

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

**«Вебпортал для пошуку роботи та підбору
персоналу»**

Виконав студент групи ІПЗ-22-1
АЛІЄВ Омар Емінович

Керівник роботи:
КОНЦИДАЙЛО Андрій Михайлович

Рецензент:
САВІЦЬКИЙ Роман Святославович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **АЛІЄВ Омар Емінович**

Керівник роботи: **КОНЦИДАЙЛО Андрій Михайлович**

Тема роботи: **«Вебпортал для пошуку роботи та підбору персоналу»**,
затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №92/с**
Вихідні дані для роботи: **методи та засоби розробки вебпорталів, зокрема використання сучасних веб-орієнтованих технологій для розробки системи пошуку роботи з можливістю створення вакансій, розміщення резюме, подання заявок та взаємодії між роботодавцями і кандидатами.**
Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Концидайло А.М.	01.03.2026р.	01.03.2026р
2	Концидайло А.М.	01.03.2026р	01.03.2026р
3	Концидайло А.М.	01.03.2026р	01.03.2026р

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з проєкту і пояснювальної записки. Пояснювальна записка до випускної роботи містить 104 сторінок, 32 ілюстрацій та 11 таблиць.

Основною метою даної роботи є проєктування та реалізація вебпорталу для пошуку роботи та підбору персоналу.

Методи, використані під час створення проєкту включали в себе аналіз потреб аудиторії, підбір засобів для програмування, розробку схеми для бази даних, роботу з графічними програмами для дизайну, розробку клієнтської частини та реалізацію серверної логіки.

Результати розробки полягають у створенні зручного середовища з доступом до інформації як для шукачів роботи, так і для роботодавців.

Вебпортал забезпечує можливість реєстрації користувачів, створення та редагування резюме, розміщення вакансій, подання заявок на вакансії, а також взаємодію між роботодавцями та кандидатами.

Під час розробки програмного забезпечення були використані сучасні вебтехнології та засоби розробки, зокрема React, Next.js, NestJS, Prisma, Redis, Python та PostgreSQL.

КЛЮЧОВІ СЛОВА : РОБОТА, ВАКАНСІЇ, РЕЗЮМЕ, ПЕРСОНАЛ, ВЕБПОРТАЛ, БАЗА ДАНИХ, КЛІЄНТ, СЕРВЕР

					ІПЗ.КР.Б-121-26-ПЗ					
Змн.	Арк.	№ докум.	Підпис	Дата	Вебпортал для пошуку роботи та підбору персоналу			Літ.	Арк.	Аркушів
Розроб.		Алієв О.Е.								
Керівник		Концидайло А.М.							3	104
								Житомирська політехніка група ІПЗ-22-1		
Реценз.		Савіцький Р.С.								
Зав.каф.		Вакалюк Т.А.								

ABSTRACT

The bachelor's qualification work consists of a project and an explanatory note. The explanatory note contains 104 pages, 32 figures, and 11 tables.

The main objective of this work is the design and implementation of a web portal for job search and personnel recruitment.

The methods used during the project development included analyzing user requirements, selecting software development tools, designing the database schema, using graphic design software, developing the client-side application, and implementing server-side logic.

The result of the development is the creation of a convenient environment that provides access to information for both job seekers and employers.

The web portal provides user registration, creation and editing of resumes, job vacancy posting, submission of job applications, as well as interaction between employers and candidates.

Modern web technologies and development tools were used during the software development process, including React, Next.js, NestJS, Prisma, Redis, Python, and PostgreSQL.

KEYWORDS: JOB, VACANCIES, RESUME, PERSONNEL, WEB PORTAL, DATABASE, CLIENT, SERVER.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ФОРМУВАННЯ ВИМОГ ДО ВЕБПОРТАЛУ	10
1.1. Постановка задачі	10
1.2. Аналіз аналогів програмного продукту.....	13
1.3. Вибір архітектури проєкту	18
1.4. Обґрунтування використаних технологій системи	22
Висновки до першого розділу	25
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБПОРТАЛУ ДЛЯ ПОШУКУ РОБОТИ ТА ПІДБОРУ ПЕРСОНАЛУ	26
2.1. Концепція архітектури та варіантів використання системи	26
2.2. Розробка бази даних системи.....	34
2.3. Проєктування та реалізація алгоритмів роботи системи.....	39
2.4. Реалізація функціоналу вебпорталу для пошуку роботи та підбору персоналу.....	44
Висновки до другого розділу.....	56
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПОРТАЛОМ	58
3.1. Локальне встановлення та налаштування системи.....	58
3.2. Розгортання та налаштування системи в мережі Інтернет.....	62
3.3. Структура інтерфейсу	67
3.4. Тестування роботи вебпорталу	76
Висновки до третього розділу	83
ВИСНОВКИ	84
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	86
ДОДАТКИ	89

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API - Application Programming Interface (інтерфейс програмування застосунків)

BFF - Backend for Frontend (проміжний шар між клієнтом і сервером)

CORS - Cross-Origin Resource Sharing (механізм обміну ресурсами між різними доменами)

CRUD - основні операції керування даними: Create (створення), Read (читання), Update (оновлення), Delete (видалення)

DTO - Data Transfer Object (об'єкт передачі даних)

HTTP - HyperText Transfer Protocol (протокол передачі гіпертексту)

MVC - Model-View-Controller (архітектурний шаблон проектування)

ORM - Object-Relational Mapping (об'єктно-реляційне відображення)

Prisma ORM - інструмент для роботи з базою даних через типізовані запити

REST - Representational State Transfer (архітектурний стиль API)

SPA - Single Page Application (односторінковий застосунок)

SSR - Server-Side Rendering (рендеринг на стороні сервера)

SQL - Structured Query Language (мова запитів до бази даних)

UI - User Interface (користувацький інтерфейс)

UX - User Experience (досвід користувача)

БД - База даних

ПЗ - Програмне забезпечення

СУБД - Система управління базами даних

HR - Human Resources (управління людськими ресурсами)

ATS - Applicant Tracking System (система відстеження кандидатів, програмне забезпечення для автоматизації збору, зберігання та обробки резюме і заявок на вакансії)

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. Сучасні вебпортали для пошуку роботи та підбору персоналу є одним із ключових інструментів цифрового ринку праці. Вони забезпечують швидку та ефективну взаємодію між роботодавцями та кандидатами, значно спрощують процес найму та підвищують загальну ефективність працевлаштування.

Найбільшу частку серед каналів для пошуку роботи займають саме спеціалізовані сайти. Вони дозволяють користувачам швидко знаходити актуальні вакансії, порівнювати пропозиції різних роботодавців та подавати заявки безпосередньо через портали. Крім того, такі ресурси забезпечують централізований доступ до великої кількості пропозицій роботи, що значно підвищує ефективність пошуку. Таким чином, вебпортали залишаються основним інструментом цифрового рекрутингу [1].

Основними користувачами таких систем є шукачі роботи, роботодавці та HR-спеціалісти. Шукачі роботи отримують можливість швидко знайти вакансії відповідно до своїх навичок та досвіду. Роботодавці можуть оперативно закривати вакансії та формувати базу перспективних кандидатів. HR-фахівці використовують платформи для аналізу ринку праці, оцінки рівня заробітних плат відповідно до досвіду та кваліфікації кандидатів, а також визначення актуальних тенденцій у галузях. Це забезпечує можливість аналізу конкурентності ринку праці та визначення орієнтовного рівня оплати праці спеціалістів відповідно до їхнього досвіду.

Вебпортал для пошуку роботи та підбору персоналу - це комплексний сучасний цифровий інструмент, який поєднує в собі функції бази даних вакансій, системи управління кандидатами (ATS) та платформи для комунікації між учасниками ринку праці. Він автоматизує процеси пошуку, відбору та аналізу кандидатів, суттєво зменшуючи часові та організаційні витрати як для роботодавців, так і для шукачів роботи.

Таким чином, розробка проєкту є актуальною та доцільною.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		7

Вона повністю відповідає сучасним вимогам цифровізації ринку праці, підвищує ефективність працевлаштування та спрощує взаємодію між усіма учасниками процесу.

Мета і завдання дослідження. Метою даної роботи є розробка алгоритмів роботи та реалізація вебпорталу для пошуку роботи та підбору персоналу.

Сформульована мета визначає наступні завдання:

1. Проаналізувати актуальні вимоги до сучасних вебпорталів для пошуку роботи та підбору персоналу та сформулювати функціональні і нефункціональні вимоги до системи.
2. Дослідити предметну область ринку праці, включаючи взаємодію між шукачами роботи, роботодавцями та HR-спеціалістами.
3. Провести порівняльний аналіз існуючих платформ для пошуку роботи та обґрунтувати власну розробку.
4. Спроектувати клієнт-серверну архітектуру проекту на основі REST API та SPA-підходу.
5. Реалізувати серверну частину системи, включаючи бізнес-логіку роботи з користувачами, вакансіями, резюме та заявками.
6. Розробити клієнтську частину вебпорталу з реалізацією інтерфейсів для шукачів роботи та роботодавців, включаючи систему автентифікації та авторизації.
7. Спроектувати та реалізувати структуру бази даних для зберігання інформації про користувачів, вакансії, резюме та відгуки.
8. Провести тестування вебпорталу (функціональне та інтеграційне) з перевіркою основних сценаріїв використання системи.

Об'єктом дослідження є методи та засоби розробки веб-орієнтованих інформаційних систем у сфері управління процесами пошуку роботи та підбору персоналу.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		8

Предметом дослідження є використання сучасних вебтехнологій для розробки вебпорталу, що забезпечує взаємодію між шукачами роботи та роботодавцями.

У роботі було використано методи об'єктно-орієнтованого проєктування, побудови компонентної архітектури вебпорталів, вебпрограмування, проєктування реляційних баз даних та розробки користувацьких інтерфейсів.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ФОРМУВАННЯ ВИМОГ ДО ВЕБПОРТАЛУ

1.1. Постановка задачі

Вебпортал для пошуку роботи та підбору персоналу є сучасною інформаційною системою, призначеною для автоматизації процесів працевлаштування та рекрутингу. Основна ідея - створення єдиного цифрового середовища, де шукачі роботи та роботодавці можуть ефективно взаємодіяти, суттєво скорочуючи час найму та підвищуючи якість підбору персоналу.

Типова функціональність подібних вебпорталів включає реєстрацію та автентифікацію користувачів, створення та редагування резюме, публікацію вакансій, подання відгуків на вакансії, а також систему пошуку та фільтрації пропозицій роботи. Додатково можуть реалізовуватися механізми повідомлень, збереження вакансій, оцінювання кандидатів та аналітичні інструменти для HR-спеціалістів, що дозволяють більш точно підбирати персонал.

Проект буде реалізований на основі клієнт-серверної архітектури з чітким поділом на логічні модулі. Серверна частина повинна забезпечувати бізнес-логіку, обробку запитів та взаємодію з базою даних. Клієнтська частина має забезпечувати зручний та швидкий інтерфейс користувача. Необхідно розробити структуру бази даних, яка включатиме ключові сутності: користувачі, ролі (кандидат, роботодавець), резюме, вакансії, заявки, освіта, досвід роботи, мови та рівні володіння ними.

Клієнтська частина проекту реалізована з використанням сучасного фреймворку Next.js, що забезпечує можливість серверного рендерингу та покращення SEO-оптимізації сторінок, які містять вакансії. Також це дозволяє поєднувати підходи SPA та SSR, забезпечуючи швидку навігацію та високу продуктивність інтерфейсу. Додатково застосовується адаптивний дизайн, що

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

гарантує коректне відображення системи на різних пристроях і роздільних здатностях екранів.

Важливим елементом системи є реалізація автоматизованого ранжування вакансій та кандидатів за допомогою алгоритмічних підходів, які частково можуть базуватися на методах штучного інтелекту. Ця функціональність є особливо актуальною, оскільки, за даними досліджень, рекрутери в середньому витрачають лише кілька секунд на первинний перегляд одного резюме. За такий короткий час повноцінно оцінити кандидата практично неможливо. Водночас детальний аналіз одного резюме може займати від кількох хвилин. При великій кількості відгуків на вакансію (що є типовим для популярних позицій) рекрутер витрачає на первинний відбір вже декілька годин. Така ситуація суттєво знижує ефективність роботи HR-фахівців і створює ризик пропуску сильних кандидатів. У зв'язку з цим автоматизація процесів відбору та ранжування є необхідною для підвищення якості та швидкості рекрутингу [2].

Окремим функціональним блоком системи є модерація контенту, яка забезпечує контроль якості та достовірності інформації, що публікується на платформі. До функцій модерації належить перевірка вакансій та резюме на відповідність правилам сервісу, видалення або приховування некоректного чи забороненого контенту, а також контроль за діями користувачів у системі. Такий підхід дозволяє підтримувати актуальність даних, запобігати публікації шахрайських оголошень та підвищувати загальний рівень довіри до вебпорталу.

На етапі реалізації передбачено тестування основних функціональних модулів системи, включаючи перевірку роботи API, коректність обробки запитів, взаємодії з базою даних та відображення даних у користувацькому інтерфейсі. Це дозволяє виявити та усунути можливі помилки до впровадження системи (табл.1.1-1.2).

Також необхідним є забезпечення високим рівнем безпеки. Для цього потрібно реалізувати хешування паролів, автентифікацію через електронну

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		11

пошту та токенизацію. Доступ до персональних даних (зокрема резюме) має бути строго обмежений і надаватися лише з дозволу власника або при поданні заявки на вакансію.

Таблиця 1.1

Основні вимоги до системи

Категорія	Вимога
Доступність та надійність	Вебпортал повинен бути доступним протягом усього часу, за винятком планового технічного обслуговування та непередбачених ситуацій.
Доступність та надійність	Доступність системи повинна становити не менше 99 % часу протягом календарного місяця.
Доступність та надійність	Максимальна норма дефектів - 1 помилка на 10 000 рядків коду.
Продуктивність	Система повинна підтримувати одночасну роботу не менше 500 користувачів.
Продуктивність	Час відгуку для типових операцій - не більше 2 секунд.
Продуктивність	Час виконання складних запитів - не більше 15 секунд.
Придатність до експлуатації	Система повинна підтримувати масштабування у разі збільшення навантаження.
Придатність до експлуатації	Оновлення версій повинно виконуватися автоматизовано через систему контролю версій.
Обмеження проєктування	Система повинна відповідати сучасним стандартам проєктування вебдодатків та UI/UX.
Обмеження проєктування	Мінімальні системні вимоги: процесор 2 ГГц, 4096 МБ RAM, 20 ГБ дискового простору, ОС Windows/Linux, доступ до Інтернету.
Обмеження проєктування	Для зберігання даних використовується реляційна база даних PostgreSQL.
Бізнес-логіка	Доступ до функціональності системи здійснюється через вебінтерфейс із використанням клієнт-серверної архітектури.

Антивимоги до системи

Категорія	Антивимога
Контентні обмеження	Система не використовує інформацію, заборонену нормами моралі та чинним законодавством.
Контентні обмеження	Система не містить матеріалів, які можуть зашкодити психічному стану користувачів.
Рекламні обмеження	Система не містить сторонньої реклами.

Така структура дозволить розробити сучасний, зручний і ефективний інструмент для ринку праці.

1.2. Аналіз аналогів програмного продукту

Аналіз існуючих аналогічних систем є важливим етапом розробки вебпорталу для пошуку роботи та підбору персоналу. Він дозволяє вивчити наявні рішення на ринку, виявити їхні сильні та слабкі сторони, а також визначити перспективні напрями вдосконалення власної системи (табл.1.3)

Work.ua - це один із найбільших українських вебпорталів для пошуку роботи та підбору персоналу, який забезпечує взаємодію між роботодавцями та шукачами роботи через розширену систему вакансій і резюме[3].

Платформа пропонує потужну систему пошуку вакансій з великою кількістю фільтрів (заробітна плата, місто, досвід роботи, тип зайнятості, графік, ключові слова) (Рис.1.1), створення детального резюме з шаблонами, швидкий відгук на вакансії, особистий кабінет, систему рекомендацій вакансій на основі профілю, а також налаштування сповіщень і підписок.

Work.ua добре реалізував баланс між функціональністю та зручністю. Особливо варто відзначити систему рекомендацій, яка допомагає кандидатам не пропустити релевантні вакансії (Рис.1.2), та розвинену систему фільтрів, яка значно економить час. Для роботодавців зручним є наявність інструментів для управління відгуками.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		13

Значна частина корисних інструментів для роботодавців (підняття вакансій, розширені статистики, пріоритет у пошуку) доступна лише на платній основі. Також через велику кількість користувачів кандидати стикаються з високою конкуренцією, а інтерфейс іноді виглядає перевантаженим через великий обсяг інформації.

Рис. 1.1 - Система фільтру вакансій

Рис. 1.2 - Система рекомендацій вакансій

OLX.ua у розділі «Робота» - універсальна класифайд-платформою, яка має окремий розділ для пошуку роботи. На відміну від спеціалізованих вебпорталів, тут акцент зроблено на простоті та швидкості [4].

Користувачі можуть переглядати вакансії у форматі оголошень, використовувати базові фільтри (Рис.1.3), зв'язуватися з роботодавцем напряму через чат або телефонні номери, а також розміщувати власні вакансії. Реєстрація та створення резюме не є обов'язковими.

Головна сила цього сайту це мінімальний бар'єр входу. Шукач роботи може відгукнутися на вакансію за кілька секунд без створення профілю. Такий підхід особливо привабливий для швидкого пошуку тимчасової або неформальної роботи.

Через відсутність обов'язкового резюме та слабкої модерації якості вакансій часто низька (Рис.1.4). Платформа майже не має інструментів для серйозного рекрутингу: відсутня система зберігання резюме, аналітика кандидатів та автоматизований підбір. Це робить OLX менш ефективним для пошуку кваліфікованих спеціалістів.

Рис.1.3 – Базовий фільтр вакансій

Опубліковано 21 травня 2026 р.

Потрібна няня для хлопчика

ЗАРПЛАТА

32 000 - 45 000 грн. / за місяць

МІСЦЕЗНАХОДЖЕННЯ

Житомир, Міськмолзавод

ТИП ЗАЙНЯТОСТІ

Повна зайнятість

ГРАФІК РОБОТИ

Повний робочий день

Досвід роботи: Ні

Підходить кандидатам: Пенсійного віку, Студентам, В декреті, Молодше 18 років

Бронювання: Ні

3 проживанням

МІСЦЕЗНАХОДЖЕННЯ

Житомир, Міськмолзавод

ОПИС

Вітаю

Шукаємо няню з проживанням у нас в Києві

Пешить на вайбер або телеграм

Рис.1.4 - Приклад некоректного розміщення вакансії на платформі (невідповідність зазначеного міста в заголовку та описі)

Jobs.ua це один з найстаріших українських порталів з пошуку роботи, який донині зберігає значну базу вакансій, але поступається сучасним конкурентам за функціональністю [5].

Платформа пропонує пошук вакансій за категоріями та регіонами, перегляд детальної інформації, базове розміщення резюме, каталог компаній, а також можливість зв'язку з роботодавцем через вказані контакти.

Серед позитивних сторін - велика накопичена база вакансій і відносна простота інтерфейсу. Для деяких користувачів зручно, що можна швидко знайти контакти роботодавця і зв'язатися з ним напряму.

Функціональність платформи виглядає застарілою. Відсутність зручної системи відгуків безпосередньо на сайті змушує користувачів переходити на зовнішні канали зв'язку (Рис.1.5). Також практично немає автоматизованого

підбору вакансій, сучасних рекомендацій та якісної інтеграції між резюме і вакансіями. UX/UI потребує суттєвого оновлення.

Продавець

★ VIP

🔥 Гаряча

📅 Термінова

30 000 грн.

🕒 11 червня

📍 Київ

Компанія:

ФЛП Пахолюк Константин (Всі вакансії)

Рубрики:

Торгівля, продажі, закупівлі; Ресторанний бізнес, купілярія

КОНТАКТИ

☎ Телефон:

+38 (097) 753-61-19

👤 Контактна особа:

Костянтин

📍 Адреса:

Київ, вул. Вербицького,30.

Даний роботодавець бажає, щоб ви зв'язалися з ним за телефонами вказаними в контактах.

ПОБАЖАННЯ ДО СПІВРОБІТНИКА

Освіта:

не має значення

Досвід роботи:

не вимагається

Графік роботи:

повний робочий день

ОПИС ВАКАНСІЇ

Потрібні продавці у м'ясний магазин на Лівий берег.

Вік бажано до 50 років.

З/п: 27000-30000 грн.

📄 ЗАВАНТАЖИТИ

🖨 РОЗДРУКУВАТИ

⚠ СКАРГА

Рис.1.5 - Сторінка вакансії на Jobs.ua без можливості подати відгук безпосередньо на сайті

Таблиця 1.3

Порівняння аналогів

Назва	Work.ua	OLX.ua (розділ “Робота”)	Jobs.ua	Мій
Розширені фільтри вакансій	+	частково	-	+
Створення резюме	+	-	-	+
Відгук на вакансію	+	-	-	+
Різні типи профілів	+	+	+	+
ATS/аналітика	+	-	+	-
Прямий контакт з роботодавцем	-	+	+	+
Модерації вакансій	+	-	+	+
Ранжування резюме та вакансій за допомогою ІІІ	-	-	-	+

1.3. Вибір архітектури проєкту

Вибір архітектури програмної системи є одним із найважливіших етапів проєктування вебпорталу, оскільки саме архітектура визначає спосіб взаємодії компонентів системи, можливості її масштабування, супроводу та подальшого розвитку. Правильно обрана архітектура дозволяє забезпечити високу продуктивність системи, спростити процес розробки та підвищити надійність роботи програмного забезпечення. Крім того, архітектурні рішення безпосередньо впливають на складність підтримки системи, швидкість внесення змін та можливість інтеграції нових функціональних можливостей у майбутньому.

Для побудови сучасних вебсистем використовуються різні архітектурні підходи. Найбільш поширеними серед них є монолітна архітектура, клієнт-серверна архітектура, мікросервісна архітектура та багаторівнева (шарова) архітектура.

Монолітна архітектура передбачає реалізацію всієї функціональності системи в межах одного програмного застосунку. У такій архітектурі всі модулі системи працюють в єдиному середовищі виконання та використовують спільну кодову базу. Основними перевагами монолітної архітектури є простота розробки на початкових етапах проєкту, спрощене розгортання та відсутність необхідності організації взаємодії між окремими сервісами. Крім того, монолітні системи зазвичай простіше тестувати та адмініструвати на початкових етапах життєвого циклу проєкту. Водночас зі збільшенням кількості функціональних можливостей і користувачів система стає складнішою для підтримки та модернізації. Будь-які зміни в окремому модулі потребують повторного розгортання всього застосунку, а масштабування окремих компонентів стає практично неможливим без масштабування всієї системи. Саме тому монолітна архітектура частіше використовується для невеликих або відносно простих програмних продуктів.

Клієнт-серверна архітектура базується на розподілі системи на дві основні частини: клієнтську та серверну. Клієнтська частина відповідає за

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		18

взаємодію з користувачем та відображення інформації, а серверна здійснює обробку даних, виконання бізнес-логіки та взаємодію зі сховищем даних. Основною перевагою такого підходу є чіткий розподіл відповідальності між компонентами системи, що спрощує підтримку програмного забезпечення та дозволяє незалежно розвивати інтерфейс користувача й серверну частину. Також клієнт-серверна архітектура забезпечує централізоване зберігання даних та контроль доступу до них. До недоліків цього підходу можна віднести залежність роботи клієнтської частини від доступності сервера, а також необхідність забезпечення надійного обміну даними між компонентами системи. Незважаючи на це, клієнт-серверна архітектура залишається одним із найпоширеніших підходів для побудови сучасних вебпорталів завдяки оптимальному балансу між складністю реалізації та можливостями масштабування.

Мікросервісна архітектура передбачає поділ системи на набір незалежних сервісів, кожен з яких виконує окрему функцію та взаємодіє з іншими через програмні інтерфейси. Такий підхід забезпечує високу гнучкість системи та дозволяє незалежно розробляти, тестувати й масштабувати окремі сервіси. Крім того, різні компоненти системи можуть бути реалізовані з використанням різних технологій залежно від поставлених задач. До переваг мікросервісної архітектури також належить підвищена відмовостійкість, оскільки збій одного сервісу не обов'язково призводить до повної недоступності всієї системи. Водночас така архітектура суттєво ускладнює процес розробки та супроводу програмного забезпечення. Зростає кількість мережових взаємодій між компонентами, виникає потреба в додаткових механізмах моніторингу, журналювання подій, балансування навантаження та забезпечення безпеки обміну даними. Для невеликих та середніх проєктів використання повноцінної мікросервісної архітектури часто є надлишковим.

