**Міністерство освіти і науки України**

**Чернівецький національний університет**

**імені Юрія Федьковича**

Факультет математики та інформатики

(повна назва інституту/факультету)

Кафедра прикладної математики та інформаційних технологій

(повна назва кафедри)

**Інформаційна система для пошуку роботи і вакансій**

**Кваліфікаційна робота**

**Рівень вищої освіти - перший (бакалаврський)**

Виконав:

студент 4 курсу, групи . 422 .

спеціальності .113 – Прикладна математика .

(назва спеціальності)

**Кацап Владислав Геннадійович**  .

(прізвище, ім’я та по-батькові)

Керівник: **к.т.н., доцент Шепетюк Б.Д**.

(науковий ступінь, вчене звання, прізвище та ініціали)

До захисту допущено:

Протокол засідання кафедри №

від „ ” грудня 2025 р.

Завідувач. кафедри \_\_\_\_\_\_\_\_\_ проф. Бігун Я.Й.

Чернівці – 2025

**АНОТАЦІЯ**

Дипломна робота присвячена розробці інформаційної системи для автоматизованого пошуку роботи. Система орієнтована на взаємодію трьох основних категорій користувачів — здобувачів роботи, роботодавців та адміністраторів. Текстова документація містить опис основних функціональних можливостей системи, включаючи реєстрацію, створення профілів, подачу заявок, керування вакансіями та адміністрування платформи. У роботі також детально описано алгоритми взаємодії між компонентами системи, архітектуру проєкту та реалізований функціонал. Окрему увагу приділено вибору та обґрунтуванню використаного стеку технологій: React — для розробки інтерфейсу користувача, Node.js — для реалізації серверної частини та MySQL — для організації зберігання даних.

Робота містить результати власних досліджень. Використання чужих ідей, результатів і текстів мають посилання на відповідне джерело

**ABSTRACT**

This work is devoted to the development of an information system for automated job search. The system is focused on the interaction of three main categories of users - job seekers, employers and administrators. The textual documentation describes the main functionalities of the system, including registration, profile creation, application submission, job management, and platform administration. It also describes in detail the algorithms of interaction between the system components, the project architecture, and the implemented functionality. Special attention is paid to the choice and justification of the technology stack used: React for user interface development, Node.js for server-side implementation, and MySQL for data storage.

The paper represents the results of original research. Others' ideas, results and books are appropriately cited

**ЗМІСТ**

[ВСТУП 4](#_Toc199692615)

[Розділ 1. Дослідження предметної області та засоби розробки 5](#_Toc199692616)

[1.1 Загальні відомості до предметної області 5](#_Toc199692617)

[1.2 Призначення та застосування Node.js та Express 7](#_Toc199692618)

[1.3 Призначення та застосування React 9](#_Toc199692619)

[1.4 Призначення та застосування MySQL 11](#_Toc199692620)

[Розділ 2. Проектування програмного застосунку 14](#_Toc199692621)

[2.1 Специфікація вимог для системи пошуку роботи і вакансій 14](#_Toc199692622)

[2.2. Функціонал програмного продукту 17](#_Toc199692623)

[2.3. Проєктування та розробка бази даних 21](#_Toc199692624)

[2.4. Компоненти програмного продукту 28](#_Toc199692625)

[Розділ 3. Робочий проєкт 30](#_Toc199692626)

[3.1 Опис розробки серверної частини 30](#_Toc199692627)

[3.2 Опис розробки клієнтської частини 33](#_Toc199692628)

[3.3 Інструкція користувача 33](#_Toc199692629)

[Висновки 34](#_Toc199692630)

[Список використаних джерел 35](#_Toc199692631)

[Додаток 36](#_Toc199692632)

# ВСТУП

У сучасному світі, де цифрові технології швидко розвиваються, стає важливо автоматизувати процеси пошуку роботи. Старі способи — через знайомих, оголошення або біржі праці — вже не такі ефективні. Тому виникає потреба у сучасних онлайн-платформах, які об'єднують здобувачів роботи, роботодавців і адміністраторів, щоб зробити взаємодію швидшою, зручнішою та безпечнішою.

Інформаційна система пошуку роботи автоматизує взаємодію між кандидатами і компаніями — від реєстрації та створення профілів до подачі заявок, співбесід і управління вакансіями. Її мета — зробити пошук роботи та підбір персоналу швидким, зручним і ефективним, а також забезпечити зручне адміністрування платформи.

Для досягнення цієї мети система передбачає наявність трьох основних типів користувачів: здобувачів роботи, роботодавців та адміністраторів. Кожна категорія має свій набір функціональних можливостей, які відповідають їхнім потребам у межах системи. Серед ключових функцій — розширений пошук, подача заявок, керування вакансіями, комунікація між сторонами, а також моніторинг і адміністрування платформи.

Розробка такої системи вимагає визначенню вимог, даних з якими програма працює. Це дозволить зробити платформу продуктивною, захищеною, зручною у користуванні та здатною працювати з великою кількістю користувачів. У підсумку система допоможе поєднувати роботодавців і кандидатів, полегшуючи пошук роботи й організовуючи процес працевлаштування..

Для розробки інформаційної системи було використано сучасний стек технологій: React — для створення інтуїтивного та динамічного інтерфейсу користувача, Node.js — для реалізації логіки бекенду та обробки запитів, а також MySQL — як реляційну базу даних для зберігання інформації про користувачів, вакансії та заявки.

# Розділ 1. Дослідження предметної області та засоби розробки

## 1.1 Загальні відомості до предметної області

У сучасному світі пошук роботи є важливою складовою життя кожної працездатної людини. З розвитком цифрових технологій традиційні методи пошуку роботи, такі як оголошення в газетах, особисті контакти чи звернення до центрів зайнятості, поступово втрачають свою ефективність. Натомість дедалі більшої популярності набувають онлайн-платформи, які надають широкі можливості для ефективного пошуку роботи та підбору персоналу.

Розробка вебсайтів у сфері працевлаштування дозволяє автоматизувати численні процеси, пов’язані з пошуком роботи. Це стосується як здобувачів, так і роботодавців: подача резюме, створення вакансій, сортування та фільтрація пропозицій, надсилання повідомлень, організація співбесід — усе це може відбуватися в межах єдиної інтегрованої системи. Завдяки цьому зменшується потреба у посередниках, прискорюється процес прийняття рішень і підвищується ефективність комунікації.

Сучасні сервіси працевлаштування дозволяють здобувачам роботи переглядати доступні вакансії, фільтрувати їх за різними критеріями, створювати резюме та швидко подавати заявки на цікаві пропозиції. З іншого боку, роботодавці мають змогу створювати вакансії, переглядати резюме кандидатів, організовувати співбесіди та приймати рішення щодо найму. Такі системи значно скорочують час пошуку та забезпечують зручну комунікацію між обома сторонами.

Онлайн-платформи для пошуку роботи також мають важливе значення для адміністраторів — вони здійснюють контроль за контентом, користувачами та загальною роботою системи, забезпечуючи безпечне та якісне функціонування сервісу.

Для реалізації сучасної інформаційної системи пошуку роботи необхідним є створення повноцінного вебсайту, який забезпечуватиме зручний та інтуїтивно зрозумілий інтерфейс, стабільну роботу серверної частини та ефективне зберігання й обробку даних. Розробка таких систем зазвичай здійснюється із використанням сучасних вебтехнологій та фреймворків, що прискорюють процес розробки, полегшують підтримку коду і забезпечують масштабованість проєкту.

Вебсайт, як правило, складається з трьох основних компонентів: фронтенду, бекенду та бази даних.

* Фронтенд — це клієнтська частина системи, з якою безпосередньо взаємодіє користувач. Для його створення широко застосовуються JavaScript-фреймворки, зокрема React, який дозволяє будувати динамічні, компонентно-орієнтовані інтерфейси з високим рівнем реактивності та зручності у використанні. Завдяки фронтенду користувачі можуть реєструватися, переглядати вакансії, створювати резюме, подавати заявки тощо.
* Бекенд — це серверна частина, що відповідає за обробку логіки програми, маршрутизацію запитів, взаємодію з базою даних, а також реалізацію авторизації, аутентифікації та контролю доступу. Часто для розробки бекенду використовуються середовища такі як Node.js, які забезпечують високу продуктивність і добре поєднуються з фронтенд-фреймворками.
* База даних — невід’ємна складова будь-якої інформаційної системи, що забезпечує зберігання інформації про користувачів, вакансії, заявки, повідомлення, права доступу та інші важливі об’єкти. У межах такого проєкту доцільно використовувати реляційні бази даних, зокрема MySQL, що забезпечують надійність, цілісність даних і зручність у використанні для складних запитів.

Таким чином, створення інформаційної системи пошуку роботи — це комплексний процес, що вимагає поєднання знань у галузі фронтенд- і бекенд-розробки, проєктування баз даних, а також розуміння логіки предметної області. Використання сучасних фреймворків та технологій дає змогу створити ефективну, масштабовану та зручну систему, яка автоматизує ключові процеси у сфері працевлаштування.

Досліджувана предметна область охоплює процеси створення, публікації та пошуку вакансій, управління профілями користувачів, взаємодію між роботодавцями та кандидатами, а також організацію та підтримку ефективного функціонування платформи. Саме автоматизація цих процесів є основною метою створення сучасної інформаційної системи пошуку роботи.

## 1.2 Призначення та застосування Node.js та Express

Node.js — це середовище виконання JavaScript на стороні сервера, яке дозволяє створювати масштабовані, високопродуктивні вебдодатки з використанням одного з найпоширеніших мов програмування — JavaScript. Завдяки своїй асинхронній та неблокуючій архітектурі, Node.js є ідеальним вибором для створення вебсервісів, які повинні ефективно обробляти велику кількість одночасних підключень. Це особливо важливо у випадках, коли система повинна обслуговувати як здобувачів, так і роботодавців у режимі реального часу.

У рамках інформаційної системи пошуку роботи Node.js доцільно використовувати для реалізації серверної логіки — обробки запитів, автентифікації, авторизації, валідації даних, а також взаємодії з базою даних.

Переваги використання Node.js:

* Єдина мова програмування (JavaScript) як на фронтенді, так і на бекенді, що спрощує розробку та зменшує поріг входу для команди розробників;
* Висока продуктивність завдяки рушію V8;
* Активна екосистема npm, яка надає тисячі готових бібліотек для прискорення розробки;
* Асинхронна обробка подій, що дозволяє будувати швидкі та чуйні API;
* Гнучкість у налаштуванні сервера: розробник має повний контроль над обробкою запитів, middleware, безпекою тощо.

Express.js як фреймворк

Express — це мінімалістичний вебфреймворк для Node.js, який забезпечує гнучку архітектуру для створення RESTful API. Його вибір зумовлений простотою у використанні, масштабованістю та гнучкістю в налаштуванні маршрутизації, обробки помилок та middleware.

Під час реалізації інформаційної системи Express дозволить:

* Створити набір ендпоінтів (HTTP-маршрутів), через які фронтенд буде взаємодіяти з сервером (наприклад, створення, редагування та перегляд вакансій, реєстрація користувачів, обробка заявок);
* Реалізувати механізми авторизації та автентифікації, включаючи використання JWT (JSON Web Tokens) для захисту маршрутизації та передачі ідентифікаційної інформації;
* Організувати структуровану логіку обробки запитів, включаючи валідацію вхідних даних, обробку помилок та логування;
* Додати middleware-функції, наприклад для контролю доступу до ресурсів залежно від ролі користувача (здобувач, роботодавець, адміністратор);
* Підключити базу даних MySQL за допомогою драйверів або ORM, забезпечивши збереження та обробку даних про користувачів, вакансії та заявки.

Процес створення API на Node.js та Express включатиме:

1. Ініціалізацію проєкту (npm init, встановлення Express, dotenv, cors тощо);
2. Налаштування маршрутизації — створення окремих файлів для маршрутів (routes/jobs.js, routes/users.js);
3. Впровадження контролерів — логіка обробки даних, перевірка прав, відповіді клієнту;
4. Підключення до бази даних — через mysql2;
5. Реалізація авторизації — за допомогою JWT токенів, middleware для перевірки прав доступу;
6. Створення системи ролей — для обмеження функціоналу залежно від типу користувача;
7. Забезпечення безпеки — обмеження CORS, обробку помилок, валідацію даних;
8. Тестування ендпоінтів — з використанням інструментів для тестування до прикладу Postman.

Node.js у зв’язці з Express дозволяє створити ефективний, гнучкий і масштабований сервер, який може бути інтегрований з клієнтською частиною на React та базою даних MySQL. Це робить його оптимальним вибором для сучасної вебплатформи, орієнтованої на велику кількість користувачів та взаємодію в реальному часі.

## 1.3 Призначення та застосування React

React — це популярна JavaScript-бібліотека з відкритим вихідним кодом, створена компанією Meta (раніше Facebook) для побудови користувацьких інтерфейсів. Вона дає змогу створювати багаторазові, динамічні та продуктивні інтерфейси вебзастосунків із використанням компонентоорієнтованої архітектури. React не є повноцінним фреймворком, як наприклад Angular, але завдяки своїй простоті, модульності й широкому екосистемному оточенню він став стандартом де-факто для фронтенд-розробки.

Обґрунтування вибору React для інформаційної системи

У контексті розробки вебплатформи для пошуку роботи React дозволяє:

* Швидко створювати інтерактивні інтерфейси для користувачів з різними ролями (здобувачі, роботодавці, адміністратори);
* Забезпечити динамічне оновлення контенту без перезавантаження сторінки завдяки віртуальному DOM;
* Повторно використовувати компоненти, наприклад, для відображення вакансій, форм подачі заявок, фільтрів пошуку;
* Забезпечити швидке оновлення UI у відповідь на зміну стану програми (наприклад, після авторизації або відправки заявки).

Компонентна структура

Однією з основних переваг React є його компонентний підхід. Кожна частина інтерфейсу — це окремий компонент, який може мати свій стан і властивості. Це дозволяє масштабувати інтерфейс, розділяючи складні екрани на логічні частини, до прикладу:

* Header, Navbar, Footer — загальні компоненти для всієї системи;
* JobList, JobCard, JobDetails — модулі для перегляду вакансій;
* UserProfile, ApplicationForm— компоненти для конкретних ролей.

Під час створення сучасного фронтенду на React доцільно використовувати й допоміжні бібліотеки та підходи:

* React Router — для реалізації клієнтської маршрутизації, що дозволяє змінювати "сторінки" без перезавантаження;
* Axios — для надсилання HTTP-запитів до бекенду на Node.js/Express;
* Tailwind CSS — для швидкої та сучасної стилізації інтерфейсу користувача без написання власних стилів;
* JWT-захист — збереження та передача токенів при здійсненні запитів;

Типовий процес створення React-застосунку

1. Ініціалізація проєкту через create-react-app;
2. Налаштування структури папок: components, pages, services, hooks, contexts;
3. Реалізація системи маршрутизації (react-router-dom): логін, реєстрація, список вакансій, профіль, адміністративна панель;
4. Підключення до API: написання сервісів для обробки запитів;
5. Створення глобального сховища стану для збереження даних про користувача, токен, список заявок;
6. Забезпечення рендерингу взалежності від авторизації, ролі користувача;
7. Обробка помилок і завантаження — спінери, повідомлення, модальні вікна;

Переваги використання React:

* Висока швидкість роботи й оновлення UI;
* Можливість масштабування проєкту без втрати структури;
* Широка спільнота, постійна підтримка та велика кількість готових рішень;
* Добре підходить для інтеграції з REST API;
* Гнучкість — можливість будувати саме те, що потрібно без зайвих обмежень.

React у поєднанні з бекендом на Express та базою даних MySQL дозволяє створити надійний, масштабований та інтерактивний вебзастосунок для пошуку роботи з розподілом ролей і гнучкою логікою. У подальших етапах розробки ця технологія забезпечить швидкість, адаптивність і зручність користування, що є ключовими вимогами до сучасної онлайн-платформи.

## 1.4 Призначення та застосування MySQL

MySQL — це система керування реляційними базами даних з відкритим кодом, яка широко використовується у веброзробці завдяки своїй надійності, високій продуктивності та сумісності з більшістю мов програмування та фреймворків. MySQL базується на структурованих таблицях, де дані організовані у вигляді рядків і стовпців, і дозволяє виконувати запити за допомогою мови SQL.

Обґрунтування вибору MySQL

В контексті створення інформаційної системи для пошуку роботи MySQL обрано як основну систему зберігання даних завдяки таким характеристикам:

* Реляційна структура даних ідеально підходить для опису об’єктів з чіткими зв’язками: користувачі, вакансії, заявки, повідомлення тощо;
* Підтримка транзакцій — критично важливо для коректної обробки операцій, наприклад, подачі заявок;
* Масштабованість — дозволяє обслуговувати зростаючу кількість користувачів без втрати продуктивності;
* Поширеність і підтримка — велика база документації, спільнота та зручна інтеграція з Node.js через драйвери mysql2;

Застосування SQL

Для взаємодії з базою даних використовуватимуться SQL-запити:

* SELECT — вибірка даних (пошук вакансій, перегляд профілю, історії заявок);
* INSERT — додавання нових записів (реєстрація користувача, створення вакансії);
* UPDATE — оновлення існуючих даних (редагування профілю, зміна статусу заявки);
* DELETE — видалення (деактивація вакансій, видалення повідомлень).

Інтеграція з бекендом (Node.js/Express) база MySQL з’єднується через mysql2 — офіційний драйвер для взаємодії з MySQL.

Приклади використання в бекенді:

* Авторизація — перевірка наявності користувача та хешування паролю;
* Ролі — обмеження доступу до певних ендпоінтів;
* Пошук — побудова фільтрів вакансій по категорії, зарплаті, локації;
* Аналітика — підрахунок кількості заявок, популярних професій тощо.

Захист та оптимізація

Планується впровадження:

* Хешування паролів (через bcrypt);
* Індексація колонок — для прискорення пошуку по email, категоріях тощо;
* Рольовий доступ — на рівні бекенду через middleware, але з урахуванням ролей у базі.

Таким чином, MySQL є логічним вибором для реалізації централізованого зберігання даних у системі пошуку роботи, з чіткою структурою, високою надійністю та широкими можливостями інтеграції. У поєднанні з Node.js/Express та React ця технологія забезпечить повноцінне функціонування системи з високим рівнем продуктивності та безпеки.

# Розділ 2. Проектування програмного застосунку

## 2.1 Специфікація вимог для системи пошуку роботи і вакансій

Система пошуку роботи і вакансій призначена для автоматизації процесу взаємодії між здобувачами роботи та роботодавцями, а також для зручного адміністрування платформи. Основна мета системи полягає у забезпеченні ефективного пошуку вакансій та кандидатів, спрощенні комунікації між користувачами та забезпеченні зручним керуванням заявками. Для реалізації поставленої мети необхідно визначити користувачів системи, функціональні вимоги для кожного користувача, нефункціональні вимоги а також вхідні та вихідні дані.

Отже, було визначено, що система матиме три основні категорії користувачів:

* Здобувачі роботи — користувачі, які шукають вакансії, подають резюме та взаємодіють із роботодавцями.
* Роботодавці — компанії та організації, які публікують вакансії, переглядають резюме кандидатів та наймають працівників.
* Адміністратори — користувачі, які керують системою, забезпечують її безперебійну роботу та слідкують за дотриманням правил платформи.

Визначено функціональні вимоги до кожного виду користувачів.

Функціонал для здобувачів роботи:

Користувачі повинні мати можливість створювати обліковий запис через email, підтверджуючи його через верифікаційне посилання. Після реєстрації вони можуть заповнити свій профіль, додаючи такі дані:

* Контактну інформацію (телефон, email, місто проживання).
* Освіту (університет, спеціальність, рік закінчення).
* Досвід роботи (посада, компанія, період роботи, обов’язки).
* Навички (технічні, софт-скіли, мови програмування тощо).
* Портфоліо (завантаження файлів, посилання на роботи).
* Можливість редагування та оновлення профілю у будь-який момент.

Кандидати повинні мати зручний механізм пошуку вакансій, який включає:

* Фільтрацію вакансій за різними критеріями: локація, рівень заробітної плати, тип зайнятості (повна, часткова, віддалена робота), галузь роботи.
* Пошук вакансій за ключовими словами для швидкого знаходження відповідних пропозицій.
* Можливість збереження обраних вакансій у "Вибране" для подальшого перегляду.

Також здобувачі роботи можуть:

* Надсилати резюме на вакансії безпосередньо через платформу.
* Переглядати статус поданих заявок (розглядається, запрошення на співбесіду, відхилено тощо).
* Оцінювати процес співбесіди, залишаючи відгуки про компанію.
* Функціональні вимоги для роботодавців:
* Реєстрація і профіль компанії
* Компанії повинні пройти процес реєстрації, який включає верифікацію для запобігання шахрайству.
* Створення профілю компанії з додаванням логотипу, опису діяльності, контактної інформації.
* Публікація вакансій
* Можливість створення та редагування вакансій, які містять опис посади, вимоги, умови праці, рівень заробітної плати.
* Управління активними вакансіями (редагування деталей, видалення, архівування).

Управління заявками

* Перегляд отриманих резюме від кандидатів.
* Надсилання відповідей (запрошення на співбесіду, відмова, уточнення деталей).
* Комунікація
* Вбудована система обміну повідомленнями для зв’язку з кандидатами.
* Планування співбесід із можливістю синхронізації з календарем.

Функціональні вимоги для адміністратора системи:

Управління користувачами

* Адміністратори мають можливість блокувати та активувати облікові записи користувачів у разі порушень правил платформи.
* Моніторинг активності
* Генерація звітів щодо кількості публікацій вакансій, реєстрації нових користувачів, активності на платформі.
* Налаштування системи
* Управління фільтрами пошуку вакансій та кандидатів.
* Створення та редагування шаблонів повідомлень для автоматичних відповідей.
* Налаштування параметрів сповіщень (частота, канали комунікації).

Також було визначено та описано вхідні та вихідні дані

Вхідні дані:

* Реєстраційні дані користувачів (ім’я, email, пароль, контакти).
* Інформація про здобувачів роботи (освіта, досвід, навички, резюме).
* Вакансії, що публікуються роботодавцями (опис, вимоги, умови праці).
* Заявки на вакансії та супровідні повідомлення.

Вихідні дані:

* Список знайдених вакансій за критеріями пошуку.
* Сповіщення про нові вакансії та зміни статусу заявок.
* Відгуки кандидатів про процес співбесіди.
* Аналітичні звіти для адміністратора.

Нефункціональні вимоги:

* Безпека: зберігання паролів у хешованому вигляді, верифікація email, захист від SQL-ін’єкцій.
* Доступність: система має бути доступною 24/7 з мінімальним простоєм.
* Масштабованість: підтримка збільшення кількості користувачів без втрати ефективності.
* Зручність використання: інтуїтивно зрозумілий інтерфейс для всіх категорій користувачів.

Система пошуку роботи і вакансій забезпечує користувачам зручний механізм взаємодії між кандидатами та роботодавцями. Завдяки широкому набору функціональних можливостей вона сприяє ефективному працевлаштуванню, полегшує комунікацію між сторонами та дозволяє адміністраторам контролювати роботу платформи. Розширені фільтри, автоматизовані сповіщення та система керування заявками роблять систему сучасним і потужним інструментом для ринку праці.

## 2.2. Функціонал програмного продукту

Для опису та моделювання функціоналу програмного продукту розроблено діаграми випадків використання, що відображають основні процеси роботи системи, взаємодію користувачів із нею.

Для візуалізації основних функціональних можливостей системи розроблено діаграми випадків використання для кожного виду користувача, що демонструють функції платформи. Здобувач роботи може зареєструвати аккаунт, верифікувати його, створювати та редагувати профіль, переглядати залишені заявки, пошук вакансій з фільтром та подання резюме на вакансії. Побудована діаграма випадків використання цього користувача представлена на рис. 2.1.

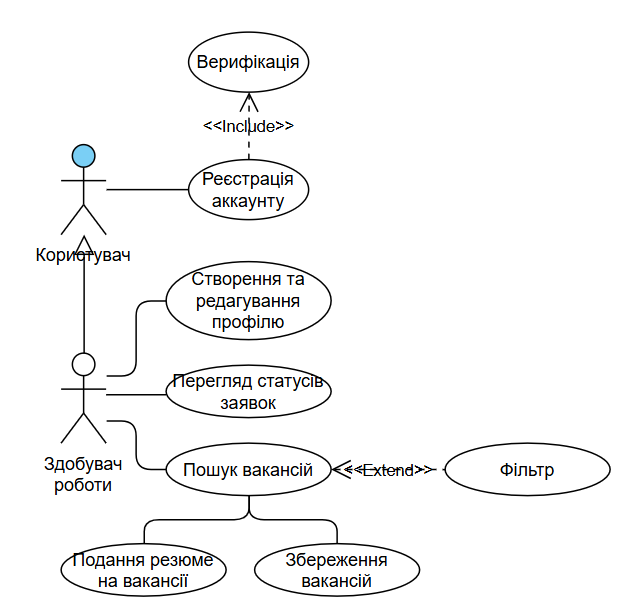


Рис. 2.1 – Діаграма випадків використання для здобувача роботи

Роботодавець також може зареєструвати аккаунт, верифікувати його. Але його основні можливості відрізняються від здобувача роботи. Він може публікувати вакансії, редагувати та архівувати їх, переглядати залишенні заяви та залишати реакцію на заяви. Побудована діаграма випадків використання користувача “роботодавець” представлена на рис. 2.2.

