НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

Комп'ютерних наук

_____ Голуб Б.Л.
(підпис) (ПІБ)

"25" 05 2025р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

Програмне забезпечення інформаційної системи рекрутингу студентів

Спеціальність 121 – «Інженерія програмного забезпечення»

ОПП Інженерія програмного забезпечення

Гарант освітньої програми

___К.т.н., доцент_____ _____ ___Вайганг Г.О.__
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

____К.т.н., доцент___ _____ ___Голуб Б.Л.__
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ ____Бурцев В.М.___
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук
_____ Голуб Б.Л.

"16" грудня 2025р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту
Бурцеву Владиславу Миколайовичу
(прізвище, ім'я, по батькові)

Спеціальність 121 – «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи «Програмне забезпечення інформаційної системи рекрутингу студентів»

затверджена наказом ректора НУБіП України від "16" грудня 2024р. № 2249.С

Термін подання завершеної роботи на кафедру 2025.05.25
(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи:

- https://work.ua/
- https://robota.ua/

Перелік питань, які потрібно розробити:

1. Дослідження предметної області та вимог .
2. Проектування інформаційної частини.
3. Проектування та розробка програмного забезпечення системи.
4. Експлуатація та впровадження системи.

Дата видачі завдання "16" грудня 2024 р.

Керівник бакалаврської кваліфікаційної роботи

К.т.н., доцент _____ Голуб Б.Л.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Бурцев В.М
(підпис) (ПІБ студента)

КИЇВ – 2025

ЗМІСТ

ВСТУП

У сучасному світі інформаційних технологій та глобалізації вища освіта і ринок праці дедалі тісніше взаємодіють між собою. Пандемія COVID-19, стрімке зростання онлайн-сервісів і конкуренція на міжнародному ринку праці підсилили потребу в ефективних інструментах пошуку та відбору молодих спеціалістів. Для студентів і випускників питання першого працевлаштування стає ключовим кроком у побудові професійної кар'єри, а для університетів і компаній — важливим фактором підвищення якості освіти та конкурентоспроможності.

Актуальність теми полягає в тому, що традиційні методи рекрутингу (вігрування резюме, відвідування ярмарків вакансій, пряма взаємодія через університетські центри кар'єри) в умовах надлишку інформації та обмежених ресурсів дедалі втрачають ефективність. За даними міжнародних досліджень, понад 60 % студентів технічних напрямів відчувають невизначеність у виборі першої роботи, а компанії витрачають до 30 % часу рекрутерів на попередню перевірку і верифікацію кандидатів. Нестача прозорості та об'єктивної аналітики призводить до невідповідності між освітніми програмами та потребами ринку праці.

Метою цієї роботи є зменшення часу рекрутерів, забезпечення більшої прозорості та об'єктивної аналітики за допомогою розробки концепції та реалізація інформаційної системи рекрутингу студентів, що об'єднує потреби трьох основних зацікавлених сторін: студентів, університетів і роботодавців. Завдання роботи включають:

1) Аналіз існуючих підходів і платформ для працевлаштування молоді.
2) Визначення функціональних і нефункціональних вимог до системи.
3) Проєктування архітектури програмного рішення з урахуванням принципів модульності, безпеки та масштабованості.
4) Реалізація ключових модулів: профілювання кандидатів, публікація вакансій та аналітика.
5) Тестування та оцінка ефективності розробленого продукту.

Методологічною основою дослідження є принципи об'єктно-орієнтованого проєктування, REST-архітектури, UX-дизайну і практики DevOps. Ідея роботи

полягає у створенні єдиної цифрової екосистеми, що забезпечує автоматизовану верифікацію даних (включно з інтеграцією до ЄДРПОУ), адаптивний підбір вакансій та аналітичні панелі для університетів.

Апробація програмного продукту. Опубліковано тези (**ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ РЕКРУТИНГУ СТУДЕНТІВ**) на Міжнародна науково-практична конференція студентів, аспірантів "Актуальні питання розвитку науки та техніки в умовах глобалізації" (місто Боярка 14.05.25).

Структура записки. У **першому розділі** «Дослідження предметної області та вимог» зібрано чотири підрозділи. Спочатку дається вступ до предметної області, де описуються загальні тенденції автоматизації процесів рекрутингу. Далі деталізуються функціональні та нефункціональні вимоги до майбутньої системи — це своєрідний контракт між замовником і розробником. Третій підрозділ присвячено формуванню уявлення про предметну область за допомогою UML-діаграм, що ілюструють основні сутності та взаємодії. І нарешті, у четвертому підрозділі наведено огляд існуючих рішень на ринку (Robota.UA, Work.ua) із виділенням їхніх сильних і слабких сторін.

Другий розділ фокусується на інформаційній частині системи — виборі бази даних і СУБД, описі структури таблиць із їхніми зв’язками, реалізації процедур та тригерів у СУБД, а також моделі даних у цілому. Тут закладається фундамент для надійного зберігання й обробки даних, що є критично важливим для будь-якого веб-застосунку.

Третій розділ присвячено проектуванню й безпосередній розробці програмного забезпечення. Спочатку коротко обґрунтовано вибір технологій (стек React, Node.js, PostgreSQL тощо), потім описано модульну структуру проекту з розподілом на пакети Web, Auth, Business, Back і Data. Далі розглядаються алгоритми взаємодії та потоки даних між цими модулями, фізична архітектура системи з діаграмою розгортання, а також інтеграція із зовнішніми сервісами (Cloudinary, перевірка ЄДРПОУ тощо).

У **четвертому розділі** «Експлуатація та впровадження системи» наводяться практичні інструкції: як підготувати середовище (Node.js, PostgreSQL, Docker), налаштувати змінні .env, запустити систему локально та в продакшені через CI/CD.

Наприкінці записки наведено **використані джерела** з усіма посиланнями на практичні й теоретичні матеріали та **додатки**, де зібрані всі діаграми, фрагменти коду й скріншоти, які ілюструють роботу системи та її архітектуру. Така послідовність розділів і підрозділів забезпечує повноту викладу і полегшує навігацію читача по документу.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИМОГ

1.1 Вступ до предметної області

Сучасний ринок праці стрімко змінюється під впливом технологій, цифрової трансформації та глобальної конкуренції, що вимагає нових рішень у сфері рекрутингу. Особливо це стосується пошуку молодих талантів — студентів і випускників, які щойно завершують навчання й роблять перші кроки у професійному житті. Саме в цьому контексті актуальним стає створення системи рекрутингу студентів, яка покликана оптимізувати та осучаснити процес найму новачків на ринку праці.

Така система — це спеціалізована IT-платформа, яка трансформує традиційні методи рекрутингу, об’єднуючи різнорідні потоки інформації з боку університетів, роботодавців та самих кандидатів. Її головна мета полягає у створенні відкритого, ефективного та зручного середовища для взаємодії між студентами, які шукають стартову позицію, та компаніями, що зацікавлені в залученні нових фахівців із сучасними знаннями. Окрім цього, система надає університетам важливі аналітичні дані, які допомагають адаптувати освітні програми до потреб ринку.

За результатами аналітичного звіту LinkedIn Global Talent Trends [2], було виявлено низку ключових проблем:

- **65% українських студентів технічних спеціальностей** не мають чіткого розуміння, як правильно організувати процес пошуку першої роботи або стажування;
- **40% компаній** вказують, що аналіз та перевірка студентських резюме часто займає більше часу, ніж інтерв’ювання досвідчених претендентів;
- **лише 15% вітчизняних університетів** мають налагоджену та формалізовану співпрацю з бізнесом, зокрема в частині оновлення навчальних програм та обміну зворотним зв’язком.

1.2 Вимоги до програмної системи

Вимоги до програмної системи виступають своєрідним фундаментом, на якому базується вся подальша розробка та впровадження платформи рекрутингу студентів. Вони окреслюють не лише перелік функцій, які повинні бути реалізовані, але й задають межі продуктивності, стабільності та безпеки системи, щоб вона справді відповідала викликам реального використання. З одного боку, точне визначення функціональних можливостей гарантує, що кінцеві користувачі – і студенти, і представники компаній – отримають усі необхідні інструменти для пошуку та взаємодії: від створення облікових записів і ведення детальних профілів до спрощених механізмів пошуку, подачі заявок та відстеження їх статусу. З іншого боку, увага до нефункціональних аспектів дозволяє забезпечити не лише правильність виконання цих дій, а й комфортну, захищену та безперебійну роботу платформи в довгостроковій перспективі.

При цьому особлива увага приділяється оптимізації швидкості відповіді системи та її масштабованості: зростаюча кількість одночасних запитів від сотень чи навіть тисяч користувачів не повинна негативно позначатися на часу обробки даних чи зручності навігації інтерфейсом. Не менш важливою є безпека збереження персональних даних та конфіденційність комунікацій, адже рекрутингова платформа оперує чутливою інформацією як про студентів, так і про компанії-замовників. Відповідність сучасним стандартам захисту даних і запровадження механізмів аутентифікації й авторизації допомагають попередити будь-які ризики несанкціонованого доступу. Нарешті, інтерфейс системи має бути інтуїтивним і доступним, а його дизайн — адаптивним до різних пристроїв, що дозволяє користувачам із будь-яким рівнем технічної підготовки швидко освоїти всі можливості платформи. У підсумку, лише комплексний підхід до опису вимог – і функціональних, і нефункціональних – забезпечує створення надійного та зручного інструменту для ефективного рекрутингу студентів.

1.2.1 Функціональні вимоги. Розробка запропонованої системи охоплює предметну область, що стосується цифрової взаємодії між трьома основними цільовими групами користувачів:

- **Студенти** зможуть створювати повноцінні профілі, які містять відомості про освіту, професійні навички, сертифікати, участь у проєктах, володіння мовами, результати тестувань тощо. Також буде доступна можливість додавання портфоліо, відеорезюме, рекомендацій.
- **Компанії** отримують зручний інтерфейс для пошуку, сортування та перегляду кандидатів за заданими параметрами. Вони зможуть публікувати вакансії та стажування, переглядати відгуки, організовувати онлайн-інтерв'ю або асесмент-центри безпосередньо через платформу.
- **Університети** матимуть доступ до аналітичної панелі з даними про подальше працевлаштування випускників, популярність спеціальностей, середній час пошуку роботи та інші метрики, які дозволять ефективніше модернізувати навчальні програми й встановлювати зв'язки з ринком праці.

До основних модулів системи належать:

- **Модуль профілювання студентів**, який підтримує завантаження мультимедійних файлів (презентацій, відео, PDF-портфоліо).
- **Аналітичний модуль** для університетів із візуалізацією даних у вигляді графіків, діаграм, звітів (наприклад, відсоток студентів, що знайшли роботу протягом 3 місяців після випуску).
- **Інструмент для рекрутерів**, що забезпечує доступ до динамічної бази кандидатів із фільтрами за факультетом, містом, рівнем англійської, наявністю досвіду або сертифікатів.

З метою гарантування безпеки та достовірності даних у систему вбудовано низку заходів:

- **Верифікація компаній** через базу ЄДРПОУ (Єдиного державного реєстру підприємств та організацій України) для захисту від фейкових роботодавців.
- **Шифрування персональної інформації** студентів та зберігання її відповідно до стандартів безпеки (зокрема, вимог GDPR).

• **Модерація контенту** — перед публікацією всі вакансії, стажування та коментарі перевіряються на відповідність правилам платформи.

Загалом, така система створює цілісну екосистему професійного зростання, де кожен учасник — студент, компанія чи університет — отримує реальні інструменти для досягнення своїх цілей. Студенти можуть презентувати свій потенціал у зручному цифровому середовищі, компанії — ефективно шукати мотивованих молодих фахівців, а університети — використовувати дані для стратегічного розвитку своїх освітніх напрямів.

По-перше, студенти отримують змогу реєструватися винятково з корпоративної пошти свого університету (наприклад, @nubip.edu.ua) і завантажувати власні резюме у зручному форматі. Це дає змогу переконатися у справжності абітурієнтів та автоматизувати збір інформації про їхню освіту й досвід. Компанії та їхні рекрутери проходять окремі процедури верифікації: одна — через перевірку коду ЄДРПОУ з бази державних реєстрів, інша — обов’язкова двофакторна аутентифікація із прив’язкою кожного рекрутера до конкретної юридичної особи. Це гарантує, що лише уповноважені фахівці зможуть створювати й оновлювати вакансії, вказуючи ключові параметри — від назви позиції й опису до вимог, рівня заробітної плати й географії. Щоб уникнути дублювання оголошень, система порівнює щойно створені записи з наявними й попереджає про можливі повтори.

Додатково абітурієнти отримують інструменти для пошуку роботи: вони можуть фільтрувати відкриті вакансії за спеціальністю, місцем навчання та містом розташування роботодавця, а також залишати відгуки та відстежувати статус своїх заявок. Університети, зі свого боку, мають доступ до детальних звітів в PDF-форматі, які містять статистику активних студентів на платформі, відсоток успішних відгуків і перелік найактивніших компаній-партнерів. Такий підхід сприяє прозорості процесів і дозволяє навчальним закладам коригувати власні стратегії співпраці з ринком праці.

1.2.2 Нефункціональні вимоги. Що стосується технічних очікувань, система повинна реагувати на запити користувачів практично миттєво: серверні запити обробляються без затримок, а інтерфейс клієнтської частини оновлюється плавно та

швидко. З метою збереження ефективності при зростанні навантаження застосовується модульна архітектура, оптимізована для розгортання в хмарних середовищах (AWS чи GCP). Інформація про користувачів і сесії захищається за допомогою JWT-токенів та надійного шифрування, водночас регулярне резервне копіювання бази даних PostgreSQL і покриття ключових функціональних модулів юніт-тестами гарантують стійкість системи до збоїв і швидке відновлення у випадку непередбачених ситуацій.

1.3 **Формування уявлення про предметну область**

Формування уявлень про предметну область починається з використання UML-діаграм — спеціалізованих графічних засобів для візуалізації компонентів системи та їхніх взаємозв’язків. UML (Unified Modeling Language) є універсальною мовою моделювання, яка задає єдиний стандарт для опису структури, поведінки та взаємодії об’єктів у будь-якій сфері розробки програмного забезпечення.

Першочергово UML-діаграми застосовують для того, щоб отримати цілісну картину майбутньої системи задовго до написання коду. Це допомагає уникнути неоднозначностей щодо того, які дані зберігатимуться, а які операції над ними виконуватимуться.

До складу UML входить 14 різновидів діаграм (рис. 1), кожна з яких фокусується на власному аспекті системи: від статичної структури класів і компонентів до динамічних сценаріїв виконання процесів і взаємодії об’єктів. Використання цих моделей на ранніх етапах проектування дозволяє чітко окреслити межі системи, розподіл відповідальності між її елементами та передбачити потенційні точки розширення або оптимізації ще до початку написання коду.

У рамках даного дослідження будуть застосовані три ключові діаграми. Діаграма прецедентів (рис. 1) покаже, які ролі (студент, компанія, рекрутер, університет) взаємодіють із системою та у яких сценаріях. Діаграма послідовностей (рис. 2) відтворить кроки обміну повідомленнями між об’єктами під час виконання

найважливішої операції — подачі заявки на вакансію. Нарешті, діаграма активностей (рис. 3) продемонструє внутрішню логіку обробки цих заявок, від перевірки на дублікати до оновлення статусу кандидата, що допоможе виявити й прибрати «вузькі місця» процесу.

1.3.1 Діаграми прецедентів. Необхідна для того, щоб на найвищому рівні зафіксувати всі ключові сценарії використання системи та ролі її учасників. Вона дозволяє одразу побачити, хто і за яких обставин взаємодіє із платформою, не заглиблюючись при цьому в технічні деталі реалізації.

На ній зображено дві основні категорії об’єктів: акторів (зовнішніх користувачів або інших систем, які ініціюють певні дії) та прецеденти (конкретні функціональні можливості або серії кроків, які виконує система у відповідь на дії актора). Актори позначаються стилізованими людиноподібними фігурами, а прецеденти — овалами з лаконічними назвами («Відгук на вакансії», «Отримання звіту про студентів» тощо). Зв’язки між акторами й прецедентами показують, хто саме може робити ту чи іншу дію, а також відображають варіанти розширення чи включення одних прецедентів в інші.

На рис. 1 наведено діаграму прецедентів досліджуваної предметної області, яка ілюструє ключові ролі користувачів (студент, рекрутер, компанія, університету) та їхні основні сценарії взаємодії з платформою. Ця модель дозволяє наочно побачити, які функції система повинна підтримувати для кожної категорії акторів, і слугує орієнтиром під час подальшого детального опрацювання технічних вимог і розробки архітектури.

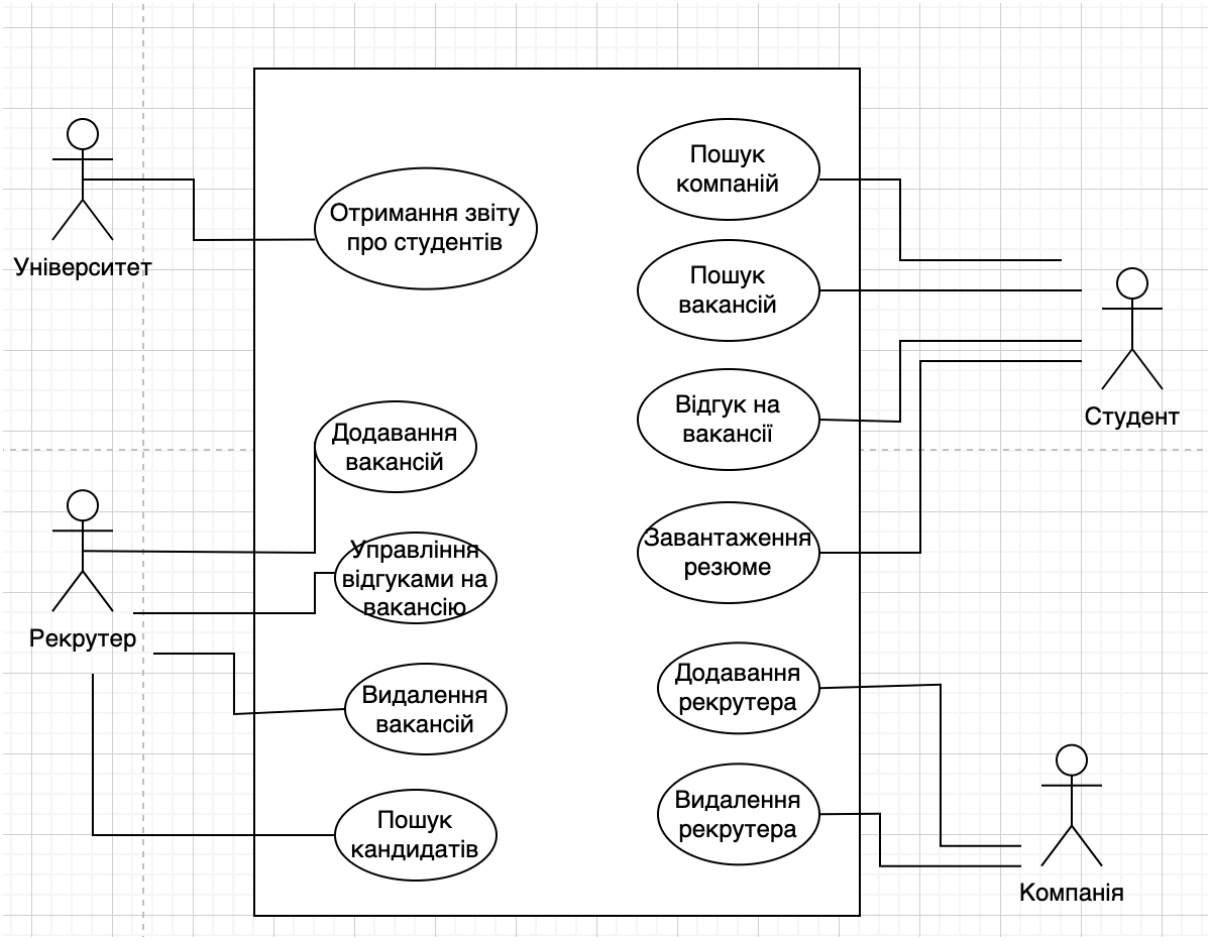


Рис. 1 Діаграма прецедентів

На діаграмі прецедентів чітко позначені чотири типи акторів, кожен із яких користується власним набором функціональних сценаріїв. Представник університету має доступ до генерування звітів про студентську активність: він отримує інформацію про кількість відгуків, успішне працевлаштування та інші показники, що дозволяють оцінити ефективність платформи та взаємодію вишу зі студентами.

