Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра

на тему: «Розроблення вебзастосунку для пошуку та підбору персоналу»

Виконав: студент групи К21-1

Спеціальність 122 Комп'ютерні науки

Дмітрієв Д.С.
(прізвище та ініціали)

Керівник к.т.н., доцент Ульяновська Ю.В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент Дніпровський державний технічний університет
(місце роботи)

доцент кафедри математичного моделювання та системного аналізу
(посада)

к.т.н., доц. Волосова Н.М.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

АНОТАЦІЯ

Дмитрієв Д. Розроблення вебзастосунку для пошуку та підбору персоналу.

Кваліфікаційна робота на здобуття освітнього ступеня бакалавр за спеціальністю 122 «Комп'ютерні науки». – Університет митної справи та фінансів, Дніпро, 2025.

У сучасних умовах стрімкої цифровізації та динамічного розвитку ринку праці актуальним стає питання ефективного пошуку та підбору персоналу. В роботі представлено розроблення вебзастосунку, який забезпечує автоматизацію та оптимізацію рекрутингових процесів із використанням сучасних вебтехнологій. Основною метою дослідження стало створення функціонального, безпечного та інтуїтивно зрозумілого інструменту, що дозволяє ефективно взаємодіяти роботодавцям і кандидатам, знижуючи часові та фінансові витрати на підбір кадрів.

У процесі розробки проаналізовано сучасні інформаційні системи та підходи до рекрутингу, визначено ключові функціональні вимоги до системи, обґрунтовано вибір технологій. Для реалізації застосунку використано мову програмування Python, мікрофреймворк Flask, систему управління базами даних SQLite3 та бібліотеки Flask-Login і Werkzeug, що забезпечили реалізацію механізмів аутентифікації, захисту даних та зручного управління користувацькими сесіями. У роботі детально описано архітектуру вебзастосунку, структуру бази даних, принципи побудови інтерфейсу користувача та алгоритми пошуку й фільтрації кандидатів і вакансій. Результати тестування підтвердили ефективність розробленого рішення для малих і середніх підприємств, які потребують швидкого й точного підбору персоналу.

Ключові слова: вебзастосунок, рекрутинг, пошук персоналу, Flask, SQLite, автоматизація, інформаційна система, підбір кадрів.

ABSTRACT

Dmytriiev D. Development of a web application for search and selection of personnel.

Qualification work for a bachelor's degree in specialty 122 «Computer Science». – University of Customs and Finance, Dnipro, 2025.

In the current conditions of rapid digitalization and dynamic development of the labor market, the issue of effective search and selection of personnel is becoming relevant. The paper presents the development of a web application that automates and optimizes recruiting processes using modern web technologies. The main goal of the study was to create a functional, secure and intuitive tool that allows employers and candidates to interact effectively, reducing the time and financial costs of recruitment.

In the course of development, we analyzed modern information systems and approaches to recruiting, identified key functional requirements for the system, and justified the choice of technologies. To implement the application, the Python programming language, the Flask microframework, the SQLite3 database management system, and the Flask-Login and Werkzeug libraries were used, which provided the implementation of authentication mechanisms, data protection, and convenient user session management. The paper describes in detail the architecture of the web application, the structure of the database, the principles of building the user interface, and algorithms for searching and filtering candidates and vacancies. The test results have confirmed the effectiveness of the developed solution for small and medium-sized enterprises that need fast and accurate recruitment.

Keywords: web application, recruiting, personnel search, Flask, SQLite, automation, information system, recruitment.

ЗМІСТ

ВСТУП 6
РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ 8
1.1 Аналіз сучасного стану ринку праці та потреб у автоматизації пошуку і підбору персоналу 8
1.2 Основні підходи та технології пошуку і підбору персоналу в інформаційних системах 10
1.3 Огляд сучасних вебзастосунків для рекрутингу. Функціональні можливості, переваги й обмеження 14
1.4 Формулювання мети, завдань, об'єкта та предмета дослідження 18
1.5 Висновки до першого розділу 19
РОЗДІЛ 2 АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ ВЕБЗАСТОСУНКУ ДЛЯ ПОШУКУ ТА ПІДБОРУ ПЕРСОНАЛУ 20
2.1 Мова програмування Python. Можливості та переваги для розробки вебзастосунків 20
2.2 Вебфреймворк Flask. Архітектура, особливості та застосування у створенні вебзастосунків 23
2.3 Система управління базами даних SQLite3. Структура, переваги та інтеграція з Flask 26
2.4 Бібліотека Flask-Login. Механізми аутентифікації та управління сесіями користувачів 30
2.5 Бібліотека Werkzeug. Інструменти для безпечного хешування паролів та обробки HTTP 32
2.6 Висновки до другого розділу 36
РОЗДІЛ 3 РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ПОШУКУ ТА ПІДБОРУ ПЕРСОНАЛУ 37
3.1 Аналіз вимог та загальна архітектура вебзастосунку 37

3.2 Проєктування бази даних. Структура таблиць users та jobs, сполучення таблиць і зв'язки 39
3.3 Реалізація серверної частини. Налаштування Flask, маршрутизація, логіка бізнес-процесів 43
3.4 Модель користувача та механізми аутентифікації. Використання Flask-Login, класи користувачів 47
3.5 Обробка запитів реєстрації та входу користувачів. Валідація, хешування паролів, повідомлення системи 49
3.6 Реалізація функціоналу публікації вакансій. Інтерфейси, валідація вводу, збереження даних 52
3.7 Висновки до третього розділу 63
ВИСНОВКИ 65
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 67
ДОДАТОК А 69
ДОДАТОК Б 70
ДОДАТОК В 77
ДОДАТОК Г 84

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій і глобалізації ринку праці, питання ефективного пошуку та підбору персоналу набуває особливої актуальності. Організації різних форм власності та сфер діяльності все частіше стикаються з необхідністю оперативного і якісного залучення кваліфікованих кадрів, що стає одним із ключових чинників їх успішного функціонування та конкурентоспроможності [1]. Традиційні методи рекрутингу, базовані переважно на ручному опрацюванні резюме та фільтрації кандидатів, втрачають свою ефективність у зв'язку з ростом обсягів інформації та підвищенням вимог до швидкості прийняття кадрових рішень. У такому контексті розроблення вебзастосунків, які автоматизують і оптимізують процеси пошуку і підбору персоналу, стає одним із пріоритетних завдань як для фахівців із інформаційних технологій, так і для кадрових служб підприємств [1, 2].

Мета даної кваліфікаційної роботи полягає у створенні сучасного вебзастосунку, призначеного для оптимізації процесу пошуку та підбору персоналу шляхом автоматизації збору, систематизації і аналізу інформації про кандидатів. Для досягнення поставленої мети необхідно розв'язати такі завдання: провести аналіз існуючих рішень на ринку систем підбору персоналу, визначити основні функціональні та нефункціональні вимоги до вебзастосунку, розробити архітектуру та дизайн системи, реалізувати основні модулі застосунку, а також провести тестування і оцінку ефективності розробленого рішення.

Об'єктом дослідження є процеси пошуку, підбору та оцінки персоналу в умовах сучасного ринку праці, а предметом дослідження – технічні та програмні засоби для автоматизації цих процесів на основі вебтехнологій.

У роботі застосовано комплексну методологію дослідження, що включає аналіз літературних джерел та сучасних інформаційних систем, методи системного аналізу для розробки архітектури додатку, програмування

з використанням сучасних вебфреймворків, а також тестування програмного забезпечення із залученням статистичних методів для оцінки продуктивності системи.

Практична значимість роботи полягає у створенні універсального і адаптивного інструменту, який може бути використаний кадровими службами малих і середніх підприємств для підвищення ефективності підбору кандидатів, зменшення часу на обробку великих обсягів резюме та підвищення якості вибору кандидатів.

Структура роботи. Кваліфікаційна робота складається зі вступу, 3-х розділів, висновків, використаних джерел з 16 найменувань, 4 додатків. Обсяг роботи складається з 91 сторінки, 23 рисунків та 3 таблиць.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Аналіз сучасного стану ринку праці та потреб у автоматизації пошуку і підбору персоналу

Сучасний ринок праці характеризується значною динамічністю та високим рівнем конкуренції, що обумовлює форму постійних змін у структурі попиту та пропозиції робочої сили [1]. Глобалізація економіки, інтенсивний розвиток інформаційних технологій, трансформація моделей виробництва та організації праці сприяють формуванню нових вимог до кваліфікаційних і професійних характеристик робітників, а також до гнучкості процесів працевлаштування. У цьому контексті виникає все більша потреба у цифрових інструментах, здатних оптимізувати процеси пошуку та підбору персоналу, забезпечуючи швидку адаптацію роботодавців до мінливих ринкових умов [1].

Важливо зауважити, що традиційні методи підбору кадрів, які переважно базувалися на особистих контактах, оголошеннях у друкованих ЗМІ та посередницьких агенціях, дедалі більше втратили ефективність в умовах сучасних вимог. Зростання обсягів інформації про вакансії та кандидатів, а також збільшення географічного та професійного розмаїття учасників ринку праці призводять до необхідності застосування масштабованих та автоматизованих рішень. Вони мають забезпечувати швидкий доступ до актуальної інформації, якісний аналіз відповідності кандидатів профілю вакансії, а також максимальне скорочення витрат часу та ресурсів на рутинні операції.

До значних факторів, що сприяють зростанню попиту на автоматизацію пошуку та підбору персоналу, належать технологічні інновації в галузі штучного інтелекту, машинного навчання та обробки природної мови. Використання таких інструментів дозволяє створити системи, здатні автономно фільтрувати великі масиви резюме, прогнозувати відповідність

кандидатів вакансіям на основі аналізу їхніх компетенцій, а також ефективно управляти взаємодією між сторонами процесу найму. Це особливо актуально в умовах стрімкого розвитку цифрових каналів комунікації і зростання ролі віддаленої роботи, що розширює географічні межі пошуку кандидатів.

Не менш важливим є той аспект, що сучасна економіка вимагає більшої гнучкості у формуванні робочих команд [1, 2]. Перехід від традиційних моделей повної зайнятості до контрактних, тимчасових, а також гібридних форматів праці обумовлює необхідність оперативного підбору кадрів з урахуванням специфічних завдань та короткострокових проєктів. Відповідно, автоматизовані системи рекрутингу повинні забезпечувати адаптивність налаштувань та можливість тонкої фільтрації за великою кількістю параметрів, включно з професійними навичками, часовою доступністю, географією та іншими релевантними факторами.

З огляду на соціально-економічні трансформації, що відбуваються впродовж останніх років, ринок праці зазнає змін як в структурі зайнятості, так і в поведінкових патернах учасників. Кандидати стають більш мобільними й активними у використанні цифрових ресурсів для пошуку роботи, одночасно з високими очікуваннями щодо прозорості та оперативності комунікації з потенційними роботодавцями. Відповідно, роботодавці змушені впроваджувати нові методи залучення працівників, фокусуючись на побудові позитивного іміджу компанії, якісному описі вакансій і використанні аналітичних інструментів для оптимізації процесів рекрутингу.

У світі сучасних технологій зростає роль платформ, що інтегрують різноманітні сервіси для пошуку та підбору персоналу, надаючи відразу комплекс можливостей – від публікації вакансій до автоматизованого скринінгу та аналітики. Це значно підвищує продуктивність та об’єктивність прийняття кадрових рішень, знижує ризики суб’єктивності, передбачає адаптацію до специфіки різних галузей та розмірів бізнесу. Особливо це помітно у малому та середньому бізнесі, де відсутність власних рекрутингових

ресурсів робить автоматизовані рішення привабливими за співвідношенням ціна-якість.

Водночас слід підкреслити, що впровадження таких систем не позбавлене викликів. Етичні аспекти обробки персональних даних, питання забезпечення конфіденційності, а також необхідність усунення технологічних упереджень у процесі автоматичного відбору кандидатів залишаються актуальними проблемами. У зв'язку з цим важливо розробляти стандарти та протоколи, орієнтовані на прозорість і справедливість систем, що сприяють підтримці довіри між усіма учасниками ринку праці.

Загалом, аналізуючи сучасний стан ринку праці, можна констатувати, що потреба в автоматизації пошуку та підбору персоналу є об'єктивною і зростаючою. Зміни у структурі зайнятості, технічний прогрес, глобалізація та трансформація соціальних моделей ведуть до появи все більш складних і масштабних процесів відбору кадрів. Відповідно, впровадження сучасних вебзастосунків, що забезпечують оптимізацію пошуку вакансій, автоматизацію фільтрації резюме та взаємодію між кандидатами і роботодавцями, стає одним із ключових чинників підвищення конкурентоспроможності як окремих організацій, так і ринку праці в цілому. Така автоматизація дає змогу ефективніше відповідати на виклики часу, мінімізувати людський фактор помилок, а також суттєво скоротити часові та фінансові витрати, що робить її невід'ємною складовою сучасної кадрової інфраструктури.

1.2 Основні підходи та технології пошуку і підбору персоналу в інформаційних системах

Пошук і підбір персоналу є складним та багатогранним процесом, який відіграє критичну роль у функціонуванні будь-якої організації. З розвитком інформаційних технологій цей процес зазнав суттєвих змін, що дозволило значно підвищити ефективність і якість найму працівників. Сучасні

інформаційні системи підбору персоналу інтегрують різні підходи і технології з метою оптимізації процесу від пошуку кандидатів до їх відбору та найму (рис. 1.1) [1, 2].

Таблиця 1.1

Підходи та технології пошуку і підбору персоналу

Основні підходи	Короткий опис
Автоматизація рекрутингу	Використання HRMS та алгоритмів AI для скринінгу резюме та оцінки компетенцій.
Бази даних резюме	Централізоване збереження та пошук кандидатів за різними параметрами.
Інтеграція із зовнішніми платформами	Імпорт даних кандидатів через API соціальних мереж і професійних порталів.
Штучний інтелект і аналітика даних	NLP, машинне навчання, Big Data для підвищення точності і швидкості відбору.
Новітні технології	Онлайн-тестування, мобільні застосунки, блокчейн, хмарні платформи для масштабованості та безпеки.

Одним із ключових підходів у цій сфері є автоматизація рутинних етапів рекрутингу, яка спирається на впровадження систем управління персоналом (HRMS – Human Resource Management Systems) та платформ для автоматизованого скринінгу резюме. При цьому, значну увагу приділяють створенню алгоритмів, здатних аналізувати великі обсяги даних, що надходять від кандидатів, та відбирати найбільш релевантні профілі за допомогою методів штучного інтелекту і машинного навчання. Такі системи володіють можливістю оцінювати компетенції, навички та досвід кандидатів, співставляючи їх із вимогами вакансій.

До класичних технологічних рішень відносять використання баз даних резюме, які дозволяють будувати і підтримувати централізовані реєстри претендентів [1, 2]. Інформаційні системи організовують збереження, пошук та фільтрацію даних, що значно скорочує часові витрати рекрутерів. Пошук у таких системах базується на ключових словах, параметрах досвіду, освіті та інших критеріїв, що забезпечує цільовий підбір потенційних кандидатів. Разом з цим, системи все частіше доповнюють наборами інструментів для більш глибокого аналізу, серед яких статистичне опрацювання, семантичний аналіз і машинне розпізнавання патернів.

Іншим вагомим підходом є інтеграція з зовнішніми платформами і соціальними мережами, такими як LinkedIn, Facebook, спеціалізовані професійні форуми та портали праці. Це дозволяє значно розширити базу потенційних кандидатів за межі локальних баз даних і автоматизувати процес збору інформації про претендентів. Інформаційні системи використовують API інтеграції для імпорту даних кандидатів, що підвищує оперативність пошуку і можливість аналізу соціального і професійного профілю кожного претендента.

Значний розвиток зазнали технології, що описують профілі кандидатів за допомогою структурованих форматів даних, таких як XML, JSON чи RDF. Це забезпечує семантичну сумісність між різними системами та дозволяє більш гнучко опрацьовувати інформацію, проводити глибинний аналіз та порівняння профілів і вакансій. Семантичні інформаційні системи здатні враховувати контекст, а не лише точні збіги ключових слів, що суттєво покращує якість підбору [2].

Використання штучного інтелекту у підборі персоналу значно розширило межі традиційних способів пошуку. Окрім класичного пошуку за ключовими словами, системи застосовують алгоритми аналізу природної мови (NLP) для розуміння контексту резюме і мотиваційних листів. Моделі машинного навчання навчаються на історичних даних, дозволяючи прогнозувати відповідність кандидатів до вакансій, їхню працездатність і

продуктивність. При цьому, все більше реалізуються системи рекомендацій, які пропонують кандидатів із найбільшою ймовірністю успішної адаптації та розвитку в організації.

Важливою складовою сучасних підходів є також використання аналітики великих даних (Big Data Analytics). Збір і обробка великих обсягів інформації, включаючи поведінкові дані, оцінки ефективності, результати тестувань та соціальні індикатори, дозволяють створювати комплексні профілі претендента. Ці дані інтегруються в системи підтримки прийняття рішень, що допомагає рекрутерам отримувати глибоке розуміння потенційних кандидатів і обирати найкращих із них.

Для підвищення інтерактивності і об'єктивності оцінки кандидатів у сучасних інформаційних системах поширене впровадження онлайн-тестувань, відеоспівбесід та гейміфікації процесу відбору. Такі технології допомагають не лише зібрати додаткові дані, але й покращують залученість користувачів та знижують суб'єктивність у прийнятті рішень, оскільки результати оцінок більш стандартизовані і легко порівнювані.

Не менш важливими є й аспекти безпеки та конфіденційності в процесах пошуку і підбору персоналу. Інформаційні системи мають відповідати вимогам законодавства щодо захисту персональних даних, включаючи шифрування інформації, контроль доступу і аудит транзакцій. Це забезпечує захист як самих кандидатів, так і роботодавців від несанкціонованого використання або витоку чутливих відомостей [1].

Використання хмарних технологій для зберігання та обробки даних стало новим етапом розвитку інформаційних систем для рекрутингу. Хмарні платформи пропонують масштабованість, доступність з будь-якого пристрою та скорочення витрат на інфраструктуру. Водночас, вони дозволяють впроваджувати інтегровані рішення з аналітикою, автоматично оновлюватися та забезпечувати безперебійну роботу.

Запровадження мобільних застосунків та адаптивних вебінтерфейсів стало вимогою часу, адже кандидати та роботодавці все частіше взаємодіють

із системами через смартфони та планшети. Зручність, швидкість і простота у використанні таких інтерфейсів дозволяють суттєво підвищити охоплення аудиторії і підвищити ефективність процесів найму.

Окремо слід згадати про роль технологій блокчейн у підборі персоналу. Ця технологія забезпечує надійне зберігання і верифікацію даних про освіту, кваліфікації і досвід кандидатів, що значно знижує ризики шахрайств і недостовірної інформації. Впровадження блокчейн-рішень у HR-системах майбутнього може сприяти створенню прозорих і довірчих відносин між усіма учасниками процесу підбору.

Таким чином, сучасні інформаційні системи пошуку і підбору персоналу поєднують у собі широкий спектр підходів – від класичних баз даних і ключового пошуку до інтелектуального аналізу, штучного інтелекту та великих даних. Вони інтегруються з широким рядом зовнішніх джерел інформації, забезпечують гнучкість та масштабованість за рахунок хмарних технологій, при цьому гарантують безпеку і зручність взаємодії через адаптивний інтерфейс. Постійний розвиток цих систем спонукає вдосконалювати алгоритми оцінки кандидатів, що у перспективі забезпечить більш точний, швидкий і ефективний підбір персоналу, що адаптований до конкретних організаційних вимог і сучасних тенденцій ринку праці.

1.3 Огляд сучасних вебзастосунків для рекрутингу. Функціональні можливості, переваги й обмеження

У сучасних умовах цифрової трансформації рекрутинг значною мірою перейшов у формат вебзастосунків, що дозволяє оптимізувати пошук і підбір персоналу, підвищити ефективність взаємодії між роботодавцями та кандидатами, а також забезпечити гнучкість процесів адміністрування вакансій. Вебзастосунки для рекрутингу є складними системами, що інтегрують різноманітні функціональні можливості, спрямовані на задоволення різнопланових вимог користувачів і ринку праці загалом. Вони

забезпечують автоматизацію ключових етапів рекрутингу: від публікації вакансій, пошуку резюме, фільтрації кандидатів, організації співбесід до моніторингу ефективності і звітності (рис. 1.1).

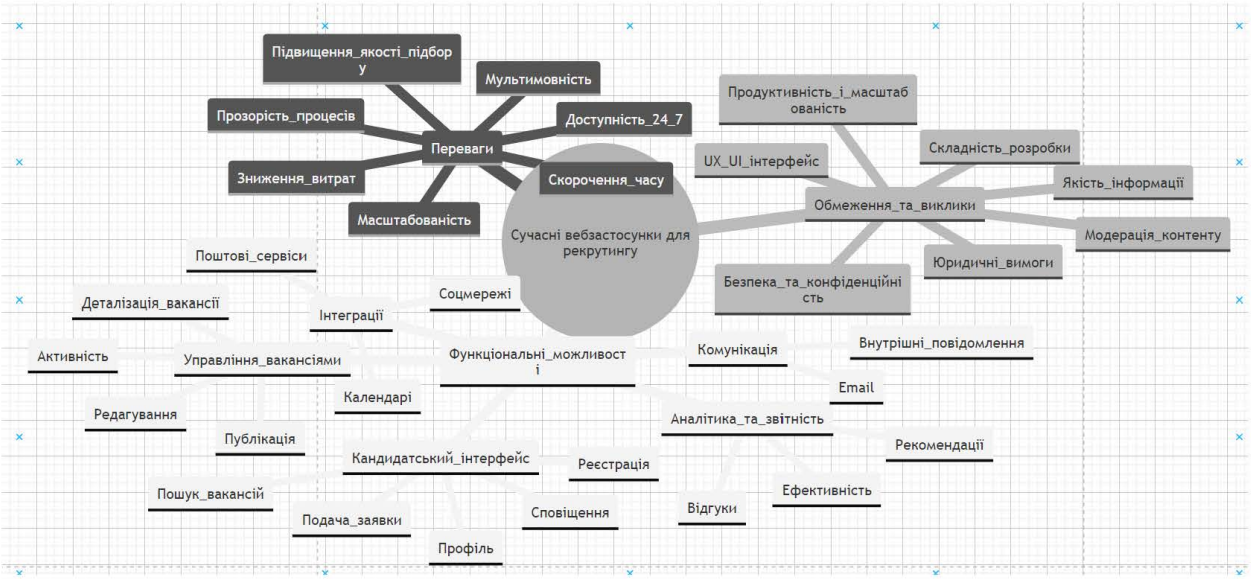

Рисунок 1.1 – Сучасні вебзастосунки для рекрутингу

Однією з головних функцій сучасних вебзастосунків для рекрутингу є функціонал публікації і управління вакансіями, який дозволяє роботодавцям створювати, редагувати, активувати або деактивувати оголошення, вказувати деталізовані параметри вакансії, такі як назва посади, опис обов'язків, вимоги до кандидатів, умови праці, рівень заробітної плати, тип зайнятості та місце роботи. Це надає можливість максимально точно інформувати потенційних кандидатів, що збільшує якість відгуків і скорочує час пошуку оптимального кандидата.

