employer/vacancies	Свої вакансії	EMPLOYER
/employer/vacancies/new, /:id/edit	Створення/редагування	EMPLOYER
/employer/vacancies/:id	Заявки на вакансію	EMPLOYER
/employer/profile	Профіль роботодавця	EMPLOYER
/employer/specialists	Пошук спеціалістів	EMPLOYER
/messages, /messages/:id	Список розмов, чат	Authenticated

Звернення клієнтської частини до серверного прикладного інтерфейсу виконуються через HTTP-клієнт Axios з базовим URL /api. Перехоплювач запитів автоматично додає Bearer-токен у заголовок Authorization; перехоплювач відповідей при отриманні статусу 401 ініціює оновлення пари токенів через POST /auth/refresh і повторює оригінальний запит. У разі невдалого оновлення виконується вихід користувача з системи.

3.1.3 Архітектурні особливості серверної частини

Серверна частина побудована на прикладному фреймворку NestJS з модульною архітектурою. Загалом до складу серверної частини входить 18 модулів NestJS, кожен з яких відповідає за окрему функціональну область. Графічне представлення структури модулів та їхніх взаємозв’язків подано на рисунку 3.1.

Модулі утворюють чотири функціональні групи. До інфраструктурних модулів належать PrismaModule, RedisModule, ElasticsearchClientModule та EmailModule; вони позначені декоратором @Global() і доступні в усіх інших модулях без повторного імпорту. Група модулів автен-

Рисунок 3.1 — Структура модулів серверної частини NestJS

тифікації та керування (AuthModule, UsersModule, FilesModule) забезпечує обробку облікових записів, JWT-токенів і завантаження файлів. Доменні модулі (SpecialistsModule, EmployersModule, VacanciesModule, ApplicationsModule, MessagesModule, SkillsModule, SpecializationsModule, LocationsModule) реалізують бізнес-логіку відповідних сутностей. Алгоритмічні модулі SearchModule та RecommendationsModule реалізують ключові науково-практичні рішення, що складають наукову новизну роботи, а саме гібридний пошук і рекомендаційну систему.

Перелік глобальних охоронців та елементи конвеєра обробки запитів подано у таблиці 3.2.

Таблиця 3.2 — Глобальні охоронці та конвеєр обробки запиту

Засіб	Призначення
JwtAuthGuard	Перевірка JWT access-токену; @Public/@OptionalJwt
RolesGuard	Перевірка ролі користувача через декоратор @Roles()
ValidationPipe	Валідація DTO (whitelist, forbidNonWhitelisted, transform)
ThrottlerModule	Rate-limiting: 100 запитів/хв загально, 10 для /auth/*
CORS	Дозволені джерела: лише APP_FRONTEND_URL

Кожен запит послідовно проходить ланцюжок JwtAuthGuard, далі RolesGuard, потім ValidationPipe, після чого передається до Controller, потім до Service, далі до Prisma і нарешті до бази даних; відповідь повертається у зворотному порядку. Якщо помилка виникає на будь-якому з етапів, запит відхиляється з відповідним статус-кодом (401, 403, 400). Для отримання інформації про поточного автентифікованого користувача в обробниках застосовується декоратор @CurrentUser().

3.1.4 Архітектура бази даних

Основним сховищем даних розробленої системи виступає реляційна СУБД PostgreSQL. Вибір саме реляційної моделі обумовлено характером предметної області: дані про користувачів, профілі, вакансії, відгуки та повідомлення мають чітко визначену структуру з численними зв’язками між сутностями. Реляційна модель забезпечує цілісність цих зв’язків на рівні самої СУБД через механізм зовнішніх ключів і дає змогу будувати ефективні запити з об’єднанням таблиць. Транзакційна модель ACID, підтримувана

PostgreSQL, є критично важливою для операцій модифікації даних, у яких потрібно одночасно змінювати декілька пов’язаних записів (наприклад, створення відгуку з одночасним оновленням лічильника відгуків на вакансію).

Схему бази даних організовано за принципом нормалізації до третьої нормальної форми, що мінімізує надлишковість даних та запобігає аномаліям оновлення. Багато-до-багатьох зв’язки (фахівець — технологія, фахівець — спеціалізація, вакансія — обов’язкова технологія, роботодавець — локація) реалізовано через окремі асоціативні таблиці, у яких додатково зберігаються атрибути зв’язку, як-от рівень володіння технологією або мінімальний рекомендований рівень для вакансії.

За функціональним призначенням сутності бази даних поділяються на сім логічних груп: обліковими записами й автентифікацією, профілями користувачів двох ролей, професійними даними фахівців (стек, освіта, досвід, сертифікати), довідниками технологій та локацій, вакансіями й відгуками, обміном повідомленнями, а також допоміжними сутностями для збереження файлів та історії взаємодій. Загалом схема містить понад двадцять моделей; повну функціональну схему з переліком атрибутів та зв’язків подано у графічних матеріалах (додаток Д2).

Доступ до бази даних здійснюється через ORM Prisma 6.5 з підходом schema-first: схему описано в декларативному форматі Prisma Schema Language, на основі якого автоматично генеруються типобезпечний клієнт TypeScript та SQL-міграції. Версіонування схеми через Prisma Migrate забезпечує відтворюваність структури бази у будь-якому середовищі (розробка, тестування, виробництво) і безпечне накочування змін без втрати даних. Усі первинні ключі реалізовано як UUID версії 4, що дає змогу генерувати ідентифікатори на стороні застосунку без звернення до СУБД і виключає колізії при масштабуванні.

Для забезпечення продуктивності пошукових та аналітичних запитів

на критичних полях створено вторинні індекси: за електронною адресою користувача, за ідентифікатором вакансії у таблицях відгуків та переглядів, за часовими полями для часових діапазонних запитів. Перелічувальні типи (ролі, рівні кваліфікації, рівні володіння технологією, типи зайнятості, статуси відгуків) реалізовано як нативні enum-типи PostgreSQL, що поєднує суворий контроль значень на рівні СУБД з компактним зберіганням.

У всій схемі застосовано підхід м'якого видалення (soft delete): ключові сутності містять поле deletedAt типу DateTime, яке заповнюється поточним моментом часу при «видаленні» запису замість фізичного вилучення рядка з таблиці. Технічну реалізацію підходу через проміжне програмне забезпечення Prisma описано у підрозділі 3.2.2.

3.2 Реалізація серверної частини

3.2.1 Організація структури модулів серверної частини

Кожен модуль NestJS оформлений за єдиним структурним шаблоном і містить такі складові: файл module.ts з декоратором @Module, файл controller.ts з методами обробки HTTP-запитів, файл service.ts з бізнес-логікою та піддиректорію dto/ з класами Data Transfer Object для валідації вхідних даних засобами бібліотеки class-validator. Зв'язок між модулями реалізовано через механізм ін'єкції залежностей NestJS, що відповідає принципу інверсії керування, описаному у підрозділі 2.2.1.

Кореневий модуль AppModule, окрім бізнес-модулів, імпортує також конфігураційні: ConfigModule з валідацією змінних оточення за схемою, ThrottlerModule для обмеження частоти запитів та SessionMiddleware для тимчасової сесії під час обміну з Google OAuth. На рівні AppModule зареєстровано глобальні охоронці JwtAuthGuard і RolesGuard через провайдер з токеном APP_GUARD.

3.2.2 Проектування та реалізація бази даних

Структура бази даних описана у файлі schema.prisma у директорії apps/backend/prisma/. На основі цієї схеми Prisma генерує типобезпечний клієнт TypeScript та SQL-міграції.

Сервіс PrismaService інкапсулює клієнт Prisma і реалізує методи onModuleInit та onModuleDestroy для коректного управління життєвим циклом з'єднання. У межах PrismaService реалізовано проміжне програмне забезпечення для автоматичного застосування підходу soft delete: перед операціями findMany, findFirst і findUnique до умов запиту автоматично додається фільтр {deletedAt: null}; операції delete та deleteMany замінюються на update з присвоєнням поточного моменту часу полю deletedAt.

Міграційну стратегію реалізовано через інструмент Prisma Migrate. Початкове заповнення довідників (навички, спеціалізації, країни, міста) виконується за допомогою скрипта seed.ts.

3.2.3 Реалізація API ендпоінтів та логіки обробки запитів

Серверна частина експонує REST API, який охоплює всі функціональні можливості системи та надає понад шістдесят ендпоінтів. Усі ендпоінти повертають дані у форматі JSON і використовують стандартні HTTP-методи відповідно до архітектурного стилю REST.

Зведений перелік ендпоінтів за функціональними групами подано у таблиці 3.3. Окрім явних перевірок ролей, усі ендпоінти, що модифікують стан системи, виконують додаткові перевірки прав доступу: ендпоінти редагування власних ресурсів перевіряють збіг ідентифікатора користувача з власником ресурсу; ендпоінти, пов'язані з відгуками, контролюють володіння відповідною вакансією; ендпоінти доступу до повідомлень перевіряють участь користувача у діалозі.

Таблиця 3.3 — Зведений перелік ендпоінтів серверного API

Група	Ендпоінти (метод і шлях)	Доступ
Автентифікація	`/auth/register`, `/auth/login`, `/auth/refresh`, `/auth/logout`, `/auth/me`, `/auth/confirm-email`, `/auth/request-password-reset`, `/auth/reset-password`, `/auth/google`, `/auth/google/callback`	Public, JWT, 10/хв
Облікові записи	PATCH `/users/me/password`, PATCH `/users/me/email`, DELETE `/users/me`	JWT
Профілі фахівців	CRUD `/specialists/profile`, GET `/specialists/:id`, CRUD підресурсів: skills, experience, education, certificates	SPECIALIST, Public
Профілі роботодавців	CRUD `/employers/profile`, GET `/employers/:id`, CRUD `/employers/profile/locations/:cityId`	EMPLOYER, Public
Вакансії	GET `/vacancies`, POST `/vacancies`, GET `/vacancies/my`, GET, PATCH, DELETE `/vacancies/:id`, CRUD `/vacancies/:id/skills/:skillId`	Public, EMPLOYER
Відгуки	POST `/applications`, GET `/applications/my`, GET `/applications/vacancy/:vacancyId`, PATCH `/applications/:id/status`, DELETE `/applications/:id`	SPECIALIST, EMPLOYER
Пошук і рекомендації	GET `/search/vacancies`, GET `/search/specialists`, POST `/search/reindex`, GET `/recommendations/vacancies`, GET `/recommendations/specialists/:vacancyId`	Public, JWT, ADMIN
Файли	POST `/files/avatar`, `/files/resume`, `/files/logo`, `/files/certificate/:id`, DELETE `/files/:fileId`	SPECIALIST, EMPLOYER

Кінець таблиці 3.3

Група	Ендпоінти (метод і шлях)	Доступ
Повідомлення	POST `/messages`, GET `/messages/conversations`, GET `/messages/conversations/:id`, PATCH `/messages/conversations/:id/read`	JWT (учасник)
Довідники	GET `/skills`, CRUD `/skills/:id`, GET `/specializations`, GET `/locations/countries`, GET `/locations/cities`	Public, ADMIN

3.2.4 Реалізація алгоритму гібридного пошуку фахівців

Гібридний пошук є ключовим алгоритмічним рішенням розробленого програмного забезпечення. Технічну основу реалізації складає пошуковий рушій Elasticsearch версії 8.19.1. У межах системи створено два пошукові індекси для вакансій і для профілів фахівців.

Індекс вакансій містить текстові поля (заголовок, опис, назва спеціалізації, назва компанії), числові поля (межі заробітної плати) та категоріальні поля (рівень кваліфікації, тип зайнятості, ознака віддаленої роботи). У документі окремо виділено вкладене (nested) поле skills з підполями skillId, skillName, isRequired та minProficiencyLevel. Індекс профілів фахівців містить аналогічні поля для опису кандидата. Індексування виконується асинхронно після кожної зміни сутності у базі даних.