Рекрутер представлений як основний виконавець завдань із набору персоналу: він створює, редагує та видаляє вакансії, паралельно керуючи судами відгуків — переглядає заявки, підтверджує кандидатів або відхиляє їх. Окрім того, йому доступний механізм пошуку претендентів за різними критеріями (спеціалізація, університет, місто тощо), що допомагає швидко знайти найбільш відповідних кандидатів.

Студент взаємодіє з платформою через завантаження власного резюме, пошук вакансій та компаній за цікавою спеціалізацією та місцем розташування, а також безпосередньо подає відгуки на обрані позиції. Кожна з цих дій позначена на діаграмі

як окремий прецедент, що відображає послідовність кроків — від вибору оголошення до відправлення заявки.

Компанія моделюється як юридична одиниця, яка делегує своїм рекрутерам право публікувати і керувати вакансіями. На діаграмі видно, що саме компанія додає або видаляє рекрутерів у своєму підрозділі, а вже через них здійснюється весь цикл роботи з позиціями та відгуками.

Окремо підкреслені взаємодії між акторами: компанія уповноважує рекрутерів на управління вакансіями, студент звертається до рекрутера через систему відгуків, а університет отримує зведені дані про студентів, щоб аналізувати їхню активність та рівень працевлаштування. У сукупності така модель ілюструє всі ключові сценарії використання платформи та демонструє, які сервіси і в якій послідовності пропонуються кожному типу користувачів.

1.3.2 Діаграма послідовностей. Діаграма послідовностей ілюструє детальний сценарій взаємодії п’яти ключових учасників процесу: Студента, Веб-платформи, Рекрутера, Компанії та Університету. Спочатку Рекрутер через інтерфейс Веб-платформи створює нову вакансію, яка одразу стає доступною для перегляду. Далі Студент, використовуючи засоби фільтрації на платформі, знаходить відповідну позицію та ініціює подачу заявки: його запити надходять до сервера, де перевіряються на коректність і повноту даних, після чого заявка зберігається в базі. У разі помилок платформа повертає відповідь із проханням виправити некоректні поля, а вдалий запит підтверджується відображенням статусу «Надіслано».

Потім Рекрутер отримує сповіщення про новий відгук і через свою лінію життя на діаграмі переглядає список кандидатів, відбираючи тих, що відповідають вимогам: на цьому етапі він може підтвердити студента для подальших кроків або відхилити заявку. У разі відбору Система ініціює обмін повідомленнями з Компанією, яка надсилає офіційну пропозицію контракту. Після того як Студент підписує документ, Веб-платформа оновлює статус заявки на «Успішно працевлаштовано» та зберігає фінальні дані в базі.

Нарешті, коли Університет потребує підсумкових даних про працевлаштування, він надсилає запит до Веб-платформи, яка вибірково формує звіт на основі інформації

з бази даних і передає його у вигляді готового PDF. Усі ці кроки, від створення вакансії до генерації звіту, викладені в хронологічному порядку на діаграмі послідовностей (Додаток А), що допомагає чітко побачити часові інтервали між обміном повідомлень і визначити потенційні «вузькі місця» у логіці бізнес-процесів.

1.3.3 Діаграма активності. У діаграмі активності (Додаток Б), яка ілюструє ключові бізнес-процеси системи рекрутингу студентів та розподіл відповідальності між основними учасниками. Кожна паралельна лінія відповідає «плавцю» (swimlane) для конкретного типу користувача: Студента, Рекрутера, Представника компанії, Працівника університету. Такий підхід дозволяє наочно відобразити не лише послідовність дій, а й взаємодію між різними ролями.

З боку Студента активність починається з входу на платформу й переходить до пошуку вакансій через фільтри та ключові слова. Після знаходження цікавого оголошення абітурієнт відкриває його картку, знайомиться з умовами та за необхідності завантажує або оновлює своє резюме, перш ніж натиснути кнопку «Відгукнутися». Далі йде етап підготовки до співбесіди — студент отримує повідомлення про запрошення від Рекрутера, проходить інтерв’ю або тестування, і в разі успішного результату отримує фінальне сповіщення про працевлаштування.

Рекрутер у своїй частині діаграми створює вакансії, встановлює їхні параметри й публікує на платформі. Після появи відгуків він переглядає заявки, відзначає тих кандидатів, яких запрошує до наступного етапу відбору, і надсилає їм відповідні повідомлення. Водночас Представник компанії контролює діяльність рекрутерів: додає або призначає нових фахівців, перевіряє опубліковані вакансії на відповідність внутрішнім стандартам та затверджує їх до розміщення.

Коли рекрутингова кампанія досягає свого завершення, Працівник університету звертається до модулю аналітики — він відправляє запит на формування звіту про працевлаштованих студентів, отримує готовий документ у форматі PDF і використовує його для оцінки ефективності освітніх програм. Окремо Адміністратор моніторить загальну роботу платформи, перевіряє достовірність акаунтів, виявляє й видаляє некоректні або дубльовані вакансії, забезпечуючи безперебійну та захищену роботу всієї системи.

Таким чином, діаграма активності наочно демонструє основні робочі потоки та взаємопов’язані дії всіх учасників, сприяючи чіткому розумінню процесів і полегшуючи їх подальшу реалізацію в архітектурі програмного забезпечення.

1.4 Огляд наявних рішень

У цьому розділі здійснюється систематичний огляд існуючих на ринку сервісів, які виконують завдання рекрутингу та підбору кадрів, з метою виявлення їх сильних і слабких сторін. Такий аналіз допомагає краще зрозуміти, які інструменти й підходи вже зарекомендували себе передовими, а де є простір для інновацій. Крім того, дослідження конкурентних платформ дозволяє дізнатися, які функції користуються найбільшим попитом серед студентів та роботодавців, як організовано взаємодію, наскільки зручним є інтерфейс і як налагоджено процес комунікації.

З огляду на вітчизняний ринок, до числа головних конкурентів обрано портали robota.ua[2] та Work.ua[2], які мають значну аудиторію й стабільно утримують лідерські позиції у сфері пошуку роботи. При розгляді цих платформ буде звернено увагу на такі аспекти: механізми реєстрації та пошуку кандидатів, можливості фільтрації вакансій, якість та швидкість отримання зворотного зв’язку, наявність мобільних застосунків і інтеграцій із зовнішніми сервісами. Аналіз дозволить виокремити кращі практики, наприклад, автоматичне підбирання релевантних пропозицій чи гнучку систему сповіщень, і визначити, які рішення потрібно вдосконалити або доповнити в нашій розробці.

Завдяки порівняльному підходу стає можливим сформувати чітке уявлення про розриви на ринку та створити платформу з унікальним набором переваг. У подальшому це допоможе спроєктувати інтерфейс та функціонал, які не лише відповідатимуть очікуванням користувачів, але й перевершать існуючі пропозиції robota.ua та Work.Ua за зручністю, швидкістю та рівнем персоналізації.

1.4.1 Недоліки robota.ua. У дослідженні robota.ua[2] (рис. 2) висвітлюються низка суттєвих недоліків, які впливають на зручність та ефективність роботи користувачів із платформою. Насамперед, безкоштовні акаунти залишаються обмеженими у своїх можливостях: абітурієнти не мають змоги бачити детальну статистику по відгуках на свої резюме і не можуть претендувати на пріоритетне відображення в списку кандидатів без додаткової оплати. До того ж сама кількість резюме, які одночасно можуть «жити» у профілі, стримана жорсткими лімітами, що не дозволяє активним шукачам утримувати кілька варіантів документів та швидко реагувати на нові вакансії.

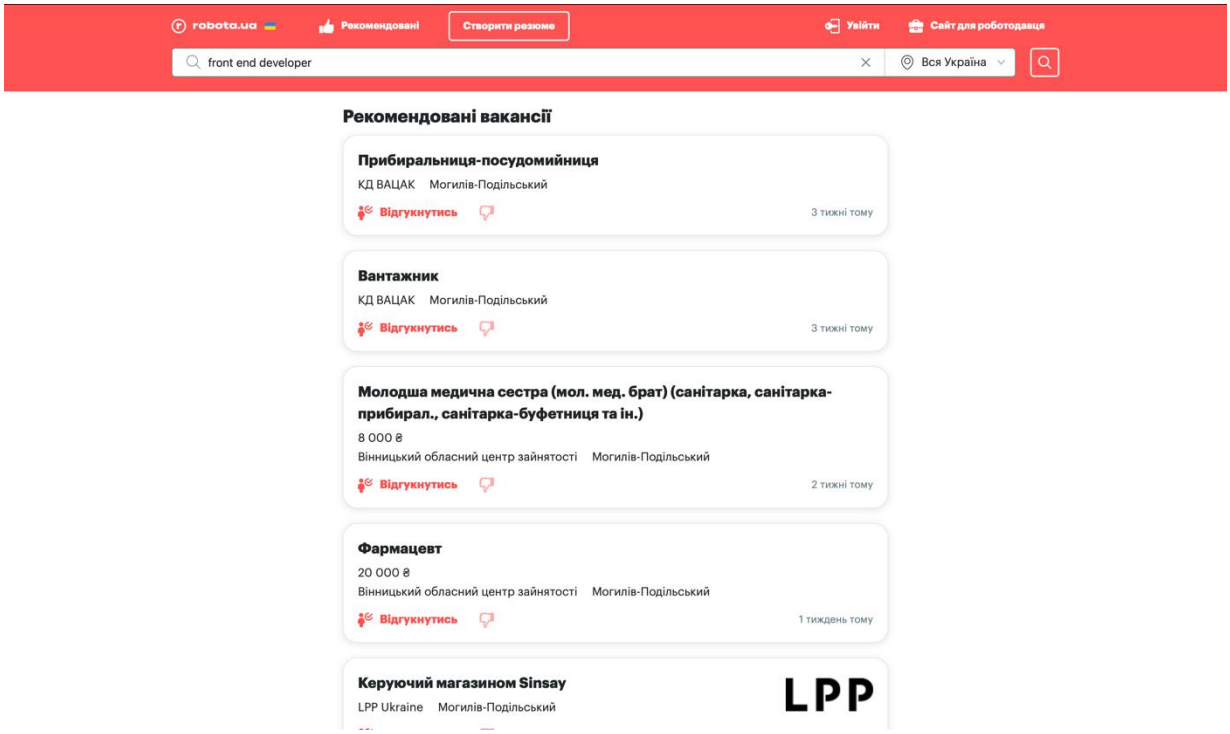


Рис. 2 Інтерфейс сайту Robota.ua

Блоки з пропозиціями та спонсорованими оголошеннями займають значну частину екрану, змушуючи користувача прокручувати сторінку в пошуках звичайних вакансій. Окрім того, відсутність гнучких фільтрів, що дозволяли б сортувати вакансії за професійними кваліфікаціями чи роками досвіду, змушує вдовжувати процес пошуку й часто призводить до необхідності вручну перелічувати сотні нерелевантних пропозицій.

Проблема контролю якості оголошень на robota.ua також є гострою: досить часто зустрічаються фейкові вакансії (рис. 3) або пропозиції із застарілою інформацією, які

не проходять жодної серйозної модерації. Опис компанії іноді дублюється з інших джерел без перевірки або підтримки актуальності, тому реально зайнятися пошуком роботи доводиться обережно й не завжди з першої спроби можна знайти достовірні контакти.

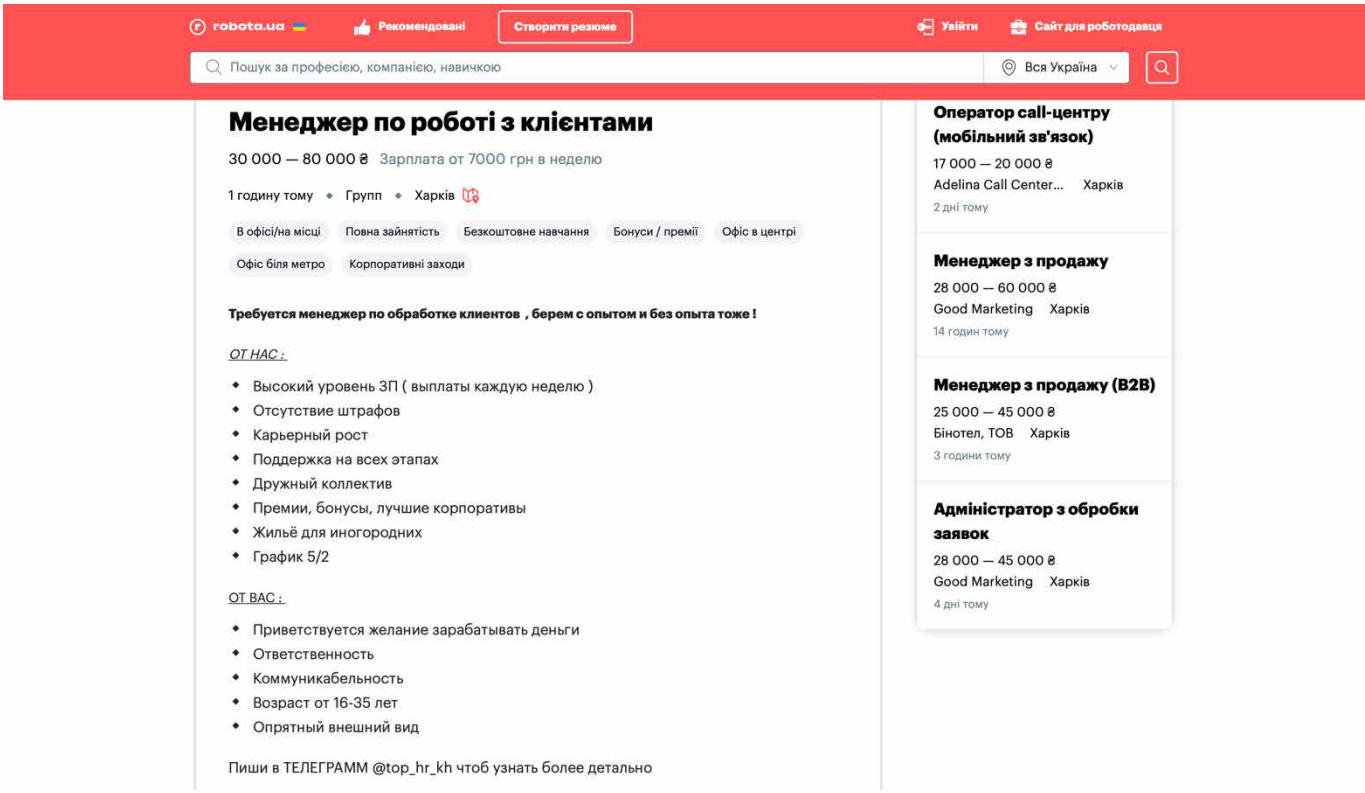


Рис. 3 Вакансія «Офісу»

Ще одним недоліком є недостатня персоналізація підбору вакансій (див. рис. 2) основуючись на профілі користувача (рис. 4). Алгоритми рекомендацій часто підсовують користувачу пропозиції, що не відповідають його навичкам чи бажаному формату роботи (наприклад, намагаються поєднати віддалену позицію з вимогою перебувати офісу чи навпаки). Опції налаштування сповіщень дуже обмежені, і навіть базова категорія «віддалена робота» не дозволяє отримувати листи тільки по тих вакансіях, які справді цікавлять.

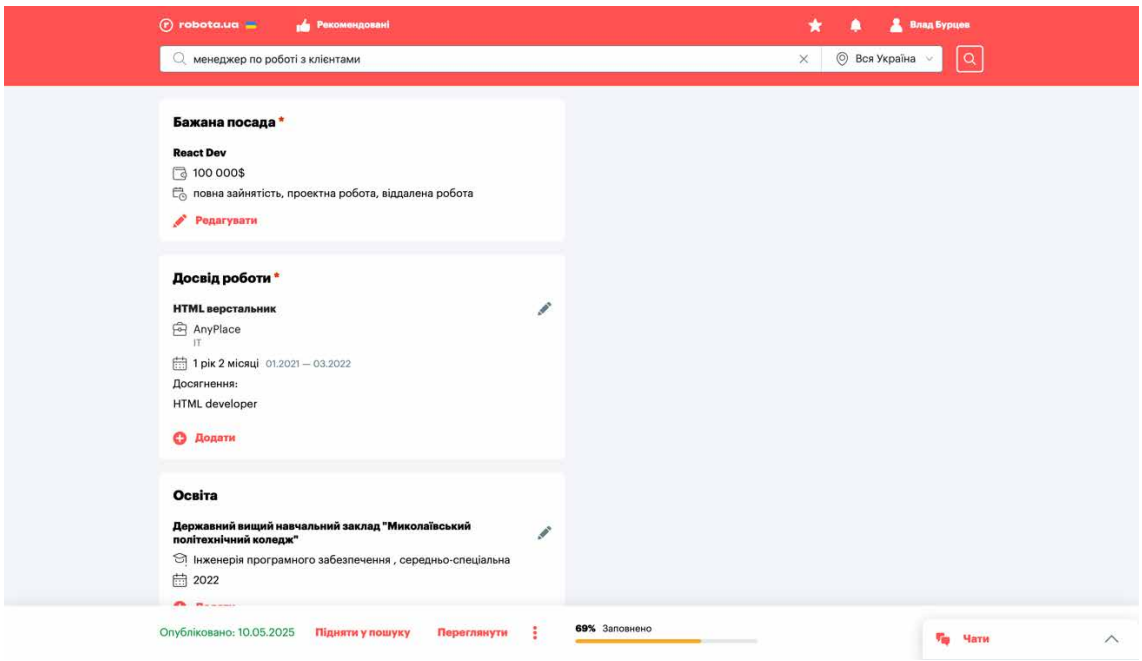


Рис. 4 Профіль користувача

Технічна сторона платформи іноді підводить користувачів: під час пікових періодів навантаження сторінки завантажуються повільно (рис. 5), а самі сервіси прийому резюме й відправки відгуків можуть кілька разів «відвалюватися». Це особливо помітно в години пік, коли одночасно на сайт заходить велика кількість студентів і роботодавців. У результаті процес пошуку та відгуку затягується, що знижує загальну ефективність використання robota.ua.