Для кандидатів вебзастосунки пропонують зручні інтерфейси для реєстрації, створення та редагування профілів, завантаження резюме, а також фільтри пошуку вакансій за різними критеріями, включаючи ключові слова, локацію, тип зайнятості, рівень заробітної плати. Часто передбачена можливість підписки на розсилки нових вакансій за заданими параметрами, що дозволяє оперативно отримувати релевантні пропозиції. Більшість

сучасних систем реалізують функцію прямої подачі заявки на вакансію, з опцією листування через внутрішні повідомлення або електронну пошту, що підвищує швидкість комунікації.

Крім базових функціональних можливостей, важливою частиною сучасних платформ для рекрутингу є використання аналітики і звітності, що дозволяє роботодавцям відслідковувати кількість і якість отриманих відгуків, аналізувати ефективність різних каналів і оголошень, а кандидатам отримувати рекомендації та пропозиції на основі їхнього профілю та поведінки у системі. Інтеграція з іншими сервісами, такими як LinkedIn [3], Facebook, Google, а також популярними календарними та поштовими сервісами, розширює функціональні межі застосунків і покращує користувацький досвід.

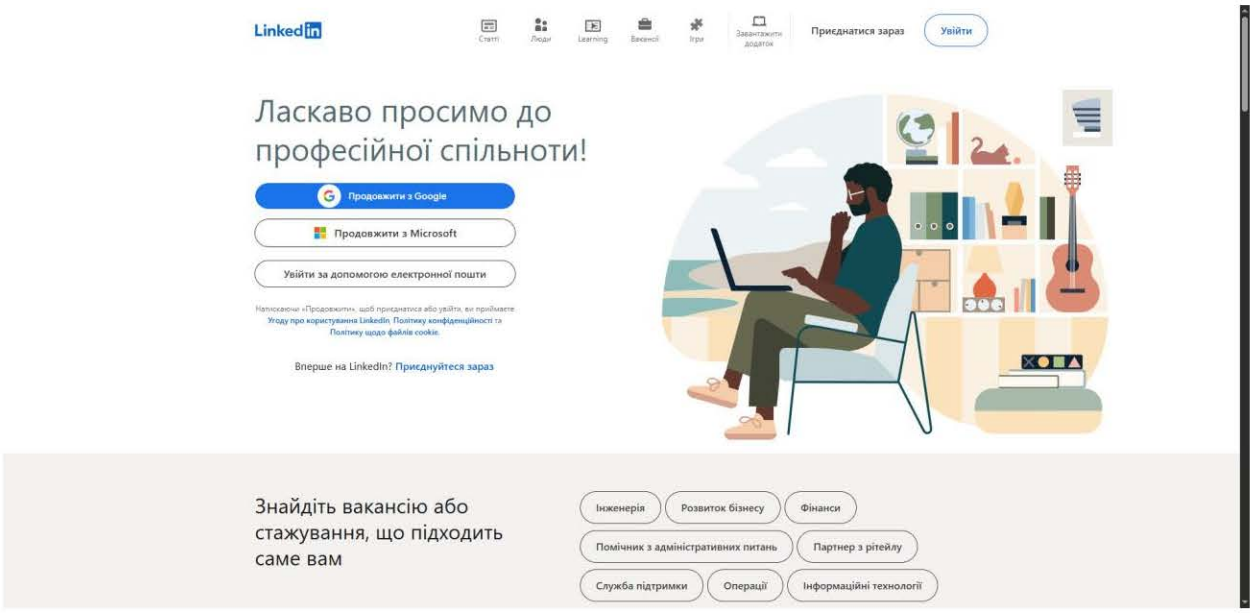

Рисунок 1.2 – Інтерфейс LinkedIn

Ефективність вебзастосунків для рекрутингу залежить також від їхньої архітектури, зокрема, наявності адаптивного дизайну, що підтримує коректне відображення на різних пристроях, зручності навігації, швидкості завантаження та безпеки даних. Безпека є критичним аспектом, адже системи містять персональні дані кандидатів і конфіденційну інформацію, тому

застосовуються сучасні стандарти шифрування, автентифікації і захисту від атак. Щодо переваг таких систем, слід відзначити значне скорочення часу, що витрачається на пошук кандидатів, підвищення якості підбору за рахунок використання фільтрів та автоматизованих алгоритмів співставлення, підвищення прозорості процесів для всіх учасників, а також зниження витрат, пов'язаних із традиційними методами рекрутингу, такими як друк оголошень чи послуги кадрових агентств. Додатково, здатність працювати 24/7, багатомовність, а також можливість швидко масштабувати функціонал під конкретні потреби бізнесу роблять вебзастосунки незамінним інструментом у сучасній HR практиці.

Незважаючи на очевидні переваги, існують і певні обмеження й виклики для сучасних вебзастосунків у сфері рекрутингу. Одним із них є необхідність підтримки високої продуктивності при великому обсязі даних і численних користувачах, що вимагає ефективної роботи з базами даних і оптимізації запитів. Особливо суттєвим є питання масштабованості при зростанні кількості вакансій і користувачів. Інша категорія проблем пов'язана з якістю інформації: некоректно оформлені резюме, недостатньо точні або застарілі оголошення можуть знижувати ефективність системи. Для розв'язання таких питань використовуються додаткові методи валідації і модерації контенту.

Також варто враховувати аспект UX/UI – навіть найкращі функції будуть марними, якщо інтерфейс користувача ускладнює навігацію або вводить в оману. Задля покращення юзабіліті багато платформ впроваджують адаптивні і персонифіковані інтерфейси, інтерактивні підказки, пошук за природньою мовою та голосовий інтерфейс. Але це збільшує складність розробки і потребує глибокої експертизи. Ще одним викликом є інтеграція вебзастосунків з юридично-правовими нормами, у тому числі у сфері захисту персональних даних (GDPR, українське законодавство), що накладає вимоги до управління персональною інформацією, зберігання її в безпечному вигляді та можливості видалення за запитом користувача. Відходи від вимог можуть спричинити великі штрафи та втрату довіри.

1.4 Формулювання мети, завдань, об'єкта та предмета дослідження

Метою даної роботи є створення інтегрованого вебзастосунку, який би ефективно оптимізував процес взаємодії між роботодавцями та кандидатами на ринку праці, сприяючи прискоренню та підвищенню якості підбору персоналу. Передумовою для досягнення цієї мети слугує розробка інтуїтивно зрозумілого, функціонального та безпечного інтерфейсу, який базується на сучасних програмних технологіях із застосуванням фреймворку Flask та системи керування базами даних SQLite. Вебзастосунок, який планується створити, покликаний вирішити ряд актуальних проблем, пов'язаних з ефективністю пошуку вакансій, доступністю інформації про робочі місця, а також зручністю реєстрації та індетифікації користувачів з різними ролями, зокрема кандидатів і роботодавців. У цьому контексті мета дослідження передбачає розроблення функціоналу, що забезпечить можливість публікації та редагування вакансій, здійснення пошуку за ключовими параметрами, такими як посада, локація, зарплатні очікування і тип зайнятості, а також формування персоналізованого інформаційного простору для кожного користувача.

Відповідно до поставленої мети, завдання дослідження охоплюють комплекс теоретичних і практичних аспектів. Зокрема, серед них варто виділити необхідність проведення аналізу наявних інформаційних систем та вебзастосунків для пошуку роботи і оцінку їхніх переваг і недоліків з точки зору функціональності, користувацького досвіду і технологічної реалізації. Далі суспільно-технологічний аналіз існуючих моделей взаємодії між роботодавцем і кандидатом стає базою для упровадження оптимальних архітектурних рішень і технологічних інструментів у рамках розробки власного продукту. Технічна реалізація вимагатиме розробки і налаштування бази даних, структурованої відповідно до специфіки предметної області, де головними об'єктами є користувачі (зокрема їхні профілі, ролі та

аутентифікаційні дані) та вакансії (із детальним описом, умовами і статусами). Крім того, значна увага приділятиметься створенню адаптивного інтерфейсу, який забезпечить зручне використання вебзастосунку на різних пристроях і платформах, а також інтеграції механізмів пошуку і фільтрації даних, що дозволить підвищити точність і швидкість отримання релевантної інформації користувачами системи.

1.5 Висновки до першого розділу

У першому розділі проведено ґрунтовне дослідження предметної області, пов'язаної з автоматизацією процесів пошуку та підбору персоналу. На основі аналізу сучасного стану ринку праці визначено ключові проблеми, з якими стикаються компанії у сфері рекрутингу, серед яких – неефективність традиційних методів відбору кандидатів, інформаційна перевантаженість та високі витрати часу на обробку великого обсягу резюме. Окрему увагу приділено дослідженню новітніх технологій, що застосовуються в сучасних інформаційних системах: штучному інтелекту, машинному навчанню, NLP, Big Data, а також технологіям інтеграції з професійними соціальними мережами. Було проаналізовано функціональні можливості існуючих вебзастосунків для рекрутингу, їх переваги та недоліки, що дозволило окреслити вимоги до майбутньої системи. У результаті сформульовано мету та завдання дослідження, визначено об'єкт і предмет роботи, що заклало основу для технічного проєктування майбутнього вебзастосунку.

РОЗДІЛ 2 АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ ВЕБЗАСТОСУНКУ ДЛЯ ПОШУКУ ТА ПІДБОРУ ПЕРСОНАЛУ

2.1 Мова програмування Python. Можливості та переваги для розробки вебзастосунків

Розгляд мови програмування Python у контексті розробки вебзастосунків вимагає всебічного аналізу її можливостей, особливостей синтаксису, екосистеми та переваг, які вона надає розробникам [4]. Python є однією з найпопулярніших мов програмування у світі, що відзначається високою читаємістю, зрозумілістю синтаксису, а також широкою підтримкою спільноти, що постійно розвиває і підтримує безліч бібліотек, фреймворків та інструментів. Ці характеристики сприяють широкому застосуванню Python у різноманітних галузях, зокрема в розробці вебзастосунків, де він проявляє себе як ефективний і гнучкий інструмент.

Перш за все, слід розглянути структуру мови з точки зору її концептуальних особливостей. Python позиціонується як інтерпретована, високорівнева, багатопарадигмова мова програмування, що підтримує процедурний, об'єктно-орієнтований та функціональний стилі програмування. Завдяки цьому розробник отримує високий ступінь свободи у виборі підходів до реалізації логіки вебзастосунку, що дозволяє адаптувати код під конкретні завдання і підвищувати продуктивність розробки [4]. Синтаксис Python відрізняється простотою і чіткістю, що досягається в тому числі завдяки відсутності зайвих дужок та обов'язковому відступу для позначення блоків коду. Це сприяє легшому розумінню і підтримці коду, що є критично важливим у розробці масштабованих вебзастосунків, де команди розробників можуть налічувати десятки або сотні фахівців.

Окрім чистоти синтаксису, Python забезпечує потужний набір вбудованих структур даних, таких як списки, множини, словники та кортежі, які активно застосовуються для обробки та зберігання інформації. У сфері

веброзробки часто виникає потреба у швидкому зберіганні, пошуку та обробці великих обсягів даних, тому доступність цих ефективних структур даних без необхідності використання зовнішніх бібліотек робить Python вигідним вибором. До того ж, мова підтримує генератори, вирази списків та лямбда-функції, що дозволяють писати компактний і ефективний код для складної логіки. Ключовою перевагою Python у розробці вебзастосунків є наявність розвиненої екосистеми фреймворків, інструментів і бібліотек, розроблених для прискорення процесу створення вебінтерфейсів, керування базами даних, аутентифікації, обробки HTTP-запитів та багатьох інших аспектів. Найвідомішими фреймворками є Django та Flask, кожен з яких має свої унікальні особливості та застосовується відповідно до масштабу та потреб проєкту. Flask, наприклад, забезпечує легкий і максимально гнучкий каркас для створення вебзастосунків, не нав'язуючи жорстких структур, що ідеально підходить для розробки проєктів із унікальними вимогами або невеликим обсягом функціоналу. Django ж пропонує комплексний набір інструментів і "зі коробки" надає стандартні механізми, що пришвидшують розробку великих, складних вебплатформ з високими вимогами до безпеки, масштабованості та підтримуваності коду [4].

У контексті розвитку сучасних вебзастосунків важливою є підтримка Python можливостей асинхронного програмування, що дозволяє ефективно обробляти одночасні запити від численних користувачів, що є критично важливим фактором для проєктів з високою відвідуваністю або специфічними вимогами до продуктивності. Ця властивість, реалізована в сучасних версіях мови, доповнюється підтримкою популярних асинхронних вебфреймворків типу FastAPI, які базуються на новітніх стандартах Python і дозволяють створювати високозавантажені API та вебсервіси з одночасно простою та гнучкою у використанні архітектурою.

Окремо варто зазначити, що Python має інтегровану підтримку різноманітних систем управління базами даних, що відіграє центральну роль у розробці вебзастосунків з урахуванням необхідності надійного зберігання,

маніпуляції та пошуку інформації. Через наявність численних ORM (Object-Relational Mapping) бібліотек, таких як SQLAlchemy чи Django ORM, розробники можуть працювати з базою даних в об'єктно-орієнтованому стилі, що значно спрощує код і підвищує його підтримуваність без компромісів у продуктивності. Такі підходи дозволяють уникнути прямого написання складних SQL-запитів, а натомість працювати із сутностями високого рівня, що ефективно корелює з бізнес-логікою застосунку.

Безпека вебзастосунків, розроблених на Python, є ще одним вагомим фактором, що обумовлює вибір цієї мови. Фреймворки, що використовуються у створенні вебрішень, містять вбудовані механізми захисту від типових вразливостей, таких як SQL-ін'єкції, міжсайтове скриптування (XSS), CSRF і підробка запитів. Крім того, Python-середовище дозволяє легко інтегрувати додаткові засоби аутентифікації та авторизації, використовувати сучасні технології шифрування даних, а також аудит безпеки, що у підсумку сприяє створенню надійних і безпечних вебсервісів [4]. Не менш суттєвим аспектом є мультиплатформність та портативність коду, що забезпечує можливість розгортання Python вебзастосунків практично у будь-якому середовищі, включаючи популярні сервери, хмарні платформи і контейнери, такі як Docker. Це дозволяє інженерам з легкістю створювати інтегровані рішення, масштабувати їх за потребами, забезпечуючи високу доступність і гнучкість у розгортанні.

Гнучкість Python проявляється також у широких можливостях інтеграції з іншими мовами та інструментами. У веброзробці часто потрібна робота з різними API, сторонніми сервісами, а також виконання складних обчислень або обробка даних за межами основного функціоналу застосунку. Завдяки численним бібліотекам HTTP-клієнтів, JSON-парсерів, а також можливості виклику нативного коду на C/C++, Python дозволяє створювати комплексні модулі, які працюють з різноманітними технологіями та стандартами.

Крім того, активне та широка спільнота користувачів і розробників Python забезпечує постійний розвиток екосистеми, доступність великих

обсягів документації, навчальних матеріалів, а також допомоги у вирішенні практичних завдань у сфері веброзробки. Такі ресурси сприяють як прискоренню процесу освоєння мови новачками, так і підтримці на високому рівні професіоналів, що працюють над складними проєктами.

Не варто оминути увагою той факт, що Python активно застосовується не лише для серверної частини вебзастосунків, а й для розробки API, побудови сервісів машинного навчання та аналітики, що інтегруються до вебплатформ. Це дозволяє створювати унікальні продукти з інтегрованою технологією обробки даних і автоматизації, що відкриває нові горизонти для вебзастосунків, підвищує їх цінність для кінцевих користувачів та бізнесу.

Підсумовуючи, можливості мови програмування Python у розробці вебзастосунків базуються на її простому, інтуїтивно зрозумілому синтаксисі, широкій підтримці різних стилів програмування, наявності потужної екосистеми фреймворків та бібліотек, що сприяє швидкій та надійній розробці інноваційних вебплатформ [4]. Висока продуктивність, комплексна підтримка безпеки, мультиплатформність та інтеграційні можливості роблять Python ідеальним вибором для розробників, котрі прагнуть створювати сучасні, масштабовані та зручні у користуванні вебзастосунки, здатні задовольняти найрізноманітніші вимоги ринку і користувачів. Завдяки цим факторам мова Python приваблює як початківців, так і досвідчених розробників, створюючи міцний фундамент для подальшого розвитку та інновацій у сфері веброзробки.

2.2 Вебфреймворк Flask. Архітектура, особливості та застосування у створенні вебзастосунків

Архітектура Flask спирається на концепцію мікрофреймворку, що визначає його мінімалістський характер та відсутність надмірної абстракції. На відміну від монолітних фреймворків, які пропонують комплексний набір інтегрованих інструментів, Flask забезпечує базовий каркас для створення вебзастосунків із можливістю розширення функціональності за допомогою

сторонніх бібліотек, так званих розширень (extensions). Серцевиною Flask є клас Flask, екземпляр якого представляє вебдодаток і відповідає за обробку HTTP-запитів, маршрутизацію, контекст застосунку, управління сесіями та іншими ключовими аспектами життєвого циклу вебсервісу [5]. Цей клас базується на Werkzeug – WSGI-утиліті бібліотеці, яка виконує роль серверного інструментарію, що слугує головною точкою взаємодії між вебсервером і самим застосунком.

Flask належить до фреймворків, які надають розробнику максимальний контроль над структурою проєкту, не нав'язуючи жорсткі стандарті внутрішньої організації. Цей аспект є суттєвим при розробці індивідуалізованих рішень або швидкій побудові прототипів. В основі Flask лежить принцип «розробляй з мінімальними обмеженнями», що дозволяє реалізовувати проєктну архітектуру у відповідності з вимогами конкретного застосунку. Кожен HTTP-запит обробляється через маршрути, які асоціюються з функціями-представленнями (view functions), що повертають відповіді, представлені у вигляді HTML-сторінок, JSON-об'єктів або інших форматів [6]. Маршрутизація здійснюється за допомогою простого і гнучкого механізму декораторів, що додає зручності у визначенні логіки обробки запитів.

Особливістю Flask є вбудована підтримка шаблонізатора Jinja2, який забезпечує потужні можливості поєднання статичного HTML з динамічним контентом. Jinja2 пропонує мову шаблонів, що включає умовні конструкції, цикли, макроси та фільтри, що суттєво полегшує організацію представлення даних на рівні користувацького інтерфейсу. Це дозволяє розробникам ефективно розділяти логіку та візуальну частину, сприяє підтримці чистого коду і гнучкому налаштуванню вигляду застосунку. Фреймворк також підтримує механізми управління станом сесії, дозволяючи зберігати інформацію про користувача між запитами, що критично для реалізації функціоналу автентифікації, збереження налаштувань користувача та інших персоналізованих рішень. Сесії у Flask обладнані захистом від підробки даних

за допомогою підпису секретним ключем, що підвищує безпеку застосунку [7].

Питання інтеграції з базами даних в Flask вирішується гнучко, оскільки він не має вбудованих ORM, проте легко поєднується з такими популярними бібліотеками, як SQLAlchemy чи Peewee. Цей підхід залишає розробнику свободу вибору та налаштування найбільш оптимальної для конкретного проєкту системи управління даними, що дозволяє ефективно працювати із різними типами баз даних, включно з реляційними та NoSQL рішеннями. У контексті використання з SQLite3, що є частим вибором для прототипів та невеликих застосунків, Flask забезпечує надійне з'єднання через простий API, що інтегрується із загальною архитектурою програми.

Однією із визначних рис Flask є простота налаштування і запуску застосунку. Розробник може швидко створити базову структуру проєкту з мінімальним об'ємом коду, що робить цей інструмент особливо привабливим для навчання та стартапів. Крім того, Flask включає в себе механізми дебагінгу та перезавантаження серверу у режимі розробки, що пришвидшує цикл тестування і корекції коду. За допомогою Flask-CLI доступна інтеграція із командним рядком для виконання типових операцій, таких як ініціалізація бази даних чи запуск тестів [8].

Безпека є одним із базових вимог, яку задовольняє Flask через використання перевірених інструментів для захисту від атак типу CSRF, XSS та інші. Для автентифікації та авторизації Flask пропонує розширення Flask-Login, яке інтегрується з уже існуючим каркасом, надаючи готові механізми для керування сесіями, захисту маршрутів і реалізації функціоналу користувацьких ролей. Це дозволяє створювати багаторівневі системи доступу, які є невід'ємною складовою сучасних вебзастосунків, що працюють із чутливою інформацією користувачів. Важливим аспектом є масштабованість та розширюваність, забезпечувані Flask. Розробники можуть додавати модулі для роботи з REST API, підтримки WebSocket, інтеграції з фронтенд-бібліотеками та фреймворками за допомогою доступного набору

плагінів та розширень. Архітектура сприяє впровадженню таких технологій, що роблять вебзастосунок не лише інтуїтивно зрозумілим, а й готовим для інтерактивної взаємодії з користувачем в режимі реального часу.

Щодо практичного застосування, Flask знайшов широке визнання серед розробників завдяки власній здатності поєднувати легкість і потужність. Його використовують для створення динамічних сайтів, корпоративних CRM-систем, систем управління контентом, а також для розробки RESTful API, необхідних в сучасних мобільних та вебдодатках. Вебзастосунок для пошуку та підбору персоналу, розроблений із використанням Flask, вигідно відрізняється швидкістю розгортання і простотою подальшої підтримки, що позначається на загальній ефективності проєкту [5, 6].

Загалом, архітектура Flask, що базується на підходах WSGI та мікрофреймворка, особливий акцент на розширюваність через розширення, підтримку шаблонізатора Jinja2, а також гнучкість у роботі з базами даних і користувацькими сесіями, створює комфортне середовище для реалізації ширшого спектру завдань у сфері веброзробки. Використання Flask у створенні вебзастосунків сприяє швидкому переходу від концепції до впровадження, дозволяє утримувати код чистим та масштабованим, що забезпечує високу якість кінцевого продукту та сприяє його успішному застосуванню в реальних бізнес-процесах. Такий підхід виступає фундаментом для оптимального балансу між гнучкістю розробки і надійністю функціоналу, що важливо для вебплатформ сучасного типу, орієнтованих на пошук та підбір персоналу.

2.3 Система управління базами даних SQLite3. Структура, переваги та інтеграція з Flask

SQLite3 є вбудованою системою управління реляційними базами даних, яка характеризується компактністю, простотою у використанні та повною відсутністю необхідності налаштування сервера. Головною особливістю

SQLite3 є збереження всієї бази даних у вигляді одного файлу на диску, що суттєво спрощує розгортання програмних продуктів і знижує витрати на адміністрування [9]. Такий підхід дозволяє розробникам уникати складнощів із підтримкою зовнішніх серверів СУБД та значно прискорює процес розробки та тестування. Структурно SQLite3 підтримує реляційну модель даних, даючи змогу створювати таблиці з чітко визначеними типами даних, індексами, зв'язками між таблицями, а також забезпечує повний набір стандартних SQL-запитів, включаючи транзакції, що гарантує цілісність даних. У межах розробки вебзастосунку, призначеного для пошуку та підбору персоналу, така структурна модель є надзвичайно важливою, оскільки вона дозволяє логічно розділити сутності доменної області, наприклад, користувачів, вакансії та їх параметри, що сприяє підтримці високої якості даних і полегшує подальший розвиток системи.

