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:

«Вебплатформа для пошуку роботи або стажування»

Виконав студент групи ВТ-22-2
ДРОЖАК Вікторія Тарасівна

Керівник роботи:
ЄФРЕМОВ Юрій Миколайович

Рецензент:
МОРОЗОВ Андрій Васильович

Житомир – 2026

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: **ДРОЖАК Вікторія Тарасівна**
Керівник роботи: **ЄФРЕМОВ Юрій Миколайович**
Тема роботи: **«Вебплатформа для пошуку роботи або стажування»**, затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., № 92/с**
Вихідні дані для роботи: об’єкт дослідження процес пошуку роботи та стажування, а також взаємодія між кандидатами та роботодавцями за допомогою веборієнтованих інформаційних систем.Предмет дослідження методи, засоби та технології розроблення вебплатформи для пошуку роботи та стажування, що забезпечує пошук вакансій, створення резюме, подання заявок і підбір кандидатів.
Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	**Консультант**	Завдання видав	Завдання прийняв
1	Єфремов Ю. М.	01.03.2026 р.	01.03.2026 р.
2	Єфремов Ю. М.	01.03.2026 р.	01.03.2026 р.
3	Єфремов Ю. М.	01.03.2026 р.	01.03.2026 р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра присвячена розробці вебплатформи для пошуку роботи або стажування. Робота складається з пояснювальної записки та програмного продукту. Пояснювальна записка містить 153 сторінки, з них 88 сторінок основного тексту, 36 ілюстрацій та 23 таблиці, 1 додаток та 30 використаних джерел

Метою роботи є створення вебплатформи, що автоматизує пошук вакансій і стажувань, взаємодію кандидатів із роботодавцями, створення резюме, подання заявок, перегляд кандидатів компаніями та адміністрування вмісту системи. Платформа забезпечує зручний пошук актуальних пропозицій, збереження вакансій і відстеження статусу заявок.

Для реалізації програмного продукту використано React і Vite для клієнтської частини, Node.js та Express.js для серверної логіки, PostgreSQL і Supabase для зберігання й обробки даних. Також передбачено автентифікацію користувачів, авторизацію через Google та інтеграцію з Google Calendar і Google Meet для організації дистанційних співбесід.

Результатом роботи є вебплатформа, яка дозволяє кандидатам шукати вакансії або стажування, створювати резюме й подавати заявки, компаніям - публікувати вакансії та переглядати кандидатів, а адміністратору - контролювати роботу системи та керувати її основними об’єктами.

КЛЮЧОВІ СЛОВА: ВЕБПЛАТФОРМА, ПОШУК РОБОТИ, СТАЖУВАННЯ, ВАКАНСІЯ, КАНДИДАТ, РОБОТОДАВЕЦЬ, РЕЗЮМЕ, ЗАЯВКА, REACT, POSTGRESQL, NODE.JS, АДМІНІСТРАТИВНА ПАНЕЛЬ.

ABSTRACT

The bachelor’s qualification thesis is devoted to the development of a web platform for job and internship search. The work consists of an explanatory note and a software product. The explanatory note contains 153 pages, including 88 pages of main text, 36 illustrations, 23 tables, 1 appendix, and 30 references.

The aim of the work is to create a web platform that automates the search for vacancies and internships, interaction between candidates and employers, resume creation, application submission, candidate review by companies, and administration of system content. The platform provides convenient search for current offers, saving vacancies, and tracking application statuses.

React and Vite were used for the client side of the software product, Node.js and Express.js for server-side logic, and PostgreSQL and Supabase for data storage and processing. User authentication, Google authorization, and integration with Google Calendar and Google Meet for organizing remote interviews are also provided.

The result of the work is a web platform that allows candidates to search for vacancies or internships, create resumes, and submit applications; companies to publish vacancies and review candidates; and the administrator to control the operation of the system and manage its main objects.

KEYWORDS: WEB PLATFORM, JOB SEARCH, INTERNSHIP, VACANCY, CANDIDATE, EMPLOYER, RESUME, APPLICATION, REACT, POSTGRESQL, NODE.JS, ADMINISTRATIVE PANEL.

Зміст

РЕФЕРАТ 3
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 6
ВСТУП 8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ РОЗРОБКИ ВЕБПЛАТФОРМ ДЛЯ ПОШУКУ РОБОТИ АБО СТАЖУВАННЯ 10
1.1. Постановка задачі 10
1.2. Аналіз аналогів вебплатформи для пошуку роботи або стажування 13
1.3. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення 18
Висновки до першого розділу 21
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБПЛАТФОРМИ ДЛЯ ПОШУКУ РОБОТИ ТА СТАЖУВАННЯ 23
2.1. Визначення варіантів використання та об’єктно-орієнтованої структури системи 23
2.2. Розробка бази даних системи 32
2.3. Проєктування та реалізація алгоритмів роботи системи 45
2.4. Реалізація функціоналу вебплатформи для пошуку роботи або стажування 53
Висновки до другого розділу 62
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБПЛАТФОРМИ ДЛЯ ПОШУКУ РОБОТИ ТА СТАЖУВАННЯ 64
3.1. Порядок встановлення та налаштування системи 64
3.2. Структура інтерфейсу та порядок взаємодії з веб-сервісом 67
3.3. Тестування функціональних можливостей вебплатформи 82
Висновки до третього розділу 86
ВИСНОВКИ 88
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 89
ДОДАТКИ 92

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API - прикладний програмний інтерфейс, що забезпечує взаємодію між клієнтською та серверною частинами системи.

БД - база даних, призначена для зберігання інформації про користувачів, вакансії, резюме, заявки та компанії.

Вебплатформа - програмна система, що працює через веббраузер і забезпечує пошук роботи або стажування.

Frontend - клієнтська частина вебплатформи, яка відповідає за інтерфейс користувача.

Backend - серверна частина вебплатформи, яка обробляє запити, виконує бізнес-логіку та взаємодіє з базою даних.

JSON - формат обміну даними між клієнтською та серверною частинами системи.

REST API - архітектурний підхід до організації взаємодії між клієнтом і сервером за допомогою HTTP-запитів.

HTTP - протокол передавання даних у мережі Інтернет.

HTTPS - захищена версія протоколу HTTP.

OAuth 2.0 - протокол авторизації, який використовується для входу через зовнішні сервіси, зокрема Google.

JWT - токен доступу, який використовується для підтвердження авторизованого користувача.

SQL - мова структурованих запитів для роботи з реляційними базами даних.

PostgreSQL - система керування реляційними базами даних, що використовується для зберігання даних вебплатформи.

Supabase - сервіс, що використовується для автентифікації користувачів, роботи з базою даних і зберігання файлів.

React - JavaScript-бібліотека для створення інтерфейсу користувача.

Vite - інструмент для швидкого запуску та збирання клієнтської частини вебзастосунку.

Node.js - середовище виконання JavaScript на сервері.

Express.js - фреймворк для створення серверної частини та API.

Google Calendar API - сервіс для створення та керування подіями календаря.

Google Meet - сервіс для проведення онлайн-співбесід.

Data.gov.ua - український портал відкритих державних даних, що використовується для перевірки інформації про компанії.

ЄДРПОУ - єдиний державний реєстр підприємств та організацій України.

UI - інтерфейс користувача.

UX - досвід користувача під час роботи з вебплатформою.

ВСТУП

У сучасному світі інтернет і цифрові технології відіграють важливу роль у житті кожної людини. Це особливо помітно на ринку праці. Сьогодні пошук роботи або стажування здебільшого відбувається онлайн. Для цього використовують різноманітні вебплатформи. Вони дозволяють швидко знаходити актуальні вакансії, подавати резюме та спілкуватися з роботодавцями. Особисті зустрічі не потрібні на початкових етапах.

З розвитком технологій зростають і вимоги користувачів до таких сервісів. Люди очікують, що платформа буде простою, зрозумілою, швидкою та зручною. Роботодавці зацікавлені у швидкому пошуку кваліфікованих кандидатів та ефективному управлінні вакансіями. Незважаючи на різноманіття рішень, багато платформ мають недоліки. Це складний інтерфейс, незручний пошук, відсутність персоналізованих рекомендацій або перевантаженість зайвими функціями.

Особливо актуальною ця проблема є для студентів та молодих спеціалістів, які лише починають кар'єру. Їм важливо швидко знайти перше місце роботи або стажування, здобути досвід і зрозуміти вимоги ринку праці. Через відсутність досвіду їм складніше орієнтуватися серед безлічі вакансій. Тому потрібна вебплатформа, яка буде простою, зрозумілою та орієнтованою на їхні потреби.

Актуальність теми кваліфікаційної роботи полягає у необхідності вдосконалення існуючих підходів до пошуку роботи та стажування шляхом розробки сучасної вебплатформи. Така платформа повинна забезпечувати зручний пошук вакансій, можливість швидкого створення профілю, подання заявок, а також ефективну взаємодію між роботодавцями та кандидатами.

Метою кваліфікаційної роботи є створення вебплатформи для пошуку роботи або стажування. Вона має бути простою, функціональною та ефективною. Користувачі зможуть швидко знаходити вакансії. Роботодавці легко публікуватимуть оголошення та обиратимуть кандидатів.

Для досягнення мети потрібно виконати такі завдання. Провести аналіз вебплатформ для пошуку роботи та стажувань. Визначити переваги та недоліки наявних рішень. Сформулювати вимоги до нової системи. Розробити структуру і

логіку платформи. Створити користувацький інтерфейс. Реалізувати ключовий функціонал: реєстрацію, авторизацію, профілі, пошук і фільтрацію вакансій, подання заявок. Провести тестування та оцінити ефективність системи.

Об'єкт дослідження - процес пошуку роботи та стажування в онлайн-середовищі.

Предмет дослідження - методи та інструменти розробки вебплатформ, що забезпечують ефективну взаємодію між шукачами та роботодавцями.

Методи дослідження у процесі виконання роботи використано методи аналізу предметної області, порівняльного аналізу існуючих вебплатформ для пошуку роботи та стажування, об'єктно-орієнтованого проєктування, моделювання структури бази даних, проєктування користувацького інтерфейсу та функціонального тестування програмного продукту.

Практичне значення роботи полягає у створенні вебплатформи. Вона може стати основою для впровадження сервісу. Такий сервіс покращить пошук роботи. Він зробить процес зручнішим, швидшим і доступнішим для багатьох користувачів.

Апробація результатів роботи. За результатами досліджень у випускній кваліфікаційній роботі було опубліковано тези:

Дрожак В. Т., Єфремов Ю. М. Вебплатформа для пошуку та організації студентських стажувань. Комп'ютерні технології: інновації, проблеми, рішення : тези доповідей VIII Всеукр. наук.-техн. конф. (м. Житомир, 2–3 груд. 2025 р)

Дрожак В. Т., Концидайло А. М. Особливості тестування вебплатформи для пошуку роботи або стажування. Інформаційно-комп'ютерні технології : тези доповідей XVI Міжнар. наук.-техн. конф. (м. Житомир, 2–3 квіт. 2026 р)

Структура та обсяг роботи. Випускна кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел із 30 найменувань, а також 1 додаток. Робота містить 23 таблиці та 36 ілюстрацій. Повний обсяг роботи становить 153 сторінки, з них 88 сторінок основного тексту

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ РОЗРОБКИ ВЕБПЛАТФОРМ ДЛЯ ПОШУКУ РОБОТИ АБО СТАЖУВАННЯ

1.1. Постановка задачі

Пошук першої роботи або стажування є важливим етапом професійного становлення студентів, випускників і молодих спеціалістів. Саме стажування дає змогу отримати практичні навички, закріпити набуті знання, краще зрозуміти особливості майбутньої професії та підготуватися до подальшого працевлаштування [1]. Перехід від навчання до професійної діяльності також є важливим етапом для молоді, тому використання сучасних вебплатформ для пошуку вакансій і стажувань має велике значення [2].

Інформація про вакансії та стажування часто розміщується на різних вебресурсах: офіційних сайтах компаній, спеціалізованих платформах, у соціальних мережах та інших онлайн-джерелах. Унаслідок цього кандидатам доводиться витрачати значний час на пошук, аналіз і порівняння пропозицій. Для підвищення ефективності цього процесу доцільним є використання єдиної вебплатформи, яка забезпечуватиме централізований доступ до вакансій, створення та завантаження резюме, подання заявок і відстеження їхнього статусу.

Сучасні цифрові платформи допомагають зробити взаємодію між кандидатами та роботодавцями простішою. Вони дають змогу швидше знаходити вакансії, подавати заявки та організовувати процес відбору кандидатів [3]. Такі платформи стали важливою частиною онлайн-пошуку роботи, оскільки поєднують кандидатів і роботодавців в одному цифровому середовищі [4].

У розроблюваній системі передбачено три основні ролі користувачів: кандидат, компанія та адміністратор. Кожна роль має свої можливості та обмеження. Це потрібно для того, щоб кожен користувач працював тільки з тими функціями, які йому потрібні.

Кандидат використовує платформу для пошуку роботи або стажування. Для нього потрібно реалізувати реєстрацію, вхід у систему, створення особистого профілю, додавання інформації про себе, завантаження резюме, перегляд

вакансій, пошук і фільтрацію пропозицій, подання заявки та перегляд статусу поданих заявок.

Компанія використовує платформу для розміщення вакансій і роботи з кандидатами. Для неї потрібно реалізувати створення профілю, додавання нових вакансій, редагування або видалення вже створених вакансій, перегляд заявок кандидатів, перегляд профілю кандидата та зміну статусу заявки. Це допоможе компаніям зручніше організовувати пошук працівників або стажерів.

Адміністратор відповідає за контроль роботи системи. Для нього потрібно створити адміністративну панель, у якій можна переглядати користувачів, перевіряти компанії, контролювати вакансії та заявки, а також додавати й редагувати інформаційні статті. Адміністратор потрібен для того, щоб підтримувати порядок у системі та перевіряти достовірність інформації про компанії.

Наступним етапом є проєктування загальної структури системи. Потрібно визначити, з яких сторінок і модулів складатиметься вебплатформа. До основних сторінок можна віднести сторінку реєстрації, сторінку входу, головну сторінку, сторінку вакансій, сторінку детального перегляду вакансії, особистий кабінет кандидата, кабінет компанії та адміністративну панель.

Окрему увагу потрібно приділити базі даних. У ній має зберігатися вся основна інформація, необхідна для роботи системи: дані користувачів, профілі кандидатів, профілі компаній, вакансії, заявки, резюме та статті. Також потрібно визначити зв’язки між цими даними. Наприклад, одна компанія може створювати багато вакансій, одна вакансія може мати багато заявок, а один кандидат може подавати заявки на різні вакансії.

Важливим завданням є реалізація реєстрації та авторизації користувачів. Користувач повинен мати можливість створити обліковий запис, увійти в систему та отримати доступ до функцій відповідно до своєї ролі. Наприклад, кандидат після входу повинен бачити можливості для пошуку вакансій і подання заявок, компанія функції для створення вакансій і перегляду кандидатів, а адміністратор інструменти для контролю роботи системи.

Для кандидата потрібно створити особистий кабінет. У ньому він зможе заповнювати інформацію про себе, вказувати освіту, навички, досвід, контактні дані та завантажувати резюме. Також у кабінеті кандидат повинен бачити власні заявки та їхній поточний статус.

Крім цього, потрібно реалізувати сторінку вакансій. На ній кандидат повинен бачити список доступних вакансій або стажувань. Для зручності варто додати пошук і фільтрацію, щоб користувач міг швидше знайти потрібну пропозицію. Наприклад, вакансії можна шукати за назвою, напрямом, форматом роботи, типом зайнятості або місцем розташування.

Також потрібно створити сторінку детального перегляду вакансії. На ній кандидат повинен бачити назву посади, опис вакансії, вимоги, обов'язки, формат роботи, місце роботи та інформацію про компанію. Після перегляду цієї інформації кандидат повинен мати можливість подати заявку.

Окремим завданням є реалізація подання заявки. Коли кандидат натискає кнопку подання заявки, система повинна створити новий запис у базі даних. У заявці має зберігатися інформація про кандидата, вакансію, резюме, дату подання та статус заявки. Початковий статус заявки може бути «подано». Після цього кандидат зможе переглядати цю заявку у своєму кабінеті.

Для компанії потрібно створити кабінет роботодавця. У ньому компанія зможе заповнювати інформацію про себе, додавати нові вакансії, редагувати або видаляти вже створені пропозиції. Під час створення вакансії компанія повинна вказати назву, опис, вимоги до кандидата, обов'язки, формат роботи, тип зайнятості та місце роботи.

Також компанія повинна мати можливість переглядати заявки кандидатів. Після того як кандидат подає заявку, компанія бачить її у своєму кабінеті. Вона може переглянути інформацію про кандидата, його резюме та прийняти рішення щодо заявки.

Потрібно реалізувати й зміну статусу заявки. Після перегляду кандидата компанія може змінити статус заявки, наприклад на «розглядається», «прийнято»

або «відхилено». Після цього кандидат повинен бачити оновлену інформацію у своєму кабінеті.

Для адміністратора потрібно створити адміністративну частину системи. Адміністратор повинен мати можливість переглядати основні дані платформи, перевіряти компанії, контролювати вакансії та заявки. Особливо важливою є перевірка компаній, оскільки це допомагає підвищити довіру до платформи та зменшити кількість недостовірної інформації.

Також адміністратор повинен мати можливість додавати інформаційні статті. Такі матеріали можуть бути корисними для кандидатів. Наприклад, у них можна розміщувати поради щодо створення резюме, підготовки до співбесіди, пошуку першої роботи або проходження стажування.

Під час розробки потрібно передбачити перевірку введених даних. Якщо користувач залишив обов’язкове поле порожнім або ввів неправильні дані, система повинна показати зрозуміле повідомлення про помилку. Це допоможе уникнути неправильного збереження інформації та зробить роботу з платформою зручнішою.

Також потрібно реалізувати розмежування прав доступу. Кандидат не повинен мати можливості створювати вакансії або керувати компаніями. Компанія не повинна мати доступу до адміністративних функцій. Адміністратор повинен мати доступ до службових функцій тільки після входу в систему. Такий підхід потрібен для безпеки та правильної роботи платформи.

У результаті виконання роботи планується створити вебплатформу для пошуку роботи та стажувань, яка забезпечить зручну взаємодію між кандидатами, компаніями та адміністратором системи.

1.2. Аналіз аналогів вебплатформи для пошуку роботи або стажування

Перед розробкою власної вебплатформи для пошуку роботи або стажування було розглянуто кілька існуючих сервісів зі схожим призначенням. Це потрібно для того, щоб зрозуміти, які можливості вже реалізовані на популярних платформах, які з них є зручними для користувачів, а які можна покращити у власній системі.

Для аналізу було обрано три відомі платформи: LinkedIn, Work.ua та robota.ua. Усі вони допомагають кандидатам знаходити вакансії, створювати профілі або резюме, подавати заявки та взаємодіяти з роботодавцями. Також ці сервіси дають компаніям змогу розміщувати вакансії, переглядати кандидатів і знаходити працівників.

LinkedIn - це міжнародна професійна платформа, яка поєднує можливості соціальної мережі та сервісу для пошуку роботи. Користувач може створити власний професійний профіль, додати інформацію про освіту, досвід роботи, навички, сертифікати, проєкти та досягнення. Також на платформі можна переглядати вакансії, подавати заявки, підписуватися на компанії та спілкуватися з рекрутерами [5]. Приклад інтерфейсу платформи LinkedIn подано на рис. 1.1.

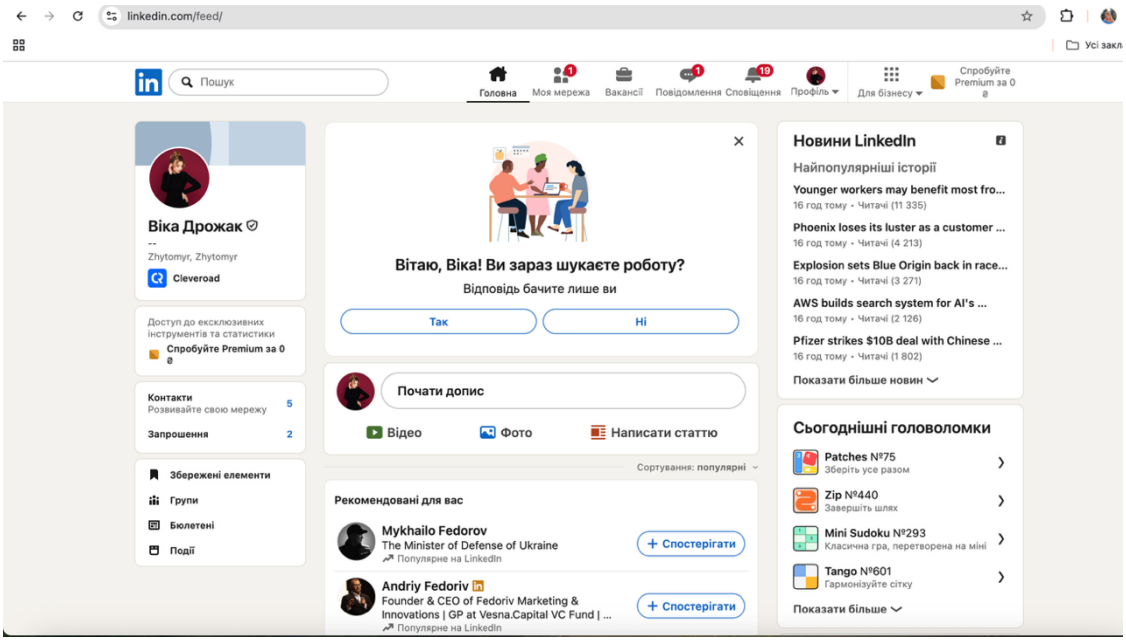


Рис. 1.1. Інтерфейс платформи LinkedIn

Головною особливістю LinkedIn є те, що він використовується не лише для пошуку роботи, а й для розвитку професійних контактів. Користувач може додавати інших спеціалістів до своєї мережі, стежити за новинами компаній, публікувати дописи та показувати свою професійну активність. Це дає можливість кандидату не просто знайти вакансію, а й сформувати власний професійний образ.

Перевагою LinkedIn є велика міжнародна аудиторія, наявність багатьох компаній і рекрутерів, можливість створити детальний профіль та розвивати

професійні зв’язки. Така платформа корисна для кандидатів, які хочуть будувати кар’єру не лише на локальному, а й на міжнародному ринку.

Недоліком LinkedIn є складність для нових користувачів. Через велику кількість функцій інтерфейс може здаватися перевантаженим. Крім того, сервіс не має чіткої орієнтації саме на стажування або кандидатів без досвіду. Для студентів чи початківців пошук відповідних пропозицій може бути менш зручним, оскільки більшість вакансій розраховані на різний рівень досвіду.

Work.ua - це одна з найпопулярніших українських платформ для пошуку роботи. Вона орієнтована переважно на український ринок праці та дозволяє кандидатам швидко знаходити вакансії в різних містах і сферах діяльності. Користувач може створити резюме, заповнити інформацію про освіту, досвід, навички, бажану посаду та рівень заробітної плати [6]. Приклад сторінки платформи Work.ua наведено на рис. 1.2.

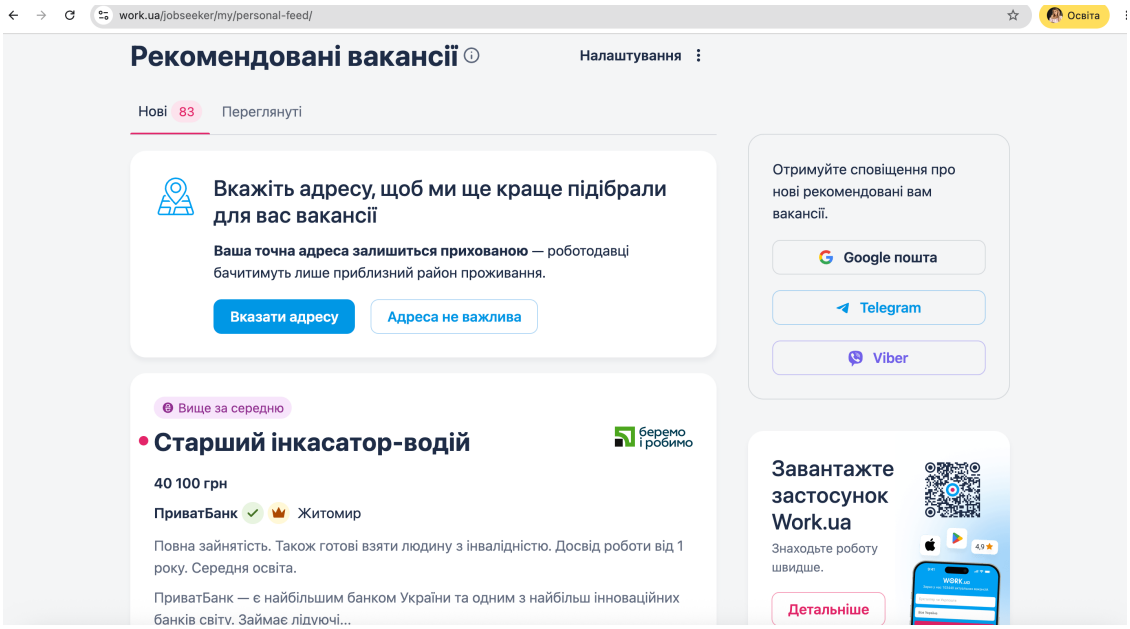


Рис. 1.2. Інтерфейс платформи Work.ua

На Work.ua реалізовано зручний пошук вакансій. Кандидат може використовувати фільтри за містом, професією, типом зайнятості, рівнем заробітної плати, досвідом роботи та іншими параметрами. Це спрощує пошук і дозволяє швидше знайти вакансії, які відповідають потребам користувача.Для роботодавців Work.ua надає можливість створювати вакансії, переглядати

резюме кандидатів і отримувати відгуки на опубліковані пропозиції. Завдяки цьому платформа є корисною як для кандидатів, так і для компаній.

Перевагою Work.ua є простий і зрозумілий інтерфейс, велика кількість вакансій в Україні, зручні фільтри та можливість створення резюме безпосередньо на сайті. Це робить платформу доступною для широкого кола користувачів.

Недоліком Work.ua є те, що сервіс орієнтований переважно на загальний пошук роботи. Він не спеціалізується саме на стажуваннях або першій роботі. Через це кандидатам без досвіду потрібно самостійно шукати відповідні пропозиції серед великої кількості вакансій. Для студентів або початківців це може бути не завжди зручно.

robota.ua - це ще одна популярна українська платформа для пошуку роботи та підбору персоналу. Вона дозволяє кандидатам створювати резюме, переглядати вакансії, використовувати фільтри пошуку та подавати заявки. Компанії, у свою чергу, можуть розміщувати вакансії, переглядати резюме та знаходити працівників відповідно до своїх вимог [7]. Приклад інтерфейсу robota.ua показано на рис. 1.3.

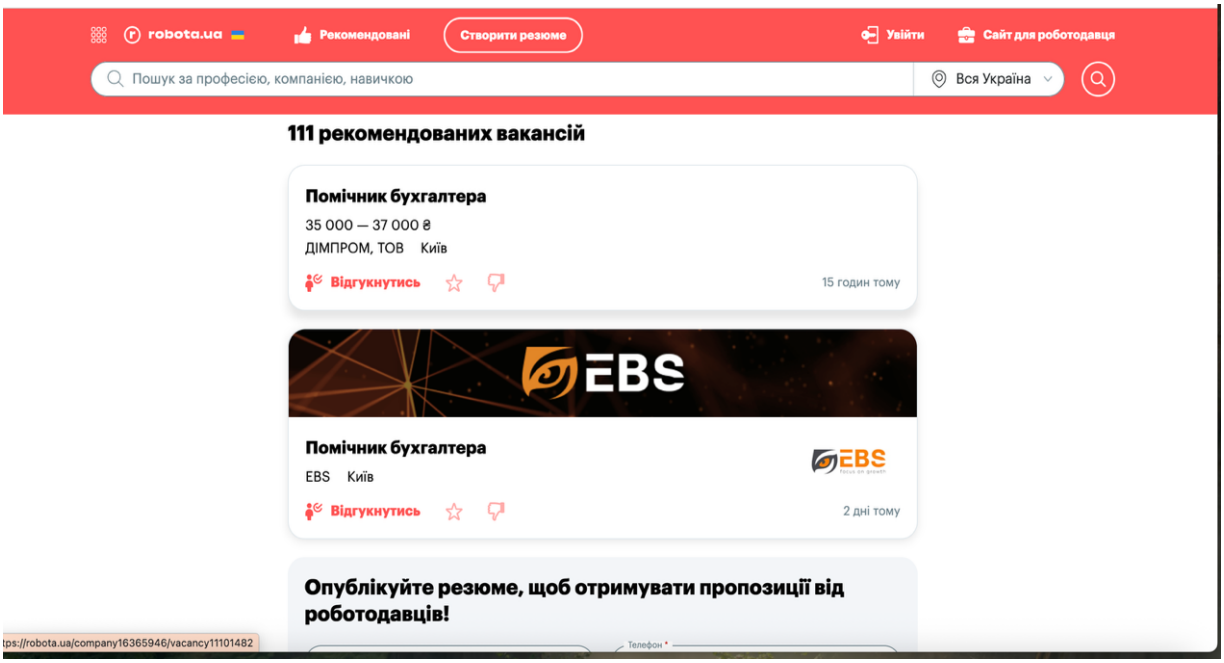


Рис. 1.3. Інтерфейс платформи robota.ua

Платформа robota.ua має велику базу вакансій і зручні інструменти пошуку. Кандидат може шукати роботу за посадою, містом, компанією, рівнем заробітної плати, форматом роботи або іншими параметрами. Це допомагає швидко знаходити потрібні пропозиції.

Перевагою robota.ua є орієнтація на український ринок, велика кількість вакансій, зручна система фільтрації та наявність функціоналу як для кандидатів, так і для роботодавців. Платформа є корисною для людей із різним рівнем досвіду та для компаній, які шукають працівників.

Недоліком robota.ua є те, що вона також більше орієнтована на широкий ринок праці, а не на стажування або кандидатів-початківців. Через це користувачам без досвіду потрібно витрачати більше часу на пошук вакансій, які підходять саме для старту кар'єри. Крім того, частина функцій може бути зайвою для користувачів, яким потрібен простий сервіс для пошуку стажування.

Таблиця 1.1

Порівняння аналогів вебплатформи для пошуку роботи або стажування

Критерій порівняння	**LinkedIn**	**Work.ua**	**robota.ua**
Основне призначення	Пошук роботи та професійні контакти	Пошук роботи в Україні	Пошук роботи в Україні
Цільова аудиторія	Спеціалісти, рекрутери, компанії	Кандидати та роботодавці	Кандидати та роботодавці
Створення профілю та резюме	Так	Так	Так
Пошук і фільтрація вакансій	Так	Так	Так
Подання заявки	Так	Так	Так
Кабінет кандидата	Так	Так	Так
Кабінет компанії	Так	Так	Так
Перегляд заявок кандидатів	Так	Так	Так
Орієнтація на український ринок	Частково	Так	Так

Продовження таблиці 1.1

Орієнтація на стажування	Частково	Частково	Частково
Простота інтерфейсу	Середня	Висока	Висока
Інформаційні статті	Так	Так	Так
Основна перевага	Міжнародна аудиторія	Багато вакансій в Україні	Зручний пошук вакансій
Основний недолік	Складний для нових користувачів	Немає спеціалізації на стажуваннях	Не орієнтований на кандидатів без досвіду

1.3. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для реалізації вебплатформи для пошуку роботи або стажування було обрано набір інструментів, які дозволяють створити повноцінний вебзастосунок. Система повинна мати зручний інтерфейс для користувачів, серверну частину для обробки запитів, базу даних для зберігання інформації, а також засоби для авторизації та завантаження файлів.

Під час вибору технологій враховувалося, наскільки вони зручні у використанні, як швидко з їхньою допомогою можна розробити проєкт і чи буде легко підтримувати систему в майбутньому. Також важливо, щоб обрані інструменти добре поєднувалися між собою та давали змогу надалі розширювати функціонал платформи.

Для створення клієнтської частини було використано мову програмування JavaScript [26] та бібліотеку React [8]. JavaScript є однією з основних мов для створення вебсайтів, оскільки вона дозволяє робити сторінки інтерактивними. Наприклад, за допомогою JavaScript можна обробляти натискання кнопок, відкривати форми, оновлювати список вакансій або показувати повідомлення користувачу. React було обрано тому, що він дозволяє створювати інтерфейс із окремих частин - компонентів. Наприклад, окремими компонентами можуть бути

форма входу, картка вакансії, кнопка подання заявки, меню, сторінка профілю кандидата або кабінет компанії. Такий підхід є зручним, тому що кожну частину можна розробляти, змінювати та використовувати повторно в різних місцях сайту [8].

Для швидкого запуску та збирання клієнтської частини було використано Vite. Цей інструмент спрощує роботу над React-проєктом і дозволяє швидко бачити зміни під час розробки. Це зручно, тому що після редагування коду сторінка оновлюється майже одразу, і розробник може швидко перевірити результат.

Для оформлення зовнішнього вигляду вебплатформи було використано CSS. За допомогою CSS задаються кольори, шрифти, відступи, розташування елементів, вигляд кнопок, форм, меню та інших частин інтерфейсу. Також CSS дає змогу адаптувати сторінки під різні розміри екрана, щоб платформа коректно відображалася на комп’ютерах, планшетах і смартфонах. Завдяки цьому інтерфейс стає більш зручним, зрозумілим і приємним для користувачів [10].

Під час розробки дизайну основна увага приділялася простоті та зручності. Інтерфейс не повинен бути перевантаженим зайвими елементами, щоб кандидат міг швидко знайти вакансії, компанія створити нову пропозицію, а адміністратор перейти до потрібного розділу керування. Для цього доцільно використовувати зрозуміле меню, помітні кнопки, однакове оформлення сторінок і логічне розміщення основних блоків [30].

Також важливо, щоб дизайн був адаптивним. Це означає, що вебплатформа повинна коректно відображатися не лише на комп’ютері або ноутбуці, а й на планшеті чи смартфоні. Адаптивність є важливою, оскільки користувачі можуть переглядати вакансії, подавати заявки або перевіряти їхній статус із різних пристроїв.

Для створення серверної частини було використано Node.js та Express.js. Node.js дає можливість виконувати JavaScript на сервері, а не лише в браузері. Завдяки цьому клієнтська та серверна частини проєкту можуть бути написані однією мовою програмування, що спрощує розробку й підтримку системи [11].

Express.js використовується як вебфреймворк для Node.js. Він допомагає створювати серверні маршрути, обробляти запити від клієнтської частини, передавати дані між інтерфейсом і базою даних та повертати користувачу потрібний результат [12].

Серверна частина відповідає за основну логіку роботи системи. Вона приймає запити від користувачів, перевіряє дані, працює з базою даних і повертає відповідь на клієнтську частину. Наприклад, коли кандидат подає заявку на вакансію, сервер отримує ці дані, перевіряє їх і зберігає заявку в базі даних. Коли компанія відкриває список кандидатів, сервер отримує потрібну інформацію з бази та передає її для відображення на сторінці.

Для зберігання даних у вебплатформі було використано сервіс Supabase. Це хмарна платформа, яка надає готові інструменти для роботи з базою даних, авторизацією користувачів і файлами. Основою Supabase є база даних PostgreSQL, тому вона підходить для зберігання структурованої інформації, яка використовується в системі [13].

Supabase було обрано тому, що він дозволяє швидко налаштувати роботу з даними без створення окремого сервера бази даних. Це спрощує розробку, оскільки значна частина необхідних можливостей уже доступна в самому сервісі. У базі даних зберігається основна інформація платформи: дані користувачів, профілі кандидатів, профілі компаній, вакансії, заявки на роботу або стажування, а також відомості про резюме.

Завдяки використанню бази даних система може зберігати, оновлювати та швидко отримувати потрібну інформацію. Наприклад, коли компанія створює вакансію, вона зберігається в базі даних і стає доступною для кандидатів. Якщо кандидат подає заявку, ця інформація також зберігається та може бути переглянута компанією.

Supabase також підтримує роботу з файлами, що є важливим для цієї платформи. Кандидат може завантажити своє резюме, а система зберігає файл у сховищі. У базі даних при цьому можна зберігати посилання на файл або коротку

інформацію про нього. Це робить роботу з резюме зручнішою та допомагає пов’язати документи з конкретними користувачами і заявками [14].

Ще однією важливою частиною є авторизація користувачів. Після входу в систему користувач отримує доступ тільки до тих можливостей, які відповідають його ролі. Наприклад, кандидат може переглядати вакансії та подавати заявки, компанія може створювати вакансії та переглядати кандидатів, а адміністратор може контролювати роботу платформи. Такий розподіл прав потрібен для безпеки та правильного функціонування системи [15].

Для зручної організації співбесід у вебплатформі передбачається можливість створення онлайн-зустрічей між кандидатом і представником компанії. Це дозволить проводити співбесіди дистанційно та не створювати посилання на зустріч вручну через окремі сервіси.

Інтеграція з Google Meet планується через Google Calendar API [16]. Під час створення співбесіди компанія зможе вибрати кандидата, дату та час зустрічі. Після цього серверна частина на основі цих даних створюватиме подію в Google Calendar. Для створення події використовується метод додавання події до календаря, а для формування онлайн-зустрічі може застосовуватися параметр створення конференції, який додає до події посилання на Google Meet [17].

Після створення події система отримає посилання на Google Meet і збереже його в базі даних. Це посилання буде відображатися в особистому кабінеті кандидата та компанії. Завдяки цьому обидві сторони зможуть швидко перейти до онлайн-співбесіди у визначений час. Такий підхід робить процес комунікації між кандидатом і роботодавцем простішим, оскільки вся інформація про співбесіду зберігається в одному місці.

Висновки до першого розділу

У першому розділі було розглянуто особливості створення вебплатформи для пошуку роботи та стажувань. Проаналізовано сервіси LinkedIn, Work.ua та robota.ua і встановлено, що вони мають широкий функціонал для пошуку роботи, створення резюме та взаємодії між кандидатами й роботодавцями. Водночас ці платформи переважно орієнтовані на загальний ринок праці та не завжди зручні

для студентів, початківців і користувачів, які шукають стажування або першу роботу.

На основі аналізу було визначено основні завдання розроблюваної системи: реєстрація й авторизація користувачів, створення профілів кандидатів і компаній, пошук та фільтрація вакансій, подання заявок, робота з резюме, перегляд статусу заявок і адміністрування платформи. Також визначено три основні ролі користувачів: кандидат, компанія та адміністратор, для яких передбачено окремі функції.

У розділі обґрунтовано вибір технологій для реалізації системи. Для клієнтської частини обрано JavaScript, React, Vite та CSS, для серверної Node.js і Express.js. Для зберігання даних, авторизації та роботи з файлами використовується Supabase. Додатково передбачено інтеграцію з Google Meet через Google Calendar API для організації онлайн-співбесід.

Отже, у першому розділі було визначено основні функції, ролі користувачів, вимоги до системи та технології її реалізації. Проведений аналіз підтвердив доцільність розробки вебплатформи, яка спрощує взаємодію між кандидатами й роботодавцями та об’єднує пошук вакансій, подання заявок і керування вакансіями в одному середовищі.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБПЛАТФОРМИ ДЛЯ ПОШУКУ РОБОТИ ТА СТАЖУВАННЯ

2.1. Визначення варіантів використання та об’єктно-орієнтованої структури системи

Розроблювана система призначена для автоматизації процесів пошуку роботи та стажувань, взаємодії кандидатів із роботодавцями, створення й опрацювання заявок, керування резюме, публікації вакансій та адміністрування вмісту платформи. Вебплатформа забезпечує єдине середовище, у якому користувачі можуть виконувати основні дії, пов’язані з пошуком роботи, стажування або кандидатів [18].