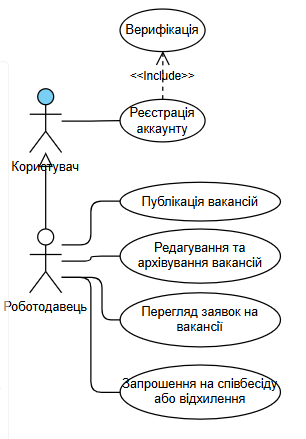


Рис. 2.2 – Діаграма випадків використання для роботодавця

Адміністратор може керуванти користувачами, моніторити активність сайту а також переглядати всі залишені вакансії. Діаграма зображена на рис.2.3.

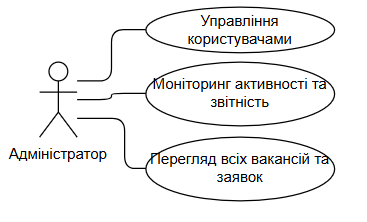


Рис. 2.3 – Діаграма випадків використання для адміністратора

Для відображення процесу пошуку роботи було створено діаграму активності. Вона ілюструє основні етапи: вхід в систему, ініціалізація пошуку, фільтрації, подання заявки на вакансію а також процеси що відбуваються у системи. Діаграма також показує можливі розгалуження в залежності від прийняття рішення щодо подальших дій. Побудована діаграма активності наведена на рис. нижче.

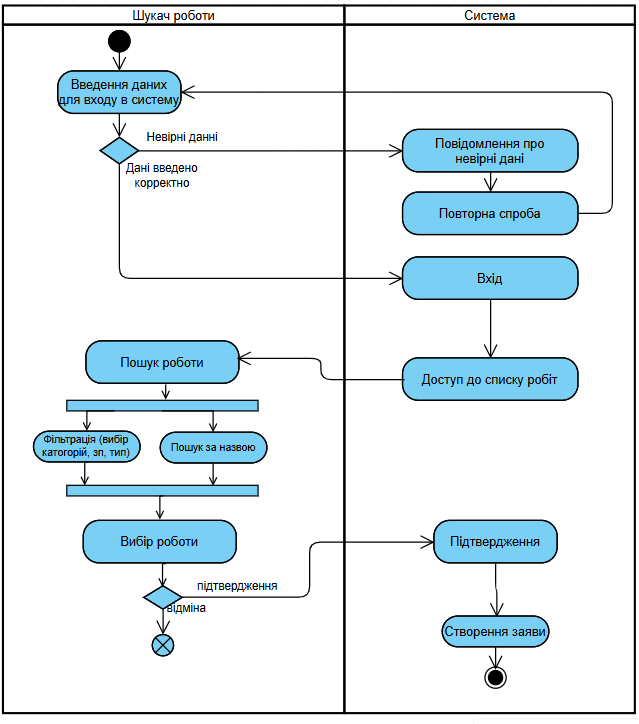


Рис. 2.4 – Діаграма активності пошуку роботи

Проєктування функціоналу програмного продукту включає розробку UML-діаграм, що дозволяють структуровано відобразити її функціональні можливості, архітектуру та динаміку роботи. Діаграми випадків використання демонструють ключові сценарії взаємодії користувачів із платформою, а діаграма активості визначає послідовність дій процесу пошуку роботи.

## 2.3. Проєктування та розробка бази даних

Під час розробки системи пошуку роботи було спроєктовано реляційну базу даних MySQL, яка дозволяє ефективно зберігати й обробляти інформацію про користувачів, роботодавців, вакансії, резюме, заявки та повідомлення.

Розроблена база даних містить такі таблиці:

**1. Таблиця Users**

Містить загальну інформацію про всіх користувачів системи. Розроблена таблиця зображена на рисунку 2.5.

**Атрибути:**

* user\_id – унікальний ідентифікатор користувача (PRIMARY KEY).
* first\_name, last\_name – ім’я та прізвище користувача.
* email – адреса електронної пошти (унікальна).
* password – хеш пароля.
* role – роль користувача: admin, employer, job\_seeker.
* phone – номер телефону.
* verified – чи підтверджено акаунт.
* verification\_token – токен для підтвердження пошти.
* registration\_date – дата реєстрації (за замовчуванням поточна).

**Зв’язки:**

* Пов’язується з таблицями JobSeekerProfiles, EmployerProfiles, Messages.



Рис. 2.5 – Таблиця користувачів

**2. Таблиця JobCategories**

Категорії, до яких можуть належати вакансії та вподобання пошукачів. Розроблена таблиця категорій зображена на рис. 2.6.

**Атрибути:**

* category\_id – унікальний ідентифікатор категорії.
* name – назва категорії (унікальна).

**Зв’язки:**

* Зв’язується з таблицями Jobs, JobSeekerProfiles.

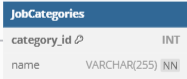


Рис. 2.6 – Таблиця категорій

**3. Таблиця JobSeekerProfiles**

Розроблена таблиця містить у собі розширена інформація про користувача, а саме користувача з ролю шукача роботи. Таблиця розроблена для можливості шукачів роботи налаштувати свій профіль, вказавши необхідну інформацію для пошуку роботи. Ця таблиця зображена на рис. 2.7.

**Атрибути:**

* user\_id – ID користувача (PRIMARY KEY, співпадає з Users.user\_id).
* prefered\_category\_1, prefered\_category\_2 – бажані категорії.
* skills – навички (текст).
* experience – кількість років досвіду.
* city – місто проживання.
* experience\_text – опис досвіду.
* education – освіта.
* expectations – побажання щодо роботи.
* portfolio – посилання або опис портфоліо.

**Зв’язки:**

* user\_id → Users(user\_id)
* prefered\_category\_1, prefered\_category\_2 → JobCategories(category\_id)
* Пов’язана з таблицею Resumes.

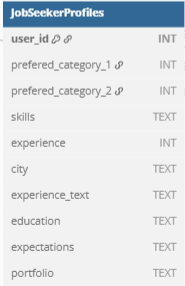


Рис. 2.7 – Таблиця профілів пошукачів роботи

**4. Таблиця Resumes**

Містить резюме які додають пошукачі роботи. Побудована таблиця зображена на рис. 2.8.

**Атрибути:**

* resume\_id – унікальний ідентифікатор резюме.
* title – назва резюме.
* job\_seeker\_id – посилання на пошукача.
* resume\_file – шлях або текстовий вміст файлу.
* upload\_date – дата завантаження.

**Зв’язки:**

* job\_seeker\_id → JobSeekerProfiles(user\_id)



Рис. 2.8 – Таблиця резюме

**5. Таблиця EmployerProfiles**

Зберігає профілі роботодавців. Розроблена таблиця з профілями роботодавців зображена на рис. 2.9.

**Атрибути:**

* user\_id – ID користувача (PRIMARY KEY).
* company\_name – назва компанії.
* logo – шлях до зображення логотипу.
* company\_description – опис компанії.

**Зв’язки:**

* user\_id → Users(user\_id)
* Пов’язана з таблицею Jobs.

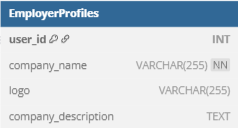


Рис. 2.9 – Таблиця профілів роботодавців

**6. Таблиця Jobs**

Описує вакансії, які створюють роботодавці. Таблиця представлена на рис. 2.10.

**Атрибути:**

* job\_id – унікальний ідентифікатор вакансії.
* employer\_id – роботодавець, що створив вакансію.
* category\_id – категорія вакансії.
* title – назва вакансії.
* description – опис посади.
* requirements – вимоги до кандидата.
* salary, salaryMax – діапазон заробітної плати.
* employment\_type – тип зайнятості.
* location – місцезнаходження.
* status – статус публікації.
* posting\_date – дата створення.

**Зв’язки:**

* employer\_id → EmployerProfiles(user\_id)
* category\_id → JobCategories(category\_id)
* Пов’язана з таблицею Applications.



Рис. 2.10 – Таблиця робіт

**7. Таблиця Applications**

Заявки пошукачів на вакансії. Таблиця зображена на рисуну 2.11.

**Атрибути:**

* application\_id – унікальний ID заявки.
* job\_seeker\_id – користувач, що подав заявку.
* job\_id – вакансія, на яку подано заявку.
* status – статус заявки (pending, accepted, rejected).
* application\_date – дата подачі заявки.

**Зв’язки:**

* job\_seeker\_id → JobSeekerProfiles(user\_id)
* job\_id → Jobs(job\_id)

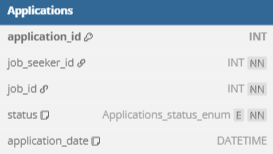


Рис. 2.11 – Таблиця заявки

**8. Таблиця Messages**

Приватні повідомлення між користувачами. Таблиця зображена на рис. 2.12.

**Атрибути:**

* message\_id – унікальний ID повідомлення.
* sender\_id – відправник.
* receiver\_id – одержувач.
* message – вміст повідомлення.
* sent\_date – дата й час надсилання.

**Зв’язки:**

* sender\_id → Users(user\_id)
* receiver\_id → Users(user\_id)

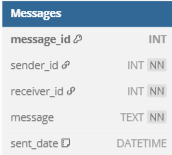


Рис. 2.12 – Таблиця повідомлень

Для зображення логічної структури бази даних та взаємозв’язків між її основними таблицями побудовано ER-діаграму (Entity-Relationship Diagram). Вона містить такі основні сутності та відображає їхні взаємозв’язки. ER-діаграма представлена на рис. 2.13.

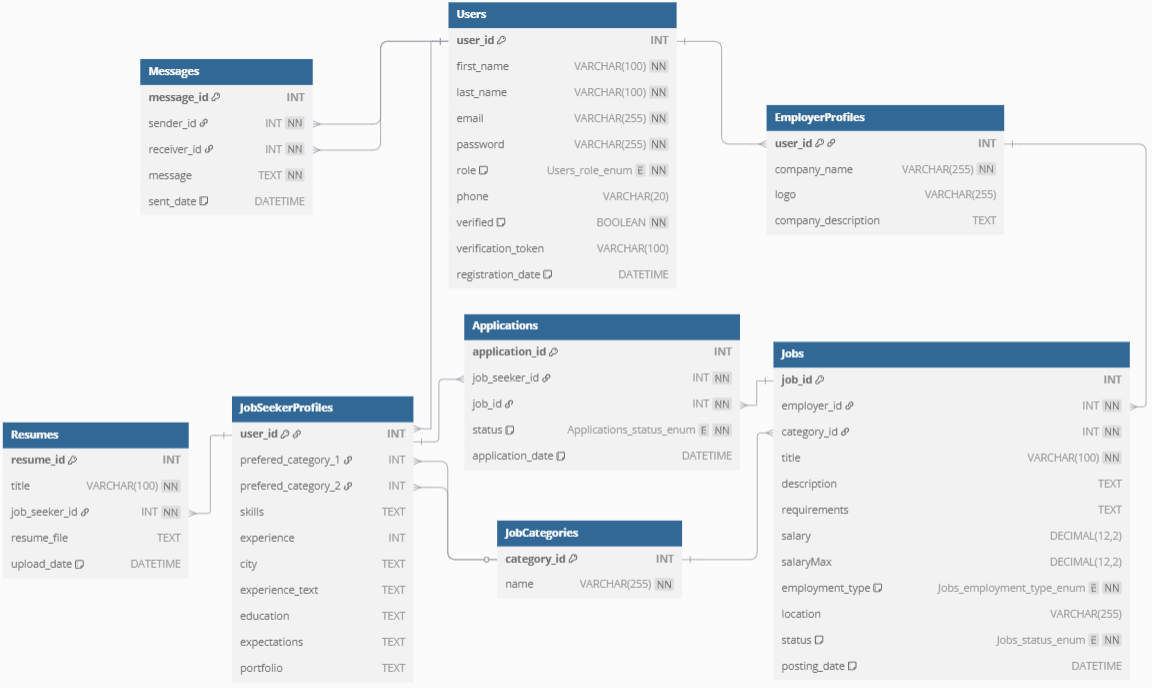


Рис. 2.13 – ER-діаграма бази даних

Отже, проєктована база даних забезпечує логічну структуру з чіткими зв’язками, підтримку ролей користувачів та зберігання резюме, вакансій, заявок і повідомлень.

## 2.4. Компоненти програмного продукту

Програмне забезпечення системи пошуку роботи складається з кількох основних компонентів, які забезпечують її стабільну та ефективну роботу. Взаємодія між компонентами здійснюється за допомогою API, що дозволяє забезпечити швидку обробку запитів та гнучкість у масштабуванні платформи.

1. База даних

Система використовуватиме реляційну базу даних для зберігання інформацій

1. Back-end (серверна частина)

* Технологічний стек: Node.js з використанням Express.js.
* Реалізація REST API для обробки запитів від клієнтів.
* Підключення до MySQL за допомогою mysql2 для зберігання даних.
* Обробка запитів до бази даних.

1. Front-end (інтерфейс користувача)

* Розробка веб-додатку з використанням React.js.
* Розробка сторінок для кожного виду користувачів.
* Використання TailwindCSS для стилізації інтерфейсу.

У підсумку, програмний продукт складається з декількох ключових компонентів, які забезпечують його ефективну роботу. База даних відповідає за зберігання інформації про користувачів, вакансії та заявки. Серверна частина реалізує логіку, обробку запитів та управління даними. Інтерфейс користувача забезпечує зручний доступ до функціоналу системи через веб-додаток, побудований на React.js. Завдяки такій архітектурі платформа зможе ефективно обслуговувати користувачів, забезпечуючи комфортний досвід взаємодії.

Для відображення всіх компонентів розроблюваного програмного та їх взаємозв’язків продукту було побудовано діаграму розгортання, що зображена на рис. 2.14.

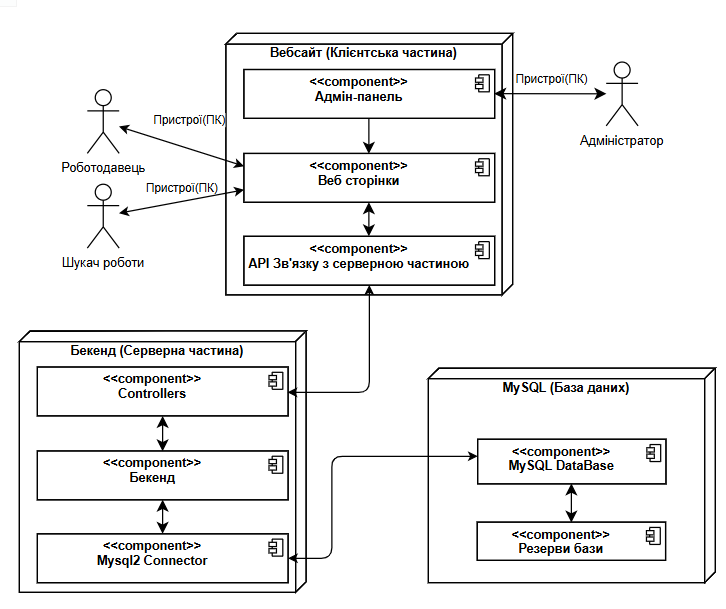


Рис. 2.14 – діаграма розгортання

# Розділ 3. Робочий проєкт

## 3.1 Опис розробки серверної частини

Серверна частина системи реалізована за допомогою Node.js з використанням фреймворку Express, що забезпечує створення масштабованого REST API для взаємодії між клієнтом (фронтендом) та базою даних. Архітектура сервера побудована за принципами модульності, розділення відповідальностей і розширюваності.

Фізична структура проєкту виглядає наступним чином:

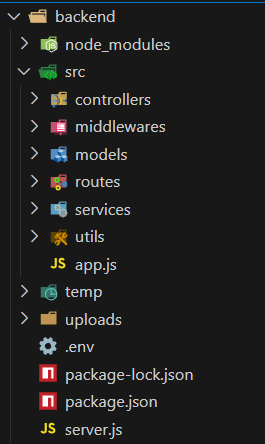


Рис. 3.1 – структура бекенд проєкту

**app.js**

Це головний вхідний файл серверного застосунку. Він відповідає за ініціалізацію додатка Express, підключення middleware, маршрутів, а також запуск сервера.

Основні дії у app.js:

* Імпорт необхідних бібліотек (express, cors, dotenv);
* Ініціалізація middleware (express.json(), cors());
* Імпорт та підключення файлів маршрутів;
* Обробка помилок та 404;
* Підключення до бази даних (через models/db.js);
* Запуск сервера на вказаному порту.

**controllers/**

У папці controllers зберігаються логічні модулі, які обробляють запити користувачів. Кожен контролер відповідає за певну функціональність системи:

* adminController.js — функції адміністратора, такі як блокування користувачів та роботи.
* analysisController.js — отримання аналітичних даних, за допомогою запитів в базу даних.
* authController.js — логіка реєстрації, авторизації користувача, генерації JWT токенів, а також перевірки облікових даних.
* profileController.js — управління профілем користувача: отримання, оновлення персональної інформації.
* jobController.js — створення, редагування, отримання або видалення вакансій. Підтримує фільтрацію за категоріями, пошук за ключовими словами.
* categoryController.js — обробка запитів до категорій вакансій. Доступна для відображення списку доступних категорій.
* applicationController.js — керування заявками здобувачів на вакансії, перевірка статусів.
* resumeController.js — функціонал, пов’язаний із завантаженням, збереженням або оновленням резюме користувачів.

**middlewares/**

* authMiddleware.js — посередник, який перевіряє авторизаційний JWT токен. Якщо токен дійсний — користувач отримує доступ до захищених ресурсів; якщо ні — повертається помилка 401 (Unauthorized).

**models/**

* db.js — модуль підключення до бази даних MySQL. Підключення виконується за допомогою бібліотеки mysql2, або ORM (наприклад, Sequelize чи Prisma в майбутньому). Модуль забезпечує конфігурацію з .env та експорт пулу з’єднань.

**routes/**

Файли в папці routes реєструють шляхи для кожної групи API-ресурсів. Вони делегують обробку контролерам.

* adminRoutes.js — /api/admin (блокування аккаунтів);
* analysisRoutes.js — /api/analysis (аналітичні дані);
* authRoutes.js — /api/auth (реєстрація, вхід);
* profileRoutes.js — /api/profile (отримання, оновлення профілю);
* jobRoutes.js — /api/jobs (керування вакансіями);
* categoryRoutes.js — /api/categories (список категорій);
* applicationRoutes.js — /api/applications (створення, перевірка заявок);
* resumeRoutes.js — /api/resumes (керування файлами резюме).
* services/
* emailService.js —модуль для надсилання електронних листів (наприклад, підтвердження реєстрації, сповіщення про нову заявку). Базується на Nodemailer.

**utils/**

* tokenUtils.js — допоміжні функції для генерації, перевірки, декодування JWT токенів. Це дозволяє уникнути повторення коду у контролерах.

**Папка uploads/**

Папка uploads призначена для зберігання файлів, які завантажують користувачі платформи. У контексті системи пошуку роботи — це, передусім, **резюме здобувачів**, які додаються під час створення або оновлення профілю. Ці файли зберігаються локально на сервері

**Ключові особливості реалізації:**

* Обробляє multipart/form-data запитів, які містять файли.
* При завантаженні файлу створюється унікальна назва (запобігання конфліктам імен).
* Зберігаються лише дозволений формат (.pdf), розмір файлу обмежений.
* При зверненні до резюме система надає захищене посилання або виконує перевірку доступу до файлу (на основі ролі або власності користувача).

Серверна частина системи організована з урахуванням принципів чистої архітектури, що дозволяє легко масштабувати систему, додавати нові функції, а також забезпечує належний рівень безпеки через розмежування доступу, токенізацію та валідацію даних. Вибрана структура дозволяє гнучко підтримувати проєкт та впроваджувати додаткову логіку з мінімальними змінами в основному коді.

## 3.2 Опис розробки клієнтської частини

Клієнтська частина є ключовим компонентом вебплатформи для пошуку роботи, оскільки саме через неї користувачі взаємодіють із системою. Вона відповідає за відображення інтерфейсу, обробку подій, передачу даних до сервера та оновлення контенту в реальному часі. З огляду на вимоги до сучасного вебзастосунку — швидкодія, адаптивність, зручність та гнучкість — для реалізації клієнтської частини було обрано бібліотеку React.

React забезпечує ефективну розробку динамічних та інтерактивних інтерфейсів, підтримує компонентно-орієнтовану архітектуру, яка сприяє повторному використанню коду, та легко інтегрується з іншими бібліотеками і бекенд-сервісами. Завдяки підтримці віртуального DOM та односторінкової навігації (SPA), застосунок забезпечує безперебійний досвід користування, зокрема без потреби перезавантаження сторінки при кожній дії.

Стилизація інтерфейсу здійснюється за допомогою Tailwind CSS, що дозволяє швидко створювати адаптивний та сучасний дизайн.

Структура папок організована наступним чином:

* src/components/ — універсальні UI-компоненти (кнопки, інпути, картки тощо)
* src/pages/ — сторінки для різних ролей (пошукач, роботодавець, адміністратор)
* src/api/ — функції для взаємодії з бекендом через HTTP-запити
* src/layouts/ — компоненти для загального макету (наприклад, з хедером і футером)
* src/tools/ — допоміжні функції (наприклад, для роботи з авторизацією)

**Система ролей**

Вся логіка клієнта побудована з урахуванням трьох основних ролей:

* Пошукач роботи (job\_seeker)
* Роботодавець (employer)
* Адміністратор (admin)

В залежності від ролі користувача, інтерфейс, доступні сторінки та функціонал відрізняються. Це реалізовано як на рівні маршрутизації (через ProtectedRoute та перевірки ролей), так і на рівні відображення компонентів.

**Основні сторінки та їх функціонал**

**1. Головна сторінка (MainPage, MainEmployerPage)**

Головна сторінка має різний вигляд залежно від типу користувача. Для пошукачів роботи вона відображає пошуковий рядок, категорії, популярні вакансії та персоналізовані рекомендації. Для роботодавців передбачена окрема головна сторінка з можливістю швидкого переходу до управління власними вакансіями. Якщо користувач не авторизований, йому показується загальна інформація про сервіс із закликом до реєстрації.

**2. Пошук та перегляд вакансій (JobSearchPage, JobViewPage)**

* JobSearchPage: дозволяє шукати вакансії за ключовими словами, категоріями, типом зайнятості, рівнем зарплати. Результати відображаються у вигляді карток (JobCard).
* JobViewPage: детальна сторінка вакансії. Для пошукача — можливість подати заявку, переглянути статус. Для роботодавця — перегляд заявок на цю вакансію, управління ними (прийняти/відхилити). Для адміністратора — можливість заблокувати/розблокувати вакансію.

**3. Профіль користувача (Profile, ProfileEdit)**

Сторінка профілю користувача відрізняється залежно від ролі. Для пошукача роботи вона дозволяє переглядати та редагувати особисту інформацію, навички, досвід, освіту, очікування, а також керувати резюме. Для роботодавця доступне редагування профілю компанії, зокрема логотипу та контактної інформації. Адміністратор має доступ до адмін-панелі.

**4. Адмін-панель (AdminPage)**

Адмін-панель надає можливість керування користувачами та вакансіями. Адміністратор може блокувати або розблоковувати пошукачів і роботодавців, здійснювати пошук за іменем, email або назвою компанії. Також доступне управління вакансіями з можливістю фільтрації за назвою, компанією чи категорією. Крім цього, передбачено редагування категорій — додавання, видалення та зміну назв. Адміністратор може переглядати статистику за категоріями у вигляді діаграм, що відображають розподіл заявок за статусами: прийняті, скасовані, очікують.

**5. Заявки та резюме (JobApplicationsPage, ResumePage, ResumeReadPage, JobCreatePage)**

* JobApplicationsPage: перегляд усіх заявок, поданих пошукачем, з відображенням статусу (очікує, прийнято, відхилено).
* ResumePage: керування власними резюме, створення, редагування, видалення.
* ResumeReadPage: перегляд резюме (наприклад, для роботодавця або адміністратора).
* JobCreatePage: форма для створення нової вакансії або редагування існуючої. Валідація полів, вибір категорії, типу зайнятості, зарплати, опису

**6. Аутентифікація та реєстрація (LoginPage, RegisterPage, EmployerRegisterPage, VerifyEmailPage)**

Аутентифікація та реєстрація включають окремі сторінки для входу, а також реєстрації пошукача роботи та роботодавця. Після реєстрації користувач має підтвердити свою електронну пошту через спеціальну сторінку верифікації. Усі форми супроводжуються валідацією введених даних та інформуванням про можливі помилки.

**Основні компоненти**

* Header — верхня панель з логотипом, навігацією, кнопками входу/виходу, випадаючим меню для профілю.
* Footer — нижня панель з інформацією про сервіс, контактами, посиланнями.
* JobCard — картка вакансії для списків, з позначкою про блокування, статусом, зарплатою, кількістю заявок.
* Input, Button, TextArea — універсальні компоненти для форм, з підтримкою стилізації через Tailwind.
* CategoryCard — картка категорії для головної сторінки та фільтрів.
* Card, CardContent — компоненти для побудови інформаційних блоків.

**Система стилізації**

Система стилізації інтерфейсу побудована з використанням Tailwind CSS, що забезпечує швидке створення адаптивного дизайну завдяки утилітарним класам для кольорів, відступів, розмірів і шрифтів. Такий підхід дозволяє змінювати кольори та інші параметри за потреби, а також гарантує однаковий вигляд інтерфейсу на різних пристроях.

