ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-застосунку для автоматизації процесів найму співробітників»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Валерій ОЛІЙНИК
(підпис)

Виконав: здобувач вищої освіти групи ПД-43

Валерій ОЛІЙНИК

Керівник: Юрій БАЖАН
викладач

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ «
____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Олійнику Валерію Олександровичу

1. Тема кваліфікаційної роботи: «Розробка web-застосунку для автоматизації процесів найму співробітників»
керівник кваліфікаційної роботи викладач кафедри ІПЗ Юрій БАЖАН,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про процеси найму персоналу, опис сучасні методи рекрутингу, інструменти автоматизації управління кандидатами, а також наведено технічну документацію з описом бібліотек і фреймворків для розробки web-застосунків (NestJS, Next.js, PostgreSQL, TypeORM, Ant Design, Tailwind CSS).

4. Зміст розрахунково-пояснювальної записки:

1. Огляд та аналіз існуючих методів і технологій автоматизації процесів найму персоналу в контексті сучасних компаній.
2. Проектування web-застосунку для автоматизації управління процесами найму співробітників.
3. Програмна реалізація та опис функціонування web-застосунку для автоматизації процесів найму співробітників.
4. Тестування web-застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Архітектура застосунку.
6. Схема бази даних.
7. Екранні форми.
8. Демонстрація роботи програми.
9. Апробація результатів дослідження .

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд існуючих алгоритмів обробки тексту та методів і технологій побудови тематичного словника	14.03.-18.03.2025	
4	Проектування застосунку для автоматизованої побудови тематичного словника навчальної дисципліни	19.03-24.03.2025	
5	Програмна реалізація застосунку	25.03.-13.04.20245	
6	Тестування застосунку	14.04.-28.04.2025	

7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Валерій ОЛІЙНИК

(підпис)

Керівник
кваліфікаційної роботи _____ Юрій БАЖАН

(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 50 стор., 8 рис., 21 джерел.

Мета роботи – підвищення ефективності процесу найму співробітників шляхом автоматизації окремих етапів рекрутингу з використанням сучасних web-технологій.

Об'єкт дослідження – процес найму персоналу у сучасних компаніях.

Предмет дослідження – інструменти та технології web-розробки, що можуть бути використані для автоматизації найму співробітників.

Короткий зміст роботи: У роботі проаналізовано процес найму та розроблено web-застосуцнок для автоматизації. Проаналізовано сучасні методи рекрутингу, платформи (Work.ua, Robota.ua, GRC.ua) та інструменти автоматизації, висвітлено їх переваги та недоліки. Розроблено трирівневу клієнт-серверну архітектуру з використанням технологічного стеку: NestJS, PostgreSQL та TypeORM на стороні сервера, Next.js, Ant Design та Tailwind CSS на стороні клієнта. Реалізовано функціональні модулі: управління користувачами, публікація вакансій, кандидатів, заявок, аутентифікація (JWT) та REST API з документацією через Swagger. Розроблено інтуїтивно зрозумілий інтерфейс, який підтримує фільтрацію, пагінацію та доступ за ролями. Проведено ручне UI/UX тестування та тестування API через Postman, щоб підтвердити стабільність та відповідність вимогам. Застосунок оптимізував процес найму та скоротив час, який витрачає HR-відділ.

Сферою використання є автоматизація найму в компаніях різного масштабу.

КЛЮЧОВІ СЛОВА: РЕКРУТИНГ, АВТОМАТИЗАЦІЯ, WEB-ЗАСТОСУНОК, NESTJS, NEXT.JS, POSTGRESQL, TYPEORM, REST API, АВТЕНТИФІКАЦІЯ, UI/UX, HR.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

HR – Human Resources

ATS – Applicant Tracking System

API – Application Programming Interface

REST – Representational State Transfer

JWT – JSON Web Token

UI – User Interface

UX – User Experience

SSR – Server-Side Rendering

SSG – Static Site Generation

CSR – Client-Side Rendering

СУБД – Система управління базами даних

Three-Tier Architecture – Трирівнева архітектура (представлення, бізнес-логіка, дані)

ВСТУП

Актуальність: У сучасному світі успіх бізнесу компанії залежить від ефективності процесу найму працівників. Ефективний рекрутинг безпосередньо впливає на розвиток організації, а також конкурентоспроможність. Оперативність закриття вакансій висококваліфікованими кандидатами та раціоналізація бюджету на залучення працівників забезпечують швидке формування професійної команди, що є одним із основних факторів досягнення визнання та впізнаваності підприємства. На жаль, застарілі методи роботи HR-відділу часто не є ефективними, займають багато часу, що призводить до утворення помилок через людський фактор. У результаті, втрачається нагода найму талановитих співробітників, що збільшує період рекрутингу та навантаження HR-команди.

Організації потребують залучення сучасних інструментів автоматизації в процес найму. Web-застосунки сприяють налаштуванню данних про кандидатів, поліпшенню комунікацій між HR-командою та претендентами, а також поліпшити рутинні задачі. Застосування новітніх технологічних стеків, зокрема NestJS для серверної частини й Next.js для інтерфейсу, сприяє розробці адаптивних, дієвих та продуктивних рішень, що є ключовим фактором відповідності умовам сучасного ринку.

Об'єктом дослідження є процес найму персоналу у сучасних компаніях.

Предметом дослідження є інструменти та технології web-розробки, що можуть бути використані для автоматизації найму співробітників.

Метою роботи є підвищення ефективності процесу найму працівників шляхом автоматизації окремих етапів рекрутингу з використанням сучасних web-технологій для оптимізації роботи HR-відділу та поліпшення комунікації з кандидатами.

Методи дослідження: першим кроком було проаналізовано науково-технічну літературу та інтернет-джерела відповідно до теми дослідження, а також аналіз

існуючих рішень для поглибленого вивчення предметної області та формулювання функціональних і нефункціональних вимог до програмного забезпечення. Для моделювання використано методи об’єктно-орієнтованого проектування, зокрема побудову ER-діаграм. Розробка програмного забезпечення велася з використанням відповідних методів, а тестування продукту здійснювалось із застосуванням методів перевірки якості програмних рішень. Дослідження реалізації застосунку відбувалося впродовж його розробки.

Наукова новизна роботи полягає у розробці особливої архітектури інтеграції модулів для забезпечення модифікованого налаштування етапів процесу найму працівників, враховуючи персональні запити конкретної компанії з використання обраного стеку технологій.

Практична значущість результатів полягає в створенні web-застосунку для HR-відділів задля підвищення продуктивності процесу найму, що дає перспективу пришвидшити процес закриття вакансій, покращення досконалості підбору кандидатів та оптимізації витрат на функціонування. Розроблений застосунок забезпечує цілісне сховище даних, покращує взаємодію та пропонує інструменти для моніторингу ефективності роботи рекрутингових процесів.

1 ДОСЛІДЖЕННЯ РЕКРУТИНГОВИХ ПРОЦЕСІВ І ТЕХНОЛОГІЧНИХ РІШЕНЬ

1.1. Структура та етапи процесу підбору персоналу

Процес найму працівників займає визначальне місце в керуванні людськими ресурсами, орієнтуючий на підбір компетентних фахівців шляхом ретельного оцінювання та найму для успішного виконання організаційних завдань. Оперативне планування, дієва координація та застосування новітніх інструментів для забезпечення конкурентоспроможності компанії на ринку праці формують результативний рекрутинг[1]. Основні етапи даного процесу включають:

- формування штатних потреб. Керівники підрозділів досліджують наявні та наступні завдання, окреслюють необхідність нового працівника, його посаду, обов’язки, а також головні повноваження. Після цих дій формується заявка на вакансію, яка направляється до співробітників HR-відділу;
- структуроване позиціонування вакансії. HR-команда створює деталізований опис посади, зокрема інформацію щодо наявності досвіду, навичок, відповідної освіти, а також деталі стосовно умов праці, компанії та переваги роботи саме в цій організації. Опис повинен бути чітким, відповідати стилю компанії, спонуккати кандидатів відгукнутися саме на цю пропозицію;
- пошук та залучення кандидатів. Вакансія розміщується на відповідних платформах (Work.ua, LinkedIn, Rabota.ua). Такі сайти можуть допомогти відстежувати прогрес кандидатів та оцінювати їхню придатність. Також вакансія можна опублікувати у соціальних мережах та корпоративному вебсайті;

Рис. 1.1 Розміщення вакансій на спеціалізованих платформах

- первинний відбір кандидатів. Даний етап включає в себе аналіз отриманих резюме, а також інформацію про відповідність запитам описаних у вакансії. Для розгляду лише релевантних кандидатів використовують фільтри за ключовими словами або ж необхідним досвідом, щоб процес найму не витрачав багато часу[2];
- проведення співбесід. Даний етап містить декілька основних форматів: - телефонна співбесіда або онлайн-інтерв'ю для перевірки наявної загальної придатності, комунікативних вмінь та мотивації у роботі на даній посаді та компанії; - технічна частина інтерв'ю, яка складається з практичних завдань та кейсів, виконання яких дозволить оцінити професійну компетентність кандидата; - фінальна співбесіда з HR-менеджером для узгодження професійних прагнень та умов співпраці;
- підготовка пропозиція щодо умов працевлаштування. Після обрання кандидата підготовується та надсилається офіційний офер, з описом умов співпраці (обов'язки, графік, зарплата, бонуси);

- онбординг. Після прийняття пропозиції автоматизована система допомагає з оформленням документів, передачею даних до HR-відділу а також організації адаптаційного процесу, що включає навчальний період та знайомство з командою[3].

Дані етапи відтворюють повний цикл рекрутингу, який вимагає чіткої узгодженості та дієвого управління задля здобутку найсприятливіших результатів.

1.2. Проблематика ручного управління рекрутингом

Багато компаній малого та середнього бізнесу не використовують автоматизовані методи управління рекрутингом. Дана помилка створює труднощі в процесі дієвого найму працівників. Враховуючи результати дослідження SHRM, рекрутери витрачають до 60% робочого часу на рутинні завдання, що негативно впливає на ефективність роботи[4].

Рис. 1.2 П'ять найбільш трудомістких завдань з найму персоналу для малого та середнього бізнесу

Основні проблеми ручного підходу містять:

- суттєві часові витрати. Оприлюднення вакансій на всіх можливих платформах, суб'єктивний аналіз резюме, комунікація з кандидатами, домовленість про співбесіди та збір зворотного зв'язку займають набагато більше часу, ніж хотілося б. Саме через це HR-фахівці відволікаються від більш важливих стратегічних завдань;
- втрата важливих даних. Зазвичай вся інформація стосовно кандидатів (резюме, опис, фідбек) зберігається в месенджерах, електронній пошті, таблицях, що може призвести до втрати інформації. Повторне залучення кандидатів витрачає зайвий час, через втрачені дані, які могли зберігатися в автоматизованих сховищах;
- недостатньо ефективна координація. Нестача простору для взаємодії в HRкоманді може призвести до затримки в обміні інформацією або наявності помилок у координації;
- ризик суб'єктивності. Часто людський фактор може призвести до упереджених рішень. Наприклад, неструктурований зворотній зв'язок про кандидата або нестандартні критерії оцінки можуть знизити якість підбору працівників;
- негативний досвід кандидатів. Здебільшого кандидати відмовляються від пропозицій роботодавців через поганий попередній досвід під час найму. Затримки у відповідях, повторні запити даних HR-фахівцями можуть погіршити ставлення кандидатів до вакансії та компанії в цілому;
- відсутність аналітичних даних. Ручний аналіз джерел залучення кандидатів не дозволяє ефективно визначити рівень залучення на кожному етапі та середній час закриття вакансії, що не дає можливості для оптимізації процесу.

Саме ці перешкоди описують потребу в автоматизації задля збільшення ефективності та конкурентоспроможності компаній.

1.3. Переваги автоматизації рекрутингових процесів

Автоматизація процесів HR-працівників, яка реалізовується за допомогою спеціалізованих web-застосунків, дозволяє усунути велику кількість труднощів ручного управління, що дає стратегічні переваги компаніям. Впровадження даних систем сприяє раціоналізації ресурсів, удосконалення процесу найму та поліпшення взаємодії всіх учасників процесу:

- прискорення процесу найму. Автоматизація монотонних процесів, таких як аналіз резюме, планування співбесід чи масова розсилка повідомлень, скорочує час закриття вакансій[5];
- покращення комунікації. Система, яка об’єднує всіх учасників процесу у єдиному цифровому просторі покращує комунікацію та забезпечує підвищення якості відбору;
- централізоване сховище даних. Єдина база кандидатів, яка містить резюме, історію комунікації та фідбеком, забезпечує швидкий доступ до інформації, яка буде доступна рекрутерам, менеджерам та іншим відповідним учасникам команди;
- зниження помилок. Автоматизація стандартних процесів мінімізує людський фактор, при обробці даних чи відбору майбутніх співробітників. Стандартизовані форми оцінки та аналітика зменшують суб’єктивність сприяючи об’єктивному вибору кандидатів, що також покращує досвід кандидатів;
- покращення досвіду кандидатів. Чіткий, налаштований процес, швидкий зворотній зв’язок, автоматизовані повідомлення формують позитивне враження про компанію, що вдосконалює бренд роботодавця;

- аналітичні можливості. Система генерує звіти про продуктивність джерел надходження кандидатів, тривалість найму, що допомагає вдосконалити стратегії рекрутингу та оптимізувати використання ресурсів.

Тобто, автоматизація покращує управління внутрішніми діями, а також допомагає виділятися від інших компаній на ринку праці, що буде залучати кандидатів.

1.4. Огляд сучасних систем управління підбору персоналу

Вивчаючи ринок, можна сказати, що існує велика кількість систем управління кандидатами (Applicant Tracking Systems – ATS) та більш складних HR-платформ. Для визначення функціональних можливостей та переваг, які має надавати розроблюваний застосунок, було проведено аналіз декількох популярних рішень.

1.4.1. Work.ua

Work.ua – це провідна українська платформа для пошуку роботи, заснована у 2006 році, яка виступає як універсальне рішення для компаній і кандидатів. Вона приваблює зручністю і простотою використання, що робить її універсальною для використання бізнесами різного масштабу – від малого бізнесу до великих організацій.