Щодо переваг, які роблять SQLite3 привабливою для малого і середнього рівня застосунків, варто відзначити не лише відсутність необхідності серверного налаштування, а й високу продуктивність при роботі з файлами маленького та середнього розміру, низьке споживання ресурсів, легкість у резервному копіюванні та відновленні. Крім того, SQLite3 має відкритий вихідний код, що сприяє широкому використанню в академічних та комерційних проєктах [10]. Вона підтримує стандартні угоди, такі як ACID (атомарність, узгодженість, ізоляція, стійкість), що вкрай важливо для забезпечення надійності вебзастосунку, який працює з приватною інформацією користувачів та потенційно конфіденційними даними рекрутингу.

Важливим аспектом є й те, що SQLite3 інтегрується з мовою програмування Python за допомогою вбудованої бібліотеки sqlite3, що значно полегшує її застосування у Flask-застосунку. Flask, як мікрофреймворк, надає достатньо гнучкості для побудови як простих, так і складних вебсервісів, і у поєднанні з SQLite3 дає можливість безпосередньо взаємодіяти з базою даних у межах легкої архітектури без надмірних накладних витрат. Задля цього

застосовують патерн підключення до бази даних через контекст додатку, що реалізовано у вигляді функції отримання з'єднання з базою даних, яка зберігається у глобальному об'єкті Flask. Це дозволяє ефективно управляти ресурсами, запобігаючи непотрібним відкриттям і закриттям з'єднань [11].

У рамках вебзастосунку структура бази даних у SQLite3 продумана таким чином, що розмежовує основні сутності за таблицями «users» та «jobs». Таблиця користувачів містить інформацію про зареєстрованих кандидатів та роботодавців, включаючи унікальні ідентифікатори, логіни, паролі, електронні адреси, ролі в системі та додаткові атрибути, такі як назва компанії для роботодавців. Таблиця вакансій розроблена для зберігання деталей позиції – назви, опису, місця розташування, типу роботи, діапазону зарплат, а також пов'язана з роботодавцем через зовнішній ключ. Така структура забезпечує не лише логічну цілісність, а і дає змогу виконувати складні SQL-запити на пошук за ключовими словами, фільтрацію за локацією, типом зайнятості та іншими параметрами.

Крім того, можливості SQLite3 дозволили застосувати механізми контролю статусу вакансій (активна чи ні) та автоматичне фіксування дат створення записів, що додає до системи зручності та гнучкості в управлінні інформацією. Газування цих атрибутів на рівні бази зменшує рівень логіки на стороні застосунку, що позитивно впливає на надійність і швидкодію. У контексті безпеки, використання SQLite3 поєднується з валідацією та хешуванням паролів на стороні застосунку; для цього застосовується бібліотека werkzeug.security [12]. Така практика забезпечує, що навіть при випадковому доступі до файлу бази даних, інформація про облікові записи залишатиметься захищеною. Робота з базою SQL реалізована за допомогою параметризованих запитів, що мінімізує ризик ін'єкцій SQL, що потенційно корисно для підтримання високих стандартів інформаційної безпеки.

Інтеграція SQLite3 із Flask у цьому вебзастосунку реалізується у вигляді створення проміжного рівня доступу до бази через функції, які ініціалізують з'єднання у глобальному контексті, автоматично його закривають при

завершенні роботи запиту, а також виконують інструкції ініціалізації бази даних виключно на вимогу. Такий підхід запобігає зайвому навантаженню і забезпечує коректне ведення транзакцій. Використання Flask-Login разом із SQLite3 підвищує рівень абстракції, надаючи зручні методи для аутентифікації та авторизації користувачів безпосередньо з таблиці users, враховуючи при цьому різні ролі системи.

Ще однією важливою перевагою використання SQLite3 у вебзастосунку є підтримка незалежності від середовища розгортання. Оскільки база представлена єдиним файлом, вона легко переноситься між різними ОС і хостингами, що зручно для розробки, тестування і виробничої експлуатації. При цьому інтеграція з Flask не вимагає додаткових специфічних драйверів чи системних налаштувань, що дуже спрощує цикл розробки і реалізації оновлень. Використання SQLite3 має також свої обмеження, з якими слід рахуватися при масштабуванні проєкту. Вона не призначена для високонавантажених систем, де потрібна одночасна робота великої кількості користувачів із високою швидкістю записів і транзакцій. Однак для задачі, пов'язаної з персональним пошуком роботи та публікацією вакансій, де обсяг даних і інтенсивність операцій не виходять за межі малих чи середніх навантажень, SQLite3 є оптимальним рішенням [9-12].

В цілому, вибір архітектури бази даних на базі SQLite3 у поєднанні з Flask, з одного боку, дає легкість і гнучкість розробки, а з іншого – забезпечує надійність, структурованість та ефективність обробки даних, що є критично важливим для функціональності платформи, яка обслуговує дві категорії користувачів – кандидатів і роботодавців. Реалізація зв'язків між сутностями, підтримка транзакцій та контроль цілісності даних створюють фундамент для подальшої масштабованості та розвитку застосунку за мінімальних технічних витрат та з максимальним комфортом для користувачів.

2.4 Бібліотека Flask-Login. Механізми аутентифікації та управління сесіями користувачів

Flask-Login розроблена з метою спрощення процесу додавання функціоналу логіна та управління сеансами користувачів, будучи при цьому максимально гнучкою та адаптивною під різноманітні конфігурації вебзастосунків [13]. Бібліотека надає можливість організувати аутентифікацію шляхом керування користувацькими сесіями без необхідності повторного введення облікових даних при кожному запиті, що реалізується за допомогою застосування кукісів, де зберігається унікальний ідентифікатор сесії. Наступні концепції є фундаментальними для роботи Flask-Login.

По-перше, бібліотека вводить абстракцію моделі користувача, яка вимагає реалізації певних методів та властивостей, через які застосунок інтерпретує користувача в контексті аутентифікації. Користувач у Flask-Login представлений класом, який імплементує інтерфейс UserMixin, забезпечуючи методи, такі як is_authenticated, is_active, is_anonymous та get_id, що дають змогу системі валідувати стан користувача і однозначно ідентифікувати його під час роботи з сесією. Ця модель взаємодіє з базою даних, де зберігаються облікові дані користувачів, для отримання необхідної інформації при аутентифікації. По-друге, ключовим механізмом взаємодії з аутентифікаційним сервером є функція user_loader – спеціальний декоратор, що реєструє зворотний виклик, який отримує унікальний ідентифікатор користувача та повертає відповідний об’єкт користувача [13]. Це реалізує процес відновлення стану користувача зі сесії після повторного звернення до сервера, що забезпечує безперервність досвіду користування вебзастосунком без необхідності повторного введення паролів. Функція user_loader виконує запит до сховища даних, витягуючи інформацію про користувача, наприклад, його ідентифікатор, ім’я, роль, та інші атрибути, необхідні для подальшої роботи.

Додатково Flask-Login надає інструменти для організації процесу реєстрації та входу, а також механізми для виходу користувачів із системи (logout), що дає змогу здійснювати контрольований доступ до ресурсів. При вході користувача до системи виконується функція login_user, яка встановлює відповідні флеш-повідомлення і зберігає ідентифікатор користувача в кукі браузеру, що дозволяє відстежувати сесію. Метод login_required служить декоратором для маршрутів, доступ до яких повинен бути обмежений лише автентифікованими користувачами; таким чином, він автоматично перенаправляє неавторизованих користувачів на сторінку входу або інший ресурс, визначений у конфігурації.

Функціонал управління сесіями у Flask-Login будується на основі збереження стану користувача на стороні сервера з ідентифікатором у клієнтських куках. При кожному запиті сервер перевіряє цей ідентифікатор через функцію завантаження користувача, що дозволяє автентифікованому користувачу залишатися у системі доти, доки діє сесія або не виконано вихід. Враховуючи, що стандартні характеристики HTTP не зберігають інформацію між запитами, такі сесії є фундаментальним механізмом забезпечення безперервного користувацького досвіду. Flask-Login координує цю взаємодію, при цьому він інтегрується із системами куками та, за можливості, з іншими механізмами управління сесіями, забезпечуючи також можливість зазначення часу життя сесії, використання remember-me функціоналу, що дає змогу користувачам залишатися залогіненими тривалий час [13].

Значне місце у Flask-Login займає система повідомлень, які відображаються користувачам в процесі аутентифікації: це як звичайні успішні повідомлення про вхід та реєстрацію, так і повідомлення про помилки у випадку неправильно введених даних, відсутності доступу чи інших проблем. Вони формуються за допомогою функції flash, що дозволяє динамічно інформувати користувача, сприяючи більшій дружності до інтерфейсу.

З технічної точки зору, бібліотека легко інтегрується з іншими компонентами Flask-застосунку, використовуючи стандартизовані інтерфейси та декоратори. Це сприяє збереженню чистоти коду та забезпечує можливість масштабування системи зростанням її складності. Flask-Login не передбачає конкретного способу зберігання паролів чи поведінки логіки користувача, залишаючи розробнику свободу в обранні методів шифрування, типів баз даних і власних моделей даних, що робить його універсальним інструментом для різних завдань. Крім того, бібліотека дає змогу працювати з ролями користувачів, що легко інтегрується у систему контролю доступу, дозволяючи обмежувати функціональні можливості залежно від типу користувача – кандидата або роботодавця. Така сегментація базується на інформації, що міститься у об'єкті користувача, забезпечує гнучкість у реалізації бізнес-логіки та сприяє підвищенню безпеки, оскільки доступ до певних сторінок чи дій на сайті контролюється на рівні сесії.

2.5 Бібліотека Werkzeug. Інструменти для безпечного хешування паролів та обробки HTTP

Сучасна розробка вебзастосунків нерозривно пов'язана з питаннями забезпечення безпеки та ефективного управління HTTP-запитами й відповідями. Одним із ключових інструментів у цій галузі для розробників на Python є бібліотека Werkzeug, що є фундаментальним нижнім рівнем для фреймворку Flask, а також може використовуватися автономно. Werkzeug забезпечує набір компонентів, які полегшують обробку даних HTTP-протоколу, а також реалізує механізми безпечного хешування паролів, що є критично важливим аспектом в системах аутентифікації користувачів. Розглянемо детальніше призначення, архітектуру та функціональні можливості Werkzeug, зокрема щодо безпеки паролів та обробки HTTP [14, 15].

Werkzeug, що у перекладі з німецької означає «інструмент», є бібліотекою низького рівня, що виступає HTTP-утилітою для Python, надаючи гнучкі й розширювані засоби для створення і обробки HTTP-запитів і відповідей. Вона конфігурується як робочий шар, що абстрагує складнощі роботи з протоколом HTTP, дозволяючи розробникам концентрувати увагу на бізнес-логіці застосунку. З моменту свого створення Werkzeug посідає вагоме місце в екосистемі Flask, оскільки забезпечує багатий API для обробки маршрутизації, сесій, куків, обробки форм, та інших важливих вебфункцій.

Коли мова заходить про безпеку у вебзастосунках, одним із фундаментальних сценаріїв є надійне зберігання паролів користувачів. Використання необроблених паролів у базі даних або їх зберігання у вигляді простого тексту є серйозною вразливістю, що загрожує компрометацією даних при несанкціонованому доступі. Вирішенням цієї проблеми являється застосування криптографічних хеш-функцій із сольовими значеннями і алгоритмами, стійкими до атак перебору чи колізій. У цьому контексті Werkzeug пропонує механізми хешування паролів через інтеграцію з пакетом Werkzeug.security, який реалізує сучасні та перевірені алгоритми, такі як PBKDF2 (Password-Based Key Derivation Function 2), що відповідають кращим практикам безпеки [14, 16].

Реалізація захисту паролів у Werkzeug спрощена через функції generate_password_hash та check_password_hash. Перша функція відповідає за генерацію хешу із заданого пароля, автоматично обираючи надійний алгоритм та генеруючи випадковий сольовий компонент, який унеможливлює застосування до хешу відкритих таблиць (rainbow tables). Друга функція здійснює безпечну перевірку відповідності хешу пароля, що вводиться користувачем, з раніше згенерованим значенням, не розкриваючи сам хеш чи пароль. Таким чином, середовище Flask, використовуючи Werkzeug, отримує простий у застосуванні, але потужний інструментарій для аутентифікації, що максимально забезпечує захист від стандартних видів атак. Крім інструментів безпечного хешування, Werkzeug виконує обробку HTTP-запитів і відповідає

за формування об'єктів request і response, які є центральними у сучасному веброзробці. Через ці об'єкти відбувається доступ до всіх характеристик HTTP-запиту, включно з методами запиту (GET, POST, PUT, DELETE), заголовками, параметрами, тілом запиту, куками, сесіями, а також формування відповіді з потрібним статусом і заголовками. Werkzeug призначена насамперед для роботи з WSGI (Web Server Gateway Interface) протоколом, який є стандартом у Python для взаємодії вебсерверів і застосунків.

Особливістю бібліотеки є її модульна архітектура та високий рівень абстракції, що дозволяє розробникам створювати власні Werkzeug-модулі для специфічних потреб, таких як обробка завантаження файлів, підтримка WebSocket, або кастомна маршрутизація. Werkzeug також підтримує обробку різних статусів HTTP-відповідей, керування куками і сесіями, що у сумі забезпечує комплексну платформу для розробки веборієнтованих застосунків. З точки зору продуктивності, Werkzeug оптимізована для збереження мінімальних затримок в обробці запитів, використовуючи потужну систему кешування і суттєво ефективно маніпулюючи потоками даних. Завдяки легкості інтеграції з Flask, бібліотека лишається одним із найпопулярніших рішень для Python-розробників, які створюють вебзастосунки будь-якої складності [16].

У практичній реалізації у проєкті Werkzeug застосовується, зокрема, для створення надійної системи реєстрації та входу користувачів. При реєстрації пароль користувача піддається хешуванню за допомогою функції generate_password_hash, що гарантує, що в базу даних потрапляють лише криптографічно захищені значення. Під час входу введений пароль порівнюється із заздалегідь збереженим хешем через check_password_hash, що унеможливлює крадіжку паролів навіть у випадку витоку бази даних. Використання бібліотеки також забезпечує збереження сесій та обробку HTTP-запитів без необхідності реалізовувати ці механізми вручну, підвищуючи тим самим безпеку і надійність застосунку. Безпека у сучасних вебзастосунках безпосередньо залежить від надійності методів обробки даних

користувача, а особливо паролів. Прикладами вразливостей є атаки типу brute force, rainbow table, а також SQL-ін'єкції та міжсайтові скриптові атаки (XSS). Використання Werkzeug як рекомендованого інструментарію не тільки сприяє мінімізації шансів успішної атаки на аутентифікаційний механізм, а й дозволяє стандартизувати і уніфікувати підхід до обробки HTTP, що в цілому підвищує безпеку і стабільність вебзастосунку.

Розглядаючи архітектурні особливості, Werkzeug орієнтується на WSGI сервери, підтримуючи стандарти Python Web, і дозволяє легко проводити відладку та тестування окремих компонентів HTTP-протоколу безпосередньо в середовищі розробки. Це суттєво знижує час розробки та зменшує кількість помилок, пов'язаних із нестандартною або некоректною обробкою запитів. Бібліотека надає зручні утиліти для роботи з URL, маршрутизації, а також інструменти для безпечної роботи з куками, що є життєво важливим для підтримання стану сесії та впровадження механізмів авторизації в застосунку.

Також слід відзначити, що Werkzeug активно підтримується спільнотою розробників, регулярно оновлюється та покращується з урахуванням сучасних вимог безпеки та новітніх вебтехнологій. На сьогодні вона залишається стандартним компонентом у стеку Flask, і її інтеграція є найоптимальнішим вибором для проєктів, де безпека аутентифікації і гнучке управління HTTP - пріоритетні [14-16].

Використання Werkzeug у проєкті означає, що система аутентифікації не лише відповідає стандартам індустрії по захисту даних користувачів, але й ідеально інтегрована в структуру вебзастосунку, забезпечуючи надійну взаємодію між клієнтом і сервером через HTTP. Бібліотека дозволяє ефективно опрацьовувати різноманітні HTTP методи, обробляти форми, підтримувати стан авторизації та враховувати безліч нюансів мережевої взаємодії, що суттєво підвищує якість кінцевого продукту.

2.6 Висновки до другого розділу

Другий розділ присвячено аналізу та обґрунтуванню вибору засобів реалізації вебзастосунку для пошуку та підбору персоналу. Було розглянуто основні можливості мови програмування Python у сфері веброзробки, її синтаксичні переваги, багата екосистема бібліотек і підтримка різних парадигм програмування. Вибір фреймворку Flask обґрунтовано його гнучкістю, простотою у використанні та здатністю до швидкої розробки прототипів. Детально досліджено особливості архітектури Flask, його взаємодію з шаблонізатором Jinja2, підтримку маршрутизації та механізми роботи із сесіями. Було проаналізовано інтеграцію з системою керування базами даних SQLite3, яка виявилася ефективним рішенням для локального зберігання даних з огляду на простоту розгортання та високий рівень надійності. Розглянуто засоби аутентифікації користувачів, реалізовані за допомогою бібліотеки Flask-Login, а також методи безпечного зберігання паролів на основі інструментів Werkzeug. Таким чином, було сформовано технологічну основу майбутнього вебзастосунку, що забезпечує його функціональність, безпеку та масштабованість.

РОЗДІЛ 3 РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ПОШУКУ ТА ПІДБОРУ ПЕРСОНАЛУ

3.1 Аналіз вимог та загальна архітектура вебзастосунку

У межах розробки вебзастосунку для пошуку та підбору персоналу необхідно перш за все здійснити детальний аналіз вимог, що передбачає визначення функціональних та нефункціональних характеристик, які будуть закладені у систему (рис. 3.1). Вебзастосунок призначений для зручного і ефективного поєднання кандидатів, які шукають роботу, та роботодавців, що пропонують вакансії. Основна мета системи полягає у створенні інтуїтивного інтерфейсу для користувачів різних ролей, забезпеченні безпечної автентифікації, можливості публікації та пошуку вакансій, а також відгуку кандидатів на пропозиції.

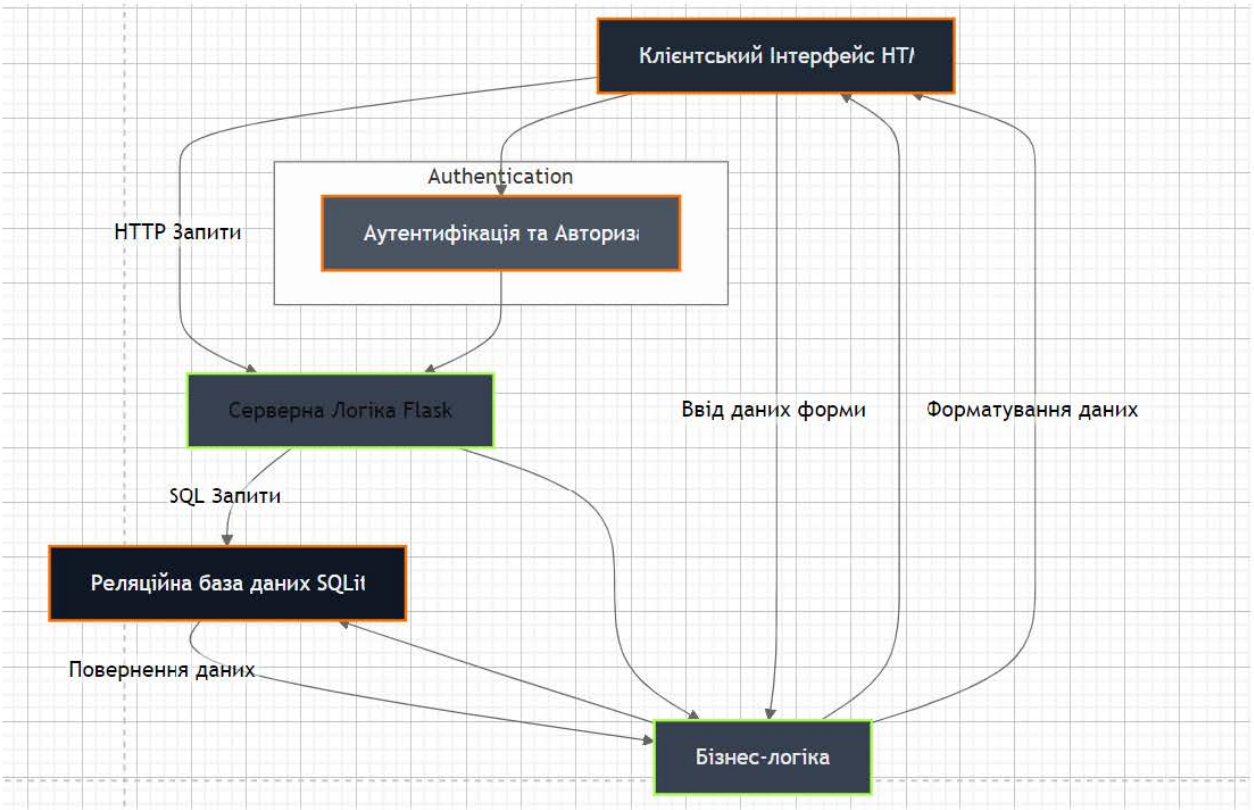

Рисунок 3.1 – Діаграма аналізу вимог

Виходячи з цього, функціональна модель передбачає наявність двох основних типів користувачів – кандидатів та роботодавців, кожен з яких має різний набір привілеїв і взаємодіє із застосунком через відповідні інтерфейси. Кандидати можуть реєструватись, увійти до системи, переглядати вакансії з функціоналом пошуку за ключовими словами й локацією, переглядати деталі вакансій та надсилати відгуки. Роботодавці, крім базових функцій, мають додатковий функціонал для публікації нових вакансій, управління ними, а також перегляду створених оголошень на власній панелі керування.

На рівні нефункціональних вимог система повинна відповідати сучасним стандартам безпеки, гарантуючи захищеність персональних даних та надійну обробку облікових записів користувачів. Також важливо забезпечити масштабованість, підтримку великих обсягів даних і користувачів, а інтерфейс має бути адаптивним, щоб коректно функціонувати на різноманітних пристроях. Враховуючи специфіку завдання, обрано архітектурний стиль клієнт-сервер із реалізацією через вебфреймворк Flask, який забезпечує гнучкість і легкість впровадження логіки бекенду. Для зберігання даних використовується реляційна база даних SQLite3, що підходить для застосуноків середнього масштабу з невеликою навантаженістю при одночасній роботі.

Загальна архітектура вебзастосунку складається з декількох шарів із розмежуванням відповідальностей. Найнижчий рівень становить база даних, у якій зберігаються користувацькі облікові записи, інформація про вакансії, рольові атрибути, а також додаткові метадані, такі як статус активності вакансії та часові позначки створення. Посередницький рівень – це серверна частина, реалізована на Flask, яка виконує обробку клієнтських запитів, взаємодіє з базою даних, проводить аутентифікацію та авторизацію, формує динамічні відповіді та маршрути. Цей рівень також реалізує контролери, які відповідають за бізнес-логіку: перевірку заповнення форм, обробку помилок, створення нових вакансій, пошук, фільтрацію та інші операції, необхідні для коректної роботи сервісу.

