Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

ВЕБЗАСТОСУНОК ДЛЯ АВТОМАТИЗАЦІЇ ПІДБОРУ ПЕРСОНАЛУ

WEB APPLICATION FOR AUTOMATING RECRUITMENT

спеціальність 122 Комп'ютерні науки

освітня програма «Комп'ютерні науки»

Виконав: здобувач вищої освіти
групи КН-42
Бондарчук Ілля Віталійович

_____
(підпис)

Керівник: д.т.н., професор
Гордєєв Олександр Олександрович

_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«2» червня 2026 р.
Гарант ОП: д.п.н., професор
Тулашвілі Юрій Йосипович

_____
(підпис)

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерних наук*
Ступінь вищої освіти: *бакалавр*
Галузь знань*: 12 Інформаційні технології*
Спеціальність: *122 Комп'ютерні науки*
Освітня програма: *«Комп'ютерні науки»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Валерій ЛІЩИНА
«16» січня 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
ПЕРШОГО (БАКАЛАВРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Бондарчук Ілля Віталійович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Вебзастосунок для автоматизації підбору персоналу*
Керівник: *д.т.н., професор Гордєєв Олександр Олександрович*
затверджено наказом закладу вищої освіти від «16» січня 2026 р. № 32/01-02
2. Строк подання кваліфікаційної роботи *«02» червня 2026 р.*
3.Вихідні дані до роботи: *аналітичне дослідження систем управління проектами.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Вступ
Розділ 1. Аналіз предметної області
Розділ 2. Структурна реалізація об'єкта проектування
Розділ 3. Програмна реалізація об'єкта проектування
Розділ 4. Спеціальні розрахунки
Висновки
Додатки
5. Перелік графічного матеріалу: _____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми за темою роботи та постановка завдань дослідження	*Ліщина В. О.*		
Теоретичне дослідження	*Ліщина В. О.*		
Практична реалізація	*Ліщина В. О.*		
Спеціальні розрахунки	*Ліщина В. О.*		
Показник запозичень тексту	_____%		
Інструментальна перевірка	*Кошелюк В. А.*		
Нормоконтроль	*Сачук В. О.*		
Гарант ОПП	*Тулашвілі Ю.Й.*		

7. Дата видачі завдання *«16» січня 2026 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної Роботи	Строк виконання етапів роботи	Примітка
1.	*Провести огляд літературних джерел по темі кваліфікаційної роботи*	*до 10.02.2026р*	
2.	*Провести аналіз загальної проблеми і вибір напрямків дослідження*	*до 24.02.2026р.*	
3.	*Розробити функціональну схему роботи програмного продукту*	*до 10.03.2026р*	
4.	*Описати засоби розробки об'єкта проектування*	*до 31.03.2026р.*	
5.	*Практична реалізація об'єкта проектування*	*до 05.05.2026р.*	
7.	*Провести спеціальні розрахунки*	*до 21.05.2026р.*	
8.	*Здача чистового варіанту кваліфікаційної роботи на кафедру*	*до 02.06.2026р.*	

Здобувач вищої освіти _____ Ілля БОНДАРЧУК

Керівник роботи _____ Олександр ГОРДЄЄВ

АНОТАЦІЯ

Бондарчук І. В. Вебзастосунок для автоматизації підбору персоналу. Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп’ютерні науки». Луцький національний технічний університет. Луцьк, 2026.

У роботі розглянуто вебзастосунок для автоматизації підбору персоналу. Рішення призначене для автоматизації процесів, пов’язаних із обробленням резюме, зіставленням компетентностей і первинним відбором кандидатів.

Ключові слова: рекрутинг, вебзастосунок, кандидат, вакансія, автоматизація відбору.

ABSTRACT

Illia Bondarchuk. Web application for automating recruitment. Manuscript.

Bachelor’s qualifying thesis of the OP "Computer Sciences". Lutsk National Technical University. Lutsk, 2026.

The thesis considers a web application for automating recruitment intended to automate processes related to resume processing, skill matching, and initial candidate screening. The relevance of the topic is substantiated, and the main system requirements, architecture, data model, and user scenarios are defined. The implemented functionality includes vacancies, candidate profiles, resumes, filtering, matching score, and interview stages. The main scenarios were tested, and an approach to deployment was prepared. The practical value of the work lies in the ability to reduce recruitment time and make candidate assessment more structured.

Keywords: recruitment, web application, candidate, vacancy, screening automation.

ЗМІСТ

ВСТУП

Актуальність теми зумовлена потребою рекрутингових команд швидко обробляти вакансії, резюме й етапи відбору без втрати інформації. Сучасні програмні системи мають не лише автоматизувати окремі операції, а й забезпечувати цілісну підтримку користувача: від введення початкових даних до отримання зрозумілого результату, контролю виконання дій та подальшого використання накопиченої інформації. У таких умовах особливого значення набувають веборієнтовані рішення, які доступні з різних пристроїв, легко масштабуються та можуть бути інтегровані з іншими сервісами.

Практична потреба у розробленні такої системи пояснюється тим, що ринок праці характеризується високою оперативністю зміни вимог до фахівців, великою кількістю відгуків на вакансії та потребою роботодавців у прозорому порівнянні кандидатів. За відсутності спеціалізованого програмного інструмента значна частина роботи виконується вручну або за допомогою універсальних сервісів, які не враховують специфіку предметної області. Це призводить до втрати часу, дублювання інформації, складності контролю актуального стану даних і зниження якості прийняття рішень. Тому створення цільового застосунку є обґрунтованим як з погляду оптимізації користувацьких процесів, так і з погляду набуття практичних навичок проєктування сучасних програмних продуктів.

Метою кваліфікаційної роботи є скорочення трудомісткості первинного відбору кандидатів шляхом розроблення вебзастосунку, що підтримує облік вакансій, резюме, критеріїв відповідності та результатів оцінювання. Досягнення цієї мети передбачає не лише реалізацію окремих функцій, а й обґрунтування архітектури системи, визначення ролей користувачів, опис інформаційних потоків, вибір технологічного стеку та перевірку працездатності створеного рішення на типових сценаріях використання.

Об’єктом дослідження є процеси пошуку, оцінювання та попереднього відбору кандидатів на вакантні посади. У межах роботи цей об’єкт розглядається

як сукупність взаємопов’язаних дій користувача, інформаційних сутностей, правил обробки даних і результатів, які повинні бути відображені у програмній системі.

Предметом дослідження є алгоритми, моделі даних і програмні засоби автоматизації рекрутингових процедур у вебсередовищі. Предметна область потребує врахування функціональних вимог, обмежень безпеки та ергономічності користування, способів зберігання даних, логіки взаємодії між клієнтською та серверною частинами, а також засобів перевірки правильності роботи основних модулів. Саме ці складові визначають якість майбутнього програмного продукту та його придатність до практичного застосування.

Для досягнення поставленої мети необхідно розв’язати такі основні завдання: проаналізувати сучасний стан предметної області та наявні підходи до вирішення подібних задач; визначити потреби цільових користувачів і сформувати перелік функціональних вимог; розробити структуру даних і логіку взаємодії компонентів; спроєктувати користувацький інтерфейс; реалізувати основні програмні модулі; провести перевірку роботи системи на типових сценаріях; сформулювати висновки щодо ефективності запропонованого рішення.

Потрібно реалізувати функції створення профілів вакансій, завантаження резюме, фільтрацію за навичками, формування рейтингу відповідності, ведення історії комунікацій і підтримку рішень рекрутера. Зазначені функції повинні працювати узгоджено, забезпечувати логічний шлях користувача та створювати основу для подальшого розвитку програмного продукту. Особливу увагу необхідно приділити простоті введення даних, зрозумілості результатів і зменшенню кількості надлишкових дій під час виконання основних операцій.

Методологічною основою роботи є методи аналізу вимог, структуризації даних, ранжування альтернатив, розробки клієнт-серверних вебсистем, проєктування баз даних та тестування сценаріїв користувача. Використання цих методів дає змогу послідовно пройти етапи від аналізу проблеми до створення працездатного програмного рішення. На етапі проєктування важливо визначити

інформаційні сутності, зв’язки між ними, сценарії використання та обмеження системи. На етапі реалізації необхідно забезпечити коректну взаємодію компонентів, обробку помилкових ситуацій і достатній рівень надійності під час виконання основних операцій.

Практична цінність отриманого результату полягає в тому, що система може бути корисною для малих і середніх компаній, навчальних центрів та кадрових служб, які потребують простого інструмента попереднього відбору персоналу. Запропоноване рішення має забезпечити зменшення рутинних дій, підвищення прозорості роботи з даними та створення більш зручного цифрового середовища для кінцевого користувача.

РОЗДІЛ 1
АНАЛІЗ ПРОЦЕСУ ПІДБОРУ ПЕРСОНАЛУ

1.1 Рекрутинг як предмет автоматизації

Підбір персоналу розглядається як послідовний інформаційний процес, що охоплює створення вакансії, отримання резюме, попередній аналіз кандидата, фіксацію етапу відбору та прийняття рекрутингового рішення. Для програмної реалізації важливо подати цей процес через чіткі сутності, стани та правила взаємодії користувача із системою.

Рекрутинг є послідовністю взаємопов’язаних дій формування вакансії, отримання резюме, попередній відбір, оцінювання відповідності, комунікація з кандидатом і фіксація рішення. У ручному режимі ці дії часто виконуються у різних таблицях, поштових скриньках і месенджерах, через що рекрутер втрачає повну картину воронки. Автоматизація має об’єднати інформацію в єдиному процесі, де кожен кандидат має поточний етап, джерело, набір навичок і історію взаємодії.

Предметна область включає декілька ролей. Рекрутер створює вакансії, додає кандидатів і веде комунікацію; HR-менеджер переглядає воронку та контролює якість процесу; керівник підрозділу оцінює релевантних кандидатів; адміністратор відповідає за довідники, права доступу й службові налаштування. Розмежування ролей важливе, оскільки резюме містять персональні дані та не повинні бути доступними всім користувачам системи.