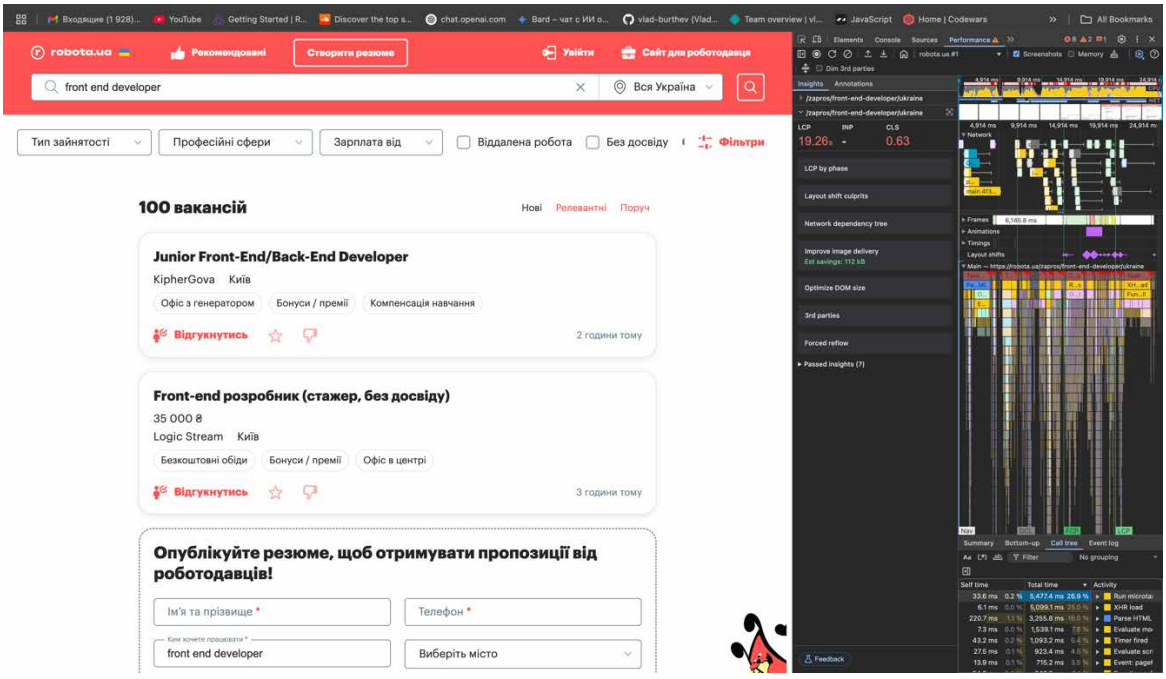


Рис. 5 Продуктивність веб-платформи robota.ua при високому навантажені

1.4.2 Недоліки Work.ua. Попри свою популярність, часто підводить студентів і молодих фахівців через низку недоліків, помітних уже на головній сторінці та в процесі повсякденного використання. По-перше, система фільтрації вакансій не дає змоги звузити пошук до потрібних параметрів: жорсткі обмеження за локацією, зарплатою чи спеціалізацією змушують переглядати сотні нерелевантних оголошень, а спроба знайти вузькоспеціалізовану позицію часто завершується нульовим результатом із пропозиціями, віддалено пов'язаними із запитом.

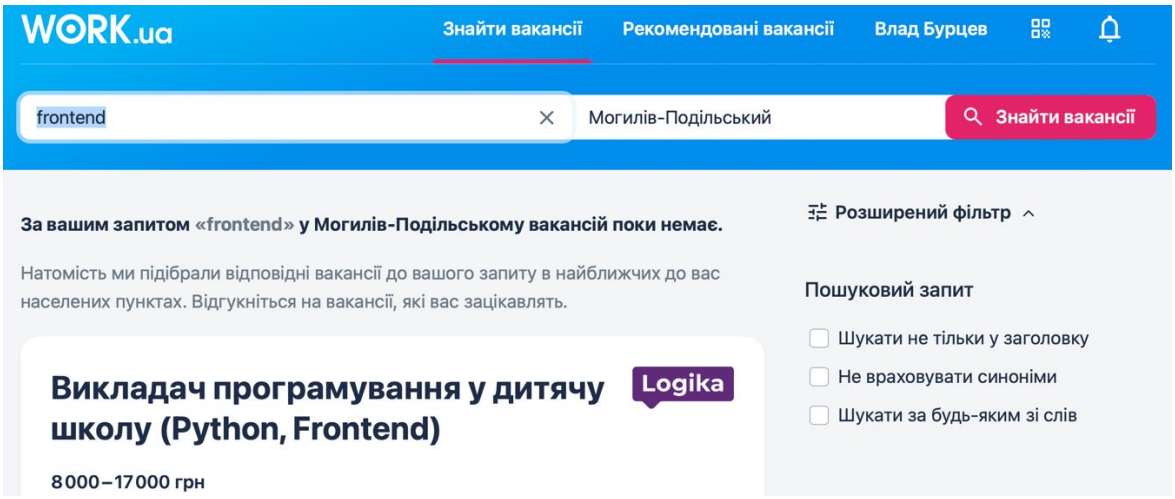


Рис. 6 Налаштування фільтрів

По-друге, після публікації власного резюме користувачі зазвичай стикаються з хвилею дзвінків і листів із пропозиціями, що не відповідають їхньому профілю, — це вказує на слабку перевірку роботодавців і низький контроль якості вакансій. У поєднанні з регіональною обмеженістю, коли в невеликих містах або взагалі немає підходящих вакансій, або доводиться задовольнятися пропозиціями із сусідніх населених пунктів, це створює відчуття марної витрати часу.

Ще один недолік — нав'язлива порада «відгукнутися на 7 вакансій» (рис. 7), яка більше тисне на кількість, аніж заохочує шукати найкращий варіант, і формує уявлення про надмірну конкуренцію. Окрім того, на порталі досі можна натрапити на застарілі оголошення, розміщені ще три тижні тому, але досі не вилучені з пошуку, хоча реальні вакансії давно закриті.

Рис. 7 Порада «Відгукніться ще на 7 вакансій»

Узагальнюючи вищезазначене, незважаючи на визнання та широку аудиторію robota.ua й Work.ua, їхні обмеження суттєво ускладнюють життя студентам, які шукають першу роботу або проходять стажування. По-перше, відсутність належної перевірки вакансій підвищує ймовірність потрапити на фейкові або неактуальні пропозиції. По-друге, алгоритми обох платформ не враховують індивідуальні вподобання та специфіку студентського профілю, через що рекомендації виявляються нерелевантними й не допомагають у цілеспрямованому пошуку. Саме ці вразливі місця обґрунтовують необхідність створення окремого рішення — спеціалізованої платформи для студентського рекрутингу з поглибленою верифікацією оголошень, багатими безкоштовними інструментами та гнучкою системою персоналізованих рекомендацій.

1.5 Формулювання завдання

Формулювання завдання є критично важливим етапом будь-якого проєкту, оскільки воно задає чіткі межі та очікування від системи ще до початку розробки. По-перше, чітко виписане завдання допомагає уніфікувати розуміння між замовником, аналітиками та розробниками: прописані цілі, вимоги та критерії успіху забезпечують усім сторонам однаковий «компас», за яким вони рухаються. По-друге, детальне завдання дозволяє звести до мінімуму ризики «роздуття» функціоналу (так званого scope creep), чітко визначивши, які можливості є обов’язковими, а які — додатковими. По-третє, на його основі можна прогнозувати терміни й ресурси: оцінити обсяг робіт, розподілити завдання між командою й скласти реалістичний графік. Нарешті,

формулювання завдання створює фундамент для тестування та валідації: саме від опису цілей і вимог відштовхуються сценарії приймальних випробувань, що гарантує, що кінцевий продукт дійсно виконає покладені на нього функції.

1.5.1 Мета розробки. Мета створення цієї системи полягає в тому, щоб об’єднати студентів, роботодавців та навчальні заклади в єдиному цифровому середовищі та оптимізувати увесь цикл працевлаштування. Розроблене програмне забезпечення автоматизує ключові етапи взаємодії — від розміщення й пошуку вакансій до подачі та опрацювання відгуків — і надає університетам інструменти для оперативного моніторингу активності своїх студентів. Завдання полягає в тому, щоб спростити рекрутингові процедури, забезпечити швидкий і точний підбір кандидатів завдяки зручним фільтрам та механізмам відстеження статусу заявок, а також сформувати високоякісні аналітичні звіти, що допоможуть як компаніям, так і вишам приймати обґрунтовані рішення для підвищення ефективності програми стажувань і працевлаштування. Зрештою, проєкт має націлити платформу на потреби студентів-новачків, для яких важливо отримати простий, інтуїтивний і безпечний сервіс з мінімальними бар’єрами на шляху до першої роботи.

1.5.2 Основні завдання. Визначаються для деталізації ключових кроків, які необхідно виконати для реалізації інформаційної системи рекрутингу студентів.

Першим завданням є проєктування архітектури системи. На цьому етапі потрібно створити модульну структуру, що включає окремі компоненти backend-сервісів, frontend-інтерфейсу та сховища даних. Визначення технологічного стеку полягає у виборі сучасних і взаємопов’язаних інструментів — наприклад, React для клієнтської частини, Node.js для серверної логіки та PostgreSQL як реляційної бази даних.

Далі слідує безпосередня реалізація функціональних модулів. Обов’язковим є розробка механізму реєстрації та автентифікації з валідацією email-доменів університетів, аби гарантувати, що лише справжні студенти можуть створити акаунт. Одночасно створюється система публікації та пошуку вакансій: користувачі зможуть додавати оголошення з фільтрацією за спеціальністю, містом та рівнем зарплати. Наступним кроком передбачено впровадження персоналізованих рекомендацій на

базі алгоритмів машинного навчання, які аналізують профіль користувача та його попередню активність. Окремим підсегментом є розробка аналітичної панелі для університетів із формуванням статистики працевлаштування та відстеженням активності студентів.

Третій етап передбачає тестування системи та її оптимізацію. Потрібно провести навантажувальні випробування для оцінки продуктивності, а також аудит безпеки, щоб переконатися в надійності захисту даних та стійкості до можливих атак.

Нарешті, останнім завданням є створення повноцінної документації. Це включає технічний опис архітектури, котрий стане опорою для майбутніх розробників, а також користувацькі посібники для студентів, рекрутерів і адміністраторів, які пояснюватимуть порядок роботи з кожним з модулів платформи.

1.5.3 Актуальність завдання. Актуальність розробки спеціалізованої платформи для студентського рекрутингу обумовлена відсутністю на ринку інструментів, які б одночасно забезпечували достовірність інформації про вакансії, гнучкість безкоштовного функціоналу та глибоку аналітику результатів. Сьогодні студенти стикаються з ризиком натрапити на фейкові чи застарілі оголошення, а університети та роботодавці не мають достатніх механізмів для оцінки ефективності взаємодії та своєчасного коригування своїх кадрових ініціатив. В умовах динамічних змін на ринку праці й зростаючої конкуренції за перші робочі місця наявність прозорої, безпечної та адаптивної системи рекрутингу стає критичною для успішного початку кар’єри студентів. Саме тому створення платформи, що інтегрує перевірку даних, персоналізовані рекомендації та детальну звітність, здатне значно підвищити якість і швидкість підбору кадрів у вищих навчальних закладах та компаніях-партнерах.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ ЧАСТИНИ

2.1 Вибір БД та СУБД

При виборі системи зберігання даних для проєкту рекрутингу студентів основним завданням було забезпечити надійність, гнучкість і безпеку обробки інформації. Реляційна модель бази даних ідеально відповідає вимогам предметної області: вона дозволяє коректно відображати зв’язки між студентами, університетами та компаніями, зручно реалізувати складні запити та гарантувати цілісність даних під час одночасного доступу великої кількості користувачів. Крім того, важливою умовою стала наявність можливості реактивного розширення структури БД у майбутньому — наприклад, за потреби додати підтримку нових аналітичних звітів або мультимедійних елементів у профілях.

Відкриті СУБД із повною підтримкою транзакційних операцій та стандартів безпеки виявилися оптимальними для освітнього проєкту, де важливі відсутність ліцензійних обмежень і прозорість у керуванні. Вибір впав на PostgreSQL завдяки її широкій екосистемі розширень, серед яких JSONB для зберігання напівструктурованих даних і інструменти повнотекстового пошуку. Ця СУБД добре масштабується і відзначається стабільністю навіть при значних навантаженнях, що робить її ідеальним рішенням для платформи з великим потоком запитів та багаторівневою верифікацією даних.

Для адміністрування бази даних обрана графічна оболонка pgAdmin, яка спрощує розробникам і адміністраторам роботу зі схемами, запитами та моніторингом продуктивності. Зручний інтерфейс дозволяє швидко створювати та змінювати таблиці, налаштовувати ролі доступу та аналізувати плани виконання SQL-запитів без необхідності звертатися до командного рядка. Такий підхід значно прискорює початкову настройку середовища та полегшує подальшу підтримку системи в процесі її експлуатації.

Крім перелічених переваг, варто відзначити, що широке та детальне офіційне документування робить процес інсталяції, налаштування та використання PostgreSQL

доступним навіть для новачків. Відкритий код системи стимулює розвиток численних форумів, блогів та спільнот (Stack Overflow, офіційні mailing-листи, локальні MeetUp-групи), де можна швидко знайти відповіді, приклади налаштувань, поради від досвідчених розробників та повідомлення про вакансії, пов’язані з PostgreSQL.

2.2 Структура БД, таблиці та зв’язки

При проєктуванні бази даних для системи рекрутингу студентів кожна сутність відображається окремою таблицею, створеною з допомогою DDL-команд CREATE TABLE (рис. 8). Використання конструкції IF NOT EXISTS у запиті гарантує, що сторінка схеми може виконуватися багаторазово без помилок у випадку вже наявних таблиць. Структура кожної таблиці визначається набором стовпців із вказаними типами даних — від UUID для унікальних ідентифікаторів до TEXT[] для масивних полів — і обмеженнями, що забезпечують цілісність і коректність інформації.

Для кожної сутності застосовано принцип використання UUID як первинних ключів із генерацією uuid_generate_v4(). Це дозволяє підвищити безпеку та уникнути послідовного перебору ідентифікаторів. Поля типу VARCHAR із обмеженнями UNIQUE NOT NULL гарантують, що критично важлива інформація — наприклад, електронні адреси — не дублюється і завжди заповнена. Додаткові атрибути, такі як password, name, surname та специфічні для кожної ролі поля (наприклад, egrpou для компаній чи university_id для студентів), чітко розмежовують бізнес-логіку платформи і дозволяють коректно ідентифікувати та верифікувати користувачів.

Зв’язки між таблицями реалізовані через зовнішні ключі з опціями дій при видаленні записів (ON DELETE CASCADE чи ON DELETE SET NULL) (Додаток В). Завдяки цьому видалення компанії автоматично видаляє пов’язаних рекрутерів та їх вакансії, а видалення університету не призводить до втрати студентських профілів, а лише очищує відповідне поле. Додаткові обмеження, такі як CHECK (salary_to >= salary_from) або перелік допустимих статусів у заявках, гарантують, що дані завжди відповідають встановленим бізнес-правилам.

```sql
CREATE TABLE IF NOT EXISTS companies (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    email VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    name VARCHAR(100) NOT NULL,
    surname VARCHAR(100) NOT NULL,
    title VARCHAR(255) NOT NULL,
    description TEXT,
    phone VARCHAR(20),
    logo VARCHAR(255),
    egrpou VARCHAR(8) NOT NULL,
    role_id INTEGER NOT NULL REFERENCES roles(id),
    is_activated BOOLEAN DEFAULT false,
    activation_link VARCHAR(255),
    activation_link_expires_at TIMESTAMP WITH TIME ZONE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);
```

Рис. 8 Приклад створення таблиці

Для відстеження життєвого циклу кожного запису передбачені поля created_at та updated_at з типовим значенням CURRENT_TIMESTAMP. Це дозволяє вести аудит змін і аналізувати активність користувачів. Усі таблиці мають поле is_activated або is_activated BOOLEAN DEFAULT true/false, яке контролює доступ та статус облікових записів. Такий підхід до створення таблиць забезпечує гнучкість моделі даних, зручність розширення схеми при додаванні нових сутностей і високий рівень захисту й узгодженості інформації в системі.

2.3 Створення уявлень, процедур, тригерів

У процесі проєктування системи було реалізовано уявлення (views), збережені процедури (functions) та тригери (triggers), які автоматизують обробку даних, спрощують аналітику та забезпечують актуальність інформації.

Уявлення — це віртуальні таблиці, які формуються на основі результату SQL-запиту. Вони не зберігають дані фізично, але дозволяють зручно об'єднувати та аналізувати інформацію з кількох таблиць.

У системі створено наступні уявлення (Додаток Д):

- company_statistics — містить статистику по кожній компанії: кількість рекрутерів, вакансій, найнятих студентів і нових заявок за поточний місяць.
- recruiter_performance — надає дані про ефективність рекрутерів: кількість вакансій, заявок, успішних наймів та нових заявок за місяць.

Процедури — це набір SQL-інструкцій, які можна повторно використовувати для виконання складних запитів або бізнес-логіки.

У системі реалізовано (Додаток Д):

- get_date_range(period) — повертає часовий інтервал залежно від вказаного періоду (current_month, last_month, last_half_year, last_year, тощо).
- get_company_metrics(company_id, period) — повертає метрики для компанії за обраний період, включаючи кількість вакансій, заявок, нових рекрутерів, активних рекрутерів та наймів.
- get_recruiter_metrics(recruiter_id, period) — надає статистику для конкретного рекрутера, включаючи кількість вакансій, заявок, наймів та коефіцієнт успішності (success rate).

Тригери — це спеціальні об'єкти в базі даних, які автоматично виконуються при виникненні певних подій (наприклад, INSERT, UPDATE або DELETE). У системі тригери використовуються для автоматичного оновлення поля updated_at при зміні записів у таблицях.