Щодо статистики, то платформу відвідали понад 4 мільйони користувачів щомісяця, що свідчить про її значний вплив на ринок праці України. Щодня 400 тисяч відвідувачів переглядають дану платформу для пошуку роботи. Work.ua підтримує весь цикл рекрутингу: від розміщення вакансій (понад 96 000 активних вакансій) до пошуку кандидатів через фільтри за спеціалізацією, досвідом і локацією.

Серед додаткових функцій – мобільний додаток із функціями перегляду вакансій і базова аналітика відвідуваності вакансій. Work.ua також пропонує розсилку свіжих вакансій для кандидатів, що підвищує їхню залученість[6].

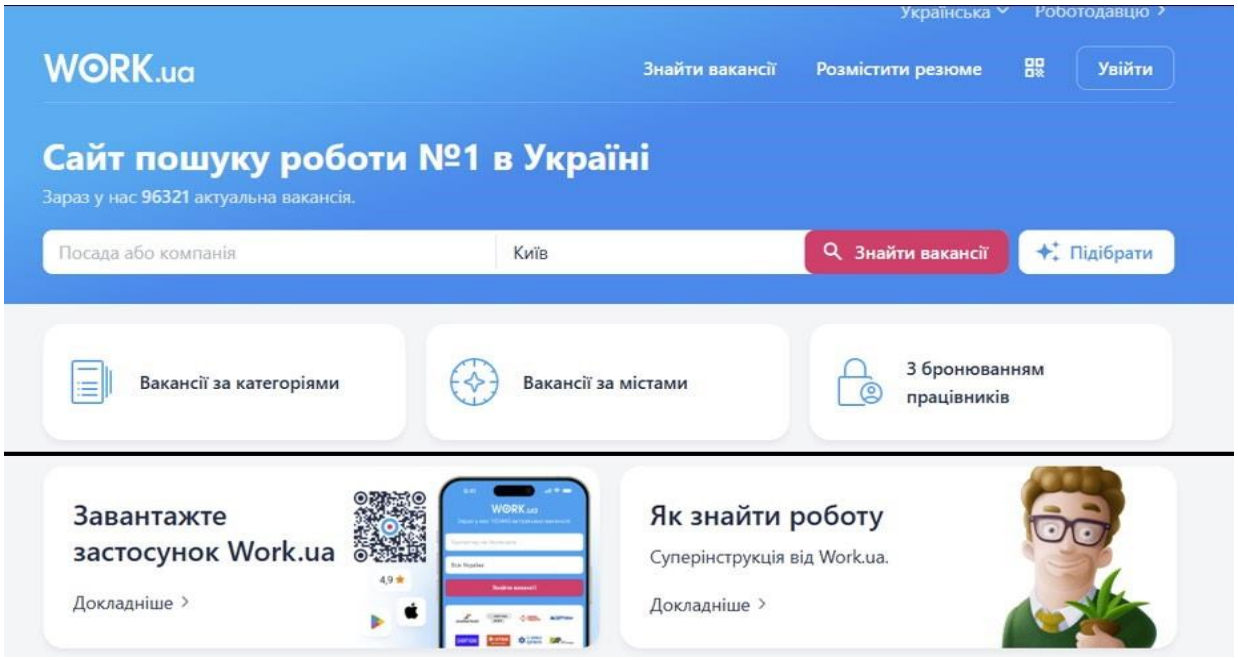


Рис. 1.3 Інтерфейс головної сторінки платформи Work.ua

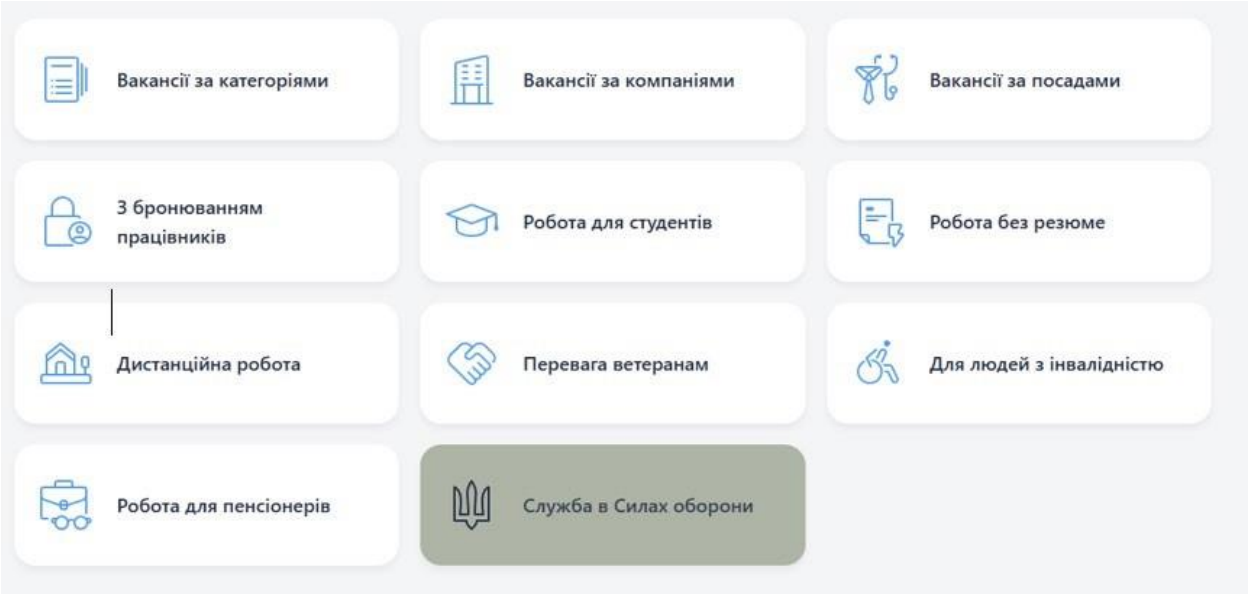


Рис. 1.4 Перелік доступних категорій вакансій для різних груп користувачів на платформі Work.ua

Функціонал:

- доступ до 200+ job-платформах для публікації вакансій. Можливість шукати кандидатів за різними параметрами(досвід, зарплата, локація);
- перегляд профілів кандидатів та засоби для комунікації з претендентами;
- мобільний додаток для автоматизованої розсилки вакансій і базовим аналізом даних.

Переваги: інтуїтивний інтерфейс, велика база кандидатів (понад 1 млн резюме), підтримка інтеграцій із HRM-системами.

Недоліки: рекрутинг із недостатньо оптимізованою автоматизацією, низька можливість для аналітики в складних рекрутингових компаній.

1.4.2. Robota.ua

Robota.ua – одна з перших онлайн-платформ пошуку роботи в Україні, заснована у 2000 році, і є лідером на ринку з приблизно 5 мільйонами відвідувань щомісяця. Платформа спрямована на компанії, які шукають кандидатів у різних регіонах України, та кандидатів, які потребують швидкого доступу до вакансій (понад 100 000 активних вакансій). Robota.ua вирізняється інтеграцією з HRM-системами, такими як PeopleForce та Bitrix24, що забезпечує автоматичний імпорт резюме та синхронізацію записів.

Система застосовує ШІ для визначення релевантних пропозицій вакансій кандидатам на основі їхньої історії пошуку та забезпечує основні інструменти для рекрутерів, а саме чат із кандидатами, сповіщення про перегляд резюме та географічну прив'язку.

Завдяки мобільному додатку Robota.ua кандидати можуть оперативно відгукуватись на вакансії та переглядати пропозиції за їх географічним розташуванням, що є зручним для мобільних користувачів [7].

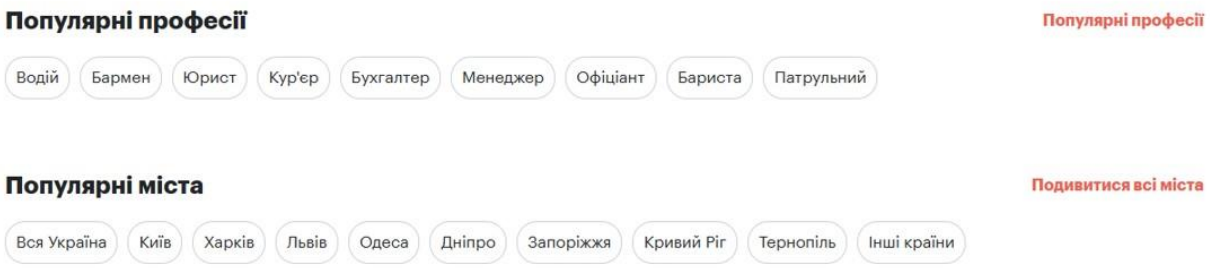


Рис. 1.5 Інтерфейс вибору популярних професій та регіонів для пошуку роботи на платформі Robota.ua

Функціонал**:**

- розміщення вакансій у провідній базі України (100 000+ вакансій);
- персоналізований підбір вакансій за допомогою аналізу ШІ;
- автоматизовані засоби взаємодії (чати, повідомлення);
- мобільний додаток із функціями геолокації та швидкого відгуку.

Переваги**:** розширена база вакансій, розгорнутий набір AI-інструментів, злиття з HR-системами для автоматизованого управління.

Недоліки**:** менш інтуїтивний інтерфейс порівняно з конкурентами, обмежена аналітика для рекрутерів.

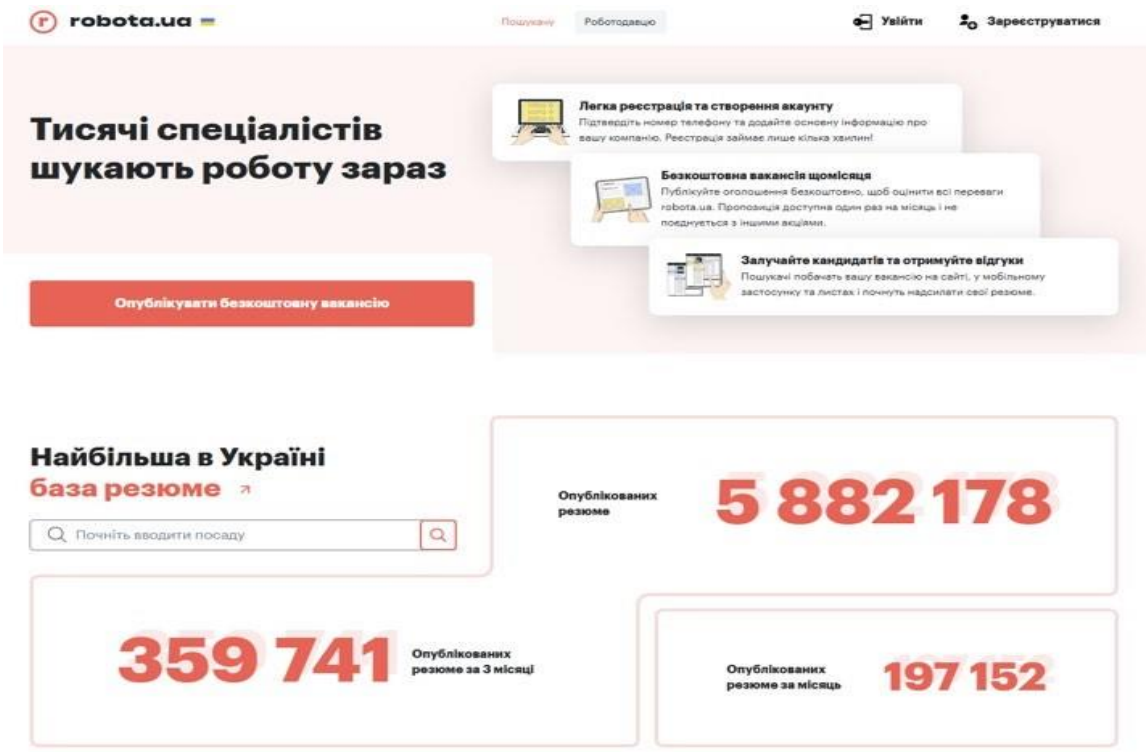


Рис. 1.5 Головна сторінка платформи Robota.ua для роботодавців

1.4.3. Headhunter.ua

Headhunter.ua (тепер відомий як GRC.ua після ребрендингу у 2019 році) – це платформа для пошуку професіоналів, яка була частиною міжнародної мережі HeadHunter Group, а наразі належить українським співвласникам.

Заснована як ресурс для спеціалістів середньої та вищої ланки, платформа фокусується на кандидатів із вищою освітою та досвідом (90% мають повну вищу освіту, 63% шукають зарплату від 9000 грн). GRC.ua надає доступ до перевіреної бази резюме (понад 160 000 перевірених профілів) і пропонує інтегровані інструменти для управління процесом пошуку персоналу.

Після ребрендингу GRC.ua позиціонується стала платформою, яка об’єднує всіх учасників ринку праці, включаючи консалтингові послуги[8].

Функціонал**:**

- Велика система понад 160 000+ верифікованих профілів для рекрутерів;
- CRM-система для управління вакансіями та моніторингу кандидатів; -
Базова аналітика та дослідження ринку праці.

Переваги: першочерговий пошук професійних спеціалістів, ретельно перевірена та контрольована база, передові інструменти для HR.
Недоліки: погіршення впізнаваності після оновлення бренду, платне API для інтеграцій.