Автоматизація первинного відбору не повинна підміняти професійне рішення рекрутера. Рейтинг відповідності є допоміжним показником, який структурує чергу опрацювання й пояснює, чому кандидат потрапив вище або нижче у списку. Остаточне рішення залишається за людиною, а система повинна фіксувати підстави зміни етапу та результат комунікації.

Основними ризиками ручного підбору є дублювання анкет, суб’єктивне зіставлення навичок, пропуск сильних кандидатів і затримка відповідей. Вебзастосунок зменшує ці ризики через уніфіковані профілі, фільтри, контроль

обов’язкових полів, журнал змін і автоматичне виявлення повторів за електронною поштою або телефоном.

1.2 Огляд ATS-систем і сервісів керування кандидатами

ATS-системи призначені для керування повним циклом роботи з кандидатами. Публікації вакансій, збору відгуків, парсингу резюме, комунікації та аналітики. Для порівняння обрано Greenhouse, Workable, Hurma та універсальні інструменти Jira/Notion, оскільки вони представляють різні підходи, корпоративну ATS, хмарний рекрутинговий сервіс, HRM/ATS для малого й середнього бізнесу та гнучкі системи загального призначення [1-2].

Greenhouse орієнтована на зрілі рекрутингові процеси й підтримує інтеграції, аналітику та налаштування етапів відбору. Workable сильний у швидкому запуску вакансій, публікації на job-платформах і командній роботі з кандидатами. Hurma поєднує HRM та ATS-функції, що корисно для компаній, яким потрібен єдиний контур персоналу. Jira або Notion можна адаптувати під воронку, але вони не мають спеціалізованого парсингу резюме та рекрутингової аналітики без додаткових налаштувань.

Порівняльна характеристика в таблиці 1.1 показує, що готові ATS мають сильну функціональність, проте можуть бути надмірними або дорогими для невеликої організації. Для навчального прототипу доцільно взяти з них базові принципи: етапи воронки, профіль кандидата, фільтрацію, оцінку відповідності та журнал дій.

Ключовим висновком огляду є потреба у простій системі, яка не повторює всі можливості комерційних ATS, але покриває первинний добір. Такий прототип має підтримувати вакансії, кандидатів, завантаження резюме, нормалізацію навичок, рейтинг відповідності, коментарі та базовий звіт по воронці.

Таблиця 1.1 – Порівняння ATS-систем і сервісів керування кандидатами

Система	Сильні сторони	Обмеження	API та інтеграції	Парсинг резюме й вартість
Greenhouse	Розвинена воронка, аналітика, командні оцінки	Надмірна складність для малого прототипу	Широкі інтеграції та API	Підтримується через інтеграції; висока вартість
Workable	Швидка публікація вакансій, зручна робота з кандидатами	Частина функцій залежить від тарифу	API та інтеграції з job-платформами	Є засоби імпорту резюме; комерційна модель
Hurma	Поєднання HRM та ATS, придатність для малого бізнесу	Менша гнучкість складних enterprise-процесів	Інтеграції з HR-інструментами	Підтримує рекрутингові картки; вартість нижча за enterprise ATS
Jira / Notion	Гнучкі дошки, просте налаштування процесу	Немає спеціалізованої ATS-логіки без ручної конфігурації	API доступний, але потребує власної моделі	Парсинг резюме відсутній; вартість залежить від команди

На відміну від універсальних дошок задач, рекрутинг потребує спеціальних обмежень. Персональні дані, унікальність кандидата, історія етапів, причина відмови та контроль комунікації. Ці властивості напряму впливають на вимоги до моделі даних і серверної логіки.

Аналіз аналогів виконано з урахуванням функцій, які є типовими для сучасних ATS-систем. Ведення вакансій, зберігання кандидатів, робота з резюме, підтримка рекрутингової воронки, командна взаємодія, інтеграції та аналітика. Такий підхід дає змогу визначити не лише загальні переваги наявних сервісів, а й вимоги до власного вебзастосунку [3-4].

1.3 Вимоги до вебзастосунку автоматизації підбору персоналу

Механізм автоматизації рейтингу уточнено як гібридний алгоритм зіставлення резюме з вакансією. Система не замінює рішення рекрутера, але автоматично нормалізує текст резюме, виділяє ключові навички, порівнює їх із вимогами вакансії та формує пояснюваний бал. Рекрутер може скоригувати результат, однак початкова оцінка не є довільною ручною позначкою.

Формула рейтингу може враховувати обов’язкові навички, бажані навички, роки досвіду, збіг ключових слів і штраф за відсутність критичних вимог. Такий підхід не претендує на повноцінний AI-рекрутинг, але виводить систему за межі простого CRUD-обліку.

Вимоги до безпеки уточнено через контекст орендаря системи. Рекрутер має бачити лише вакансії, кандидатів і журнали тієї організації, до якої належить його обліковий запис. Тому роль користувача є необхідною, але недостатньою умовою доступу, її потрібно доповнювати перевіркою tenantId, ownerId та прав на конкретний об’єкт.

В-1. Система повинна дозволяти рекрутеру створювати вакансію із назвою, описом, обов’язковими та бажаними навичками, рівнем досвіду, форматом роботи й статусом.

В-2. Система повинна дозволяти додавати кандидата вручну або через завантаження резюме у форматі PDF/DOCX із подальшим уточненням полів.

В-3. Система повинна перевіряти дублікати кандидатів за електронною поштою, телефоном або посиланням на профіль.

В-4. Система повинна формувати рейтинг відповідності кандидата до вакансії на основі збігу обов’язкових навичок, бажаних навичок, досвіду та ключових слів резюме.

В-5. Рекрутер повинен бачити пояснення рейтингу: які критерії збіглися, які відсутні та який ваговий внесок має кожен блок.

В-6. Рейтинг не повинен автоматично відхиляти кандидата без рішення користувача.

В-7. Система повинна підтримувати етапи воронки: новий кандидат, скринінг, технічна оцінка, співбесіда, офер, відмова та резерв.

В-8. Кожна зміна етапу повинна зберігати автора, час і коментар.

В-9. Керівник підрозділу може переглядати кандидатів, переданих на оцінювання, але не змінює службові налаштування вакансій.

Нефункціональні вимоги включають захист персональних даних, розмежування доступу, журналювання критичних операцій, зручну роботу з

фільтрами та передбачувану обробку помилок. Інтерфейс має бути адаптивним, підтримувати клавіатурну навігацію, показувати порожні стани й не втрачати введені дані після помилки.

Для якості результатів важливо нормалізувати назви навичок. Наприклад, JavaScript, JS і ECMAScript повинні зіставлятися через довідник синонімів, а не оцінюватися як різні компетентності. Це зменшує шум у рейтингу та робить пояснення відповідності більш корисним для рекрутера.

Вимоги до системи сформовано з урахуванням того, що вебзастосунок має підтримувати не лише збереження рекрутингових даних, а й попереднє зіставлення кандидата з вимогами вакансії. Рейтинг відповідності формується на основі нормалізованого тексту резюме, переліку навичок, досвіду та ключових ознак, а остаточне рішення залишається за рекрутером [5-7].

1.4 Технічне завдання

Задача розроблення полягає у створенні вебзастосунку, який забезпечує цілісний цикл роботи з вакансіями та кандидатами. Система повинна підтримувати створення вакансії, додавання кандидата, аналіз резюме, розрахунок рейтингу відповідності, зміну етапу воронки та збереження історії дій користувачів.

Постановка задачі передбачає створення узгодженого прототипу ATS-рівня: вакансія, кандидат, резюме, оцінювання, етап воронки й журнал аудиту мають працювати як пов’язана система. Такий підхід дозволяє аргументовано захистити архітектурні рішення.

Задача розроблення полягає у створенні вебзастосунку, який підтримує керування вакансіями, профілями кандидатів, резюме, етапами відбору та рейтингом відповідності. Основний сценарій починається зі створення вакансії, продовжується додаванням кандидата й завершується переглядом поясненого рейтингу та зміною етапу воронки.

Модель рейтингу задається як зважена сума нормалізованих критеріїв. Обов’язкові навички, бажані навички, релевантний досвід, ключові слова резюме та негативні обмеження. Якщо кандидат не має обов’язкової навички, система зменшує підсумковий бал і показує причину.

Для реалізації потрібно спроєктувати сутності Vacancy, Candidate, Resume, Skill, PipelineStage, Evaluation та AuditLog. Вони мають зберігати не лише поточний стан, а й історію змін, оскільки рекрутингові рішення потребують пояснюваності. Окремо визначаються ролі користувачів і правила доступу до персональних даних.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ АВТОМАТИЗАЦІЇ РЕКРУТИНГУ

2.1 Архітектура сервісу вакансій і кандидатів

Архітектуру побудовано як клієнт-серверну систему з відокремленими рівнями Web UI, REST API, сервісної логіки та сховища даних. Клієнт відповідає за введення й візуалізацію, але не приймає остаточних рішень щодо прав доступу, допустимих етапів або рейтингу. Сервер централізовано перевіряє запити й застосовує предметні правила.

API поділено на маршрути вакансій, кандидатів, резюме, етапів воронки та звітів. Контролери приймають запит і передають перевірені параметри до сервісів, а сервіси координують операції зі сховищем, журналом і модулем оцінювання. Репозиторії приховують SQL-запити та дозволяють тестувати бізнес-логіку окремо [8].

Операції створення кандидата й зміни етапу виконуються транзакційно. Якщо під час додавання кандидата виявлено дубль або не вдалося зберегти резюме, система не повинна створити напівпорожній профіль. Якщо етап змінено, одночасно фіксується запис AuditLog із попереднім і новим значенням.