Основними користувачами системи є кандидати, представники компаній та адміністратор. Кожна роль має власний набір функціональних можливостей. Кандидат може переглядати вакансії, застосовувати фільтри, зберігати цікаві пропозиції, створювати або завантажувати резюме, подавати заявки та відстежувати їхній статус. Представник компанії може створювати й редагувати профіль компанії, публікувати вакансії, переглядати заявки кандидатів, аналізувати резюме, змінювати статуси заявок і організовувати подальшу комунікацію. Адміністратор забезпечує контроль роботи платформи, керує користувачами, компаніями, вакансіями, заявками, скаргами та інформаційними матеріалами.

Основною метою системи є створення зручного вебсередовища, яке поєднує кандидатів і роботодавців. Кандидат отримує можливість швидко знайти актуальні вакансії або стажування, переглянути детальну інформацію про пропозицію та подати заявку. Компанія, у свою чергу, може розміщувати вакансії, опрацьовувати відгуки, переглядати резюме та шукати потенційних працівників або стажерів.

Для наочного подання взаємодії користувачів із системою використовується діаграма варіантів використання. Вона відображає основні ролі користувачів і функції, доступні кожній із них. Діаграма варіантів використання вебплатформи для пошуку роботи та стажування наведена на рис. 2.1.

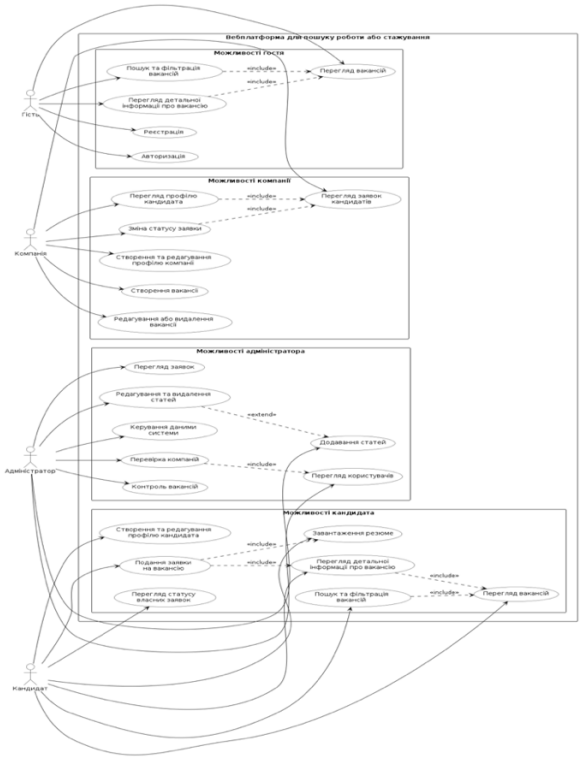


Рис.2.1. Діаграма варіантів використання вебплатформи для пошуку роботи або стажування

У межах вебплатформи для пошуку роботи або стажування основними акторами є гість, кандидат, компанія та адміністратор. Гість має доступ лише до публічних функцій системи, кандидат використовує платформу для пошуку вакансій і подання заявок, компанія розміщує вакансії та опрацьовує кандидатів, а адміністратор контролює роботу системи та здійснює модерацію даних.

Для визначення функціональних вимог до вебплатформи було розроблено прецеденти використання для кожної категорії користувачів. Першою категорією є гість, який має доступ лише до відкритих функцій системи. Опис основних прецедентів використання для гостя наведено в таблиці 2.1.

Таблиця 2.1

Опис прецедентів для гостя

№	Назва прецеденту	Актор	Передумови	Основний сценарій	Результат
1	Перегляд вакансій	Гість	Користувач відкрив вебплатформу	Гість переходить на сторінку вакансій і переглядає доступні пропозиції роботи або стажування	Система відображає список актуальних вакансій

Продовження таблиці 2.1

№	Назва прецеденту	Актор	Передумови	Основний сценарій	Результат
2	Пошук та фільтрація вакансій	Гість	У системі наявні опубліковані вакансії	Гість вводить ключові слова або обирає фільтри за напрямом, містом, форматом роботи чи типом зайнятості	Система показує список вакансій, що відповідають заданим критеріям
3	Перегляд детальної інформації про вакансію	Гість	Гість обрав вакансію зі списку	Система відкриває сторінку з повним описом вакансії, вимогами, умовами та інформацією про компанію	Гість отримує детальну інформацію про вакансію
4	Реєстрація	Гість	Користувач не має облікового запису	Гість обирає тип облікового запису, вводить необхідні дані та підтверджує реєстрацію	У системі створюється новий обліковий запис
5	Авторизація	Гість	Користувач має зареєстрований обліковий запис	Гість вводить електронну пошту та пароль або використовує інший доступний спосіб входу	Користувач входить у систему та отримує доступ до функцій відповідної ролі

Як видно з таблиці 2.1, гість може переглядати вакансії, здійснювати їх пошук і фільтрацію, переглядати детальну інформацію про вакансії, а також виконувати реєстрацію та авторизацію. Реалізація зазначених прецедентів забезпечує ознайомлення користувача з можливостями вебплатформи та створює умови для подальшої роботи після входу до системи.

Після авторизації кандидат отримує доступ до розширеного функціоналу вебплатформи. Основні прецеденти використання, які характеризують взаємодію кандидата із системою, наведено в таблиці 2.2

Таблиця 2.2

Опис прецедентів для кандидата

№	Назва прецеденту	Актор	Передумови	Основний сценарій	Результат
1	Створення та редагування профілю кандидата	Кандидат	Кандидат авторизований у системі	Кандидат відкриває особистий профіль, заповнює або змінює особисті дані, контактну інформацію, навички, освіту та досвід	Дані профілю кандидата зберігаються в системі
2	Завантаження резюме	Кандидат	Кандидат авторизований у системі	Кандидат обирає файл резюме та завантажує його до системи	Резюме зберігається та може бути використане під час подання заявки
3	Пошук та фільтрація вакансій	Кандидат	Кандидат авторизований або перебуває у відкритій частині системи	Кандидат вводить пошуковий запит або застосовує фільтри	Система відображає релевантні вакансії
4	Перегляд детальної інформації про вакансію	Кандидат	Кандидат обрав вакансію зі списку	Система показує опис вакансії, вимоги, умови роботи, компанію та доступні дії	Кандидат отримує повну інформацію для прийняття рішення
5	Подання заявки на вакансію	Кандидат	Кандидат авторизований і має створене або завантажене резюме	Кандидат відкриває вакансію, обирає резюме та надсилає заявку	У системі створюється заявка на вакансію
6	Перегляд статусу власних заявок	Кандидат	Кандидат подав хоча б одну заявку	Кандидат переходить до розділу власних заявок	Система відображає список заявок та їхні поточні статуси
7	Збереження вакансії	Кандидат	Кандидат авторизований у системі	Кандидат натискає кнопку збереження вакансії	Вакансія додається до списку обраних
8	Приховування нецікавої вакансії	Кандидат	Кандидат авторизований у системі	Кандидат позначає вакансію як нецікаву	Вакансія приховується зі списку рекомендацій або основного перегляду

Відповідно до таблиці 2.2, кандидат може керувати власним профілем і резюме, здійснювати пошук вакансій, подавати заявки, переглядати їхній статус,

а також формувати персональний список вакансій. Наведені прецеденти охоплюють повний цикл взаємодії кандидата з вебплатформою.

Наступною категорією користувачів є компанія, яка використовує вебплатформу для пошуку кандидатів та керування вакансіями. Основні прецеденти використання для компанії наведено в таблиці 2.3.

Таблиця 2.3

Опис прецедентів для компанії

№	Назва прецеденту	Актор	Передумови	Основний сценарій	Результат
1	Створення та редагування профілю компанії	Компанія	Представник компанії авторизований у системі	Компанія заповнює або оновлює назву, опис, контактні дані, сайт, місто та іншу інформацію	Профіль компанії зберігається або оновлюється
2	Створення вакансії	Компанія	Компанія авторизована та має заповнений профіль	Представник компанії вводить назву вакансії, опис, вимоги, умови, місто, формат роботи та тип зайнятості	Нова вакансія додається до системи
3	Редагування або видалення вакансії	Компанія	Компанія має створену вакансію	Представник компанії відкриває вакансію, змінює її дані або видаляє її	Дані вакансії оновлюються або вакансія видаляється
4	Перегляд заявок кандидатів	Компанія	На вакансії компанії подано заявки	Компанія відкриває список заявок на свої вакансії	Система відображає кандидатів, які подали заявки
5	Перегляд профілю кандидата	Компанія	Кандидат подав заявку на вакансію компанії	Компанія відкриває профіль кандидата або його резюме	Компанія отримує інформацію для оцінки кандидата
6	Зміна статусу заявки	Компанія	Є отримана заявка від кандидата	Представник компанії змінює статус заявки, наприклад на «переглянуто», «запрошено» або «відхилено»	Статус заявки оновлюється та стає доступним кандидату

Як показано в таблиці 2.3, компанія має можливість створювати та редагувати власний профіль, публікувати вакансії, переглядати кандидатів, опрацьовувати заявки та змінювати їхній статус. Реалізація цих прецедентів забезпечує ефективну організацію процесу підбору персоналу.

Окрему роль у роботі вебплатформи виконує адміністратор, який відповідає за контроль, модерацію та підтримку коректної роботи системи. Основні прецеденти використання адміністратора наведено в таблиці 2.4.

Таблиця 2.4

Опис прецедентів для адміністратора

№	Назва прецеденту	Актор	Передумови	Основний сценарій	Результат
1	Перевірка компаній	Адміністратор	Адміністратор авторизований у системі	Адміністратор переглядає зареєстровані компанії та перевіряє коректність їхніх даних	Компанія отримує відповідний статус перевірки
2	Контроль вакансій	Адміністратор	У системі наявні створені вакансії	Адміністратор переглядає вакансії, перевіряє їхній зміст і за потреби виконує модерацію	Некоректні або застарілі вакансії редагуються, блокуються або видаляються
3	Перегляд заявок	Адміністратор	У системі наявні заявки кандидатів	Адміністратор переглядає інформацію про подані заявки	Адміністратор отримує загальне уявлення про активність користувачів
4	Перегляд користувачів	Адміністратор	У системі є зареєстровані користувачі	Адміністратор відкриває список користувачів і переглядає їхні ролі та статуси	Система відображає інформацію про користувачів
5	Додавання статей	Адміністратор	Адміністратор авторизований у системі	Адміністратор створює інформаційну статтю, додає заголовок, текст і публікує її	Нова стаття стає доступною користувачам

Продовження таблиці 2.4

6	Редагування та видалення статей	Адміністратор	У системі наявні інформаційні статті	Адміністратор змінює зміст статті або видаляє її	Інформаційний матеріал оновлюється або видаляється
7	Керування даними системи	Адміністратор	Адміністратор має відповідні права доступу	Адміністратор виконує службові дії з підтримки актуальності та коректності даних	Система підтримується в актуальному та працездатному стані

З таблиці 2.4 видно, що адміністратор виконує функції перевірки компаній, контролю вакансій, перегляду користувачів і заявок, керування інформаційними статтями та підтримки актуальності даних. Реалізація зазначених прецедентів забезпечує стабільне функціонування вебплатформи, контроль достовірності інформації та ефективне адміністрування системи.

Наведені прецеденти відображають основну функціональність вебплатформи та демонструють взаємодію між користувачами й системою. Вони дозволяють визначити, які модулі необхідно реалізувати в програмному продукті, а також які дані мають зберігатися та оброблятися. Зокрема, для реалізації описаних сценаріїв необхідні модулі автентифікації, керування профілями, роботи з вакансіями, обробки заявок, керування резюме, адміністрування та модерації.

Після визначення варіантів використання та основних вимог до системи доцільно перейти до побудови діаграми класів.

Діаграма класів відображає внутрішню структуру програмного продукту, основні сутності предметної області, їхні властивості та взаємозв’язки між ними.

Діаграма класів також дає змогу відобразити взаємозв'язки між основними сутностями системи та визначити їхню роль у забезпеченні функціональних можливостей вебплатформи. Для вебплатформи така діаграма дозволяє показати, які об'єкти беруть участь у процесах реєстрації користувачів, створення профілів, публікації вакансій, завантаження резюме, подання заявок і адміністрування системи (рис. 2.2).

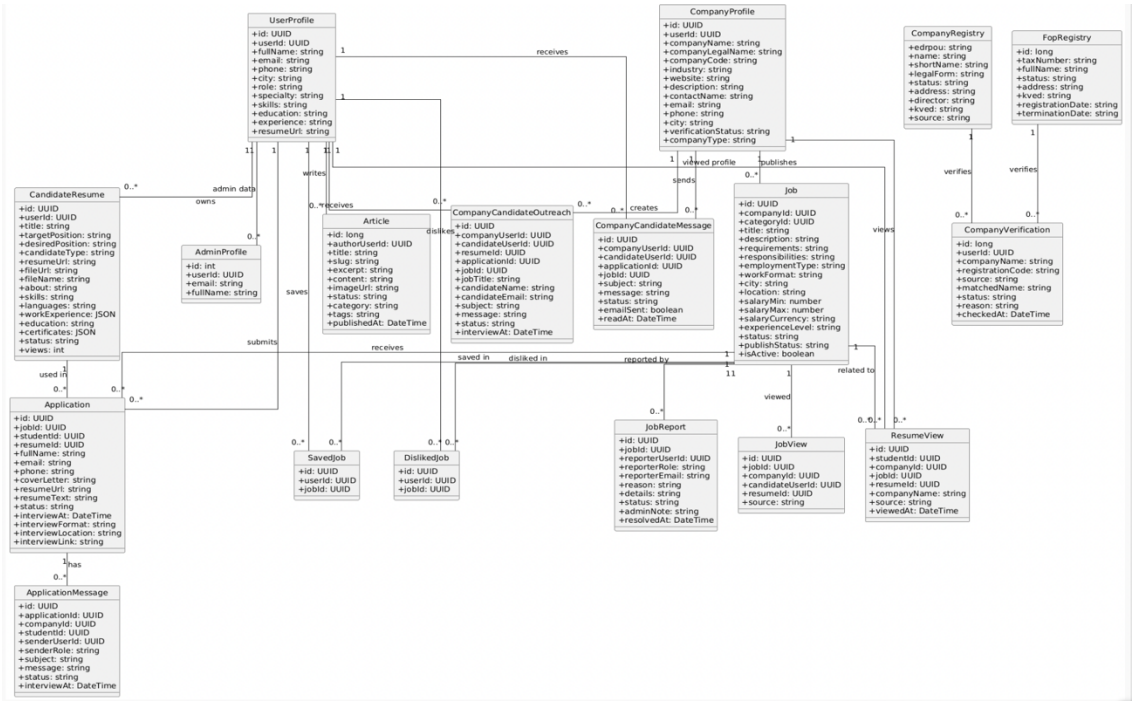


Рис.2.2. Діаграма класів вебплатформи для пошуку роботи або стажування

Основною сутністю діаграми класів є клас UserProfile, який описує кандидата та містить основні персональні дані користувача: ідентифікатор, ім'я, контактну інформацію, місто, спеціальність, навички, освіту, досвід роботи та інші відомості, необхідні для пошуку роботи або стажування. Користувач може створювати резюме, подавати заявки на вакансії, переглядати статус своїх заявок, зберігати цікаві вакансії та взаємодіяти з роботодавцями.

Для зберігання інформації про адміністраторів використовується клас AdminProfile, який містить службову інформацію про користувачів із адміністративними правами. Адміністратори здійснюють модерацію контенту, контролюють роботу вебплатформи, перевіряють інформацію про роботодавців і забезпечують коректне функціонування системи.

Клас CompanyProfile представляє роботодавця та містить інформацію про компанію: назву, код ЄДРПОУ, сферу діяльності, опис, контактні дані, вебсайт, місто, тип компанії та статус перевірки. Представники компаній мають можливість створювати вакансії, переглядати заявки кандидатів, надсилати повідомлення та запрошення на співбесіди.

Клас CandidateResume описує резюме кандидата. Він містить інформацію про бажану посаду, професійні навички, освіту, досвід роботи, знання мов, сертифікати, приклади виконаних робіт, а також файл резюме або посилання на нього. Один користувач може створити декілька резюме, при цьому під час подання заявки використовується одне обране резюме.

Клас Job описує вакансію або стажування. До його атрибутів належать назва вакансії, опис, вимоги, обов'язки, формат роботи, тип зайнятості, місто, рівень досвіду, розмір заробітної плати та статус публікації. Кожна вакансія належить одній компанії, тоді як одна компанія може створювати необмежену кількість вакансій.

Клас Application описує заявку кандидата на вакансію. Він пов'язує між собою кандидата, вакансію та резюме, що використовується під час подання заявки. Заявка містить контактні дані кандидата, супровідний лист, дату подання, поточний статус, інформацію про заплановану співбесіду та інші службові відомості. Один кандидат може подати декілька заявок, а одна вакансія може містити заявки від багатьох кандидатів.

Для організації взаємодії між роботодавцями та кандидатами використовуються класи ApplicationMessage та CompanyCandidateMessage. Перший забезпечує листування в межах конкретної заявки на вакансію, тоді як другий використовується для прямого обміну повідомленнями між роботодавцем і кандидатом незалежно від наявності заявки. Крім цього, клас CompanyCandidateOutreach реалізує механізм активного пошуку кандидатів, дозволяючи компаніям самостійно надсилати пропозиції щодо працевлаштування.

Для реалізації додаткових можливостей вебплатформи використовуються класи SavedJob, DislikedJob, ResumeView, JobView, JobReport та Article. Клас SavedJob забезпечує збереження вакансій, які зацікавили кандидата, DislikedJob використовується для приховування вакансій, що не відповідають його інтересам, ResumeView та JobView фіксують перегляди резюме й вакансій, а JobReport дозволяє користувачам повідомляти адміністрацію про вакансії, що

порушують правила платформи або містять недостовірну інформацію. Клас Article використовується для публікації інформаційних матеріалів, рекомендацій щодо пошуку роботи, написання резюме та проходження співбесід.

Для перевірки роботодавців у системі використовуються класи CompanyRegistry, CompanyVerification та FopRegistry. Вони забезпечують зберігання інформації про юридичних осіб і фізичних осіб-підприємців, а також результатів перевірки роботодавців, що сприяє підвищенню достовірності інформації, розміщеної на вебплатформі.

Таким чином, діаграма класів відображає логічну структуру вебплатформи, демонструє основні сутності системи, їхні атрибути та взаємозв'язки. Вона є основою для подальшого проєктування структури бази даних, реалізації серверної логіки та побудови програмної архітектури вебплатформи.

2.2. Розробка бази даних системи

База даних є одним із ключових компонентів вебплатформи для пошуку роботи та стажування, оскільки забезпечує централізоване зберігання, обробку та взаємозв'язок інформації, необхідної для функціонування всіх модулів системи. Саме база даних відповідає за збереження відомостей про користувачів, профілі компаній, резюме кандидатів, вакансії, заявки, повідомлення між учасниками платформи, перегляди вакансій і резюме, інформаційні статті, а також службові дані, необхідні для роботи адміністративної частини системи.

Під час проєктування бази даних було враховано особливості роботи трьох основних категорій користувачів: кандидатів, представників компаній та адміністраторів. Кандидати використовують платформу для створення особистого профілю, завантаження або формування резюме, пошуку вакансій, збереження цікавих пропозицій, подання заявок та комунікації з роботодавцями. Представники компаній створюють профілі організацій, публікують вакансії, переглядають резюме кандидатів, працюють із заявками, надсилають повідомлення та запрошення на співбесіду. Адміністратори здійснюють контроль роботи системи, перевірку компаній, модерацію вакансій і статей, а також опрацювання скарг користувачів.

Основними вимогами до бази даних є забезпечення цілісності інформації, швидкого доступу до даних, підтримка механізмів пошуку та фільтрації вакансій, масштабованість системи, розмежування прав доступу відповідно до ролей користувачів і підтримка взаємозв'язків між усіма основними сутностями. Для реалізації цих вимог було обрано реляційну модель даних, яка забезпечує ефективне зберігання інформації та мінімізує дублювання даних завдяки використанню зовнішніх ключів і логічних зв'язків між таблицями.

Як систему керування базами даних використано PostgreSQL. Вибір саме цієї СКБД обумовлений її високою продуктивністю, підтримкою транзакцій, механізмів забезпечення цілісності даних, індексування, складних SQL-запитів, а також широкими можливостями роботи з типами даних JSON і JSONB. PostgreSQL забезпечує надійну роботу багатокористувацьких інформаційних систем, підтримує масштабування та добре інтегрується із сучасними вебтехнологіями, що робить її доцільним рішенням для реалізації вебплатформи пошуку роботи та стажування.

На рисунку 2.3 наведено діаграму структури бази даних вебплатформи. Вона відображає основні таблиці, їх атрибути, первинні та зовнішні ключі, а також зв'язки між сутностями, що забезпечують цілісність і узгодженість даних. Діаграма демонструє логічну організацію бази даних та взаємодію між основними функціональними модулями системи. До її складу входять таблиці профілів користувачів і компаній, резюме кандидатів, вакансій, заявок, повідомлень, переглядів вакансій і резюме, збережених вакансій, скарг на вакансії, інформаційних статей, а також таблиці, що забезпечують перевірку роботодавців і ведення державних реєстрів. Така структура забезпечує ефективне зберігання даних, підтримує процеси пошуку роботи, взаємодії між кандидатами та роботодавцями, а також створює основу для подальшого розвитку й масштабування вебплатформи.

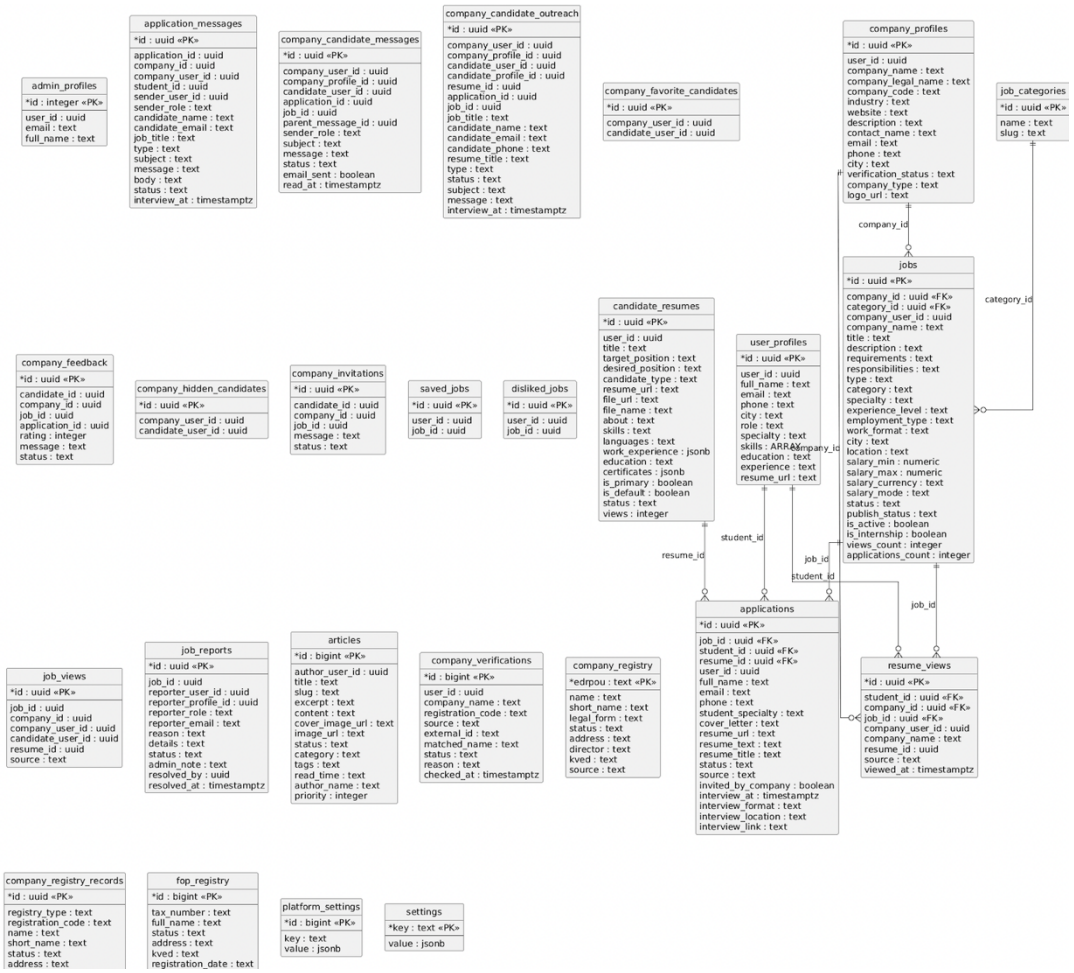


Рис. 2.3. Діаграма структури бази даних

У процесі проєктування бази даних було сформовано набір взаємопов'язаних таблиць, кожна з яких відповідає окремій сутності предметної області. Основними таблицями системи є user_profiles, company_profiles, candidate_resumes, jobs, job_categories, applications, application_messages, company_candidate_messages, company_candidate_outreach, saved_jobs, disliked_jobs, job_reports, articles, company_registry, company_registry_records, company_verifications, fop_registry та інші службові таблиці. Такий поділ дозволяє забезпечити чітке розмежування інформації між різними функціональними модулями системи та спрощує її подальший супровід і масштабування.

Основною таблицею, що містить інформацію про користувачів платформи, є user_profiles. У ній зберігаються персональні дані кандидатів, контактна інформація, професійні навички, освіта, досвід роботи та інші відомості,

необхідні для формування особистого профілю користувача. Таблиця також містить унікальний ідентифікатор користувача, що використовується для встановлення зв'язків з іншими таблицями бази даних. Структуру таблиці user_profiles наведено в табл. 2.5.

Таблиця 2.5

Структура таблиці user_profiles

Поле	**Тип даних**	**Опис**
id	UUID	Унікальний ідентифікатор профілю користувача
user_id	UUID	Ідентифікатор користувача в системі автентифікації
full_name	TEXT	Повне ім'я користувача
email	TEXT	Адреса електронної пошти
phone	TEXT	Контактний номер телефону
city	TEXT	Місто проживання
specialty	TEXT	Спеціальність або професійний напрям
education	TEXT	Відомості про освіту
experience	TEXT	Інформація про професійний досвід
skills	TEXT	Професійні навички користувача
resume_url	TEXT	Посилання на завантажене резюме

Таблиця user_profiles використовується для зберігання персональної інформації кандидатів. Дані цієї таблиці відображаються в особистому кабінеті користувача та використовуються під час створення резюме, подання заявок на вакансії, формування списку рекомендованих вакансій і пошуку кандидатів роботодавцями. Поле user_id забезпечує зв'язок профілю з обліковим записом користувача, а унікальний ідентифікатор id використовується як зовнішній ключ в інших таблицях бази даних.

Для зберігання інформації про роботодавців використовується таблиця company_profiles. Вона містить відомості про організацію, контактну інформацію, напрям діяльності, статус перевірки та інші дані, необхідні для публікації вакансій і взаємодії з кандидатами. Структуру таблиці company_profiles наведено в табл. 2.6.

Таблиця 2.6

Структура таблиці company_profiles

Поле	**Тип даних**	**Опис**
id	UUID	Унікальний ідентифікатор компанії
user_id	UUID	Ідентифікатор користувача, який представляє компанію
company_name	TEXT	Назва компанії
company_legal_name	TEXT	Повна юридична назва
company_code	TEXT	Код ЄДРПОУ
industry	TEXT	Галузь діяльності
website	TEXT	Офіційний вебсайт компанії
description	TEXT	Опис діяльності
contact_name	TEXT	Контактна особа
email	TEXT	Контактна електронна пошта
phone	TEXT	Контактний телефон
city	TEXT	Місто розташування
verification_status	TEXT	Статус перевірки компанії

Таблиця company_profiles використовується для зберігання інформації про роботодавців, які розміщують вакансії на платформі. Дані цієї таблиці застосовуються під час відображення профілю компанії, формування сторінки роботодавця, публікації вакансій і перевірки достовірності інформації адміністраторами. Кожна компанія пов'язана з конкретним користувачем системи через поле user_id**.**

Для зберігання інформації про резюме кандидатів використовується таблиця candidate_resumes**.** Вона дозволяє одному користувачеві створювати декілька резюме для різних напрямів працевлаштування та використовувати їх під час подання заявок. Структуру таблиці candidate_resumes наведено в табл. 2.7.

Таблиця 2.7

Структура таблиці candidate_resumes

Поле	**Тип даних**	**Опис**
id	UUID	Унікальний ідентифікатор резюме
user_id	UUID	Ідентифікатор власника резюме
title	TEXT	Назва резюме
target_position	TEXT	Бажана посада

Продовження таблиці 2.7

Поле	**Тип даних**	**Опис**
candidate_type	TEXT	Тип кандидата
resume_url	TEXT	Посилання на файл резюме
about	TEXT	Коротка інформація про кандидата
skills	TEXT	Професійні навички
languages	TEXT	Володіння мовами
work_experience	JSONB	Інформація про досвід роботи
education	TEXT	Освіта
certificates	JSONB	Сертифікати та додаткові досягнення
status	TEXT	Поточний статус резюме

Таблиця candidate_resumes забезпечує зберігання професійної інформації кандидатів та використовується під час формування особистого профілю, перегляду резюме компаніями й подання заявок на вакансії. Одне резюме може використовуватися для кількох заявок, що забезпечує повторне використання вже введених користувачем даних без необхідності їх повторного заповнення.

Основною таблицею, що містить інформацію про вакансії, є jobs. У ній зберігаються відомості про роботодавця, назву вакансії, вимоги до кандидатів, умови праці, формат роботи, рівень заробітної плати та інші характеристики вакансії. Структуру таблиць jobs наведено відповідно в табл. 2.8

Таблиця 2.8

Структура таблиці jobs

Поле	**Тип даних**	**Опис**
id	UUID	Унікальний ідентифікатор вакансії
company_id	UUID	Посилання на компанію
category_id	UUID	Посилання на категорію вакансій
title	TEXT	Назва вакансії
description	TEXT	Опис вакансії
requirements	TEXT	Вимоги до кандидата
responsibilities	TEXT	Основні обов'язки
employment_type	TEXT	Тип зайнятості
work_format	TEXT	Формат роботи
city	TEXT	Місто роботи
salary_min	NUMERIC	Мінімальний рівень заробітної плати
salary_max	NUMERIC	Максимальний рівень заробітної плати

Продовження таблиці 2.8

Поле	Тип даних	Опис
experience_level	TEXT	Необхідний рівень досвіду
status	TEXT	Статус вакансії

Таблиця jobs є однією з центральних таблиць бази даних. У ній зберігаються всі відомості про вакансії, що публікуються роботодавцями. Кожна вакансія належить конкретній компанії та пов'язана з категорією через зовнішній ключ category_id. Використання окремих полів для мінімального та максимального розміру заробітної плати забезпечує можливість реалізації пошуку вакансій за бажаним рівнем доходу, а статус вакансії визначає її поточний стан (активна, закрита або прихована).

Для реалізації процесу працевлаштування використовується таблиця applications, яка містить інформацію про заявки кандидатів на вакансії. Саме ця таблиця поєднує користувача, його резюме та вакансію, на яку було подано заявку. Структуру таблиці applications наведено в табл. 2.9.

Таблиця 2.9

Структура таблиці applications

Поле	Тип даних	Опис
id	UUID	Унікальний ідентифікатор заявки
job_id	UUID	Посилання на вакансію
student_id	UUID	Посилання на кандидата
resume_id	UUID	Посилання на резюме
cover_letter	TEXT	Супровідний лист
status	TEXT	Поточний статус заявки
full_name	TEXT	Ім'я кандидата
email	TEXT	Контактна електронна пошта
phone	TEXT	Контактний номер телефону
interview_at	TIMESTAMPTZ	Дата та час співбесіди
interview_format	TEXT	Формат проведення співбесіди
interview_location	TEXT	Місце проведення співбесіди
interview_link	TEXT	Посилання на онлайн-зустріч

Таблиця applications забезпечує реалізацію основного бізнес-процесу вебплатформи - подання кандидатом заявки на вакансію. Кожна заявка пов'язана

одночасно з вакансією, користувачем і вибраним резюме, що забезпечує цілісність інформації та можливість відстеження всіх етапів проходження відбору. Крім цього, таблиця містить інформацію щодо організації співбесіди, тому окрема таблиця для її зберігання не використовується.

Для забезпечення взаємодії між кандидатами та представниками компаній у системі використовується декілька таблиць повідомлень. Основною є application_messages, яка забезпечує обмін повідомленнями в межах конкретної заявки. Для реалізації додаткової комунікації використовуються таблиці company_candidate_messages та company_candidate_outreach, що дозволяють компаніям ініціювати контакт із кандидатами незалежно від наявності заявки, надсилати запрошення та узгоджувати проведення співбесіди. Структуру таблиці application_messages наведено в табл. 2.10.

Таблиця 2.10

Структура таблиці application_messages

Поле	Тип даних	Опис
id	UUID	Унікальний ідентифікатор повідомлення
application_id	UUID	Посилання на заявку
company_id	UUID	Ідентифікатор компанії
student_id	UUID	Ідентифікатор кандидата
sender_user_id	UUID	Відправник повідомлення
sender_role	TEXT	Роль відправника
subject	TEXT	Тема повідомлення
message	TEXT	Текст повідомлення
status	TEXT	Статус повідомлення
interview_at	TIMESTAMPTZ	Запланована дата співбесіди

Таблиця application_messages забезпечує зберігання історії листування між кандидатом і роботодавцем у межах конкретної заявки. Повідомлення можуть містити інформацію щодо уточнення деталей вакансії, запрошення на співбесіду, зміни статусу заявки або інших питань, пов'язаних із процесом працевлаштування. Збереження всієї історії комунікації дозволяє учасникам процесу переглядати попередні повідомлення та забезпечує прозорість взаємодії.

Для організації персонального пошуку вакансій у системі використовуються таблиці saved_jobs та disliked_jobs. Перша таблиця призначена для формування списку обраних вакансій, які користувач планує переглянути або розглянути пізніше. Друга використовується для збереження вакансій, які користувач більше не бажає бачити серед результатів пошуку. Такий підхід дозволяє персоналізувати процес пошуку роботи та зробити його більш зручним для кандидата. Структуру таблиці saved_jobs наведено в табл. 2.11.

Таблиця 2.11

Структура таблиці saved_jobs

Поле	**Тип даних**	**Опис**
id	UUID	Унікальний ідентифікатор запису
user_id	UUID	Посилання на користувача
job_id	UUID	Посилання на вакансію

Таблиця saved_jobs використовується для формування індивідуального списку обраних вакансій. Користувач може додавати цікаві вакансії до списку збережених і повертатися до них у будь-який момент без необхідності повторного пошуку. Це підвищує зручність роботи з платформою та дозволяє швидко отримувати доступ до вакансій, які зацікавили кандидата. Структуру таблиці disliked_jobs наведено в табл. 2.12.

Таблиця 2.12

Структура таблиці disliked_jobs

Поле	**Тип даних**	**Опис**
id	UUID	Унікальний ідентифікатор запису
user_id	UUID	Посилання на користувача
job_id	UUID	Посилання на вакансію

Таблиця disliked_jobs використовується для збереження вакансій, які кандидат більше не бажає переглядати. Інформація з цієї таблиці використовується під час формування результатів пошуку, що дозволяє не відображати користувачу вже переглянуті або нецікаві вакансії. Такий механізм сприяє персоналізації рекомендацій і покращує зручність користування вебплатформою.

Для контролю якості вакансій та забезпечення безпеки користувачів використовується таблиця job_reports. Вона призначена для зберігання скарг користувачів на вакансії, які можуть містити недостовірну інформацію, порушувати правила платформи або викликати підозру щодо шахрайства. Структуру таблиці job_reports наведено в табл. 2.13.

Таблиця 2.13

Структура таблиці job_reports

Поле	Тип даних	Опис
id	UUID	Унікальний ідентифікатор скарги
job_id	UUID	Посилання на вакансію
reporter_user_id	UUID	Користувач, який подав скаргу
reporter_role	TEXT	Роль користувача
reason	TEXT	Причина подання скарги
details	TEXT	Детальний опис проблеми
status	TEXT	Статус розгляду скарги
admin_note	TEXT	Коментар адміністратора
resolved_at	TIMESTAMPTZ	Дата завершення розгляду

Таблиця job_reports використовується адміністраторами для модерації вакансій. Після надходження скарги адміністратор може переглянути її зміст, проаналізувати зазначені причини, прийняти рішення щодо подальших дій та зафіксувати результат розгляду. Використання окремої таблиці забезпечує прозорість процесу модерації та дозволяє вести історію опрацювання звернень користувачів.

Для перевірки достовірності інформації про роботодавців у системі реалізовано окремий модуль перевірки компаній. До його складу входять таблиці company_registry**,** company_registry_records**,** company_verifications та fop_registry, які забезпечують зберігання відомостей про юридичних осіб, результати перевірок і службову інформацію щодо використаних відкритих державних реєстрів. Структуру таблиці company_registry наведено в табл. 2.14.

Таблиця 2.14

Структура таблиці company_registry

Поле	Тип даних	Опис
edrpou	TEXT	Код ЄДРПОУ (первинний ключ)
name	TEXT	Повна назва юридичної особи
short_name	TEXT	Скорочена назва
legal_form	TEXT	Організаційно-правова форма
status	TEXT	Статус компанії
address	TEXT	Юридична адреса
director	TEXT	Керівник
kved	TEXT	Основний вид економічної діяльності
source	TEXT	Джерело отримання інформації

Таблиця company_registry використовується для локального зберігання відомостей про юридичних осіб, отриманих із відкритих державних реєстрів. Наявність локальної копії інформації дозволяє зменшити кількість звернень до зовнішніх сервісів, прискорити процес перевірки роботодавців та підвищити стабільність роботи системи. Структуру таблиці company_verifications наведено в табл. 2.15.

Таблиця 2.15

Структура таблиці company_verifications

Поле	Тип даних	Опис
id	BIGINT	Унікальний ідентифікатор перевірки
user_id	UUID	Користувач, який проходить перевірку
company_name	TEXT	Назва компанії
registration_code	TEXT	Реєстраційний код
source	TEXT	Джерело перевірки
matched_name	TEXT	Назва, знайдена у реєстрі
status	TEXT	Результат перевірки
reason	TEXT	Причина відмови або підтвердження
checked_at	TIMESTAMPTZ	Дата проведення перевірки

Таблиця company_verifications містить результати автоматичної перевірки компаній під час реєстрації або оновлення профілю роботодавця. Збереження історії перевірок дозволяє адміністраторам контролювати процес підтвердження роботодавців та аналізувати результати виконаних перевірок.

Для реалізації інформаційного розділу вебплатформи використовується таблиця articles, призначена для зберігання інформаційних матеріалів. У ній містяться статті, рекомендації щодо пошуку роботи, створення резюме, проходження співбесід, професійного розвитку та інші публікації, які допомагають користувачам ефективніше користуватися можливостями платформи. Крім основного тексту статті, таблиця містить службові дані, необхідні для її публікації, категоризації та відображення на вебплатформі. Структуру таблиці articles наведено в табл. 2.16.