Верхній рівень архітектури – клієнтська частина, представлена у вигляді HTML-шаблонів із використанням Jinja2, що надає можливість динамічного формування вебсторінок відповідно до даних, отриманих із бекенду. Візуальне оформлення забезпечує каскадний стиль CSS із фокусом на темний помаранчево-салатовий дизайн, що реалізує унікальний фірмовий стиль проєкту. Клієнтська частина комунікує з сервером переважно через HTTP через стандартні GET- та POST-запити, при цьому дані користувача вводяться через форми із відповідною валідацією як на клієнтській, так і на серверній стороні.

Особливість застосунку полягає у застосуванні Flask-Login для керування сесіями користувачів, що підвищує рівень безпеки й зручності авторизації. Таким чином, зв'язок між клієнтом і сервером структурований навколо маршрутів, які обробляють різні типи запитів: від відкриття головної сторінки зі списком вакансій до приватних панелей користувачів і публікації нових позицій роботодавцями. Зокрема, механізм пошуку реалізований за допомогою параметричних SQL-запитів, що дозволяє гнучко відфільтрувати вакансії за ключовими словами і географією.

3.2 Проєктування бази даних. Структура таблиць users та jobs, сполучення таблиць і зв'язки

Проєктування бази даних починається з визначення необхідних сутностей і атрибутів, які відповідають вимогам предметної області. У контексті системи пошуку і підбору персоналу первинними сутностями виступають користувачі (users) та вакансії (jobs). Таблиця users містить дані про всіх зареєстрованих користувачів, які можуть бути поділені на дві категорії: кандидати та роботодавці. Для кожного користувача зберігаються унікальний ідентифікатор, ім'я користувача, хешований пароль, електронна пошта, роль користувача, а також додатковий атрибут «назва компанії» для роботодавців, що дозволяє ідентифікувати організацію, яку вони

представляють. Атрибут role має тип тексту та визначає, чи є користувач кандидатом, чи роботодавцем. Для забезпечення унікальності ідентифікації користувачів і виключення дублювань, в таблиці встановлено обмеження унікальності для полів username та email. Поле created_at фіксує час створення запису і автоматично заповнюється поточним таймстампом у момент додавання нового рядка (табл. 3.1).

Таблиця 3.1

Структура таблиць баз даних

Таблиця	Поле	Тип даних	Опис
users	id	INTEGER PRIMARY KEY AUTOINCREMENT	Унікальний ідентифікатор користувача
users	username	TEXT UNIQUE NOT NULL	Ім’я користувача, унікальне
users	password	TEXT NOT NULL	Хешований пароль
users	email	TEXT UNIQUE NOT NULL	Електронна пошта, унікальна
users	role	TEXT NOT NULL DEFAULT candidate	Роль користувача: кандидат або роботодавець
users	company_ name	TEXT	Назва компанії для роботодавців
users	created_at	TIMESTAMP NOT NULL DEFAULT CURRENT_TIMES TAMP	Дата і час реєстрації користувача

Продовження таблиці 3.1

jobs	id	INTEGER PRIMARY KEY AUTOINCREMENT	Унікальний ідентифікатор вакансії
jobs	employer_id	INTEGER NOT NULL	Ідентифікатор роботодавця (зв'язок з users.id)
jobs	title	TEXT NOT NULL	Назва посади вакансії
jobs	description	TEXT NOT NULL	Опис вакансії
jobs	location	TEXT	Місцезнаходження вакансії
jobs	salary_range	TEXT	Діапазон заробітної плати
jobs	job_type	TEXT	Тип зайнятості (повна, часткова і т.д.)
jobs	company _name _display	TEXT NOT NULL	Назва компанії для відображення у вакансії
jobs	posted_date	TIMESTAMP NOT NULL DEFAULT CURRENT_TIMES TAMP	Дата публікації вакансії
jobs	is_active	BOOLEAN NOT NULL DEFAULT 1	Статус активності вакансії

Структура таблиці jobs відображає інформацію про вакансії, які публікуються роботодавцями. Зокрема, таблиця містить такі поля, як унікальний ідентифікатор вакансії (id), зовнішній ключ employer_id, що посилається на таблицю users і вказує на користувача-роботодавця, який розмістив вакансію, назва посади (title), детальний опис вакансії (description),

місцезнаходження (location), діапазон зарплати (salary_range), тип зайнятості (job_type), назва компанії для відображення (company_name_display), дата розміщення вакансії (posted_date) та статус активності вакансії (is_active). Зовнішній ключ employer_id забезпечує посилання на конкретного роботодавця, тим самим встановлюючи зв'язок «один до багатьох» між таблицями users і jobs: один користувач-роботодавець може мати кілька вакансій, водночас кожна вакансія належить одному роботодавцю. Поле company_name_display, яке може відрізнятись від офіційної назви компанії, дає можливість явного визначення відображуваного імені компанії у вакансії, що підвищує гнучкість в дизайні інтерфейсу. Для дата поля використовується тип TIMESTAMP з автоматичним заповненням поточною датою і часом у момент публікації вакансії. Поле is_active вказує на те, чи активна вакансія, і є логічним прапором для фільтрації доступних позицій.

Зв'язок між таблицями забезпечується за допомогою механізму зовнішніх ключів, який гарантує цілісність бази даних та забороняє створення вакансії без прив'язки до існуючого роботодавця. Схема реалізована через зовнішній ключ maxemployer_id, що посилається на поле id у таблиці users, з огляду на те, що лише користувачі з роллю «employer» мають право публікувати вакансії. Цей зв'язок реалізує ієрархічну структуру, де кожна вакансія є підпорядкованою певному роботодавцю, що відображає реальний процес роботи системи.

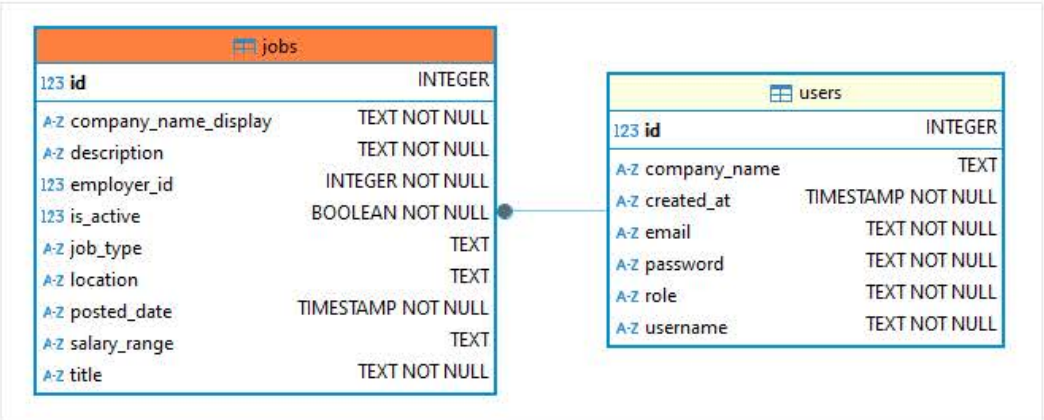

Рисунок 3.2 – Схема БД

Якщо розглядати детальніше типи даних полів, таблиці орієнтовані на оптимальний баланс між ефективністю зберігання та гнучкістю: текстові поля для імен та описів, поля типу TIMESTAMP для фіксації часу, тип INTEGER для ідентифікаторів і логічного прапора. Усі текстові провідні поля, які є унікальними або критичними (username, email), проєктовані з унікальними обмеженнями, що дозволяють виключити дублікати і тим самим забезпечити коректність даних. Хешування паролів користувачів впроваджено на рівні логіки застосунку, що забезпечує безпеку збереження інформації, уникаючи прямого зберігання відкритих паролів у базі.

Логіка взаємодії між цими двома таблицями дозволяє виконувати корисні операції, включаючи пошук вакансій за ключовими словами, фільтрацію за місцем розташування, а також відображення інформації про роботодавця на сторінках вакансій. При цьому реляційна модель БД забезпечує цілісність та консистентність даних через суворе дотримання зовнішніх ключів та унікальних індексів. Відфільтровані та відсортовані результати запитів налаштовуються на рівні SQL-запитів у коді застосунку.

3.3 Реалізація серверної частини. Налаштування Flask, маршрутизація, логіка бізнес-процесів

Реалізація серверної частини вебзастосунку для пошуку та підбору персоналу базується на використанні мікрофреймворку Flask, який забезпечує гнучкість та легкість у створенні RESTful API, а також можливості розширення та масштабованості. Першим етапом є базове налаштування самого середовища Flask, що передбачає визначення конфігураційних параметрів застосунку, створення об'єкта застосунку, налаштування секретного ключа для сесій і з'єднання з базою даних. Використання секретного ключа унікальним та непередбачуваним способом є необхідним для підтримки безпеки сесій користувачів, що зберігає конфіденційність їхніх

персональних даних і захищає від атак типу CSRF. Конфігурація підключення до бази даних здійснюється шляхом вказання шляху до файлу SQLite, котрий містить всю інформацію про користувачів, вакансії та інші пов'язані дані (рис. 3.3).

Рисунок 3.3 – Блок-схема серверної частини

Важливою складовою сервера є реалізація обробки маршрутів, яка відповідає за реакцію на запити користувачів і формування відповідей. У Flask маршрутизація реалізована через декоратори, які прямують конкретним функціям, котрі опрацьовують HTTP-запити різних методів, зокрема GET та POST. На головній сторінці за маршрутом / реалізована логіка виводу списку вакансій із можливістю пошуку за ключовими словами та фільтрації за місцем розташування. Запити до бази даних здійснюються з урахуванням наданих

параметрів для вибірки актуальних вакансій, що позитивно впливає на користувацький досвід через швидке та релевантне відображення інформації.

Обробка реєстрації та автентифікації користувачів передбачає окремі маршрути /register і /login відповідно. В процесі реєстрації забезпечується валідація введених даних: унікальність імені користувача та електронної пошти, відповідність паролів, а також додаткова перевірка для ролі роботодавця з вимогою надати назву компанії. Паролі користувачів зберігаються у базі даних у хешованому вигляді, що виключає ризик їх компрометації у разі несанкціонованого доступу. Для входу користувачів застосовані механізми перевірки відповідності електронної пошти та паролю з хешованою версією, а також підтримується запам'ятовування сесії користувача.

Модель користувача імплементована у вигляді класу, який наслідується від UserMixin бібліотеки Flask-Login, що забезпечує стандартизовані методи для перевірки автентичності та ідентифікації користувача в системі. Завантажувач користувача завантажує інформацію про користувача при кожному запиті, що дозволяє здійснювати плавний перехід між сторінками без необхідності повторного запиту авторизаційних даних.

Важливою частиною бізнес-логіки є розмежування функціональності за ролями користувачів. У разі ролі роботодавця активуються додаткові можливості, зокрема публікація нових вакансій через маршрут /post_job. При цьому формується перевірка обов'язкових полів, таких як назва вакансії, опис і компанія для відображення, щоб уникнути помилок і некоректних записів у базі даних. Вакансії зберігаються з посиланням на ідентифікатор користувача-роботодавця, що дозволяє відслідковувати приналежність до певного підприємства та керувати їхнім набором персоналу. Для кандидатів передбачена стандартна інформаційна панель без можливості додавання вакансій.

Додатковим маршрутом є /dashboard, який виконує функцію персонального кабінету користувача. Для роботодавців у цьому розділі

відображаються всі їхні опубліковані вакансії з можливістю подальшого їх перегляду та адміністрування. Кандидати ж бачать більш обмежену інформацію, що відповідає їх ролі у застосунку. Налагоджена система перенаправлень в залежності від статусу користувача створює комфортну навігацію та уникнення розгалужених сторінок без відповідного доступу.

Детальний перегляд вакансії реалізований на маршруті з використанням динамічного сегменту URL, який включає ідентифікатор вакансії. У цьому маршруті здійснюється вибірка з бази даних з об'єднанням інформації про вакансію та роботодавця для відображення повного опису, контактів та умов. Застосовується фільтр форматування дати для забезпечення зручного візуального сприйняття дати публікації та оновлення інформації, що підвищує довіру користувачів до платформи.

Серверна частина також керує безпекою застосунку через застосування декораторів login_required, які обмежують доступ до певних маршрутів лише автентифікованим користувачам, запобігаючи несанкціонованому доступу й захищаючи конфіденційність та цілісність даних. Застосунок використовує також механізм флеш-повідомлень для інформування користувача щодо результатів дій, таких як успішна авторизація, помилки у формі чи повідомлення про вихід з системи.

Взаємодія з базою даних забезпечується через відкриття та закриття з'єднання у межах обробників запитів, що дозволяє ефективно використовувати ресурси та уникнути втрат з'єднань. Функція ініціалізації бази даних виконує виконання SQL-скрипту, який створює необхідну структуру таблиць, включаючи таблиці користувачів та вакансій. Модель даних налаштована з урахуванням вимог бізнес-логіки, передбачаючи унікальні поля, зв'язки між таблицями та додаткові атрибути для різноманітних ролей користувачів.

3.4 Модель користувача та механізми аутентифікації. Використання Flask-Login, класи користувачів

Процес моделювання користувача починається з визначення класу, який репрезентує користувача у системі. У цьому застосунку створено клас User, що наслідує інтерфейс UserMixin із Flask-Login. Це дозволяє інтегруватися з API бібліотеки для управління сесією. Структура забезпечує гнучкість у представлення категорій користувачів, що є загальноприйнятою практикою у побудові систем управління доступом, базуючись на ролях (табл 3.2).

Таблиця 3.2

Аспекти моделі користувача

Аспект	Опис
Модель користувача	Клас User з атрибутами id, username, email, role та company_name; інтеграція з Flask-Login через UserMixin.
Завантаження користувача	Функція load_user отримує дані з SQLite, створює об'єкт User для управління сесією.
Безпека аутентифікації	Хешування паролів за допомогою werkzeug.security; generate_password_hash і check_password_hash.
Реєстрація	Валідація полів, перевірка на унікальність username і email; роль і company_name для роботодавців.
Вхід в систему	Перевірка email і пароля, ініціалізація сесії через login_user з remember=True.
Управління ролями	Розмежування доступу на рівні маршрутів залежно від ролі (candidate або employer).
Інтеграція Flask-Login	Централізоване управління сесіями, захист маршрутів декоратором login_required, гнучкість у розширенні.

Клас User містить атрибути, які відображають сутність користувача у системі, зокрема унікальний ідентифікатор, ім'я користувача, електронну пошту, роль, а також назву компанії (опціонально для роботодавців).

Ініціалізація об'єкта класу User відбувається на основі даних, отриманих з бази даних, що зберігається у SQLite3. Для запиту даних використовується функція load_user, яка є обов'язковим компонентом у Flask-Login і відповідає за завантаження користувача за унікальним ідентифікатором. Ця функція виконує вибірку запису з таблиці користувачів, реконструюючи об'єкт User із відповідними полями. Завдяки використанню row factory для курсора SQLite, результат запиту повертається у вигляді об'єкту, до полів якого можливо звертатися за іменем, що підвищує читабельність та зручність подальшої роботи з даними. Система аутентифікації реалізована за допомогою Flask-Login, яка транспарентно управляє сесіями користувачів – створює, зберігає і завершує їх, а також контролює доступ до захищених маршрутів шляхом декоратора login_required. В цьому проєкті налаштовано поведінку, яка перенаправляє неавторизованих користувачів до сторінки входу, а також надаються повідомлення про необхідність авторизації з відповідними категоріями для візуального відображення помилок та інформації.

Варто зазначити, що безпека аутентифікації забезпечується криптографічним хешуванням паролів із застосуванням бібліотеки werkzeug.security. Конкретно для збереження пароля використовується функція generate_password_hash, що генерує захищений хеш, а для перевірки пароля застосовується check_password_hash. Таким чином, у базі даних ніколи не зберігаються паролі у відкритому вигляді, що мінімізує ризики компрометації даних у разі потенційних атак.

Реєстрація нового користувача включає в себе валідацію введених даних, що передбачає, зокрема, перевірку непорожніх полів, відповідність паролів і відсутність вже існуючих користувачів із тим самим іменем або електронною поштою. Для користувачів із роллю роботодавця додатково вимагається наявність назви компанії, що логічно відповідає ролі створювача

вакансій. У разі відсутності або невідповідності введених даних відбувається відображення відповідних повідомлень про помилки, що полегшує взаємодію користувача із системою.

Маршрут входу реалізовано з перевіркою користувача за електронною поштою, оскільки це унікальний ідентифікатор, більш зручний у застосуванні порівняно із логіном. Після успішної аутентифікації відбувається ініціалізація сесії користувача через функцію login_user із опціональним параметром remember=True, що зберігає сесію навіть після закриття браузера, якщо це передбачено логікою застосунку. В протилежному випадку користувачу відображаються повідомлення про помилки: неправильна електронна пошта або пароль. Інтеграція Flask-Login у Flask-застосунок забезпечує централізоване управління станом аутентифікації, що суттєво спрощує реалізацію системи доступу. Крім того, ця бібліотека передбачає можливість легкого розширення – наприклад, додаткову авторизацію на основі ролей. У подальшому розробнику доступні методи для визначення поведінки при різних подіях аутентифікації, що підвищує гнучкість та контроль безпеки.

З огляду на ролі користувачів у системі визначено два основних типи: кандидати та роботодавці. Кандидати можуть шукати і відгукуватися на вакансії, у той час як роботодавці мають можливість створювати і керувати своїми вакансіями. В моделі користувача роль зберігається у відповідному полі бази даних, що дає змогу контролювати права доступу на рівні маршрутів, реалізованих через Flask. Наприклад, сторінка створення вакансії доступна лише для користувачів із роллю роботодавця, що реалізовано за допомогою перевірок у відповідних view-функціях.

3.5 Обробка запитів реєстрації та входу користувачів. Валідація, хешування паролів, повідомлення системи

Розпочинаючи з процесу реєстрації користувача, слід зауважити, що він слугує початковою точкою формування унікального облікового запису у

системі. Під час його обробки запроси, які надходять від клієнтської частини застосунку, проходять специфічний алгоритм перевірок, спрямований на підтвердження коректності та повноти наданої інформації. Це реалізовано за допомогою механізмів валідації, які здійснюються безпосередньо на серверній стороні у функції маршруту реєстрації та передбачають перевірку обов'язкових полів, таких як ім'я користувача, електронна пошта, пароль та підтвердження пароля, а також специфічних для певних ролей користувачів додаткових даних, наприклад, назви компанії у випадку роботодавця. Валідація цих параметрів передбачає контроль наявності непорожніх значень, коректності формату електронної пошти та відповідності введених паролів, оскільки саме через ці дані забезпечується подальша безперебійна робота системи ідентифікації та авторизації. Особлива увага приділяється неповторності унікальних ідентифікаторів, таких як ім'я користувача та електронна пошта, для запобігання дублюванню втрат, що реалізується за допомогою відповідних запитів до бази даних для перевірки наявності записів зі співпадаючими значеннями.

Наступним найважливішим елементом у процесі реєстрації є забезпечення безпеки паролів користувачів, що досягається через застосування криптографічного хешування. У рамках даного вебзастосунку використовується функція генерації захищеного хешу пароля зі спеціалізованої бібліотеки, що реалізує сучасні алгоритми хешування з урахуванням солі (додавання унікального випадкового рядка до пароля перед хешуванням), що значно ускладнює проведення атак перебору або словникових атак. Таким чином, жоден відкритий пароль не зберігається у базі даних, що є однією з фундаментальних вимог інформаційної безпеки. Збережений у вигляді хешу пароль згодом під час спроби входу гарантовано автентифікує лише користувачів, які вводять правильні початкові дані. Важливо відзначити, що генерація хешу здійснюється лише після успішного проходження всіх перевірок валідації, що дозволяє уникнути непотрібного навантаження на процесор та забезпечує оптимальну роботу системи.

У процесі реалізації функціоналу авторизації користувача на основі введених даних електронної пошти та пароля здійснюється пошук відповідного запису у базі даних, після чого перевіряється збіг хешованого пароля з введеним користувачем. Порівняння здійснюється з використанням захищених методів, доступних у бібліотеці для перевірки хешів, що гарантує рівень безпеки, відповідний сучасним стандартам. У разі, якщо відповідний користувач не знайдено, або введений пароль не співпадає з хешем, функціонал передбачає формування чітких та зрозумілих повідомлень про помилки, які ознайомлюють користувача з причинами відмови у доступі без розкриття специфічної внутрішньої інформації, що могло б бути використано зловмисниками. Так, наприклад, повідомлення відрізняють між неправильною електронною поштою і помилковим паролем, однак при цьому зберігають граничну обережність, щоб не допускати фішингових або інших атак із визначення зареєстрованих облікових записів.

Важливою деталлю є інтеграція з бібліотекою Flask-Login, яка забезпечує менеджмент сесій користувачів, включаючи збереження стану, обробку успішного входу шляхом виклику відповідних методів, а також можливість підтримки функції «запам'ятати користувача», що покращує користувацький досвід. Після завершення автентифікації система друкує користувачеві повідомлення успішного входу та автоматично перенаправляє його до персональної панелі керування або іншого релевантного ресурсу. В разі ж виникнення помилок – система коректно інформує користувача за допомогою флеш-повідомлень, які виводяться у вигляді яскравих та інформативних банерів у головній області сторінки, що стимулює повторні спроби введення коректної інформації.

Система також реалізує валідацію на стороні клієнта з використанням динамічної зміни форми, наприклад, через JavaScript, що відповідає за показ або приховування додаткового поля «Назва компанії» у разі вибору ролі «роботодавець» у процесі реєстрації. Такий підхід не тільки покращує юзабіліті, проте й служить додатковим бар'єром для помилок введення або

пропуску важливої інформації. Проте все одно існує повний контроль на серверній стороні, що виключає спроби обходу клієнтської валідації та гарантує цілісність отриманих даних.

Важливо підкреслити, що в процесі розробки враховані аспекти обробки помилок, які можуть виникнути при виконанні запитів до бази даних, а також логіку взаємодії з користувачем, що дозволяє унаочнити виниклі ситуації та висвітлити конкретні інструкції щодо їх виправлення. Такий системний підхід оптимізує взаємодію з кінцевим користувачем і знижує ймовірність випадкових помилок чи неправильного використання сервісу.

Крім того, в цілому дизайн системи обробки вхідних запитів орієнтований на підтримку розмежування прав доступу, що відображено в тому, що лише користувачі з роллю «роботодавець» мають повноваження з публікації вакансій, а кандидати отримують доступ до пошуку і перегляду вакансій та власної інформаційної панелі. Це має пряме відношення до процесів реєстрації, адже при виборі ролі відбувається відповідна обробка введених даних та подальша маршрутизація. Таким чином, обробка запитів автентифікації й реєстрації є не лише інструментом доступу користувача, а й механізмом реалізації бізнес-логіки, що диктує призначені ролі і відповідні їм функціональні можливості.