Багаторівнева (шарова) архітектура передбачає поділ програмного забезпечення на окремі логічні рівні, наприклад рівень представлення, рівень бізнес-логіки та рівень доступу до даних. Такий підхід дозволяє спростити

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						19
Змн.	Арк.	№ докум.	Підпис	Дата		

підтримку системи та підвищити її модульність. Основною перевагою багаторівневої архітектури є чітке розмежування відповідальності між компонентами системи, що полегшує внесення змін до окремих модулів без впливу на інші частини програмного забезпечення. Крім того, поділ на рівні підвищує повторне використання програмного коду та спрощує тестування функціональних компонентів. До недоліків можна віднести збільшення кількості програмних шарів та необхідність організації взаємодії між ними, що в окремих випадках може призводити до незначного зниження продуктивності системи. Незважаючи на це, багаторівнева архітектура широко використовується в сучасних інформаційних системах завдяки високій підтримуваності та зрозумілій структурі програмного коду.

Слід зазначити, що на практиці сучасні вебсистеми часто поєднують декілька архітектурних підходів одночасно. Наприклад, система може використовувати клієнт-серверну модель взаємодії між користувачем та сервером, багаторівневу організацію програмного коду для розподілу відповідальності між компонентами, а також окремі незалежні сервіси для реалізації спеціалізованого функціоналу. Такий підхід дозволяє поєднати переваги різних архітектурних рішень та забезпечити баланс між простотою реалізації, продуктивністю і можливістю подальшого розвитку системи.

Для вибору архітектурного рішення було проведено аналіз особливостей предметної області та функціональних вимог до системи. Розроблювана платформа повинна забезпечувати одночасну роботу великої кількості користувачів, централізоване зберігання інформації про вакансії та резюме, підтримку процесів реєстрації, автентифікації та взаємодії між шукачами роботи й роботодавцями. Крім того, система повинна мати можливість подальшого розширення функціональності та інтеграції інтелектуальних алгоритмів аналізу даних.

Монолітна архітектура не повною мірою відповідає зазначеним вимогам через складність масштабування окремих компонентів системи. Використання повноцінної мікросервісної архітектури також не є оптимальним рішенням,

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

оскільки значно підвищує складність реалізації та супроводу програмного забезпечення. З урахуванням особливостей предметної області найбільш доцільним є використання клієнт-серверної архітектури з модульною структурою компонентів системи.

Для розроблюваного вебпорталу було обрано клієнт-серверну архітектуру з модульною організацією програмного забезпечення. Система складається з клієнтської частини, серверної частини, підсистеми зберігання даних та окремого модуля інтелектуальної обробки інформації. Кожен із компонентів виконує власний набір функцій та взаємодіє з іншими компонентами через визначені програмні інтерфейси.

Клієнтська частина системи забезпечує взаємодію користувачів із платформою та відповідає за відображення інформації, введення даних і формування запитів до серверної частини. Через вебінтерфейс користувачі отримують доступ до всіх функціональних можливостей системи.

Серверна частина реалізує основну бізнес-логіку платформи, виконує обробку запитів користувачів, здійснює перевірку прав доступу та координує роботу інших компонентів системи. Саме серверна частина забезпечує централізоване управління даними та контроль виконання бізнес-процесів.

Окремий модуль інтелектуальної обробки інформації призначений для виконання задач аналізу та ранжування даних. Виділення такого функціоналу в окремий компонент дозволяє незалежно розвивати алгоритми обробки інформації та забезпечує можливість подальшого вдосконалення механізмів підбору кандидатів і вакансій без внесення змін до основної логіки системи.

Таким чином, для реалізації вебпорталу пошуку роботи було обрано клієнт-серверну архітектуру з модульною структурою та окремим модулем інтелектуальної обробки інформації. Цей підхід забезпечує достатню гнучкість, спрощує підтримку програмного забезпечення, дозволяє ефективно розподілити функціональність між компонентами системи та створює можливість подальшого масштабування окремих частин платформи відповідно до майбутніх потреб користувачів.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						21
Змн.	Арк.	№ докум.	Підпис	Дата		

1.4. Обґрунтування використаних технологій системи

При розробці вебпорталу для пошуку роботи та підбору персоналу було обрано сучасний стек технологій, що забезпечує високу продуктивність, масштабованість та зручність підтримки системи. Вибір конкретних технологічних рішень здійснювався з урахуванням вимог до продуктивності, безпеки, гнучкості розширення функціоналу та зручності супроводу програмного забезпечення. Серверна частина реалізується з використанням NestJS, клієнтська частина - на базі Next.js, а для збереження та обробки даних використовується PostgreSQL.

NestJS було обрано як основний фреймворк для реалізації серверної логіки, оскільки він забезпечує чітку архітектурну структуру проєкту та підтримує принципи модульності [6]. Його основні компоненти - Controller, Service та Module - дозволяють розділити бізнес-логіку, обробку запитів та організацію коду на незалежні блоки. Такий підхід значно спрощує масштабування системи, тестування та подальший розвиток проєкту. Крім того, NestJS базується на Node.js і TypeScript, що забезпечує типізацію, стабільність та високу продуктивність серверної частини. Додатковою перевагою є підтримка Dependency Injection, що дозволяє зменшити зв'язаність компонентів системи та підвищити її модульність.

Next.js було обрано для реалізації клієнтської частини завдяки його гібридному підходу до рендерингу [7]. У системі використовується поєднання SPA (Single Page Application) для динамічних частин інтерфейсу, таких як профіль користувача, особистий кабінет та взаємодія з резюме, і SSR (Server-Side Rendering) для сторінок вакансій. SSR дозволяє покращити SEO-оптимізацію, що є важливим для сторінок вакансій, оскільки вони повинні індексуватися пошуковими системами та залучати органічний трафік. Водночас SPA-підхід забезпечує швидку взаємодію користувача з інтерфейсом без перезавантаження сторінок [8]. Додатково Next.js забезпечує оптимізацію продуктивності за рахунок автоматичного поділу коду (code

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		22

splitting), що зменшує час завантаження сторінок та підвищує загальний користувацький досвід.

Для збереження та обробки даних використовується PostgreSQL - реляційна система управління базами даних, яка забезпечує високу продуктивність при роботі зі складними запитам. Вибір PostgreSQL зумовлений необхідністю реалізації розширеної системи фільтрації вакансій, включаючи пошук за містом, досвідом, заробітною платою, типом зайнятості та іншими параметрами. Завдяки підтримці індексів, складних SQL-запитів та повнотекстового пошуку PostgreSQL дозволяє ефективно обробляти великі обсяги даних і забезпечує швидку вибірку релевантних вакансій [9]. Додатковою перевагою є підтримка транзакційності та механізмів цілісності даних, що є критично важливим для системи, яка оперує великою кількістю взаємопов'язаних сутностей (користувачі, вакансії, резюме, заявки).

Для зберігання користувацьких сесій у системі використовується Redis. Використання Redis дозволяє швидко отримувати дані сесій, зменшити навантаження на основну базу даних та забезпечити стабільну роботу механізмів автентифікації й авторизації користувачів. Завдяки зберіганню даних у оперативній пам'яті Redis забезпечує мінімальний час доступу до інформації, що особливо важливо для систем з великою кількістю одночасних користувачів. Крім того, Redis підтримує механізми TTL (time-to-live), що дозволяє автоматично керувати життєвим циклом сесій.

Окремим важливим компонентом системи є модуль штучного інтелекту, реалізований з використанням Python. Даний модуль відповідає за інтелектуальне ранжування резюме та вакансій. У межах цього модуля застосовуються методи векторного пошуку та семантичного аналізу тексту. Резюме та вакансії перетворюються на векторні представлення (embeddings), після чого здійснюється обчислення косинусної подібності між векторами кандидата та вакансії. Це дозволяє визначати ступінь релевантності між ними на основі змісту, а не лише ключових слів. Такий підхід забезпечує більш

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		23

точний та інтелектуальний відбір кандидатів порівняно з класичними алгоритмами TF-IDF або простим пошуком за ключовими словами.

Модуль штучного інтелекту інтегрується з основною системою через REST API та використовується для формування рейтингу кандидатів і вакансій у реальному часі. Така інтеграція дозволяє розділити відповідальність між основною серверною частиною та ML-сервісом, забезпечуючи незалежний розвиток алгоритмів машинного навчання та їх масштабування без впливу на інші компоненти системи.

Отже, обраний технологічний стек формує цілісну та логічно структуровану екосистему, в якій кожен компонент виконує чітко визначену роль і забезпечує ефективну взаємодію з іншими елементами. Використані технології демонструють високу ступінь сумісності, що дозволило реалізувати як клієнтську, так і серверну частини системи без надмірної складності інтеграції.

Комбінація NestJS, Next.js, PostgreSQL, Redis та окремого Python-сервісу машинного навчання забезпечує оптимальне співвідношення продуктивності, масштабованості та зручності підтримки. NestJS та Next.js, побудовані на TypeScript, дозволяють використовувати єдину мову програмування на всьому стеку, що суттєво спрощує розробку та зменшує кількість помилок. PostgreSQL виступає надійним реляційним сховищем даних, а Redis ефективно вирішує задачі кешування та роботи з сесіями в реальному часі. Окремий Python-сервіс дає можливість повноцінно використовувати потужні бібліотеки машинного навчання без впливу на основну кодову базу.

Обрані технології є широко поширеними в сучасній веб-розробці, мають активну спільноту та розвинену екосистему інструментів, що сприяє стабільності системи та спрощує її подальшу підтримку й розвиток. Крім того, такий стек забезпечує високу швидкість розробки нових функціональних можливостей та зручність інтеграції додаткових сервісів у майбутньому.

Таким чином, обраний технологічний стек є раціональним, надійним та перспективним рішенням для реалізації сучасного вебпорталу пошуку роботи,

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						24
Змн.	Арк.	№ докум.	Підпис	Дата		

що поєднує необхідний рівень продуктивності, гнучкості та зручності супроводу. Він повністю відповідає вимогам проєкту щодо обробки великих обсягів даних, рекомендаційних алгоритмів та забезпечення комфортного користувацького досвіду.

Висновки до першого розділу

У першому розділі проведено комплексне дослідження предметної області та сучасного стану ринку вебпорталів для пошуку роботи. Було проаналізовано ключові потреби основних учасників ринку праці — шукачів роботи, роботодавців та HR-спеціалістів.

У другому підрозділі проведено аналіз популярних аналогів - Work.ua, OLX.ua та Jobs.ua. Було виявлено, що існуючі платформи добре реалізують базовий функціонал, проте мають суттєві обмеження в автоматизації підбору, інтелектуальному ранжуванні кандидатів та гнучкості пошуку.

У третьому підрозділі обґрунтовано вибір архітектури проєкту. Система побудована за клієнт-серверною архітектурою з багаторівневою організацією програмного забезпечення та виділеним сервісом машинного навчання. Такий підхід дозволяє поєднати простоту реалізації основної системи з можливістю винесення окремих функціональних модулів у незалежні сервіси, зокрема модуля штучного інтелекту.

У четвертому підрозділі обґрунтовано вибір інструментальних засобів та визначено вимоги до апаратного забезпечення. Для реалізації клієнтської частини використовується Next.js, серверної частини - NestJS, для збереження даних - PostgreSQL, для зберігання сесій - Redis, а для реалізації модуля штучного інтелекту - Python. Вибір даного технологічного стеку забезпечує продуктивність, масштабованість та зручність подальшого розвитку системи. Таким чином, перший розділ дозволив комплексно вивчити проблематику, проаналізувати конкурентне середовище та обрати сучасні технології, які забезпечать створення конкурентоспроможного вебпорталу з акцентом на автоматизацію рекрутингу.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						25
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБПОРТАЛУ ДЛЯ ПОШУКУ РОБОТИ ТА ПІДБОРУ ПЕРСОНАЛУ

2.1. Концепція архітектури та варіантів використання системи

Аналіз вимог до вебпорталу для пошуку роботи та підбору персоналу включає визначення і детальний опис функціональних та нефункціональних вимог, оцінку швидкості реакції системи на дії користувачів, опис основних сценаріїв її використання, а також створення узагальненої схеми елементів системи з їх структурою, взаємозв'язками та принципами роботи.

Цей етап дозволяє глибше зрозуміти потреби користувачів і технічні особливості, необхідні для розробки зручного та ефективного вебпорталу.

Функціональні вимоги описують конкретні дії та можливості, які повинна реалізовувати система (табл.2.1). Вони визначають, що саме має робити вебпортал для пошуку роботи та підбору персоналу.

Таблиця 2.1

Основні функціональні вимоги до системи

№	Функціональна вимога	Опис
1	Реєстрація та авторизація користувачів	Авторизація за логіном і паролем або через OAuth-провайдерів
		Реєстрація користувачів із використанням електронної пошти та пароля
		Реєстрація користувачів через OAuth-провайдерів.
		Додавання інформації про освіту, досвід роботи, навички та знання мов
		Відновлення пароля
		Підтвердження електронної пошти при реєстрації
2	Можливості шукача роботи	Створення та редагування особистого профілю
		Створення та редагування резюме

№	Функціональна вимога	Опис
		Перегляд історії поданих відгуків на вакансії
		Доступ до списку збережених вакансій
3	Можливості роботодавця	Створення та редагування профілю компанії
		Створення нових вакансій
		Перегляд списку створених вакансій
		Можливість перегляду відгуків на свою вакансію
		Сортувати та фільтрувати отримані резюме за допомогою модуля штучного інтелекту
		Надсилати кандидатам запрошення на співбесіду або відмови
		Закривати або архівувати вакансії
4	Можливості модерації	Перевірка нових вакансій на правильність заповнення
		Можливість давати статус верифікації роботодавцям
		Можливість блокування роботодавців
		Перегляд статистики нових користувачів і активності подання відгуків
		Перегляд статистики по зарплатам
5	Інтелектуальне ранжування резюме	Аналіз відповідності резюме вимогам вакансії
		Автоматичне обчислення відсотка релевантності
		Сортування кандидатів за рівнем відповідності вакансії
6	Система сповіщень	Надсилання повідомлень про зміну статусу заявки
		Надсилання повідомлень про зміну статусу вакансії

Нефункціональні вимоги стосуються характеристик системи, таких як продуктивність, безпека, зручність використання та надійність. Вони визначають, якою має бути система з точки зору швидкості роботи, захисту даних користувачів, доступності та масштабованості.

До ключових нефункціональних вимог належать висока швидкість завантаження сторінок, надійний захист персональних даних, інтуїтивно зрозумілий інтерфейс та можливість одночасної роботи великої кількості користувачів (табл.2.2).

Нефункціональні вимоги до системи

№	Нефункціональна вимога	Опис
1	Продуктивність	Час відкриття основних сторінок не повинен перевищувати 2 секунд.
		Час виконання пошуку вакансій не повинен перевищувати 5 секунд.
		Система повинна підтримувати одночасну роботу не менше 500 користувачів.
		Час виконання
		Час обробки складних запитів, пов'язаних з аналізом та ранжуванням резюме за допомогою ML-моделі, не повинен перевищувати 15 секунд.
2	Безпека	Паролі користувачів повинні зберігатися у хешованому вигляді
		Для авторизації повинні використовуватися session-based auth
		Захист від несанкціонованого доступу до персональних даних
		Обмеження кількості спроб входу для захисту від атак перебором
3	Надійність	Збереження даних у реляційній базі даних
		Коректне відновлення роботи після перезапуску сервера
		Захист від втрати даних при помилках користувача
4	Зручність використання	Адаптивний інтерфейс для мобільних пристроїв та персональних комп'ютерів
		Інтуїтивно зрозуміла навігація між сторінками
		Мінімальна кількість дій для подання заявки на вакансію
5	Масштабованість	Можливість розгортання окремих сервісів (Frontend, Backend, ML Service) незалежно один від одного.
		Можливість розширення функціоналу без зміни існуючої архітектури.

Часові характеристики системи

№	Операція	Час виконання
1	Реєстрація нового користувача (з урахуванням повідомлення на пошту)	Не більше 6 секунд
2	Авторизація користувача	Не більше 3 секунд
3	Операції пошуку та фільтрації даних	Не більше 5 секунд
4	Операції додавання нових записів у базу даних	Не більше 3 секунд
5	Операції редагування існуючих записів	Не більше 3 секунд
6	Операції видалення записів з бази даних	Не більше 3 секунд
7	Операції аналітичної обробки та ранжування даних	Не більше 15 секунд

Часові характеристики системи визначають швидкість обробки запитів та час відгуку (табл.2.3).

Специфікація варіантів використання системи описує основні сценарії взаємодії користувачів з програмною системою. Вона є одним із ключових документів на етапі аналізу вимог і призначена для чіткого розуміння того, як користувачі різних ролей (актори) можуть використовувати систему для досягнення своїх цілей. У специфікації детально описуються основні бізнес-процеси, які підтримує система, з акцентом на функціональність, доступну для кожного типу користувача.

У цій специфікації для кожного варіанту використання наводиться опис основного потоку подій (основного сценарію), альтернативних сценаріїв, а також можливих виключних ситуацій. Крім того, визначаються умови передвиконання, які повинні бути виконані перед початком сценарію, та умови післявиконання, які гарантують результат успішного завершення. Такий підхід дозволяє максимально чітко формалізувати очікувану поведінку системи.

З метою кращого розуміння принципів роботи розробленої системи, а також для створення надійної основи для подальших етапів проєктування, реалізації та тестування програмного забезпечення була побудована відповідна діаграма (рис. 2.1).

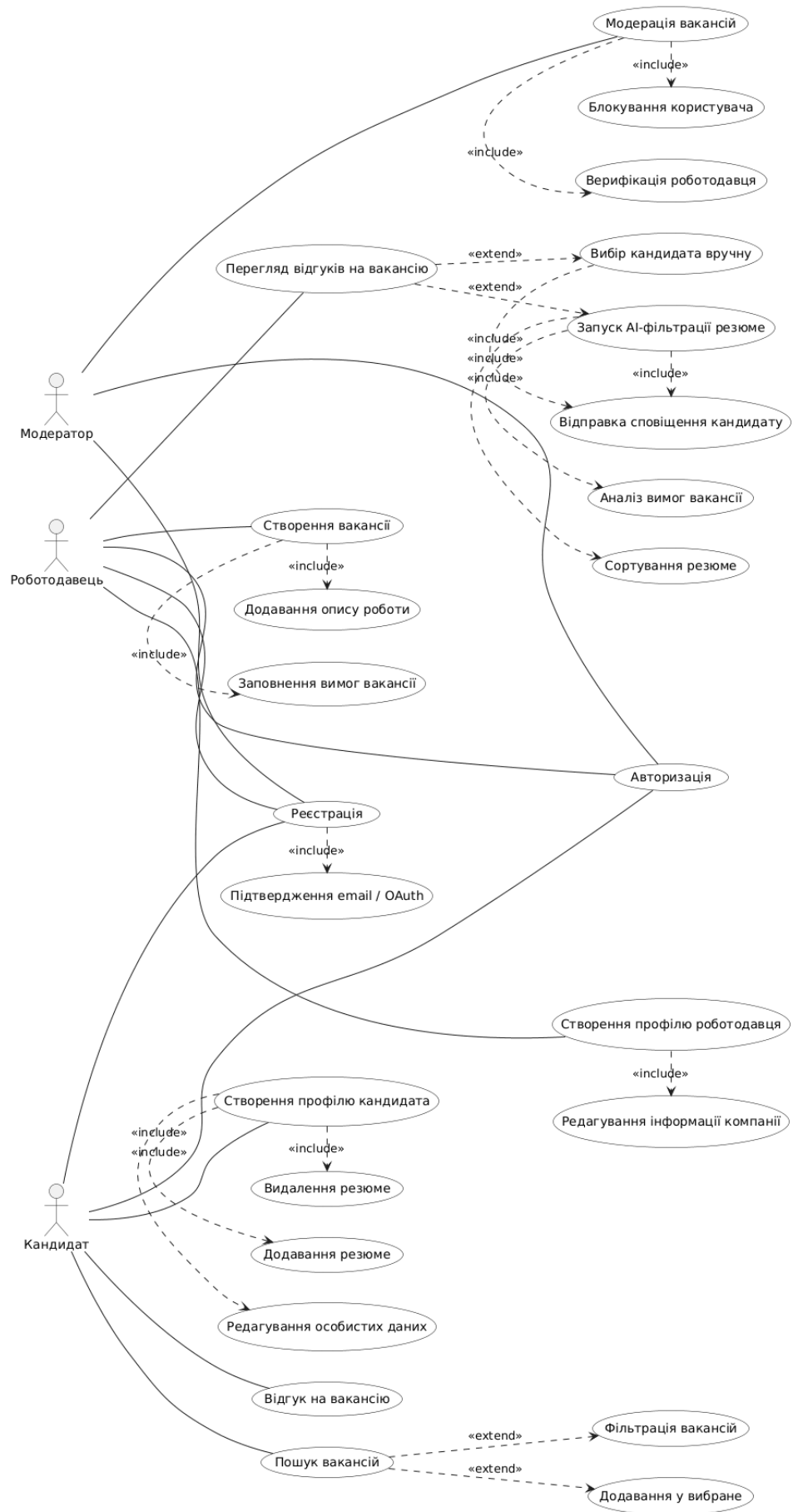


Рис.2.1 - Діаграма варіантів використання

Змн.	Арк.	№ докум.	Підпис	Дата

ІПЗ.КР.Б-121-26-ПЗ

Арк.

30

Нижче наведено основні сценарії використання, які ілюструють типові випадки взаємодії користувачів з вебпорталом для пошуку роботи та підбору персоналу (табл. 2.4).

Таблиця 2.4

Основні сценарії використання

№	Актор	Варіант використання	Опис
1	Гість	Реєстрація та авторизація користувача	Користувач створює обліковий запис із вибором ролі шукача роботи або роботодавця та проходить процедуру підтвердження електронної пошти. Після успішної авторизації користувач отримує доступ до функціональних можливостей системи відповідно до своєї ролі.
2	Кандидат	Створення та управління резюме	Шукач роботи заповнює інформацію про освіту, досвід роботи, професійні навички та знання мов. Створене резюме може використовуватися для подання заявок на вакансії.
3	Кандидат	Пошук вакансій та подання заявки	Користувач здійснює пошук вакансій за допомогою фільтрів та переглядає детальну інформацію про них. Після вибору відповідної вакансії він може подати заявку, використовуючи одне зі своїх резюме.
4	Роботодавець	Створення та управління вакансіями	Роботодавець створює вакансії, вказуючи необхідні вимоги до кандидатів та умови роботи. Виходячи зі статусу роботодавця, вакансія потрапляє у список перевірки модератором, або відразу стає активною для відгуків
5	Роботодавець	Відбір кандидатів за допомогою інтелектуального модуля	Роботодавець отримує результати автоматичного аналізу резюме кандидатів. Інтелектуальний модуль оцінює їх відповідність вимогам вакансії та формує рейтинг кандидатів.

№	Актор	Варіант використання	Опис
6	Модератор	Модерація контенту та управління користувачами	Модератор здійснює контроль опублікованих вакансій та перевіряє їх відповідність правилам платформи. Також він може керувати статусом роботодавців, обмежувати доступ порушникам та переглядати статистичну інформацію щодо роботи системи.

Таким чином, діаграма варіантів використання є важливою складовою етапу проєктування системи, оскільки дозволяє наочно відобразити взаємодію користувачів із програмним продуктом. Вона сприяє кращому розумінню потреб користувачів та визначенню основних функціональних можливостей системи. Аналіз сценаріїв взаємодії між користувачами та системою дає змогу виявити ключові вимоги до функціональності та сформулювати чітке уявлення про очікувану поведінку програмного забезпечення. Це забезпечує відповідність реалізованої системи реальним потребам користувачів та зменшує ризик виникнення невідповідностей між вимогами і реалізацією.

Для відображення об'єктно-орієнтованої структури розробленого вебпорталу використовується діаграма класів. Діаграма класів є однією з ключових статичних структурних UML-діаграм, яка надає цілісне уявлення про внутрішню архітектуру системи [10]. Вона відображає основні сутності (класи), їхні атрибути, методи, а також типи взаємозв'язків між ними.