Таблиця 2.16

Структура таблиці articles

Поле	**Тип даних**	**Опис**
id	BIGINT	Унікальний ідентифікатор статті
author_user_id	UUID	Автор статті
title	TEXT	Заголовок статті
slug	TEXT	Унікальна адреса статті
excerpt	TEXT	Короткий опис
content	TEXT	Основний текст
category	TEXT	Категорія статті
tags	TEXT	Ключові слова
status	TEXT	Статус публікації
published_at	TIMESTAMPTZ	Дата публікації

Таблиця articles забезпечує функціонування інформаційного розділу вебплатформи. Завдяки їй користувачі можуть отримувати корисні матеріали щодо працевлаштування, розвитку кар'єри, підготовки резюме та проходження співбесід, а адміністратори мають можливість створювати, редагувати та публікувати нові інформаційні матеріали.

Окрім основних таблиць, база даних містить низку допоміжних сутностей, які забезпечують додатковий функціонал вебплатформи. До них належать таблиці company_candidate_messages, company_candidate_outreach, fop_registry та admin_profiles.

Таблиця company_candidate_messages призначена для зберігання повідомлень між представниками компаній і кандидатами. На відміну від таблиці

application_messages, яка використовується для листування в межах конкретної заявки, ця таблиця забезпечує можливість прямої комунікації між роботодавцем і кандидатом. У ній зберігається інформація про відправника, одержувача, тему повідомлення, текст повідомлення, статус доставки та дату його створення. Завдяки цьому забезпечується історія спілкування між сторонами та можливість повторного перегляду попередніх повідомлень.

Таблиця company_candidate_outreach використовується для реалізації механізму активного пошуку кандидатів роботодавцями. За допомогою цієї таблиці представники компаній можуть надсилати потенційним кандидатам пропозиції щодо працевлаштування незалежно від того, чи подавали вони заявку на вакансію. У таблиці зберігається інформація про вакансію, кандидата, тему повідомлення, текст запрошення та статус взаємодії.

Між таблицями бази даних реалізовано систему зв'язків, яка забезпечує цілісність даних і підтримує взаємодію між основними функціональними модулями вебплатформи. Основні зв'язки реалізовано за допомогою зовнішніх ключів, що забезпечують коректне поєднання взаємопов'язаних сутностей та підтримують референтну цілісність інформації. Структуру таблиці зв'язки між таблицями бази даних наведено в табл. 2.17.

Таблиця 2.17

Основні зв'язки між таблицями бази даних

Батьківська таблиця	**Дочірня таблиця**	**Поле зовнішнього ключа**	**Тип зв'язку**	**Опис**
company_profiles	jobs	company_id	1 : N	Одна компанія може створити багато вакансій
jobs	applications	job_id	1 : N	На одну вакансію може бути подано багато заявок
user_profiles	applications	student_id	1 : N	Один кандидат може подати багато заявок на різні вакансії
candidate_resumes	applications	resume_id	1 : N	Одне резюме може використовуватися під час подання кількох заявок

Продовження таблиці 2.17

Батьківська таблиця	**Дочірня таблиця**	**Поле зовнішнього ключа**	**Тип зв'язку**	**Опис**
jobs	resume_views	job_id	1 : N	Одна вакансія може бути пов'язана з багатьма переглядами резюме
company_profiles	resume_views	company_id	1 : N	Одна компанія може переглядати резюме багатьох кандидатів
user_profiles	resume_views	student_id	1 : N	Одне резюме кандидата може переглядатися різними роботодавцями

Наведені зв'язки забезпечують взаємодію між основними модулями вебплатформи. Кожна вакансія належить певній компанії, кожна заявка пов'язана з вакансією, кандидатом та резюме, а інформація про перегляди резюме дозволяє фіксувати активність роботодавців. Використання зовнішніх ключів гарантує узгодженість інформації між таблицями, запобігає появі записів із посиланнями на неіснуючі об'єкти та забезпечує коректну роботу системи під час виконання операцій додавання, оновлення й видалення даних.

Спроєктована структура бази даних повністю забезпечує реалізацію функціональних можливостей вебплатформи для пошуку роботи та стажування. Вона підтримує зберігання інформації про користувачів, компанії, вакансії, резюме, заявки, повідомлення, перегляди, статті та інші дані, необхідні для функціонування системи. Використання реляційної моделі даних, механізмів зовнішніх ключів, індексування та сучасних типів даних PostgreSQL забезпечує цілісність інформації, високу продуктивність роботи та створює основу для подальшого розвитку й масштабування вебплатформи.

2.3. Проєктування та реалізація алгоритмів роботи системи

Проєктування алгоритмів роботи системи є одним із ключових етапів розроблення вебплатформи для пошуку роботи та стажування, оскільки саме алгоритми визначають порядок взаємодії користувачів із системою, обробку введених даних, перевірку прав доступу, виконання основних бізнес-процесів, пошук вакансій, створення резюме, подання заявок, керування вакансіями та

адміністрування контенту. Від коректності побудови цих алгоритмів залежить стабільність роботи платформи, зручність користування нею та безпека обробки даних.

Розроблена система орієнтована на три основні категорії користувачів: кандидатів, представників компаній та адміністратора. Для кожної ролі передбачено окремі сценарії використання, набір доступних функцій і відповідні обмеження доступу. Кандидат може створювати профіль, формувати резюме, переглядати вакансії, зберігати зацікавлені пропозиції та подавати заявки.

Представник компанії має змогу створювати профіль компанії, публікувати вакансії, переглядати заявки кандидатів, змінювати їхні статуси та самостійно здійснювати пошук резюме. Адміністратор відповідає за контроль роботи платформи, керування користувачами, компаніями, вакансіями, заявками, резюме та інформаційними матеріалами.

Основним принципом проєктування алгоритмів є розподіл відповідальності між клієнтською та серверною частинами. Клієнтська частина відповідає за відображення інтерфейсу, навігацію між сторінками, збір даних із форм, первинну перевірку введеної інформації, відображення повідомлень користувачу та надсилання HTTP-запитів до серверного API. Серверна частина виконує автентифікацію, авторизацію, перевірку ролей користувачів, обробку бізнес-логіки, взаємодію з базою даних, роботу із зовнішніми сервісами та формування відповіді у форматі JSON. Такий підхід забезпечує чіткий поділ логіки відображення та логіки обробки даних, що спрощує підтримку й подальше розширення системи.

Загальний алгоритм взаємодії користувача із системою побудований за клієнт-серверною моделлю. Користувач виконує певну дію в інтерфейсі вебзастосунку, наприклад заповнює форму, відкриває сторінку вакансії або натискає кнопку подання заявки. Після цього клієнтська частина формує HTTP-запит до серверного API. Якщо дія потребує авторизації, до запиту додається токен доступу. Сервер перевіряє дійсність токена, визначає користувача та його роль, виконує необхідні перевірки прав доступу, звертається до бази даних і

повертає результат клієнтській частині. Після отримання відповіді інтерфейс оновлюється, а користувач отримує повідомлення про успішне виконання дії або причину помилки.

Алгоритм реєстрації користувача починається з вибору ролі: кандидат або компанія. Залежно від обраної ролі система відображає відповідну форму реєстрації. Кандидат вводить ім’я, прізвище, електронну пошту, пароль та інші необхідні дані. Представник компанії вказує назву організації, опис діяльності, сайт, контактну особу, електронну пошту, пароль та реєстраційні дані компанії. На клієнтській частині виконується первинна перевірка: наявність обов’язкових полів, коректність формату електронної пошти, мінімальна довжина пароля та правильність заповнення основних даних.

Якщо введені дані є некоректними, система відображає користувачу відповідне повідомлення про помилку. Якщо перевірка успішна, дані надсилаються на сервер. На серверній частині перевіряється унікальність електронної пошти, створюється обліковий запис користувача, формується відповідний профіль у базі даних і користувачу надсилається лист для підтвердження електронної пошти. Такий алгоритм дозволяє уникнути дублювання облікових записів і забезпечує базову перевірку достовірності користувача.

Алгоритм автентифікації використовується для підтвердження особи користувача, а алгоритм авторизації - для перевірки його прав доступу до певних функцій системи. Під час входу користувач вводить електронну пошту та пароль. Система передає ці дані до сервісу автентифікації, який перевіряє їхню коректність. У разі успішного входу створюється сесія користувача та формується токен доступу. Надалі цей токен використовується для виконання захищених запитів до серверного API [19].

Після отримання токена система визначає роль користувача та перенаправляє його до відповідного розділу [22]. Кандидат отримує доступ до пошуку вакансій, особистого кабінету, розділу резюме, збережених вакансій і заявок. Представник компанії переходить до панелі керування профілем

компанії, вакансіями, заявками та пошуком кандидатів. Адміністратор отримує доступ до адміністративної панелі, у якій реалізовано керування користувачами, компаніями, вакансіями, резюме, заявками та статтями.

Під час звернення до захищених маршрутів система перевіряє наявність активної сесії та дійсність токена доступу. Якщо користувач не авторизований, він перенаправляється на сторінку входу або отримує повідомлення про необхідність авторизації. Якщо роль користувача не відповідає вимогам маршруту, доступ блокується. Наприклад, кандидат не може створювати вакансії від імені компанії, компанія не може керувати адміністративними даними, а неавторизований користувач не може подавати заявки. Такий алгоритм забезпечує розмежування прав доступу між різними типами користувачів.

Однією з основних функцій системи є пошук вакансій або стажувань. Користувач може переглядати список доступних пропозицій, застосовувати фільтри та відкривати детальну інформацію про конкретну вакансію. Пошук може виконуватися за ключовими словами, назвою посади, містом, форматом роботи, типом зайнятості, компанією, рівнем досвіду або напрямом діяльності. Це дає змогу кандидатам швидко знаходити найбільш релевантні пропозиції.

Алгоритм пошуку вакансій починається з відкриття сторінки зі списком пропозицій. Клієнтська частина надсилає запит до серверного API для отримання активних вакансій. Якщо користувач задає фільтри, ці параметри додаються до запиту. Сервер формує SQL-запит до бази даних з урахуванням переданих умов, відбирає відповідні записи та повертає список вакансій у форматі JSON. На стороні клієнта отримані дані відображаються у вигляді карток із короткою інформацією: назвою посади, компанією, містом, типом зайнятості, форматом роботи та коротким описом.

Після вибору конкретної вакансії користувач може перейти на сторінку її детального перегляду. У цьому випадку клієнтська частина надсилає запит із ідентифікатором вакансії. Сервер перевіряє існування відповідного запису в базі даних і повертає повну інформацію: опис вакансії, вимоги до кандидата, обов’язки, умови роботи, рівень досвіду, формат зайнятості, місце роботи та дані

про компанію. Це дозволяє кандидату оцінити відповідність вакансії власним навичкам і професійним інтересам.

Алгоритм подання заявки на вакансію доступний лише авторизованому користувачу з роллю кандидата. Після натискання кнопки подання заявки система перевіряє активну сесію, роль користувача, існування вакансії, її актуальність, наявність резюме кандидата та відсутність повторної заявки на цю саму вакансію. Якщо кандидат має декілька резюме, він може обрати потрібне резюме для конкретної заявки. Також користувач може додати супровідний лист.

Після успішної перевірки сервер створює новий запис у таблиці заявок зі статусом «на розгляді». У заявці зберігається інформація про кандидата, вакансію, компанію, обране резюме, супровідний лист, дату подання та поточний статус. Після створення заявки кандидат отримує повідомлення про успішне подання, а компанія бачить нову заявку у своєму особистому кабінеті. Якщо ж кандидат уже подавав заявку на цю вакансію, система повідомляє про неможливість повторного подання.

Для кандидатів реалізовано алгоритм створення та редагування резюме. Кандидат переходить до відповідного розділу особистого кабінету, заповнює інформацію про освіту, досвід роботи, професійні навички, знання мов, контактні дані, бажану посаду, портфоліо, сертифікати та інші відомості. Клієнтська частина виконує первинну перевірку заповнення форми, після чого дані надсилаються на сервер. Сервер перевіряє, чи користувач авторизований і чи має роль кандидата, після чого створює або оновлює запис резюме в базі даних.

У системі передбачена можливість створення кількох резюме одним кандидатом. Це дозволяє користувачу готувати окремі резюме для різних напрямів працевлаштування або різних типів вакансій. Одне з резюме може бути позначене як основне. Основне резюме може використовуватися за замовчуванням під час подання заявки, що спрощує процес взаємодії кандидата із системою.

Також передбачено алгоритм роботи із завантаженими файлами резюме. Кандидат може додати документ у підтримуваному форматі. Клієнтська частина

передає файл на сервер, де він обробляється за допомогою механізму завантаження файлів. Сервер перевіряє тип і розмір файлу, після чого зберігає його або використовує для подальшої обробки. Такий функціонал дає змогу кандидату доповнити структуроване резюме готовим документом.

Для покращення користувацького досвіду реалізовано алгоритм збереження вакансій. Якщо кандидат зацікавився певною пропозицією, але ще не готовий подавати заявку, він може додати її до списку обраних. Під час збереження сервер перевіряє авторизацію користувача, роль кандидата, існування вакансії та відсутність дубліката в списку збережених. Якщо запису ще немає, система створює новий запис у таблиці збережених вакансій. Надалі кандидат може переглядати обрані вакансії в особистому кабінеті та швидко повертатися до них.

Крім збереження вакансій, у системі передбачено можливість приховування нецікавих пропозицій. Цей алгоритм дозволяє кандидату прибрати з відображення вакансії, які не відповідають його інтересам. Завдяки цьому користувач може персоналізувати список пропозицій і зосередитися на найбільш релевантних варіантах.

Для представників компаній реалізовано алгоритм створення та редагування профілю компанії. Після реєстрації компанія переходить до особистого кабінету та заповнює дані про організацію: назву, опис діяльності, сайт, контактні дані, місцезнаходження, логотип, код ЄДРПОУ та інші реквізити. Клієнтська частина перевіряє обов’язкові поля, після чого сервер зберігає або оновлює профіль компанії в базі даних. Цей профіль надалі відображається разом із вакансіями та використовується для представлення роботодавця кандидатам.

Окремим процесом є перевірка компанії. Після створення профілю компанія може отримати статус «очікує перевірки». У системі передбачено використання реєстраційних даних, зокрема коду ЄДРПОУ, для перевірки інформації про компанію. Перевірка може виконуватися на основі відкритих державних даних України. Після аналізу даних компанії її статус може бути

змінено на підтверджений або відхилений. Такий алгоритм дозволяє підвищити довіру до роботодавців і зменшити ризик появи недостовірних вакансій [20].

Після створення профілю компанія отримує можливість публікувати вакансії. Алгоритм створення вакансії починається із заповнення форми, у якій представник компанії вказує назву посади, опис, вимоги до кандидата, обов'язки, умови роботи, місто, формат роботи, тип зайнятості, рівень досвіду та інші характеристики. Після надсилання форми сервер перевіряє авторизацію користувача, його роль, належність до відповідної компанії та коректність введених даних. Якщо всі умови виконано, створюється новий запис вакансії в базі даних.

Компанія також може редагувати, публікувати, приховувати або закривати власні вакансії. Під час редагування сервер перевіряє, чи належить вакансія саме цій компанії. Це не дозволяє одному роботодавцю змінювати записи іншої компанії. Після успішної перевірки дані вакансії оновлюються в базі даних, а зміни стають доступними користувачам платформи.

Алгоритм перегляду заявок компанією реалізовано через особистий кабінет роботодавця. Система завантажує лише ті заявки, які належать до вакансій конкретної компанії. Для кожної заявки відображаються дані кандидата, обране резюме, супровідний лист, дата подання, вакансія та поточний статус. Представник компанії може переглянути детальну інформацію про кандидата й ухвалити рішення щодо подальшого розгляду.

Після аналізу заявки компанія може змінити її статус. Наприклад, заявка може отримати статус «переглянута», «запрошено на співбесіду», «відхилено» або «прийнято». Під час зміни статусу сервер перевіряє, чи має користувач право керувати цією заявкою. Після успішного оновлення новий статус зберігається в базі даних і стає доступним як для компанії, так і для кандидата. Це забезпечує прозорість процесу розгляду заявок.

Крім роботи із заявками, компанія має доступ до загального списку резюме, розміщених у системі. Алгоритм пошуку кандидатів дозволяє роботодавцю переглядати резюме, аналізувати навички, досвід, освіту, знання мов та інші

професійні дані кандидатів. Такий функціонал підтримує активний пошук працівників або стажерів з боку роботодавця й розширює можливості платформи порівняно з класичною моделлю, коли лише кандидат подає заявку на вакансію.

Для організації співбесід у системі передбачено алгоритм інтеграції з Google Calendar і Google Meet. Після вибору кандидата компанія може створити подію для співбесіди, указавши дату, час, учасників і додаткову інформацію. Серверна частина за допомогою Google Calendar API створює подію в календарі та може сформувати посилання на відеоконференцію Google Meet. Після створення події інформація про зустріч може бути збережена в системі й використана для подальшої комунікації між кандидатом і роботодавцем [23].

Адміністратор відповідає за контроль роботи платформи та керування її основними сутностями. Алгоритм доступу до адміністративної панелі передбачає обов'язкову перевірку автентифікації та адміністративної ролі користувача. Якщо користувач не має відповідних прав, доступ до адміністративних функцій блокується. Це дозволяє захистити системні дані від несанкціонованого редагування.

В адміністративній панелі адміністратор може переглядати користувачів, компанії, вакансії, резюме, заявки та інформаційні матеріали. Алгоритми адміністрування передбачають можливість зміни статусів, модерування опублікованої інформації, перевірки компаній, контролю вакансій і керування контентом. Наприклад, адміністратор може підтвердити або відхилити компанію, перевірити коректність вакансії, переглянути інформацію про користувача або керувати статтями.

Окремо реалізовано алгоритм роботи зі статтями та інформаційними матеріалами. Адміністратор може створювати, редагувати, публікувати або видаляти статті, присвячені пошуку роботи, підготовці резюме, проходженню співбесід, розвитку професійних навичок і кар'єрному зростанню. Користувачі можуть переглядати ці матеріали в публічній частині платформи. Завдяки цьому система виконує не лише функцію пошуку вакансій, а й роль інформаційного ресурсу для кандидатів.

Усі алгоритми системи передбачають обробку помилок. Якщо користувач вводить некоректні дані, система відображає повідомлення про помилку ще на клієнтській частині або після відповіді сервера. Якщо сервер не знаходить потрібний запис, користувач отримує відповідне повідомлення. Якщо дія заборонена через недостатні права доступу, система блокує її виконання. Такий підхід підвищує надійність роботи платформи та робить взаємодію з нею зрозумілішою для користувача.

Для оптимізації роботи окремих процесів може використовуватися кешування. Дані, які часто запитуються та не змінюються після кожного звернення, можуть тимчасово зберігатися в пам’яті сервера. Це зменшує кількість повторних звернень до бази даних або зовнішніх джерел, пришвидшує відповідь API та знижує навантаження на сервер.

Таким чином, реалізовані алгоритми забезпечують повний цикл роботи вебплатформи: від реєстрації користувача, авторизації та створення профілю до пошуку вакансій, формування резюме, подання заявок, керування вакансіями, перегляду кандидатів, організації співбесід і адміністрування системи. Запропонована логіка дозволяє чітко розмежувати права доступу між кандидатами, компаніями та адміністраторами, забезпечити коректну обробку даних, підвищити безпеку системи та створити зручний механізм взаємодії між усіма учасниками процесу працевлаштування.

2.4. Реалізація функціоналу вебплатформи для пошуку роботи або стажування

Функціонал вебплатформи для пошуку роботи та стажування було реалізовано з урахуванням потреб трьох основних груп користувачів: кандидатів, роботодавців та адміністратора системи. Основною метою розробки є створення єдиного вебсередовища, у якому кандидати можуть знаходити актуальні вакансії та стажування, створювати й редагувати власні резюме, подавати заявки на обрані пропозиції, зберігати цікаві вакансії та керувати своїм профілем. Роботодавці, зі свого боку, отримують можливість створювати профілі компаній, публікувати вакансії, переглядати заявки кандидатів, аналізувати резюме та

організовувати співбесіди. Адміністратор системи забезпечує контроль за користувачами, компаніями, вакансіями, заявками, резюме та інформаційними матеріалами платформи.

Система побудована за клієнт-серверною архітектурою, що дозволяє розділити логіку відображення інтерфейсу та логіку обробки даних. Клієнтська частина відповідає за взаємодію з користувачем, відображення сторінок, роботу форм, навігацію, перегляд вакансій, резюме, заявок, профілів компаній та адміністративних розділів. Серверна частина приймає HTTP-запити від клієнта, перевіряє права доступу, виконує бізнес-логіку, взаємодіє з базою даних, зовнішніми сервісами та повертає результат у форматі JSON. Такий підхід спрощує підтримку системи, дає змогу окремо розвивати frontend і backend, а також полегшує масштабування платформи в майбутньому .

Клієнтська частина реалізована за допомогою бібліотеки React [8].Вона використовується для побудови компонентного інтерфейсу, у якому окремі частини сторінок винесені в повторно використовувані компоненти. Для запуску та збирання frontend-частини застосовується Vite [9], що забезпечує швидку розробку, миттєве оновлення сторінки під час змін у коді та оптимізовану production-збірку.

Для маршрутизації використовується React Router DOM. Завдяки цьому в системі реалізовано переходи між головною сторінкою, сторінками авторизації та реєстрації, сторінками вакансій, профілем кандидата, кабінетом компанії, сторінками створення та редагування вакансій, переглядом резюме, заявками та адміністративною панеллю.

У frontend-частині передбачено розмежування доступу до сторінок залежно від ролі користувача. Неавторизовані користувачі можуть переглядати публічні сторінки, список вакансій, сторінки окремих вакансій, інформаційні матеріали та сторінки авторизації. Кандидати отримують доступ до особистого кабінету, профілю, резюме, збережених вакансій та заявок. Компанії мають власний кабінет для керування профілем роботодавця, вакансіями, заявками кандидатів і пошуком резюме. Адміністратор має доступ до окремих адміністративних

сторінок, де може контролювати користувачів, компанії, вакансії, резюме, заявки та статті.

Для оформлення інтерфейсу використовуються CSS-стилі [10], а також бібліотека Lucide React, яка надає набір іконок для кнопок, меню, карток вакансій, блоків статистики, форм та інших елементів інтерфейсу. Це покращує візуальне сприйняття системи, робить сторінки зрозумілішими та зручнішими для користувачів. Інтерфейс побудований таким чином, щоб користувач міг швидко знайти потрібну дію: переглянути вакансії, створити резюме, подати заявку, перейти до кабінету або виконати адміністративну дію [29].

Серверна частина реалізована на платформі Node.js [11] із використанням фреймворку Express.js [12]. Express.js використовується для створення API-маршрутів, підключення middleware, обробки запитів, перевірки авторизації, формування відповідей та обробки помилок. Серверна частина виконує основну бізнес-логіку системи: реєстрацію користувачів, перевірку токенів, роботу з профілями, вакансіями, резюме, заявками, компаніями, адміністративними функціями, файлами, зовнішніми API та базою даних.

Для зберігання конфігураційних параметрів використовується бібліотека dotenv. Вона дає змогу винести службові дані у змінні середовища: параметри підключення до бази даних, ключі Supabase, налаштування SMTP, Google OAuth, адреси клієнтської частини та інші параметри. Такий підхід дозволяє не зберігати конфіденційні дані безпосередньо в коді та спрощує налаштування системи для різних середовищ: локальної розробки, тестування та production.

Основною базою даних системи є PostgreSQL [25] . Підключення до неї на серверній частині реалізовано за допомогою бібліотеки pg. Через SQL-запити сервер створює, читає, оновлює та видаляє дані, пов’язані з користувачами, профілями, компаніями, вакансіями, резюме, заявками, збереженими вакансіями, статтями та іншими сутностями. PostgreSQL забезпечує надійне зберігання структурованих даних і дозволяє будувати складні запити для фільтрації, пошуку та аналітики.

У системі також використовується Supabase [13]. Він застосовується для автентифікації користувачів, роботи з обліковими записами, перевірки токенів доступу та взаємодії з даними. Supabase Authentication дозволяє реалізувати реєстрацію, вхід, підтвердження електронної пошти, відновлення доступу та авторизацію через зовнішніх провайдерів. Серверна частина перевіряє токени користувачів, визначає їхню особу та роль, після чого дозволяє або забороняє доступ до відповідних API-маршрутів.

Для підвищення рівня безпеки серверної частини використовується helmet, який додає захисні HTTP-заголовки. Бібліотека cors застосовується для налаштування доступу до API з боку клієнтської частини та контролю дозволених джерел запитів. Для обмеження кількості запитів використовується express-rate-limit. Це допомагає зменшити ризик надмірного навантаження на сервер, автоматизованих атак і повторюваних запитів до чутливих маршрутів.

У системі реалізовано модуль реєстрації, авторизації та підтвердження користувачів. Користувач може створити обліковий запис за допомогою електронної пошти та пароля. Після реєстрації на вказану електронну адресу надсилається лист із посиланням для підтвердження пошти. Це дозволяє переконатися, що користувач має доступ до вказаної електронної адреси, і зменшує ризик створення фейкових облікових записів.

Окрім стандартної авторизації, у платформі реалізовано вхід через Google за допомогою протоколу OAuth 2.0 [19]. Користувач може обрати авторизацію через Google, після чого його буде перенаправлено на сторінку входу Google. Після успішного підтвердження система отримує необхідні дані профілю та створює або оновлює обліковий запис користувача на платформі. Такий спосіб входу спрощує процес авторизації та робить платформу зручнішою для користувачів [28].

Після успішної авторизації користувач отримує токен доступу, який використовується для виконання захищених запитів до серверного API. Сервер перевіряє цей токен, визначає користувача, його профіль і роль у системі. У платформі передбачено три основні ролі: кандидат, компанія та адміністратор.

Завдяки цьому реалізовано розмежування доступу до функцій системи. Наприклад, кандидат може подавати заявки та створювати резюме, компанія може публікувати вакансії та переглядати кандидатів, а адміністратор може модерувати контент і керувати системними даними.

Для кандидатів реалізовано функціонал створення та редагування особистого профілю. У профілі користувач може вказати персональні дані, контактну інформацію, місто проживання, освіту, професійні навички, досвід роботи, бажану сферу діяльності та інші відомості. Ці дані формують професійне представлення кандидата в системі та можуть використовуватися під час подання заявок або перегляду профілю роботодавцем.

Окремим важливим модулем є робота з резюме. Кандидат може створювати декілька резюме для різних напрямів працевлаштування. У резюме можуть зазначатися бажана посада, короткий опис професійного досвіду, освіта, навички, знання мов, сертифікати, приклади виконаних робіт, посилання на портфоліо, контактні дані та файл резюме. Також передбачено можливість позначити одне резюме як основне. Основне резюме може використовуватися за замовчуванням під час подання заявки на вакансію.

Для роботи з файлами резюме та документами в проєкті використовуються спеціалізовані бібліотеки. Зокрема, pdf-parse і pdfjs-dist можуть застосовуватися для обробки PDF-документів, а mammoth - для роботи з DOCX-файлами. Для завантаження файлів використовується multer, який дозволяє приймати файли на сервері та обробляти їх перед збереженням або подальшим аналізом. Для роботи з архівами використовуються adm-zip та unzipper. Це дає змогу платформі обробляти документи, які кандидати додають до своїх резюме або які використовуються в інших частинах системи.

Модуль перегляду вакансій дозволяє кандидатам і неавторизованим користувачам переглядати список доступних пропозицій роботи або стажування. На сторінці вакансії відображаються назва посади, опис, вимоги до кандидата, тип зайнятості, рівень досвіду, формат роботи, місце роботи, умови праці, інформація про компанію та інші важливі характеристики. Це дає змогу

кандидату швидко оцінити, чи відповідає вакансія його навичкам, досвіду та професійним інтересам.

Для зручності пошуку реалізовано фільтрацію вакансій за ключовими параметрами. Користувач може здійснювати пошук за назвою посади, ключовими словами, містом, типом зайнятості, рівнем досвіду, форматом роботи або іншими доступними критеріями. Після введення параметрів пошуку клієнтська частина формує запит до серверного API, а сервер повертає список вакансій, що відповідають заданим умовам. Завдяки цьому користувач може швидко знайти релевантні пропозиції серед великої кількості вакансій.

Однією з ключових функцій платформи є подання заявки на вакансію. Кандидат може обрати вакансію, додати резюме та супровідний лист. Перед створенням заявки сервер перевіряє, чи користувач авторизований, чи має він роль кандидата, чи належить обране резюме саме цьому користувачу, а також чи не подавав він уже заявку на цю саму вакансію. Якщо всі перевірки успішні, система створює запис заявки зі статусом очікування розгляду. Надалі заявка може бути переглянута роботодавцем, а її статус може змінюватися залежно від результатів розгляду.

Для кандидатів також реалізовано можливість збереження вакансій. Якщо користувач зацікавився певною пропозицією, але ще не готовий подавати заявку, він може додати її до списку обраних. Збережені вакансії доступні в особистому кабінеті кандидата, що дозволяє швидко повернутися до них пізніше. Цей функціонал підвищує зручність користування платформою та допомагає кандидатам планувати процес пошуку роботи.

Для представників компаній реалізовано окремий функціональний блок. Компанія може створити власний профіль, указавши назву організації, опис діяльності, контактні дані, вебсайт, місцезнаходження, логотип, реєстраційні дані та іншу інформацію. Профіль компанії використовується для представлення роботодавця кандидатам і відображається разом із вакансіями. Наявність детального профілю підвищує довіру кандидатів і дозволяє краще зрозуміти діяльність компанії.

Особливу увагу в системі приділено механізму підтвердження компаній. Під час реєстрації або заповнення профілю компанія може вказати код ЄДРПОУ та інші реєстраційні дані. У системі передбачено перевірку компаній за допомогою відкритих державних даних України, зокрема наборів Data.gov.ua [20]. Для обробки таких даних використовуються бібліотеки fast-xml-parser, xml2js та sax, які дозволяють читати й аналізувати XML-дані [21]. Для роботи з архівами державних наборів даних застосовуються adm-zip та unzipper. Після перевірки компанія може отримати статус підтвердженої, що підвищує довіру кандидатів до роботодавця та його вакансій.

Після створення профілю компанія отримує можливість публікувати вакансії. Представник компанії може створювати нові вакансії, редагувати вже опубліковані записи, оновлювати опис, вимоги, умови роботи, рівень досвіду, формат зайнятості, місце роботи або видаляти неактуальні пропозиції. Для кожної вакансії зберігається інформація про посаду, компанію, вимоги до кандидата, тип зайнятості, формат роботи, рівень досвіду, опис обов'язків та інші характеристики, необхідні для повного представлення вакансії.

Для роботодавців реалізовано модуль перегляду заявок. Компанія може бачити кандидатів, які відгукнулися на її вакансії, переглядати їхні резюме, супровідні листи, контактну інформацію та основні дані профілю. Після аналізу заявки представник компанії може змінити її статус відповідно до результатів розгляду. Це дозволяє компаніям організовано керувати процесом відбору кандидатів і відстежувати стан кожної заявки.

Крім роботи із заявками, компанія має можливість самостійно переглядати резюме кандидатів, розміщені в системі. Такий функціонал дозволяє роботодавцю активно шукати потенційних працівників або стажерів, аналізувати їхній досвід, навички, освіту, знання мов, портфоліо та інші професійні характеристики. Завдяки цьому платформа підтримує не лише пасивну модель, коли кандидат сам подає заявку, а й активну модель пошуку кандидатів з боку роботодавця.

Для організації співбесід у системі реалізовано інтеграцію з сервісами Google. На серверній частині використовується бібліотека googleapis, яка забезпечує доступ до Google Calendar API [23]. За допомогою цього API система може створювати події календаря, указувати дату й час співбесіди, додавати учасників, формувати опис зустрічі та зберігати інформацію про подію. Така інтеграція допомагає автоматизувати процес планування співбесід між кандидатами та роботодавцями.

Додатково використовується інтеграція з Google Meet [24]. Під час створення події для співбесіди система може сформувати посилання на відеоконференцію Google Meet і прив'язати його до події календаря. Це дозволяє кандидату та роботодавцю проводити дистанційні співбесіди безпосередньо за посиланням, створеним системою. Такий функціонал особливо корисний для віддалених вакансій, стажувань і компаній, які працюють з кандидатами з різних міст.

Для оптимізації роботи окремих запитів використовується node-cache. Ця бібліотека дозволяє тимчасово зберігати часто використовувані дані в пам'яті сервера. Завдяки кешуванню можна зменшити кількість повторних звернень до бази даних або зовнішніх джерел, прискорити відповідь API та знизити навантаження на сервер. Кешування є особливо корисним для даних, які часто читаються, але не змінюються після кожного запиту.

Для адміністраторів реалізовано адміністративний модуль. Адміністратор може переглядати користувачів, компанії, вакансії, резюме, заявки та інформаційні матеріали. Також адміністратор може контролювати якість опублікованої інформації, перевіряти або змінювати статуси компаній, модерувати вакансії, переглядати скарги або проблемні записи, працювати зі статтями та підтримувати стабільну роботу системи. Наявність адміністративної панелі дозволяє забезпечити контроль за контентом і дотриманням правил платформи.

Окремо реалізовано функціонал роботи зі статтями. У системі можуть публікуватися інформаційні матеріали щодо пошуку роботи, підготовки резюме,

проходження співбесід, розвитку професійних навичок, кар'єрного зростання та інших тем, корисних для кандидатів. Це робить платформу не лише сервісом для пошуку вакансій, а й інформаційним ресурсом, який допомагає користувачам краще підготуватися до працевлаштування.

У системі також передбачено роботу з електронною поштою. Для цього використовується nodemailer, який дозволяє надсилати службові повідомлення користувачам. Електронні листи можуть використовуватися для підтвердження реєстрації, системних повідомлень, сповіщень або інших процесів, пов'язаних із взаємодією між платформою та користувачем. Це забезпечує додатковий канал комунікації та підвищує зручність використання системи.

Під час розробки використовуються інструменти контролю якості коду. ESLint застосовується для перевірки JavaScript-коду [26] та виявлення потенційних помилок, невикористаних змінних, некоректного використання React Hooks та інших проблем. Для frontend-частини використовуються @vitejs/plugin-react, eslint-plugin-react-hooks та eslint-plugin-react-refresh. Для backend-частини під час локальної розробки застосовується nodemon, який автоматично перезапускає сервер після внесення змін у код. Це прискорює процес розробки та полегшує тестування змін.

У коді також реалізовано централізовану взаємодію frontend із backend API. Клієнтська частина використовує окремий сервісний шар для формування HTTP-запитів, додавання заголовків авторизації, передавання токена доступу, обробки JSON-відповідей і помилок. Такий підхід зменшує дублювання коду, спрощує підтримку API-викликів і дозволяє в одному місці змінювати логіку взаємодії з сервером.

З погляду архітектури, проєкт містить поділ на функціональні модулі. У frontend-частині окремо виділено функції авторизації, кандидатів, компаній, публічних сторінок, адміністративних сторінок, сервісів, контекстів і стилів. Така структура дозволяє групувати код відповідно до його призначення. У backend-частині також передбачено модульний підхід, пов'язаний із конфігурацією, middleware, маршрутами, роботою з базою даних, авторизацією та окремими

бізнес-сутностями. Це створює основу для подальшого рефакторингу та масштабування системи.

Таким чином, реалізований функціонал вебплатформи охоплює повний цикл взаємодії між кандидатами, роботодавцями та адміністраторами. Система підтримує реєстрацію користувачів, підтвердження електронної пошти, вхід через Google, створення профілів, формування кількох резюме, пошук і фільтрацію вакансій, перегляд детальної інформації про вакансії, подання заявок, збереження вакансій, створення профілів компаній, перевірку роботодавців за державними даними, публікацію та редагування вакансій, перегляд заявок, активний пошук кандидатів компаніями, організацію співбесід через Google Calendar і Google Meet, публікацію статей та адміністративне керування платформою.

Використання React, Vite, React Router DOM, Node.js, Express.js, PostgreSQL, Supabase, Google APIs, multer, nodemailer, node-cache, helmet, cors, express-rate-limit та інших бібліотек забезпечує гнучкість, функціональність і можливість подальшого розвитку системи. Проєкт має достатньо широку функціональну основу для вебплатформи з пошуку роботи та стажувань, а його архітектура дозволяє поступово розширювати можливості, покращувати безпеку, оптимізувати продуктивність і додавати нові модулі відповідно до потреб користувачів.

Висновки до другого розділу

У другому розділі було розглянуто проєктування вебплатформи для пошуку роботи та стажування. Було визначено основні типи користувачів системи: гість, кандидат, компанія та адміністратор, а також описано їхні можливості.

Гість може переглядати вакансії та зареєструватися. Кандидат може створювати профіль, додавати резюме, шукати вакансії, зберігати цікаві пропозиції та подавати заявки. Компанія має змогу створювати вакансії, переглядати заявки кандидатів і самостійно шукати резюме в системі. Адміністратор контролює роботу платформи, перевіряє компанії, вакансії та інформаційні матеріали.

У розділі було описано діаграму варіантів використання та діаграму класів, які показують взаємодію користувачів із системою та її основні об’єкти. Також було спроєктовано базу даних, у якій зберігаються дані про користувачів, компанії, вакансії, резюме, заявки та статті.

Окремо розглянуто основні алгоритми роботи системи: реєстрацію та авторизацію користувачів, пошук вакансій, подання заявок, створення вакансій, роботу з резюме, збереження вакансій і адміністрування платформи. Усі ці процеси працюють за клієнт-серверною моделлю.

Також було описано реалізацію функціоналу платформи. Для клієнтської частини використано React і Vite, для серверної Node.js та Express.js, а для зберігання даних PostgreSQL і Supabase. Додатково передбачено вхід через Google, підтвердження пошти, перевірку компаній та інтеграцію з Google Calendar і Google Meet.

Отже, у другому розділі було сформовано основу вебплатформи: визначено її структуру, базу даних, алгоритми роботи та основний функціонал. Запропоноване рішення дозволяє кандидатам зручно шукати вакансії та подавати заявки, а компаніям - публікувати вакансії, переглядати кандидатів і керувати процесом відбору.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБПЛАТФОРМИ ДЛЯ ПОШУКУ РОБОТИ ТА СТАЖУВАННЯ

3.1. Порядок встановлення та налаштування системи

Для забезпечення коректної роботи вебплатформи для пошуку роботи та стажування необхідно виконати налаштування програмного середовища, бази даних та зовнішніх сервісів, які використовуються в системі. Розроблена система побудована за клієнт-серверною архітектурою та складається з клієнтської частини, серверної частини, бази даних і зовнішніх сервісів автентифікації та інтеграції.

Клієнтська частина реалізована за допомогою бібліотеки React та інструмента Vite. Вона забезпечує взаємодію користувачів із системою через вебінтерфейс і відповідає за відображення сторінок, форм, вакансій, профілів компаній, резюме та інших елементів платформи. Frontend-частина є основним інтерфейсом взаємодії користувача з вебплатформою. Через неї користувачі переглядають вакансії, проходять реєстрацію та авторизацію, створюють профілі, працюють із резюме, подають заявки та керують власними даними.

Клієнтська частина побудована за компонентним принципом. Окремі елементи інтерфейсу, зокрема форми, кнопки, картки вакансій, сторінки профілю та списки заявок, реалізовані як незалежні компоненти. Це спрощує підтримку коду та подальше розширення функціональних можливостей системи.

Серверна частина реалізована на платформі Node.js із використанням фреймворку Express.js. Вона відповідає за обробку запитів від клієнтської частини, перевірку прав доступу користувачів, взаємодію з базою даних та реалізацію бізнес-логіки системи. Серверна частина забезпечує роботу з вакансіями, резюме, профілями користувачів, заявками та іншими даними, необхідними для функціонування платформи.

Для зберігання інформації використовується база даних PostgreSQL та сервіси Supabase. У базі даних зберігаються відомості про користувачів, компанії, вакансії, резюме, заявки, збережені вакансії та інформаційні матеріали. Перед початком експлуатації системи необхідно створити таблиці бази даних і

налаштувати зв’язки між ними відповідно до структури, описаної в попередньому розділі.