Підсумкова оцінка релевантності фахівця обчислюється як зважена сума п’яти компонентів:

$$\begin{aligned}\text{Score} = W_{\text{BM25}} \cdot \text{BM25}_{\text{norm}} + W_{\text{SKILL}} \cdot \text{SkillMatch} + \\ + W_{\text{LEVEL}} \cdot \text{LevelMatch} + W_{\text{LOCATION}} \cdot \text{LocationMatch} + \\ + W_{\text{SALARY}} \cdot \text{SalaryMatch}, \quad (3.1)\end{aligned}$$

де

$BM25_{norm}$ – нормалізована оцінка текстової релевантності, обчислена Elasticsearch за формулою (2.4);

SkillMatch – оцінка відповідності технологічного стеку вимогам вакансії;

LevelMatch – оцінка відповідності рівня кваліфікації;

LocationMatch – оцінка географічної відповідності;

SalaryMatch – оцінка відповідності зарплатних очікувань;

$W_{BM25}, W_{SKILL}, W_{LEVEL}, W_{LOCATION}, W_{SALARY}$ – вагові коефіцієнти, винесені у бібліотеку shared-constants.

Значення вагових коефіцієнтів у поточній реалізації наведено в таблиці 3.4.

Таблиця 3.4 — Вагові коефіцієнти алгоритму гібридного пошуку

Коефіцієнт	Значення	Компонент
W_{BM25}	0,30	Текстова релевантність
W_{SKILL}	0,35	Збіг технологій
W_{LEVEL}	0,15	Рівень кваліфікації
$W_{LOCATION}$	0,10	Локація
W_{SALARY}	0,10	Зарплатні очікування

Компонент SkillMatch обчислюється як середнє внесків кожної обов'язкової технології, заявленої у вакансії. Внесок визначається залежно від наявності технології у профілі кандидата та відповідності його рівня володіння вимагуваному: для рівнів INTERMEDIATE і вище приймається значення 1,0; для рівня BEGINNER – 0,5; за відсутності технології у профілі – 0,0:

$$\text{SkillMatch} = \frac{1}{n} \cdot \sum_{i=1}^{n} \text{contrib}(i), \tag{3.2}$$

де

n – кількість обов'язкових технологій вакансії;

contrib(i) – внесок i-ї технології.

Компонент LevelMatch обчислюється з рівнями, кодованими цілими числами від 1 (Junior) до 5 (Architect): точному збігу відповідає значення 1,0; різниці в один рівень – 0,7; різниці у два рівні – 0,3; більшій різниці – 0,0. Компонент LocationMatch набуває таких значень: збіг міст дає 1,0; збіг лише за країною – 0,5; сумісність формату віддаленої роботи – 0,5; інакше – 0,0. Компонент SalaryMatch: якщо очікувана заробітна плата не перевищує максимальної межі бюджету, приймається значення 1,0; при перевищенні до 20 % значення лінійно зменшується; при більшому перевищенні приймається 0,0.

Об'єднання перелічених компонентів на рівні Elasticsearch виконується через механізм function_score, описаний у підрозділі 2.4.3, з додаванням функції загасання за часом створення документа на основі функції Гауса (масштаб 30 днів, коефіцієнт 0,5). Загальну послідовність дій алгоритму пошуку подано у графічних матеріалах (додаток Д3).

3.2.5 Реалізація рекомендаційної системи

Рекомендаційна система реалізує гібридний підхід, що поєднує контент-базовану фільтрацію на основі косинусної подібності векторів навичок з колаборативною складовою, яка використовує історію переглядів профілів.

Рекомендаційна функціональність зосереджена у модулі RecommendationsModule, який містить п'ять основних сервісів: VectorizerService (побудова векторних подань), ContentBasedService (контент-базовані рекомендації), CollaborativeService (колаборативні рекомендації), RecommendationsService (об'єднання двох підходів),

ProfileViewsService (фіксація переглядів).

Вектор фахівця формується як перелік числових значень рівнів володіння кожною технологією: 1 для BEGINNER, 2 для INTERMEDIATE, 3 для ADVANCED, 4 для EXPERT; технології, якими кандидат не володіє, представлено нульовими компонентами. Вектор вакансії будується аналогічно, проте значення компонентів, що відповідають обов'язковим технологіям, подвоюються, унаслідок чого вага цих технологій у косинусній подібності зростає.

Базова контент-базована оцінка обчислюється як косинусна подібність вектора фахівця та вектора вакансії за формулою (2.7). Показник покриття coverage обчислюється як частка обов'язкових технологій, наявних у фахівця на достатньому рівні:

$$\text{coverage} = \frac{|\text{required} \cap \text{specialist\_skills}|}{|\text{required}|}. \tag{3.3}$$

Підсумкова контент-базована оцінка є зваженою сумою:

$$\text{ContentScore} = 0{,}7 \times \text{cosine} + 0{,}3 \times \text{coverage}. \tag{3.4}$$

Колаборативну складову реалізовано на основі сутності ProfileView. Подібність між фахівцями обчислюється як коефіцієнт Жаккара множин роботодавців, які переглядали їхні профілі, відповідно до формули (2.8). Поєднання двох підходів виконується у вигляді зваженої лінійної комбінації. Ключовою особливістю реалізації є динамічне визначення коефіцієнта α:

$$\text{FinalScore} = \alpha \cdot \text{ContentScore} + (1 - \alpha) \cdot \text{CollabScore}, \tag{3.5}$$

$$\alpha = \begin{cases} 1{,}0, & \text{якщо count(views)} < 5, \\ 0{,}6, & \text{якщо count(views)} \geq 5. \end{cases} \tag{3.6}$$

Якщо у користувача накопичено менше п’яти переглядів, обирається режим $\alpha = 1{,}0$ (cold start, тільки контент-базовані рекомендації). Після досягнення порогу значення коефіцієнта переходить до $\alpha = 0{,}6$ (warm start з домінуванням контент-базованої складової). Результати обчислень кешуються у Redis з часом життя 3600 секунд. Ключі кешу формуються за шаблонами rec:vacancies:{userId}:{limit} та rec:specialists:{userId}:{vacancyId}:{limit}. Передбачено механізм коректної деградації при тимчасовій недоступності Redis. Загальну послідовність дій рекомендаційної системи подано у графічних матеріалах (додаток Д3).

3.2.6 Механізми безпеки серверної частини

Безпеку серверної частини організовано на чотирьох рівнях.

Перший рівень охоплює автентифікацію та керування сесіями. Застосовано пару токенів JWT: короткоживучий access-токен з терміном дії 15 хвилин і довгоживучий refresh-токен з терміном дії 7 днів. Refresh-токени зберігаються в окремій таблиці RefreshToken; підтримуються ротація токенів (нове значення видається при кожному використанні) та примусове відкликання. Паролі хешуються алгоритмом bcrypt з 12 раундами. Окремо реалізовано блокування облікового запису після п’яти послідовних невдалих спроб входу (поле isLocked моделі User) з аудитом у таблиці LoginAttempt. Підтвердження електронних адрес та скидання паролів виконується через одноразові токени EmailConfirmationToken та PasswordResetToken з обмеженим терміном дії.

Другий рівень відповідає за авторизацію. Глобальний JwtAuthGuard перевіряє валідність access-токену для всіх ендпоінтів, окрім позначених @Public() або @OptionalJwt(). Глобальний RolesGuard перевіряє відповідність ролі поточного користувача переліку, заданому декоратором @Roles(). Підтримуються три ролі: SPECIALIST, EMPLOYER, ADMIN. Інтеграцію з Google OAuth 2.0 реалізовано через бібліотеку Passport.js зі стратегіями local,

jwt та google.

Третій рівень охоплює валідацію та обмеження частоти запитів. Глобальний ValidationPipe з параметрами whitelist, forbidNonWhitelisted і transform забезпечує суворе дотримання контракту прикладного інтерфейсу. ThrottlerModule обмежує загальну частоту запитів значенням 100 запитів на хвилину з однієї IP-адреси та 10 запитів на хвилину для ендпоінтів автентифікації, що ускладнює атаки brute force.

Четвертий рівень відповідає за захист від міждоменних запитів. Налаштування CORS обмежує дозволені джерела виключно адресою клієнтської частини, заданою через змінну оточення APP_FRONTEND_URL.

3.2.7 Обробка завантаження файлів через Firebase Storage

Підсистему завантаження файлів реалізовано у модулі FilesModule на основі Firebase Admin SDK. Інтеграція налаштовується через сервісний обліковий запис Firebase, JSON-файл якого вказується змінною оточення FIREBASE_SERVICE_ACCOUNT_PATH, та ідентифікатор сховища, заданий змінною FIREBASE_STORAGE_BUCKET.

Завантаження файлу відбувається через мультипарт-запит HTTP з використанням перехоплювача FileInterceptor бібліотеки multer (memoryStorage). Послідовність обробки файлу між компонентами системи подана на рисунку 3.2.

Унікальний ключ об’єкта генерується за шаблоном {category}/{userId}/{uuid}.{ext}, де category набуває одного зі значень avatars, resumes, logos або certificates. Після завантаження файл стає публічно доступним через виклик makePublic(), що дає змогу отримати прямий URL для подальшого використання. Видалення виконується у зворотному порядку: спочатку об’єкт вилучається з Firebase Storage, після чого видаляється відповідний запис у таблиці File. Аватарки та логотипи приймаються

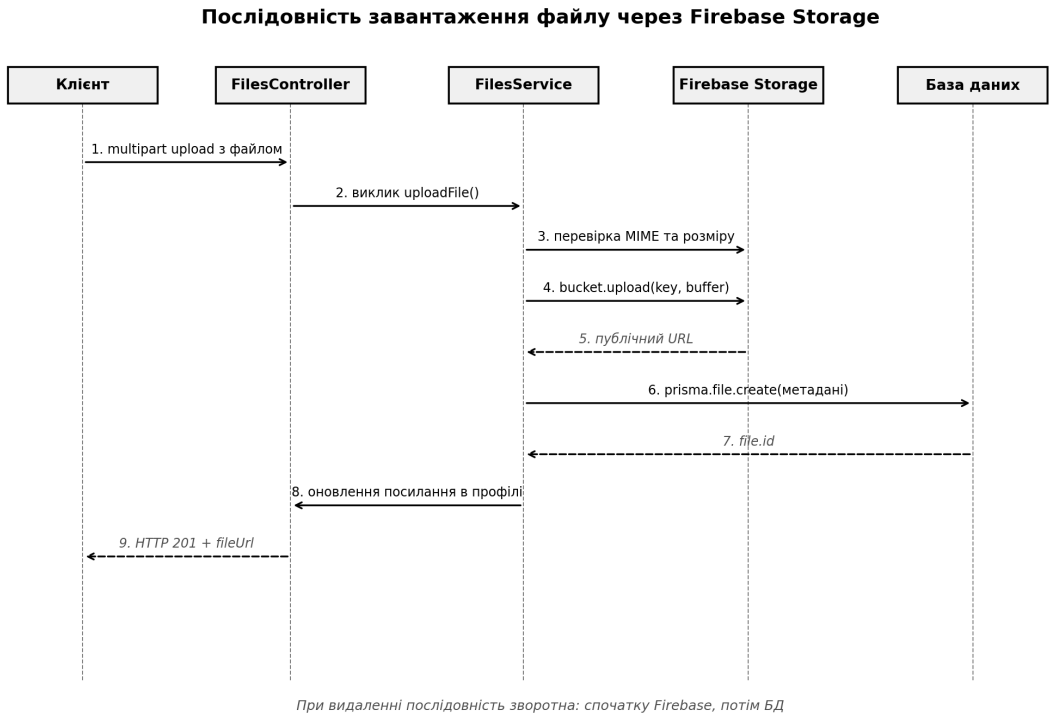


Рисунок 3.2 — Послідовність завантаження файлу через Firebase Storage

у форматах JPEG, PNG, WEBP з максимальним розміром 5 МБ; резюме та сертифікати – у форматі PDF або зображень з максимальним розміром 20 МБ.

3.3 Реалізація клієнтської частини

3.3.1 Організація структури модулів клієнтської частини

Клієнтська частина має ієрархічну структуру директорій у apps/frontend/src/. Маршрутні сторінки зосереджено в директорії pages/, у якій виокремлено піддиректорії auth/, specialist/, employer/ та messages/. Перевикористовувані компоненти розміщено в components/, поділеній за призначенням: layout/ містить макети, ui/ – базові елементи інтерфейсу, shared/ – спеціалізовані компоненти на кшталт MatchScore та LevelBar. Шар api/ інкапсулює HTTP-клієнт Axios та модулі за доменами; директорія store/

містить Zustand-сховище, а lib/ – допоміжні утиліти.