Модуль рейтингу відповідності працює як окремий сервіс. Він отримує вакансію, профіль кандидата й нормалізовані навички, повертає числовий бал та пояснення. Такий підхід дозволяє змінити формулу або ваги без переписування контролерів та інтерфейсу.

Захист персональних даних реалізується через автентифікацію, рольову авторизацію, обмеження доступу до резюме та журналювання переглядів. Архітектура має підтримувати локальний запуск для демонстрації й подальше розгортання через Docker-контейнери.

Рисунок 2.1 подає узагальнену архітектуру програмної системи. На ньому показано, як користувацький інтерфейс взаємодіє з API, як запити передаються до бізнес-логіки, як дані зберігаються у сховищі та які службові компоненти

підтримують надійність роботи. Діаграма не замінює детального опису класів або таблиць, але задає основу для подальшого проєктування.

Рисунок 2.1 – Архітектура вебзастосунку для керування вакансіями та кандидатами

Архітектура вебзастосунку побудована за клієнт-серверним принципом. Клієнтська частина відповідає за інтерфейс рекрутера, серверна частина реалізує бізнес-логіку та API, а база даних зберігає вакансії, кандидатів, резюме, результати оцінювання й журнал подій.

У ресурсній моделі створення кандидата для конкретної вакансії розглядається як створення application-запису, а не як запуск абстрактного сценарію. Це дає змогу природно пов’язати HTTP-запит із сутностями бази даних, повернути код 201 Created, забезпечити журналювання targetId і повторно використати той самий маршрут у клієнтському інтерфейсі та інтеграційних тестах.

REST API у проєкті трактується як набір ресурсів, а не як набір довільних команд. Тому основні маршрути мають вигляд /api/vacancies, /api/candidates, /api/vacancies/:vacancyId/applications, /api/candidates/:candidateId/evaluations і

/api/candidates/:candidateId/stage. Така структура краще відповідає ресурсній моделі, ніж узагальнений виклик на кшталт узагальненого RPC-виклику.

Архітектурно система поєднує RBAC і ABAC: RBAC визначає базову роль користувача, а ABAC перевіряє атрибути запиту, організацію, власника й цільовий ресурс. Це для хмарного середовища, де кілька компаній можуть працювати в одній інсталяції, але не повинні бачити дані одна одної [9-11].

2.2 Сценарії рекрутера та модель даних відбору

Сценарії роботи рекрутера визначають логіку моделі даних. Основними об’єктами системи є Vacancy, Candidate, Resume, Evaluation, PipelineStage та AuditLog. Їхні зв’язки забезпечують збереження профілю кандидата, історії оцінювання, поточного етапу відбору та службової інформації про виконані дії.

Для AuditLog також доцільно передбачити індекси за tenantId, resourceType, targetId і timestamp. Така структура потрібна не тільки для перегляду історії в картці кандидата, а й для службової перевірки: хто змінив етап, з якого значення на яке, у межах якої вакансії та з якої організації було виконано дію.

Модель AuditLog доповнено логікою поліморфного посилання на об’єкт дії. Запис аудиту повинен містити actorId, tenantId, resourceType, targetId, action, beforeValue, afterValue і timestamp. Без resourceType та targetId неможливо коректно відновити історію конкретного кандидата або вакансії.

Для рекрутингової воронки доцільно зберігати не лише поточний етап кандидата, а й допустимі переходи між етапами. Це перетворює сценарій добору на керовану state machine: система дозволяє лише ті переходи, які описані в конфігурації вакансії або організації.

Основний сценарій рекрутера включає створення вакансії, додавання кандидата, завантаження резюме, автоматичне зіставлення навичок, перегляд рейтингу й переведення кандидата на наступний етап. Кожна дія має відображатися у стані кандидата та бути придатною для подальшого аудиту.

Use Case-діаграма відображає три групи дій: роботу рекрутера з вакансіями та кандидатами, оцінювання керівником підрозділу й адміністрування довідників. Важливо, що керівник не керує всією воронкою, а лише переглядає переданих кандидатів і додає оцінку. Це запобігає змішуванню ролей.

Модель даних відокремлює Candidate від Resume, оскільки один кандидат може мати кілька версій резюме. Vacancy містить набір критеріїв, а Evaluation зберігає результат зіставлення на конкретний момент часу. PipelineStage фіксує поточний стан, а AuditLog містить історію зміни етапів, коментарів і оцінок.

Діаграма послідовності описує виклик POST-запиту створення кандидата: Web UI надсилає форму, REST API виконує валідацію, сервіс перевіряє дублікати, модуль оцінювання розраховує початковий рейтинг, репозиторій зберігає дані, після чого клієнт отримує профіль із результатом.

Така модель підтримує повторне оцінювання. Якщо вакансія змінює вимоги або кандидат оновлює резюме, система може створити новий Evaluation без втрати попереднього результату. Це важливо для прозорості, оскільки рекрутер бачить не лише поточний бал, а й причину його зміни.

Рисунок 2.2 – Варіанти використання для рекрутера, керівника та адміністратора

Рисунок 2.2 відображає узагальнену модель варіантів використання. Він показує, які актори беруть участь у роботі системи, які дії належать до основного

сценарію, а які пов’язані з адмініструванням або переглядом результатів. Така діаграма корисна на етапі погодження вимог, оскільки дозволяє оперативно побачити, чи не пропущено важливу роль або дію.

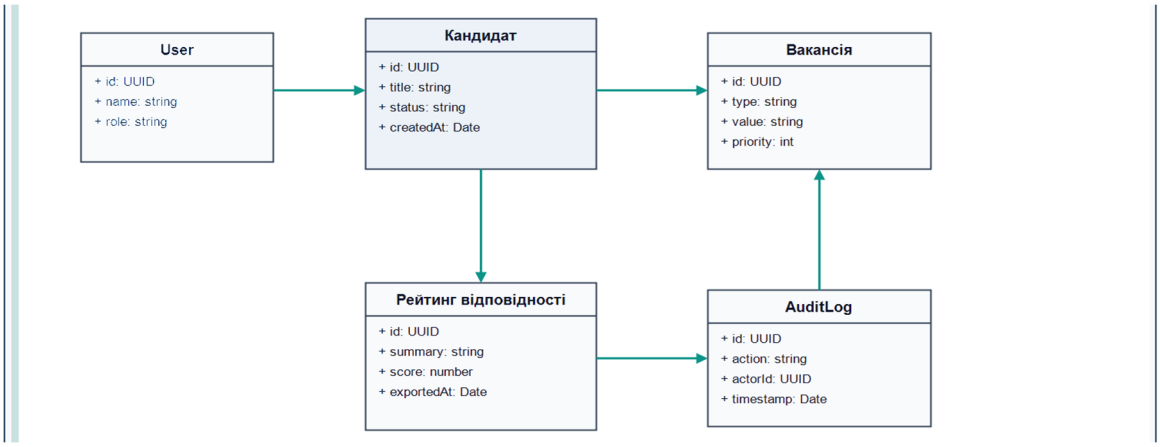

Рисунок 2.3 – Модель даних вакансій, кандидатів, резюме та етапів відбору

Рисунок 2.3 подає концептуальну модель даних. Він не є остаточною фізичною схемою бази даних, але демонструє логічний склад предметної області. На наступних етапах цю модель обґрунтовано деталізувати у вигляді таблиць, колекцій або класів. Важливо, щоб кожна сутність мала зрозуміле призначення і не дублювала дані іншої сутності.

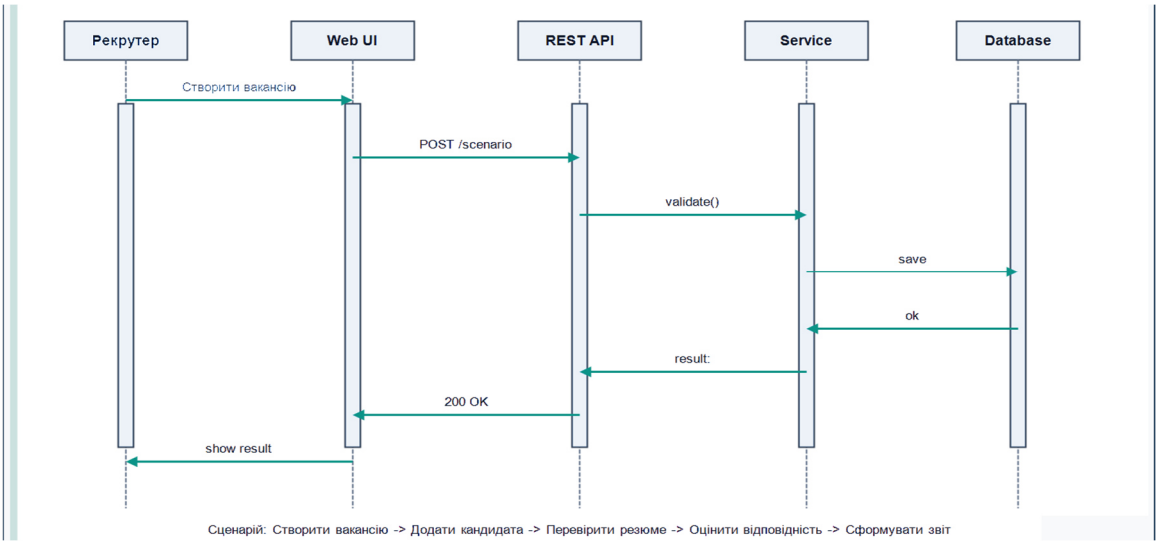

Рисунок 2.4 – REST-послідовність додавання кандидата до вакансії

Рисунок 2.4 демонструє послідовність додавання кандидата до вакансії через ресурсний REST-маршрут /api/vacancies/:vacancyId/applications. Такий варіант не є RPC-викликом і узгоджується з архітектурним описом REST API.