Важливою складовою системи є сервіс Supabase Authentication, який використовується для реєстрації користувачів, входу до системи, підтвердження електронної пошти та авторизації через Google. Для його налаштування необхідно створити проєкт у Supabase, активувати механізми автентифікації та налаштувати параметри підтвердження електронної пошти.

Для спрощення процедури входу реалізовано підтримку авторизації через Google. Після налаштування Google OAuth користувачі можуть входити до системи не лише за допомогою електронної пошти та пароля, а й через власний Google-акаунт. Це підвищує зручність використання платформи та скорочує час реєстрації нових користувачів.

У системі також реалізовано механізм перевірки компаній. Для цього використовуються відкриті державні дані України. Налаштування цього функціоналу передбачає забезпечення доступу до відповідних наборів даних, які використовуються для перевірки інформації про роботодавців. Це дозволяє підвищити достовірність даних про компанії та зменшити ризик появи недостовірних вакансій.

Для організації дистанційних співбесід платформа інтегрується із сервісами Google Calendar та Google Meet. Після налаштування відповідних сервісів система може автоматично створювати події календаря, призначати дату й час співбесіди, додавати учасників зустрічі та генерувати посилання для підключення до відеоконференції. Це спрощує планування співбесід між кандидатом і роботодавцем та забезпечує зручну комунікацію між учасниками відбору.

На рис. 3.1 зображено діаграму розгортання вебплатформи для пошуку роботи та стажування. Вона демонструє взаємодію між основними частинами системи: користувацьким пристроєм, frontend-частиною, backend-сервером, базою даних, сервісами Supabase, Google-сервісами та відкритими державними даними. Користувач працює із системою через браузер, frontend надсилає запити

до backend API, сервер обробляє їх, звертається до бази даних або зовнішніх сервісів і повертає результат клієнтській частині.

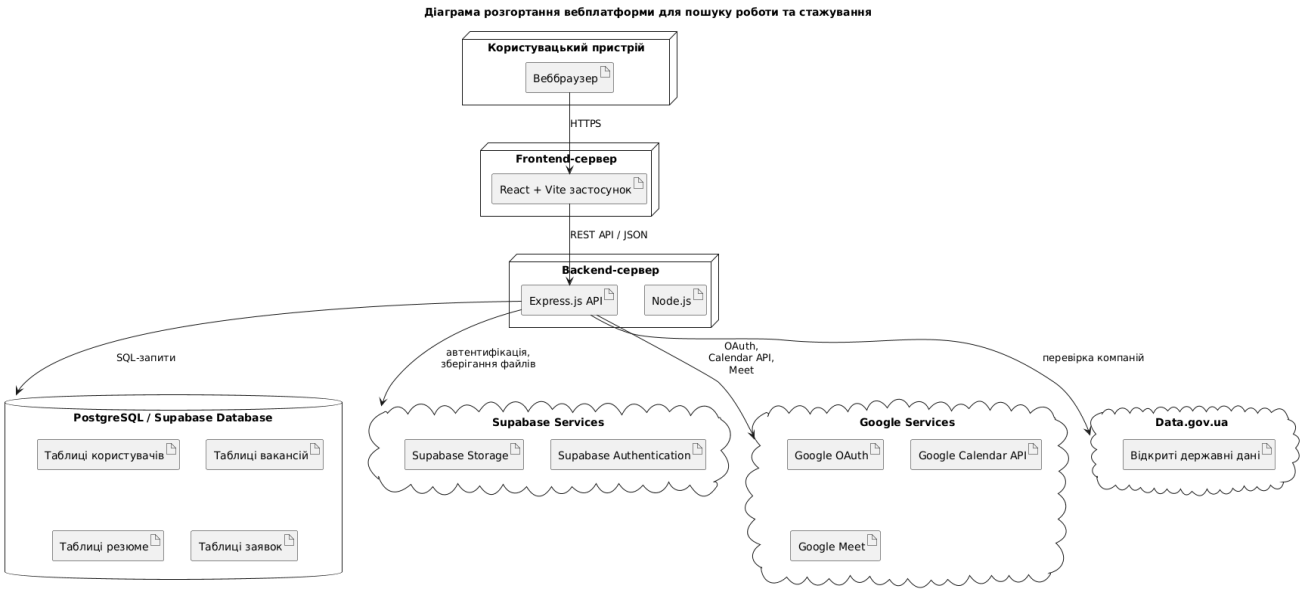


Рис.3.1. Діаграма розгортання вебплатформи для пошуку роботи та стажування

Після завершення налаштування системи необхідно перевірити працездатність її основних функціональних модулів. Насамперед виконується тестування реєстрації користувачів, підтвердження електронної пошти, стандартної авторизації та входу через Google. Також перевіряється створення профілю кандидата, додавання й редагування резюме, створення профілю компанії, публікація вакансій роботодавцями та подання заявок кандидатами.

Окремо необхідно перевірити коректність роботи пошуку вакансій, застосування фільтрів, перегляду детальної інформації про вакансії, збереження обраних пропозицій, перегляду резюме компаніями, зміни статусів заявок, організації співбесід і роботи адміністративної панелі. Важливо також переконатися, що система правильно розмежовує права доступу між кандидатами, компаніями та адміністратором.

У разі успішного проходження всіх перевірок система вважається готовою до використання. Отже, процес встановлення та налаштування вебплатформи включає підготовку клієнтської та серверної частин, налаштування бази даних, сервісів автентифікації, зовнішніх інтеграцій і перевірку основних сценаріїв

роботи. Після виконання цих дій система забезпечує повноцінну роботу всіх функціональних модулів і може бути передана в експлуатацію користувачам.

3.2. Структура інтерфейсу та порядок взаємодії з веб-сервісом

Для більш наочного представлення структури вебплатформи доцільно доповнити опис інтерфейсу основними зображеннями сторінок системи. Такі ілюстрації дозволяють продемонструвати зовнішній вигляд вебсервісу, логіку розміщення елементів, порядок взаємодії користувача з платформою та відмінності між інтерфейсами різних ролей.

Головну сторінку вебплатформи наведено на рис. 3.2. Вона є початковим екраном взаємодії користувача із системою та містить основні елементи навігації. На сторінці відображаються навігаційне меню, короткий опис призначення платформи, кнопки швидкого переходу до пошуку вакансій, авторизації або реєстрації, а також інформаційні блоки з актуальними пропозиціями, рекомендованими вакансіями та статтями. Завдяки такій структурі користувач може швидко ознайомитися з можливостями системи й перейти до потрібного розділу.

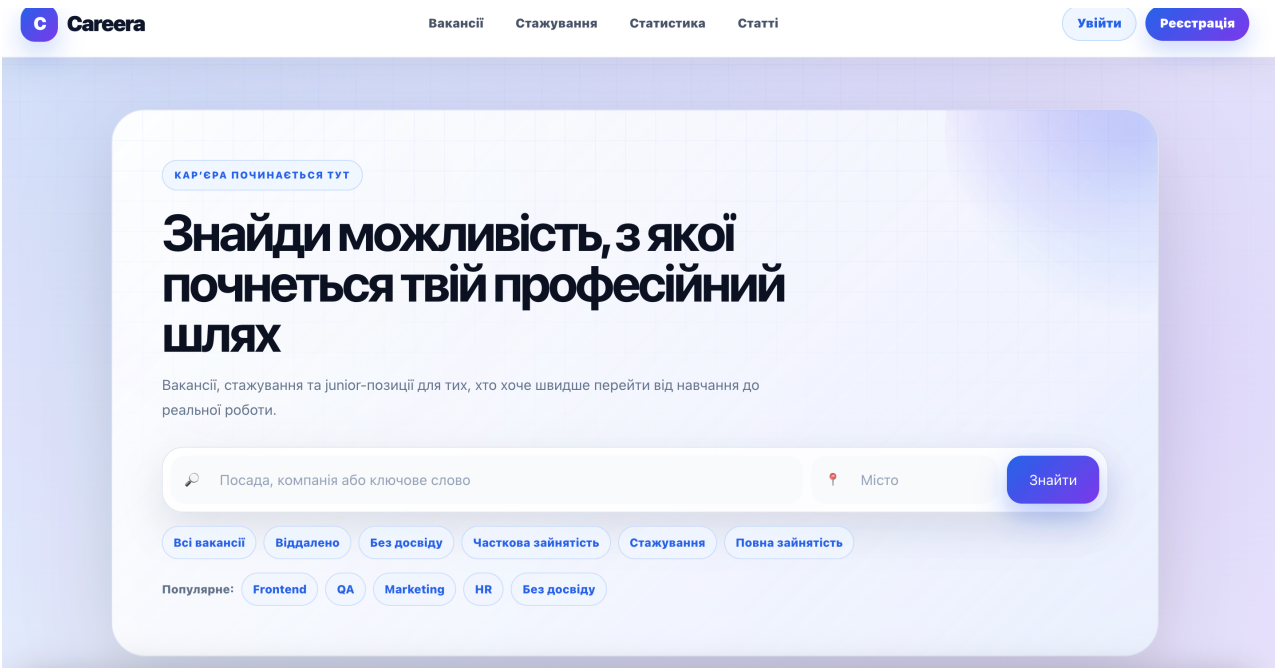


Рис. 3.2. Головна сторінка вебплатформи

Блок гарячих вакансій, розміщений на головній сторінці, наведено на рис. 3.3. У ньому відображаються найбільш актуальні або пріоритетні пропозиції для

кандидатів. Картки вакансій містять коротку інформацію про посаду, компанію, локацію, формат роботи та кнопку переходу до детального перегляду.

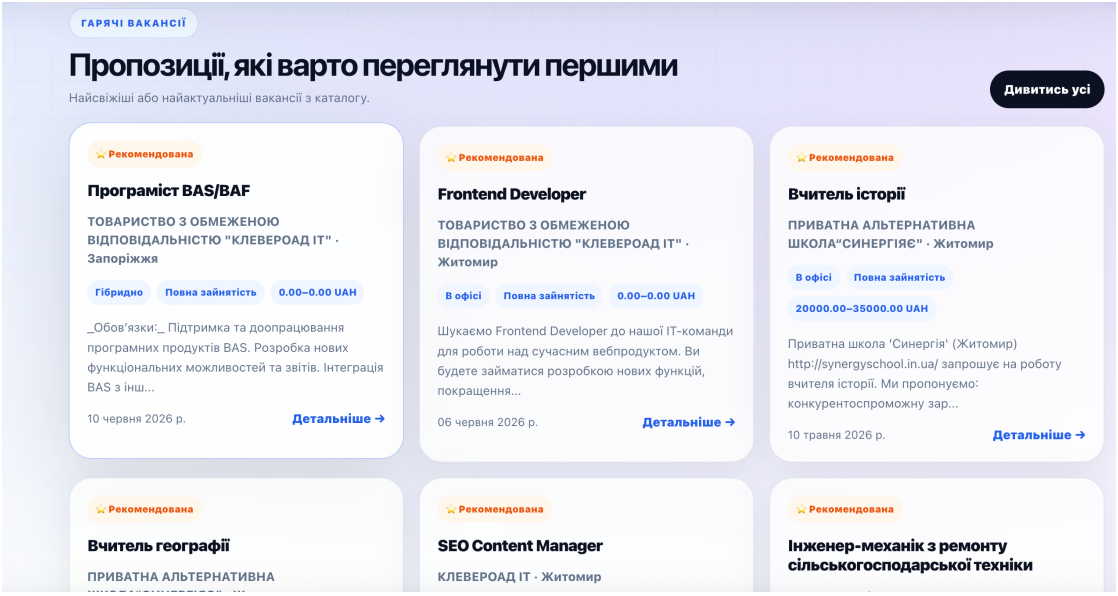


Рис. 3.3. Блок гарячих вакансій на головній сторінці

Блок статей на головній сторінці наведено на рис. 3.4. Він містить інформаційні матеріали щодо пошуку роботи, створення резюме, проходження співбесід і професійного розвитку. Цей блок показує, що платформа виконує не лише функцію пошуку вакансій, а й роль інформаційного ресурсу для кандидатів.

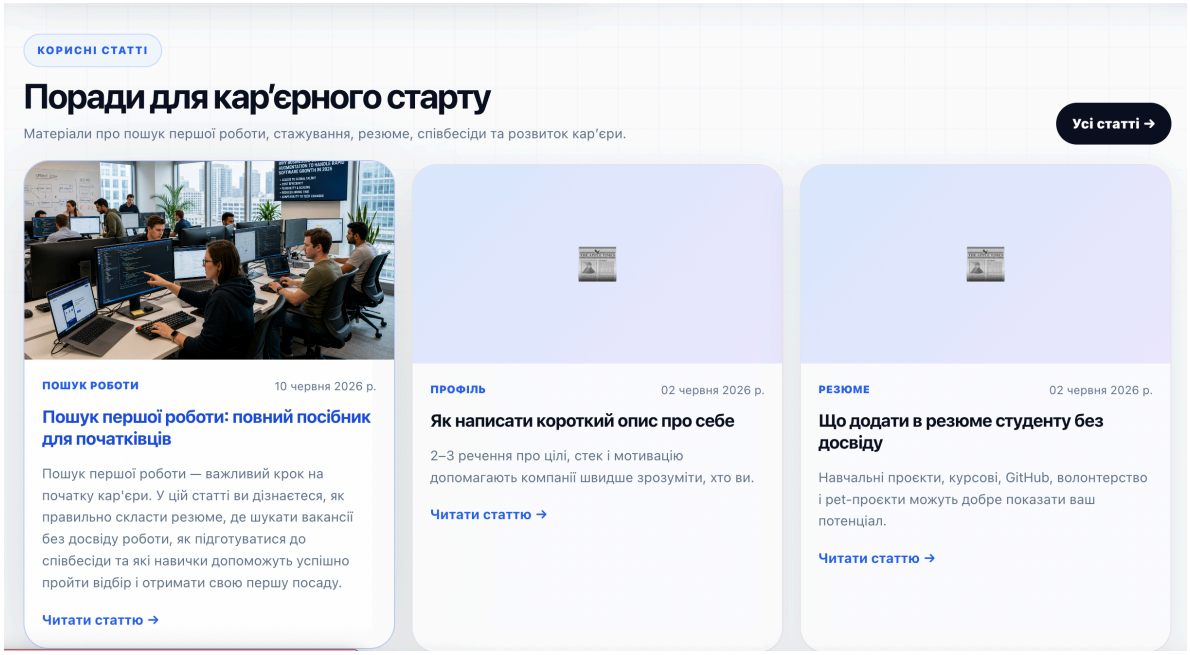


Рис. 3.4. Блок статей на головній сторінці

Сторінку авторизації користувача наведено на рис. 3.5. На ній розміщено поля для введення електронної пошти та пароля, кнопку входу, посилання для

реєстрації, можливість відновлення пароля та кнопку авторизації через Google. Ця сторінка забезпечує доступ до персоналізованих функцій системи.

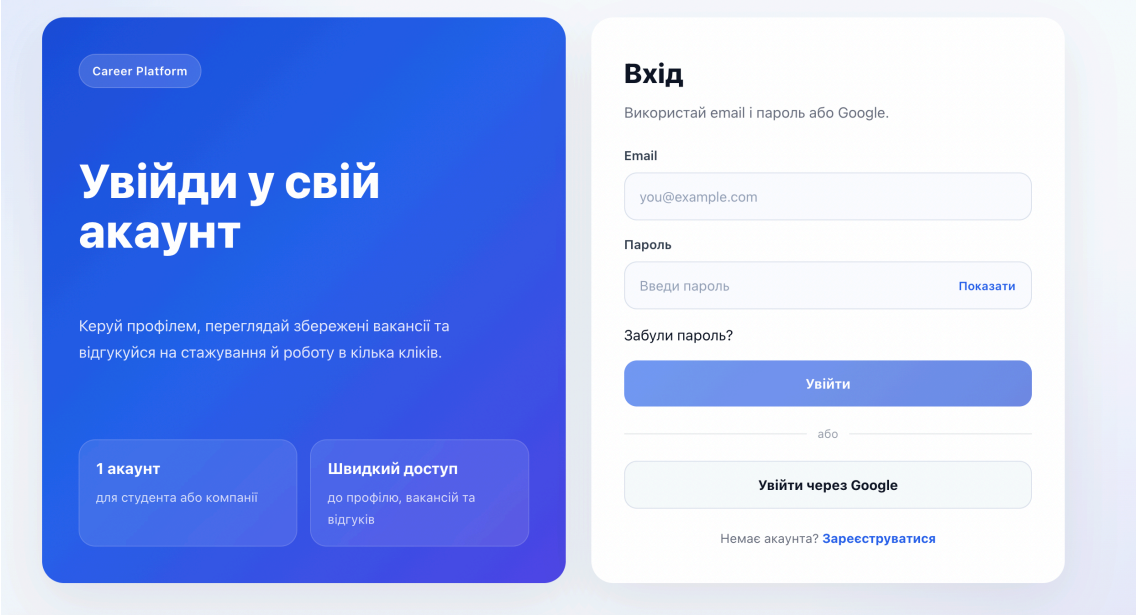


Рис. 3.5. Сторінка авторизації користувача

Сторінку реєстрації користувача наведено на рис. 3.6. На цій сторінці користувач обирає роль - кандидат або компанія. Залежно від вибору система відображає відповідні поля. Для кандидата це персональні дані, електронна пошта та пароль, а для компанії - назва організації, контактна особа, сайт, опис діяльності та інші реквізити.

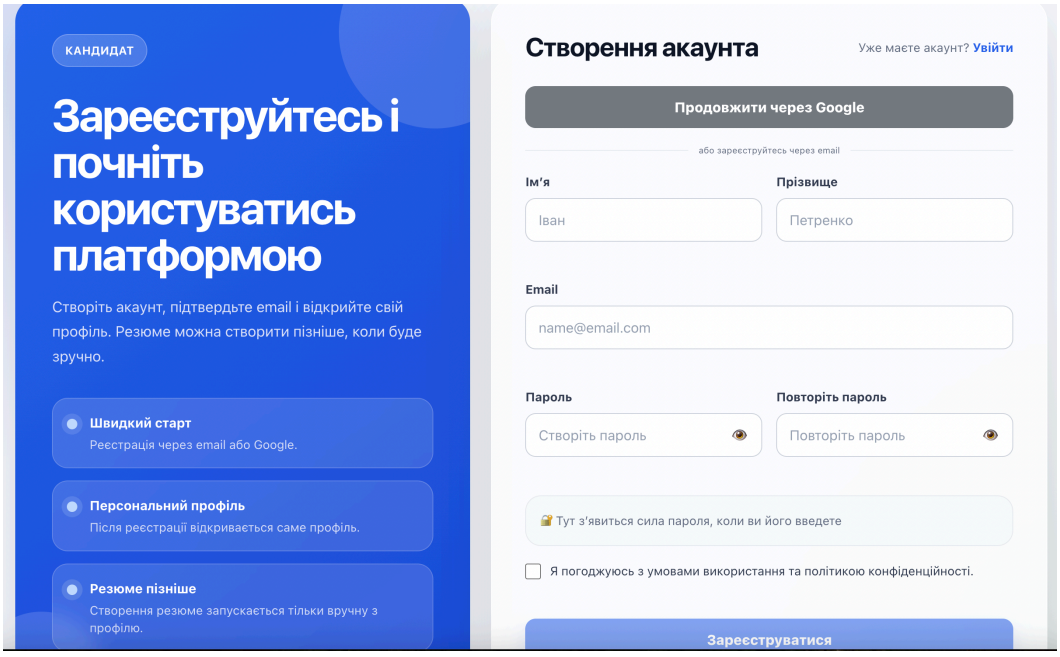


Рис. 3.6. Сторінка реєстрації користувача

Сторінку пошуку вакансій наведено на рис. 3.7. На ній відображається пошуковий рядок, фільтри, список вакансій і картки пропозицій. Користувач може виконувати пошук за ключовими словами, містом, типом зайнятості, форматом роботи, рівнем досвіду або компанією. Це одна з головних сторінок публічного інтерфейсу.

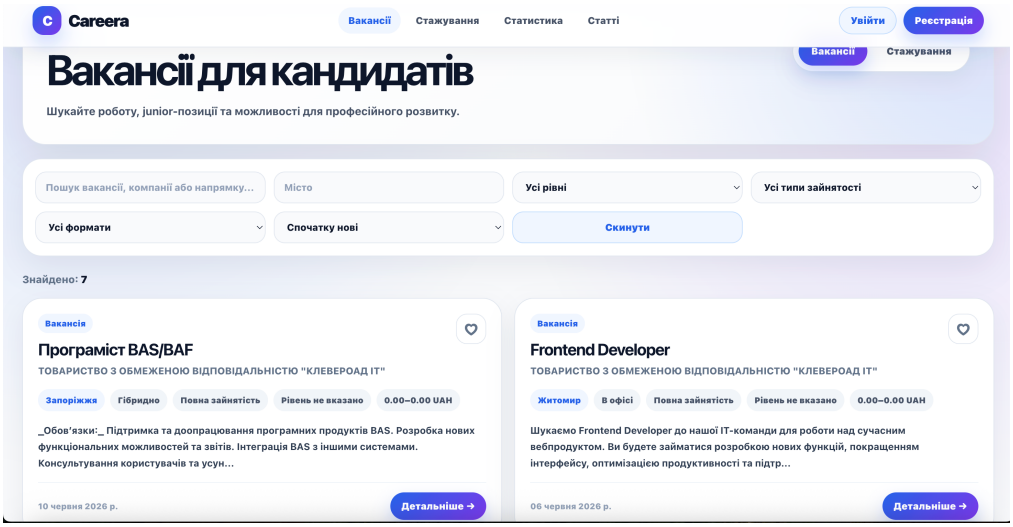


Рис. 3.7. Сторінка пошуку вакансій

Сторінку детального перегляду вакансії наведено на рис. 3.8. На ній відображається повний опис посади, вимоги до кандидата, умови роботи, інформація про компанію, а також кнопки дій: зберегти вакансію, подати заявку або повідомити про проблему. Ця сторінка є важливою для прийняття кандидатом рішення щодо подання заявки.

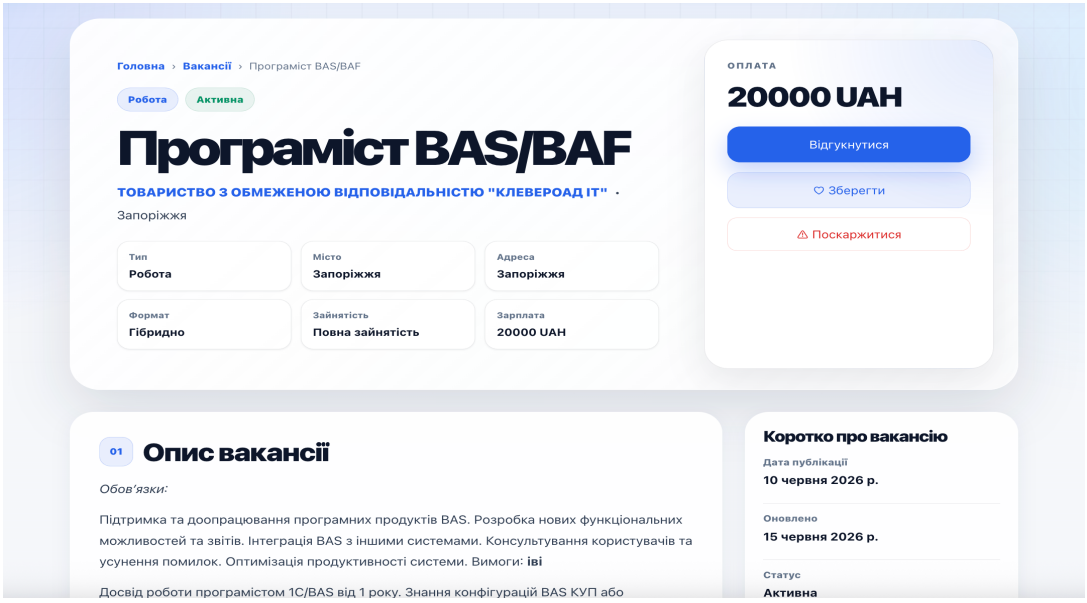


Рис. 3.8. Сторінка детального перегляду вакансії

Форму подання заявки на вакансію наведено на рис. 3.9. У ній кандидат може обрати резюме, додати супровідний лист і підтвердити надсилання заявки. Це зображення демонструє один із головних сценаріїв роботи кандидата з платформою.

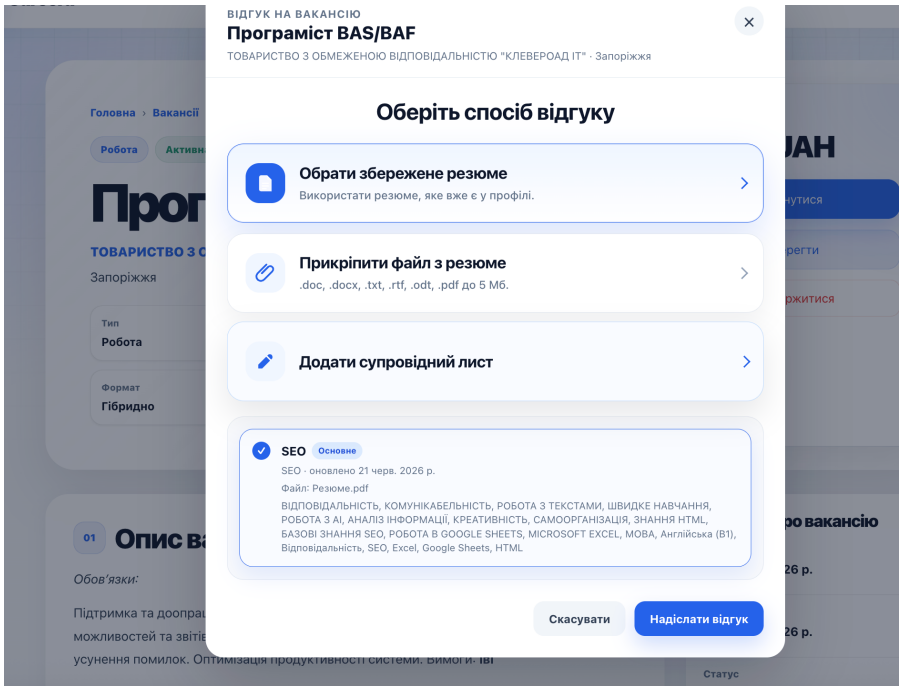


Рис. 3.9. Форма подання заявки на вакансію

Особистий кабінет кандидата наведено на рис. 3.10. У ньому користувач може переглядати та редагувати власний профіль, переходити до резюме, заявок і збережених вакансій. Особистий кабінет є центральним розділом для керування даними кандидата.

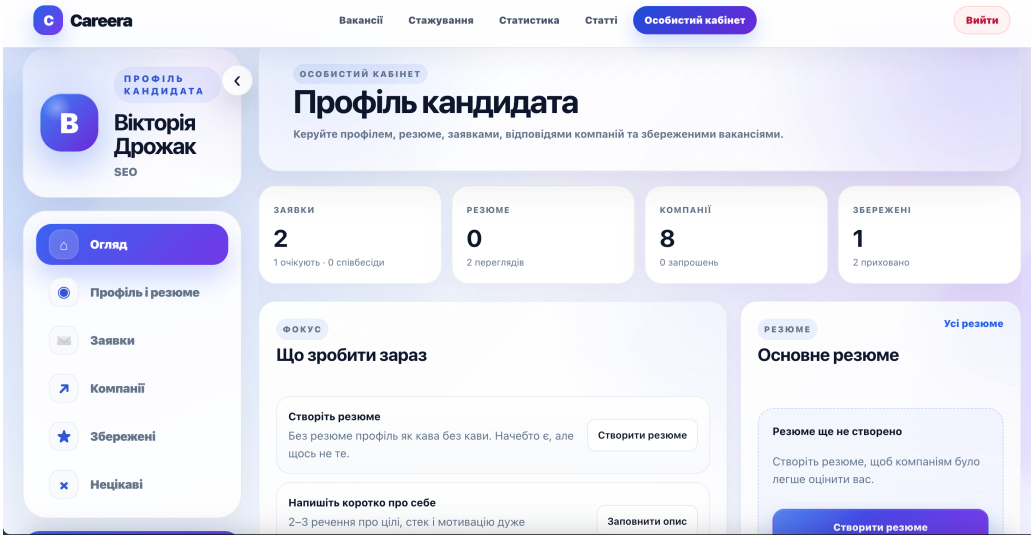


Рис. 3.10. Особистий кабінет кандидата

Окремим елементом інтерфейсу кандидата є сторінка створення резюме. Її наведено на рис. 3.11. На цій сторінці користувач може сформувати нове резюме, заповнивши основні персональні та професійні дані. Інтерфейс сторінки містить поля для введення бажаної посади, короткого опису про себе, контактної інформації, освіти, досвіду роботи, професійних навичок, знання мов, сертифікатів, посилань на портфоліо та додаткових матеріалів.

Також передбачене поле для завантаження файлу резюме. Після заповнення форми кандидат натискає кнопку збереження, після чого дані передаються на сервер і зберігаються в базі даних.

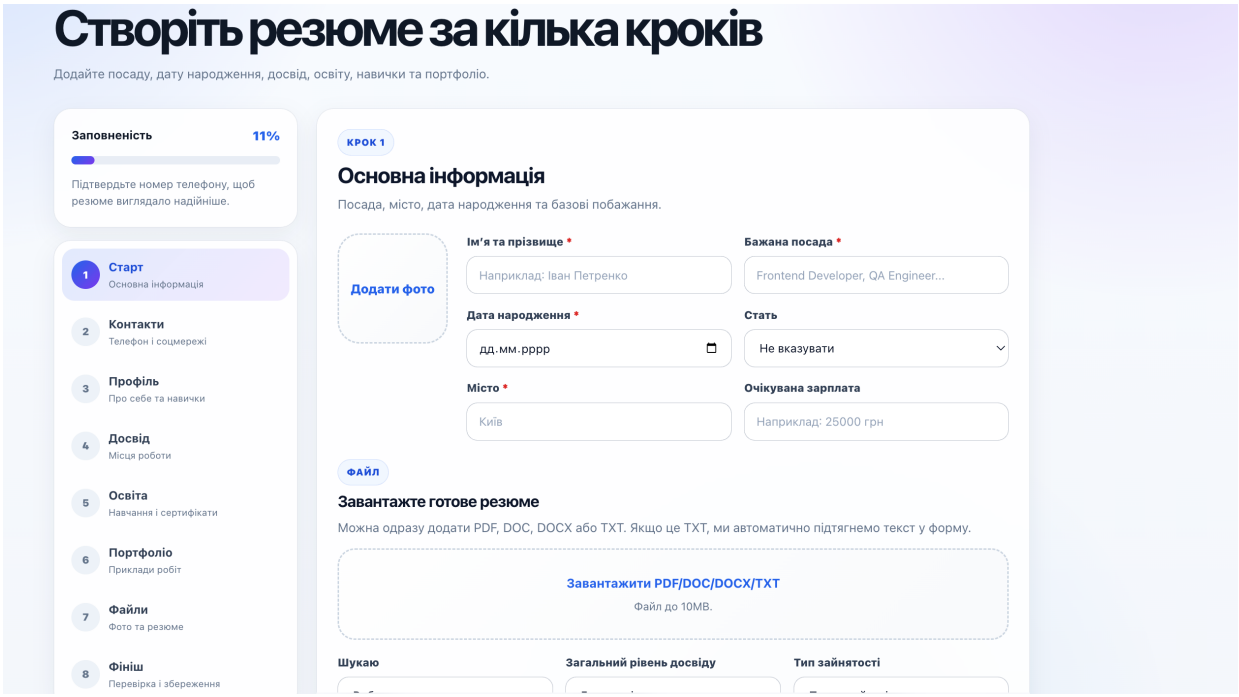


Рис. 3.11. Сторінка створення резюме

Сторінку керування резюме наведено на рис. 3.12. У цьому розділі кандидат може створювати нові резюме, переглядати вже додані, редагувати їх, видаляти неактуальні варіанти або позначати одне резюме як основне. Інтерфейс сторінки містить поля для зазначення бажаної посади, короткого опису про себе, освіти, досвіду роботи, професійних навичок, знання мов, сертифікатів, посилання на портфоліо та завантаження файлу резюме. Наявність такого розділу дозволяє кандидату зручно керувати власними професійними даними та використовувати підготовлене резюме під час подання заявок на вакансії.

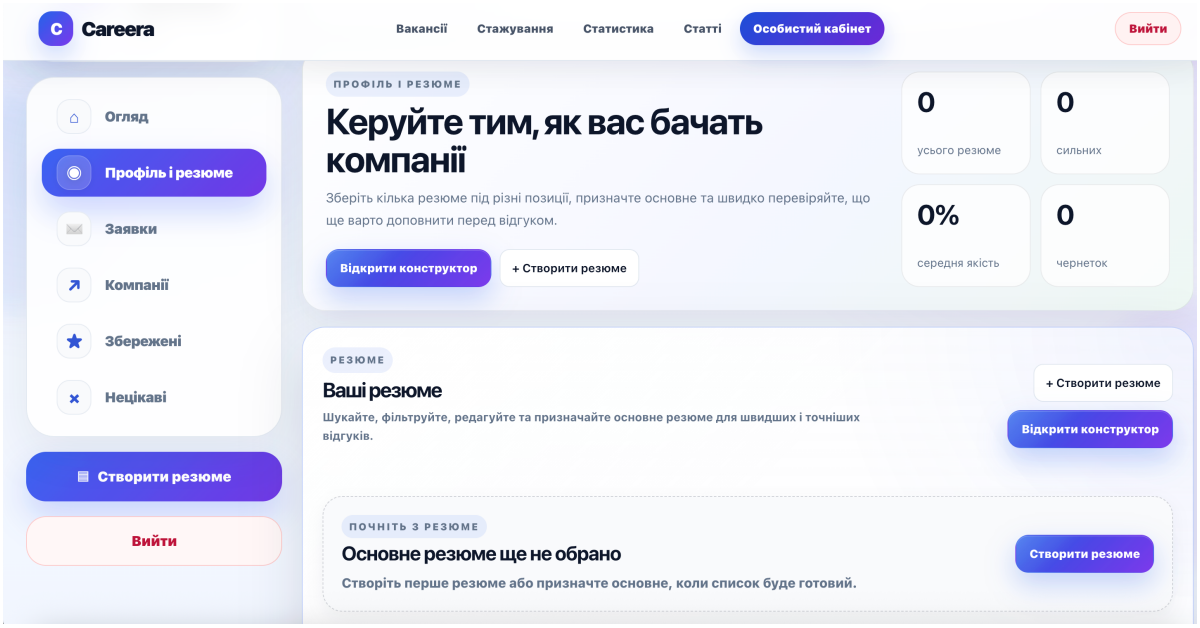


Рис. 3.12. Сторінка керування резюме

Сторінку заявок кандидата наведено на рис. 3.13. У цьому розділі користувач бачить вакансії, на які вже подав відгук, назву компанії, дату подання, використане резюме та поточний статус заявки. Це дозволяє кандидату відстежувати процес розгляду своїх відгуків.

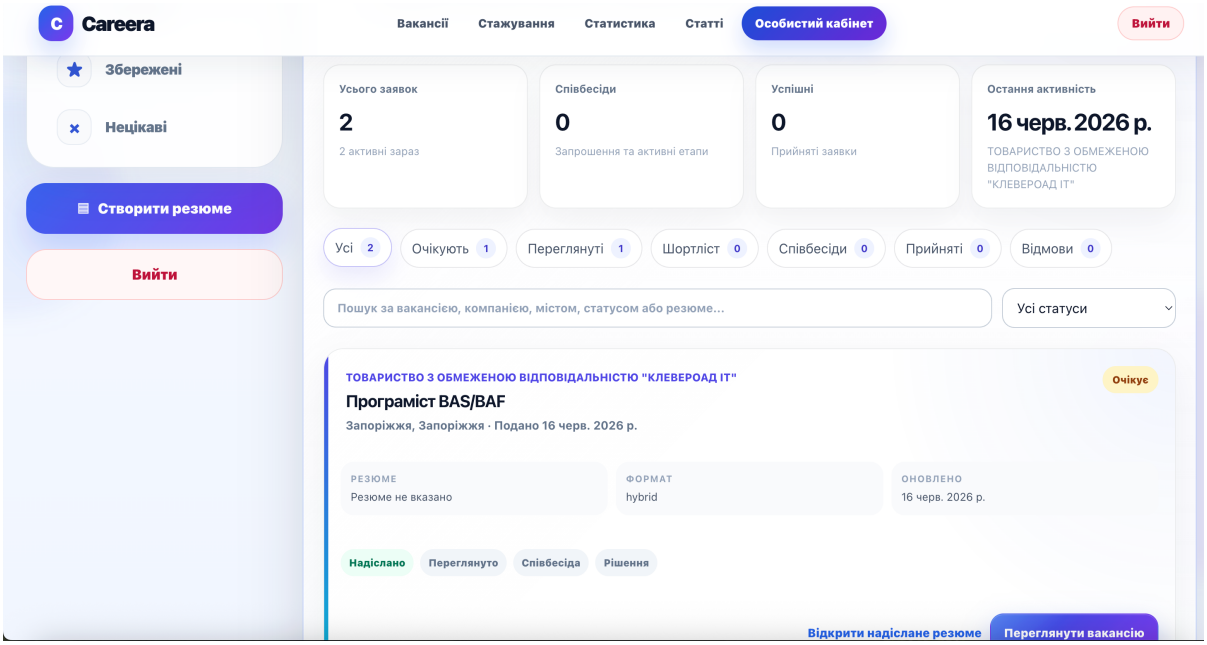


Рис. 3.13. Сторінка заявок кандидата

Сторінку збережених вакансій наведено на рис. 3.14. У ній кандидат бачить усі пропозиції, які додав до обраних. Користувач може повернутися до детального перегляду вакансії або видалити її зі списку збережених.

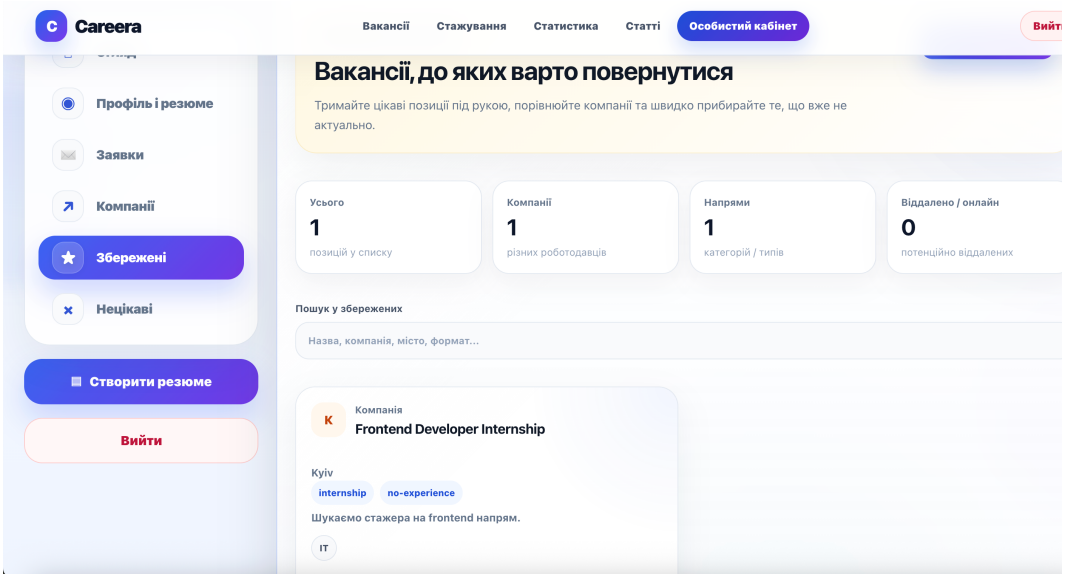


Рис. 3.14. Сторінка збережених вакансій

Особистий кабінет компанії наведено на рис. 3.15. У ньому відображається інформація про профіль компанії, створені вакансії, отримані заявки, статистика активності та доступні дії. Кабінет компанії є основним робочим простором представника роботодавця.