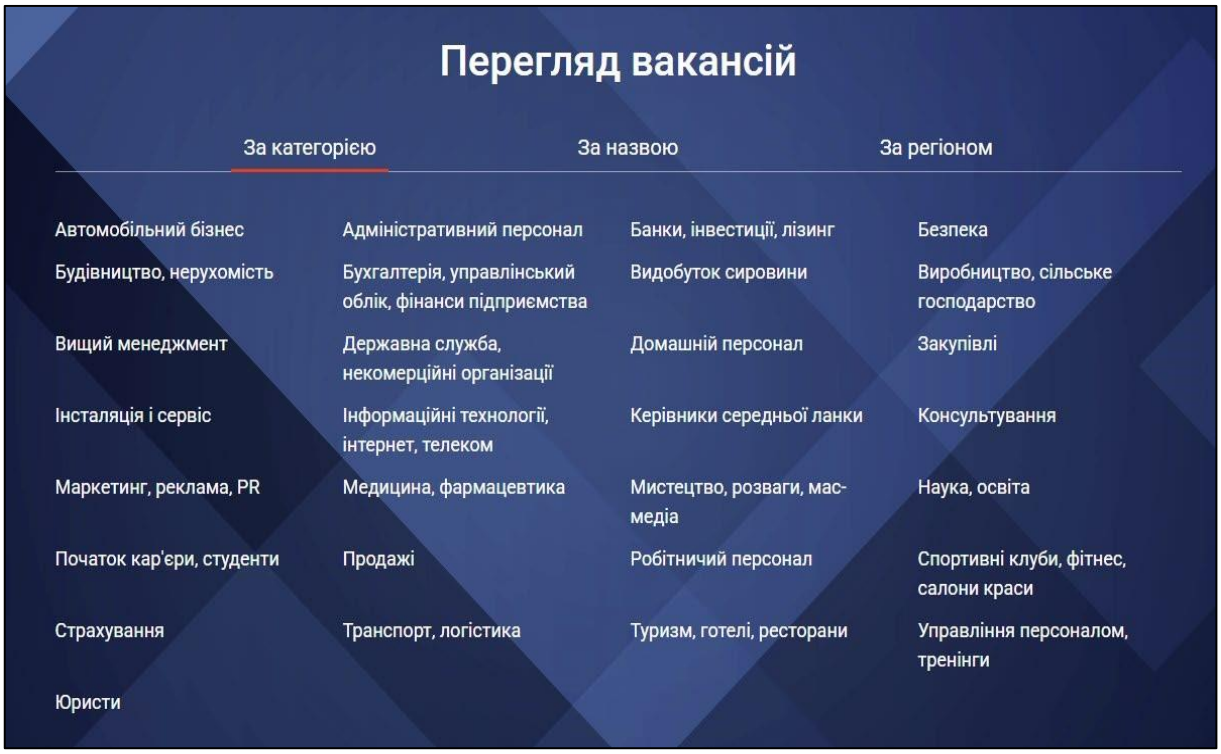


Рис. 1.6 Категоризація вакансій на платформі GRC.ua

1.5. Визначення вимог до web-застосунку

Враховуючи аналіз предметної області, проблем ручного управління та розгляд ATS, сформульовано вимоги до web-застосунку, який автоматизує процеси рекрутингу.

1.5.1. Функціональні вимоги

Система повинна мати такі можливості:

- Реєстрація, автентифікація користувачів;
- Контроль рівнів доступу відповідно до ролі (Рекрутер, Кандидат).
- Створення, редагування та видалення вакансій;
- Формування вичерпної інформації (назва, вимоги, обов’язки, умови, відповідальні особи);
- Можливість прикріплення файлів (тестові завдання, описи);
- Фільтрація та сортування списку вакансій за статусом, датою чи іншими критеріями;
- Зберігання даних кандидатів (ПІБ, контакти, резюме); - Пошук кандидатів за ключовими словами.

1.5.2. Нефункціональні вимоги

- Час завантаження сторінки не повинен перевищувати 2 секунд за нормальних умов;
- Підтримка 100 одночасних користувачів без затримок
- Доступність системи на рівні 99.7% часу;
- Дані регулярно резервуються і є можливість їх відновлення;
- Захищений від веб-вразливостей (XSS, CSRF, SQL-ін’єкції) відповідно до стандартів OWASP [9];
- Хешування паролів за допомогою алгоритмів (наприклад, bcrypt);

- Шифрування трафіку за протоколом HTTPS і контроль доступу на основі ролей;
- Можливість додавання нових модулів без значних змін у коді;
- Наявність сучасного, інтуїтивного, зрозумілого інтерфейсу з логічною навігацією;
- Адаптивний дизайн для коректного відображення на десктопах, планшетах і смартфонах;
- Код укомплектований чіткою структурою модулів;
- Спрощене оновлення та підтримка завдяки стандартам кодування;
- Коректна робота в останніх версіях браузерів (Chrome, Firefox, Safari, Edge).

Підводячи підсумки, можна сказати, що переваги автоматизованих процесів в контексті процесу найму встановлено на основі того, що запровадження webзастосунків може дуже прискорити процес рекрутингу, підвищити організацію процесів, реалізувати прозорість та покращити якість відбору кандидатів.

Аналіз популярних систем для управління кандидатами на роботу (ATS), таких як Work.ua, Robota.ua та GRC.ua, свідчить про те, що прогресивні системи надають різноманітний асортимент функціональності.

Вичерпні вимоги до web-застосунку, утворено на основі дослідження, охоплюючи функціональні та нефункціональні аспекти. Функціональні вимоги охоплюють керування користувачами та кандидатами, а також автоматизацію всього процесу найму шляхом автоматизації підбору персоналу, включаючи пропозиції роботи та співбесіди.

Нефункціональні вимоги забезпечують продуктивність і безпеку додатку, а також масштабованість і зручність для користувачів, що є ключовими факторами успіху програмного забезпечення в сучасному світі.

Ці вимоги ляжуть в основу проектування архітектури системи, вибору набору технічних інструментів, таких як NestJS, PostgreSQL і Next.js, і розробки програмного продукту. Впровадження додатку не лише оптимізує внутрішні процеси компанії, але й підвищує її конкурентоспроможність на ринку праці завдяки залученню найкращих талантів.

На наступних етапах необхідно буде приділити увагу детальній розробці користувацького інтерфейсу, елементів безпеки, пов'язаних з аутентифікацією користувачів та перевіркою паролів, а також тестуванню фактичної реалізації функціоналу в реальному середовищі.

2. ПРОЕКТУВАННЯ СИСТЕМИ ТА ВИБІР ТЕХНОЛОГІЙ

2.1. Обґрунтування вибору архітектури застосунку

2.1.1. Трирівнева клієнт-серверна архітектура (Three-Tier Architecture)

Створений web-застосунок спрямований на автоматизацію процесів рекрутингу, тому було обрано трирівневу клієнт-серверну архітектуру (Three-Tier Architecture).

Дана структура є стандартною для новітніх web-систем, так як вона ефективна, гнучка та модульна. Згадана архітектура відмежовує систему на три логічні рівні: представлення, бізнес-логіки та даних, що гарантує чітке розмежовування функцій і полегшує розробку, масштабування та підтримку[10]. Такий вибір обґрунтований взаємозв’язком архітектури функціональним вимогам і нефункціональним вимогам проєкту.

Рівень представлення (Frontend) оснащує взаємодію з користувачем через webбраузер. Інтерфейс користувача виконаний за допомогою Next.js, відтворює зовнішній вигляд системи для рекрутерів, надаючи можливість створювати вакансії, керувати кандидатами, координувати співбесіди та переглядати звітність аналітики. Frontend обробляє запити до сервера, формуючи дані у зрозумілому та зручному вигляді для користувача.

На рівні логіки можна ознайомитись із основною логікою застосунку, обробленими запитати від клієнтами. Рівень Backend виконує обчислення та взаємодію з базою даних. Дана серверна частина надає API для управління даними, пов’язаними з клієнтською частиною. Цей рівень гарантує автентифікацію, авторизацію та стандартизовану обробку даних.

Рівень даних (Database) відповідальний за зберігання та впорядкування даними. Це система управління базами даних (СУБД), яка зберігає інформацію стосовно користувачів, кандидатів, опублікованих вакансій та залишених заявок.

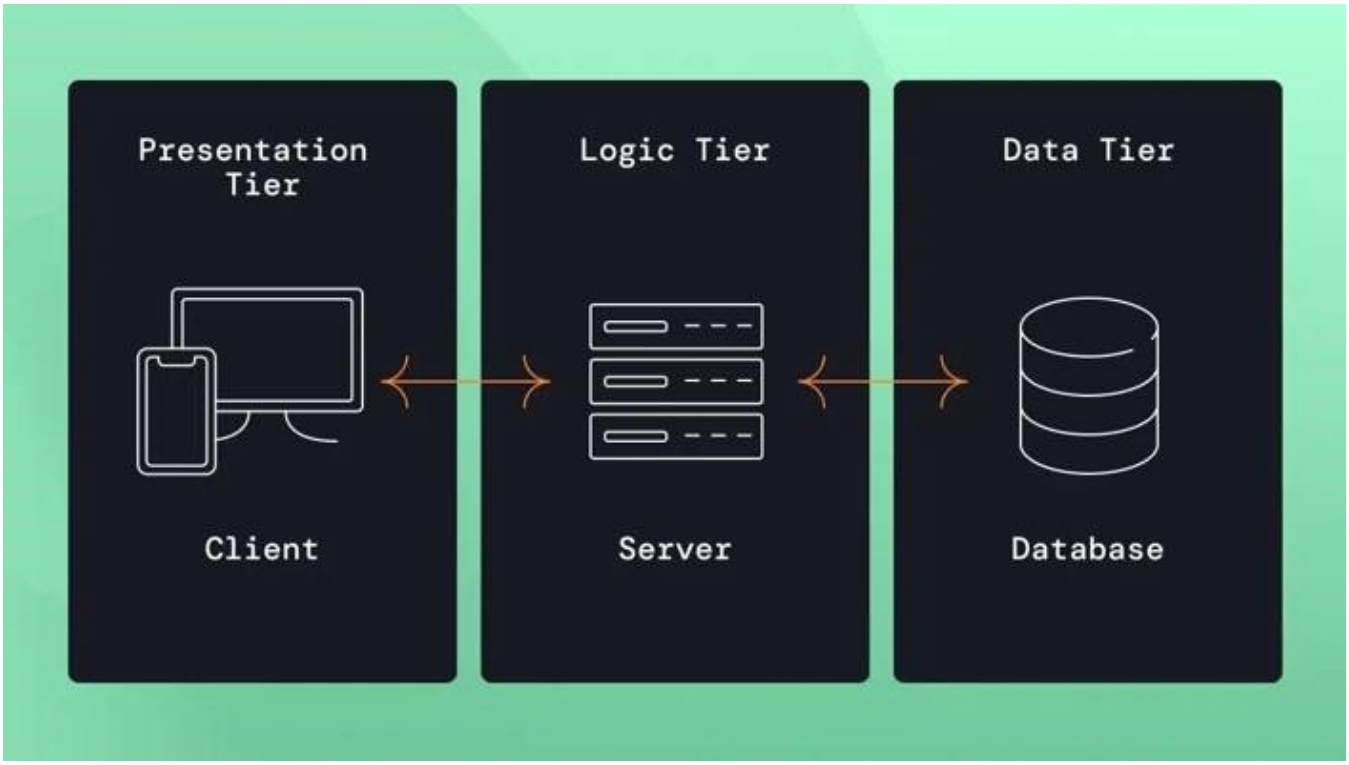


Рис. 2.1. Three-Tier Architecture

2.1.2. Переваги Three-Tier Architecture

Трирівнева архітектура має багато переваг, які відповідають потребам проєкту:

- Кожен рівень має чітку модульність та розділення відповідальностей, завдяки чому облегшується процес розробки та тестування;
- Система може залишатися стабільною, без значних змін, навіть при заміні технологій на будь-якому рівні. За потреби в майбутньому можна створити клієнтський інтерфейс у вигляді мобільного додатку з використанням того самого backend API;
- Backend є посередником, перевіряючи всі запити, тому рівень даних відокремлено від зовнішнього доступу;

- Backend API дозволяє використовувати систему з іншими клієнтами або ж модулями;
- Загальноприйнята архітектура спрощує інтеграцію розробників та впровадження існуючих інструментів.

2.1.3. Порівняння 1-рівневої, 2-рівневої та 3-рівневої архітектури 1-рівнева архітектура (Monolithic):

- Усі компоненти програми знаходяться в одного блоці або ж програмі; - Першочергова розробка простіша, особливо для невеликих застосувань; - Складно підтримувати та масштабувати, якщо зростає складність.

2-рівнева архітектура (Client-Server Applications):

- Застосунок поділяється на дві частини: клієнт та сервер, який як правило обробляє логіку та дані;
- Наявна певна модульність та покращена масштабність, якщо порівнювати з однорівневою архітектурою;
- Для більш складних систем можуть виникнути труднощі в масштабності, оскільки серверний рівень поєднує бізнес-логіку та доступ до даних, що потенційно створює вузьке місце.

3-рівнева архітектура:

- Розділяє застосунок на рівні представлення, застосунку (бізнес-логіка) та даних;
- Забезпечує найвищий рівень розділення, сприяючи масштабованості, зручності обслуговування та гнучкості;
- Зазвичай вимагає більших накладних витрат на розробку порівняно з простішими архітектурами[11].

2.2. Обґрунтування вибору технологічного стеку

Вибір технологічного стеку є важливою дією для визначення рівня продуктивності, стабільності, адаптивності та гнучкості розробки web-застосунку. Обраний стек відповідає критеріям новітніх і перевірених технологій, які підходять під вимоги проєкту та гарантують ефективну реалізацію всього функціоналу.

2.2.1. Технології серверної частини (Backend)

NestJS - це просунутий фреймворк для створення ефективних, надійних і масштабованих серверних додатків на Node.js. Він побудований з використанням TypeScript для забезпечення статичної типізації та покращення якості коду.

Даний фреймворк поєднує простоту із сильними інструментами, що чудово підходить для автоматизації рекрутингу[12].

Переваги:

- модульність (логічна структура модулів);
- підтримка мікросервісної архітектури; попередній контроль типізації, що зменшує відсоток помилок і полегшує технічне оновлення;
- неблокуюча модель Node.js означає, що він може опрацьовувати велику масив запитів, що важливо для кадрових систем, які стикаються з екстремальними навантаженнями.

PostgreSQL — це об’єктно-реляційна СУБД із відкритим кодом, якій характерні такі особливості як: надійність, гнучкість, підтримка складних запитів.

Дана система оптимізована для управління структурованими даними щодо кандидатів, вакансій та процесу рекрутингу[13].

Переваги:

- цілісність даних та надійність операцій, завдяки ACID-транзакціям;
- надійне управління великими масивами інформації з резервуванням копій;

- відкритий код, що дозволяє економити ресурси; широка база детальної документації та розширень.

2.2.2. Технології клієнтської частини (Frontend)

Next.js (React-фреймворк). Next.js являє собою фреймворк React, який функціонує як його основа. Фреймворк використовує React як основу для забезпечення гнучких можливостей рендерингу та оптимізації продуктивності, а також зручності розробки. Розробники вважають Next.js придатним для створення інтерактивних інтерфейсів систем рекрутингу[14].