2.3 Інтерфейс рекрутингової воронки та стек технологій

Інтерфейс рекрутингового сервісу проєктується навколо основних дій користувача: перегляду вакансій, роботи з картками кандидатів, аналізу рейтингу відповідності та керування етапами воронки. Важливо, щоб користувач бачив не лише підсумковий бал кандидата, а й пояснення чинників, які вплинули на результат.

Інтерфейс організовано навколо рекрутингової воронки. На головній панелі рекрутер бачить вакансії, кількість кандидатів на етапах, прострочені комунікації та швидкі фільтри. Картка кандидата містить контактні дані, резюме, навички, рейтинг відповідності, історію етапів і коментарі.

Рисунок 2.5 – Макет інтерфейсу вакансій, кандидатів і співбесід.

Рисунок 2.5 подає макет ключових екранів рекрутингового сервісу: воронку кандидатів, картку вакансії, профіль кандидата і календар співбесід.

React обрано для клієнтської частини через компонентний підхід, зручну побудову SPA та можливість повторного використання елементів картки, фільтра й модального вікна. TypeScript формалізує структуру кандидата, вакансії й помилок API, що зменшує ризик неузгоджених полів між клієнтом і сервером.

Node.js та Express доцільні для швидкої побудови REST API, асинхронної роботи з файлами резюме й інтеграції з модулем парсингу. FastAPI може бути альтернативою для Python-орієнтованої обробки тексту, але для єдиного TypeScript-стеку Express спрощує спільні типи та локальну розробку.

PostgreSQL обрано як основне сховище, оскільки рекрутингова модель має чіткі зв’язки між вакансіями, кандидатами, оцінками та історією етапів. Для пошуку за навичками можна застосувати індекси й додаткові таблиці нормалізованих компетентностей. MongoDB підходив би для гнучких резюме, але складні зв’язки та транзакційність важливіші для цього прототипу [12-14].

Критерії якості включають швидке відкриття списку кандидатів, зрозумілі фільтри, пояснюваний рейтинг, контроль дублювання, доступність інтерфейсу та захист персональних даних. Візуальні матеріали оформлено у формальному стилі з нейтральною палітрою, щоб схема підтримувала зміст, а не відволікала декоративними акцентами.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ МОДУЛІВ РЕКРУТИНГОВОГО СЕРВІСУ

3.1 Середовище розробки та структура модулів сервісу

Реалізація програмної системи передбачає поділ проєкту на клієнтський, серверний і спільний модулі. Така структура спрощує супровід коду, оскільки компоненти інтерфейсу, маршрути API, сервіси бізнес-логіки, репозиторії та типи даних мають окремі зони відповідальності.

Структуру проєкту поділено на клієнтський, серверний і спільний модулі. Клієнт містить компоненти воронки, картки кандидата, форми вакансії та звіту. Сервер містить маршрути, сервіси, репозиторії, модуль рейтингу, middleware авторизації та централізовану обробку помилок. Спільний модуль містить типи DTO та перелік допустимих статусів.

Конфігурацію винесено у змінні середовища. Адреса бази, параметри JWT, шлях до сховища резюме та режим запуску. Міграції створюють таблиці вакансій, кандидатів, резюме, навичок, оцінок та журналу. Демонстраційні дані відокремлено від тестових, щоб автоматичні перевірки не залежали від презентаційного набору.

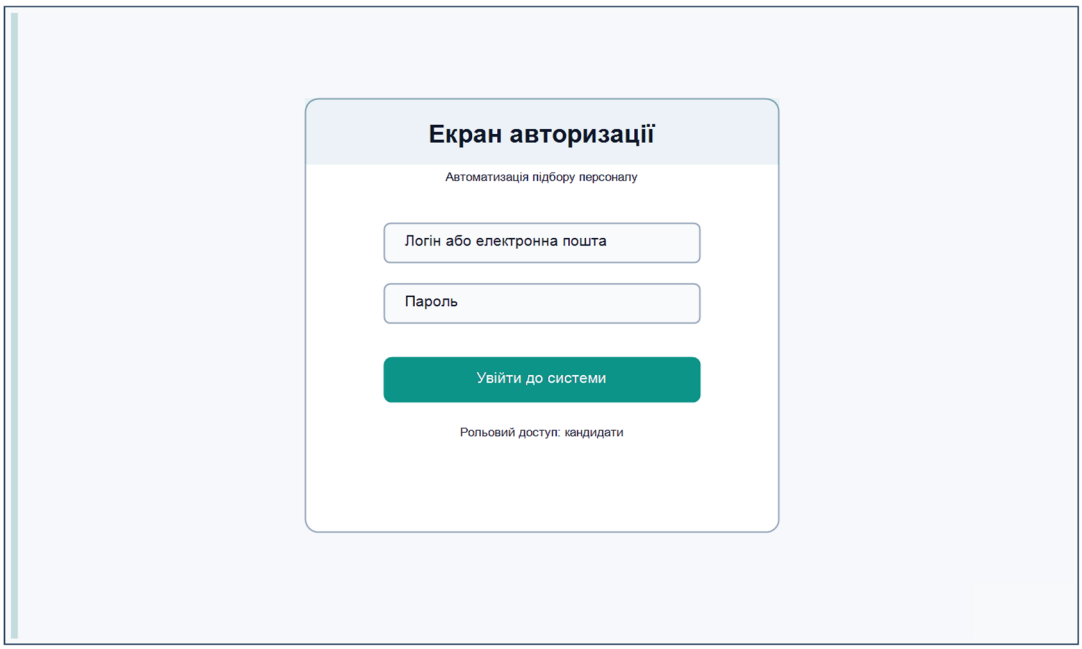

Рисунок 3.1 – Екран входу рекрутера

Рисунок 3.1 відображає початковий екран авторизації користувача рекрутингового сервісу. Його призначення полягає у перевірці облікових даних рекрутера та наданні доступу до функцій системи відповідно до ролі користувача. Наявність окремого екрана входу є важливою для захисту персональних даних кандидатів і розмежування доступу між учасниками рекрутингового процесу.

Лістинг 3.1 – Опис базової моделі даних для об’єкта «кандидат»

```
export type CandidateStatus = 'new' | 'screening' | 'interview' |
'offer' | 'rejected' | 'reserve';

export interface CandidateRecord {
  id: string;
  fullName: string;
  status: CandidateStatus;
  ownerId: string;
  vacancyId?: string;
  matchingScore: number;
  createdAt: string;
  updatedAt: string;
  metadata: {
    domainObject: 'candidate';
    mainModule: 'vacancies';
    resultType: 'matchingScore';
  };
}

export const emptyCandidate = (ownerId: string): CandidateRecord
=> {
  const now = new Date().toISOString();
  return {
    id: crypto.randomUUID(),
    fullName: '',
    status: 'new',
    ownerId,
    matchingScore: 0,
    createdAt: now,
    updatedAt: now,
    metadata: { domainObject: 'candidate', mainModule:
'vacancies', resultType: 'matchingScore' }
  };
};
```

кінець лістингу 3.1

Лістинг 3.1 подає базову TypeScript-модель кандидата, яка визначає основні поля запису: ідентифікатор, ім’я, статус, власника, пов’язану вакансію, рейтинг відповідності та часові мітки. Наявність такого інтерфейсу формалізує структуру даних, уніфікує обмін між клієнтською та серверною частинами й зменшує ризик неузгодженості під час розробки. У ній застосовано англомовні enum-подібні значення та окреме поле `matchingScore`, що відповідає за результат первинного зіставлення з вакансією.

Рисунок 3.2 демонструє головну панель вебзастосунку, на якій рекрутер отримує узагальнену інформацію про відкриті вакансії, кількість кандидатів і поточний стан відбору. Такий екран виконує роль робочого центру системи, оскільки з нього користувач переходить до карток кандидатів, фільтрів, аналітики та керування етапами рекрутингової воронки.

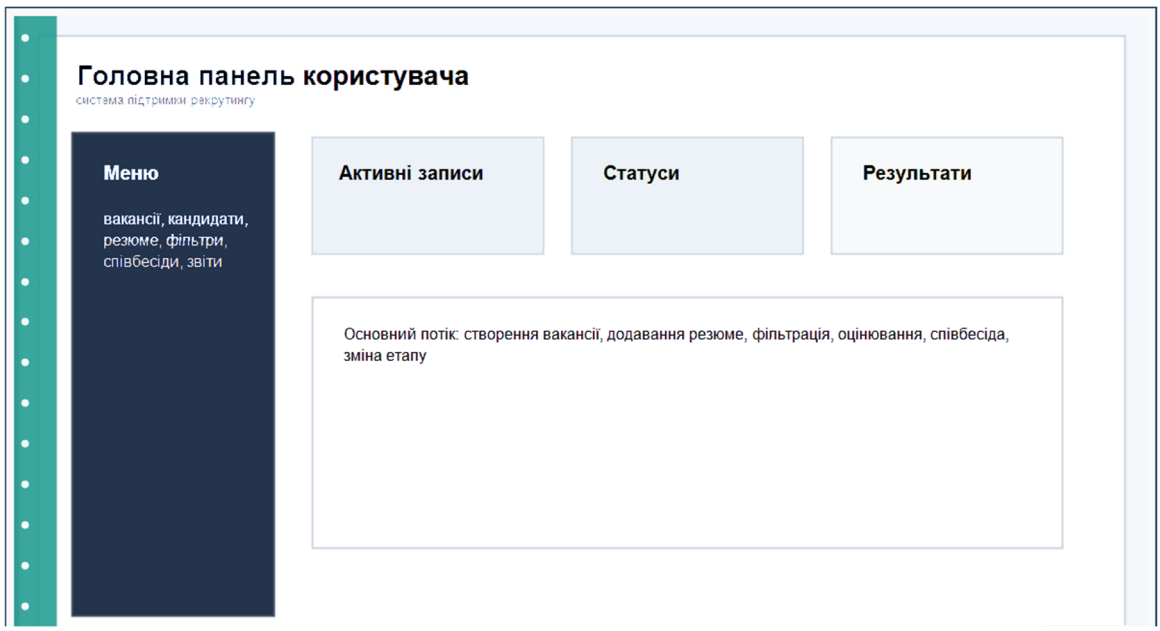

Рисунок 3.2 – Головна панель вакансій і кандидатів

Для контролю якості під час розробки використовуються перевірка типів, форматування, модульні тести сервісів і інтеграційні перевірки API. Журналювання не повинно містити повний текст резюме або конфіденційні контактні дані, але фіксує ідентифікатор кандидата, дію, автора й час.