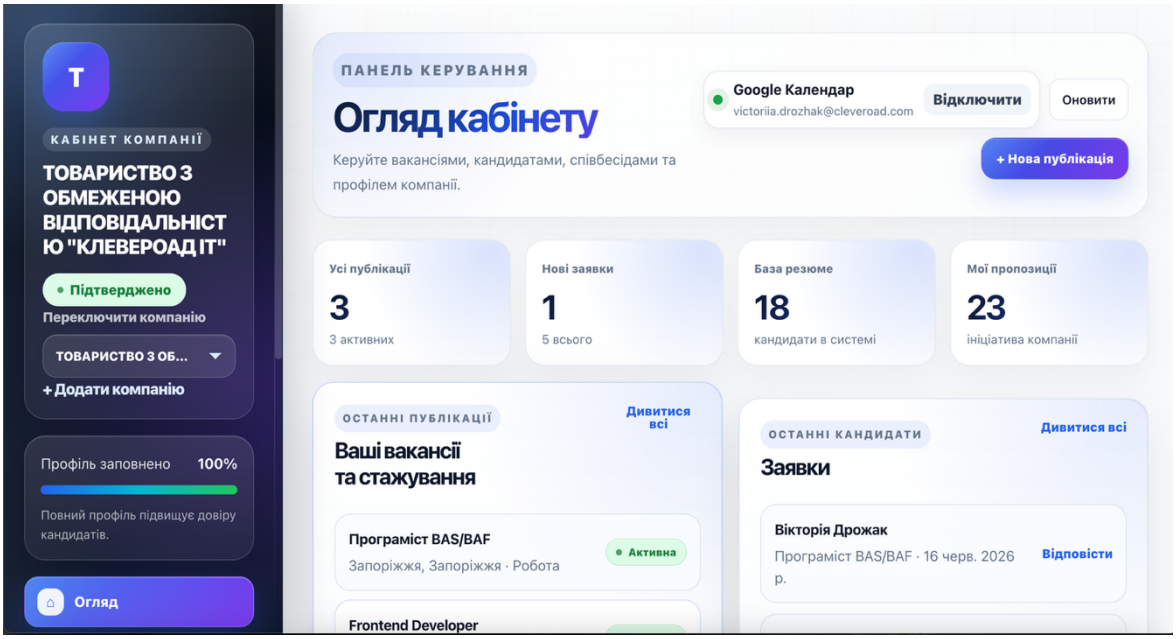


Рис. 3.15. Особистий кабінет компанії

Сторінку редагування профілю компанії наведено на рис. 3.16. На ній представник роботодавця може вказати назву організації, опис діяльності, сайт, контактні дані, місцезнаходження, логотип, код ЄДРПОУ та інші реєстраційні дані. Також на цій сторінці може відображатися статус підтвердження компанії.

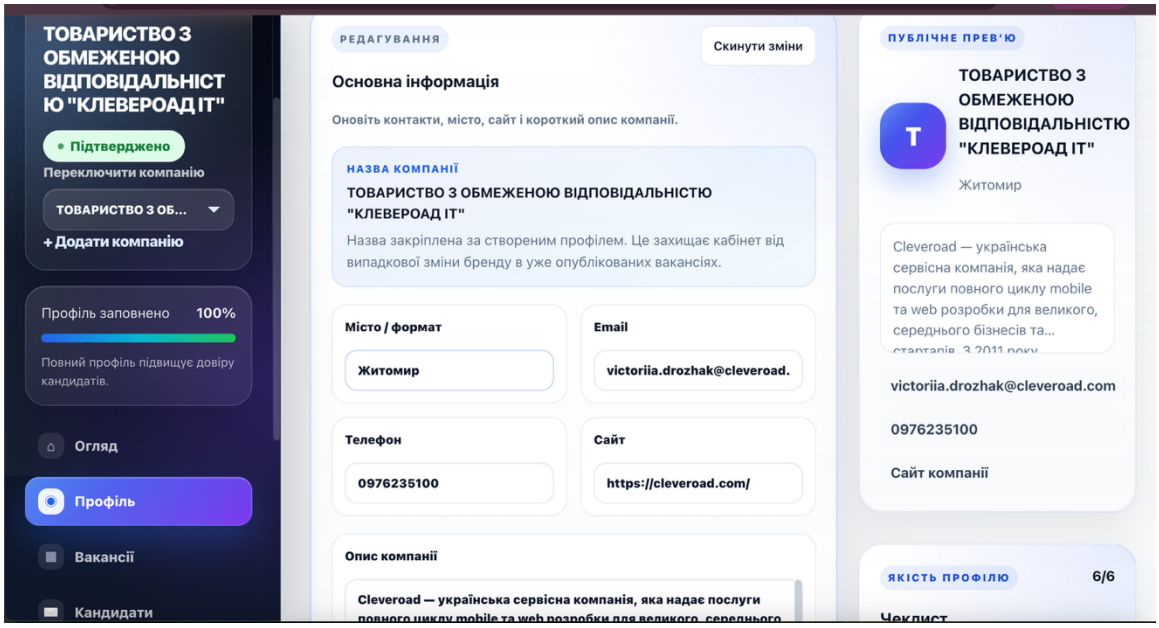


Рис. 3.16. Сторінка редагування профілю компанії

Сторінку створення вакансії наведено на рис. 3.17. Вона містить форму, у якій роботодавець вводить назву посади, опис, обов'язки, вимоги, умови роботи, місто, формат зайнятості, тип роботи та рівень досвіду. Це зображення демонструє процес публікації нової вакансії.

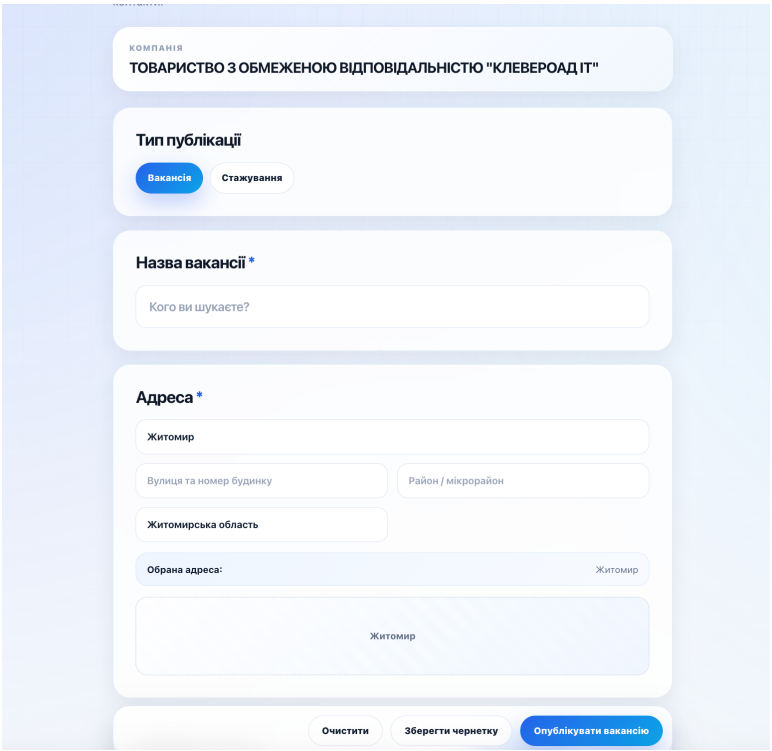


Рис. 3.17. Сторінка створення вакансії

Сторінку керування вакансіями компанії наведено на рис. 3.18. У ній відображаються всі створені вакансії, їхній статус, дата публікації, кількість

заявок і доступні дії. Компанія може редагувати, закривати, приховувати або видаляти вакансії.

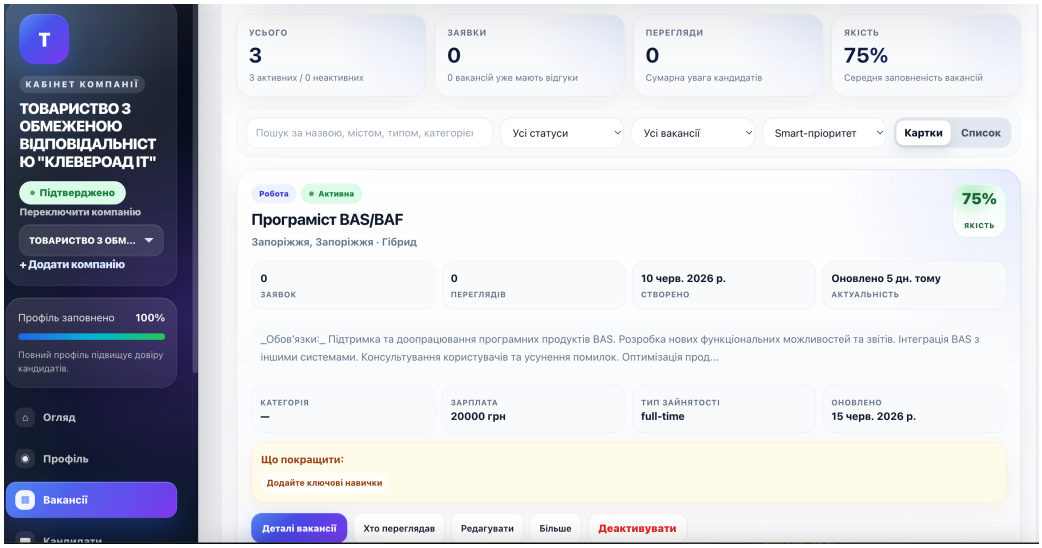


Рис. 3.18. Сторінка керування вакансіями компанії

Сторінку перегляду заявок компанією наведено на рис. 3.19. Тут роботодавець бачить кандидатів, які подали заявки на його вакансії, переглядає резюме, супровідні листи, дату подання та статус заявки. Представник компанії може змінити статус заявки або запросити кандидата на співбесіду.

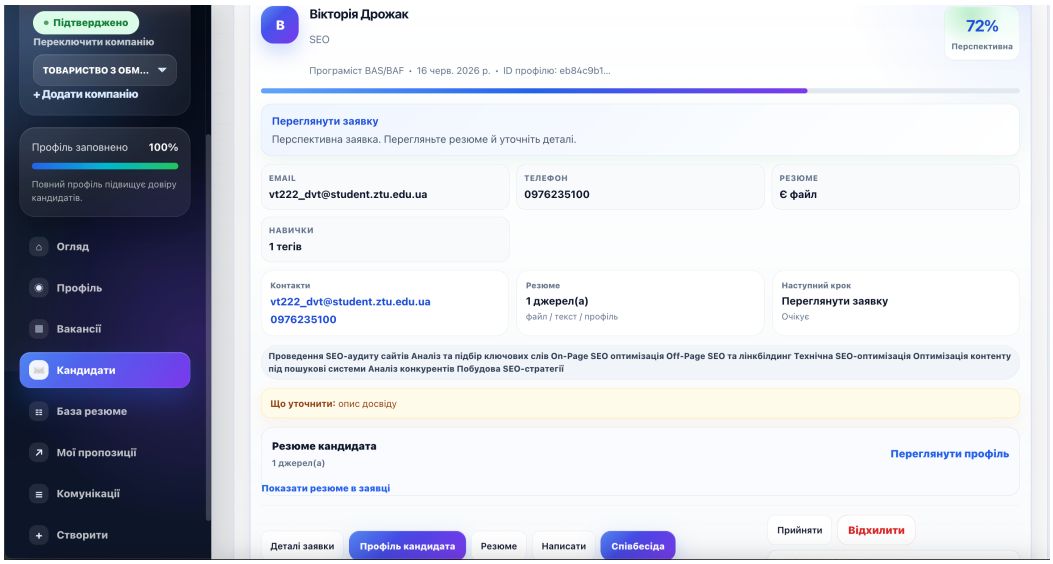


Рис. 3.19. Сторінка перегляду заявок компанією

Сторінку пошуку кандидатів наведено на рис. 3.20. У цьому розділі компанія може переглядати резюме, розміщені в системі, аналізувати навички, досвід, освіту та інші професійні дані кандидатів. Це демонструє активну модель пошуку працівників з боку роботодавця.

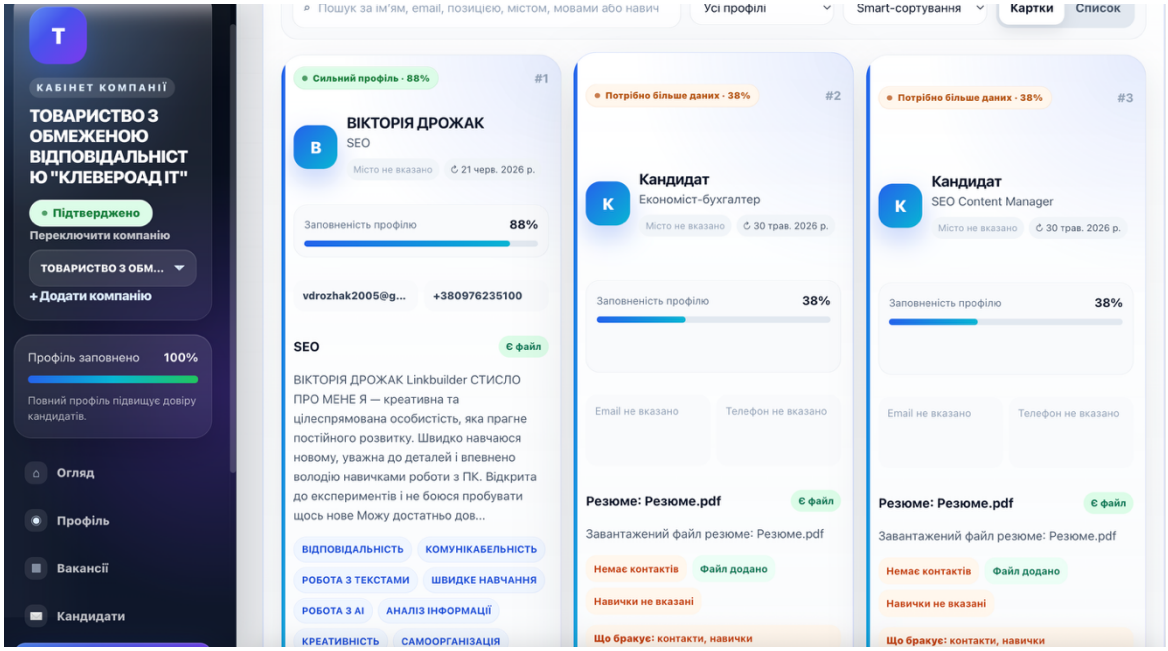


Рис. 3.20. Сторінка пошуку кандидатів

Форму планування співбесіди наведено на рис. 3.21. У ній компанія може обрати кандидата, вказати дату, час, опис зустрічі та створити подію з посиланням на відеоконференцію.

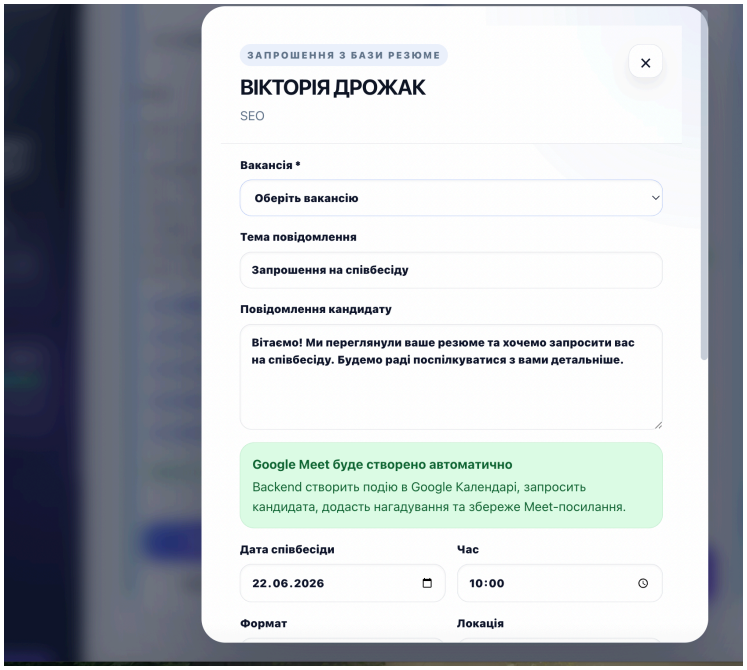


Рис. 3.21. Форма планування співбесіди

Сторінку мої пропозиції наведено на рис. 3.22. У цьому розділі компанія може переглядати всі створені нею вакансії та пропозиції. Для кожної пропозиції відображається назва посади, дата створення, статус публікації, кількість отриманих заявок і доступні дії. Представник компанії може перейти до

перегляду вакансії, редагувати її, оновити умови, закрити, приховати або видалити неактуальну пропозицію. Наявність такого розділу дозволяє роботодавцю зручно керувати всіма опублікованими вакансіями в одному місці.

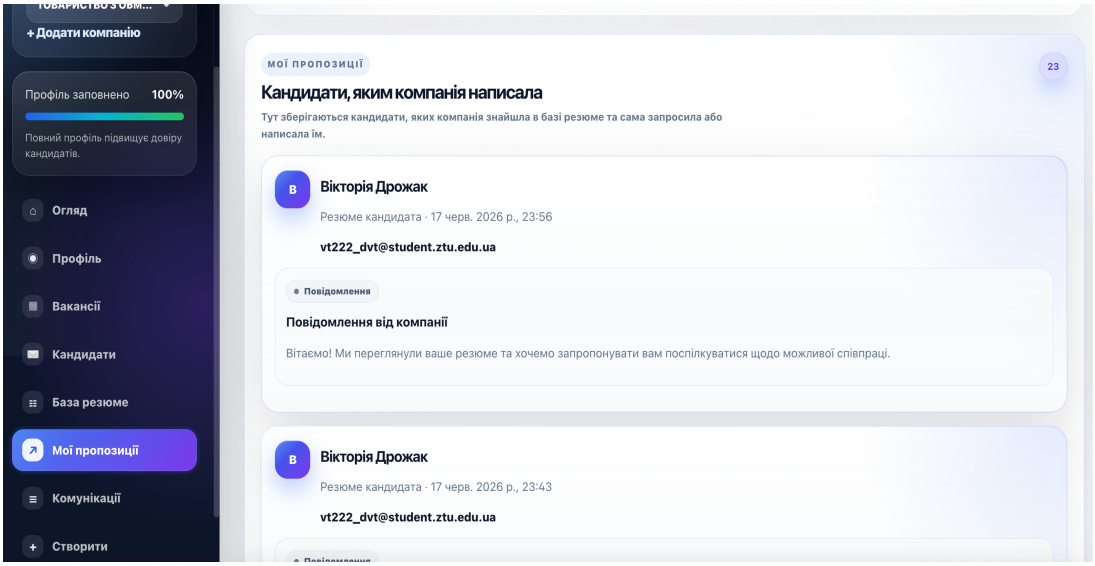


Рис. 3.22. Сторінка мої пропозиції

Сторінку комунікації наведено на рис. 3.23. Цей розділ призначений для організації взаємодії між компанією та кандидатами. У ньому можуть відображатися повідомлення, відповіді на заявки, запрошення на співбесіду, інформація про заплановані зустрічі та інші комунікаційні елементи. Через цей інтерфейс роботодавець може швидше зв’язуватися з кандидатами, уточнювати деталі заявки, надсилати запрошення або координувати подальші етапи відбору.

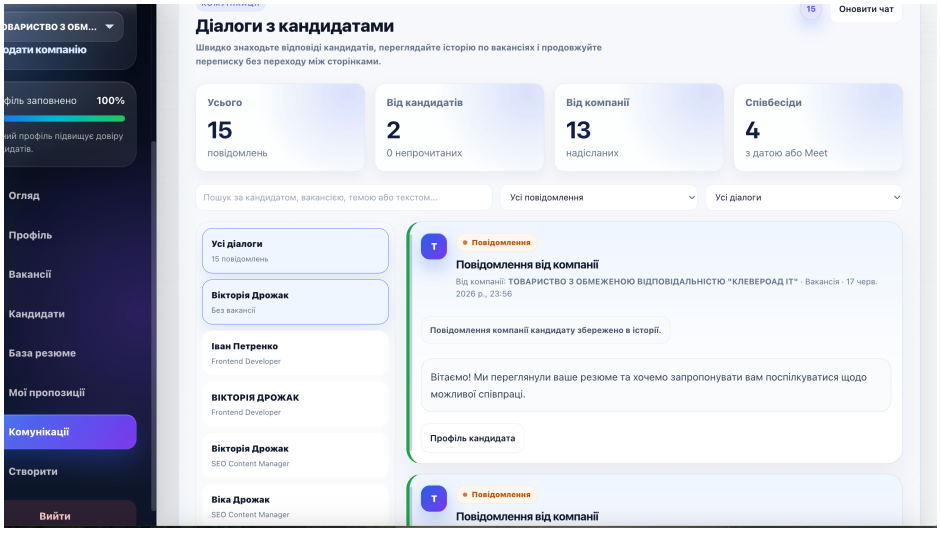


Рис. 3.23. Сторінка комунікації

Адміністративну панель наведено на рис. 3.24. Вона містить загальні інструменти керування системою, статистичні блоки, навігацію між адміністративними розділами та доступ до керування основними сутностями платформи.

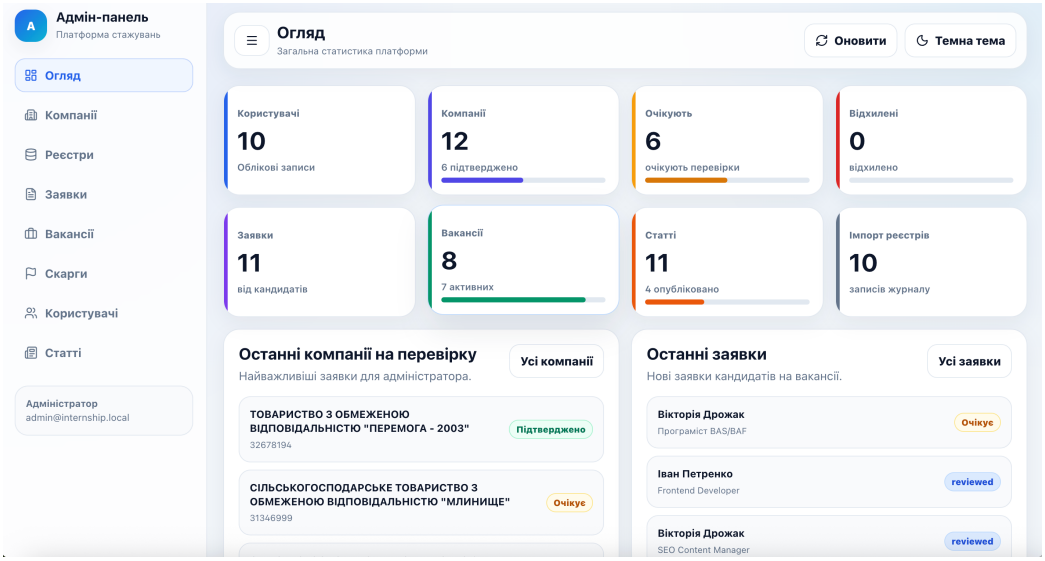


Рис. 3.24. Адміністративна панель

Сторінку керування користувачами наведено на рис. 3.25. У ній адміністратор бачить список зареєстрованих користувачів, їхні ролі, електронні адреси, статуси та основну інформацію. Цей розділ дозволяє контролювати облікові записи користувачів.

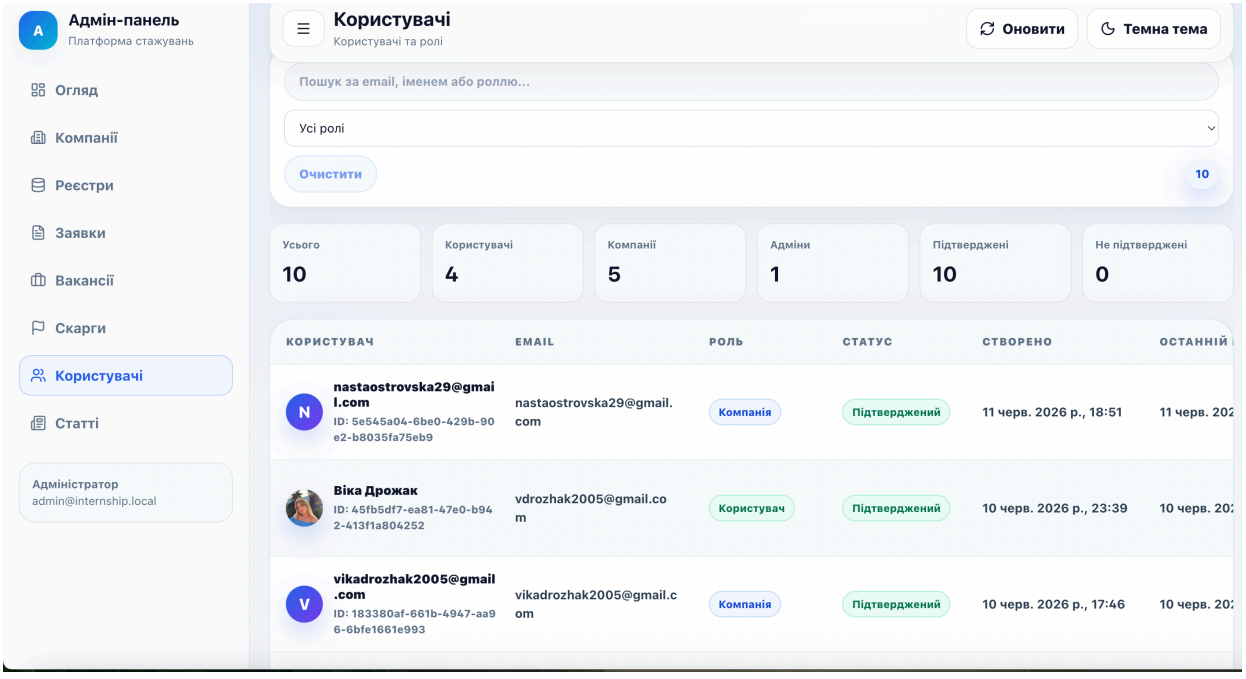


Рис. 3.25. Сторінка керування користувачами

Сторінку перевірки компаній наведено на рис. 3.26. У цьому розділі адміністратор переглядає інформацію про роботодавців, перевіряє їхні реєстраційні дані, код ЄДРПОУ та змінює статус підтвердження. Це зображення демонструє механізм контролю достовірності компаній.

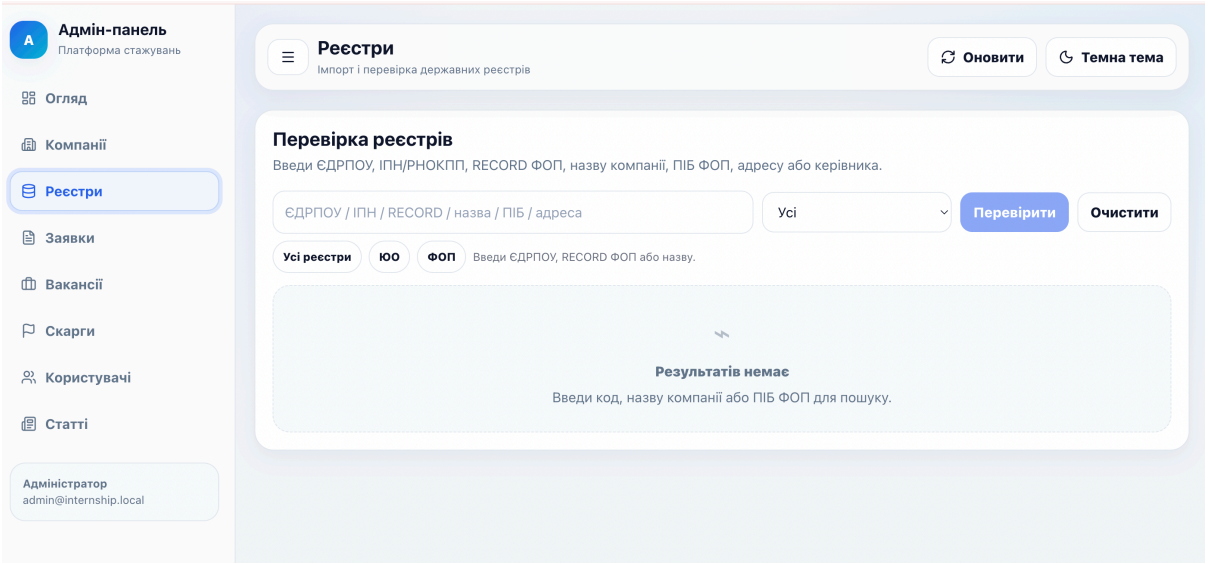


Рис. 3.26. Сторінка перевірки компаній

Сторінку керування вакансіями в адміністративній панелі наведено на рис. 3.27. На ній адміністратор може переглядати всі вакансії, перевіряти їхній зміст, статус і за потреби виконувати дії модерації. Такий розділ потрібний для підтримки якості опублікованих пропозицій.

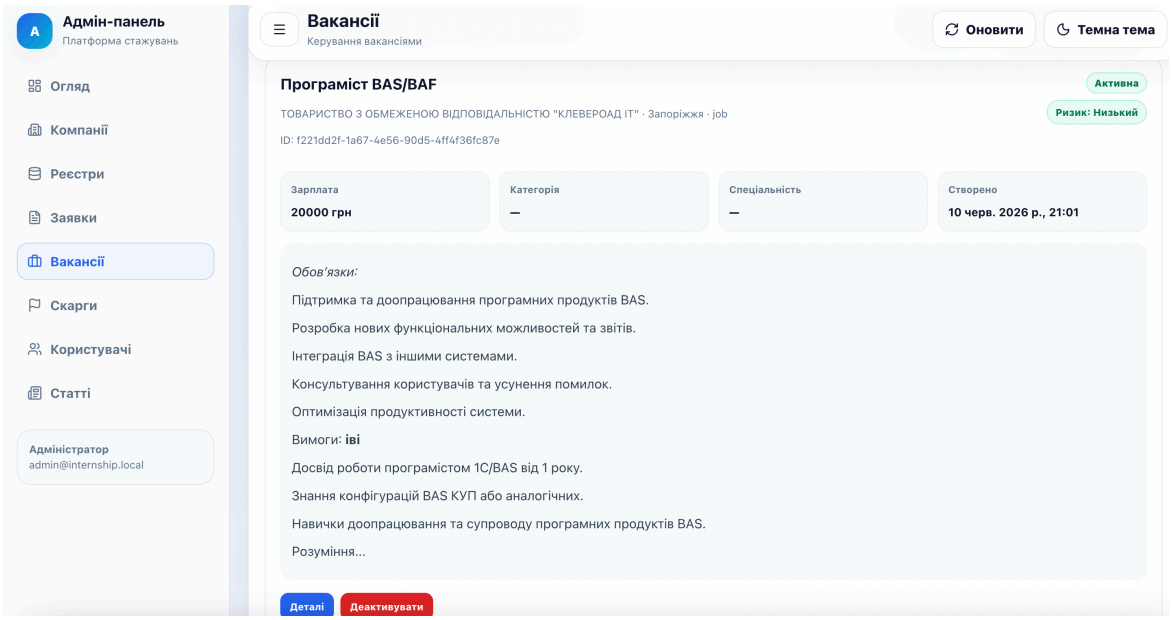


Рис. 3.27. Сторінка керування вакансіями в адміністративній панелі

Сторінку перегляду скарг на вакансії наведено на рис. 3.28. У ній адміністратор бачить скарги користувачів, причину звернення, пов’язану вакансію та може ухвалити рішення щодо подальших дій. Це демонструє механізм підтримки якості контенту.

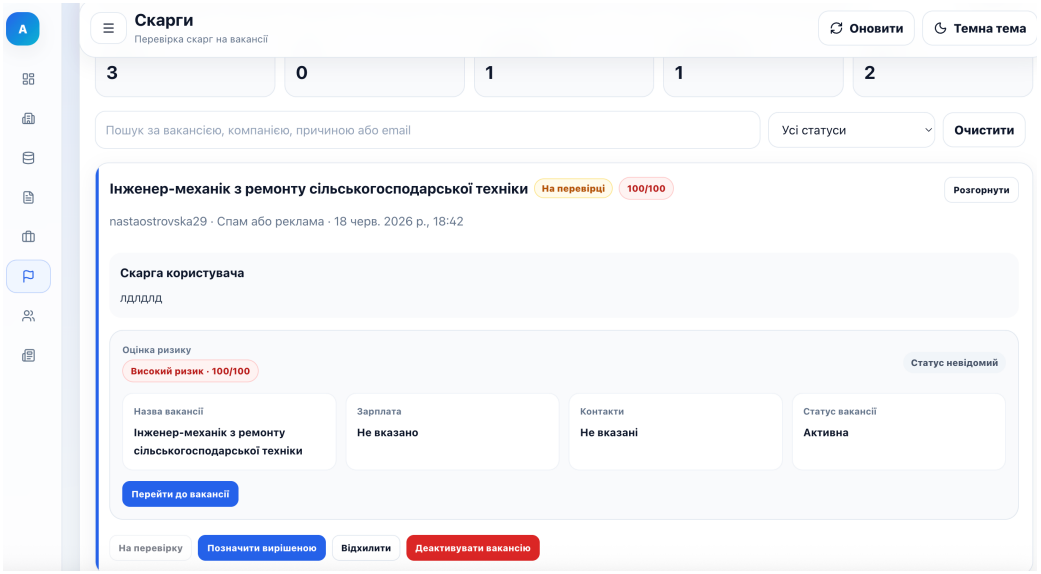


Рис. 3.28. Сторінка перегляду скарг на вакансії

Сторінку створення або редагування статті наведено на рис. 3.29. Вона містить поля заголовка, короткого опису, основного тексту, зображення та кнопки збереження або публікації. Це зображення демонструє, як адміністратор наповнює платформу інформаційними матеріалами.

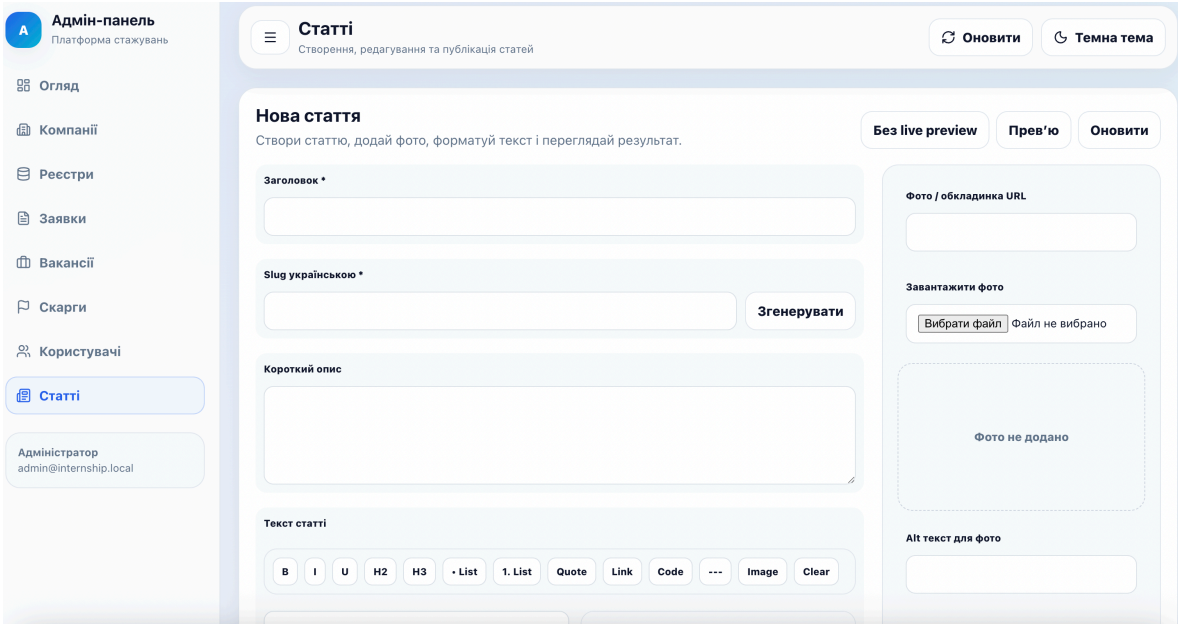


Рис. 3.29. Сторінка створення або редагування статті

Сторінку перегляду статті наведено на рис. 3.30. Вона призначена для ознайомлення користувачів з інформаційними матеріалами, що стосуються пошуку роботи, підготовки резюме, проходження співбесід, професійного розвитку та особливостей працевлаштування. На сторінці відображаються заголовок статті, дата публікації, основний текст, зображення або ілюстративні матеріали, а також додаткова інформація про автора чи джерело публікації. Такий розділ доповнює функціональність платформи та допомагає користувачам отримувати корисні поради для успішного пошуку роботи або стажування.

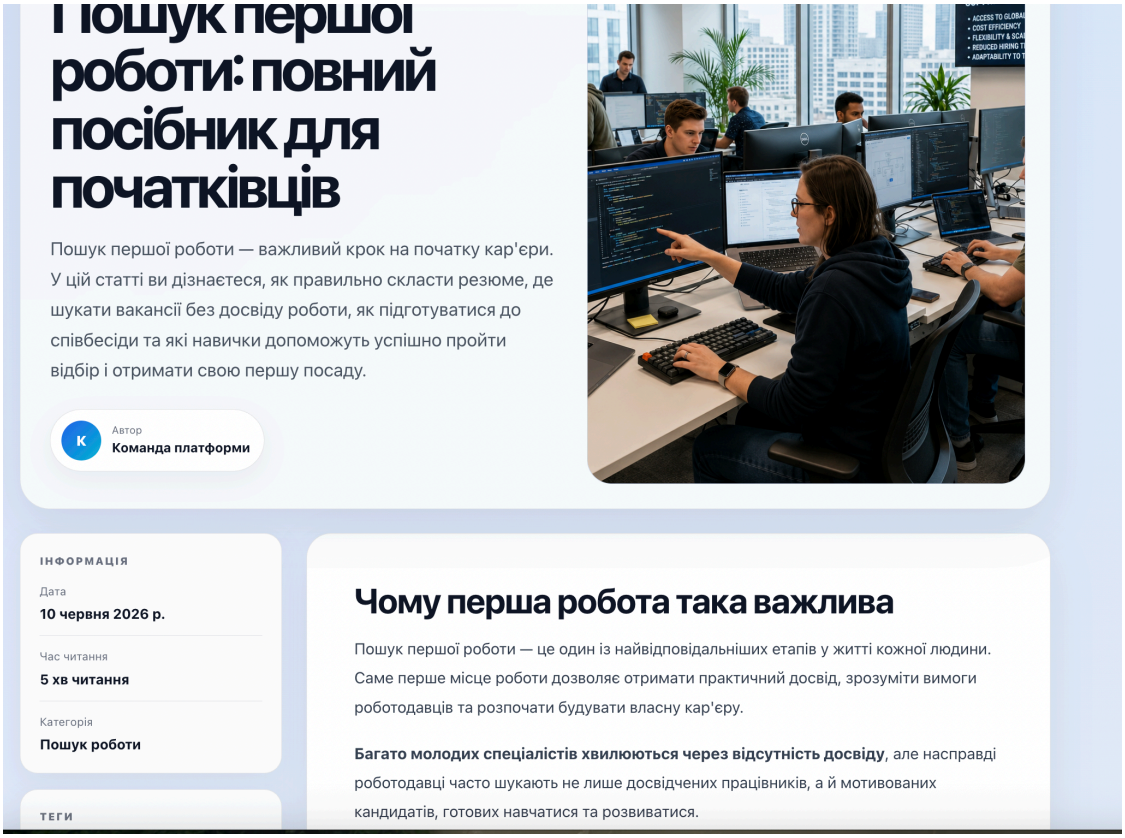


Рис. 3.30. Сторінка перегляду статті

3.3. Тестування функціональних можливостей вебплатформи

Тестування програмного комплексу виконувалося з метою перевірки правильності роботи вебплатформи для пошуку роботи та стажування, а також підтвердження відповідності реалізованого функціоналу поставленим вимогам. Основну увагу було приділено працездатності ключових модулів системи, коректності обробки введених даних, взаємодії клієнтської частини із сервером, роботі бази даних, зовнішніх інтеграцій та перевірці доступу користувачів відповідно до їхніх ролей.