**Взаємодія з сервером**

Для взаємодії з бекендом були створені окремі модулі API, такі як api/admin.js та api/category.js, які містять функції для основних дій, зокрема отримання списків, створення, редагування, блокування та пошуку. Усі запити до сервера виконуються асинхронно за допомогою fetch або axios з обов’язковою обробкою помилок і відображенням відповідних повідомлень користувачу. Для виконання захищених дій використовується токен авторизації

Клієнтська частина проєкту має важливі особливості, які роблять її зручною, надійною та масштабованою. Всі сторінки та компоненти адаптуються під різні розміри екранів, тому дизайн завжди виглядає добре. Форми мають обов’язкову валідацію з повідомленнями про помилки, що підвищує зручність використання. Після дій користувача, наприклад подачі заявки або блокування, дані оновлюються автоматично без перезавантаження сторінки.

Функції чітко розділені за ролями — для пошукачів, роботодавців і адміністраторів, що забезпечує правильний доступ до різних розділів. У проєкті використовуються сучасні React-хуки (useState, useEffect, useMemo, useCallback), що допомагає ефективно керувати станом і покращує швидкість роботи.

Також у інтерфейсі відображаються статуси вакансій, заявок і користувачів з кольоровими позначками та підказками для кращої інформативності. Завдяки React і Tailwind CSS, клієнтська частина працює швидко, адаптивна та легко розширюється у майбутньому.

## 3.3 Інструкція користувачу

Для початку роботи з платформою на головному меню відображаються категорії вакансій, що дозволяє швидко обрати цікаву сферу (рис. 3.2).

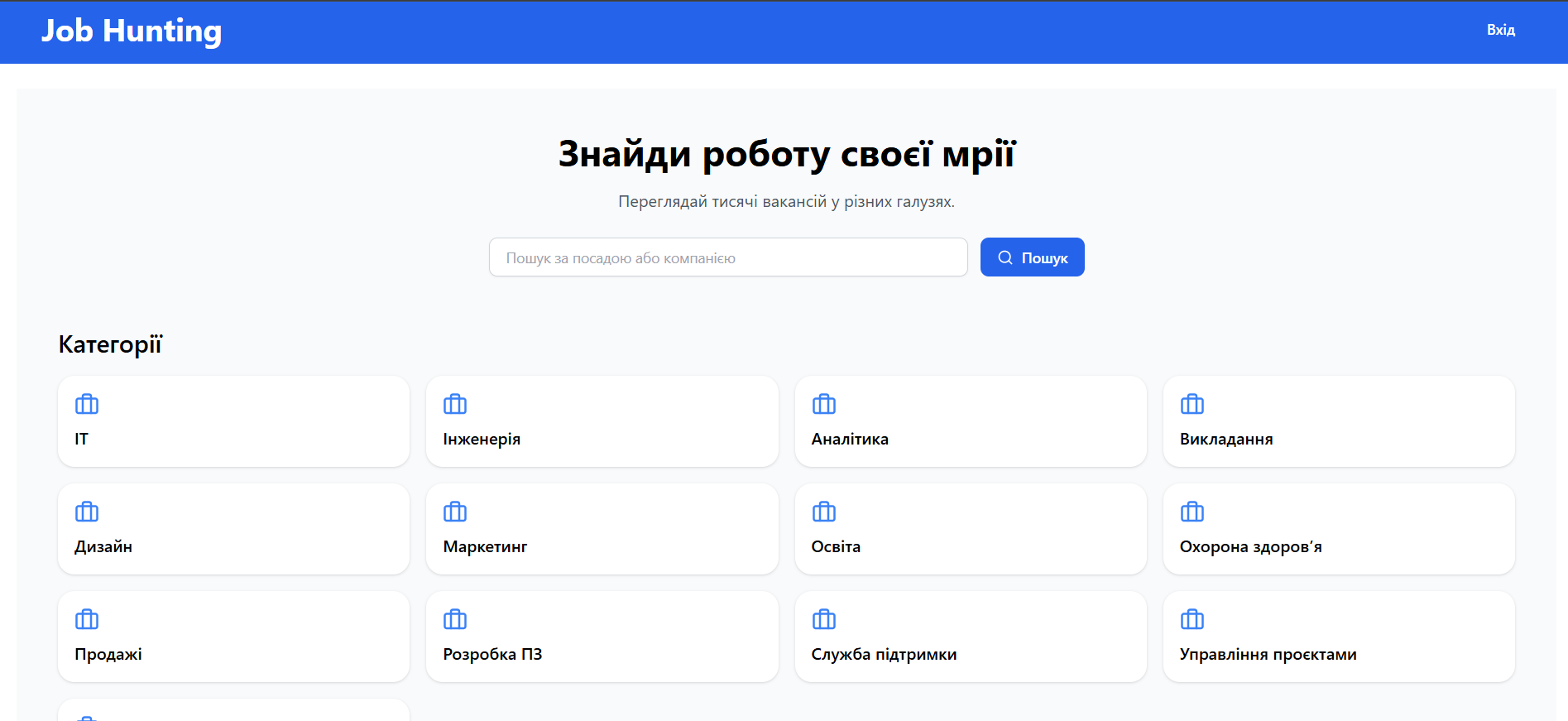


Рис. 3.2 – головне меню

Пошук роботи здійснюється за допомогою пошукового рядка з можливістю фільтрації за категорією, назвою посади, типом зайнятості та зарплатою, що допомагає звузити результати і знайти відповідну вакансію (рис. 3.3).

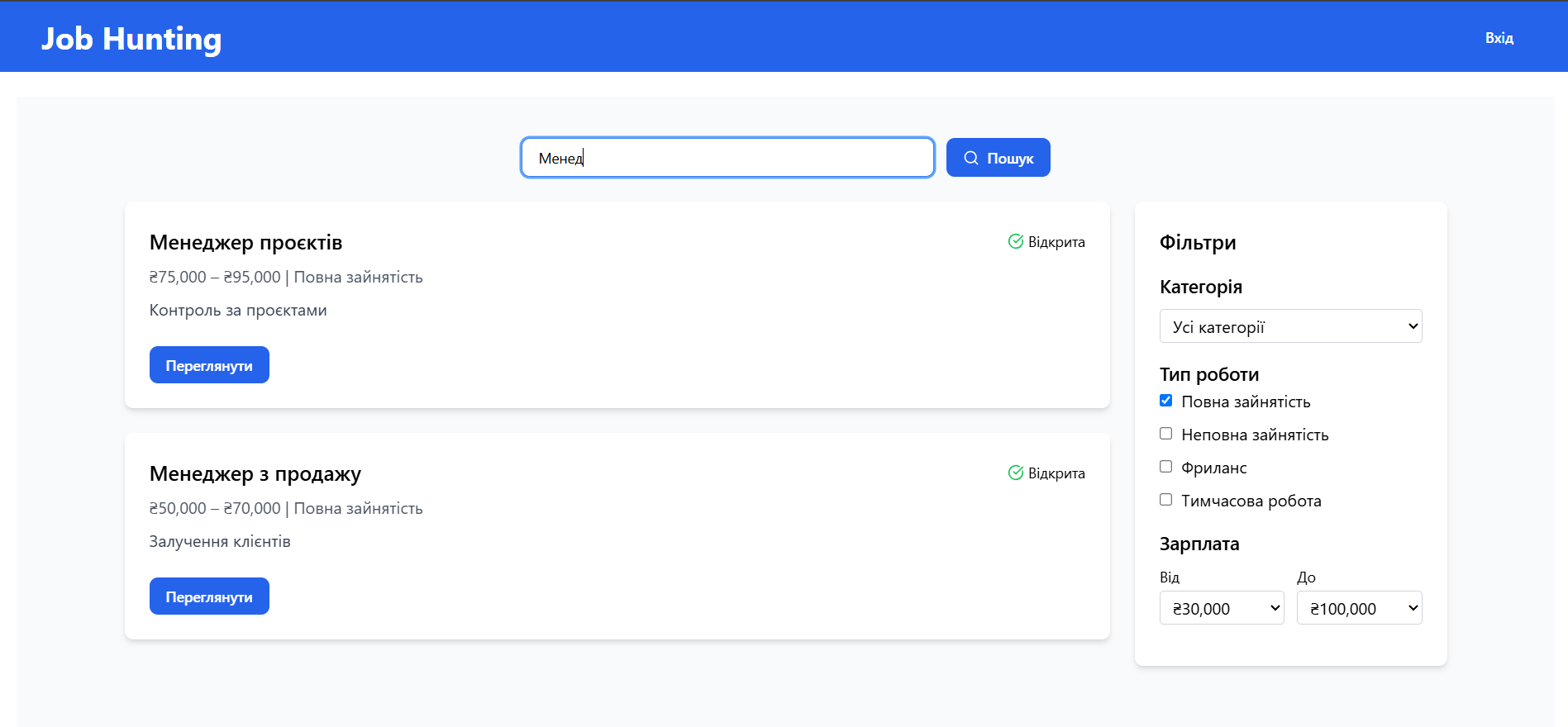


Рис. 3.3 – пошук

Для перегляду детальної інформації про вакансію потрібно натиснути на неї, що відкриває сторінку з описом та умовами роботи (рис. 3.4).

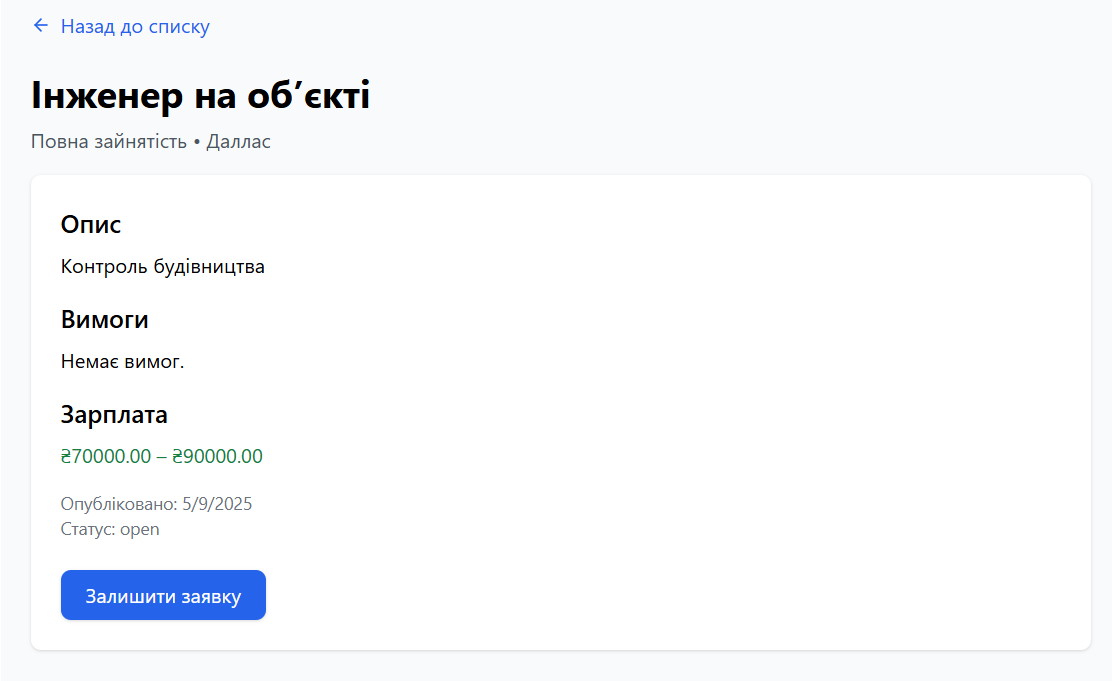


Рис. 3.4 – інформація про вакансію

Реєстрація користувача проводиться через окремі сторінки, де необхідно заповнити основні дані (рис. 3.5).

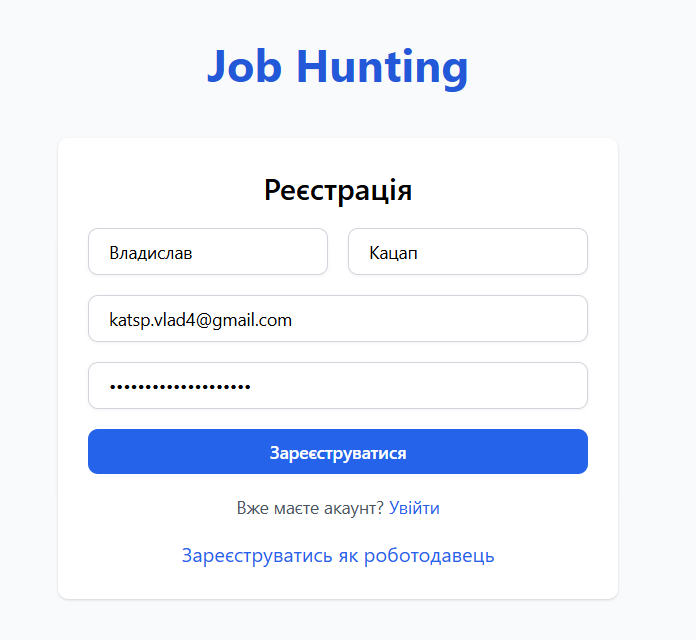


Рис. 3.5 – реєстрація

Після реєстрації користувачу надсилається лист для підтвердження електронної пошти (рис. 3.6).

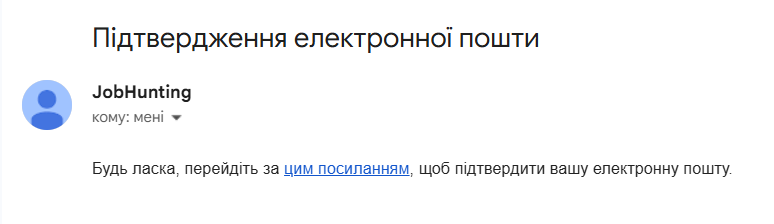


Рис. 3.6 – лист про підтвердження пошти

Результат підтвердження відображається у вигляді відповідного повідомлення (рис. 3.7).

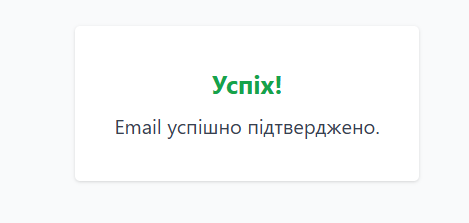


Рис. 3.7 – підтвердження пошти

Пошукач роботи може редагувати свій профіль, додаючи бажані категорії, навички, досвід та іншу особисту інформацію, а також завантажити файли резюме (рис. 3.8).

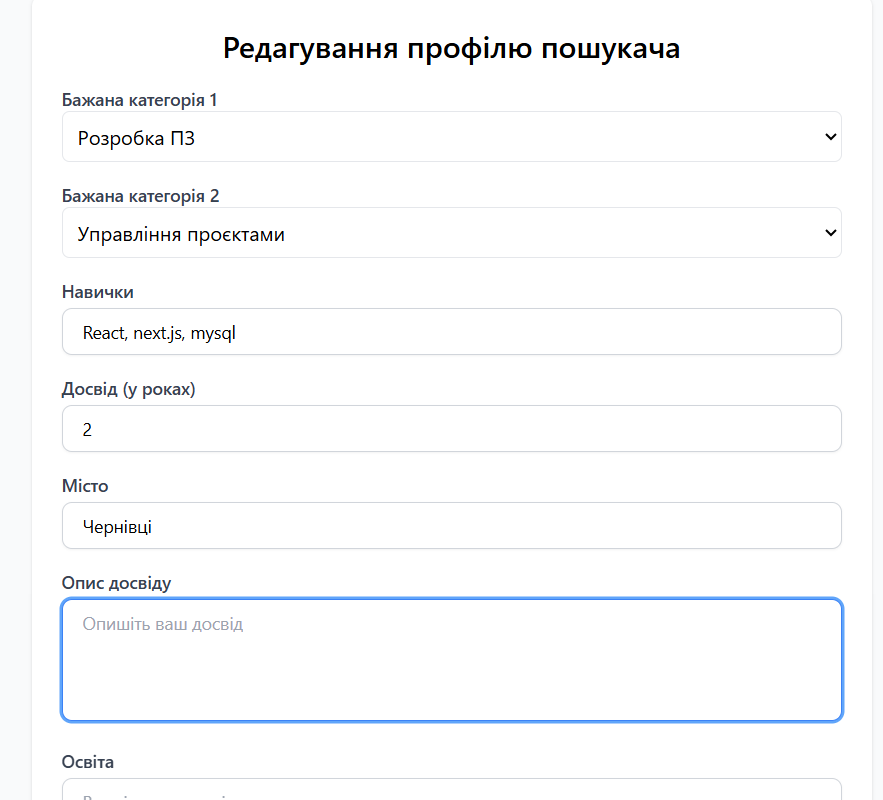


Рис. 3.8 – редагування профілю

Після авторизації система пропонує вакансії саме у обраних категоріях (рис. 3.9).

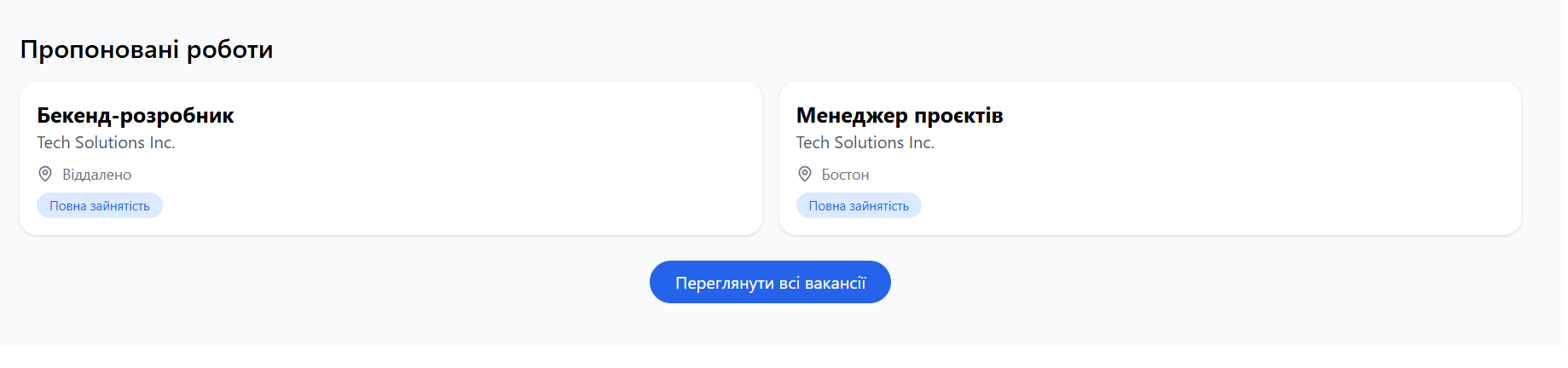


Рис. 3.9 – вакансії

Користувачі можуть переглядати статус своїх заявок на вакансії у відповідному розділі (рис. 3.10).

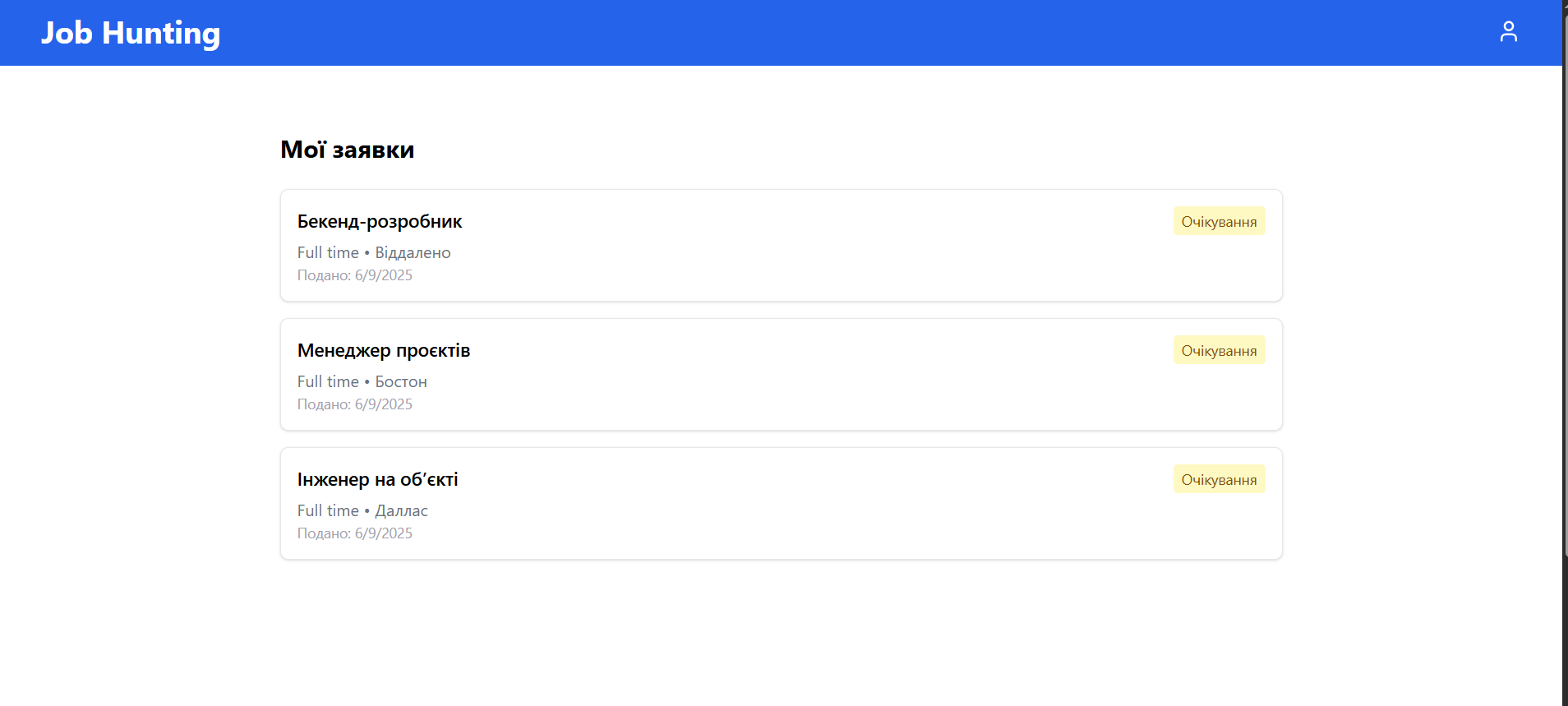


Рис. 3.10 – заявки

Повідомлення про розгляд заявки надходять на електронну пошту (рис. 3.11).

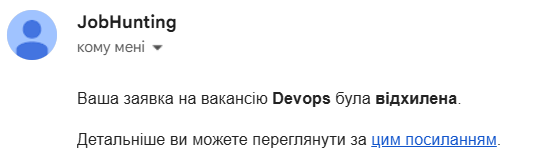


Рис. 3.11 – повідомлення про розгляд заявки

Для роботодавців доступна функція перегляду заявок на свої вакансії з можливістю управління ними (рис. 3.12).

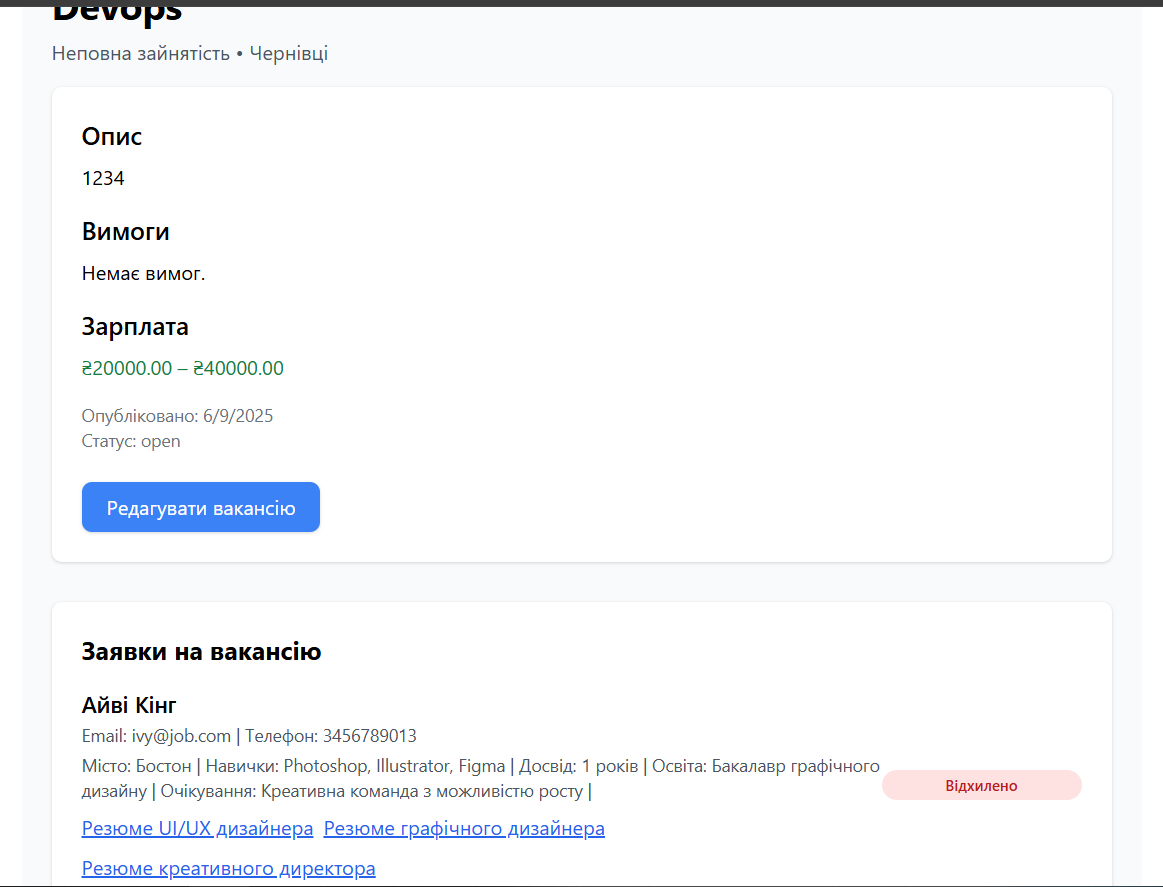


Рис. 3.12 – заявки на вакансії

Адміністратори можуть блокувати акаунти користувачів (пошукачів або роботодавців) для контролю якості сервісу (рис. 3.13).