3.2 Серверна логіка вакансій, кандидатів і етапів відбору

Серверна логіка забезпечує створення вакансій, додавання кандидатів, перевірку вхідних даних, розрахунок рейтингу відповідності та зміну етапів рекрутингової воронки. Окремі сервіси відповідають за політику доступу, роботу з резюме, оцінювання кандидата та запис подій до журналу аудиту.

Логіка переходів у воронці реалізується через конфігурацію станів, а не через абстрактний execute-метод із текстовим маркером. Це дозволяє прямо в коді побачити, з якого стану в який кандидат може перейти, які умови перевіряються і який запис AuditLog створюється після успішної операції.

Транзакційність у серверній логіці потрібна для уникнення розходження між поточним етапом кандидата і журналом подій. Якщо оновлення етапу відбулося, але запис AuditLog не створено, інтерфейс показуватиме новий стан без історії. Якщо ж транзакція відкочується цілком, система зберігає узгодженість навіть у разі помилки сервісу оцінювання.

Серверна частина не повинна покладатися лише на authorize(recruiter). Після перевірки ролі кожна операція додатково звіряє tenantId, доступність вакансії, право користувача працювати з кандидатом і допустимість переходу етапу. Це закриває ризик, коли рекрутер однієї організації випадково або навмисно звертається до запису іншої організації.

Лістинг 3.4 демонструє реалізацію переходів між етапами рекрутингової воронки на основі state machine. Код перевіряє допустимість переходу, враховує наявність розрахованого matchingScore, виконує зміну стану в транзакції та записує AuditLog із посиланням на конкретний ресурс.

Створення вакансії виконується через маршрут `/api/vacancies`, який приймає перевірені дані й додає власника з авторизованого користувача. Сервер не довіряє службовим полям із клієнта, тому роль, власник і початковий статус визначаються на бекенді. Помилки валідації повертаються у вигляді словника полів.

Додавання кандидата перевіряє коректність контактів, унікальність email/телефону та наявність вакансії. Якщо резюме додано файлом, сервіс зберігає метадані документа й запускає нормалізацію навичок. Після цього модуль рейтингу повертає бал і пояснення, яке зберігається в Evaluation.

Зміна етапу воронки дозволена лише для допустимих переходів. Наприклад, кандидат не може перейти до оферу без завершеної співбесіди, якщо така вимога встановлена вакансією. Кожна зміна етапу виконується транзакційно разом із записом у AuditLog, щоб історія не відставала від поточного статусу [15].

Серверні помилки поділено на валідаційні, авторизаційні, помилки відсутнього запису та внутрішні збої. Користувач отримує зрозуміле повідомлення, а технічні деталі залишаються в журналі. Такий підхід враховує персональний характер рекрутингових даних.

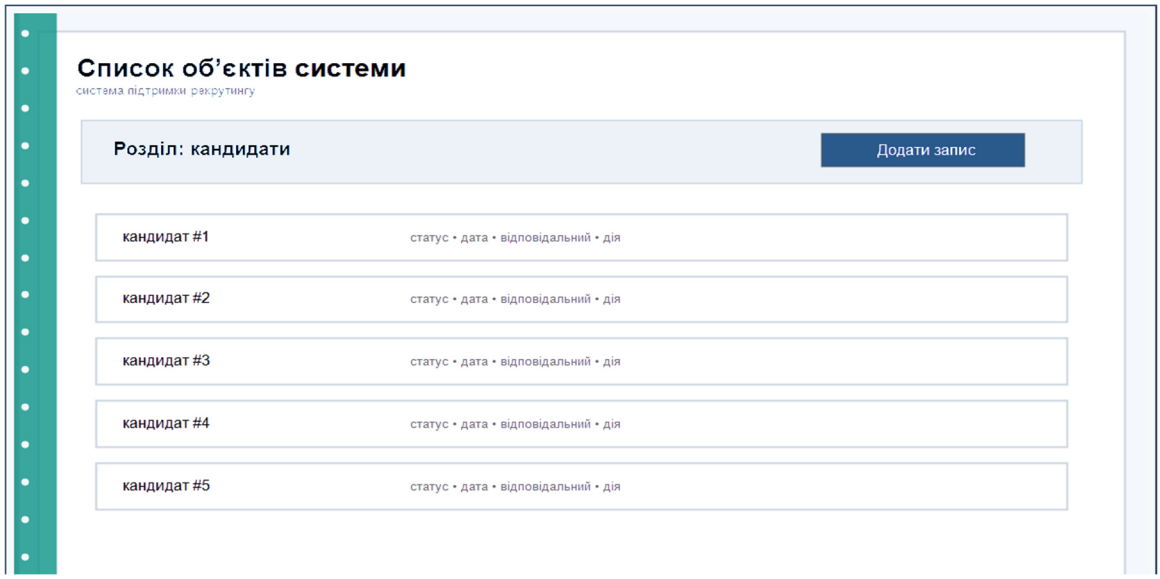

Рисунок 3.3 – Список кандидатів за вакансіями та статусами

Рисунок 3.3 ілюструє представлення списку кандидатів із можливістю групування або фільтрації за вакансією, статусом, рейтингом відповідності та етапом відбору. Такий інтерфейс дає змогу швидко знаходити потрібних кандидатів, порівнювати їх між собою та контролювати наповнення кожної вакансії без ручного перегляду розрізнених таблиць.

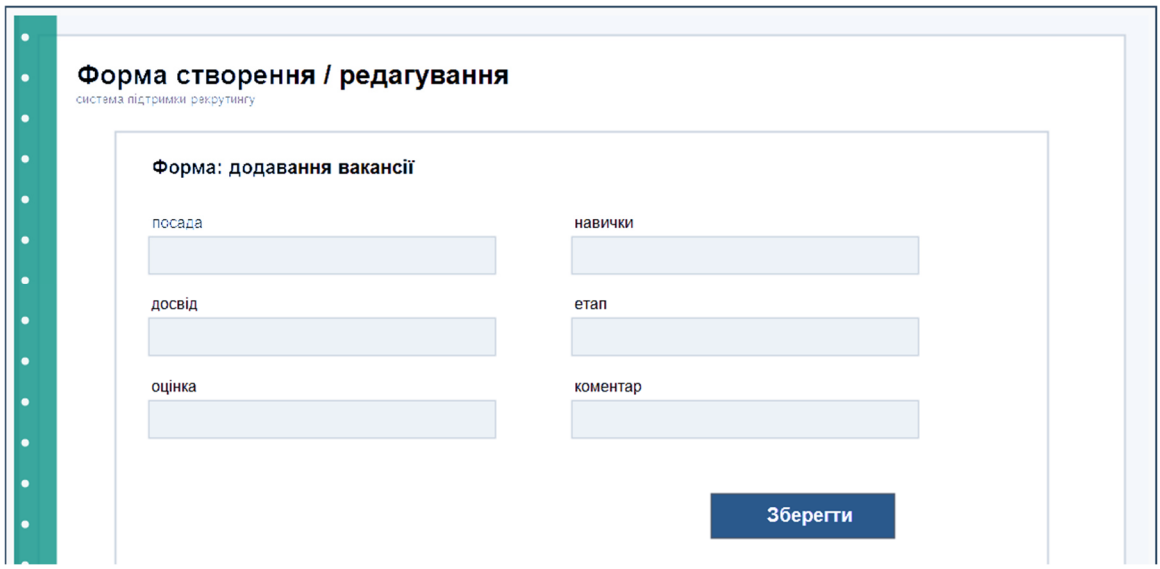

Рисунок 3.4 – Форма додавання кандидата або вакансії

Рисунок 3.4 показує форму введення даних, яка використовується для створення нового кандидата або вакансії. У ній зосереджено ключові поля, необхідні для подальшої обробки запису: контактні дані, назва вакансії, навички, опис вимог і службові параметри. Форма є важливою частиною системи, оскільки саме на цьому етапі забезпечується первинна структуризація даних.

Рисунок 3.5 – Картка кандидата з історією комунікацій

Рисунок 3.5 відображає детальну картку кандидата, у якій поєднано персональні дані, резюме, поточний етап відбору, рейтинг відповідності та історію взаємодії. Такий екран забезпечує рекрутеру повний контекст щодо кандидата й дозволяє приймати рішення не лише за числовим показником, а й за сукупністю збережених коментарів, подій та результатів оцінювання.

Лістинг 3.2 демонструє серверний маршрут REST API, який відповідає за створення заявки кандидата для конкретної вакансії. У фрагменті показано перевірку авторизації, отримання tenant-контексту, пошук вакансії, валідацію вхідних даних і створення запису через сервісний шар. Такий підхід відокремлює HTTP-контролер від бізнес-логіки та забезпечує контроль доступу до ресурсів.