Переваги:

- злагоджена візуалізація контенту на клієнтському рівні доступна завдяки підтримці SSR, SSG і CSR, яка дозволяє оптимізувати швидкість завантаження;
- впорядкована організація сторінок відповідно до файлового розташування, полегшує створення сторінок для кандидатів, публікації вакансій та аналітики;
- вбудовані механізми оптимізації автоматично вдосконалюють обробку зображення;
- наявний розширений набір бібліотек, таких як Redux для управління станом.

Tailwind CSS – це utility-first CSS-фреймворк, орієнтований на практичність, який дозволяє швидку розробку користувацьких інтерфейсів завдяки підходу без CSS, що пришвидшує розробку та забезпечує узгодженість дизайну[15].

Переваги:

- гнучке оформлення інтерфейсу дозволяє застосування стилів в розмітці для пришвидшення розробки, оптимізації налаштування дизайну та мінімізації зовнішніх стилів;

- неважке налаштування елементів дизайну завдяки конфігураційним налаштуванням;
- вбудована адаптивність дизайну під екрани різних розмірів.

Ant Design (AntD) – це популярна бібліотека UI-компонентів для React, яка орієнтована саме на корпоративні застосунки. Вона покращує взаємодію з користувачами і разом з цим підвищує продуктивність. Дана бібліотека буде ефективно працювати з складними користувацькими інтерфейсами для систем управління процесів найму працівників[16].

Переваги:

- бібліотека має готові компоненти, такі як таблиці, форми, що набагато пришвидшує розробку;
- усі компоненти створені відповідно до стандартів UX;
- зручна кастомізація кольорів та стилів для відповідності корпоративному стилю;
- інтеграція з TypeScript.

2.3. Концептуальне проектування бази даних

Проектування бази даних є головною стадією, яка формулює ефективність зберігання, обробки та пошуку даних. Для web-застосунку застосована реляційна модель у PostgreSQL, яка оптимізує структурованість управління рекрутинговими даними.

На концептуальному рівні окреслено ключове значення та їхні зв’язки:

- Користувач (Users). Відображає працівника компанії із визначенням ролі та доступом до системи;
- Вакансія (JobVacancies). Інформація про актуальну вакансію з урахуванням параметрів користувача;

- Кандидат (JobSeekerProfiles). Особа, яка подає заявку на відкриту вакансію;
- Роботодавець (EmployerProfiles). Особа, яка створює вакансії, а також переглядає заявки;
- Заявка (JobApplications). Зв’язок між кандидатом та вакансією.

2.3.1. Опис основних таблиць бази даних

Таблиця Users (Користувачі):

- id (PK, SERIAL): Унікальний ідентифікатор користувача;
- email (VARCHAR): Електронна адреса;
- passwordHash (VARCHAR): Хеш пароля;
- role (UserRole (ENUM)): Роль;
- isEmailVerified (BOOLEAN): Пошта підтверджена;
- isActive (BOOLEAN): Активний;
- lastLoginAt (DATETIME «nullable»): Останній вхід;
- createdAt (DATETIME): Створено;
- updatedAt (DATETIME): Оновлено.

Призначення: зберігання персональних даних.

Таблиця EmployerProfiles (Профілі роботодавців):

- id (PK, SERIAL): Унікальний ідентифікатор роботодавця;
- userId (INT «FK»): ID користувача;
- companyName (VARCHAR): Назва компанії;
- website (VARCHAR «nullable»): Вебсайт;
- description (TEXT «nullable»): Опис компанії;
- logoUrl (VARCHAR «nullable»): Логотип URL;
- companySize (VARCHAR «nullable»): Розмір компанії;

- industry (VARCHAR «nullable»): Індустрія;
- foundedYear (INT «nullable»): Рік заснування;
- createdAt (DATETIME): Створено;
- updatedAt (DATETIME): Оновлено.

Призначення: зберігання персональних даних роботодавців, а також їх логотипи компаній.

Таблиця JobSeekerProfiles (Профілі шукачів):

- id (PK, SERIAL): Унікальний ідентифікатор шукача;
- userId (INT «FK»): ID користувача;
- fullName (VARCHAR «nullable»): Повне ім'я;
- phone (VARCHAR «nullable»): Телефон;
- portfolioUrl (VARCHAR «nullable»): Портфоліо URL;
- experience (TEXT «nullable»): Досвід роботи;
- education (TEXT «nullable»): Освіта;
- skills (TEXT[] «nullable»): Навички, масив рядків;
- resumeUrl (VARCHAR «nullable»): Резюме URL;
- avatarUrl (VARCHAR «nullable»): Аватар URL;
- createdAt (DATETIME): Створено;
- updatedAt (DATETIME): Оновлено.

Призначення: зберігання персональних та контактних даних шукачів, а також їх резюме.

Таблиця JobVacancies (Вакансії):

- id (PK, SERIAL): Унікальний ідентифікатор вакансії;
- employerProfileId (INT «FK»): ID профілю роботодавця;
- title (VARCHAR): Назва вакансії;

- description (TEXT): Опис;
- location (VARCHAR): Локація;
- salary (VARCHAR «nullable»): Зарплатня;
- category (VARCHAR «nullable»): Категорія;
- jobType (TEXT[] «nullable»): Тип роботи, масив Enum JobType;
- onSite (BOOLEAN): В офісі;
- requirements (TEXT «nullable»): Вимоги;
- benefits (TEXT «nullable»): Переваги;
- createdAt (DATETIME): Створено;
- updatedAt (DATETIME): Оновлено.

Призначення: зберігання інформації про всі відкриті та закриті вакансії в компанії.

Таблиця JobApplications (Заявки на вакансії):

- id (PK, SERIAL): Унікальний ідентифікатор відгуку.
- vacancyId (INT «FK»): ID вакансії.
- jobSeekerProfileId (INT «FK»): ID профілю шукача.
- status (ApplicationStatus (ENUM)): Статус заявки.
- coverLetter (TEXT «nullable»): Супровідний лист.
- appliedAt (DATETIME): Подано.

Призначення: зв'язує кандидатів з вакансіями.

2.4. Проектування API

API (Application Programming Interface) серверної частини створено відповідно до принципів REST (Representational State Transfer). Обмін даними відбуватиметься

через стандартні HTTP-методи (GET, POST, PUT, PATCH, DELETE) та формату JSON для передачі даних.

Основні принципи проектування API:

- Використання іменників для ресурсів: URL-адреси будуть встановлювати відповідності ресурсів (наприклад, /vacancies, /candidates);
- Використання HTTP статус-кодів: Для оповіщення клієнта про результат операції (наприклад, 200 OK, 201 Created, 400 Bad Request, 401 Unauthorized, 404 Not Found, 500 Internal Server Error);
- Автоматизація API: Додано автоматичний префікс api до всіх енпоінтів.
- Пагінація та фільтрація для списків ресурсів: Для ефективної роботи з великими масивом даних;
- Автентифікація та авторизація: Доступ до ресурсів буде захищено (наприклад, за допомогою JWT-токенів).

Приклади основних ендпоінтів:

Користувачі та Автентифікація:

- POST /auth/register – Реєстрація нового користувача;
- POST /auth/login – Вхід користувача, отримання JWT токена;
- GET /users – Отримання інформації про всіх користувачів.

Роботодавці:

- GET /profile/employer/me - Отримати профіль поточного роботодавця;
- PUT /profile/employer – Оновити профіль роботодавця;
  POST /profile/employer/logo-upload-ur1 – Створити посилання на логотип (image) на AWS S3;
- PUT /profile/employer/logo – Оновити профіль роботодавця (посилання на логотип).

Кандидати:

- GET /profile/job-seeker/me – Отримати профіль поточного шукача;
- PUT /profile/job-seeker – Оновити профіль шукача;
- POST /profile/job-seeker/resume-upload-ur1 – Створити посилання на резюме (pdf) на AWS S3;
- PUT /profile/job-seeker/resume – Оновити профіль шукача (посилання на резюме).

Вакансії:

- GET /vacancies - Отримати список всіх вакансій (з фільтрацією та пагінацією);
- POST /vacancies - Створити нову вакансію;
- GET /vacancies/my - Отримати вакансії створені поточним роботодавцем;
- GET /vacancies/{id} - Отримати детальну інформацію про вакансію за ID; - PUT /vacancies/{id} - Оновити інформацію про вакансію; - DELETE /vacancies/{id} - Видалити вакансію.

Відгуки/Заявки (Applications):

- POST /applications – Створити новий відгук кандидата на вакансію;
- GET /applications/my – Отримати відгуки поточного шукача;
- GET /applications/{vacancyId}/applications – Отримати список відгуків на конкретну вакансію;
- PATCH /applications/vacancy/{vacancyId}/{appId} – Змінити статус конкретного відгуку.

2.5. Проектування інтерфейсу користувача (UI/UX)

2.5.1. Мета та значення UI/UX дизайну

Проектування інтерфейсу користувача та досвіду користувача є важливою стадією розробки web-застосунку для автоматизації процесів рекрутингу, так як продуктивність роботи HR-відділу та позитивний досвід кандидатів залежить від якості взаємодії.

UI формує графічну складову та інтерактивність, а UX спрямований на створенні плавного та логічного користувацького досвіду. Ціллю є забезпечення інтуїтивного, візуально привабливого та простого у використанні рішення, що оптимізує адаптацію рекрутерів і покращити досвід кандидатів.

Згідно з принципами usability Якоба Нільсена, інтерфейс має бути ефективним, результативним і задовольняючим для користувачів[17].

2.5.2. Цільові користувачі

Для гарантування відповідності інтерфейсу вимогам користувачів визначено дві основні категорії цільової категорії:

- Рекрутери. HR-фахівці є основними користувачами системи, які відповідають за повний цикл процесу найму співробітників – від формування та публікації вакансії до обробки резюме, планування співбесід та затвердження кандидатів. Рекрутери вимагають оперативного доступу до бази вакансій та кандидатів, а також засоби фільтрації, пошуку, аналітики та автоматизації рутинних процесів, таких як надсилання сповіщень та планування співбесід. Для них важлива чітка навігація, проста візуальна ієрархія та можливість роботи з масштабними обсягами інформації;

  Кандидати. Користувачі, які використовують портал для подання резюме та отримання інформації про запрошення на співбесіду. Для майбутніх працівників визначальним є простота у використанні, зрозумілий інтерфейс,

швидкий зворотній зв’язок для забезпечення позитивного досвіду та підтримки репутації компанії як роботодавця.

Раціоналізація процесів взаємодії користувачів та рекрутерів визначило вимоги до інтерфейсу: логічна навігація, простота використання, адаптивність та мінімізація можливих помилок.

2.5.3. Основні принципи UI/UX дизайну

Дизайн інтерфейсу базується на сучасних принципах UI/UX, які забезпечують високу ефективність і відповідність стандартам usability[18]. Основні принципи включають:

- простота та інтуїтивність. Продумана архітектура UI мінімізує потребу в додатковому навчанні, що дозволяє користувачам швидко адаптуватися.
- послідовність системи стилізації та функціональності. Цілісність дизайну гарантують узгоджений вигляд та єдині кольори, шрифти та стандартизовані елементи. Ant Design надає набір готових компонентів, що підтримує необхідні стандарти інтерфейсу та функціональності.
- ефективність. Автоматизація та продуманий дизайн інтерфейсу спрощує рутинні завдання.
- емоційна привабливість. Дизайн враховує всі естетичні аспекти, такі як гармонійна кольорова палітра та новітні шрифти, що залишає позитивну думку для кандидатів.

2.5.4. Ключові інтерфейси застосунку

Основні екрани/розділи застосунку:

- список вакансій: Таблиця або картки з усіма вакансіями, можливість фільтрації (за статусом, відділом, рекрутером), пошуку, створення нової вакансії;

- сторінка вакансії: Детальна інформація про вакансію, список кандидатів на цю вакансію. Можливість редагувати вакансію;
- сторінка кандидата: Детальна інформація про кандидата (резюме, контакти).

Використання Ant Design надасть готові компоненти (таблиці, форми, модальні вікна, кнопки, навігаційні елементи), що значно прискорить розробку UI та забезпечить професійний вигляд. Tailwind CSS буде використовуватися для кастомної стилізації, створення унікальних лейаутів та забезпечення адаптивності.

Дійшовши до висновку, можна виділити те, що розробка web-застосунку для автоматизації процесів найму засноване на трирівневій клієнт-серверній архітектурі (Three-Tier Architecture).

Архітектурне рішення сприяє модульному принципу, можливістю масштабування та чітко організованому функціоналу між представленням, бізнеслогікою та даними. Ця модель забезпечує стабільність та гнучкість системи, відповідаючи функціональним і нефункціональним вимогам проєкту.

Для реалізації було обрано технологічний стек, який включає сучасні перевірені інструменти: NestJS разом з PostgreSQL для серверної функціональності та Next.js разом з Ant Design і Tailwind CSS для розробки фронтенду, що забезпечує чуйний користувацький інтерфейс та елегантний дизайн.

Структурне моделювання в PostgreSQL забезпечує чітку організацію даних для профілів користувачів, списків вакансій, кандидатів та заявок, що спричиняє до ефективного зберігання та обробки інформації.

Дизайн REST API, що використовує стандартні методи HTTP та формат JSON, забезпечує прямий зв'язок між клієнтською та серверною системами та гарантує безпеку даних за допомогою автентифікації та авторизації.

Дизайн інтерфейсу та користувацького досвіду орієнтований на потреби двох основних груп користувачів: рекрутерів та шукачів роботи.

Основні екрани програми розроблені таким чином, щоб підтримувати інтуїтивно зрозумілу навігацію, забезпечувати швидкий доступ до даних і мінімізувати помилки користувачів, що полегшує процес підбору персоналу та покращує користувацький досвід.

Обрана архітектура системи разом зі стеком технологій, базою даних та методами проектування інтерфейсу забезпечує міцну основу для створення ефективного веб-додатку, що відповідає сучасним стандартам та вимогам користувачів.

3. РОЗРОБКА ТА РЕАЛІЗАЦІЯ WEB-ЗАСТОСУНКУ

3.1. Налаштування середовища розробки

Налаштування середовища розробки є ключовим етапом створення webзастосунку для автоматизації процесів найму, так як воно гарантує ефективну, постійну та інтегровану платформу для повного циклу розробки – від тестування до розгортання. Середовище охоплює операційні системи, інструменти коду, управління залежностями, бази даних, тестування та контейнери.