3.3.2 Розробка ключових сторінок та компонентів інтерфейсу

Сторінки клієнтського застосунку поділяються на чотири групи відповідно до ролі користувача. До сторінок фахівця належать: VacanciesListPage (список вакансій з фільтрами та повнотекстовим пошуком), VacancyDetailPage (деталі вакансії з формою відгуку), RecommendationsPage (персоналізовані рекомендації з відсотком відповідності), ApplicationsPage (список власних відгуків), SpecialistProfilePage та SpecialistProfileEditPage (власний профіль), SpecialistViewPage (публічний перегляд чужого профілю).

Сторінки роботодавця охоплюють EmployerVacanciesPage (список власних вакансій), VacancyFormPage (універсальну форму створення і редагування вакансії), VacancyApplicantsPage (список відгуків з можливістю зміни статусу), EmployerProfilePage (профіль компанії) та SpecialistsSearchPage (пошук фахівців за гібридним алгоритмом). Сторінки повідомлень – це ConversationsPage (список діалогів) та ChatPage (вікно діалогу з формою надсилання повідомлень).

До ключових перевикористовуваних компонентів належать: VacancyCard (картка вакансії з основною інформацією та технологіями), MatchScore (візуалізація відсотка відповідності у вигляді горизонтальної смуги прогресу), LevelBar (графічне зображення рівня кваліфікації від JUNIOR до ARCHITECT), Header (верхня навігаційна панель з посиланнями, що відрізняються для різних ролей).

Стилізацію компонентів виконано засобами CSS-фреймворку Tailwind за принципом utility-first. Колірна палітра, типографіка та параметри інтерфейсу витримані у єдиному стилі чистого мінімалізму.

3.3.3 Керування станом та організація навігації

У клієнтській частині керування станом поділено на дві категорії. Глобальний клієнтський стан зберігається у Zustand-сховищі auth.store.ts і містить поля user (об'єкт з ідентифікатором, електронною адресою та роллю або null), accessToken та refreshToken з функціями-сетерами setTokens, setUser і logout. Через проміжне програмне забезпечення persist стан зберігається у браузерному локальному сховищі, що забезпечує автоматичне відновлення сесії після перезавантаження сторінки.

Серверний стан обслуговується бібліотекою TanStack React Query з часом застарівання кешу (staleTime) 30 секунд. Інвалідація кешу після операцій модифікації виконується через виклик queryClient.invalidateQueries з ключем відповідного запиту.

Маршрутизацію реалізовано засобами бібліотеки React Router 6 за допомогою конструктора createBrowserRouter з підтримкою вкладених маршрутів. Компонент AppLayout виступає батьківським макетом для всіх автентифікованих сторінок, AuthLayout – для сторінок входу та реєстрації. Захист маршрутів виконується через перевірку наявності користувача у Zustand-сховищі з перенаправленням на сторінку входу за відсутності автентифікації. Додатково передбачено перевірку ролі для маршрутів, призначених для конкретного типу користувача.

3.3.4 Взаємодія з серверним API на клієнті

Звернення клієнтської частини до серверного прикладного інтерфейсу організовано через шар api/, який містить модулі для кожної доменної області системи: auth.api.ts, specialists.api.ts, employers.api.ts, vacancies.api.ts, applications.api.ts, search.api.ts, recommendations.api.ts, messages.api.ts, skills.api.ts, specializations.api.ts та locations.api.ts. Кожен модуль містить

типізовані функції з імпортом типів аргументів і результатів з бібліотеки shared-types.

Як приклад розглянемо повний потік обробки запиту на подання відгуку на вакансію фахівцем. Послідовність взаємодій між компонентами системи від натискання кнопки до оновлення стану інтерфейсу подана на рисунку 3.3.

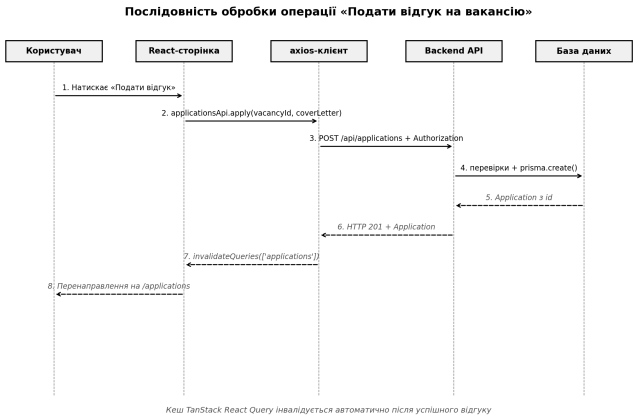


Рисунок 3.3 — Послідовність обробки операції «Подати відгук на вакансію»

Серверна частина приймає запит, послідовно проходить ланцюг охоронців для перевірки автентифікації та ролі SPECIALIST, після чого передає управління сервісу ApplicationsService для виконання бізнес-логіки: перевірки наявності профілю фахівця, відсутності дублікату відгуку та опублікованості вакансії. Створений запис повертається клієнтові зі статус-кодом 201. На клієнтському боці TanStack React Query інвалідує кеш списку відгуків фахівця, унаслідок чого сторінка ApplicationsPage автоматично оновлюється при наступному переході. Такий підхід забезпечує консистентність даних між сторінками без необхідності ручного оновлення стану.

ВИСНОВКИ ДО РОЗДІЛУ

У третьому розділі подано детальний опис розробленого програмного комплексу: архітектурні рішення, реалізацію серверної та клієнтської частин, а також деталі реалізації гібридного пошуку та рекомендаційної системи.

Серверну частину побудовано на прикладному фреймворку NestJS і організовано у вісімнадцять функціонально відокремлених модулів. Базу даних реалізовано на СУБД PostgreSQL з використанням ORM Prisma; у схемі міститься понад двадцять моделей з підтримкою «м'якого видалення». Серверний прикладний інтерфейс експонує понад шістдесят ендпоінтів, що повністю покривають визначені у першому розділі функціональні вимоги.

Реалізовано гібридний алгоритм пошуку фахівців на основі Elasticsearch, у якому текстова релевантність за моделлю BM25 поєднується з чотирма додатковими компонентами оцінки відповідності, та гібридну рекомендаційну систему з контент-базованою фільтрацією і колаборативною складовою, у якій коефіцієнт об'єднання визначається динамічно залежно від обсягу історії взаємодій користувача. Безпеку серверної частини організовано на чотирьох рівнях (автентифікація JWT, авторизація за ролями, валідація вхідних даних, обмеження частоти запитів); клієнтську частину побудовано як SPA на React з поділом стану на глобальний (Zustand) та серверний (TanStack React Query).

Розроблений програмний комплекс повністю відповідає функціональним та нефункціональним вимогам, сформульованим у першому розділі, та реалізує всі ключові механізми сучасного підбору IT-фахівців. Оцінка ефективності розроблених алгоритмів і функціональне тестування розглянуто у наступному розділі пояснювальної записки.

РОЗДІЛ 4
ТЕСТУВАННЯ ТА АНАЛІЗ РОЗРОБЛЕНОГО ПРОЄКТУ

4.1 Демонстрація роботи та функціональне тестування веб-системи

У цьому підрозділі наведено результати функціонального тестування реалізованого програмного комплексу за основними сценаріями взаємодії користувачів. Для кожного сценарію подано короткий опис послідовності дій, очікуваний результат та екранне відображення відповідного стану інтерфейсу. Тестування виконано на локальному розгортанні системи через Docker Compose; усі сценарії перевірено на коректну поведінку при штатному використанні та на обробку помилкових ситуацій.

4.1.1 Реєстрація та автентифікація користувача

Сценарій починається з відкриття сторінки реєстрації, на якій користувач вводить електронну адресу, пароль із підтвердженням та обирає роль (фахівець або роботодавець). Після надсилання форми серверна частина виконує валідацію вхідних даних, створює запис у таблиці User зі статусом непідтвердженого облікового запису та надсилає повідомлення з посиланням для активації на вказану адресу. Альтернативний сценарій передбачає автентифікацію через Google за протоколом OAuth 2.0 у потоці Authorization Code Grant. Сторінка реєстрації наведена на рисунку 4.1.

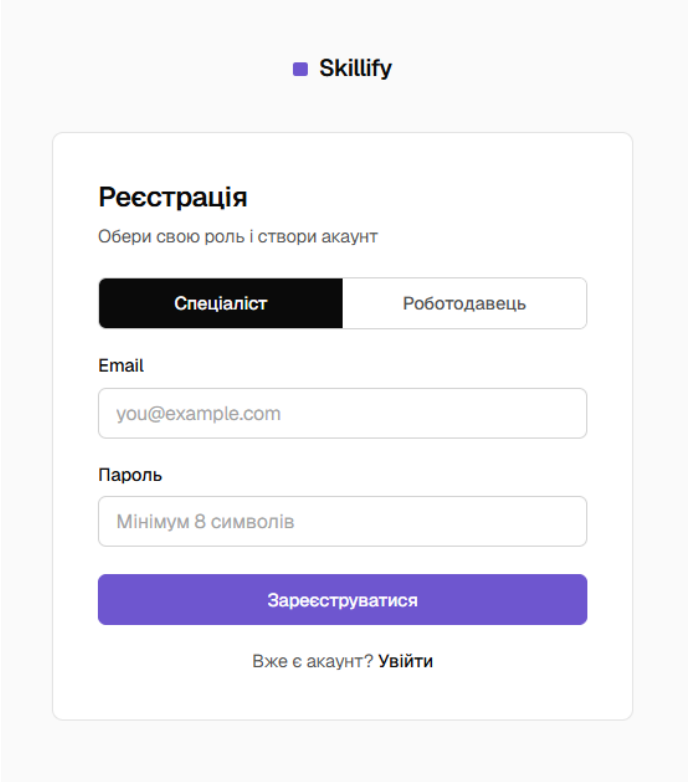


Рисунок 4.1 — Сторінка реєстрації користувача

Після підтвердження адреси користувач здійснює вхід через стандартну форму з полями електронної адреси й паролю. У разі успішної автентифікації серверна частина видає пару токенів JWT (access на 15 хвилин та refresh на 7 днів) і відповідно до ролі переадресовує користувача: фахівців на сторінку перегляду доступних вакансій, роботодавців на сторінку пошуку фахівців. Перевірено реакцію системи на помилкові сценарії: введення непідтвердженої адреси повертає статус 403 з повідомленням про необхідність активації; п’ять послідовних невдалих спроб входу призводять до тимчасового блокування облікового запису та запису події у таблицю LoginAttempt.

4.1.2 Створення та налаштування профілю фахівця

Після першого входу новостворений фахівець перенаправляється на сторінку покрокового заповнення профілю. На послідовних кроках вказуються особисті дані, технологічний стек з рівнями володіння, обрана спеціалізація, рівень кваліфікації, досвід роботи, освіта, сертифікати, бажана заробітна плата, тип зайнятості та географічна локація. Файли (аватарка, документ резюме, зображення сертифікатів) завантажуються у Firebase Storage через відповідні ендпоінти серверної частини.