Лістинг 3.2 – Метод API для створення запису в системі

```
       router.post('/api/vacancies/:vacancyId/applications', authenticate,
           authorize('recruiter'), async (req, res, next) => {  try {
    const tenantId = req.user.tenantId;    const vacancy = await
vacancyRepository.findById(req.params.vacancyId, { tenantId });
    if (!vacancy) throw new NotFoundError('Вакансію не знайдено');
    accessPolicy.assertCanCreateApplication({ actor: req.user, vacancy });
    const payload = validateCandidatePayload(req.body);
    const application = await candidateService.createApplication({
      ...payload,
      tenantId,
      vacancyId: vacancy.id,
      ownerId: req.user.id
    });
    res.status(201).json({ ok: true, data: application });
  } catch (error) {
    next(error);
  }});
```

кінець лістингу 3.2

Лістинг 3.3 описує функцію перевірки даних вакансії перед їх збереженням у системі. Валідація контролює мінімальну довжину назви, допустимість статусу, наявність обов’язкових навичок і коректність додаткових полів. Завдяки цьому некоректні дані відхиляються на серверному рівні, а користувач отримує структуровані повідомлення про помилки.

Лістинг 3.3 – Функція серверної валідації вхідних даних

```
function validateVacancyPayload(payload: unknown) {
  const input = payload as Record<string, unknown>;
  const errors: Record<string, string> = {};
  const allowedStatuses = ['draft', 'published', 'paused', 'closed'];

  if (typeof input?.title !== 'string' || input.title.trim().length < 3) {
    errors.title = 'Назва вакансії повинна містити щонайменше 3 символи';
  }
  if (typeof input?.status !== 'string' ||
!allowedStatuses.includes(input.status)) {
    errors.status = 'Указано неприпустимий статус вакансії';
  }
  if (!Array.isArray(input.requiredSkills) || input.requiredSkills.length === 0)
{
    errors.requiredSkills = 'Потрібно вказати хоча б одну обов’язкову навичку';
  }
  if (Object.keys(errors).length > 0) {
    throw new ValidationError('Помилка перевірки вакансії', errors);
  }

  return {
    title: (input.title as string).trim(),
    status: input.status as string,
    requiredSkills: input.requiredSkills,
    description: typeof input.description === 'string' ?
input.description.trim() : ''
  };
}
```

кінець лістингу 3.3

Код визначає дозволені переходи, перевіряє наявність розрахованого рейтингу, виконує оновлення стану кандидата в межах транзакції та створює запис AuditLog. Така структура робить бізнес-логіку прозорою, контрольованою та придатною для модульного тестування.

3.3 Клієнтський інтерфейс рекрутера та картки кандидата

Клієнтська частина реалізує робоче середовище рекрутера. Вона повинна забезпечувати зрозумілу навігацію між вакансіями, кандидатами, фільтрами та звітами, а також коректно відображати стани завантаження, помилки введення й результати виконання операцій.

Клієнтський інтерфейс відображає воронку як набір етапів із картками кандидатів. Картка показує ім’я, вакансію, поточний етап, рейтинг відповідності

та коротке пояснення ключових збігів. Рекрутер може відкрити детальний профіль, переглянути резюме, додати коментар або змінити етап.

Форма створення вакансії перевіряє обов'язкові поля на клієнті, але остаточну валідацію виконує сервер. Якщо сервер повертає помилку, введені значення зберігаються, а проблемні поля підсвічуються. Під час запиту кнопка збереження блокується, щоб уникнути повторного створення запису.

Фільтри дозволяють обмежити список кандидатів за вакансією, етапом, рейтингом, навичками та датою останньої взаємодії. Порожній результат не повинен виглядати як помилка: інтерфейс пояснює, що кандидатів за вибраними умовами не знайдено, і пропонує змінити фільтр.

Екран звіту показує кількість кандидатів на кожному етапі, середній рейтинг, частку відмов і прострочені комунікації. Показники мають підписи та джерело даних, щоб користувач розумів, як сформовано результат. Рейтинг не приховує фактичні навички кандидата, а лише допомагає впорядкувати список.

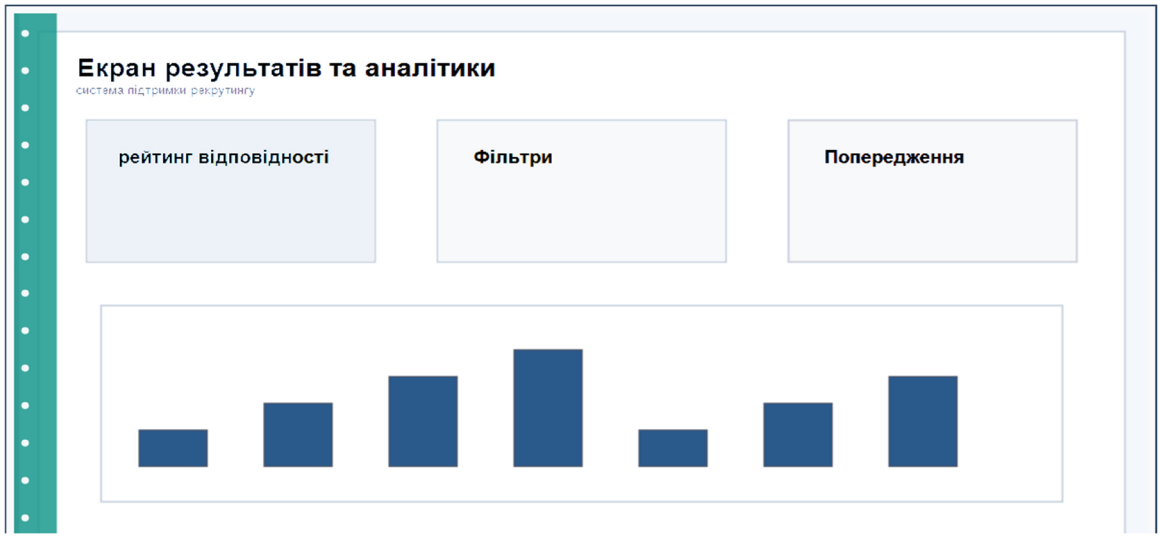

Рисунок 3.6 – Аналітика проходження кандидатів етапами відбору

Рисунок 3.6 подає аналітичне представлення руху кандидатів рекрутинговою воронкою. На ньому узагальнюються показники кількості кандидатів на етапах, частки переходів, відмов і успішних проходжень. Така

аналітика допомагає оцінити ефективність процесу підбору персоналу та виявити етапи, на яких накопичуються затримки або втрати кандидатів.

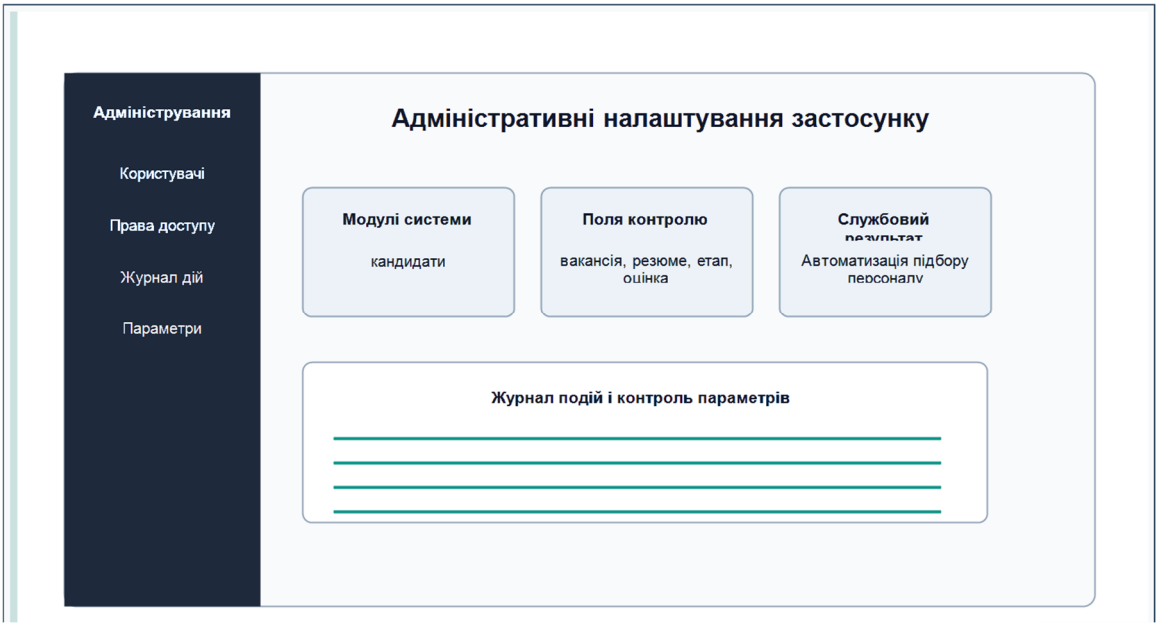

Рисунок 3.7 – Налаштування етапів, ролей і прав доступу

Рисунок 3.7 демонструє службову частину інтерфейсу, призначену для налаштування етапів воронки, ролей користувачів і правил доступу. Цей екран важливий для адаптації системи до конкретної організації, оскільки різні компанії можуть мати різну структуру відбору, набір відповідальних осіб і вимоги до захисту рекрутингових даних.

Лістинг 3.4 – Клієнтський метод збереження даних через REST API

```
export async function saveVacancy(formState: Record<string, unknown>) {
  const response = await fetch('/api/vacancies', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(formState)
  });
  const body = await response.json();
  if (!response.ok || !body.ok) {
    throw new ApiError(body.message ?? 'Не вдалося зберегти вакансію',
body.errors ?? {});
  }
  return body.data;
}
```

кінець лістингу 3.4

Лістинг 3.4 демонструє клієнтський метод, який надсилає дані форми на сервер за допомогою HTTP-запиту. У фрагменті показано формування JSON-запиту, обробку відповіді API та перехоплення помилок, якщо сервер повертає неуспішний результат. Такий метод є посередником між інтерфейсом користувача та серверною логікою застосунку.