3.6 Реалізація функціоналу публікації вакансій. Інтерфейси, валідація вводу, збереження даних

Валідація процесу введення є невід'ємною частиною функціоналу та спрямована на запобігання введенню некоректних, неповних чи потенційно шкідливих даних. Розроблена система валідації працює в декілька рівнів і включає як клієнтську, так і серверну перевірку. Окрім базової перевірки присутності основних полів, система контролює логічну послідовність і семантичне наповнення, наприклад, забезпечується ненульова і не порожня назва вакансії та опис, що має бути достатньо інформативним для потенційних

кандидатів. Особливо увагу приділено валідації поля з назвою компанії для відображення, яке є обов'язковим, оскільки це безпосередньо впливає на зовнішній вигляд оголошення. Якщо перевірка знаходить недоліки або порушення правил, користувач отримує інформативне повідомлення з підказкою, яке допомагає усунути помилки, завдяки чому підвищується якість і повнота інформації, що вводиться. Такий підхід мінімізує ризики появи невалідних оголошень та спрощує подальшу модерацію.

Щодо деталей реалізації, технологічна основа використовує Flask – легкий вебфреймворк на Python, а для зберігання даних – SQLite3, що є оптимальним варіантом для невеликих і середніх проєктів. Форма має метод POST, щоб безпечно передавати інформацію на сервер. В процесі обробки на сервері спочатку відбувається отримання інформації із форми через об'єкт request. Потім проводиться валідація, описана вище. Якщо усі перевірки пройдені успішно, система використовує параметризовані SQL-запити для вставки нової вакансії в таблицю бази даних «jobs». Такий спосіб запобігає SQL-ін'єкціям, підвищуючи безпеку застосунку. Вставка включає ідентифікатор роботодавця, отриманий з контексту автентифікованого користувача, що визначає зв'язок між вакансією та профілем компанії. Окремо зберігається дата публікації, яку база даних генерує автоматично, гарантуючи точність хронології.

Збережені дані є структурованими й стандартизованими, втім передбачено гнучкість у вигляді необов'язкових для заповнення полів, таких як локація та діапазон заробітної плати, що дозволяє роботодавцям формувати вакансії під свої потреби. Таким чином, система підтримує багато варіацій вакансій, адаптуючись до реальних вимог ринку праці. Детальний опис, збережений у текстовому форматі, підтримує багаторядкове введення, зберігаючи форматування, що позитивно впливає на читабельність і загальне сприйняття оголошення. Для більшої безпеки вхідні тексти при відображенні на інших сторінках обробляються із застосуванням фільтрів, що попереджають уразливості, зокрема XSS-атаки. З боку користувача, у разі

успішного виконання операції, система повідомляє про успішність публікації за допомогою flash-повідомлення, що відображається у верхній частині екрану, підвищуючи довіру до інструменту і спонукання до подальшої активності. У разі помилок або виявлення некоректних даних користувач отримує чітку, контекстно-залежну інформацію, що допомагає виправити недоліки. Використання сесійних повідомлень та механізму flash вплинуло на зручність роботи з формами й покращення користувацького досвіду.

Цілий функціонал тісно інтегрований із системою автентифікації, реалізованою через Flask-Login, завдяки чому лише аутентифіковані користувачі з роллю роботодавця мають доступ до сторінки публікації вакансій. Забезпечується перевірка прав доступу прямо на сервері, що виключає можливість обходу авторизації і несанкціонованого додавання вакансій. Впроваджена логіка корегує користувацький інтерфейс на стороні клієнта, наприклад, автоматично приховує або відображає додаткове поле назви компанії залежно від ролі, що знижує ризик помилок і покращує інтуїтивність. Дизайн форми має унікальний стиль, який підкреслює сучасність і унікальність проєкту, сприяючи приємній взаємодії навіть при тривалому використанні. Тим самим, комплексна реалізація функціоналу публікації вакансій, що поєднує продумані вебінтерфейси, надійну та гнучку валідацію вводу, а також безпечне й структуроване збереження даних, формує фундамент для подальшого розвитку платформи. Вона створює передумови для масштабування, інтеграції нових сервісів, таких як пошукові фільтри, аналітика активності вакансій або інтеграція з зовнішніми ресурсами, що значно розширить можливості користувачів і підвищить ефективність роботи вебзастосунку в контексті підбору персоналу. Така архітектура й практична реалізація відповідають сучасним стандартам розробки, орієнтуються на потреби кінцевого користувача і забезпечують необхідну надійність і масштабованість продукту.

На рисунках 3.4-3.21 наведено реалізований функціонал вебзастосунку.