На основі аналізу функціональних вимог і структури бази даних були визначені ключові сутності платформи. У реалізації вони представлені у вигляді класів бізнес-логіки (рис. 2.2).

Такий підхід дозволяє чітко зрозуміти структуру системи та взаємодію між її основними компонентами.

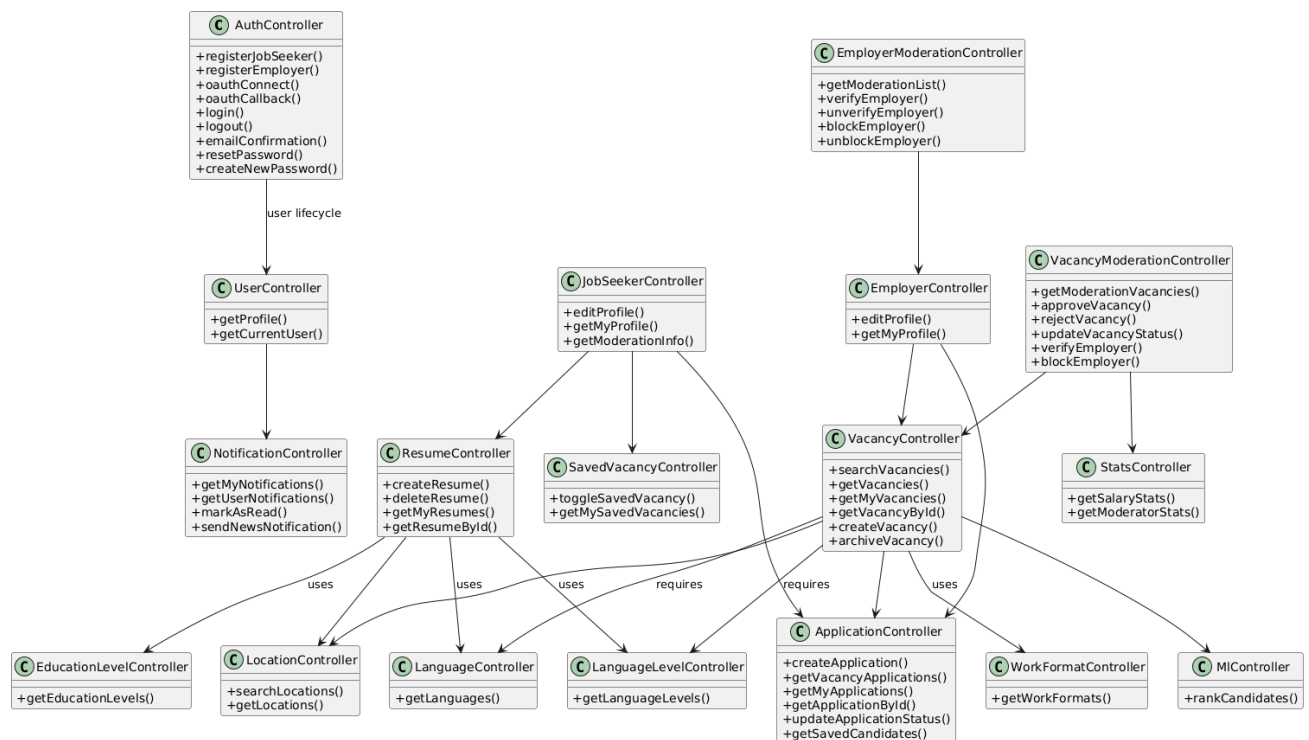


Рис. 2.2. – Діаграма класів

З діаграми можна виділити наступні компоненти системи. AuthController та UserController відповідають за автентифікацію та управління користувачами. Вони забезпечують реєстрацію користувачів, у тому числі через OAuth-провайдерів, авторизацію, підтвердження електронної пошти та відновлення пароля. Окрім цього, UserController надає доступ до поточного профілю користувача.

JobSeekerController та EmployerController реалізують функціонал різних ролей користувачів. JobSeekerController відповідає за управління профілем шукача роботи, тоді як EmployerController забезпечує керування профілем роботодавця, включаючи редагування даних компанії.

ResumeController відповідає за створення, перегляд та видалення резюме користувача, а також забезпечує зв'язок резюме з додатковими даними, такими як освіта, мови та навички.

VacancyController та VacancyModerationController керують функціоналом вакансій. VacancyController реалізує створення, пошук, перегляд та архівацію вакансій, тоді як VacancyModerationController

відповідає за модерацію вакансій і роботодавців, включаючи підтвердження, відхилення та блокування.

ApplicationController відповідає за роботу із заявками кандидатів на вакансії. Він дозволяє створювати заявки, переглядати їх статуси та отримувати список кандидатів для роботодавця, а також змінювати статус заявки.

SavedVacancyController забезпечує функціонал збереження вакансій для подальшого перегляду користувачем. NotificationController відповідає за систему сповіщень, включаючи отримання повідомлень користувачем, їх позначення як прочитаних та надсилання системних повідомлень.

LocationController, LanguageController, LanguageLevelController, WorkFormatController та EducationLevelController є довідниковими компонентами системи. Вони забезпечують роботу зі статичними або напівстатичними даними, які використовуються при створенні резюме та вакансій, а також для фільтрації та пошуку.

MIController реалізує інтелектуальний модуль ранжування кандидатів і виконує аналіз відповідності резюме вимогам вакансій, формуючи рейтинг на основі релевантності. StatsController забезпечує отримання аналітичної інформації системи, зокрема статистики щодо заробітних плат та даних для адміністративного аналізу.

2.2. Розробка бази даних системи

Зберігання всіх даних вебпорталу здійснюється за допомогою СУБД PostgreSQL. Дана система керування базами даних забезпечує надійне зберігання інформації, підтримує реляційну модель даних, механізми індексації, забезпечення цілісності даних та ефективне виконання складних запитів. Вибір саме реляційної моделі зумовлений необхідністю роботи з великою кількістю взаємопов'язаних сутностей та забезпеченням строгих зв'язків між ними.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		34

Для взаємодії з базою даних у серверній частині проєкту використовується ORM-фреймворк Prisma ORM, який забезпечує об'єктно-орієнтовану роботу з даними та автоматичне формування SQL-запитів [11]. Використання ORM дозволяє зменшити кількість ручного SQL-коду, мінімізувати ймовірність помилок при роботі з базою даних та підвищити читабельність і підтримуваність коду.

Під час проєктування бази даних було прийнято рішення використовувати підхід нормалізації даних до третьої нормальної форми. Це дозволило уникнути дублювання інформації, забезпечити цілісність даних та спростити їх подальшу обробку. Окрему увагу було приділено розділенню логічних сутностей на незалежні таблиці, що відповідає принципам реляційного моделювання.

Дотримання третьої нормальної форми забезпечує відсутність транзитивних залежностей між атрибутами, що означає зберігання кожного факту лише в одному місці бази даних. Завдяки цьому спрощується процес оновлення інформації, оскільки зміни в довідникових таблицях автоматично відображаються у пов'язаних сутностях без необхідності масового редагування записів. Крім того, такий підхід зменшує ризик виникнення аномалій вставки, оновлення та видалення даних.

Використання нормалізованої структури також позитивно впливає на продуктивність складних запитів, оскільки дозволяє ефективно застосовувати зв'язки між таблицями через зовнішні ключі та індекси. Це особливо важливо для системи, що передбачає інтенсивну роботу з фільтрацією та пошуком вакансій і резюме.

Таким чином, обрана модель даних забезпечує баланс між структурованістю, продуктивністю та простотою супроводу, що є критично важливим для довгострокового розвитку вебпорталу.

Всього база даних системи містить 18 основних сутностей: User, Account, Token, JobSeeker, Employer, Resume, ResumeEducation,

EducationLevel, Vacancy, WorkFormat, Location, Language, LanguageLevel, ResumeLanguage, VacancyLanguage, Application, SavedVacancy та Notification.

Нижче наведено загальний опис структури бази даних вебпорталу пошуку роботи та підбору персоналу. Основними сутностями системи є користувачі, шукачі роботи, роботодавці, резюме, вакансії та заявки на вакансії, а також допоміжні довідникові таблиці, які забезпечують коректну роботу системи та реалізацію пошуку і фільтрації даних.

Базовою таблицею є User, яка містить загальну інформацію про всіх користувачів системи. Вона включає ідентифікатор користувача (id як первинний ключ), електронну пошту, хеш пароля, роль користувача, статус підтвердження email та спосіб авторизації. Рішення про виділення окремої таблиці користувача та подальший поділ на профілі обумовлено необхідністю реалізації різних типів користувачів (шукач роботи та роботодавець), які мають різні набори атрибутів. Такий підхід дозволяє уникнути надмірної кількості NULL-полів у єдиній таблиці та забезпечує більш гнучку модель даних.

Таблиці JobSeeker та Employer реалізують профілі користувачів залежно від їх ролі. Обидві таблиці містять зовнішній ключ user_id, який посиляється на таблицю User, що реалізує зв'язок "один до одного". JobSeeker зберігає персональні дані шукача роботи, такі як ім'я, прізвище, дату народження, телефон та фото профілю, тоді як Employer містить інформацію про компанію, включаючи назву, опис, вебсайт, логотип та статус верифікації або блокування.

Поділ користувачів на окрему сутність User та профільні сутності JobSeeker і Employer є обґрунтованим рішенням, яке дозволяє розділити загальні дані користувача та дані, що залежать від його ролі в системі. Такий підхід запобігає змішуванню різних типів інформації в одній таблиці та робить структуру бази даних більш впорядкованою.

Винесення профільних даних в окремі таблиці дозволяє уникнути ситуації, коли частина полів є неактуальною для певного типу користувача. Це

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						36
Змн.	Арк.	№ докум.	Підпис	Дата		

також зменшує складність структури таблиць і спрощує подальшу роботу з даними, оскільки кожна сутність містить лише необхідний набір атрибутів.

Окреме зберігання профілів також полегшує розширення системи, оскільки нові атрибути для одного типу користувача можна додавати без впливу на інші типи. Це знижує ризик порушення існуючої структури даних при подальшій розробці.

Додаткові дані, такі як освіта та знання мов, винесені в окремі таблиці ResumeEducation та ResumeLanguage. Таке рішення було прийнято з метою забезпечення нормалізації та уникнення зберігання повторюваних структурованих даних у межах одного запису. Кожна з цих таблиць пов'язана з відповідними довідниковими сутностями EducationLevel, Language та LanguageLevel, що дозволяє забезпечити стандартизовані значення та уникнути неоднозначності у введенні даних [12].

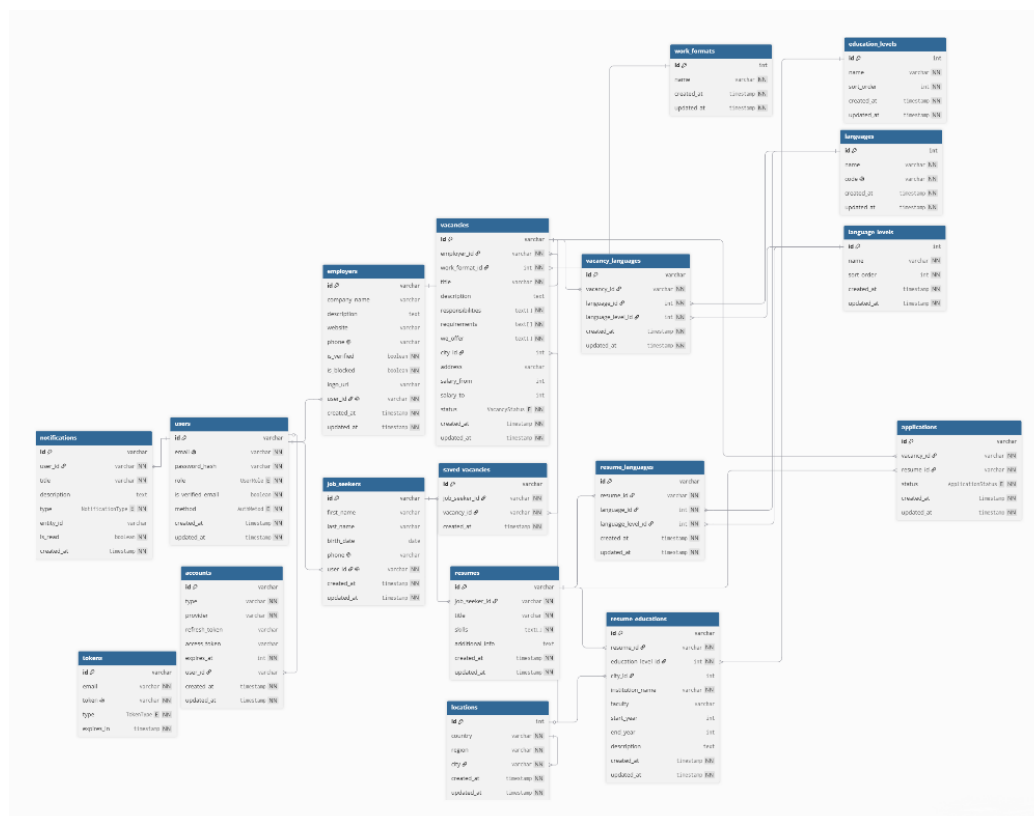
Таблиця Vacancy пов'язана з Employer через зовнішній ключ employer_id, що реалізує зв'язок "один до багатьох", оскільки один роботодавець може створювати декілька вакансій. Вакансія також містить зовнішні ключі work_format_id та location_id, які посилаються на довідникові таблиці WorkFormat та Location. Використання окремих довідникових таблиць є обґрунтованим рішенням, оскільки дозволяє централізовано керувати значеннями, що часто використовуються у фільтрації та пошуку.

Для опису мовних вимог використовується таблиця VacancyLanguage, яка реалізує зв'язок «багато-до-багатьох» між вакансіями та мовами. Це дозволяє гнучко описувати вимоги до кандидатів без обмеження кількості мов для однієї вакансії.

Особливу увагу під час проєктування бази даних було приділено стандартизації іменування полів. Усі поля таблиць використовують стиль snake_case, що забезпечує єдину стилістичну модель даних та підвищує читабельність як у SQL-запитах, так і в ORM-шарі. Це рішення також зменшує ймовірність помилок при інтеграції з серверною частиною проєкту.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						37
Змн.	Арк.	№ докум.	Підпис	Дата		

Загальна структура бази даних системи представлена у вигляді ER-діаграми (рис. 2.3), яка відображає основні сутності, їх атрибути та зв'язки між ними. Вона дозволяє наочно проаналізувати логіку побудови бази даних та взаємодію між її компонентами. На основі цієї діаграми реалізується фізична структура бази даних у СУБД та формуються відповідні таблиці.



Окрім основної бази даних, у системі використовується Redis, який використовується для зберігання тимчасових даних, необхідних під час

автентифікації користувачів. На відміну від основної бази даних, Redis працює в оперативній пам'яті, що забезпечує значно швидший доступ до інформації [13]. Це дозволяє прискорити виконання операцій автентифікації, зменшити навантаження на основну базу даних та підвищити загальну продуктивність вебпорталу.

Redis також використовується для реалізації короткоживучих даних сесій, що дозволяє автоматично керувати їх життєвим циклом без додаткового навантаження на основну СУБД.

2.3. Проєктування та реалізація алгоритмів роботи системи

Вебпортал пошуку роботи та підбору персоналу має широкий набір функціональних можливостей, які суттєво відрізняються залежно від ролі користувача. Для опису логіки роботи системи та моделювання взаємодії користувачів із її компонентами в роботі використовуються діаграми діяльності UML.

Діаграми активності дозволяють представити послідовність дій, бізнес-процеси та сценарії використання платформи для різних ролей користувачів.

Розроблена система підтримує чотири основні категорії користувачів: неавторизованих відвідувачів, шукачів роботи, роботодавців та модераторів.

Для кожної з ролей передбачено окремий набір функціональних можливостей. Алгоритм роботи вебпорталу для неавторизованого користувача починається з переходу на головну сторінку системи, де користувач може переглядати список вакансій та інформацію про роботодавців. Також йому доступні пошук і фільтрація вакансій за різними параметрами. Для того щоб отримати можливість створювати резюме, подавати заявки на вакансії або зберігати вакансії, користувач повинен пройти процедуру реєстрації та авторизації. Під час реєстрації він може створити обліковий запис і обрати роль шукача роботи або роботодавця.

Процес реєстрації починається із заповнення користувачем реєстраційної форми. Після надсилання даних система виконує перевірку їх

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		39

коректності та перевіряє відсутність облікового запису з вказаною електронною поштою. У разі успішного проходження перевірок створюється новий обліковий запис користувача та формується відповідний профіль залежно від обраної ролі. Після завершення реєстрації користувач отримує можливість авторизуватися та працювати з функціоналом системи відповідно до своїх прав доступу.

Після авторизації для шукача роботи відкривається доступ до особистого кабінету. Користувач може створювати та видаляти власні резюме. Під час створення резюме він заповнює інформацію про освіту, професійні навички, досвід роботи та рівень володіння іноземними мовами. Також шукач роботи має можливість переглядати вакансії, використовувати пошук і фільтрацію, а на сторінці вакансії подавати заявку на працевлаштування. Окрім цього, користувач може переглядати історію поданих заявок, відстежувати їх поточний статус, зберігати цікаві вакансії для подальшого перегляду та отримувати сповіщення про зміни статусу заявок і інші важливі події.

Алгоритм створення резюме передбачає заповнення користувачем набору форм, що містять основну інформацію про кандидата, його освіту, професійні навички, попередній досвід роботи та знання іноземних мов. Після завершення введення даних система виконує їх перевірку та зберігає інформацію в базі даних. Один користувач може створювати декілька резюме, що дозволяє адаптувати їх під різні вакансії або професійні напрямки.

Пошук вакансій реалізується за допомогою механізму фільтрації, який дозволяє виконувати відбір пропозицій за різними параметрами. Користувач може здійснювати пошук за назвою вакансії, місцем розташування, форматом роботи, рівнем заробітної плати та іншими характеристиками. Після вибору вакансії відкривається сторінка детального перегляду, на якій відображається повна інформація про пропозицію роботи та роботодавця.

Для подання заявки кандидат обирає одне зі створених резюме та підтверджує відгук на вакансію. Після цього система створює запис про заявку

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						40
Змн.	Арк.	№ докум.	Підпис	Дата		

та пов'язує його з вакансією, резюме та користувачем. Створена заявка отримує початковий статус, який свідчить про її успішне надсилання роботодавцю. Надалі користувач має можливість відстежувати всі зміни статусів через особистий кабінет.

Для роботодавця після авторизації відкривається доступ до особистого кабінету компанії. Він може редагувати інформацію про компанію та керувати її профілем. Також роботодавець має можливість створювати, публікувати та архівувати вакансії. Під час створення вакансії він вказує вимоги до кандидата, рівень заробітної плати, формат роботи, місце розташування та необхідні мовні навички. Роботодавець може переглядати список отриманих заявок і змінювати статус кожної з них відповідно до результатів розгляду. Додатково він має можливість сортувати подані резюме за допомогою інтелектуального модуля ранжування кандидатів.

Алгоритм створення вакансії починається із заповнення роботодавцем спеціальної форми. У процесі створення вакансії користувач вказує необхідну інформацію про посаду, вимоги до кандидатів, умови праці та інші параметри. Після надсилання форми система виконує перевірку введених даних та створює запис у базі даних.

Після створення вакансія не відображається іншим користувачам відразу. Для забезпечення якості контенту та контролю достовірності інформації передбачено процес модерації. Створена вакансія передається модератору для перевірки, після чого приймається рішення щодо її подальшої публікації.

Для модератора система передбачає окремий функціонал, який починається з авторизації та доступу до модераційної панелі. Модератор може переглядати інформацію про всіх зареєстрованих користувачів системи, блокувати або розблоковувати роботодавців, а також змінювати статус їхньої верифікації. Крім того, модератор має можливість керувати вакансіями та видаляти записи, які порушують правила використання платформи.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						41
Змн.	Арк.	№ докум.	Підпис	Дата		

Алгоритм модерації вакансій передбачає аналіз інформації, доданої роботодавцем. У разі відповідності вакансії правилам платформи модератор підтверджує її публікацію. Після цього вакансія отримує статус активної та стає доступною для перегляду всім користувачам системи. Якщо під час перевірки виявлено порушення або некоректну інформацію, вакансія може бути відхилена або видалена. Такий підхід дозволяє підтримувати якість інформації, що публікується на платформі.

Після публікації вакансії роботодавець починає отримувати заявки від кандидатів. Для кожної вакансії формується окремий список відгуків, який містить інформацію про кандидатів та їхні резюме. Під час відкриття роботодавцем конкретної заявки система автоматично фіксує факт її перегляду та змінює статус на відповідний. Це дозволяє кандидату отримувати актуальну інформацію щодо стану розгляду його заявки.

Для спрощення роботи з великою кількістю кандидатів у системі реалізовано механізм інтелектуального ранжування резюме. Алгоритм ранжування запускається за запитом роботодавця та використовує дані вакансії разом із резюме кандидатів, які подали заявки. Серверна частина системи передає необхідну інформацію до окремого сервісу машинного навчання, який виконує аналіз змісту документів.

У процесі аналізу текстова інформація вакансії та резюме перетворюється у векторні представлення. Після цього виконується обчислення ступеня схожості між вакансією та кожним резюме кандидата. На основі отриманих значень формується рейтинг відповідності кандидатів вимогам вакансії. Результат роботи алгоритму повертається до основної системи та використовується для сортування списку кандидатів від найбільш релевантного до найменш релевантного.

Після аналізу отриманих заявок роботодавець приймає рішення щодо подальшого розгляду кандидатів. Він може змінювати статус заявки відповідно до результатів відбору, наприклад переводити кандидата на наступний етап розгляду, запрошувати на співбесіду або відхиляти заявку.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						42
Змн.	Арк.	№ докум.	Підпис	Дата		

Зміна статусу виконується безпосередньо через інтерфейс системи та автоматично зберігається в базі даних.

Після кожної зміни статусу система формує відповідне сповіщення для кандидата. Механізм сповіщень дозволяє інформувати користувачів про важливі події без необхідності постійно перевіряти сторінки вакансій або власних заявок. У повідомленні міститься інформація про зміну статусу та вакансію, до якої стосується відповідна заявка.

Таким чином, алгоритм роботи вебпорталу охоплює повний цикл взаємодії між роботодавцем та кандидатом - від створення вакансії та її модерації до подання заявки, розгляду кандидатів, інтелектуального ранжування резюме та надсилання повідомлень про результати відбору. Така послідовність дій забезпечує впорядкованість бізнес-процесів та дозволяє автоматизувати основні етапи підбору персоналу.

На основі наведених алгоритмів роботи користувачів побудовано діаграму активності вебпорталу пошуку роботи та підбору персоналу (рис. 2.4) [14]. Вона відображає основні сценарії взаємодії між користувачами різних ролей та системою, демонструючи послідовність виконання ключових процесів - від реєстрації користувачів до обробки заявок на вакансії.

У діаграмі чітко розмежовано логіку роботи трьох основних ролей: роботодавця, шукача роботи та модератора, а також показано роль системи як центрального компонента, відповідального за збереження даних і обробку запитів. Окремо виділено процес модерації вакансій, включаючи етапи аналізу та прийняття рішення про публікацію чи відхилення оголошення.