Лістинг 3.5 – Обробник форми користувацького інтерфейсу

```
async function handleSubmit(event: React.FormEvent<HTMLFormElement>) {
  event.preventDefault();
  setLoading(true);
  setErrors({});
  try {
    const saved = await saveVacancy(formState);
    notify('Вакансію збережено');
    navigate(`/vacancies/${saved.id}`);
  } catch (error) {
    const apiError = error as ApiError;
    setErrors(apiError.details ?? {});
    notify(apiError.message ?? 'Не вдалося виконати операцію');
  } finally {
    setLoading(false);
  }
}
```

кінець лістингу 3.5

Лістинг 3.5 описує обробник події надсилання форми в клієнтському інтерфейсі. Він блокує стандартну поведінку браузера, вмикає стан завантаження, очищує попередні помилки, викликає метод збереження даних і після успішної операції перенаправляє користувача до створеного запису. Така логіка забезпечує передбачувану поведінку форми та зрозумілий зворотний зв’язок для рекрутера.

Лістинги 3.4 і 3.5 узгоджено з маршрутом `/api/vacancies` та форматом помилок API. Обробник форми використовує типізовану подію, контроль завантаження, повідомлення про успіх і гарантоване скидання стану у блоці `finally`.

Демонстраційні дані призначені для перевірки основних сценаріїв роботи системи. Вони охоплюють вакансії з різними вимогами, кандидатів із різним

рівнем відповідності, приклади дублювання контактів і ситуації, у яких потрібно перевірити зміну етапу воронки та запис події до журналу.

Демонстраційний набір містить дві вакансії: Frontend Developer і QA Engineer. Для кожної вакансії задано обов’язкові навички, бажані навички, рівень досвіду та етапи воронки. До набору додано кандидатів із різним ступенем відповідності, щоб показати роботу рейтингу та фільтрів.

Приклад кандидата для демонстрації має поля `candidateId: c-001`, `fullName: Ілля Бондарчук`, `email: illia.bondarchuk@example.com`, `skills: [React, TypeScript, HTML, CSS]`, `experienceYears: 2`, `vacancyId: v-frontend`, `stage: screening`, `matchingScore: 82`. Такий запис дозволяє перевірити профіль, рейтинг, фільтр за навичками й зміну етапу.

Окремо підготовлено негативні демонстраційні випадки: кандидат без email, дубльований номер телефону, резюме без потрібних навичок і спроба змінити чужу вакансію. Вони потрібні для перевірки повідомлень і стійкості системи.

Контрольний запуск виконується у послідовності: вхід рекрутера, створення вакансії, додавання кандидата, перегляд рейтингу, зміна етапу, формування звіту. Після кожного кроку перевіряється видимий результат і запис у сховищі або журналі.

РОЗДІЛ 4
ТЕСТУВАННЯ І РОЗГОРТАННЯ ПРОЕКТУ

4.1 Написання тестів

План тестування сформовано відповідно до функціональних і нефункціональних вимог системи. Перевірка охоплює авторизацію, створення вакансії, додавання кандидата, розрахунок рейтингу, зміну етапів воронки, роботу API, обробку помилок і запуск застосунку в контейнеризованому середовищі.

План тестування (рис. 4.1) побудовано за вимогами до вакансій, кандидатів, рейтингу, етапів воронки, звітів і розгортання. Для кожної групи визначено позитивні, негативні та граничні випадки. Така структура усуває проблему шаблонних тестів, у яких очікуваний результат не відповідає самій дії.

Рисунок 4.1 – Схема тестування рекрутингового процесу

Модульні тести перевіряють валідацію вакансії, нормалізацію навичок, формулу рейтингу та допустимі переходи етапів. Інтеграційні тести перевіряють маршрути API, транзакції, унікальність кандидата й запис AuditLog. Наскрізний тест відтворює шлях рекрутера від створення вакансії до перегляду звіту.

Для ТС-01 очікуваним результатом є успішна авторизація та перенаправлення на головну панель рекрутера, а не загальний опис стабільності системи. Для ТС-02 очікуваним результатом є створена вакансія або кандидат із відображенням у списку та записом у базі. Саме такі конкретні результати внесено до таблиці 4.1.

Таблиця 4.1 – Контрольні тестові випадки

ID	Перевірка	Очікуваний результат
ТС-01	Вхід рекрутера	Успішна авторизація, перенаправлення на головну панель
ТС-02	Створення вакансії	Вакансію збережено та показано у списку
ТС-03	Некоректний email кандидата	Показано повідомлення біля поля email
ТС-04	Дубль кандидата	Створення заблоковано, показано існуючий профіль
ТС-05	Зміна етапу воронки	Етап оновлено, запис AuditLog створено

Граничні перевірки включають мінімальну довжину назви вакансії, резюме без навичок, нульовий рейтинг, кандидата з повним збігом і повторне надсилання форми. Негативні сценарії перевіряють неправильний формат email, дубль кандидата та заборонений доступ.

Результати тестів фіксуються з початковими умовами, дією, очікуваною поведінкою та фактичним результатом. Для дефектів зазначається короткий спосіб відтворення, щоб після виправлення можна було повторити той самий сценарій.

4.2 Результати тестування функцій підбору персоналу

Результати тестування подано за групами вимог: функціональність, API, інтерфейс, безпека та розгортання. Для кожної групи визначено набір контрольних перевірок, а відсоткові показники на рисунку 4.2 відображають частку успішно виконаних тестів у відповідній групі.

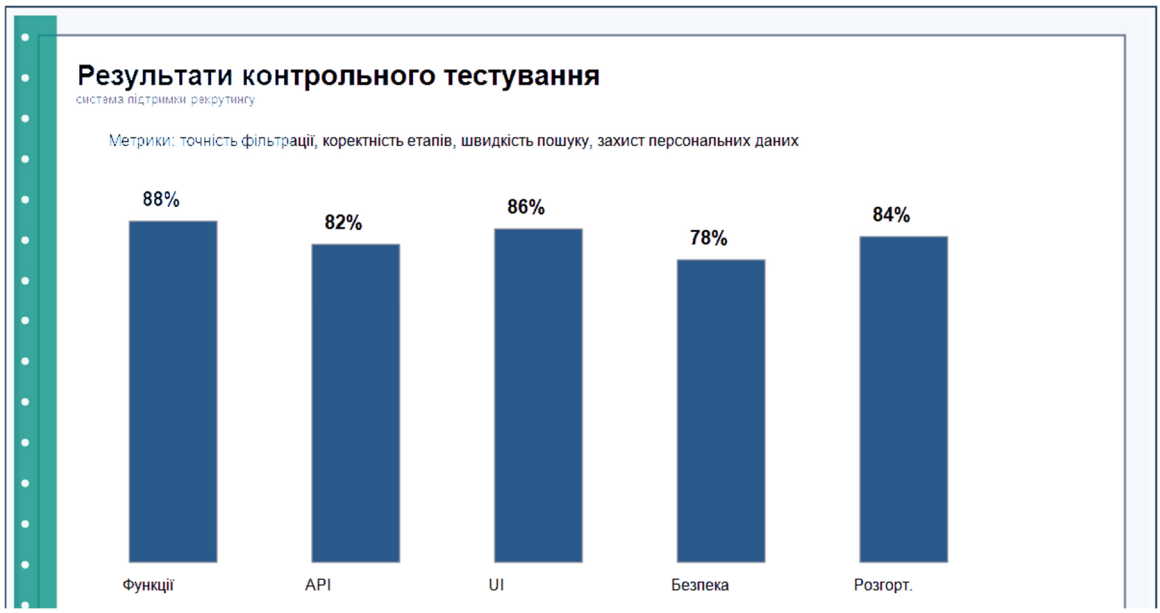

Рисунок 4.2 – Результати контрольного тестування підбору персоналу

Показник «Ефективність API – 82 %» у такому трактуванні є не швидкістю сервера, а агрегованим коефіцієнтом проходження вимог групи API: функціональні перевірки, REST-структура маршруту, обробка помилок, перевірка tenantId, аудит і базові latency-пороги. Показник «Надійність розгортання – 84 %» формується за чек-листом запуску контейнерів, health-check, міграцій, повторного старту та збереження даних у PostgreSQL.

Кожний відсотковий показник є агрегованим коефіцієнтом за групою вимог. Наприклад, група API охоплює створення вакансії, додавання кандидата, перевірку tenantId, обробку невалідного DTO, помилку відсутнього ресурсу та запис аудиту. Якщо частина сценаріїв не проходить перевірку, коефіцієнт зменшується пропорційно кількості непройдених тестів.

Відсоткові показники на рисунку 4.2 трактуються як коефіцієнт покриття вимог успішними тестами, а не як суб’єктивна оцінка якості. Формула має вигляд: Coverage = PassedRequirementTests / PlannedRequirementTests * 100 %. Тому показник 88 % для функцій означає 22 успішні перевірки з 25 запланованих, 82 % для API – 18 із 22, 86 % для UI – 19 із 22, 78 % для безпеки 14 із 18, 84 % для розгортання – 16 із 19.

Показник безпеки означає частку успішно пройдених сценаріїв перевірки доступу, ізоляції tenantId, заборони чужих ресурсів, обробки некоректного токена й відсутності конфіденційних даних у журналі.

Таблиця 4.2 показує критерії готовності прототипу до демонстрації. Виявлені зауваження не блокують основний сценарій і можуть бути використані для подальшого розвитку.

Таблиця 4.2 – Критерії готовності до демонстрації

Критерій	Стан	Коментар
Основний сценарій	Виконано	Вакансія, кандидат, рейтинг і звіт працюють узгоджено
Рейтинг відповідності	Перевірено	Бал супроводжується поясненням критеріїв
Помилки введення	Перевірено	Система показує проблемні поля без втрати даних
Контроль доступу	Перевірено	Ролі рекрутера, керівника й адміністратора розмежовано
Розгортання	Підготовлено	Docker-контейнери запускаються, API й база доступні

Контрольні перевірки підтвердили, що рекрутер може створити вакансію, додати кандидата, отримати рейтинг відповідності й змінити етап воронки. Після оновлення сторінки профіль кандидата та історія етапів відтворюються зі сховища. Повторний запит не створює дубльованого кандидата.