Використані технології та інструменти розробки забезпечують відповідність вимогам проекту щодо продуктивності, масштабованості та командної взаємодії, а також забезпечують підтримку обраного технологічного стеку (NestJS, Next.js, PostgreSQL).

3.1.1. Операційна система Linux (Manjaro)

Manjaro Linux визначено за основну операційну систему для розробки через її високу надійність в роботі, масштабність та інноваційним підходам до керування. Manjaro гарантує швидкий доступ до актуальних версій програмного забезпечення, що є дуже важливим для роботи з сучасними фреймворками, наприклад з NestJS і Next.js. Також можна сказати, що Linux-середовище відповідає стандартам розробки серверних застосунків, що оптимізує процес інтеграції з Docker і впровадженню у хмарному середовищі.

Переваги Linux (Manjaro):

- Зручність для користувача. Manjaro Linux було розроблено з чіткою метою додати графічні інструменти до сильних сторін Arch Linux і таким чином створити привабливу альтернативу для багатьох користувачів. Незважаючи на те, що операційна система вже давно переросла цей підхід, його все ще

можна знайти у Manjaro Linux. Дистрибутив також робить його дуже простим для початківців і, таким чином, полегшує перехід з Windows.

- Змінний реліз. Втім, Manjaro Linux аж ніяк не є чистим Linux для початківців. Завдяки моделі випуску релізів, дистрибутив також чітко орієнтований на користувачів, які завжди хочуть бути в курсі останніх подій. Користувачі можуть самі вирішувати, чи хочуть вони використовувати стабільні версії, чи лише частково протестовані нові можливості.
- Середовище робочого столу. Користувачі можуть вибирати, коли йдеться про робочі столи. Наявні також версії для різних середовищ, які також відрізняються за своїм програмним забезпеченням і темами. Це ще одна причина, чому Manjaro Linux приваблює багатьох різних користувачів.
- Продуктивність. За красивою оболонкою ховається потужне ядро. Економна операційна система дуже швидка, працює безпечно і безперебійно на більшості комп'ютерів: Manjaro Linux - це надійний дистрибутив.
- Програмне забезпечення. Попередньо встановлене програмне забезпечення у кожному випуску гарантує, що лише кілька програм потрібно встановити додатково. Для додаткового програмного забезпечення доступний менеджер пакунків Pacman з власним інтерфейсом. Він також може бути використаний для встановлення пакунків зі сховища користувачів Arch (AUR).
- Спільнота. Спільнота Manjaro Linux має репутацію корисної та добре організованої. По-перше, є швидка допомога для вирішення питань і проблем, а по-друге, кілька документацій доступні багатьма мовами.

3.1.2. Середовище виконання JavaScript

Спершу варто розглянути JavaScript загалом. Тож, JavaScript (JS) - це легка інтерпретована (або скомпільована просто в часі) мова програмування з передовими функціями.

Дана мова визнана як мова сценаріїв для веб-сторінок, багато небраузерних середовищ також використовують її, наприклад, Node.js, Apache CouchDB та Adobe Acrobat. JavaScript - це багатопарадигмальна, однопотокова, динамічна мова, що базується на прототипах, підтримує об'єктно-орієнтовані, імперативні та декларативні (наприклад, функціональне програмування) стилі.

Динамічні можливості JavaScript включають конструювання об'єктів під час виконання, списки параметрів змінних, змінні функцій, динамічне створення сценаріїв, самоаналіз об'єктів та відновлення вихідного коду[19].

Розробка фреймворків JavaScript, що складаються з бібліотек коду JavaScript, дозволяє розробникам використовувати попередньо написаний код JavaScript у своїх проектах. Це заощаджує їм час і зусилля, оскільки вони не потребують написання програмних функцій з нуля. Кожен JavaScript-фреймворк має функції, спрямовані на спрощення процесу розробки та налагодження. Наприклад, фронтенд-фреймворки JavaScript, такі як jQuery та ReactJS, підвищують ефективність дизайну. Вони дозволяють розробникам повторно використовувати та оновлювати компоненти коду, не впливаючи один на одного, ні на функції, ні на значення. Тим часом, фреймворки для розробки мобільних додатків, такі як Cordova та Titanium , дозволяють створювати нативні або гібридні додатки .

Реалізація коду JavaScript у Node.js також відіграє важливу роль у веб-розробці. Node.js може зменшити час відгуку сервера завдяки своїй однопотоковій природі та неблокуючій архітектурі, а також уникнути затримок. Node.js також достатньо легкий, щоб служити масштабованим інструментом для мікросервісів, дозволяючи розробляти єдиний застосунок, що складається з невеликих сервісів з окремими процесами[20].

Для запуску серверного (NestJS) та клієнтського (Next.js) застосування потрібне середовище виконання JavaScript. Версію LTS 20.x було вибрано через її стабільність та сумісність з сучасними функціями ECMAScript та фреймворками. Модель

вводувиводу є асинхронною для оброблення великої кількості одночасних запитів та використовується для системи рекрутингу для навантажень.

3.1.3. Менеджер пакетів

npm— це стандартний менеджер пакетів для Node.js. Залежності – це готовий набір бібліотек та пакетів, необхідних для функціонування програми на Node.js.

npm складається з трьох окремих компонентів:

- Вебсайт
- інтерфейс командного рядка (CLI)
- реєстр

npm можна використовувати для:

- Налаштовувати пакунки коду для ваших програм або вбудовувати пакунки як є.
- Завантажуйте окремі інструменти, які можна використовувати одразу.
- Запускайте пакунки без завантаження за допомогою npx .
- Діліться кодом з будь-яким користувачем npm будь-де.
- Обмежуйте доступ до коду для певних розробників.
- Створюйте організації для координації підтримки пакунків, кодування та розробників.
- Створюйте віртуальні команди за допомогою організацій.
- Керуйте декількома версіями коду та залежностями коду.
- Легко оновлюйте додатки, одночасно оновлюючи основний код.
- Знаходьте кілька способів розв'язання однієї і тієї ж задачі.
- Знаходьте інших розробників, які працюють над схожими проблемами та проектами[21].

3.1.4. Інтегроване середовище розробки та система контролю версій

WebStorm – це комплексне програмне середовище (IDE) із вбудованими засобами для налагодження, тестування та управління версіями. Дані функції є вкрай вигідним для великих та складніших проектів. WebStorm включає розширені функції автодоповнення коду, перепроектування структур та виявлення помилок, які можуть зекономити час розробників і збільшити продуктивність. Ще однією поширеною ситуацією рефакторингу є перейменування файлу. Webstorm має призначення безпечного перейменування, де він збирає всі екземпляри файлу та оптимізує код імпортовані дані. Ще однією потужною функцією продуктивності є функція пошуку будь-де.

Git (з репозиторієм на GitHub) – це надійна платформа для контролю версіями, яка гарантує відстеження змін, розподіл гілок та командну співпрацю. Платформа GitHub забезпечує зручне управління репозиторієм, інтеграцію з CI/CD та підтримку pull requests, що робить її оптимальним рішенням для розробників.

WebStorm має вбудовану інтеграцію з GitHub, що дозволяє керувати гілками та коммітами безпосередньо з IDE. Можна переглядати, створювати та об’єднувати інтуїтивну роботу із запитати на злиття через зручний UI.

3.2. Реалізація серверної частини (Backend) на NestJS

Серверна частина застосунку, що відповідає за бізнес-логіку та взаємодію з базою даних, була реалізована на фреймворку NestJS. NestJS дозволяє створювати масштабовані та легко підтримуваний код, що є важливим для системи автоматизації рекрутингу.

3.2.1. Структура проекту NestJS

Проєкт **NestJS** було згенеровано за допомогою NestJS CLI та має типову модульну структуру. Основна директорія з вихідним кодом — src, де зберігаються всі

основні файли та модулі. Файл main.ts є вхідною точкою додатку та відповідає за ініціалізацію NestJS-додатку. У файлі app.module.ts знаходиться кореневий модуль застосунку, який імпортує інші функціональні модулі. Базовий контролер розташовано у файлі app.controller.ts, а логіка обробки — у app.service.ts. Директорія config використовується для зберігання конфігураційних файлів, таких як підключення до бази даних або налаштування JWT-секретів.

Модуль auth реалізує механізми автентифікації та авторизації, включає файли auth.module.ts, auth.controller.ts, auth.service.ts, а також підкаталоги guards (файли jwt.guard.ts, roles.guard.ts) та strategy (файл jwt.strategy.ts).

Модуль user відповідає за управління користувачами та включає user.module.ts, user.controller.ts, user.service.ts, а також сутність entities/user.entity.ts, реалізовану за допомогою TypeORM.

Модуль job-vacancy забезпечує функціональність управління вакансіями і містить job-vacancy.module.ts, job-vacancy.controller.ts та job-vacancy.service.ts.

Для управління профілями кандидатів створено модуль job-seeker-profile, який включає job-seeker-profile.module.ts, job-seeker-profile.controller.ts і job-seekerprofile.service.ts.

Окремий модуль job-application відповідає за управління заявками (відгуками на вакансії) та складається з файлів job-application.module.ts, job-application.controller.ts і job-application.service.ts.

3.2.2. Реалізація моделей даних та взаємодія з PostgreSQL (TypeORM)

Для взаємодії з базою даних PostgreSQL було обрано TypeORM. Були створені класи-сутності (entities) для кожної таблиці бази даних. Ці сутності декоруються за допомогою декораторів TypeORM (@Entity(), @PrimaryGeneratedColumn(), @Column(), @ManyToOne(), @OneToMany()) для опису структури таблиць та зв'язків між ними.

Налаштовано підключення до PostgreSQL в app.module.ts або окремому модулі бази даних через TypeOrmModule.forRoot({...}).

Для роботи з даними в сервісах використовуються репозиторії TypeORM (Repository<Entity>), які надають методи для CRUD-операцій та побудови складних запитів.

```
export enum JobType {
   FULL_TIME = 'Full Time',
   PART_TIME = 'Part Time',
   CONTRACT = 'Contract',
   INTERNSHIP = 'Internship',
   REMOTE = 'Remote',
   EASY_APPLY = 'Easy Apply',
   MULTIPLE_CANDIDATE = 'Multiple Candidate'
}

@Entity({ name: 'job_vacancies' }) export
class JobVacancy {
   @PrimaryGeneratedColumn()
   @ApiProperty({  description:  'Unique  ID'  })
id: number;

   @ManyToOne(() => EmployerProfile, (employer) => employer.vacancies, {
eager: true,        nullable: false,        onDelete: 'CASCADE'
   })
   @JoinColumn({ name: 'employerProfileId' })
@ApiProperty({ type: () => EmployerProfile })    employer:
EmployerProfile;
```

```
  @Column()
  @ApiProperty({ example: 'Senior Frontend Developer' })
title: string;

  @Column({ type: 'text' })
  @ApiProperty({ example: 'Develop amazing user interfaces...' })
description: string;

  @OneToMany(() => JobApplication, app => app.vacancy)
applications:JobApplication[];

  @Column()
  @ApiProperty({ example: 'Porto, Portugal' })
location: string;

  @Column({ nullable: true })
  @ApiPropertyOptional({ example: '$80,000 - $120,000' })
salary?: string;

  @Column({ nullable: true })
  @ApiPropertyOptional({ example: 'Technology' })
category?: string;

  @Column({
    type: 'simple-array',
nullable: true,  })
  @ApiPropertyOptional({ type: [String], enum: JobType, example:
```

```
[JobType.FULL_TIME, JobType.REMOTE] })
    jobType?: JobType[];

    @Column({ default: true })
    @ApiProperty({ example: true, description: 'True if On Site, False if Remote' })
  onSite: boolean;

    @Column({ type: 'text', nullable: true })
    @ApiPropertyOptional()
  requirements?: string;

    @Column({ type: 'text', nullable: true })
    @ApiPropertyOptional()
  benefits?: string;

    @CreateDateColumn()
  createdAt: Date;

    @UpdateDateColumn()
  updatedAt: Date;
  }
```

3.2.3. Розробка основних модулів

Кожен функціональний блок системи було реалізовано як окремий модуль NestJS.

Модуль AuthModule:

- Реалізовано реєстрацію (/auth/register) та логін (/auth/login) користувачів.

- Для автентифікації використовується механізм JWT (JSON Web Tokens). При успішному логіні генерується JWT, який клієнт зберігає та надсилає з кожним наступним запитом.
- Використано бібліотеку @nestjs/jwt та @nestjs/passport з passport-jwt стратегією для валідації токенів.
- Реалізовано JwtAuthGuard для захисту ендпоінтів, що потребують автентифікації.
- Реалізовано хешування паролів за допомогою бібліотеки argon2.

Модуль UserModule:

- Надає функціонал для управління користувачами.
- Визначає ролі користувачів (наприклад, через зв'язок з таблицею Roles або поле в User).

Модуль JobVacanciesModule:

- Реалізує CRUD-операції для вакансій.
- Валідація вхідних даних для створення/оновлення вакансій за допомогою DTO (Data Transfer Objects) та class-validator, class-transformer.
- Логіка для фільтрації та пагінації списку вакансій.
- Зв'язок з користувачами (хто створив, хто наймаючий менеджер).

Модуль JobSeekerModule:

- Реалізує CRUD-операції для кандидатів.
- Логіка для уникнення дублікатів кандидатів (за email).

Модуль JobApplicationModule:

- Керує відгуками кандидатів на вакансії.

3.2.4. Реалізація REST API ендпоінтів

Для кожного модуля були створені контролери (*.controller.ts), які визначають маршрути (ендпоінти) та обробляють HTTP-запити.

Використання декораторів NestJS (@Controller(), @Get(), @Post(), @Put(), @Delete(), @Param(), @Body(), @Query(), @UseGuards()) для визначення маршрутів, отримання параметрів запиту та захисту ендпоінтів.

Впровадження сервісів в контролери через Dependency Injection.

Обробка помилок та повернення відповідних HTTP статус-кодів.