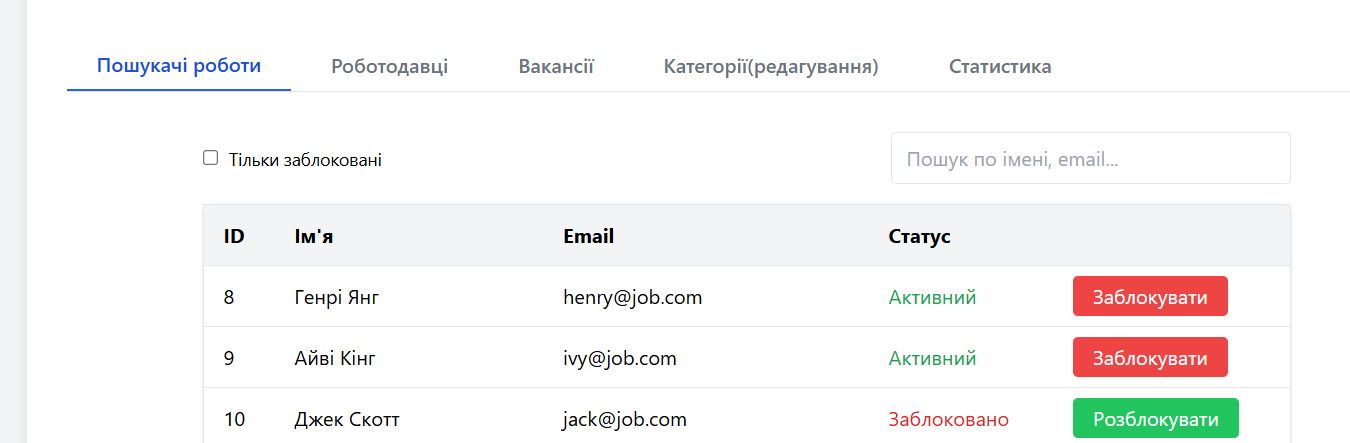


Рис. 3.13 – блокування користувачів

Також доступна статистика по категоріях вакансій із розподілом за статусами (скасовані, прийняті, в очікуванні) (рис. 3.14).

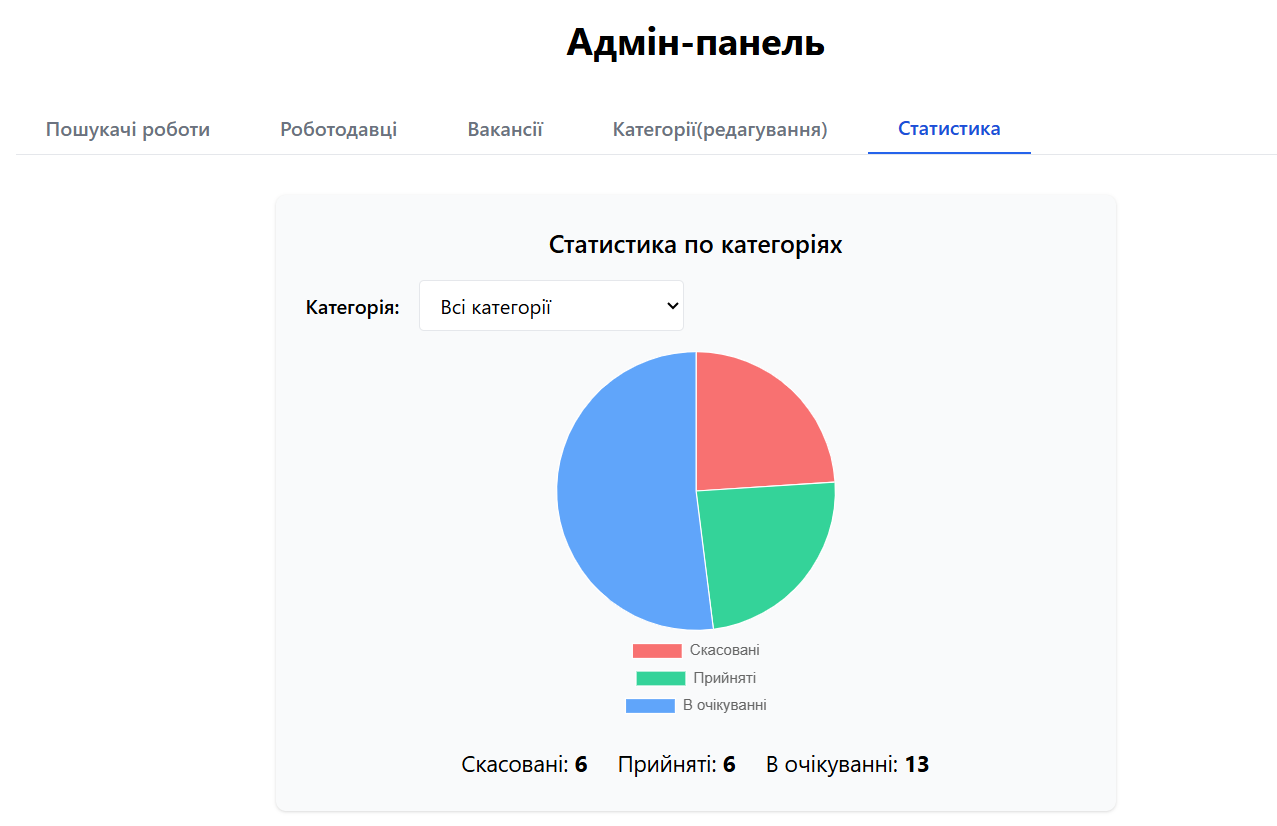


Рис. 3.14 – статистика в адмін-панелі

Адміністратори можуть додавати та редагувати категорії вакансій через відповідний інтерфейс (рис. 3.15).

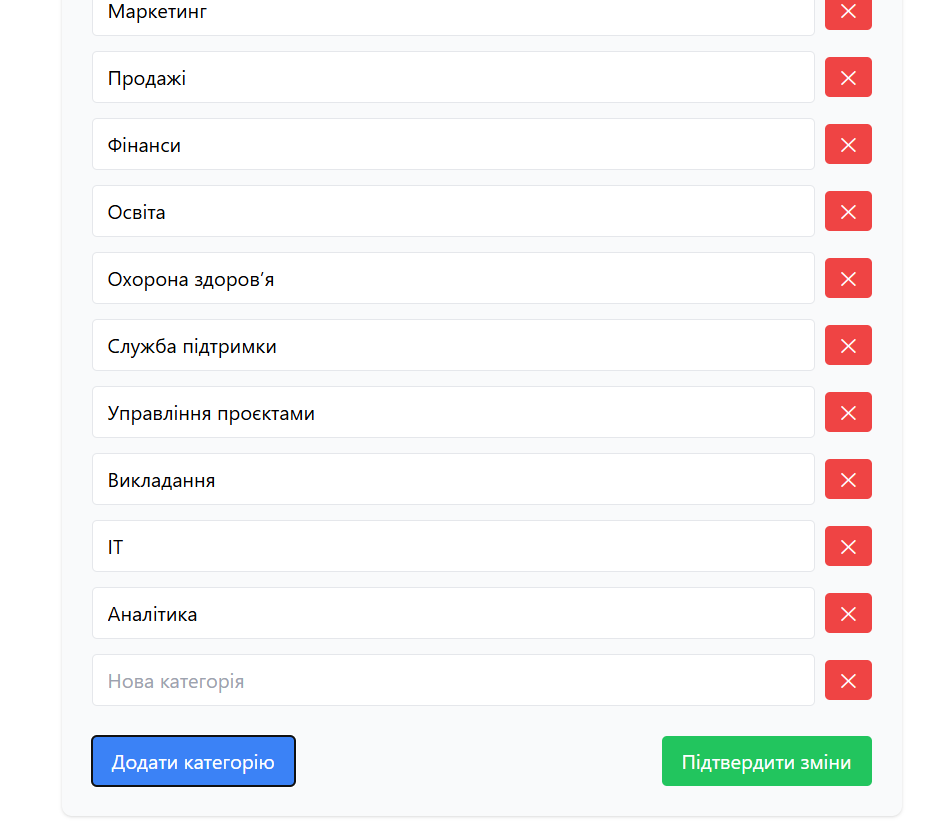


Рис. 3.15 – редагування категорій

# Висновки

В ході виконання проєкту була створена сучасна веб-платформа для взаємодії між пошукачами роботи, роботодавцями та адміністраторами, яка охоплює всі необхідні функціональні компоненти для ефективного пошуку та пропозиції вакансій, управління профілями та вакансіями, а також адміністрування системи. Розробка здійснена із застосуванням сучасних технологій, таких як Node.js Express для бекенду, СУБД MySQL, React для фронтенду та Tailwind CSS для стилізації, що забезпечують високу продуктивність, адаптивність та зручність користувацького інтерфейсу.

Для реалізації функціоналу було чітко розмежовано ролі користувачів, що дозволяє забезпечити безпеку і зручність у роботі з платформою. Асинхронна взаємодія з бекендом через API, використання токенів авторизації та валідація форм підвищують надійність системи і комфорт користувачів. Застосування React-хуків сприяло оптимізації керування станом і покращенню продуктивності.

Проєкт розроблено у вигляді масштабованої та гнучкої системи, що дозволяє легко додавати нові функції і підтримувати стабільну роботу. У процесі розробки були виконані всі основні етапи життєвого циклу програмного забезпечення: аналіз вимог, проєктування, реалізація, тестування та документування.

Створена платформа значно спрощує процес пошуку роботи і рекрутингу, підвищує ефективність комунікації між учасниками ринку праці та має потенціал для подальшого розвитку, зокрема шляхом інтеграції додаткових сервісів і розширення функціоналу.

Таким чином, поставлені цілі проєкту досягнуті — розроблено сучасний, зручний та функціональний сервіс, який відповідає вимогам сучасного ринку праці та має хороші перспективи для масштабування і вдосконалення..

# Список використаних джерел

1. React – A JavaScript library for building user interfaces [Електронний ресурс] – Доступ: <https://react.dev/>
2. Tailwind CSS – Rapidly build modern websites without ever leaving your HTML [Електронний ресурс] – Доступ: <https://tailwindcss.com/>
3. React Router – Declarative routing for React.js [Електронний ресурс] – Доступ: <https://reactrouter.com/>
4. React Documentation [Електронний ресурс] – Доступ: <https://legacy.reactjs.org/docs/getting-started.html>
5. Chart.js Documentation [Електронний ресурс] – Доступ: <https://www.chartjs.org/docs/latest/>
6. Building a RESTful API with Node.js and Express [Електронний ресурс] – Доступ: <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs>
7. React Hook Form – Performant, flexible and extensible forms with easy-to-use validation [Електронний ресурс] – Доступ: <https://react-hook-form.com/>

# Додаток

Нижче додано код компонентів та сторінки додатку. Повний код для додату знаходиться в репозиторії GitHub та доступний за посиланнями.  
Клієнтська частина (frontend):   
Серверна частина (backend):

Компоненти

Компонент Header(/components/Header.jsx)

import React, { useState } from "react";

import { Link, useNavigate } from "react-router-dom";

import { getUser } from "../tools/auth";

import Button from "../components/ui/Button";

import { Menu, User } from "lucide-react";

const Header = () => {

const user = getUser();

const navigate = useNavigate();

const [dropdownOpen, setDropdownOpen] = useState(false);

const handleLogout = () => {

localStorage.removeItem("token");

setDropdownOpen(false);

navigate("/login");

};

const handleSecondButtonClick = () => {

setDropdownOpen(false);

if (user?.role === "employer") {

navigate("/");

} else {

navigate("/applications");

}

};

return (

<header className="bg-blue-600 p-4 px-10">

<div className="flex justify-between items-center">

<Link to="/" className="text-white text-3xl font-bold">

Job Hunting

</Link>

{user ? (

<div className="relative">

<button

onClick={() => setDropdownOpen((prev) => !prev)}

className="text-white flex items-center gap-2 hover:opacity-80"

>

<User className="w-6 h-6" />

</button>

{dropdownOpen && (

<div className="absolute right-0 mt-2 bg-white rounded shadow-md py-2 w-40 z-10">

<button

className="block w-full text-left px-4 py-2 text-sm hover:bg-gray-100"

onClick={() => {

setDropdownOpen(false);

if (user?.role === "admin") {

navigate("/profile");

} else {

navigate("/profile");

}

}}

>

{user?.role === "admin" ? "Адмін-панель" : "Мій профіль"}

</button>

{user?.role === "job\_seeker" && (

<button

className="block w-full text-left px-4 py-2 text-sm hover:bg-gray-100"

onClick={handleSecondButtonClick}

>

Мої заявки

</button>

)}

{user?.role === "employer" && (

<button

className="block w-full text-left px-4 py-2 text-sm hover:bg-gray-100"

onClick={handleSecondButtonClick}

>

Мої роботи

</button>

)}

<hr />

<button

className="block w-full text-left px-4 py-2 text-sm hover:bg-gray-100 text-red-500"

onClick={handleLogout}

>

Вийти

</button>

</div>

)}

</div>

) : (

<Link to="/login">

<Button>Вхід</Button>

</Link>

)}

</div>

</header>

);

};

export default Header;

Компонент Footer (/components/Footer.jsx)  
import React from "react";

import { Link } from "react-router-dom";

const Footer = () => {

return (

<footer className="bg-gray-800 text-white py-6 mt-10">

<div className="max-w-7xl mx-auto px-4 flex flex-col md:flex-row justify-between items-center gap-4">

<Link to="/" className="text-xl font-semibold hover:text-blue-400 transition">

Job Finder

</Link>

<div className="flex gap-6 text-sm">

<Link to="/" className="hover:text-blue-400 transition">Home</Link>

<Link to="/login" className="hover:text-blue-400 transition">Login</Link>

<Link to="/register" className="hover:text-blue-400 transition">Register</Link>

<Link to="/profile" className="hover:text-blue-400 transition">Account</Link>

</div>

<p className="text-xs text-gray-400">&copy; {new Date().getFullYear()} Job Finder. All rights reserved.</p>

</div>

</footer>

);

};

export default Footer;

Компонент Button (/components/ui/Button.jsx)

import React from "react";

const Button = React.forwardRef(({ children, className = "", ...props }, ref) => {

return (

<button

ref={ref}

className={`

inline-flex items-center justify-center rounded-lg bg-blue-600 px-4 py-2 text-sm font-medium

text-white hover:bg-blue-700 focus:outline-none focus:ring-2 focus:ring-blue-400 focus:ring-offset-2

disabled:opacity-50 disabled:cursor-not-allowed transition ${className}

`}

{...props}

>

{children}

</button>

);

});

Button.displayName = "Button";

export default Button;

Компонент Input (/components/ui/Input.jsx)

import React from "react";

const Input = React.forwardRef(({ label, className = "", ...props }, ref) => {

return (

<div className={className}>

{label && (

<label className="block text-sm font-medium text-gray-700 mb-1">

{label}

</label>

)}

<input

ref={ref}

className={`

w-full rounded-lg border border-gray-300 bg-white px-4 py-2 text-sm

placeholder-gray-400 shadow-sm focus:border-blue-500 focus:outline-none focus:ring-2 focus:ring-blue-400

transition duration-150 ease-in-out

`}

{...props}

/>

</div>

);

});

Input.displayName = "Input";

export default Input;

Компонент JobCard (/components/ui/JobCard.jsx)

import React from 'react';

import Button from "./Button";

import { Link } from 'react-router-dom';

import { Clock, CheckCircle, XCircle } from "lucide-react";

const typeMap = {

full\_time: "Повна зайнятість",

part\_time: "Неповна зайнятість",

freelance: "Фриланс",

temporary: "Тимчасова робота",

};

const formatSalary = (min, max) => {

const minFormatted = parseInt(min).toLocaleString();

if (max) {

const maxFormatted = parseInt(max).toLocaleString();

return `₴${minFormatted} – ₴${maxFormatted}`;

} else {

return `₴${minFormatted}`;

}

};

const statusOptions = [

{ value: "draft", label: "Чернетка", icon: <Clock className="inline w-4 h-4 text-gray-400" /> },

{ value: "open", label: "Відкрита", icon: <CheckCircle className="inline w-4 h-4 text-green-500" /> },

{ value: "closed", label: "Закрита", icon: <XCircle className="inline w-4 h-4 text-red-500" /> },

];

const getStatus = (status) => statusOptions.find(s => s.value === status);

const JobCard = ({ job }) => {

const { title, salary, salaryMax, employment\_type, description, status, pending\_applications, blocked } = job;

const statusObj = getStatus(status);

return (

<div className="bg-white p-6 mb-6 rounded-lg shadow-md hover:shadow-lg transition relative">

{blocked && (

<span className="absolute top-3 right-3 bg-red-100 text-red-700 px-3 py-1 rounded text-xs font-semibold shadow">

Заблоковано адміністратором

</span>

)}

<div className="flex items-center justify-between mb-2">

<h3 className="text-xl font-semibold">{title}</h3>

{!blocked && statusObj && (

<span className="ml-2 text-sm flex items-center gap-1">

{statusObj.icon}

{statusObj.label}

</span>

)}

</div>

<p className="text-gray-600">

{formatSalary(salary, salaryMax)} | {typeMap[employment\_type] || employment\_type}

</p>

<p className="text-gray-700 mt-2">

{description.length > 200 ? `${description.slice(0, 200)}...` : description}

</p>

<div className="flex items-center mt-2 gap-2">

{typeof pending\_applications === "number" && (

<span className="flex items-center text-blue-600 text-sm font-medium">

<Clock className="w-4 h-4 mr-1" />

{pending\_applications}

</span>

)}

</div>

<Link to={`/job/${job.job\_id}`}>

<Button className="mt-4">Переглянути</Button>

</Link>

</div>

);

};

export default JobCard;

Компонент TextArea (/components/ui/TextArea.jsx)

import React from "react";

const TextArea = React.forwardRef(({ label, className = "", rows = 4, ...props }, ref) => {

return (

<div className={className}>

{label && (

<label className="block text-sm font-medium text-gray-700 mb-1">

{label}

</label>

)}

<textarea

ref={ref}

rows={rows}

className={`

w-full rounded-lg border border-gray-300 bg-white px-4 py-2 text-sm

placeholder-gray-400 shadow-sm focus:border-blue-500 focus:outline-none focus:ring-2 focus:ring-blue-400

transition duration-150 ease-in-out resize-none

`}

{...props}

/>

</div>

);

});

TextArea.displayName = "TextArea";

export default TextArea;

Компонент CategoryCard (/components/ui/CategoryCard.jsx)

import React from "react";

import { Briefcase } from "lucide-react";

import { Link } from "react-router-dom";

const CategoryCard = ({ category }) => {

return (

<Link to={`/jobs?category\_id=${category.category\_id}`}>

<div className="bg-white p-4 rounded-2xl shadow hover:shadow-md transition">

<Briefcase className="text-blue-500 mb-2" />

<h3 className="font-medium">{category.name}</h3>

</div>

</Link>

);

};

export default CategoryCard;

Сторінки

Сторінка підтвердження пошти(/pages/VerifyEmailPage.jsx)

import React, { useState } from "react";

import { useParams } from "react-router-dom";

import axios from "../api/api";

const VerifyEmailPage = () => {

const { token } = useParams();

const [status, setStatus] = useState("loading");

const [message, setMessage] = useState("");

if (status === "loading" && token) {

setStatus("pending");

axios

.get(`/auth/verify/${token}`)

.then((res) => {

setStatus("success");

setMessage(res.data.message);

})

.catch((err) => {

setStatus("error");

setMessage(err.response?.data?.message || "Something went wrong.");

});

}

return (

<div className="min-h-screen flex items-center justify-center bg-gray-50 p-6">

<div className="bg-white p-8 rounded shadow max-w-md text-center">

{status === "loading" || status === "pending" ? (

<p className="text-gray-600">Перевірка...</p>

) : status === "success" ? (

<>

<h2 className="text-xl font-bold text-green-600 mb-2">Успішно!</h2>

<p className="text-gray-700">{message}</p>

</>

) : (

<>

<h2 className="text-xl font-bold text-red-600 mb-2">Помилка</h2>

<p className="text-gray-700">{message}</p>

</>

)}

</div>

</div>

);

};

export default VerifyEmailPage;

Сторінка адміністратора(/pages/AdminPage.jsx)

import React, { useState, useEffect } from 'react';

import { getCategories, updateCategories } from '../api/category';

import {

getAllJobSeekers,

getAllEmployers,

getAllJobs,

blockUser,

unblockUser,

blockJob,

unblockJob,

} from '../api/admin';

import { Pie } from 'react-chartjs-2';

import {

Chart as ChartJS,

ArcElement,

Tooltip,

Legend,

} from 'chart.js';

ChartJS.register(ArcElement, Tooltip, Legend);

const TABS = [

{ key: 'seekers', label: 'Пошукачі роботи' },

{ key: 'employers', label: 'Роботодавці' },

{ key: 'jobs', label: 'Вакансії' },

{ key: 'categories', label: 'Категорії(редагування)' },

{ key: 'stats', label: 'Статистика' },

];

const AdminPage = () => {

const [activeTab, setActiveTab] = useState('seekers');

const [seekers, setSeekers] = useState([]);

const [seekersSearch, setSeekersSearch] = useState('');

const [employers, setEmployers] = useState([]);

const [employersSearch, setEmployersSearch] = useState('');

const [jobs, setJobs] = useState([]);

const [jobsSearch, setJobsSearch] = useState('');

const [categories, setCategories] = useState([]);

const [catLoading, setCatLoading] = useState(false);

const [catError, setCatError] = useState('');

const [catSuccess, setCatSuccess] = useState('');

const [statsCategories, setStatsCategories] = useState([]);

const [selectedStatCategory, setSelectedStatCategory] = useState('');

const [statsData, setStatsData] = useState(null);

const [statsLoading, setStatsLoading] = useState(false);

const [statsError, setStatsError] = useState('');

const [showBlockedSeekers, setShowBlockedSeekers] = useState(false);

const [showBlockedEmployers, setShowBlockedEmployers] = useState(false);

const [showBlockedJobs, setShowBlockedJobs] = useState(false);

useEffect(() => {

if (activeTab === 'seekers') {

getAllJobSeekers().then(setSeekers);

}

if (activeTab === 'employers') {

getAllEmployers().then(setEmployers);

}

if (activeTab === 'jobs') {

getAllJobs().then(setJobs);

}

if (activeTab === 'categories') {

setCatLoading(true);

setCatError('');

getCategories()

.then(data => {

const sorted = [...data].sort((a, b) => a.category\_id - b.category\_id);

setCategories(sorted.map(c => c.name));

setCatLoading(false);

})

.catch(() => {

setCatError('Не вдалося завантажити категорії');

setCatLoading(false);

});

}

if (activeTab === 'stats') {

setStatsLoading(true);

setStatsError('');

fetch('http://localhost:5000/api/categories')

.then(res => res.json())

.then(data => {

const sorted = [...data].sort((a, b) => a.category\_id - b.category\_id);

setStatsCategories(sorted);

setSelectedStatCategory('');

setStatsLoading(false);

})

.catch(() => {

setStatsError('Не вдалося завантажити категорії');

setStatsLoading(false);

});

}

}, [activeTab]);

useEffect(() => {

if (activeTab !== 'stats') {

setStatsData(null);

return;

}

setStatsLoading(true);

setStatsError('');

let url = 'http://localhost:5000/api/analysis/applications';

if (selectedStatCategory) {

url += `?category=${selectedStatCategory}`;

}

fetch(url)

.then(res => res.json())

.then(data => {

setStatsData(data && data[0] ? data[0] : null);

setStatsLoading(false);

})

.catch(() => {

setStatsError('Не вдалося отримати статистику');

setStatsLoading(false);

});

}, [activeTab, selectedStatCategory]);

const handleCatChange = (idx, value) => {

setCategories(prev => prev.map((c, i) => (i === idx ? value : c)));

};

const handleCatRemove = idx => {

setCategories(prev => prev.filter((\_, i) => i !== idx));

};

const handleCatAdd = () => {

setCategories(prev => [...prev, '']);

};

const handleCatSave = async () => {

setCatLoading(true);

setCatError('');

setCatSuccess('');

try {

await updateCategories({ categories });

setCatSuccess('Категорії оновлено');