Діаграма активності дозволяє наочно представити повний життєвий цикл вакансії на платформі та візуалізувати паралельні процеси й альтернативні сценарії роботи системи. Така форма представлення сприяє кращому розумінню бізнес-логіки проєкту, полегшує виявлення можливих вузьких місць та служить важливим інструментом для подальшого тестування й документування функціональності вебпорталу. Крім того, вона підтверджує цілісність і завершеність розробленої функціональної моделі системи.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						43
Змн.	Арк.	№ докум.	Підпис	Дата		

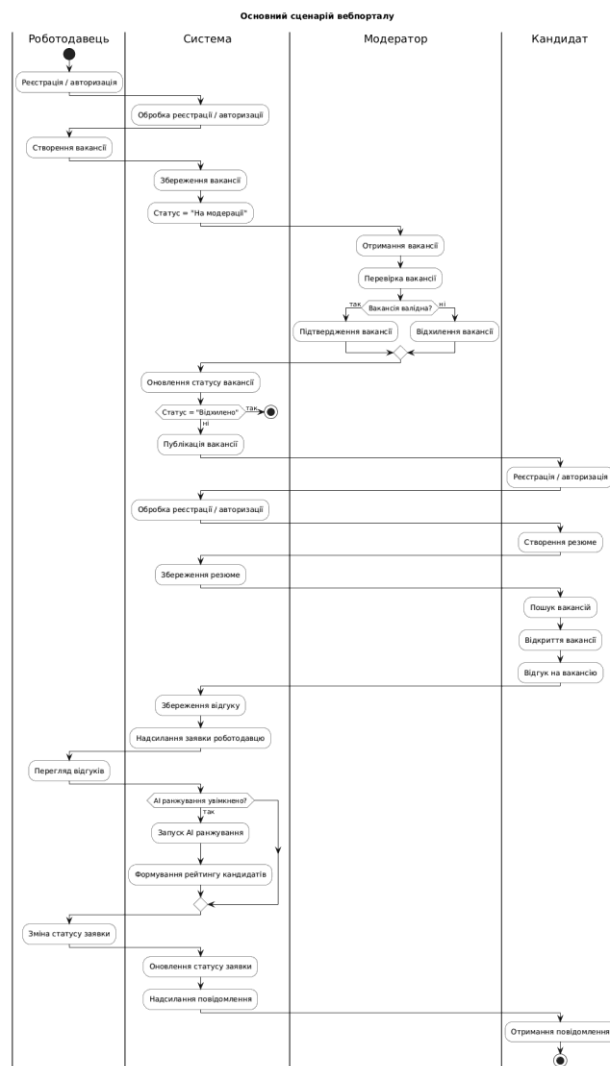


Рис.2.4 – Основна діаграма активності вебпорталу

2.4. Реалізація функціоналу вебпорталу для пошуку роботи та підбору персоналу

Для реалізації серверної частини вебпорталу пошуку роботи та підбору персоналу було обрано фреймворк NestJS. Він базується на Node.js та пропонує модульну архітектуру, яка дозволяє чітко розділити код на контролери, сервіси та модулі. Такий підхід значно полегшує підтримку, тестування та подальше розширення системи. Обмін даними між клієнтською та серверною частинами здійснюється через REST API [15]. Кожен функціональний блок системи надає чітко визначений набір ендпоінтів, доступ до яких реалізовано за допомогою стандартних HTTP-методів: GET, POST, PATCH та DELETE. Архітектура серверної частини побудована за

принципом модульності. Кожна основна предметна область виділена в окремий модуль: VacancyModule - для роботи з вакансіями, ResumeModule - для резюме, AuthModule - для автентифікації та авторизації, NotificationModule - для сповіщень. Така структура забезпечує низьку зв'язність компонентів і дозволяє масштабувати окремі частини системи незалежно одна від одної. У кожному модулі є два ключові шари. Контролери відповідають за обробку вхідних HTTP-запитів, валідацію параметрів маршруту та заголовків, а також формування відповідей клієнту. Вони виступають лише тонким шаром між клієнтом і бізнес-логікою. Основна логіка зосереджена в сервісах. Саме там відбувається перевірка бізнес-правил, взаємодія з базою даних, обробка файлів, розрахунки відповідності вакансій і резюме, робота з пошуковими запитами та інші операції. Винесення бізнес-логіки в окремі сервіси дозволяє зменшити складність контролерів та підвищити читабельність коду. У коді нижче представлено приклад реалізації VacancyController. У ньому визначено маршрути API та здійснюється делегування запитів відповідним методам сервісу.

```
@Authorization(UserRole.EMPLOYER)
@Post('create')
@Throttle({ default: { limit: 2, ttl: 2 * 60 * 1000} })
@HttpCode(HttpStatus.OK)
public async create(
    @Authorized() user: User,
    @Body() dto: CreateVacancyDto
) {
    return this.vacancyService.create(user.id, dto)
}
```

У коді нижче наведено фрагмент VacancyService. У сервісі реалізовано ключові операції: створення та оновлення вакансії, перевірка прав доступу роботодавця тощо.

```
public async create(userId: string, dto: CreateVacancyDto) {
    const employer = await this.findEmployerById(userId);
```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						45
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    if (employer.isBlocked)
      throw new ForbiddenException('Аккаунт заблоковано');
    await this.validateVacancyLanguages(dto);
    return this.prismaService.vacancy.create({
      data: {
        employerId: employer.id, workFormatId: dto.workFormatId, title: dto.title,
        description: dto.description, responsibilities: dto.responsibilities ?? [],
requirements: dto.requirements ?? [],
        weOffer: dto.weOffer ?? [], salaryFrom: dto.salaryFrom, salaryTo: dto.salaryTo,
        locationId: dto.locationId, address: dto.address,
        status: employer.isVerified ? 'APPROVED' : 'PENDING',
        vacancyLanguages: { create: dto.languages?.map(lang => ({ languageId:
lang.languageId, languageLevelId: lang.languageLevelId })) ?? [] }, },
      include: { vacancyLanguages: true, employer: true, workFormat: true, location: true
    });}

```

Використання NestJS дозволило створити добре структурований, модульний і масштабований бекенд, який відповідає сучасним практикам розробки бізнес рівня. Чітке розмежування відповідальностей між шарами значно спрощує подальшу розробку нових функцій та підтримку проєкту.

Для взаємодії серверної частини з базою даних у розробленій системі обрано Prisma ORM. Це сучасний інструмент, який забезпечує типізований та безпечний доступ до даних безпосередньо з TypeScript, значно зменшуючи потребу в написанні raw SQL-запитів. Центральним елементом Prisma є файл схеми (schema.prisma), у якому описується вся структура бази даних. У ньому визначено всі моделі, їхні поля, типи даних, обмеження, індекси та зв'язки між таблицями. На основі цієї схеми Prisma автоматично генерує повністю типізований клієнт, що дозволяє працювати з базою даних за допомогою ланцюжків методів і отримувати підказки IDE на етапі розробки. У схемі представлено ключові моделі системи: User - базова модель користувача, JobSeeker та Employer - розширені профілі відповідно для кандидатів і роботодавців, Resume - детальна інформація про резюме, Vacancy - вакансії. Крім того, створено допоміжні моделі для зберігання даних про досвід роботи,

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		46

освіту, навички, мови, заявки на вакансії (Application), збережені вакансії (SavedVacancy) та сповіщення (Notification).

Особливу увагу приділено моделюванню зв'язків між сутностями. Prisma підтримує відношення один-до-одного, один-до-багатьох та багато-до-багатьох. Наприклад, один роботодавець (Employer) може мати багато вакансій, а одна вакансія може бути пов'язана з кількома вимогами до мов чи навичок. Для багатьох-до-багатьох зв'язків (наприклад, між резюме та навичками) використовуються проміжні таблиці. Такий підхід забезпечує цілісність даних на рівні бази та дозволяє ефективно виконувати складні запити з включенням пов'язаних даних.

Для полів зі строго обмеженим набором значень (роль користувача, статус вакансії, статус заявки, тип сповіщення тощо) застосовуються перерахування (enum) [16]. Це дозволяє контролювати допустимі значення вже на рівні бази даних і уникати некоректних станів у системі. У коді нижче наведено фрагмент файлу schema.prisma з моделлю Language та її зв'язками з іншими сутностями.

```
model Language {
  id Int @id @default(autoincrement())
  name String
  code String @unique
  resumeLanguages ResumeLanguage[]
  vacancyLanguages VacancyLanguage[]
  createdAt DateTime @default(now()) @map("created_at")
  updatedAt DateTime @updatedAt @map("updated_at")
  @@unique([name])
  @@map("languages") }
```

Основні операції з даними реалізуються через методи Prisma, такі як findMany, findUnique, create, update та delete. Вони дозволяють виконувати типові CRUD-операції без необхідності написання SQL-запитів, що значно спрощує розробку та підвищує читабельність коду. Наприклад, при отриманні списку вакансій використовується метод findMany, який дозволяє не лише

отримати дані, але й застосувати фільтрацію, сортування та пагінацію. Це забезпечує ефективну роботу з великими обсягами даних.

```
const [applications, total] = await Promise.all([
  this.prisma.application.findMany({
    where,
    select: {
      id: true, vacancyId: true, resumeId: true, status: true, updatedAt: true,
      vacancy: {
        select: {
          id: true, title: true, description: true,
          salaryFrom: true, salaryTo: true, updatedAt: true,
          location: { select: { city: true } },
          workFormat: { select: { name: true } },
          employer: { select: { companyName: true } },
        }
      },
    },
    orderBy: { updatedAt: "desc" },
    skip, take: limit
  }),
  this.prisma.application.count({ where })
]);
```

У наведеному прикладі сервіс звертається до PrismaClient для отримання даних з таблиці Vacancy. Додатково можуть застосовуватись умови фільтрації, наприклад за статусом вакансії, роботодавцем або іншими параметрами. Отримані дані повертаються у контролер, який передає їх клієнтській частині.

Використання Prisma ORM дозволило суттєво прискорити розробку серверної частини, забезпечити високу типізацію всіх запитів до бази даних, зменшити кількість помилок на етапі роботи з даними та спростити підтримку проєкту в довгостроковій перспективі. Завдяки чіткій та декларативній схемі будь-які зміни в моделях даних швидко відображаються в коді, що є критичним для динамічної системи підбору персоналу.

Під час реалізації механізму реєстрації користувачів у системі застосовується транзакційний підхід до запису даних [17]. Це зумовлено тим, що створення нового користувача передбачає виконання декількох пов'язаних операцій: створення основного запису в таблиці User та обов'язкове створення профілю відповідно до обраної ролі - JobSeeker для кандидатів або Employer

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		48

для роботодавців. Для гарантування цілісності даних використовується транзакція Prisma. Вона дозволяє виконати весь ланцюжок операцій як єдине атомарне ціле: або всі зміни успішно застосовуються, або жодна з них не фіксується. У разі виникнення помилки на будь-якому етапі Prisma автоматично виконує rollback, запобігаючи появі неповних записів у базі даних. У коді нижче наведено реалізацію методу реєстрації користувача з використанням транзакції. У наведеному прикладі спочатку створюється базовий запис користувача, після чого, залежно від обраної ролі, у межах тієї самої транзакції створюється відповідний профіль у таблиці JobSeeker або Employer. Для цього застосовується метод \$transaction, який забезпечує виконання всіх запитів в одному контексті. Використання транзакцій є критичним для даної системи, оскільки роль користувача визначає доступні йому функціональні можливості платформи. Без пов'язаного профілю ролі подальша робота користувача була б неможливою. Транзакційний підхід повністю виключає ситуацію, коли користувач зареєстрований, але не має відповідного профілю.

```
const newUser = await this.prismaService.$transaction(async prisma => {
  const newUser = await this.userService.create(
    prisma, dto.email, dto.password, role, false, AuthMethod.CREDENTIALS
  );
  if (role === UserRole.EMPLOYER)
    await this.employerService.create(prisma, newUser.id);
  if (role === UserRole.JOB_SEEKER)
    await this.jobSeekerService.create(prisma, newUser.id);
  return newUser;
});
```

У серверній частині вебпорталу для обробки результатів виконання операцій використовуються HTTP статус-коди [18]. Вони дозволяють клієнтській частині отримувати стандартизовану інформацію про результат запиту та коректно реагувати на нього. Такий підхід є важливою частиною реалізації REST API та забезпечує зрозумілу взаємодію між клієнтом і

сервером. У системі використовуються стандартні HTTP статус-коди. Наприклад, код 200 OK застосовується у випадку успішного виконання запиту, 201 Created - при створенні нового ресурсу, 400 Bad Request - у разі некоректних вхідних даних, а 401 Unauthorized - при спробі доступу без авторизації.

```
@Recaptcha()
```

```
@Post('register/jobseeker')
```

```
@HttpCode(HttpStatus.CREATED)
```

```
public async registerJobSeeker(@Req() req: Request, @Body() dto: RegisterDto) {  
    return this.authService.register(req, dto, UserRole.JOB_SEEKER) }  
}
```

У розробленому вебпорталі реалізовано механізм розмежування доступу до функціоналу системи на основі ролей користувачів. Для цього використовується спеціальний декоратор, який перевіряє роль авторизованого користувача перед виконанням запиту до відповідного ендпоінта. Такий підхід дозволяє обмежити доступ до окремих функцій системи залежно від ролі користувача, зокрема шукача роботи, роботодавця або модератора. Наприклад, створення вакансії доступне лише роботодавцю, тоді як модератор має доступ до функцій перевірки та управління контентом.

```
@Authorization(UserRole.MODERATOR)
```

```
@Get('moderate/:id')
```

```
@HttpCode(HttpStatus.OK)
```

```
async moderateGetById(@Param('id') id: string) {  
    return this.vacancyService.getVacancyById(id, true); }  
}
```

Реалізація перевірки ролей базується на використанні механізму guards у NestJS. Guard виконується перед обробкою запиту контролером і визначає, чи має користувач достатній рівень доступу для виконання конкретної операції. У випадку невідповідності ролі користувача доступ до ресурсу блокується, і система повертає відповідний HTTP статус-код (403 Forbidden). Це забезпечує захист бізнес-логіки від несанкціонованого доступу.

Для організації обміну даними між клієнтською та серверною частинами вебпорталу активно використовуються об'єкти передачі даних (DTO - Data

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						50
Змн.	Арк.	№ докум.	Підпис	Дата		

Transfer Object). DTO слугують чітким контрактом між клієнтом і сервером, визначаючи структуру вхідних і вихідних даних для кожного ендпоінту REST API, а також забезпечуючи їхню валідацію на вході [19]. У системі створено окремі DTO для основних операцій: створення та оновлення вакансій (CreateVacancyDto, UpdateVacancyStatusDto), резюме (CreateResumeDto), заявок на вакансії (CreateApplicationDto) та профілів користувачів. Кожен DTO чітко вказує дозволені поля, їхні типи та обмеження, що виключає можливість передачі зайвих або некоректних даних на сервер.

```
export class CreateApplicationDto {
  @ApiProperty({
    description: "ID вакансії, на яку подається заявка",
    example: "clx9a8b2c0001v8kq3m1",
  })
  @IsString()
  vacancy_id: string;
  @ApiProperty({
    description: "ID резюме, яке використовується для відгуку",
    example: "clx9a8f4d0002v8kq7z9p",
  })
  @IsString()
  resume_id: string; }
```

Валідація даних у DTO реалізована за допомогою бібліотеки class-validator. Правила валідації задаються декларативно через декоратори (@IsNotEmpty(), @IsEmail(), @IsString(), @MinLength(), @MaxLength(), @IsEnum() тощо). Це дозволяє перевіряти формат даних, обов'язковість полів, довжину, числові діапазони та інші умови ще до потрапляння запиту в бізнес-логіку сервісу. Додатково, у поєднанні з бібліотекою class-transformer, DTO автоматично трансформують вхідні дані (наприклад, перетворюють рядки у числа, дати або enum-значення), що забезпечує коректну типізацію в контролерах NestJS.

Для ефективної роботи з великими обсягами даних у вебпорталі реалізовано механізм пагінації при отриманні списків вакансій та резюме.

Пагінація дозволяє обмежувати кількість записів, що повертаються в одному

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						51
Змн.	Арк.	№ докум.	Підпис	Дата		

HTTP-запиті, знижувати навантаження на базу даних і значно прискорювати відгук API. У системі за замовчуванням на одну сторінку повертається 15 записів. Клієнтська частина передає параметри page (номер сторінки) та limit (кількість елементів на сторінці). Такий підхід дає змогу користувачам зручно переглядати результати по частинах, а серверу - оптимально обробляти запити навіть за умови тисяч активних вакансій. У VacancyService пагінація реалізується за допомогою методів Prisma ORM - skip та take. Параметр skip визначає зміщення (кількість пропущених записів), а take - кількість записів, які потрібно повернути. Додатково виконується окремий запит для підрахунку загальної кількості елементів (count), що дозволяє повертати мета-інформацію: поточну сторінку, загальну кількість сторінок, загальну кількість записів та інші.

Для підвищення релевантності підбору кандидатів у вебпорталі було розроблено окремий сервіс семантичного ранжування резюме. Він дозволяє автоматично оцінювати, наскільки конкретне резюме відповідає вимогам вакансії, використовуючи сучасні методи обробки природної мови. В основі сервісу лежить нейронна модель intfloat/multilingual-e5-large з бібліотеки sentence-transformers. Ця модель призначена для створення семантичних векторних представлень (embeddings) тексту [20]. На відміну від простого пошуку за ключовими словами, вона здатна розуміти сенс тексту, синоніми та контекст, що особливо важливо для задачі підбору персоналу. Модель добре підходить для висунутих вимог, оскільки: підтримує велику кількість мов (включаючи українську), ефективно працює з професійною лексикою (навички, посади, обов'язки), показує хороші результати при порівнянні коротких і середніх текстів, якими є опис вакансій і резюме.

Під час тестування сервісу ранжування кандидатів було виявлено критичну проблему продуктивності: обробка 50 резюме займала в середньому 52 секунди. Така затримка робила використання моделі в реальному часі практично неможливим. Аналіз показав, що основною причиною низької швидкості роботи є неефективна реалізація кодування текстів. У початковій

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						52
Змн.	Арк.	№ докум.	Підпис	Дата		

версії метод `model.encode()` викликався окремо для кожного поля (`skills`, `info`, `language`) кожного резюме всередині циклу. Таким чином, для обробки 50 резюме виконувалось 150 незалежних викликів методу `encode()`. Кожен виклик супроводжувався значними накладними витратами: підготовкою тензорів, переміщенням даних і ініціалізацією обчислень у моделі

```
skills_clean, info_clean, lang_clean = preprocess(resume.skills), preprocess(resume.info),
preprocess(resume.language)
```

```
skills_vec = model.encode(skills_clean, convert_to_tensor=True)
```

```
info_vec = model.encode(info_clean, convert_to_tensor=True)
```

```
lang_vec = model.encode(lang_clean, convert_to_tensor=True)
```

```
score_skills = util.cos_sim(vacancy_vec, skills_vec)[0][0].item()
```

```
score_info = util.cos_sim(vacancy_vec, info_vec)[0][0].item()
```

```
score_lang = util.cos_sim(vacancy_vec, lang_vec)[0][0].item()
```

Для вирішення цієї проблеми було впроваджено підхід `batch processing` (пакетної обробки даних) [21]. Замість поелементного кодування всі тексти кожного типу (`skills`, `info`, `language`) попередньо збираються у списки та передаються в модель одним викликом `encode()`.

```
skills_texts=[preprocess(r.skills) for r in data.resumes]; info_texts=[preprocess(r.info) for
r in data.resumes]; lang_texts=[preprocess(r.language) for r in data.resumes]
```

```
skills_vecs=model.encode(skills_texts,convert_to_tensor=True,batch_size=32);
```

```
info_vecs=model.encode(info_texts,convert_to_tensor=True,batch_size=32);
```

```
lang_vecs=model.encode(lang_texts,convert_to_tensor=True,batch_size=32)
```

```
scores_skills=util.cos_sim(vacancy_vec,skills_vecs)[0];
```

```
scores_info=util.cos_sim(vacancy_vec,info_vecs)[0];
```

```
scores_lang=util.cos_sim(vacancy_vec,lang_vecs)[0]
```

Такий підхід дозволяє моделі обробляти дані у вигляді матриць, використовуючи внутрішню паралелізацію обчислень, що є значно ефективнішим за послідовну обробку кожного елемента окремо. Це дозволяє суттєво зменшити кількість звернень до моделі та мінімізувати накладні витрати на кожен виклик. З точки зору алгоритмічної складності, у попередній реалізації кількість операцій зростала лінійно від кількості резюме, тобто мала вигляд $O(N \times F)$, де N - кількість резюме, а F - кількість текстових полів [22].

У новій реалізації кількість викликів моделі є сталою відносно N і дорівнює $O(F)$, що значно покращує масштабованість системи.

У вебпорталі реалізовано систему сповіщень, яка забезпечує інформування користувачів про важливі події, такі як зміна статусу заявки на вакансію, отримання відповіді від роботодавця або оновлення стану вакансії. Дана підсистема відіграє важливу роль у забезпеченні оперативної взаємодії між учасниками платформи. Архітектурно система сповіщень побудована на основі підходу, подібного до патерну Observer [23]. У даному випадку об'єкти-користувачі виступають у ролі “підписників” які отримують повідомлення про зміни стану певних сутностей системи. Події, що виникають у бізнес-логіці (наприклад, зміна статусу заявки), виступають як тригери для генерації сповіщень. При зміні статусу заявки роботодавцем система автоматично створює запис сповіщення у базі даних та прив'язує його до відповідного користувача. Після цього сповіщення стає доступним у клієнтській частині у розділі повідомлень.

```
if (application.status === ApplicationStatus.APPLIED) {
  const updated = await this.prisma.application.update({
    where: {
      id: application.id,
    },
    data: {
      status: ApplicationStatus.VIEWED,
    },
    include: this.applicationInclude,
  });
  await this.notificationsService.notifyApplicationStatusChanged(
    updated.id,
    updated.status,
  );
  return this.formatApplication(updated);
}
```

У вебпорталі застосовано комбінований підхід до побудови клієнтської частини, який поєднує SPA (Single Page Application) та SSR (Server-Side Rendering). Така архітектура дозволяє одночасно забезпечити високу швидкість взаємодії з інтерфейсом та покращену індексацію сторінок пошуковими системами. Основна частина проєкту реалізована як SPA [24]. Це

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						54
Змн.	Арк.	№ докум.	Підпис	Дата		

означає, що після першого завантаження сторінки подальша навігація між розділами виконується без повного перезавантаження сторінки. Дані підвантажуються асинхронно через REST API, що забезпечує швидку реакцію інтерфейсу та покращує користувацький досвід. Такий підхід особливо ефективний для особистих кабінетів користувачів, де відбувається часта взаємодія з даними.

Водночас для сторінок вакансій було реалізовано SSR (Server-Side Rendering). Це рішення було прийнято з метою покращення SEO-оптимізації, оскільки вакансії повинні бути доступні для індексації пошуковими системами та швидко відображатися при переході з зовнішніх джерел [25]. При використанні SSR HTML-сторінка формується на сервері ще до передачі клієнту, що дозволяє пошуковим роботам отримувати повністю готовий контент без необхідності виконання JavaScript.

```
<div className="grid grid-cols-1 lg:grid-cols-3 gap-8 items-start">
  <div className="lg:col-span-2 space-y-8">
    <VacancyHero vacancy={vacancy} />
    <VacancyDetails vacancy={vacancy} />
    <VacancyMap vacancy={vacancy} />
  </div>
  <div className="lg:col-span-1">
    <VacancySidebar vacancy={vacancy} />
  </div>
</div>
```

Поєднання SPA та SSR дозволило досягти балансу між продуктивністю інтерфейсу та SEO-ефективністю платформи. SPA забезпечує швидку та динамічну взаємодію користувача із системою, тоді як SSR гарантує коректну індексацію публічних сторінок та високу видимість вакансій у пошукових системах.

Для візуалізації статистичних даних у клієнтській частині вебпорталу використовується бібліотека Chart.js у поєднанні з бібліотекою-адаптером react-chartjs-2 [26]. Використання графіків дозволяє наочно відображати

інформацію про активність користувачів, кількість нових користувачів, подані заявки та інші показники роботи системи. У розробленому проєкті для побудови графіків використовуються компоненти Line та Bar, які забезпечують створення лінійних і стовпчикових діаграм відповідно. Лінійні графіки застосовуються для відображення змін показників у часі, тоді як стовпчикові діаграми використовуються для порівняння значень між окремими категоріями. Для формування графіка клієнтська частина отримує статистичні дані через REST API, після чого виконується підготовка набору даних у форматі, який підтримується бібліотекою Chart.js. Отримані значення передаються до компонентів графіків, де автоматично формується відповідна візуалізація.