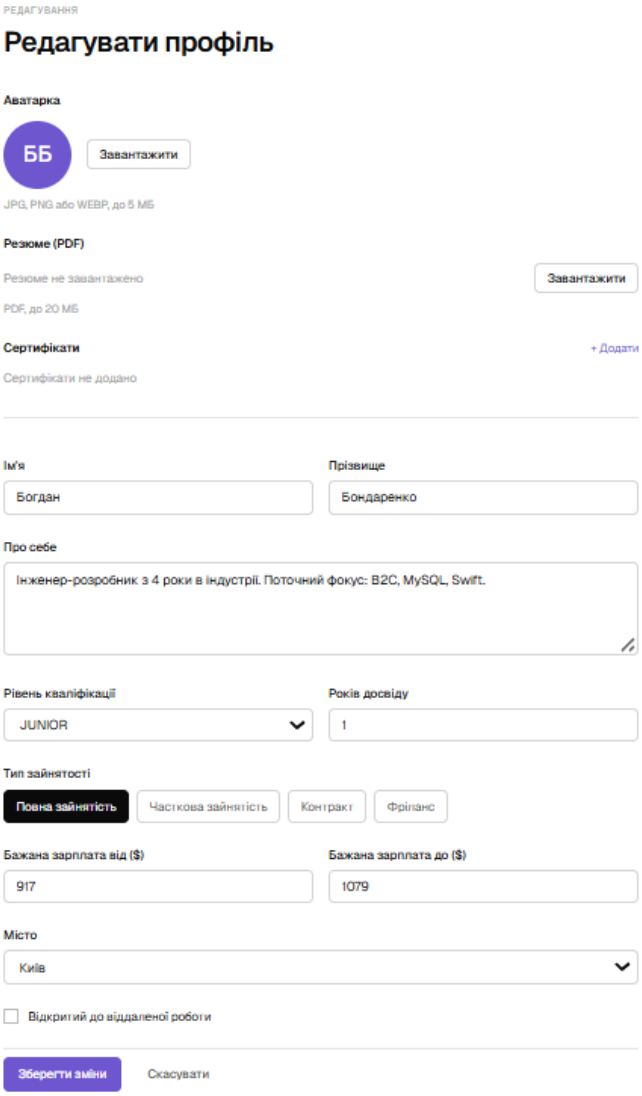


Рисунок 4.2 — Форма редагування профілю фахівця

Після збереження дані профілю індексуються у пошуковому рушії Elasticsearch для подальшого використання в алгоритмі гібридного пошуку, а векторне подання навичок зберігається для рекомендаційної підсистеми. Готовий профіль відображається в режимі перегляду з усіма заповненими розділами.

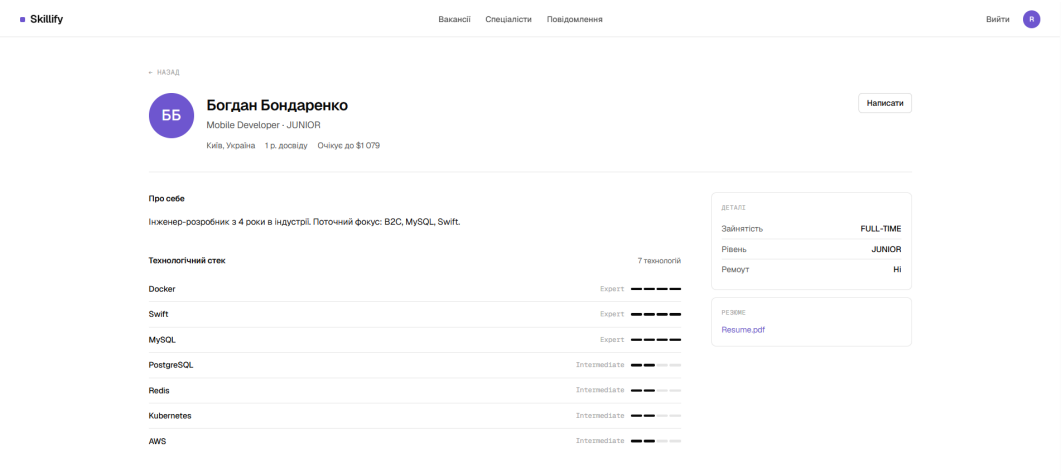


Рисунок 4.3 — Сторінка перегляду профілю фахівця

4.1.3 Створення та публікація вакансії роботодавцем

Роботодавець після заповнення профілю компанії переходить до форми створення вакансії. У формі задаються заголовок, опис, рівень кваліфікації, тип зайнятості, межі заробітної плати, географічна локація та ознака можливості віддаленої роботи. На окремому кроці вказуються обов’язкові технології з мінімальним рівнем володіння для кожної з них.

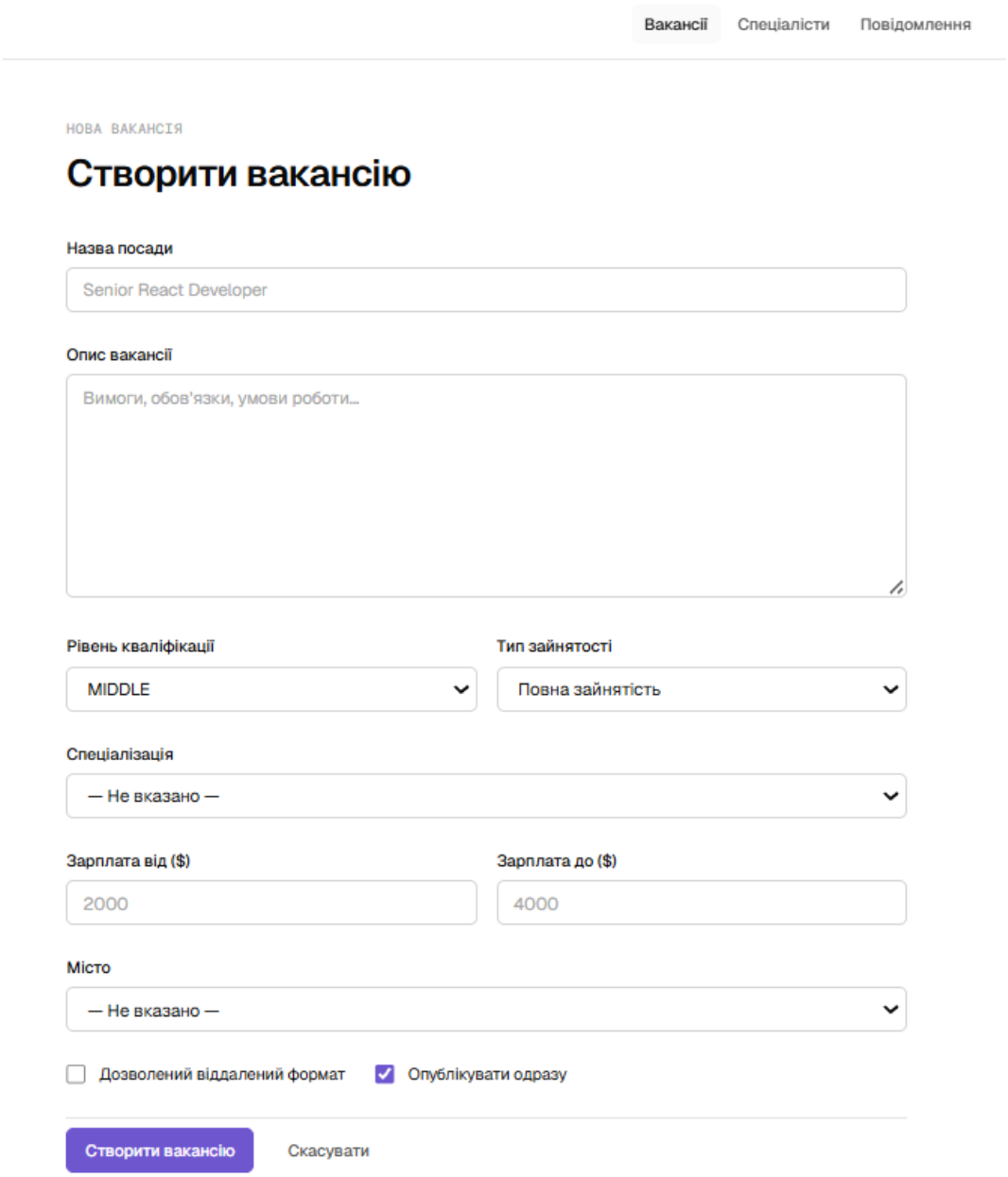


Рисунок 4.4 — Форма створення вакансії роботодавцем

Після збереження вакансія переводиться у статус опублікованої, додається до загального переліку та індексується у пошуковому рушії. Реакція системи на спробу збереження вакансії з порожніми обов’язковими полями повертає валідаційну помилку зі статусом 400; спроба несанкціонованого створення з облікового запису ролі SPECIALIST повертає статус 403.

4.1.4 Пошук фахівців за критеріями

Для перевірки роботи гібридного алгоритму пошуку роботодавець відкриває сторінку пошуку фахівців, вводить пошуковий запит та обирає додаткові фільтри: рівень кваліфікації, локація, межі бажаної оплати, обов'язкові технології. Клієнтська частина надсилає запит на серверну частину, яка побудовує запит до Elasticsearch з механізмом function_score, де BM25-релевантність текстових полів комбінується з обчисленими у часі запиту функціями SkillMatch, LevelMatch, LocationMatch та SalaryMatch відповідно до формули (3.1).

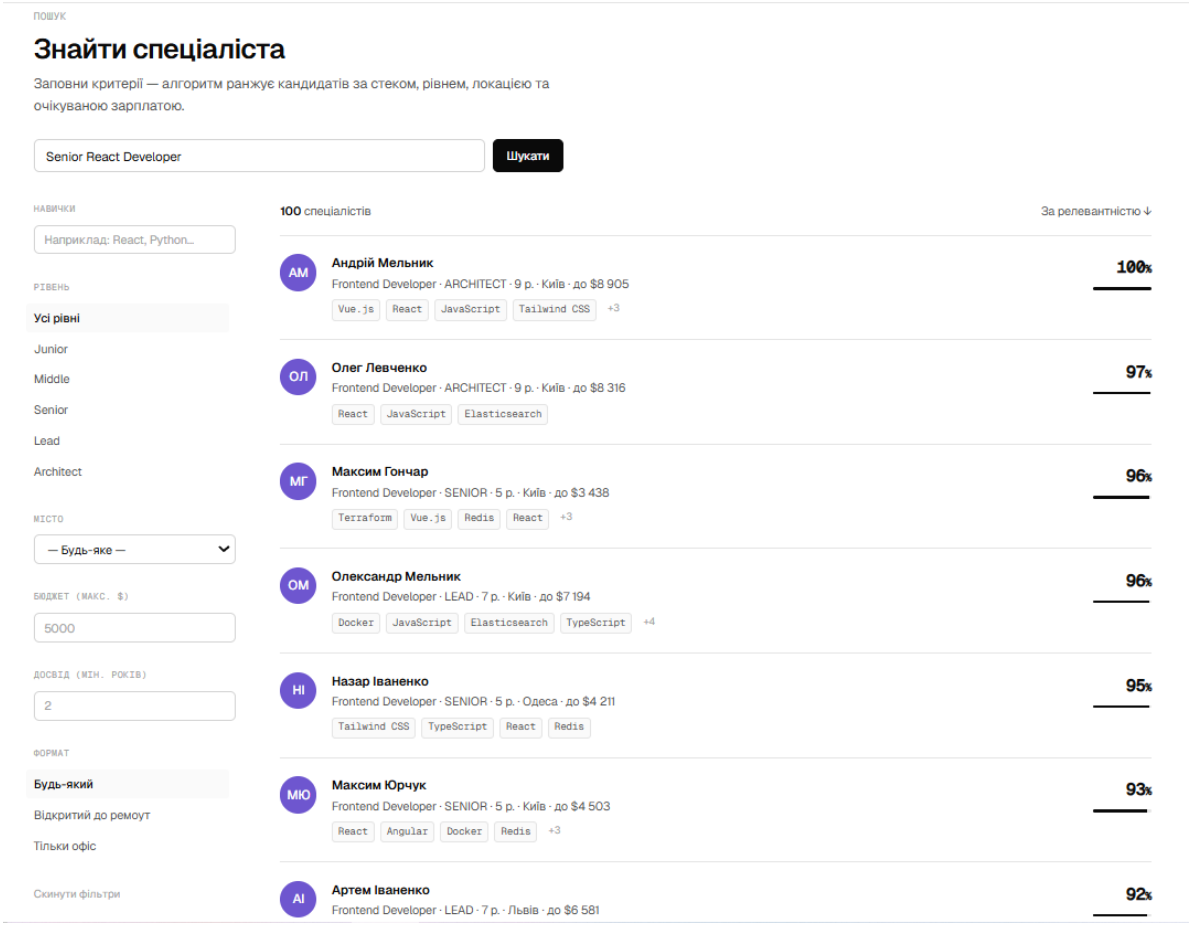


Рисунок 4.5 — Результати гібридного пошуку фахівців з відсотками відповідності

Біля кожного результату відображається числовий відсоток відповід-

ності, що дозволяє роботодавцю візуально оцінити релевантність кандидатів і прийняти рішення про звернення. Перелік упорядкований за спаданням підсумкової оцінки.