Використання Swagger для автоматичної генерації документації API. Модуль @nestjs/swagger дозволяє документувати ендпоінти, моделі DTO та відповіді.

```
@Post()
@ApiBearerAuth()
@UseGuards(JwtAuthGuard, RolesGuard)
@Roles(UserRole.EMPLOYER)
@ApiOperation({ summary: 'Create a new job vacancy (Employer only)' }) async
create(@Req() req, @Body() dto: CreateJobVacancyDto) {
  if (!req.user || !req.user.id) throw new ForbiddenException('User ID not found');
return this.vacancyService.create(req.user.id, dto);  }

@Put(':id')
@ApiBearerAuth()
@UseGuards(JwtAuthGuard, RolesGuard)
@Roles(UserRole.EMPLOYER)
@ApiOperation({ summary: 'Update an existing job vacancy (Employer only)' })
async update(
    @Req() req,
```

```
    @Param('id', ParseIntPipe) id: number,
    @Body() dto: Partial<CreateJobVacancyDto>,
  ) {
    if (!req.user || !req.user.id) throw new ForbiddenException('User ID not found');
  return this.vacancyService.update(req.user.id, id, dto);   }
```

3.3. Реалізація клієнтської частини (Frontend) на Next.js

Клієнтська частина застосунку, що забезпечує інтерфейс користувача, була розроблена на фреймворку Next.js з використанням бібліотек React, Tailwind CSS та Ant Design.

3.3.1. Структура проекту Next.js

Проєкт **Next.js** було створено за допомогою утиліти create-next-app і він має типову модульну структуру, орієнтовану на масштабовану розробку з використанням сучасних підходів до маршрутизації та компонентного підходу. Основна директорія app відповідає за маршрутизацію — кожен файл у цій директорії з розширенням .ts або .tsx автоматично стає окремим маршрутом. У директорії employer-dashboard розміщені сторінки, що стосуються панелі керування роботодавця, зокрема піддиректорія profile для сторінки профілю роботодавця, vacancies для управління вакансіями, динамічний маршрут [vacancyId] для перегляду конкретної вакансії та сторінка create для створення нової. Основна сторінка списку вакансій представлена у файлі page.tsx. У директорії find-jobs знаходиться сторінка пошуку вакансій для кандидатів, також реалізована у page.tsx. Аналогічно, job-seeker-dashboard містить сторінку профілю кандидата та головну сторінку дашборду. Окремі директорії login і sign-up відповідають за автентифікацію та реєстрацію користувачів, відповідно. У корені проєкту також присутні файли favicon.ico (іконка сайту), globals.css (глобальні стилі з підтримкою Tailwind CSS), layout.tsx (глобальний лейаут з навігаційною

панеллю, футером та перемикачем тем) та page.tsx, який слугує вхідною сторінкою застосунку.

Директорія components містить перевикористовувані React-компоненти, структуровані за тематичними піддиректоріями. У jobs розміщені компоненти JobCard.tsx (картка вакансії), JobDetails.tsx (деталі вакансії) та JobFilters.tsx (фільтрація вакансій за різними параметрами). У layout зберігаються компоненти інтерфейсу, такі як Footer.tsx, Header.tsx і Preloader.tsx. Секції лендінг-сторінки згруповані в sections, серед яких — AboutSection.tsx, CategoriesSection.tsx, HeroSection.tsx, HowItWorksSection.tsx і StatsSection.tsx. Компоненти для роботи з темами знаходяться в theme, зокрема ThemeApplication.tsx та ThemeSwitcher.tsx. У ui містяться кастомні інтерфейсні елементи, наприклад, NeonButton.tsx.

Директорія lib містить допоміжні функції, зокрема файл AntdRegistry.tsx для коректної інтеграції бібліотеки Ant Design. Управління глобальним станом застосунку здійснюється в store через файли authStore.ts, jobStore.ts та themeStore.ts, які відповідають за автентифікацію, обробку даних про вакансії та теми відповідно. У types визначаються користувацькі TypeScript-типи, що забезпечує типобезпечність на всіх рівнях проєкту. Нарешті, файл middleware.ts містить налаштування для обробки запитів, таких як автентифікація або перенаправлення, що підвищує безпеку та зручність навігації в застосунку.

3.3.2. Налаштування Tailwind CSS та Ant Design

Tailwind CSS

- Цей файл створюється як залежність.
- Файл конфігурації tailwind.config.js створюється для налаштування теми (кольори, шрифти, точки) та підключення плагінів.
- Створює файл postcss.config.js.
- Імпортує базові стилі Tailwind у глобальний CSS-файл (наприклад, styles/globals.css).

- Tailwind використовується шляхом додавання допоміжних класів безпосередньо до елементів JSX.

Ant Design:

- Встановлено як залежність (antd).
- Компоненти AntD (наприклад, Button, Table, Form, Modal, Menu) імпортувалися та використовувалися безпосередньо в React-компонентах.

3.3.3. Розробка основних компонентів інтерфейсу

Були розроблені як сторінкові компоненти (в app/), так і пере-використовувані UI-компоненти (в components/).

Layout-компонент: Загальна структура сторінки (хедер, сайдбар з навігацією, футер), яка обгортає вміст кожної сторінки.

Компоненти для відображення даних: VacancyCard, VacancyTable.

Форми: Компоненти форм для створення/редагування вакансій, кандидатів, використовуючи компоненти Form, Input, Select, DatePicker з Ant Design та бібліотеку для управління формами (react-hook-form).

Модальні вікна: Для підтвердження дій, відображення додаткової інформації.

3.3.4. Реалізація ключових сторінок

Список вакансій (/find-jobs): Відображення вакансій з пагінацією, сортуванням та фільтрацією. Кнопки для переходу до деталей вакансії, редагування, створення нової.

Сторінка вакансії (/employer-dashboard/vacancies/[vacancyId]/applicants): Відображення детальної інформації про відгуки на вакансію.

Сторінки автентифікації (/auth/login, /auth/sign-up): Форми для входу та реєстрації, валідація полів, обробка відповідей від сервера, збереження JWT токена.

Реалізація захищених маршрутів (redirect на логін, якщо користувач не автентифікований).

3.4. Опис ключових функціональних можливостей застосунку

Розроблений web-застосунок надає наступний ключовий функціонал:

- Створення та управління вакансіями: Рекрутери можуть створювати нові вакансії, вказуючи назву, детальний опис, вимоги, відповідальних осіб. Вакансії можна редагувати, архівувати або видаляти. Система відображає список всіх вакансій зі статусами.
- Рольова модель доступу: Рекрутер та Шукач мають різні права доступу до функціоналу та даних системи.
- Пошук та фільтрація: Реалізовано можливості пошуку та фільтрації вакансій за різними критеріями.

Розробка web-застосунку для автоматизації рекрутингу була успішно реалізована завдяки створенню зручного для користувача середовища на базі Manjaro Linux, Node.js LTS 20.x, npm та інтегрованого середовища WebStorm з контролем версій через Git та GitHub. Серверна частина на основі NestJS включає модульну структуру з TypeORM для взаємодії з PostgreSQL, JWT-аутентифікацію, управління вакансіями, кандидатами та заявками, а також REST API з документацією Swagger.

Побудована на Next.js з використанням Tailwind CSS та Ant Design, клієнтська частина надає інтуїтивно зрозумілий інтерфейс з динамічними шляхами, елементами відображення даних та форм, а також сторінками безпечної автентифікації. Розробка основної функціональності, такої як створення завдань, рольовий доступ та фільтрація, підтвердила, що система готова до тестування та розгортання.

4 ТЕСТУВАННЯ ТА РОЗГОРТАННЯ ЗАСТОСУНКУ

Застосунок тестувався вручну, послідовно тестуючи всі кнопки та сторінки для перевірки коректності інтерфейсу, навігації та взаємодії з серверною частиною. Такий підхід дозволив виявити потенційні помилки в UI/UX і забезпечити стабільність роботи ключових модулів, таких як аутентифікація, управління вакансіями, дані кандидатів і додаток. Тестування включає перевірку правильності відображення фільтрів, форм обробки, динамічних шляхів і даних з бекенд API. Бекенд також використовує Postman для тестування всіх кінцевих точок API, включаючи методи GET, POST, PUT, PATCH і DELETE, а також валідацію відповідей, валідацію токенів JWT і обробку помилок. Після завершення тестування мануалу та API ми були готові до розробки застосунку.

4.1. Стратегія тестування

Стратегія тестування базувалася на поєднанні ручного тестування на стороні клієнта та автоматизованого тестування на стороні сервера. Етап ручного тестування був зосереджений на перевірці користувацького інтерфейсу (UI) та користувацького досвіду (UX). Для цього були протестовані всі функціональні елементи, такі як кнопки, форми, фільтри та навігаційні посилання, щоб переконатися, що вони працюють належним чином. Особливу увагу було приділено модулям аутентифікації (вхід, реєстрація), управлінню оголошеннями про вакансії та профілями пошукачів, а також коректному відображенню даних на сторінках з динамічними шляхами (наприклад, вакансії/[vacancyId]).

На стороні сервера використовувався Postman як основний інструмент тестування API. Була створена серія тестових запитів для всіх кінцевих точок, включаючи аутентифікацію (POST /auth/login, POST /auth/register), управління

вакансіями (GET /vacancies, POST /vacancies), файли (GET /profile/employer/me, PUT /profile/job-seeker) і програми (POST /applications, PATCH /applications/vacancy/jobseeker). Тести охоплюють стан додатку (GET /profile/employer/me, PUT /profile/jobseeker) і додатків (POST /applications, PATCH /applications/vacancy/{vacancyId}/{appId}). Тести включають перевірку коректності кодів статусу (наприклад, 200 OK, 401 несанкціоновано), форматів відповідей (JSON) і обробку помилок у разі невірних даних або несанкціонованого доступу. Для автентифікації токенів JWT виконуються окремі перевірки, включаючи валідність токенів, термін дії та доступ до захищених кінцевих точок на основі ролі користувача.

Така стратегія дозволила гарантувати стабільність роботи застосунку, виявити та усунути помилки в логіці API та забезпечити відповідність інтерфейсу вимогам юзабіліті. Ручне тестування користувацького інтерфейсу доповнює інструментальне тестування API, забезпечуючи комплексний підхід до тестування системи перед розгортанням.

Тестування системи автоматизації рекрутингу передбачає ручне маніпулювання всіма кнопками та сторінками для перевірки надійності інтерфейсу та модулів, а також перевірку кінцевих точок та автентифікації внутрішніх API за допомогою тестування Postman. Ця стратегія тестування забезпечує повну перевірку функціональності, точності реакції та обробки помилок, щоб підготувати систему до успішного розгортання.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було проведено вичерпний аналіз найсучасніших підходів до автоматизації процесів найму, що включав огляд актуальних досліджень, методик та технологій у сфері HR. В рамках аналізу було досліджено як звичні методи найму (наприклад, ручний відбір резюме та співбесіди), так інноваційні рішення, зокрема застосування штучного інтелекту для оцінки кандидатів, автоматизовані системи відстеження заявників (ATS), платформи для управління талантами та інтеграція з соціальними мережами для полегшення реєстрації. Це дало змогу ясно визначити ключові вимоги до функціональності та структури web-застосунку для HR-агентства, такі як потреба в інтуїтивному інтерфейсі для обох типів користувачів, гнучка система фільтрування вакансій, підтримка безпечного обміну даними між сторонами, а також забезпечення високої швидкості обробки запитів для зручної роботи.

Досліджено наявні рішення на ринку, такі як платформи Work.ua, Robota.ua та GRC.ua, з акцентом на їхні сильні та слабкі сторони. Було проаналізовано звичні архітектурні моделі, зокрема дво- та трирівневі архітектури, мікросервісний підхід і монолітні системи. На основі цього аналізу обрано оптимальний технологічний стек, що містить сучасні інструменти розробки (Next.js, NestJS, PostgreSQL, TypeORM), та побудовано ефективну архітектуру системи з урахуванням принципів масштабованості, безпеки та продуктивності. Використання REST API для взаємодії між клієнтською та серверною частинами гарантувало гнучкість і можливість подальшого розширення функціоналу.

Визначено функціональні та нефункціональні потреби до web-застосунку, які стали основою для подальшого проектування і розробки застосунку. До функціональних вимог віднесено створення двох типів користувачів (роботодавець і

шукач), здатність створення та пошуку вакансій, відгуків на вакансії, а також управління профілями.

Нефункціональні вимоги містять забезпечення високої продуктивності, адаптивності інтерфейсу для різних пристроїв, захист даних користувачів згідно з стандартів GDPR, а також підтримку стабільної роботи під час одночасного застосування великою кількістю користувачів.

Клієнтська частина застосунку була здійснена з використанням фреймворку Next.js, що гарантувало високий рівень продуктивності завдяки оптимізації завантаження сторінок, адаптивність інтерфейсу для різних пристроїв (десктоп, планшет, смартфон) та підтримку серверного рендерингу (SSR) для покращення SEO та швидкості завантаження. Для стилізації використано Tailwind CSS, що дало змогу швидко створювати сучасний та консистентний дизайн.

Серверна частина розроблена на основі фреймворку NestJS, що надало можливість організувати захищену авторизацію, ефективну взаємодію з базою даних PostgreSQL, а також оптимізувати обробку запитів завдяки модульній структурі NestJS.

Створено систему з двома ролями користувачів — роботодавець і шукач, яка забезпечує чітку взаємодію між сторонами в процесі найму персоналу. Роботодавці можуть створювати та керувати вакансіями, переглядати профілі кандидатів. Шукачі, в свою чергу, мають можливість заповнювати профілі, завантажувати резюме, шукати вакансії та відгукуватися на них із супровідним листом. Система побудована таким чином, щоб забезпечити інтуїтивність і ефективність для обох сторін.

Реалізовано захист особистих даних користувачів згідно зі стандартами безпеки, враховуючи шифрування паролів, застосування HTTPS для передавання даних, а також обмеження доступу до резюме кандидатів.

Здійснено комплексне тестування розробленого застосунку. Результати тестування підтвердили коректність роботи всіх функціональних модулів,

стабільність взаємодії між клієнтською та серверною частинами, а також відповідність системи поставленим функціональним і нефункціональним вимогам.

Таким чином, розроблений web-застосунок для HR-агентства став ефективним інструментом для автоматизації процесів найму персоналу, забезпечивши зручність для користувачів, безпеку даних і потенціал для подальшого розвитку.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Олійник В.О., Бажан Ю.П. Аналіз впливу адаптивного дизайну на користувацький досвід веб-додатків для пошуку роботи на смартфонах та персональних комп'ютерах // Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24 квітня 2025 року, ДУІКТ, м.Київ – С. 153-154.
2. Олійник В.О., Бажан Ю.П. Вплив автоматизації на ефективність процесу підбору персоналу в HR-системах: скорочення часу та покращення якості відбору кандидатів // Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24 квітня 2025 року, ДУІКТ, м.Київ – С. 155-157.