Було створено загальну функцію update_updated_at_column() (Додаток Е), яка оновлює час зміни. Ця функція використовується у тригерах:

- update_companies_updated_at
- update_universities_updated_at
- update_students_updated_at
- update_recruiters_updated_at
- update_vacancies_updated_at
- update_applications_updated_at

Ці тригери забезпечують автоматичне оновлення поля updated_at при кожному редагуванні відповідного запису, що дозволяє точно відслідковувати зміни в базі даних.

В цілому реалізація уявлень, процедур і тригерів дозволяє централізовано зберігати логіку обробки даних, забезпечити автоматизацію типових операцій і підтримувати актуальність інформації без необхідності ручного втручання.

2.4 Модель даних системи

У базі даних системи рекрутингу студентів реалізовано шість основних сутностей (Додаток Ж), які відображають ключові об’єкти предметної області: **Компанія**, **Рекрутер**, **Університет**, **Студент**, **Вакансія** та **Заявка**. Кожна з цих таблиць відповідає за зберігання окремого набору атрибутів: від облікових даних і контактної інформації до статусів активації й часових міток створення та оновлення записів. Таке розділення забезпечує зрозумілість моделі та полегшує подальше розширення системи новими сутностями або властивостями.

Сутність **Компанія** виступає центром екосистеми: вона має унікальний ідентифікатор, реквізити (ЄДРПОУ, назва, опис, лінк активації) та відповідає за формування вакансій. Через зовнішній ключ company_id до таблиці **Компанія** прив’язуються як таблиця **Рекрутер**, так і **Вакансія**, що відображає відносини “один–до–багатьох”. При видаленні компанії всі пов’язані з нею рекрутери та вакансії видаляються автоматично (CASCADE), що гарантує чистоту даних.

Таблиця **Рекрутер** містить персональні дані користувача, який працює від імені компанії, та посилання на материнську таблицю **Компанія**. Кожен рекрутер може створювати кілька записів у таблиці **Вакансія**, що дає змогу моделювати різні відкриті позиції. За необхідності рекрутер також може бути деактивований, і цю інформацію відображає булеве поле is_activated.

Сутність **Університет** зберігає дані про навчальні заклади, а через зовнішній ключ у таблиці **Студент** указується, в якому університеті навчається користувач. При

цьому, якщо заклад видаляється, у студента просто очищається значення university_id (SET NULL), що запобігає втраті профілів випускників.

Таблиця **Вакансія** описує самі позиції: від назви і опису до діапазону зарплати, валюти, типу роботи та налаштувань активації. Для кожної вакансії зберігається інформація про компанію та рекрутера, який її опублікував. Універсальні обмеження CHECK, зокрема на валюту та діапазон зарплат, забезпечують коректність вводимих даних.

Сутність **Заявка** моделює відгук студента на вакансію. Вона поєднує зовнішні ключі vacancy_id і student_id та містить статус (pending, accepted, rejected). Унікальне комбіноване обмеження гарантує, що один студент не зможе двічі відгукнутися на одну вакансію. Завдяки такій структурі реалізовано повний життєвий цикл рекрутингової взаємодії — від публікації позиції до прийому або відхилення кандидата.

3 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Огляд технологій

Під час вибору технологій для клієнтської частини було звернуто увагу на поєднання продуктивності, зручності розробки та багатої екосистеми. React у поєднанні з TypeScript став очевидним вибором завдяки своїй здатності надавати строгі типи й ранню перевірку помилок ще до запуску коду. Це допомогло знизити ризик регресій під час рефакторингу та полегшило розробку завдяки автодоповненню та документації, яка виходить “з коробки”. Альтернативні бібліотеки, як Vue чи Angular, також мають свої переваги, проте React виграв конкуренцію завдяки простоті концепції компонентів і великій кількості готових до використання рішень.

Для оформлення інтерфейсу було обран Material-UI замість кастомних стилізацій чи інших фреймворків UI, оскільки він пропонує добре продумані компоненти та адаптивні теми, які дозволяють зосередитися саме на логіці, а не на дрібних деталях CSS. HTML і CSS застосовуються для тонкого налаштування стилів там, де потрібна унікальна візуальна обробка. React-Router був обраний через свій декларативний підхід до маршрутизації – він надзвичайно простий в інтеграції й дозволяє легко організувати динамічні шляхи та охорону маршрутів.

Щодо управління станом та взаємодії з сервером, Redux Toolkit і RTK Query забезпечують автоматизацію рутинних операцій: зменшують шаблонний код у порівнянні з класичним Redux, автоматично генерують action creators та slice reducers, а RTK Query додає потужний кеш і механізми повторних запитів замість того, щоб писати власний fetch-логіку. Axios обраний як HTTP-клієнт через його простий API та розширюваність (інтерсептори, таймаути), хоча можна було використати вбудований fetch, але у випадку складних запитів Axios забезпечує більше контролю.

На сервері вирішальним фактором став Express із TypeScript: цей мінімалістичний фреймворк дозволяє легко налаштувати middleware, чітко описати

контракти API та швидко розгорнути сервіс без зайвих абстракцій. Хоча існують більш структуровані рішення по типу NestJS, Express дав більшу свободу вибору архітектурних рішень без “вбудованих” підходів. Для зберігання даних було обрано на PostgreSQL із прямими SQL-запитами: вона гарантує цілісність транзакцій та високу продуктивність складних запитів, порівняно із NoSQL-базами, де складніше реалізувати суворі зв’язки між таблицями. З точки зору безпеки, JWT забезпечують масштабовану модель безсерверної автентифікації замість класичних сесій у базі даних, що зменшує навантаження на сервер при зберіганні сесійних даних.

3.2 **Загальний огляд компонентів**

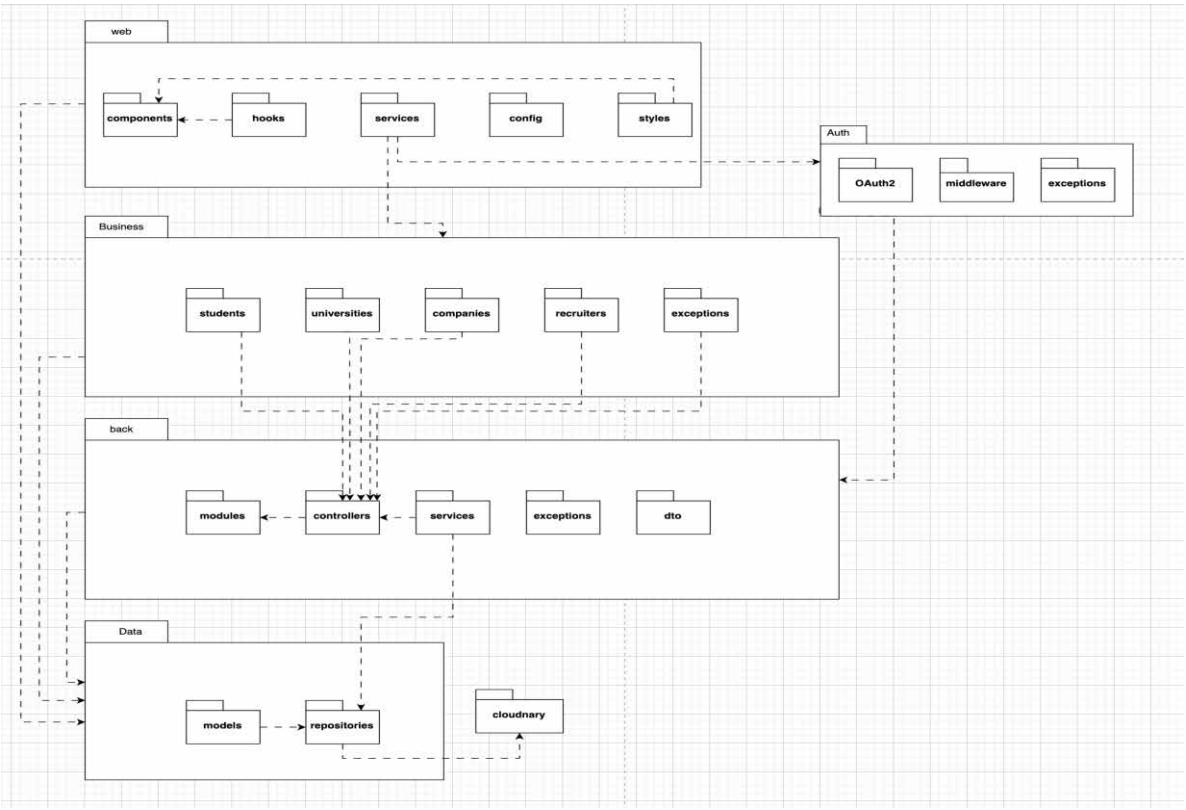


Рис. 9 Діаграма пакетів

3.2.1 Клієнтська частина. Клієнтська частина організована у пакеті **Web** як центральну точку взаємодії користувача з платформою. Саме тут реалізовано відображення списків вакансій, профілів студентів, форм реєстрації та будь-яких інших елементів інтерфейсу — кнопок, модальних вікон, сповіщень. Взаємодія з

екранами й обробка кліків відбуваються безпосередньо в браузері, де додано валідацію вводу та підготовку запитів до серверу.

Папка **components** наповнена усіма повторно використовуваними блоками: картками вакансій, полями вводу, елементами навігації та іншими UI-модулями. Це дозволило мені будувати сторінки швидко й уніфіковано, дотримуючись єдиного стилю. У директорії **hooks** розміщено власні React-хуки, які інкапсулюють логіку роботи з формами, обробку таймаутів у запитах та підписку на оновлення даних — усе, що не стосується безпосередньо розмітки.

У папці **services** описані функції для відправки HTTP-запитів за допомогою Axios: отримання списку вакансій, відправка відгуків, завантаження резюме та запит аналітики для університетів. Для кожного API-ендпоінта реалізовано окремий модуль, де реалізовано обробку типових помилок (наприклад, повторні спроби при збої мережі або коректне повідомлення про помилки 4xx/5xx).

Каталог **config** містить змінні середовища (.env), налаштування базового URL для API, правила CORS і константи для ключів локального сховища. Налаштовано його так, щоб у різних середовищах (розробка, тест, продакшен) весь код бізнес-логіки залишався незмінним — достатньо змінити один файл конфігурації.

У директорії **styles** зібрані глобальні CSS-файли та конфігурація Tailwind, де визначено кольорову палітру, типографіку й правила адаптивності. Це гарантує, що інтерфейс виглядатиме гармонійно на будь-якому пристрої — від великого монітора до екрану смартфона.

Щоб узгодити всі частини інтерфейсу, використовується **Redux store** як «source of truth»: центр зберігання стану дозволяє мені відстежувати зміну даних, виконувати відкат і синхронізувати інформацію між компонентами. Перехід між різними розділами реалізовано через **react-router**, що дає змогу плавно змінювати URL без перезавантаження сторінки та підтримувати прямі посилання на будь-яку вкладку. Це робить користувацький досвід швидким і передбачуваним.

3.1.2 Авторизація. У пакеті **Auth** зосереджено усі компоненти, пов’язані з безпекою та контролем доступу, щоб відокремити цю відповідальність від решти бізнес-логіки. Імплементовано перевірку JWT-токенів у middleware: кожен запит

проходить через ланцюжок перевірок, де аналізується його підпис і строк дії, а також витягаю з токена інформацію про користувача — його роль і привілеї. У разі виявлення недійсного чи простроченого токена користувач одразу отримує зрозуміле повідомлення про помилку авторизації, що мінімізує ризики несанкціонованого доступу.

Окрім стандартної роботи з токенами, інтегровано в **Auth** підтримку протоколу OAuth2, щоб мати змогу авторизовувати користувачів через зовнішні сервіси (наприклад, Google чи LinkedIn). Це рішення дало змогу клієнтам платформи обирати найзручніший спосіб входу та водночас забезпечило високий рівень захисту завдяки перевіреним механізмам сторонніх провайдерів. Всі налаштування клієнтських і секретних ключів винесені у конфігураційні файли, що дозволяє легко додавати нові провайдери без переробки коду.

Щоб зробити обробку запитів максимально прозорою та зручною, модуль **Auth** було створено, як окремий «проксі»-пакет, який перехоплює HTTP-запити ще до того, як вони досягнуть шару бізнес-логіки. У middleware-компонентах розміщені усі перевірки прав доступу, обробку помилок і редиректи, а також логування спроб доступу для аудиту. Такий підхід дозволив мені централізувати безпекові правила в одному місці й гарантував, що жоден запит, що не пройшов валідацію, не потрапить далі до контролерів чи сервісів. Це значно спростило підтримку коду та дало впевненість у захищеності системи.

3.1.3 Business (Бізнес-логіка). У пакеті **Business** зібрано усі ключові правила предметної області так, щоб вони діяли однаково і на клієнті, і на сервері. По-перше, тут реалізована перевірка ролей: коли надходить запит від користувача, система звертається до модуля **role-check**, де визначається, чи дійсно ця людина має статус студента, рекрутера або представника університету. Це дає змогу тонко налаштувати права: наприклад, блокувати рекрутера від видалення чужих вакансій або перешкоджати студенту подавати більше ніж одну заявку на одну вакансію.

По-друге, у **Business** закладена валідація всіх бізнес-процесів. Туди винесено логіку, що перевіряє послідовність дій: чи існує у студента завантажене резюме перед подачею заявки, чи не минув термін публікації вакансії, чи правильний формат даних

в полях контракту тощо. Наприклад, модуль **application-rules** контролює, щоб студент не зміг створити дві заявки на ту саму позицію, а сервіс **vacancy-rules** — щоб рекрутер не опублікував вакансію з від'ємною зарплатою чи без вказаного місця розташування.

Третій важливий аспект — **Business** слугує єдиним джерелом правди для фронтенду й бекенду. Він був реалізований окремою бібліотекою, яку підключають обидві частини проєкту. Це гарантує, що навіть якщо інтерфейс застосовує ту саму валідацію форм, що і сервер, користувач отримає одноманітний досвід, а ймовірність розбіжностей у правилах знижується до нуля. Такий підхід економить час на тестування та підтримку, адже зміну в одному місці — наприклад, додавання нової ролі чи зміна умов відбору кадрів — достатньо внести лише в **Business**, і вона одразу почне діяти скрізь.

3.1.4 Back (Серверна частина). У серверній частині, яка реалізована у пакеті **Back**, логіка така, щоб кожен HTTP-запит проходив чітко визначений життєвий цикл. Спочатку запит потрапляє до одного з **контролерів** (controllers), де проводиться попередня валідація вхідних даних згідно зі схемами, описаними в **DTO** (dto). Тільки після успішної перевірки контролер передає дані далі — у **сервіси** (services), де відбувається основна обробка: виклик бізнес-логіки з пакету Business, взаємодія з репозиторіями пакету Data та побудова відповіді.

Щоб гарантувати стійкість і передбачуваність, усі помилки централізовано обробляються в модулі **exceptions**. Там описані класи користувацьких винятків для технічних, авторизаційних та бізнес-помилок, і кожен із них перетворюється на відповідний HTTP-статус із зрозумілим повідомленням. Завдяки цьому зменшено дублювання коду обробки помилок у контролерах і сервісах, а також забезпечую єдине місце для коригування текстів чи логіки відповіді у разі виключних ситуацій.

Окрім цього, було об'єднано всі контролери, сервіси, DTO та обробники помилок у **модулі** (modules), щоб легко масштабувати функціонал: достатньо додати новий модуль із власними контролерами і сервісами, і система автоматично підхопить їх під час завантаження. Така структура робить бекенд прозорим і зрозумілим для

будь-якого нового учасника команди, оскільки чітко показує, де починається вхідний запит, де відбувається логіка й куди повертається результат.

3.1.5 Data (Сховище даних). У цьому пакеті винесено усю роботу з базою та зовнішніми сервісами, щоб відокремити деталі зберігання інформації від бізнес-логіки. По-перше, у папці **models** описана структуру кожної сутності — студентів, компаній, вакансій, заявок тощо — у вигляді SQL-схем, що відповідають таблицям PostgreSQL. Тут зазначено типи полів, обов’язкові обмеження й зв’язки (наприклад, один-до-багатьох між вакансіями та відгуками), що гарантує цілісність даних на рівні бази.

Далі, у директорії **repositories**, реалізовані класи, які інкапсулюють усі CRUD-операції над моделями. Кожен репозиторій містить методи find, create, update і delete із чітким інтерфейсом, аби не тягнути SQL-запити в сервіси або контролери. Всередині репозиторіїв використано SQL-запити, через query builder, що дає змогу оптимізувати складні вибірки без втрати зручності.

Окремою важливою частиною є інтеграція з **Cloudinary** для зберігання резюме або інших файлів. У підпакеті **cloudinary** налаштовано клієнтський SDK, описано методи для завантаження, оновлення та видалення медіафайлів, а також механізми обробки помилок від стороннього сервісу. Таким чином, у коді бекенду можна просто викликати cloudinary.upload(file) замість того, щоб думати про формування multipart-запиту чи ручну передачу токенів.

Крім того, у **Data** імплементовано підтримку транзакцій для критичних груп операцій (наприклад, створення вакансії разом із початковими статистичними записами), а також налаштував систему міграцій. Це дозволяє оновлювати структуру бази даних без втрати даних і завжди мати можливість відкотитися до попередньої версії. Така організація коду робить роботу з даними прозорою, надійною та легко підтримуваною.

3.2 Алгоритми взаємодії та потоки даних

У цьому підрозділі описано низку характерних сценаріїв, які ілюструють, як запити рухаються крізь усі шари системи — від клієнта до бази даних і назад.

Одним із найчастіших патернів є стандартний запит на отримання списку вакансій. Коли користувач відкриває сторінку «Вакансії», фронтенд через свій vacanciesService ініціює HTTP GET-запит до ендпоінта /api/vacancies. Цей запит спочатку перехоплює шар **Auth**, який підписує його даними про поточного користувача, а потім передає в контролер VacanciesController.list(). Контролер звертається до VacanciesService.getAll(), який у свою чергу викликає бізнес-модуль **Business** для фільтрації та сортування згідно з правами та уподобаннями користувача. Репозиторій з пакета **Data** формує оптимізований SQL-запит, повертає масив вакантних позицій, а контролер надсилає його назад клієнту. RTK Query на клієнті отримує відповідь та оновлює кеш, а компоненти інтерфейсу, підписані на цей slice, миттєво візуалізують нові дані.

Інший типовий потік пов’язаний із оновленням профілю користувача. Коли студент або рекрутер заходить у налаштування акаунта й змінює свій телефон чи фото, фронтенд формує PATCH-запит через userService.updateProfile(). Запит знову проходить авторизацію в **Auth**, потім досягає UsersController.update(), де UsersService.updateUser() звертається до **Business**-модуля для валідації нових даних (наприклад, перевірити формат номера телефону чи розмір файлу аватару). Після перевірки відбувається запис у базу через UserRepository, і сервіс повертає оновлену модель користувача. Фронтенд отримує нові дані й синхронізує їх із Redux store, щоб усі компоненти відобразили зміни без перезавантаження.