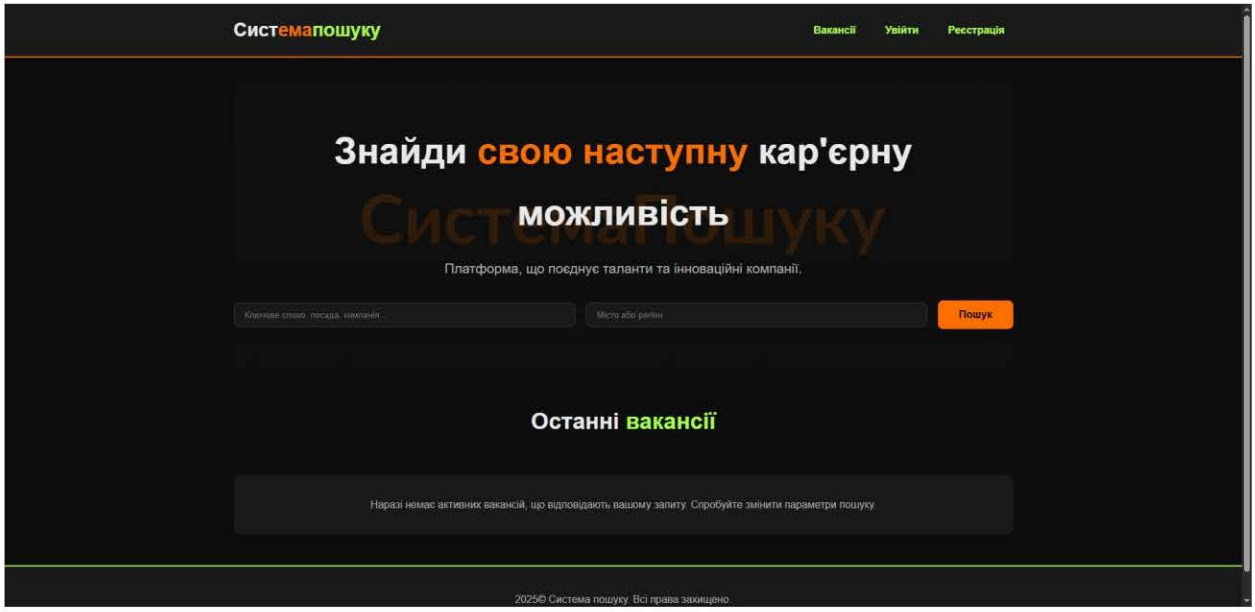

Рисунок 3.4 – Головна сторінка

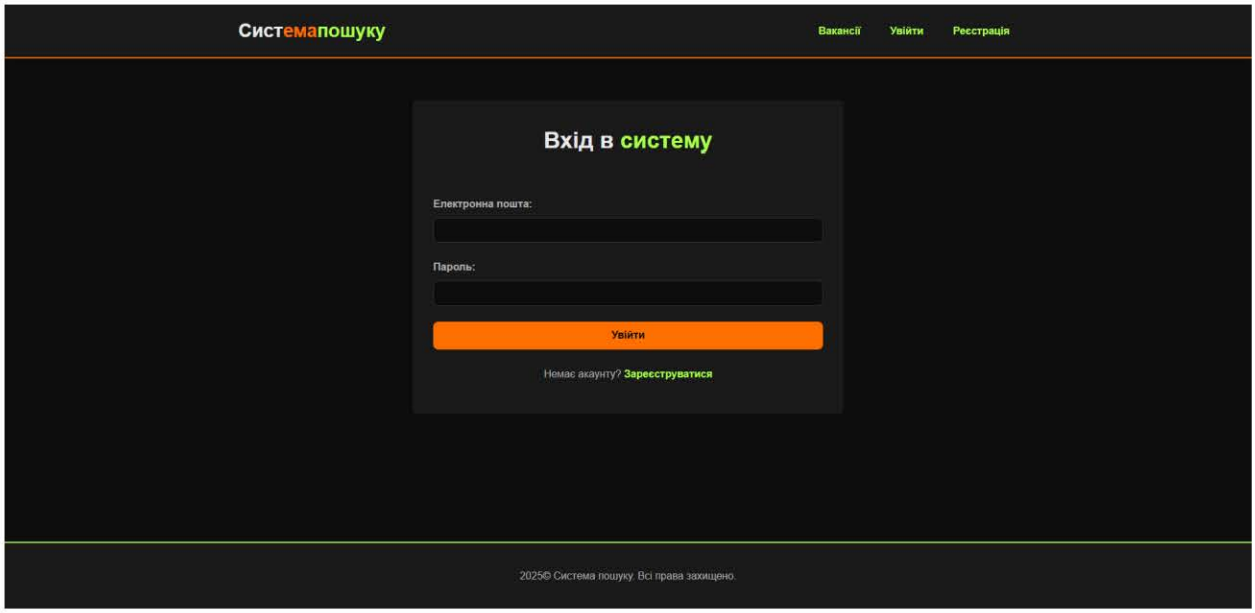

Рисунок 3.5 – Сторінка авторизації

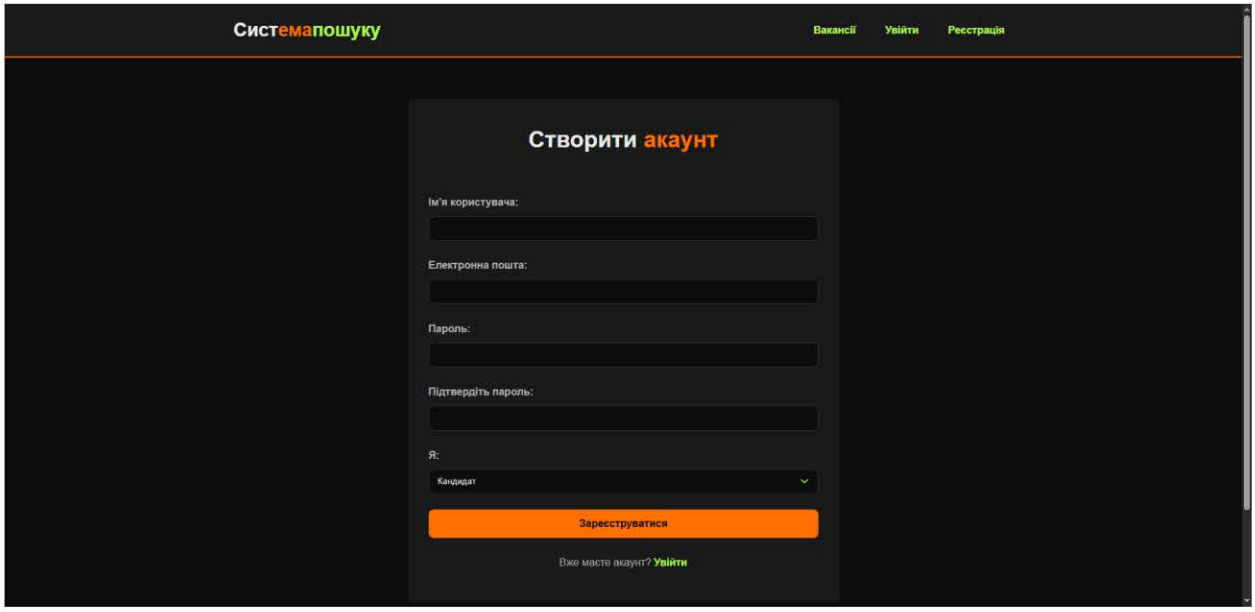

Рисунок 3.6 – Сторінка реєстрації

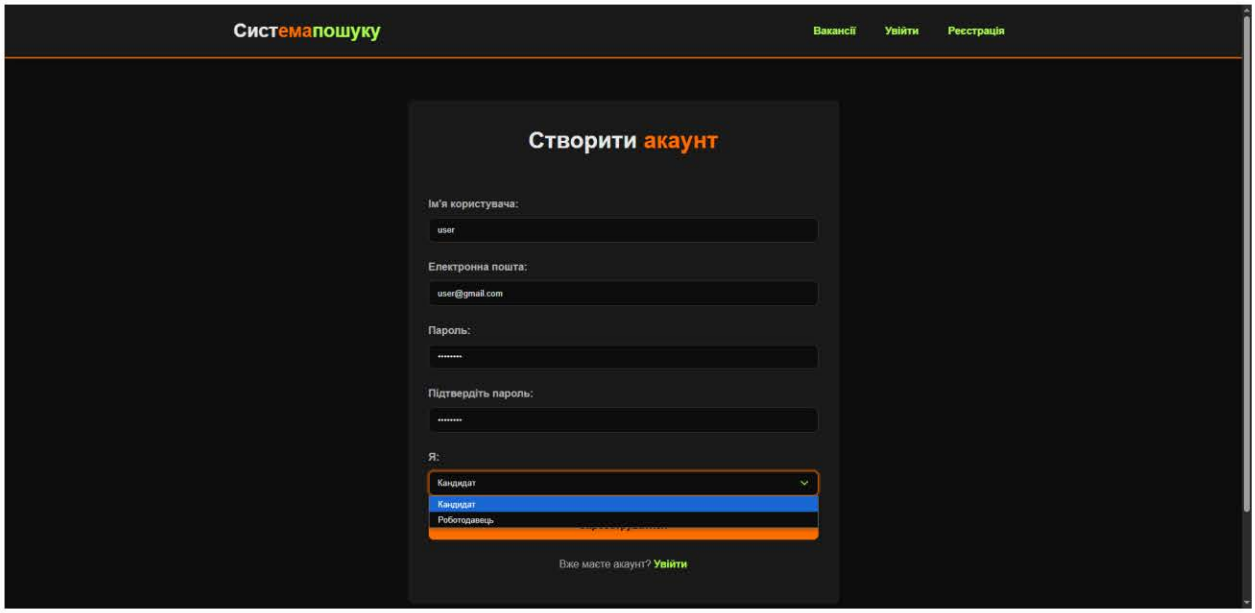

Рисунок 3.7 – Вибір ролі користувача під час реєстрації

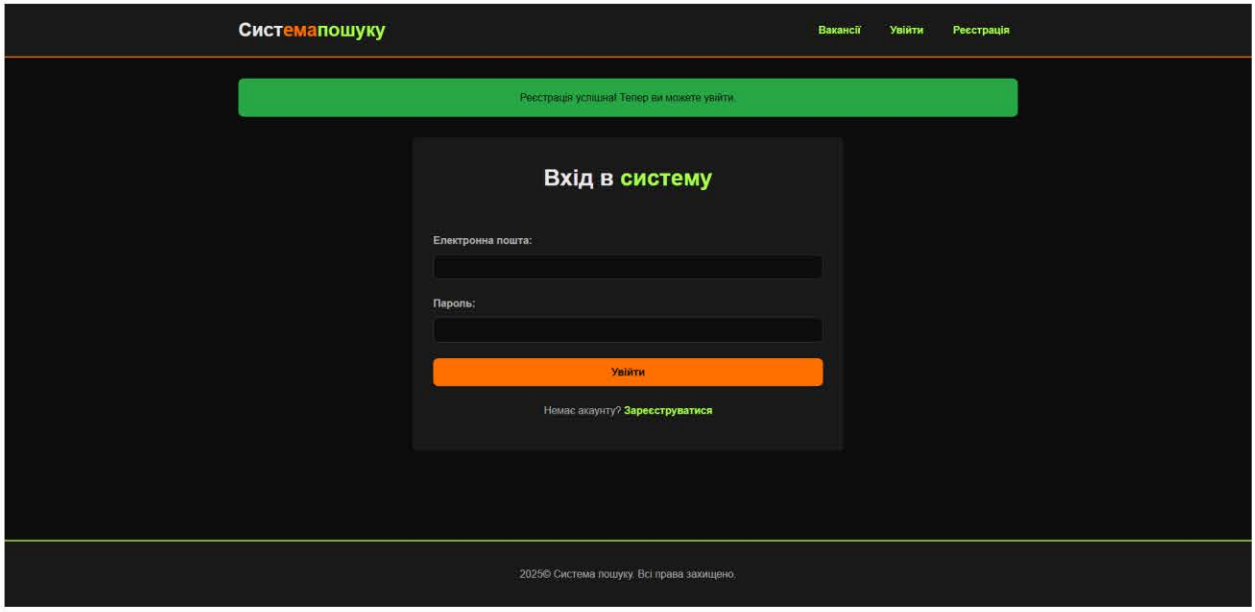

Рисунок 3.8 – Успішна реєстрація

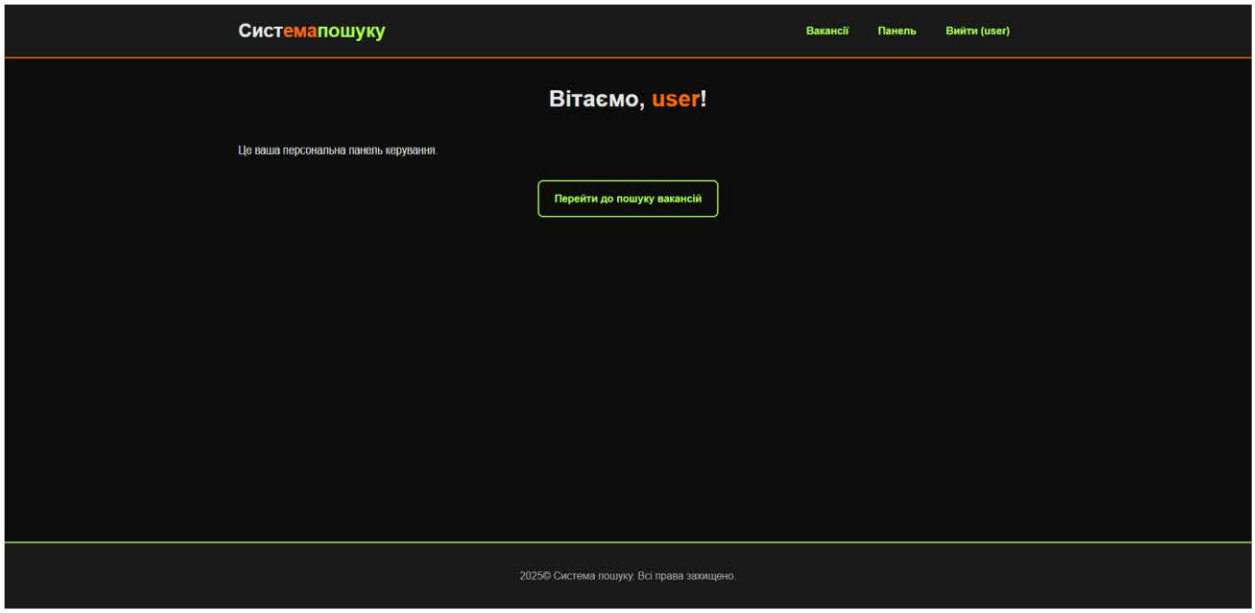

Рисунок 3.9 – Успішна авторизація

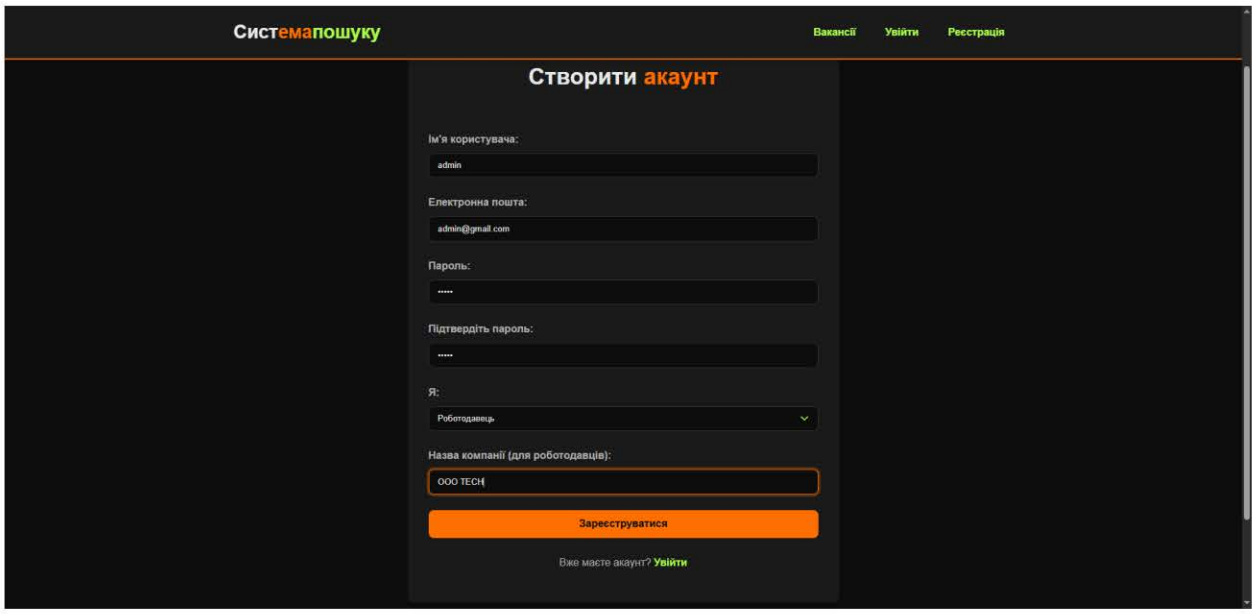

Рисунок 3.10 – Створення акаунта як роботодавець

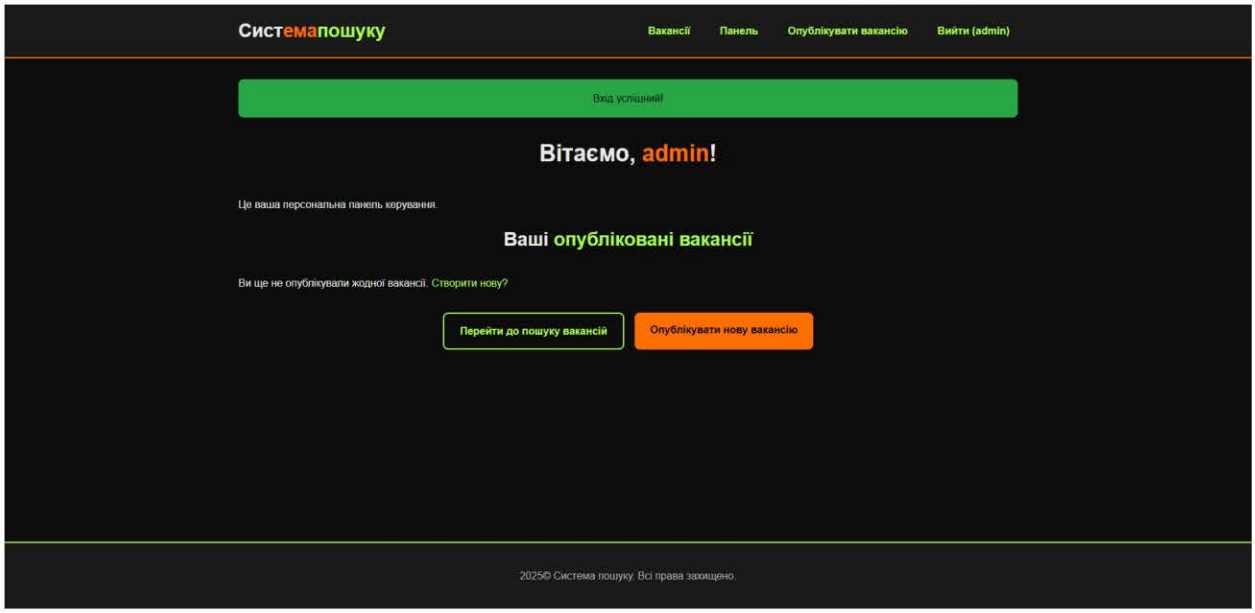

Рисунок 3.11 – Авторизація як роботодавець

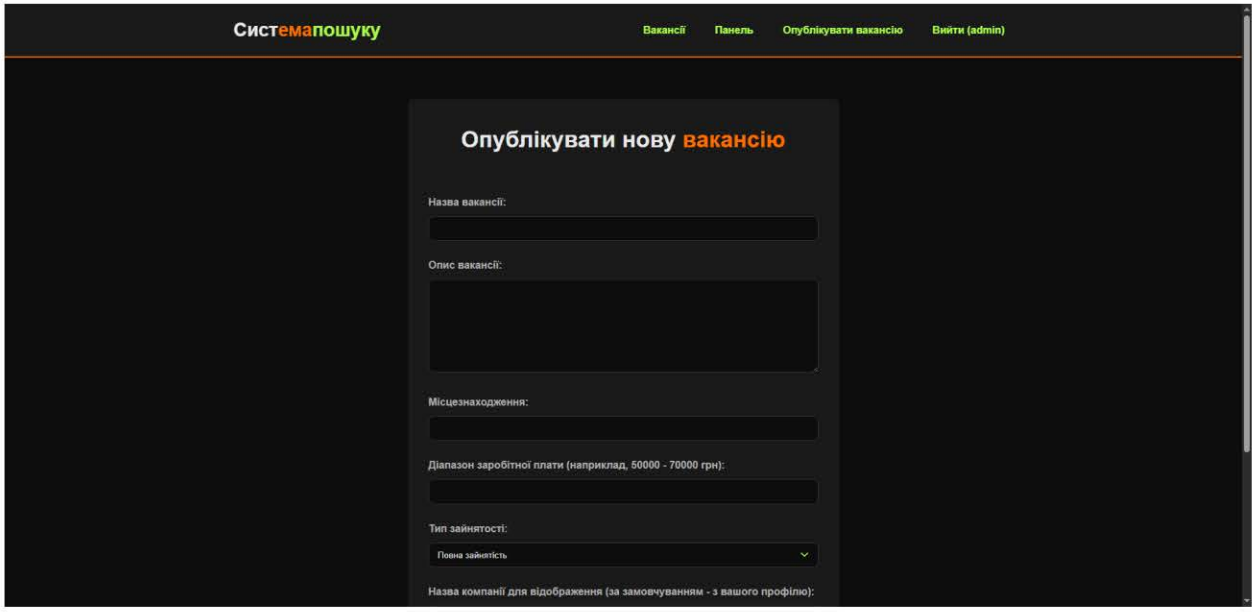

Рисунок 3.12 – Сторінка створення нової вакансії

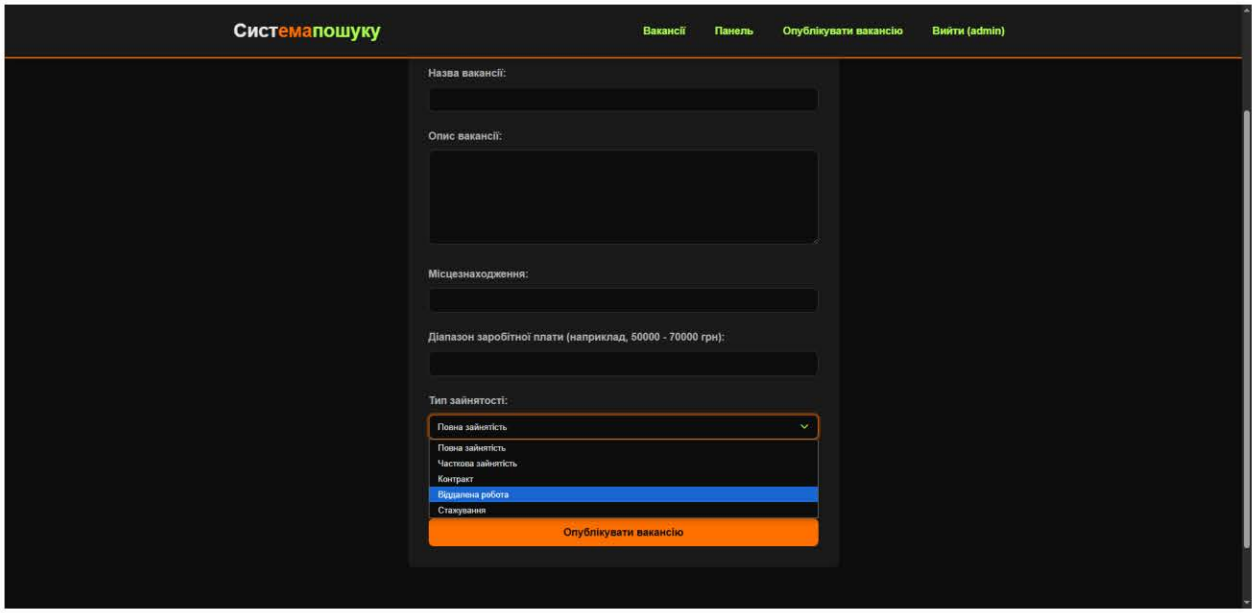

Рисунок 3.13 – Вибір типу занятості

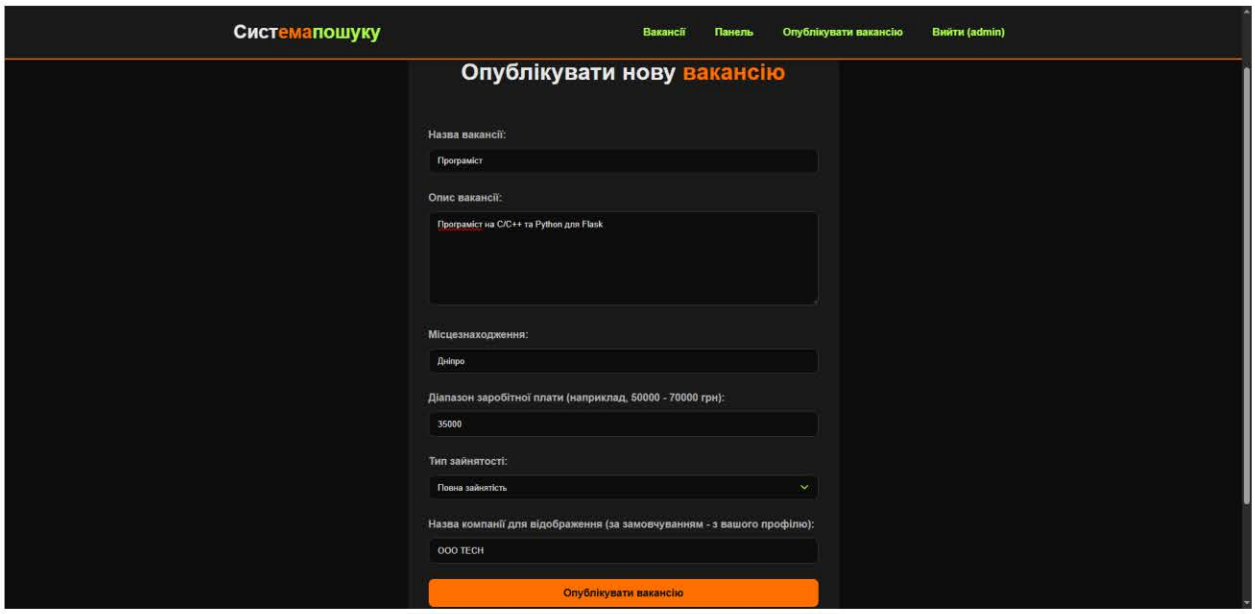

Рисунок 3.14 – Заповнені дані вакансії

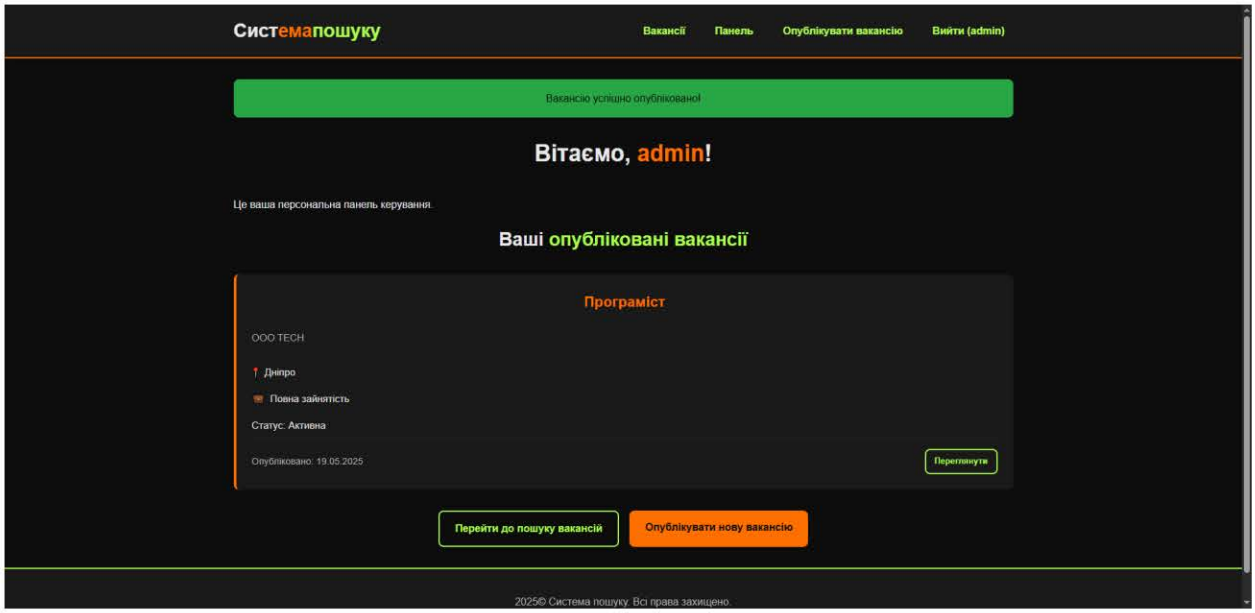

Рисунок 3.15 – Успішне додавання нової вакансії

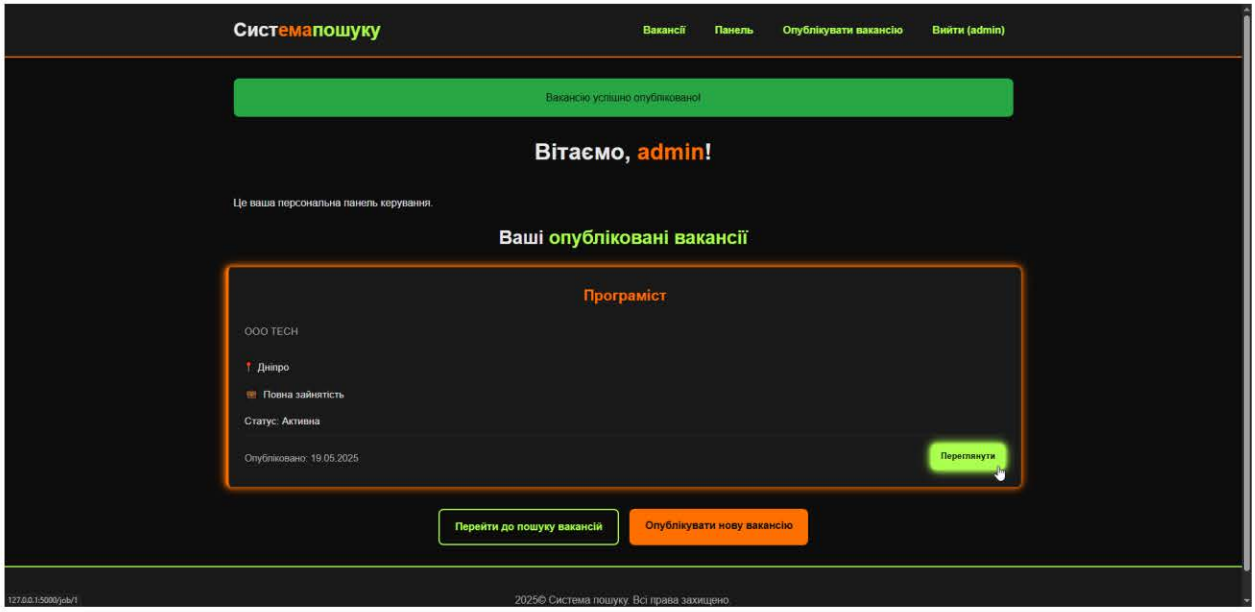

Рисунок 3.16 – Опублікована вакансія на головній сторінці

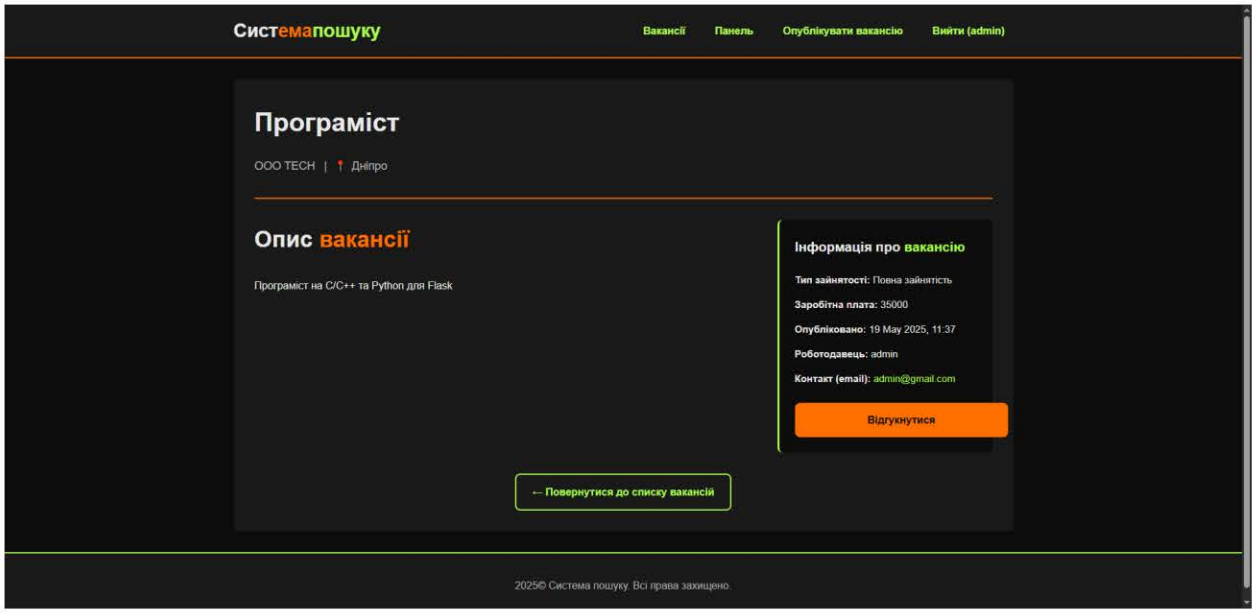

Рисунок 3.17 – Сторінка вакансії

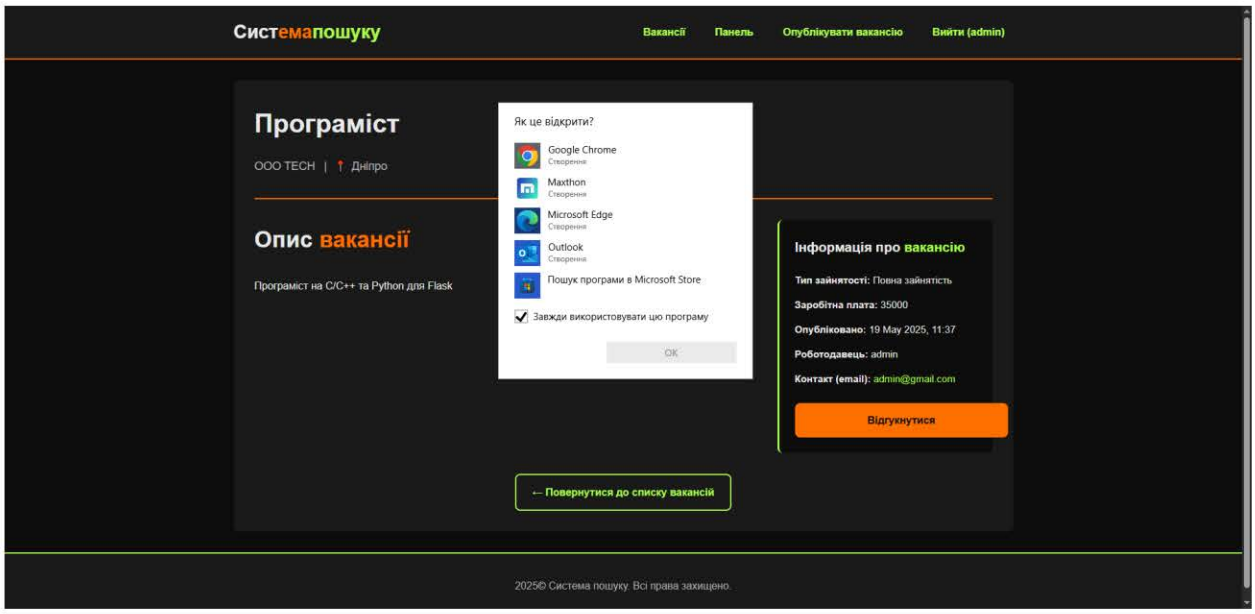

Рисунок 3.18 – Зв’язок з роботодавцем

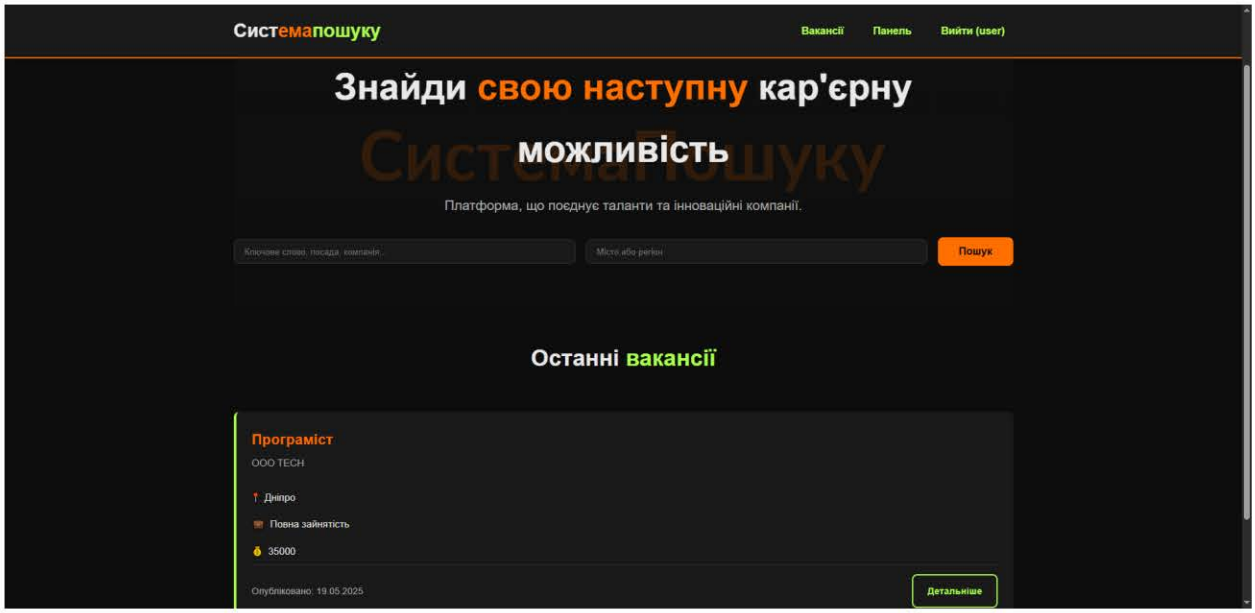

Рисунок 3.19 – Перегляд з боку користувача

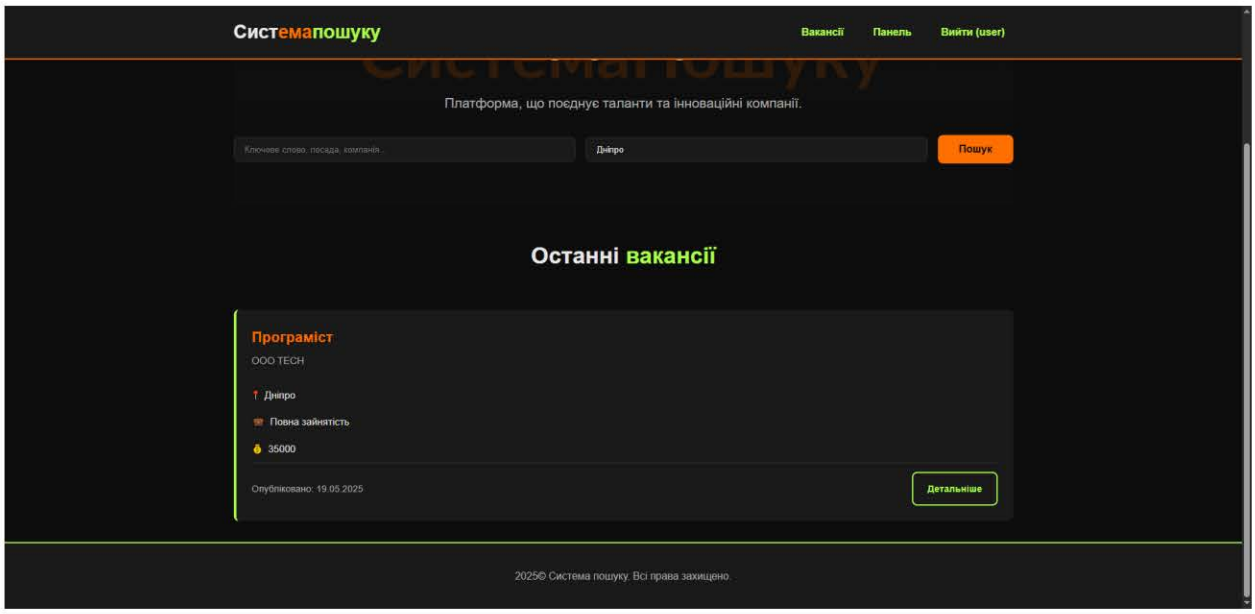

Рисунок 3.20 – Пошук вакансій

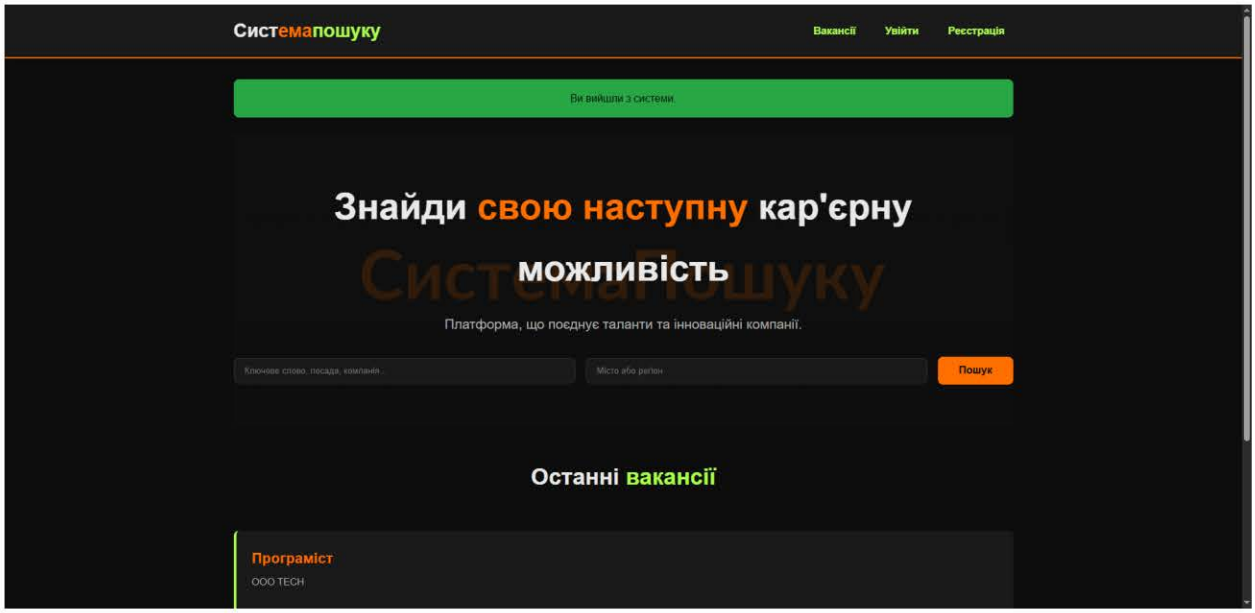

Рисунок 3.21 – Вихід з системи

3.7 Висновки до третього розділу

У третьому розділі реалізовано процес створення вебзастосунку для пошуку та підбору персоналу відповідно до поставлених цілей і завдань. Розроблено архітектуру програми з урахуванням ролей користувачів, основних бізнес-процесів та вимог до функціоналу. Було створено інтуїтивно