4.1.5 Перегляд персоналізованих рекомендацій

Для фахівця сторінка персоналізованих рекомендацій містить підбірку вакансій, сформовану рекомендаційною підсистемою. Для нового користувача з невеликою історією переглядів використовується режим $\alpha = 1{,}0$, тобто виключно контент-базована фільтрація; після накопичення достатньої історії взаємодій система переходить у режим $\alpha = 0{,}6$, в якому контент-базована складова доповнюється колаборативними сигналами.

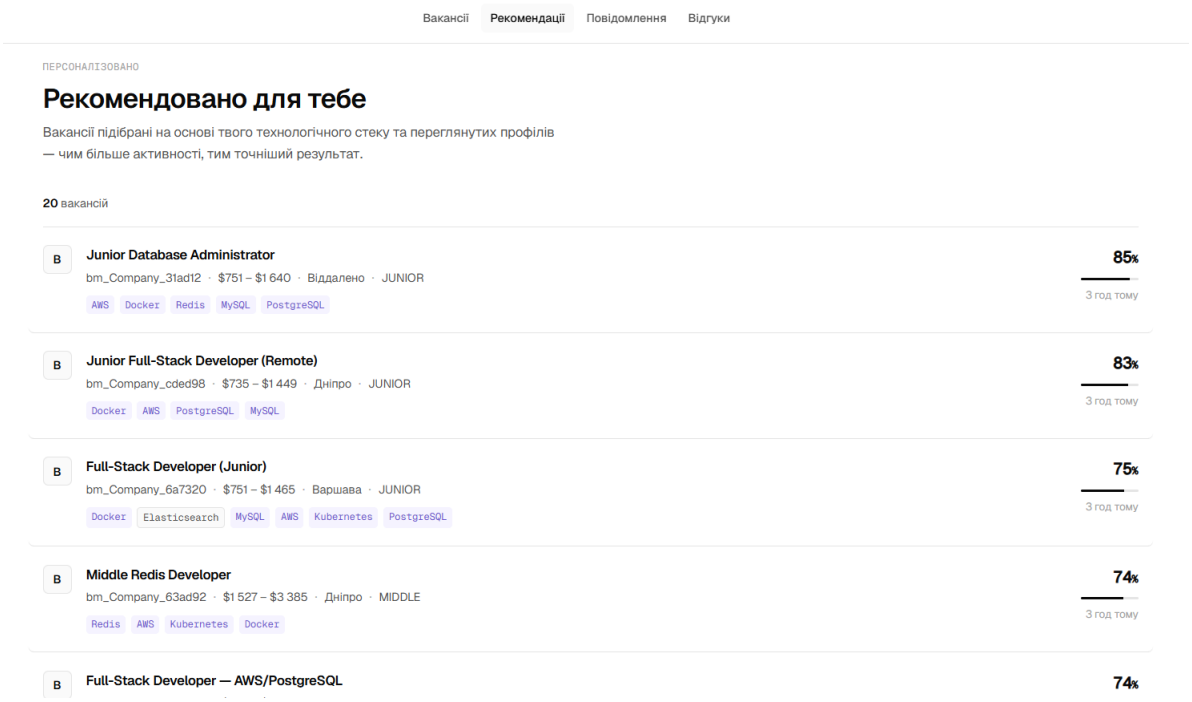


Рисунок 4.6 — Персоналізовані рекомендації вакансій для фахівця

Аналогічна функціональність реалізована для роботодавця: на сторінці деталей вакансії додається блок рекомендованих фахівців, упорядкований за спаданням оцінки відповідності. Результати обчислень кешуються у Redis

з часом життя 3600 секунд, що забезпечує миттєвий відгук при повторних запитах у межах однієї години.

4.1.6 Надсилання відгуку на вакансію

Фахівець, переглядаючи деталі вакансії, ініціює подання відгуку через відповідну кнопку. У формі заповнюється необов'язковий супровідний лист, після чого виконується надсилання. Серверна частина перевіряє наявність профілю кандидата, відсутність попереднього відгуку на цю вакансію та її поточний статус, після чого створює запис Application зі статусом PENDING.

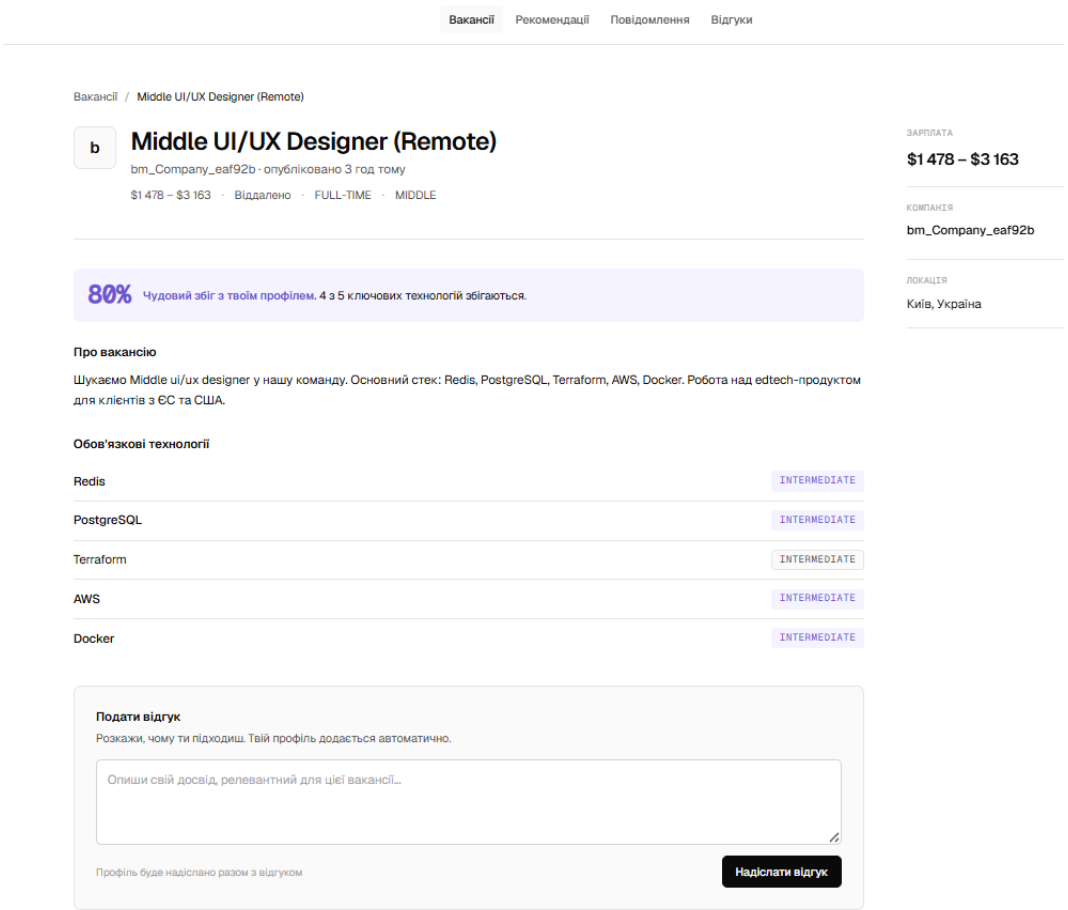


Рисунок 4.7 — Сторінка вакансії з формою подання відгуку

Подальша робота з відгуками з боку роботодавця відбувається на окремій сторінці, де відображається список відгуків на конкретну вакан-

сію з можливістю зміни статусу: PENDING, REVIEWED, INTERVIEW, ACCEPTED або REJECTED. Фахівець має дзеркальний перегляд власних відгуків з поточним статусом обробки.

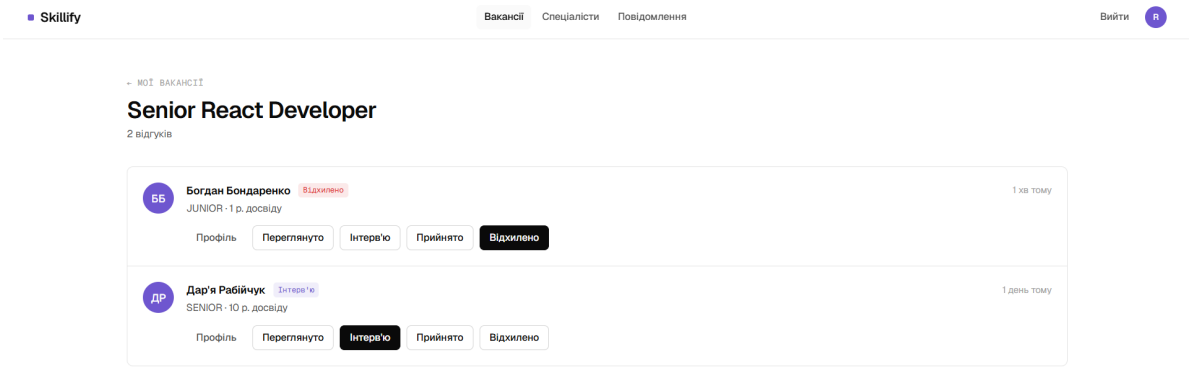


Рисунок 4.8 — Список відгуків на вакансію з боку роботодавця

4.1.7 Обмін повідомленнями між користувачами

Між фахівцями та роботодавцями реалізовано систему діалогів. Перехід на сторінку профілю іншого користувача містить кнопку ініціації розмови, натискання якої створює (або відкриває раніше створений) діалог. Список усіх діалогів поточного користувача відображається на окремій сторінці.

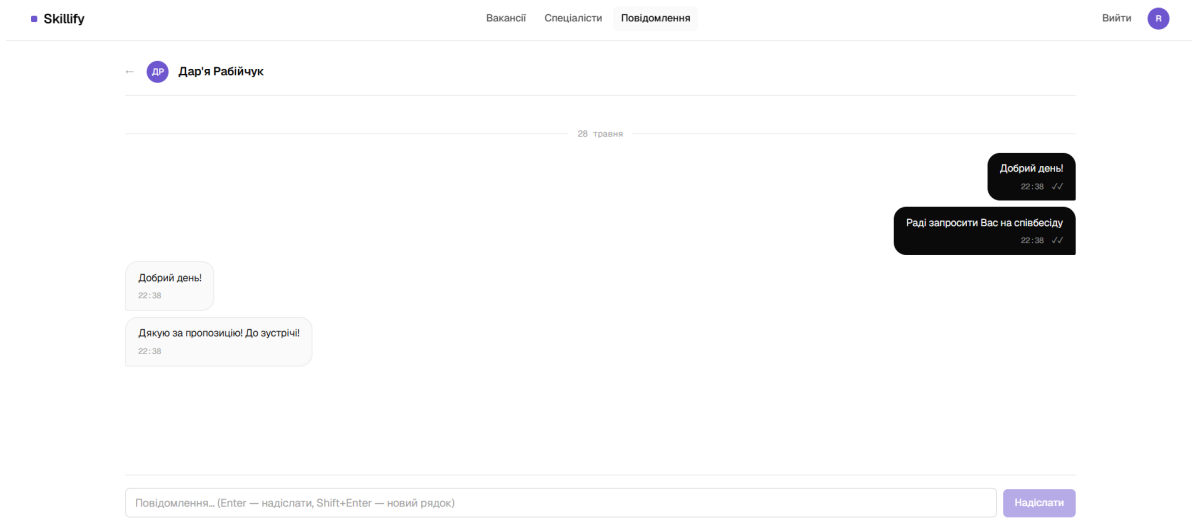


Рисунок 4.9 — Інтерфейс обміну повідомленнями між користувачами

Оновлення стану діалогу у відкритому вікні виконується методом polling з періодом 5 секунд, що відповідає прийнятому компромісу між простотою реалізації та оперативністю доставки повідомлень. Прочитані повідомлення позначаються відповідним маркером, що оновлюється через окремий ендпоінт серверної частини.