Завантаження резюме — ще один ключовий сценарій. Користувач вибирає файл на клієнті, і через resumesService.upload() формується мультипарт-запит до /api/resumes/upload. Цей запит уперше маршрутизують у **Auth**, потім у ResumesController.upload(), де перед записом у Cloudinary перевіряють тип і розмір файлу. Після успішного завантаження сервіс бекенду записує URL файлу в базу даних за допомогою ResumeRepository.save(). Повернутий URL надсилається клієнту, де оновлюється відповідний slice RTK Query, і компонент резюме автоматично відображає новий документ.

Не менш важливий патерн — процес аутентифікації й відновлення сесії. При вході через AuthService.login() дані форми передаються на /api/auth/login. Після перевірки логіна та пароля сервер генерує пару JWT (access та refresh) і повертає їх клієнту. Axios-інтерсептори з мого коду автоматично зберігають токени в пам’яті та локальному сховищі. У разі закінчення терміну дії access-токена інтерсептор відправляє refresh-запит на /api/auth/refresh, отримує новий токен і повторює початковий запит, який виконався в бекграунді без участі користувача.

Нарешті, обробка помилок і виняткових ситуацій сама по собі є окремим патерном. Усі HTTP-помилки (4xx, 5xx) ловляться в Axios-інтерсепторах, де їх розділено на технічні (сервер недоступний, таймаут) та бізнес-помилки (недостатньо прав, дублікати заявок тощо). Перші відправляють глобальний сповіщувач Toast із пропозицією повторити дію, а другі проксовано повертаються з чітким повідомленням до контролера помилок у фронтенді, де формується human-friendly текст. Цей потік гарантує, що незалежно від рівня, на якому трапилася помилка, користувач отримає зрозумілу відповідь і інтерфейс збереже стабільність.

3.3 **Фізична архітектура системи**

Для демонстрації фізичної архітектуру системи, було створено діаграму розгортання (рис.10), щоб показати, як різні компоненти взаємодіють між собою на рівні серверів, контейнерів та зовнішніх сервісів. На цій схемі кожен вузол відповідає окремому середовищу виконання або інфраструктурному елементу, а стрілки позначають типи мережевих з’єднань і протоколів, якими обмінюються дані.

Першим у ланцюжку варто **Клієнтський пристрій**: десктоп або мобільний браузер користувача, де запущений SPA-застосунок на React. Саме звідси надходять HTTPS-запити до зовнішнього **Load Balancer** або проксі-сервера (наприклад, Nginx), який розподіляє трафік між декількома інстансами фронтенд-і бекенд-контейнерів.

Далі йдуть **Контейнери з клієнтським додатком** – це Docker-контейнери, що віддають статичні файли HTML, JavaScript і CSS. Вони налаштовані на кешування

через HTTP заголовки і працюють за допомогою Nginx як веб-сервера для максимально швидкої доставки ресурсів. Ця частина відповідає за відображення інтерфейсу та налагоджує захищене з'єднання з API через HTTPS.

Наступним вузлом є **API-сервер** (Back), упакований також у Docker-контейнер з Node.js і Express. Він слухає порт 3000 та приймає запити від фронтенду. Передусім запити проходять через **Auth Middleware**, а потім потрапляють у контролери та сервіси. Docker-контейнер API-сервера під'єднаний до внутрішньої Docker-мережі, де розташована база, щоб мінімізувати затримки та ізолювати трафік від зовнішніх загроз.

Поряд із API-сервером на діаграмі розгортання представлений **Сервер бази даних PostgreSQL**. Це окремий контейнер або в managed-сервісі, підключений тільки до API-сервера за допомогою TCP-порту 5432. У схемі видно, що доступ до БД має обмежуватися лише репозиторіями бекенду, а всі операції відбуваються через надійний TLS-канал для шифрування даних у транзиті.

Окремою гілкою йде засіб **Cloudinary** – зовнішній хмарний сервіс для зберігання та обробки медіафайлів. API-сервер через спеціальні клієнтські бібліотеки ініціює HTTPS-запити до Cloudinary для завантаження резюме та зображень. У схемі це позначено пунктирною лінією, оскільки Cloudinary не входить у наш Docker-кластер, але є критичною частиною розгортання.

Завершують діаграму **Сервіси моніторингу й логування** (наприклад, Prometheus, Grafana, ELK-stack), які під'єднані до API-сервера й бази для збору метрик продуктивності та логів. Вони дозволяють у режимі реального часу відстежувати стан контейнерів, швидкість обробки запитів і можливі помилки в роботі.

Загалом, діаграма розгортання дає змогу чітко уявити собі фізичне розташування компонентів, канали зв'язку між ними та обмеження доступу. Така наочна модель допомагає планувати масштабування, забезпечувати безпеку на кожному рівні та швидко орієнтуватися в інфраструктурі під час підтримки та оновлень.

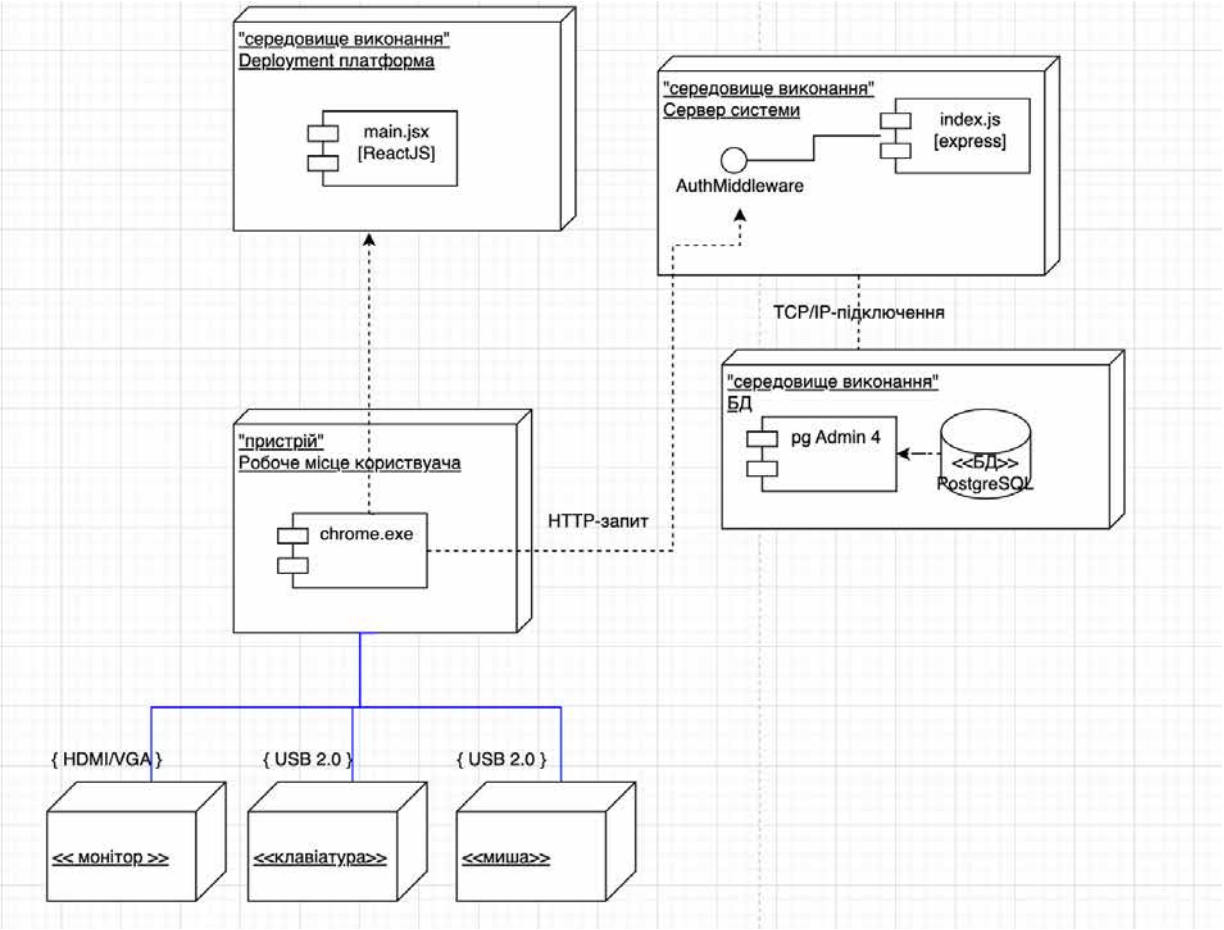


Рис. 10 Діаграма розгортання

3.4 Інтеграція із зовнішніми сервісами

Передбачено кілька зовнішніх сервісів, які забезпечують зберігання медіафайлів та підтвердження легітимності організацій. Завдяки цим інтеграціям значно розширюється функціональність системи, а також автоматизуються перевірки даних і оптимізується робота серверної частини.

Для обробки та зберігання зображень (логотипів компаній, фото студентів, скани документів тощо) використовується Cloudinary. Користувацькі файли відразу надсилаються з клієнта на бекенд у вигляді FormData, де відбувається їх оптимізація в сучасний формат WebP. Отримані посилання на картинки повертаються назад і відображаються в інтерфейсі, а завдяки CDN знімальне навантаження з основного сервера суттєво знижується.

Cloudinary також дає змогу управляти вже завантаженими файлами: замінювати їх або видаляти при оновленні профілів чи видаленні об’єктів у системі. Крім того, за допомогою вбудованих трансформаційних налаштувань можна гнучко задавати розміри, якість та формат зображень, що позитивно впливає на швидкість їх завантаження та відображення на стороні клієнта.

Для перевірки юридичних осіб (компаній та університетів) інтегровано API ЄДРПОУ — державного реєстру підприємств і організацій України. Після заповнення реєстраційної форми сервер робить запит до відповідного REST-інтерфейсу за кодом ЄДРПОУ, отримуючи офіційну назву, статус, адресу та перелік видів діяльності (КВЕД). Ця інформація автоматично підставляється в профіль організації та стає підставою для присвоєння їй статусу «перевірена компанія», що підвищує довіру користувачів.

Для інформування користувачів та інші сповіщення використовується обліковий запис Gmail через SMTP. Платформа надсилає листи зі власної адреси, формуючи теми та тіло повідомлень на основі шаблонів. Завдяки цьому всі повідомлення доходять до одержувачів швидко та надійно, а адміністратори можуть відстежувати статус доставки та обробку помилок через логи поштового сервера.

4 ЕКСПЛУАТАЦІЯ ТА ВПРОВАДЖЕННЯ СИСТЕМИ

4.1 Підготовка середовища виконання

Перед тим як приступити до розгортання та тестування інформаційної системи рекрутингу студентів, потрібно впевнитися, що локальна машина або сервер відповідають мінімальним системним вимогам. Ключовими компонентами, необхідними для коректної роботи клієнтської й серверної частин, є середовище виконання Node.js, реляційна база даних PostgreSQL та інструменти контейнеризації Docker / Docker Compose.

По-перше, для збірки фронтенд-бандлу та запуску серверної частини було обрано Node.js мінімум версії 18.x. Такий вибір гарантує сумісність із актуальними пакетами, зазначеними у файлі package.json клієнтської частини.

Зокрема, у списку залежностей присутні сучасні інструменти: Vite (версія ^5.3.4), React 18 (версія ^18.3.1) та TypeScript 5 (версія ^5.2.2). Враховуючи, що ці пакети розроблені з урахуванням останніх можливостей ECMAScript, використання Node.js 18.x дозволяє безперешкодно працювати з ESM-бандлерами і скористатися новими фічами мови.

Крім того, наявність LTS-релізу підвищує стабільність і безпеку проєкту. Node.js 18.x підтримує сучасні підходи до роботи з модулями та забезпечує автоматичні оновлення безпеки, що особливо важливо при використанні зовнішніх бібліотек і фреймворків.

Для серверної частини також застосовується Node.js 18.x, аби уникнути можливих конфліктів між різними версіями залежностей. Це стосується таких пакетів як dotenv (версія ^16.4.x), express (версія ^4.18.x) та ORM/драйверів на кшталт sequelize або pg. Використання однакової версії середовища дозволяє знизити ризик несумісності та спрощує керування спільними модулями.

У проекті для роботи зі структурованими даними обрано PostgreSQL. Для налаштування локального середовища розробник може встановити PostgreSQL через офіційний інсталятор або скористатися Docker-контейнером, а параметри підключення (ім’я бази, користувач, пароль) прописати у файлі .env.development. Це дозволяє швидко отримати робочу СУБД із необхідними параметрами без додаткових складнощів.

По-друге, щоб уніфікувати локальне середовище для всіх учасників проєкту та забезпечити відтворюваність, застосовано Docker і Docker Compose. У корені репозиторію розміщено файл docker-compose.yml, який описує конфігурацію контейнерів для фронтенду (Nginx із проксі для React-бандлу), бекенду (Node.js/Express), бази даних PostgreSQL, кешу та черг (Redis), а також, за потреби, допоміжних сервісів, наприклад, Mailhog для тестування email-розсилок. Після виконання команди docker-compose up --build усі сервіси автоматично піднімаються, міжконтейнерна мережа налаштовується, а порти проєкту (API на 3000, СУБД на 5432, фронтенд на 80/3001) мапляться відповідно до прописаних у конфігурації. Завдяки такому підходу кожен учасник команди може за лічені хвилини отримати повноцінне робоче середовище.

Для належного функціонування клієнтської та серверної частин системи необхідно створити файл із змінними середовища (наприклад, .env, чи розподілити на кілька файлів — .env.development, .env.staging, .env.production). У цьому файлі збираються всі ключі й значення, без яких додаток не зможе працювати правильно.

По-перше, варто вказати тип середовища через змінну NODE_ENV (можливі варіанти — development або production). Це дозволяє підлаштовувати логіку програми під режим розробки чи продакшену. Далі необхідно задати порт, на якому Express-сервер очікує запити, — змінна PORT (наприклад, 3000).

По-друге, для підключення до PostgreSQL слід задати п’ять параметрів: DB_HOST, DB_PORT, DB_USER, DB_PASSWORD та DB_NAME. Саме на їхній основі утворюється рядок з’єднання, що використовується ORM чи драйвером для взаємодії з базою даних.

Також критичною є змінна JWT_SECRET, яка використовується для підпису JSON Web Token. Вона повинна бути довгою щонайменше 32 символи, аби забезпечити достатній рівень безпеки при генерації токенів для автентифікації користувачів.

Окрім того, якщо застосунок інтегрується з Cloudinary для роботи із зображеннями, необхідно визначити три налаштування: CLOUDINARY_CLOUD_NAME, CLOUDINARY_API_KEY та CLOUDINARY_API_SECRET. Це дозволяє серверу виконувати запити до API Cloudinary, відправляти файли, отримувати URL-адреси завантажених медіа та керувати ними.

Для перевірки юридичних осіб за кодом ЄДРПОУ достатньо вказати базовий адресний рядок — змінну EGRPOU_API_URL. Саме на її значення спираються запити до державного реєстру, аби підтвердити справжність компаній чи університетів.

Нарешті, якщо система використовує SMTP для надсилання листів (через SendGrid, Gmail чи інший поштовий сервер), у файлі .env мають опинитися EMAIL_HOST, EMAIL_PORT, EMAIL_USER та EMAIL_PASS. Ці змінні визначають конфігурацію поштового транспорту, що дає змогу відправляти повідомлення користувачам (активації акаунтів, скидання паролів тощо).

4.2 **Розгортання системи**

Для початку необхідно скопіювати репозиторій до локальної машини та встановити всі залежності. Після клонування проєкту у вибрану директорію, слід перейти до папки з клієнтською частиною (зазвичай назва папки – web) і запустити команду для інсталяції залежностей. Аналогічні дії потрібно виконати у кореневій папці серверної частини (папка back), щоб інсталювати пакети, необхідні для роботи бекенду. Таким чином, обидва середовища – клієнтське та серверне – отримають потрібні бібліотеки та фреймворки для своєї роботи.

Наступним кроком є налаштування файлів із змінними середовища. У кожному з підпакетів (web та back) зазвичай міститься шаблон .env.example. Його слід скопіювати у файл .env і заповнити реальними значеннями – наприклад, вказати в порожні поля власні ключі для підключення до баз даних, сертифікати або адреси зовнішніх сервісів. Це гарантує, що при запуску програма матиме доступ до всіх необхідних налаштувань, не виключаючи секретних ключів і конфіденційних параметрів.

Для швидкої розробки та одночасного запуску клієнта і сервера існує простий спосіб із застосуванням npm-скриптів. Потрібно відкрити два окремих термінали: в одному перейти до папки клієнта і виконати команду для старту режиму розробки (дозволяє миттєво бачити зміни у фронтенді), в іншому – перейти до папки сервера і також запустити цей режим (зазвичай використовують команду, яка перезапускає сервер при кожному зміненні коду, наприклад через nodemon). Завдяки такій організації можна відразу перевіряти, як зміни у коді клієнта впливають на загальний процес і реагувати на помилки в режимі реального часу.

Окрім локального запуску через npm, для одноманітного середовища розробки зручно використовувати Docker Compose. У корені репозиторію розташований файл docker-compose.yml, що описує конфігурацію всіх необхідних контейнерів: фронтенд-клієнта з проксі-сервером, бекенд-додатку, бази даних, кешу та інших допоміжних сервісів. Достатньо виконати одну команду, яка автоматично побудує образи, підніме контейнери та об’єднає їх у спільну мережу. Після цього фронтенд буде доступний за заздалегідь зазначеним портом (наприклад, 3001), сервер API працюватиме на своєму порту (наприклад, 3000), а якщо передбачено – адміністративний інтерфейс для бази даних (наприклад, pgAdmin) відкриватиметься окремо, як вказано в конфігурації.

Щоб переконатися, що розгортання відбулося успішно, достатньо відкрити у браузері адресу фронтенд-інтерфейсу, створити тестовий акаунт студента чи рекрутера і пройти всі критичні сценарії: від реєстрації до публікації вакансії, подачі заявки та формування звіту. Такий підхід дозволяє переконатися, що всі сервіси

взаємодіють коректно і дані коректно передаються між клієнтом, сервером та іншими компонентами системи.

Завдяки чіткій послідовності налаштувань: починаючи від встановлення залежностей, налаштування файлів середовища і закінчуючи способами запуску через npm або Docker Compose, розробники швидко можуть розгорнути систему в локальному середовищі, а тестувальники та інші учасники процесу без зайвих зусиль відтворити ту ж конфігурацію для перевірки та демонстрації.

4.3 **Адміністрування системи**

У системі передбачені два рівні адміністративного доступу — роль **admin** та роль **superadmin**, які відрізняються за масштабом повноважень і відповідальністю.