```
const totals = useMemo(() => ({
  newJobSeekers: stats.reduce((s, d) => s + d.newJobSeekers, 0),
  newEmployers: stats.reduce((s, d) => s + d.newEmployers, 0),
  newApplications: stats.reduce((s, d) => s + d.newApplications, 0),
}), [stats]);

{ label: "Шукачі",
  data: stats.map((d) => d.newJobSeekers),
  borderColor: "#3f55ff",
  backgroundColor: "rgba(63,85,255,0.08)",
  borderWidth: 2,
  pointRadius: range === 30 ? 2 : 4,
  fill: true,
  tension: 0.35, },
```

Висновки до другого розділу

У другому розділі виконано проєктування вебпорталу для пошуку роботи та підбору персоналу.

У першому підрозділі визначено основні варіанти використання системи, описано ролі користувачів та побудовано UML-діаграми, що відображають взаємодію між основними компонентами програмного комплексу.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		56

У другому підрозділі розроблено структуру бази даних системи. Спроектовано сутності, необхідні для зберігання інформації про користувачів, роботодавців, шукачів роботи, вакансії, резюме, заявки та повідомлення. Також описано використання Redis для зберігання тимчасових даних, пов'язаних з автентифікацією користувачів та службовими процесами системи.

У третьому підрозділі розроблено алгоритми роботи вебпорталу для різних категорій користувачів, зокрема шукачів роботи, роботодавців, модераторів та неавторизованих відвідувачів. Для наочного відображення бізнес-процесів побудовано діаграми активності, які демонструють послідовність виконання основних операцій системи.

У четвертому підрозділі розглянуто практичну реалізацію програмного забезпечення. Описано архітектуру REST API на основі NestJS, використання Prisma ORM для взаємодії з базою даних, механізми авторизації та розмежування доступу, застосування DTO для валідації даних, реалізацію пагінації та системи сповіщень. Також розглянуто інтеграцію окремого ML-сервісу для інтелектуального ранжування кандидатів та використання комбінованого підходу SPA і SSR для забезпечення високої продуктивності інтерфейсу та покращення SEO-показників вебпорталу.

Отримані результати проєктування стали основою для подальшої реалізації та тестування проєкту, забезпечивши її функціональність, масштабованість та відповідність поставленим вимогам.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		57

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПОРТАЛОМ

3.1. Локальне встановлення та налаштування системи

Для забезпечення працездатності вебпорталу було виконано розгортання всіх її компонентів на серверному середовищі. Фізична архітектура системи та взаємодія між її складовими представлені на діаграмі розгортання (рис. 3.1).

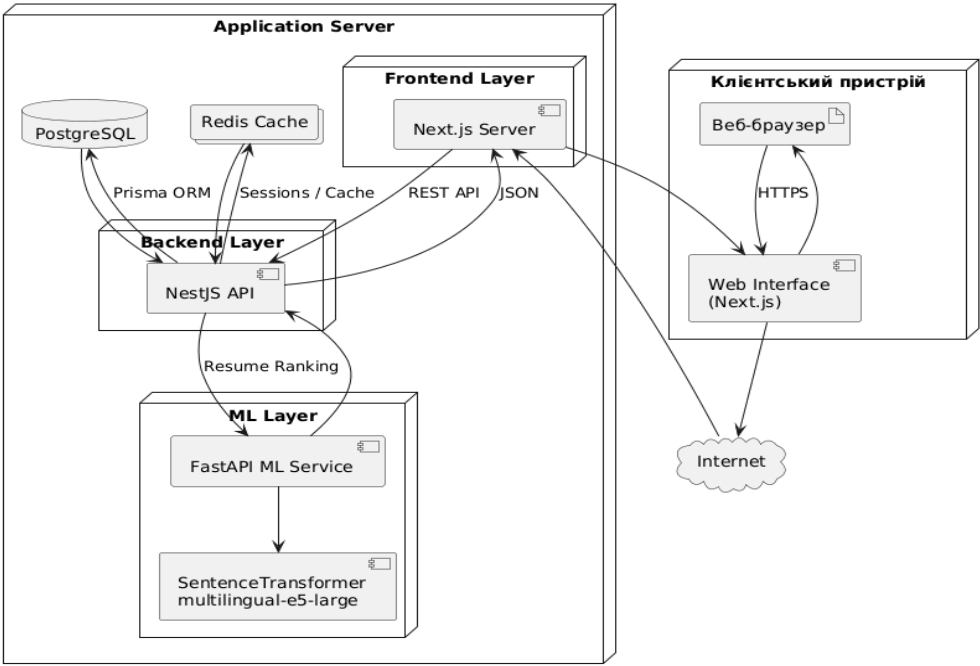


Рис. 3.1 – Діаграма розгортання вебпорталу на локальному пристрої

Діаграма розгортання відображає розміщення програмних компонентів системи на обчислювальних ресурсах та взаємозв'язки між ними. Архітектура системи побудована за клієнт-серверним принципом та складається з основного вебпорталу, реалізованого на платформі NestJS, і окремого сервісу машинного навчання, реалізованого на FastAPI. Взаємодія між компонентами здійснюється через HTTP API.

Основними компонентами системи є клієнтський рівень, серверна частина, ML-сервіс, база даних та система кешування. Клієнтський рівень представлений веб-браузером користувача та frontend-частиною проєкту, яка розроблена на фреймворку Next.js і забезпечує інтерфейс взаємодії з

платформою. Саме через цей рівень користувачі отримують доступ до функціоналу системи та виконують основні дії.

Серверна частина реалізована у вигляді Backend API на фреймворку NestJS і включає основні функціональні модулі, такі як автентифікація, управління вакансіями та резюме. Вона відповідає за обробку запитів від клієнтської частини, реалізацію бізнес-логіки та взаємодію з базою даних.

Окремим компонентом системи є ML-сервіс, реалізований на базі FastAPI. Він використовується для семантичного ранжування резюме та вакансій. У межах цього сервісу застосовується модель SentenceTransformer (intfloat/multilingual-e5-large), яка виконує обчислення релевантності між вакансіями та резюме на основі їхнього змісту.

Для збереження даних використовується PostgreSQL, яка є основною реляційною базою даних системи. Вона забезпечує зберігання інформації про користувачів, вакансії, резюме, заявки та сповіщення, а також підтримує складні запити та зв'язки між сутностями.

Для підвищення продуктивності системи використовується Redis, який виконує роль системи кешування та застосовується для зберігання користувацьких сесій. Це дозволяє зменшити навантаження на основну базу даних і пришвидшити обробку запитів.

Користувач взаємодіє з платформою через веб-браузер. Клієнтська частина (Next.js) надсилає HTTP-запити до Backend API. Серверна частина обробляє бізнес-логіку, виконує автентифікацію та авторизацію, після чого взаємодіє з базою даних PostgreSQL через Prisma ORM. Для пришвидшення роботи використовуються запити в Redis для підтвердження актуальності сесії.

Для безпечного та зручного розгортання системи всі конфіденційні параметри винесено у файл .env [27]. До них належать: параметри підключення до PostgreSQL та Redis, секретні ключі сесій, налаштування OAuth-провайдерів, ключі Google reCAPTCHA, параметри SMTP-сервера для надсилання листів

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						59
Змн.	Арк.	№ докум.	Підпис	Дата		

Для налаштування OAuth-автентифікації необхідно зареєструвати застосунок у консолі відповідного провайдера. Для Google OAuth використовується Google Cloud Console [28]. Після створення проєкту розробник переходить до розділу налаштування OAuth Consent Screen, де визначаються базові параметри застосунку (назва, email підтримки, домен). Далі у секції Credentials створюється OAuth 2.0 Client ID, де вказуються дозволені URI перенаправлення (redirect URI), які використовуються після успішної авторизації користувача (рис.3.2) .

The image shows a dark-themed interface for configuring OAuth. It has two main sections: 'Authorized JavaScript origins' and 'Authorized redirect URIs'. Each section has a subtitle indicating its use (browser requests vs. web server requests). Both sections contain two input fields with pre-filled URLs and a '+ Add URI' button below them.

Рис.3.2 – Поля для вводу URI перенаправлень (OAuth Google)

У результаті генеруються два ключі: Client ID та Client Secret, які додаються до конфігураційного файлу середовища (.env). Після цього Backend API використовує ці ключі для ініціації OAuth-процесу, перенаправлення користувача на сторінку авторизації та обміну authorization code на access token.

Для захисту вебпорталу від ботів використовується сервіс Google reCAPTCHA [29]. Для його підключення необхідно перейти до Google reCAPTCHA Admin Console та зареєструвати новий сайт. Під час реєстрації вказується домен, на якому буде використовуватись система, та тип reCAPTCHA (reCAPTCHA v2). Після створення проєкту генеруються два ключі: Site Key та Secret Key (рис.3.3). Site Key використовується на клієнтській частині (Frontend) для ініціалізації reCAPTCHA та отримання

токена підтвердження, тоді як Secret Key використовується на серверній частині (Backend API) для перевірки валідності цього токена через запит до API Google.

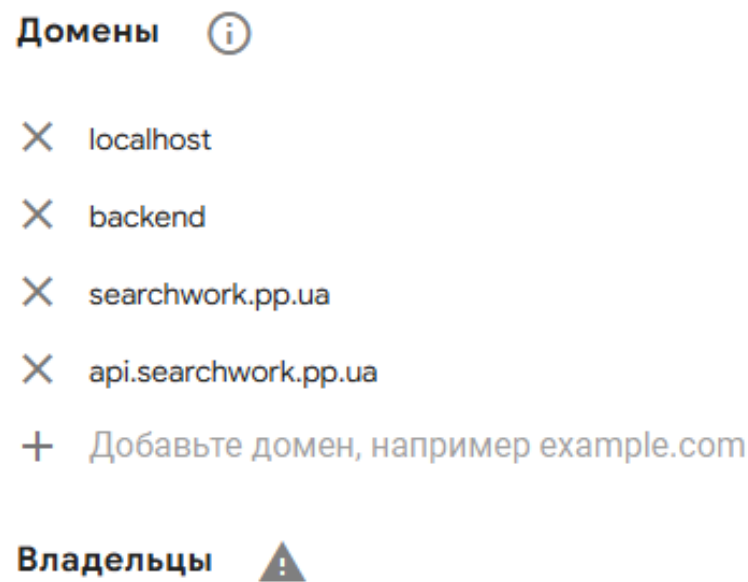


Рис.3.3 – Домени для використання на цих сайтах reCAPTCHA

Для забезпечення зручного локального розгортання та уніфікації середовища виконання всі компоненти системи можуть бути запущені в контейнерах Docker [30]. Це дозволяє ізолювати сервіси (Frontend, Backend, ML-сервіс, PostgreSQL та Redis) та гарантує ідентичність середовища на різних етапах розробки та тестування. Кожен компонент запускається як окремий контейнер із власними конфігураційними параметрами (рис.3.4).

☐		Name	Container ID	Image	Port(s)	Actions
☐	▼	diplomaproject	-	-	-	
☐		frontend	39c2284e195e	diplomaproject	3000:3000 ↗	
☐		backend	0b3c5a0c2e8f	diplomaproject	4000:4000 ↗	
☐		ml_service	b2ea7fb26619	diplomaproject	8000:8000 ↗	
☐		postgres_v2	fa78b0b6acce	postgres:1	5433:5432 ↗	
☐		redis	338fdebad0eb	redis:5.0	6379:6379 ↗	

Рис.3.4 – Docker контейнери вебпорталу

3.2. Розгортання та налаштування системи в мережі Інтернет

Архітектура розгортання проєкту представлена на діаграмі (рис. 3.5), яка відображає взаємодію між клієнтською частиною, мережею Інтернет, сервісом Cloudflare Tunnel та локально розгорнутим середовищем виконання. Дана діаграма дозволяє наочно продемонструвати шлях проходження запиту від користувача до серверних компонентів системи. Вона також ілюструє спосіб публікації локального проєкту в мережі Інтернет через доменне ім'я.

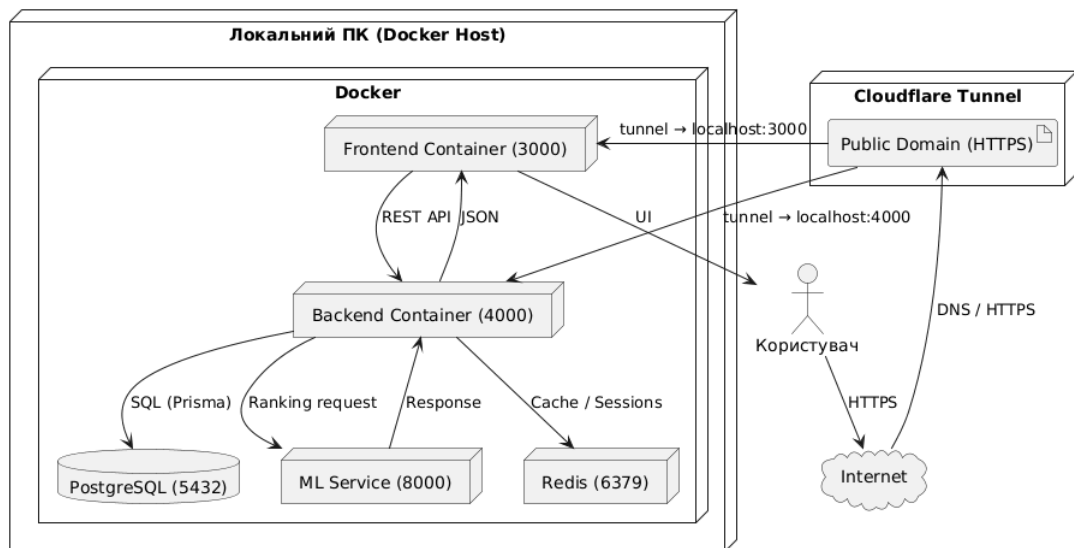


Рис.3.5 – Діаграма розгортання вебпорталу в мережі Інтернет

Для забезпечення доступу до локально розгорнутої системи з мережі Інтернет було використано механізм тунелювання через Cloudflare Tunnel. Цей підхід дозволяє опублікувати локальний сервер без необхідності відкриття прямих входних портів на маршрутизаторі або налаштування статичної публічної IP-адреси.

На першому етапі домен було зареєстровано у стороннього DNS-провайдера. Після цього було змінено NS-записи (Name Server) домену, які перенаправили управління DNS-зоною до сервісу Cloudflare (рис.3.6). Це дозволило централізовано керувати доменом через панель Cloudflare та використовувати його інфраструктуру для маршрутизації трафіку [31].

Додатково слід зазначити, що процес зміни NS-записів є критичним етапом делегування домену, оскільки саме він визначає, який DNS-провайдер

буде відповідальним за обробку всіх запитів до доменного імені. Після внесення змін у панелі реєстратора домену відбувається період оновлення DNS-кешів, який може тривати від кількох хвилин до кількох годин залежно від TTL-параметрів. Після завершення цього процесу домен повністю починає обслуговуватися інфраструктурою Cloudflare, що дає можливість гнучко керувати маршрутизацією та безпекою запитів.

The screenshot shows the 'NS-сервери' (NS servers) section in the Cloudflare dashboard. At the top, there is a dropdown menu labeled 'Власні сервери імен' (Own domain servers). Below it, a section titled 'Список своїх серверів імен' (List of your domain servers) contains a table with two entries:

№	Ім'я сервера
1	pete.ns.cloudflare.com
2	gina.ns.cloudflare.com

Below the table is a '+ Додати' (Add) button. At the bottom of the section are two buttons: 'Зберегти' (Save) in green and 'Відмінити' (Cancel) in blue.

Рис. 3.6 – Зміна NS-записів у панелі доменного реєстратора

Після підключення домену до Cloudflare було створено Cloudflare Tunnel, який встановлює захищене вихідне з'єднання з локального середовища розробки до інфраструктури Cloudflare. Тунель працює за принципом ініціації з'єднання з локального сервера назовні, після чого Cloudflare виступає як проміжний проксі-сервер, що перенаправляє вхідні запити з доменного імені на локально запущений проєкт.

Цей підхід є особливо важливим у випадках, коли локальна машина знаходиться за NAT або не має можливості отримати публічну IP-адресу. У такій конфігурації відсутня необхідність у відкритті портів на роутері, що значно зменшує потенційну поверхню атак та підвищує загальний рівень безпеки системи.

Після визначення загальної логіки маршрутизації запитів було виконано детальне локальне налаштування Cloudflare Tunnel на стороні середовища розробки. Для цього використовувалася утиліта `cloudflared`, яка дозволяє керувати тунелями безпосередньо з командного рядка та інтегрувати їх у локальну файлову систему конфігурації.

На першому етапі було ініційовано створення нового тунелю за допомогою команди створення з унікальним іменем проєкту: `cloudflared`

tunnel create searchwork». У результаті виконання цієї команди система згенерувала унікальний ідентифікатор тунелю та автоматично створила службовий JSON-файл із обліковими даними автентифікації. Даний файл містить криптографічні ключі, що використовуються для встановлення довіреного з'єднання між локальним середовищем та інфраструктурою Cloudflare. Саме цей ідентифікатор надалі використовується для прив'язки конфігурації до конкретного тунелю.

Наступним етапом стало прив'язування створеного тунелю до доменної інфраструктури шляхом налаштування DNS-маршрутизації. Для цього було виконано команду: «cloudflared tunnel route dns searchwork». Дана операція автоматично створює відповідні DNS-записи у зоні Cloudflare та зв'язує доменні імена з ідентифікатором тунелю. У результаті цього запити, що надходять на визначені піддомени, починають маршрутизуватися через створений тунель без додаткових ручних налаштувань DNS-записів.

Після цього було створено локальний конфігураційний файл config.yml, який визначає поведінку тунелю під час запуску на локальній машині. Даний файл містить основні параметри підключення, включаючи назву тунелю та шлях до файлу автентифікаційних даних, який забезпечує доступ до Cloudflare Tunnel сервісу. Зокрема, у конфігурації було явно вказано параметр credentials-file, який містить абсолютний шлях до JSON-файлу облікових даних, розташованого у локальній файловій системі користувача: «C:\Users\Aliev\.cloudflared\6b4b6273-f4eb-447d-b401-3be08ebd78cf.json». Цей файл використовується при кожному запуску тунелю для підтвердження автентичності клієнта та встановлення захищеного з'єднання з мережею Cloudflare. Відсутність або некоректність цього файлу унеможливило ініціалізацію тунелю.

Ключовим елементом конфігурації є блок маршрутизації вхідного трафіку, який визначає відповідність між доменними іменами та локальними сервісами. У межах реалізованої системи було налаштовано розділення на два незалежні сервіси: клієнтський інтерфейс та серверну API-частину. Для

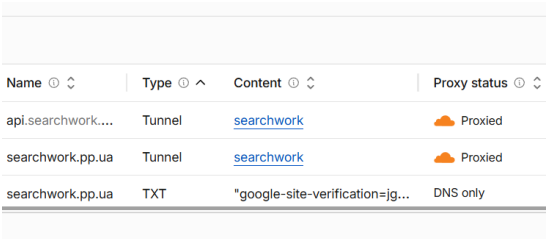
					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						64
Змн.	Арк.	№ докум.	Підпис	Дата		

основного домену було визначено наступне правило маршрутизації: «hostname: searchwork.pp.ua | service: http://localhost:3000». Ця конфігурація забезпечує перенаправлення всіх HTTP-запитів до локального frontend-застосунку, який працює на порту 3000.