У процесі тестування було перевірено основні сценарії використання системи: реєстрацію та авторизацію користувачів, підтвердження електронної пошти, вхід через Google, створення профілю кандидата, додавання й редагування резюме, пошук і фільтрацію вакансій, подання заявок, збереження вакансій, створення профілю компанії, публікацію вакансій, перегляд заявок, пошук кандидатів, перевірку компаній, роботу адміністративної панелі та інтеграцію з Google Calendar і Google Meet.

Спочатку було перевірено модуль реєстрації та входу до системи. Користувач вводив необхідні дані, обирав роль і надсилав форму. Якщо всі обов'язкові поля були заповнені коректно, система створювала новий обліковий запис. У разі пропуску обов'язкових полів, введення некоректної електронної адреси або неправильного пароля користувач отримував відповідне повідомлення про помилку. Також було протестовано підтвердження електронної пошти та можливість авторизації через Google-акаунт.

Окремо перевірялося розмежування прав доступу. Було встановлено, що кандидат має доступ лише до функцій пошуку вакансій, роботи з профілем, резюме, збереженими вакансіями та заявками. Представник компанії отримує доступ до створення й редагування профілю компанії, публікації вакансій, перегляду заявок і пошуку кандидатів. Адміністратор має доступ до адміністративної панелі та функцій керування основними об'єктами системи. Якщо користувач намагався виконати дію, яка не відповідає його ролі, система обмежувала доступ.

Під час перевірки функціоналу кандидата було протестовано створення та редагування профілю. Користувач додавав персональні дані, контактну інформацію, відомості про освіту, досвід і навички. Після збереження внесені дані коректно відображалися в особистому кабінеті. Також було перевірено створення, редагування, перегляд, видалення резюме та можливість позначення одного резюме як основного.

Функціонал пошуку вакансій перевірявся шляхом перегляду загального списку пропозицій, введення пошукових запитів і застосування фільтрів. Система

коректно відображала вакансії відповідно до заданих параметрів: ключових слів, міста, типу зайнятості, формату роботи, рівня досвіду та інших критеріїв. Також було перевірено сторінку детального перегляду вакансії, на якій відображається повна інформація про посаду, компанію, вимоги та умови роботи.

Подання заявки тестувалося шляхом вибору вакансії, резюме та надсилання супровідного листа. Система перевіряла, чи авторизований користувач, чи має він роль кандидата, чи належить йому вибране резюме та чи не подавав він заявку на цю вакансію раніше. Після успішної перевірки заявка створювалася в системі та відображалася як у кабінеті кандидата, так і в кабінеті компанії.

Також було перевірено функцію збереження вакансій. Кандидат міг додати вакансію до списку обраних і повернутися до неї пізніше. У разі повторного збереження тієї самої вакансії дублікати не створювалися, що підтвердило коректність роботи механізму контролю повторних записів.

Для представника компанії було протестовано створення та редагування профілю роботодавця. Після внесення інформації про компанію дані зберігалися та відображалися у відповідних розділах системи. Також було перевірено створення вакансій: після заповнення форми та збереження даних вакансія ставала доступною кандидатам у загальному списку пропозицій.

Окремо перевірялася робота із заявками кандидатів. Компанія могла переглядати заявки, які надійшли на її вакансії, відкривати резюме кандидатів, читати супровідні листи та змінювати статус заявки. Після зміни статусу оновлена інформація коректно відображалася і для представника компанії, і для кандидата.

Було протестовано можливість пошуку кандидатів компанією. Представник роботодавця міг переглядати резюме, розміщені в системі, оцінювати навички, освіту, досвід та інші дані кандидатів. Це підтвердило правильність роботи механізму активного пошуку потенційних працівників або стажерів.

Адміністративна панель перевірялася через функції перегляду користувачів, компаній, вакансій, заявок, скарг та інформаційних матеріалів.

Також було протестовано можливість перевірки компаній, модерації вакансій і керування статтями. Доступ до цих функцій був відкритий лише користувачу з роллю адміністратора.

Під час тестування перевірки компаній було встановлено, що статус роботодавця може бути змінений після перевірки його реєстраційних даних. Після підтвердження компанія отримувала відповідний статус у системі, що підвищує рівень довіри до роботодавця на платформі.

Також було перевірено роботу інтеграції з Google Calendar і Google Meet. Під час створення співбесіди система формувала подію календаря, зберігала дату й час зустрічі та створювала посилання на онлайн-зустріч. Це підтвердило можливість організації дистанційних співбесід безпосередньо через платформу.

Для узагальнення результатів тестування основних функцій складено таблицю 3.1.

Таблиця 3.1

Результати тестування основних функцій вебплатформи

	Функція системи	**Очікуваний результат**	**Результат тестування**
1	Реєстрація користувача	Створення нового облікового запису за умови коректного заповнення форми	Виконано успішно
2	Авторизація користувача	Вхід до системи з правильними обліковими даними	Виконано успішно
3	Підтвердження електронної пошти	Активація облікового запису після підтвердження пошти	Виконано успішно
4	Вхід через Google	Авторизація користувача за допомогою Google-акаунта	Виконано успішно
5	Розмежування прав доступу	Надання доступу відповідно до ролі користувача	Виконано успішно
6	Створення профілю кандидата	Збереження персональних і професійних даних кандидата	Виконано успішно
7	Керування резюме	Створення, редагування, видалення та вибір основного резюме	Виконано успішно
8	Пошук і фільтрація вакансій	Відображення вакансій відповідно до заданих критеріїв	Виконано успішно
9	Подання заявки на вакансію	Створення заявки та її відображення в кабінетах кандидата й компанії	Виконано успішно

Продовження таблиці 3.1

	Функція системи	**Очікуваний результат**	**Результат тестування**
10	Збереження вакансій	Додавання вакансії до списку обраних без дублювання записів	Виконано успішно
11	Створення профілю компанії	Збереження та відображення інформації про роботодавця	Виконано успішно
12	Публікація вакансії	Додавання нової вакансії до системи	Виконано успішно
13	Опрацювання заявок компанією	Перегляд заявок, резюме та зміна статусу заявки	Виконано успішно
14	Пошук кандидатів	Перегляд резюме кандидатів компанією	Виконано успішно
15	Перевірка компаній	Зміна статусу роботодавця після перевірки даних	Виконано успішно
16	Робота адміністративної панелі	Керування користувачами, компаніями, вакансіями, заявками та статтями	Виконано успішно
17	Інтеграція з Google Calendar і Google Meet	Створення події календаря та посилання на онлайн-зустріч	Виконано успішно

Проведене тестування показало, що основні функціональні модулі вебплатформи працюють коректно. Система правильно обробляє дані користувачів, забезпечує розмежування доступу між ролями, дозволяє кандидатам працювати з резюме та заявками, компаніям - з вакансіями й кандидатами, а адміністратору - контролювати вміст платформи. Отримані результати підтверджують готовність системи до подальшого використання та демонструють відповідність реалізованого функціоналу поставленим вимогам.

Висновки до третього розділу

У третьому розділі було розглянуто особливості впровадження, налаштування та перевірки працездатності вебплатформи для пошуку роботи та стажування. Описано загальну структуру розгортання системи, її основні програмні компоненти, інтерфейс користувача та результати тестування реалізованого функціоналу.

Було визначено, що система складається з клієнтської частини, серверної частини, бази даних і зовнішніх сервісів. Клієнтська частина забезпечує відображення сторінок вебплатформи та взаємодію користувача з інтерфейсом. Серверна частина відповідає за обробку запитів, автентифікацію, перевірку прав доступу, виконання бізнес-логіки та взаємодію з базою даних. База даних використовується для збереження інформації про користувачів, компанії, вакансії, резюме, заявки та інші об’єкти системи.

Також у розділі було описано основні сторінки вебплатформи, зокрема головну сторінку, сторінки входу та реєстрації, сторінку пошуку вакансій, особистий кабінет кандидата, кабінет компанії, сторінки керування резюме й заявками, а також адміністративну панель. Для кожної ролі користувача передбачено окремий набір функцій, що забезпечує зручність роботи із системою та розмежування доступу до її можливостей.

Під час тестування було перевірено основні сценарії використання вебплатформи: реєстрацію та авторизацію користувачів, підтвердження електронної пошти, створення профілю кандидата, додавання й редагування резюме, пошук і фільтрацію вакансій, подання заявок, збереження вакансій, створення вакансій компанією, перегляд кандидатів, зміну статусів заявок, роботу адміністративної панелі та інтеграцію із зовнішніми сервісами.

Результати тестування показали, що основні функціональні модулі системи працюють коректно. Вебплатформа правильно зберігає й обробляє дані, відображає необхідну інформацію, розмежовує доступ відповідно до ролей користувачів і повідомляє про помилки у разі некоректного введення даних або спроби виконання недозволених дій.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено вебплатформу для пошуку роботи та стажування. Система призначена для спрощення взаємодії між кандидатами та роботодавцями, пошуку вакансій, створення резюме, подання заявок і керування процесом відбору.

У першому розділі було проаналізовано предметну область та існуючі сервіси для пошуку роботи, зокрема LinkedIn, Work.ua та robota.ua. Встановлено, що такі платформи мають широкий функціонал, однак не завжди достатньо орієнтовані на студентів, початківців і користувачів, які шукають стажування або першу роботу. Це підтвердило доцільність розробки окремої вебплатформи.

У другому розділі було виконано проєктування системи. Визначено основні ролі користувачів: гість, кандидат, компанія та адміністратор. Також було описано варіанти використання, діаграму класів, структуру бази даних і основні алгоритми роботи системи.

У третьому розділі було розглянуто реалізацію та тестування вебплатформи. Клієнтська частина створена з використанням React і Vite, серверна - на основі Node.js та Express.js. Для зберігання даних і автентифікації використано PostgreSQL та Supabase. Також реалізовано підтвердження електронної пошти, вхід через Google, перевірку компаній та інтеграцію з Google Calendar і Google Meet.

Розроблена вебплатформа забезпечує основні функції для кандидатів, компаній та адміністратора. Кандидати можуть створювати профіль, формувати резюме, шукати вакансії та подавати заявки. Компанії можуть публікувати вакансії, переглядати заявки, змінювати їхній статус і самостійно шукати кандидатів. Адміністратор контролює роботу платформи, перевіряє компанії та керує інформаційними матеріалами.

Проведене тестування підтвердило коректну роботу основних функцій системи. Було перевірено реєстрацію, авторизацію, роботу з профілями, резюме, вакансіями, заявками, адміністративною панеллю та зовнішніми інтеграціями.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. OECD. Transition from education to work: Where are today's youth? Education at a Glance 2025. OECD Publishing, 2025. URL: https://www.oecd.org/en/publications/education-at-a-glance-2025_1c0d9c79-en/full-report/transition-from-education-to-work-where-are-today-s-youth_b90719d0.html (дата звернення: 22.03.2026).
2. International Labour Organization. Global Employment Trends for Youth 2024. Geneva: ILO, 2024. URL: https://www.ilo.org/publications/major-publications/global-employment-trends-youth-2024 (дата звернення: 26.03.2026).
3. The Use of Advanced Technology in Job Matching Platforms. World Bank, Solutions for Youth Employment : вебсайт. URL: https://thedocs.worldbank.org/en/doc/ceb5c5792ad0d874e9b1c3cc71362f46-0460012023/digital-job-matching-platforms-s4ye-draft-note-for-discussion (дата звернення: 05.04.2026).
4. Signore C., Della Piana B., Di Vincenzo F. Digital Job Searching and Recruitment Platforms: A Semi-systematic Literature Review // Methodologies and Intelligent Systems for Technology Enhanced Learning, Workshops. MIS4TEL 2023. Lecture Notes in Networks and Systems. Cham : Springer Nature Switzerland, 2023. Vol. 769. P. 313–322. URL: https://link.springer.com/chapter/10.1007/978-3-031-42134-1_31 (дата звернення: 05.04.2026).
5. LinkedIn Jobs : вебсайт. URL: https://www.linkedin.com/jobs/ (дата звернення: 13.03.2026).
6. Work.ua — сайт пошуку роботи в Україні : вебсайт. URL: https://www.work.ua/ (дата звернення: 13.03.2026).
7. robota.ua — сайт пошуку роботи в Україні : вебсайт. URL: https://robota.ua/ (дата звернення: 13.03.2026).
8. React : документація. URL: https://react.dev/ (дата звернення: 05.04.2026).
9. Vite. Next Generation Frontend Tooling : документація. URL: https://vite.dev/ (дата звернення: 20.04.2026).

10. CSS: Cascading Style Sheets. MDN Web Docs : документація. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 20.04.2026).
11. Node.js. Run JavaScript Everywhere : документація. URL: https://nodejs.org/ (дата звернення: 20.04.2026).
12. Express.js. Node.js Web Application Framework : документація. URL: https://expressjs.com/ (дата звернення: 20.04.2026).
13. Supabase Docs : документація. URL: https://supabase.com/docs (дата звернення: 20.04.2026).
14. Supabase Storage : документація. URL: https://supabase.com/docs/guides/storage (дата звернення: 20.04.2026).
15. Supabase Auth : документація. URL: https://supabase.com/docs/guides/auth (дата звернення: 20.04.2026).
16. Google Calendar API Overview. Google for Developers : документація. URL: https://developers.google.com/workspace/calendar/api/guides/overview (дата звернення: 24.05.2026).
17. Events: insert. Google Calendar API. Google for Developers : документація. URL: https://developers.google.com/workspace/calendar/api/v3/reference/events/insert (дата звернення: 24.05.2026).
18. Sommerville I. Software Engineering. 10th ed. Boston : Pearson Education, 2016. 816 p.
19. OAuth 2.0 Authorization Framework. RFC 6749. URL: https://datatracker.ietf.org/doc/html/rfc6749 (дата звернення: 24.05.2026).
20. Єдиний державний вебпортал відкритих даних України. Data.gov.ua : вебсайт. URL: https://data.gov.ua (дата звернення: 24.05.2026).
21. Extensible Markup Language (XML) 1.0 (Fifth Edition). World Wide Web Consortium (W3C). URL: https://www.w3.org/TR/xml/ (дата звернення: 24.05.2026).
22. JSON Web Token (JWT). RFC 7519. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 24.05.2026).

23. Google Calendar API Documentation. Google for Developers : документація. URL: https://developers.google.com/calendar (дата звернення: 24.05.2026).
24. Google Meet Documentation. Google for Developers : документація. URL: https://developers.google.com/workspace (дата звернення: 24.05.2026).
25. PostgreSQL Documentation : документація. URL: https://www.postgresql.org/docs/ (дата звернення: 24.05.2026).
26. JavaScript Guide. MDN Web Docs : документація. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide (дата звернення: 24.05.2026).
27. REST API Tutorial : вебсайт. URL: https://restfulapi.net/ (дата звернення: 24.05.2026).
28. OAuth 2.0 Simplified : вебсайт. URL: https://www.oauth.com/ (дата звернення: 24.05.2026).
29. Nielsen J. Usability Engineering. San Francisco : Morgan Kaufmann Publishers, 1994. 362 p.
30. Krug S. Don’t Make Me Think, Revisited: A Common Sense Approach to Web Usability. 3rd ed. Berkeley : New Riders, 2014. 216 p.