} catch (e) {

setCatError('Помилка при збереженні');

}

setCatLoading(false);

};

useEffect(() => {

if (catSuccess) {

const timer = setTimeout(() => setCatSuccess(''), 5000);

return () => clearTimeout(timer);

}

}, [catSuccess]);

const handleUserBlockToggle = async (userId, blocked) => {

if (blocked) {

await unblockUser(userId);

} else {

await blockUser(userId);

}

if (activeTab === 'seekers') getAllJobSeekers().then(setSeekers);

if (activeTab === 'employers') getAllEmployers().then(setEmployers);

};

const handleJobBlockToggle = async (jobId, blocked) => {

if (blocked) {

await unblockJob(jobId);

} else {

await blockJob(jobId);

}

getAllJobs().then(setJobs);

};

const filteredSeekers = seekers

.filter(s =>

[s.first\_name, s.last\_name, s.email]

.join(' ')

.toLowerCase()

.includes(seekersSearch.toLowerCase())

)

.filter(s => (showBlockedSeekers ? s.blocked : true));

const filteredEmployers = employers

.filter(e =>

[e.company\_name, e.first\_name, e.last\_name, e.email]

.join(' ')

.toLowerCase()

.includes(employersSearch.toLowerCase())

)

.filter(e => (showBlockedEmployers ? e.blocked : true));

const filteredJobs = jobs

.filter(j =>

[j.title, j.company\_name, j.category\_name]

.join(' ')

.toLowerCase()

.includes(jobsSearch.toLowerCase())

)

.filter(j => (showBlockedJobs ? j.blocked : true));

return (

<div className="flex justify-center items-start min-h-screen bg-gray-100 py-10">

<div className="w-full max-w-6xl bg-white rounded-xl shadow-lg p-8">

<h1 className="text-3xl font-bold mb-8 text-center">Адмін-панель</h1>

<div className="flex gap-2 mb-8 border-b">

{TABS.map(tab => (

<button

key={tab.key}

onClick={() => setActiveTab(tab.key)}

className={`px-6 py-2 text-base font-medium transition

${activeTab === tab.key

? 'border-b-2 border-blue-600 text-blue-700'

: 'text-gray-500 hover:text-blue-600'}

bg-transparent focus:outline-none`}

>

{tab.label}

</button>

))}

</div>

{activeTab === 'seekers' && (

<div>

<div className="flex justify-center">

<div className="mb-4 flex items-center gap-4 justify-between w-4/5">

<label className="flex items-center gap-2 text-sm">

<input

type="checkbox"

checked={showBlockedSeekers}

onChange={e => setShowBlockedSeekers(e.target.checked)}

/>

Тільки заблоковані

</label>

<input

type="text"

placeholder="Пошук по імені, email..."

value={seekersSearch}

onChange={e => setSeekersSearch(e.target.value)}

className="px-3 py-2 border rounded w-80 focus:outline-none focus:ring focus:border-blue-400"

/>

</div>

</div>

<div className="flex justify-center">

<table className="w-4/5 bg-white rounded-lg shadow border border-gray-200">

<thead>

<tr className="bg-gray-100">

<th className="py-3 px-4 text-left">ID</th>

<th className="py-3 px-4 text-left">Ім'я</th>

<th className="py-3 px-4 text-left">Email</th>

<th className="py-3 px-4 text-left">Статус</th>

<th className="py-3 px-4"></th>

</tr>

</thead>

<tbody>

{filteredSeekers.map(s => (

<tr key={s.user\_id} className="border-t hover:bg-gray-50">

<td className="py-2 px-4">{s.user\_id}</td>

<td className="py-2 px-4">{s.first\_name} {s.last\_name}</td>

<td className="py-2 px-4">{s.email}</td>

<td className="py-2 px-4">

{s.blocked ? (

<span className="text-red-600">Заблоковано</span>

) : (

<span className="text-green-600">Активний</span>

)}

</td>

<td className="py-2 px-4">

<button

onClick={() => handleUserBlockToggle(s.user\_id, s.blocked)}

className={`px-4 py-1 rounded transition ${

s.blocked

? 'bg-green-500 hover:bg-green-600 text-white'

: 'bg-red-500 hover:bg-red-600 text-white'

}`}

>

{s.blocked ? 'Розблокувати' : 'Заблокувати'}

</button>

</td>

</tr>

))}

</tbody>

</table>

</div>

</div>

)}

{activeTab === 'employers' && (

<div>

<div className="flex justify-center">

<div className="mb-4 flex items-center gap-4 justify-between w-4/5">

<label className="flex items-center gap-2 text-sm">

<input

type="checkbox"

checked={showBlockedEmployers}

onChange={e => setShowBlockedEmployers(e.target.checked)}

/>

Тільки заблоковані

</label>

<input

type="text"

placeholder="Пошук по компанії, імені, email..."

value={employersSearch}

onChange={e => setEmployersSearch(e.target.value)}

className="px-3 py-2 border rounded w-80 focus:outline-none focus:ring focus:border-blue-400"

/>

</div>

</div>

<div className="flex justify-center">

<table className="w-4/5 bg-white rounded-lg shadow border border-gray-200">

<thead>

<tr className="bg-gray-100">

<th className="py-3 px-4 text-left">ID</th>

<th className="py-3 px-4 text-left">Компанія</th>

<th className="py-3 px-4 text-left">Ім'я</th>

<th className="py-3 px-4 text-left">Email</th>

<th className="py-3 px-4 text-left">Статус</th>

<th className="py-3 px-4"></th>

</tr>

</thead>

<tbody>

{filteredEmployers.map(e => (

<tr key={e.user\_id} className="border-t hover:bg-gray-50">

<td className="py-2 px-4">{e.user\_id}</td>

<td className="py-2 px-4">{e.company\_name}</td>

<td className="py-2 px-4">{e.first\_name} {e.last\_name}</td>

<td className="py-2 px-4">{e.email}</td>

<td className="py-2 px-4">

{e.blocked ? (

<span className="text-red-600">Заблоковано</span>

) : (

<span className="text-green-600">Активний</span>

)}

</td>

<td className="py-2 px-4">

<button

onClick={() => handleUserBlockToggle(e.user\_id, e.blocked)}

className={`px-4 py-1 rounded transition ${

e.blocked

? 'bg-green-500 hover:bg-green-600 text-white'

: 'bg-red-500 hover:bg-red-600 text-white'

}`}

>

{e.blocked ? 'Розблокувати' : 'Заблокувати'}

</button>

</td>

</tr>

))}

</tbody>

</table>

</div>

</div>

)}

{activeTab === 'jobs' && (

<div>

<div className="flex justify-center">

<div className="mb-4 flex items-center gap-4 justify-between w-4/5">

<label className="flex items-center gap-2 text-sm">

<input

type="checkbox"

checked={showBlockedJobs}

onChange={e => setShowBlockedJobs(e.target.checked)}

/>

Тільки заблоковані

</label>

<input

type="text"

placeholder="Пошук по назві, компанії або категорії..."

value={jobsSearch}

onChange={e => setJobsSearch(e.target.value)}

className="px-3 py-2 border rounded w-80 focus:outline-none focus:ring focus:border-blue-400"

/>

</div>

</div>

<div className="flex justify-center">

<table className="w-4/5 bg-white rounded-lg shadow border border-gray-200">

<thead>

<tr className="bg-gray-100">

<th className="py-3 px-4 text-left">ID</th>

<th className="py-3 px-4 text-left">Назва</th>

<th className="py-3 px-4 text-left">Компанія</th>

<th className="py-3 px-4 text-left">Категорія</th>

<th className="py-3 px-4 text-left">Статус</th>

<th className="py-3 px-4"></th>

</tr>

</thead>

<tbody>

{filteredJobs.map(j => (

<tr key={j.job\_id} className="border-t hover:bg-gray-50">

<td className="py-2 px-4">{j.job\_id}</td>

<td className="py-2 px-4">{j.title}</td>

<td className="py-2 px-4">{j.company\_name}</td>

<td className="py-2 px-4">{j.category\_name}</td>

<td className="py-2 px-4">

{j.blocked ? (

<span className="text-red-600">Заблоковано</span>

) : (

<span className="text-green-600">Активна</span>

)}

</td>

<td className="py-2 px-4">

<button

onClick={() => handleJobBlockToggle(j.job\_id, j.blocked)}

className={`px-4 py-1 rounded transition ${

j.blocked

? 'bg-green-500 hover:bg-green-600 text-white'

: 'bg-red-500 hover:bg-red-600 text-white'

}`}

>

{j.blocked ? 'Розблокувати' : 'Заблокувати'}

</button>

</td>

</tr>

))}

</tbody>

</table>

</div>

</div>

)}

{activeTab === 'stats' && (

<div className="flex flex-col items-center w-full">

<div className="w-full max-w-2xl bg-gray-50 rounded-lg shadow p-6">

<h2 className="text-xl font-semibold mb-4 text-center">Статистика по категоріях</h2>

{statsError && <div className="text-red-500 mb-2">{statsError}</div>}

<div className="mb-4 flex flex-col sm:flex-row items-center gap-4">

<label className="font-medium">Категорія:</label>

<select

className="px-3 py-2 border rounded focus:outline-none focus:ring focus:border-blue-400"

value={selectedStatCategory}

onChange={e => setSelectedStatCategory(e.target.value)}

disabled={statsLoading}

>

<option value="">Всі категорії</option>

{statsCategories.map(cat => (

<option key={cat.category\_id} value={cat.category\_id}>

{cat.name}

</option>

))}

</select>

</div>

{statsLoading && <div className="text-blue-500 mb-2">Завантаження...</div>}

{statsData && (

<div className="flex flex-col items-center">

<div className="w-40 sm:w-56">

<Pie

data={{

labels: ['Скасовані', 'Прийняті', 'В очікуванні'],

datasets: [

{

data: [

statsData.cancelled || 0,

statsData.accepted || 0,

statsData.pending || 0,

],

backgroundColor: [

'#f87171', // red-400

'#34d399', // green-400

'#60a5fa', // blue-400

],

borderWidth: 1,

},

],

}}

options={{

plugins: {

legend: { position: 'bottom' },

},

maintainAspectRatio: false,

}}

height={300}

/>

</div>

<div className="mt-4 flex gap-6 text-lg">

<span>Скасовані: <b>{statsData.cancelled || 0}</b></span>

<span>Прийняті: <b>{statsData.accepted || 0}</b></span>

<span>В очікуванні: <b>{statsData.pending || 0}</b></span>

</div>

</div>

)}

</div>

</div>

)}

{activeTab === 'categories' && (

<div className="flex flex-col items-center w-full">

<div className="w-full max-w-2xl bg-gray-50 rounded-lg shadow p-6">

<h2 className="text-xl font-semibold mb-4 text-center">Категорії</h2>

{catLoading && <div className="text-blue-500 mb-2">Завантаження...</div>}

{catError && <div className="text-red-500 mb-2">{catError}</div>}

<div className="space-y-3">

{categories.map((cat, idx) => (

<div key={idx} className="flex items-center gap-2">

<input

type="text"

placeholder="Нова категорія"

value={cat}

onChange={e => handleCatChange(idx, e.target.value)}

className="flex-1 px-3 py-2 border rounded focus:outline-none focus:ring focus:border-blue-400"

/>

<button

onClick={() => handleCatRemove(idx)}

className="bg-red-500 hover:bg-red-600 text-white px-3 py-1 rounded"

disabled={categories.length === 1}

title="Видалити"

>

&#10005;

</button>

</div>

))}

</div>

<div className="flex gap-3 mt-6">

<button

onClick={handleCatAdd}

className="bg-blue-500 hover:bg-blue-600 text-white px-4 py-2 rounded"

>

Додати категорію

</button>

<button

onClick={handleCatSave}

className="bg-green-500 hover:bg-green-600 text-white px-4 py-2 rounded ml-auto"

disabled={catLoading}

>

Підтвердити зміни

</button>

</div>

{/\* Повідомлення про успіх внизу форми \*/}

{catSuccess && (

<div className="fixed left-0 right-0 bottom-8 flex justify-center pointer-events-none">

<div className="bg-green-100 border border-green-300 text-green-700 rounded px-6 py-3 shadow text-center pointer-events-auto">

{catSuccess}

</div>

</div>

)}

</div>

</div>

)}

</div>

</div>

);

};

export default AdminPage;

Сторінка реєстрації роботодавця(/pages/EmployerRegisterPage.jsx)

import React, { useState, useRef } from "react";

import { Link, useNavigate } from "react-router-dom";

import Input from "../components/ui/Input";

import Button from "../components/ui/Button";

import { registerEmployer } from "../api/auth";

import TextArea from "../components/ui/TextArea";

import Logo from "../components/Logo";

import Notification from "../components/Notification";

export default function EmployerRegisterPage() {

const [form, setForm] = useState({

first\_name: "",

last\_name: "",

email: "",

password: "",

company\_name: "",

logo: "",

company\_description: "",

});

const [error, setError] = useState("");

const [preview, setPreview] = useState("");

const [loading, setLoading] = useState(false);

const [notification, setNotification] = useState("");

const fileInputRef = useRef(null);

const navigate = useNavigate();

const handleChange = (e) => {

setForm({ ...form, [e.target.name]: e.target.value });

};

const handlePhotoClick = () => {

fileInputRef.current.click();

};

const resizeImage = (file, maxSize = 250) =>

new Promise((resolve) => {

const img = new window.Image();

const reader = new FileReader();

reader.onload = (e) => {

img.onload = () => {

const canvas = document.createElement("canvas");

const ctx = canvas.getContext("2d");

let width = img.width;

let height = img.height;

if (width > height) {

if (width > maxSize) {

height \*= maxSize / width;

width = maxSize;

}

} else {

if (height > maxSize) {

width \*= maxSize / height;

height = maxSize;

}

}

canvas.width = width;

canvas.height = height;

ctx.drawImage(img, 0, 0, width, height);

resolve(canvas.toDataURL("image/png"));

};

img.src = e.target.result;

};

reader.readAsDataURL(file);

});

const handlePhotoChange = async (e) => {

const file = e.target.files[0];

if (file) {

const resized = await resizeImage(file, 250);

setForm((prev) => ({ ...prev, logo: resized }));

setPreview(resized);

}

};

const isBase64Image = (str) =>

typeof str === "string" && str.startsWith("data:image/");

const handleSubmit = async (e) => {

e.preventDefault();

setError("");

let dataToSend = { ...form };

console.log("Form data to send:", dataToSend);

if (dataToSend.logo && !isBase64Image(dataToSend.logo)) {

try {

const response = await fetch(dataToSend.logo);

const blob = await response.blob();

const reader = new FileReader();

dataToSend.logo = await new Promise((resolve) => {

reader.onloadend = () => resolve(reader.result);

reader.readAsDataURL(blob);

});

} catch {

}

}

try {

setLoading(true);

await registerEmployer(dataToSend);

setNotification("Реєстрація успішна. Перевірте пошту для підтвердження.");

setTimeout(() => {

navigate("/login");

}, 1000);

} catch (err) {

setLoading(false);

setError(err.response?.data?.message || "Не вдалося зареєструватися");

} finally {

setLoading(false);

}

};

return (

<div className="min-h-screen bg-gray-50 px-6 py-10">

<Logo />

<Notification

message={notification}

type="success"

onClose={() => setNotification("")}

/>

<form

onSubmit={handleSubmit}

className="bg-white p-6 rounded-lg shadow max-w-md mx-auto mt-10"

>

<h2 className="text-2xl font-semibold mb-4 text-center">

Реєстрація роботодавця

</h2>

{error && <p className="text-red-500 text-sm mb-3">{error}</p>}

<div className="flex flex-col items-center mb-6">

<div

className="relative group cursor-pointer"

onClick={handlePhotoClick}

title="Змініть фото"

>

<img

src={

preview ||

form.logo ||

process.env.PUBLIC\_URL + "/default.png"

}

alt="Логотип компанії"

className="w-24 h-24 rounded-full object-cover border-2 border-gray-300"

/>

<div className="absolute inset-0 bg-black bg-opacity-40 rounded-full flex items-center justify-center opacity-0 group-hover:opacity-100 transition-opacity">

<span className="text-white text-sm">Змініть фото</span>

</div>

<input

type="file"

accept="image/\*"

className="hidden"

ref={fileInputRef}

onChange={handlePhotoChange}

/>

</div>

</div>

<div className="mb-4 flex gap-4">

<Input

name="first\_name"

placeholder="Ім'я"

value={form.first\_name}

onChange={handleChange}

required

className="flex-1"

/>

<Input

name="last\_name"

placeholder="Прізвище"

value={form.last\_name}

onChange={handleChange}

required

className="flex-1"

/>

</div>

<div className="mb-4">

<Input

name="email"

type="email"

placeholder="Електронна пошта"

value={form.email}

onChange={handleChange}

required

/>

</div>

<div className="mb-4">

<Input

name="password"

type="password"

placeholder="Пароль"

value={form.password}

onChange={handleChange}

required

/>

</div>

<div className="mb-4">

<Input

name="company\_name"

placeholder="Назва компанії"

value={form.company\_name}

onChange={handleChange}

required

/>

</div>

<div className="mb-4">

<TextArea

name="company\_description"

placeholder="Опис компанії"

value={form.company\_description}

onChange={handleChange}

/>

</div>

<Button type="submit" className="w-full" disabled={loading}>

{loading ? (

<span className="flex items-center justify-center gap-2">

<span className="w-5 h-5 border-2 border-white border-t-blue-500 rounded-full animate-spin"></span>

Реєстрація...

</span>

) : (

"Зареєструватися"

)}

</Button>

<div className="mt-4 text-center">

<Link to="/register" className="text-blue-600 hover:underline">

Зареєструватись як пошукач роботи

</Link>

</div>

</form>

</div>

);

}

Сторінка реєстрації(/pages/RegisterPage.jsx)

import React, { useState } from "react";

import { Link } from "react-router-dom";

import Input from "../components/ui/Input";

import Button from "../components/ui/Button";

import Logo from "../components/Logo";

import { registerUser } from "../api/auth";

import Notification from "../components/Notification";

export default function RegisterForm() {

const [form, setForm] = useState({

first\_name: "",

last\_name: "",

email: "",

password: "",

role: "job\_seeker"

});

const [error, setError] = useState("");

const [loading, setLoading] = useState(false);

const [notification, setNotification] = useState("");

const handleChange = (e) =>

setForm({ ...form, [e.target.name]: e.target.value });

const handleRegister = async (e) => {

e.preventDefault();

setError("");

setLoading(true);

try {

const result = await registerUser(form);

setNotification("Користувача зареєстровано. Підтвердіть свою пошту.");

setTimeout(() => {

window.location.href = "/login";

}, 1000);

} catch (err) {

setError(err.response?.data?.message || "Registration failed");

} finally {

setLoading(false);

}

};

return (

<div className="min-h-screen bg-gray-50 px-6 py-10">

<Logo />

<Notification

message={notification}

type="success"

onClose={() => setNotification("")}

/>

<form

onSubmit={handleRegister}

className="bg-white p-6 rounded-lg shadow max-w-md mx-auto mt-10"

>

<h2 className="text-2xl font-semibold mb-4 text-center">Реєстрація</h2>

{error && <p className="text-red-500 text-sm mb-3">{error}</p>}

<div className="mb-4 flex gap-4">

<Input

name="first\_name"

placeholder="Ім'я"

value={form.first\_name}

onChange={handleChange}

required

className="flex-1"

/>

<Input

name="last\_name"

placeholder="Прізвище"

value={form.last\_name}

onChange={handleChange}

required

className="flex-1"

/>

</div>

<div className="mb-4">

<Input

name="email"

type="email"

placeholder="Електронна пошта"

value={form.email}

onChange={handleChange}

required

/>

</div>

<div className="mb-4">

<Input

name="password"

type="password"

placeholder="Пароль"

value={form.password}

onChange={handleChange}

required

/>

</div>

<Button type="submit" className="w-full" disabled={loading}>

{loading ? (

<span className="flex items-center justify-center gap-2">

<span className="w-5 h-5 border-2 border-white border-t-blue-500 rounded-full animate-spin"></span>

Реєстрація...

</span>

) : (

"Зареєструватися"

)}

</Button>

<p className="mt-4 text-sm text-gray-600 text-center">

Вже маєте акаунт?{" "}

<Link to="/login" className="text-blue-600 hover:underline">

Увійти

</Link>

</p>

<div className="mt-4 text-center">

<Link to="/register/employer" className="text-blue-600 hover:underline">

Зареєструватись як роботодавець

</Link>

</div>

</form>

</div>

);

}

Сторінка авторизації(/pages/LoginPage.jsx)

import React, { useState } from "react";

import { Link } from "react-router-dom";

import Input from "../components/ui/Input";

import Button from "../components/ui/Button";

import Logo from "../components/Logo";

import { loginUser } from "../api/auth";

import Notification from "../components/Notification";

export default function LoginPage() {

const [email, setEmail] = useState("");

const [password, setPassword] = useState("");

const [error, setError] = useState("");

const [loading, setLoading] = useState(false);

const [notification, setNotification] = useState("");

const handleLogin = async (e) => {

e.preventDefault();

setError("");

setLoading(true);

try {

const { token } = await loginUser({ email, password });

localStorage.setItem("token", token);

setNotification("Вхід успішний!");

setTimeout(() => {

window.location.href = "/profile";

}, 1000);

} catch (err) {

setError(err.response?.data?.message || "Не вдалося увійти");

} finally {

setLoading(false);

}

};

return (

<div className="min-h-screen bg-gray-50 px-6 py-10">

<Logo />

<Notification

message={notification}

type="success"

onClose={() => setNotification("")}

/>

<form

onSubmit={handleLogin}

className="bg-white p-6 rounded-lg shadow max-w-md mx-auto mt-10"

>

<h2 className="text-2xl font-semibold mb-4 text-center">Вхід</h2>

{error && <p className="text-red-500 text-sm mb-3">{error}</p>}

<div className="mb-4">

<Input

type="email"

placeholder="Електронна пошта"

value={email}

onChange={(e) => setEmail(e.target.value)}

required

/>

</div>

<div className="mb-4">

<Input

type="password"

placeholder="Пароль"

value={password}

onChange={(e) => setPassword(e.target.value)}

required

/>

</div>

<Button type="submit" className="w-full" disabled={loading}>

{loading ? (

<span className="flex items-center justify-center gap-2">

<span className="w-5 h-5 border-2 border-white border-t-blue-500 rounded-full animate-spin"></span>

Вхід...