Окремо було налаштовано маршрутизацію для API-шару системи через піддомен: «hostname: api.searchwork.pp.ua | service: http://localhost:4000». У цьому випадку всі запити до API автоматично перенаправляються до локального backend-сервера, що працює на порту 4000, забезпечуючи логічне розділення клієнтської та серверної логіки в межах однієї інфраструктури розгортання.

Після збереження конфігураційного файлу тунель запускається як фоновий процес на локальному комп'ютері, після чого Cloudflare починає використовувати визначені правила маршрутизації для обробки вхідного трафіку. Це дозволяє забезпечити стабільний доступ до окремих компонентів системи без необхідності ручного керування мережевими параметрами під час кожного запуску проєкту.

У конфігурації тунелю було визначено маршрутизацію HTTP-запитів до відповідних локальних сервісів, зокрема до frontend та backend API, які працюють на різних портах. Таким чином, користувач отримує доступ до системи через доменне ім'я, тоді як фізичне розгортання залишається локальним (рис.3.7).



Name	Type	Content	Proxy status
api.searchwork...	Tunnel	searchwork	Proxied
searchwork.pp.ua	Tunnel	searchwork	Proxied
searchwork.pp.ua	TXT	*google-site-verification=jg...	DNS only

Рис.3.7 – Налаштовані маршрути запитів

Окремо варто зазначити, що така схема дозволяє масштабувати архітектуру без зміни зовнішньої точки входу: при додаванні нових сервісів достатньо лише оновити конфігурацію маршрутизації в Cloudflare Tunnel без необхідності зміни DNS або інфраструктури доступу.

Після налаштування тунелю було виконано конфігурацію HTTPS-захисту через Cloudflare SSL/TLS. У межах цього етапу було активовано використання захищеного з'єднання між користувачем та сервером, що реалізується через автоматично згенеровані SSL-сертифікати Cloudflare.

У панелі керування SSL/TLS було обрано режим повного шифрування з перевіркою сертифіката на стороні походження (Full (strict)), що забезпечує наскрізне шифрування трафіку між клієнтом, Cloudflare та локальним сервером (рис.3.8). Це дозволяє гарантувати цілісність і конфіденційність переданих даних, а також захищає систему від атак типу man-in-the-middle

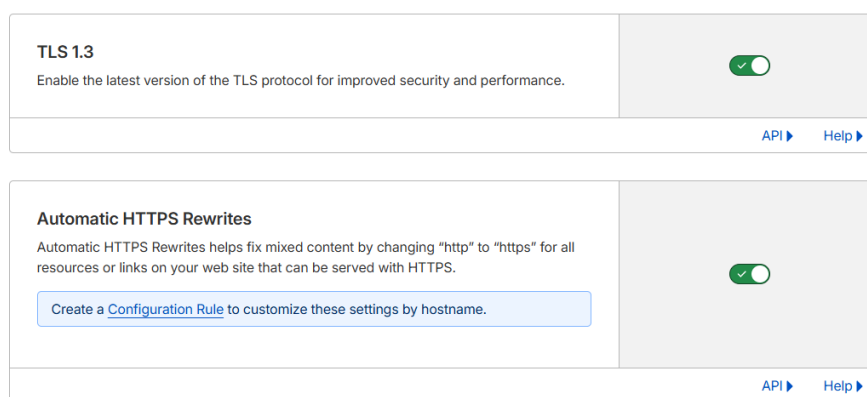


Рис. 3.8 – Налаштування SSL/TLS у панелі Cloudflare

Додатково було налаштовано примусове перенаправлення всіх HTTP-запитів на HTTPS, що забезпечує використання виключно захищеного протоколу передачі даних. Це є стандартною практикою для сучасних вебсистем, оскільки дозволяє уникнути передачі незашифрованої інформації між клієнтом та сервером.

У процесі роботи системи всі запити користувачів проходять через інфраструктуру Cloudflare, яка виконує роль reverse проху. Спочатку запит надходить до глобальної мережі Cloudflare, після чого відповідно до DNS-конфігурації та правил маршрутизації він перенаправляється через встановлений тунель до локального середовища виконання. Там запит обробляється відповідним компонентом системи, після чого відповідь повертається у зворотному напрямку через той самий канал зв'язку.

Таким чином реалізується багаторівнева архітектура доступу до системи, яка поєднує локальне розгортання з глобальною мережевою інфраструктурою Cloudflare. Це дозволяє досягти високого рівня доступності, стабільності та безпеки без необхідності використання складної серверної інфраструктури.

3.3. Структура інтерфейсу

Для забезпечення зручної взаємодії користувачів із функціональними можливостями платформи було розроблено вебінтерфейс, який надає доступ до всіх основних операцій системи. Залежно від ролі користувача набір доступних функцій може відрізнятися, проте базовий перегляд вакансій доступний усім відвідувачам сайту.

Після відкриття головної сторінки користувачу відображається форма пошуку вакансій, яка є основним елементом інтерфейсу платформи (рис. 3.9). Вона містить поле для введення назви посади або ключових слів, а також поле для вибору міста. Введені параметри використовуються для формування пошукового запиту та подальшого відображення вакансій, що відповідають заданим критеріям.

Рис.3.9 – Головна сторінка вебпорталу

Після введення параметрів пошуку та натискання кнопки пошуку користувач переходить на сторінку результатів, де відображаються вакансії, що відповідають заданим критеріям (рис. 3.10). Для кожної вакансії відображається основна інформація, зокрема назва посади, назва компанії, місце роботи, формат зайнятості та рівень заробітної плати.

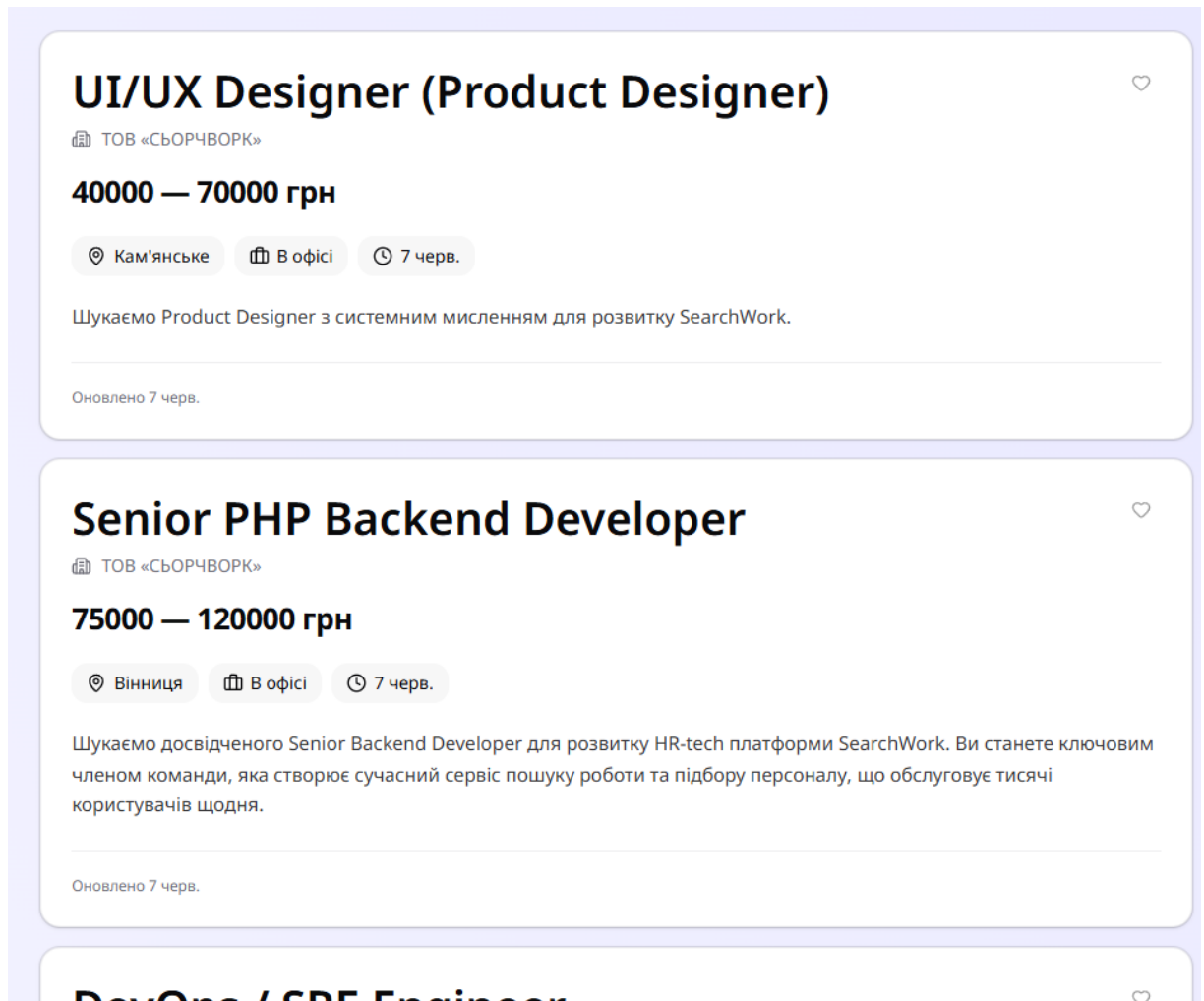


Рис. 3.10 – Сторінка результатів пошуку

Для уточнення результатів пошуку користувач може скористатися додатковими фільтрами. Система підтримує фільтрацію вакансій за рівнем заробітної плати, вимогами до знання мов, за містом розташування та за назвою вакансії. У результаті застосування фільтрів користувач отримує оновлений набір даних у режимі реального часу без необхідності перезавантаження сторінки. (рис. 3.11).

Фільтри

Знайди потрібну вакансію

Очистити

Посада

Місто

Зарплата

Мови

☐ Азербайджанська
 ☐ Англійська
 ☐ Арабська
 ☐ Болгарська
 ☐ В'єтнамська
 ☐ Грецька

Застосувати фільтри

Рис.3.11 – Фільтрація вакансій

Після вибору вакансії користувач переходить на сторінку її детального перегляду. На цій сторінці відображається повний опис вакансії, перелік вимог до кандидата, посадові обов'язки, умови роботи, інформація про роботодавця та інші відомості, необхідні для прийняття рішення щодо подання заявки (рис. 3.12).

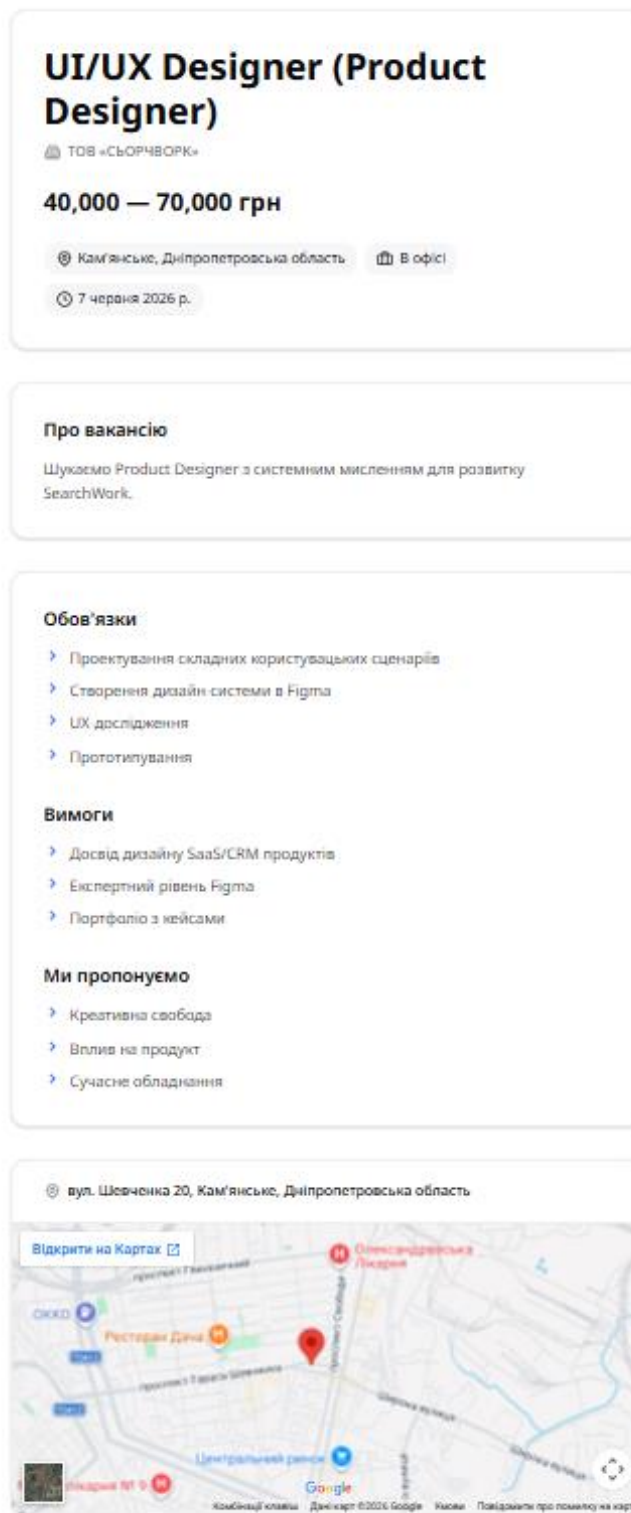


Рис.3.12 – Сторінка перегляду вакансії

Для отримання доступу до повного функціоналу системи користувач повинен пройти процедуру реєстрації або авторизації. Сторінка авторизації дозволяє виконати вхід за допомогою електронної пошти та пароля або скористатися зовнішніми сервісами автентифікації через OAuth (рис. 3.13).

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		70

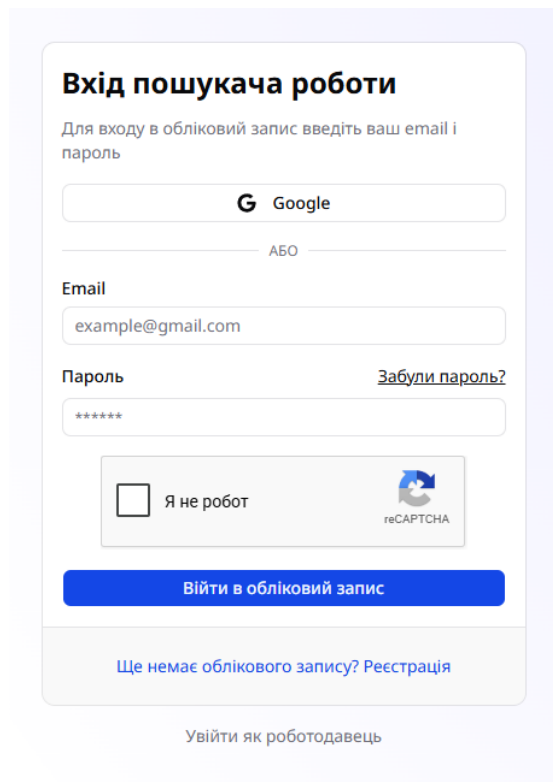


Рис.3.13 – Сторінка авторизації

Якщо користувач не має облікового запису, він може пройти процедуру реєстрації. Під час реєстрації необхідно вказати основні персональні дані та підтвердити адресу електронної пошти. Після успішного створення облікового запису користувач отримує доступ до особистого кабінету (рис. 3.14).

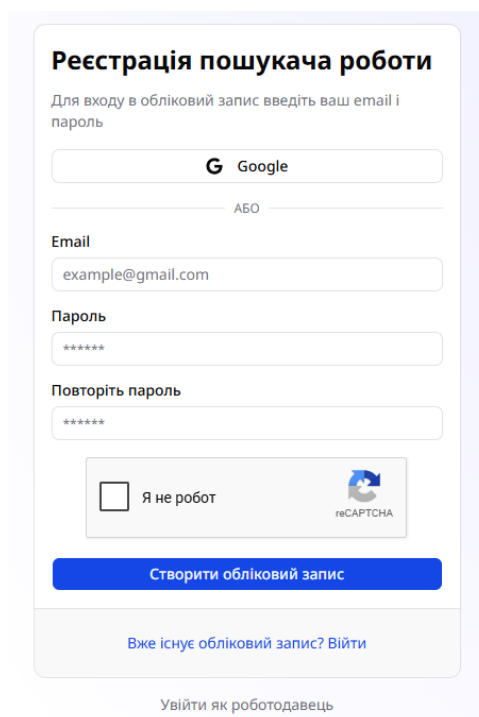


Рис.3.14 – Сторінка реєстрації

Після авторизації роботодавець отримує доступ до списку створених ним вакансій. У даному розділі відображаються всі вакансії, які були додані користувачем, із зазначенням їх основних параметрів, зокрема назви, дати створення та поточного статусу (рис. 3.15).

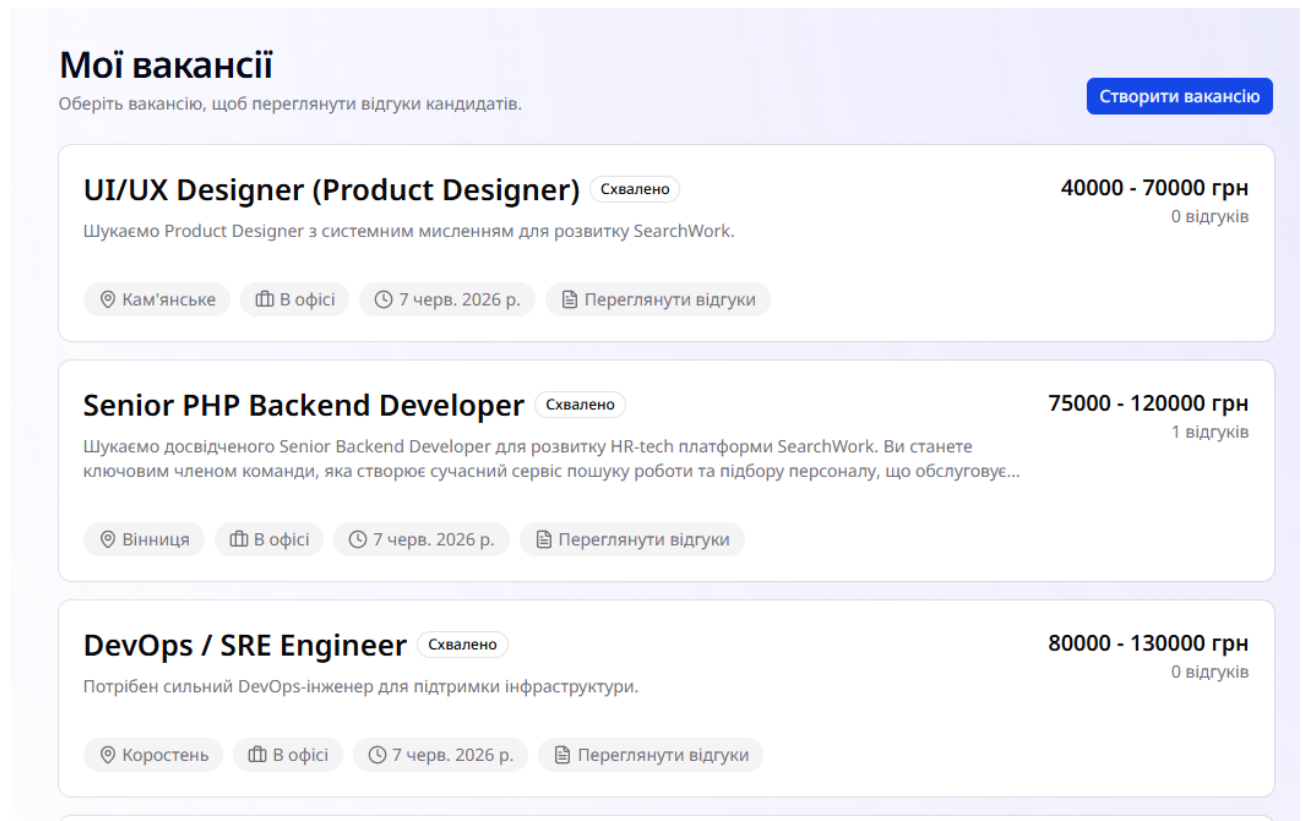


Рис.3.15 – Список створених вакансій роботодавця

Кожна вакансія у списку має власний статус, який визначає її доступність у системі. Статус “На перевірці” означає, що вакансія перебуває на етапі перевірки або очікує підтвердження. Статус “Схвалено” вказує на те, що вакансія опублікована та доступна для перегляду користувачами платформи. У разі необхідності вакансія може мати статус “Архівовано”, що означає її деактивацію та приховування з публічного списку.

Після переходу до конкретної вакансії роботодавцю відображається список усіх поданих резюме кандидатів. У даному розділі зібрані всі заявки, які були надіслані на обрану вакансію, що дозволяє роботодавцю ефективно керувати процесом відбору кандидатів (рис. 3.16).

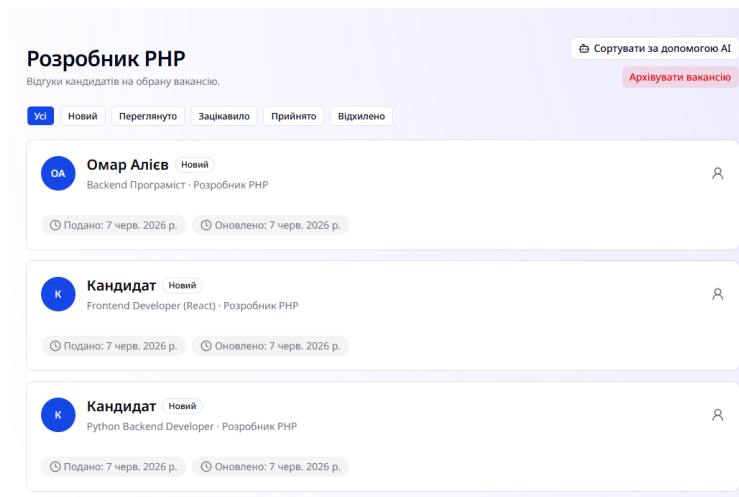


Рис.3.16 – Список поданих резюме на вакансію

У системі також реалізовано можливість архівації самої вакансії. Архівування означає переведення вакансії в неактивний стан, після чого вона більше не відображається у публічному списку та не приймає нові відгуки. При цьому дані вакансії та всі пов'язані з нею заявки зберігаються в системі для подальшого перегляду або аналітики.

Окремо передбачено функціонал автоматизованого ранжування резюме за допомогою модуля машинного навчання. Даний механізм дозволяє впорядкувати кандидатів за рівнем відповідності вакансії, використовуючи семантичний аналіз тексту резюме та опису вакансії. Результатом роботи алгоритму є сортування списку резюме за релевантністю, що значно спрощує процес первинного відбору кандидатів (рис.3.17).

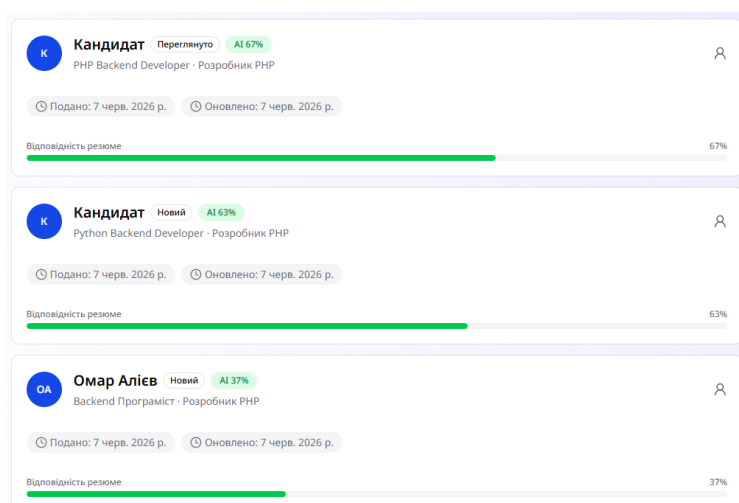


Рис.3.17 – Відсортований список за допомогою модуля машинного навчання

Окремим функціональним блоком системи є інтерфейс шукача роботи, який надає можливість створення та управління власними резюме. Після авторизації користувач отримує доступ до розділу резюме, де може створити нове резюме або видалити існуюче (рис.3.18).

Рис.3.18 – Інтерфейс створення резюме шукача роботи

Під час створення резюме користувач заповнює основні структуровані поля, які відображають його професійний профіль. Зокрема, передбачено введення інформації про навички, які характеризують технічні та професійні компетенції кандидата. Додатково користувач може зазначити рівень освіти, що дозволяє оцінити його академічну підготовку.

Також у резюме реалізовано можливість додавання мовних знань із зазначенням рівня володіння кожною мовою, що є важливим параметром при автоматизованому відборі кандидатів. Окремим текстовим полем передбачено опис про себе, де користувач може в довільній формі вказати додаткову інформацію про свій досвід, професійні інтереси та кар'єрні цілі.

Сформоване резюме зберігається у системі та використовується під час подачі заявок на вакансії, а також для подальшого автоматизованого аналізу та ранжування за допомогою модуля машинного навчання.

Окремим функціональним модулем системи є інтерфейс модератора, який призначений для загального контролю та моніторингу стану вебпорталу. Після входу в систему модератор отримує доступ до головної панелі керування, яка містить узагальнену статистичну інформацію про роботу сервісу (рис.3.19).

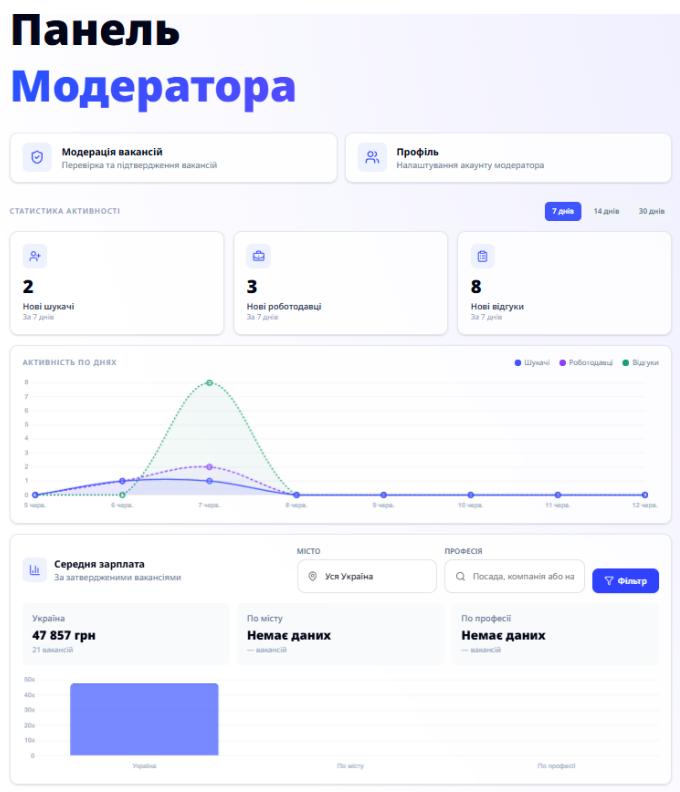


Рис.3.19 – Панель модератора зі статистикою системи

На головній сторінці панелі відображаються ключові аналітичні показники, зокрема кількість зареєстрованих користувачів, кількість нових користувачів за певний період, а також загальна кількість активних відгуків на вакансії. Ці дані дозволяють оцінити динаміку розвитку платформи та активність її користувачів.

Окремо реалізовано блок аналітики заробітних плат, який відображає середні значення запропонованої заробітної плати по вакансіях. Даний показник дозволяє модератору або адміністратору системи аналізувати ринок

праці в межах платформи та відстежувати загальні тенденції у пропозиціях роботодавців.

Наразі функціональність даного модуля перебуває на етапі тестування, яке здійснюється методом поступового впровадження (feature rollout) для обмеженої групи користувачів. На поточному етапі доступ до аналітичного блоку надано виключно користувачам із роллю модератора, що дозволяє перевірити коректність розрахунків, стабільність роботи та зручність інтерфейсу перед повномасштабним запуском.

Окремим елементом інтерфейсу модератора є список неперевірених вакансій, який формується з усіх новостворених вакансій до моменту їх публічного доступу (рис.3.20). Даний розділ призначений для попередньої перевірки коректності та відповідності вакансій встановленим вимогам платформи перед їх публікацією.

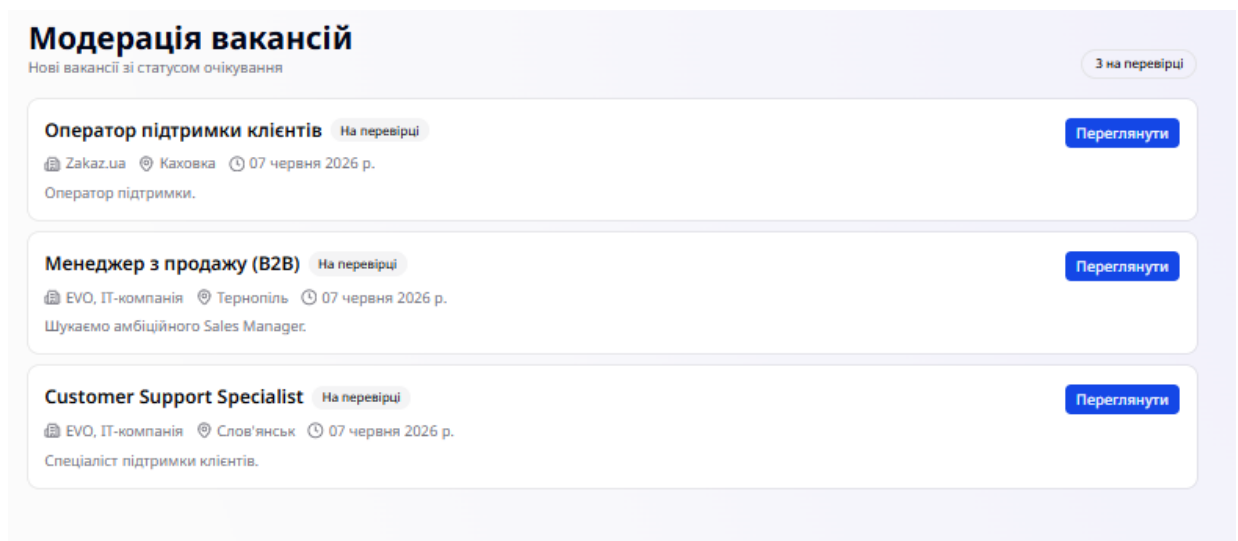


Рис.3.20 – Список неперевірених вакансій

3.4. Тестування роботи вебпорталу

Тестування є однією з ключових складових життєвого циклу розробки програмного забезпечення. Воно спрямоване на перевірку коректності функціонування системи, виявлення можливих помилок та оцінювання відповідності реалізованого функціоналу вимогам технічного завдання.

Проведення тестування дозволяє забезпечити стабільність роботи вебпорталу та підтвердити правильність реалізації основних користувацьких сценаріїв.

У межах розробленого вебпорталу було застосовано два види тестування: функціональне та навантажувальне. Функціональне тестування виконувалося вручну за допомогою тест-кейсів, які містили послідовність дій користувача, очікуваний результат та фактичний результат виконання (табл 3.1 – 3.4) [32]. Такий підхід дозволив перевірити коректність роботи основних функцій системи, зокрема реєстрації та авторизації користувачів, створення та пошуку вакансій, подачі заявок, роботи з резюме та модулів модерації.

Таблиця 3.1

Тест-кейс 1. Перевірка функціоналу для неавторизованого користувача

№	Кроки	Очікуваний результат	Отриманий результат	Статус
1	Перехід на головну сторінку вебпорталу	Відображається форма пошуку вакансій	Як і очікувалось	Виконано
2	Виконання пошуку вакансій за назвою посади	Відображаються вакансії, що відповідають пошуковому запиту	Як і очікувалось	Виконано
3	Перехід на сторінку створення резюме без авторизації	Виконується перенаправлення на сторінку входу	Як і очікувалось	Виконано
4	Перехід на сторінку створення вакансії без авторизації	Виконується перенаправлення на сторінку входу	Як і очікувалось	Виконано
5	Перегляд детальної інформації про вакансію	Відображається сторінка вакансії з повним описом	Як і очікувалось	Виконано

Результати функціонального тестування вебпорталу для неавторизованого користувача наведено у таблиці 3.1. Наведений тест-кейс дозволяє перевірити базові сценарії взаємодії з системою без проходження процедури авторизації, зокрема пошук вакансій, перегляд їх детальної інформації та контроль доступу до функцій створення резюме і вакансій.

Тест-кейс 2. Перевірка функціоналу модератора

№	Кроки	Очікуваний результат	Отриманий результат	Статус
1	Авторизація під обліковим записом модератора	Успішний вхід до панелі модератора	Як і очікувалось	Виконано
2	Перехід до списку неперевірених вакансій	Відображається перелік вакансій зі статусом PENDING	Як і очікувалось	Виконано
3	Підтвердження вакансії	Статус вакансії змінюється на APPROVED	Як і очікувалось	Виконано
4	Перегляд статистики вебпорталу	Відображаються статистичні показники системи	Як і очікувалось	Виконано
5	Перегляд аналітики середніх заробітних плат	Відображаються розраховані показники заробітних плат	Як і очікувалось	Виконано