4.2 Оцінка ефективності алгоритму пошуку

Оскільки публічних анотованих наборів даних для задачі підбору IT-фахівців не існує, для оцінювання запропонованого алгоритму сформовано синтетичний бенчмарк зі 1 500 профілів фахівців і 150 модельних вакансій. Розподіли параметрів (рівні кваліфікації, діапазони заробітних плат, географічний розподіл) сформовано на основі відкритих даних звіту «IT Research Ukraine 2025» [1] та опитування ринку праці DOU [2].

На сформованому наборі даних проведено порівняння трьох стратегій ранжування: лише BM25, BM25 з компонентом SkillMatch та повна гібридна стратегія з усіма п’ятьма компонентами. Усереднені значення стандартних метрик інформаційного пошуку за першими десятьма результатами наведено

у таблиці 4.1.

Таблиця 4.1 — Метрики якості гібридного пошуку фахівців

Стратегія	Точність	Повнота	F-міра	MAP
Лише BM25	0,38	0,21	0,27	0,29
BM25 + SkillMatch	0,57	0,33	0,42	0,46
Повна гібридна	0,71	0,42	0,53	0,58

Дані таблиці 4.1 засвідчують поступове зростання якості пошуку при послідовному додаванні компонентів. Перехід від базової стратегії до проміжної (додавання SkillMatch) збільшує точність з 0,38 до 0,57, оскільки структуроване зіставлення технологій з рівнями володіння точніше моделює природу IT-добору, ніж лексичний збіг у тексті. Перехід до повної гібридної стратегії дає додатковий приріст на 14 п. п. за рахунок внеску компонентів LevelMatch, LocationMatch та SalaryMatch.

Медіанний час відгуку гібридного пошуку становить 240 мс на робочій станції з процесором AMD Ryzen 7 і 32 ГБ оперативної пам’яті з Elasticsearch у локальному Docker-контейнері, що укладається у нефункціональну вимогу «не більше ніж 1 с», сформульовану у першому розділі.

4.3 Оцінка якості рекомендаційної системи

Для оцінювання якості рекомендаційного модуля сформовано окремий тестовий набір даних, що моделює історію взаємодій користувачів із системою. На основі описаного раніше набору з 1 500 профілів згенеровано 7 500 подій перегляду профілів роботодавцями; розподіл активності відповідає типовій моделі ринку: 20 % роботодавців забезпечують 80 % переглядів. Поділ історії взаємодій між тренувальною (80 %) та тестовою (20 %) частинами виконано шляхом часового зсуву.

Порівнювалися три варіанти рекомендаційного підходу: лише контент-базований, лише колаборативний та гібридний з адаптивним ба-

лансуванням. Усереднені значення стандартних метрик якості за першими десятьма результатами наведено у таблиці 4.2.

Таблиця 4.2 — Метрики якості рекомендаційної системи

Стратегія	Точність	Повнота	F-міра	Частота попадань
Лише контент-базована	0,46	0,28	0,35	0,72
Лише колаборативна	0,52	0,31	0,39	0,74
Гібридна (адаптивна)	0,64	0,39	0,49	0,86

Гібридна стратегія перевершує кожну з базових за всіма метриками. Для користувачів з малою історією переглядів (менше п’яти) адаптивна гібридна схема забезпечує точність на рівні 0,42 проти 0,11 для чисто колаборативної. Це підтверджує доцільність динамічного перемикання режимів, описаного у підрозділі 3.2.5: проблема холодного старту практично усувається без шкоди для якості у режимі warm start.

Окремо досліджено вплив кешування у Redis на час відгуку: без кешу медіанний час обчислення рекомендацій становить 95 мс, з кешем — 35 мс. Експеримент з примусовим відключенням Redis підтвердив коректність деградації — система переходить у режим прямого обчислення без помилок користувача.

Узагальнюючи, експериментальне порівняння підтверджує доцільність обох ключових алгоритмічних рішень роботи: гібридне ранжування забезпечує помітно вищі показники якості порівняно з ізольованим BM25-ранжуванням, а гібридна рекомендаційна схема з адаптивним балансуванням ефективно поєднує переваги обох підходів і коректно обробляє ситуацію холодного старту.

ВИСНОВКИ ДО РОЗДІЛУ

У четвертому розділі виконано функціональне тестування реалізованого програмного комплексу та експериментальне оцінювання ефективності запропонованих алгоритмів. Функціональне тестування охопило сім основних сценаріїв взаємодії користувачів із системою — від реєстрації до обміну повідомленнями — усі сценарії виконано без помилок, що підтверджує відповідність реалізації функціональним вимогам.

Експериментальна оцінка гібридного пошуку на синтетичному наборі з 1 500 профілів і 150 вакансій засвідчила зростання точності з 0,38 до 0,71 та F-міри з 0,27 до 0,53 при послідовному додаванні компонентів у функцію ранжування; медіанний час відгуку повної гібридної стратегії становить 240 мс, що укладається у ліміт 1 с. Оцінка рекомендаційної системи на 7 500 подій перегляду показала, що гібридна схема з адаптивним балансуванням перевершує кожен з базових підходів за всіма метриками (точність — 0,64, повнота — 0,39, частота попадань — 0,86) і практично знімає проблему холодного старту; кешування у Redis скорочує час відгуку повторних запитів з 95 мс до 35 мс.

Отримані результати підтверджують ефективність запропонованих алгоритмічних рішень і їх відповідність функціональним та нефункціональним вимогам до програмного комплексу.

ВИСНОВКИ

У дипломній роботі розв’язано задачу алгоритмічного вдосконалення процесу пошуку та підбору фахівців у сфері інформаційних технологій. Спроєктовано архітектуру, програмно реалізовано прототип розподіленого програмного комплексу та виконано його експериментальну перевірку.

У ході виконання роботи отримано такі основні результати.

1. Проведено аналіз провідних платформ підбору IT-фахівців (Djinni, DOU.ua, LinkedIn, Robota.ua) та встановлено, що жодне з наявних рішень одночасно не поєднує BM25-ранжування, зваженого багатокритеріального оцінювання з налаштовуваними коефіцієнтами та гібридних рекомендаційних алгоритмів. На цій основі сформульовано вимоги до програмного забезпечення.
2. Розроблено модульну архітектуру системи: серверну частину організовано у вісімнадцять модулів NestJS, клієнтську — як SPA на React з поділом стану на глобальний (Zustand) та серверний (TanStack Query). Обрано стек: TypeScript, Node.js, NestJS, PostgreSQL з Prisma, Elasticsearch, Redis, React, Vite.
3. Запропоновано та реалізовано гібридний алгоритм ранжування пошуку фахівців, що обчислює інтегральну оцінку як зважену суму BM25-релевантності та чотирьох компонентів (SkillMatch, LevelMatch, LocationMatch, SalaryMatch) з налаштовуваними коефіцієнтами.
4. Запропоновано та реалізовано гібридну рекомендаційну систему, що поєднує контент-базовану фільтрацію на основі косинусної подібності з item-based колаборативною фільтрацією. Адаптивне балансування між складовими забезпечує плавний перехід від холодного старту до повноцінного гібридного режиму.

5. Виконано функціональне тестування за сімома сценаріями та експериментальну оцінку алгоритмів. Гібридне ранжування забезпечує точність 0,71 проти 0,38 для ізольованого BM25; адаптивна рекомендаційна схема дає точність 0,64 і ефективно усуває проблему холодного старту.

Наукова новизна одержаних результатів полягає у запропонованому гібридному алгоритмі ранжування з можливістю незалежного налаштування ваг п’яти компонентів та у гібридній рекомендаційній схемі з адаптивним балансуванням між контент-базованою та колаборативною складовими.

Практичне значення результатів полягає у створенні робочого прототипу системи, придатного для застосування у компаніях, рекрутингових агенціях та кадрових службах підприємств.

Перспективи подальшого розвитку охоплюють налаштування процесу безперервного розгортання (CD) на основі GitHub Actions з автоматичним випуском нових версій у хмарне середовище, контейнерним розгортанням через Docker та послідовним просуванням артефактів через середовища staging і production. Окремим напрямом є інтеграція додаткових сигналів зворотного зв’язку у рекомендаційну модель та автоматизація моніторингу якості пошуку у виробничих умовах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *Lviv IT Cluster*. IT Research Ukraine 2025: From Adaptation to Transformation [Електронний ресурс] / Lviv IT Cluster. — 2025. — Режим доступу до ресурсу: https://itcluster.lviv.ua/en/ukrainian-tech-industry-in-the-fourth-year-of-the-war-stabilization-new-strategies-and-a-shift-in-market-architecture-results-of-it-research-ukraine-2025-from-adaptation-to-transformatio/.
2. *Редакція DOU*. Як айтівці шукають роботу в 2026 році: 44% пошукачів знизили зарплатні очікування — результати опитування DOU [Електронний ресурс] / Редакція DOU. — 2026. — Режим доступу до ресурсу: https://dou.ua/lenta/articles/job-market-2026-part-2/.
3. *Редакція DOU*. Як айтівці шукають роботу у 2025 році: 42% пошукачів знизили зарплатні очікування — результати опитування DOU [Електронний ресурс] / Редакція DOU. — 2025. — Режим доступу до ресурсу: https://dou.ua/lenta/articles/job-market-2025-part-2/.
4. Djinni — сервіс пошуку роботи в IT [Електронний ресурс]. — Режим доступу до ресурсу: https://djinni.co/.
5. DOU — спільнота розробників [Електронний ресурс]. — Режим доступу до ресурсу: https://dou.ua/.
6. LinkedIn [Електронний ресурс]. — Режим доступу до ресурсу: https://www.linkedin.com/.
7. Robota.ua [Електронний ресурс]. — Режим доступу до ресурсу: https://robota.ua/.
8. *Fielding R.* HTTP Semantics [Електронний ресурс] : RFC / R. Fielding, M. Nottingham, J. Reschke ; RFC Editor. — 06.2022. — № 9110. — DOI: 10.17487/RFC9110. — Режим доступу до ресурсу: https://www.rfc-editor.org/rfc/rfc9110.html.

9. *Fielding R. T.* Architectural Styles and the Design of Network-based Software Architectures [Електронний ресурс] : дис. … док. / Fielding Roy Thomas. — University of California, Irvine, 2000. — Режим доступу до ресурсу: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.
10. TypeScript: JavaScript With Syntax For Types [Електронний ресурс]. — Режим доступу до ресурсу: https://www.typescriptlang.org/.
11. Node.js: JavaScript runtime built on Chrome's V8 JavaScript engine [Електронний ресурс]. — Режим доступу до ресурсу: https://nodejs.org/.
12. NestJS: A progressive Node.js framework [Електронний ресурс]. — Режим доступу до ресурсу: https://nestjs.com/.
13. Prisma: Next-generation Node.js and TypeScript ORM [Електронний ресурс]. — Режим доступу до ресурсу: https://www.prisma.io/.
14. React: The library for web and native user interfaces [Електронний ресурс]. — Режим доступу до ресурсу: https://react.dev/.
15. Vite: Next Generation Frontend Tooling [Електронний ресурс]. — Режим доступу до ресурсу: https://vitejs.dev/.
16. Tailwind CSS: A utility-first CSS framework [Електронний ресурс]. — Режим доступу до ресурсу: https://tailwindcss.com/.
17. Zustand: Bear necessities for state management in React [Електронний ресурс]. — Режим доступу до ресурсу: https://zustand-demo.pmnd.rs/.
18. TanStack Query: Powerful asynchronous state management [Електронний ресурс]. — Режим доступу до ресурсу: https://tanstack.com/query/latest.
19. PostgreSQL: The World's Most Advanced Open Source Relational Database [Електронний ресурс]. — Режим доступу до ресурсу: https://www.postgresql.org/.
20. Redis: The Real-time Data Platform [Електронний ресурс]. — Режим доступу до ресурсу: https://redis.io/.