Роль **admin** зосереджена насамперед на виконанні повсякденних операційних завдань. Адміністратор цієї категорії відповідає за перевірку нових заявок на реєстрацію компаній і університетів, підтвердження достовірності наданих ними даних, а також контролює, щоб до публікації вакансій допускалися лише ті організації, які пройшли валідацію.

У свою чергу роль **superadmin** має розширені права, що виходять за межі функціоналу звичайного адміністратора. Superadmin, крім здійснення усіх дій, доступних admin, може змінювати рівні доступу інших адміністраторів, коригувати налаштування системи в цілому та переглядати аудиторські журнали, аби забезпечувати безпеку й цілісність даних.

Таким чином, поділ на два рівні адміністрування дозволяє делегувати стандартні операційні процеси ролі admin, водночас залишаючи за superadmin можливість здійснювати глобальний контроль над системою і керувати правами інших користувачів.

Процедура верифікації організацій у системі побудована як кілька послідовних етапів, які поєднують автоматичні та ручні перевірки. Спершу платформа надсилає запит до державного реєстру ЄДРПОУ, щоб отримати базову інформацію про

компанію або університет. Завдяки цьому вже на початковому етапі система фільтрує очевидні невідповідності або відсутні записи.

Після отримання даних від ЄДРПОУ відбувається ручна оцінка, під час якої адміністратор порівнює інформацію з реєстру з документами, завантаженими користувачем. Якщо виникають сумніви щодо точності даних чи термінів реєстрації, адміністратор може звернутися безпосередньо до представника організації, щоб уточнити деталі або попросити додаткові свідчення.

Такий двоетапний підхід—поєднання автоматичного запиту й вручну проведеної верифікації—дозволяє суттєво зменшити кількість фейкових або неповноцінно зареєстрованих компаній на платформі. Завдяки цьому студенти та університети можуть бути впевнені, що взаємодіють лише з тих організаціями, які мають офіційний статус і підтверджені державними реєстрами.

У випадку, коли кількість заявок значно зростає і з’являється потреба опрацьовувати велику кількість даних одночасно, можлива інтеграція механізму автоімпорту з директорії або імпорт CSV-файлів. Проте навіть за використання такого способу завантаження інформації ключові записи лишаються під контролем: остаточне підтвердження здійснює superadmin, який має остаточне рішення про статус організації.

4.4 Демонстрація роботи програми

Перший екран, з яким стикається користувач, — це **лендінгова сторінка** (рис. 11) платформи. Вона виконана в мінімалістичному стилі з приглушеними кольорами та чіткою типографією, щоб не відволікати увагу від головних закликів до дії. У верхній частині розташовано логотип і навігаційне меню: «Про нас», «Функціонал», «Підтримка», а з правого краю — дві яскраві кнопки «Зареєструватися» і «Увійти». Центральна частина лендінгу заповнена коротким слоганом «Швидкий рекрутинг для студентів та компаній» і ілюстрацією, що символізує взаємодію між студентом і

роботодавцем. Нижче наведено три ключові переваги платформи — перевірені компанії, простий інтерфейс, миттєві сповіщення — у вигляді іконок із підписами.

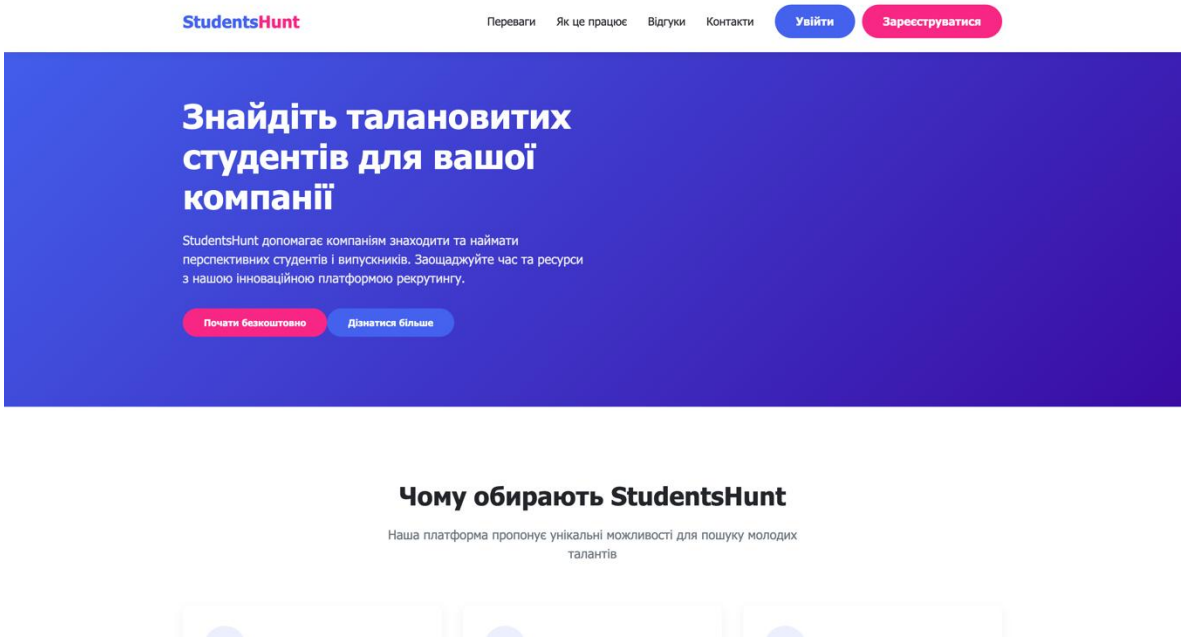


Рис. 11 Лендінг сторінка сайту

Користувач на цьому етапі може обрати одну з двох опцій: створити новий акаунт або авторизуватися в уже наявному. Натискання кнопки «Зареєструватися» миттєво перенаправляє на сторінку реєстрації, де потрібно ввести корпоративну пошту університету або коду ЄДРПОУ для компаній, а також обрати роль (студент, рекрутер чи університет). Якщо ж користувач вже зареєстрований, він може натиснути «Увійти» — і через декілька мілісекунд опиниться на формі входу. Обидві кнопки ведуть на єдину сторінку аутентифікації, де за допомогою вкладок можна перемикатися між реєстрацією та авторизацією без перезавантаження сторінки.

Сторінка аутентифікації розроблена таким чином, щоб максимально прискорити доступ до функціоналу. У верхній частині — підзаголовок «Вхід в систему» або «Реєстрація нового акаунту», а нижче — велика форма з полями для електронної пошти, пароля та підтвердження пароля (для реєстрації), а також чекбоксом для згоди з умовами користування. Після заповнення форми й натискання кнопки «Підтвердити» відбувається валідація на стороні клієнта: перевіряється коректність формату email, мінімальна довжина пароля та збіг полів. Такий підхід забезпечує швидку та інтуїтивну навігацію від лендінгу до основного функціоналу системи.

4.4.1 Інтерфейс та функціонал компанії. Після успішної аутентифікації користувача (наприклад компанії) автоматично перенаправляється на власний **Dashboard** (рис. 12), який слугує центральною точкою контролю й моніторингу її рекрутингової активності. Верхня панель містить чотири ключові метрики: загальну кількість відкритих вакансій, сумарну кількість отриманих відгуків, число зареєстрованих рекрутерів та кількість заявок у статусі «очікують» (Pending). Таке стисле представлення дозволяє швидко оцінити поточний стан процесу та негайно звернути увагу на вузькі місця — наприклад, якщо багато заявок залишаються без обробки.

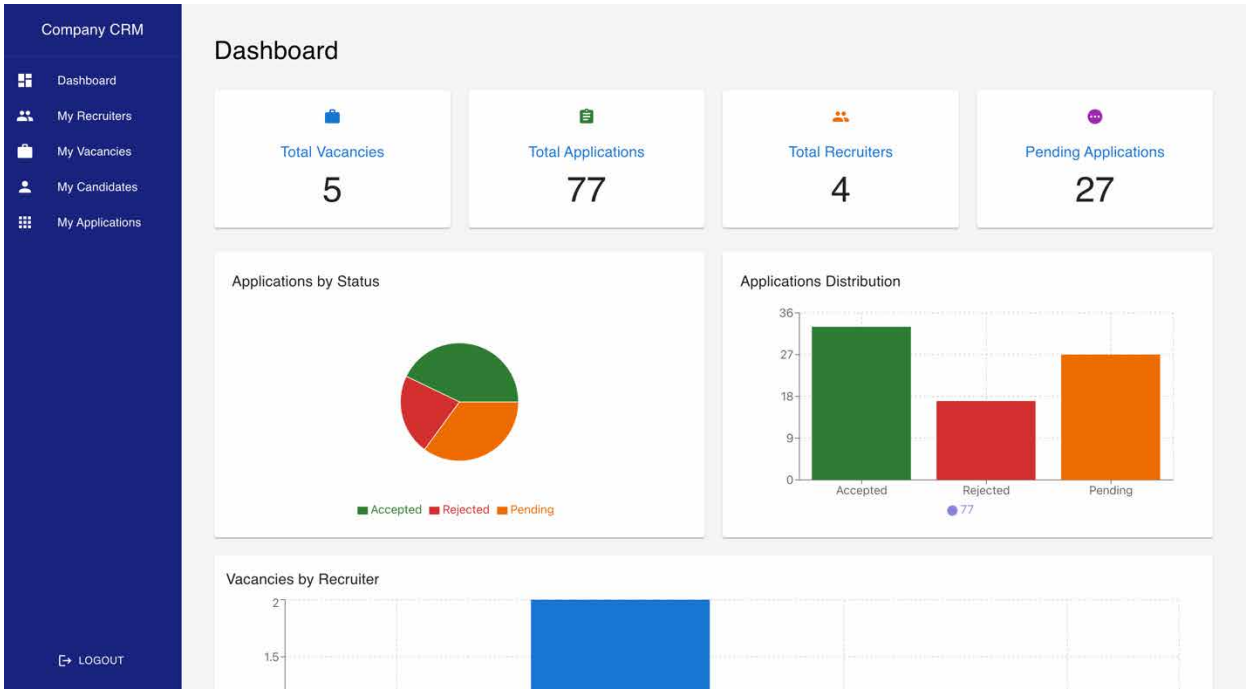


Рис. 12 «Dashboard» компанії

Нижче наведено дві візуалізації (див. рис. 12) для поглибленого аналізу: кругова діаграма “Applications by Status” показує співвідношення прийнятих, відхилених і тих, що очікують рішення, а сусідня стовпчаста діаграма “Applications Distribution” дає абсолютні значення цих категорій. Завдяки такому поєднанню графіків менеджер компанії може не лише відстежити загальні тренди, але й оцінити, з яким типом відгуків працює її команда найактивніше, та при необхідності скоригувати свої підходи до відбору.

Ще один блок відображає “Vacancies by Recruiter” — стовпчастий графік, у якому кожна колонка відповідає окремому рекрутеру й показує число вакансій, які

він створив чи веде. Це допомагає власнику бізнесу чи HR-директору побачити розподіл навантаження всередині команди та за потреби перенаправити ресурси чи надати допомогу тим рекрутерам, які мають найменші показники.

У лівій колонці знаходиться основне навігаційне меню, де окрім Dashboard доступні розділи «My Recruiters» (Додаток К), «My Vacancies» (Додаток Л), «My Candidates» (Додаток М) та «My Applications» (Додаток Н). Кожен із них відкриває окремий список із деталями: профілі рекрутерів із контактами та статусами, перелік вакансій із можливістю створення чи редагування, картки студентів-кандидатів та журнал усіх відгуків із фільтрами за датою, статусом і спеціалізацією.

Наприкінці панелі розміщено кнопку «LOGOUT», яка безпечно завершує сесію та повертає користувача на головну сторінку лендінгу. Таким чином, Dashboard компанії поєднує в собі загальний огляд діяльності та швидкий доступ до всіх важливих розділів, забезпечуючи керівникам і рекрутерам зручний, інформативний інтерфейс для щоденної роботи.

4.4.2 Інтерфейс та функціонал студента. Після успішного входу студент потрапляє на **Dashboard** (рис. 13) — персоналізовану панель керування, де зібрано найактуальнішу інформацію про його рекрутингову активність. Першим блоком ідуть **Нещодавні заявки**: тут відображаються останні вакансій, на які студент подав відгуки, з позначками статусів (наприклад, «Pending», «Accepted», «Rejected»). Одразу поруч розміщено **Нещодавні вакансії** — свіжі оголошення від компаній із назвою позиції, ім’ям роботодавця та містом розташування. Нижче розташувався розділ **Швидкі дії**, у якому великими кнопками винесено основні переходи: «Переглянути всі вакансії», «Список компаній», «Редагувати профіль», що дозволяє одним кліком перейти до потрібної функції.

Перехід у розділ **Vacancies (Вакансії)** відкриває студенту зручний каталог усіх доступних пропозицій. Угорі сторінки знаходяться поля пошуку та фільтри: рядок для введення ключових слів або назви посади, випадаючий список із вибором міста і радіокнопки для сортування за датою публікації (від нових до старих або навпаки). Кожна вакансія у списку представлена карткою з найважливішими даними: назва ролі, логотип і назва компанії, локація, короткий опис та вказівка зарплатного

діапазону. Для ознайомлення з деталями передбачена кнопка **View Details**, яка відкриває повний опис позиції, перелік вимог і умов, а також форму для подачі заявки.

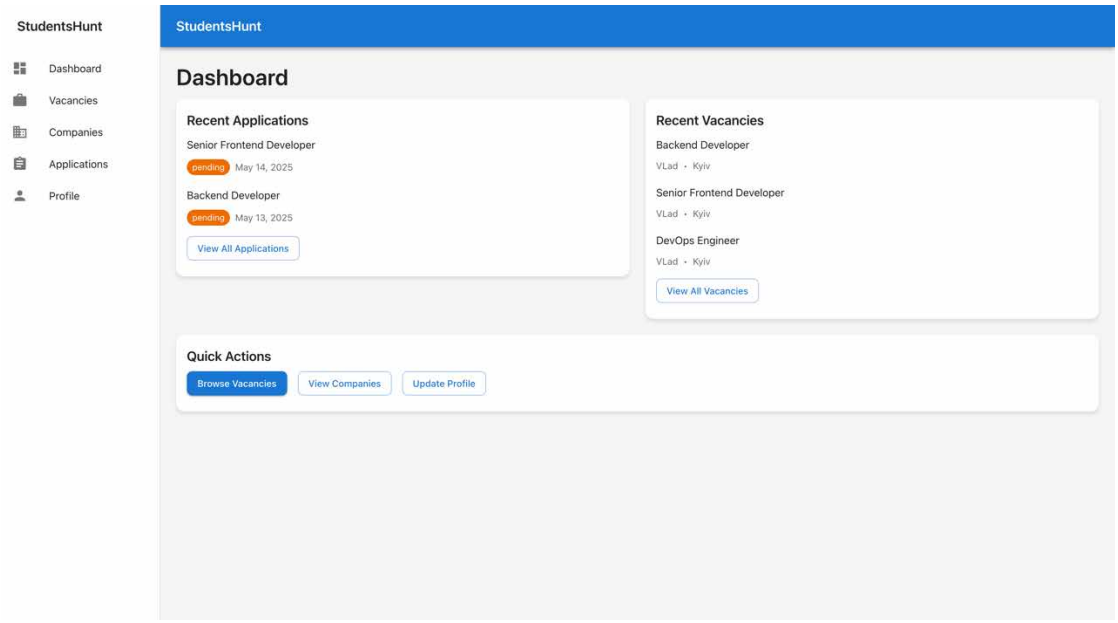


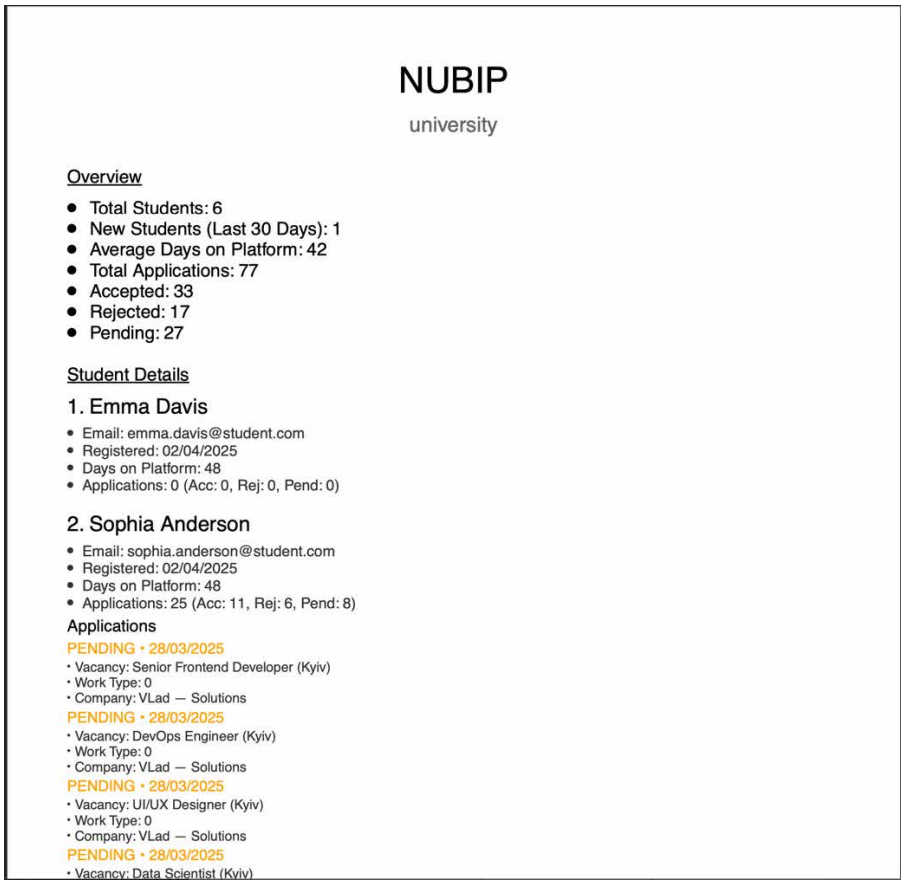
Рис. 13 “Dashboard” студента

У розділі **Applications (Заявки)** сконцентровано всі відгуки, надіслані студентом. Тут можна скористатися фільтрами за статусом (всі, «На розгляді», «Прийняті», «Відхилені») та впорядкувати записи за датою подачі. Кожен рядок у таблиці містить базову інформацію: назву вакансії, назву компанії, рівень зарплати, місто, тип зайнятості (наприклад, стажування чи повний день), а також дату, коли було відправлено заявку. При розгортанні деталізації студент бачить повний опис оголошення, контакт рекрутера, унікальні ідентифікатори заявки та вакансії, а також кнопки швидкого переходу до профілю компанії або сторінки самої вакансії.

Таким чином, студент має широкий набір інструментів для ефективного пошуку роботи: він може шукати та фільтрувати вакансії за різними критеріями, подавати заявки на обрані позиції, відстежувати їхній статус і завжди мати під рукою контакти рекрутерів. Окрім того, у своєму профілі він може оновлювати резюме, додавати навички та персональні дані, щоб підтримувати актуальність інформації та підвищувати шанси на успіх у конкуренції за вакансії.