Результати перевірки функціональності модератора представлені у таблиці 3.2. Даний тест-кейс охоплює основні операції модерації, включаючи авторизацію в системі, перегляд вакансій зі статусом очікування перевірки, їх затвердження, а також доступ до аналітичних даних та статистичних показників роботи вебпорталу. Це дозволяє забезпечити контроль якості контенту, що публікується на платформі, та підтримувати актуальність і достовірність інформації для кінцевих користувачів. Крім того, перевірка адміністративного функціоналу є важливою з точки зору стабільності системи при виконанні критичних операцій, пов'язаних з управлінням даними. Реалізація таких перевірок також гарантує коректний розподіл прав доступу між різними ролями користувачів. В цілому це підтверджує готовність системи до експлуатації в умовах реального використання.

Таблиця 3.3

Тест-кейс 3. Перевірка функціоналу шукача роботи

№	Кроки	Очікуваний результат	Отриманий результат	Статус
1	Створення нового резюме	Резюме успішно створюється та зберігається в системі	Як і очікувалось	Виконано
2	Додавання навичок, освіти та мов до резюме	Інформація успішно зберігається	Як і очікувалось	Виконано
3	Видалення існуючого резюме	Видалення успішно стирається з бази даних	Як і очікувалось	Виконано
4	Подання заявки на вакансію	Створюється новий відгук на вакансію	Як і очікувалось	Виконано
5	Перегляд статусу поданої заявки	Відображається актуальний статус заявки	Як і очікувалось	Виконано

У таблиці 3.3 наведено результати тестування функціоналу шукача роботи. Тест-кейс охоплює ключові користувацькі сценарії, пов'язані зі створенням та управлінням резюме, подачею заявок на вакансії, а також переглядом їх поточного статусу. Це дозволяє підтвердити коректність реалізації основних функцій взаємодії кандидата з платформою.

Таблиця 3.4

Тест-кейс 4. Перевірка функціоналу роботодавця

№	Кроки	Очікуваний результат	Отриманий результат	Статус
1	Створення нової вакансії	Вакансія успішно створюється	Як і очікувалось	Виконано
2	Перегляд списку власних вакансій	Відображаються всі вакансії роботодавця	Як і очікувалось	Виконано

№	Кроки	Очікуваний результат	Отриманий результат	Статус
3	Перегляд списку кандидатів на вакансію	Відображаються всі отримані резюме	Як і очікувалось	Виконано
4	Сортування кандидатів за допомогою AI-ранжування	Резюме впорядковуються за релевантністю	Як і очікувалось	Виконано
5	Архівація вакансії	Статус вакансії змінюється на ARCHIVED та вона приховується з пошуку	Як і очікувалось	Виконано

Результати тестування функціональності роботодавця наведено у таблиці 3.4. У межах даного тест-кейсу перевіряється процес створення та керування вакансіями, перегляд списку кандидатів, використання механізму інтелектуального ранжування резюме, а також архівація вакансій. Отримані результати підтверджують коректну роботу основних бізнес-процесів, пов'язаних із роботодавцем.

Окрім функціонального тестування, було проведено навантажувальне тестування з використанням програмного засобу Apache JMeter [33]. Даний інструмент дозволив змодельовати одночасну роботу великої кількості користувачів та оцінити продуктивність серверної частини вебпорталу під навантаженням. Під час тестування аналізувалися такі показники, як середній час відповіді сервера, кількість оброблених запитів, пропускна здатність системи та стабільність роботи API при збільшенні кількості одночасних звернень. Додатково здійснювався моніторинг використання ресурсів серверу, зокрема завантаження процесора та оперативної пам'яті.

У процесі навантажувального тестування в Apache JMeter було змодельовано сценарій звернення до фронтенд-сторінки searchwork.pp.ua/vacancies. Тестування передбачало імітацію 500 одночасних

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		80

віртуальних користувачів із часом нарощування навантаження 5 секунд та без повторного виконання запитів у межах одного циклу (рис.3.21). Така конфігурація дозволила оцінити продуктивність вебінтерфейсу при різкому збільшенні кількості одночасних відвідувань. Параметри тестування були визначені відповідно до даних, наведених у таблиці 2.2, що містить нефункціональні вимоги та сценарії їх перевірки.

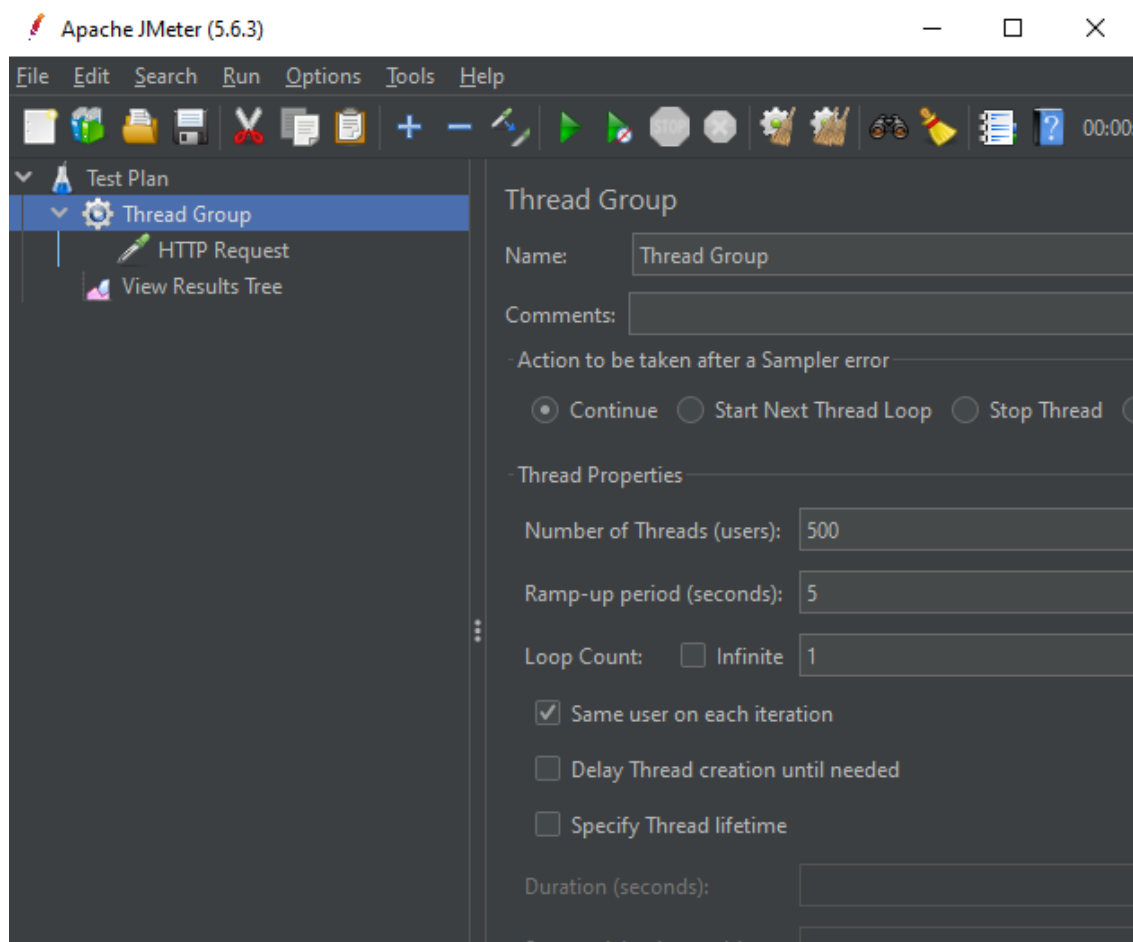


Рис.3.21 – Налаштування Apache Jmeter перед навантажувальним тестуванням

За результатами навантажувального тестування в Apache JMeter було встановлено, що система коректно обробила задане навантаження без виникнення помилок або втрати запитів. Середній час завантаження сторінки (load time) склав 148 мс, що є допустимим показником для даного типу вебпорталу (рис.3.22). Отримані значення повністю відповідають нефункціональним вимогам та підтверджують стабільну роботу системи при заданих умовах навантаження.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						81
Змн.	Арк.	№ докум.	Підпис	Дата		

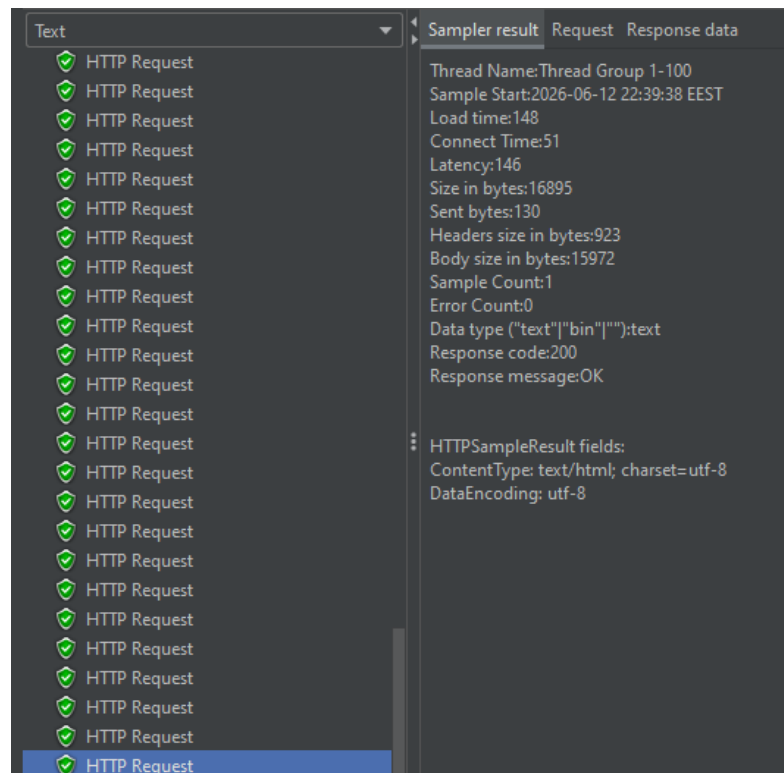


Рис.3.22 – Результат перевірки вебпорталу

Під час навантажувального тестування додатково здійснювався моніторинг стану серверної інфраструктури з метою оцінки впливу одночасних запитів на обчислювальні ресурси системи (рис.3.23). Зокрема, аналізувалося завантаження процесора, яке в пікових значеннях досягало 79%, що свідчить про достатній запас продуктивності та відсутність критичних перевантажень. Отримані результати підтверджують, що система здатна стабільно працювати під заданим навантаженням без деградації продуктивності.

☐			Name	Container ID	Image	Port(s)	CPU (%)
☐			diplomaproject - xxxxxxxxxxxxxxxx	-	-	-	79.6%
☐			frontend	39c2284e195e	diplomaprc	3000:3000	79.12%
☐			backend	0b3c5a0c2e8f	diplomaprc	4000:4000	0%
☐			ml_service	b2ea7fb26619	diplomaprc	8000:8000	0.28%
☐			postgres_v2	fa78b0b6acce	postgres:1	5433:5432	0.01%
☐			redis	338fdebad0eb	redis:5.0	6379:6379	0.19%

Висновки до третього розділу

У третьому розділі розроблено та представлено інтерфейс вебпорталу для пошуку роботи та підбору персоналу. Детально описано архітектуру системи, побудовану за клієнт-серверним підходом, яка включає клієнтську частину, серверну логіку, окремий ML-модуль для ранжування резюме, базу даних PostgreSQL та систему кешування Redis.

Також розглянуто порядок розгортання, налаштування та експлуатації застосунку. Зокрема, описано процес локального запуску системи за допомогою Docker та механізм публікації вебпорталу в мережі Інтернет через Cloudflare Tunnel. Такий підхід забезпечує безпечний доступ до локальної інфраструктури без необхідності відкривати порти назовні. Окрему увагу приділено налаштуванню автентифікації через зовнішні сервіси (OAuth) та захисту від автоматизованих атак за допомогою Google reCAPTCHA.

Завершальною частиною розділу стало тестування вебпорталу, яке охопило функціональне та навантажувальне тестування. Отримані результати підтвердили коректність реалізації основних сценаріїв використання, стабільність роботи системи під навантаженням, а також відповідність нефункціональним вимогам, визначеним у технічному завданні.

ВИСНОВКИ

У процесі виконання даної роботи було досягнуто поставленої мети - розробки та впровадження вебпорталу для пошуку роботи та підбору персоналу, що відповідає визначеним функціональним і нефункціональним вимогам. Робота охоплювала повний життєвий цикл розробки програмного продукту: від аналізу предметної області до реалізації, тестування та розгортання системи.

У першому розділі було проаналізовано предметну область, визначено основні концепції та особливості функціонування систем пошуку роботи. На основі аналізу існуючих рішень сформовано вимоги до майбутньої системи, визначено її ключовий функціонал для різних категорій користувачів, а також обґрунтовано вибір технологічного стеку.

У другому розділі було виконано проектування програмної системи. Розроблено архітектуру проекту, спроектовано структуру бази даних, описано основні сутності та зв'язки між ними. Також розроблено алгоритми роботи ключових модулів системи, зокрема механізмів автентифікації, управління вакансіями та резюме, а також модуль автоматизованого ранжування кандидатів на основі методів машинного навчання.

У третьому розділі було представлено інтерфейс користувацької взаємодії та описано порядок розгортання системи. Реалізовано механізми локального запуску та розгортання в мережі Інтернет із використанням Cloudflare Tunnel. Детально розглянуто структуру інтерфейсу для різних ролей користувачів, що забезпечує зручну та логічну взаємодію з системою. Окрему увагу приділено тестуванню вебпорталу, яке включало функціональне та навантажувальне тестування. Отримані результати підтвердили коректність роботи основних функцій системи, стабільність під час пікового навантаження та відповідність встановленим нефункціональним вимогам.

Таким чином, у межах роботи було реалізовано повноцінний вебпортал для пошуку роботи та підбору персоналу, який є функціонально завершеним,

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						84
Змн.	Арк.	№ докум.	Підпис	Дата		

стабільним у роботі та готовим до подальшого розширення і практичного використання. Результати розробки мають практичну цінність і можуть бути використані для подальшого вдосконалення систем подібного типу.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						85
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ElectroIQ. Recruitment Statistics By Work Culture, AI, Cost Of Hiring And Facts (2026). URL: <https://electroiq.com/stats/recruitment-statistics/> (дата звернення: 10.05.2026).
2. Vettio. Average Time Recruiters Spend Reviewing Resumes: Key Insights & Hiring Data. URL: <https://vettio.com/blog/average-time-recruiter-spends-reviewing-resumes/> (дата звернення: 10.05.2026).
3. Work.ua. URL: <https://www.work.ua/> (дата звернення: 15.05.2026).
4. OLX. OLX Робота. URL: <https://www.olx.ua/uk/rabota/> (дата звернення: 15.05.2026).
5. Jobs.ua. URL: <https://jobs.ua/> (дата звернення: 15.05.2026).
6. NestJS. NestJS Documentation. URL: <https://docs.nestjs.com> (дата звернення: 03.05.2026).
7. Next.js. Next.js Documentation. URL: <https://nextjs.org/docs> (дата звернення: 03.05.2026).
8. Thakkar M. Building React Apps with Server-Side Rendering: Use React, Redux, and Next to Build Full Server-Side Rendering Applications. Berkeley: Apress, 2020. 350 p. ISBN 978-1-4842-5869-9.
9. PostgreSQL Global Development Group. PostgreSQL 18.4, 17.10, 16.14, 15.18, and 14.23 Released. URL: <https://www.postgresql.org/about/news/postgresql-184-1710-1614-1518-and-1423-released-3297/> (дата звернення: 03.05.2026).
10. Object Management Group (OMG). Unified Modeling Language (UML) Specification. URL: <https://www.omg.org/spec/UML/> (дата звернення: 20.05.2026).
11. Prisma. Prisma Documentation. URL: <https://www.prisma.io/docs> (дата звернення: 03.05.2026).
12. Microsoft. Designing and implementing lookup tables and reference data in relational databases. URL: <https://learn.microsoft.com/en-us/sql/relational-databases/tables/> (дата звернення: 20.05.2026).

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		86

13. Redis. Redis Documentation. URL: <https://redis.io/docs/> (дата звернення: 03.05.2026).

14. Visual Paradigm. UML Activity Diagram Tutorial. URL: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-activity-diagram-tutorial> (дата звернення: 20.05.2026).

15. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures. Chapter 5: Representational State Transfer (REST). URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (дата звернення: 15.06.2026).

16. Microsoft. TypeScript Handbook - Enums. URL: <https://www.typescriptlang.org/docs/handbook/enums.html> (дата звернення: 03.05.2026).

17. Prisma. Transactions (Interactive transactions and batch operations). URL: <https://www.prisma.io/docs/orm/prisma-client/queries/transactions> (дата звернення: 06.05.2026).

18. Mozilla Developer Network (MDN). HTTP response status codes. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status> (дата звернення: 10.05.2026).

19. Oracle. Core J2EE Patterns — Data Transfer Object. URL: <https://www.oracle.com/java/technologies/data-transfer-object.html> (дата звернення: 15.06.2026).

20. Wang L., et al. Multilingual E5 Text Embeddings: A Technical Report. arXiv:2402.05672, 2024. URL: <https://arxiv.org/abs/2402.05672> (дата звернення: 19.06.2026).

21. Sentence Transformers. SentenceTransformer.encode() - batch processing. URL: https://sbert.net/docs/package_reference/sentence_transformer/model.html (дата звернення: 19.06.2026).

22. University of Wisconsin–Madison. Complexity Analysis (Big-O Notation and Algorithm Efficiency). URL:

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						87
Змн.	Арк.	№ докум.	Підпис	Дата		

<https://pages.cs.wisc.edu/~vernon/cs367/notes/3.COMPLEXITY.html> (дата звернення: 19.06.2026).

23. Refactoring.Guru. Observer Design Pattern. URL: <https://refactoring.guru/design-patterns/observer> (дата звернення: 19.06.2026).

24. Mozilla Developer Network (MDN). Single-page application (SPA). URL: <https://developer.mozilla.org/en-US/docs/Glossary/SPA> (дата звернення: 19.06.2026).

25. Entri Blog. Client-side vs Server-side Rendering in SEO – Differences, Pros & SEO Impact. URL: <https://entri.app/blog/client-side-vs-server-side-rendering-seo/> (дата звернення: 19.06.2026).

26. Chart.js. Chart.js Documentation. URL: <https://www.chartjs.org/docs/latest/> (дата звернення: 25.05.2026).

27. dotenv. Environment Variables (.env) – dotenv documentation. URL: <https://www.dotenv.org/docs/> (дата звернення: 19.06.2026).

28. Google. OAuth 2.0 for Web Server Applications. URL: <https://developers.google.com/identity/protocols/oauth2/web-server> (дата звернення: 10.05.2026).

29. Google. reCAPTCHA Documentation. URL: <https://developers.google.com/recaptcha/docs/v3> (дата звернення: 10.05.2026).

30. Docker Inc. Docker Documentation. URL: <https://docs.docker.com/> (дата звернення: 20.05.2026).

31. Cloudflare. Cloudflare Tunnel Documentation (formerly Argo Tunnel). URL: <https://developers.cloudflare.com/cloudflare-one/connections/connect-networks/> (дата звернення: 10.06.2026).

32. ISTQB. Foundation Level Syllabus — Software Testing Fundamentals (Test Cases and Functional Testing). URL: <https://www.istqb.org/certifications/certified-tester-foundation-level.html> (дата звернення: 19.06.2026).

33. Apache Software Foundation. Apache JMeter User Manual. URL: <https://jmeter.apache.org/usermanual/> (дата звернення: 10.06.2026).

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						88
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						89
Змн.	Арк.	№ докум.	Підпис	Дата		

Auth module