зрозумілий інтерфейс користувача, адаптований до потреб як кандидатів, так і роботодавців. Здійснено проєктування і реалізацію бази даних із таблицями користувачів та вакансій, а також забезпечено логіку їх взаємодії. Реалізовано функціонал реєстрації, входу до системи, публікації та редагування вакансій, пошуку за ключовими параметрами, що дозволяє користувачам ефективно взаємодіяти із системою. Особливу увагу приділено питанням безпеки: впроваджено захист автентифікації, хешування паролів, контроль доступу до ресурсів. Проведене тестування підтвердило коректність роботи основних модулів та стабільність системи. Таким чином, розроблений вебзастосунок повністю відповідає технічному завданню, демонструє практичну цінність та потенціал для подальшого розвитку.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи досягнуто поставленої мети – створено повноцінний вебзастосунок для пошуку та підбору персоналу, який відповідає сучасним вимогам до ефективності, функціональності та безпеки. Проведене дослідження підтвердило актуальність тематики в умовах цифрової трансформації ринку праці, зростання попиту на автоматизовані рішення в рекрутингу та потреби бізнесу в скороченні витрат часу і ресурсів на залучення кваліфікованих кадрів. На основі аналізу існуючих інформаційних систем і технологій було визначено оптимальні інструменти реалізації, що дозволило забезпечити гнучку архітектуру, адаптивний інтерфейс користувача та стабільну роботу всіх функціональних компонентів системи.

Під час розробки застосунку використано сучасні технології програмування, зокрема мову Python, мікрофреймворк Flask, систему управління базами даних SQLite3, а також бібліотеки Flask-Login і Werkzeug, які забезпечили реалізацію механізмів аутентифікації, захисту даних, управління сесіями та формування зручного для користувача середовища. У процесі проєктування було враховано специфіку взаємодії між роботодавцем і кандидатом, що дозволило сформувати логічну структуру бази даних, передбачити ролі користувачів та реалізувати точковий пошук і фільтрацію вакансій. Реалізований функціонал дозволяє публікувати, редагувати та переглядати вакансії, здійснювати пошук за ключовими параметрами, реєструвати та ідентифікувати користувачів, а також забезпечує базові елементи персоналізації й безпеки.

Оцінка працездатності системи показала її ефективність у вирішенні завдань пошуку персоналу для малого та середнього бізнесу. Система характеризується простотою в експлуатації, швидкою реакцією на дії користувачів та можливістю масштабування в разі потреби. Розроблений вебзастосунок є прикладом успішного поєднання теоретичних знань і

практичних навичок програмної інженерії, а також демонструє потенціал для подальшого вдосконалення, зокрема шляхом впровадження штучного інтелекту, систем рекомендацій або інтеграції з зовнішніми API.

Кваліфікаційна робота виконана у відповідності до стандарту спеціальності 122 – «Комп'ютерні науки» і демонструє володіння такими компетентностями як:

– здатність проєктувати та розробляти програмне забезпечення із застосуванням різних парадигм програмування: узагальненого, об'єктноорієнтованого, функціонального, логічного, з відповідними моделями, методами й алгоритмами обчислень, структурами даних і механізмами управління;

– здатність реалізувати багаторівневу обчислювальну модель на основі архітектури клієнт-сервер, включаючи бази даних, знань і сховища даних, виконувати розподілену обробку великих наборів даних на кластерах стандартних серверів для забезпечення обчислювальних потреб користувачів, у тому числі на хмарних сервісах.

Серед результатів навчання, визначених стандартом, кваліфікаційна робота реалізує наступні:

– розробляти програмні моделі предметних середовищ, вибирати парадигму програмування з позицій зручності та якості застосування для реалізації методів та алгоритмів розв'язання задач в галузі комп'ютерних наук;

– використовувати інструментальні засоби розробки клієнт-серверних застосувань, проєктувати концептуальні, логічні та фізичні моделі баз даних, розробляти та оптимізувати запити до них, створювати розподілені бази даних, сховища та вітрини даних, бази знань, у тому числі на хмарних сервісах, із застосуванням мов вебпрограмування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Rabiul T. Development of a Web-based Job Recruitment System. 2024. URL: https://www.researchgate.net/publication/388566213_Development_of_a_Web-based_Job_Recruitment_System

2. Walek B., Pektor O., Farana R. Proposal of the Web Application for Selection of Suitable Job Applicants Using Expert System. Advances in Intelligent Systems and Computing. Cham, 2016. С. 363–373. URL: https://doi.org/10.1007/978-3-319-33622-0_33

3. LinkedIn. URL: https://ua.linkedin.com/

4. Python Documentation. URL: https://docs.python.org/3/

5. Flask. URL: https://flask.palletsprojects.com/en/stable/

6. Grinberg M. Flask Web Development: Developing Web Applications with Python. 2018. 312 p.

7. Carroll T. Python Web Applications with Flask: Hand-on your Flask skills with advanced techniques and build dynamic web applications. 2024. 473 p.

8. Grinberg M. The New and Improved Flask Mega-Tutorial (2024 Edition). 2023. 326 p.

9. SQLite3. URL: https://www.sqlite.org/

10. sqlite3 module. URL: https://docs.python.org/3/library/sqlite3

11. Kreibich J. A. Using SQLite: Small. Fast. Reliable. Choose Any Three. 2010. 526 p.

12. Basu S. Learn SQLite with Python in 24 hours For Beginners - Simple, Concise & Easy Guide To Using Database with Python. 2021. 80 p.

13. Flask Login Documentation. URL: https://flask-login.readthedocs.io/en/latest/

14. Werkzeug. URL: https://werkzeug.palletsprojects.com/en/stable/

15. Werkzeug pypi. URL: https://pypi.org/project/Werkzeug/

16. Lathkar M. Building Web Apps with Python and Flask : Learn to Develop and Deploy Responsive RESTful Web Applications Using Flask Framework (English Edition). 2021. 307 p.

ДОДАТОК А

ІНІЦІАЛІЗАЦІЯ БАЗИ ДАНИХ

```
DROP TABLE IF EXISTS users;
DROP TABLE IF EXISTS jobs;

CREATE TABLE users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT UNIQUE NOT NULL,
    password TEXT NOT NULL,
    email TEXT UNIQUE NOT NULL,
    role TEXT NOT NULL DEFAULT 'candidate',
    company_name TEXT,
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE jobs (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    employer_id INTEGER NOT NULL,
    title TEXT NOT NULL,
    description TEXT NOT NULL,
    location TEXT,
    salary_range TEXT,
    job_type TEXT,
    company_name_display TEXT NOT NULL,
    posted_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    is_active BOOLEAN NOT NULL DEFAULT 1,
    FOREIGN KEY (employer_id) REFERENCES users (id)
);
```

ДОДАТОК Б

ЛІСТИНГ СЕРВЕРНОЇ ЧАСТИНИ

```
import sqlite3
import os
from flask import Flask, render_template, request, redirect, url_for, flash, g, session
from werkzeug.security import generate_password_hash, check_password_hash
from flask_login import LoginManager, UserMixin, login_user, login_required, logout_user, current_user


app = Flask(__name__)
app.config['SECRET_KEY'] = os.urandom(24)
app.config['DATABASE'] = os.path.join(app.root_path, 'database.db')

login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'login'
login_manager.login_message = «Будь ласка, увійдіть, щоб отримати доступ до цієї сторінки.»
login_manager.login_message_category = «info»


class User(UserMixin):
    def __init__(self, id, username, email, role, company_name=None):
        self.id = id
        self.username = username
        self.email = email
        self.role = role
        self.company_name = company_name

@login_manager.user_loader
def load_user(user_id):
    db = get_db()
    user_data = db.execute('SELECT id, username, email, role, company_name FROM users WHERE id = ?', (user_id,)).fetchone()
    if user_data:
        return User(id=user_data['id'], username=user_data['username'], email=user_data['email'], role=user_data['role'], company_name=user_data['company_name'])
    return None
```

```
def get_db():
    if 'db' not in g:
        g.db = sqlite3.connect(
            app.config['DATABASE'],
            detect_types=sqlite3.PARSE_DECLTYPES
        )
        g.db.row_factory = sqlite3.Row
    return g.db

@app.teardown_appcontext
def close_db(exception):
    db = g.pop('db', None)
    if db is not None:
        db.close()

def init_db():
    db = get_db()
    with app.open_resource('schema.sql', mode='r') as f:
        db.cursor().executescript(f.read())
    db.commit()
    print(«Базу даних ініціалізовано.»)

@app.cli.command('initdb')
def initdb_command():
    ««««Ініціалізує базу даних.»»««
    init_db()
    print('Базу даних ініціалізовано.')


@app.route('/')
def index():
    db = get_db()
    search_query = request.args.get('search', '')
    location_query = request.args.get('location', '')

    query = 'SELECT j.id, j.title, j.location, j.salary_range, j.job_type,
j.company_name_display, j.posted_date, u.username as employer_username FROM
jobs j JOIN users u ON j.employer_id = u.id WHERE j.is_active = 1'
    params = []

    if search_query:
        query += ' AND (j.title LIKE ? OR j.description LIKE ?)'
```

```
        params.extend(['%' + search_query + '%', '%' + search_query + '%'])
    if location_query:
        query += ' AND j.location LIKE ?'
        params.append('%' + location_query + '%')

    query += ' ORDER BY j.posted_date DESC'

    jobs = db.execute(query, tuple(params)).fetchall()
    return render_template('index.html', jobs=jobs, search_query=search_query,
location_query=location_query)

@app.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('index'))
    if request.method == 'POST':
        username = request.form['username']
        email = request.form['email']
        password = request.form['password']
        confirm_password = request.form['confirm_password']
        role = request.form['role']
        company_name = request.form.get('company_name', None)

        db = get_db()
        error = None

        if not username:
            error = «Ім'я користувача обов'язкове.»
        elif not email:
            error = «Електронна пошта обов'язкова.»
        elif not password:
            error = «Пароль обов'язковий.»
        elif password != confirm_password:
            error = «Паролі не співпадають.»
        elif db.execute('SELECT id FROM users WHERE username = ?',
(username,)).fetchone() is not None:
            error = f»Користувач {username} вже зареєстрований.»
        elif db.execute('SELECT id FROM users WHERE email = ?',
(email,)).fetchone() is not None:
            error = f»Електронна пошта {email} вже зареєстрована.»
        elif role == 'employer' and not company_name:
            error = «Назва компанії обов'язкова для роботодавця.»

        if error is None:
```

```
        hashed_password = generate_password_hash(password)
        db.execute(
          'INSERT INTO users (username, email, password, role, company_name)
VALUES (?, ?, ?, ?, ?)',
          (username, email, hashed_password, role, company_name if role ==
'employer' else None)
        )
        db.commit()
        flash('Реєстрація успішна! Тепер ви можете увійти.', 'success')
        return redirect(url_for('login'))

      flash(error, 'error')

  return render_template('register.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
  if current_user.is_authenticated:
    return redirect(url_for('index'))
  if request.method == 'POST':
    email = request.form['email']
    password = request.form['password']
    db = get_db()
    error = None
    user_data = db.execute('SELECT * FROM users WHERE email = ?',
(email,)).fetchone()

    if user_data is None:
      error = 'Неправильна електронна пошта.'
    elif not check_password_hash(user_data['password'], password):
      error = 'Неправильний пароль.'

    if error is None:
      user_obj = User(id=user_data['id'], username=user_data['username'],
email=user_data['email'], role=user_data['role'],
company_name=user_data['company_name'])
      login_user(user_obj, remember=True)
      flash('Вхід успішний!', 'success')
      next_page = request.args.get('next')
      return redirect(next_page or url_for('dashboard'))

    flash(error, 'error')

  return render_template('login.html')
```

```
@app.route('/logout')
@login_required
def logout():
    logout_user()
    flash('Ви вийшли з системи.', 'success')
    return redirect(url_for('index'))

@app.route('/dashboard')
@login_required
def dashboard():


    if current_user.role == 'employer':
        db = get_db()
        jobs = db.execute('SELECT * FROM jobs WHERE employer_id = ? ORDER
BY posted_date DESC', (current_user.id,)).fetchall()
        return render_template('dashboard.html', user_jobs=jobs)
    return render_template('dashboard.html')


@app.route('/post_job', methods=['GET', 'POST'])
@login_required
def post_job():
    if current_user.role != 'employer':
        flash('Тільки роботодавці можуть публікувати вакансії.', 'warning')
        return redirect(url_for('index'))

    if request.method == 'POST':
        title = request.form['title']
        description = request.form['description']
        location = request.form['location']
        salary_range = request.form['salary_range']
        job_type = request.form['job_type']
        company_name_display = request.form.get('company_name_display') or
current_user.company_name

        db = get_db()
        error = None

        if not title:
            error = «Назва вакансії обов'язкова.»
        elif not description:
            error = «Опис вакансії обов'язковий.»
```

```
        elif not company_name_display:
            error = «Назва компанії для відображення обов'язкова.»


        if error is None:
            db.execute(
                'INSERT INTO jobs (employer_id, title, description, location,
salary_range, job_type, company_name_display) VALUES (?, ?, ?, ?, ?, ?, ?)',
                (current_user.id, title, description, location, salary_range, job_type,
company_name_display)
            )
            db.commit()
            flash('Вакансію успішно опубліковано!', 'success')
            return redirect(url_for('dashboard'))

        flash(error, 'error')

    return render_template('post_job.html')

@app.route('/job/<int:job_id>')
def job_detail(job_id):
    db = get_db()
    job = db.execute(
        'SELECT j.*, u.username as employer_username, u.email as employer_email
'
        'FROM jobs j JOIN users u ON j.employer_id = u.id WHERE j.id = ? AND
j.is_active = 1',
        (job_id,)
    ).fetchone()

    if job is None:
        flash('Вакансію не знайдено або вона неактивна.', 'error')
        return redirect(url_for('index'))

    return render_template('job_detail.html', job=job)


from datetime import datetime

@app.template_filter('datetimeformat')
def datetimeformat(value, format='%d %B %Y, %H:%M'):
    if isinstance(value, str):

        try:
```

```
            value = datetime.fromisoformat(value)
        except ValueError:

            return value
    if isinstance(value, datetime):
        return value.strftime(format)
    return value


if __name__ == '__main__':
    app.run(debug=True)
```