</span>

) : (

"Увійти"

)}

</Button>

<p className="mt-4 text-sm text-gray-600 text-center">

Ще не маєте акаунта?{" "}

<Link to="/register" className="text-blue-600 hover:underline">

Зареєструйтесь тут

</Link>

</p>

</form>

</div>

);

}

Головна сторінка(/pages/MainPage.jsx)

import React, { useEffect, useState } from "react";

import { Link, useNavigate } from "react-router-dom";

import Input from "../components/ui/Input";

import Button from "../components/ui/Button";

import Card from "../components/ui/Card";

import CardContent from "../components/ui/CardContent";

import CategoryCard from "../components/ui/CategoryCard";

import { MapPin, Search } from "lucide-react";

import { getCategories } from "../api/category";

import { getSuggestedJobs } from "../api/job";

const employmentTypeMap = {

full\_time: "Повна зайнятість",

part\_time: "Неповна зайнятість",

freelance: "Фриланс",

temporary: "Тимчасова робота",

};

export default function MainPage() {

const [categories, setCategories] = useState([]);

const [suggestedJobs, setSuggestedJobs] = useState([]);

const [searchTerm, setSearchTerm] = useState("");

const [preferredCategory, setPreferredCategory] = useState(null);

const navigate = useNavigate();

useEffect(() => {

const fetchInitialData = async () => {

try {

const [categoryData, jobData, profileData] = await Promise.all([

getCategories(),

getSuggestedJobs(),

]);

setCategories(categoryData);

setSuggestedJobs(jobData);

if (profileData?.prefered\_category\_1) {

setPreferredCategory(profileData.prefered\_category\_1);

}

} catch (error) {

console.error("Помилка при завантаженні даних:", error);

}

};

fetchInitialData();

}, []);

const handleSearch = () => {

const query = searchTerm.trim();

if (query) {

navigate(`/jobs?search=${encodeURIComponent(query)}`);

} else {

navigate("/jobs");

}

};

const handleSeeAllClick = () => {

if (preferredCategory) {

navigate(`/jobs?category\_id=${preferredCategory}`);

} else {

navigate("/jobs");

}

};

return (

<div className="min-h-screen bg-gray-50 px-10 py-10">

<section className="text-center mb-12">

<h2 className="text-4xl font-bold mb-4">Знайди роботу своєї мрії</h2>

<p className="text-gray-600 mb-6">Переглядай тисячі вакансій у різних галузях.</p>

<div className="max-w-xl mx-auto flex gap-3">

<Input

value={searchTerm}

onChange={(e) => setSearchTerm(e.target.value)}

placeholder="Пошук за посадою або компанією"

className="flex-1"

/>

<Button onClick={handleSearch}>

<Search className="mr-2 h-4 w-4" />Пошук

</Button>

</div>

</section>

<section className="mb-12">

<h2 className="text-2xl font-semibold mb-4">Категорії</h2>

<div className="grid grid-cols-2 md:grid-cols-4 gap-4">

{categories.map((category) => (

<CategoryCard key={category.category\_id} category={category} />

))}

</div>

</section>

<section>

<h2 className="text-2xl font-semibold mb-4">Пропоновані роботи</h2>

{suggestedJobs.length === 0 ? (

<p className="text-gray-500">Наразі немає пропонованих вакансій.</p>

) : (

<>

<div className="grid gap-4 md:grid-cols-2">

{suggestedJobs.map((job) => (

<Link to={`/job/${job.job\_id}`} key={job.job\_id}>

<Card className="hover:shadow-md transition-shadow duration-200 cursor-pointer">

<CardContent className="p-4">

<h3 className="text-xl font-bold">{job.title}</h3>

<p className="text-gray-600">{job.company\_name}</p>

<div className="flex items-center gap-2 text-sm mt-2 text-gray-500">

<MapPin className="h-4 w-4" /> {job.location}

</div>

<span className="mt-2 inline-block bg-blue-100 text-blue-600 px-3 py-1 rounded-full text-xs">

{employmentTypeMap[job.employment\_type]}

</span>

</CardContent>

</Card>

</Link>

))}

</div>

<div className="text-center mt-6">

<button

onClick={handleSeeAllClick}

className="inline-block bg-blue-600 text-white px-6 py-2 rounded-full hover:bg-blue-700 transition"

>

Переглянути всі вакансії

</button>

</div>

</>

)}

</section>

</div>

);

}

Сторінка профілю(/pages/Profile.jsx)

import React from "react";

import { getUser } from "../tools/auth";

import JobSeekerProfile from "./jobseeker/JobSeekerProfile";

import EmployerProfile from "./employer/EmoloyerProfile";

import AdminPage from "./AdminPage";

export default function Profile() {

const user = getUser();

if (!user) return <p>Користувач не авторизований</p>;

return (

<>

{user.role === "job\_seeker" ? (

<JobSeekerProfile />

) : user.role === "employer" ? (

<EmployerProfile />

) : user.role === "admin" ? (

<AdminPage />

) : (

<p>Немає профілю для цієї ролі</p>

)}

</>

);

}

Сторінка профілю пошукача роботи(/pages/JobSeekerProfile.jsx)

import React, { useEffect, useState } from "react";

import { Link } from "react-router-dom";

import Button from "../../components/ui/Button";

import API from "../../api/api";

import { getUser } from "../../tools/auth";

import { getJobSeekerProfile } from "../../api/profile";

export default function JobSeekerProfile() {

const user = getUser();

const [resumes, setResumes] = useState([]);

const [profile, setProfile] = useState(null);

useEffect(() => {

const fetchData = async () => {

try {

const [resumesRes, profileRes] = await Promise.all([

API.get("/resumes"),

getJobSeekerProfile(),

]);

setResumes(resumesRes.data.resumes);

setProfile(profileRes);

} catch (err) {

console.error("Error loading data:", err);

}

};

fetchData();

}, []);

return (

<div className="min-h-screen bg-gray-50 px-6 py-10">

<div className="bg-white p-6 rounded-lg shadow max-w-3xl mx-auto">

<h2 className="text-2xl font-semibold mb-6 text-center">Профіль користувача</h2>

<div className="mb-6">

<p><strong>Імʼя:</strong> {user.first\_name} {user.last\_name}</p>

<p><strong>Email:</strong> {user.email}</p>

</div>

{profile ? (

<div className="mb-8 space-y-2">

<h3 className="text-xl font-medium mb-3">Профіль пошукача</h3>

<p><strong>Місто:</strong> {profile.city}</p>

<p><strong>Категорія 1:</strong> {profile.prefered\_category\_1\_name}</p>

<p><strong>Категорія 2:</strong> {profile.prefered\_category\_2\_name}</p>

<p><strong>Навички:</strong> {profile.skills}</p>

<p><strong>Досвід (років):</strong> {profile.experience}</p>

<p><strong>Опис досвіду:</strong> {profile.experience\_text}</p>

<p><strong>Освіта:</strong> {profile.education}</p>

<p><strong>Очікування:</strong> {profile.expectations}</p>

<p><strong>Портфоліо:</strong> <a href={profile.portfolio} target="\_blank" className="text-blue-500 underline">{profile.portfolio}</a></p>

<Link to="/profile/edit">

<Button className="mt-4">Редагувати профіль</Button>

</Link>

</div>

) : (

<p className="text-gray-500 mb-6">Інформація профілю не знайдена.</p>

)}

<hr/>

<div className="mt-8">

<h3 className="text-xl font-medium mb-3">Ваші резюме</h3>

<Link to="/resume">

<Button className="mb-4">Додати резюме</Button>

</Link>

{resumes.length === 0 ? (

<p className="text-gray-500 text-sm">Немає резюме.</p>

) : (

<ul className="space-y-4">

{resumes.map((resume) => (

<li

key={resume.resume\_id}

className="bg-gray-100 p-4 rounded-lg flex justify-between items-center"

>

<span>{resume.title || `Резюме #${resume.resume\_id}`}</span>

<div className="space-x-2">

<a

href={`/resume/${resume.resume\_id}`}

rel="noopener noreferrer"

>

<Button size="sm">Переглянути</Button>

</a>

<Button

variant="destructive"

size="sm"

onClick={async () => {

if (window.confirm("Ви впевнені, що хочете видалити це резюме?")) {

try {

await API.delete(`/resumes/${resume.resume\_id}`);

setResumes((prev) => prev.filter((r) => r.resume\_id !== resume.resume\_id));

} catch (err) {

console.error("Помилка при видаленні резюме", err);

}

}

}}

>

Видалити

</Button>

</div>

</li>

))}

</ul>

)}

</div>

</div>

</div>

);

}

Сторінка редагування аккаунта(/pages/JobSeekerProfileEdit.jsx)

import React, { useState, useEffect } from "react";

import { useNavigate } from "react-router-dom";

import Input from "../../components/ui/Input";

import TextArea from "../../components/ui/TextArea";

import Button from "../../components/ui/Button";

import { getCategories } from "../../api/category";

import { getJobSeekerProfile, upsertJobSeekerProfile } from "../../api/profile";

export default function JobSeekerProfileEdit() {

const [profileData, setProfileData] = useState({

prefered\_category\_1: "",

prefered\_category\_2: "",

skills: "",

experience: "",

city: "",

experience\_text: "",

education: "",

expectations: "",

portfolio: "",

});

const [categories, setCategories] = useState([]);

const [loading, setLoading] = useState(true);

const navigate = useNavigate();

useEffect(() => {

const fetchData = async () => {

try {

const profileResponse = await getJobSeekerProfile();

const categoriesResponse = await getCategories();

setProfileData({

...profileResponse,

prefered\_category\_1: profileResponse.prefered\_category\_1,

prefered\_category\_2: profileResponse.prefered\_category\_2,

});

setCategories(categoriesResponse);

setLoading(false);

} catch (error) {

console.error("Error fetching profile or categories:", error);

setLoading(false);

}

};

fetchData();

}, []);

const handleInputChange = (e) => {

const { name, value } = e.target;

setProfileData((prev) => ({

...prev,

[name]: value,

}));

};

const handleSubmit = async (e) => {

e.preventDefault();

try {

await upsertJobSeekerProfile(profileData);

navigate("/profile");

} catch (error) {

console.error("Error updating profile:", error);

}

};

if (loading) {

return <p>Завантаження...</p>;

}

return (

<div className="min-h-screen bg-gray-50 px-6 py-10">

<form onSubmit={handleSubmit} className="bg-white p-6 rounded-lg shadow max-w-2xl mx-auto">

<h2 className="text-2xl font-semibold mb-4 text-center">Редагування профілю пошукача</h2>

{/\* Categories \*/}

<div className="mb-4">

<label htmlFor="prefered\_category\_1" className="block text-sm font-medium text-gray-700">Бажана категорія 1</label>

<select

id="prefered\_category\_1"

name="prefered\_category\_1"

value={profileData.prefered\_category\_1}

onChange={handleInputChange}

className="w-full border p-2 rounded-md"

>

<option value="">Оберіть категорію</option>

{categories.map((category) => (

<option key={category.category\_id} value={category.category\_id}>

{category.name}

</option>

))}

</select>

</div>

<div className="mb-4">

<label htmlFor="prefered\_category\_2" className="block text-sm font-medium text-gray-700">Бажана категорія 2</label>

<select

id="prefered\_category\_2"

name="prefered\_category\_2"

value={profileData.prefered\_category\_2}

onChange={handleInputChange}

className="w-full border p-2 rounded-md"

>

<option value="">Оберіть категорію</option>

{categories.map((category) => (

<option key={category.category\_id} value={category.category\_id}>

{category.name}

</option>

))}

</select>

</div>

{/\* Inputs \*/}

<Input

label="Навички"

placeholder="Введіть ваші навички"

name="skills"

value={profileData.skills}

onChange={handleInputChange}

className="mb-4"

/>

<Input

label="Досвід (у роках)"

placeholder="Введіть ваш досвід"

name="experience"

type="number"

min="0"

value={profileData.experience}

onChange={handleInputChange}

className="mb-4"

/>

<Input

label="Місто"

placeholder="Введіть ваше місто"

name="city"

value={profileData.city}

onChange={handleInputChange}

className="mb-4"

/>

{/\* TextAreas \*/}

<TextArea

label="Опис досвіду"

placeholder="Опишіть ваш досвід"

name="experience\_text"

value={profileData.experience\_text}

onChange={handleInputChange}

className="mb-4"

/>

<TextArea

label="Освіта"

placeholder="Вкажіть вашу освіту"

name="education"

value={profileData.education}

onChange={handleInputChange}

className="mb-4"

/>

<TextArea

label="Очікування"

placeholder="Опишіть ваші кар'єрні очікування"

name="expectations"

value={profileData.expectations}

onChange={handleInputChange}

className="mb-4"

/>

<Input

label="Посилання на портфоліо"

placeholder="Введіть посилання на портфоліо"

name="portfolio"

value={profileData.portfolio}

onChange={handleInputChange}

className="mb-4"

/>

<Button type="submit" className="w-full">

Оновити профіль

</Button>

</form>

</div>

);

}

Сторінка профілю роботодавця(/pages/EmployerProfile.jsx)

import React, { useState, useEffect, useRef } from "react";

import Input from "../../components/ui/Input";

import Button from "../../components/ui/Button";

import TextArea from "../../components/ui/TextArea";

import { getEmployerProfile, upsertEmployerProfile } from "../../api/profile";

import Logo from "../../components/Logo";

export default function EmployerProfile() {

const [form, setForm] = useState({

first\_name: "",

last\_name: "",

email: "",

company\_name: "",

logo: "",

company\_description: "",

});

const [error, setError] = useState("");

const [success, setSuccess] = useState("");

const [preview, setPreview] = useState("");

const fileInputRef = useRef(null);

useEffect(() => {

const fetchProfile = async () => {

try {

const data = await getEmployerProfile();

setForm({

first\_name: data.first\_name || "",

last\_name: data.last\_name || "",

email: data.email || "",

company\_name: data.company\_name || "",

logo: data.logo || "",

company\_description: data.company\_description || "",

});

setPreview(data.logo || "");

console.log("Профіль завантажено:", data);

} catch (err) {

setError("Не вдалося завантажити профіль");

}

};

fetchProfile();

}, []);

const handleChange = (e) => {

setForm({ ...form, [e.target.name]: e.target.value });

};

const handlePhotoClick = () => {

fileInputRef.current.click();

};

const resizeImage = (file, maxSize = 250) =>

new Promise((resolve) => {

const img = new window.Image();

const reader = new FileReader();

reader.onload = (e) => {

img.onload = () => {

const canvas = document.createElement("canvas");

const ctx = canvas.getContext("2d");

let width = img.width;

let height = img.height;

if (width > height) {

if (width > maxSize) {

height \*= maxSize / width;

width = maxSize;

}

} else {

if (height > maxSize) {

width \*= maxSize / height;

height = maxSize;

}

}

canvas.width = width;

canvas.height = height;

ctx.drawImage(img, 0, 0, width, height);

resolve(canvas.toDataURL("image/png"));

};

img.src = e.target.result;

};

reader.readAsDataURL(file);

});

const handlePhotoChange = async (e) => {

const file = e.target.files[0];

if (file) {

const resized = await resizeImage(file, 250);

setForm((prev) => ({ ...prev, logo: resized }));

setPreview(resized);

}

};

const isBase64Image = (str) =>

typeof str === "string" && str.startsWith("data:image/");

const handleSubmit = async (e) => {

e.preventDefault();

setError("");

setSuccess("");

let dataToSend = { ...form };

if (dataToSend.logo && !isBase64Image(dataToSend.logo)) {

try {

const response = await fetch(dataToSend.logo);

const blob = await response.blob();

const reader = new FileReader();

dataToSend.logo = await new Promise((resolve) => {

reader.onloadend = () => resolve(reader.result);

reader.readAsDataURL(blob);

});

} catch {

}

}

try {

await upsertEmployerProfile(dataToSend);

setSuccess("Профіль оновлено!");

} catch (err) {

setError(err.response?.data?.message || "Не вдалося оновити профіль");

}

};

return (

<div className="min-h-screen bg-gray-50 px-6 py-10">

<Logo />

<form

onSubmit={handleSubmit}

className="bg-white p-6 rounded-lg shadow max-w-md mx-auto mt-10"

>

<h2 className="text-2xl font-semibold mb-4 text-center">

Профіль роботодавця

</h2>

{error && <p className="text-red-500 text-sm mb-3">{error}</p>}

{success && <p className="text-green-600 text-sm mb-3">{success}</p>}

<div className="flex flex-col items-center mb-6">

<div

className="relative group cursor-pointer"

onClick={handlePhotoClick}

title="Змініть фото"

>

<img

src={form.logo || "/default.png"}

alt="Логотип компанії"

className="w-24 h-24 rounded-full object-cover border-2 border-gray-300"

/>

<div className="absolute inset-0 bg-black bg-opacity-40 rounded-full flex items-center justify-center opacity-0 group-hover:opacity-100 transition-opacity">

<span className="text-white text-sm">Змініть фото</span>

</div>

<input

type="file"

accept="image/\*"

className="hidden"

ref={fileInputRef}

onChange={handlePhotoChange}

/>

</div>

</div>

<div className="mb-4 flex gap-4">

<Input

name="first\_name"

placeholder="Ім'я"

value={form.first\_name}

onChange={handleChange}

required

className="flex-1"

/>

<Input

name="last\_name"

placeholder="Прізвище"

value={form.last\_name}

onChange={handleChange}

required

className="flex-1"

/>

</div>

<div className="mb-4">

<Input

name="email"

type="email"

placeholder="Електронна пошта"

value={form.email}

onChange={handleChange}

required

disabled

/>

</div>

<div className="mb-4">

<Input

name="company\_name"

placeholder="Назва компанії"

value={form.company\_name}

onChange={handleChange}

required

/>

</div>

<div className="mb-4">

<TextArea

name="company\_description"

placeholder="Опис компанії"

value={form.company\_description}

onChange={handleChange}

/>

</div>

<Button type="submit" className="w-full">

Зберегти зміни

</Button>

</form>

</div>

);

}

Сторінка створення роботи(/pages/JobCreatePage.jsx)

import React, { useState, useEffect } from "react";

import { useNavigate, useSearchParams } from "react-router-dom";

import Input from "../../components/ui/Input";

import Button from "../../components/ui/Button";

import { createJob, updateJob, getJobById } from "../../api/job";

import { getCategories } from "../../api/category";

import TextArea from "../../components/ui/TextArea";

const initialState = {

title: "",

description: "",

requirements: "",

salary: "",

salaryMax: "",

employment\_type: "full\_time",

location: "",

category\_id: "",

status: "draft",

};

const employmentTypes = [

{ value: "full\_time", label: "Повна зайнятість" },

{ value: "part\_time", label: "Неповна зайнятість" },

{ value: "freelance", label: "Фриланс" },

{ value: "temporary", label: "Тимчасова робота" },

];

const statusOptions = [

{ value: "draft", label: "Чернетка" },

{ value: "open", label: "Відкрита" },

{ value: "closed", label: "Закрита" },

];

export default function JobCreatePage() {

const [form, setForm] = useState(initialState);

const [error, setError] = useState("");

const [loading, setLoading] = useState(false);

const [editing, setEditing] = useState(false);

const [categories, setCategories] = useState([]);

const navigate = useNavigate();

const [searchParams] = useSearchParams();

const edit = searchParams.get("edit");

useEffect(() => {

getCategories()

.then((res) => setCategories(res))

.catch(() => setCategories([]));

}, []);

useEffect(() => {

if (edit) {

setEditing(true);

setLoading(true);

getJobById(edit)

.then((data) => {

setForm({

title: data.title || "",

description: data.description || "",

requirements: data.requirements || "",

salary: data.salary || "",

salaryMax: data.salaryMax || "",

employment\_type: data.employment\_type || "full\_time",

location: data.location || "",

category\_id: data.category\_id || "",

status: data.status || "draft",

});

})

.catch(() => setError("Не вдалося завантажити вакансію"))

.finally(() => setLoading(false));

}

}, [edit]);

const handleChange = (e) => {

const { name, value } = e.target;

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

};

const handleSubmit = async (e) => {

e.preventDefault();

setError("");

setLoading(true);

try {

if (editing) {

await updateJob(edit, form);

alert("Вакансію оновлено!");

navigate(`/job/${edit}`);

} else {

await createJob(form);

alert("Вакансію створено!");

navigate("/");

}

} catch (err) {

setError(err.response?.data?.message || "Помилка при збереженні вакансії");

} finally {

setLoading(false);

}

};

return (

<div className="max-w-xl mx-auto py-8">

<h1 className="text-2xl font-bold mb-6">{editing ? "Редагувати вакансію" : "Створити вакансію"}</h1>

<form onSubmit={handleSubmit} className="space-y-4 bg-white p-6 rounded shadow">

<Input

label="Назва вакансії"

name="title"

value={form.title}

onChange={handleChange}

required

/>

<div>

<label className="block mb-1 font-medium">Категорія</label>

<select

name="category\_id"

value={form.category\_id}

onChange={handleChange}

className="w-full border rounded px-3 py-2"

required

>

<option value="">Оберіть категорію</option>

{categories.map((cat) => (

<option key={cat.category\_id} value={cat.category\_id}>

{cat.name}

</option>

))}

</select>

</div>

<TextArea

label="Опис"

name="description"

value={form.description}

onChange={handleChange}

as="textarea"

rows={6}

required

/>

<TextArea

label="Вимоги"

name="requirements"

value={form.requirements}

onChange={handleChange}

as="textarea"

rows={4}

/>

<div className="flex gap-2">

<Input

label="Зарплата від"

name="salary"

type="number"

value={form.salary}

onChange={handleChange}

required

/>

<Input

label="до"

name="salaryMax"

type="number"

value={form.salaryMax}

onChange={handleChange}

/>

</div>

<div>

<label className="block mb-1 font-medium">Тип зайнятості</label>

<select

name="employment\_type"

value={form.employment\_type}

onChange={handleChange}

className="w-full border rounded px-3 py-2"

>

{employmentTypes.map((t) => (

<option key={t.value} value={t.value}>{t.label}</option>

))}

</select>

</div>

<Input

label="Локація"

name="location"

value={form.location}

onChange={handleChange}

required

/>

<div>

<label className="block mb-1 font-medium">Статус вакансії</label>

<select

name="status"

value={form.status}

onChange={handleChange}

className="w-full border rounded px-3 py-2"

required

>

{statusOptions.map((s) => (

<option key={s.value} value={s.value}>{s.label}</option>

))}

</select>

</div>

{error && <div className="text-red-600">{error}</div>}

<Button type="submit" disabled={loading}>

{editing ? "Оновити" : "Створити"}

</Button>

</form>

</div>

);

}

Головна сторінка роботодавця(/pages/MainEmployerPage.jsx)

import React, { useEffect, useState } from "react";

import Input from "../../components/ui/Input";

import Button from "../../components/ui/Button";

import JobCard from "../../components/ui/JobCard";

import { getEmployerJobs } from "../../api/job";

import { EyeOff } from "lucide-react";

import { useNavigate } from "react-router-dom";

export default function MainEmployerPage() {

const [jobs, setJobs] = useState([]);

const [filteredJobs, setFilteredJobs] = useState([]);

const [searchTerm, setSearchTerm] = useState("");

const [showOnlyWithUnviewed, setShowOnlyWithUnviewed] = useState(false);

const navigate = useNavigate();

useEffect(() => {

fetchJobs();

}, []);

useEffect(() => {

applyFilters();

}, [jobs, searchTerm, showOnlyWithUnviewed]);

const fetchJobs = async () => {

try {

const res = await getEmployerJobs();

setJobs(res);

} catch (err) {

setJobs([]);

}

};

const applyFilters = () => {

let filtered = [...jobs];

if (searchTerm) {

const term = searchTerm.toLowerCase();

filtered = filtered.filter(

(job) =>

job.title.toLowerCase().includes(term) ||

job.description.toLowerCase().includes(term)

);

}

if (showOnlyWithUnviewed) {

filtered = filtered.filter(

(job) => Number(job.pending\_applications) > 0

);

}

setFilteredJobs(filtered);

};

return (

<div className="min-h-screen bg-gray-50 px-6 py-10">

<section className="mb-6">

<div className="max-w-lg mx-auto flex gap-3">

<Input

value={searchTerm}

onChange={(e) => setSearchTerm(e.target.value)}

placeholder="Пошук вакансії..."

className="flex-1"

/>

<Button

type="button"

variant="outline"

onClick={() => setShowOnlyWithUnviewed((v) => !v)}

title="Показати лише вакансії з новими заявами"

className={

showOnlyWithUnviewed

? "bg-blue-600 text-white border-blue-600 !border-2"

: "!bg-white !text-blue-600 !border-blue-600 !border-2 hover:!bg-blue-50"

}

>

Є нові заявки

</Button>

<Button

type="button"

variant="primary"

onClick={() => navigate("/jobs/add")}

title="Додати вакансію"

>

Додати вакансію

</Button>

</div>

</section>

<div className="max-w-screen-xl mx-auto grid grid-cols-1 md:grid-cols-3 gap-6">

<div className="col-span-2">

{filteredJobs.length === 0 ? (

<p>Нічого не знайдено.</p>

) : (

filteredJobs.map((job) => (

<JobCard

job={{

...job,

status: job.status || "draft",

pending\_applications: job.pending\_applications

}}

key={job.job\_id}

/>

))

)}

</div>

<div className="col-span-1 bg-white p-6 rounded-lg shadow-md">

<h3 className="text-xl font-semibold mb-4">Ваші вакансії</h3>

<p className="text-gray-600">

Тут ви можете переглядати та керувати своїми опублікованими вакансіями.