```

import {forwardRef, Module} from '@nestjs/common';
import { AuthService } from '../auth.service';
import { AuthController } from '../auth.controller';
import { UserModule } from '@user/user.module';
import { EmployerModule } from '@employer/employer.module';
import { JobSeekerModule } from '@job-seeker/job-seeker.module';
import { PrismaModule } from '@prisma/prisma.module';
import { GoogleRecaptchaModule } from '@nestlab/google-recaptcha';
import { ConfigModule, ConfigService } from '@nestjs/config';
import { getRecaptchaConfig } from '@config/recaptcha.config';
import { ProviderModule } from '@auth/provider/provider.module';
import { getProviderConfig } from '@config/providers.config';
import { EmailConfirmationModule } from '@auth/email-confirmation/email-confirmation.module';
import { UserService } from '@user/user.service';
import { MailService } from '@libs/mail/mail.service';

@Module({
  controllers: [AuthController],
  providers: [AuthService, UserService, MailService],
  exports: [AuthService],
  imports: [
    UserModule,
    EmployerModule,
    JobSeekerModule,
    PrismaModule,
    ProviderModule.registerAsync({
      imports: [ConfigModule],
      useFactory: getProviderConfig,
      inject: [ConfigService]
    }),
    GoogleRecaptchaModule.forRootAsync({
      imports: [ConfigModule],
      useFactory: getRecaptchaConfig,
      inject: [ConfigService]
    }),
    forwardRef(() => EmailConfirmationModule),
  ],
})
export class AuthModule {}

```

```

import {
  BadRequestException,
  Body,
  Controller,
  Get,
  HttpStatusCode,
  HttpStatus,
  Param,
  Post,
  Query,
  Req,
  Res,
  UseGuards
} from '@nestjs/common';
import { AuthService } from '../auth.service';
import { RegisterDto } from '@auth/dto/register.dto';
import { Request, Response } from 'express';

```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						90
Змн.	Арк.	№ докум.	Підпис	Дата		

```

import {LoginDto} from "@/auth/dto/login.dto";
import {Recaptcha} from "@nestlab/google-recaptcha";
import {AuthProviderGuard} from "@/auth/guard/provider.guard";
import {ProviderService} from "@/auth/provider/provider.service";
import {ConfigService} from "@nestjs/config";
import {UserRole} from "@prisma/__generated__/enums";
import {Throttle} from "@nestjs/throttler";

@Controller('auth')
export class AuthController {
  constructor(
    private readonly authService: AuthService,
    private readonly providerService: ProviderService,
    private readonly configService: ConfigService
  ) {}

  @Recaptcha()
  @Post('register/jobseeker')
  @HttpCode(HttpStatus.CREATED)
  public async registerJobSeeker(@Req() req: Request, @Body() dto:
RegisterDto) {
    return this.authService.register(req, dto, UserRole.JOB_SEEKER)
  }

  @Recaptcha()
  @Post('register/employer')
  @HttpCode(HttpStatus.CREATED)
  public async registerEmployer(@Req() req: Request, @Body() dto:
RegisterDto) {
    return this.authService.register(req, dto, UserRole.EMPLOYER)
  }
}

```

```

@UseGuards(AuthProviderGuard)
@Get('/oauth/connect/:provider/:role')
public async connect(
  @Param('provider') provider: string,
  @Param('role') role: UserRole
) {
  const rolesMap = {
    'JOB_SEEKER': UserRole.JOB_SEEKER,
    'EMPLOYER': UserRole.EMPLOYER
  } as const

  const userRole = rolesMap[role]
  if (!userRole) {
    throw new BadRequestException('Invalid role')
  }

  const providerInstance = this.providerService.findByService(provider)

  return{
    url: providerInstance.getAuthUrl(userRole)
  }
}

@UseGuards(AuthProviderGuard)
@Get('/oauth/callback/:provider')
public async callback(
  @Req() req: Request,
  @Res({passthrough: true}) res: Response,
  @Query('code') code: string,
  @Param('provider') provider: string,

```

Арк.

ІПЗ.КР.Б-121-26-ПЗ

91

Змн.	Арк.	№ докум.	Підпис	Дата
------	------	----------	--------	------

```

        @Query('state') role: UserRole
    ) {
        if(!code) {
            throw new BadRequestException('Not provided provider');
        }
        if (role === 'EMPLOYER'){
            await this.authService.extractProfileFromCode(req,provider,code,
UserRole.EMPLOYER);
        }
        else if (role === 'JOB_SEEKER'){
            await this.authService.extractProfileFromCode(req,provider,code,
UserRole.JOB_SEEKER);
        }
        else {
            throw new BadRequestException('Invalid role')
        }

        return
res.redirect(`${this.configService.getOrThrow<string>('ALLOWED_ORIGIN')}/profile`);
    }

    @Recaptcha()
    @Post('login')
    @Throttle({ default: { limit: 5, ttl: 60000 } })
    @HttpCode(HttpStatus.OK)
    public async login(@Req() req: Request, @Body() dto: LoginDto) {
        return this.authService.login(req, dto)
    }

    @Post('logout')
    @HttpCode(HttpStatus.OK)
    public async logout(
        @Req() req: Request,
        @Res({ passthrough: true }) res: Response
    ) {
        return this.authService.logout(req, res)
    }
}

import {
    ConflictException,
    Injectable,
    InternalServerErrorException,
    NotFoundException,
    UnauthorizedException
} from '@nestjs/common';

import {RegisterDto} from './dto/register.dto';
import {UserService} from '@user/user.service';
import {EmployerService} from '@employer/employer.service'
import {JobSeekerService} from '@job-seeker/job-seeker.service'

```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		92

```

import {AuthMethod, UserRole} from "@prisma/__generated__/enums";
import {PrismaService} from "@prisma/prisma.service";
import {User} from "@prisma/__generated__/client";
import {Request, Response} from 'express';
import {LoginDto} from "@auth/dto/login.dto";
import {verify} from "argon2";
import {ConfigService} from "@nestjs/config";
import {ProviderService} from "@auth/provider/provider.service";
import {EmailConfirmationService} from "@auth/email-confirmation/email-confirmation.service";

@Injectable()
export class AuthService {

  public constructor(
    private readonly userService: UserService,
    private readonly employerService: EmployerService,
    private readonly jobSeekerService: JobSeekerService,
    private readonly prismaService: PrismaService,
    private readonly configService: ConfigService,
    private readonly providerService: ProviderService,
    private readonly emailConfirmationService: EmailConfirmationService,
  ) {

  }

  public async register(req: Request, dto: RegisterDto, role: UserRole) {
    const isExists = await this.userService.findByEmail(dto.email)

    if (isExists) {
      throw new ConflictException('Така пошта вже зареєстрована.')
    }

    const newUser = await this.prismaService.$transaction(async prisma =>
    {
      const newUser = await this.userService.create(
        prisma,
        dto.email,
        dto.password,
        role,
        false,
        AuthMethod.CREDENTIALS
      )

      if (role === UserRole.EMPLOYER) {
        await this.employerService.create(prisma, newUser.id)
      }

      if (role === UserRole.JOB_SEEKER) {
        await this.jobSeekerService.create(prisma, newUser.id)
      }

      return newUser
    })
    await this.emailConfirmationService.sendVerificationToken(newUser)

    return {
      message: ' Успішно зареєстровано! Очікую підтвердження електронною поштою'
    }
  }

  public async login(req: Request, dto: LoginDto) {
    const user = await this.userService.findByEmail(dto.email);
  }

```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						93
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        if (!user || !user.passwordHash) {
            throw new NotFoundException("Помилка. Користувача не знайдено або не
вірний пароль");
        }
        const isValidPassword = await verify(user.passwordHash, dto.password);

        if (!isValidPassword) {
            throw new UnauthorizedException("Помилка. Користувача не знайдено або
не вірний пароль");
        }

        if (!user.isVerifiedEmail) {
            await this.emailConfirmationService.sendVerificationToken(user)
            throw new UnauthorizedException("Обліковий запис не підтверджено.
Перевірте вашу поштову скриньку.");
        }

        return this.saveSession(req, user)
    }

    public async extractProfileFromCode(
        req: Request,
        provider: string,
        code: string,
        role: UserRole
    ) {
        const providerInstance = this.providerService.findByService(provider)
        const profile = await providerInstance.findUserByCode(code)

        const account = await this.prismaService.account.findFirst({
            where: {
                id: profile.id,
                provider: profile.provider
            }
        })

        let user = account?.userId ? await
this.userService.findById(account.userId) : await
this.userService.findByEmail(profile.email)

        if (user) {
            if (account) {
                await this.prismaService.account.update({
                    where: {
                        id: account.id
                    },
                    data: {
                        accessToken: profile.access_token,
                        expiresAt: profile.expires_at,

                        ...(profile.refresh_token && {
                            refreshToken: profile.refresh_token
                        })
                    }
                })
            } else {
                await this.prismaService.account.create({
                    data: {
                        id: profile.id,
                        userId: user.id,

```

```

        type: 'oauth',
        provider: profile.provider,
        accessToken: profile.access_token,
        refreshToken: profile.refresh_token,
        expiresAt: profile.expires_at
    }
    })
}

return this.saveSession(req, user)
}

user = await this.prismaService.$transaction(async prisma => {
    const newUser = await this.userService.create(
        prisma,
        profile.email,
        '',
        role,
        true,
        AuthMethod[profile.provider.toUpperCase()]
    )
    if (role === UserRole.EMPLOYER) {
        await this.employerService.create(prisma, newUser.id)
    }
    if (role === UserRole.JOB_SEEKER) {
        await this.jobSeekerService.create(prisma, newUser.id)
    }

    await prisma.account.create({
        data: {
            userId: newUser.id,
            type: 'oauth',
            provider: profile.provider,
            accessToken: profile.access_token,
            refreshToken: profile.refresh_token,
            expiresAt: profile.expires_at
        }
    })
    return newUser
})

return this.saveSession(req, user)
}

public async logout(req: Request, res: Response): Promise<void> {
    return new Promise((resolve, reject) => {
        req.session.destroy(err => {
            if (err) {
                return reject(
                    new InternalServerErrorException(err.message)
                );
            }
        })
    })

    res.clearCookie(this.configService.getOrThrow<string>('SESSION_NAME'))
    resolve()
}

public async saveSession(req: Request, user: User) {
    return new Promise((resolve, reject) => {
        req.session.userId = user.id;

        req.session.save(err => {
            if (err) {

```

```

        return reject(
            new InternalServerErrorException(err.message)
        );
    }
    resolve({
        user
    })
})
})
}
}

```

user module

```

import { Module } from '@nestjs/common';
import { UserService } from '../user.service';
import { UserController } from '../user.controller';

@Module({
  controllers: [UserController],
  providers: [UserService],
  exports: [UserService]
})
export class UserModule {}

```

```

import { Controller, Get, HttpStatusCode, HttpStatus } from '@nestjs/common';
import { UserService } from '../user.service';
import { Authorization } from '@auth/decorators/auth.decorator';
import { Authorized } from '@auth/decorators/authorized.decorator';
import { UserRole } from '@prisma/___generated___/enums';

@Controller('users')
export class UserController {
  constructor(private readonly userService: UserService) {}

  @Authorization()
  @HttpStatusCode(HttpStatus.OK)
  @Get('profile')
  public async findProfile(@Authorized('id') userId: string) {
    return this.userService.findById(userId);
  }

  @Authorization()
  @HttpStatusCode(HttpStatus.OK)
  @Get('me')
  public async findProfileSecure(@Authorized('id') userId: string) {
    return this.userService.findByIdSecure(userId);
  }
}

```

```

import { Injectable, NotFoundException } from '@nestjs/common';
import { PrismaService } from '@prisma/prisma.service';
import { AuthMethod, UserRole } from '@prisma/___generated___/enums';

import { hash } from 'argon2'

```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						96
Змн.	Арк.	№ докум.	Підпис	Дата		

```

import {PrismaTransaction} from "../../prisma/types";

@Injectable()
export class UserService {
  public constructor(private readonly prismaService: PrismaService) {}

  public async findById(id: string) {
    const user = await this.prismaService.user.findUnique({
      where: { id },
    })

    if (!user) {
      throw new NotFoundException(`Користувача з ідентифікатором ${id}
не знайдено`)
    }

    let profile = null

    if (user.role === "JOB SEEKER") {
      profile = await this.prismaService.jobSeeker.findUnique({
        where: { userId: id },
      })
    }

    if (user.role === "EMPLOYER") {
      profile = await this.prismaService.employer.findUnique({
        where: { userId: id },
      })
    }

    return {
      ...user,
      profile,
    }
  }

  public async findByIdSecure(id: string) {
    const user = await this.prismaService.user.findUnique({
      where: {
        id
      },
      include: {
        accounts: true
      }
    })

    if (!user) {
      throw new NotFoundException(`Користувача з ідентифікатором ${id}
не знайдено`)
    }

    return {
      email: user.email,
      role: user.role,
    };
  }

  public async findByEmail(email: string) {
    const user = await this.prismaService.user.findUnique({
      where: {
        email
      },
      include: {

```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						97
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        accounts: true
    }

    })

    return user;
}

public async create(
    prisma: PrismaTransaction,
    email: string,
    password: string,
    role: UserRole,
    isVerifiedEmail: boolean,
    method: AuthMethod
) {

    return prisma.user.create({
        data: {
            email,
            passwordHash: password ? await hash(password) : '',
            role,
            isVerifiedEmail,
            method
        },
        include: {
            accounts: true
        }
    })
}
}

```

resume module

```

import {Body, Controller, Delete, Get, HttpStatusCode, HttpStatus, Param, Post}
from '@nestjs/common';
import { ResumeService } from '../resume.service';
import { Authorization } from '@auth/decorators/auth.decorator';
import { Authorized } from '@auth/decorators/authorized.decorator';
import { User, UserRole } from '@prisma/__generated__/client';
import { CreateResumeDto } from '@resume/dto/create-resume.dto';
import { ApiOkResponse } from '@nestjs/swagger';
import { ResumePreviewDto } from '@resume/dto/resume-preview.dto';
import { ResumeDetailDto } from '@resume/dto/resume-detail.dto';

@Controller('resume')
export class ResumeController {
    constructor(private readonly resumeService: ResumeService) {}

    @Authorization(UserRole.JOB_SEEKER)
    @Post('create')
    @HttpCode(HttpStatus.OK)
    public async create(
        @Authorized() user: User,
        @Body() dto: CreateResumeDto
    ) {
        return this.resumeService.create(user.id, dto)
    }

    @Authorization(UserRole.JOB_SEEKER)

```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						98
Змн.	Арк.	№ докум.	Підпис	Дата		

```

@Delete('/:id')
@HttpCode(HttpStatus.OK)
public async delete(
    @Authorized() user: User,
    @Param('id') resumeId: string
) {
    return this.resumeService.delete(user.id, resumeId)
}

@Authorization(UserRole.JOB_SEEKER)
@Get('my')
@ApiOkResponse({
    type: [ResumePreviewDto]
})
@HttpCode(HttpStatus.OK)
public async findMy(@Authorized() user: User) {
    return this.resumeService.findMyList(user.id)
}

@Authorization(UserRole.JOB_SEEKER)
@Get('my/:id')
@ApiOkResponse({
    type: [ResumeDetailDto]
})
@HttpCode(HttpStatus.OK)
public async findMyById(
    @Authorized() user: User,
    @Param('id') resumeId: string
) {
    return this.resumeService.findMyById(user.id, resumeId)
}
}

```

```

import { Module } from '@nestjs/common';
import { ResumeService } from '../resume.service';
import { ResumeController } from '../resume.controller';
import { PrismaModule } from '@prisma/prisma.module';
import { AuthModule } from '@auth/auth.module';
import { UserModule } from '@user/user.module';

@Module({
    imports: [PrismaModule, AuthModule, UserModule],
    controllers: [ResumeController],
    providers: [ResumeService],
    exports: [ResumeService]
})
export class ResumeModule {}

```

```

import {
    BadRequestException, ForbiddenException,
    Injectable,
    NotFoundException
} from '@nestjs/common'

```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		99

```

import { PrismaService } from '@prisma/prisma.service'
import { CreateResumeDto } from '@resume/dto/create-resume.dto'
import { ResumePreviewDto } from '@resume/dto/resume-preview.dto';
import { ResumeDetailDto } from '@resume/dto/resume-detail.dto';

@Injectable()
export class ResumeService {
  public constructor(private readonly prismaService: PrismaService) {}

  public async create(userId: string, dto: CreateResumeDto) {
    const jobSeeker = await this.findJobSeekerByUserId(userId)

    await this.validateEducations(dto)
    await this.validateLanguages(dto)

    return this.prismaService.resume.create({
      data: {
        jobSeekerId: jobSeeker.id,
        title: dto.title,
        skills: dto.skills ?? [],
        additionalInfo: dto.additionalInfo,

        resumeEducations: {
          create:
            dto.educations?.map(education => ({
              educationLevelId: education.educationLevelId,
              institutionName: education.institutionName,
              faculty: education.faculty,
              city: education.city,
              startYear: education.startYear,
              endYear: education.endYear,
              description: education.description
            })) ?? []
        },

        resumeLanguages: {
          create:
            dto.languages?.map(language => ({
              languageId: language.languageId,
              languageLevelId: language.languageLevelId
            })) ?? []
        }
      },
      include: {
        resumeEducations: true,
        resumeLanguages: true
      }
    })
  }

  public async findMyList(userId: string): Promise<ResumePreviewDto[]> {
    const jobSeeker = await this.findJobSeekerByUserId(userId)

    const resumes = await this.prismaService.resume.findMany({
      where: {
        jobSeekerId: jobSeeker.id
      },
      select: {
        id: true,
        title: true,
        createdAt: true,
        updatedAt: true
      },
      orderBy: {

```

```

        createdAt: 'desc'
    }
})

return resumes
}

public async findMyById(userId: string, resumeId: string):
Promise<ResumeDetailDto> {
    const jobSeeker = await this.findJobSeekerByUserId(userId)

    const resume = await this.prismaService.resume.findFirst({
        where: {
            id: resumeId,
            jobSeekerId: jobSeeker.id
        },
        include: this.getResumeInclude()
    })

    if (!resume) {
        throw new NotFoundException('Резюме не знайдено')
    }

    return {
        id: resume.id,
        title: resume.title,
        skills: resume.skills,
        additionalInfo: resume.additionalInfo,
        createdAt: resume.createdAt,
        updatedAt: resume.updatedAt,

        educations: resume.resumeEducations.map(education => ({
            id: education.id,
            level: education.educationLevel.name,
            institutionName: education.institutionName,
            faculty: education.faculty,
            locationId: education.locationId,
            startYear: education.startYear,
            endYear: education.endYear,
            description: education.description
        })),

        languages: resume.resumeLanguages.map(language => ({
            name: language.language.name,
            code: language.language.code,
            level: language.languageLevel.name
        })))
    }
}

public async delete(userId: string, resumeId: string) {
    const jobSeeker = await this.findJobSeekerByUserId(userId);

    const resume = await this.prismaService.resume.findFirst({
        where: {
            id: resumeId,
            jobSeekerId: jobSeeker.id,
        },
    });

    if (!resume) {
        throw new NotFoundException("Резюме не знайдено");
    }
}

```

```

    }

    const application = await this.prismaService.application.findFirst({
      where: {
        resumeId: resumeId,
      },
      select: {
        id: true,
      },
    });

    if (application) {
      throw new ForbiddenException(
        "Ви вже відгукнулись цим резюме. Видалення неможливе."
      );
    }

    return this.prismaService.resume.delete({
      where: {
        id: resumeId,
      },
    });
  }

  private getResumeInclude() {
    return {
      resumeEducations: {
        include: {
          educationLevel: {
            select: {
              id: true,
              name: true,
              sortOrder: true
            },
          },
        },
      },
      orderBy: {
        startYear: 'desc' as const
      },
    },
    resumeLanguages: {
      include: {
        language: {
          select: {
            id: true,
            name: true,
            code: true
          },
        },
        languageLevel: {
          select: {
            id: true,
            name: true,
            sortOrder: true
          },
        },
      },
    },
  }
}

```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						102
Змн.	Арк.	№ докум.	Підпис	Дата		

```

private async findJobSeekerByUserId(userId: string) {
  const jobSeeker = await this.prismaService.jobSeeker.findUnique({
    where: {
      userId
    }
  })

  if (!jobSeeker) {
    throw new NotFoundException('профіль шукача роботи не знайдено')
  }

  return jobSeeker
}

private async validateEducations(dto: CreateResumeDto): Promise<void> {
  const educations = dto.educations ?? []

  if (!educations.length) {
    return
  }

  const educationLevelIds = educations.map(
    education => education.educationLevelId
  )

  const existingEducationLevels =
    await this.prismaService.educationLevel.findMany({
      where: {
        id: {
          in: educationLevelIds
        }
      },
      select: {
        id: true
      }
    })

  const existingIds = existingEducationLevels.map(item => item.id)

  const invalidIds = educationLevelIds.filter(
    id => !existingIds.includes(id)
  )

  if (invalidIds.length) {
    throw new BadRequestException({
      message: 'Недійсний ідентифікатор рівня освіти',
      invalidEducationLevelIds: [...new Set(invalidIds)]
    })
  }
}

private async validateLanguages(dto: CreateResumeDto): Promise<void> {
  const languages = dto.languages ?? []

  if (!languages.length) {
    return
  }

  const languageIds = languages.map(language => language.languageId)

  const languageLevelIds = languages.map(
    language => language.languageLevelId
  )
}

```

```

        this.validateUniqueLanguages(languageIds)

        await this.validateLanguageIds(languageIds)
        await this.validateLanguageLevelIds(languageLevelIds)
    }

    private validateUniqueLanguages(languageIds: number[]): void {
        const uniqueIds = new Set(languageIds)

        if (uniqueIds.size !== languageIds.length) {
            const duplicatedIds = languageIds.filter(
                (id, index) => languageIds.indexOf(id) !== index
            )

            throw new BadRequestException({
                message: 'Дублікації мов в одному резюме не допускаються',
                duplicatedLanguageIds: [...new Set(duplicatedIds)]
            })
        }
    }

    private async validateLanguageIds(languageIds: number[]): Promise<void> {
        const existingLanguages = await
this.prismaService.language.findMany({
            where: {
                id: {
                    in: languageIds
                }
            },
            select: {
                id: true
            }
        })

        const existingIds = existingLanguages.map(item => item.id)

        const invalidIds = languageIds.filter(id =>
!existingIds.includes(id))

        if (invalidIds.length) {
            throw new BadRequestException({
                message: 'Недійсний ідентифікатор мови',
                invalidLanguageIds: [...new Set(invalidIds)]
            })
        }
    }

    private async validateLanguageLevelIds(languageLevelIds: number[]):
Promise<void> {
        const existingLanguageLevels =
            await this.prismaService.languageLevel.findMany({
                where: {
                    id: {
                        in: languageLevelIds
                    }
                },
                select: {
                    id: true
                }
            })

        const existingIds = existingLanguageLevels.map(item => item.id)

```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		104

```

    const invalidIds = languageLevelIds.filter(
      id => !existingIds.includes(id)
    )

    if (invalidIds.length) {
      throw new BadRequestException({
        message: 'Недійсний ідентифікатор рівня мови',
        invalidLanguageLevelIds: [...new Set(invalidIds)]
      })
    }
  }
}

```

docker-compose

```

version: "3.9"

services:
  # ===== POSTGRES =====
  postgres:
    image: postgres:16
    container_name: postgres_v2
    restart: always
    environment:
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB}
    ports:
      - "5433:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
      - app_network

  # ===== REDIS =====
  redis:
    image: redis:5.0
    container_name: redis
    restart: always
    command: redis-server --requirepass ${REDIS_PASSWORD}
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data
    networks:
      - app_network

  # ===== BACKEND =====
  backend:
    build:
      context: ./backend
    args:
      POSTGRES_URL: ${POSTGRES_URL}
    container_name: backend
    restart: always
    ports:
      - "4000:4000"
    environment:
      GOOGLE_CLIENT_ID: ${GOOGLE_CLIENT_ID}
      GOOGLE_CLIENT_SECRET: ${GOOGLE_CLIENT_SECRET}
      APPLICATION_URL: https://api.searchwork.pp.ua
      PORT: 4000
    env_file:

```

```

- ./env
depends_on:
- postgres
- redis
- ml
networks:
- app_network

# ===== ML SERVICE =====
ml:
  build: ./ml_service
  container_name: ml_service
  restart: always
  ports:
  - "8000:8000"
  environment:
    PYTHONUNBUFFERED: "1"
  networks:
  - app_network

# ===== FRONTEND =====
frontend:
  build:
    context: ./frontend
    args:
      NEXT_PUBLIC_API_URL: https://api.searchwork.pp.ua
      NEXT_PUBLIC_RECAPTCHA_SITE_KEY: ${NEXT_PUBLIC_RECAPTCHA_SITE_KEY}
  container_name: frontend
  restart: always
  ports:
  - "3000:3000"
  environment:
    NEXT_PUBLIC_API_URL: https://api.searchwork.pp.ua
    NEXT_PUBLIC_RECAPTCHA_SITE_KEY: ${NEXT_PUBLIC_RECAPTCHA_SITE_KEY}
  depends_on:
  - backend
  networks:
  - app_network

# ===== VOLUMES =====
volumes:
  postgres_data:
  redis_data:

# ===== NETWORK =====
networks:
  app_network:
    driver: bridge

```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						106
Змн.	Арк.	№ докум.	Підпис	Дата		