ДОДАТОК В

ЛІСТИНГ СТИЛІВ

```
:root {
  --color-dark-bg: #0D0D0D;
  --color-dark-bg-secondary:
#1A1A1A;
  --color-primary-orange: #FF6F00;
  --color-primary-orange-hover:
#E66300;
  --color-secondary-lime: #ABFF4F;
  --color-secondary-lime-hover:
#9AE647;
  --color-text-light: #EAEAEA;
  --color-text-medium: #B0B0B0;
  --color-text-dark: #333333;
  --color-border: #303030;
  --color-success: #28a745;
  --color-error: #dc3545;
  --color-info: #17a2b8;
  --color-warning: #ffc107;

  --font-primary: 'Poppins', sans-
serif;
  --border-radius: 8px;
  --card-shadow: 0 5px 15px
rgba(0,0,0,0.2);
  --neon-orange-glow: 0 0 5px var(--
color-primary-orange), 0 0 10px var(--
color-primary-orange), 0 0 15px var(--
color-primary-orange);
  --neon-lime-glow: 0 0 5px var(--
color-secondary-lime), 0 0 10px var(--
color-secondary-lime), 0 0 15px var(--
color-secondary-lime);
}


body {
  font-family: var(--font-primary);
  background-color: var(--color-dark-
bg);
  color: var(--color-text-light);
  line-height: 1.7;
  margin: 0;
  padding: 0;
  display: flex;
  flex-direction: column;
  min-height: 100vh;
  font-size: 16px;
}

.container {
  width: 90%;
  max-width: 1200px;
  margin: 0 auto;
  padding: 0 15px;
}

a {
  color: var(--color-secondary-lime);
  text-decoration: none;
  transition: color 0.3s ease;
}

a:hover {
  color: var(--color-primary-orange);
}

h1, h2, h3, h4, h5, h6 {
  margin-top: 0;
  margin-bottom: 0.75em;
  font-weight: 600;
  color: var(--color-text-light);
}
h1 { font-size: 2.8em; }
h2 { font-size: 2.2em; }
```

```
h3 { font-size: 1.8em; }

.highlight-orange { color: var(--color-primary-orange); }
.highlight-lime { color: var(--color-secondary-lime); }


.site-header {
  background-color: var(--color-dark-bg-secondary);
  padding: 1em 0;
  border-bottom: 2px solid var(--color-primary-orange);
  box-shadow: 0 2px 10px rgba(0,0,0,0.3);
  position: sticky;
  top: 0;
  z-index: 1000;
}

.nav-container {
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.logo {
  font-size: 1.8em;
  font-weight: 700;
  color: var(--color-text-light);
}
.logo .logo-accent1 { color: var(--color-primary-orange); }
.logo .logo-accent2 { color: var(--color-secondary-lime); }
.logo:hover { color: var(--color-text-light); }


.main-nav ul {
  list-style: none;
  padding: 0;
  margin: 0;
  display: flex;
}

.main-nav li {
  margin-left: 20px;
}

.main-nav a {
  font-weight: 600;
  padding: 0.5em 0.8em;
  border-radius: var(--border-radius);
  transition: background-color 0.3s ease, color 0.3s ease;
}

.main-nav a:hover, .main-nav a.active {
  background-color: var(--color-primary-orange);
  color: var(--color-dark-bg);
}


.site-main {
  flex-grow: 1;
  padding: 2em 0;
}


.site-footer {
  background-color: var(--color-dark-bg-secondary);
  color: var(--color-text-medium);
  padding: 2em 0;
  text-align: center;
  border-top: 2px solid var(--color-secondary-lime);
  margin-top: auto;
}
.site-footer p {
  margin: 0.3em 0;
}
```

```
.btn {
  display: inline-block;
  padding: 0.8em 1.5em;
  border: none;
  border-radius: var(--border-radius);
  font-family: var(--font-primary);
  font-weight: 600;
  cursor: pointer;
  transition: all 0.3s ease;
  text-align: center;
  font-size: 1em;
}

.btn-primary {
  background-color: var(--color-primary-orange);
  color: var(--color-dark-bg);
  box-shadow: 0 2px 5px rgba(0,0,0,0.2);
}
.btn-primary:hover {
  background-color: var(--color-primary-orange-hover);
  box-shadow: var(--neon-orange-glow);
  transform: translateY(-2px);
}

.btn-secondary {
  background-color: transparent;
  color: var(--color-secondary-lime);
  border: 2px solid var(--color-secondary-lime);
}
.btn-secondary:hover {
  background-color: var(--color-secondary-lime);
  color: var(--color-dark-bg);
  box-shadow: var(--neon-lime-glow);
  transform: translateY(-2px);
}

.btn-block {
  display: block;
  width: 100%;
}
.btn-sm {
  padding: 0.5em 1em;
  font-size: 0.9em;
}



.form-container {
  background-color: var(--color-dark-bg-secondary);
  padding: 2em;
  border-radius: var(--border-radius);
  box-shadow: var(--card-shadow);
  max-width: 600px;
  margin: 2em auto;
}
.form-container h2 {
  text-align: center;
  margin-bottom: 1.5em;
}

.form-group {
  margin-bottom: 1.5em;
}

.form-label, label {
  display: block;
  margin-bottom: 0.5em;
  font-weight: 600;
  color: var(--color-text-medium);
}

.form-control {
  width: 100%;
  padding: 0.8em 1em;
  background-color: var(--color-dark-bg);
  border: 1px solid var(--color-border);
  border-radius: var(--border-radius);
  color: var(--color-text-light);
```

```
    font-family: var(--font-primary);
    box-sizing: border-box;
    transition: border-color 0.3s ease, box-shadow 0.3s ease;
}
.form-control:focus {
    outline: none;
    border-color: var(--color-primary-orange);
    box-shadow: 0 0 0 3px rgba(255, 111, 0, 0.3);
}
textarea.form-control {
    min-height: 120px;
    resize: vertical;
}
select.form-control {
    appearance: none;
    background-image: url(«data:image/svg+xml,%3Csvg xmlns='http://www.w3.org/2000/svg' viewBox='0 0 16 16'%3E%3Cpath fill='none' stroke='%23ABFF4F' stroke-linecap='round' stroke-linejoin='round' stroke-width='2' d='M2 5l6 6 6-6'/%3E%3C/svg%3E»);
    background-repeat: no-repeat;
    background-position: right 1em center;
    background-size: 1em;
    padding-right: 2.5em;
}


.auth-switch {
    text-align: center;
    margin-top: 1.5em;
    color: var(--color-text-medium);
}
.auth-switch a {
    font-weight: 600;
}

.flash-messages {
    margin-bottom: 1.5em;
}
.flash {
    padding: 1em;
    margin-bottom: 1em;
    border-radius: var(--border-radius);
    color: var(--color-dark-bg);
    text-align: center;
}
.flash-success { background-color: var(--color-success); }
.flash-error { background-color: var(--color-error); }
.flash-info { background-color: var(--color-info); color: var(--color-text-light); }
.flash-warning { background-color: var(--color-warning); }



.hero-section {
    text-align: center;
    padding: 4em 0;
    background: linear-gradient(rgba(13, 13, 13, 0.8), rgba(13, 13, 13, 0.9)), url(«https://placehold.co/1200x400/1a1a1a/FF6F00?text=Workplace») no-repeat center center/cover;
    border-radius: var(--border-radius);
    margin-bottom: 3em;
}
.hero-section h1 {
    font-size: 3.5em;
    margin-bottom: 0.3em;
    font-weight: 700;
}
.hero-section .subtitle {
    font-size: 1.3em;
    color: var(--color-text-medium);
    margin-bottom: 1.5em;
```

```
}


.search-form {
  display: flex;
  gap: 1em;
  justify-content: center;
  align-items: center;
  margin-top: 2em;
  flex-wrap: wrap;
}
.search-form input[type=«text»] {
  padding: 0.8em 1.2em;
  border: 1px solid var(--color-
border);
  border-radius: var(--border-radius);
  background-color: var(--color-dark-
bg-secondary);
  color: var(--color-text-light);
  min-width: 250px;
  flex-grow: 1;
}
.search-form input[type=«text»]:focus
{
  border-color: var(--color-
secondary-lime);
  box-shadow: 0 0 0 3px rgba(171,
255, 79, 0.3);
}
.search-form .btn-primary {
  padding: 0.8em 2em;
}


.job-listings h2 {
  text-align: center;
  margin-bottom: 1.5em;
}
.job-grid {
  display: grid;
  grid-template-columns: repeat(auto-
fit, minmax(320px, 1fr));
  gap: 1.5em;
}

.job-card {
  background-color: var(--color-dark-
bg-secondary);
  border-radius: var(--border-radius);
  padding: 1.5em;
  box-shadow: var(--card-shadow);
  transition: transform 0.3s ease, box-
shadow 0.3s ease;
  display: flex;
  flex-direction: column;
  border-left: 4px solid var(--color-
secondary-lime);
}
.job-card:hover {
  transform: translateY(-5px)
scale(1.02);
  box-shadow: 0 8px 25px
rgba(0,0,0,0.3), var(--neon-lime-
glow);
}

.job-card-header h3 {
  margin-bottom: 0.2em;
  font-size: 1.4em;
}
.job-card-header h3 a {
  color: var(--color-primary-orange);
}
.job-card-header h3 a:hover {
  text-decoration: underline;
}
.job-card .company-name {
  color: var(--color-text-medium);
  font-size: 0.95em;
  display: block;
  margin-bottom: 0.8em;
}

.job-card-body p {
  margin-bottom: 0.5em;
  font-size: 0.95em;
  display: flex;
  align-items: center;
```

```
}
.job-card-body .icon {
  margin-right: 0.5em;
  color: var(--color-secondary-lime);
}

.job-card-footer {
  margin-top: auto;
  padding-top: 1em;
  border-top: 1px solid var(--color-
border);
  display: flex;
  justify-content: space-between;
  align-items: center;
  font-size: 0.9em;
}
.job-card .posted-date {
  color: var(--color-text-medium);
}
.no-jobs {
  text-align: center;
  padding: 2em;
  background-color: var(--color-dark-
bg-secondary);
  border-radius: var(--border-radius);
  color: var(--color-text-medium);
}


.job-detail-container {
  background-color: var(--color-dark-
bg-secondary);
  padding: 2em;
  border-radius: var(--border-radius);
  box-shadow: var(--card-shadow);
}
.job-detail-header {
  border-bottom: 2px solid var(--
color-primary-orange);
  margin-bottom: 2em;
  padding-bottom: 1em;
}
.job-detail-header h1 {
  font-size: 2.5em;
  margin-bottom: 0.2em;
}
.job-detail-header .company-info {
  font-size: 1.1em;
  color: var(--color-text-medium);
}
.job-detail-header .company-info
.separator { margin: 0 0.5em; }
.job-detail-header .company-info .icon
{ color: var(--color-secondary-lime);
margin-right: 0.3em; }

.job-detail-main {
  display: flex;
  gap: 2em;
  flex-wrap: wrap;
}
.job-detail-description {
  flex: 3;
  min-width: 300px;
}
.job-detail-description h2 {
  margin-bottom: 0.8em;
}
.job-detail-description p {
  white-space: pre-wrap;
  line-height: 1.8;
}

.job-detail-sidebar {
  flex: 1;
  background-color: var(--color-dark-
bg);
  padding: 1.5em;
  border-radius: var(--border-radius);
  border-left: 3px solid var(--color-
secondary-lime);
  min-width: 280px;
}
.job-detail-sidebar h3 {
  margin-bottom: 1em;
  font-size: 1.3em;
}
.job-detail-sidebar ul {
```

```
    list-style: none;
    padding: 0;
    margin-bottom: 1.5em;
}
.job-detail-sidebar li {
    margin-bottom: 0.8em;
    font-size: 0.95em;
}
.job-detail-sidebar li strong {
    color: var(--color-text-light);
}
.job-detail-sidebar .btn-apply {
    width: 100%;
}
.back-link-container {
    margin-top: 2em;
    text-align: center;
}
.dashboard-section h2, .dashboard-section h3 {
    text-align: center;
    margin-bottom: 1em;
}
.dashboard-jobs .job-card {
    border-left-color: var(--color-primary-orange);
}
.dashboard-jobs .job-card:hover {
    box-shadow: 0 8px 25px rgba(0,0,0,0.3), var(--neon-orange-glow);
}
.dashboard-actions {
    margin-top: 2em;
    display: flex;
    gap: 1em;
    justify-content: center;
    flex-wrap: wrap;
}
@media (max-width: 768px) {
    .nav-container {
        flex-direction: column;
        align-items: flex-start;
    }
    .main-nav ul {
        flex-direction: column;
        width: 100%;
        margin-top: 1em;
    }
    .main-nav li {
        margin-left: 0;
        margin-bottom: 0.5em;
        width: 100%;
    }
    .main-nav a {
        display: block;
        text-align: center;
    }
    .hero-section h1 { font-size: 2.5em;
}
    .hero-section .subtitle { font-size: 1.1em; }
    .search-form { flex-direction: column; }
    .search-form input[type=«text»], .search-form .btn-primary {
        width: 100%;
        min-width: unset;
    }
    .job-detail-main {
        flex-direction: column;
    }
    .job-detail-sidebar {
        margin-top: 2em;
    }
}
@media (max-width: 480px) {
    .container { width: 95%; }
    h1 { font-size: 2em; }
    h2 { font-size: 1.8em; }
    .hero-section h1 { font-size: 2em; }
    .job-card { padding: 1em; }
    .job-card-header h3 { font-size: 1.2em; }
    .form-container { padding: 1.5em; }
}
```

ДОДАТОК Г

ЛІСТИНГ ВЕБСТОРІНОК

```
<!DOCTYPE html>
<html lang=«uk»>
<head>
  <meta charset=«UTF-8»>
  <meta name=«viewport» content=«width=device-width, initial-scale=1.0»>
  <title>{% block title %} Пошук Роботи{% endblock %}</title>
  <link href=«[https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;600;700&display=swap](https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;600;700&display=swap)» rel=«stylesheet»>
  <link rel=«stylesheet» href=«{{ url_for(‘static’, filename=‘css/style.css’) }}»>
  {% block head_extra %}{% endblock %}
</head>
<body>
  <header class=«site-header»>
    <div class=«container nav-container»>
      <a href=«{{ url_for(‘index’) }}» class=«logo»> <span class=«logo-accent1»> </span><span class=«logo-accent2»>Jobs</span></a>
      <nav class=«main-nav»>
        <ul>
          <li><a href=«{{ url_for(‘index’) }}»>Вакансії</a></li>
          {% if current_user.is_authenticated %}
            <li><a href=«{{ url_for(‘dashboard’) }}»>Панель</a></li>
            {% if current_user.role == ‘employer’ %}
              <li><a href=«{{ url_for(‘post_job’) }}»>Опублікувати вакансію</a></li>
            {% endif %}
            <li><a href=«{{ url_for(‘logout’) }}»>Вийти ({{ current_user.username }})</a></li>
          {% else %}
            <li><a href=«{{ url_for(‘login’) }}»>Увійти</a></li>
            <li><a href=«{{ url_for(‘register’) }}»>Реєстрація</a></li>
          {% endif %}
        </ul>
      </nav>
    </div>
  </header>

  <main class=«site-main»>
    <div class=«container»>
      {% with messages = get_flashed_messages(with_categories=true) %}
        {% if messages %}
          <div class=«flash-messages»>
            {% for category, message in messages %}
              <div class=«flash flash-{{ category }}»>{{ message }}</div>
```

```
                {% endfor %}
              </div>
            {% endif %}
          {% endwith %}
          {% block content %}{% endblock %}
        </div>
    </main>

    <footer class=«site-footer»>
        <div class=«container»>
            <p>&copy; {{ SCRIPT_YEAR }} Bloom Jobs. Всі права захищено.</p>
            <p>Дизайн, що виходить за межі буденності.</p>
        </div>
    </footer>
    <script>


document.addEventListener('DOMContentLoaded', function() {
        const yearSpan = document.querySelector('.site-footer p:first-child');
        if (yearSpan) {
            yearSpan.innerHTML = yearSpan.innerHTML.replace('{{ SCRIPT_YEAR }}', new Date().getFullYear());
        }
    });
    </script>
    {% block scripts_extra %}{% endblock %}
</body>
</html>


{% macro render_field(field, icon_class=None) %}
  <div class=«form-group»>
    {{ field.label(class=«form-label») }}
    {% if icon_class %}
      <div class=«input-group»>
        <span class=«input-group-icon»><i class=«{{ icon_class }}»></i></span> {{ field(class=«form-control « + («is-invalid» if field.errors else ««), **kwargs) }}
      </div>
    {% else %}
      {{ field(class=«form-control « + («is-invalid» if field.errors else ««), **kwargs) }}
    {% endif %}
    {% if field.errors %}
      <ul class=«errors»>
      {% for error in field.errors %}
        <li class=«error-message»>{{ error }}</li>
      {% endfor %}
      </ul>
    {% endif %}
  </div>
{% endmacro %}


{% extends «layout.html» %}

{% block title %}Головна - Вакансії{% endblock %}

{% block content %}
<section class=«hero-section»>
    <h1>Знайди <span class=«highlight-orange»>свою наступну</span> кар'єрну можливість</h1>
    <p class=«subtitle»>Платформа, що поєднує таланти та інноваційні компанії.</p>
```

```
  <form method=«GET» action=«{{ url_for('index') }}» class=«search-form»>
    <input type=«text» name=«search» placeholder=«Ключове слово, посада, компанія...» value=«{{ search_query or '' }}»>
    <input type=«text» name=«location» placeholder=«Місто або регіон» value=«{{ location_query or '' }}»>
    <button type=«submit» class=«btn btn-primary»>Пошук</button>
  </form>
</section>

<section class=«job-listings»>
  <h2>Останні <span class=«highlight-lime»>вакансії</span></h2>
  {% if jobs %}
    <div class=«job-grid»>
      {% for job in jobs %}
      <div class=«job-card»>
        <div class=«job-card-header»>
          <h3><a href=«{{ url_for('job_detail', job_id=job.id) }}»>{{ job.title }}</a></h3>
          <span class=«company-name»>{{ job.company_name_display }}</span>
        </div>
        <div class=«job-card-body»>
          <p class=«location»><span class=«icon»>⚲</span> {{ job.location or 'Не вказано' }}</p>
          <p class=«job-type»><span class=«icon»>💼</span> {{ job.job_type or 'Не вказано' }}</p>
          {% if job.salary_range %}
          <p class=«salary»><span class=«icon»>💰</span> {{ job.salary_range }}</p>
          {% endif %}
        </div>
        <div class=«job-card-footer»>
          <span class=«posted-date»>Опубліковано: {{ job.posted_date | datetimeformat('%d.%m.%Y') }}</span>
          <a href=«{{ url_for('job_detail', job_id=job.id) }}» class=«btn btn-secondary»>Детальніше</a>
        </div>
      </div>
      {% endfor %}
    </div>
  {% else %}
    <p class=«no-jobs»>Наразі немає активних вакансій, що відповідають вашому запиту. Спробуйте змінити параметри пошуку.</p>
  {% endif %}
</section>
{% endblock %}


{% extends «layout.html» %}

{% block title %}Реєстрація{% endblock %}

{% block content %}
<div class=«form-container auth-form»>
```

```
<h2>Створити <span class=«highlight-orange»>акаунт</span></h2>
<form method=«POST» action=«{{ url_for(‘register’) }}»>
<div class=«form-group»>
<label for=«username»>Ім’я користувача:</label>
<input type=«text» id=«username» name=«username» required class=«form-control» value=«{{ request.form.username }}»>
</div>
<div class=«form-group»>
<label for=«email»>Електронна пошта:</label>
<input type=«email» id=«email» name=«email» required class=«form-control» value=«{{ request.form.email }}»>
</div>
<div class=«form-group»>
<label for=«password»>Пароль:</label>
<input type=«password» id=«password» name=«password» required class=«form-control»>
</div>
<div class=«form-group»>
<label for=«confirm_password»>Підтвердіть пароль:</label>
<input type=«password» id=«confirm_password» name=«confirm_password» required class=«form-control»>
</div>
<div class=«form-group»>
<label for=«role»>Я:</label>
<select id=«role» name=«role» required class=«form-control» onchange=«toggleCompanyName(this.value)»>
<option value=«candidate» {% if request.form.role == ‘candidate’ %}selected{% endif %}>Кандидат</option>
<option value=«employer» {% if request.form.role == ‘employer’ %}selected{% endif %}>Роботодавець</option>
</select>
</div>
<div class=«form-group» id=«company_name_group» style=«display: {% if request.form.role == ‘employer’ %}block{% else %}none{% endif %};»>
<label for=«company_name»>Назва компанії (для роботодавців):</label>
<input type=«text» id=«company_name» name=«company_name» class=«form-control» value=«{{ request.form.company_name }}»>
</div>
<button type=«submit» class=«btn btn-primary btn-block»>Зареєструватися</button>
</form>
<p class=«auth-switch»>Вже маєте акаунт? <a href=«{{ url_for(‘login’) }}»>Увійти</a></p>
</div>

<script>
function toggleCompanyName(role) {
const companyNameGroup = document.getElementById(‘company_name_group’);
if (role === ‘employer’) {
```

```
companyNameGroup.style.display = 'block';

document.getElementById('company_name').required = true;
        } else {

companyNameGroup.style.display = 'none';

document.getElementById('company_name').required = false;
        }
    }


document.addEventListener('DOMContentLoaded', function() {
        const roleSelect = document.getElementById('role');
        if (roleSelect) {

toggleCompanyName(roleSelect.value);
        }
    });
</script>
{% endblock %}


{% extends «layout.html» %}

{% block title %}Вхід{% endblock %}

{% block content %}
<div class=«form-container auth-form»>
    <h2>Вхід в <span class=«highlight-lime»>систему</span></h2>
    <form method=«POST» action=«{{ url_for('login') }}»>
        <div class=«form-group»>
            <label for=«email»>Електронна пошта:</label>
            <input type=«email» id=«email» name=«email» required class=«form-control» value=«{{ request.form.email }}»>
        </div>
        <div class=«form-group»>
            <label for=«password»>Пароль:</label>
            <input type=«password» id=«password» name=«password» required class=«form-control»>
        </div>
        <button type=«submit» class=«btn btn-primary btn-block»>Увійти</button>
    </form>
    <p class=«auth-switch»>Немає акаунту? <a href=«{{ url_for('register') }}»>Зареєструватися</a></p>
</div>
{% endblock %}


{% extends «layout.html» %}

{% block title %}Опублікувати вакансію{% endblock %}

{% block content %}
<div class=«form-container post-job-form»>
    <h2>Опублікувати нову <span class=«highlight-orange»>вакансію</span></h2>
    <form method=«POST» action=«{{ url_for('post_job') }}»>
        <div class=«form-group»>
            <label for=«title»>Назва вакансії:</label>
```

```
            <input type=«text» id=«title» name=«title» required class=«form-control» value=«{{ request.form.title }}»>
        </div>
        <div class=«form-group»>
            <label for=«description»>Опис вакансії:</label>
            <textarea id=«description» name=«description» rows=«8» required class=«form-control»>{{ request.form.description }}</textarea>
        </div>
        <div class=«form-group»>
            <label for=«location»>Місцезнаходження:</label>
            <input type=«text» id=«location» name=«location» class=«form-control» value=«{{ request.form.location }}»>
        </div>
        <div class=«form-group»>
            <label for=«salary_range»>Діапазон заробітної плати (наприклад, 50000 - 70000 грн):</label>
            <input type=«text» id=«salary_range» name=«salary_range» class=«form-control» value=«{{ request.form.salary_range }}»>
        </div>
        <div class=«form-group»>
            <label for=«job_type»>Тип зайнятості:</label>
            <select id=«job_type» name=«job_type» class=«form-control»>
                <option value=«Повна зайнятість» {% if request.form.job_type == ‘Повна зайнятість’ %}selected{% endif %}>Повна зайнятість</option>
                <option value=«Часткова зайнятість» {% if request.form.job_type == ‘Часткова зайнятість’ %}selected{% endif %}>Часткова зайнятість</option>
                <option value=«Контракт» {% if request.form.job_type == ‘Контракт’ %}selected{% endif %}>Контракт</option>
                <option value=«Віддалена робота» {% if request.form.job_type == ‘Віддалена робота’ %}selected{% endif %}>Віддалена робота</option>
                <option value=«Стажування» {% if request.form.job_type == ‘Стажування’ %}selected{% endif %}>Стажування</option>
            </select>
        </div>
        <div class=«form-group»>
            <label for=«company_name_display»>Назва компанії для відображення (за замовчуванням - з вашого профілю):</label>
            <input type=«text» id=«company_name_display» name=«company_name_display» class=«form-control» value=«{{ request.form.company_name_display or current_user.company_name }}»>
        </div>
        <button type=«submit» class=«btn btn-primary btn-block»>Опублікувати вакансію</button>
    </form>
</div>
{% endblock %}
```

```
{% extends «layout.html» %}

{% block title %}{{ job.title }} - Деталі вакансії{% endblock %}

{% block content %}
<div class=«job-detail-container»>
  <div class=«job-detail-header»>
    <h1>{{ job.title }}</h1>
    <p class=«company-info»>
      <span class=«company-name»>{{ job.company_name_display }}</span>
      {% if job.location %}
      <span class=«separator»>|</span> <span class=«location»><span class=«icon»> ⚲ </span> {{ job.location }}</span>
      {% endif %}
    </p>
  </div>

  <div class=«job-detail-main»>
    <div class=«job-detail-description»>
      <h2>Опис <span class=«highlight-orange»>вакансії</span></h2>
      <p>{{ job.description | safe }}</p> {
    </div>

    <aside class=«job-detail-sidebar»>
      <h3>Інформація про <span class=«highlight-lime»>вакансію</span></h3>
      <ul>
        {% if job.job_type %}<li><strong>Тип зайнятості:</strong> {{ job.job_type }}</li>{% endif %}
        {% if job.salary_range %}<li><strong>Заробітна плата:</strong> {{ job.salary_range }}</li>{% endif %}

        <li><strong>Опубліковано:</strong> {{ job.posted_date | datetimeformat }}</li>

        <li><strong>Роботодавець:</strong> {{ job.employer_username }}</li>
        <li><strong>Контакт (email):</strong> <a href=«mailto:{{ job.employer_email }}»>{{ job.employer_email }}</a></li>

      </ul>
      <a href=«mailto:{{ job.employer_email }}?subject=Відгук на вакансію: {{ job.title }}» class=«btn btn-primary btn-apply»>Відгукнутися</a>
    </aside>
  </div>

  <div class=«back-link-container»>
    <a href=«{{ url_for('index') }}» class=«btn btn-secondary»>&larr; Повернутися до списку вакансій</a>
  </div>
</div>
{% endblock %}


{% extends «layout.html» %}

{% block title %}Панель керування{% endblock %}

{% block content %}
<section class=«dashboard-section»>
```

```
  <h2>Вітаємо, <span class=«highlight-orange»>{{ current_user.username }}</span>!</h2>
  <p>Це ваша персональна панель керування.</p>

  {% if current_user.role == ‘employer’ %}
    <h3>Ваші <span class=«highlight-lime»>опубліковані вакансії</span></h3>
    {% if user_jobs %}
      <div class=«job-grid dashboard-jobs»>
        {% for job in user_jobs %}
        <div class=«job-card»>
          <div class=«job-card-header»>
            <h3><a href=«{{ url_for(‘job_detail’, job_id=job.id) }}»>{{ job.title }}</a></h3>
            <span class=«company-name»>{{ job.company_name_display }}</span>
          </div>
          <div class=«job-card-body»>
            <p class=«location»><span class=«icon»>📍</span> {{ job.location or ‘Не вказано’ }}</p>
            <p class=«job-type»><span class=«icon»>💼</span> {{ job.job_type or ‘Не вказано’ }}</p>
            <p class=«status»>Статус: {% if job.is_active %}Активна{% else %}Неактивна{% endif %}</p>
          </div>
          <div class=«job-card-footer»>
            <span class=«posted-date»>Опубліковано: {{ job.posted_date | datetimeformat(‘%d.%m.%Y’) }}</span>
            {
            <a href=«{{ url_for(‘job_detail’, job_id=job.id) }}» class=«btn btn-secondary btn-sm»>Переглянути</a>
          </div>
        </div>
        {% endfor %}
      </div>
    {% else %}
      <p>Ви ще не опублікували жодної вакансії. <a href=«{{ url_for(‘post_job’) }}»>Створити нову?</a></p>
    {% endif %}
  {% elif current_user.role == ‘candidate’ %}
  {% endif %}

  <div class=«dashboard-actions»>
    <a href=«{{ url_for(‘index’) }}» class=«btn btn-secondary»>Перейти до пошуку вакансій</a>
    {% if current_user.role == ‘employer’ %}
    <a href=«{{ url_for(‘post_job’) }}» class=«btn btn-primary»>Опублікувати нову вакансію</a>
    {% endif %}
  </div>
</section>
{% endblock %}
```