</p>

</div>

</div>

</div>

);

}

Сторінка пошуку роботи(/pages/JobSearchPage.jsx)

import React, { useEffect, useState } from "react";

import { useSearchParams, useNavigate, useParams } from "react-router-dom";

import Input from "../components/ui/Input";

import Button from "../components/ui/Button";

import { Search } from "lucide-react";

import JobCard from "../components/ui/JobCard";

import { getJobsByCategory } from "../api/job";

import { getCategories } from "../api/category";

const salaryOptions = [0, 10000, 2000, 30000, 50000, 70000, 90000, 100000, 130000, 150000, 200000];

const typeMap = {

full\_time: "Повна зайнятість",

part\_time: "Неповна зайнятість",

freelance: "Фриланс",

temporary: "Тимчасова робота",

};

const JobSearchPage = () => {

const [searchParams] = useSearchParams();

const { categoryId } = useParams();

const navigate = useNavigate();

const [categories, setCategories] = useState([]);

const [selectedCategory, setSelectedCategory] = useState(categoryId || searchParams.get("category\_id") || "");

const [searchTerm, setSearchTerm] = useState(searchParams.get("search") || "");

const [selectedFilters, setSelectedFilters] = useState({

jobType: searchParams.get("jobType")?.split(",") || [],

salaryRange: [

parseInt(searchParams.get("minSalary")) || 0,

parseInt(searchParams.get("maxSalary")) || 200000,

],

});

const [allJobs, setAllJobs] = useState([]);

const [filteredJobs, setFilteredJobs] = useState([]);

const fetchJobs = async () => {

try {

const jobs = await getJobsByCategory(selectedCategory || 0);

setAllJobs(jobs);

} catch (err) {

console.error("Помилка при завантаженні вакансій:", err);

}

};

const fetchCategories = async () => {

try {

const res = await getCategories();

setCategories(res);

} catch (error) {

console.error("Помилка при завантаженні категорій:", error);

}

};

const updateURLParams = () => {

const params = {};

if (searchTerm) params.search = searchTerm;

if (selectedFilters.jobType.length > 0) params.jobType = selectedFilters.jobType.join(",");

if (selectedFilters.salaryRange[0] !== 0) params.minSalary = selectedFilters.salaryRange[0];

if (selectedFilters.salaryRange[1] !== 200000) params.maxSalary = selectedFilters.salaryRange[1];

if (selectedCategory) params.category\_id = selectedCategory;

navigate(`/jobs?${new URLSearchParams(params).toString()}`, { replace: true });

};

const toggleFilter = (filterCategory, value) => {

setSelectedFilters((prevState) => {

const updated = { ...prevState };

const exists = updated[filterCategory].includes(value);

updated[filterCategory] = exists

? updated[filterCategory].filter((item) => item !== value)

: [...updated[filterCategory], value];

return updated;

});

};

const handleSalaryChange = (which, value) => {

setSelectedFilters((prev) => {

const min = which === "min" ? value : prev.salaryRange[0];

const max = which === "max" ? value : prev.salaryRange[1];

return {

...prev,

salaryRange: [Math.min(min, max), Math.max(min, max)],

};

});

};

const handleCategoryChange = (e) => {

setSelectedCategory(e.target.value);

};

const applyFilters = () => {

if (!Array.isArray(allJobs)) {

setFilteredJobs([]);

return;

}

let jobs = [...allJobs];

if (selectedFilters.jobType.length > 0) {

jobs = jobs.filter((job) => selectedFilters.jobType.includes(job.employment\_type));

}

const [minSalary, maxSalary] = selectedFilters.salaryRange;

jobs = jobs.filter(

(job) =>

job.salary >= minSalary &&

(job.salaryMax || job.salary) <= maxSalary

);

if (searchTerm) {

const term = searchTerm.toLowerCase();

jobs = jobs.filter(

(job) =>

job.title.toLowerCase().includes(term) ||

job.description.toLowerCase().includes(term)

);

}

setFilteredJobs(jobs);

};

useEffect(() => {

fetchCategories();

}, []);

useEffect(() => {

fetchJobs();

}, [selectedCategory]);

useEffect(() => {

applyFilters();

}, [allJobs, selectedFilters, searchTerm]);

useEffect(() => {

updateURLParams();

}, [selectedFilters, selectedCategory, searchTerm]);

return (

<div className="min-h-screen bg-gray-50 px-6 py-10">

<section className="mb-6">

<div className="max-w-lg mx-auto flex gap-3">

<Input

value={searchTerm}

onChange={(e) => setSearchTerm(e.target.value)}

placeholder="Пошук вакансії..."

className="flex-1"

/>

<Button>

<Search className="mr-2 h-4 w-4" />

Пошук

</Button>

</div>

</section>

<div className="max-w-screen-xl mx-auto grid grid-cols-1 md:grid-cols-4 gap-6">

<div className="col-span-3">

{filteredJobs.length === 0 ? (

<p>Нічого не знайдено.</p>

) : (

filteredJobs.map((job) => (

<JobCard job={job} key={job.job\_id} />

))

)}

</div>

<div className="col-span-1 bg-white p-6 rounded-lg shadow-md">

<h3 className="text-xl font-semibold mb-4">Фільтри</h3>

<div className="mb-4">

<h4 className="text-lg font-medium mb-2">Категорія</h4>

<select

value={selectedCategory}

onChange={handleCategoryChange}

className="w-full border border-gray-300 rounded px-2 py-1">

<option value="">Усі категорії</option>

{categories.map((cat) => (

<option key={cat.category\_id} value={cat.category\_id}>

{cat.name}

</option>

))}

</select>

</div>

<div className="mb-4">

<h4 className="text-lg font-medium">Тип роботи</h4>

<div className="space-y-2">

{Object.keys(typeMap).map((type) => (

<label key={type} className="flex items-center space-x-2">

<input

type="checkbox"

checked={selectedFilters.jobType.includes(type)}

onChange={() => toggleFilter("jobType", type)}/>

<span>{typeMap[type]}</span>

</label>

))}

</div>

</div>

<div className="mb-4">

<h4 className="text-lg font-medium mb-2">Зарплата</h4>

<div className="flex gap-3">

<div className="flex-1">

<label className="block text-sm mb-1">Від</label>

<select

className="w-full border border-gray-300 rounded px-2 py-1"

value={selectedFilters.salaryRange[0]}

onChange={(e) => handleSalaryChange("min", parseInt(e.target.value))}

>

{salaryOptions.map((salary) => (

<option key={salary} value={salary}>

₴{salary.toLocaleString()}

</option>

))}

</select>

</div>

<div className="flex-1">

<label className="block text-sm mb-1">До</label>

<select

className="w-full border border-gray-300 rounded px-2 py-1"

value={selectedFilters.salaryRange[1]}

onChange={(e) => handleSalaryChange("max", parseInt(e.target.value))}

>

{salaryOptions.map((salary) => (

<option key={salary} value={salary}>

₴{salary.toLocaleString()}

</option>

))}

</select>

</div>

</div>

</div>

</div>

</div>

</div>

);

};

export default JobSearchPage;

Сторінка перегляду роботи(/pages/JobViewPage.jsx)

import React, { useEffect, useState, useMemo } from "react";

import { useParams, Link, useNavigate } from "react-router-dom";

import { getJobById } from "../api/job";

import { applyForJob, getJobApplications, manageApplication, getJobSeekerApplications } from "../api/application";

import { ArrowLeft } from "lucide-react";

import { getUser } from "../tools/auth";

import { blockJob, unblockJob } from "../api/admin";

const employmentTypeMap = {

full\_time: "Повна зайнятість",

part\_time: "Неповна зайнятість",

freelance: "Фриланс",

temporary: "Тимчасова робота",

};

const JobViewPage = () => {

const { jobId } = useParams();

const navigate = useNavigate();

const [job, setJob] = useState(null);

const [loading, setLoading] = useState(true);

const [applying, setApplying] = useState(false);

const [applicationStatus, setApplicationStatus] = useState(null);

const [applications, setApplications] = useState([]);

const [appLoading, setAppLoading] = useState(false);

const [blockLoading, setBlockLoading] = useState(false);

const user = getUser();

const fetchJob = async () => {

try {

const data = await getJobById(jobId);

if (data.status === "draft" && (!user || user.role !== "employer")) {

navigate("/jobs");

} else {

setJob(data);

}

} catch (error) {

console.error("Не вдалося завантажити вакансію:", error);

navigate("/jobs");

} finally {

setLoading(false);

}

};

const fetchApplications = async () => {

setAppLoading(true);

try {

const data = await getJobApplications(jobId);

setApplications(data.applications || []);

} catch (error) {

setApplications([]);

} finally {

setAppLoading(false);

}

};

const checkIfApplied = async () => {

try {

const applications = await getJobSeekerApplications();

const application = applications.find((a) => String(a.job\_id) === String(jobId));

if (application) {

setApplicationStatus(application.status);

}

} catch (error) {

console.error("Не вдалося перевірити заявки:", error);

}

};

const handleApply = async () => {

if (!window.confirm("Ви впевнені, що хочете залишити заявку?")) return;

try {

setApplying(true);

await applyForJob({ job\_id: job.job\_id });

alert("Заявка успішно відправлена!");

setApplicationStatus("pending");

} catch (error) {

console.error("Помилка при подачі заявки:", error);

alert("Не вдалося залишити заявку.");

} finally {

setApplying(false);

}

};

const handleManage = async (application\_id, status) => {

if (!window.confirm(`Ви впевнені, що хочете ${status === "accepted" ? "прийняти" : "відхилити"} цю заявку?`)) return;

try {

await manageApplication({ application\_id, status });

fetchApplications();

} catch (error) {

alert("Не вдалося оновити статус заявки.");

}

};

const handleBlockToggle = async () => {

if (!job) return;

setBlockLoading(true);

try {

if (job.blocked) {

await unblockJob(job.job\_id);

setJob({ ...job, blocked: false });

} else {

await blockJob(job.job\_id);

setJob({ ...job, blocked: true });

}

} catch (e) {

alert("Не вдалося змінити статус блокування.");

}

setBlockLoading(false);

};

useEffect(() => {

fetchJob();

}, [jobId]);

useEffect(() => {

if (user && user.role === "job\_seeker") {

checkIfApplied();

}

}, [user, jobId]);

const isEmployerOfThisJob = useMemo(() => {

return user && user.role === "employer" && job && user.user\_id === job.employer\_id;

}, [user, job]);

useEffect(() => {

if (isEmployerOfThisJob) {

fetchApplications();

}

}, [isEmployerOfThisJob]);

if (loading) {

return <div className="min-h-screen flex items-center justify-center">Завантаження...</div>;

}

if (!job) {

return <p className="text-center mt-10 text-gray-500">Вакансія не знайдена.</p>;

}

return (

<div className="max-w-4xl mx-auto px-6 py-10 bg-gray-50 min-h-screen">

<button

onClick={() => navigate(-1)}

className="flex items-center text-blue-600 hover:underline mb-6"

>

<ArrowLeft className="mr-2 w-4 h-4" />

Назад до списку

</button>

<h1 className="text-3xl font-bold mb-2">{job.title}</h1>

<p className="text-gray-600 mb-4">

{employmentTypeMap[job.employment\_type]} • {job.location || "Локація не вказана"}

</p>

<div className="bg-white rounded-lg shadow p-6">

<h2 className="text-xl font-semibold mb-2">Опис</h2>

<p className="mb-4 whitespace-pre-line">{job.description || "Немає опису."}</p>

<h2 className="text-xl font-semibold mb-2">Вимоги</h2>

<p className="mb-4 whitespace-pre-line">{job.requirements || "Немає вимог."}</p>

<h2 className="text-xl font-semibold mb-2">Зарплата</h2>

<p className="mb-4 text-green-700">

₴{job.salary.toLocaleString()}

{job.salaryMax && job.salaryMax !== job.salary ? ` – ₴${job.salaryMax.toLocaleString()}` : ""}

</p>

<p className="text-sm text-gray-500">

Опубліковано: {new Date(job.posting\_date).toLocaleDateString()}

</p>

<p className="text-sm text-gray-500">Статус: {job.status}</p>

{/\* --- КНОПКА ДЛЯ АДМІНА --- \*/}

{user?.role === "admin" && (

<button

onClick={handleBlockToggle}

disabled={blockLoading}

className={`mt-4 px-5 py-2 rounded-lg shadow text-white ${

job.blocked

? "bg-green-500 hover:bg-green-600"

: "bg-red-500 hover:bg-red-600"

}`}

>

{blockLoading

? "Зміна..."

: job.blocked

? "Розблокувати"

: "Заблокувати"}

</button>

)}

{job.blocked ? (

<div className="mt-6 px-5 py-2 rounded-lg bg-red-100 text-red-700 font-semibold text-center">

Вакансія заблокована

</div>

) : (

<>

{user ? (

user.role === "employer" && user.user\_id === job.employer\_id ? (

<button

onClick={() => navigate(`/jobs/add?edit=${job.job\_id}`)}

className="mt-6 px-5 py-2 bg-blue-500 hover:bg-blue-600 text-white rounded-lg shadow"

>

Редагувати вакансію

</button>

) : applicationStatus ? (

<div className={`mt-6 px-5 py-2 rounded-lg inline-block

${applicationStatus === "pending" ? "bg-yellow-100 text-yellow-700"

: applicationStatus === "accepted" ? "bg-green-100 text-green-700"

: "bg-red-100 text-red-700"}`}>

{applicationStatus === "pending" && "Ви вже подали заявку. Очікуйте результатів"}

{applicationStatus === "accepted" && "Вашу заявку одобрено!"}

{applicationStatus === "rejected" && "На жаль, вам відмовили."}

</div>

) : user.role === "job\_seeker" && (

<button

onClick={handleApply}

disabled={applying || job.status === "closed"}

className={`mt-6 px-5 py-2 rounded-lg shadow text-white ${

job.status === "closed"

? "bg-gray-400 cursor-not-allowed"

: "bg-blue-600 hover:bg-blue-700"

}`}

>

{applying ? "Відправка..." : "Залишити заявку"}

</button>

)

) : (

<button

onClick={() => navigate("/login")}

className="mt-6 px-5 py-2 bg-yellow-500 hover:bg-yellow-600 text-white rounded-lg shadow"

>

Авторизуйтесь, щоб подати заявку

</button>

)}

</>

)}

</div>

{/\* Заявки для роботодавця \*/}

{isEmployerOfThisJob && (

<div

className={`bg-white rounded-lg shadow p-6 mt-8 ${

job.blocked ? "pointer-events-none select-none" : ""

}`}

>

<h2 className="text-xl font-bold mb-4">Заявки на вакансію</h2>

{job.blocked ? (

<div className="text-red-500 font-semibold mb-4">

Вакансія заблокована. Перегляд заявок недоступний.

</div>

) : appLoading ? (

<div>Завантаження заявок...</div>

) : applications.length === 0 ? (

<div className="text-gray-500">Заявок ще немає.</div>

) : (

<div className="space-y-6">

{applications.map((app) => (

<div key={app.application\_id} className="border-b pb-4 mb-4">

<div className="flex flex-col md:flex-row md:items-center md:justify-between">

<div>

<div className="font-semibold text-lg">

{app.user.first\_name} {app.user.last\_name}

</div>

<div className="text-gray-600 text-sm mb-1">

Email: {app.user.email}

{app.user.phone && <> | Телефон: {app.user.phone}</>}

</div>

{app.user.profile && (

<div className="text-gray-600 text-sm mb-1">

{app.user.profile.city && <>Місто: {app.user.profile.city} | </>}

{app.user.profile.skills && <>Навички: {app.user.profile.skills} | </>}

{app.user.profile.experience && <>Досвід: {app.user.profile.experience} років | </>}

{app.user.profile.education && <>Освіта: {app.user.profile.education} | </>}

{app.user.profile.expectations && <>Очікування: {app.user.profile.expectations} | </>}

</div>

)}

<div className="flex flex-wrap gap-2 mt-2">

{app.user.resumes && app.user.resumes.length > 0 ? (

app.user.resumes.map((resume) => (

<a

key={resume.resume\_id}

href={`/resume/${resume.resume\_id}`}

target="\_blank"

rel="noopener noreferrer"

className="text-blue-600 underline hover:text-blue-800"

>

{resume.title || `Резюме #${resume.resume\_id}`}

</a>

))

) : (

<span className="text-gray-400">Резюме відсутні</span>

)}

</div>

</div>

<div className="mt-4 md:mt-0 flex flex-col gap-2 min-w-[160px]">

<span

className={`px-3 py-1 rounded-full text-xs font-semibold text-center ${

app.status === "pending"

? "bg-yellow-100 text-yellow-700"

: app.status === "accepted"

? "bg-green-100 text-green-700"

: "bg-red-100 text-red-700"

}`}

>

{app.status === "pending"

? "Очікує"

: app.status === "accepted"

? "Прийнято"

: "Відхилено"}

</span>

{app.status === "pending" && (

<div className="flex gap-2 mt-2">

<button

onClick={() => handleManage(app.application\_id, "accepted")}

className="px-3 py-1 bg-green-500 hover:bg-green-600 text-white rounded"

disabled={job.blocked}

>

Прийняти

</button>

<button

onClick={() => handleManage(app.application\_id, "rejected")}

className="px-3 py-1 bg-red-500 hover:bg-red-600 text-white rounded"

disabled={job.blocked}

>

Відхилити

</button>

</div>

)}

</div>

</div>

<div className="text-xs text-gray-400 mt-2">

Подано: {new Date(app.application\_date).toLocaleString()}

</div>

</div>

))}

</div>

)}

</div>

)}

</div>

);

};

export default JobViewPage;

Сторінка резюме(/pages/ResumePage.jsx)

import React, { useState } from "react";

import { useNavigate } from "react-router-dom";

import Button from "../../components/ui/Button";

import { createResume } from "../../api/resume";

import { saveAs } from "file-saver";

export default function ResumePage() {

const navigate = useNavigate();

const [pdfFile, setPdfFile] = useState(null);

const [pdfUrl, setPdfUrl] = useState(null);

const handlePdfChange = (e) => {

const file = e.target.files[0];

setPdfFile(file);

const reader = new FileReader();

reader.onloadend = () => {

setPdfUrl(reader.result);

};

if (file) reader.readAsDataURL(file);

};

const handleSubmit = async (e) => {

e.preventDefault();

if (!pdfFile) {

alert("Please select a PDF file.");

return;

}

const formData = new FormData();

formData.append("resumeFile", pdfFile);

try {

await createResume(formData);

navigate("/profile");

} catch (error) {

console.error("Error uploading resume:", error);

}

};

return (

<div className="min-h-screen bg-gray-50 px-6 py-10">

<form

onSubmit={handleSubmit}

className="bg-white p-6 rounded-lg shadow max-w-lg mx-auto"

>

<h2 className="text-2xl font-semibold mb-4 text-center">Upload Resume</h2>

{/\* PDF File Upload \*/}

<div className="mb-4">

<input

type="file"

accept="application/pdf"

onChange={handlePdfChange}

className="w-full border p-2 rounded-md"

/>

</div>

{/\* PDF Preview \*/}

{pdfUrl && (

<div className="mb-4">

<h3 className="text-lg font-semibold mb-2">PDF Preview</h3>

<iframe

title="pdf"

src={pdfUrl}

width="100%"

height="400"

className="border rounded-md"

/>

</div>

)}

<Button type="submit" className="w-full">

Upload Resume

</Button>

</form>

</div>

);

}

Сторінка перегляду резюме(/pages/ResumeReadPage.jsx)

import React, { useEffect, useState } from "react";

import { useParams, useNavigate } from "react-router-dom";

import { getResume, getResumeFile } from "../../api/resume";

export default function ResumeReadPage() {

const { id } = useParams();

const [resumeInfo, setResumeInfo] = useState(null);

const [loading, setLoading] = useState(true);

const navigate = useNavigate();

useEffect(() => {

const fetchResume = async () => {

try {

const data = await getResume(id);

setResumeInfo(data);

} catch (error) {

console.error("Error fetching resume metadata:", error);

} finally {

setLoading(false);

}

};

fetchResume();

}, [id]);

if (loading) return <div>Loading...</div>;

if (!resumeInfo) return <div>Resume not found.</div>;

const resumeUrl = getResumeFile(id);

return (

<div className="min-h-screen bg-gray-50 px-6 py-10">

<div className="bg-white p-6 rounded-lg shadow max-w-4xl mx-auto">

<h2 className="text-2xl font-semibold mb-4 text-center">Resume Preview</h2>

<iframe

src={resumeUrl}

width="100%" height="600"

className="border rounded-md"

title="Resume PDF" />

<div className="flex justify-between mt-4">

<button

onClick={() => navigate("/profile")}

className="text-gray-600 hover:underline">

Back to Profile

</button>

</div>

</div>

</div>

);}

Сторінка заявок(/pages/JobApplicationsPage.jsx)

import React, { useEffect, useState } from "react";

import { getJobSeekerApplications } from "../api/application";

import { useNavigate } from "react-router-dom";

const employmentTypeMap = {

full\_time: "Full time",

part\_time: "Part time",

freelance: "Freelance",

temporary: "Temporary",

};

const statusMap = {

pending: { text: "Очікування", color: "bg-yellow-100 text-yellow-800" },

accepted: { text: "Прийнято", color: "bg-green-100 text-green-700" },

rejected: { text: "Відхилено", color: "bg-red-100 text-red-700" },

};

const JobApplicationsPage = () => {

const [applications, setApplications] = useState([]);

const [loading, setLoading] = useState(true);

const navigate = useNavigate();

const fetchApplications = async () => {

try {

const data = await getJobSeekerApplications();

setApplications(data);

} catch (error) {

console.error("Не вдалося завантажити заявки:", error);

} finally {

setLoading(false);

}

};

useEffect(() => {

fetchApplications();

}, []);

if (loading) {

return <div className="text-center py-20 text-gray-500">Завантаження...</div>;

}

if (applications.length === 0) {

return (

<div className="text-center py-20 text-gray-600">

<p className="mb-4">Ви ще не подали жодної заявки.</p>

<button

onClick={() => navigate("/jobs")}

className="px-4 py-2 bg-blue-600 hover:bg-blue-700 text-white rounded"

>

Пошук роботи

</button>

</div>

);

}

return (

<div className="max-w-5xl mx-auto px-6 py-10">

<h1 className="text-2xl font-bold mb-6">Мої заявки</h1>

<div className="space-y-4">

{applications.map((app) => {

const statusInfo = statusMap[app.status] || { text: app.status, color: "bg-gray-100 text-gray-600" };

return (

<div

key={app.application\_id}

className="p-4 rounded-lg shadow bg-white border hover:bg-gray-50 transition cursor-pointer"

onClick={() => navigate(`/job/${app.job\_id}`)}

>

<div className="flex justify-between items-center mb-1">

<h2 className="text-lg font-semibold">{app.job\_title}</h2>

<span className={`text-sm px-2 py-1 rounded ${statusInfo.color}`}>

{statusInfo.text}

</span>

</div>

<p className="text-gray-500">

{employmentTypeMap[app.employment\_type]} • {app.location || "Локація не вказана"}

</p>

<p className="text-sm text-gray-400">

Подано: {new Date(app.application\_date).toLocaleDateString()}

</p>

</div>

);

})}

</div>

</div>

);

};

export default JobApplicationsPage;

Серверна частина

Сервіс відправки на пошту (/services/emailService.js)

import nodemailer from 'nodemailer';

const createTransporter = () => {

return nodemailer.createTransport({

service: 'Gmail',

secure: true,

auth: {

user: process.env.EMAIL\_USER,

pass: process.env.EMAIL\_PASS

}

});

};

const sendMail = async ({ to, subject, html }) => {

const transporter = createTransporter();

await transporter.sendMail({

from: `"JobHunting" <${process.env.EMAIL\_USER}>`,

to,

subject,

html

});

};

const sendVerificationEmail = async (to, token) => {

const verificationLink = `${process.env.FRONTEND\_URL}/verify/${token}`;

await sendMail({

to,

subject: 'Підтвердження електронної пошти',

html: `<p>Будь ласка, перейдіть за <a href="${verificationLink}">цим посиланням</a>, щоб підтвердити вашу електронну пошту.</p>`

});

};

export { sendMail, sendVerificationEmail };

Міддлвейр авторизації (/middlewares/authMiddleware.js)

import jwt from 'jsonwebtoken';

const authMiddleware = (req, res, next) => {

const token = req.header('Authorization')?.replace('Bearer ', '');

if (!token) {

return res.status(401).json({ message: 'Access denied. No token provided.' });

}

try {

const decoded = jwt.verify(token, process.env.JWT\_SECRET);

req.userId = decoded.userId;

req.role = decoded.role;

next();

} catch (error) {

return res.status(400).json({ message: 'Invalid or expired token.' });

}

};

const authMiddlewareNotReq = (req, res, next) => {

const token = req.header('Authorization')?.replace('Bearer ', '');

if (!token) {

return next();

}

try {

const decoded = jwt.verify(token, process.env.JWT\_SECRET);

req.userId = decoded.userId;

req.role = decoded.role;

next();

} catch (error) {

return next();

}

};

const isEmployer = (req, res, next) => {

if (req.role !== 'employer') {

return res.status(403).json({ message: 'Access denied. You must be an employer to perform this action.' });

}

next();

};

const isAdmin = (req, res, next) => {

if (req.role !== 'admin') {

return res.status(403).json({ message: 'Access denied. You must be an employer to perform this action.' });

}

next();

};

const isJobSeeker = (req, res, next) => {

if (req.role !== 'job\_seeker') {

return res.status(403).json({ message: 'Access denied. You must be an job\_seeker to perform this action.' });

}

next();

};

export { authMiddleware, authMiddlewareNotReq, isAdmin, isEmployer, isJobSeeker };

Підключення до бд (/db.js)

import mysql from 'mysql2';

import dotenv from 'dotenv';

dotenv.config();

const pool = mysql.createPool({

host: process.env.DB\_HOST,

user: process.env.DB\_USER,

password: process.env.DB\_PASSWORD,

database: process.env.DB\_NAME,

waitForConnections: true,

connectionLimit: 10,

queueLimit: 0

});

const promisePool = pool.promise();

export default promisePool;

Контролери

Контролер адміністратора (/controllers/adminController.js)

import db from '../models/db.js';

export const blockUser = async (req, res) => {

const { userId } = req.params;

try {

await db.query('UPDATE Users SET blocked = TRUE WHERE user\_id = ?', [userId]);

res.json({ message: 'Користувача заблоковано.' });

} catch (error) {

res.status(500).json({ message: 'Помилка сервера.' });

}

};

export const unblockUser = async (req, res) => {

const { userId } = req.params;

try {

await db.query('UPDATE Users SET blocked = FALSE WHERE user\_id = ?', [userId]);

res.json({ message: 'Користувача розблоковано.' });

} catch (error) {

res.status(500).json({ message: 'Помилка сервера.' });

}

};

export const getAllJobSeekers = async (req, res) => {

try {

const [rows] = await db.query(

`SELECT u.user\_id, u.first\_name, u.last\_name, u.email, u.blocked, u.registration\_date

FROM Users u

WHERE u.role = 'job\_seeker'`

);

res.json(rows);

} catch (error) {

res.status(500).json({ message: 'Помилка сервера.' });

}

};

export const getAllEmployers = async (req, res) => {

try {

const [rows] = await db.query(

`SELECT u.user\_id, u.first\_name, u.last\_name, u.email, u.blocked, u.registration\_date, e.company\_name

FROM Users u

JOIN EmployerProfiles e ON u.user\_id = e.user\_id

WHERE u.role = 'employer'`

);

res.json(rows);

} catch (error) {

res.status(500).json({ message: 'Помилка сервера.' });

}

};

export const blockJob = async (req, res) => {

const { jobId } = req.params;

try {

await db.query('UPDATE Jobs SET blocked = TRUE WHERE job\_id = ?', [jobId]);

res.json({ message: 'Вакансію заблоковано.' });

} catch (error) {

res.status(500).json({ message: 'Помилка сервера.' });

}

};

export const unblockJob = async (req, res) => {

const { jobId } = req.params;

try {

await db.query('UPDATE Jobs SET blocked = FALSE WHERE job\_id = ?', [jobId]);

res.json({ message: 'Вакансію розблоковано.' });

} catch (error) {

res.status(500).json({ message: 'Помилка сервера.' });

}

};

export const getAllJobs = async (req, res) => {

try {

const [rows] = await db.query(

`SELECT j.job\_id, j.title, j.blocked, j.status, j.posting\_date, e.company\_name, c.name AS category\_name

FROM Jobs j

JOIN EmployerProfiles e ON j.employer\_id = e.user\_id

JOIN JobCategories c ON j.category\_id = c.category\_id`

);

res.json(rows);

} catch (error) {

res.status(500).json({ message: 'Помилка сервера.' });

}

};

Контролер аналізу (/controller/analysisController.js)

import db from '../models/db.js';

const getApplicationsStats = async (req, res) => {

try {

const { category } = req.query;

let query, params = [];

if (category) {

query = `

SELECT

jc.category\_id AS category\_id,

jc.name AS category,

SUM(CASE WHEN a.status = 'rejected' THEN 1 ELSE 0 END) AS cancelled,

SUM(CASE WHEN a.status = 'accepted' THEN 1 ELSE 0 END) AS accepted,

SUM(CASE WHEN a.status = 'pending' THEN 1 ELSE 0 END) AS pending

FROM JobCategories jc

LEFT JOIN Jobs j ON j.category\_id = jc.category\_id

LEFT JOIN Applications a ON a.job\_id = j.job\_id

WHERE jc.category\_id = ?