ДОДАТКИ

Лістинг коду

```
import { useMemo, useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import { useAuth } from "../../../context/AuthContext";
import "./Login.css";

function isEmail(value) {
    return /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(String(value || "").trim());
}

function normalizeRole(role) {
    const value = String(role || "").toLowerCase();

    if (value === "candidate" || value === "jobseeker" || value === "user") {
        return "student";
    }

    if (value === "company") {
        return "company";
    }

    if (value === "admin") {
        return "admin";
    }

    return "student";
}

function getRedirectPath(role) {
    const normalizedRole = normalizeRole(role);

    if (normalizedRole === "admin") {
        return "/admin";
    }

    if (normalizedRole === "company") {
        return "/company/dashboard";
    }


    return "/profile";
}

function isBadStudentRedirect(path) {
    const value = String(path || "").toLowerCase();

    return (
        value === "/jobs" ||
        value === "/resume/create" ||
        value === "/student/resume/create" ||
        value === "/register/candidate/goal" ||
        value.includes("resume/create")
    );
}

function makeFunnyError(err) {
    const raw = String(
        err?.message || err?.originalMessage || err?.title || ""
    ).toLowerCase();

    if (
        raw.includes("invalid login credentials") ||
        raw.includes("invalid credentials") ||
        raw.includes("пароль")
    ) {
        return {
            title: "Пароль пішов за кавою ☕",
```

```
            message:
                "Акаунт є, але пароль не збігається. Спробуй ще раз — без паніки.",
            actionText: null,
            actionPath: null,
        };
    }

    if (
        raw.includes("not found") ||
        raw.includes("акаунт не знайдено") ||
        raw.includes("не знайдено")
    ) {
        return {
            title: "Ми все обшукали 🕵️",
            message: "Акаунта з такою поштою немає. Саме час створити новий.",
            actionText: "Зареєструватися",
            actionPath: "/register",
        };
    }

    if (
        raw.includes("email not confirmed") ||
        raw.includes("not confirmed") ||
        raw.includes("підтвердіть")
    ) {
        return {
            title: "Пошта чекає свого героя 💌",
            message:
                "Зайди в пошту й підтвердь акаунт. Лист міг сховатися у спамі.",
            actionText: null,
            actionPath: null,
        };
    }

    if (raw.includes("email") || raw.includes("пошта")) {
        return {
            title: "Пошта виглядає підозріло 👀",
            message: "Перевір, чи не загубилась крапка, @ або домен.",
            actionText: null,
            actionPath: null,
        };
    }

    if (raw.includes("network") || raw.includes("failed to fetch")) {
        return {
            title: "Інтернет робить вигляд, що працює 🛜",
            message: "Перевір з’єднання або спробуй ще раз за кілька секунд.",
            actionText: "Спробувати ще",
            actionPath: null,
        };
    }

    if (raw.includes("too many") || raw.includes("rate limit")) {
        return {
            title: "Ти занадто швидкий для нас 🚀",
            message: "Система попросила паузу. Дай їй хвилинку видихнути.",
            actionText: null,
            actionPath: null,
        };
    }

    return {
        title: err?.title || "Щось пішло не за планом 🫠",
        message:
            err?.message ||
            "Ми не впали, ми просто прилягли. Спробуй ще раз.",
        actionText: err?.actionText,
        actionPath: err?.actionPath,
    };
```

```
}

export default function Login() {
    const navigate = useNavigate();
    const auth = useAuth();

    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const [showPassword, setShowPassword] = useState(false);
    const [submitting, setSubmitting] = useState(false);
    const [checkingEmail, setCheckingEmail] = useState(false);
    const [emailHint, setEmailHint] = useState(null);
    const [error, setError] = useState(null);
    const [success, setSuccess] = useState("");

    const canSubmit = useMemo(() => {
        return isEmail(email) && password.trim().length >= 6 && !submitting;
    }, [email, password, submitting]);

    async function handleEmailBlur() {
        const normalizedEmail = email.trim().toLowerCase();

        setEmailHint(null);

        if (!normalizedEmail || !isEmail(normalizedEmail)) return;
        if (!auth.checkEmailExists) return;

        try {
            setCheckingEmail(true);

            const result = await auth.checkEmailExists(normalizedEmail);

            if (result?.exists) {
                setEmailHint({
                    type: "success",
                    title: "Акаунт знайдено ✅",
                    message: "Ця пошта вже зареєстрована. Можна входити.",
                });
            } else if (result?.ok && !result.exists) {
                setEmailHint({
                    type: "info",
                    title: "Акаунта ще немає 🐣",
                    message: "З цією поштою ще не реєструвалися.",
                    actionText: "Зареєструватися",
                    actionPath: "/register",
                });
            }
        } catch {
            setEmailHint(null);
        } finally {
            setCheckingEmail(false);
        }
    }

    async function handleSubmit(e) {
        e.preventDefault();

        setError(null);
        setSuccess("");

        const normalizedEmail = email.trim().toLowerCase();

        if (!isEmail(normalizedEmail)) {
            setError({
                title: "Пошта виглядає підозріло 👀",
                message: "Перевір формат. Наприклад: name@gmail.com",
            });
            return;
        }
```

```
    if (password.trim().length < 6) {
        setError({
            title: "Пароль замалий 🐣",
            message: "Пароль має містити щонайменше 6 символів.",
        });
        return;
    }

    try {
        setSubmitting(true);

        const result = await auth.signIn(normalizedEmail, password);

        const nextRole = normalizeRole(result?.role || auth.role || "student");


        let redirectTo = getRedirectPath(nextRole);

        if (nextRole !== "student" && result?.redirectTo) {
            redirectTo = result.redirectTo;
        }

        if (nextRole === "student") {
            redirectTo = "/profile";
        }

        if (nextRole === "student" && isBadStudentRedirect(result?.redirectTo)) {
            redirectTo = "/profile";
        }

        setSuccess(
            nextRole === "student"
                ? "Вхід успішний. Відкриваємо профіль 🚀"
                : result?.message || "Вхід успішний. Відкриваємо кабінет 🚀"
        );

        navigate(redirectTo, { replace: true });
    } catch (err) {
        setError(makeFunnyError(err));
    } finally {
        setSubmitting(false);
    }
}

async function handleGoogleLogin() {
    setError(null);
    setSuccess("");

    try {
        setSubmitting(true);

        if (!auth.signInWithProvider && !auth.signInWithGoogle) {
            throw {
                title: "Google-вхід не налаштований",
                message: "У AuthContext немає функції signInWithProvider.",
            };
        }


        localStorage.setItem(
            "user_signup_draft",
            JSON.stringify({
                role: "student",
                next: "/profile",
            })
        );

        if (auth.signInWithGoogle) {
            await auth.signInWithGoogle("student");
        } else {
```

```
            await auth.signInWithProvider("google", "student");
        }
    } catch (err) {
        setError(makeFunnyError(err));
        setSubmitting(false);
    }
}

return (
    <div className="login-page">
        <div className="login-shell">
            <section className="login-hero">
                <span className="login-badge">Career Platform</span>

                <h1 className="login-hero__title">Увійди у свій акаунт</h1>

                <p className="login-hero__text">
                    Керуй профілем, переглядай збережені вакансії та відгукуйся
                    на стажування й роботу в кілька кліків.
                </p>

                <div className="login-hero__stats">
                    <div className="login-stat">
                        <strong>1 акаунт</strong>
                        <span>для студента або компанії</span>
                    </div>

                    <div className="login-stat">
                        <strong>Швидкий доступ</strong>
                        <span>до профілю, вакансій та відгуків</span>
                    </div>
                </div>
            </section>

            <section className="login-card">
                <div className="login-card__header">
                    <h2>Вхід</h2>
                    <p>Використай email і пароль або Google.</p>
                </div>

                <form className="login-form" onSubmit={handleSubmit}>
                    <label className="login-field">
                        <span>Email</span>

                        <input
                            type="email"
                            placeholder="you@example.com"
                            value={email}
                            onChange={(e) => {
                                setEmail(e.target.value);
                                setEmailHint(null);
                                setError(null);
                                setSuccess("");
                            }}
                            onBlur={handleEmailBlur}
                            autoComplete="email"
                        />
                    </label>

                    {checkingEmail ? (
                        <div className="login-alert login-alert--info">
                            Перевіряємо пошту...
                        </div>
                    ) : null}

                    {emailHint ? (
                        <div
                            className={
                                emailHint.type === "success"
                                    ? "login-alert login-alert--success"
```

```
                    : "login-alert login-alert--info"
                }
            >
                <strong>{emailHint.title}</strong>
                <span>{emailHint.message}</span>

                {emailHint.actionText && emailHint.actionPath ? (
                    <Link to={emailHint.actionPath}>
                        {emailHint.actionText}
                    </Link>
                ) : null}
            </div>
        ) : null}

        <label className="login-field">
            <span>Пароль</span>

            <div className="login-password-wrap">
                <input
                    type={showPassword ? "text" : "password"}
                    placeholder="Введи пароль"
                    value={password}
                    onChange={(e) => {
                        setPassword(e.target.value);
                        setError(null);
                        setSuccess("");
                    }}
                    autoComplete="current-password"
                />

                <button
                    type="button"
                    className="login-password-toggle"
                    onClick={() => setShowPassword((prev) => !prev)}
                >
                    {showPassword ? "Сховати" : "Показати"}
                </button>
            </div>
        </label>

        {error ? (
            <div className="login-alert login-alert--error">
                <strong>{error.title}</strong>
                <span>{error.message}</span>

                {error.actionText && error.actionPath ? (
                    <Link to={error.actionPath}>
                        {error.actionText}
                    </Link>
                ) : null}
            </div>
        ) : null}

        {success ? (
            <div className="login-alert login-alert--success">
                {success}
            </div>
        ) : null}

        <button
            type="submit"
            className="login-submit"
            disabled={!canSubmit}
        >
            {submitting ? "Входимо..." : "Увійти"}
        </button>

        <div className="login-divider">
            <span>або</span>
        </div>
```

```
                        <button
                            type="button"
                            className="login-google"
                            onClick={handleGoogleLogin}
                            disabled={submitting}
                        >
                            Увійти через Google
                        </button>

                        <p className="login-help">

                        </p>
                    </form>

                    <div className="login-footer">
                        <p>
                            Немає акаунта? <Link to="/register">Зареєструватися</Link>
                        </p>
                    </div>
                </section>
            </div>
        </div>
    );
}

import { useEffect, useMemo, useState } from "react";
import { Link, useLocation, useNavigate } from "react-router-dom";
import { apiFetch } from "../../../services/api/api";
import { useAuth } from "../../../context/AuthContext";
import { supabase } from "../../../services/supabaseClient";
import "../../../styles/registration.css";

const initialCompany = {
    company_name: "",
    company_type: "",
    registration_code: "",
    industry: "",
    website: "",
    description: "",
    contact_name: "",
    phone: "",
    city: "",
};

const INDUSTRIES = [
    "FMCG",
    "IT",
    "Інтернет",
    "Реклама та PR-послуги",
    "Транспорт та Логістика",
    "HR",
    "Банки",
    "Фінансові послуги",
    "Державний сектор",
    "Медицина та Охорона здоров'я",
    "Освіта",
    "Промисловість та Виробництво",
    "Будівництво",
    "Торгівля роздрібна / Retail",
    "Юридичні послуги",
    "Консалтинг / Аудит",
    "NGO",
    "АПК",
    "Оборонна промисловість",
    "Інше",
];

const PENDING_COMPANY_REGISTRATION_KEY = "pendingCompanyRegistration";
```

```
function savePendingCompanyRegistration(payload, email = "") {
    try {
        localStorage.setItem(
            PENDING_COMPANY_REGISTRATION_KEY,
            JSON.stringify({
                payload,
                email: String(email || "").trim().toLowerCase(),
                created_at: new Date().toISOString(),
            })
        );
    } catch {}
}

function clearPendingCompanyRegistration() {
    try {
        localStorage.removeItem(PENDING_COMPANY_REGISTRATION_KEY);
    } catch {}
}

function normalizeName(value) {
    return String(value || "").trim().replace(/\s+/g, " ");
}

function normalizeCompanyType(value) {
    return String(value || "").trim();
}

function isFop(companyType) {
    const value = String(companyType || "").trim().toUpperCase().replace(/\./g, "");
    return value === "ФОП" || value.includes("ФІЗИЧНА ОСОБА");
}

function getSubjectType(companyType) {
    return isFop(companyType) ? "fop" : "company";
}

function normalizeRegistrationCode(value, companyType = "") {
    const digits = String(value || "").replace(/\D/g, "");
    return isFop(companyType) ? digits.slice(0, 10) : digits.slice(0, 8);
}

function normalizePhone(value) {
    const raw = String(value || "").replace(/[^\d+]/g, "");
    if (raw.startsWith("+")) return raw.slice(0, 13);
    return raw.slice(0, 12);
}

function isValidEmail(value) {
    return /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(String(value || "").trim());
}

function isValidUkrainianPhone(value) {
    const raw = String(value || "").trim();
    const digits = raw.replace(/\D/g, "");

    if (raw.startsWith("+380")) return digits.length === 12;
    if (raw.startsWith("380")) return digits.length === 12;
    if (raw.startsWith("0")) return digits.length === 10;

    return digits.length >= 9 && digits.length <= 15;
}

function isValidWebsite(value) {
    const website = String(value || "").trim();
    if (!website) return true;

    try {
        const url = new URL(website);
        return url.protocol === "http:" || url.protocol === "https:";
    } catch {
```

```
        return false;
    }
}

function getRegistrationCodeLabel(companyType) {
    return isFop(companyType) ? "ІПН ФОП" : "ЄДРПОУ компанії";
}

function getRegistrationCodeHint(companyType) {
    return isFop(companyType)
        ? "Введіть 10 цифр ІПН ФОП"
        : "Введіть 8 цифр ЄДРПОУ";
}

function getPasswordStrength(password = "") {
    let score = 0;

    if (password.length >= 8) score += 1;
    if (/[A-ZА-ЯІЇЄҐ]/.test(password)) score += 1;
    if (/[a-za-яіїєґ]/i.test(password)) score += 1;
    if (/\d/.test(password)) score += 1;
    if (/[^A-Za-zA-Яа-яІіЇїЄєҐґ0-9]/.test(password)) score += 1;

    if (!password) {
        return {
            score: 0,
            label: "Пароль ще не придумали",
            className: "is-empty",
        };
    }

    if (score <= 2) {
        return {
            score,
            label: "Слабенький, як чай без заварки 🫖",
            className: "is-weak",
        };
    }

    if (score <= 4) {
        return {
            score,
            label: "Вже краще, пароль качається 💪",
            className: "is-medium",
        };
    }

    return {
        score,
        label: "Ого, це пароль-бодібілдер 🦾",
        className: "is-strong",
    };
}

function isStrongPassword(password) {
    return getPasswordStrength(password).score >= 4;
}

function getRawError(error) {
    return (
        error?.message ||
        error?.error ||
        error?.data?.error ||
        error?.response?.error ||
        String(error || "")
    );
}

function getErrorText(error = "") {
    const raw = getRawError(error);
```

```
    const lower = raw.toLowerCase();

    if (
        lower.includes("user already registered") ||
        lower.includes("already registered") ||
        lower.includes("already exists") ||
        lower.includes("duplicate key") ||
        lower.includes("23505")
    ) {
        return "🐣 А цей email уже вилупився. Не треба реєструватися вдруге — просто
увійдіть.";
    }

    if (lower.includes("rate limit") || lower.includes("too many requests")) {
        return "🐢 Трошки повільніше. Поштовий голуб втомився — спробуйте пізніше.";
    }

    if (lower.includes("smtp") || lower.includes("confirmation email")) {
        return "📮 Лист застряг у поштовій трубі. SMTP, здається, пішов на каву.";
    }

    if (lower.includes("network") || lower.includes("fetch") || lower.includes("сервер")) {
        return "🌐 Сервер десь загубився. Перевірте backend, інтернет і API URL.";
    }

    if (lower.includes("token") || lower.includes("jwt") || lower.includes("session")) {
        return "🪪 Перепустка протермінована. Увійдіть ще раз — і все оживе.";
    }

    if (
        lower.includes("компан") ||
        lower.includes("фоп") ||
        lower.includes("єдрпоу") ||
        lower.includes("ipн") ||
        lower.includes("реєстр")
    ) {
        return `🧐 Реєстр примружився: ${raw}`;
    }

    return `😅 Щось пішло не за планом. ${raw || "Спробуйте ще раз."}`;
}

function getVerificationColor(status) {
    if (status === "verified") return "#16803c";
    if (status === "rejected") return "#c62828";
    return "#b26a00";
}

function getVerificationTitle(status, subjectType = "company") {
    if (status === "verified") {
        return subjectType === "fop" ? "ФОП підтверджено" : "Компанію підтверджено";
    }

    if (status === "pending_review") return "Потрібна ручна перевірка";

    if (status === "rejected") {
        return subjectType === "fop" ? "ФОП не підтверджено" : "Компанію не підтверджено";
    }

    return "Перевірку виконано";
}

function getVerificationText(verification) {
    if (!verification) return "";
    if (verification.reason) return verification.reason;

    if (verification.status === "verified") {
        return verification.subject_type === "fop"
            ? "ФОП знайдено в реєстрі."
```

```
            : "Компанію знайдено в реєстрі.";
    }

    if (verification.status === "pending_review") {
        return "Реєстр не свариться, але просить ручну перевірку.";
    }

    if (verification.status === "rejected") {
        return "Дані не пройшли перевірку.";
    }

    return "Перевірку виконано.";
}

export default function EmployerRegister() {
    const navigate = useNavigate();
    const location = useLocation();
    const { signUp, signInWithProvider, refreshMe } = useAuth();

    const [step, setStep] = useState(1);
    const [oauthReady, setOauthReady] = useState(Boolean(location.state?.oauthReady));

    const [companyForm, setCompanyForm] = useState(initialCompany);
    const [authForm, setAuthForm] = useState({
        email: "",
        password: "",
        confirmPassword: "",
        agree: false,
    });

    const [fieldErrors, setFieldErrors] = useState({});
    const [showPassword, setShowPassword] = useState(false);
    const [showConfirmPassword, setShowConfirmPassword] = useState(false);

    const [loading, setLoading] = useState(false);
    const [oauthLoading, setOauthLoading] = useState(false);
    const [verifyLoading, setVerifyLoading] = useState(false);

    const [error, setError] = useState("");
    const [message, setMessage] = useState("");
    const [verification, setVerification] = useState(null);

    useEffect(() => {
        if (location.state?.oauthReady) {
            setOauthReady(true);
            setMessage("OAuth вхід успішний. Завершіть реєстрацію компанії.");
        }
    }, [location.state]);

    const subjectType = useMemo(
        () => getSubjectType(companyForm.company_type),
        [companyForm.company_type]
    );

    const registrationCode = useMemo(
        () =>
            normalizeRegistrationCode(
                companyForm.registration_code,
                companyForm.company_type
            ),
        [companyForm.registration_code, companyForm.company_type]
    );

    const isCodeValid = useMemo(() => {
        return subjectType === "fop"
            ? registrationCode.length === 10
            : registrationCode.length === 8;
    }, [subjectType, registrationCode]);

    const companyIsVerifiedEnough = verification?.status === "verified";
```

```
    const canVerifyCompany =
        companyForm.company_name.trim().length >= 2 &&
        companyForm.company_type.trim().length >= 2 &&
        isCodeValid &&
        !verifyLoading;

    const canGoToContacts =
        companyForm.company_name.trim().length >= 2 &&
        companyForm.company_type.trim().length >= 2 &&
        companyForm.industry.trim().length > 0 &&
        isCodeValid &&
        companyIsVerifiedEnough;

    const canGoToAuth =
        companyForm.contact_name.trim().length >= 2 &&
        isValidUkrainianPhone(companyForm.phone) &&
        companyForm.city.trim().length >= 2;

    const canSubmit =
        !loading &&
        authForm.agree &&
        canGoToContacts &&
        canGoToAuth &&
        (oauthReady ||
            (isValidEmail(authForm.email) &&
                authForm.password === authForm.confirmPassword &&
                isStrongPassword(authForm.password)));

    function clearStatus() {
        setError("");
        setMessage("");
    }

    function clearFieldError(name) {
        setFieldErrors((prev) => ({
            ...prev,
            [name]: "",
        }));
    }

    function handleCompanyChange(e) {
        const { name, value } = e.target;

        setCompanyForm((prev) => {
            const next = { ...prev, [name]: value };

            if (name === "company_type") {
                next.company_type = normalizeCompanyType(value);
                next.registration_code = normalizeRegistrationCode(
                    prev.registration_code,
                    value
                );
            }

            if (name === "registration_code") {
                next.registration_code = normalizeRegistrationCode(value,
prev.company_type);
            }

            if (name === "phone") {
                next.phone = normalizePhone(value);
            }

            return next;
        });

        if (["company_name", "company_type", "registration_code"].includes(name)) {
            setVerification(null);
        }
```

```
        clearFieldError(name);
        clearStatus();
    }

    function handleAuthChange(e) {
        const { name, type, value, checked } = e.target;

        setAuthForm((prev) => ({
            ...prev,
            [name]: type === "checkbox" ? checked : value,
        }));

        clearFieldError(name);
        clearStatus();
    }

    function validateCompanyStep() {
        const errors = {};
        const companyName = normalizeName(companyForm.company_name);
        const companyType = normalizeCompanyType(companyForm.company_type);
        const code = normalizeRegistrationCode(companyForm.registration_code, companyType);
        const currentSubjectType = getSubjectType(companyType);

        if (companyName.length < 2) {
            errors.company_name =
                currentSubjectType === "fop"
                    ? "🪪 Вкажіть ПІБ ФОП. Реєстр любить повні імена."
                    : "🏢 Вкажіть назву компанії. Без імені реєстр соромиться.";
        }

        if (companyType.length < 2) {
            errors.company_type = "🧬 Вкажіть тип: ТОВ, ФОП, ПП, АТ або інший.";
        }

        if (!companyForm.industry.trim()) {
            errors.industry = "🧭 Оберіть сферу діяльності. Бізнес без сфери трохи
підозрілий.";
        }

        if (currentSubjectType === "fop") {
            if (code.length !== 10) {
                errors.registration_code = "🔢 ІПН ФОП має містити рівно 10 цифр.";
            }
        } else if (code.length !== 8) {
            errors.registration_code = "🔢 ЄДРПОУ має містити рівно 8 цифр.";
        }

        if (!isValidWebsite(companyForm.website)) {
            errors.website = "🌍 Сайт має починатися з http:// або https://.";
        }

        if (verification?.status !== "verified") {
            errors.verification =
                "🧐 Спочатку натисніть «Перевірити компанію». Реєстр хоче сказати своє
слово.";
        }

        setFieldErrors((prev) => ({ ...prev, ...errors }));
        return Object.keys(errors).length === 0;
    }

    function validateContactStep() {
        const errors = {};

        if (normalizeName(companyForm.contact_name).length < 2) {
            errors.contact_name = "👤 Вкажіть контактну особу. Роботам теж треба знати, кому
писати.";
        }
```

```
        if (!isValidUkrainianPhone(companyForm.phone)) {
            errors.phone = "📞 Номер виглядає дивно. Приклад: +380991112233 або
0991112233.";
        }

        if (normalizeName(companyForm.city).length < 2) {
            errors.city = "📍 Вкажіть місто. Карта без точки сумує.";
        }

        setFieldErrors((prev) => ({ ...prev, ...errors }));
        return Object.keys(errors).length === 0;
    }

    function validateAccessStep() {
        const errors = {};

        if (!oauthReady) {
            if (!authForm.email.trim()) {
                errors.email = "📧 Без email акаунт не створимо.";
            } else if (!isValidEmail(authForm.email)) {
                errors.email = "📧 Email виглядає підозріло. Наприклад: company@email.com";
            }

            if (!authForm.password) {
                errors.password = "🔐 Придумайте пароль. Тільки не 123456, він уже
втомився.";
            } else if (!isStrongPassword(authForm.password)) {
                errors.password = "🔐 Пароль слабенький, як чай без заварки.";
            }

            if (!authForm.confirmPassword) {
                errors.confirmPassword = "🙈 Повторіть пароль ще раз.";
            } else if (authForm.password !== authForm.confirmPassword) {
                errors.confirmPassword = "🙈 Паролі розбіглися. Введіть однакові.";
            }
        }

        if (!authForm.agree) {
            errors.agree = "📝 Без галочки ніяк. Юристи вже хвилюються.";
        }

        setFieldErrors((prev) => ({ ...prev, ...errors }));
        return Object.keys(errors).length === 0;
    }

    function validateCompanyInfo({ requireVerification = false } = {}) {
        const companyName = normalizeName(companyForm.company_name);
        const companyType = normalizeCompanyType(companyForm.company_type);
        const industry = companyForm.industry.trim();
        const currentSubjectType = getSubjectType(companyType);
        const code = normalizeRegistrationCode(companyForm.registration_code, companyType);

        if (companyName.length < 2) throw new Error("Вкажіть назву компанії або ПІБ ФОП.");
        if (companyType.length < 2) throw new Error("Вкажіть тип компанії.");
        if (!industry) throw new Error("Оберіть сферу діяльності компанії.");

        if (currentSubjectType === "fop" && code.length !== 10) {
            throw new Error("Для ФОП потрібно вказати ІПН з 10 цифр.");
        }

        if (currentSubjectType === "company" && code.length !== 8) {
            throw new Error("ЄДРПОУ має містити рівно 8 цифр.");
        }

        if (!isValidWebsite(companyForm.website)) {
            throw new Error("Сайт має починатися з http:// або https://.");
        }
```

```
        if (requireVerification && verification?.status !== "verified") {
            throw new Error(verification?.reason || "Спочатку натисніть «Перевірити
компанію».");
        }

        return {
            company_name: companyName,
            company_type: companyType,
            subject_type: currentSubjectType,
            registration_code: code,
            industry,
        };
    }

    function buildCompanyPayload() {
        const companyInfo = validateCompanyInfo({ requireVerification: true });

        return {
            role: "company",
            ...companyInfo,
            website: companyForm.website.trim() || "",
            description: companyForm.description.trim() || null,
            contact_name: normalizeName(companyForm.contact_name),
            phone: companyForm.phone.trim(),
            city: normalizeName(companyForm.city),
            verification,
        };
    }

    async function handleVerifyCompany() {
        try {
            setVerifyLoading(true);
            setError("");
            setMessage("");
            setVerification(null);

            const companyName = normalizeName(companyForm.company_name);
            const companyType = normalizeCompanyType(companyForm.company_type);
            const currentSubjectType = getSubjectType(companyType);
            const code = normalizeRegistrationCode(companyForm.registration_code,
companyType);

            const errors = {};

            if (companyName.length < 2) errors.company_name = "🏢 Вкажіть назву або ПІБ
ФОП.";
            if (companyType.length < 2) errors.company_type = "🧬 Вкажіть тип компанії.";
            if (currentSubjectType === "fop" && code.length !== 10) {
                errors.registration_code = "🔢 ІПН ФОП має містити 10 цифр.";
            }
            if (currentSubjectType === "company" && code.length !== 8) {
                errors.registration_code = "🔢 ЄДРПОУ має містити 8 цифр.";
            }

            if (Object.keys(errors).length) {
                setFieldErrors((prev) => ({ ...prev, ...errors }));
                return;
            }

            const response = await apiFetch("/auth/verify-company", {
                method: "POST",
                skipAuth: true,
                body: {
                    company_name: companyName,
                    company_type: companyType,
                    subject_type: currentSubjectType,
                    registration_code: code,
                },
            });
```

```
            const nextVerification = response?.verification;

            if (!nextVerification) {
                throw new Error("Сервер не повернув результат перевірки.");
            }

            setVerification(nextVerification);

            if (nextVerification.status !== "verified") {
                setFieldErrors((prev) => ({
                    ...prev,
                    verification:
                        nextVerification.reason ||
                        "Реєстр не підтвердив дані. Перевірте назву і код.",
                }));
                return;
            }

            setMessage(
                nextVerification.subject_type === "fop"
                    ? "ФОП знайдено в реєстрі. Податкова махнула рукою ✅"
                    : "Компанію знайдено в реєстрі. Бюрократичний дракон переможений ✅"
            );
        } catch (err) {
            const verificationFromError =
                err?.verification || err?.data?.verification || err?.response?.verification
|| null;

            if (verificationFromError) {
                setVerification(verificationFromError);
                setFieldErrors((prev) => ({
                    ...prev,
                    verification:
                        verificationFromError.reason ||
                        "Реєстр підозріло мовчить. Перевірте дані ще раз.",
                }));
                return;
            }

            setError(getErrorText(err));
        } finally {
            setVerifyLoading(false);
        }
    }

    function goToStep(nextStep) {
        setError("");
        setMessage("");

        if (nextStep >= 2 && !validateCompanyStep()) return;
        if (nextStep >= 3 && !validateContactStep()) return;

        setStep(nextStep);
    }

    async function getSessionAccessToken() {
        const { data, error } = await supabase.auth.getSession();

        if (error) throw error;

        if (!data?.session?.access_token) {
            throw new Error("Не вдалося отримати access token.");
        }

        return data.session.access_token;
    }

    async function handleOAuth(provider) {
        try {
            setOauthLoading(true);
```

```
            setError("");
            setMessage("");

            if (!validateCompanyStep()) {
                setStep(1);
                return;
            }

            if (!validateContactStep()) {
                setStep(2);
                return;
            }

            if (!validateAccessStep()) {
                setStep(3);
                return;
            }

            const payload = buildCompanyPayload();

            savePendingCompanyRegistration(payload, authForm.email);

            await signInWithProvider(provider, "company");
        } catch (err) {
            setError(getErrorText(err));
            setOauthLoading(false);
        }
    }

    async function handleSubmit(e) {
        e.preventDefault();

        try {
            setLoading(true);
            setError("");
            setMessage("");

            if (!validateCompanyStep()) {
                setStep(1);
                return;
            }

            if (!validateContactStep()) {
                setStep(2);
                return;
            }

            if (!validateAccessStep()) {
                setStep(3);
                return;
            }

            const payload = buildCompanyPayload();
            savePendingCompanyRegistration(payload, authForm.email);

            let accessToken = null;

            if (!oauthReady) {
                const signUpResult = await signUp(
                    authForm.email.trim().toLowerCase(),
                    authForm.password,
                    { role: "company" }
                );

                if (signUpResult?.type === "already_registered") {
                    setFieldErrors((prev) => ({
                        ...prev,
                        email: "🐣 Цей email уже має акаунт. Просто увійдіть — він вас
чекає.",
                    }));
```

```
                    return;
                }

                const session = signUpResult?.session || signUpResult?.data?.session;

                if (!session?.access_token) {
                    setMessage(
                        "Акаунт створено. Підтвердіть email і потім увійдіть. Поштова
ракета вже полетіла 🚀"
                    );
                    return;
                }

                accessToken = session.access_token;
            } else {
                accessToken = await getSessionAccessToken();
            }

            localStorage.setItem("access_token", accessToken);
            localStorage.setItem("token", accessToken);

            const response = await apiFetch("/auth/complete-registration", {
                method: "POST",
                token: accessToken,
                body: payload,
            });

            await refreshMe();
            clearPendingCompanyRegistration();

            const companyVerification = response?.company_verification || verification;

            navigate("/company/dashboard", {
                replace: true,
                state: {
                    verificationStatus: companyVerification?.status || "verified",
                    verificationReason: companyVerification?.reason || null,
                },
            });
        } catch (err) {
            setError(getErrorText(err));
        } finally {
            setLoading(false);
        }
    }

    return (
        <div className="auth-shell">
            <div className="auth-container">
                <aside className="auth-aside">
                    <div className="auth-badge">Employer account</div>
                    <h1 className="auth-aside__title">Зареєструйте компанію</h1>
                    <p className="auth-aside__text">
                        Створіть акаунт роботодавця, підтвердьте компанію або ФОП
                        і отримайте доступ до публікації вакансій.
                    </p>
                </aside>

                <section className="auth-card">
                    <div className="auth-card__head">
                        <div>
                            <button
                                type="button"
                                className="auth-link-button"
                                onClick={() => navigate("/register")}
                            >
                                ← Назад
                            </button>
                            <h2>Реєстрація роботодавця</h2>
                        </div>
```

```
                        <p>
                            Уже маєте акаунт? <Link to="/login">Увійти</Link>
                        </p>
                    </div>

                    <CompanySteps step={step} />

                    <form className="auth-form" onSubmit={handleSubmit}>
                        {step === 1 && (
                            <CompanyInfoStep
                                companyForm={companyForm}
                                industries={INDUSTRIES}
                                registrationCode={registrationCode}
                                isCodeValid={isCodeValid}
                                subjectType={subjectType}
                                canVerifyCompany={canVerifyCompany}
                                canGoNext={canGoToContacts}
                                verifyLoading={verifyLoading}
                                verification={verification}
                                fieldErrors={fieldErrors}
                                onChange={handleCompanyChange}
                                onVerify={handleVerifyCompany}
                                onNext={() => goToStep(2)}
                            />
                        )}

                        {step === 2 && (
                            <ContactStep
                                companyForm={companyForm}
                                canGoNext={canGoToAuth}
                                fieldErrors={fieldErrors}
                                onChange={handleCompanyChange}
                                onBack={() => goToStep(1)}
                                onNext={() => goToStep(3)}
                            />
                        )}

                        {step === 3 && (
                            <AccessStep
                                authForm={authForm}
                                oauthReady={oauthReady}
                                oauthLoading={oauthLoading}
                                loading={loading}
                                canSubmit={canSubmit}
                                fieldErrors={fieldErrors}
                                showPassword={showPassword}
                                showConfirmPassword={showConfirmPassword}
                                onTogglePassword={() => setShowPassword((value) => !value)}
                                onToggleConfirmPassword={() =>
                                    setShowConfirmPassword((value) => !value)
                                }
                                onAuthChange={handleAuthChange}
                                onOAuth={handleOAuth}
                                onBack={() => goToStep(2)}
                            />
                        )}

                        {error && <div className="auth-alert auth-alert--
error">{error}</div>}

                        {message && !error && (
                            <div
                                className={
                                    verification?.status === "verified"
                                        ? "auth-alert auth-alert--success"
                                        : "auth-alert auth-alert--warning"
                                }
                            >
                                {message}
```

```
                            </div>
                        )}
                    </form>
                </section>
            </div>
        </div>
    );
}

function CompanySteps({ step }) {
    return (
        <div className="employer-steps">
            <StepItem number={1} label="Компанія" active={step === 1} done={step > 1} />
            <StepLine active={step > 1} />
            <StepItem number={2} label="Контакти" active={step === 2} done={step > 2} />
            <StepLine active={step > 2} />
            <StepItem number={3} label="Доступ" active={step === 3} done={false} />
        </div>
    );
}

function StepItem({ number, label, active, done }) {
    return (
        <div className="employer-step">
            <span
                className={
                    active || done
                        ? "employer-step__circle employer-step__circle--active"
                        : "employer-step__circle"
                }
            >
                {done ? "✓" : number}
            </span>

            <span
                className={
                    active
                        ? "employer-step__label employer-step__label--active"
                        : "employer-step__label"
                }
            >
                {label}
            </span>
        </div>
    );
}

function StepLine({ active }) {
    return (
        <span
            className={
                active
                    ? "employer-step__line employer-step__line--active"
                    : "employer-step__line"
            }
        />
    );
}

function CompanyInfoStep({
                             companyForm,
                             industries,
                             registrationCode,
                             isCodeValid,
                             subjectType,
                             canVerifyCompany,
                             canGoNext,
                             verifyLoading,
                             verification,
                             fieldErrors,
```

```
                        onChange,
                        onVerify,
                        onNext,
                    }) {
return (
    <>
        <div className="auth-section-title">
            <h3>Інформація про компанію</h3>
            <p>Вкажіть назву, тип, сферу діяльності та код для перевірки.</p>
        </div>

        <label className="auth-field">
            <span>{subjectType === "fop" ? "ПІБ ФОП" : "Назва компанії"}</span>
            <input
                name="company_name"
                placeholder='ТОВ "Приклад" або Іваненко Іван Іванович'
                value={companyForm.company_name}
                onChange={onChange}
                className={fieldErrors.company_name ? "input-error" : ""}
            />
            <FieldError message={fieldErrors.company_name} />
        </label>

        <label className="auth-field">
            <span>Тип</span>
            <input
                name="company_type"
                placeholder="ТОВ, ФОП, ПП, АТ"
                value={companyForm.company_type}
                onChange={onChange}
                className={fieldErrors.company_type ? "input-error" : ""}
            />
            <FieldError message={fieldErrors.company_type} />
        </label>

        <label className="auth-field">
            <span>Сфера діяльності</span>
            <select
                name="industry"
                value={companyForm.industry}
                onChange={onChange}
                className={fieldErrors.industry ? "input-error" : ""}
            >
                <option value="">Оберіть сферу діяльності</option>
                {industries.map((industry) => (
                    <option key={industry} value={industry}>
                        {industry}
                    </option>
                ))}
            </select>
            <FieldError message={fieldErrors.industry} />
        </label>

        <label className="auth-field">
            <span>{getRegistrationCodeLabel(companyForm.company_type)}</span>
            <input
                name="registration_code"
                value={companyForm.registration_code}
                onChange={onChange}
                inputMode="numeric"
                maxLength={subjectType === "fop" ? 10 : 8}
                className={fieldErrors.registration_code ? "input-error" : ""}
            />
            <small>{getRegistrationCodeHint(companyForm.company_type)}</small>
            <FieldError message={fieldErrors.registration_code} />
        </label>

        {registrationCode && !isCodeValid && (
            <p className="auth-warning">
                {subjectType === "fop"
```

```
            ? "ІПН ФОП має містити 10 цифр."
            : "ЄДРПОУ має містити 8 цифр."}
    </p>
)}

<button
    type="button"
    className="auth-button auth-button--secondary"
    onClick={onVerify}
    disabled={!canVerifyCompany}
>
    {verifyLoading
        ? "Перевірка..."
        : subjectType === "fop"
            ? "Перевірити ФОП"
            : "Перевірити компанію"}
</button>

<FieldError message={fieldErrors.verification} />

{verification && (
    <div
        className="verification-box"
        style={{ borderColor: getVerificationColor(verification.status) }}
    >
        <strong style={{ color: getVerificationColor(verification.status) }}>
            {getVerificationTitle(
                verification.status,
                verification.subject_type || subjectType
            )}
        </strong>

        <span>{getVerificationText(verification)}</span>

        {verification.company?.name && (
            <span>
                Назва в реєстрі: <strong>{verification.company.name}</strong>
            </span>
        )}

        {verification.company?.full_name && (
            <span>
                ПІБ ФОП: <strong>{verification.company.full_name}</strong>
            </span>
        )}

        {verification.company?.status && (
            <span>
                Статус: <strong>{verification.company.status}</strong>
            </span>
        )}
    </div>
)}

<label className="auth-field">
    <span>Сайт</span>
    <input
        name="website"
        placeholder="https://example.com"
        value={companyForm.website}
        onChange={onChange}
        className={fieldErrors.website ? "input-error" : ""}
    />
    <FieldError message={fieldErrors.website} />
</label>

<label className="auth-field">
    <span>Опис</span>
    <textarea
        name="description"
```

```
                    placeholder="Коротко опишіть компанію"
                    value={companyForm.description}
                    onChange={onChange}
                />
            </label>

            <div className="auth-actions auth-actions--right">
                <button
                    type="button"
                    className="auth-button auth-button--primary"
                    onClick={onNext}
                    disabled={!canGoNext}
                >
                    Продовжити
                </button>
            </div>
        </>
    );
}

function ContactStep({ companyForm, canGoNext, fieldErrors, onChange, onBack, onNext }) {
    return (
        <>
            <div className="auth-section-title">
                <h3>Контактна особа</h3>
                <p>Ці дані потрібні для зв’язку з роботодавцем.</p>
            </div>

            <label className="auth-field">
                <span>Ім’я контактної особи</span>
                <input
                    name="contact_name"
                    placeholder="Ім’я та прізвище"
                    value={companyForm.contact_name}
                    onChange={onChange}
                    className={fieldErrors.contact_name ? "input-error" : ""}
                />
                <FieldError message={fieldErrors.contact_name} />
            </label>

            <label className="auth-field">
                <span>Телефон</span>
                <input
                    name="phone"
                    placeholder="+380991112233"
                    value={companyForm.phone}
                    onChange={onChange}
                    className={fieldErrors.phone ? "input-error" : ""}
                />
                <FieldError message={fieldErrors.phone} />
            </label>

            <label className="auth-field">
                <span>Місто</span>
                <input
                    name="city"
                    placeholder="Київ"
                    value={companyForm.city}
                    onChange={onChange}
                    className={fieldErrors.city ? "input-error" : ""}
                />
                <FieldError message={fieldErrors.city} />
            </label>

            <div className="auth-actions">
                <button type="button" className="auth-button auth-button--secondary"
onClick={onBack}>
                    Назад
                </button>
```

```
                <button
                    type="button"
                    className="auth-button auth-button--primary"
                    onClick={onNext}
                    disabled={!canGoNext}
                >
                    Продовжити
                </button>
            </div>
        </>
    );
}

function AccessStep({
                        authForm,
                        oauthReady,
                        oauthLoading,
                        loading,
                        canSubmit,
                        fieldErrors,
                        showPassword,
                        showConfirmPassword,
                        onTogglePassword,
                        onToggleConfirmPassword,
                        onAuthChange,
                        onOAuth,
                        onBack,
                    }) {
    const passwordsMatch = authForm.confirmPassword && authForm.password ===
authForm.confirmPassword;

    return (
        <>
            <div className="auth-section-title">
                <h3>Дані для входу</h3>
                <p>Створіть доступ до кабінету роботодавця.</p>
            </div>

            {!oauthReady && (
                <>
                    <label className="auth-field">
                        <span>Email</span>
                        <input
                            type="email"
                            name="email"
                            placeholder="company@email.com"
                            value={authForm.email}
                            onChange={onAuthChange}
                            className={fieldErrors.email ? "input-error" : ""}
                        />
                        <FieldError message={fieldErrors.email} />
                    </label>

                    <PasswordField
                        label="Пароль"
                        name="password"
                        value={authForm.password}
                        show={showPassword}
                        error={fieldErrors.password}
                        onToggle={onTogglePassword}
                        onChange={onAuthChange}
                    />

                    <PasswordChecklist password={authForm.password} />

                    <PasswordField
                        label="Повторіть пароль"
                        name="confirmPassword"
                        value={authForm.confirmPassword}
                        show={showConfirmPassword}
```

```
                        error={fieldErrors.confirmPassword}
                        success={passwordsMatch ? "✅ Паролі збіглися. Вони подружилися." :
""}
                        onToggle={onToggleConfirmPassword}
                        onChange={onAuthChange}
                    />
                </>
            )}

            <label className="auth-checkbox">
                <input
                    type="checkbox"
                    name="agree"
                    checked={authForm.agree}
                    onChange={onAuthChange}
                />
                <span>Я погоджуюсь з правилами користування та політикою
конфіденційності</span>
            </label>
            <FieldError message={fieldErrors.agree} />

            {!oauthReady && (
                <div className="auth-socials">
                    <button
                        type="button"
                        className="auth-button auth-button--secondary"
                        onClick={() => onOAuth("google")}
                        disabled={oauthLoading || !authForm.agree}
                    >
                        {oauthLoading ? "Переадресація..." : "Продовжити через Google"}
                    </button>

                    <button
                        type="button"
                        className="auth-button auth-button--secondary"
                        onClick={() => onOAuth("linkedin_oidc")}
                        disabled={oauthLoading || !authForm.agree}
                    >
                        {oauthLoading ? "Переадресація..." : "Продовжити через LinkedIn"}
                    </button>
                </div>
            )}

            <div className="auth-actions">
                <button type="button" className="auth-button auth-button--secondary"
onClick={onBack}>
                    Назад
                </button>

                <button type="submit" className="auth-button auth-button--primary"
disabled={!canSubmit}>
                    {loading ? "Створення акаунта..." : "Зареєструвати компанію"}
                </button>
            </div>
        </>
    );
}

function PasswordField({
                           label,
                           name,
                           value,
                           show,
                           error,
                           success,
                           onToggle,
                           onChange,
                       }) {
    return (
        <label className="auth-field auth-field--stable">
```

```
            <span>{label}</span>

            <div className="password-input-wrap">
                <input
                    type={show ? "text" : "password"}
                    name={name}
                    placeholder={label}
                    value={value}
                    onChange={onChange}
                    autoComplete="new-password"
                    className={error ? "input-error" : ""}
                />

                <button
                    type="button"
                    className="password-toggle"
                    onClick={onToggle}
                    aria-label={show ? "Сховати пароль" : "Показати пароль"}
                >
                    {show ? "🙈" : "👁"}
                </button>
            </div>

            {success ? <FieldSuccess message={success} /> : <FieldError message={error} />}
        </label>
    );
}

function FieldError({ message }) {
    return (
        <small className={message ? "field-error" : "field-error field-error--empty"}>
            {message || " "}
        </small>
    );
}

function FieldSuccess({ message }) {
    return (
        <small className={message ? "field-success" : "field-success field-success--
empty"}>
            {message || " "}
        </small>
    );
}

function PasswordChecklist({ password }) {
    const strength = getPasswordStrength(password);

    const checks = [
        { label: "Мінімум 8 символів", valid: password.length >= 8 },
        { label: "Велика літера", valid: /[A-ZА-ЯІЇЄҐ]/.test(password) },
        { label: "Мала літера", valid: /[a-za-яіїєґ]/i.test(password) },
        { label: "Цифра", valid: /\d/.test(password) },
        {
            label: "Спецсимвол — бонус до броні",
            valid: /[^A-Za-zA-Яа-яІіЇїЄєҐґ0-9]/.test(password),
        },
    ];

    return (
        <div className={password ? "password-panel" : "password-panel password-panel--
empty"}>
            {password ? (
                <>
                    <div className="password-strength">
                        <div className="password-strength__top">
                            <span>Сила пароля</span>
                            <strong>{strength.label}</strong>
                        </div>
```

```
                        <div className={`password-strength__bar ${strength.className}`}>
                            <span style={{ width: `${Math.min(strength.score, 5) * 20}%` }}
/>
                        </div>
                    </div>

                    <ul className="password-checklist">
                        {checks.map((item) => (
                            <li
                                key={item.label}
                                className={
                                    item.valid
                                        ? "password-checklist__item is-valid"
                                        : "password-checklist__item"
                                }
                            >
                                <span className="password-checklist__dot">
                                    {item.valid ? "✓" : ""}
                                </span>
                                {item.label}
                            </li>
                        ))}
                    </ul>
                </>
            ) : (
                <span className="password-panel__placeholder">
                    🔐 Тут з’явиться сила пароля, коли ви його введете
                </span>
            )}
        </div>
    );
}
import { useEffect, useMemo, useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import { useAuth } from "../../../context/AuthContext";
import { supabase } from "../../../services/supabaseClient";
import "../../../styles/registration.css";

const CANDIDATE_ROLE = "student";

function isValidEmail(email) {
    return /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(String(email || "").trim());
}

function isStrongPassword(password) {
    return (
        password.length >= 8 &&
        /[a-za-яіїєґ]/i.test(password) &&
        /[A-ZA-ЯІЇЄҐ]/.test(password) &&
        /\d/.test(password)
    );
}

function getPasswordStrength(password = "") {
    let score = 0;

    if (password.length >= 8) score += 1;
    if (/[A-ZA-ЯІЇЄҐ]/.test(password)) score += 1;
    if (/[a-za-яіїєґ]/i.test(password)) score += 1;
    if (/\d/.test(password)) score += 1;
    if (/[^A-Za-zA-Яа-яІіЇїЄєҐґ0-9]/.test(password)) score += 1;

    if (!password) {
        return {
            score: 0,
            label: "Пароль ще не придумали",
            className: "is-empty",
        };
    }
```

```
    if (score <= 2) {
        return {
            score,
            label: "Слабенький, як чай без заварки 🫖",
            className: "is-weak",
        };
    }

    if (score <= 4) {
        return {
            score,
            label: "Вже краще, пароль качається 💪",
            className: "is-medium",
        };
    }

    return {
        score,
        label: "Ого, це пароль-бодібілдер 🦾",
        className: "is-strong",
    };
}

function getErrorText(error = "") {
    const raw =
        error?.message ||
        error?.error ||
        error?.data?.error ||
        error?.response?.error ||
        String(error || "");

    const lower = raw.toLowerCase();

    if (
        lower.includes("user already registered") ||
        lower.includes("already registered") ||
        lower.includes("already exists") ||
        lower.includes("duplicate key") ||
        lower.includes("23505")
    ) {
        return `
🐣 А цей email уже вилупився

Акаунт із такою поштою вже існує.
Не треба реєструватися вдруге — просто увійдіть.
        `;
    }

    if (
        lower.includes("email rate limit exceeded") ||
        lower.includes("rate limit") ||
        lower.includes("too many requests")
    ) {
        return `
🦕 Трошки повільніше

Ми вже надіслали багато листів.
Дайте поштовому голубу хвилинку перепочити й спробуйте пізніше.
        `;
    }

    if (
        lower.includes("error sending confirmation email") ||
        lower.includes("smtp")
    ) {
        return `
🥊 Лист застряг у поштовій трубі

Не вдалося надіслати лист підтвердження.
```

```
Схоже, SMTP у Supabase пішов на каву.
        `;
    }

    if (
        lower.includes("network") ||
        lower.includes("fetch") ||
        lower.includes("failed to fetch") ||
        lower.includes("server") ||
        lower.includes("backend") ||
        lower.includes("сервер")
    ) {
        return `
🌐 Сервер десь загубився

Перевірте:
• чи запущений backend
• чи працює інтернет
• чи правильний API URL

Ми теж його шукаємо 👀
        `;
    }

    if (lower.includes("oauth") || lower.includes("provider")) {
        return `
🔑 Соцмережа не пустила нас у гості

Спробуйте ще раз або зареєструйтесь через email і пароль.
        `;
    }

    if (
        lower.includes("jwt") ||
        lower.includes("token") ||
        lower.includes("session")
    ) {
        return `
🪪 Перепустка протермінована

Сесія завершилась.
Увійдіть ще раз — і все знову запрацює.
        `;
    }

    return `
😅 Щось пішло не за планом

${raw || "Спробуйте ще раз. Якщо не вийде — зробимо вигляд, що так і було."}
    `;
}

function ErrorAlert({ error }) {
    if (!error) return null;

    return (
        <div className="auth-alert auth-alert--error">
            {error
                .split("\n")
                .filter((line) => line.trim())
                .map((line, index) => (
                    <p key={index}>{line}</p>
                ))}
        </div>
    );
}

function SuccessAlert({ message }) {
    if (!message) return null;
```

```
    return <div className="auth-alert auth-alert--success">{message}</div>;
}

function FieldError({ message }) {
    return (
        <small className={message ? "field-error" : "field-error field-error--empty"}>
            {message || " "}
        </small>
    );
}

function FieldSuccess({ message }) {
    return (
        <small className={message ? "field-success" : "field-success field-success--
empty"}>
            {message || " "}
        </small>
    );
}

function PasswordChecklist({ password }) {
    const strength = getPasswordStrength(password);

    const checks = useMemo(
        () => [
            { label: "Мінімум 8 символів", valid: password.length >= 8 },
            {
                label: "Велика літера",
                valid: /[A-ZA-ЯІЇЄҐ]/.test(password),
            },
            {
                label: "Мала літера",
                valid: /[a-za-яіїєґ]/i.test(password),
            },
            { label: "Цифра", valid: /\d/.test(password) },
            {
                label: "Спецсимвол — бонус до броні",
                valid: /[^A-Za-zA-Яа-яІіЇїЄєҐг0-9]/.test(password),
            },
        ],
        [password]
    );

    return (
        <div className={password ? "password-panel" : "password-panel password-panel--
empty"}>
            {password ? (
                <>
                    <div className="password-strength">
                        <div className="password-strength__top">
                            <span>Сила пароля</span>
                            <strong>{strength.label}</strong>
                        </div>

                        <div className={`password-strength__bar ${strength.className}`}>
                            <span
                                style={{
                                    width: `${Math.min(strength.score, 5) * 20}%`,
                                }}
                            />
                        </div>
                    </div>

                    <ul className="password-checklist">
                        {checks.map((item) => (
                            <li
                                key={item.label}
                                className={
                                    item.valid
```

```
                                        ? "password-checklist__item is-valid"
                                        : "password-checklist__item"
                                    }
                                >
                                    <span className="password-checklist__dot">
                                        {item.valid ? "✓" : ""}
                                    </span>
                                    {item.label}
                                </li>
                            ))}
                        </ul>
                    </>
                ) : (
                    <span className="password-panel__placeholder">
                        🔐 Тут з’явиться сила пароля, коли ви його введете
                    </span>
                )}
            </div>
        );
}

export default function CandidateRegisterPage() {
    const navigate = useNavigate();
    const auth = useAuth();

    const [form, setForm] = useState({
        first_name: "",
        last_name: "",
        email: "",
        password: "",
        confirmPassword: "",
    });

    const [agree, setAgree] = useState(false);
    const [fieldErrors, setFieldErrors] = useState({});

    const [showPassword, setShowPassword] = useState(false);
    const [showConfirmPassword, setShowConfirmPassword] = useState(false);

    const [loading, setLoading] = useState(false);
    const [oauthLoading, setOauthLoading] = useState(false);
    const [resendLoading, setResendLoading] = useState(false);

    const [error, setError] = useState("");
    const [message, setMessage] = useState("");
    const [signupCompleted, setSignupCompleted] = useState(false);

    const [canResendAt, setCanResendAt] = useState(0);
    const [remainingSeconds, setRemainingSeconds] = useState(0);

    useEffect(() => {
        if (auth.loading) return;

        if (!auth.user) return;

        if (auth.role === "company") {
            navigate("/company/dashboard", { replace: true });
            return;
        }

        if (auth.role === "admin") {
            navigate("/admin", { replace: true });
            return;
        }

        if (
            auth.role === "student" ||
            auth.role === "candidate" ||
            auth.role === "jobseeker" ||
            auth.role === "user" ||