4.4.3 Генерація звіту для університету. Після реєстрації або авторизації університет може отримати звіт про студентів (рис. 14), які зареєстровані на

платформі. Цей документ починається з блоку **Загальних показників**, де зібрані ключові метрики, що відображають активність усіх користувачів-студентів. Зокрема, вказується **загальна кількість активних студентів** (наприклад, 6), а також **число нових реєстрацій за останні 30 днів** (1), що допомагає оцінити темпи зростання аудиторії. Крім того, виводиться **середня кількість днів**, які студенти проводять у системі (42), — цей показник демонструє рівень залученості й регулярності використання платформи.



NUBIP

university

Overview

- Total Students: 6
- New Students (Last 30 Days): 1
- Average Days on Platform: 42
- Total Applications: 77
- Accepted: 33
- Rejected: 17
- Pending: 27

Student Details

1. Emma Davis

- Email: emma.davis@student.com
- Registered: 02/04/2025
- Days on Platform: 48
- Applications: 0 (Acc: 0, Rej: 0, Pend: 0)

2. Sophia Anderson

- Email: sophia.anderson@student.com
- Registered: 02/04/2025
- Days on Platform: 48
- Applications: 25 (Acc: 11, Rej: 6, Pend: 8)

Applications

PENDING • 28/03/2025
• Vacancy: Senior Frontend Developer (Kyiv)
• Work Type: 0
• Company: VLad — Solutions

PENDING • 28/03/2025
• Vacancy: DevOps Engineer (Kyiv)
• Work Type: 0
• Company: VLad — Solutions

PENDING • 28/03/2025
• Vacancy: UI/UX Designer (Kyiv)
• Work Type: 0
• Company: VLad — Solutions

PENDING • 28/03/2025
• Vacancy: Data Scientist (Kyiv)

Рис. 14 Згенерований звіт для університету

В рамках загального огляду подається **сукупна кількість заявок** (77) і їх розподіл за статусами: **Accepted** (33), **Rejected** (17) та **Pending** (27). Такі дані дають змогу побачити, скільки вакансій було відкрито студентами, який відсоток відгуків завершився працевлаштуванням чи стажуванням, а також який обсяг заявок ще знаходиться на розгляді. Для керівництва університету це важливо при плануванні кар’єрних заходів і відборі партнерів-роботодавців.

Друга частина звіту — **Деталізовані дані про кожного студента** — містить індивідуальні довідки. Для кожного зареєстрованого користувача виводяться **ім’я та**

прізвище, **електронна пошта**, **дата реєстрації** (наприклад, 02.04.2025) і **кількість днів**, проведених у системі (48). Поруч відображено **загальну кількість заявок** того чи того студента та їх розподіл за статусами (Acc: A, Rej: R, Pend: P), що допомагає відслідкувати активність кожного індивідуально.

У завершальній частині деталізації для кожного студента наводиться повний перелік усіх поданих заявок із такими атрибутами: **статус заявки** (PENDING / ACCEPTED / REJECTED), **дата подання** (наприклад, 28.03.2025), **назва вакансії** (Vacancy: Senior Frontend Developer (Kyiv)), **тип роботи** (Work Type: 0), а також **назва компанії** (Company: Vlad – Solutions). Така глибока інформація дає можливість університету аналізувати, на які галузі орієнтуються студенти, як змінюється успіх їхніх відгуків із часом і з якими компаніями вони найчастіше взаємодіють.

У підсумку звіт поєднує високорівневий огляд і детальну аналітику, що дозволяє університету приймати обґрунтовані рішення щодо вдосконалення освітніх програм, активізації співпраці з роботодавцями та своєчасної підтримки студентів у процесі працевлаштування.

ВИСНОВКИ

У першому розділі було проведено аналіз предметної області та виявлено ключові вимоги до майбутньої системи. Було вивчено особливості взаємодії між компаніями, студентами та університетами, а також сформульовано основні цілі: створення зручного інструменту для рекрутингу молодих спеціалістів, автоматизації пошуку за ключовими параметрами (університет, факультет, спеціальність, місто тощо), а також забезпечення достовірності даних у системі.

У другому розділі було спроєктовано архітектуру системи та базу даних. Розроблено моделі для компаній, студентів, університетів, вакансій, рекрутерів і токенів, які забезпечують авторизацію та зв’язок між сутностями. Особливу увагу приділено реалізації перевірки компаній і університетів через адміністратора, а також обмеженню дій для неперевірених акаунтів. У базі даних передбачено можливість зберігання ЄДРПОУ для обох типів організацій у єдиній структурі.

У третьому розділі реалізовано серверну та клієнтську частини проєкту. Сервер створено з використанням express, реалізовано API для взаємодії з базою даних, логіку обробки зображень та збереження їх у форматі WebP. На фронтенді використано React, Vite, TypeScript — для побудови зручного, адаптивного інтерфейсу з урахуванням ролей користувачів. Передбачено окремий функціонал для адміністратора, рекрутерів, студентів та університетів.

У четвертому розділі виконано тестування основної функціональності, перевірку коректної авторизації та прав доступу, створення та перегляду вакансій, відгуків студентів, підтвердження акаунтів. Також протестовано завантаження зображень, а також перегляд аналітики щодо зареєстрованих студентів університетом.

Отже, врезультаті виконання бакалаврської кваліфікаційної роботи було повністю реалізовано вебплатформу для пошуку студентів і працівників, що відповідає сучасним вимогам щодо функціональності, безпеки та зручності використання. Розроблена система дозволяє компаніям ефективно знаходити кандидатів за заданими критеріями, університетам — відстежувати активність своїх студентів, а студентам — легко шукати вакансії та відгукуватися на них.

ВИКОРИСТАНІ ДЖЕРЕЛА

1. LinkedIn Global Talent Trends [Електронний ресурс]. – Режим доступу: https://business.linkedin.com/talent-solutions/global-talent-trends

2. Платформа пошуку роботи Robota.ua [Електронний ресурс]. – Режим доступу: https://robota.ua/

3. Платформа пошуку роботи Work.ua [Електронний ресурс]. – Режим доступу: https://www.work.ua/

4. Система управління базами даних PostgreSQL [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/

5. Типи діаграм UML (DOU.ua) [Електронний ресурс]. – Режим доступу: https://dou.ua/forums/topic/40575/

6. Stack Overflow — технічні обговорення [Електронний ресурс]. – Режим доступу: https://stackoverflow.com/

7. Інформація про SQL на QAGROUP [Електронний ресурс]. – Режим доступу: https://qagroup.com.ua/publications/sql-info/

8. TypeScript — офіційна документація [Електронний ресурс]. – Режим доступу: https://www.typescriptlang.org/docs/

9. React — офіційна документація [Електронний ресурс]. – Режим доступу: https://react.dev/

10. Redux — офіційна документація [Електронний ресурс]. – Режим доступу: https://redux.js.org/

11. Redux Toolkit — сучасний інструмент для роботи з Redux [Електронний ресурс]. – Режим доступу: https://redux-toolkit.js.org/

12. RTK Query — бібліотека для запитів у Redux Toolkit [Електронний ресурс]. – Режим доступу: https://redux-toolkit.js.org/rtk-query/overview

13. Express — документація фреймворку [Електронний ресурс]. – Режим доступу: https://expressjs.com/

14. Material UI — React компоненти [Електронний ресурс]. – Режим доступу: https://mui.com/

15. Vite — сучасний інструмент для фронтенд-розробки [Електронний ресурс]. – Режим доступу: https://vitejs.dev/

16. JWT (JSON Web Tokens) — офіційна документація [Електронний ресурс]. – Режим доступу: https://jwt.io/introduction

17. REST API — основи та принципи [Електронний ресурс]. – Режим доступу: https://restfulapi.net/

18. Docker — офіційна документація [Електронний ресурс]. – Режим доступу: https://docs.docker.com/

19. GitHub — документація та репозиторії [Електронний ресурс]. – Режим доступу: https://github.com/

20. Axios — HTTP клієнт для браузера та Node.js [Електронний ресурс]. – Режим доступу: https://axios-http.com/

21. Figma — інструмент дизайну інтерфейсів [Електронний ресурс]. – Режим доступу: https://www.figma.com/

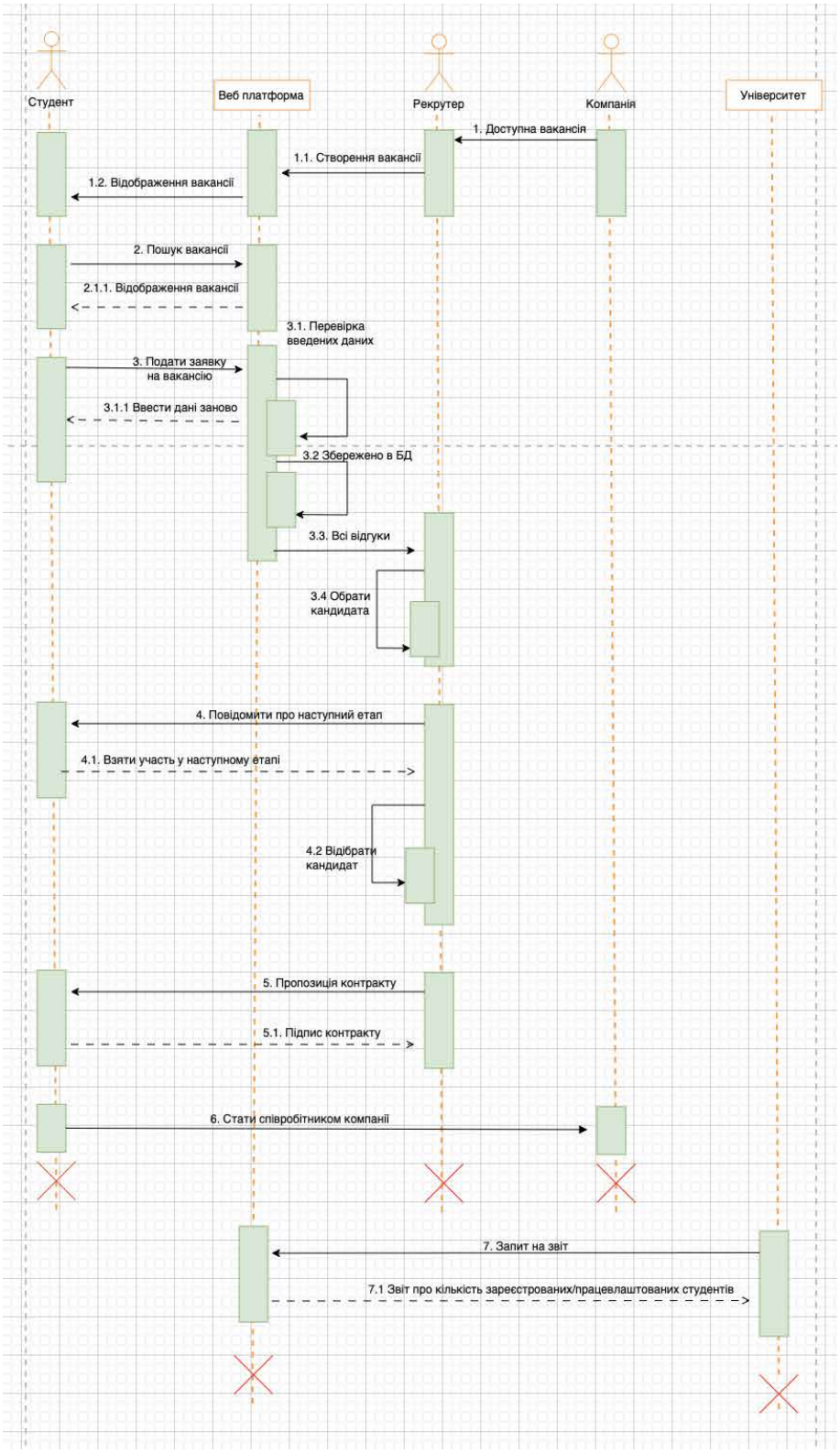
22. Cloudinary — сервіс зберігання та обробки зображень [Електронний ресурс]. – Режим доступу: https://cloudinary.com/documentation

23. Модулі для React-додатків — Gen Tech Journal [Електронний ресурс]. – Режим доступу: https://journal.gen.tech/post/modules-for-react-app

ДОДАТКИ

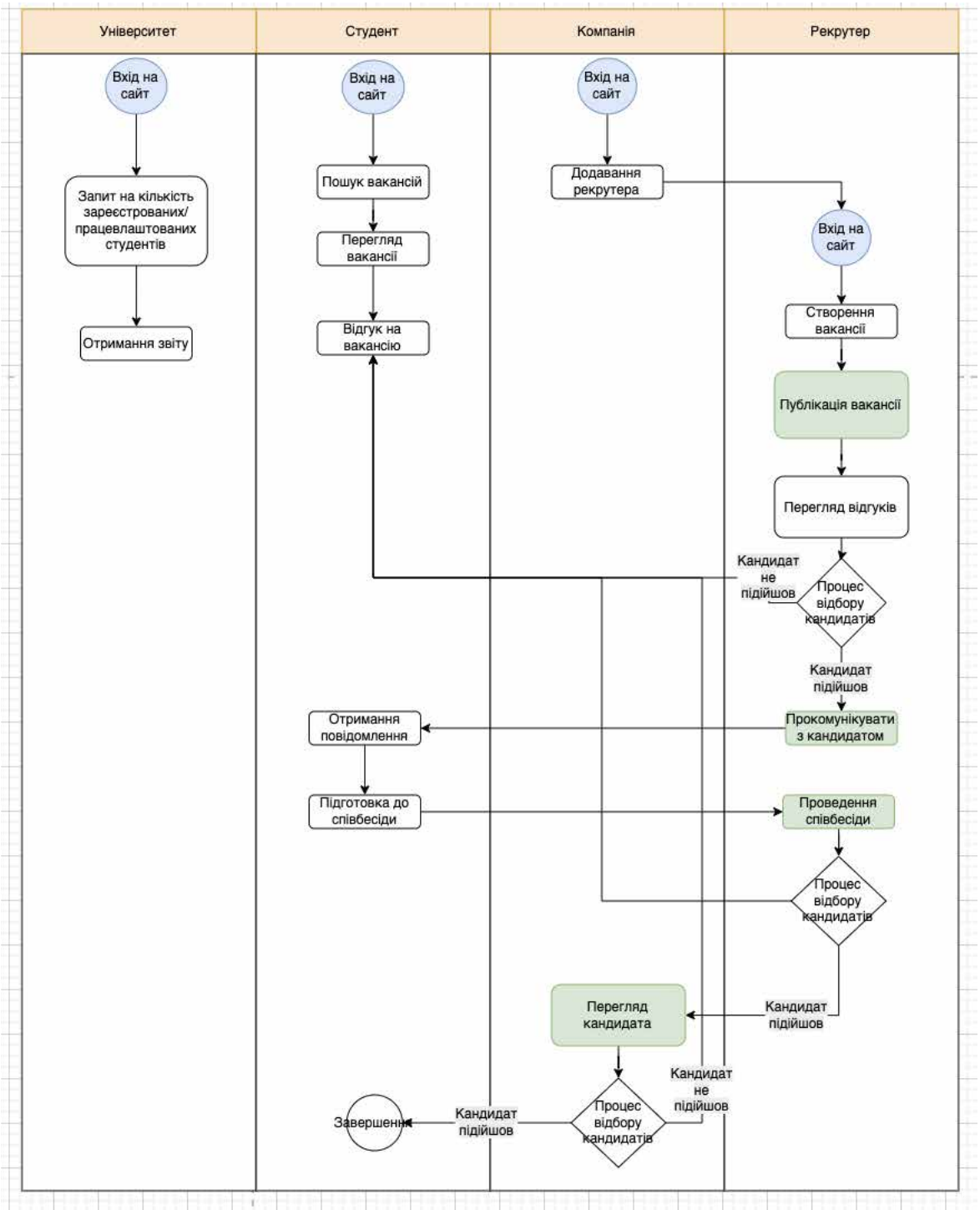
Додаток А

Діаграма послідовності



Додаток Б

Діаграма активності



Додаток В

DDL-схема створення для системи

```
CREATE TABLE IF NOT EXISTS companies (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    email VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    name VARCHAR(100) NOT NULL,
    surname VARCHAR(100) NOT NULL,
    title VARCHAR(255) NOT NULL,
    description TEXT,
    phone VARCHAR(20),
    logo VARCHAR(255),
    egrpou VARCHAR(8) NOT NULL,
    role_id INTEGER NOT NULL REFERENCES roles(id),
    is_activated BOOLEAN DEFAULT false,
    activation_link VARCHAR(255),
    activation_link_expires_at TIMESTAMP WITH TIME ZONE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS universities (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    email VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    name VARCHAR(100) NOT NULL,
    surname VARCHAR(100) NOT NULL,
    role_id INTEGER NOT NULL REFERENCES roles(id),
    is_activated BOOLEAN DEFAULT false,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS students (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    email VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    name VARCHAR(100) NOT NULL,
    surname VARCHAR(100) NOT NULL,
    role_id INTEGER NOT NULL REFERENCES roles(id),
    is_activated BOOLEAN DEFAULT false,
    university_id UUID REFERENCES universities(id) ON DELETE SET NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS recruiters (
```

```
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    company_id UUID NOT NULL REFERENCES companies(id) ON DELETE CASCADE,
    email VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    name VARCHAR(100) NOT NULL,
    surname VARCHAR(100) NOT NULL,
    role_id INTEGER NOT NULL REFERENCES roles(id),
    is_activated BOOLEAN DEFAULT false,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS vacancies (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    company_id UUID NOT NULL REFERENCES companies(id) ON DELETE CASCADE,
    recruiter_id UUID NOT NULL REFERENCES recruiters(id) ON DELETE CASCADE,
    title VARCHAR(255) NOT NULL,
    description TEXT NOT NULL,
    requirements TEXT[] NOT NULL,
    salary_from INTEGER NOT NULL,
    salary_to INTEGER NOT NULL,
    currency VARCHAR(3) NOT NULL CHECK (currency IN ('USD', 'EUR', 'UAH')),
    location VARCHAR(100) NOT NULL,
    work_type INTEGER NOT NULL,
    is_activated BOOLEAN DEFAULT true,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT salary_range CHECK (salary_to >= salary_from)
);

CREATE TABLE IF NOT EXISTS applications (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    vacancy_id UUID NOT NULL REFERENCES vacancies(id) ON DELETE CASCADE,
    student_id UUID NOT NULL REFERENCES students(id) ON DELETE CASCADE,
    status VARCHAR(20) NOT NULL DEFAULT 'pending' CHECK (status IN ('pending',
'accepted', 'rejected')),
    cover_letter TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(vacancy_id, student_id)
);
```