GROUP BY jc.category\_id, jc.name

`;

params.push(category);

} else {

query = `

SELECT

SUM(CASE WHEN a.status = 'rejected' THEN 1 ELSE 0 END) AS cancelled,

SUM(CASE WHEN a.status = 'accepted' THEN 1 ELSE 0 END) AS accepted,

SUM(CASE WHEN a.status = 'pending' THEN 1 ELSE 0 END) AS pending

FROM Applications a

`;

}

const [rows] = await db.query(query, params);

res.json(rows);

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Server error.' });

}

};

export { getApplicationsStats };

Контролер авторизації (/controller/authController.js)

import bcrypt from 'bcrypt';

import jwt from 'jsonwebtoken';

import { sendVerificationEmail } from '../services/emailService.js';

import { generateToken } from '../utils/tokenUtils.js';

import db from '../models/db.js';

const register = async (req, res) => {

const { first\_name, last\_name, email, password, role } = req.body;

const connection = await db.getConnection();

try {

await connection.beginTransaction();

const [existingUser] = await connection.query('SELECT \* FROM Users WHERE email = ?', [email]);

if (existingUser.length > 0) {

return res.status(400).json({ message: 'Цей email вже зареєстровано.' });

}

const hashedPassword = await bcrypt.hash(password, 10);

const verificationToken = generateToken();

await connection.query(

'INSERT INTO Users (first\_name, last\_name, email, password, role, verified, verification\_token) VALUES (?, ?, ?, ?, ?, ?, ?)',

[first\_name, last\_name, email, hashedPassword, role, false, verificationToken]

);

await sendVerificationEmail(email, verificationToken);

await connection.commit();

res.status(201).json({ message: 'Користувача зареєстровано. Будь ласка, підтвердіть email.' });

} catch (error) {

await connection.rollback();

console.error(error);

res.status(500).json({ message: 'Помилка сервера.' });

} finally {

connection.release();

}

};

const registerEmployer = async (req, res) => {

const { first\_name, last\_name, email, password, company\_name, logo, company\_description } = req.body;

const role = 'employer';

const connection = await db.getConnection();

try {

await connection.beginTransaction();

const [existingUser] = await connection.query('SELECT \* FROM Users WHERE email = ?', [email]);

if (existingUser.length > 0) {

return res.status(400).json({ message: 'Цей email вже зареєстровано.' });

}

const hashedPassword = await bcrypt.hash(password, 10);

const verificationToken = generateToken();

const [userResult] = await connection.query(

'INSERT INTO Users (first\_name, last\_name, email, password, role, verified, verification\_token) VALUES (?, ?, ?, ?, ?, ?, ?)',

[first\_name, last\_name, email, hashedPassword, role, false, verificationToken]

);

const user\_id = userResult.insertId;

await connection.query(

'INSERT INTO EmployerProfiles (user\_id, company\_name, logo, company\_description) VALUES (?, ?, ?, ?)',

[user\_id, company\_name, logo, company\_description]

);

await sendVerificationEmail(email, verificationToken);

await connection.commit();

res.status(201).json({ message: 'Роботодавця зареєстровано. Будь ласка, підтвердіть email.' });

} catch (error) {

await connection.rollback();

console.error(error);

res.status(500).json({ message: 'Помилка сервера.' });

} finally {

connection.release();

}

};

const verifyEmail = async (req, res) => {

const { token } = req.params;

try {

const [userRows] = await db.query('SELECT \* FROM Users WHERE verification\_token = ?', [token]);

if (userRows.length === 0) {

return res.status(400).json({ message: 'Недійсний або прострочений токен.' });

}

const user = userRows[0];

await db.query('UPDATE Users SET verified = ?, verification\_token = ? WHERE user\_id = ?', [true, null, user.user\_id]);

res.json({ message: 'Email успішно підтверджено.' });

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Помилка сервера.' });

}

};

const login = async (req, res) => {

const { email, password } = req.body;

try {

const [userRows] = await db.query('SELECT \* FROM Users WHERE email = ?', [email]);

if (userRows.length === 0) {

return res.status(400).json({ message: 'Невірний email або пароль.' });

}

const user = userRows[0];

const isPasswordValid = await bcrypt.compare(password, user.password);

if (!isPasswordValid) {

return res.status(400).json({ message: 'Невірний email або пароль.' });

}

if (user.blocked) {

return res.status(403).json({ message: 'Ваш акаунт заблоковано. Зверніться до адміністратора.' });

}

if (!user.verified) {

return res.status(400).json({ message: 'Будь ласка, підтвердіть свій email.' });

}

const jwtToken = jwt.sign(

{ userId: user.user\_id,

email: user.email,

first\_name: user.first\_name,

last\_name: user.last\_name,

role: user.role },

process.env.JWT\_SECRET,

{ expiresIn: '7d' }

);

res.json({ token: jwtToken });

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Помилка сервера.' });

}

};

export { register, registerEmployer, verifyEmail, login };

Контролер заявок (/controller /applicationController.js)

import db from '../models/db.js';

import { sendMail } from '../services/emailService.js';

const applyForJob = async (req, res) => {

const jobSeekerId = req.userId;

const { job\_id } = req.body;

console.log(req.userId);

console.log(req.body);

try {

const [existingApplication] = await db.query('SELECT \* FROM Applications WHERE job\_seeker\_id = ? AND job\_id = ?', [jobSeekerId, job\_id]);

if (existingApplication.length > 0) {

return res.status(400).json({ message: 'You have already applied for this job.' });

}

const [result] = await db.query(

'INSERT INTO Applications (job\_seeker\_id, job\_id) VALUES (?, ?)',

[jobSeekerId, job\_id]

);

res.status(201).json({ message: 'Application submitted successfully.', applicationId: result.insertId });

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Server error.' });

}

};

const getJobSeekerApplications = async (req, res) => {

const jobSeekerId = req.userId;

try {

const [applications] = await db.query(

`SELECT

a.application\_id,

a.job\_id,

a.status,

a.application\_date,

j.title AS job\_title,

j.location,

j.employment\_type,

j.salary,

j.salaryMax

FROM Applications a

JOIN Jobs j ON a.job\_id = j.job\_id

WHERE a.job\_seeker\_id = ?

ORDER BY a.application\_date DESC`,

[jobSeekerId]

);

res.json(applications);

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Server error.' });

}

};

const manageApplication = async (req, res) => {

const employerId = req.userId;

const { application\_id, status } = req.body;

const applicationId = application\_id;

if (status !== 'accepted' && status !== 'rejected') {

return res.status(400).json({ message: 'Invalid status. Must be either "accepted" or "rejected".' });

}

try {

const [application] = await db.query('SELECT \* FROM Applications WHERE application\_id = ?', [applicationId]);

if (application.length === 0) {

return res.status(404).json({ message: 'Application not found.' });

}

const applicationData = application[0];

const [jobPosting] = await db.query('SELECT \* FROM Jobs WHERE job\_id = ?', [applicationData.job\_id]);

if (jobPosting.length === 0 || jobPosting[0].employer\_id !== employerId) {

return res.status(403).json({ message: 'You are not authorized to manage this application.' });

}

await db.query('UPDATE Applications SET status = ? WHERE application\_id = ?', [status, applicationId]);

const [userRows] = await db.query('SELECT email, first\_name FROM Users WHERE user\_id = ?', [applicationData.job\_seeker\_id]);

const user = userRows[0];

const job = jobPosting[0];

if (user && user.email) {

const statusText = status === 'accepted' ? 'прийнята' : 'відхилена';

const jobLink = `${process.env.FRONTEND\_URL || 'http://localhost:3000'}/job/${job.job\_id}`;

const subject = `Ваша заявка на вакансію "${job.title}" була розглянута`;

const html = `

<p>Ваша заявка на вакансію <b>${job.title}</b> була <b>${statusText}</b>.</p>

<p>Детальніше ви можете переглянути за <a href="${jobLink}">цим посиланням</a>.</p>

`;

await sendMail({ to: user.email, subject, html });

}

res.json({ message: `Application ${status} successfully.` });

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Server error.' });

}

};

const getJobApplications = async (req, res) => {

const employerId = req.userId;

const { job\_id } = req.params;

try {

const [jobRows] = await db.query(

'SELECT \* FROM Jobs WHERE job\_id = ? AND employer\_id = ?',

[job\_id, employerId]

);

if (jobRows.length === 0) {

return res.status(404).json({ message: 'Job not found or you are not authorized to view applications for this job.' });

}

const job = jobRows[0];

const [applications] = await db.query(

`SELECT

a.application\_id, a.job\_seeker\_id, a.status, a.application\_date

FROM Applications a

WHERE a.job\_id = ?`,

[job\_id]

);

const detailedApplications = await Promise.all(applications.map(async (app) => {

const [userRows] = await db.query(

'SELECT user\_id, first\_name, last\_name, email, phone FROM Users WHERE user\_id = ?',

[app.job\_seeker\_id]

);

const user = userRows[0] || {};

// Профіль пошукача роботи

const [profileRows] = await db.query(

`SELECT

prefered\_category\_1, prefered\_category\_2, skills, experience, city,

experience\_text, education, expectations, portfolio

FROM JobSeekerProfiles WHERE user\_id = ?`,

[app.job\_seeker\_id]

);

const profile = profileRows[0] || {};

// Всі резюме користувача

const [resumes] = await db.query(

'SELECT resume\_id, title, resume\_file, upload\_date FROM Resumes WHERE job\_seeker\_id = ?',

[app.job\_seeker\_id]

);

return {

application\_id: app.application\_id,

status: app.status,

application\_date: app.application\_date,

user: {

...user,

profile,

resumes: resumes || []

}

};

}));

res.json({

job,

applications: detailedApplications

});

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Server error.' });

}

};

export { applyForJob, manageApplication, getJobSeekerApplications, getJobApplications };

Контролер категорій (/controller /categoryController.js)

import db from '../models/db.js';

const getCategories = async (req, res) => {

try {

const [categories] = await db.query('SELECT \* FROM JobCategories');

res.json(categories);

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Server error.' });

}

};

const updateCategories = async (req, res) => {

try {

const { categories } = req.body;

if (!Array.isArray(categories)) {

return res.status(400).json({ message: 'Categories must be an array.' });

}

const [currentRows] = await db.query('SELECT name FROM JobCategories');

const current = currentRows.map(row => row.name);

const toAdd = categories.filter(cat => !current.includes(cat));

const toDelete = current.filter(cat => !categories.includes(cat));

for (const name of toAdd) {

await db.query('INSERT INTO JobCategories (name) VALUES (?)', [name]);

}

for (const name of toDelete) {

await db.query('DELETE FROM JobCategories WHERE name = ?', [name]);

}

res.json({ added: toAdd, deleted: toDelete });

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Server error.' });

}

};

export { getCategories, updateCategories };

Контролер робіт (/controller/jobController.js)

import db from '../models/db.js';

const createJob = async (req, res) => {

const { title, description, requirements, salary, salaryMax, employment\_type, location, category\_id } = req.body;

const employer\_id = req.userId;

if (!title || !description || !requirements || !salary || !employment\_type || !location || !category\_id) {

return res.status(400).json({ message: 'All fields are required.' });

}

try {

const [result] = await db.query(

`INSERT INTO Jobs (employer\_id, category\_id, title, description, requirements, salary, salaryMax, employment\_type, location, status)

VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)`,

[employer\_id, category\_id, title, description, requirements, salary, salaryMax, employment\_type, location, 'draft']

);

res.status(201).json({ message: 'Job posting created successfully.', jobId: result.insertId });

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Server error.' });

}

};

const updateJob = async (req, res) => {

const { id } = req.params;

const employerId = req.userId;

const { title, description, requirements, salary, salaryMax, employment\_type, location, category\_id, status } = req.body;

try {

const [existing] = await db.query('SELECT \* FROM Jobs WHERE job\_id = ?', [id]);

if (!existing.length) {

return res.status(404).json({ message: 'Job posting not found.' });

}

if (existing[0].employer\_id !== employerId) {

return res.status(403).json({ message: 'Unauthorized. You do not own this job posting.' });

}

await db.query(

`UPDATE Jobs

SET status = ?, title = ?, description = ?, requirements = ?, salary = ?, salaryMax = ?, employment\_type = ?, location = ?, category\_id = ?

WHERE job\_id = ?`,

[status, title, description, requirements, salary, salaryMax, employment\_type, location, category\_id, id]

);

res.json({ message: 'Job posting updated successfully.' });

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Server error.' });

}

};

const updateJobStatus = async (req, res) => {

const { id } = req.params;

const { status } = req.body;

const employerId = req.userId;

const allowedStatuses = ['open', 'closed', 'draft'];

if (!allowedStatuses.includes(status)) {

return res.status(400).json({ message: 'Invalid status.' });

}

try {

const [existing] = await db.query('SELECT \* FROM Jobs WHERE job\_id = ?', [id]);

if (!existing.length) {

return res.status(404).json({ message: 'Job posting not found.' });

}

if (existing[0].employer\_id !== employerId) {

return res.status(403).json({ message: 'Unauthorized. You do not own this job posting.' });

}

await db.query(

'UPDATE Jobs SET status = ? WHERE job\_id = ?',

[status, id]

);

res.json({ message: 'Job posting status updated successfully.' });

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Server error.' });

}

};

const getEmployerJobs = async (req, res) => {

const employerId = req.userId;

try {

const [postings] = await db.query(

`SELECT j.job\_id, j.blocked, j.category\_id, j.title, j.description, j.salary, j.salaryMax, j.employment\_type, j.location, j.status, j.posting\_date, jc.name AS category\_name,

(SELECT COUNT(\*) FROM Applications a WHERE a.job\_id = j.job\_id AND a.status = 'pending') AS pending\_applications

FROM Jobs j

JOIN JobCategories jc ON j.category\_id = jc.category\_id

WHERE j.employer\_id = ?`,

[employerId]

);

console.log(postings);

if (postings.length === 0) {

return res.status(404).json({ message: 'No jobs found for this employer.' });

}

res.json(postings);

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Server error.' });

}

};

const getJobsByCategory = async (req, res) => {

let categoryId = req.params.id;

console.log(categoryId);

try {

let query = `

SELECT j.job\_id, j.category\_id, j.title, j.description, j.salary, j.salaryMax,

j.employment\_type, j.location, j.status, j.posting\_date,

e.company\_name, jc.name AS category\_name

FROM Jobs j

JOIN EmployerProfiles e ON j.employer\_id = e.user\_id

JOIN JobCategories jc ON j.category\_id = jc.category\_id

WHERE j.status = 'open' and j.blocked = false

`;

let params = [];

if (categoryId !== "0") {

query += ` AND j.category\_id = ${categoryId}`;

}

const [jobs] = await db.query(query, params);

res.json(jobs);

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Server error.' });

}

};

const getJobById = async (req, res) => {

let id = req.params.id;

try {

let query = `

SELECT j.job\_id, j.blocked, j.category\_id, j.title, j.description, j.salary, j.salaryMax,

j.employment\_type, j.location, j.status, j.posting\_date,

e.company\_name, jc.name AS category\_name

FROM Jobs j

JOIN EmployerProfiles e ON j.employer\_id = e.user\_id

JOIN JobCategories jc ON j.category\_id = jc.category\_id

WHERE j.job\_id = ?`;

let params = [id];

const [jobs] = await db.query(query, params);

if (jobs.length === 0) {

res.json({});

return;

}

res.json(jobs[0])

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Server error.' });

}

};

const getLatestJobs = async (req, res) => {

const userId = req.userId;

console.log(userId);

try {

let query = `

SELECT j.job\_id, j.title, j.description, j.salary, j.salaryMax,

j.employment\_type, j.location, j.status, j.posting\_date,

e.company\_name, jc.name AS category\_name

FROM Jobs j

JOIN EmployerProfiles e ON j.employer\_id = e.user\_id

JOIN JobCategories jc ON j.category\_id = jc.category\_id

WHERE j.status = 'open' and j.blocked = false

`;

let params = [];

if (userId) {

const [preferences] = await db.query(

`SELECT prefered\_category\_1, prefered\_category\_2 FROM JobSeekerProfiles WHERE user\_id = ?`,

[userId]

);

if (preferences.length > 0) {

const { prefered\_category\_1, prefered\_category\_2 } = preferences[0];

const categoryConditions = [];

if(prefered\_category\_1) categoryConditions.push(`j.category\_id = ${prefered\_category\_1}`);

if (prefered\_category\_2 && prefered\_category\_2 !== prefered\_category\_1)

categoryConditions.push(`j.category\_id = ${prefered\_category\_2}`);

if (categoryConditions.length > 0) {

query += ` AND (${categoryConditions.join(" OR ")})`;

}

}

}

query += ` ORDER BY j.posting\_date DESC LIMIT 10`;

const [jobs] = await db.query(query, params);

res.json(jobs);

} catch (error) {

console.error(error);

res.status(500).json({ message: "Server error." });

}

};

export {

createJob,

updateJob,

updateJobStatus,

getEmployerJobs,

getJobsByCategory,

getJobById,

getLatestJobs

};

Контролер резюме (/controller /resumeController.js)

import db from '../models/db.js';

import path from 'path';

import fs from 'fs';

const createResume = async (req, res) => {

const userId = req.userId;

const file = req.file;

console.log(req.file.originalname)

if (!file) {

return res.status(400).json({ message: 'Resume file is required.' });

}

try {

const fileName = `${userId}\_${Date.now()}\_${file.originalname}`;

const filePath = path.join(process.cwd(), 'uploads/resumes', fileName);

const title = file.originalname;

fs.renameSync(file.path, filePath);

const [result] = await db.query(

'INSERT INTO Resumes (job\_seeker\_id, title, resume\_file) VALUES (?, ?, ?)',

[userId, title, fileName]

);

res.status(201).json({ message: 'Resume uploaded successfully.', resumeId: result.insertId });

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Server error.' });

}

};

const deleteResume = async (req, res) => {

const userId = req.userId;

const { id } = req.params;

try {

const [resume] = await db.query('SELECT \* FROM Resumes WHERE resume\_id = ?', [id]);

if (!resume.length) {

return res.status(404).json({ message: 'Resume not found.' });

}

if (resume[0].job\_seeker\_id !== userId) {

return res.status(403).json({ message: 'Unauthorized. You do not own this resume.' });

}

const filePath = path.join(process.cwd(), 'uploads/resumes', resume[0].resume\_file); // оновлено для надійності

if (fs.existsSync(filePath)) {

fs.unlinkSync(filePath);

}

await db.query('DELETE FROM Resumes WHERE resume\_id = ?', [id]);

res.json({ message: 'Resume deleted successfully.' });

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Server error.' });

}

};

const getResumes = async (req, res) => {

const userId = req.userId;

try {

const [resumes] = await db.query('SELECT \* FROM Resumes WHERE job\_seeker\_id = ?', [userId]);

res.json({ resumes });

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Server error.' });

}

};

const getResumeFile = async (req, res) => {

const { resume\_id } = req.params;

try {

const [resume] = await db.query('SELECT \* FROM Resumes WHERE resume\_id = ?', [resume\_id]);

if (!resume.length) {

return res.status(404).json({ message: 'Resume not found.' });

}

const filePath = path.join(process.cwd(), 'uploads/resumes', resume[0].resume\_file);

console.log(filePath);

if (!fs.existsSync(filePath)) {

return res.status(404).json({ message: 'Resume file not found on server.' });

}

res.sendFile(resume[0].resume\_file, {

root: path.join(process.cwd(), 'uploads', 'resumes'),

});

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Server error while retrieving file.' });

}

};

const getResume = async (req, res) => {

const { id } = req.params;

try {

const [resume] = await db.query('SELECT \* FROM Resumes WHERE resume\_id = ?', [id]);

if (!resume.length) {

return res.status(404).json({ message: 'Resume not found.' });

}

res.json(resume[0]);

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Server error.' });

}

};

export { createResume, deleteResume, getResumes, getResumeFile, getResume };

Контролер профілів (/controller /profileController.js)

import db from '../models/db.js';

const upsertJobSeekerProfile = async (req, res) => {

const {

education,

experience,

experience\_text,

skills,

city,

expectations,

portfolio,

prefered\_category\_1,

prefered\_category\_2,

} = req.body;

const userId = req.userId;

try {

const [existingProfile] = await db.query(

'SELECT \* FROM JobSeekerProfiles WHERE user\_id = ?',

[userId]

);

if (existingProfile.length > 0) {

await db.query(

`UPDATE JobSeekerProfiles

SET education = ?, experience = ?, experience\_text = ?, skills = ?, city = ?, expectations = ?, portfolio = ?, prefered\_category\_1 = ?, prefered\_category\_2 = ?

WHERE user\_id = ?`,

[

education,

experience,

experience\_text,

skills,

city,

expectations,

portfolio,

prefered\_category\_1,

prefered\_category\_2,

userId,

]

);

return res.json({ message: 'Job seeker profile updated successfully.' });

} else {

await db.query(

`INSERT INTO JobSeekerProfiles

(user\_id, education, experience, experience\_text, skills, city, expectations, portfolio, prefered\_category\_1, prefered\_category\_2)

VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)`,

[

userId,

education,

experience,

experience\_text,

skills,

city,

expectations,

portfolio,

prefered\_category\_1,

prefered\_category\_2,

]

);

return res.status(201).json({ message: 'Job seeker profile created successfully.' });

}

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Server error.' });

}

};

const upsertEmployerProfile = async (req, res) => {

const { company\_name, logo, company\_description } = req.body;

const userId = req.userId;

try {

const [existingProfile] = await db.query('SELECT \* FROM EmployerProfiles WHERE user\_id = ?', [userId]);

if (existingProfile.length > 0) {

await db.query(

'UPDATE EmployerProfiles SET company\_name = ?, logo = ?, company\_description = ? WHERE user\_id = ?',

[company\_name, logo, company\_description, userId]

);

return res.json({ message: 'Employer profile updated successfully.' });

} else {

await db.query(

'INSERT INTO EmployerProfiles (user\_id, company\_name, logo, company\_description) VALUES (?, ?, ?, ?)',

[userId, company\_name, logo, company\_description]

);

return res.status(201).json({ message: 'Employer profile created successfully.' });

}

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Server error.' });

}

};

const getJobSeekerProfile = async (req, res) => {

const userId = req.userId;

try {

const [profile] = await db.query(`

SELECT

p.\*,

c1.name AS prefered\_category\_1\_name,

c2.name AS prefered\_category\_2\_name

FROM JobSeekerProfiles p

LEFT JOIN JobCategories c1 ON p.prefered\_category\_1 = c1.category\_id

LEFT JOIN JobCategories c2 ON p.prefered\_category\_2 = c2.category\_id

WHERE p.user\_id = ?

`, [userId]);

if (profile.length === 0) {

await db.query('INSERT INTO JobSeekerProfiles (user\_id) VALUES (?)', [userId]);

const [newProfile] = await db.query('SELECT \* FROM JobSeekerProfiles WHERE user\_id = ?', [userId]);

return res.status(201).json(newProfile[0]);

}

res.json(profile[0]);

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Server error.' });

}

};

const getEmployerProfile = async (req, res) => {

const userId = req.userId;

try {

const [profileRows] = await db.query(

`SELECT

e.\*,

u.first\_name, u.last\_name, u.email, u.phone, u.verified, u.registration\_date, u.role

FROM EmployerProfiles e

JOIN Users u ON e.user\_id = u.user\_id

WHERE e.user\_id = ?`,

[userId]

);

if (profileRows.length === 0) {

await db.query('INSERT INTO EmployerProfiles (user\_id) VALUES (?)', [userId]);

const [newProfileRows] = await db.query(

`SELECT

e.\*,

u.first\_name, u.last\_name, u.email, u.phone, u.verified, u.registration\_date, u.role

FROM EmployerProfiles e

JOIN Users u ON e.user\_id = u.user\_id

WHERE e.user\_id = ?`,

[userId]

);

return res.status(201).json(newProfileRows[0]);

}

res.json(profileRows[0]);

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Server error.' });

}

};

export { upsertJobSeekerProfile, upsertEmployerProfile, getJobSeekerProfile, getEmployerProfile };