ПЕРЕЛІК ПОСИЛАНЬ

1. Сочинська-Сибірцева І. М., Доренська А. О., Тушевська Т. В. HRменеджмент. DSpace KNTU. URL: https://dspace.kntu.kr.ua/server/api/core/bitstreams/62c3e322-b0b4-4c76-bd18-1716f974282f/content (date of access: 25.05.2025)

2. Холодницька А. В. Сучасні технології підбору персоналу та можливості їхнього практичного використання. Науковий вісник полісся. URL: http://www.irbisnbuv.gov.ua/cgi-bin/irbis_nbuv/cgiirbis_64.exe?C21COM=2&I21DBN=UJRN&P21DBN=UJRN&IMAGE_FILE_DOWNLOAD=1&Image_file_name=PDF/nvp_2015_1_11.pdf (date of access: 25.05.2025)

3. Cable D. M., Gino F., Staats B. R. Reinventing employee onboarding. MIT Sloan Management Review. URL: https://www.hbs.edu/ris/Publication%20Files/Reinventing%20the%20onboarding%20process_3b5ac7ce-f71b-40d9-bb8a-93276b979570.pdf (date of access: 25.05.2025)

4. How Small Businesses Attract and Hire Top Talent. LinkedIn Talent Solutions. URL: https://business.linkedin.com/talent-solutions/resources/talent-strategy/smbattracting-top-talent#hiringchallenges (date of access: 25.05.2025)

5. Silva G. L. L. et al. An automated system for employee recruitment management. IEEE Xplore. URL: https://ieeexplore.ieee.org/document/10025159 (date of access: 25.05.2025)

6. About Us - Work.ua. Work.ua. URL: https://www.work.ua/about-us/ (date of access: 25.05.2025)

7. Robota.ua - Job Search Platform. Robota.ua. URL: https://robota.ua/?utm_source=google&utm_medium=cpc&utm_campaign=Search_Brand

_UA_All_NP%20Exp_MAX_conv&utm_content=657013579878&utm_term=%D1%80%D0%BE%D0%B1%D0%BE%D1%82%D0%B0%20%D1%8E%D0%B0&gad_source=1&gad_campaignid=22505977118&gbraid=0AAAADjAphXp-ig3jKM3chiUxSnR9A0QG&gclid=Cj0KCQjw_8rBBhCFARIsAJrc9yBqRtPBHBK_f1911HVDiQMYDQ6AIn0-ksUUAw4zfh_IVcElnnVU9Z4aAsQqEALw_wcB (date of access: 25.05.2025)

8. GRC.ua - Recruitment Platform. GRC.ua. URL: https://grc.ua/ (date of access: 25.05.2025)

9. OWASP Top Ten Web Application Security Risks. OWASP Foundation. URL: https://owasp.org/www-project-top-ten/ (date of access: 25.05.2025)

10. Gallaugher J. M., Ramanathan S. C. Choosing a client/server architecture: a comparison of two-and three-tier systems. Information Systems Management. URL: https://doi.org/10.1080/10580539608906981 (date of access: 25.05.2025)

11. Kgil T., Mudge T. FlashCache: A NAND flash memory file cache for low power web servers. ResearchGate. URL: https://www.researchgate.net/figure/A-Typical-3-Tier-Server-Architecture-Tier-1-Web-Server-Tier-2-Application-Server-Tier_fig1_221147997 (date of access: 25.05.2025)

12. NestJS Official Documentation. NestJS - A progressive Node.js framework. URL: https://docs.nestjs.com/ (date of access: 25.05.2025)

13. PostgreSQL Official Documentation. PostgreSQL: The world's most advanced open source database. URL: https://www.postgresql.org/ (date of access: 25.05.2025)

14. Next.js Official Documentation. Next.js by Vercel. URL: https://nextjs.org/docs/ (date of access: 25.05.2025)

15. Gerchev I. Tailwind CSS. Google Books. URL: https://books.google.com.ua/books?hl=uk&lr=&id=GczDEAAAQBAJ&oi=fnd&pg=PT3&dq=Tailwind+CSS+(CSS+framework).&ots=hTDf9TMIJb&sig=qiggEFUkOHgKBDaS9guCtDd1SEQ&redir_esc=y#v=onepage&q=Tailwind%20CSS%20(CSS%20framework).

&f=false (date of access: 25.05.2025)

16. Venancio D. A. Enterprise React Development with UmiJS. IEEE Xplore. URL: https://ieeexplore.ieee.org/abstract/document/10163203 (date of access: 25.05.2025)

17. Nielsen J. Usability engineering. Google Books. URL: https://books.google.com.ua/books?hl=uk&lr=&id=95As2OF67f0C&oi=fnd&pg=PR9&dq=According+to+Jakob+Nielsen%27s+usability+principles,+an+interface+should+be+effective,+efficient,+and+satisfying+for+users.&ots=3dCFzqdv1u&sig=dAU6IaQnUkvsJ4CWE_c24a6JKwE&redir_esc=y#v=onepage&q&f=false (date of access: 25.05.2025)

18. Norman D. A. The Design of Everyday Things. ICDST. URL: https://dl.icdst.org/pdfs/files4/4bb8d08a9b309df7d86e62ec4056ceef.pdf (date of access: 25.05.2025)

19. Morales J. F. et al. Lightweight compilation of (C) LP to JavaScript. Cambridge Core. URL: https://doi.org/10.1017/S1471068412000336 (date of access: 25.05.2025)

20. Heino P. Node.js v23.5 in Database Handling Perspective. Theseus. URL: https://www.theseus.fi/bitstream/handle/10024/877439/Heino_Puja.pdf?sequence=2 (date of access: 25.05.2025)

21. Rappl F. Modern Frontend Development with Node.js. Google Books. URL: https://books.google.com.ua/books?hl=uk&lr=&id=HyqdEAAAQBAJ&oi=fnd&pg=PP1&dq=npm+is+the+standard+package+manager+for+Node.js.&ots=jbDpryvvLU&sig=3Tb3DiBs22S1W_Uvy6iEHiVgKbQ&redir_esc=y#v=onepage&q=npm%20is%20the%20standard%20package%20manager%20for%20Node.js.&f=false (date of access: 25.05.2025)

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-застосунку для автоматизації процесів найму співробітників

Виконав студент 4 курсу
група ПД-43
Олійник Валерій Олександрович
Керівник роботи
викладач кафедри ІПЗ Бажан Юрій Павлович

Київ – 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи:

Підвищення ефективності процесу найму співробітників шляхом автоматизації окремих етапів рекрутингу з використанням сучасних web-технологій.

Об'єкт дослідження:

Процес найму персоналу у сучасних компаніях.

Предмет дослідження:

Інструменти та технології web-розробки, що можуть бути використані для автоматизації найму співробітників.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Аналіз сучасних підходів до автоматизації процесів найму.
2. Дослідження та аналіз існуючих web-рішень для керування процесом найму.
3. Визначення вимог та проектування архітектури web-застосунку для автоматизації процесу найму.
4. Розробка клієнтської частини застосунку з використанням Next.js.
5. Реалізація серверної частини системи із застосуванням NestJs, PostgreSQL та TypeORM.
6. Розробка функціональних модулів для створення та управління вакансіями.
7. Реалізація модуля обробки заявок кандидатів.
8. Проведення тестування функціональності та працездатності розробленого web-застосунку .

3

АНАЛІЗ АНАЛОГІВ

Функціональність	Work.ua	Robota.ua	GRC.ua	Head Hunter League
Реєстрація та Типи Користувачів	+	+	+	+
Профілі Користувачів	+	+	+	+
Публікація та Керування Вакансіями	+	+	+	+
Керування Кандидатами	+	+	+	+
Додаткові Сервіси	Статті, аналітика, мобільний додаток	AI, статті, мобільний додаток	Дослідження ринку, консалтинг	Темна/світла тема, анімації
Цільова аудиторія	Кандидати всіх типів по регіонах	Кандидати всіх типів по регіонах	Фахівці середньої та вищої ланки з вищою освітою	IT-фахівці

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

- Реєстрація та авторизація користувачів (рекрутер, кандидат);
- Створення та керування вакансіями;
- Надсилання заявок кандидатами через веб-інтерфейс;
- Перегляд та фільтрація резюме.

Нефункціональні вимоги:

- Масштабованість: можливість обробки великої кількості заявок одночасно;
- Безпека: захист персональних даних користувачів (JWT, шифрування паролю за допомогою argon2);
- Швидкодія: мінімальні затримки при виконанні запитів;
- Зручність користування: Сучасний UI/UX дизайн, побудований за принципами мінімалізму, інтуїтивної навігації, адаптивності (mobile-first), з акцентом на простоту форм взаємодії та контрастну візуалізацію важливої інформації.

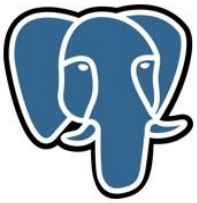
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Postman