21. Docker: Accelerated Container Application Development [Електронний ресурс]. — Режим доступу до ресурсу: https://www.docker.com/.
22. nginx: HTTP and reverse proxy server [Електронний ресурс]. — Режим доступу до ресурсу: https://nginx.org/.
23. Firebase Cloud Storage [Електронний ресурс]. — Режим доступу до ресурсу: https://firebase.google.com/docs/storage.
24. *Jones M.* JSON Web Token (JWT) [Електронний ресурс] : RFC / M. Jones, J. Bradley, N. Sakimura ; RFC Editor. — 05.2015. — № 7519. — DOI: 10.17487/RFC7519. — Режим доступу до ресурсу: https://www.rfc-editor.org/rfc/rfc7519.html.
25. *Provos N.* A Future-Adaptable Password Scheme [Електронний ресурс] / N. Provos, D. Mazieres // Proceedings of the 1999 USENIX Annual Technical Conference. — 1999. — С. 81—91. — Режим доступу до ресурсу: https://www.usenix.org/legacy/event/usenix99/provos/provos.pdf.
26. *Hardt D.* The OAuth 2.0 Authorization Framework [Електронний ресурс] : RFC / D. Hardt ; RFC Editor. — 10.2012. — № 6749. — DOI: 10.17487/RFC6749. — Режим доступу до ресурсу: https://www.rfc-editor.org/rfc/rfc6749.html.
27. Passport.js: Simple, unobtrusive authentication for Node.js [Електронний ресурс]. — Режим доступу до ресурсу: https://www.passportjs.org/.
28. *Robertson S.* The Probabilistic Relevance Framework: BM25 and Beyond [Електронний ресурс] / S. Robertson, H. Zaragoza // Foundations and Trends in Information Retrieval. — 2009. — Т. 3, № 4. — С. 333—389. — DOI: 10.1561/1500000019.
29. Elasticsearch: The Distributed Search and Analytics Engine [Електронний ресурс]. — Режим доступу до ресурсу: https://www.elastic.co/elasticsearch/.

30. *Hu Y.* Collaborative Filtering for Implicit Feedback Datasets [Електронний ресурс] / Y. Hu, Y. Koren, C. Volinsky // Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008). — IEEE Computer Society, 2008. — С. 263—272. — DOI: 10.1109/ICDM.2008.22.

31. Recommender Systems Handbook [Електронний ресурс] / за ред. F. Ricci, L. Rokach, B. Shapira. — 2-е вид. — Boston, MA : Springer, 2015. — DOI: 10.1007/978-1-4899-7637-6.

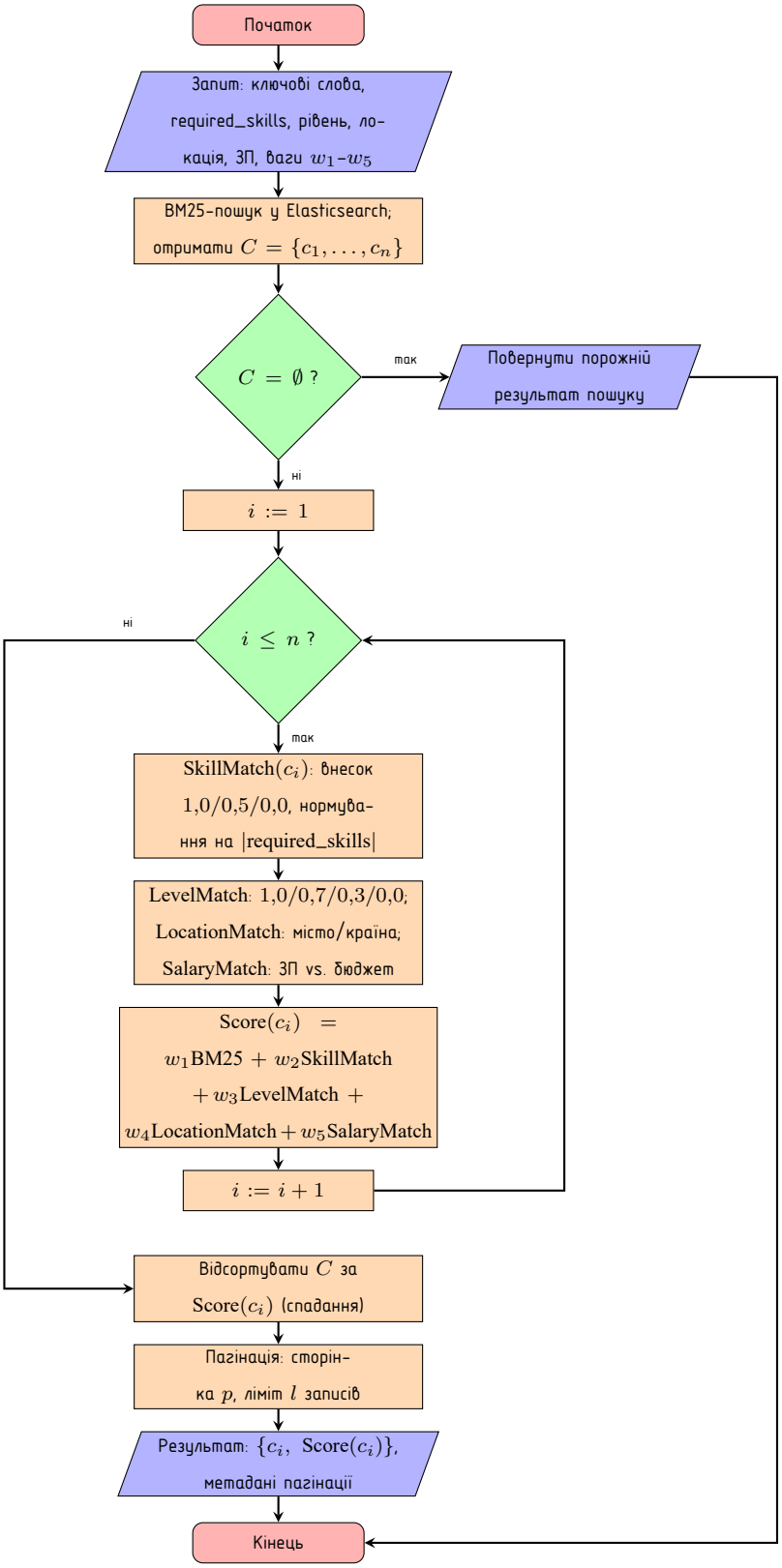
ДОДАТОК А

Веб-система для пошуку та підбору фахівців у сфері інформаційних технологій

Принципова схема

ІАЛЦ.467200.004 Д1

Київ — 2026 р.



					ІАЛЦ.467200.004 Д1	
					Веб-система для пошуку та підбору фахівців у сфері інформаційних технологій Принципова схема	Лит. / Маса / Масштаб
Зм.	Лист	№ докум.	Підп.	Дата		
Розробив		Рабійчук Д. О.				Аркуш 1 / Аркушів 1
Перевірив		Шульга М. В.				
Т. контр.						НТУУ "КПІ ім. Ігоря Сікорського", ФІОТ ІМ-21
Н. контр.		Гончаренко О. О.				
Затвердив		Волокита А. М.				

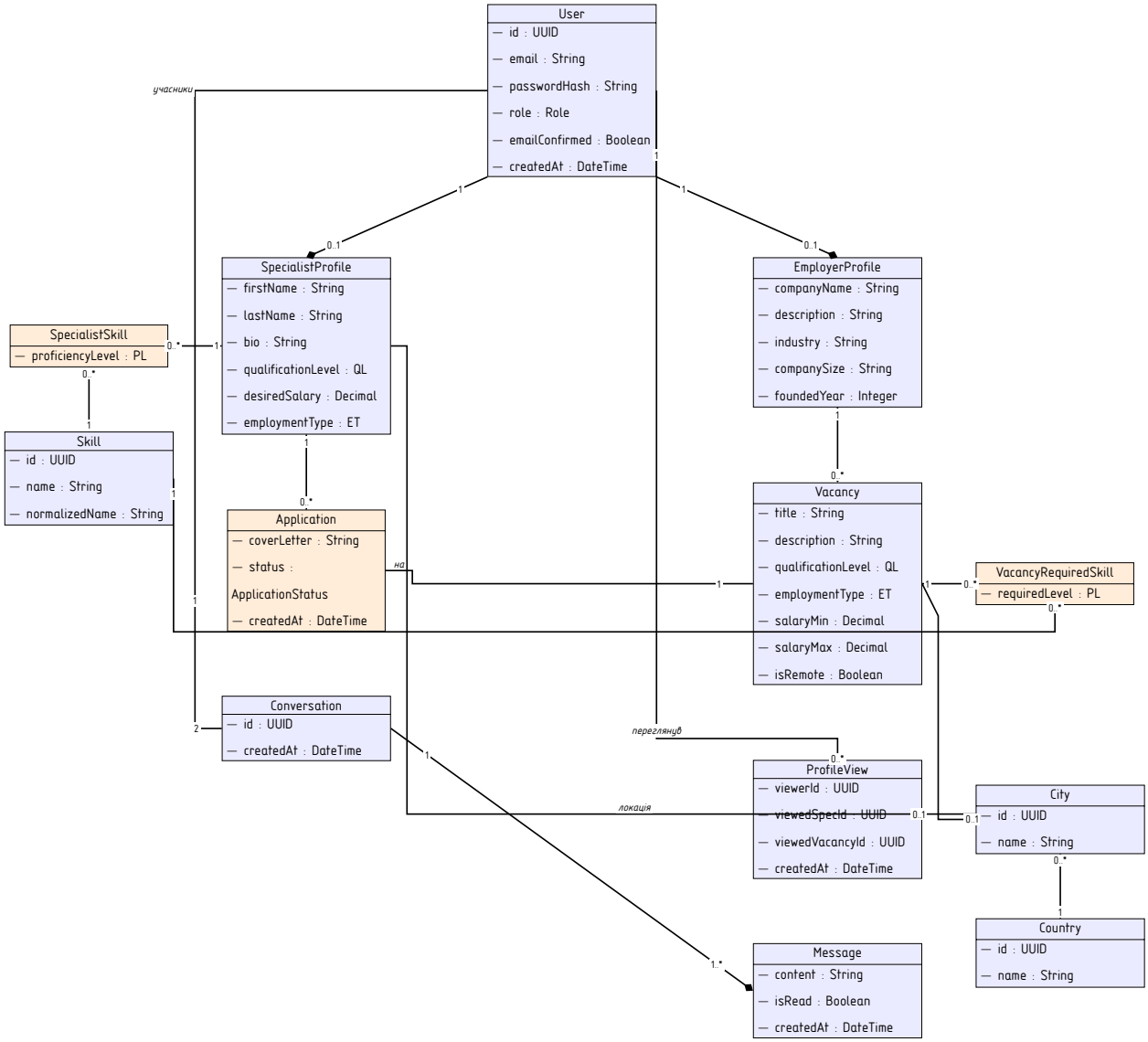
ДОДАТОК Б

Веб-система для пошуку та підбору фахівців у сфері інформаційних технологій

Функціональна схема

ІАЛЦ.467200.005 Д2

Київ — 2026 р.



					ІАЛЦ.467200.005 Д2	
					Веб-система для пошуку та підбору фахівців у сфері інформаційних технологій Функціональна схема	Лит. / Маса / Масштаб
Зм.	Лист	№ докум.	Підп.	Дата		
Розробив		Рабійчук Д. О.				Аркуш 1 / Аркушів 1
Перевірив		Шульга М. В.				
Т. контр.						НТУУ "КПІ ім. Ігоря Сікорського", ФІОТ ІМ-21
Н. контр.		Гончаренко О. О.				
Затвердив		Волокита А. М.				