Додаток Д

Процедури та уявлення для обчислення статистики компаній і рекрутерів

```
CREATE OR REPLACE FUNCTION get_date_range(period time_period)
RETURNS TABLE (start_date TIMESTAMP WITH TIME ZONE, end_date TIMESTAMP WITH
TIME ZONE) AS $$
BEGIN
  RETURN QUERY
  SELECT
    CASE period
      WHEN 'current_month' THEN date_trunc('month', CURRENT_DATE)
      WHEN 'last_month' THEN date_trunc('month', CURRENT_DATE - INTERVAL '1 month')
      WHEN 'last_half_year' THEN date_trunc('month', CURRENT_DATE - INTERVAL '6
months')
      WHEN 'last_year' THEN date_trunc('month', CURRENT_DATE - INTERVAL '1 year')
      ELSE '1970-01-01'::timestamp
    END,
    CASE period
      WHEN 'current_month' THEN date_trunc('month', CURRENT_DATE + INTERVAL '1
month')
      WHEN 'last_month' THEN date_trunc('month', CURRENT_DATE)
      WHEN 'last_half_year' THEN date_trunc('month', CURRENT_DATE)
      WHEN 'last_year' THEN date_trunc('month', CURRENT_DATE)
      ELSE CURRENT_TIMESTAMP
    END;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE VIEW company_statistics AS
SELECT
  c.id as company_id,
  COUNT(DISTINCT r.id) as total_recruiters,
  COUNT(DISTINCT v.id) as total_vacancies,
  COUNT(DISTINCT CASE WHEN a.status = 'accepted' THEN a.id END) as
total_hired_students,
  COUNT(DISTINCT CASE WHEN a.created_at >= date_trunc('month', CURRENT_DATE)
THEN a.id END) as new_applications_this_month
FROM companies c
LEFT JOIN recruiters r ON c.id = r.company_id
LEFT JOIN vacancies v ON c.id = v.company_id
LEFT JOIN applications a ON v.id = a.vacancy_id
GROUP BY c.id;

CREATE OR REPLACE VIEW recruiter_performance AS
SELECT
  r.id as recruiter_id,
  r.company_id,
  COUNT(DISTINCT v.id) as total_vacancies,
```

```
        COUNT(DISTINCT a.id) as total_applications,
        COUNT(DISTINCT CASE WHEN a.status = 'accepted' THEN a.id END) as successful_hires,
        COUNT(DISTINCT CASE WHEN a.created_at >= date_trunc('month', CURRENT_DATE)
THEN a.id END) as new_applications_this_month
    FROM recruiters r
    LEFT JOIN vacancies v ON r.id = v.recruiter_id
    LEFT JOIN applications a ON v.id = a.vacancy_id
    GROUP BY r.id, r.company_id;

    CREATE OR REPLACE FUNCTION get_company_metrics(
        company_id INTEGER,
        period time_period
    )
    RETURNS TABLE (
        total_vacancies BIGINT,
        total_applications BIGINT,
        successful_hires BIGINT,
        new_recruiters BIGINT,
        active_recruiters BIGINT
    ) AS $$
    DECLARE
        date_range RECORD;
    BEGIN
        date_range := get_date_range(period);

        RETURN QUERY
        SELECT
            COUNT(DISTINCT v.id),
            COUNT(DISTINCT a.id),
            COUNT(DISTINCT CASE WHEN a.status = 'accepted' THEN a.id END),
            COUNT(DISTINCT CASE WHEN r.created_at >= date_range.start_date THEN r.id END),
            COUNT(DISTINCT CASE WHEN r.is_activated = true THEN r.id END)
        FROM companies c
        LEFT JOIN vacancies v ON v.company_id = c.id
        LEFT JOIN applications a ON a.vacancy_id = v.id
        LEFT JOIN recruiters r ON r.company_id = c.id
        WHERE c.id = company_id
        AND (a.created_at IS NULL OR a.created_at >= date_range.start_date)
        AND (r.created_at IS NULL OR r.created_at >= date_range.start_date);
    END;
    $$ LANGUAGE plpgsql;

    CREATE OR REPLACE FUNCTION get_recruiter_metrics(
        recruiter_id UUID,
        period time_period
    )
    RETURNS TABLE (
        total_vacancies BIGINT,
        total_applications BIGINT,
        successful_hires BIGINT,
        success_rate NUMERIC
    ) AS $$
```

```
DECLARE
   date_range RECORD;
BEGIN
   date_range := get_date_range(period);

   RETURN QUERY
   SELECT
      COUNT(DISTINCT v.id),
      COUNT(DISTINCT a.id),
      COUNT(DISTINCT CASE WHEN a.status = 'accepted' THEN a.id END),
      ROUND(
         COUNT(DISTINCT CASE WHEN a.status = 'accepted' THEN a.id END)::numeric /
         NULLIF(COUNT(DISTINCT a.id), 0) * 100,
         2
      )
   FROM recruiters r
   LEFT JOIN vacancies v ON r.id = v.recruiter_id
   LEFT JOIN applications a ON v.id = a.vacancy_id
   WHERE r.id = recruiter_id
   AND (a.created_at BETWEEN date_range.start_date AND date_range.end_date
       OR v.created_at BETWEEN date_range.start_date AND date_range.end_date);
END;
$$ LANGUAGE plpgsql;
```

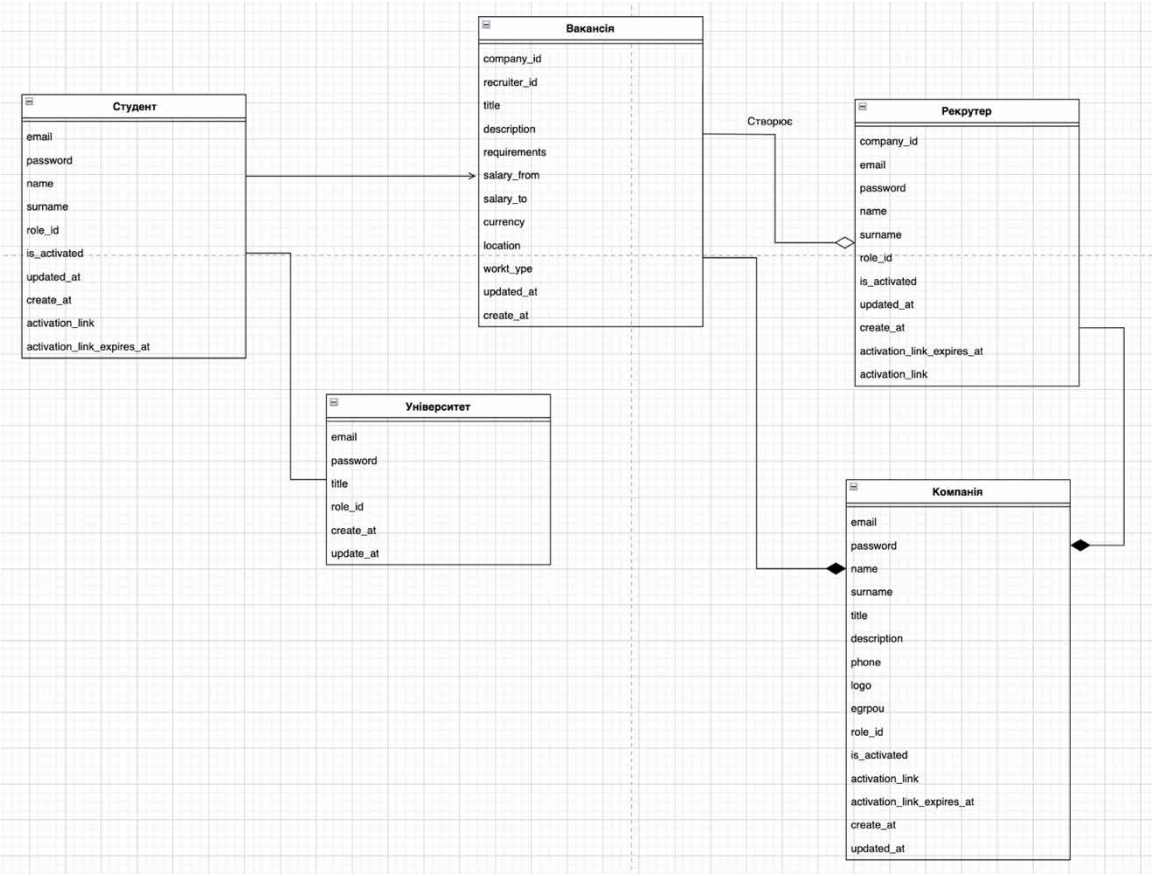
Додаток Е

Загальна функція update_updated_at_column

```
CREATE OR REPLACE FUNCTION update_updated_at_column()
RETURNS TRIGGER AS $$
BEGIN
    NEW.updated_at = CURRENT_TIMESTAMP;
    RETURN NEW;
END;
$$ language 'plpgsql';

CREATE TRIGGER update_companies_updated_at
    BEFORE UPDATE ON companies
    FOR EACH ROW
    EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_universities_updated_at
    BEFORE UPDATE ON universities
    FOR EACH ROW
    EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_students_updated_at
    BEFORE UPDATE ON students
    FOR EACH ROW
    EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_recruiters_updated_at
    BEFORE UPDATE ON recruiters
    FOR EACH ROW
    EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_vacancies_updated_at
    BEFORE UPDATE ON vacancies
    FOR EACH ROW
    EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_applications_updated_at
    BEFORE UPDATE ON applications
    FOR EACH ROW
    EXECUTE FUNCTION update_updated_at_column();
```

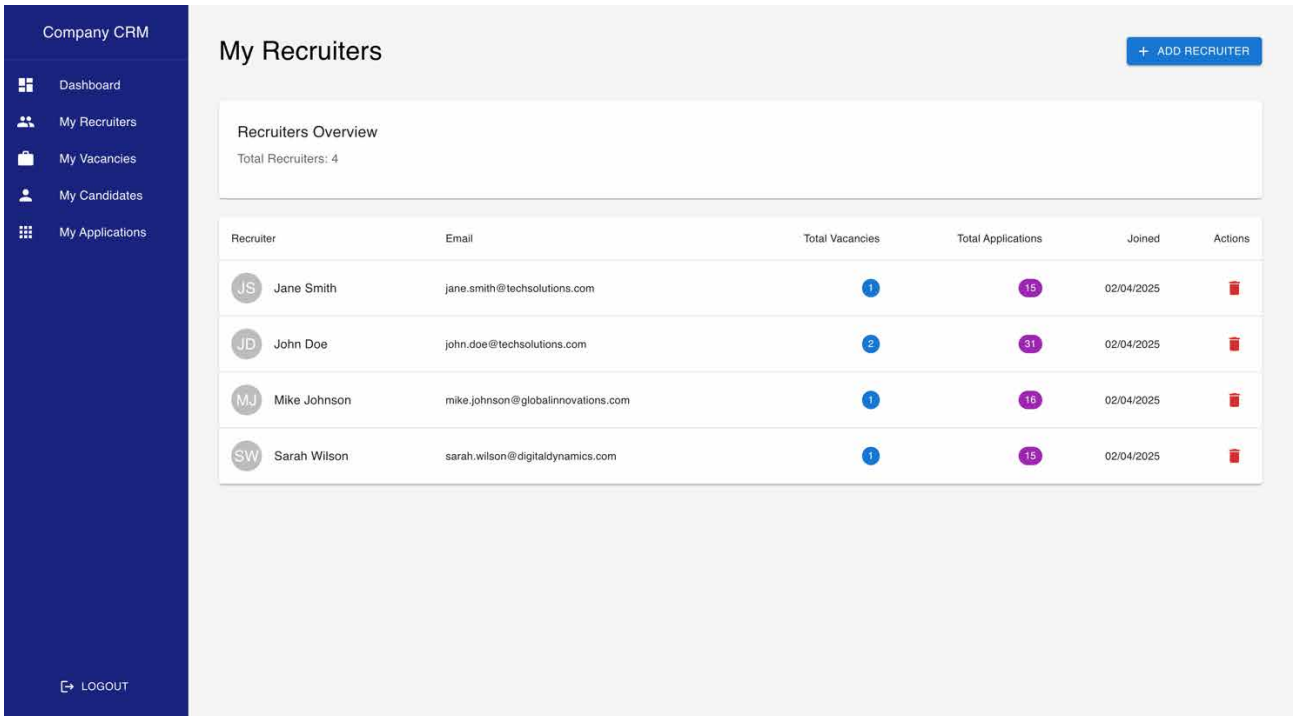
Додаток Ж

Основні сутності системи



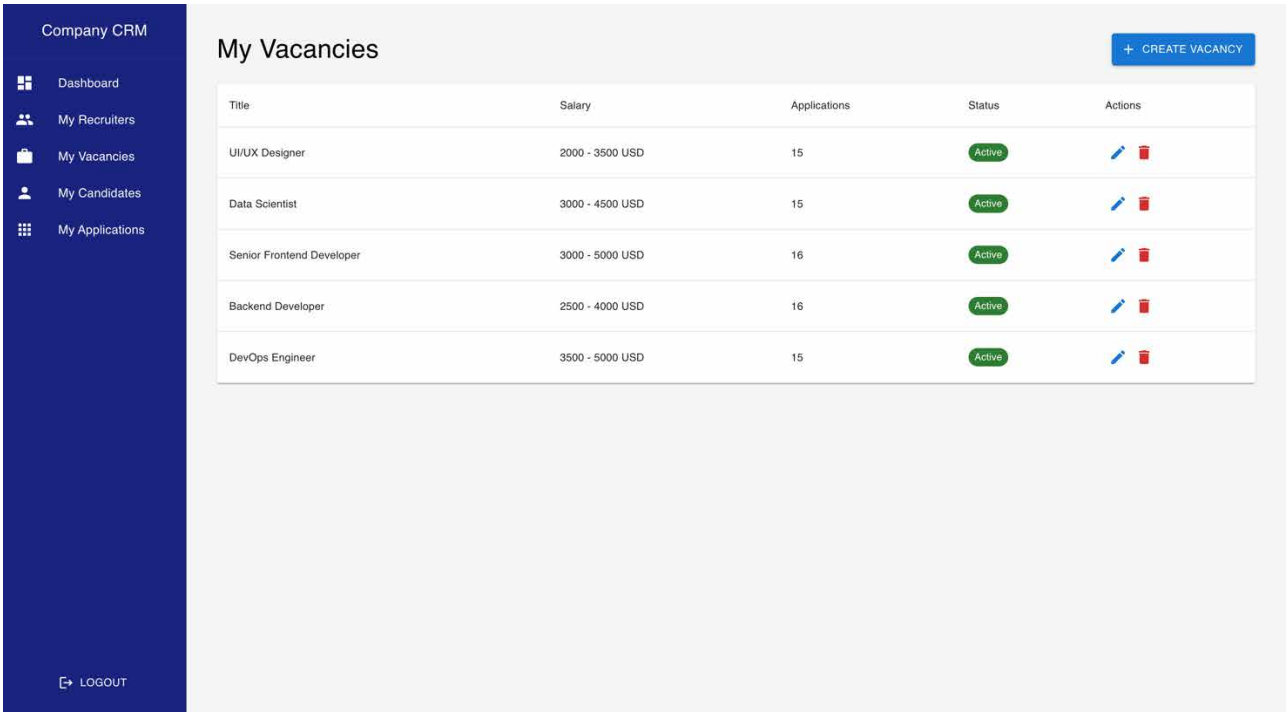
Додаток К

Сторінка «My Recruiters»



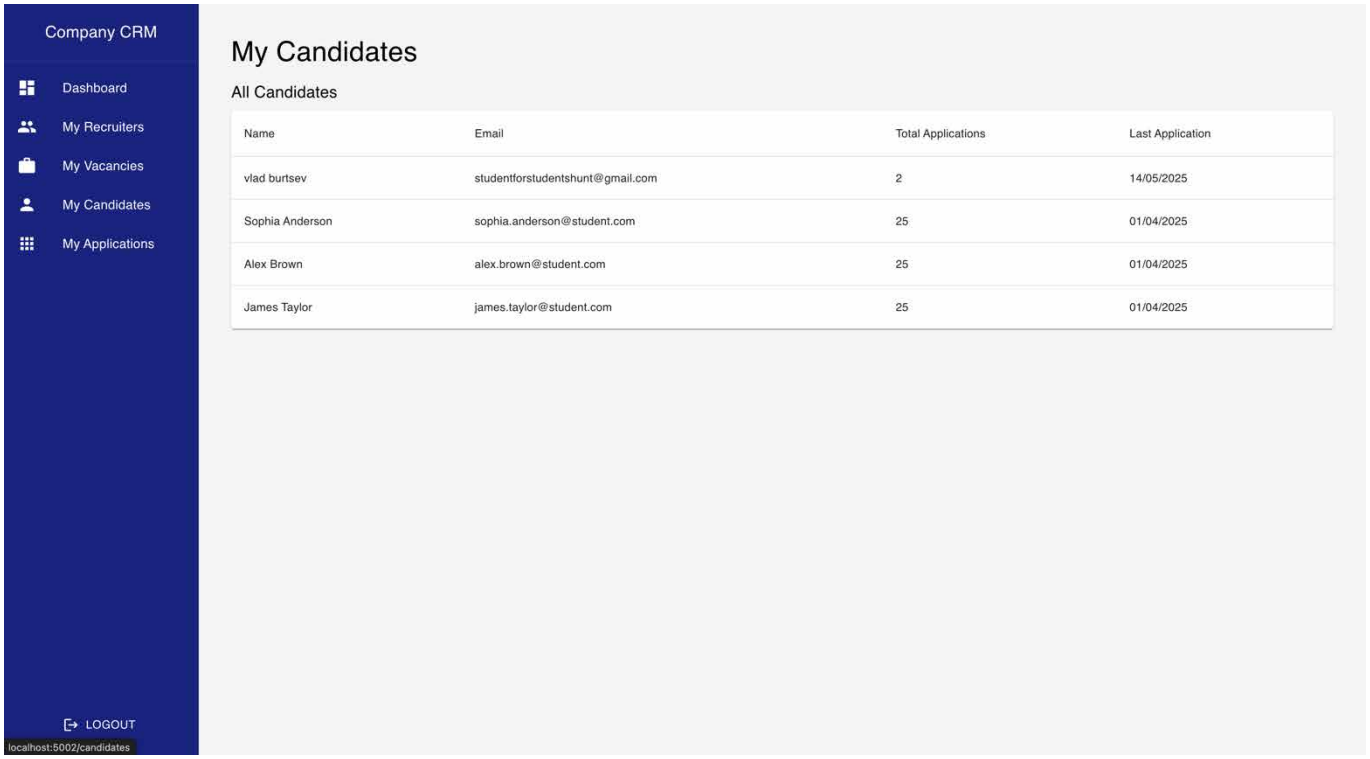
Додаток Л

Сторінка «My Vacancies»



Додаток М

Сторінка «My Candidates»



Додаток Н

Сторінка «My Applications»