Перевірка рейтингу показала, що система розділяє збіг обов’язкових і бажаних навичок. Кандидат із відсутньою критичною навичкою отримує нижчий бал, а пояснення показує, який критерій не виконано. Завдяки цьому результат оцінювання є прозорим для рекрутера.

Розгортання вебзастосунку робимо через контейнеризоване середовище, яке містить клієнтську частину, серверний API та базу даних PostgreSQL. Такий підхід забезпечує відтворюваність запуску, спрощує перенесення системи між середовищами та дає змогу перевірити роботу основних модулів після старту контейнерів.

Розгортання доповнено контрольним чек-листом: міграції застосовані, змінні середовища задані, API повертає /health, frontend відкриває головну

сторінку, а тестова вакансія з кандидатом зберігається після перезапуску контейнерів.

Розгортання через Docker забезпечує відтворюване середовище для клієнта, API та PostgreSQL. Мінімальна структура містить `frontend/Dockerfile`, `backend/Dockerfile`, `docker-compose.yml` і файл `.env.example` із назвами потрібних змінних без секретних значень.

Frontend Dockerfile може використовувати Node.js для збирання React-застосунку та nginx для віддавання статичних файлів: `FROM node:20-alpine AS build`, `RUN npm ci && npm run build`, `FROM nginx:alpine`, `COPY --from=build /app/dist /usr/share/nginx/html`. Така схема відокремлює етап збирання від виконання [15].

Backend Dockerfile запускає Express API: `FROM node:20-alpine`, `WORKDIR /app`, `COPY package*.json ./`, `RUN npm ci --omit=dev`, `COPY dist ./dist`, `CMD ["node", "dist/server.js"]`. Перед запуском контейнера застосовуються міграції бази або окрема службова команда, щоб схема відповідала версії застосунку.

`docker-compose.yml` об’єднує сервіси `frontend`, `backend` і `postgres`. Backend отримує `DATABASE_URL`, `JWT_SECRET`, шлях до сховища резюме та порт API. Після старту перевіряються маршрут `/health`, доступність головної сторінки та створення тестової вакансії.

Контрольний запуск після розгортання включає авторизацію, створення вакансії, додавання демонстраційного кандидата й перегляд звіту. Успішним результатом є доступність контейнерів, коректне підключення до бази, відсутність критичних помилок у логах і збереження даних після перезапуску.

ВИСНОВКИ

У кваліфікаційній роботі було вирішено прикладну задачу, пов’язану з розробленням вебзастосунку для автоматизації підбору персоналу. У процесі виконання роботи проаналізовано предметну область, визначено основну проблему, яка полягає у тому, що великий обсяг резюме, складність зіставлення компетентностей із вакансіями та суб’єктивність первинного відбору. Проведений аналіз підтвердив актуальність теми, оскільки обрана система спрямована на автоматизацію реальних користувацьких процесів, зменшення кількості ручних операцій, упорядкування даних і підвищення ергономічності прийняття рішень.

Досліджено сучасний стан предметної області, розглянуто типові підходи до вирішення подібних задач, визначено цільових користувачів, інформаційні сутності, ризики та обмеження. На основі цього сформовано вимоги до системи підтримки рекрутингу, які охоплюють функціональні можливості, якість інтерфейсу, безпеку, роботу з даними та готовність системи до подальшого розвитку.

Виконано проєктування програмної системи. Було визначено архітектуру застосунку, логіку взаємодії клієнтської і серверної частин, структуру основних модулів, модель даних, користувацькі сценарії та підходи до побудови інтерфейсу. Запропоновані проєктні рішення дали змогу узгодити функціональність із предметною областю та підготувати основу для практичної реалізації.

Реалізовано основну функціональність застосунку, що включає створення вакансій, облік кандидатів, завантаження резюме, фільтрацію, рейтинг відповідності, етапи співбесід і звітність. Розроблена структура програмного проєкту забезпечує поділ відповідальності між модулями, підтримує роботу з даними, обробку помилок, перевірку введення та виконання основного сценарію користувача. Окрему увагу приділено інтерфейсу, який має забезпечувати послідовну роботу користувача від входу в систему до отримання результату.

Проведено тестування та підготовку проєкту до розгортання. Перевірено позитивні, негативні та граничні сценарії, роботу основних модулів, реакцію системи на некоректні дані, доступність інтерфейсу, взаємодію з API та збереження результатів. Тестування підтвердило, що система здатна виконувати основний сценарій добору, скорочувати час первинного опрацювання кандидатів і забезпечувати процес оцінювання більш структурованим.

Практичне значення отриманого результату полягає в тому, що розроблена система дозволяє скоротити час первинного добору кандидатів і забезпечити процес оцінювання більш структурованим. Запропоноване рішення може бути використане як навчальний прототип, основа для подальшого вдосконалення або база для створення більш повної прикладної системи. Подальший розвиток обґрунтовано спрямувати на розширення функціональності, покращення аналітики, інтеграцію із зовнішніми сервісами, підвищення рівня безпеки та адаптацію системи до реальних умов експлуатації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Greenhouse. Applicant tracking software and hiring platform. URL: https://www.greenhouse.com/ (дата звернення: 17.02.2026).

2. Greenhouse. Harvest API documentation. URL: https://developers.greenhouse.io/harvest.html (дата звернення: 17.02.2026).

3. Workable. Applicant Tracking Software. URL: https://get.workable.com/ (дата звернення: 17.02.2026).

4. Workable. Applicant tracking system guide: from A to Z. URL: https://resources.workable.com/tutorial/applicant-tracking-systems-atoz (дата звернення: 17.02.2026).

5. HURMA. Recruiting system or ATS – Recruitment Automation. URL: https://hurma.work/en/capabilities/recruiting/ (дата звернення: 17.02.2026).

6. LinkedIn Talent Solutions. Hire the people you need. URL: https://business.linkedin.com/hire (дата звернення: 17.02.2026).

7. Indeed Partner Docs. ATS integration with Indeed Apply. URL: https://docs.indeed.com/indeed-apply/ats (дата звернення: 07.05.2026).

8. OpenAPI Initiative. OpenAPI Specification. URL: https://spec.openapis.org/oas/ (дата звернення: 07.05.2026).

9. MDN Web Docs. HTTP: Hypertext Transfer Protocol. URL: https://developer.mozilla.org/en-US/docs/HTTP (дата звернення: 07.05.2026).

10. TypeScript. The TypeScript Handbook. URL: https://www.typescriptlang.org/docs/handbook/intro.html (дата звернення: 07.05.2026).

11. React. React Reference Overview. URL: https://react.dev/reference/react (дата звернення: 07.05.2026).

12. Express.js. Routing guide. URL: https://expressjs.com/en/guide/routing/ (дата звернення: 07.05.2026).

13. Node.js. Introduction to Node.js. URL: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs (дата звернення: 07.05.2026).

14. PostgreSQL Global Development Group. PostgreSQL documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 07.05.2026).

15. Docker Docs. Docker Compose overview. URL: https://docs.docker.com/compose/ (дата звернення: 12.05.2026).

ДОДАТКИ

ДОДАТОК А

Фрагменти програмного коду ілюструють реалізацію основних методів розробки. Лістинги відображають структуру моделі даних, API-методу та сервісу виконання основної операції.

Лістинг А.1 – Опис базової моделі даних.

```
export interface Candidate {
  id: string;
  fullName: string;
  email: string;
  phone?: string;
  status: 'new' | 'screening' | 'interview' | 'offer' | 'rejected' | 'reserve';
  skills: string[];
  matchingScore: number;
  createdAt: string;
  updatedAt: string;
}
```

Лістинг А.2 – Приклад API-методу створення запису.

```
router.post('/api/demo/vacancy-applications', authenticateDemoUser, async (req, res, next)
=> {
  try {
    const demoTenantId = seedContext.tenantId;
    const vacancy = await vacancyRepository.findBySlug('frontend-developer', { tenantId:
demoTenantId });
    if (!vacancy) throw new NotFoundError('Демонстраційну вакансію не знайдено');
    const payload = validateCandidatePayload(req.body);
    const application = await demoCandidateService.createSeedApplication({
      ...payload,
      tenantId: demoTenantId,
      vacancyId: vacancy.id,
      createdBy: req.user.id
    });
    res.status(201).json({ ok: true, demo: true, data: application });
  } catch (error) {
    next(error);
  }
});
  } catch (error) {
    next(error);
  }
});
```

Лістинг А.3 – Сервіс виконання основної операції.

```
type DemoStage = 'draft' | 'screening' | 'interview' | 'offer' | 'rejected';
const demoTransitions = new Map<DemoStage, DemoStage[]>([
  ['draft', ['screening']],
```

```
  ['screening', ['interview', 'rejected']],
  ['interview', ['offer', 'rejected']]
]);
function canMoveDemoCandidate(from: DemoStage, to: DemoStage) {
  return demoTransitions.get(from)?.includes(to) === true;
}
async function runDemoStageMove(candidateId: string, next: DemoStage, actor: UserContext) {
  const candidate = await demoRepository.findCandidate(candidateId, actor.tenantId);
  if (!candidate) throw new NotFoundError('Демонстраційного кандидата не знайдено');
  if (!canMoveDemoCandidate(candidate.stage, next)) throw new DomainError('Недопустимий
перехід етапу');
  const score = await demoMatchingService.calculateScore(candidate.resumeText,
candidate.vacancyId);
  await demoRepository.updateCandidateStage(candidate.id, next, score);
  await auditLog.write({ tenantId: actor.tenantId, actorId: actor.id, resourceType:
'candidate', targetId: candidate.id, action: 'demoStageMove', beforeValue: candidate.stage,
afterValue: next });
  return { candidateId: candidate.id, stage: next, score };
}
```