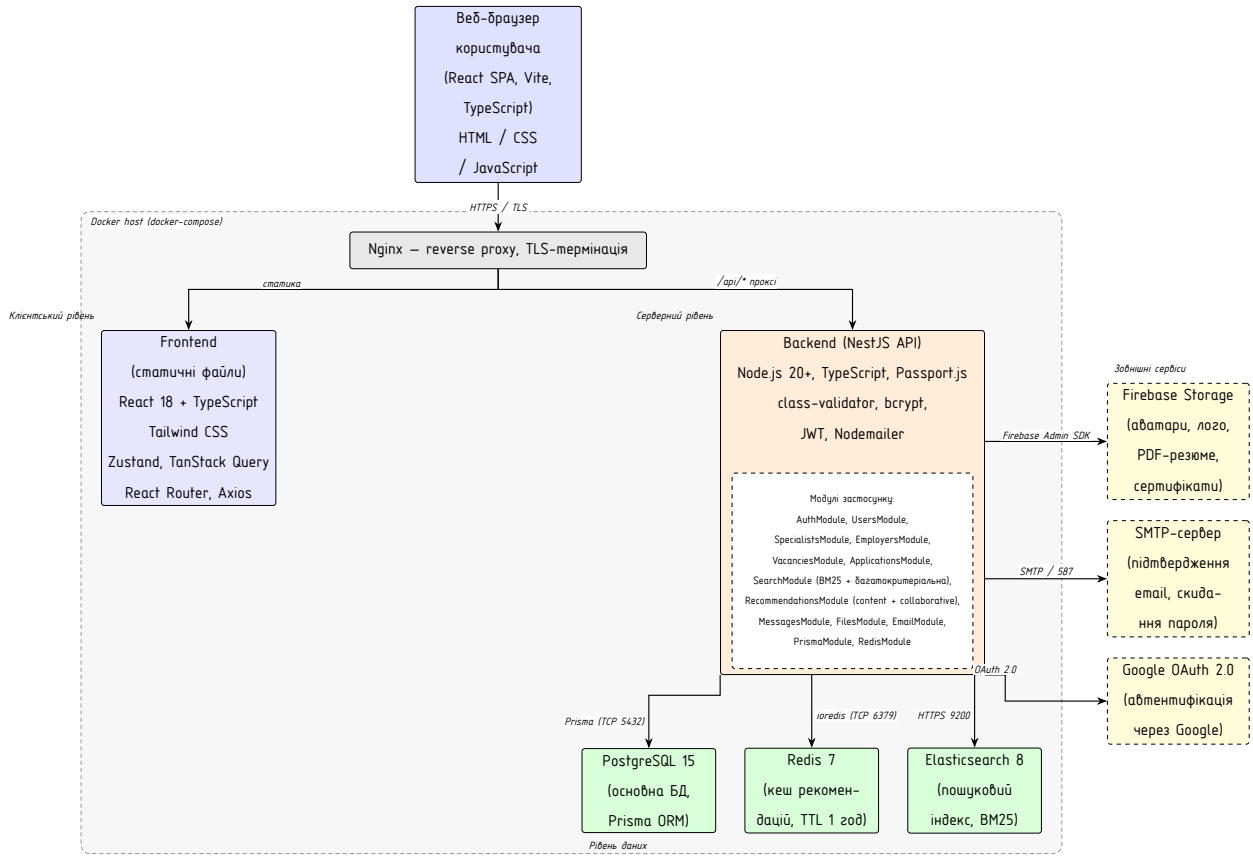
ДОДАТОК В

Веб-система для пошуку та підбору фахівців у сфері інформаційних технологій

Структурна схема

ІАЛЦ.467200.006 Д3

Київ — 2026 р.



					ІАЛЦ.467200.006 ДЗ	
					Веб-система для пошуку та підбору фахівців у сфері інформаційних технологій Структурна схема	Лит. / Маса / Масштаб
Зм.	Лист	№ докум.	Підп.	Дата		
Розробив		Рабійчук Д. О.				Аркуш 1 / Аркушів 1
Перевірив		Шульга М. В.				
Т. контр.						НТУУ "КПІ ім. Ігоря Сікорського", ФІОТ ІМ-21
Н. контр.		Гончаренко О. О.				
Затвердив		Волокита А. М.				

ДОДАТОК Г

Веб-система для пошуку та підбору фахівців у сфері інформаційних технологій

Текст програмного коду

ІАЛЦ.467200.007 Д4

Київ — 2026 р.

apps/backend/src/search/search.service.ts

```ts
async searchSpecialists(dto: SearchSpecialistsDto) {
  const limit = dto.limit ?? 20;
  const page = dto.page ?? 1;
  const candidateSize = 100;

  const esResponse = await this.es.search({
    index: SPECIALIST_INDEX,
    size: candidateSize,
    query: this.buildBaseQuery(dto),
  });

  const hits = esResponse.hits.hits;
  if (hits.length === 0) return { items: [], total: 0, page, limit };

  const maxBm25 = hits.reduce((m, h) => Math.max(m, h._score ?? 0), 0);
  const bm25Map = new Map(
    hits.map((h) => [h._id!, maxBm25 > 0 ? (h._score ?? 0) / maxBm25 : 1]),
  );
  const ids = hits.map((h) => h._id!);

  const [profiles, queryCityCountryId] = await Promise.all([
    this.prisma.specialistProfile.findMany({
      where: { id: { in: ids }, deletedAt: null },
      include: { skills: { include: { skill: true } }, city: { include: { country: true } } },
    }),
    dto.cityId
      ? this.prisma.city
          .findUnique({ where: { id: dto.cityId }, select: { countryId: true } })
          .then((c) => c?.countryId)
      : Promise.resolve(undefined),
  ]);

  const profileMap = new Map(profiles.map((p) => [p.id, p]));

  const scored = ids
    .map((id) => {
      const profile = profileMap.get(id);
      if (!profile) return null;

      const bm25 = bm25Map.get(id) ?? 0;
      const skillMatch = computeSkillMatch(profile.skills, dto.skillIds ?? []);
      const levelMatch = computeLevelMatch(profile.qualificationLevel, dto.qualificationLevel);
```

					ІАЛЦ.467200.007 Д4		
Зм.	Лист	№ докум.	Підп.	Дата			
Розробив		Рабійчук Д. О.			Веб-система для пошуку та підбору фахівців у сфері інформаційних технологій Текст програмного коду	Лит.	Аркуш 1 / Аркушів 6
Перевірив		Шульга М. В.					
Н. контр.		Гончаренко О. О.				НТУУ "КПІ ім. Ігоря Сікорського", ФІОТ ІМ-21	
Затвердив		Волокита А. М.					

```
      const locationMatch = computeLocationMatch(
        {
          cityId: profile.cityId,
          cityCountryId: profile.city?.countryId,
          openToRemote: profile.openToRemote,
        },
        { cityId: dto.cityId, queryCityCountryId, openToRemote: dto.openToRemote },
      );
      const salaryMatch = computeSalaryMatch(profile.desiredSalaryMax, dto.salaryMax);

      const finalScore =
        W_BM25 * bm25 +
        W_SKILL_MATCH * skillMatch +
        W_LEVEL_MATCH * levelMatch +
        W_LOCATION_MATCH * locationMatch +
        W_SALARY_MATCH * salaryMatch;

      return { profile, finalScore };
    })
    .filter((x): x is NonNullable<typeof x> => x !== null)
    .sort((a, b) => b.finalScore - a.finalScore);

  const paged = scored.slice((page - 1) * limit, page * limit);
  return {
    items: paged.map(({ profile, finalScore }) => ({
      ...profile,
      _score: Math.round(finalScore * 100) / 100,
    })),
    total: scored.length,
    page,
    limit,
  };
}
```

apps/backend/src/recommendations/recommendations.service.ts

```
const PROFICIENCY_SCORE: Record<string, number> = {
  BEGINNER: 1, INTERMEDIATE: 2, ADVANCED: 3, EXPERT: 4,
};

function cosine(a: Map<string, number>, b: Map<string, number>): number {
  let dot = 0;
  let magA = 0;
  let magB = 0;
  for (const [id, va] of a) {
    magA += va * va;
    const vb = b.get(id);
    if (vb !== undefined) dot += va * vb;
  }
  for (const [, vb] of b) magB += vb * vb;
  if (magA === 0 || magB === 0) return 0;
  return dot / (Math.sqrt(magA) * Math.sqrt(magB));
}

function computeAlpha(historySize: number): number {
  return historySize >= MIN_HISTORY_THRESHOLD ? ALPHA_WARM :
  ALPHA_COLD_START;
}

async recommendVacancies(userId: string, limit = 20) {
  const cacheKey = `rec:v2:vacancies:${userId}:${limit}`;
  const cached = await this.redis.get(cacheKey);
  if (cached) return cached;

  const specialist = await this.prisma.specialistProfile.findUnique({
    where: { userId },
    include: { skills: true },
  });
  if (!specialist) throw new NotFoundException('Specialist profile not
  found');

  const specialistVector = new Map<string, number>(
    specialist.skills.map((s) => [s.skillId,
  PROFICIENCY_SCORE[s.proficiencyLevel] ?? 1]),
  );
  if (specialistVector.size === 0) return [];

  const vacancies = await this.prisma.vacancy.findMany({
    where: { deletedAt: null, isPublished: true },
    include: { requiredSkills: { include: { skill: true } } },
    orderBy: { createdAt: 'desc' },
    take: 300,
  });

  const vacancyIds = vacancies.map((v) => v.id);
  const [collaborativeScores, viewedCount] = await Promise.all([

  this.collaborative.computeCollaborativeScoresForVacancies(vacancyIds,
  userId),
```

```
    this.prisma.profileView.count({
      where: { viewerId: userId, viewedVacancyId: { not: null } },
    }),
  ]);

  const alpha = computeAlpha(viewedCount);

  const scored = vacancies
    .map((vacancy) => {
      const vacancyVector = new Map<string, number>(
        vacancy.requiredSkills.map((s) => [
          s.skillId,
          (PROFICIENCY_SCORE[s.minProficiencyLevel] ?? 1) * (s.isRequired ? 2 : 1),
        ]),
      );

      const similarityScore = cosine(specialistVector, vacancyVector);

      const required = vacancy.requiredSkills.filter((s) => s.isRequired);
      const coverage =
        required.length === 0
          ? 1
          : required.filter(
              (s) =>
                specialistVector.has(s.skillId) &&
                (specialistVector.get(s.skillId) ?? 0) >=
                  (PROFICIENCY_SCORE[s.minProficiencyLevel] ?? 1),
            ).length / required.length;

      const contentScore = similarityScore * CONTENT_WEIGHT + coverage * COVERAGE_WEIGHT;
      const collaborativeScore = collaborativeScores.get(vacancy.id) ?? 0;
      const finalScore = alpha * contentScore + (1 - alpha) * collaborativeScore;

      return { vacancy, finalScore };
    })
    .filter((r) => r.finalScore > 0)
    .sort((a, b) => b.finalScore - a.finalScore)
    .slice(0, limit);

  const result = scored.map(({ vacancy, finalScore }) => ({
    ...vacancy,
    _score: Math.round(finalScore * 100) / 100,
  }));
  await this.redis.set(cacheKey, result, REC_TTL);
  return result;
}
```

apps/backend/src/recommendations/collaborative.service.ts

```ts
@Injectable()
export class CollaborativeService {
  constructor(private readonly prisma: PrismaService) {}

  computeJaccardSimilarity(a: Set<string>, b: Set<string>): number {
    if (a.size === 0 && b.size === 0) return 0;
    let intersection = 0;
    for (const v of a) {
      if (b.has(v)) intersection++;
    }
    const union = a.size + b.size - intersection;
    return union === 0 ? 0 : intersection / union;
  }

  async computeCollaborativeScoresForVacancies(
    candidateIds: string[],
    specialistUserId: string,
  ): Promise<Map<string, number>> {
    const scores = new Map(candidateIds.map((id) => [id, 0]));

    const viewed = await this.prisma.profileView.findMany({
      where: { viewerId: specialistUserId, viewedVacancyId: { not: null } },
      select: { viewedVacancyId: true },
      distinct: ['viewedVacancyId'],
    });

    const viewedIds = viewed.map((v) => v.viewedVacancyId!);
    if (viewedIds.length === 0) return scores;

    const allViews = await this.prisma.profileView.findMany({
      where: {
        viewedVacancyId: { in: [...new Set([...candidateIds, ...viewedIds])] },
      },
      select: { viewedVacancyId: true, viewerId: true },
    });

    const viewerSets = new Map<string, Set<string>>();
    for (const v of allViews) {
      const vid = v.viewedVacancyId!;
      if (!viewerSets.has(vid)) viewerSets.set(vid, new Set());
      viewerSets.get(vid)!.add(v.viewerId);
    }

    for (const candidateId of candidateIds) {
      const viewersOfCandidate = viewerSets.get(candidateId) ?? new Set<string>();
      let total = 0;
      for (const viewedId of viewedIds) {
        const viewersOfViewed = viewerSets.get(viewedId) ?? new Set<string>();
        total += this.computeJaccardSimilarity(viewersOfCandidate,
```

```
    viewersOfViewed);
50        }
51        scores.set(candidateId, total / viewedIds.length);
52      }
53
54      return scores;
55    }
56  }
```