GitHub

TaiwindCSS



PostgreSQL

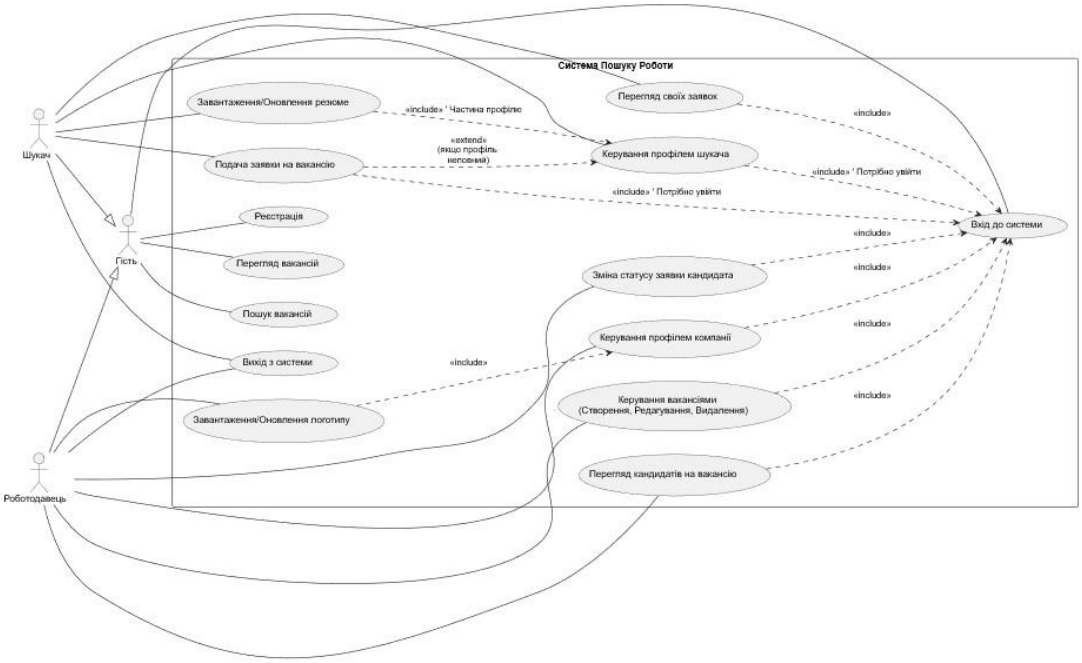
WebStorm

Ant Design

TypeScript

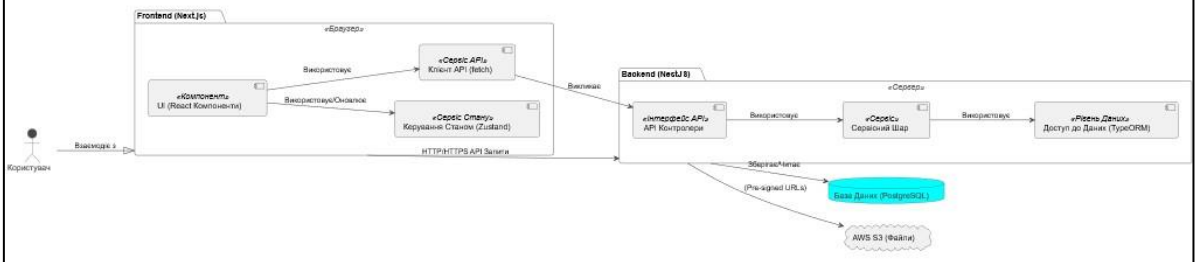
6

Діаграми варіантів використання



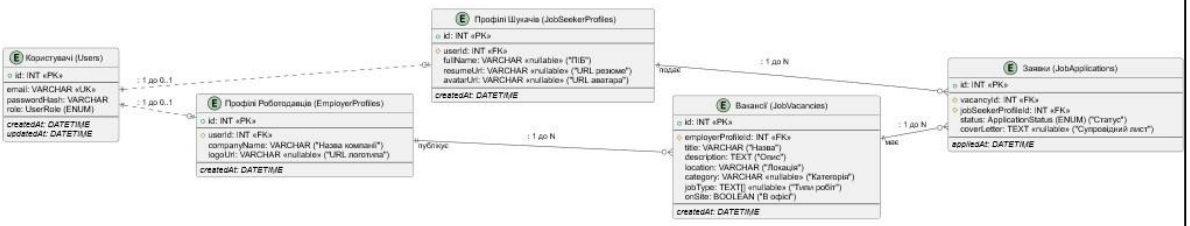
7

Архітектура web-застосунку



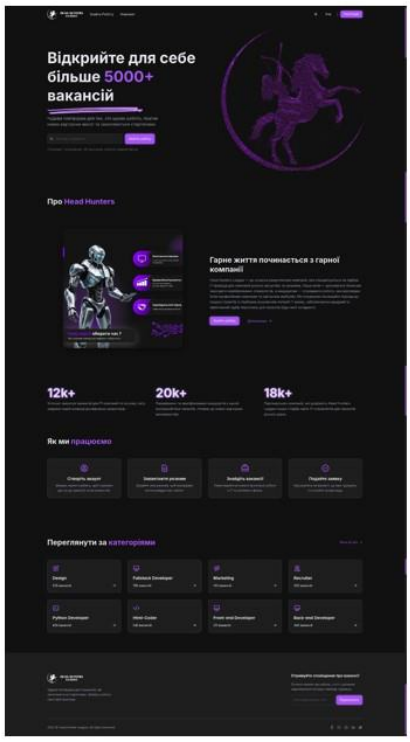
8

Схема бази даних



9

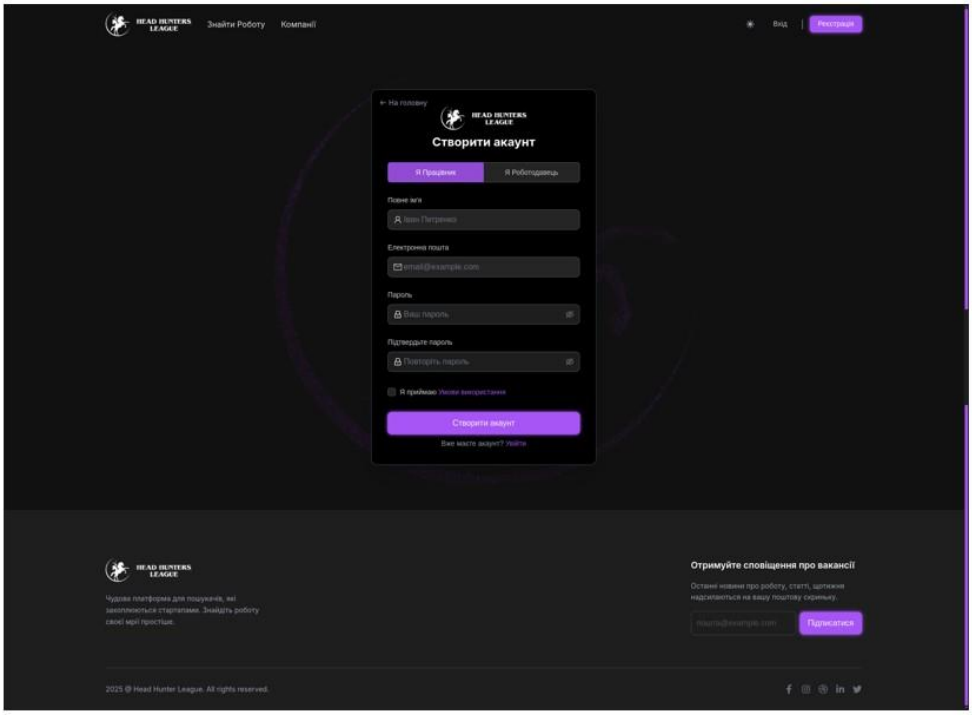
ЕКРАННІ ФОРМИ



Головний екран додатку

10

ЕКРАННІ ФОРМИ



Сторінка реєстрації користувача

11

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези:

1. *Олійник В.О., Бажан Ю.П.*

Аналіз впливу адаптивного дизайну на користувацький досвід веб-додатків для пошуку роботи на смартфонах та персональних комп'ютерах // Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24 квітня 2025 року, ДУІКТ, м.Київ – С. 152-153 (подано до друку).

2. *Олійник В.О., Бажан Ю.П.*

Вплив автоматизації на ефективність процесу підбору персоналу в HR-системах: скорочення часу та покращення якості відбору кандидатів // Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24 квітня 2025 року, ДУІКТ, м.Київ – С. 154-155 (подано до друку).

12

ВИСНОВКИ

1. Проведено аналіз сучасних підходів до автоматизації процесів пошуку та підбору персоналу, що дало змогу визначити вимоги до функціональності та структури web-застосунку для HR-агентства.
2. Досліджено існуючі рішення та типові архітектурні моделі, на основі чого обрано оптимальний технологічний стек і побудовано ефективну архітектуру системи.
3. Визначено функціональні та нефункціональні вимоги до web-застосунку, які стали основою для подальшого проєктування і розробки сервісу.
4. Клієнтська частина реалізована з використанням Next.js, що забезпечило високий рівень продуктивності, адаптивність інтерфейсу для різних пристроїв та підтримку серверного рендерингу (SSR) для покращення SEO.
5. Розроблено серверну частину на основі NestJS, що дало змогу організувати захищену авторизацію, ефективну взаємодію з базою даних через TypeORM та оптимізувати обробку запитів.
6. Створено систему з двома ролями користувачів: роботодавець і кандидат, яка забезпечує організовану взаємодію між сторонами в процесі найму персоналу.
7. Реалізовано захист персональних даних користувачів, функціонал відгуків на вакансії та доступ роботодавців до резюме лише після відгуку кандидата.
8. Проведено тестування розробленого застосунку, яке підтвердило коректність роботи функціональних модулів, стабільність взаємодії між клієнтською та серверною частинами, а також відповідність системи поставленим вимогам.
9. Робота пройшла апробацію на науково-технічних конференціях.
10. Результати дипломної роботи можуть бути використані як основа для подальшого розвитку системи в рамках комерційного проєкту, стартапу або впровадження в HR-процеси кадрових агентств.

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

```
import {
Column,
  CreateDateColumn,
  Entity,
  JoinColumn,
  OneToOne,
  PrimaryGeneratedColumn,
  OneToMany
} from
"typeorm";
import { User } from
"../../user/entities/user.entity";
import { JobVacancy } from "../../job-vacancy/entities/job-vacancy.entity"; import {
 ApiProperty, ApiPropertyOptional } from
"@nestjs/swagger";

@Entity({ name: 'employer_profiles' }) // Явное имя таблицы export class EmployerProfile { @PrimaryGeneratedColumn()  @ApiProperty()
  id: number;

  @OneToOne(() => User, (user) => user.employerProfile)
  @JoinColumn({ name: 'userId' })
  user: User;

  @Column()
  @ApiProperty({ example: 'Acme Corporation'
})  companyName: string;

  @Column({ nullable: true })
  @ApiPropertyOptional({ example:
'https://acme.com'
})  website?: string;

  @Column({ type: 'text', nullable: true })
  @ApiPropertyOptional({  example: 'Leading provider of innovative solutions...' })  description?: string;

  @Column({ nullable: true })
  @ApiPropertyOptional({ example:
'/logos/acme.png'
})  logoUrl?: string;

  @OneToMany(() => JobVacancy, (vacancy) => vacancy.employer, {
    eager: false,
  })
  vacancies: JobVacancy[];

  @CreateDateColumn()
  createdAt: Date;

  @Column({ nullable: true })
  @ApiPropertyOptional({  example:
'501-1000 employees' })  companySize?: string;

  @Column({ nullable: true })
  @ApiPropertyOptional({ example:
'Technology' })
  industry?: string;

  @Column({ nullable: true })
@ApiPropertyOptional({ example: 1999 })
foundedYear?: number;
}

// src/job-application/entities/job-application.entity.ts
import  {  Entity,  PrimaryGeneratedColumn, CreateDateColumn, ManyToOne, Column } from 'typeorm';
import  {  JobVacancy  }  from  '../../job-vacancy/entities/job-vacancy.entity'; import {
 JobSeekerProfile } from '../../job-seeker-profile/entities/job-seeker-profile.entity'; import {
 ApiProperty } from '@nestjs/swagger';

export enum ApplicationStatus {
PENDING = 'Pending',
  REVIEWED = 'Reviewed',
  INTERVIEWING = 'Interviewing',
  OFFERED = 'Offered',
  REJECTED = 'Rejected',
  HIRED = 'Hired',
}

@Entity({ name: 'job_applications' })
export class JobApplication {
@PrimaryGeneratedColumn()
@ApiProperty()
  id: number;

  @ManyToOne(() => JobVacancy, vacancy => vacancy.applications, { eager: false, onDelete: 'CASCADE' })  @ApiProperty({ type: () => JobVacancy })
  vacancy: JobVacancy; // Связь с вакансией

  @ManyToOne(() => JobSeekerProfile, profile => profile.applications, { eager: true, onDelete: 'CASCADE' })  @ApiProperty({ type: () => JobSeekerProfile })
  jobSeekerProfile: JobSeekerProfile;
```

```
@Column({ type: 'enum', enum:
ApplicationStatus, default: ApplicationStatus.PENDING })
@ApiProperty({ enum: ApplicationStatus,
example:
ApplicationStatus.PENDING })
status: ApplicationStatus;

@Column({ type: 'text', nullable: true })
@ApiProperty({ description: 'Cover letter or
message from applicant', nullable: true }) coverLetter?:
string;

@CreateDateColumn()
appliedAt: Date;
}

import {
Column,
CreateDateColumn,
Entity,
JoinColumn,
OneToMany,
OneToOne,
PrimaryGeneratedColumn,

UpdateDateColumn,
} from "typeorm";
import { User } from
"../../user/entities/user.entity";
import { JobApplication } from "../../job-
application/entities/job-application.entity";

@Entity({ name:
'job_seeker_profile' }) export class
JobSeekerProfile {
@PrimaryGeneratedColumn()
id: number;


@OneToOne(() => User, (user) =>
user.jobSeekerProfile, {
nullable: false,
onDelete: 'CASCADE',
})
@JoinColumn({ name: 'userId' })
user: User;

@Column({ nullable: true })

fullName?: string;

@Column({ nullable: true })
phone?: string;

@Column({ nullable: true })
portfolioUrl?: string;

@Column({ type: 'text', nullable: true })
experience?: string;

@Column({ type: 'text', nullable: true })
education?: string;

@Column({ type: 'simple-array', nullable: true
})
skills?: string[];

@Column({ nullable: true })
resumeUrl?: string;

@CreateDateColumn()
createdAt: Date;

@UpdateDateColumn()
updatedAt: Date;

@OneToMany(() => JobApplication,
(application) => application.jobSeekerProfile, {
eager: false,
})
applications: JobApplication[];
}

import {
Column,
CreateDateColumn,
Entity,
ManyToOne,
PrimaryGeneratedColumn,
UpdateDateColumn,
JoinColumn,
OneToMany, } from
"typeorm";
import { EmployerProfile } from "../../employer-
profile/entities/employer-profile.entity"; import {
ApiProperty, ApiPropertyOptional } from
"@nestjs/swagger";
import {JobApplication} from "../../job-
application/entities/job-application.entity";

export enum JobType {
FULL_TIME = 'Full Time',
PART_TIME = 'Part Time',
CONTRACT = 'Contract',
INTERNSHIP = 'Internship',
REMOTE = 'Remote',
EASY_APPLY = 'Easy Apply',
MULTIPLE_CANDIDATE = 'Multiple
Candidate'
}

@Entity({ name: 'job_vacancies' })
export class JobVacancy {
@PrimaryGeneratedColumn()
@ApiProperty({ description: 'Unique ID' })
```

```
    id: number;

    @ManyToOne(() => EmployerProfile,
(employer) => employer.vacancies, {
      eager: true,
      nullable: false,
      onDelete: 'CASCADE'
    })
    @JoinColumn({ name: 'employerProfileId' })
  @ApiProperty({ type: () => EmployerProfile })
  employer: EmployerProfile;

    @Column()
   @ApiProperty({   example:   'Senior   Frontend
Developer' })
    title: string;

    @Column({ type: 'text' })
    @ApiProperty({ example: 'Develop amazing
  user
interfaces...' })
  description: string;

    @OneToMany(()        =>
  JobApplication,  app      => app.vacancy)
applications:JobApplication[];

    @Column()
    @ApiProperty({ example: 'Porto, Portugal' })
    location: string;

    @Column({ nullable: true })
    @ApiPropertyOptional({          example:
  '$80,000          - $120,000' })
    salary?: string;

    @Column({ nullable: true })
    @ApiPropertyOptional({ example:
  'Technology' })
    category?: string;

    @Column({
      type: 'simple-
  array',      nullable:
  true,    })
    @ApiPropertyOptional({          type:
  [String],          enum: JobType, example:
[JobType.FULL_TIME, JobType.REMOTE]
})   jobType?: JobType[];

    @Column({ default: true })
    @ApiProperty({ example: true, description:
  'True if
On Site, False if Remote' })
  onSite: boolean;

    @Column({ type: 'text', nullable: true })
  @ApiPropertyOptional()
    requirements?: string;

    @Column({ type: 'text', nullable:
  true })   @ApiPropertyOptional()
  benefits?: string;

    @CreateDateColumn()
    createdAt: Date;

    @UpdateDateColumn()
  updatedAt: Date;
  }

  // src/user/entities/user.entity.ts (Пример пути)

  import {
  Column,
    CreateDateColumn,
    Entity,
    PrimaryGeneratedColumn,
    OneToOne,

  UpdateDateColumn,
  } from "typeorm";
  import { JobSeekerProfile } from "../../job-seeker-
profile/entities/job-seeker-profile.entity";
  import { EmployerProfile } from "../../employer-
profile/entities/employer-profile.entity";

  export enum UserRole {
  EMPLOYEE = 'employee',
    EMPLOYER = 'employer',
    ADMIN = 'admin'
  }

  @Entity({ name: 'users' })
  export class User {
  @PrimaryGeneratedColumn(
  )   id: number;

    @Column({ unique: true, nullable: false })
    email: string;

    @Column({ nullable: false })
    password: string;

    @Column({
      type: 'enum',
  enum: UserRole,
      nullable: false,
      default: UserRole.EMPLOYEE,
    })   role:
  UserRole;

    @OneToOne(() => JobSeekerProfile, (profile)
  =>
profile.user, {      cascade: true,
  nullable: true,      eager:
```

```
        false,        onDelete:
        'CASCADE',
            onUpdate: 'CASCADE',
          })
          jobSeekerProfile?: JobSeekerProfile;

          @OneToOne(() => EmployerProfile, (profile)
        =>
profile.user, {        cascade: true,
        nullable: true,        eager:
        false,        onDelete:
        'CASCADE',
            onUpdate: 'CASCADE',
          })
          employerProfile?: EmployerProfile;

          @Column({ default: false })
          isEmailVerified: boolean;

          @Column({ nullable: true })
          verificationToken?: string;

          @Column({ nullable: true })
          verificationTokenExpires?: Date;

          @Column({ nullable: true })
          resetPasswordToken?: string;

          @Column({ nullable: true })
          resetPasswordTokenExpires?: Date;

          @Column({ default: true })
          isActive: boolean;

          @Column({ nullable: true })
          lastLoginAt?: Date;

          @CreateDateColumn()
          createdAt: Date;

          @UpdateDateColumn()
        updatedAt: Date;
        }
```