```

```
            !auth.role
        ) {
            navigate("/profile", { replace: true });
        }
    }, [auth.loading, auth.user, auth.role, navigate]);

    useEffect(() => {
        if (!canResendAt) {
            setRemainingSeconds(0);
            return;
        }

        const updateCountdown = () => {
            const diff = Math.max(0, Math.ceil((canResendAt - Date.now()) / 1000));
            setRemainingSeconds(diff);

            if (diff <= 0) {
                setCanResendAt(0);
            }
        };

        updateCountdown();

        const timer = setInterval(updateCountdown, 1000);

        return () => clearInterval(timer);
    }, [canResendAt]);

    function clearStatus() {
        setError("");
        setMessage("");
        setFieldErrors({});
    }

    function handleChange(event) {
        const { name, value } = event.target;

        setForm((prev) => ({
            ...prev,
            [name]: name === "email" ? value.trimStart() : value,
        }));

        setError("");
        setMessage("");

        setFieldErrors((prev) => ({
            ...prev,
            [name]: "",
        }));
    }

    function validateFields() {
        const errors = {};

        const firstName = form.first_name.trim();
        const lastName = form.last_name.trim();
        const email = form.email.trim().toLowerCase();

        if (!firstName) {
            errors.first_name = "👋 А як вас звати? Введіть ім’я.";
        }

        if (!lastName) {
            errors.last_name = "🪪 Прізвище теж потрібне. Формальності, але важливі.";
        }

        if (!email) {
            errors.email = "📧 Без email ми не зможемо створити акаунт.";
        } else if (!isValidEmail(email)) {
            errors.email = "📧 Email виглядає підозріло. Наприклад: name@email.com";
```

```
    }

    if (!form.password) {
        errors.password = "🔐 Придумайте пароль. Тільки не 123456, він уже втомився.";
    } else if (!isStrongPassword(form.password)) {
        errors.password = "🔐 Пароль слабенький, як чай без заварки.";
    }

    if (!form.confirmPassword) {
        errors.confirmPassword = "🙈 Повторіть пароль ще раз.";
    } else if (form.password !== form.confirmPassword) {
        errors.confirmPassword = "🙈 Паролі розбіглися. Введіть однакові.";
    }

    if (!agree) {
        errors.agree = "📝 Без галочки ніяк. Юристи вже хвилюються.";
    }

    setFieldErrors(errors);

    return Object.keys(errors).length === 0;
}

async function handleOAuth(provider) {
    try {
        clearStatus();

        if (!agree) {
            setFieldErrors({
                agree: "📝 Без галочки ніяк. Погодьтеся з умовами, і рушимо далі.",
            });
            return;
        }

        if (typeof auth.signInWithProvider !== "function") {
            throw new Error("OAuth provider is not configured.");
        }

        setOauthLoading(true);

        localStorage.setItem(
            "user_signup_draft",
            JSON.stringify({
                role: CANDIDATE_ROLE,
                provider,
                next: "/profile",
            })
        );

        await auth.signInWithProvider(provider);
    } catch (err) {
        setError(getErrorText(err));
        setOauthLoading(false);
    }
}

async function handleSubmit(event) {
    event.preventDefault();

    try {
        setLoading(true);
        setError("");
        setMessage("");
        setFieldErrors({});

        if (!validateFields()) {
            setLoading(false);
            return;
        }
```

```
            const firstName = form.first_name.trim();
            const lastName = form.last_name.trim();
            const email = form.email.trim().toLowerCase();
            const fullName = `${firstName} ${lastName}`.trim();

            localStorage.setItem(
                "user_signup_draft",
                JSON.stringify({
                    full_name: fullName,
                    email,
                    role: CANDIDATE_ROLE,
                    next: "/profile",
                })
            );

            const signUpData = await auth.signUp(email, form.password, {
                role: CANDIDATE_ROLE,
                full_name: fullName,
            });

            if (
                signUpData?.type === "already_registered" ||
                signUpData?.user?.identities?.length === 0
            ) {
                setFieldErrors({
                    email: "🐣 Цей email уже має акаунт. Просто увійдіть — він вас чекає.",
                });

                return;
            }

            if (signUpData?.session?.access_token) {
                await auth.completeUserProfile({
                    role: CANDIDATE_ROLE,
                    full_name: fullName,
                    email,
                });

                setMessage("Акаунт створено. Перенаправляємо у профіль...");
                navigate("/profile", { replace: true });
                return;
            }

            setSignupCompleted(true);
            setMessage(
                "Акаунт створено. Ми надіслали лист для підтвердження email. Відкрийте лист
і перейдіть за посиланням, після цього відкриється ваш профіль."
            );
            setCanResendAt(Date.now() + 60_000);
            setRemainingSeconds(60);
        } catch (err) {
            const friendly = getErrorText(err);

            if (
                String(err?.message || "")
                    .toLowerCase()
                    .includes("already")
            ) {
                setFieldErrors({
                    email: "🐣 Цей email уже зареєстрований. Спробуйте увійти.",
                });
            } else {
                setError(friendly);
            }
        } finally {
            setLoading(false);
        }
    }
```

```
    async function handleResend() {
        try {
            setResendLoading(true);
            setError("");
            setMessage("");
            setFieldErrors({});

            const email = form.email.trim().toLowerCase();

            if (!email) {
                setFieldErrors({
                    email: "📧 Введіть email, щоб ми знали, куди відправляти лист.",
                });
                return;
            }

            if (!isValidEmail(email)) {
                setFieldErrors({
                    email: "📧 Email виглядає дивно. Перевірте адресу.",
                });
                return;
            }

            if (remainingSeconds > 0) {
                throw new Error(`Повторно надіслати лист можна через ${remainingSeconds}
секунд.`);
            }

            const { error: resendError } = await supabase.auth.resend({
                type: "signup",
                email,
                options: {
                    emailRedirectTo:
`${window.location.origin}/auth/callback?role=${CANDIDATE_ROLE}&next=/profile`,
                },
            });

            if (resendError) throw resendError;

            setMessage("Лист підтвердження надіслано повторно. Поштова ракета вже в дорозі
🚀");
            setCanResendAt(Date.now() + 60_000);
            setRemainingSeconds(60);
        } catch (err) {
            setError(getErrorText(err));
        } finally {
            setResendLoading(false);
        }
    }

    const resendDisabled = resendLoading || loading || remainingSeconds > 0;

    const passwordsMatch =
        form.confirmPassword && form.password === form.confirmPassword;

    return (
        <div className="auth-shell">
            <div className="auth-container">
                <aside className="auth-aside">
                    <div className="auth-badge">Кандидат</div>

                    <h1 className="auth-aside__title">
                        Зареєструйтесь і почніть користуватись платформою
                    </h1>

                    <p className="auth-aside__text">
                        Створіть акаунт, підтвердьте email і відкрийте свій профіль.
                        Резюме можна створити пізніше, коли буде зручно.
                    </p>
```

```
                    <div className="auth-benefits">
                        <div className="auth-benefit">
                            <span className="auth-benefit__dot" />
                            <div>
                                <strong>Швидкий старт</strong>
                                <p>Реєстрація через email або Google.</p>
                            </div>
                        </div>

                        <div className="auth-benefit">
                            <span className="auth-benefit__dot" />
                            <div>
                                <strong>Персональний профіль</strong>
                                <p>Після реєстрації відкривається саме профіль.</p>
                            </div>
                        </div>

                        <div className="auth-benefit">
                            <span className="auth-benefit__dot" />
                            <div>
                                <strong>Резюме пізніше</strong>
                                <p>Створення резюме запускається тільки вручну з
профілю.</p>
                            </div>
                        </div>
                    </div>
                </aside>

                <section className="auth-card">
                    <div className="auth-card__head">
                        <h2>Створення акаунта</h2>
                        <p>
                            Уже маєте акаунт? <Link to="/login">Увійти</Link>
                        </p>
                    </div>

                    {signupCompleted ? (
                        <div className="auth-success-card">
                            <h3>Підтвердіть email</h3>

                            <p>{message}</p>

                            <ErrorAlert error={error} />

                            <div className="auth-success-card__actions">
                                <button
                                    type="button"
                                    className="auth-button auth-button--primary"
                                    onClick={handleResend}
                                    disabled={resendDisabled}
                                >
                                    {remainingSeconds > 0
                                        ? `Надіслати повторно через ${remainingSeconds} с`
                                        : resendLoading
                                            ? "Надсилаємо..."
                                            : "Надіслати лист повторно"}
                                </button>

                                <Link
                                    className="auth-button auth-button--ghost"
                                    to="/login"
                                    state={{
                                        email: form.email.trim().toLowerCase(),
                                        registered: true,
                                    }}
                                >
                                    Перейти до входу
                                </Link>
                            </div>
                        </div>
```

```
                    ) : (
                        <form className="auth-form" onSubmit={handleSubmit}>
                            <div className="auth-socials">
                                <button
                                    type="button"
                                    className="auth-button auth-button--social"
                                    onClick={() => handleOAuth("google")}
                                    disabled={oauthLoading}
                                >
                                    {oauthLoading
                                        ? "Переадресація..."
                                        : "Продовжити через Google"}
                                </button>
                            </div>

                            <div className="auth-divider">
                                <span />
                                <small>або зареєструйтесь через email</small>
                                <span />
                            </div>

                            <div className="auth-grid auth-grid--2 auth-grid--stable">
                                <div className="auth-field auth-field--stable">
                                    <label htmlFor="first_name">Ім’я</label>
                                    <input
                                        id="first_name"
                                        name="first_name"
                                        type="text"
                                        placeholder="Іван"
                                        value={form.first_name}
                                        onChange={handleChange}
                                        autoComplete="given-name"
                                        className={fieldErrors.first_name ? "input-error" :
""}
                                    />
                                    <FieldError message={fieldErrors.first_name} />
                                </div>

                                <div className="auth-field auth-field--stable">
                                    <label htmlFor="last_name">Прізвище</label>
                                    <input
                                        id="last_name"
                                        name="last_name"
                                        type="text"
                                        placeholder="Петренко"
                                        value={form.last_name}
                                        onChange={handleChange}
                                        autoComplete="family-name"
                                        className={fieldErrors.last_name ? "input-error" :
""}
                                    />
                                    <FieldError message={fieldErrors.last_name} />
                                </div>
                            </div>

                            <div className="auth-field auth-field--stable">
                                <label htmlFor="email">Email</label>
                                <input
                                    id="email"
                                    name="email"
                                    type="email"
                                    placeholder="name@email.com"
                                    value={form.email}
                                    onChange={handleChange}
                                    autoComplete="email"
                                    className={fieldErrors.email ? "input-error" : ""}
                                />
                                <FieldError message={fieldErrors.email} />
                            </div>
```

```
                        <div className="auth-grid auth-grid--2 auth-grid--stable">
                            <div className="auth-field auth-field--stable">
                                <label htmlFor="password">Пароль</label>

                                <div className="password-input-wrap">
                                    <input
                                        id="password"
                                        name="password"
                                        type={showPassword ? "text" : "password"}
                                        placeholder="Створіть пароль"
                                        value={form.password}
                                        onChange={handleChange}
                                        autoComplete="new-password"
                                        className={fieldErrors.password ? "input-error"
: ""}
                                    />

                                    <button
                                        type="button"
                                        className="password-toggle"
                                        onClick={() => setShowPassword((prev) =>
!prev)}

                                        aria-label={
                                            showPassword
                                                ? "Сховати пароль"
                                                : "Показати пароль"
                                        }
                                    >
                                        {showPassword ? "🙈" : "👁"}
                                    </button>
                                </div>

                                <FieldError message={fieldErrors.password} />
                            </div>

                            <div className="auth-field auth-field--stable">
                                <label htmlFor="confirmPassword">Повторіть
пароль</label>

                                <div className="password-input-wrap">
                                    <input
                                        id="confirmPassword"
                                        name="confirmPassword"
                                        type={showConfirmPassword ? "text" :
"password"}

                                        placeholder="Повторіть пароль"
                                        value={form.confirmPassword}
                                        onChange={handleChange}
                                        autoComplete="new-password"
                                        className={
                                            fieldErrors.confirmPassword
                                                ? "input-error"
                                                : ""
                                        }
                                    />

                                    <button
                                        type="button"
                                        className="password-toggle"
                                        onClick={() =>
                                            setShowConfirmPassword((prev) => !prev)
                                        }
                                        aria-label={
                                            showConfirmPassword
                                                ? "Сховати пароль"
                                                : "Показати пароль"
                                        }
                                    >
                                        {showConfirmPassword ? "🙈" : "👁"}
                                    </button>
```

```
                                    </div>

                                    {passwordsMatch ? (
                                        <FieldSuccess message="✅ Паролі збіглися. Вони
подружилися." />
                                    ) : (
                                        <FieldError message={fieldErrors.confirmPassword}
/>
                                    )}
                                </div>
                            </div>

                            <PasswordChecklist password={form.password} />

                            <div className="checkbox-block">
                                <label className="auth-checkbox">
                                    <input
                                        type="checkbox"
                                        checked={agree}
                                        onChange={(e) => {
                                            setAgree(e.target.checked);
                                            setFieldErrors((prev) => ({
                                                ...prev,
                                                agree: "",
                                            }));
                                            setError("");
                                        }}
                                    />
                                    <span>
                                        Я погоджуюсь з умовами використання та політикою
                                        конфіденційності.
                                    </span>
                                </label>

                                <FieldError message={fieldErrors.agree} />
                            </div>

                            <ErrorAlert error={error} />
                            <SuccessAlert message={!error ? message : ""} />

                            <button
                                type="submit"
                                className="auth-button auth-button--primary"
                                disabled={loading}
                            >
                                {loading ? "Створюємо акаунт..." : "Зареєструватися"}
                            </button>
                        </form>
                    )}
                </section>
            </div>
        </div>
    );
}
import { useState } from "react";
import { supabase } from "../../../services/supabaseClient";
import { api } from "../../../services/api/api";
import "../../../styles/auth.css";

export default function ResumeUpload() {
    const [error, setError] = useState("");
    const [ok, setOk] = useState("");
    const [loading, setLoading] = useState(false);

    async function handleFile(e) {
        const file = e.target.files?.[0];
        if (!file) return;

        setError("");
        setOk("");
```

```
        const allowed = ["application/pdf", "application/msword",
            "application/vnd.openxmlformats-officedocument.wordprocessingml.document"];
        if (!allowed.includes(file.type)) return setError("Only PDF/DOC/DOCX allowed");
        if (file.size > 5 * 1024 * 1024) return setError("Max 5MB");

        setLoading(true);
        try {
            const { data } = await supabase.auth.getSession();
            const uid = data?.session?.user?.id;
            if (!uid) throw new Error("Not logged in");

            const ext = file.name.split(".").pop().toLowerCase();
            const path = `${uid}/resume.${ext}`;

            const { error: uploadError } = await supabase.storage
                .from("resumes")
                .upload(path, file, { upsert: true });

            if (uploadError) throw uploadError;

            await api.patch("/me/resume-file", { resume_path: path });
            setOk("Resume uploaded ✅");
        } catch (err) {
            setError(err.message);
        } finally {
            setLoading(false);
        }
    }

    return (
        <div className="auth-field">
            <label className="auth-label">Upload resume (PDF/DOC/DOCX)</label>
            <input className="auth-input" type="file" onChange={handleFile} />
            {loading && <div className="auth-hint">Uploading...</div>}
            {ok && <div className="auth-success">{ok}</div>}
            {error && <div className="auth-error">{error}</div>}
        </div>
    );
}
import { useEffect, useMemo, useState } from "react";
import { Link, useNavigate, useParams } from "react-router-dom";
import { apiFetch } from "../../../services/api/api";
import "../../../styles/article-details.css";

function formatDate(value) {
    if (!value) return "Дата не вказана";

    const date = new Date(value);

    if (Number.isNaN(date.getTime())) return "Дата не вказана";

    return date.toLocaleDateString("uk-UA", {
        day: "2-digit",
        month: "long",
        year: "numeric",
    });
}

function normalizeText(value) {
    return String(value || "").trim();
}

function getReadingTime(text) {
    const words = normalizeText(text)
        .split(/\s+/)
        .filter(Boolean).length;

    const minutes = Math.max(1, Math.ceil(words / 180));
```

```
    return `${minutes} хв читання`;
}

function normalizeTags(tags) {
    if (Array.isArray(tags)) return tags.filter(Boolean);

    if (typeof tags === "string") {
        return tags
            .split(",")
            .map((tag) => tag.trim())
            .filter(Boolean);
    }

    return [];
}

function normalizeList(response, key) {
    if (Array.isArray(response)) return response;
    if (Array.isArray(response?.[key])) return response[key];
    if (Array.isArray(response?.items)) return response.items;
    if (Array.isArray(response?.results)) return response.results;
    if (Array.isArray(response?.data)) return response.data;
    return [];
}

function getArticleImage(article) {
    return (
        article?.image_url ||
        article?.cover_url ||
        article?.cover_image ||
        article?.image ||
        ""
    );
}

function getArticleContent(article) {
    return (
        article?.content ||
        article?.body ||
        article?.description ||
        article?.text ||
        ""
    );
}

function getArticleId(article) {
    return article?.slug || article?.id || article?._id || "";
}

function getErrorMessage(error) {
    if (error?.response?.error) return error.response.error;
    if (error?.response?.message) return error.response.message;
    if (error?.error) return error.error;
    if (error?.message) return error.message;
    return "Не вдалося завантажити статтю";
}

export default function ArticlePage() {
    const { id, slug } = useParams();
    const navigate = useNavigate();

    const [article, setArticle] = useState(null);
    const [relatedArticles, setRelatedArticles] = useState([]);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState("");

    const articleId = id || slug;

    const content = useMemo(() => getArticleContent(article), [article]);
    const tags = useMemo(() => normalizeTags(article?.tags), [article]);
```

```
const image = useMemo(() => getArticleImage(article), [article]);

useEffect(() => {
    loadArticle();
    // eslint-disable-next-line react-hooks/exhaustive-deps
}, [articleId]);

async function loadArticle() {
    if (!articleId) {
        setLoading(false);
        setError("Не вказано ID статті");
        return;
    }

    try {
        setLoading(true);
        setError("");

        const response = await apiFetch(`/articles/${articleId}`);

        const loadedArticle =
            response?.article ||
            response?.item ||
            response?.data ||
            response;

        if (!loadedArticle || typeof loadedArticle !== "object") {
            setArticle(null);
            setError("Статтю не знайдено");
            return;
        }

        setArticle(loadedArticle);

        try {
            const category = loadedArticle?.category || "";

            const relatedResponse = await apiFetch(
                `/articles?limit=3&category=${encodeURIComponent(category)}`
            );

            const list = normalizeList(relatedResponse, "articles");

            const currentArticleId = getArticleId(loadedArticle);

            setRelatedArticles(
                list
                    .filter((item) => getArticleId(item) !== currentArticleId)
                    .slice(0, 3)
            );
        } catch (relatedError) {
            console.warn("Could not load related articles:", relatedError);
            setRelatedArticles([]);
        }
    } catch (requestError) {
        setArticle(null);
        setError(getErrorMessage(requestError));
    } finally {
        setLoading(false);
    }
}

if (loading) {
    return (
        <main className="article-details-page">
            <section className="article-details-loading">
                <div className="article-loader" />
                <h1>Завантажуємо статтю...</h1>
                <p>Будь ласка, зачекайте кілька секунд.</p>
            </section>
```

```
        </main>
    );
}

if (error || !article) {
    return (
        <main className="article-details-page">
            <section className="article-details-error">
                <span>!</span>
                <h1>Статтю не знайдено</h1>
                <p>
                    {error ||
                        "Можливо, статтю було видалено або посилання неправильне."}
                </p>

                <div className="article-error-actions">
                    <button type="button" onClick={() => navigate(-1)}>
                        Назад
                    </button>

                    <Link to="/articles">До всіх статей</Link>
                </div>
            </section>
        </main>
    );
}

return (
    <main className="article-details-page">
        <section className="article-details-hero">
            <button
                type="button"
                className="article-back-button"
                onClick={() => navigate(-1)}
            >
                ← Назад
            </button>

            <div className="article-hero-grid">
                <div className="article-hero-content">
                    <div className="article-meta-row">
                        {article.category ? (
                            <span className="article-category">
                                {article.category}
                            </span>
                        ) : null}

                        <span>
                            {formatDate(
                                article.created_at ||
                                article.published_at ||
                                article.createdAt
                            )}
                        </span>

                        <span>{getReadingTime(content)}</span>
                    </div>

                    <h1>{article.title || "Без назви"}</h1>

                    {article.excerpt || article.subtitle ? (
                        <p className="article-lead">
                            {article.excerpt || article.subtitle}
                        </p>
                    ) : null}

                    <div className="article-author-card">
                        <div className="article-author-avatar">
                            {(article.author_name || article.author || "A")
                                .slice(0, 1)
```

```
                        .toUpperCase()}
                </div>

                <div>
                    <span>Автор</span>
                    <strong>
                        {article.author_name ||
                            article.author ||
                            "Адміністрація платформи"}
                    </strong>
                </div>
            </div>
        </div>

        <div className="article-hero-image">
            {image ? (
                <img
                    src={image}
                    alt={article.title || "Зображення статті"}
                />
            ) : (
                <div className="article-image-placeholder">
                    <span>Article</span>
                </div>
            )}
        </div>
    </div>
</section>

<section className="article-details-layout">
    <aside className="article-details-sidebar">
        <div className="article-sidebar-card">
            <p className="article-sidebar-label">Інформація</p>

            <div className="article-sidebar-item">
                <span>Дата</span>
                <strong>
                    {formatDate(
                        article.created_at ||
                        article.published_at ||
                        article.createdAt
                    )}
                </strong>
            </div>

            <div className="article-sidebar-item">
                <span>Час читання</span>
                <strong>{getReadingTime(content)}</strong>
            </div>

            {article.category ? (
                <div className="article-sidebar-item">
                    <span>Категорія</span>
                    <strong>{article.category}</strong>
                </div>
            ) : null}
        </div>

        {tags.length ? (
            <div className="article-sidebar-card">
                <p className="article-sidebar-label">Теги</p>

                <div className="article-tags">
                    {tags.map((tag) => (
                        <span key={tag}>{tag}</span>
                    ))}
                </div>
            </div>
        ) : null}
    </aside>
```

```
    <article className="article-content-card">
        {content ? (
            <div className="article-content">
                {content.split("\n").map((paragraph, index) => {
                    const text = paragraph.trim();

                    if (!text) return null;

                    if (text.startsWith("### ")) {
                        return (
                            <h3 key={`paragraph-${index}`}>
                                {text.replace("### ", "")}
                            </h3>
                        );
                    }

                    if (text.startsWith("## ")) {
                        return (
                            <h2 key={`paragraph-${index}`}>
                                {text.replace("## ", "")}
                            </h2>
                        );
                    }

                    return (
                        <p key={`paragraph-${index}`}>
                            {text}
                        </p>
                    );
                })}
            </div>
        ) : (
            <div className="article-empty-content">
                <h2>Контент статті поки відсутній</h2>
                <p>Автор ще не додав повний текст цієї статті.</p>
            </div>
        )}
    </article>
</section>

{relatedArticles.length ? (
    <section className="related-articles-section">
        <div className="related-articles-header">
            <p>Читайте також</p>
            <h2>Схожі статті</h2>
        </div>

        <div className="related-articles-grid">
            {relatedArticles.map((item) => (
                <Link
                    key={getArticleId(item)}
                    to={`/articles/${getArticleId(item)}`}
                    className="related-article-card"
                >
                    <div>
                        <span>{item.category || "Стаття"}</span>

                        <h3>{item.title || "Без назви"}</h3>

                        <p>
                            {item.excerpt ||
                                item.description ||
                                "Перейдіть, щоб прочитати більше."}
                        </p>
                    </div>

                    <strong>Читати →</strong>
                </Link>
            ))}
```

```
                    </div>
                </section>
            ) : null}
        </main>
    );
}
import { useEffect, useMemo, useState } from "react";
import { Link, useNavigate } from "react-router-dom";

import Header from "../../../components/layout/Header";
import Footer from "../../../components/layout/Footer";
import { apiFetch } from "../../../services/api/api";
import { useAuth } from "../../../context/AuthContext";

import "./JobsPage.css";

function slugify(value) {
    return String(value || "")
        .toLowerCase()
        .trim()
        .replace(/\s+/g, "-")
        .replace(/[^\w-]+/g, "");
}

function formatDate(value) {
    if (!value) return "Дата не вказана";

    const date = new Date(value);

    if (Number.isNaN(date.getTime())) return "Дата не вказана";

    return date.toLocaleDateString("uk-UA", {
        day: "2-digit",
        month: "long",
        year: "numeric",
    });
}

function normalizeList(response) {
    if (Array.isArray(response)) return response;
    if (Array.isArray(response?.jobs)) return response.jobs;
    if (Array.isArray(response?.items)) return response.items;
    if (Array.isArray(response?.results)) return response.results;
    if (Array.isArray(response?.data)) return response.data;
    return [];
}

function formatSalary(job) {
    if (job.salary_min && job.salary_max) {
        return `${job.salary_min}-${job.salary_max} ${job.salary_currency || "USD"}`;
    }

    if (job.salary_min) {
        return `від ${job.salary_min} ${job.salary_currency || "USD"}`;
    }

    if (job.salary_max) {
        return `до ${job.salary_max} ${job.salary_currency || "USD"}`;
    }

    return job.is_paid ? "Оплачувано" : "Не вказано";
}

function formatExperience(value) {
    if (value === "no-experience") return "Без досвіду";
    if (value === "junior") return "Junior";
    if (value === "middle") return "Middle";
    if (value === "senior") return "Senior";
    return "Рівень не вказано";
}
```

```
function formatEmployment(value) {
    if (value === "full-time") return "Повна зайнятість";
    if (value === "part-time") return "Часткова зайнятість";
    if (value === "contract") return "Проєктна робота";
    if (value === "temporary") return "Позмінна робота";
    if (value === "internship") return "Стажування";
    return "Тип не вказано";
}

function formatWorkFormat(value) {
    if (value === "remote") return "Віддалено";
    if (value === "office") return "В офісі";
    if (value === "hybrid") return "Гібридно";
    return "Формат не вказано";
}

function getPageCopy(fixedType) {
    if (fixedType === "internship") {
        return {
            label: "Стажування",
            title: "Стажування для старту кар'єри",
            text: "Обирайте актуальні стажування, trainee-програми та можливості для
студентів і початківців.",
            emptyTitle: "Стажувань не знайдено",
            emptyText: "Спробуйте змінити пошук, місто або формат стажування.",
        };
    }

    return {
        label: "Вакансії",
        title: "Вакансії для кандидатів",
        text: "Шукайте роботу, junior-позиції та можливості для професійного розвитку.",
        emptyTitle: "Вакансій не знайдено",
        emptyText: "Спробуйте змінити пошук, місто або фільтри.",
    };
}

export default function JobsPage({ fixedType = "job" }) {
    const navigate = useNavigate();
    const { user, role } = useAuth();

    const page = getPageCopy(fixedType);

    const [jobs, setJobs] = useState([]);
    const [savedIds, setSavedIds] = useState([]);

    const [search, setSearch] = useState("");
    const [city, setCity] = useState("");
    const [experienceLevel, setExperienceLevel] = useState("");
    const [employmentType, setEmploymentType] = useState("");
    const [workFormat, setWorkFormat] = useState("");
    const [sortBy, setSortBy] = useState("newest");

    const [loading, setLoading] = useState(true);
    const [savingId, setSavingId] = useState(null);
    const [error, setError] = useState("");

    useEffect(() => {
        if (user && role === "company") {
            navigate("/company/dashboard", { replace: true });
        }
    }, [user, role, navigate]);

    useEffect(() => {
        loadJobs();
        // eslint-disable-next-line react-hooks/exhaustive-deps
    }, [fixedType, search, city, experienceLevel, employmentType, workFormat]);

    useEffect(() => {
```

```
    if (!user || role !== "student") {
        setSavedIds([]);
        return;
    }

    loadSavedJobs();
    // eslint-disable-next-line react-hooks/exhaustive-deps
}, [user?.id, role]);

async function loadJobs() {
    try {
        setLoading(true);
        setError("");

        const params = new URLSearchParams();

        params.set("type", fixedType);

        if (search.trim()) params.set("q", search.trim());
        if (city.trim()) params.set("city", city.trim());
        if (experienceLevel) params.set("experience_level", experienceLevel);
        if (employmentType) params.set("employment_type", employmentType);
        if (workFormat) params.set("work_format", workFormat);

        const response = await apiFetch(`/jobs?${params.toString()}`, {
            token: null,
        });

        setJobs(normalizeList(response));
    } catch (err) {
        setError(err.message || "Не вдалося завантажити дані");
    } finally {
        setLoading(false);
    }
}

async function loadSavedJobs() {
    try {
        const response = await apiFetch("/saved");

        const ids = normalizeList(response)
            .map((item) => item.job_id || item.job?.id || item.id)
            .filter(Boolean)
            .map(String);

        setSavedIds(ids);
    } catch {
        setSavedIds([]);
    }
}

async function handleSave(jobId) {
    if (!user || role !== "student") {
        navigate("/login");
        return;
    }

    const id = String(jobId);

    try {
        setSavingId(id);

        if (savedIds.includes(id)) {
            await apiFetch(`/saved/${id}`, {
                method: "DELETE",
            });

            setSavedIds((current) =>
                current.filter((item) => String(item) !== id)
            );
```

```
            return;
        }

        await apiFetch("/saved", {
            method: "POST",
            body: {
                job_id: id,
            },
        });

        setSavedIds((current) =>
            current.includes(id) ? current : [...current, id]
        );
    } catch (err) {
        setError(err.message || "Не вдалося оновити збережені");
    } finally {
        setSavingId(null);
    }
}

function clearFilters() {
    setSearch("");
    setCity("");
    setExperienceLevel("");
    setEmploymentType("");
    setWorkFormat("");
    setSortBy("newest");
}

const preparedJobs = useMemo(() => {
    const list = jobs.map((job) => ({
        ...job,
        companyText: job.company_name || "Компанія",
        locationText: job.city || job.location || "Локацію не вказано",
        salaryText: formatSalary(job),
        experienceText: formatExperience(job.experience_level),
        employmentText: formatEmployment(job.employment_type),
        workFormatText: formatWorkFormat(job.work_format),
        detailsPath:
            fixedType === "internship"
                ? `/internships/${job.id}/${slugify(job.title)}`
                : `/jobs/${job.id}/${slugify(job.title)}`,
    }));

    if (sortBy === "oldest") {
        return [...list].sort(
            (a, b) =>
                new Date(a.created_at || 0).getTime() -
                new Date(b.created_at || 0).getTime()
        );
    }

    if (sortBy === "salary") {
        return [...list].sort(
            (a, b) =>
                Number(b.salary_max || b.salary_min || 0) -
                Number(a.salary_max || a.salary_min || 0)
        );
    }

    return [...list].sort(
        (a, b) =>
            new Date(b.created_at || 0).getTime() -
            new Date(a.created_at || 0).getTime()
    );
}, [jobs, sortBy, fixedType]);

if (user && role === "company") {
    return null;
```

```
}

return (
    <>
        <Header />

        <main className="jobs-page">
            <section className="jobs-hero">
                <div>
                    <p className="jobs-eyebrow">{page.label}</p>
                    <h1>{page.title}</h1>
                    <p>{page.text}</p>
                </div>

                <div className="jobs-hero-actions">
                    <Link
                        to="/jobs"
                        className={
                            fixedType === "job"
                                ? "jobs-switch is-active"
                                : "jobs-switch"
                        }
                    >
                        Вакансії
                    </Link>

                    <Link
                        to="/internships"
                        className={
                            fixedType === "internship"
                                ? "jobs-switch is-active"
                                : "jobs-switch"
                        }
                    >
                        Стажування
                    </Link>
                </div>
            </section>

            <section className="jobs-filters">
                <input
                    value={search}
                    onChange={(event) => setSearch(event.target.value)}
                    placeholder={
                        fixedType === "internship"
                            ? "Пошук стажування, компанії або напрямку..."
                            : "Пошук вакансії, компанії або напрямку..."
                    }
                />

                <input
                    value={city}
                    onChange={(event) => setCity(event.target.value)}
                    placeholder="Місто"
                />

                <select
                    value={experienceLevel}
                    onChange={(event) => setExperienceLevel(event.target.value)}
                >
                    <option value="">Усі рівні</option>
                    <option value="no-experience">Без досвіду</option>
                    <option value="junior">Junior</option>
                    <option value="middle">Middle</option>
                    <option value="senior">Senior</option>
                </select>

                <select
                    value={employmentType}
                    onChange={(event) => setEmploymentType(event.target.value)}
```

```
        >
            <option value="">Усі типи зайнятості</option>
            <option value="full-time">Повна зайнятість</option>
            <option value="part-time">Часткова зайнятість</option>
            <option value="contract">Проєктна робота</option>
            <option value="temporary">Позмінна робота</option>
            <option value="internship">Стажування</option>
        </select>

        <select
            value={workFormat}
            onChange={(event) => setWorkFormat(event.target.value)}
        >
            <option value="">Усі формати</option>
            <option value="remote">Віддалено</option>
            <option value="office">В офісі</option>
            <option value="hybrid">Гібридно</option>
        </select>

        <select
            value={sortBy}
            onChange={(event) => setSortBy(event.target.value)}
        >
            <option value="newest">Спочатку нові</option>
            <option value="oldest">Спочатку старі</option>
            <option value="salary">За зарплатою</option>
        </select>

        <button type="button" onClick={clearFilters}>
            Скинути
        </button>
    </section>

    {error ? <div className="jobs-alert">{error}</div> : null}

    <section className="jobs-results-head">
        <p>
            Знайдено: <strong>{preparedJobs.length}</strong>
        </p>
    </section>

    {loading ? (
        <section className="jobs-grid">
            {[1, 2, 3, 4].map((item) => (
                <article className="jobs-card jobs-card--loading" key={item}>
                    <div />
                    <div />
                    <div />
                </article>
            ))}
        </section>
    ) : null}

    {!loading && !preparedJobs.length ? (
        <section className="jobs-empty">
            <h2>{page.emptyTitle}</h2>
            <p>{page.emptyText}</p>
            <button type="button" onClick={clearFilters}>
                Очистити фільтри
            </button>
        </section>
    ) : null}

    {!loading && preparedJobs.length ? (
        <section className="jobs-grid">
            {preparedJobs.map((job) => {
                const isSaved = savedIds.includes(String(job.id));

                return (
                    <article className="jobs-card" key={job.id}>
```

```
                        <div className="jobs-card-top">
                            <div>
                                <span>
                                    {fixedType === "internship"
                                        ? "Стажування"
                                        : "Вакансія"}
                                </span>

                                <h2>{job.title}</h2>
                                <p>{job.companyText}</p>
                            </div>

                            {role === "student" ? (
                                <button
                                    type="button"
                                    className={
                                        isSaved
                                            ? "jobs-save is-active"
                                            : "jobs-save"
                                    }
                                    onClick={() => handleSave(job.id)}
                                    disabled={savingId === String(job.id)}
                                    title={
                                        isSaved
                                            ? "Прибрати зі збережених"
                                            : "Зберегти"
                                    }
                                >
                                    {savingId === String(job.id)
                                        ? "..."
                                        : isSaved
                                            ? "♥"
                                            : "♡"}
                                </button>
                            ) : null}
                        </div>

                        <div className="jobs-card-tags">
                            <span>{job.locationText}</span>
                            <span>{job.workFormatText}</span>
                            <span>{job.employmentText}</span>
                            <span>{job.experienceText}</span>
                            <span>{job.salaryText}</span>
                        </div>

                        <p className="jobs-card-description">
                            {job.description
                                ? `${job.description.slice(0, 190)}${
                                    job.description.length > 190
                                        ? "..."
                                        : ""
                                }`
                                : "Опис відсутній"}
                        </p>

                        <div className="jobs-card-footer">
                            <small>{formatDate(job.created_at)}</small>

                            <Link to={job.detailsPath}>
                                Детальніше →
                            </Link>
                        </div>
                    </article>
                );
            })}
        </section>
    ) : null}
</main>

<Footer />
```

```
        </>
    );
}
import { useEffect, useMemo, useState } from "react";
import { Link, Navigate, useParams } from "react-router-dom";
import Header from "../../../components/layout/Header";
import ApplyModal from "../../jobseeker/components/ApplyModal.jsx";
import { apiFetch } from "../../../services/api/api.js";
import { useAuth } from "../../../context/AuthContext";
import "./JobDetailsPage.css";

function getJobDirection(job) {
    return job?.specialty || job?.category_name || job?.category || "";
}

function formatWorkFormat(value) {
    switch (value) {
        case "remote":
            return "Віддалено";
        case "office":
            return "В офісі";
        case "hybrid":
            return "Гібридно";
        default:
            return value || "Не вказано";
    }
}

function formatEmployment(value) {
    switch (value) {
        case "full-time":
            return "Повна зайнятість";
        case "part-time":
            return "Часткова зайнятість";
        case "contract":
            return "Проєктна робота";
        case "temporary":
            return "Позмінна робота";
        case "internship":
            return "Стажування";
        default:
            return value || "Не вказано";
    }
}

function formatExperience(value) {
    switch (value) {
        case "no-experience":
            return "Без досвіду";
        case "junior":
            return "Junior";
        case "middle":
            return "Middle";
        case "senior":
            return "Senior";
        default:
            return value || "Не вказано";
    }
}

function formatSalary(job) {
    if (!job) return "Не вказано";

    if (job.salary_min && job.salary_max) {
        return `${job.salary_min}-${job.salary_max} ${job.salary_currency || "UAH"}`;
    }

    if (job.salary_min) {
        return `від ${job.salary_min} ${job.salary_currency || "UAH"}`;
    }
```

```
    if (job.salary_max) {
        return `до ${job.salary_max} ${job.salary_currency || "UAH"}`;
    }

    return job.is_paid ? "Оплачувано" : "Не вказано";
}

function normalizeSavedIds(data) {
    const list = Array.isArray(data)
        ? data
        : Array.isArray(data?.saved)
            ? data.saved
            : Array.isArray(data?.items)
                ? data.items
                : [];

    return list
        .map((item) => item.job_id || item.job?.id || item.id || item)
        .filter(Boolean)
        .map(String);
}

function parseBenefits(value) {
    if (!value) return [];

    if (Array.isArray(value)) {
        return value.map((item) => String(item).trim()).filter(Boolean);
    }

    if (typeof value === "string") {
        try {
            const parsed = JSON.parse(value);

            if (Array.isArray(parsed)) {
                return parsed.map((item) => String(item).trim()).filter(Boolean);
            }
        } catch {
            return value
                .split(",")
                .map((item) => item.trim())
                .filter(Boolean);
        }
    }

    return [];
}

export default function JobDetailsPage() {
    const { id } = useParams();
    const { user, role, loading: authLoading } = useAuth();

    const [job, setJob] = useState(null);
    const [similarJobs, setSimilarJobs] = useState([]);

    const [loading, setLoading] = useState(true);
    const [similarLoading, setSimilarLoading] = useState(false);

    const [error, setError] = useState("");
    const [similarError, setSimilarError] = useState("");

    const [savedIds, setSavedIds] = useState([]);
    const [saving, setSaving] = useState(false);

    const [applyOpen, setApplyOpen] = useState(false);
    const [applied, setApplied] = useState(false);
    const [message, setMessage] = useState("");

    useEffect(() => {
        let active = true;
```

```
    async function loadJob() {
        try {
            setLoading(true);
            setError("");
            setJob(null);

            const data = await apiFetch(`/jobs/${id}`, {
                token: null,
            });

            if (!active) return;

            setJob(data);
        } catch (err) {
            if (!active) return;

            setError(err.message || "Вакансію не знайдено");
        } finally {
            if (active) {
                setLoading(false);
            }
        }
    }

    if (id) {
        loadJob();
    }

    return () => {
        active = false;
    };
}, [id]);

useEffect(() => {
    if (!job?.id) return;

    let active = true;

    async function loadSimilarJobs() {
        try {
            setSimilarLoading(true);
            setSimilarError("");

            const direction = getJobDirection(job);
            let result = [];

            if (direction) {
                const params = new URLSearchParams();
                params.set("specialty", direction);

                const data = await apiFetch(`/jobs?${params.toString()}`, {
                    token: null,
                });

                const list = Array.isArray(data) ? data : [];

                result = list.filter((item) => String(item.id) !== String(job.id));
            }

            if (result.length === 0) {
                const data = await apiFetch("/jobs", {
                    token: null,
                });

                const list = Array.isArray(data) ? data : [];

                result = list.filter((item) => String(item.id) !== String(job.id));
            }
```

```
            if (!active) return;

            setSimilarJobs(result.slice(0, 3));
        } catch (err) {
            if (!active) return;

            console.error("Не вдалося завантажити схожі вакансії:", err);
            setSimilarError("Не вдалося завантажити схожі вакансії");
            setSimilarJobs([]);
        } finally {
            if (active) {
                setSimilarLoading(false);
            }
        }
    }

    loadSimilarJobs();

    return () => {
        active = false;
    };
}, [job?.id]);

useEffect(() => {
    if (authLoading) return;

    if (!user || role !== "student") {
        setSavedIds([]);
        return;
    }

    let active = true;

    async function loadSaved() {
        try {
            const data = await apiFetch("/saved");

            if (!active) return;

            setSavedIds(normalizeSavedIds(data));
        } catch {
            if (!active) return;
            setSavedIds([]);
        }
    }

    loadSaved();

    return () => {
        active = false;
    };
}, [authLoading, user?.id, role]);

const direction = useMemo(() => getJobDirection(job), [job]);
const benefits = useMemo(() => parseBenefits(job?.benefits), [job]);
const isSaved = job?.id ? savedIds.includes(String(job.id)) : false;

async function saveJob() {
    if (!job?.id) return;

    if (!user) {
        setMessage("Спочатку увійдіть, щоб зберегти вакансію.");
        return;
    }

    if (role !== "student") {
        setMessage("Зберігати вакансії може тільки шукач роботи.");
        return;
    }
```

```
    const jobId = String(job.id);

    try {
        setSaving(true);
        setMessage("");

        if (isSaved) {
            await apiFetch(`/saved/${jobId}`, {
                method: "DELETE",
            });

            setSavedIds((prev) => prev.filter((item) => String(item) !== jobId));
            return;
        }

        await apiFetch("/saved", {
            method: "POST",
            body: {
                job_id: jobId,
            },
        });

        setSavedIds((prev) => (prev.includes(jobId) ? prev : [...prev, jobId]));
    } catch (err) {
        setMessage(err.message || "Не вдалося зберегти вакансію");
    } finally {
        setSaving(false);
    }
}

function openApply() {
    setMessage("");

    if (!user) {
        setMessage("Щоб відгукнутися на вакансію, увійдіть в акаунт.");
        return;
    }

    if (role !== "student") {
        setMessage("Відгукуватися на вакансії може тільки шукач роботи.");
        return;
    }

    setApplyOpen(true);
}

if (loading || authLoading) {
    return (
        <div className="job-details-page">
            <Header />

            <main className="job-details-main">
                <div className="job-details-state">Завантаження...</div>
            </main>
        </div>
    );
}

if (error) {
    return (
        <div className="job-details-page">
            <Header />

            <main className="job-details-main">
                <div className="job-details-state">{error}</div>
            </main>
        </div>
    );
}
```

```
if (!job) {
    return <Navigate to="/jobs" replace />;
}

return (
    <div className="job-details-page">
        <Header />

        <main className="job-details-main">
            <section className="job-details-card">
                <div className="job-details-breadcrumbs">
                    <Link to="/">Головна</Link>
                    <span>›</span>
                    <Link to="/jobs">Вакансії</Link>
                    <span>›</span>
                    <span>{job.title}</span>
                </div>

                <div className="job-details-top">
                    <div>
                        <span className="job-details-type">
                            {job.type === "internship" ? "Стажування" : "Робота"}
                        </span>

                        <h1>{job.title}</h1>

                        <p className="job-details-meta">
                            {job.company_name || "Компанія"} •{" "}
                            {job.city || job.location || "Україна"}
                        </p>

                        {direction ? (
                            <div className="job-details-direction">
                                {direction}
                            </div>
                        ) : null}
                    </div>

                    <div className="job-details-side">
                        <strong>{formatSalary(job)}</strong>

                        <button
                            type="button"
                            onClick={openApply}
                            disabled={applied}
                            className="job-details-apply-btn"
                        >
                            {applied ? "Відгук надіслано" : "Відгукнутись"}
                        </button>

                        <button
                            type="button"
                            onClick={saveJob}
                            disabled={saving}
                            className={`job-details-save-btn ${
                                isSaved ? "job-details-save-btn--active" : ""
                            }`}
                        >
                            {saving
                                ? "Збереження..."
                                : isSaved
                                    ? "♥ Збережено"
                                    : "♡ Зберегти"}
                        </button>

                        {message ? (
                            <div className="job-details-message">
                                {message}

                                {!user ? (
```

```
                                    <div>
                                        <Link to="/login">Увійти</Link>
                                        <span> або </span>
                                        <Link to="/register/candidate">
                                            зареєструватися
                                        </Link>
                                    </div>
                                ) : null}
                            </div>
                        ) : null}
                    </div>
                </div>

                <div className="job-details-info-grid">
                    <div>
                        <span>Формат</span>
                        <strong>{formatWorkFormat(job.work_format ||
job.format)}</strong>
                    </div>

                    <div>
                        <span>Зайнятість</span>
                        <strong>{formatEmployment(job.employment_type)}</strong>
                    </div>

                    <div>
                        <span>Досвід</span>
                        <strong>{formatExperience(job.experience_level)}</strong>
                    </div>
                </div>

                <section className="job-details-section">
                    <h2>Опис вакансії</h2>
                    <p className="job-details-description">
                        {job.description || "Опис вакансії відсутній"}
                    </p>
                </section>

                {job.requirements ? (
                    <section className="job-details-section">
                        <h2>Вимоги</h2>
                        <p className="job-details-description">
                            {job.requirements}
                        </p>
                    </section>
                ) : null}

                {job.responsibilities ? (
                    <section className="job-details-section">
                        <h2>Обов'язки</h2>
                        <p className="job-details-description">
                            {job.responsibilities}
                        </p>
                    </section>
                ) : null}

                {benefits.length > 0 ? (
                    <section className="job-details-section">
                        <h2>Переваги</h2>

                        <ul className="job-details-benefits">
                            {benefits.map((item) => (
                                <li key={item}>{item}</li>
                            ))}
                        </ul>
                    </section>
                ) : null}
            </section>

            <section className="similar-jobs-section">
```

```
                    <div className="similar-jobs-header">
                        <div>
                            <h2>Схожі вакансії</h2>

                            <p>
                                {direction
                                    ? `За напрямом: ${direction}`
                                    : "Інші актуальні пропозиції"}
                            </p>
                        </div>

                        <Link to="/jobs">Всі вакансії →</Link>
                    </div>

                    {similarLoading ? (
                        <div className="similar-jobs-state">
                            Завантаження схожих вакансій...
                        </div>
                    ) : null}

                    {!similarLoading && similarError ? (
                        <div className="similar-jobs-state">{similarError}</div>
                    ) : null}

                    {!similarLoading && !similarError && similarJobs.length === 0 ? (
                        <div className="similar-jobs-state">
                            Схожих вакансій поки немає.
                        </div>
                    ) : null}

                    {!similarLoading && !similarError && similarJobs.length > 0 ? (
                        <div className="similar-jobs-grid">
                            {similarJobs.map((item) => {
                                const itemDirection = getJobDirection(item);

                                return (
                                    <Link
                                        key={item.id}
                                        to={`/jobs/${item.id}`}
                                        className="similar-job-card"
                                    >
                                        <h3 className="line-clamp-2">
                                            {item.title}
                                        </h3>

                                        <p className="similar-job-meta">
                                            {item.company_name || "Компанія"} •{" "}
                                            {item.city || item.location || "Україна"}
                                        </p>

                                        <div className="similar-job-tags">
                                            {itemDirection ? (
                                                <span className="similar-job-tag similar-
job-tag--blue">
                                                    {itemDirection}
                                                </span>
                                            ) : null}

                                            {item.work_format || item.format ? (
                                                <span className="similar-job-tag">
                                                    {formatWorkFormat(
                                                        item.work_format || item.format
                                                    )}
                                                </span>
                                            ) : null}
                                        </div>

                                        <p className="similar-job-description line-clamp-
3">
                                            {item.description || "Опис вакансії відсутній"}
```

```
                                        </p>
                                    </Link>
                                );
                            })}
                        </div>
                    ) : null}
                </section>
            </main>

            <ApplyModal
                job={job}
                open={applyOpen}
                onClose={() => setApplyOpen(false)}
                onSuccess={() => {
                    setApplied(true);
                    setMessage("Відгук успішно надіслано.");
                }}
            />
        </div>
    );
}
```